



ЦИФРОВЫЕ
РЕШЕНИЯ

Библиотека OSCAT



Руководство пользователя

10.2026
версия 1.1

Содержание

1 Введение	13
2 Цель документа	14
3 Установка библиотеки	15
4 Описание библиотеки OSCAT	16
5 Глобальные переменные	18
6 Типы данных	19
6.1 CALENDAR	19
6.2 COMPLEX	19
6.3 CONSTANTS_LANGUAGE	20
6.4 CONSTANTS_LOCATION	20
6.5 CONSTANTS_MATH	20
6.6 CONSTANTS_PHYS	21
6.7 CONSTANTS_SETUP	21
6.8 ESR_DATA	21
6.9 FRACTION	21
6.10 HOLIDAY_DATA	22
6.11 REAL2	23
6.12 SDT	23
6.13 VECTOR_3	23
7 Специальные функции	24
7.1 ESR_COLLECT	24
7.2 ESR_MON_B8	24
7.3 ESR_MON_R4	25
7.4 ESR_MON_X8	25
7.5 OSCAT_VERSION	26
7.6 STATUS_TO_ESR	27
8 Математические функции	28
8.1 ACOSH	28
8.2 ACOTH	28
8.3 AGDF	28
8.4 ASINH	29
8.5 ATAN2	29
8.6 ATANH	30
8.7 BETA	30
8.8 BINOM	31
8.9 CAUCHY	31
8.10 CAUCHYCD	32
8.11 CEIL	32
8.12 CEIL2	32
8.13 CMP	33
8.14 COSH	33
8.15 COTH	34
8.16 D_TRUNC	34
8.17 DEC1	34
8.18 DEG	35
8.19 DIFFER	35
8.20 ERF	36
8.21 ERFC	36
8.22 EVEN	37
8.23 EXP10	37
8.24 EXPN	38
8.25 FACT	38
8.26 FIB	38
8.27 FLOOR	39

8.28 FLOOR2	39
8.29 FRACT	40
8.30 GAMMA	40
8.31 GAUSS	41
8.32 GAUSSCD	41
8.33 GCD	42
8.34 GDF	42
8.35 GOLD	43
8.36 HYPOT	43
8.37 INC	43
8.38 INC1	44
8.39 INC2	44
8.40 INV	45
8.41 LAMBERT_W	45
8.42 LANGEVIN	46
8.43 MAX3	46
8.44 MID3	47
8.45 MIN3	47
8.46 MODR	48
8.47 MUL_ADD	48
8.48 NEGX	49
8.49 RAD	49
8.50 RDM	50
8.51 RDM2	50
8.52 RDMDW	51
8.53 REAL_TO_FRAC	51
8.54 RND	52
8.55 ROUND	52
8.56 SGN	53
8.57 SIGMOID	53
8.58 SIGN_I	54
8.59 SIGN_R	54
8.60 SINC	55
8.61 SINH	55
8.62 SQRTN	55
8.63 TANC	56
8.64 TANH	56
8.65 WINDOW	57
8.66 WINDOW2	57
9 Работа с массивами	59
9.1 _ARRAY_ABS	59
9.2 _ARRAY_ADD	59
9.3 _ARRAY_INIT	60
9.4 _ARRAY_MEDIAN	60
9.5 _ARRAY_MUL	61
9.6 _ARRAY_SHUFFLE	61
9.7 _ARRAY_SORT	62
9.8 ARRAY_AVG	62
9.9 ARRAY_GAV	63
9.10 ARRAY_HAV	63
9.11 ARRAY_MAX	64
9.12 ARRAY_MIN	64
9.13 ARRAY_SDV	65
9.14 ARRAY_SPR	65
9.15 ARRAY_SUM	65
9.16 ARRAY_TREND	66

9.17 ARRAY_VAR.....	66
9.18 IS_SORTED	67
10 Комплексная арифметика	68
10.1 CABS	68
10.2 CACOS	68
10.3 CACOSH	69
10.4 CADD	69
10.5 CARG	70
10.6 CASIN	70
10.7 CASINH	71
10.8 CATAN	71
10.9 CATANH	72
10.10 CCON	72
10.11 CCOS	73
10.12 CCOSH	73
10.13 CDIV	74
10.14 CEXP	74
10.15 CINV	74
10.16 CLOG	75
10.17 CMUL	75
10.18 CPOL	76
10.19 CPOW	76
10.20 CSET	77
10.21 CSIN	77
10.22 CSINH	78
10.23 CSQRT	78
10.24 CSUB	78
10.25 CTAN	79
10.26 CTANH	79
11 Арифметика чисел двойной точности	80
11.1 R2_ABS	80
11.2 R2_ADD	80
11.3 R2_ADD2	81
11.4 R2_MUL	81
11.5 R2_SET	82
12 Арифметические функции	83
12.1 F_LIN	83
12.2 F_LIN2	83
12.3 F_POLY	84
12.4 F_POWER	84
12.5 F_QUAD	85
12.6 FRMP_B	85
12.7 FT_AVG	86
12.8 FT_MIN_MAX	87
12.9 FT_RMP	87
12.10 LINEAR_INT	88
12.11 POLYNOM_INT	89
13 Геометрические функции	91
13.1 CIRCLE_A	91
13.2 CIRCLE_C	91
13.3 CIRCLE_SEG	92
13.4 CONE_V	93
13.5 ELLIPSE_A	93
13.6 ELLIPSE_C	94
13.7 SPHERE_V	95
13.8 TRIANGLE_A	96

14 Операции над векторами	98
14.1 V3_ABS	98
14.2 V3_ADD	98
14.3 V3_ANG	99
14.4 V3_DPRO	99
14.5 V3_NORM	99
14.6 V3_NUL	100
14.7 V3_PAR	100
14.8 V3_REV	101
14.9 V3_SMUL	101
14.10 V3_SUB	101
14.11 V3_XANG	102
14.12 V3_XPRO	102
14.13 V3_YANG	103
14.14 V3_ZANG	103
15 Дата и время	104
15.1 CALENDAR_CALC	104
15.2 DATE_ADD	104
15.3 DAY_OF_DATE	105
15.4 DAY_OF_MONTH	105
15.5 DAY_OF_WEEK	105
15.6 DAY_OF_YEAR	106
15.7 DAY_TO_TIME	106
15.8 DAYS_DELTA	106
15.9 DAYS_IN_MONTH	107
15.10 DAYS_IN_YEAR	107
15.11 DCF77	107
15.12 DST	108
15.13 DT_TO_SDT	109
15.14 DT2_TO_SDT	109
15.15 EASTER	110
15.16 EVENTS	110
15.17 HOLIDAY	111
15.18 HOUR	112
15.19 HOUR_OF_DT	113
15.20 HOUR_TO_TIME	113
15.21 HOUR_TO_TOD	113
15.22 JD2000	114
15.23 LEAP_DAY	114
15.24 LEAP_OF_DATE	114
15.25 LEAP_YEAR	115
15.26 LTIME_TO_UTC	115
15.27 MINUTE	116
15.28 MINUTE_OF_DT	116
15.29 MINUTE_TO_TIME	116
15.30 MONTH_BEGIN	117
15.31 MONTH_END	117
15.32 MONTH_OF_DATE	117
15.33 MULTIME	118
15.34 PERIOD	118
15.35 PERIOD2	119
15.36 REFRACTION	120
15.37 RTC_2	120
15.38 RTC_MS	121
15.39 SDT_TO_DATE	122
15.40 SDT_TO_DT	122

15.41 SDT_TO_TOD	123
15.42 SECOND	123
15.43 SECOND_OF_DT	124
15.44 SECOND_TO_TIME	124
15.45 SET_DATE	124
15.46 SET_DT	125
15.47 SET_TOD	126
15.48 SUN_MIDDAY	126
15.49 SUN_POS	126
15.50 SUN_TIME	127
15.51 TIMECHECK	128
15.52 UTC_TO_LTIME	129
15.53 WORK_WEEK	130
15.54 YEAR_BEGIN	130
15.55 YEAR_END	130
15.56 YEAR_OF_DATE	131
16 Работа со строками	132
16.1 BIN_TO_BYTE	132
16.2 BIN_TO_DWORD	132
16.3 BYTE_TO_STRB	132
16.4 BYTE_TO_STRH	133
16.5 CAPITALIZE	133
16.6 CHARCODE	133
16.7 CHARNAME	135
16.8 CHR_TO_STRING	135
16.9 CLEAN	136
16.10 CODE	136
16.11 COUNT_CHAR	136
16.12 COUNT_SUBSTRING	137
16.13 DEC_TO_BYTE	137
16.14 DEC_TO_DWORD	138
16.15 DEC_TO_INT	138
16.16 DEL_CHARS	138
16.17 DT_TO_STRF	139
16.18 DWORD_TO_STRB	140
16.19 DWORD_TO_STRF	140
16.20 DWORD_TO_STRH	141
16.21 EXEC	141
16.22 FILL	142
16.23 FIND_CHAR	142
16.24 FIND_CTRL	143
16.25 FIND_NONUM	143
16.26 FIND_NUM	143
16.27 FINDB	144
16.28 FINDB_NONUM	144
16.29 FINDB_NUM	145
16.30 FINDP	145
16.31 FIX	145
16.32 FLOAT_TO_REAL	146
16.33 FSTRING_TO_BYTE	147
16.34 FSTRING_TO_DT	147
16.35 FSTRING_TO_DWORD	148
16.36 FSTRING_TO_MONTH	149
16.37 FSTRING_TO_WEEK	149
16.38 FSTRING_TO_WEEKDAY	150
16.39 HEX_TO_BYTE	150

16.40 HEX_TO_DWORD	151
16.41 IS_ALNUM.....	151
16.42 IS_ALPHA.....	152
16.43 IS_CC.....	152
16.44 IS_CTRL.....	153
16.45 IS_HEX.....	153
16.46 IS_LOWER.....	154
16.47 IS_NCC	154
16.48 IS_NUM.....	155
16.49 IS_UPPER.....	155
16.50 ISC_ALPHA.....	156
16.51 ISC_CTRL	156
16.52 ISC_HEX.....	157
16.53 ISC_LOWER.....	157
16.54 ISC_NUM	158
16.55 ISC_UPPER.....	158
16.56 LOWERCASE	159
16.57 MESSAGE_8	159
16.58 MESSAGE_4R.....	159
16.59 MIRROR.....	160
16.60 MONTH_TO_STRING	160
16.61 OCT_TO_BYTE	161
16.62 OCT_TO_DWORD	161
16.63 REAL_TO_STRF.....	162
16.64 REPLACE_ALL	162
16.65 REPLACE_CHARS	163
16.66 REPLACE_UML	163
16.67 TICKER	164
16.68 TO_LOWER.....	164
16.69 TO_UML.....	165
16.70 TO_UPPER.....	165
16.71 TRIM	165
16.72 TRIM1	166
16.73 TRIME.....	166
16.74 UPPERCASE.....	167
16.75 WEEKDAY_TO_STRING	167
17 Модули памяти	168
17.1 FIFO_16	168
17.2 FIFO_32	168
17.3 STACK_16.....	168
17.4 STACK_32.....	169
18 Генераторы импульсов.....	170
18.1 A_TRIG	170
18.2 B_TRIG	170
18.3 CLICK_CNT.....	171
18.4 CLICK_DEC.....	171
18.5 CLK_DIV	172
18.6 CLK_N.....	173
18.7 CLK_PRG.....	174
18.8 CLK_PULSE.....	174
18.9 CYCLE_4	175
18.10 D_TRIG	176
18.11 GEN_BIT	177
18.12 GEN_SQ.....	177
18.13 SCHEDULER.....	178
18.14 SCHEDULER_2	180

18.15 SEQUENCE_4	182
18.16 SEQUENCE_8	183
18.17 SEQUENCE_64	184
18.18 TMAX	184
18.19 TMIN	185
18.20 TOF_1	185
18.21 TONOF	186
18.22 TP_1	187
18.23 TP_1D	187
18.24 TP_X	188
19 Логические модули	190
19.1 BCDC_TO_INT	190
19.2 BIT_COUNT	190
19.3 BIT_LOAD_B	190
19.4 BIT_LOAD_B2	191
19.5 BIT_LOAD_DW	191
19.6 BIT_LOAD_DW2	192
19.7 BIT_LOAD_W	192
19.8 BIT_LOAD_W2	193
19.9 BIT_OF_DWORD	193
19.10 BIT_TOGGLE_B	194
19.11 BIT_TOGGLE_DW	194
19.12 BIT_TOGGLE_W	194
19.13 BYTE_OF_BIT	195
19.14 BYTE_OF_DWORD	195
19.15 BYTE_TO_BITS	196
19.16 BYTE_TO_GRAY	196
19.17 CHECK_PARITY	197
19.18 CHK_REAL	197
19.19 CRC_GEN	197
19.20 DEC_2	200
19.21 DEC_4	200
19.22 DEC_8	201
19.23 DW_TO_REAL	202
19.24 DWORD_OF_BYTE	202
19.25 DWORD_OF_WORD	203
19.26 GRAY_TO_BYTE	203
19.27 INT_TO_BCDC	204
19.28 MATRIX	204
19.29 MUX_2	205
19.30 MUX_4	206
19.31 PARITY	207
19.32 REAL_TO_DW	207
19.33 REFLECT	207
19.34 REVERSE	208
19.35 SHL1	208
19.36 SHR1	208
19.37 SWAP_BYTE	209
19.38 SWAP_BYTE2	209
19.39 WORD_OF_BYTE	210
19.40 WORD_OF_DWORD	210
20 Триггеры, элементы хранения и регистры сдвига	211
20.1 COUNT_BR	211
20.2 COUNT_DR	212
20.3 FF_D2E	213
20.4 FF_D4E	213

20.5 FF_DRE	214
20.6 FF_JKE	214
20.7 FF_RSE.....	214
20.8 LTCH.....	215
20.9 LTCH_4	215
20.10 SELECT_8.....	215
20.11 SHR_4E.....	216
20.12 SHR_4UDE.....	217
20.13 SHR_8PLE	217
20.14 SHR_8UDE.....	218
20.15 STORE_8	218
20.16 TOGGLE.....	218
21 Генераторы сигналов.....	219
21.1 _RMP_B.....	219
21.2 _RMP_NEXT	220
21.3 _RMP_W	221
21.4 GEN_PULSE	222
21.5 GEN_PW2.....	222
21.6 GEN_RDM.....	223
21.7 GEN_RDT	224
21.8 GEN_RMP.....	225
21.9 GEN_SIN.....	226
21.10 GEN_SQR	227
21.11 PWM_DC.....	229
21.12 PWM_PW	230
21.13 RMP_B.....	230
21.14 RMP_SOFT	231
21.15 RMP_W	232
22 Обработка сигналов.....	233
22.1 AIN.....	233
22.2 AIN1.....	234
22.3 AOUT	235
22.4 AOUT1	235
22.5 BYTE_TO_RANGE	236
22.6 DELAY.....	237
22.7 DELAY_4.....	238
22.8 FADE.....	239
22.9 FILTER_DW.....	240
22.10 FILTER_I	241
22.11 FILTER_MAV_DW	241
22.12 FILTER_MAV_W	242
22.13 FILTER_W	243
22.14 FILTER_WAV	243
22.15 MIX.....	243
22.16 MUX_R2.....	244
22.17 MUX_R4.....	244
22.18 OFFSET	245
22.19 OFFSET2	246
22.20 OVERRIDE	247
22.21 RANGE_TO_BYTE.....	248
22.22 RANGE_TO_WORD.....	249
22.23 SCALE.....	249
22.24 SCALE_B	250
22.25 SCALE_B2.....	251
22.26 SCALE_B4.....	252
22.27 SCALE_B8.....	252

22.28 SCALE_D	253
22.29 SCALE_R	254
22.30 SCALE_X2.....	255
22.31 SCALE_X4.....	256
22.32 SCALE_X8.....	256
22.33 SEL2_OF_3	257
22.34 SEL2_OF_3B.....	257
22.35 SH.....	257
22.36 SH_1	258
22.37 SH_2	259
22.38 SH_T	260
22.39 STAIR.....	260
22.40 STAIR2.....	260
22.41 TREND	261
22.42 TREND_DW.....	261
22.43 WORD_TO_RANGE.....	261
23 Термометры сопротивления	263
23.1 MULTI_IN	263
23.2 RES_NI	264
23.3 RES_NTC.....	265
23.4 RES_PT	266
23.5 RES_SI	267
23.6 SENSOR_INT	268
23.7 TEMP_NI.....	269
23.8 TEMP_NTC	269
23.9 TEMP_PT	270
23.10 TEMP_SI	271
24 Модули измерения и отсчета времени.....	272
24.1 ALARM_2	272
24.2 BAR_GRAPH.....	274
24.3 CALIBRATE	275
24.4 CYCLE_TIME	275
24.5 DT_SIMU.....	276
24.6 FLOW_METER	276
24.7 M_D	277
24.8 M_T.....	277
24.9 M_TX	278
24.10 METER.....	278
24.11 METER_STAT	279
24.12 ONTIME	280
24.13 T_PLC_MS	280
24.14 T_PLC_US.....	281
24.15 TC_MS	281
24.16 TC_S.....	281
24.17 TC_US.....	281
25 Конвертация величин	282
25.1 ASTRO.....	282
25.2 BFT_TO_MS.....	283
25.3 C_TO_F.....	283
25.4 C_TO_K	284
25.5 DEG_TO_DIR	284
25.6 DIR_TO_DEG	285
25.7 ENERGY	286
25.8 F_TO_C.....	287
25.9 F_TO_OM.....	287
25.10 F_TO_PT.....	288

25.11 GEO_TO_DEG.....	288
25.12 K_TO_C.....	289
25.13 KMH_TO_MS.....	289
25.14 LENGHT.....	290
25.15 MS_TO_BFT.....	291
25.16 MS_TO_KMH.....	292
25.17 OM_TO_F.....	292
25.18 PRESSURE.....	293
25.19 PT_TO_F.....	294
25.20 SPEED.....	294
25.21 TEMPERATURE.....	295
26 Регуляторы.....	298
26.1 BAND_B.....	298
26.2 CONTROL_SET1.....	299
26.3 CONTROL_SET2.....	299
26.4 CTRL_IN.....	300
26.5 CTRL_OUT.....	301
26.6 CTRL_PI.....	302
26.7 CTRL_PID.....	304
26.8 CTRL_PWM.....	305
26.9 DEAD_BAND.....	306
26.10 DEAD_BAND_A.....	307
26.11 DEAD_ZONE.....	308
26.12 DEAD_ZONE2.....	309
26.13 FT_DERIV.....	309
26.14 FT_IMP.....	309
26.15 FT_INT.....	310
26.16 FT_INT2.....	311
26.17 FT_PD.....	311
26.18 FT_PDT1.....	311
26.19 FT_PI.....	312
26.20 FT_PID.....	312
26.21 FT_PIDW.....	313
26.22 FT_PIDWL.....	314
26.23 FT_PIW.....	314
26.24 FT_PIWL.....	315
26.25 FT_PT1.....	315
26.26 FT_PT2.....	316
26.27 FT_TN8.....	316
26.28 FT_TN16.....	317
26.29 FT_TN64.....	317
26.30 HYST.....	317
26.31 HYST_1.....	317
26.32 HYST_2.....	318
26.33 HYST_3.....	318
26.34 INTEGRATE.....	319
27 Модули управления.....	320
27.1 DRIVER_1.....	320
27.2 DRIVER_4.....	320
27.3 DRIVER_4C.....	320
27.4 FLOW_CONTROL.....	321
27.5 FT_PROFILE.....	322
27.6 INC_DEC.....	322
27.7 INTERLOCK.....	323
27.8 INTERLOCK_4.....	324
27.9 MANUAL.....	325

27.10 MANUAL_1	325
27.11 MANUAL_2	326
27.12 MANUAL_4	326
27.13 PARSET	327
27.14 PARSET2	328
27.15 SIGNAL	328
27.16 SIGNAL_4	328
27.17 SRAMP	329
27.18 TUNE	329
27.19 TUNE2	330
28 Работа с буфером	332
28.1 _BUFFER_CLEAR	332
28.2 _BUFFER_INIT	332
28.3 _BUFFER_INSERT	333
28.4 _BUFFER_UPPERCASE	334
28.5 _STRING_TO_BUFFER	335
28.6 BUFFER_COMP	336
28.7 BUFFER_SEARCH	337
28.8 BUFFER_TO_STRING	338
29 Работа со списками	340
29.1 LIST_ADD	340
29.2 LIST_CLEAN	341
29.3 LIST_GET	342
29.4 LIST_INSERT	343
29.5 LIST_LEN	344
29.6 LIST_NEXT	345
29.7 LIST_RETRIEVE	346
29.8 LIST_RETRIEVE_LAST	346

1 Введение

OSCAT (Open Source Community for Automation Technology) — это открытая библиотека программных компонентов для промышленной автоматизации. Авторство библиотеки принадлежит сообществу **OSCAT** oscat.de.

При написании данного руководства использовался перевод описания библиотеки OSCAT Basic от сайта oscat.ru. Выражаем огромную благодарность авторам перевода за проделанную работу.

2 Цель документа

Библиотека OSCAT содержит программные модули (функции и функциональные блоки), используемые для разработки прикладного программного обеспечения ПЛК,

Данный документ содержит описание алгоритмов работы блоков, входные/выходные параметры и типы данных каждого блока, и предназначен для ознакомления с составом библиотеки.

Для полного понимания принципов работы в среде необходимо ознакомиться с руководством пользователя ALTA IDE.

3 Установка библиотеки

Чтобы подключить библиотеку к проекту:

1. Откройте редактор **Менеджер библиотек** одним из способов:
 - дважды нажмите ЛКМ на системную папку **Менеджер библиотек** в дереве проекта;
 - или нажмите ПКМ на системную папку **Менеджер библиотек** в дереве проекта и выберите в контекстном меню пункт **Открыть**:

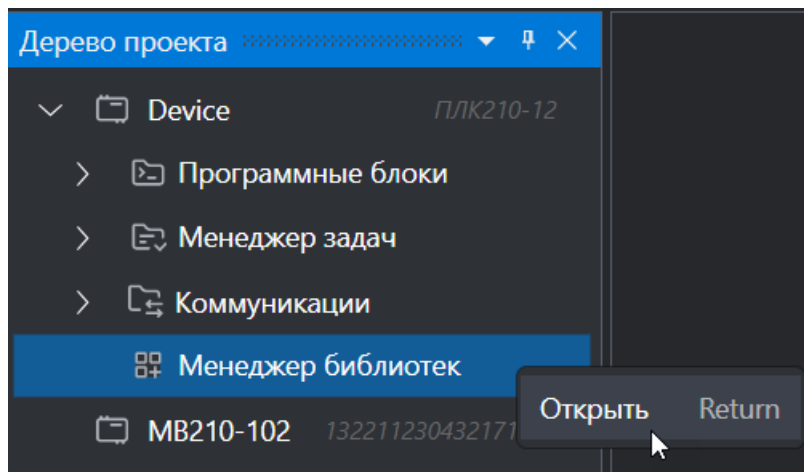


Рисунок 3.1

- В редакторе **Менеджер библиотек** откройте вкладку **Магазин**.
- Выберите библиотеку **OSCAT** и нажмите кнопку **Подключить** на карточке библиотеки.
- Библиотека добавится в проект и отобразится на вкладке **Мои библиотеки** в редакторе **Менеджер библиотек**.

4 Описание библиотеки OSCAT

Прямой доступ к памяти

Названия модулей, изменяющих значения своих входных переменных с помощью указателей (например, `_ARRAY_SORT`), начинаются с символа нижнего подчеркивания («_»). Функция `_ARRAY_SORT` сортирует массив с помощью прямого доступа к памяти, что является очень эффективным решением, поскольку позволяет обойтись без промежуточных массивов (и, соответственно, сэкономить память), а также уменьшить время сортировки (т.к. не потребуется копировать данные из входного массива в промежуточные). Тем не менее, данные модули должны использоваться только опытными специалистами, поскольку их некорректный вызов может привести к серьезным ошибкам (вплоть до прекращения работы приложения). В частности, при вызове таких функций необходимо делать проверку на то, что значения указателей не являются нулевыми или неопределенными.

Названия модулей

Модули, названия которых начинаются с префикса **FT_**, подразумевают использование на некотором отрезке времени (например, ФБ `FT_AVG` вычисляет среднее значение переменной по заданному количеству срезов, и его однократный вызов не приведет к желаемому результату).

Модули, названия которых начинаются с префикса **F_**, подразумевают однократный вызов (например, функция `F_LIN` возвращает значение линейной функции с заданными коэффициентами, и ее вызов имеет смысл только в том случае, если значение одной из входных переменных изменилось).



ПРИМЕЧАНИЕ

Данное соглашение распространяется на модули из раздела Арифметические функции и Регуляторы.

Параметры модулей

Чтобы облегчить процесс разработки и упростить работу со сложными функциями, многие модули библиотеки **OSCAT** имеют параметры. Параметры представляют переменные класса **VAR_INPUT CONSTANT**, которые задаются при вызове модуля и не могут изменить свои значения в процессе его работы.

На языке **ST** значения параметров задаются непосредственно при вызове модуля:

Пример вызова модуля с параметрами:

```
OSCAT_BASIC.SCHEDULER_0(T := tT1, T3 := T#200MS);
```

Отчеты о событиях и ошибках (ESR)

Модули, выполняющие сложные функции, обычно имеют выход **Error**, который используется для сигнализации о возникших ошибках. Значение **0** соответствует нормальной работе модуля. При возникновении ошибок выход принимает значение из диапазона **1...99** – это значение соответствует коду ошибки. Расшифровка кодов приводится в описании соответствующего модуля.

Выход **Status** выполняет аналогичную функцию, но содержит коды не ошибок, а состояний модуля (например, включен/остановлен/выключен). Диапазон кодов состояний: **100... 199**. Диапазон **200... 255** зарезервирован для отладочных сообщений.

ESR-модули позволяют собирать информацию об ошибках и состояниях с метками времени, которая при необходимости может быть сохранена в архив или передана на верхний уровень АСУ (реализация архивации и передачи должны быть выполнена пользователем). Это позволяет с помощью стандартных возможностей библиотеки вести аварийные и оперативные журналы, не используя дополнительный функционал системы. Описание ESR-модулей приведено в разделе [специальные функции](#).

Библиотека OSCAT содержит:

- [глобальные переменные](#);
- [типы данных](#);
- [специальные функции](#);
- [математические функции](#);
- [работа с массивами](#);
- [комплексная арифметика](#);
- [арифметика чисел двойной точности](#);
- [арифметические функции](#);
- [геометрические функции](#);
- [операции над векторами](#);
- [дата и время](#);
- [работа со строками](#);
- [модули памяти](#);
- [генераторы импульсов](#);
- [логические модули](#);

- триггеры, элементы хранения и регистры сдвига;
- генераторы сигналов;
- обработка сигналов;
- термометры сопротивления;
- модули изменения и отсчета времени;
- конвертация величин;
- регуляторы;
- модули управления;
- работа с буфером;
- работа со списками.

5 Глобальные переменные

В библиотеке OSCAT физические и математические константы представлены в виде глобальных переменных.

Название переменной	Тип	Описание
Глобальные переменные		
MATH	CONSTANTS_MATH	Структура CONSTANTS_MATH определяет математические константы.
PHYS	CONSTANTS_PHYS	Структура CONSTANTS_PHYS определяет физические константы.
LANGUAGE	CONSTANTS_LANGUAGE	Структура CONSTANTS_LANGUAGE определяет языковые константы.
SETUP	CONSTANTS_SETUP	Структура CONSTANTS_SETUP определяет вспомогательные константы для работы со строками и временем.
LOCATION	CONSTANTS_LOCATION	Структура CONSTANTS_LOCATION определяет языки регионов.
Глобальные константы		
String_Length	INT	Используется в функциях работы со строками для определения их размеров. Значение по умолчанию – 250. Рекомендуется использовать эту константу при объявлении всех строковых переменных приложения.
List_Length	INT	Используется для определения размеров списков . Значение по умолчанию – 250.

6 Типы данных

6.1 CALENDAR

Структура **CALENDAR** содержит подробную информацию о текущей дате и времени. Она используется при работе с ФБ [CALENDAR_CALC](#).

Переменная структуры	Тип	Описание
UTC	DT	Дата и время по UTC
LDT	DT	Местные дата и время
LDATE	DATE	Дата по LDT
LTOD	TOD	Время по LDT
YEAR	INT	Год по LDT
MONTH	INT	Месяц по LDT
DAY	INT	День по LDT
WEEKDAY	INT	День недели по LDT (1 – понедельник, 7 – воскресенье)
OFFSET	INT	Смещение LDT относительно UTC в минутах
DST_EN	BOOL	Управление режимом летнего времени (TRUE – включить, FALSE – выключить).
DST_ON	BOOL	Флаг режима летнего времени (TRUE – включен, FALSE – отключен)
NAME	STRING(5)	Название часового пояса
LANGUAGE	INT	Язык региона. Используется для определения названий месяцев и дней
LONGITUDE	REAL	Долгота
LATITUDE	REAL	Широта
SUN_RISE	TOD	Время восхода солнца по LDT
SUN_SET	TOD	Время заката солнца по LDT
SUN_MIDDAY	TOD	Время солнечного полудня по UTC
SUN_HEIGHT	REAL	Склонение солнца над горизонтом в градусах
SUN_HOR	REAL	Азимут
SUN_VER	REAL	Высота солнца над горизонтом в градусах с учетом рефракции
NIGHT	BOOL	Флаг «Сейчас ночь»
HOLIDAY	BOOL	Флаг «Сегодня праздник»
HOLY_NAME	STRING(30)	Название праздника
WORK_WEEK	INT	Номер недели в году (начиная с 1)

6.2 COMPLEX

Структура **COMPLEX** используется для представления комплексных чисел. Модули работы с комплексными числами описаны в разделе [Комплексная арифметика](#).

Переменная структуры	Тип	Описание
re	REAL	Действительная часть комплексного числа
im	REAL	Мнимая часть комплексного числа

6.3 CONSTANTS_LANGUAGE

Структура **CONSTANTS_LANGUAGE** используется для задания параметров языков проекта. По умолчанию структура определяет три языка – английский, немецкий и французский. Для работы со структурой используется [глобальная переменная LANGUAGE](#).

Переменная структуры	Тип	Описание
DEFAULT	INT	Язык по умолчанию. Языковые модули библиотеки в качестве входа принимают код языка. Значение 0 соответствует языку по умолчанию, значения от 1 до LMAX – одному из языков проекта
LMAX	INT	Количество языков проекта. Значение по умолчанию – 3 (см. размерность следующих массивов)
WEEKDAYS	ARRAY [1..3, 1..7] OF STRING(10)	Полные названия дней недели (Monday) для всех языков проекта
WEEKDAYS2	ARRAY [1..3, 1..7] OF STRING(2)	Сокращенные названия дней недели (Mo) для всех языков проекта
MONTHS	ARRAY [1..3, 1..12] OF STRING(10)	Полные названия месяцев (January) для всех языков проекта
MONTHS3	ARRAY [1..3, 1..12] OF STRING(3)	Сокращенные названия месяцев (Jan) для всех языков проекта
DIRS	ARRAY [1..3, 0..15] OF STRING(3)	Обозначения румбов для всех языков проекта

6.4 CONSTANTS_LOCATION

Структура **CONSTANTS_LOCATION** используется для задания языков регионов. По умолчанию структура определяет пять регионов, в каждом из которых используется немецкий язык. Для работы со структурой применяется [глобальная переменная LOCATION](#).

Переменная структуры	Тип	Описание
DEFAULT	INT	Регион по умолчанию.
LMAX	INT	Количество языков проекта. Значение по умолчанию – 5 .
LANGUAGE	ARRAY [1..5] OF INT	Коды языков для регионов проекта.

6.5 CONSTANTS_MATH

Структура **CONSTANTS_MATH** определяет математические константы. Для работы со структурой применяется [глобальная переменная MATH](#).

Переменная структуры	Тип	Точность (количество знаков после запятой)	Описание
PI	REAL	35	Число Пи
PI2	REAL	35	Число Пи · 2
PI4	REAL	35	Число Пи · 4
PI05	REAL	13	Число Пи / 2
PI025	REAL	15	Число Пи / 4
PI_INV	REAL	15	1 / число Пи
E	REAL	35	Число Эйлера
E_INV	REAL	15	1 / число Эйлера

Переменная структуры	Тип	Точность (количество знаков после запятой)	Описание
SQ2	REAL	13	Квадратный корень из 2
FACTS	ARRAY [0..12] OF DINT	-	Факториалы чисел от 0 до 12

6.6 CONSTANTS_PHYS

Структура **CONSTANTS_PHYS** определяет физические константы. Для работы со структурой применяется [глобальная переменная PHYS](#).

Переменная структуры	Тип	Точность (количество знаков после запятой)	Описание
C	REAL	-	Скорость света, м/с
E	REAL	8	Элементарный электрический заряд, Кл
G	REAL	5	Ускорение свободного падения, м/с ²
T0	REAL	2	Абсолютный нуль температуры, °C
RU	REAL	6	Универсальная газовая постоянная, Дж/(моль · K).
PN	REAL	-	Нормальное атмосферное давление, Па

6.7 CONSTANTS_SETUP

Структура **CONSTANTS_SETUP** определяет вспомогательные константы для работы со строками и временем. Для работы со структурой применяется [глобальная переменная SETUP](#).

Переменная структуры	Тип	Описание
EXTENDED_ASCII	BOOL	Флаг использования верхней половины таблицы ASCII
CHARNAME	ARRAY [1..4] OF STRING(253)	Содержит символы Юникода
MTH_OFS	ARRAY [1..12] OF INT	Содержит номера последних дней месяцев календарного года (0, 31, 59, 70 и т.д.)
DECADES	ARRAY [0..8] OF REAL	Массив результатов возведения в степень числа 10

6.8 ESR_DATA

Структура **ESR_DATA** используется [ESR-модулями](#) для сбора информации об ошибках и изменении состояний модулей, поддерживающих данный функционал.

Переменная структуры	Тип	Описание
TYP	BYTE	Тип события
ADRESS	STRING(10)	Имя контролируемого параметра
DS	DT	Дата и время события
TS	TIME	Относительное время события (в мс после старта проекта)
DATA	ARRAY [0..7] OF BYTE	Информация о событии

6.9 FRACTION

Структура **FRACTION** используется для представления дробей.

Переменная структуры	Тип	Описание
NUMERATOR	INT	Числитель дроби
DENOMINATOR	INT	Знаменатель дроби

6.10 HOLIDAY_DATA

Структура **HOLIDAY_DATA** содержит информацию о праздниках. Крайне сложно дать емкое описание переменных структуры – гораздо проще объяснить принцип их использования на примерах.

Переменная структуры	Тип	Описание
NAME	STRING(30)	Название праздника
DAY	SINT	См. ниже
MONTH	SINT	См. ниже
USE	SINT	См. ниже

С календарной точки зрения праздники можно разделить на две группы:

1. Праздники с фиксированной датой.
2. Переходящие праздники.

Для праздников с фиксированной датой переменные **DAY** и **MONTH** определяют дату проведения, а переменная **USE** характеризует, следует ли учитывать праздник модулям библиотеки (**1** – следует, **0** – не следует).

Пример объявления:

```
ALL_SAINTS: OSCAT_BASIC.HOLIDAY_DATA := (NAME := 'All Saints Day',
                                         DAY  := 1,
                                         MONTH := 11,
                                         USE   := 1);
```

Для переходящих праздников возможно два варианта определения:

1. Относительно заданного опорного праздника. Опорным считается праздник с **DAY=0** и **MONTH=0**:

```
ALL_SAINTS: OSCAT_BASIC.HOLIDAY_DATA := (NAME := 'Пасха',
                                         DAY   := 0,
                                         MONTH := 0,
                                         USE    := 1);
```

День Святой Троицы является переходящим праздником – он отмечается спустя 50 дней после Пасхи (при этом Пасха считается 1-м днем). Тогда объявить его можно следующим образом:

```
SAINT_TRINITY_DAY: OSCAT_BASIC.HOLIDAY_DATA := (NAME := 'Святая Троица',
                                                  DAY   := 49,
                                                  MONTH := 0,
                                                  USE    := 1);
```

В переменной **DAY** указывается смещение (положительное или отрицательное) относительно опорного праздника, переменная **MONTH** задается нулем.

2. Относительно заданной даты. Например, немецкий праздник День покаяния и молитвы празднуется в последнюю среду перед началом Адвента. Соответственно, если дата Адвента известна (в 2016-м году это **27-ое ноября**), то праздник можно определить следующим образом:

```
BUSS_UND_BETAG: OSCAT_BASIC.HOLIDAY_DATA := (NAME := 'Buss und Betag',
                                                DAY   := 27,
                                                MONTH := 11,
                                                USE    := -3);
```

Переменная **USE** в данном случае определяет день недели перед заданной датой (где **-1** соответствует понедельнику ... **-7** соответствует воскресенью).

В приведенных выше примерах праздники объявляются по одному, но в большинстве случаев удобнее представить их сразу в виде массива:

```
HOLIDAYS_RU: OSCAT_BASIC.ARRAY [0..29] OF HOLIDAY_DATA;
```

6.11 REAL2

Структура **REAL2** используется для представления чисел двойной точности на системах без поддержки переменных типа **LREAL**, но только при работе с определенными функциями и рядом ограничений. Модули работы с числами двойной точности описаны в разделе [Арифметика чисел двойной точности](#).

Пример представления: $11.222222233 = RX + R1$, где:

$RX=11.2222222$,

$R1=0.0000000033$.

Переменная структуры	Тип	Описание
R1	REAL	Разность между числом двойной точности и R1
RX	REAL	Число двойной точности, округленное до REAL

6.12 SDT

Структура **SDT** используется для определения разрядов даты и времени.

Переменная структуры	Тип	Описание
YEAR	INT	Год
MONTH	INT	Месяц
DAY	INT	День
WEEKDAY	INT	День недели (1 – понедельник, 7 – воскресенье)
HOURL	INT	Часы
MINUTE	INT	Минуты
SECOND	INT	Секунды
MS	INT	Миллисекунды

6.13 VECTOR_3

Структура **VECTOR_3** определяет радиус-вектор в трехмерном пространстве. Модули работы с векторами описаны в разделе [Операции над векторами](#).

Переменная структуры	Тип	Описание
X	REAL	Координата X
Y	REAL	Координата Y
Z	REAL	Координата Z

7 Специальные функции

7.1 ESR_COLLECT

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	ESR_0...ESR_7	ARRAY [0...3] OF ESR_DATA	Массив входных данных
	rst	BOOL	Сигнал удаления данных
Выходы	ESR_OUT	ARRAY [0...31] OF ESR_DATA	Массив выходных данных
Входы-выходы	pos	INT	Номер последнего записанного элемента выходного массива
Используемые модули	INC1		



ВНИМАНИЕ

В текущей версии библиотеки ФБ работает **некорректно**. Для корректной работы необходимо отредактировать код ФБ следующим образом:

```
IF rst OR cnt < 0 THEN
  pos := -1;
  cnt := 0;    //bug-fix
```

Функциональный блок **ESR_COLLECT** используется для сбора данных других ESR-модулей и сохранения их в массив. Входы **ESR_0...ESR_7** используются как значения для входов-выходов **ESR_Out** других ESR-модулей. При обновлении информации ESR-модулей, данные заносятся в массив **ESR_COLLECT.ESR_OUT**, при этом значение входа-выхода **pos** инкрементируется. Массив **ESR_COLLECT.ESR_OUT** может хранить до 32-х элементов типа [ESR_DATA](#). При превышении этого количества начинается перезапись элементов (начиная с нулевого). По переднему фронту на входе **rst** происходит удаление всех данных модуля, после чего **pos** принимает значение -1.

7.2 ESR_MON_B8

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	s_0...s_7	BOOL	Сигналы для мониторинга
	DT_in	DT	Текущие дата и время
Выходы	ESR_Flag	BOOL	Признак обновления данных
Входы-выходы	ESR_out	ARRAY [0...3] OF ESR_DATA	Информация мониторинга
Параметры	a0...a7	STRING(10)	Названия сигналов
Используемые модули	T_PLC_MS		

Функциональный блок **ESR_MON_B8** применяется для мониторинга от одной до восьми переменных типа BOOL (**s0...s7**). На вход **DT_in** подается значение системного времени в формате DT (считанного с помощью одной из доступных пользователю библиотек). При изменении значения переменных **s0...s7** данные мониторинга сохраняются в массив **ESR_Out**, элементы которого являются экземплярами структуры типа [ESR_DATA](#). Массив содержит 4 элемента и, соответственно, позволяет хранить данные мониторинга для четырех одновременно изменившихся переменных. Иными словами, при одновременном (произошедшем в одном цикле ПЛК) изменении всех восьми переменных будет сохранена информация только о четырех (приоритет отдается сигналам с меньшим номером). Для ведения архива мониторинга можно использовать ФБ [ESR_COLLECT](#). Выход **ESR_flag** принимает значение **TRUE** на один цикл при обновлении данных мониторинга.

Структура [ESR_DATA](#) включает следующую информацию:

Переменная структуры	Тип	Описание
TYP	BYTE	Тип изменения переменной типа BOOL. 11 – с FALSE на TRUE, 10 – с TRUE на FALSE Изменяется только на один цикл, после чего сбрасывается в 0
ADRESS	STRING(10)	Описание переменной (указывается в параметрах модуля)
DS	DT	Дата и время изменения
TS	TIME	Относительное время изменения (в мс после старта проекта)
DATA	ARRAY [0..7] OF BYTE	Не используется данным ФБ

7.3 ESR_MON_R4

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	R0...R3	BOOL	Сигналы для мониторинга
	DT_in	DT	Текущие дата и время
Выходы	ESR_Flag	BOOL	Признак обновления данных
Входы-выходы	ESR_out	ARRAY [0...3] OF ESR_DATA	Информация мониторинга
Параметры	a0...a3	STRING(10)	Названия сигналов
	s0...s3	REAL	Гистерезис сигналов
Используемые модули	T_PLC_MS , DIFFER		

Функциональный блок **ESR_MON_R4** применяется для мониторинга от одной до четырех переменных типа REAL (**R0...R3**). На вход **DT_in** подается значение системного времени в формате DT (считанного с помощью одной из доступных пользователю библиотек). При изменении значения переменных **R0...R3** на величину **s0...s3** или выше данные мониторинга сохраняются в массив **ESR_Out**, элементы которого являются экземплярами структуры типа [ESR_DATA](#). Массив содержит 4 элемента и, соответственно, позволяет хранить данные о последнем изменении для каждой переменной. Для ведения архива мониторинга можно использовать ФБ [ESR_COLLECT](#). Выход **ESR_flag** принимает значение **TRUE** на один цикл при обновлении данных мониторинга.

Структура [ESR_DATA](#) включает следующую информацию:

Переменная структуры	Тип	Описание
TYP	BYTE	Принимает значение 20 при изменении переменной. Изменяется только на один цикл, после чего сбрасывается в 0
ADRESS	STRING(10)	Описание переменной (указывается в параметрах модуля)
DS	DT	Дата и время изменения
TS	TIME	Относительное время изменения (в мс после старта проекта)
DATA	ARRAY [0..7] OF BYTE	В первых четырех байтах содержится новое значение переменной в формате IEEE 754

7.4 ESR_MON_X8

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	s0...s7	BYTE	Сигналы для мониторинга
	DT_in	DT	Текущие дата и время
	Mode	BYTE	Режим работы блока

Тип модуля: ФБ	Переменная	Тип	Описание
Выходы	ESR_Flag	BOOL	Признак обновления данных
Входы-выходы	ESR_out	ARRAY [0...3] OF ESR_DATA	Информация мониторинга
Параметры	a0...a7	STRING(10)	Названия сигналов
Используемые модули	T_PLC_MS , STATUS_TO_ESR		

Функциональный блок **ESR_MON_X8** применяется для мониторинга от одной до восьми переменных типа BYTE (**s0... s7**), которые содержат информацию о состоянии других ФБ. Такие переменные обычно называются **Status**. В качестве примера можно привести выход **Status** ФБ [SEQUENCE_4](#). На вход **DT_in** подается значение системного времени в формате DT (считанного с помощью одной из доступных пользователю библиотек). Вход **Mode** определяет режим работы блока:

- Mode=1 – обрабатываются только сообщения об ошибках;
- Mode=2 – обрабатываются сообщения об ошибках и состоянии ФБ;
- Mode=3 – обрабатываются сообщения об ошибках, состоянии ФБ.

При изменении значения переменных **s0... s7** данные мониторинга сохраняются в массив **ESR_Out**, элементы которого являются экземплярами структуры типа [ESR_DATA](#). Массив содержит 4 элемента и, соответственно, позволяет хранить данные мониторинга для четырех одновременно изменившихся переменных. Иными словами, при одновременном (произошедшем в одном цикле ПЛК) изменении всех восьми переменных будет сохранена информация только о четырех (приоритет отдается сигналам с меньшим номером). Для ведения архива мониторинга можно использовать ФБ [ESR_COLLECT](#). Выход **ESR_flag** принимает значение **TRUE** на один цикл при обновлении данных мониторинга.

Структура [ESR_DATA](#) включает следующую информацию:

Переменная структуры	Тип	Описание
TYP	BYTE	Код статуса. Изменяется только на один цикл, после чего сбрасывается в 0
ADRESS	STRING(10)	Описание переменной (указывается в параметрах модуля)
DS	DT	Дата и время изменения
TS	TIME	Относительное время изменения (в мс, после старта ПЛК)
DATA	ARRAY [0..7] OF BYTE	В нулевом байте содержится код статуса

7.5 OSCAT_VERSION

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	BOOL	Вход функции
Выходы	OSCAT_VERSION	DWORD	Версия библиотеки/дата релиза

Функция **OSCAT_VERSION** возвращает версию библиотеки (в формате DWORD) при значении IN=FALSE и дату релиза данной версии (в формате DWORD) при значении IN=TRUE.

```

VAR
  xIn      : BOOL;
  dwVer    : DWORD;  (* версия библиотеки *)
  dwRel    : DWORD;  (* дата релиза *)
END_VAR

xIn := FALSE;
dwVer := OSCAT_BASIC.OSCAT_VERSION(IN := xIn);  (* вернёт версию библиотеки (DWORD),
зависит от установленной версии *)

xIn := TRUE;
dwRel := OSCAT_BASIC.OSCAT_VERSION(IN := xIn);  (* вернёт дату релиза (DWORD),
зависит от установленной версии *)

END_PROGRAM

```

7.6 STATUS_TO_ESR

Тип модуля: функция	Переменная	Тип	Описание
Входы	status	BYTE	Код статуса
	adress	STRING(10)	Описание переменной
	DT_in	DT	Дата и время изменения
	TS	TIME	Относительное время изменения (в мс, после старта ПЛК)
Выходы	STATUS_TO_ESR	ESR_DATA	Информация мониторинга

Функция **STATUS_TO_ESR** собирает элемент структуры **ESR_DATA** из отдельных переменных.

Коды статуса **1...99** соответствуют сообщениям об ошибках, **100... 199** – состояниям ФБ, **200... 255** – отладочным сообщениям.

```

VAR
  bStatus  : BYTE := 101;          (* 100..199 – состояния ФБ *)
  sAddr    : STRING(10) := 'FB1.STATE'; (* описание/имя параметра *)
  dtChange : DT := DT#2026-06-18-12:00:00; (* дата/время события *)
  tStamp   : TIME := T#1234ms;      (* мс после старта *)

  esr      : OSCAT_BASIC.ESR_DATA;  (* результат *)
  typ      : BYTE;
  addrOut  : STRING(10);
  dsOut    : DT;
  tsOut    : TIME;
  b0       : BYTE;
END_VAR

esr := OSCAT_BASIC.STATUS_TO_ESR(
  status := bStatus,
  adress := sAddr,
  DT_in  := dtChange,
  TS     := tStamp
);

(* Пример чтения полей структуры ESR_DATA *)
typ      := esr.TYP;
addrOut  := esr.ADRRESS;
dsOut    := esr.DS;
tsOut    := esr.TS;
b0       := esr.DATA[0];

END_PROGRAM

```

8 Математические функции

8.1 ACOSH

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Аргумент
Выходы	ACOSH	REAL	Значение гиперболического арккосинуса

Функция **ACOSH** возвращает значение гиперболического арккосинуса, вычисленное по формуле

$$\text{ACOSH}(X) = \ln(X + \sqrt{X^2 - 1})$$

Область определения: $1 \leq X < +\infty$

Пример

```
VAR
  x  : REAL := 2.0;    (* Аргумент, должен быть >= 1.0 *)
  y  : REAL;           (* Результат ACOSH(x) *)
END_VAR

y := OSCAT_BASIC.ACOSH(x);    (* y ≈ 1.3169579 *)
END_PROGRAM
```

8.2 ACOTH

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Аргумент
Выходы	ACOTH	REAL	Значение гиперболического аркотангенса

Функция **ACOTH** возвращает значение гиперболического аркотангенса, вычисленное по формуле

$$\text{ACOTH}(X) = \frac{1}{2} \ln\left(\frac{X+1}{X-1}\right)$$

Область определения: $|X| > 1$

```
VAR
  x  : REAL := 2.0;    (* |x| > 1 *)
  y  : REAL;
END_VAR

y := OSCAT_BASIC.ACOTH(X := x);    (* y ≈ 0.5493061 *)
END_PROGRAM
```

8.3 AGDF

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Аргумент
Выходы	AGDF	REAL	Значение обратной функции Гудермана

Функция **AGDF** возвращает значение обратной функции Гудермана, вычисленное по формуле

$$\text{AGDF}(X) = \ln \left(\frac{1 + \sin(X)}{\cos(X)} \right)$$

Текущая реализация функции гарантирует приемлемую точность вычислений только для x , принадлежащих интервалу $-\pi/2 \leq X \leq \pi/2$

```
VAR
  x : REAL := 0.5;
  y : REAL;
END_VAR

y := OSCAT_BASIC.AGDF(X := x); (* y=0.5222381*)

END_PROGRAM
```

8.4 ASINH

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Аргумент
Выходы	ASINH	REAL	Значение гиперболического арксинуса

Функция **ASINH** возвращает значение гиперболического арксинуса, вычисленное по формуле

$$\text{ASINH}(X) = \ln(X + \sqrt{X^2 + 1})$$

Область определения: $-\infty < X < +\infty$

```
VAR
  x : REAL := 2;
  y : REAL;
END_VAR

y := OSCAT_BASIC.ASINH(X := x); (* y=1.44363546*)

END_PROGRAM
```

8.5 ATAN2

Тип модуля: функция	Переменная	Тип	Описание
Входы	Y	REAL	Координата Y
	X	REAL	Координата X
Выходы	ATAN	REAL	Значение арктангенса для отношения Y/X

Функция **ATAN2** возвращает значение арктангенса для заданных координат X и Y. Арктангенс – это угол между осью X и линией, проведенной из начала координат (0,0) в точку с координатами (X, Y). Угол

определяется в радианах в диапазоне $-\pi/2 \leq \text{ATAN2} \leq \pi/2$

```

VAR
  x  : REAL := 1.0;
  y  : REAL := 1.0;
  a1 : REAL;

  x2 : REAL := 1.0;
  y2 : REAL := 0.0;
  a2 : REAL;
END_VAR

a1 := OSCAT_BASIC.ATAN2(Y := y, X := x);  (* a1 ≈ 0.7853982 рад (π/4) *)
a2 := OSCAT_BASIC.ATAN2(Y := y2, X := x2); (* a2 = 0 рад *)

END_PROGRAM

```

8.6 ATANH

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Аргумент
Выходы	ATANH	REAL	Значение гиперболического арктангенса

Функция **ATANH** возвращает значение гиперболического арктангенса, вычисленное по формуле

$$\text{ATANH}(X) = \frac{1}{2} \ln \left(\frac{1+X}{1-X} \right)$$

Область определения: $-1 < X < 1$

```

VAR
  X  : REAL := 0.5;  (* Аргумент (в диапазоне -1..1, не включая ±1) *)
  Y  : REAL;          (* Результат *)
END_VAR

Y := OSCAT_BASIC.ATANH(X);  (* Y = ATANH(0.5) ≈ 0.549306 *)

END_PROGRAM

```

8.7 BETA

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Аргумент X
	Y	REAL	Аргумент Y
Выходы	BETA	REAL	Значение бета-функции
Используемые модули	GAMMA		

Функция **BETA** возвращает значение бета-функции, вычисленное по формуле

$$\text{BETA}(X, Y) = \frac{\Gamma(X) \cdot \Gamma(Y)}{\Gamma(X + Y)},$$

где Γ — гамма-функция.

Область определения: $X > 1, Y > 1$

```

VAR
  x : REAL := 0.5;
  y : REAL := 0.5;
  b : REAL;
END_VAR

b := OSCAT_BASIC.BETA(X := x, Y := y);  (* b = 3.10966635 *)

END_PROGRAM

```

8.8 BINOM

Тип модуля: функция	Переменная	Тип	Описание
Входы	N	INT	Аргумент N
	K	INT	Аргумент K
Выходы	BINOM	DINT	Биномиальный коэффициент из N по K

Функция **BINOM** возвращает значение биномиального коэффициента из N по K, вычисленное по формуле

$$\text{BINOM}(N, K) = \frac{N!}{K! \cdot (N - K)!}$$

Функция возвращает корректные значения только для $N \geq K > 0$

```

VAR
  n : INT := 4;
  k : INT := 2;
  c : DINT;
END_VAR

c := OSCAT_BASIC.BINOM(N := n, K := k);  (* c = 6 *)

END_PROGRAM

```

8.9 CAUCHY

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Аргумент
	T	REAL	Коэффициент сдвига
	U	REAL	Коэффициент масштаба
Выходы	CAUCHY	REAL	Плотность вероятности для распределения Коши

Функция **CAUCHY** возвращает значение плотности вероятности для распределения Коши, вычисленное по

$$\text{CAUCHY}(X) = \frac{1}{\pi} \cdot \frac{U}{U^2 + (X - T)^2}, \quad U > 0$$

формуле

```

VAR
  x : REAL := 1.0;  (* аргумент *)
  t : REAL := 0.0;  (* сдвиг *)
  u : REAL := 0.5;  (* масштаб, > 0 *)
  p : REAL;
END_VAR

p := OSCAT_BASIC.CAUCHY(X := x, T := t, U := u);  (* p ≈ 0.127323955 *)

END_PROGRAM

```

8.10 CAUCHYCD

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Аргумент
	T	REAL	Коэффициент сдвига
	U	REAL	Коэффициент масштаба
Выходы	CAUCHYCD	REAL	Функция распределения Коши

Функция **CAUCHYCD** возвращает значение функции распределения Коши, вычисленное по формуле

$$\text{CAUCHYCD}(X) = \frac{1}{\pi} \cdot \arctg\left(\frac{X - T}{U}\right) + \frac{1}{2}, U > 0$$

8.11 CEIL

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Округляемое значение
Выходы	CEIL	INT	Округленное значение

Функция **CEIL** округляет переменную типа REAL в сторону ближайшего большего целого значения.



ПРИМЕЧАНИЕ

Функция возвращает значение типа INT, что накладывает соответствующие ограничения на величину входной переменной (диапазон возможных значений для типа INT составляет –32768..32767).

```
VAR
  x1 : REAL := 3.14;
  x2 : REAL := -2.30;
  r1 : INT;
  r2 : INT;
END_VAR

r1 := OSCAT_BASIC.CEIL(X := x1);  (* r1 = 4 *)
r2 := OSCAT_BASIC.CEIL(X := x2);  (* r2 = -2 *)

END_PROGRAM
```

8.12 CEIL2

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Округляемое значение
Выходы	CEIL	DINT	Округленное значение

Функция **CEIL2** округляет переменную типа REAL в сторону ближайшего большего целого значения. В отличие от функции **CEIL**, данная функция возвращает значение типа DINT, что увеличивает диапазон возможных значений входной переменной.

```
VAR
  x1 : REAL := 33000.22;
  x2 : REAL := -33000.22;
  r1 : DINT;
  r2 : DINT;
END_VAR

r1 := OSCAT_BASIC.CEIL2(X := x1);  (* r1 = 33001 *)
r2 := OSCAT_BASIC.CEIL2(X := x2);  (* r2 = -33000 *)

END_PROGRAM
```


8.13 CMP

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	1-я сравниваемая величина
	Y	REAL	2-я сравниваемая величина
	N	INT	Кол-во сравниваемых знаков
Выходы	CMP	BOOL	Результат сравнения

Функция **CMP** сравнивает значения X и Y типа REAL и возвращает **TRUE**, если их первые N символов совпадают. В противном случае функция возвращает **FALSE**.

Необходимо отметить, что тип REAL обеспечивает относительную точность в 7-8 десятичных цифр после запятой в заданном диапазоне, поэтому нельзя гарантировать корректную работу функции в 100% случаев.

```

VAR
  x1 : REAL := 12.34567;
  y1 : REAL := 12.34512;
  n1 : INT  := 5;
  eq1 : BOOL;

  x2 : REAL := 1.234567;
  y2 : REAL := 1.229999;
  n2 : INT  := 4;
  eq2 : BOOL;
END_VAR

eq1 := OSCAT_BASIC.CMP(X := x1, Y := y1, N := n1); (* eq1 = TRUE *)
eq2 := OSCAT_BASIC.CMP(X := x2, Y := y2, N := n2); (* eq2 = FALSE *)

END_PROGRAM

```

8.14 COSH

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Аргумент
Выходы	COSH	REAL	Значение гиперболического косинуса

Функция **COSH** возвращает значение гиперболического косинуса, вычисленное по формуле

$$\text{COSH}(X) = \frac{e^X + e^{-X}}{2}$$

Область определения: $-\infty < X < +\infty$

```

VAR
  x : REAL := 2.0;
  y : REAL;
END_VAR

y := OSCAT_BASIC.COSH(X := x); (* y ≈ 3.76219583 *)

END_PROGRAM

```

8.15 COTH

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Аргумент
Выходы	COTH	REAL	Значение гиперболического котангенса

Функция **COTH** возвращает значение гиперболического котангенса, вычисленное по формуле

$$\text{COTH}(X) = \frac{e^{2X} + 1}{e^{2X} - 1}$$

Область определения: $X \neq 0$

```
VAR
    x : REAL := 2.0;
    y : REAL;
END_VAR

y := OSCAT_BASIC.COTH(X := x);    (* y ≈ 1.03731477 *)

END_PROGRAM
```

8.16 D_TRUNC

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Округляемое значение
Выходы	D_TRUNC	DINT	Округленное значение

Функция **D_TRUNC** возвращает целую часть переменной типа REAL (обратите внимание, что речь идет не об округлении до ближайшего целочисленного значения). Функция **TRUNC ()** из стандарта МЭК 61131-3 поддерживает тип DINT не во всех средах программирования; более того, даже преобразование **REAL_TO_DINT** на разных системах может приводить к разному результату. Функция **D_TRUNC** анализирует работу преобразования и возвращает корректное значение.

```
VAR
    x1 : REAL := 3.99;
    x2 : REAL := -3.99;
    r1 : DINT;
    r2 : DINT;
END_VAR

r1 := OSCAT_BASIC.D_TRUNC(X := x1);    (* r1 = 3 *)
r2 := OSCAT_BASIC.D_TRUNC(X := x2);    (* r2 = -3 *)

END_PROGRAM
```

8.17 DEC1

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	INT	Декрементируемое значение 1
	N	INT	Декрементируемое значение 2
Выходы	DEC1	INT	Результат декремента

Функция **DEC1** декрементирует значение переменной X. Если $X = 0$, то функция возвращает декрементированное значение переменной N.

```

VAR
  x1 : INT := 5;
  n1 : INT := 10;
  r1 : INT;

  x2 : INT := 0;
  n2 : INT := 10;
  r2 : INT;
END_VAR

r1 := OSCAT_BASIC.DEC1(X := x1, N := n1);    (* r1 = 4 *)
r2 := OSCAT_BASIC.DEC1(X := x2, N := n2);    (* r2 = 9 *)

END_PROGRAM

```

8.18 DEG

Тип модуля: функция	Переменная	Тип	Описание
Входы	rad	REAL	Значение угла в радианах
Выходы	DEG	REAL	Значение угла в градусах

Функция **DEG** преобразует значение угла из радиан в градусы. Если входная переменная rad превышает 2π , то из нее будет вычитаться по π до тех пор, пока ее значение не окажется в интервале $0 < \text{rad} < 2\pi$.

```

VAR
  r1 : REAL := 3.1415927;    (* π *)
  d1 : REAL;
END_VAR

d1 := OSCAT_BASIC.DEG(rad := r1);    (* d1 ≈ 180.0 *)

END_PROGRAM

```

8.19 DIFFER

Тип модуля: функция	Переменная	Тип	Описание
Входы	in1	REAL	Сравниваемое значение 1
	in2	REAL	Сравниваемое значение 2
	X	REAL	Допустимая разность значений
Выходы	DIFFER	TRUE	Флаг сравнения

Функция **DIFFER** возвращает значение **TRUE**, если значения переменных **in1** и **in2** отличаются на величину, превышающую **X**. В противном случае функция возвращает **FALSE**.

```

VAR
  a : REAL := 10.0;
  b : REAL := 10.3;
  x : REAL := 0.2;
  r1 : BOOL;

  c : REAL := 10.0;
  d : REAL := 10.1;
  r2 : BOOL;
END_VAR

r1 := OSCAT_BASIC.DIFFER(in1 := a, in2 := b, X := x);
r2 := OSCAT_BASIC.DIFFER(in1 := c, in2 := d, X := x);

END_PROGRAM

```

(* r1 = TRUE,
 |10.0-10.3| = 0.3 > 0.2 *)
 (* r2 = FALSE,
 |10.0-10.1| = 0.1 <= 0.2 *)

8.20 ERF

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Аргумент
Выходы	ERF	REAL	Функция ошибок
Используемые модули	SGN		

Функция **ERF** возвращает значение функции ошибок для аргумента **X**, вычисленное по формуле

$$\text{ERF}(X) = \pm \sqrt{1 - \exp\left(-x^2 \cdot \frac{\frac{4}{\pi} + ax^2}{1 + ax^2}\right)}, \text{ где } a = \frac{8}{3\pi} \cdot \frac{3 - \pi}{\pi - 4}$$

Относительная ошибка вычисления не превышает 0.00013.

```

VAR
  x : REAL := 0.05;
  y : REAL;
END_VAR

y := OSCAT_BASIC.ERF(X := x);
(* y = 0.5637164 *)

END_PROGRAM

```

8.21 ERFC

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Аргумент
Выходы	ERFC	REAL	Дополнительная функция ошибок
Используемые модули	ERF		

Функция **ERFC** возвращает значение дополнительной функции ошибок для аргумента **X**, вычисленное по формуле

$$\text{ERFC}(X) = 1 - \text{ERF}(X)$$

```

VAR
  x : REAL := 0.05;
  y : REAL;

END_VAR

y := OSCAT_BASIC.ERFC(x := x);    (* y = 0.9436284 *)

END_PROGRAM

```

8.22 EVEN

Тип модуля: функция	Переменная	Тип	Описание
Входы	in	DINT	Анализируемое значение
Выходы	EVEN	BOOL	Флаг четности

Функция **EVEN** возвращает **TRUE**, если значение переменной **in** является четным, и **FALSE** — если нечетным.

```

VAR
  a : DINT := 10;
  b : DINT := 11;
  r1 : BOOL;
  r2 : BOOL;
END_VAR

r1 := OSCAT_BASIC.EVEN(in := a);    (* r1 = TRUE *)
r2 := OSCAT_BASIC.EVEN(in := b);    (* r2 = FALSE *)

END_PROGRAM

```

8.23 EXP10

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Показатель степени
Выходы	EXP10	REAL	Показательная функция

Функция **EXP10** возвращает значение показательной функции с основанием степени 10, вычисленное по формуле

$$\text{EXP10}(X) = 10^X$$

```

VAR
  x1 : REAL := 2.0;
  y1 : REAL;

  x2 : REAL := -1.0;
  y2 : REAL;
END_VAR

y1 := OSCAT_BASIC.EXP10(X := x1);    (* y1 = 100.0 *)
y2 := OSCAT_BASIC.EXP10(X := x2);    (* y2 = 0.1 *)

END_PROGRAM

```

8.24 EXPN

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Основание степени
	N	INT	Показатель степени
Выходы	EXPN	REAL	Показательная функция

Функция **EXPN** возвращает значение показательной функции с целочисленным основанием степени, вычисленное по формуле

$$\text{EXPN}(X) = X^N$$

Функция **EXPN** разработана специально для ПЛК без FPU (математического сопроцессора) и выполняется приблизительно в 30 раз быстрее функции **EXPT**, определенной стандартом МЭК 61131-3.

```

VAR
  x1 : REAL := 2.0;
  n1 : INT  := 10;
  y1 : REAL;

  x2 : REAL := 5.0;
  n2 : INT  := 0;
  y2 : REAL;
END_VAR

y1 := OSCAT_BASIC.EXPON(X := x1, N := n1);  (* y1 = 1024.0 *)
y2 := OSCAT_BASIC.EXPON(X := x2, N := n2);  (* y2 = 1.0 *)

END_PROGRAM

```

8.25 FACT

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	INT	Аргумент
Выходы	FACT	DINT	Значение факториала

Функция **FACT** возвращает значение факториала **X**. Функция возвращает корректные значения при **X**, принадлежащем интервалу $0 \leq X \leq 12$. Во всех остальных случаях функция возвращает **-1**.

```

VAR
  x1 : INT := 5;
  f1 : DINT;

  x2 : INT := 13;
  f2 : DINT;
END_VAR

f1 := OSCAT_BASIC.FACT(X := x1);  (* f1 = 120 *)
f2 := OSCAT_BASIC.FACT(X := x2);  (* f2 = -1 (вне диапазона 0..12) *)

END_PROGRAM

```

8.26 FIB

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	INT	Номер элемента последовательности
Выходы	FIB	DINT	Значение элемента последовательности

Функция **FIB** возвращает значение **X**-го элемента последовательности Фибоначчи. Функция возвращает корректные значения при **X**, принадлежащему интервалу $0 \leq X \leq 46$. Во всех остальных случаях функция возвращает -1.

```
VAR
  x1 : INT := 10;
  f1 : DINT;

  x2 : INT := -1;
  f2 : DINT;
END_VAR

f1 := OSCAT_BASIC.FIB(X := x1);  (* f1 = 55 *)
f2 := OSCAT_BASIC.FIB(X := x2);  (* f2 = -1 (вне допустимого диапазона) *)

END_PROGRAM
```

8.27 FLOOR

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Округляемое значение
Выходы	FLOOR	INT	Округленное значение

Функция **FLOOR** округляет переменную типа REAL в сторону ближайшего меньшего целого значения. Обратите внимание, что функция возвращает значение типа INT, что накладывает соответствующие ограничения на величину входной переменной (диапазон возможных значений для типа INT составляет -32768..32767).

Пример

```
VAR
  x1 : REAL := 3.14;
  x2 : REAL := -2.30;
  r1 : INT;
  r2 : INT;
END_VAR

r1 := OSCAT_BASIC.FLOOR(X := x1);  (* r1 = 3 *)
r2 := OSCAT_BASIC.FLOOR(X := x2);  (* r2 = -3 *)

END_PROGRAM
```

8.28 FLOOR2

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Округляемое значение
Выходы	FLOOR2	DINT	Округленное значение

Функция **FLOOR2** округляет переменную типа REAL в сторону ближайшего меньшего целого значения. В отличие от функции **FLOOR**, данная функция возвращает значение типа DINT, что увеличивает диапазон возможных значений входной переменной.

Пример

```

VAR
  x1 : REAL := 35000.25;
  x2 : REAL := -35000.25;
  r1 : DINT;
  r2 : DINT;
END_VAR

r1 := OSCAT_BASIC.FLOOR2(X := x1); (* r1 = 35000 *)
r2 := OSCAT_BASIC.FLOOR2(X := x2); (* r2 = -35001 *)

END_PROGRAM

```

8.29 FRACT

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Значение с плавающей точкой
Выходы	FRACT	REAL	Дробная часть значения
Используемые модули	D_TRUNC		

Функция **FRACT** возвращает дробную часть значения с плавающей точкой. Необходимо отметить, что тип REAL обеспечивает относительную точность в 7-8 десятичных цифр после запятой в заданном диапазоне, поэтому нельзя гарантировать корректную работу функции в 100% случаев.

Пример

```

VAR
  x1 : REAL := 12.345;
  f1 : REAL;

  x2 : REAL := -12.345;
  f2 : REAL;
END_VAR

f1 := OSCAT_BASIC.FRACT(X := x1); (* f1 ≈ 0.345 *)
f2 := OSCAT_BASIC.FRACT(X := x2); (* f2 ≈ -0.345 *)

END_PROGRAM

```

8.30 GAMMA

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Аргумент
Выходы	GAMMA	REAL	Значение гамма-функции

Функция **GAMMA** возвращает значение гамма-функции, вычисленное по аппроксимирующей формуле Герго Немеса:

$$\text{GAMMA}(X) = \sqrt{\frac{2\pi}{X}} \cdot \left[\frac{1}{e} \cdot \left(X + \frac{1}{12X - \frac{1}{10X}} \right) \right]$$

Область определения: $X > 0$

Функция может использоваться для приближенного вычисления факториалов чисел, что будет полезным в тех случаях, когда не может быть использована функция **FACT**. Следует помнить, что гамма-функция возвращает не факториал X, а факториал его декремента.

Пример


```

VAR
  x1 : REAL := 5.0;
  g1 : REAL;

  x2 : REAL := 6.0;
  g2 : REAL;
END_VAR

g1 := OSCAT_BASIC.GAMMA(X := x1);   (* g1 ≈ 24.0  (Γ(5) = 4! ) *)
g2 := OSCAT_BASIC.GAMMA(X := x2);   (* g2 ≈ 120.0 (Γ(6) = 5! ) *)

END_PROGRAM

```

8.31 GAUSS

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Аргумент
	U	REAL	Коэффициент сдвига
	SI	REAL	Коэффициент масштаба
Выходы	GAUSS	REAL	Плотность вероятности для распределения Гаусса

Функция **GAUSS** возвращает значение плотности вероятности для распределения Гаусса (нормального распределения), вычисленное по формуле

$$\text{GAUSS}(X) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{(X-\mu)^2}{2\sigma^2}}, \quad \sigma > 0$$

Пример

```

VAR
  x : REAL := 0.0;   (* аргумент *)
  u : REAL := 0.0;   (* среднее (сдвиг) *)
  si : REAL := 1.0;  (* σ (масштаб), > 0 *)
  p : REAL;
END_VAR

p := OSCAT_BASIC.GAUSS(X := x, U := u, SI := si);   (* p ≈ 0.3989423 *)

END_PROGRAM

```

8.32 GAUSSCD

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Аргумент
	U	REAL	Параметр сдвига
	SI	REAL	Параметр масштаба
Выходы	GAUSSCD	REAL	Функция распределения Гаусса
Используемые модули	ERF		

Функция **GAUSSCD** возвращает значение функции распределения Гаусса (нормального распределения), вычисленное по формуле

$$\text{GAUSSCD}(X) = \frac{1}{2} \left[1 + \text{ERF} \left(\frac{X - \mu}{\sigma\sqrt{2}} \right) \right], \quad U > 0$$

Пример

```

VAR
  x : REAL := 0.0;    (* аргумент *)
  u : REAL := 0.0;    (* среднее (сдвиг) *)
  si : REAL := 1.0;   (* σ (масштаб), > 0 *)
  f : REAL;
END_VAR

f := OSCAT_BASIC.GAUSSCD(X := x, U := u, SI := si);  (* f = 0.5 *)

END_PROGRAM

```

8.33 GCD

Тип модуля: функция	Переменная	Тип	Описание
Входы	A	DINT	1-ое значение
	B	DINT	2-ое значение
Выходы	GCD	INT	Наибольший общий делитель для A и B

Функция GCD возвращает значение наибольшего общего делителя для чисел A и B.

Пример

```

VAR
  a : DINT := 48;
  b : DINT := 18;
  g : INT;
END_VAR

g := OSCAT_BASIC.GCD(A := a, B := b);  (* g = 6 *)

END_PROGRAM

```

8.34 GDF

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Аргумент
Выходы	GDF	REAL	Значение функции Гудермана

Функция **GDF** возвращает значение функции Гудермана, вычисленное по формуле

$$GDF(X) = 2 \operatorname{atan}(e^X) - \frac{\pi}{2}$$

Область определения: $-\infty \leq X < +\infty$

Пример

```

VAR
  x : REAL := 5.0;
  y : REAL;
END_VAR

y := OSCAT_BASIC.GDF(X := x);  (* y = 1.55732048 *)

END_PROGRAM

```

8.35 GOLD

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Аргумент
Выходы	GOLD	REAL	Значение функции

$$\text{GOLD}(X) = \frac{X + \sqrt{X^2 + 4}}{2}$$

Функция **GOLD** возвращает значение, вычисленное по формуле

При X=1 функция возвращает значение золотого сечения.

Пример

```
VAR
  x : REAL := 1.0;
  y : REAL;
END_VAR

y := OSCAT_BASIC.GOLD(X := x);  (* y ≈ 1.6180339 *)

END_PROGRAM
```

8.36 HYPOT

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Длина катета X
	Y	REAL	Длина катета Y
Выходы	HYPOT	REAL	Длина гипотенузы

Функция **HYPOT** возвращает значение длины гипотенузы прямоугольного треугольника, вычисленное по

теореме Пифагора $\text{HYPOT} = \sqrt{X^2 + Y^2}$

Пример

```
VAR
  x : REAL := 3.0;
  y : REAL := 4.0;
  h : REAL;
END_VAR

h := OSCAT_BASIC.HYPOT(X := x, Y := y);  (* h = 5.0 *)

END_PROGRAM
```

8.37 INC

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	INT	Входное значение
	D	INT	Прибавляемое значение
	M	INT	Максимальное возвращаемое значение
Выходы	INC	INT	Возвращаемое значение

Функция **INC** складывает значения **X** и **D**, и проверяет результат. Если результат сложения превышает **M**, то функция возвращает **0**. Если результат сложения отрицателен, то функция возвращает **M**. Во всех остальных случаях функция возвращает сумму **X** и **D**. Функция может быть полезной при работе с массивами и буферами, а также абсолютными энкодерами.

Пример

```

VAR
  x1 : INT := 5;
  d1 : INT := 3;
  m : INT := 10;
  r1 : INT;

  x2 : INT := 9;
  d2 : INT := 3;
  r2 : INT;

  x3 : INT := 1;
  d3 : INT := -5;
  r3 : INT;
END_VAR

r1 := OSCAT_BASIC.INC(X := x1, D := d1, M := m); (* r1 = 8 *)
r2 := OSCAT_BASIC.INC(X := x2, D := d2, M := m); (* r2 = 0 (12 > 10) *)
r3 := OSCAT_BASIC.INC(X := x3, D := d3, M := m); (* r3 = 10 (-4 < 0) *)
END_PROGRAM

```

8.38 INC1

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	INT	Инкрементируемое значение 1
	N	INT	Инкрементируемое значение 2
Выходы	INC1	INT	Результат инкремента

Функция **INC1** инкрементирует значение переменной **X** до **N-1**, после чего на выход подается **0**, и продолжается циклический инкремент.

Пример

```

VAR
  x1 : INT := 3;
  n : INT := 5;
  r1 : INT;

  x2 : INT := 4;
  r2 : INT;
END_VAR

r1 := OSCAT_BASIC.INC1(X := x1, N := n); (* r1 = 4 *)
r2 := OSCAT_BASIC.INC1(X := x2, N := n); (* r2 = 0 (т.к. 4 -> 0 при N=5) *)
END_PROGRAM

```

8.39 INC2

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	INT	Входное значение
	D	INT	Прибавляемое значение
	L	INT	Нижний предел
	U	INT	Верхний предел
Выходы	INC2	INT	Возвращаемое значение

Функция **INC2** складывает значения **X** и **D**, и проверяет результат. Если результат сложения превышает **U**, то функция возвращает **L**. Если результат сложения меньше **L**, то функция возвращает **U**. Во всех остальных случаях функция возвращает сумму **X** и **D**. Функция может оказаться полезной при работе с массивами и буферами, а также абсолютными энкодерами.

Пример

```

VAR
  x1 : INT := 5;
  d1 : INT := 3;
  l  : INT := 0;
  u  : INT := 10;
  r1 : INT;

  x2 : INT := 9;
  d2 : INT := 3;
  r2 : INT;

  x3 : INT := 1;
  d3 : INT := -5;
  r3 : INT;
END_VAR

r1 := OSCAT_BASIC.INC2(X := x1, D := d1, L := l, U := u); (*r1 = 8*)
r2 := OSCAT_BASIC.INC2(X := x2, D := d2, L := l, U := u); (*r2 = 0 (12 > 10 -> L)*)
r3 := OSCAT_BASIC.INC2(X := x3, D := d3, L := l, U := u); (*r3 = 10 (-4 < 0 -> U)*)
END_PROGRAM

```

8.40 INV

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	INT	Аргумент
Выходы	INV	REAL	Обратное число

Функция **INV** возвращает обратное число для **X**.

$$\text{INV}(X) = \frac{1}{X}$$

Пример

```

VAR
  x : INT := 4;
  y : REAL;
END_VAR

y := OSCAT_BASIC.INV(X := x); (* y = 0.25 *)
END_PROGRAM

```

8.41 LAMBERT_W

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Аргумент
Выходы	LAMBERT_W	REAL	Значение W-функции Ламберта

Функция **LAMBERT_W** возвращает значение W-функции Ламберта, вычисленное по аппроксимационной формуле

$$\text{LAMBERT_W}(X) = \begin{cases} 0.665 \cdot (1 + 0.0195 \cdot \ln(X + 1)) \cdot \ln(X + 1) + 0.04, & 0 < X \leq 500 \\ \ln(X - 4) - \left(1 - \frac{1}{\ln(X)}\right) \cdot \ln(\ln(X)), & X > 500 \end{cases}$$

Относительная точность вычисления по данной формуле превышает 10%.

Пример

```
VAR
  x : REAL := 3.0;
  w : REAL;
END_VAR

w := OSCAT_BASIC.LAMBERT_W(X := x);  (* w ≈ 1.04990888 *)

END_PROGRAM
```

8.42 LANGEVIN

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Аргумент
Выходы	LANGEVIN	REAL	Значение функции Ланжевена
Используемые модули	COTH		

Функция **LANGEVIN** возвращает значение функции Ланжевена, вычисленное по формуле

$$\text{LANGEVIN}(X) = \text{COTH}(X) - \frac{1}{X}$$

Пример

```
VAR
  x : REAL := 5.0;
  y : REAL;
END_VAR

y := OSCAT_BASIC.LANGEVIN(X := x);  (* y ≈ 0.800090849 *)

END_PROGRAM
```

8.43 MAX3

Тип модуля: функция	Переменная	Тип	Описание
Входы	in1	REAL	Сравниваемое значение 1
	in2	REAL	Сравниваемое значение 2
	in3	REAL	Сравниваемое значение 3
Выходы	MAX3	REAL	Наибольшее из сравниваемых значений

Функция **MAX3** возвращает наибольшее из трех сравниваемых значений. Согласно стандарту МЭК 61131-3 оператор **MAX()** должен иметь настраиваемое число входов, но в некоторых средах разработки их число ограничено двумя – по этой причине в библиотеку **OSCAT** включена функция **MAX3**.

Пример

```

VAR
  a : REAL := 1.2;
  b : REAL := 5.6;
  c : REAL := 3.4;
  m : REAL;
END_VAR

m := OSCAT_BASIC.MAX3(in1 := a, in2 := b, in3 := c);  (* m = 5.6 *)

END_PROGRAM

```

8.44 MID3

Тип модуля: функция	Переменная	Тип	Описание
Входы	in1	REAL	Сравниваемое значение 1
	in2	REAL	Сравниваемое значение 2
	in3	REAL	Сравниваемое значение 3
Выходы	MID3	REAL	Среднее из сравниваемых значений

Функция **MID3** возвращает среднее (*но не усредненное*) из трех сравниваемых значений.

Пример

```

VAR
  a : REAL := 1.2;
  b : REAL := 5.6;
  c : REAL := 3.4;
  m : REAL;
END_VAR

m := OSCAT_BASIC.MID3(in1 := a, in2 := b, in3 := c);  (* m = 3.4 *)

END_PROGRAM

```

8.45 MIN3

Тип модуля: функция	Переменная	Тип	Описание
Входы	in1	REAL	Сравниваемое значение 1
	in2	REAL	Сравниваемое значение 2
	in3	REAL	Сравниваемое значение 3
Выходы	MIN3	REAL	Наименьшее из сравниваемых значений

Функция **MIN3** возвращает наименьшее из трех сравниваемых значений. Согласно стандарту МЭК 61131-3 оператор **MIN ()** должен иметь настраиваемое число входов, но в некоторых средах разработки их число ограничено двумя – по этой причине в библиотеку включена функция **MIN3**.

Пример

```

VAR
  a : REAL := 1.2;
  b : REAL := 5.6;
  c : REAL := 3.4;
  m : REAL;
END_VAR

m := OSCAT_BASIC.MIN3(in1 := a, in2 := b, in3 := c);  (* m = 1.2 *)

END_PROGRAM

```

8.46 MODR

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	REAL	Сравниваемое значение 1
	DIVI	REAL	Сравниваемое значение 2
Выходы	MODR	REAL	Наименьшее из сравниваемых значений
Используемые модули	FLOOR2		

Функция **MODR** возвращает остаток от деления значения **IN** на значение **DIVI**. В отличие от стандартного оператора **MOD** (), в качестве аргументов используются переменные типа **REAL**. При **DIVI=0** функция возвращает **0**.

Пример

```

VAR
  a1 : REAL := 5.5;
  b1 : REAL := 2.0;
  r1 : REAL;

  a2 : REAL := 5.5;
  b2 : REAL := 0.0;
  r2 : REAL;
END_VAR

r1 := OSCAT_BASIC.MODR(IN := a1, DIVI := b1);  (* r1 = 1.5 *)
r2 := OSCAT_BASIC.MODR(IN := a2, DIVI := b2);  (* r2 = 0.0 (DIVI=0) *)

END_PROGRAM

```

8.47 MUL_ADD

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Входное значение
	K	REAL	Коэффициент масштабирования
	O	REAL	Сдвиг
Выходы	MUL_ADD	REAL	Отмасштабированное значение

Функция **MUL_ADD** возвращает значение линейной функции с коэффициентом **K** и сдвигом **O** для аргумента **X**.

$$MUL_ADD = K \cdot X + O$$

Пример


```

VAR
  x : REAL := 10.0;
  k : REAL := 2.5;
  o : REAL := -3.0;
  y : REAL;
END_VAR

y := OSCAT_BASIC.MUL_ADD(X := x, K := k, O := o);  (* y = 22.0 *)

END_PROGRAM

```

8.48 NEGX

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Аргумент
Выходы	NEGX	REAL	Значение аргумента с противоположным знаком

Функция **NEGX** возвращает значение аргумента с противоположным знаком.

Пример

```

VAR
  x : REAL := 12.5;
  y : REAL;
END_VAR

y := OSCAT_BASIC.NEGX(X := x);  (* y = -12.5 *)

END_PROGRAM

```

8.49 RAD

Тип модуля: функция	Переменная	Тип	Описание
Входы	DEG	REAL	Значение угла в градусах
Выходы	RAD	REAL	Значение угла в радианах
Используемые модули	MODR, DEG		

Функция **RAD** преобразует значение угла в градусах в радианы. Если входная переменная **DEG** превышает 2π , то из нее будет вычитаться по π до тех пор, пока ее значение не окажется в интервале $0 < \text{DEG} < 2\pi$.

Пример

```

VAR
  d1 : REAL := 180.0;
  r1 : REAL;

  d2 : REAL := 450.0;  (* > 360°, будет приведено к диапазону 0..360° *)
  r2 : REAL;
END_VAR

r1 := OSCAT_BASIC.RAD(DEG := d1);  (* r1 ≈ 3.1415927 *)
r2 := OSCAT_BASIC.RAD(DEG := d2);  (* r2 ≈ 1.5707963 (т.к. 450° -> 90°) *)

END_PROGRAM

```

8.50 RDM

Тип модуля: функция	Переменная	Тип	Описание
Входы	last	REAL	Начальное значение (зерно)
Выходы	RDM	REAL	Псевдослучайное число
Используемые модули	T_PLC_MS, BIT_COUNT, FRACT		

Функция **RDM** используется для генерации псевдослучайных чисел с плавающей точкой в диапазоне **[0,1)**. В качестве источника энтропии используется системный таймер ПЛК. Поскольку модуль **RDM** представляет собой функцию, а не ФБ, он не содержит информацию о своем предыдущем вызове, и поэтому должен использоваться с осторожностью. Если функция вызывается более одного раза в течение цикла, то возвращает одинаковые числа, т.к. значение системного таймера остается тем же. Этого можно избежать, вызывая функцию с разным значением зерна **last**. В качестве зерна можно использовать значение функции, возвращенное при предыдущем вызове.

Пример

```

VAR
    seed : REAL := 0.12345; (* зерно (last) *)
    r1    : REAL;
    r2    : REAL;
END_VAR

(* Первый вызов: получаем псевдослучайное число в диапазоне [0,1) *)
r1 := OSCAT_BASIC.RDM(last := seed); (* 0.0 <= r1 < 1.0 *)

(* Обновляем зерно значением предыдущего вызова *)
seed := r1;

(* Следующий вызов (в следующем цикле ПЛК): новое число, используя seed *)
r2 := OSCAT_BASIC.RDM(last := seed); (* 0.0 <= r2 < 1.0, обычно r2 <> r1 *)

END_PROGRAM

```

8.51 RDM2

Тип модуля: функция	Переменная	Тип	Описание
Входы	last	DWORD	Начальное значение (зерно)
	low	INT	Нижний предел
	high	INT	Верхний предел
Выходы	RDM2	INT	Псевдослучайное число
Используемые модули	RDM, FRACT		

Функция **RDM2** используется для генерации псевдослучайных целых чисел в диапазоне **[low, high]**. В качестве источника энтропии используется системный таймер ПЛК. Поскольку модуль **RDM2** представляет собой функцию, а не ФБ, он не содержит информацию о своем предыдущем вызове, и поэтому должен использоваться с осторожностью. Если функция вызывается более одного раза в течение цикла, то возвращает одинаковые числа, т.к. значение системного таймера остается тем же. Этого можно избежать, вызывая функцию с разным значением зерна **last**. В качестве зерна можно использовать значение функции, возвращенное при предыдущем вызове.

Пример

```

VAR
  seed : INT := 12345;  (* зерно (last) *)
  low  : INT := 10;
  high : INT := 20;

  r1   : INT;
  r2   : INT;
END_VAR

(* Первый вызов: целое число в диапазоне [10..20] *)
r1 := OSCAT_BASIC.RDM2(last := seed, low := low, high := high);  (* 10 <= r1 <= 20 *)

(* Обновляем зерно значением предыдущего результата *)
seed := r1;

(* Следующий вызов (в следующем цикле ПЛК): новое число, используя seed *)
r2 := OSCAT_BASIC.RDM2(last := seed, low := low, high := high);  (* 10 <= r2 <= 20,
                                                                    обычно r2 <> r1 *)

END_PROGRAM

```

8.52 RDMDW

Тип модуля: функция	Переменная	Тип	Описание
Входы	last	REAL	Начальное значение (зерно)
Выходы	RDMDW	DWORD	Псевдослучайный набор бит
Используемые модули	RDM, T_PLC_MS, BIT_COUNT, FRACT		

Функция **RDMDW** используется для генерации псевдослучайных 32-битных наборов бит. В качестве источника энтропии используется системный таймер ПЛК. Поскольку модуль **RDMDW** представляет собой функцию, а не ФБ, он не содержит информацию о своем предыдущем вызове, и поэтому должен использоваться с осторожностью. Если функция вызывается более одного раза в течение цикла, то возвращает одинаковые числа, т.к. значение системного таймера остается тем же. Этого можно избежать, вызывая функцию с разным значением зерна **last**. В качестве зерна можно использовать значение функции, возвращенное при предыдущем вызове.

Пример

```

VAR
  seed : DWORD := 0.12345;  (* зерно (last) *)
  dw   : DWORD;
END_VAR

(* Получаем псевдослучайный 32-битный набор *)
dw := OSCAT_BASIC.RDMDW(last := seed);

(* В качестве зерна для следующего цикла можно использовать предыдущее значение seed,
   либо обновлять seed (например, значением OSCAT_BASIC.RDM(last := seed)) *)

END_PROGRAM

```

8.53 REAL_TO_FRAC

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Число с плавающей точкой
	N	INT	Максимальное значение знаменателя
Выходы	REAL_TO_FRAC	FRACTION	Число с плавающей точкой в виде дроби

Функция **REAL_TO_FRAC** представляет значение переменной **X** типа REAL в виде дроби, знаменатель которой не превышает **N**. Возвращаемое значение представляет собой структуру типа **FRACTION**.

Пример

```
VAR
  x : REAL := 0.75;
  n : INT  := 100;
  f : OSCAT_BASIC.FRACTION;
END_VAR

f := OSCAT_BASIC.REAL_TO_FRAC(X := x, N := n); (* ожидаемо: f = 3/4
                                                (знаменатель <= 100) *)

END_PROGRAM
```

8.54 RND

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Округляемое значение
	N	INT	Число значащих цифр
Выходы	RND	REAL	Округленное значение
Используемые модули	EXPN, CEIL		

Функция **RND** возвращает число с плавающей точкой **X**, округленное до **N** значащих цифр. Для чисел 0-4 округление выполняется в меньшую сторону, для 5-9 – в большую. Для округления до заданного количества десятичных знаков может использоваться функция **ROUND**.

Пример

```
VAR
  x1 : REAL := 1234.56;
  n1 : INT  := 3;
  y1 : REAL;

  x2 : REAL := 0.012345;
  n2 : INT  := 2;
  y2 : REAL;
END_VAR

y1 := OSCAT_BASIC.RND(X := x1, N := n1); (* y1 ≈ 1230.0 (3 значащие цифры) *)
y2 := OSCAT_BASIC.RND(X := x2, N := n2); (* y2 ≈ 0.012 (2 значащие цифры) *)

END_PROGRAM
```

8.55 ROUND

Тип модуля: функция	Переменная	Тип	Описание
Входы	in	REAL	Округляемое значение
	N	INT	Число знаков после запятой
Выходы	ROUND	REAL	Округленное значение

Функция **ROUND** возвращает число с плавающей точкой **X**, округленное до **N** знаков после запятой. Для чисел 0-4 округление выполняется в меньшую сторону, для 5-9 – в большую. Для округления до заданного количества значащих цифр может использоваться функция **RND**.

Пример

```

VAR
  x1 : REAL := 12.3456;
  n1 : INT  := 2;
  y1 : REAL;

  x2 : REAL := 12.344;
  n2 : INT  := 2;
  y2 : REAL;
END_VAR

y1 := OSCAT_BASIC.ROUND(in := x1, N := n1);  (* y1 = 12.35 *)
y2 := OSCAT_BASIC.ROUND(in := x2, N := n2);  (* y2 = 12.34 *)

END_PROGRAM

```

8.56 SGN

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Анализируемое значение
Выходы	SGN	INT	Код знака

Функция **SGN** возвращает код знака переменной **X**.

$$\begin{cases} \text{SGN}(X) = 1 & \text{при } X > 0 \\ \text{SGN}(X) = 0 & \text{при } X = 0 \\ \text{SGN}(X) = -1 & \text{при } X < 0 \end{cases}$$

Пример

```

VAR
  x1 : REAL := 10.0;
  x2 : REAL := 0.0;
  x3 : REAL := -2.5;

  s1 : INT;
  s2 : INT;
  s3 : INT;
END_VAR

s1 := OSCAT_BASIC.SGN(X := x1);  (* s1 = 1 *)
s2 := OSCAT_BASIC.SGN(X := x2);  (* s2 = 0 *)
s3 := OSCAT_BASIC.SGN(X := x3);  (* s3 = -1 *)

END_PROGRAM

```

8.57 SIGMOID

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Аргумент
Выходы	SIGMOID	REAL	Значение сигмоиды

Функция **SIGMOID** возвращает значение сигмоиды, вычисленное по формуле

$$\text{SIGMOID}(X) = \frac{1}{1 + e^{-X}}$$

Пример

```

VAR
  x : REAL := 0.5;
  y : REAL;

END_VAR

y := OSCAT_BASIC.SIGMOID(X := x);  (* y = 0.622459352 *)

END_PROGRAM

```

8.58 SIGN_I

Тип модуля: функция	Переменная	Тип	Описание
Входы	in	DINT	Анализируемое значение
Выходы	SIGN_I	BOOL	Флаг знака

Функция **SIGN_I** возвращает **TRUE**, если значение переменной **in** типа DINT является отрицательным, и **FALSE** – если положительным.

Пример

```

VAR
  a : DINT := -10;
  b : DINT := 10;
  r1 : BOOL;
  r2 : BOOL;
END_VAR

r1 := OSCAT_BASIC.SIGN_I(in := a);  (* r1 = TRUE *)
r2 := OSCAT_BASIC.SIGN_I(in := b);  (* r2 = FALSE *)

END_PROGRAM

```

8.59 SIGN_R

Тип модуля: функция	Переменная	Тип	Описание
Входы	in	REAL	Анализируемое значение
Выходы	SIGN_R	BOOL	Флаг знака

Функция **SIGN_R** возвращает **TRUE**, если значение переменной **in** типа REAL является отрицательным, и **FALSE** – если положительным.

Пример

```

VAR
  a : REAL := -10.0;
  b : REAL := 10.0;
  r1 : BOOL;
  r2 : BOOL;
END_VAR

r1 := OSCAT_BASIC.SIGN_R(in := a);  (* r1 = TRUE *)
r2 := OSCAT_BASIC.SIGN_R(in := b);  (* r2 = FALSE *)

END_PROGRAM

```

8.60 SINC

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Аргумент
Выходы	SINC	REAL	Значение кардинального синуса

Функция **SINC** возвращает значение ненормированной функции кардинального синуса, вычисленное по формуле

$$\begin{cases} \text{SINC}(X) = \frac{\text{SIN}(X)}{X} & X \neq 0 \\ 1 & X = 0 \end{cases}$$

Пример

```
VAR
  x : REAL := 0.5;
  y : REAL;
END_VAR

y := OSCAT_BASIC.SINC(X := x);    (* y = 0.9588511 *)

END_PROGRAM
```

8.61 SINH

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Аргумент
Выходы	SINH	REAL	Значение гиперболического синуса

Функция **SINH** возвращает значение гиперболического синуса, вычисленное по формуле

$$\text{SINH}(X) = \frac{e^X - e^{-X}}{2}$$

Область определения: $-\infty < X < +\infty$

Пример

```
VAR
  x : REAL := 0.5;
  y : REAL;
END_VAR

y := OSCAT_BASIC.SINH(X := x);    (* y ≈ 0.5210953 *)

END_PROGRAM
```

8.62 SQRTN

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Аргумент
	N	REAL	Степень корня
Выходы	SQRTN	REAL	Корень N-й степени

Функция **SQRTN** возвращает значение корня **N**-й степени из **X**.

Пример

```

VAR
  x : REAL := 27.0;
  n : REAL := 3.0;
  y : REAL;

END_VAR

y := OSCAT_BASIC.SQRTN(X := x, N := n);  (* y = 3.0 *)

END_PROGRAM

```

8.63 TANC

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Аргумент
Выходы	TANC	REAL	Значение функции

Функция **TANC** возвращает значение одноименной математической функции, вычисленное по формуле

$$\begin{cases} \text{TANC}(X) = \frac{\text{TANC}(X)}{X} & X \neq 0 \\ 1 & X = 0 \end{cases}$$

Пример

```

VAR
  x : REAL := 0.5;
  y : REAL;

END_VAR

y := OSCAT_BASIC.TANC(X := x);  (* y = 1.092605 *)

END_PROGRAM

```

8.64 TANH

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Аргумент
Выходы	TANH	REAL	Значение гиперболического тангенса

Функция **TANH** возвращает значение гиперболического тангенса, вычисленное по формуле

$$\text{TANH}(X) = 1 - \frac{2}{e^{2X} + 1}$$

Область определения: $-\infty < X < +\infty$

Пример


```

VAR
  x : REAL := 0.5;
  y : REAL;

END_VAR

y := OSCAT_BASIC.TANH(x := x);  (* y = 0.462117136 *)

END_PROGRAM

```

8.65 WINDOW

Тип модуля: функция	Переменная	Тип	Описание
Входы	low	REAL	Нижний предел
	in	REAL	Анализируемое значение
	high	REAL	Верхний предел
Выходы	WINDOW	BOOL	Флаг принадлежности интервалу

Функция **WINDOW** возвращает **TRUE**, если значение **X** принадлежит интервалу (**low**, **high**). В противном случае функция возвращает **FALSE**.

Пример

```

VAR
  low  : REAL := 0.5;
  high : REAL := 10.0;

  x1   : REAL := 5.0;
  x2   : REAL := 0.0;
  r1   : BOOL;
  r2   : BOOL;
END_VAR

r1 := OSCAT_BASIC.WINDOW(low := low, in := x1, high := high);  (* r1 = TRUE *)
r2 := OSCAT_BASIC.WINDOW(low := low, in := x2, high := high);  (* r2 = FALSE *)

END_PROGRAM

```

8.66 WINDOW2

Тип модуля: функция	Переменная	Тип	Описание
Входы	low	REAL	Нижний предел
	in	REAL	Анализируемое значение
	high	REAL	Верхний предел
Выходы	WINDOW2	BOOL	Флаг принадлежности интервалу

Функция **WINDOW2** возвращает **TRUE**, если значение **X** принадлежит интервалу [**low**, **high**]. В противном случае функция возвращает **FALSE**.

Пример

```
VAR
  low  : REAL := 0.0;
  high : REAL := 10.0;

  x1   : REAL := 0.0;
  x2   : REAL := 10.0;
  x3   : REAL := 10.1;

  r1   : BOOL;
  r2   : BOOL;
  r3   : BOOL;
END_VAR

r1 := OSCAT_BASIC.WINDOW2(low := low, in := x1, high := high); (* r1 = TRUE *)
r2 := OSCAT_BASIC.WINDOW2(low := low, in := x2, high := high); (* r2 = TRUE *)
r3 := OSCAT_BASIC.WINDOW2(low := low, in := x3, high := high); (* r3 = FALSE *)

END_PROGRAM
```

9 Работа с массивами

Функции, описанные в данной главе, используются для операций над массивами значений типа **REAL**. Входными переменными для каждой функции являются указатель на массив (**pt**) и размер массива (**size**). В большинстве случаев представляется удобным использовать операторы **ADR ()** и **SIZEOF ()**, которые возвращают соответственно адрес и размер массива в байтах. Тогда вызов функции для обработки массива **aArr** можно представить следующим образом:

```
VAR
  aArr   : ARRAY[0..9] OF REAL;    (* обрабатываемый массив *)
  rRes   : REAL;                   (* результат функции, если она возвращает REAL *)
END_VAR

(* Пример вызова: имя функции условное *)
rRes := OSCAT_BASIC.MyArrayFunc(
  pt   := ADR(aArr),               (* указатель на массив *)
  size := SIZEOF(aArr)             (* размер массива в байтах *)
);
END_PROGRAM
```

Максимальный размер обрабатываемого массива: **ARRAY [0..32000] OF REAL**.

Обработка массива, полученного по указателю, производится путем прямых манипуляций с памятью ПЛК. Этот тип обработки является крайне эффективным, так как не требует копирования содержимого массива.

9.1 _ARRAY_ABS

Тип модуля: функция	Переменная	Тип	Описание
Входы	pt	POINTER TO ARRAY [...] OF REAL	Указатель на массив
	size	UINT	Размер массива
Выходы	_ARRAY_ABS	BOOL	Флаг окончания обработки массива

Функция **_ARRAY_ABS** вычисляет модули элементов входного массива. Результаты записываются в тот же массив по указателю. После окончания обработки выход функции принимает значение **TRUE**.

Пример: массив [-2, 12, -4, 8, 6] после обработки функцией примет вид [2, 12, 4, 8, 6].

```
VAR
  a   : ARRAY[0..4] OF REAL := [-2.0, 12.0, -4.0, 8.0, 6.0];
  done: BOOL;
END_VAR

done := OSCAT_BASIC._ARRAY_ABS(
  pt   := ADR(a),
  size := SIZEOF(a)
); (* после обработки: a = [2.0, 12.0, 4.0, 8.0, 6.0], done = TRUE *)
END_PROGRAM
```

9.2 _ARRAY_ADD

Тип модуля: функция	Переменная	Тип	Описание
Входы	pt	POINTER TO ARRAY [...] OF REAL	Указатель на массив
	size	UINT	Размер массива
	X	REAL	Прибавляемое значение
Выходы	_ARRAY_ADD	BOOL	Флаг окончания обработки массива

Функция **_ARRAY_ADD** увеличивает значение каждого из элементов входного массива на заданное значение **X**. Результаты записываются в тот же массив по указателю. После окончания обработки выход функции принимает значение **TRUE**.

Пример: массив [-2, 12, -4, 8, 6] при **X=10** после однократной обработки функцией примет вид [8, 22, 6, 18, 16].

```
VAR
  a      : ARRAY[0..4] OF REAL := [-2.0, 12.0, -4.0, 8.0, 6.0];
  xAdd   : REAL := 10.0;
  done   : BOOL;
END_VAR

done := OSCAT_BASIC._ARRAY_ADD(
  pt    := ADR(a),
  size  := SIZEOF(a),
  X     := xAdd
); (* после обработки: a = [8.0, 22.0, 6.0, 18.0, 16.0], done = TRUE *)

END_PROGRAM
```

9.3 _ARRAY_INIT

Тип модуля: функция	Переменная	Тип	Описание
Входы	pt	POINTER TO ARRAY [...] OF REAL	Указатель на массив
	size	UINT	Размер массива
	init	REAL	Прибавляемое значение
Выходы	_ARRAY_INIT	BOOL	Флаг окончания обработки массива

Функция **_ARRAY_INIT** присваивает каждому из элементов входного массива заданное значение **init**. Результаты записываются в тот же массив по указателю. После окончания обработки выход функции принимает значение **TRUE**.

Пример: массив [-2, 12, -4, 8, 6] при **init=0** после обработки функцией примет вид [0, 0, 0, 0, 0].

```
VAR
  a      : ARRAY[0..4] OF REAL := [-2.0, 12.0, -4.0, 8.0, 6.0];
  init   : REAL := 0.0;
  done   : BOOL;
END_VAR

done := OSCAT_BASIC._ARRAY_INIT(
  pt    := ADR(a),
  size  := SIZEOF(a),
  init  := init
); (* после обработки: a = [0.0, 0.0, 0.0, 0.0, 0.0], done = TRUE *)

END_PROGRAM
```

9.4 _ARRAY_MEDIAN

Тип модуля: функция	Переменная	Тип	Описание
Входы	pt	POINTER TO ARRAY [...] OF REAL	Указатель на массив
	size	UINT	Размер массива
Выходы	_ARRAY_MEDIAN	REAL	Медиана массива
Используемые модули	EVEN		

Функция **_ARRAY_MEDIAN** возвращает медиану массива, а также сортирует его по возрастанию.

Пример: массив [-2, 12, -4, 8, 6] после обработки функцией примет вид [-4, -2, 6, 8, 12].

```
VAR
  a      : ARRAY[0..4] OF REAL := [-2.0, 12.0, -4.0, 8.0, 6.0];
  median : REAL;
END_VAR

median := OSCAT_BASIC._ARRAY_MEDIAN(
  pt  := ADR(a),
  size := SIZEOF(a)
); (* после обработки: a = [-4.0, -2.0, 6.0, 8.0, 12.0], median = 6.0 *)

END_PROGRAM
```

9.5 _ARRAY_MUL

Тип модуля: функция	Переменная	Тип	Описание
Входы	pt	POINTER TO ARRAY [...] OF REAL	Указатель на массив
	size	UINT	Размер массива
	X	REAL	Множитель
Выходы	_ARRAY_MUL	BOOL	Флаг окончания обработки массива

Функция **_ARRAY_MUL** умножает значение каждого из элементов входного массива на заданное значение **X**. Результаты записываются в тот же массив по указателю. После окончания обработки выход функции принимает значение **TRUE**.

Пример: массив [-2, 12, -4, 8, 6] при **X=3** после однократной обработки функцией примет вид [-6, 36, -12, 24, 18].

```
VAR
  a      : ARRAY[0..4] OF REAL := [-2.0, 12.0, -4.0, 8.0, 6.0];
  xMul   : REAL := 3.0;
  done   : BOOL;
END_VAR

done := OSCAT_BASIC._ARRAY_MUL(
  pt  := ADR(a),
  size := SIZEOF(a),
  X   := xMul
); (* после обработки: a = [-6.0, 36.0, -12.0, 24.0, 18.0], done = TRUE *)

END_PROGRAM
```

9.6 _ARRAY_SHUFFLE

Тип модуля: функция	Переменная	Тип	Описание
Входы	pt	POINTER TO ARRAY [...] OF REAL	Указатель на массив
	size	UINT	Размер массива
Выходы	_ARRAY_SHUFFLE	BOOL	Флаг окончания обработки массива
Используемые модули	RDM2		

Функция **_ARRAY_SHUFFLE** переставляет элементы входного массива местами в случайном порядке. Результаты записываются в тот же массив по указателю. После окончания обработки выход функции принимает значение **TRUE**.

Пример

```

VAR
  a      : ARRAY[0..4] OF REAL := [-2.0, 12.0, -4.0, 8.0, 6.0];
  done   : BOOL;
END_VAR

done := OSCAT_BASIC._ARRAY_SHUFFLE(
  pt    := ADR(a),
  size  := SIZEOF(a)
);

(* После обработки элементы массива a будут перемешаны в случайном порядке.
   Конкретный порядок заранее не фиксирован, done = TRUE. *)

END_PROGRAM

```

9.7 _ARRAY_SORT

Тип модуля: функция	Переменная	Тип	Описание
Входы	pt	POINTER TO ARRAY [...] OF REAL	Указатель на массив
	size	UINT	Размер массива
Выходы	_ARRAY_SORT	BOOL	Флаг окончания обработки массива

Функция **_ARRAY_SORT** сортирует входной массив, располагая его элементы в порядке возрастания. Результаты записываются в тот же массив по указателю. После окончания обработки выход функции принимает значение **TRUE**.

Пример: массив [-2, 12, -4, 8, 6] после обработки функцией примет вид [-4, -2, 6, 8, 12].

```

VAR
  a      : ARRAY[0..4] OF REAL := [-2.0, 12.0, -4.0, 8.0, 6.0];
  done   : BOOL;
END_VAR

done := OSCAT_BASIC._ARRAY_SORT(
  pt    := ADR(a),
  size  := SIZEOF(a)
); (* после обработки: a = [-4.0, -2.0, 6.0, 8.0, 12.0], done = TRUE *)

END_PROGRAM

```

9.8 ARRAY_AVG

Тип модуля: функция	Переменная	Тип	Описание
Входы	pt	POINTER TO ARRAY [...] OF REAL	Указатель на массив
	size	UINT	Размер массива
Выходы	ARRAY_AVG	REAL	Среднее арифметическое массива

Функция **ARRAY_AVG** возвращает среднее арифметическое входного массива. Функция не изменяет содержимое входного массива.

Пример: для массива [-2, 12, -4, 8, 6] функция вернет значение **4**.

```

VAR
  a   : ARRAY[0..4] OF REAL := [-2.0, 12.0, -4.0, 8.0, 6.0];
  avg : REAL;
END_VAR

avg := OSCAT_BASIC.ARRAY_AVG(
  pt  := ADR(a),
  size := SIZEOF(a)
); (* avg = 4.0, массив a не изменяется *)

END_PROGRAM

```

9.9 ARRAY_GAV

Тип модуля: функция	Переменная	Тип	Описание
Входы	pt	POINTER TO ARRAY [...] OF REAL	Указатель на массив
	size	UINT	Размер массива
Выходы	ARRAY_GAV	REAL	Среднее геометрическое массива
Используемые модули	SQRTN		

Функция **ARRAY_GAV** возвращает среднее геометрическое входного массива. Функция не изменяет содержимое входного массива.

Пример: для массива [2, 12, 4, 8, 6] функция вернет значение **5.40383959**.

```

VAR
  a   : ARRAY[0..4] OF REAL := [2.0, 12.0, 4.0, 8.0, 6.0];
  gav : REAL;
END_VAR

gav := OSCAT_BASIC.ARRAY_GAV(
  pt  := ADR(a),
  size := SIZEOF(a)
); (* gav ≈ 5.40383959, массив a не изменяется *)

END_PROGRAM

```

9.10 ARRAY_HAV

Тип модуля: функция	Переменная	Тип	Описание
Входы	pt	POINTER TO ARRAY [...] OF REAL	Указатель на массив
	size	UINT	Размер массива
Выходы	ARRAY_HAV	REAL	Среднее гармоническое массива

Функция **ARRAY_HAV** возвращает среднее гармоническое массива. Функция не изменяет содержимое массива.

Пример: для массива [-2, 12, -4, 8, 6] функция вернет значение **-13.333334**.

```

VAR
  a   : ARRAY[0..4] OF REAL := [-2.0, 12.0, -4.0, 8.0, 6.0];
  hav : REAL;
END_VAR

hav := OSCAT_BASIC.ARRAY_HAV(
  pt   := ADR(a),
  size := SIZEOF(a)
); (* hav ≈ -13.333334, массив a не изменяется *)

END_PROGRAM

```

9.11 ARRAY_MAX

Тип модуля: функция	Переменная	Тип	Описание
Входы	pt	POINTER TO ARRAY [...] OF REAL	Указатель на массив
	size	UINT	Размер массива
Выходы	ARRAY_MAX	REAL	Значение наибольшего элемента

Функция **ARRAY_MAX** возвращает значение наибольшего элемента входного массива. Функция не изменяет содержимое массива.

Пример: для массива [-2, 12, -4, 8, 6] функция вернет значение **12**.

```

VAR
  a     : ARRAY[0..4] OF REAL := [-2.0, 12.0, -4.0, 8.0, 6.0];
  aMax  : REAL;
END_VAR

aMax := OSCAT_BASIC.ARRAY_MAX(
  pt   := ADR(a),
  size := SIZEOF(a)
); (* aMax = 12.0, массив a не изменяется *)

END_PROGRAM

```

9.12 ARRAY_MIN

Тип модуля: функция	Переменная	Тип	Описание
Входы	pt	POINTER TO ARRAY [...] OF REAL	Указатель на массив
	size	UINT	Размер массива
Выходы	ARRAY_MIN	REAL	Значение наименьшего элемента

Функция **ARRAY_MIN** возвращает значение наименьшего элемента входного массива. Функция не изменяет содержимое массива.

Пример: для массива [-2, 12, -4, 8, 6] функция вернет значение **-4**.

```

VAR
  a     : ARRAY[0..4] OF REAL := [-2.0, 12.0, -4.0, 8.0, 6.0];
  aMin  : REAL;
END_VAR

aMin := OSCAT_BASIC.ARRAY_MIN(
  pt   := ADR(a),
  size := SIZEOF(a)
); (* aMin = -4.0, массив a не изменяется *)

END_PROGRAM

```


9.13 ARRAY_SDV

Тип модуля: функция	Переменная	Тип	Описание
Входы	pt	POINTER TO ARRAY [...] OF REAL	Указатель на массив
	size	UINT	Размер массива
Выходы	ARRAY_SDV	REAL	Среднеквадратичное отклонение
Используемые модули	ARRAY_VAR		

Функция **ARRAY_SDV** возвращает среднеквадратичное отклонение для входного массива. Функция не изменяет содержимое массива.

Пример: для массива [-2, 12, -4, 8, 6] функция вернет значение **6.78233**.

```
VAR
  a   : ARRAY[0..4] OF REAL := [-2.0, 12.0, -4.0, 8.0, 6.0];
  sdv : REAL;
END_VAR

sdv := OSCAT_BASIC.ARRAY_SDV(
  pt := ADR(a),
  size := SIZEOF(a)
); (* sdv ≈ 6.78233, массив a не изменяется *)

END_PROGRAM
```

9.14 ARRAY_SPR

Тип модуля: функция	Переменная	Тип	Описание
Входы	pt	POINTER TO ARRAY [...] OF REAL	Указатель на массив
	size	UINT	Размер массива
Выходы	ARRAY_SPR	REAL	Абсолютное отклонение
Используемые модули	ARRAY_MIN, ARRAY_MAX		

Функция **ARRAY_SPR** возвращает абсолютное отклонение для входного массива. Функция не изменяет содержимое массива.

Пример: для массива [-2, 12, -4, 8, 6] функция вернет значение 16 (полученное следующим образом: 12 - (-4)).

```
VAR
  a   : ARRAY[0..4] OF REAL := [-2.0, 12.0, -4.0, 8.0, 6.0];
  spr : REAL;
END_VAR

spr := OSCAT_BASIC.ARRAY_SPR(
  pt := ADR(a),
  size := SIZEOF(a)
); (* spr = 16.0, массив a не изменяется *)

END_PROGRAM
```

9.15 ARRAY_SUM

Тип модуля: функция	Переменная	Тип	Описание
Входы	pt	POINTER TO ARRAY [...] OF REAL	Указатель на массив
	size	UINT	Размер массива
Выходы	ARRAY_SUM	REAL	Сумма элементов массива

Функция **ARRAY_SUM** возвращает сумму элементов входного массива. Функция не изменяет содержимое массива.

Пример: для массива [-2, 12, -4, 8, 6] функция вернет значение **20**.

```
VAR
  a      : ARRAY[0..4] OF REAL := [-2.0, 12.0, -4.0, 8.0, 6.0];
  sum    : REAL;
END_VAR

sum := OSCAT_BASIC.ARRAY_SUM(
  pt := ADR(a),
  size := SIZEOF(a)
); (* sum = 20.0 *)

END_PROGRAM
```

9.16 ARRAY_TREND

Тип модуля: функция	Переменная	Тип	Описание
Входы	pt	POINTER TO ARRAY [...] OF REAL	Указатель на массив
	size	UINT	Размер массива
Выходы	ARRAY_TREND	REAL	Абсолютное отклонение
Используемые модули	EVEN		

Функция **ARRAY_TREND** возвращает значение тренда входного массива. Тренд представляет собой разность между средними значениями второй и первой половин массива. Если количество элементов в массиве является нечетным, то «центральный» элемент включается в обе половины. Функция не изменяет содержимое массива.

Пример: для массива [-2, 12, -4, 8, 6] функция вернет значение **1.33333337**.

$$\frac{-4 + 8 + 6}{3} - \frac{-2 + 12 - 4}{3} = \frac{4}{3} = 1.333(3)$$

```
VAR
  a      : ARRAY[0..4] OF REAL := [-2.0, 12.0, -4.0, 8.0, 6.0];
  trend  : REAL;
END_VAR

trend := OSCAT_BASIC.ARRAY_TREND(
  pt := ADR(a),
  size := SIZEOF(a)
); (* trend ≈ 1.33333337, массив a не изменяется *)

END_PROGRAM
```

9.17 ARRAY_VAR

Тип модуля: функция	Переменная	Тип	Описание
Входы	pt	POINTER TO ARRAY [...] OF REAL	Указатель на массив
	size	UINT	Размер массива
Выходы	ARRAY_VAR	REAL	Дисперсия

Функция **ARRAY_VAR** возвращает значение дисперсии входного массива. Функция не изменяет содержимое массива.

Пример: для массива [-2, 12, -4, 8, 6] функция вернет значение **46**.

```

VAR
  a      : ARRAY[0..4] OF REAL := [-2.0, 12.0, -4.0, 8.0, 6.0];
  xVar   : REAL;
END_VAR

xVar := OSCAT_BASIC.ARRAY_VAR(
  pt    := ADR(a),
  size  := SIZEOF(a)
); (* var = 46.0, массив a не изменяется *)

END_PROGRAM

```

9.18 IS_SORTED

Тип модуля: функция	Переменная	Тип	Описание
Входы	pt	POINTER TO ARRAY [...] OF REAL	Указатель на массив
	size	UINT	Размер массива
Выходы	IS_SORTED	BOOL	Флаг сортировки массива

Функция **IS_SORTED** проверяет, отсортированы ли элементы входного массива по возрастанию. Функция возвращает **TRUE**, если массив отсортирован; во всех остальных случаях функция возвращает **FALSE**.

Пример: для массива [-2, 12, -4, 8, 6] функция вернет **FALSE**.

```

VAR
  a      : ARRAY[0..4] OF REAL := [-2.0, 12.0, -4.0, 8.0, 6.0];
  sorted : BOOL;
END_VAR

sorted := OSCAT_BASIC.IS_SORTED(
  pt    := ADR(a),
  size  := SIZEOF(a)
); (* sorted = FALSE *)

END_PROGRAM

```

10 Комплексная арифметика

Комплексным числом называется выражение вида $x + i*y$, где x и y – любые действительные числа, а i – специальное число, которое называется мнимой единицей. Величины x и y называются **действительной** и **мнимой** частью комплексного числа. Комплексные числа нашли применение в ряде областей математики и физики (в частности, в расчетах цепей переменного тока). В библиотеке OSCAT комплексные числа представляют собой экземпляры структуры **COMPLEX**, которая состоит из двух переменных: **re** (целая часть комплексного числа) и **im** (мнимая часть действительного числа). В данной главе описаны функции, используемые для работы с комплексными числами.

10.1 CABS

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	COMPLEX	Комплексное число
Выходы	CABS	REAL	Модуль комплексного числа
Используемые модули	HYPOT		

Функция **CABS** возвращает модуль комплексного числа X, вычисленный по формуле

$$CABS = \sqrt{(X.re)^2 + (X.im)^2}$$

Пример

```
VAR
  z : OSCAT_BASIC.COMPLEX;   (* z = 2.5 + i*0.5 *)
  m : REAL;                  (* |z| *)
END_VAR

z.re := 2.5;
z.im := 0.5;

m := OSCAT_BASIC.CABS(X := z); (* m ≈ 2.5495098 *)

END_PROGRAM
```

10.2 CACOS

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	COMPLEX	Комплексное число
Выходы	CACOS	COMPLEX	Арккосинус комплексного числа
Используемые модули	CACOSH		

Функция **CACOS** возвращает арккосинус комплексного числа X. Действительная часть полученного результата принадлежит интервалу $[0, \pi]$, мнимая принадлежит интервалу $[-\pi, \pi]$.

Пример

```
VAR
  z : OSCAT_BASIC.COMPLEX;   (* z = 2.5 + i*0.5 *)
  res : OSCAT_BASIC.COMPLEX; (* res = CACOS(z) *)
END_VAR

z.re := 2.5;
z.im := 0.5;

res := OSCAT_BASIC.CACOS(X := z); (* результат комплексный: res.re ≈ 0.214,
                                   res.im ≈ -1.59*)

END_PROGRAM
```

10.3 CACOSH

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	COMPLEX	Комплексное число
Выходы	CACOSH	COMPLEX	Гиперболический арккосинус комплексного числа
Используемые модули	CSQRT, CLOG		

Функция **CACOSH** возвращает гиперболический арккосинус комплексного числа **X**. Мнимая часть полученного результата принадлежит интервалу $[-\pi, \pi]$.

Пример

```

VAR
  z      : OSCAT_BASIC.COMPLEX;    (* z = 2.5 + i*0.5 *)
  res    : OSCAT_BASIC.COMPLEX;    (* res = CACOSH(z) *)
END_VAR

z.re := 2.5;
z.im := 0.5;

res := OSCAT_BASIC.CACOSH(X := z);  (* результат комплексный: res.re ≈ 1.59,
                                     res.im ≈ 0.214*)

END_PROGRAM

```

10.4 CADD

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	COMPLEX	Комплексное число
	Y	COMPLEX	Комплексное число
Выходы	CADD	COMPLEX	Сумма комплексных чисел

Функция **CADD** возвращает сумму комплексных чисел **X** и **Y**.

Пример

```

VAR
  x      : OSCAT_BASIC.COMPLEX := (re := 2.5, im := 0.5);
  y      : OSCAT_BASIC.COMPLEX := (re := 4.5, im := 1.5);
  res    : OSCAT_BASIC.COMPLEX;
  res_re : REAL;
  res_im : REAL;
END_VAR

res := OSCAT_BASIC.CADD(X := x, Y := y);  (* res.re = 7.0, res.im = 2.0*)
res_re := res.re;
res_im := res.im;

END_PROGRAM

```

10.5 CARG

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	COMPLEX	Комплексное число
Выходы	CARG	REAL	Угол между осью действительных чисел и радиус-вектором геометрического представления комплексного числа X в радианах
Используемые модули	ATAN2		

Функция **CARG** возвращает значение угла (в радианах) между осью действительных чисел и радиус-вектором геометрического представления комплексного числа **X**.

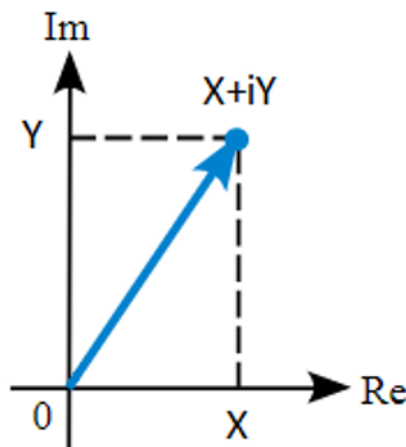


Рисунок 10.1 – Геометрическое представление комплексного числа

Пример

```

VAR
  z      : OSCAT_BASIC.COMPLEX;  (* z = 2.5 + i*0.5 *)
  ang    : REAL;                 (* аргумент z, рад *)
END_VAR

z.re := 2.5;
z.im := 0.5;

ang := OSCAT_BASIC.CARG(X := z);  (* ang = atan2(0.5, 2.5) ≈ 0.19739556 рад *)
END_PROGRAM

```

10.6 CASIN

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	COMPLEX	Комплексное число
Выходы	CASIN	COMPLEX	Арксинус комплексного числа
Используемые модули	CASINH		

Функция **CASIN** возвращает арксинус комплексного числа **X**. Мнимая часть полученного результата

принадлежит интервалу $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$.

Пример

```

VAR
  z      : OSCAT_BASIC.COMPLEX;    (* z = 2.5 + i*0.5 *)
  res    : OSCAT_BASIC.COMPLEX;    (* результат CASIN(z) *)
END_VAR

z.re := 2.5;
z.im := 0.5;

res := OSCAT_BASIC.CASIN(X := z);

END_PROGRAM

```

10.7 CASINH

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	COMPLEX	Комплексное число
Выходы	CASINH	COMPLEX	Гиперболический арксинус комплексного числа

Функция **CASINH** возвращает гиперболический арксинус комплексного числа **X**. Мнимая часть полученного результата принадлежит интервалу $[-\pi, \pi]$.

Пример

```

VAR
  z      : OSCAT_BASIC.COMPLEX;    (* входное комплексное число *)
  res    : OSCAT_BASIC.COMPLEX;    (* результат CASINH(z) *)
END_VAR

z.re := 2.5;
z.im := 0.5;

res := OSCAT_BASIC.CASINH(X := z);  (* res.re ≈ 1.66, res.im ≈ 0.184 *)

END_PROGRAM

```

10.8 CATAN

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	COMPLEX	Комплексное число
Выходы	CATAN	COMPLEX	Арктангенс комплексного числа

Функция **CATAN** возвращает арктангенс комплексного числа **X**. Мнимая часть полученного результата

принадлежит интервалу $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$.

Пример

```

VAR
  z      : OSCAT_BASIC.COMPLEX;    (* z = 2.5 + i*0.5 *)
  res    : OSCAT_BASIC.COMPLEX;    (* res = CATAN(z) *)
END_VAR

z.re := 2.5;
z.im := 0.5;

res := OSCAT_BASIC.CATAN(X := z);  (* результат комплексный res.re ≈ -0.369,
                                     res.im ≈ 0.0671 *)

END_PROGRAM

```

10.9 CATANH

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	COMPLEX	Комплексное число
Выходы	CATANH	COMPLEX	Гиперболический арктангенс комплексного числа

Функция **CATANH** возвращает гиперболический арктангенс комплексного числа **X**. Мнимая часть полученного

результата принадлежит интервалу $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$.

Пример

```

VAR
  z      : OSCAT_BASIC.COMPLEX;    (* z = 2.5 + i*0.5 *)
  res    : OSCAT_BASIC.COMPLEX;    (* res = CATANH(z) *)
END_VAR

z.re := 2.5;
z.im := 0.5;

res := OSCAT_BASIC.CATANH(X := z); (* результат комплексный res.re ≈ 0.402,
                                     res.im ≈ -0.0899 *)

END_PROGRAM

```

10.10 CCON

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	COMPLEX	Комплексное число
Выходы	CCON	COMPLEX	Сопряженное число

Функция **CCON** возвращает сопряженное число для комплексного числа **X**.

$$\begin{cases} \text{CCON.RE} = \text{X.RE} \\ \text{CCON.IM} = -\text{X.IM} \end{cases}$$

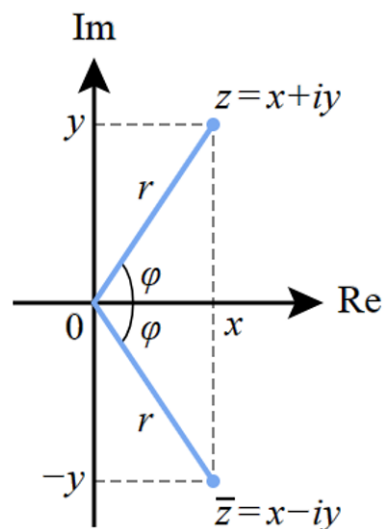


Рисунок 10.2 – Геометрическое представление сопряженных чисел

Пример


```

VAR
  z      : OSCAT_BASIC.COMPLEX;
  conj   : OSCAT_BASIC.COMPLEX;
END_VAR

z.re := 2.5;
z.im := 0.5;

conj := OSCAT_BASIC.CCON(X := z);  (* conj = 2.5 + i*(-0.5) *)

END_PROGRAM

```

10.11 CCOS

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	COMPLEX	Комплексное число
Выходы	CCOS	COMPLEX	Косинус комплексного числа
Используемые модули	CCOSH, CSET		

Функция **CCOS** возвращает косинус комплексного числа **X**.

Пример

```

VAR
  z      : OSCAT_BASIC.COMPLEX;  (* z = 2.5 + i*0.5 *)
  res    : OSCAT_BASIC.COMPLEX;  (* res = cos(z) *)
END_VAR

z.re := 2.5;
z.im := 0.5;

res := OSCAT_BASIC.CCOS(X := z);  (* результат комплексный res.re ≈ -0.903,
                                   res.im ≈ -0.312 *)

END_PROGRAM

```

10.12 CCOSH

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	COMPLEX	Комплексное число
Выходы	CCOSH	COMPLEX	Гиперболический косинус комплексного числа
Используемые модули	COSH, SINH		

Функция **CCOSH** возвращает гиперболический косинус комплексного числа **X**.

Пример

```

VAR
  z      : OSCAT_BASIC.COMPLEX;  (* z = 2.5 + i*0.5 *)
  res    : OSCAT_BASIC.COMPLEX;  (* res = cosh(z) *)
END_VAR

z.re := 2.5;
z.im := 0.5;

res := OSCAT_BASIC.CCOSH(X := z);  (* результат комплексный res.re ≈ 5.38,
                                   res.im ≈ 2.9 *)

END_PROGRAM

```

10.13 CDIV

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	COMPLEX	Комплексное число
	Y	COMPLEX	Комплексное число
Выходы	CDIV	COMPLEX	Частное комплексных чисел

Функция **CDIV** возвращает частное комплексных чисел **X** и **Y**.

Пример

```
VAR
  X  : OSCAT_BASIC.COMPLEX := (Re := 2.5, Im := 0.5);
  Y  : OSCAT_BASIC.COMPLEX := (Re := 4.5, Im := 1.5);
  res : OSCAT_BASIC.COMPLEX;  (* Результат деления *)
END_VAR

(* Вызов функции CDIV для деления X на Y *)
res := OSCAT_BASIC.CDIV(X := X, Y := Y); (*res.re ≈ 0.533, res.im ≈ -0.0667 *)

END_PROGRAM
```

10.14 CEXP

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	COMPLEX	Комплексное число
Выходы	CEXP	COMPLEX	Экспонента комплексного числа

Функция **CEXP** возвращает значение экспоненты комплексного числа **X**.

Пример

```
VAR
  x : OSCAT_BASIC.COMPLEX;  (* входное комплексное число *)
  y : OSCAT_BASIC.COMPLEX;  (* результат вычисления экспоненты *)
END_VAR

x.re := 2.5;
x.im := 0.5;

y := OSCAT_BASIC.CEXP(x := x); (* y.re ≈ 10.691, y.im ≈ 5.84 *)

END_PROGRAM
```

10.15 CINV

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	COMPLEX	Комплексное число
Выходы	CINV	COMPLEX	Обратное комплексное число

Функция **CINV** возвращает обратное число для комплексного числа **X**.

Пример

```

VAR
  x : OSCAT_BASIC.COMPLEX;  (* входное комплексное число *)
  y : OSCAT_BASIC.COMPLEX;  (* результат *)
END_VAR

x.re := 2.5;
x.im := 0.5;

y := OSCAT_BASIC.CINV(x := x);  (* y.re ≈ 0.385, y.im ≈ -0.0769 *)

END_PROGRAM

```

10.16 CLOG

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	COMPLEX	Комплексное число
Выходы	CLOG	COMPLEX	Натуральный логарифм комплексного числа
Используемые модули	HYPOT, ATAN2		

Функция **CLOG** возвращает значение натурального логарифма для комплексного числа **X**.

Пример

```

VAR
  x : OSCAT_BASIC.COMPLEX;  (* входное комплексное число *)
  y : OSCAT_BASIC.COMPLEX;  (* значение натурального логарифма *)
END_VAR

x.re := 2.5;
x.im := 0.5;

y := OSCAT_BASIC.CLOG(x := x);  (* y.re ≈ 0.936, y.im ≈ 0.197 *)

END_PROGRAM

```

10.17 CMUL

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	COMPLEX	Комплексное число
	Y	COMPLEX	Комплексное число
Выходы	CMUL	COMPLEX	Произведение комплексных чисел

Функция **CMUL** возвращает произведение комплексных чисел **X** и **Y**.

Пример

```

VAR
  x : OSCAT_BASIC.COMPLEX := (re := 2.5, im := 0.5);
  y : OSCAT_BASIC.COMPLEX := (re := 4.5, im := 1.5);
  res : OSCAT_BASIC.COMPLEX;
  res_re : REAL;
  res_im : REAL;
END_VAR

res := OSCAT_BASIC.CMUL(X := x, Y := y);  (* res.re = 10.5, res.im = 6.0 *)
res_re := res.re;
res_im := res.im;

END_PROGRAM

```

10.18 CPOL

Тип модуля: функция	Переменная	Тип	Описание
Входы	L	REAL	Радиус-вектор
	A	REAL	Угол между осью действительных чисел и радиус-вектором в радианах
Выходы	CPOL	COMPLEX	Комплексное число

Функция **CPOL** возвращает комплексное число, представленное в геометрическом виде радиус-вектором **L** и углом (в радианах) между осью действительных чисел и радиус-вектором **A**.

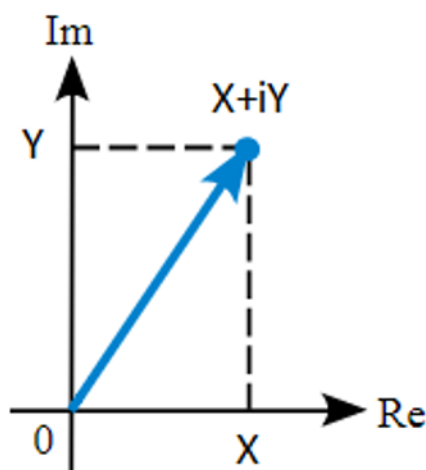


Рисунок 10.3 – Геометрическое представление комплексного числа

```

VAR
  L : REAL := 5.0; (* радиус-вектор*)
  A : REAL := 0.785; (* угол в радианах (≈ 45 градусов)*)
  res : OSCAT_BASIC.COMPLEX;
END_VAR

(* вызов функции для преобразования полярных координат в комплексное число*)
res := OSCAT_BASIC.CPOL(L := L, A := A); (* res ≈ 3.5355 + 3.5355i*)

END_PROGRAM

```

10.19 CPOW

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	COMPLEX	Комплексное число
	Y	COMPLEX	Показатель степени комплексного числа
Выходы	CPOW	COMPLEX	Комплексное число, возведенное в степень
Используемые модули	CEXP, CMUL, CLOG		

Функция **CPOW** возвращает комплексное число **X**, возведенное в степень **Y**. Показатель степени **Y** также является комплексным числом.

Пример

```

VAR
  x   : OSCAT_BASIC.COMPLEX := (re := 2.5, im := 0.5);
  y   : OSCAT_BASIC.COMPLEX := (re := 4.5, im := 1.5);
  res : OSCAT_BASIC.COMPLEX;
  res_re : REAL;
  res_im : REAL;
END_VAR

res := OSCAT_BASIC.CPOW(X := x, Y := y);  (* res.re ≈ -33.1, res.im ≈ 37.7*)
res_re := res.re;
res_im := res.im;

END_PROGRAM

```

10.20 CSET

Тип модуля: функция	Переменная	Тип	Описание
Входы	RE	REAL	Действительная часть комплексного числа
	IM	REAL	Мнимая часть комплексного числа
Выходы	CSET	COMPLEX	Комплексное число

Функция **CSET** возвращает комплексное число с действительной частью **X** и мнимой частью **Y**.

```

VAR
  RE : REAL := 2.5; (* радиус-вектор*)
  IM : REAL := 0.5; (* угол в радианах (≈ 45 градусов)*)
  res : OSCAT_BASIC.COMPLEX;
END_VAR

(* вызов функции для преобразования вещественных чисел в комплексное число*)
res := OSCAT_BASIC.CSET(RE := RE, IM := IM);  (* res = 2.5 + 0.5i*)

END_PROGRAM

```

10.21 CSIN

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	COMPLEX	Комплексное число
Выходы	CSIN	COMPLEX	Синус комплексного числа
Используемые модули	CCOSH, CSINH		

Функция **CSIN** возвращает синус комплексного числа **X**.

Пример

```

VAR
  z   : OSCAT_BASIC.COMPLEX;  (* z = 2.5 + i*0.5 *)
  res : OSCAT_BASIC.COMPLEX;  (* res = sin(z) *)
END_VAR

z.re := 2.5;
z.im := 0.5;

res := OSCAT_BASIC.CSIN(X := z);  (* результат комплексный res.re ≈ 0.675,
                                   res.im ≈ -0.417 *)

END_PROGRAM

```

10.22 CSINH

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	COMPLEX	Комплексное число
Выходы	CSINH	COMPLEX	Гиперболический синус комплексного числа
Используемые модули	COSH, SINH		

Функция **CSINH** возвращает гиперболический синус комплексного числа **X**.

Пример

```

VAR
  z      : OSCAT_BASIC.COMPLEX;    (* z = 2.5 + i*0.5 *)
  res    : OSCAT_BASIC.COMPLEX;    (* res = CSINH(z) *)
END_VAR

z.re := 2.5;
z.im := 0.5;

res := OSCAT_BASIC.CSINH(X := z);  (* результат комплексный: res.re ≈ 5.31,
                                   res.im ≈ 2.94 *)

END_PROGRAM

```

10.23 CSQRT

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	COMPLEX	Комплексное число
Выходы	CSQRT	COMPLEX	Квадратный корень комплексного числа
Используемые модули	HYPOT, SGN		

Функция **CSQRT** возвращает квадратный корень комплексного числа **X**. Пример

```

VAR
  z      : OSCAT_BASIC.COMPLEX;    (* z = 2.5 + i*0.5 *)
  res    : OSCAT_BASIC.COMPLEX;    (* квадратный корень комплексного числа *)
END_VAR

z.re := 2.5;
z.im := 0.5;

res := OSCAT_BASIC.CSQRT(X := z);  (* результат комплексный: res.re ≈ 1.59,
                                   res.im ≈ 0.157 *)

END_PROGRAM

```

10.24 CSUB

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	COMPLEX	Комплексное число
	Y	COMPLEX	Комплексное число
Выходы	CSUB	COMPLEX	Разность комплексных чисел

Функция **CSUB** возвращает разность комплексных чисел **X** и **Y**.

Пример

```

VAR
  x   : OSCAT_BASIC.COMPLEX := (re := 2.5, im := 0.5);
  y   : OSCAT_BASIC.COMPLEX := (re := 4.5, im := 1.5);
  res : OSCAT_BASIC.COMPLEX;
  res_re : REAL;
  res_im : REAL;
END_VAR

res := OSCAT_BASIC.CSUB(X := x, Y := y);  (* res.re = -2.0, res.im = -1.0*)
res_re := res.re;
res_im := res.im;

END_PROGRAM

```

10.25 CTAN

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	COMPLEX	Комплексное число
Выходы	CTAN	COMPLEX	Тангенс комплексного числа
Используемые модули	COSH, SINH		

Функция **CTAN** возвращает тангенс комплексного числа **X**.

Пример

```

VAR
  z   : OSCAT_BASIC.COMPLEX;  (* z = 2.5 + i*0.5 *)
  res : OSCAT_BASIC.COMPLEX;  (* res = tan(z) *)
END_VAR

z.re := 2.5;
z.im := 0.5;

res := OSCAT_BASIC.CTAN(X := z);  (* результат комплексный res.re ≈ -0.525,
                                   res.im ≈ 0.643 *)

END_PROGRAM

```

10.26 CTANH

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	COMPLEX	Комплексное число
Выходы	CTANH	COMPLEX	Гиперболический тангенс комплексного числа
Используемые модули	COSH, SINH		

Функция **CTANH** возвращает гиперболический тангенс комплексного числа **X**.

Пример

```

VAR
  z   : OSCAT_BASIC.COMPLEX;  (* z = 2.5 + i*0.5 *)
  res : OSCAT_BASIC.COMPLEX;  (* res = CTANH(z) *)
END_VAR

z.re := 2.5;
z.im := 0.5;

res := OSCAT_BASIC.CTANH(X := z);  (* результат комплексный res.re ≈ 0.993,
                                   res.im ≈ 0.0113 *)

END_PROGRAM

```

11 Арифметика чисел двойной точности

В соответствии со стандартом МЭК 61131-3 числа с плавающей точкой хранятся в ПЛК в виде переменных типа **REAL**. Переменные типа **REAL** соответствуют числам одинарной точности стандарта IEEE 754 и занимают **32 бита**. Они обеспечивают точность в 7-8 десятичных цифр в диапазоне от 10^{-39} до 10^{38} . Для большинства задач этого оказывается достаточно, но в отдельных ситуациях может потребоваться повышенная точность вычислений. МЭК 61131-3 предусматривает тип **LREAL**, переменные которого занимают **64 бита** и соответствуют числам двойной точности стандарта IEEE 754, но поддержка данного типа реализована не на всех ПЛК. Для подобных случаев библиотека **OSCAT** предоставляет тип данных **REAL2**, эмулирующий поддержку чисел двойной точности. **REAL2** представляет собой структуру из двух переменных типа **REAL**, одна из которых – **RX** – содержит «грубую» часть значения (в нее входит целая часть и 7-8 знаков после запятой), а вторая – **R1** – «точную» (содержит оставшиеся знаки после запятой). При этом каждая операция над подобной структурой может привести к потере точности. В данной главе описаны функции, которые применяются для работы с этой структурой.

11.1 R2_ABS

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL2	Входное значение
Выходы	R2_ABS	REAL2	Значение по модулю

Функция **R2_ABS** возвращает модуль числа двойной точности **X** типа **REAL2**.

Пример

```

VAR
  x      : OSCAT_BASIC.REAL2;    (* исходное значение REAL2 *)
  x_abs  : OSCAT_BASIC.REAL2;    (* модуль x *)
END_VAR

(* Пример задания отрицательного значения *)
x.RX := -12.345678;              (* «грубая» часть *)
x.R1 := -0.0000000009876;       (* «точная» часть *)

(* Вызов функции модуля *)
x_abs := OSCAT_BASIC.R2_ABS(x);  (* «грубая» часть = 12.345678,
                                   «точная» часть = 0.0000000009876 *)

END_PROGRAM

```

11.2 R2_ADD

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL2	Входное значение
	Y	REAL	Входное значение
Выходы	R2_ADD	REAL2	Сумма X и Y

Функция **R2_ADD** возвращает сумму числа двойной точности **X** типа **REAL2** и значения **Y** типа **REAL**.

Пример


```

VAR
  X   : OSCAT_BASIC.REAL2;  (* число двойной точности *)
  Y   : REAL;               (* слагаемое REAL *)
  S   : OSCAT_BASIC.REAL2;  (* результат: X + Y *)
END_VAR

(* Пример: X = 1000.123456789 *)
X.RX := 1000.123456;  (* «грубая» часть *)
X.R1 := 0.000000789; (* «точная» часть *)

Y := 2.5;

S := OSCAT_BASIC.R2_ADD(X := X, Y := Y); (* Сумма X и Y = 1002.623456789 *)
END_PROGRAM

```

11.3 R2_ADD2

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL2	Входное значение
	Y	REAL2	Входное значение
Выходы	R2_ADD2	REAL2	Сумма X и Y

Функция **R2_ADD2** возвращает сумму чисел двойной точности **X** и **Y** типа **REAL2**.

Пример

```

VAR
  X   : OSCAT_BASIC.REAL2;  (* первое слагаемое *)
  Y   : OSCAT_BASIC.REAL2;  (* второе слагаемое *)
  S   : OSCAT_BASIC.REAL2;  (* результат: X + Y *)
END_VAR

(* X = 1000.123456789 *)
X.RX := 1000.123456;  (* «грубая» часть *)
X.R1 := 0.000000789; (* «точная» часть *)

(* Y = 2.500000111 *)
Y.RX := 2.5;  (* «грубая» часть *)
Y.R1 := 0.000000111; (* «точная» часть *)

S := OSCAT_BASIC.R2_ADD2(X := X, Y := Y); (* Сумма X и Y = 1002.623456900 *)
END_PROGRAM

```

11.4 R2_MUL

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL2	Входное значение
	Y	REAL	Входное значение
Выходы	R2_MUL	REAL2	Произведение X и Y

Функция **R2_MUL** возвращает произведение числа двойной точности **X** типа **REAL2** и значения **Y** типа **REAL**.

Пример

```

VAR
  X   : OSCAT_BASIC.REAL2;    (* первый множитель *)
  Y   : REAL;                 (* второй множитель *)
  S   : OSCAT_BASIC.REAL2;    (* результат: X * Y *)
END_VAR

(* Пример: X = 12.345678901 *)
X.RX := 12.345678;  (* «грубая» часть *)
X.R1 := 0.000000901; (* «точная» часть *)

Y := 3.0;

S := OSCAT_BASIC.R2_MUL(X := X, Y := Y); (* Произведение X и Y = 37.037036703 *)
END_PROGRAM

```

11.5 R2_SET

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Входное значение
Выходы	R2_SET	REAL2	Выходное значение

Функция **R2_SET** присваивает значение входной переменной **X** типа **REAL** выходной переменной двойной точности **R2_SET** типа **REAL2**.

Пример

```

VAR
  X   : REAL := 123.456;
  R2  : OSCAT_BASIC.REAL2;    (* результат преобразования REAL -> REAL2 *)
END_VAR

(* Присвоить значение REAL переменной типа REAL2 *)
R2 := OSCAT_BASIC.R2_SET(X := X);

(* После преобразования:
  R2.RX = 123.456
  R2.R1 = 0.0
*)
END_PROGRAM

```

12 Арифметические функции

12.1 F_LIN

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Аргумент
	A	REAL	Угловой коэффициент
	B	REAL	Коэффициент сдвига
Выходы	F_LIN	REAL	Значение линейной функции

Функция **F_LIN** возвращает для аргумента **X** значение линейной функции, вычисленное по формуле

$$F_LIN(X) = A \cdot X + B$$

Пример

```

VAR
  X : REAL := 2.0;      (* аргумент *)
  A : REAL := 1.5;      (* угловой коэффициент *)
  B : REAL := -3.0;     (* коэффициент сдвига *)
  Y : REAL;             (* результат *)
END_VAR

(* F_LIN(X) = A * X + B = 1.5 * 2.0 - 3.0 = 0.0 *)
Y := OSCAT_BASIC.F_LIN(X := X, A := A, B := B);
END_PROGRAM

```

12.2 F_LIN2

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Аргумент
	X1	REAL	Координата X для точки 1
	Y1	REAL	Координата Y для точки 1
	X2	REAL	Координата X для точки 2
	Y2	REAL	Координата Y для точки 2
Выходы	F_LIN2	REAL	Линейно интерполированное значение Y для X

Функция **F_LIN2** возвращает для точки с координатой **X** результат линейной интерполяции координаты **Y** между узловыми точками (**X1, Y1**) и (**X2, Y2**). Если значение **X** не входит в диапазон (**X1...X2**), то значение **Y** (**F_LIN2**) экстраполируется.

Пример

```

VAR
  X : REAL := 5.0;      (* точка, для которой ищем Y *)
  X1 : REAL := 0.0;
  Y1 : REAL := 0.0;
  X2 : REAL := 10.0;
  Y2 : REAL := 100.0;
  Y : REAL;             (* результат *)
END_VAR

(* Линейная интерполяция между (0,0) и (10,100):
   при X = 5 получаем Y = 50 *)
Y := OSCAT_BASIC.F_LIN2(X := X, X1 := X1, Y1 := Y1, X2 := X2, Y2 := Y2);
END_PROGRAM

```

12.3 F_POLY

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Аргумент
	C	ARRAY [0..7] OF REAL	Коэффициенты полинома
Выходы	F_POLY	REAL	Значение полинома 7-ой степени

Функция **F_POLY** возвращает значение полинома 7-ой степени с коэффициентами **C** (где **C** – массив из 8-ми значений) для аргумента **X**, вычисленное по формуле

$$F_POLY(X) = \sum_{i=0}^7 C[i] \cdot X^i$$

Пример

```

VAR
  X : REAL := 1.5;
  C : ARRAY [0..7] OF REAL := [1.0, 1.0, 1.0, 1.0, 2.0, 2.0, 3.0, 3.0];
  Y : REAL;
END_VAR

(* Полином:
   Y = C0 + C1*X + C2*X^2 + C3*X^3 + C4*X^4 + C5*X^5 + C6*X^6 + C7*X^7
   При X = 1.5 и C = [1,1,1,1,2,2,3,3]:
   Y = 1 + 1.5 + 2.25 + 3.375 + 2*5.0625 + 2*7.59375 + 3*11.390625 + 3*17.0859375
     = 118.0078125
*)
Y := OSCAT_BASIC.F_POLY(X := X, C := C);

END_PROGRAM

```

12.4 F_POWER

Тип модуля: функция	Переменная	Тип	Описание
Входы	a	REAL	Коэффициент
	X	REAL	Аргумент
	n	REAL	Показатель степени
Выходы	F_POWER	REAL	Значение степенной функции

Функция **F_POWER** возвращает для аргумента **X** значение степенной функции, вычисленное по формуле

$$F_POWER(X) = a \cdot X^n$$

Пример

```

VAR
  a : REAL := 2.0;
  x : REAL := 3.0;
  n : REAL := 4.0;
  y : REAL;
END_VAR

y := OSCAT_BASIC.F_POWER(a := a, X := x, n := n);  (* y = 2 * 3^4 = 162 *)

END_PROGRAM

```

12.5 F_QUAD

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Аргумент
	A	REAL	Коэффициент
	B	REAL	Коэффициент
	C	REAL	Коэффициент
Выходы	F_QUAD	REAL	Значение квадратичной функции

Функция **F_QUAD** возвращает для аргумента X значение квадратичной функции, вычисленное по формуле $F_QUAD(X) = A \cdot X^2 + B \cdot X + C$

Пример

```

VAR
  x : REAL := 2.0;
  a : REAL := 3.0;
  b : REAL := -4.0;
  c : REAL := 1.0;
  y : REAL;
END_VAR

y := OSCAT_BASIC.F_QUAD(X := x, A := a, B := b, C := c);
(* y = 3*2^2 + (-4)*2 + 1 = 5 *)

END_PROGRAM

```

12.6 FRMP_B

Тип модуля: функция	Переменная	Тип	Описание
Входы	START	BYTE	Начальное значение
	DIR	BOOL	Направление изменения выхода
	TD	TIME	Прошедшее время
	TR	TIME	Полное время генерации (от 0 до 255)
Выходы	FRMP_B	BYTE	Выход блока

Функция **FRMP_B** используется для реализации генератора линейной функции. Пока функция находится в работе, значение ее выхода линейно изменяется от текущего (**START**) до максимального (при **DIR=TRUE**) или минимального (при **DIR=FALSE**). Минимальное и максимальное значения определяются диапазоном типа **BYTE** и соответственно составляют **0** и **255**. Вход **TR** определяет полное время генерации (от 0 до 255 секунд), а вход **TD** – время, прошедшее с начала работы функции. Таким образом, значение входа **TD** должно увеличиваться во время работы функции для изменения ее выхода.

См. также генератор **_RMP_B**, построенный на базе данной функции.

Пример

```

VAR
  start : BYTE := 100;      (* стартовое значение *)
  dirUp  : BOOL := TRUE;    (* TRUE: к 255, FALSE: к 0 *)
  td     : TIME := T#0s;    (* прошедшее время *)
  tr     : TIME := T#10s;   (* полное время генерации *)
  y      : BYTE;
END_VAR

(* Пример: эмуляция прошедшего времени *)
td := T#4s;

y := OSCAT_BASIC.FRMP_B(START := start, DIR := dirUp, TD := td, TR := tr);
(* При START=100, DIR=TRUE, TR=10s и TD=4s:
   y ≈ 100 + (255-100)*4/10 = 162 *)

END_PROGRAM

```

12.7 FT_AVG

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	REAL	Контролируемое значение
	E	BOOL	Сигнал управления блоком
	N	N	Число точек усреднения сигнала (0...32)
	RST	BOOL	Сигнал сброса блока
Выходы	FT_AVG	REAL	Значение скользящей средней

Функциональный блок **FT_AVG** представляет собой фильтр скользящей средней. Если вход **E** имеет значение **TRUE**, то блок находится в работе, и на выход **AVG** подается значение входа **IN**, усредненное за последние **N** циклов ПЛК (**K** – номер текущего цикла ПЛК):

$$Y_K = \frac{1}{N} \sum_{i=K-N}^K X_i$$

Если $N=0$, то $AVG=IN$. По переднему фронту на входе **RST** данные блока обнуляются, и усреднение начинается заново.

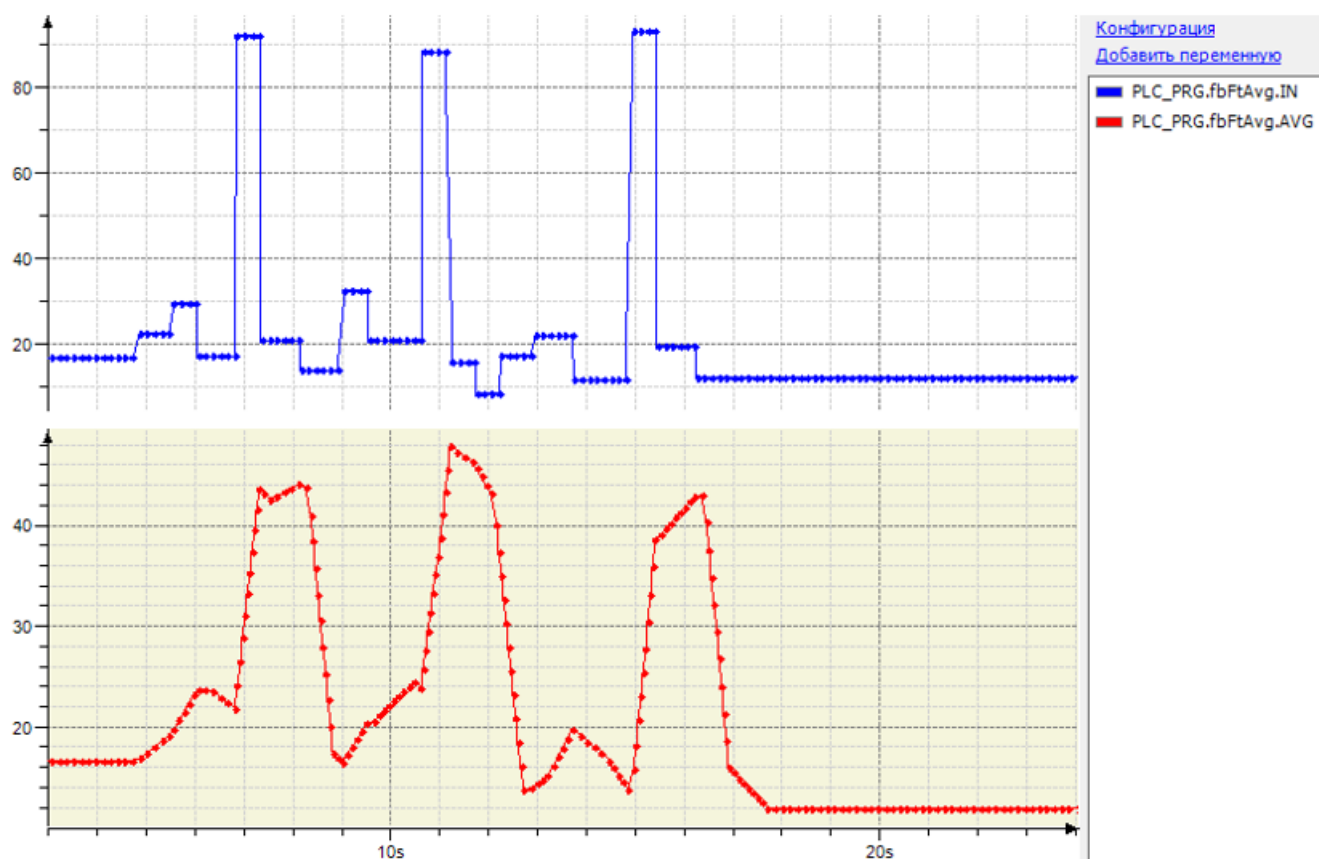


Рисунок 12.1 – Трассировка работы ФБ FT_AVG (N=30, время цикла ПЛК=50 мс)

12.8 FT_MIN_MAX

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	REAL	Контролируемое значение
	RST	BOOL	Сигнал сброса блока
Выходы	MX	REAL	Максимальное значение
	MN	REAL	Минимальное значение

Функциональный блок **FT_MIN_MAX** собирает статистику о входе **IN**, сохраняя его максимальное и минимальное значение на выходах **MX** и **MN**. По переднему фронту на входе **RST** значения выходов обнуляются, после чего сбор статистики начинается заново.

12.9 FT_RMP

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	Rmp	BOOL	Сигнал управления блоком
	IN	REAL	Контролируемый сигнал
	KR	REAL	Допустимая скорость нарастания, ед/с
	KF	REAL	Допустимая скорость спада, ед/с
Выходы	out	REAL	Сглаженное значение сигнала
	busy	BOOL	Флаг «блок в работе»
	UD	BOOL	Направление изменения сигнала

Функциональный блок **FT_RMP** используется для сглаживания сигналов. Пока вход **Rmp** имеет значение **TRUE** блок находится в работе и значение выхода **out** следует за значением входа **IN**, при этом допустимая максимальная скорость увеличения значения на выходе (в ед./с) определяется входом **KR**, а допустимая

максимальная скорость уменьшения – входом **KF**. Выход **busy** имеет значение **TRUE**, пока происходит изменение выхода **out**. Выход **UD** имеет значение **TRUE**, если значение выхода **out** увеличивается, и **FALSE** – если уменьшается. Величина изменения выхода **out** зависит от времени цикла ПЛК; см. также блоки **RMP_B** и **RMP_W** с фиксированной величиной изменения выходов.

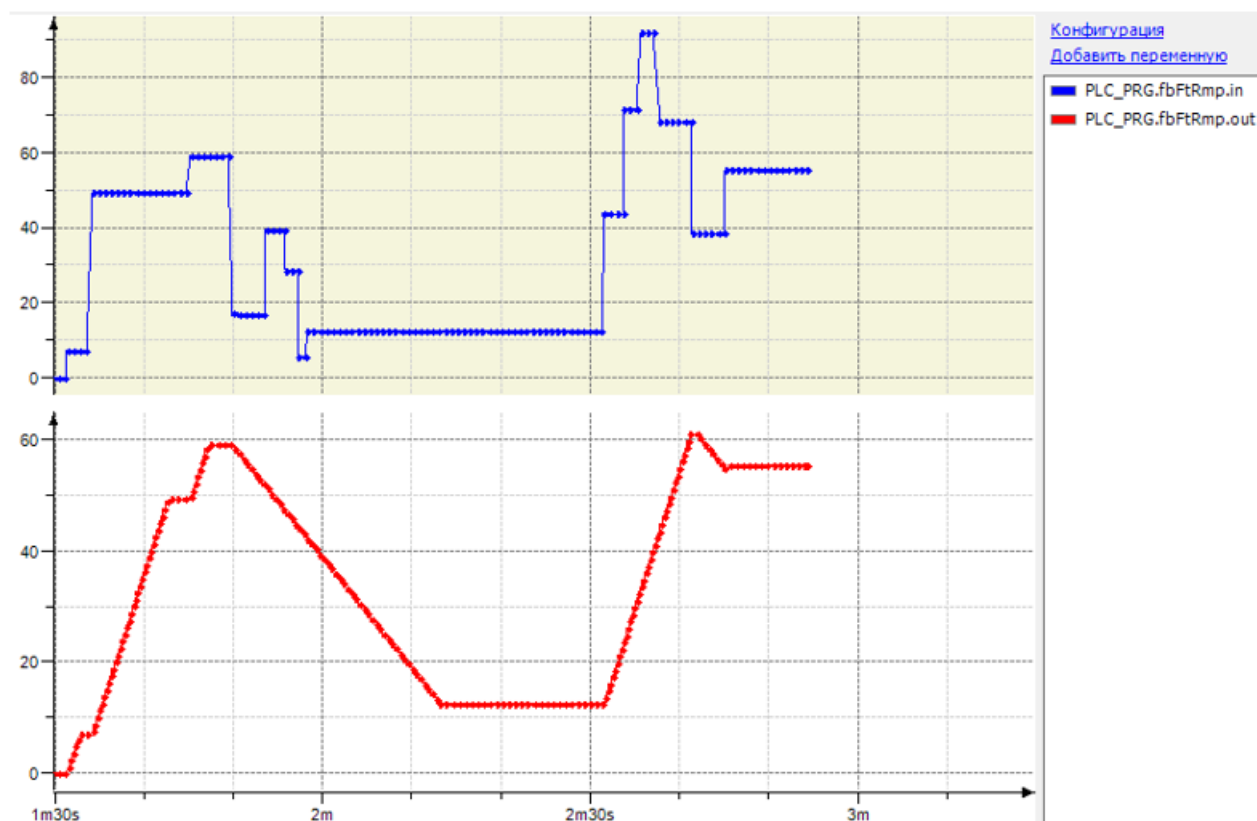


Рисунок 12.2 – Трассировка работы ФБ FT_RMP (KR=5.0, KF=2.0, время цикла ПЛК=50 мс)

12.10 LINEAR_INT

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Аргумент
	XY	ARRAY [1..20, 0..1] OF REAL	Таблица узловых точек
	Pts	INT	Число используемых узловых точек (3..20)
Выходы	LINEAR_INT	REAL	Линейно интерполированное значение Y для X

Функция **LINEAR_INT** выполняет для значения **X** кусочно-линейную интерполяцию по зависимости, определяемой таблицей узловых точек **XY**. Значения **X** в таблице должны быть отсортированы по возрастанию. Вход **Pts** определяет число используемых для интерполяции узловых точек и может принимать значение в диапазоне **[3...20]**. Если значение **X** не входит в диапазон таблицы, то значение **Y** (**LINEAR_INT**) экстраполируется.

См. график в разделе [POLYNOM_INT](#)

Пример


```

VAR      (* Таблица узловых точек: [номер_точки, 0 для X, 1 для Y] *)
XY_Table : ARRAY [1..20, 0..1] OF REAL := [
    0.0, 0.0, (* Точка 1: X=0.0, Y=0.0 *)
    1.0, 2.0, (* Точка 2: X=1.0, Y=2.0 *)
    2.0, 4.0, (* Точка 3: X=2.0, Y=4.0 *)
    3.0, 8.0, (* Точка 4: X=3.0, Y=8.0 *)
    4.0, 16.0, (* Точка 5: X=4.0, Y=16.0 *)
    5.0, 32.0, (* Точка 6: X=5.0, Y=32.0 *)
];

InputX : REAL := 1.5; (* Аргумент для интерполяции *)
Pts     : INT  := 6;  (* Количество используемых точек *)
ResultY : REAL;      (* Результат интерполяции *)

END_VAR

(* Вызов функции интерполяции из библиотеки OSCAT_BASIC *)
ResultY := OSCAT_BASIC.LINEAR_INT(X := InputX, XY := XY_Table, Pts := Pts);

(* После выполнения:
   ResultY будет равен 3.0, так как X=1.5 находится посередине
   между точками 1.0 (Y=2.0) и 2.0 (Y=4.0).
   *)

END_PROGRAM

```

12.11 POLYNOM_INT

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Аргумент
	XY	ARRAY [1..5, 0..1] OF REAL	Таблица узловых точек
	Pts	INT	Число используемых узловых точек (3..5)
Выходы	POLYNOM_INT	REAL	Полиномиально интерполированное значение Y для X

Функция **POLYNOM_INT** выполняет для значения **X** полиномиальную интерполяцию по зависимости, определяемой таблицей узловых точек **XY**. Значения **X** в таблице должны быть отсортированы по возрастанию. Вход **Pts** определяет число используемых для интерполяции узловых точек и может принимать значение в диапазоне **[3...5]**. Если значение **X** не входит в диапазон таблицы, то значение **Y** (**POLYNOM_INT**) экстраполируется.

Пример

```

VAR      (* Таблица узловых точек: [номер_точки, 0 для X, 1 для Y] *)
XY_Table : ARRAY [1..5, 0..1] OF REAL := [
    0.0, 1.0,      (* Точка 1: X=0.0, Y=1.0 *)
    1.0, 2.0,      (* Точка 2: X=1.0, Y=2.0 *)
    2.0, 5.0,      (* Точка 3: X=2.0, Y=5.0 *)
    3.0, 10.0,     (* Точка 4: X=3.0, Y=10.0 *)
    4.0, 17.0,     (* Точка 5: X=4.0, Y=17.0 *)
];

InputX : REAL := 2.5;  (* Аргумент для интерполяции *)
Pts     : INT  := 5;   (* Количество используемых точек (3..5) *)
ResultY : REAL;        (* Результат интерполяции *)

END_VAR

(* Вызов функции полиномиальной интерполяции *)
ResultY := OSCAT_BASIC.POLYNOM_INT(X := InputX, XY := XY_Table, Pts := Pts);

(*
Результат:
При X=2.5 и 5 точках, полиномиальная интерполяция даст значение,
которое точно проходит через все заданные точки.

Для данных точек функция соответствует полиному  $Y = X^2 + 1$ 
При X=2.5:  $Y = 6.25 + 1 = 7.25$ 
*)

END_PROGRAM

```

13 Геометрические функции

13.1 CIRCLE_A

Тип модуля: функция	Переменная	Тип	Описание
Входы	rx	REAL	Радиус окружности
	ax	REAL	Угол сектора в градусах
Выходы	CIRCLE_A	REAL	Площадь сектора окружности

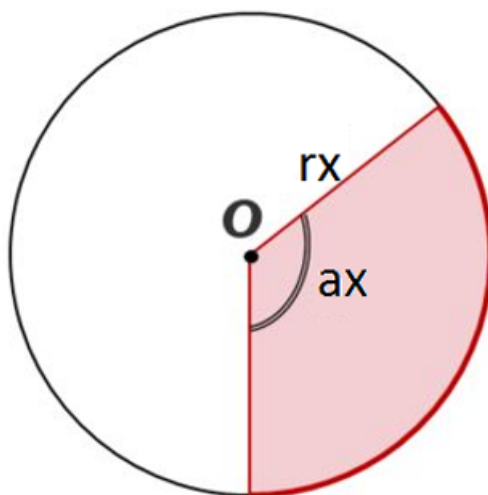


Рисунок 13.1

Функция **CIRCLE_A** возвращает для окружности с радиусом **rx** площадь сектора с углом **ax**, вычисленную по формуле:

$$\text{CIRCLE_A} = \frac{\pi \cdot rx^2 \cdot ax}{360}$$

Пример

```

VAR
  rRadius : REAL := 5.0;      // Радиус окружности
  rAngle  : REAL := 90.0;    // Угол сектора в градусах
  rArea   : REAL;            // Площадь сектора
END_VAR

rArea := OSCAT_BASIC.CIRCLE_A(rx := rRadius, ax := rAngle);
// Результат: 19.63495 (площадь сектора 90° при радиусе 5)

END_PROGRAM

```

13.2 CIRCLE_C

Тип модуля: функция	Переменная	Тип	Описание
Входы	rx	REAL	Радиус окружности
	ax	REAL	Угол сектора в градусах
Выходы	CIRCLE_C	REAL	Длина дуги сектора окружности

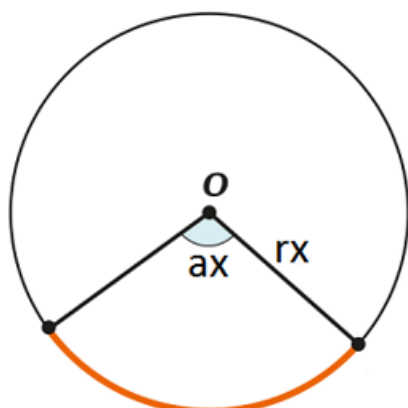


Рисунок 13.2

Функция **CIRCLE_C** возвращает для окружности с радиусом **rx** длину дуги сектора с углом **ax**, вычисленную по формуле:

$$\text{CIRCLE_C} = \frac{2\pi \cdot rx \cdot ax}{360}$$

Пример

```
VAR
  rRadius    : REAL := 5.0;    // Радиус окружности = 5 единиц
  rAngle     : REAL := 90.0;   // Угол сектора = 90 градусов
  rArcLength : REAL;           // Длина дуги сектора
END_VAR

rArcLength := OSCAT_BASIC.CIRCLE_C( rx := rRadius, ax := rAngle );
// Результат: 7.85398 (длина дуги 90° при радиусе 5)

END_PROGRAM
```

13.3 CIRCLE_SEG

Тип модуля: функция	Переменная	Тип	Описание
Входы	rx	REAL	Радиус окружности
	HX	REAL	Высота сегмента
Выходы	CIRCLE_SEG	REAL	Площадь сегмента окружности

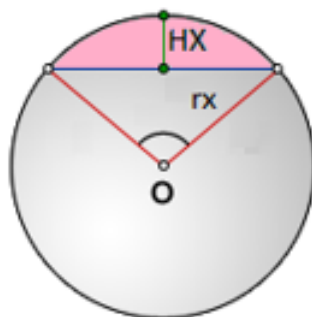


Рисунок 13.3

Функция **CIRCLE_SEG** возвращает для окружности с радиусом **rx** площадь сегмента с высотой **HX**.

Пример

```

VAR
  rRadius  : REAL := 5.0;      // Радиус окружности = 5 единиц
  rHeight  : REAL := 1.0;      // Высота сегмента = 1 единица
  rSegArea : REAL;             // Площадь сегмента
END_VAR

rSegArea := OSCAT_BASIC.CIRCLE_SEG( rx := rRadius, hx := rHeight );
// Результат: 4.08752775 (площадь сегмента при радиусе 5 и высоте 1)

END_PROGRAM

```

13.4 CONE_V

Тип модуля: функция	Переменная	Тип	Описание
Входы	rx	REAL	Радиус основания конуса
	hx	REAL	Высота конуса
Выходы	CONE_V	REAL	Объем конуса

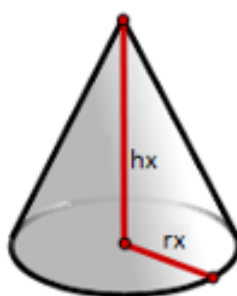


Рисунок 13.4

Функция **CONE_V** возвращает объем конуса с радиусом основания **rx** и высотой **hx**, вычисленный по формуле

$$\text{CONE_V} = \frac{\pi \cdot rx^2 \cdot hx}{3}$$

Пример

```

VAR
  rRadius  : REAL := 3.0;      // Радиус основания конуса = 3 единицы
  rHeight  : REAL := 5.0;      // Высота конуса = 5 единиц
  rVolume  : REAL;             // Объем конуса
END_VAR

rVolume := OSCAT_BASIC.CONE_V( rx := rRadius, hx := rHeight );
// Результат: 47.12389 (объем конуса с радиусом 3 и высотой 5)

END_PROGRAM

```

13.5 ELLIPSE_A

Тип модуля: функция	Переменная	Тип	Описание
Входы	R1	REAL	Малая полуось эллипса
	R2	REAL	Большая полуось эллипса
Выходы	ELLIPSE_A	REAL	Площадь эллипса

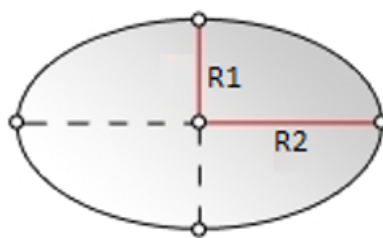


Рисунок 13.5

Функция **ELLIPSE_A** возвращает площадь эллипса с полуосями **R1** и **R2**, вычисленную по формуле

$$\text{ELLIPSE_A} = \pi \cdot R1 \cdot R2$$

Пример

```
VAR
  rSemiAxis1 : REAL := 3.0;      // Малая полуось эллипса = 3 единицы
  rSemiAxis2 : REAL := 5.0;      // Большая полуось эллипса = 5 единиц
  rArea      : REAL;             // Площадь эллипса
END_VAR

rArea := OSCAT_BASIC.ELLIPSE_A( R1 := rSemiAxis1, R2 := rSemiAxis2 );
// Результат: 47.12389 (площадь эллипса с полуосями 3 и 5)

END_PROGRAM
```

13.6 ELLIPSE_C

Тип модуля: функция	Переменная	Тип	Описание
Входы	R1	REAL	Малая полуось эллипса
	R2	REAL	Большая полуось эллипса
Выходы	ELLIPSE_C	REAL	Периметр эллипса

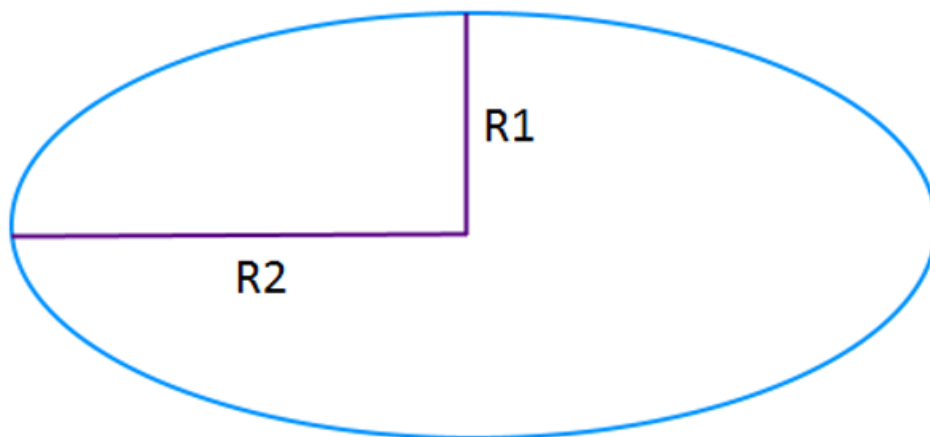


Рисунок 13.6

Функция **ELLIPSE_C** возвращает периметр эллипса с полуосями **R1** и **R2**, вычисленный по формуле

$$\text{ELLIPSE_C} = \pi \cdot [3 \cdot (R1 + R2) - \sqrt{(3R1 + R2) \cdot (R1 + 3R2)}]$$

Пример

```

VAR
  rSemiAxis1 : REAL := 3.0;      // Малая полуось эллипса = 3 единицы
  rSemiAxis2 : REAL := 5.0;      // Большая полуось эллипса = 5 единиц
  rPerimeter : REAL;             // Периметр эллипса
END_VAR

rPerimeter := OSCAT_BASIC.ELLIPSE_C( R1 := rSemiAxis1, R2 := rSemiAxis2 );
// Результат: 25.52763 (периметр эллипса с полуосями 3 и 5)

END_PROGRAM

```

13.7 SPHERE_V

Тип модуля: функция	Переменная	Тип	Описание
Входы	rx	REAL	Радиус сферы
Выходы	SPHERE_V	REAL	Объем сферы

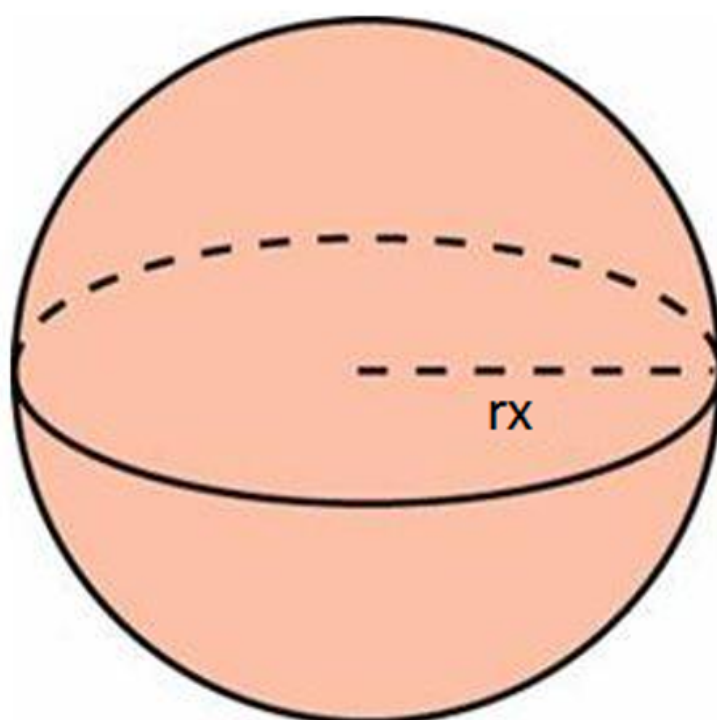


Рисунок 13.7

Функция **SPHERE_V** возвращает объем сферы с радиусом **rx**, вычисленный по формуле

$$\text{SPHERE_V} = \frac{4}{3} \pi \cdot rx^3$$

Пример

```

VAR
  rRadius : REAL := 3.0;      // Радиус сферы = 3 единицы
  rVolume : REAL;             // Объем сферы
END_VAR

rVolume := OSCAT_BASIC.SPHERE_V( rx := rRadius );
// Результат: 113.09734 (объем сферы с радиусом 3)

END_PROGRAM

```

13.8 TRIANGLE_A

Тип модуля: функция	Переменная	Тип	Описание
Входы	S1	REAL	Длина стороны 1
	S2	REAL	Длина стороны 2
	A	REAL	Угол между сторонами 1 и 2, градусы
	S3	REAL	Длина стороны 3
Выходы	TRIANGLE_A	REAL	Площадь треугольника

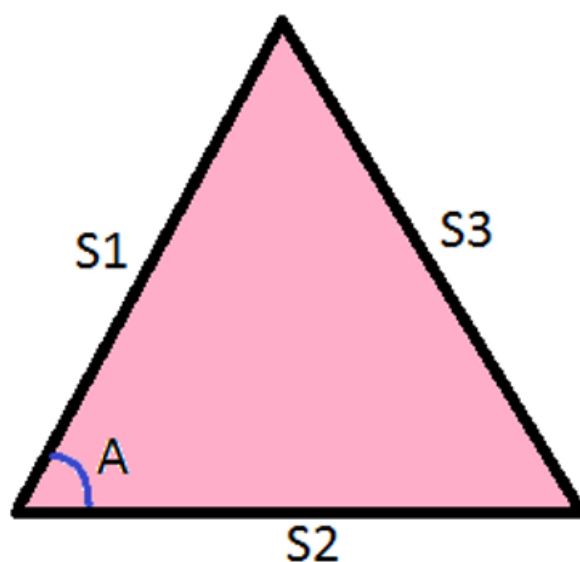


Рисунок 13.8

Функция **TRIANGLE_A** возвращает площадь треугольника, вычисленную одним из двух способов:

1. По двум сторонам и углу между ними (**S1**, **S2**, **A**). В этом случае значение входа **S3** должно быть равно **0**.

$$\text{TRIANGLE_A} = \frac{1}{2} \cdot S1 \cdot S2 \cdot \sin(A)$$

2. По трем сторонам (**S1**, **S2**, **S3**). В этом случае значение входа **A** должно быть равно **0**.

$$\text{TRIANGLE_A} = \sqrt{p \cdot (p - S1) \cdot (p - S2) \cdot (p - S3)}$$

$$p = \frac{S1 + S2 + S3}{2}$$

Примеры


```
VAR
  rSide1 : REAL := 5.0;      // Сторона 1 = 5 единиц
  rSide2 : REAL := 7.0;      // Сторона 2 = 7 единиц
  rAngle  : REAL := 60.0;    // Угол между сторонами = 60 градусов
  rSide3  : REAL := 0.0;    // Не используется (должна быть 0)
  rArea   : REAL;           // Площадь треугольника
END_VAR

// Вычисление площади по двум сторонам и углу между ними
rArea := OSCAT_BASIC.TRIANGLE_A( S1 := rSide1,
  S2 := rSide2, A := rAngle, S3 := rSide3 );
// Результат: 15.155 (площадь треугольника со сторонами 5 и 7 и углом 60°)

END_PROGRAM
```

14 Операции над векторами

Функции, описанные в данной главе, используются для работы с радиус-векторами трехмерного пространства. Координаты радиус-вектора описываются структурой [VECTOR_3](#).

14.1 V3_ABS

Тип модуля: функция	Переменная	Тип	Описание
Входы	A	VECTOR_3	Радиус-вектор
Выходы	V3_ABS	REAL	Длина радиус-вектора

Функция **V3_ABS** возвращает модуль радиус-вектора **A** (равный длине соответствующего отрезка), вычисленный по формуле

$$V3_ABS(A) = \sqrt{(A.X)^2 + (A.Y)^2 + (A.Z)^2}$$

Пример

```
VAR
  a      : OSCAT_BASIC.VECTOR_3;
  len    : REAL;
END_VAR

a.X := 3.0;
a.Y := 4.0;
a.Z := 0.0;

len := OSCAT_BASIC.V3_ABS(A := a); (* len = 5.0 *)

END_PROGRAM
```

14.2 V3_ADD

Тип модуля: функция	Переменная	Тип	Описание
Входы	A	VECTOR_3	Радиус-вектор A
	B	VECTOR_3	Радиус-вектор B
Выходы	V3_ADD	VECTOR_3	Сумма радиус-векторов A и B

Функция **V3_ADD** возвращает сумму радиус-векторов **A** и **B** (суммирование векторов представляет собой суммирование их координат).

Пример

```
VAR
  a      : OSCAT_BASIC.VECTOR_3;
  b      : OSCAT_BASIC.VECTOR_3;
  sum    : OSCAT_BASIC.VECTOR_3;
END_VAR

a.X := 3.0;
a.Y := 4.0;
a.Z := 0.0;

b.X := 1.0;
b.Y := 2.0;
b.Z := 2.0;

sum := OSCAT_BASIC.V3_ADD(A := a, B := b); (* sum = (4, 6, 2) *)

END_PROGRAM
```

14.3 V3_ANG

Тип модуля: функция	Переменная	Тип	Описание
Входы	A	VECTOR_3	Радиус-вектор A
	B	VECTOR_3	Радиус-вектор B
Выходы	V3_ANG	REAL	Угол между радиус-векторами A и B (в радианах)
Используемые модули	V3_DPRO		

Функция **V3_ANG** возвращает значение угла (в радианах) между радиус-векторами **A** и **B**.

Пример

```

VAR
  a   : OSCAT_BASIC.VECTOR_3;
  b   : OSCAT_BASIC.VECTOR_3;
  ang : REAL;
END_VAR

a.X := 1.0; a.Y := 0.0; a.Z := 0.0;
b.X := 0.0; b.Y := 1.0; b.Z := 0.0;

ang := OSCAT_BASIC.V3_ANG(A := a, B := b); (* ang ≈ 1.5707963 (π/2) *)

END_PROGRAM

```

14.4 V3_DPRO

Тип модуля: функция	Переменная	Тип	Описание
Входы	A	VECTOR_3	Радиус-вектор A
	B	VECTOR_3	Радиус-вектор B
Выходы	V3_DPRO	REAL	Скалярное произведение радиус-векторов A и B

Функция **V3_DPRO** возвращает скалярное произведение радиус-векторов **A** и **B**.

Пример

```

VAR
  a   : OSCAT_BASIC.VECTOR_3;
  b   : OSCAT_BASIC.VECTOR_3;
  ang : REAL;
END_VAR

a.X := 3.0; a.Y := 4.0; a.Z := 0.0;
b.X := 1.0; b.Y := 2.0; b.Z := 2.0;
ang := OSCAT_BASIC.V3_DPRO(A := a, B := b); (* dot = 11.0 *)

END_PROGRAM

```

14.5 V3_NORM

Тип модуля: функция	Переменная	Тип	Описание
Входы	A	VECTOR_3	Одномерный радиус-вектор
Выходы	V3_NORM	VECTOR_3	Единичный радиус-вектор
Используемые модули	V3_ABS, V3_SMUL		

Функция **V3_NORM** конвертирует одномерный радиус-вектор **A** в единичный вектор, ориентированный в том же направлении.

Пример

```

VAR
  a      : OSCAT_BASIC.VECTOR_3;
  unit   : OSCAT_BASIC.VECTOR_3;
END_VAR

a.X := 3.0;
a.Y := 4.0;
a.Z := 0.0;

unit := OSCAT_BASIC.V3_NORM(A := a);  (* unit ≈ (0.6, 0.8, 0.0) *)

END_PROGRAM

```

14.6 V3_NUL

Тип модуля: функция	Переменная	Тип	Описание
Входы	A	VECTOR_3	Радиус-вектор
Выходы	V3_NUL	BOOL	Флаг «нулевой вектор»

Функция **V3_NUL** возвращает **TRUE**, если радиус-вектор **A** является нулевым вектором.

Пример

```

VAR
  a      : OSCAT_BASIC.VECTOR_3;
  isNul  : BOOL;
END_VAR

a.X := 0.0;
a.Y := 0.0;
a.Z := 0.0;

isNul := OSCAT_BASIC.V3_NUL(A := a);  (* isNul ≈ TRUE *)

END_PROGRAM

```

14.7 V3_PAR

Тип модуля: функция	Переменная	Тип	Описание
Входы	A	VECTOR_3	Радиус-вектор A
	B	VECTOR_3	Радиус-вектор B
Выходы	V3_PAR	BOOL	Флаг «коллинеарные векторы»
Используемые модули	V3_ABS, V3_XPRO		

Функция **V3_PAR** возвращает **TRUE**, если радиус-векторы **A** и **B** являются коллинеарными. Нулевой вектор коллинеарен любому вектору, т.к. не имеет направления.

Пример

```

VAR
  a      : OSCAT_BASIC.VECTOR_3;
  b      : OSCAT_BASIC.VECTOR_3;
  isPar  : BOOL;
END_VAR

a.X := 1.0; a.Y := 2.0; a.Z := 3.0;
b.X := 2.0; b.Y := 4.0; b.Z := 6.0;
isPar := OSCAT_BASIC.V3_PAR(A := a, B := b);  (* isPar = TRUE *)

END_PROGRAM

```

14.8 V3_REV

Тип модуля: функция	Переменная	Тип	Описание
Входы	A	VECTOR_3	Радиус-вектор
Выходы	V3_REV	VECTOR_3	Противонаправленный радиус-вектор

Функция **V3_REV** конвертирует радиус-вектор **A** в радиус-вектор с равной длиной, но ориентированный в противоположном направлении.

Пример

```
VAR
  a   : OSCAT_BASIC.VECTOR_3;
  rev : OSCAT_BASIC.VECTOR_3;
END_VAR

a.X := 3.0;
a.Y := 4.0;
a.Z := 0.0;

rev := OSCAT_BASIC.V3_REV(A := a);  (* rev = (-3, -4, 0) *)

END_PROGRAM
```

14.9 V3_SMUL

Тип модуля: функция	Переменная	Тип	Описание
Входы	A	VECTOR_3	Радиус-вектор A
	M	REAL	Число
Выходы	V3_SMUL	VECTOR_3	Произведение радиус-векторов A и числа M

Функция **V3_SMUL** возвращает произведение радиус-вектора **A** и числа **M**.

Пример

```
VAR
  a   : OSCAT_BASIC.VECTOR_3;
  m   : REAL := 2.0;
  res : OSCAT_BASIC.VECTOR_3;
END_VAR

a.X := 3.0;
a.Y := 4.0;
a.Z := 0.0;

res := OSCAT_BASIC.V3_SMUL(A := a, M := m);  (* res = (6, 8, 0) *)

END_PROGRAM
```

14.10 V3_SUB

Тип модуля: функция	Переменная	Тип	Описание
Входы	A	VECTOR_3	Радиус-вектор A
	B	VECTOR_3	Радиус-вектор B
Выходы	V3_SUB	VECTOR_3	Разность радиус-векторов A и B

Функция **V3_SUB** возвращает разность радиус-векторов **A** и **B**.

Пример

```

VAR
  a   : OSCAT_BASIC.VECTOR_3;
  b   : OSCAT_BASIC.VECTOR_3;
  dif : OSCAT_BASIC.VECTOR_3;
END_VAR

a.X := 3.0; a.Y := 4.0; a.Z := 0.0;
b.X := 1.0; b.Y := 2.0; b.Z := 2.0;
dif := OSCAT_BASIC.V3_SUB(A := a, B := b); (* dif = (2, 2, -2) *)

END_PROGRAM

```

14.11 V3_XANG

Тип модуля: функция	Переменная	Тип	Описание
Входы	A	VECTOR_3	Радиус-вектор
Выходы	V3_XANG	REAL	Угол между радиус-вектором и осью X
Используемые модули	V3_ABS		

Функция **V3_XANG** возвращает угол между радиус-вектором **A** и осью **X** (в радианах).

Пример

```

VAR
  a   : OSCAT_BASIC.VECTOR_3;
  xAng : REAL;
END_VAR

a.X := 3.0;
a.Y := 4.0;
a.Z := 0.0;

xAng := OSCAT_BASIC.V3_XANG(A := a); (* xAng ≈ 0.9272952 рад *)

END_PROGRAM

```

14.12 V3_XPRO

Тип модуля: функция	Переменная	Тип	Описание
Входы	A	VECTOR_3	Радиус-вектор A
	B	VECTOR_3	Радиус-вектор B
Выходы	V3_XPRO	VECTOR_3	Векторное произведение радиус-векторов A и B

Функция **V3_XPRO** возвращает векторное произведение радиус-векторов **A** и **B**.

Пример

```

VAR
  a   : OSCAT_BASIC.VECTOR_3;
  b   : OSCAT_BASIC.VECTOR_3;
  cross : OSCAT_BASIC.VECTOR_3;
END_VAR

a.X := 1.0; a.Y := 0.0; a.Z := 0.0;
b.X := 0.0; b.Y := 1.0; b.Z := 0.0;
cross := OSCAT_BASIC.V3_XPRO(A := a, B := b); (* cross = (0, 0, 1) *)

END_PROGRAM

```

14.13 V3_YANG

Тип модуля: функция	Переменная	Тип	Описание
Входы	A	VECTOR_3	Радиус-вектор
Выходы	V3_YANG	REAL	Угол между радиус-вектором и осью Y
Используемые модули	V3_ABS		

Функция **V3_YANG** возвращает угол между радиус-вектором **A** и осью Y (в радианах).

Пример

```

VAR
  a      : OSCAT_BASIC.VECTOR_3;
  yAng   : REAL;
END_VAR

a.X := 3.0;
a.Y := 4.0;
a.Z := 0.0;

yAng := OSCAT_BASIC.V3_YANG(A := a); (* yAng ≈ 0.6435011 рад *)

END_PROGRAM

```

14.14 V3_ZANG

Тип модуля: функция	Переменная	Тип	Описание
Входы	A	VECTOR_3	Радиус-вектор
Выходы	V3_ZANG	REAL	Угол между радиус-вектором и осью Z
Используемые модули	V3_ABS		

Функция **V3_ZANG** возвращает угол между радиус-вектором **A** и осью Z (в радианах).

Пример

```

VAR
  a      : OSCAT_BASIC.VECTOR_3;
  zAng   : REAL;
END_VAR

a.X := 3.0;
a.Y := 4.0;
a.Z := 0.0;

zAng := OSCAT_BASIC.V3_ZANG(A := a); (* zAng ≈ 1.5707963 рад *)

END_PROGRAM

```

15 Дата и время

В ALTA IDE значение календарной даты и местного времени, вычисляется относительно нулевой точки 0000-0-0-0:0:0 (DATE_AND_TIME (DT)).

15.1 CALENDAR_CALC

Тип модуля: функция	Переменная	Тип	Описание
Входы	SPE	BOOL	Сигнал включения расчета азимута и высоты солнца над горизонтом
	H	REAL	Высота солнца над горизонтом на восходе в градусах
Выходы	XCAL	CALENDAR	Данные календаря
	HOLIDAYS	ARRAY [0...29] OF HOLIDAY_DATA	Список праздников
Используемые модули	UTC_TO_LTIME, DAY_OF_DATE, HOUR, DST, YEAR_OF_DATE, MONTH_OF_DATE, DAY_OF_MONTH, DAY_OF_WEEK, WORK_WEEK, HOLIDAY, SUN_TIME, SUN_POS		

Функциональный блок **CALENDAR_CALC** принимает на вход-выход **XCAL** экземпляр структуры типа **CALENDAR**, для которого пользователь определяет значения переменных **UTC**, **OFFSET**, **DST_EN**, **NAME**, **LANGUAGE**, **LONGITUDE**, **LATITUDE** (достаточно определить только нужные переменные). На основании этих данных блок рассчитывает значения всех остальных переменных структуры. Вход **SPE** включает (при значении **TRUE**) и отключает (при значении **FALSE**) расчет параметров позиции солнца (поскольку эта операция может оказаться ресурсоемкой, особенно для ПЛК, не оснащенных FPU). При включенном расчете параметров обновление переменных происходит каждые 25 секунд, при этом точность составляет $\pm 0.1^\circ$. Вход **H** определяет высоту центра солнечного диска в градусах (значение по умолчанию: **-0.8333333333**), используемую при расчете параметров позиции солнца. Вход-выход **HOLIDAYS** представляет собой массив типа **HOLIDAY_DATA** и используется для определения дат праздничных дней.

15.2 DATE_ADD

Тип модуля: функция	Переменная	Тип	Описание
Входы	IDATE	DATE	Исходная дата
	D	INT	Число прибавляемых дней
	W	INT	Число прибавляемых недель
	M	INT	Число прибавляемых месяцев
	Y	INT	Число прибавляемых лет
Выходы	DATE_ADD	DATE	Вычисленная дата
Используемые модули	YEAR_OF_DATE, MONTH_OF_DATE, DAY_OF_MONTH, SET_DATE,		

Функция **DATE_ADD** прибавляет к начальной дате **IDATE** заданное число дней **D**, недель **W**, месяцев **M** и лет **Y**. Прибавляемые числа могут быть как положительными, так и отрицательными. При этом следует помнить, что диапазон возможных значений переменной типа **DATE** составляет D#0000-0-0 - D#9999-12-31.

```

VAR
  StartDate : DATE := D#2026-06-18; (* начальная дата *)
  ResultDate : DATE; (* результат *)
END_VAR
(* Прибавить: +10 дней, +3 недели, +1 месяц, +0 лет *)
ResultDate := OSCAT_BASIC.DATE_ADD(
  IDATE := StartDate,
  D := 10,
  W := 3,
  M := 1,
  Y := 0
); (* Результат: D#2026-08-19 *)
END_PROGRAM

```


15.3 DAY_OF_DATE

Тип модуля: функция	Переменная	Тип	Описание
Входы	IDATE	DATE	Исходная дата
Выходы	DAY_OF_DATE	DINT	Число дней, прошедших с 01.01.1970 до исходной даты

Функция **DAY_OF_DATE** возвращает число дней, прошедших с **01.01.1970** до исходной даты **IDATE**.

Пример

```

VAR
  d0      : DATE := D#1970-01-01;  (* начальная дата эпохи *)
  d1      : DATE := D#2026-06-18;  (* исходная дата IDATE *)
  days0   : DINT;                  (* дней с 01.01.1970 до d0 *)
  days1   : DINT;                  (* дней с 01.01.1970 до d1 *)
END_VAR

days0 := OSCAT_BASIC.DAY_OF_DATE(IDATE := d0);  (* результат 0 *)
days1 := OSCAT_BASIC.DAY_OF_DATE(IDATE := d1);  (* результат 20622 *)

END_PROGRAM

```

15.4 DAY_OF_MONTH

Тип модуля: функция	Переменная	Тип	Описание
Входы	IDATE	DATE	Исходная дата
Выходы	DAY_OF_MONTH	INT	Номер дня в месяце исходной даты
Используемые модули	DAY_OF_YEAR, LEAP_OF_DATE		

Функция **DAY_OF_MONTH** возвращает номер дня в месяце исходной даты **IDATE**.

Пример

```

VAR
  d      : DATE := D#2026-06-18;  (* исходная дата IDATE *)
  dayNum : INT;                    (* номер дня в месяце *)
END_VAR

dayNum := OSCAT_BASIC.DAY_OF_MONTH(IDATE := d);  (* вернёт 18 *)

END_PROGRAM

```

15.5 DAY_OF_WEEK

Тип модуля: функция	Переменная	Тип	Описание
Входы	IDATE	DATE	Исходная дата
Выходы	DAY_OF_WEEK	INT	Номер дня недели исходной даты

Функция **DAY_OF_WEEK** возвращает номер дня недели исходной даты **IDATE** (**1** – понедельник, **7** – воскресенье).

Пример

```

VAR
  d      : DATE := D#2026-06-18;  (* исходная дата IDATE *)
  dow    : INT;                    (* номер дня недели *)
END_VAR

dow := OSCAT_BASIC.DAY_OF_WEEK(IDATE := d);  (* вернёт 4 (четверг) *)

END_PROGRAM

```

15.6 DAY_OF_YEAR

Тип модуля: функция	Переменная	Тип	Описание
Входы	IDATE	DATE	Исходная дата
Выходы	DAY_OF_YEAR	INT	Номер дня в году исходной даты

Функция **DAY_OF_YEAR** возвращает номер дня в году исходной даты **IDATE**.

Пример

```

VAR
  d      : DATE := D#2026-06-18;  (* исходная дата IDATE *)
  dayOfYear: INT;                  (* номер дня в году *)
END_VAR

dayOfYear := OSCAT_BASIC.DAY_OF_YEAR(IDATE := d);  (* вернёт 169 *)

END_PROGRAM

```

15.7 DAY_TO_TIME

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	REAL	Число дней в формате REAL
Выходы	DAY_TO_TIME	TIME	Число дней в формате TIME

Функция **DAY_TO_TIME** конвертирует число дней **IN** типа **REAL** в значение типа **TIME**.

Пример

```

VAR
  days : REAL := 1.5;  (* число дней *)
  t     : TIME;         (* результат *)
END_VAR

t := OSCAT_BASIC.DAY_TO_TIME(IN := days);  (* вернёт T#1d12h *)

END_PROGRAM

```

15.8 DAYS_DELTA

Тип модуля: функция	Переменная	Тип	Описание
Входы	DATE_1	DATE	Дата 1
	DATE_2	DATE	Дата 2
Выходы	DAYS_DELTA	DINT	Число дней между двумя датами

Функция **DAYS_DELTA** возвращает разницу в днях между двумя датами типа **DATE**.

Пример

```

VAR
  d1    : DATE := D#2026-06-18; (* первая дата *)
  d2    : DATE := D#2026-06-28; (* вторая дата *)
  delta: DINT; (* разница в днях *)
END_VAR

delta := OSCAT_BASIC.DAYS_DELTA(d1, d2); (* вернёт 10 *)

END_PROGRAM

```

15.9 DAYS_IN_MONTH

Тип модуля: функция	Переменная	Тип	Описание
Входы	IDATE	DATE	Исходная дата
Выходы	DAYS_IN_MONTH	INT	Число дней в месяце исходной даты
Используемые модули	DAY_OF_YEAR, LEAP_OF_DATE		

Функция **DAYS_IN_MONTH** возвращает число дней в месяце исходной даты **IDATE**.

Пример

```

VAR
  d          : DATE := D#2026-02-10; (* исходная дата IDATE *)
  daysInMonth: INT; (* число дней в месяце *)
END_VAR

daysInMonth := OSCAT_BASIC.DAYS_IN_MONTH(IDATE := d); (* вернёт 28 *)

END_PROGRAM

```

15.10 DAYS_IN_YEAR

Тип модуля: функция	Переменная	Тип	Описание
Входы	IDATE	DATE	Исходная дата
Выходы	DAYS_IN_YEAR	INT	Число дней в году исходной даты
Используемые модули	LEAP_OF_DATE		

Функция **DAYS_IN_YEAR** возвращает число дней в году исходной даты **IDATE**.

Пример

```

VAR
  d          : DATE := D#2024-06-18; (* исходная дата IDATE *)
  daysInYear : INT; (* число дней в году *)
END_VAR

daysInYear := OSCAT_BASIC.DAYS_IN_YEAR(IDATE := d); (* вернёт 366 *)

END_PROGRAM

```

15.11 DCF77

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	REC	BOOL	Сигнал передатчика DCF77
	SET	BOOL	Сигнал для инициализации часов RTC/RTC1
	SDT	DT	Значение для инициализации часов RTC/RTC1

Тип модуля: ФБ	Переменная	Тип	Описание
	DSI	BOOL	Сигнал «летнее время» для значений инициализации
Выходы	TP	BOOL	Флаг получения данных от передатчика
	DS	BOOL	Флаг «летнее время»
	WDAY	INT	Номер дня недели (1 – понедельник, 7 – воскресенье)
	ERROR	BOOL	Флаг «отсутствие сигнала от передатчика»
	RTC	DT	Текущее время
	RTC1	DT	Текущее время с учетом часового пояса и летнего времени
	MSEC	INT	Миллисекунды для текущего времени
	SYNC	BOOL	Флаг «часы синхронизированы с DCF77»
Параметры	SYNC_TIMEOUT	TIME	Таймаут синхронизации
	TIME_OFFSET	INT	Смещение времени (в часах) для RTC1
	DST_EN	BOOL	Сигнал учета летнего времени для RTC1
Используемые модули	T_PLC_MS, EVEN, SET_DT, DAY_OF_WEEK		

Функциональный блок **DCF77** получает на вход REC сигнал передатчика DCF77 (в виде последовательности бит) и декодирует его для извлечения текущей даты и времени. Передача сигнала осуществляется раз в минуту, для подтверждения корректности данных ФБ ожидает двух последовательных корректных передач, что может занять до трех минут.

В случае подтверждения корректности входных данных, полученные дата и время используются для синхронизации часов **RTC/ RTC1**, при этом выход **TP** на один цикл принимает значение **TRUE** (для синхронизации внешних устройств). Отсчет секунд для выходов **RTC/ RTC1** осуществляется с помощью часов ПЛК. **RTC** содержит дату и время в UTC+1 (исходный сигнал передатчика), а **RTC1** – дату и время со смещением **TIME_OFFSET** (в часах относительно UTC+1) и возможностью учета перехода на летнее время (с помощью параметра **DST_EN**). При переходе на летнее время выход **DS** принимает значение **TRUE**.

Выход **MSEC** содержит текущее значение миллисекунд для **RTC/ RTC1**, выход **WDAY** – номер текущего дня недели (1 – понедельник, 7 – воскресенье). Выход **ERROR** принимает значение **TRUE** при отсутствии сигнала от передатчика на входе **REC**. На вход **SDT** подается дата и время, с которыми будут инициализированы часы **RTC/ RTC1** в первом цикле ФБ. В дальнейшем можно осуществлять инициализацию часов по переднему фронту входа **SET**. Вход **DSI** определяет, учитывают ли значения инициализации часов переход на летнее время. Если в течение времени **SYNC_TIMEOUT** не происходило синхронизации **RTC/ RTC1** с сигналом передатчика, то выход **SYNC** принимает значение **FALSE**.

15.12 DST

Тип модуля: функция	Переменная	Тип	Описание
Входы	UTC	DT	Исходные дата и время
Выходы	DST	BOOL	Флаг «летнее время»
Используемые модули	YEAR_OF_DATE		

Функция **DST** возвращает **TRUE**, если дата и время **UTC** относится к летнему времени. Во всех остальных случаях возвращается **FALSE**. Функция может быть полезна при использовании часов, не поддерживающих переход на летнее время. Переход на летнее время осуществляется в 1:00 по UTC последнего воскресенья марта, возвращение на зимнее – в 1:00 по UTC последнего воскресенья октября.

Пример

```

VAR
  utc   : DT := DT#2026-03-29-01:30:00; (* время UTC *)
  isDst : BOOL;                          (* признак летнего времени *)
END_VAR

isDst := OSCAT_BASIC.DST(IN := utc); (* вернёт TRUE *)

END_PROGRAM

```

15.13 DT_TO_SDT

Тип модуля: функция	Переменная	Тип	Описание
Входы	DTI	DATE	Исходные дата и время
Выходы	DT_TO_SDT	SDT	Дата и время в формате SDT
Используемые модули	YEAR_OF_DATE, MONTH_OF_DATE, , DAY_OF_MONTH, DAY_OF_WEEK		

Функция **DT_TO_SDT** конвертирует исходную дату и время **DTI** в структуру **SDT**, содержащую разряды времени в виде переменных типа **INT**.

Пример

```

VAR
  dti : DT := DT#2026-06-18-13:45:20; (* исходная дата/время *)
  sdt : OSCAT_BASIC.SDT;              (* разряды даты/времени *)
END_VAR

sdt := OSCAT_BASIC.DT_TO_SDT(DTI := dti); (* YEAR=2026, MONTH=6, DAY=18, WEEKDAY=4,
                                          HOUR=13, MINUTE=45, SECOND=20, MS=0 *)

END_PROGRAM

```

15.14 DT2_TO_SDT

Тип модуля: функция	Переменная	Тип	Описание
Входы	DI	DATE	Исходная дата
	TI	TOD	Исходное время
Выходы	DT2_TO_SDT	SDT	Дата и время в формате SDT
Используемые модули	YEAR_OF_DATE, MONTH_OF_DATE, , DAY_OF_MONTH, DAY_OF_WEEK		

Функция **DT2_TO_SDT** конвертирует исходную дату **DI** и время **TI** в структуру **SDT**, содержащую разряды времени в виде переменных типа **INT**.

Пример

```

VAR
  di      : DATE := D#2026-06-18;      (* дата *)
  ti      : TOD := TOD#13:45:20.250;  (* время *)
  sdt     : OSCAT_BASIC.SDT;           (* разряды даты/времени *)

  y       : INT;
  mo      : INT;
  da      : INT;
  wd      : INT;
  hh      : INT;
  mi      : INT;
  ss      : INT;
  ms      : INT;
END_VAR

sdt := OSCAT_BASIC.DT2_TO_SDT(
  DI := di,
  TI := ti
);

y := sdt.YEAR;      (* 2026 *)
mo := sdt.MONTH;    (* 6 *)
da := sdt.DAY;      (* 18 *)
wd := sdt.WEEKDAY;  (* 4 *)
hh := sdt.HOUR;     (* 13 *)
mi := sdt.MINUTE;   (* 45 *)
ss := sdt.SECOND;   (* 20 *)
ms := sdt.MS;       (* 250 *)

END_PROGRAM

```

15.15 EASTER

Тип модуля: функция	Переменная	Тип	Описание
Входы	year	INT	Год
Выходы	EASTER	DATE	Дата Пасхи для заданного года
Используемые модули	SET_DATE		

Функция **EASTER** возвращает дату Пасхи для заданного года. Дата Пасхи является опорной для большинства религиозных праздников.

Пример (православная Пасха)

```

VAR
  year   : INT := 2026;      (* заданный год *)
  eDate  : DATE;             (* дата Пасхи *)
END_VAR

eDate := OSCAT_BASIC.EASTER(year); (* вернёт D#2026-04-12 *)

END_PROGRAM

```

15.16 EVENTS

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	DATE_IN	DATE	Текущая дата
	ENA	BOOL	Сигнал включения ФБ
Выходы	Y	BOOL	Флаг «сегодня мероприятие»
	NAME	STRING(30)	Название мероприятия

Тип модуля: ФБ	Переменная	Тип	Описание
Входы-выходы	ELIST	ARRAY [0..49] OF HOLIDAY_DATA	Список мероприятий
Используемые модули	DAY_OF_DATE, YEAR_OF_DATE, SET_DATE		

Функциональный блок **EVENTS** используется для создания календаря мероприятий и контроля их наступления. Информация о мероприятиях содержится в массиве **ELIST**, каждый элемент которого представляет собой запись типа **HOLIDAY_DATA**. На вход **DATE_IN** поступает текущая дата. Вход **ENA** управляет работой блока (**TRUE** – блок включен, **FALSE** – отключен). Если текущей дате соответствует какое-либо мероприятие, то выход **Y** принимает значение **TRUE**, а на выходе **NAME** отображается название этого мероприятия. Каждой дате может соответствовать только одно мероприятие.

Пример

```

VAR
  fbEvents : OSCAT_BASIC.EVENTS;
  ELIST     : ARRAY[0..2] OF OSCAT_BASIC.HOLIDAY_DATA;
  dateIn    : DATE;
  ena       : BOOL;
  y         : BOOL;
  name      : STRING(30);
All_Saints: OSCAT_BASIC.HOLIDAY_DATA:=
(NAME:='All Saints Day', DAY :=1, MONTH:=11, USE :=1);
Easter: OSCAT_BASIC.HOLIDAY_DATA:=
(NAME:='Пасха', DAY :=0, MONTH:=0, USE :=1);
SAINT_TRINITY_DAY: OSCAT_BASIC.HOLIDAY_DATA:=
(NAME:='Святая троица', DAY :=49, MONTH:=0, USE :=1);
END_VAR
(* Заполняем календарь мероприятий *)
ELIST[0] := All_Saints;

ELIST[1] := Easter;

ELIST[2] := SAINT_TRINITY_DAY;
ena := TRUE;

(* Пример 1: 01.11.2026 -> All Saints Day *)
dateIn := D#2026-11-01;
fbEvents(ELIST := ELIST, DATE_IN := dateIn, ENA := ena);
y := fbEvents.Y;      (* TRUE *)
name := fbEvents.NAME; (* 'All Saints Day' *)

(* Пример 2: 31.05.2026 -> Святая троица (Пасха 12.04.2026 + 49 дней) *)
dateIn := D#2026-05-31;
fbEvents(ELIST := ELIST, DATE_IN := dateIn, ENA := ena);
y := fbEvents.Y;      (* TRUE *)
name := fbEvents.NAME; (* 'Святая троица' *)
END_PROGRAM

```

15.17 HOLIDAY

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	DATE_IN	DATE	Текущая дата
	LANGU	INT	Язык приложения
	FRIDAY	BOOL	Сигнал «пятница – праздник»
	SATURDAY	BOOL	Сигнал «суббота – праздник»
	SUNDAY	BOOL	Сигнал «воскресенье – праздник»
Выходы	Y	BOOL	Флаг «сегодня праздник»
	NAME	STRING(30)	Название праздника

Тип модуля: ФБ	Переменная	Тип	Описание
Входы-выходы	HOLIDAYS	ARRAY [0..49] OF HOLIDAY_DATA	Список праздников
Используемые модули	DAY_OF_WEEK , EASTER , YEAR_OF_DATE , SET_DATE , DATE_ADD		

Функциональный блок **HOLIDAY** используется для создания календаря праздников и контроля их наступления. Информация о мероприятиях содержится в массиве **HOLIDAYS**, каждый элемент которого представляет собой запись типа [HOLIDAY_DATA](#). На вход **DATE_IN** поступает текущая дата. Если текущей дате соответствует какой-либо праздник, то выход **Y** принимает значение **TRUE**, а на выходе **NAME** отображается название этого праздника. Если входы **FRIDAY**, **SATURDAY** и **SUNDAY** имеют значение **TRUE**, то соответствующие дни недели также считаются праздниками. Для таких случаев вход **LANGU** определяет язык, на котором будет отображаться название дня недели на выходе **NAME**.

Пример УТОЧНИТЬ РЕАЛИЗАЦИЮ LANGU!

```

VAR
  fbHoliday : OSCAT_BASIC.HOLIDAY;
  aHolidays : ARRAY[0..29] OF OSCAT_BASIC.HOLIDAY_DATA;
  dToday     : DATE := D#2026-06-19; (* пятница *)
  iLangu     : INT  := 1;           (* язык приложения *)
  xFriday    : BOOL := TRUE;
  xSaturday  : BOOL := FALSE;
  xSunday    : BOOL := FALSE;
  xY         : BOOL;
  sName      : STRING(30);
END_VAR

(* Заполняем список праздников (показываем 1 элемент, остальные можно оставить
пустыми/неиспользуемыми*)
aHolidays[0] := (
  NAME := 'All Saints Day',
  DAY  := 1,
  MONTH := 11,
  USE  := 1
);

(* Вызов FB HOLIDAY *)
fbHoliday(
  DATE_IN := dToday,
  LANGU   := iLangu,
  FRIDAY  := TRUE,
  SATURDAY := FALSE,
  SUNDAY  := FALSE
);

xY := fbHoliday.Y; (* TRUE, т.к. FRIDAY=TRUE и сегодня пятница *)
sName := fbHoliday.NAME; (* 'Friday' / 'Пятница' — зависит от LANGU
и реализации *)
END_PROGRAM

```

15.18 HOUR

Тип модуля: функция	Переменная	Тип	Описание
Входы	itod	TOD	Исходное время суток
Выходы	HOUR	INT	Число часов

Функция **HOUR** возвращает число часов, вырезанное из исходного значения времени суток **itod** типа **TOD**.

Пример


```

VAR
  t      : TOD := TOD#13:45:20; (* исходное время суток *)
  h      : INT;                  (* число часов *)
END_VAR

h := OSCAT_BASIC.HOUR(itod := t); (* вернёт 13 *)

END_PROGRAM

```

15.19 HOUR_OF_DT

Тип модуля: функция	Переменная	Тип	Описание
Входы	XDT	DT	Исходные дата и время
Выходы	HOUR_OF_DT	INT	Число часов

Функция **HOUR_OF_DT** возвращает число часов, вырезанное из исходного значения даты и времени **XDT** типа **DT**.

Пример

```

VAR
  dt : DT := DT#2026-06-18-13:45:20; (* исходные дата и время *)
  h  : INT;                          (* число часов *)
END_VAR

h := OSCAT_BASIC.HOUR_OF_DT(XDT := dt); (* вернёт 13 *)

END_PROGRAM

```

15.20 HOUR_TO_TIME

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	REAL	Количество часов
Выходы	HOUR_TO_TIME	TIME	Время

Функция **HOUR_TO_TIME** конвертирует заданное количество часов **IN** типа **REAL** (в виде числа с плавающей точкой) в значение переменной типа **TIME**.

Пример

```

VAR
  hours : REAL := 1.5; (* количество часов *)
  t      : TIME;       (* результат *)
END_VAR

t := OSCAT_BASIC.HOUR_TO_TIME(IN := hours); (* вернёт T#1h30m *)

END_PROGRAM

```

15.21 HOUR_TO_TOD

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	REAL	Количество часов
Выходы	HOUR_TO_TOD	TOD	Время суток

Функция **HOUR_TO_TOD** конвертирует заданное количество часов **IN** типа **REAL** (в виде числа с плавающей точкой) в значение переменной типа **TOD**.

Пример

```

VAR
  hours : REAL := 13.75; (* количество часов *)
  xtod  : TOD;          (* время суток *)
END_VAR

xtod := OSCAT_BASIC.HOUR_TO_TOD(IN := hours); (* вернёт TOD#13:45:00 *)

END_PROGRAM

```

15.22 JD2000

Тип модуля: функция	Переменная	Тип	Описание
Входы	DTI	DT	Заданные дата и время по грегорианскому календарю
Выходы	JD2000	REAL	Заданные дата и время по юлианскому календарю

Функция **JD2000** конвертирует значение заданной даты и времени по грегорианскому календарю **DTI** в значение по юлианскому календарю – в виде числа дней (с плавающей точкой), прошедших с полудня 1 января 2000 года.

Точкой отсчета для юлианского календаря является полдень 1 января 4713 г. до нашей эры. Поскольку в настоящий момент количество прошедших с того момента дней превышает диапазон типа **REAL**, то данная функция использует в качестве опорной точки полдень 1 января 2000 года нашей эры, которому соответствует юлианская дата JD 2451544.5. При суммировании данного значения с результатом функции можно получить абсолютную дату по юлианскому календарю.

ПРИМЕР НЕ ПРИВЕДЕН!

15.23 LEAP_DAY

Тип модуля: функция	Переменная	Тип	Описание
Входы	IDATE	DATE	Заданная дата
Выходы	LEAP_DAY	BOOL	Флаг «високосный день»

Функция **LEAP_DAY** возвращает **TRUE**, если заданная дата **IDATE** является високосным днем (29-м февраля). Во всех остальных случаях функция возвращает **FALSE**.

Пример

```

VAR
  d          : DATE := D#2024-02-29; (* заданная дата IDATE *)
  isLeapDay  : BOOL;                  (* флаг високосного дня *)
END_VAR

isLeapDay := OSCAT_BASIC.LEAP_DAY(IDATE := d); (* вернёт TRUE *)

END_PROGRAM

```

15.24 LEAP_OF_DATE

Тип модуля: функция	Переменная	Тип	Описание
Входы	IDATE	DATE	Заданная дата
Выходы	LEAP_OF_DATE	BOOL	Флаг «високосный год»

Функция **LEAP_OF_DATE** возвращает значение **TRUE**, если год заданной даты **IDATE** является високосным. Во всех остальных случаях функция возвращает **FALSE**.

Пример

```

VAR
  d      : DATE := D#2024-06-18;  (* заданная дата IDATE *)
  isLeapYear: BOOL;                (* флаг високосного года *)
END_VAR

isLeapYear := OSCAT_BASIC.LEAP_OF_DATE(IDATE := d);  (* вернёт TRUE *)

END_PROGRAM

```

15.25 LEAP_YEAR

Тип модуля: функция	Переменная	Тип	Описание
Входы	yr	INT	Заданный год
Выходы	LEAP_YEAR	BOOL	Флаг «високосный год»

Функция **LEAP_YEAR** возвращает **TRUE**, если заданный год **yr** является високосным. Во всех остальных случаях функция возвращает **FALSE**.

Пример

```

VAR
  yr      : INT := 2024;  (* заданный год *)
  isLeap   : BOOL;        (* флаг високосного года *)
END_VAR

isLeap := OSCAT_BASIC.LEAP_YEAR(yr);  (* вернёт TRUE *)

END_PROGRAM

```

15.26 LTIME_TO_UTC

Тип модуля: функция	Переменная	Тип	Описание
Входы	LTIME	DT	Местное дата и время
	DST	BOOL	Сигнал учета летнего времени
	TIME_ZONE_OFFSET	INT	Смещение местного времени относительно UTC в минутах
Выходы	LEAP_YEAR	DT	Дата и время в UTC

Функция **LTIME_TO_UTC** конвертирует значение местного даты и времени, смещенного относительно UTC на число минут, определяемым значением переменной **TIME_ZONE_OFFSET**. Если вход **DST** имеет значение **TRUE**, то при конвертации учитывается переход на летнее время. Переход на летнее время регламентируется по-разному в различных странах; функция предполагает, что при переходе на летнее время смещение местного времени относительно **UTC** увеличивается на один час.

Пример

```

VAR
  lt      : DT := DT#2026-06-18-12:00:00;  (* местное дата/время *)
  dst     : BOOL := FALSE;                  (* учет летнего времени *)
  tzOff   : INT := 180;                     (* UTC+3 = 180 минут *)
  utc     : DT;                             (* результат UTC *)
END_VAR

utc := OSCAT_BASIC.LTIME_TO_UTC(
  LOCAL_TIME      := lt,
  DST             := FALSE,
  TIME_ZONE_OFFSET := tzOff
);
(* вернёт DT#2026-06-18-09:00:00 *)

END_PROGRAM

```

15.27 MINUTE

Тип модуля: функция	Переменная	Тип	Описание
Входы	itod	TOD	Заданное время
Выходы	MINUTE	INT	Число минут

Функция **MINUTE** возвращает число минут, вырезанное из исходного значения времени суток **itod** типа **TOD**.

Пример

```
VAR
  t      : TOD := TOD#13:45:20; (* заданное время суток *)
  m      : INT;                  (* число минут *)
END_VAR

m := OSCAT_BASIC.MINUTE(itod := t); (* вернёт 45 *)

END_PROGRAM
```

15.28 MINUTE_OF_DT

Тип модуля: функция	Переменная	Тип	Описание
Входы	XDT	DT	Заданные дата и время
Выходы	MINUTE_OF_DT	INT	Число минут

Функция **MINUTE_OF_DT** возвращает число минут, вырезанное из исходного значения даты и времени **XDT** типа **DT**.

Пример

```
VAR
  ndt : DT := DT#2026-06-18-13:45:20; (* заданные дата и время *)
  m    : INT;                          (* число минут *)
END_VAR

m := OSCAT_BASIC.MINUTE_OF_DT(XDT := ndt); (* вернёт 45 *)

END_PROGRAM
```

15.29 MINUTE_TO_TIME

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	REAL	Количество минут
Выходы	MINUTE_TO_TIME	TIME	Время

Функция **MINUTE_TO_TIME** конвертирует заданное количество минут **IN** типа **REAL** (в виде числа с плавающей точкой) в значение переменной типа **TIME**.

Пример

```
VAR
  minutes : REAL := 36; (* количество минут *)
  t        : TIME;      (* результат *)
END_VAR

t := OSCAT_BASIC.MINUTE_TO_TIME(IN := minutes); (* вернёт T#36m *)

END_PROGRAM
```

15.30 MONTH_BEGIN

Тип модуля: функция	Переменная	Тип	Описание
Входы	idate	DATE	Заданная дата
Выходы	MONTH_BEGIN	DATE	Дата первого дня заданного месяца
Используемые модули	DAY_OF_MONTH		

Функция **MONTH_BEGIN** возвращает дату первого дня месяца, заданного переменной **idate** типа **DATE**.

Пример

```

VAR
  d      : DATE := D#2026-06-18;  (* заданная дата idate *)
  mb     : DATE;                  (* начало месяца *)
END_VAR

mb := OSCAT_BASIC.MONTH_BEGIN(idate := d);  (* вернёт D#2026-06-01 *)

END_PROGRAM

```

15.31 MONTH_END

Тип модуля: функция	Переменная	Тип	Описание
Входы	idate	DATE	Заданная дата
Выходы	MONTH_END	DATE	Дата последнего дня заданного месяца
Используемые модули	SET_DATE, YEAR_OF_DATE, MONTH_OF_DATE		

Функция **MONTH_END** возвращает дату последнего дня месяца, заданного переменной **idate** типа **DATE**.

Пример

```

VAR
  d      : DATE := D#2026-06-18;  (* заданная дата idate *)
  me     : DATE;                  (* конец месяца *)
END_VAR

me := OSCAT_BASIC.MONTH_END(idate := d);  (* вернёт D#2026-06-30 *)

END_PROGRAM

```

15.32 MONTH_OF_DATE

Тип модуля: функция	Переменная	Тип	Описание
Входы	idate	DATE	Заданная дата
Выходы	MONTH_OF_DATE	INT	Число месяцев
Используемые модули	DAY_OF_YEAR, LEAP_OF_DATE		

Функция **MONTH_OF_DATE** возвращает число месяцев, вырезанное из исходного значения даты **idate** типа **DATE**.

Пример

```

VAR
  d      : DATE := D#2026-06-18; (* заданная дата idate *)
  m      : INT;   (* номер месяца *)
END_VAR

m := OSCAT_BASIC.MONTH_OF_DATE(idate := d); (* вернёт 6 *)

END_PROGRAM

```

15.33 MULTIME

Тип модуля: функция	Переменная	Тип	Описание
Входы	t	TIME	Заданное время
	M	REAL	Коэффициент
Выходы	MULTIME	TIME	Отмасштабированное значение времени

Функция **MULTIME** возвращает произведение заданного времени **T** типа **TIME** на коэффициент **M** типа **REAL**.

Пример

```

VAR
  tIn  : TIME := T#2s; (* заданное время t *)
  m    : REAL := 1.5;  (* коэффициент M *)
  tOut : TIME;  (* результат *)
END_VAR

tOut := OSCAT_BASIC.MULTIME(
  t := tIn,
  M := m
); (* вернёт T#3s *)

END_PROGRAM

```

15.34 PERIOD

Тип модуля: функция	Переменная	Тип	Описание
Входы	d1	DATE	Нижняя граница интервала
	dx	DATE	Проверяемая дата
	d2	DATE	Верхняя граница интервала
Выходы	PERIOD	BOOL	Флаг принадлежности даты интервалу
Используемые модули	DAY_OF_YEAR, LEAP_OF_DATE		

Функция **PERIOD** возвращает **TRUE**, если дата **dx** принадлежит интервалу **[d1, d2]** (включая границы интервала и при условии, что $d1 < d2$). Функция не учитывает значение года, т.е. может использоваться только для дат, относящихся к одному году.

Пример

```

VAR
  d1 : DATE := D#2026-06-01; (* нижняя граница dx1 *)
  dx : DATE := D#2026-06-18; (* проверяемая дата dx *)
  d2 : DATE := D#2026-06-30; (* верхняя граница dx2 *)
  xIn: BOOL; (* результат *)
END_VAR

xIn := OSCAT_BASIC.PERIOD(
  d1 := d1,
  dx := dx,
  d2 := d2
); (* вернёт TRUE *)

END_PROGRAM

```

15.35 PERIOD2

Тип модуля: функция	Переменная	Тип	Описание
Входы	DP	ARRAY [0..3, 0..1] OF DATE	Интервалы проверки
	dx	DATE	Проверяемая дата
Выходы	PERIOD2	BOOL	Флаг принадлежности даты одному из интервалов

Функция **PERIOD2** возвращает **TRUE**, если дата **dx** принадлежит одному из 4-х интервалов, определенных массивом дат **DP**. В отличие от функции **PERIOD**, функция **PERIOD2** учитывает год даты при проверке принадлежности интервалу.

Пример

```

VAR
  DP : ARRAY[0..3, 0..1] OF DATE; (* 4 интервала: [начало, конец] *)
  dx : DATE := D#2026-06-18; (* проверяемая дата *)
  xIn: BOOL; (* результат *)
END_VAR

(* Интервал 0: июнь 2026 *)
DP[0, 0] := D#2026-06-01;
DP[0, 1] := D#2026-06-30;

(* Интервал 1: январь 2025 *)
DP[1, 0] := D#2025-01-01;
DP[1, 1] := D#2025-01-31;

(* Интервал 2: один день (пример) *)
DP[2, 0] := D#2026-12-31;
DP[2, 1] := D#2026-12-31;

(* Интервал 3: произвольный интервал (пример) *)
DP[3, 0] := D#2027-03-01;
DP[3, 1] := D#2027-03-31;

xIn := OSCAT_BASIC.PERIOD2(
  DP := DP,
  dx := dx
); (* вернёт TRUE *)

END_PROGRAM

```

15.36 REFRACTION

Тип модуля: функция	Переменная	Тип	Описание
Входы	ELEV	REAL	Истинная высота светила в градусах
Выходы	REFRACTION	REAL	Значение рефракции в градусах

Функция **REFRACTION** возвращает значение атмосферной рефракции для светила с истинной высотой **ELEV** по формуле Смартсона. Обе величины измеряются в градусах. Диапазон допустимых значений для функции: $-1.9 < \text{ELEV} < 80$

Пример

```

VAR
    elev : REAL := 5.0;
    r    : REAL;      (* рефракция, градусы *)
END_VAR

r := OSCAT_BASIC.REFRACTION(ELEV := elev); (* вернёт 0.166*)

END_PROGRAM

```

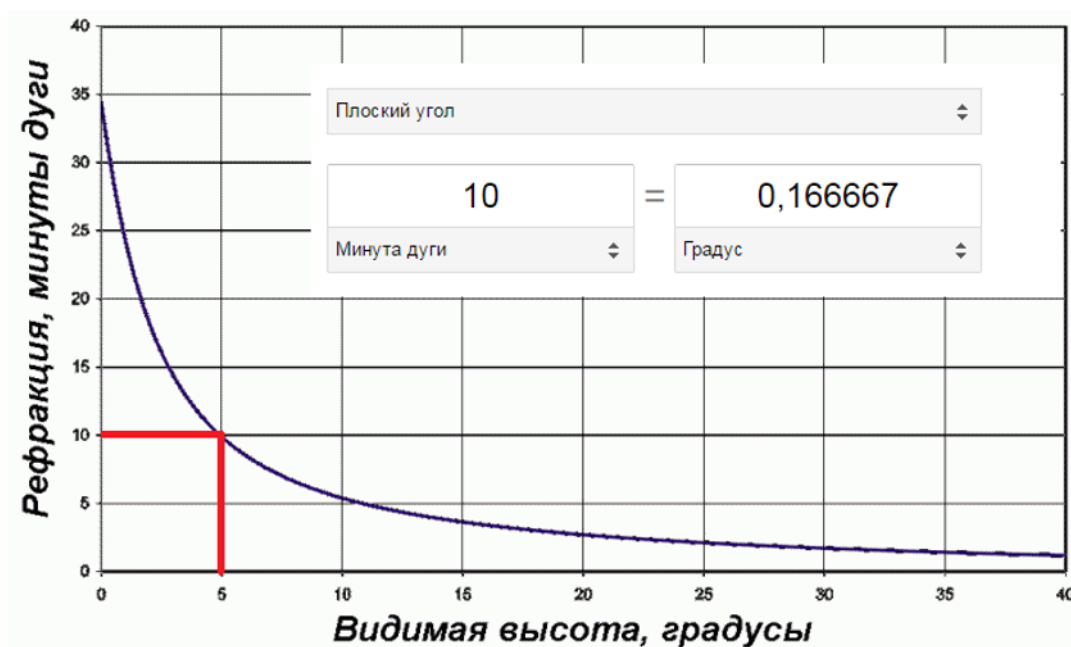


Рисунок 15.1

15.37 RTC_2

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	SET	BOOL	Сигнал установки опорной даты и времени
	SDT	DT	Дата и время начала отсчета
	SMS	INT	Значение миллисекунд
	DEN	BOOL	Сигнал включения учета летнего времени
	OFS	INT	Смещение местного времени относительно SDT в минутах
Выходы	UDT	DT	Текущие дата и время в UTC
	LOCAL_DT	DT	Местные дата и время
	DSO	BOOL	Флаг «учет летнего времени»

Тип модуля: ФБ	Переменная	Тип	Описание
	XMS	INT	Текущее значение миллисекунд
Используемые модули	RTC_MS		

Функциональный блок **RTC_2** представляет собой программные часы реального времени. При вызове блока начинается отсчет времени с момента, определяемого значениями входов **SDT** (дата и время в UTC) и **SMS** (миллисекунды). Текущее значение даты и времени в UTC подается на выход **UDT**, текущее значение миллисекунд – на выход **XMS**. По переднему фронту переменной **SET** в часы записываются новые опорные дата и время (**SDT** и **SMS**), относительно которых начинается отсчет. Если вход **DEN** имеет значение **TRUE**, то блок учитывает переход на летнее время, при этом выход **DSO** имеет значение **TRUE**. Вход **OFS** позволяет задать смещение местного времени относительно UTC в минутах. Текущие значения местного даты и времени подаются на выход **LOCAL_DT**.

Пример

```

VAR
  fbRtc  : OSCAT_BASIC.RTC_2;
  xSet   : BOOL;
  dtBase : DT    := DT#2026-06-18-12:00:00; (* импульс SET *)
  iSms   : INT   := 250;                    (* SDT (UTC) *)
  xDen   : BOOL  := FALSE;                  (* SMS *)
  iOfs   : INT   := 180;                    (* учет DST отключен *)
  udt    : DT;
  ldt    : DT;
  dso    : BOOL;
  xms    : INT;
END_VAR

(* Формируем импульс SET на один цикл *)
xSet := TRUE;
fbRtc(
  SET := xSet,
  SDT := dtBase,
  SMS := iSms,
  DEN := xDen,
  OFS := iOfs
);
udt := fbRtc.UDT; (* сразу после SET: UDT = DT#2026-06-18-12:00:00
                  (далее увеличивается) *)
ldt := fbRtc.LOCAL_DT; (* сразу после SET: LOCAL_DT = DT#2026-06-18-15:00:00
                       (при OFS=180 и DEN=FALSE) *)
dso := fbRtc.DSO; (* DSO = FALSE, т.к. DEN=FALSE *)
xms := fbRtc.XMS; (* сразу после SET: XMS = 250 (далее меняется) *)
(* Сбрасываем SET, чтобы импульс был только один цикл *)
xSet := FALSE;
END_PROGRAM

```

15.38 RTC_MS

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	SET	BOOL	Сигнал установки опорной даты и времени
	SDT	DT	Дата и время начала отсчета
	SMS	INT	Значение миллисекунд
Выходы	XDT	DT	Текущие дата и время в UTC
	XMS	INT	Текущее значение миллисекунд
Используемые модули	T_PLC_MS		

Функциональный блок **RTC_MS** представляет собой программные часы реального времени. При вызове блока начинается отсчет времени с момента, определяемого значениями входов **SDT** (дата и время в UTC) и

SMS (миллисекунды). Текущее значение даты и времени в UTC подается на выход **UDT**, текущее значение миллисекунд – на выход **XMS**. По переднему фронту переменной **SET** в часы записываются новые опорные дата и время (**SDT** и **SMS**), относительно которых начинается отсчет.

Пример

```
VAR
  fbRtcMs: OSCAT_BASIC. RTC_MS;
  xSet    : BOOL;           (* импульс SET *)
  dtBase  : DT := DT#2026-06-18-12:00:00; (* SDT (UTC) *)
  iSms    : INT := 250;     (* SMS *)
  xdt     : DT;             (* выход XDT *)
  xms     : INT;            (* выход XMS *)
END_VAR

(* Формируем импульс SET на один цикл *)
xSet := TRUE;
fbRtcMs(
  SET := xSet,
  SDT := dtBase,
  SMS := iSms
);
xdt := fbRtcMs.XDT; (* сразу после SET: XDT = DT#2026-06-18-12:00:00
                     (далее увеличивается) *)
xms := fbRtcMs.XMS; (* сразу после SET: XMS = 250 (далее меняется) *)
(* Сбрасываем SET, чтобы импульс был только один цикл *)
xSet := FALSE;
END_PROGRAM
```

15.39 SDT_TO_DATE

Тип модуля: функция	Переменная	Тип	Описание
Входы	DTI	SDT	Дата в формате SDT
Выходы	SDT_TO_DATE	DATE	Дата в формате DATE
Используемые модули	SET_DATE		

Функция **SDT_TO_DATE** конвертирует исходную дату **DTI** в формате **SDT** в переменную типа **DATE**.

Пример

```
VAR
  sdt : OSCAT_BASIC.SDT; (* дата в формате SDT *)
  d   : DATE;             (* результат DATE *)
END_VAR

(* Заполняем только разряды даты, необходимые для конвертации *)
sdt.YEAR := 2026;
sdt.MONTH := 6;
sdt.DAY := 18;

d := OSCAT_BASIC.SDT_TO_DATE(DTI := sdt); (* вернёт D#2026-06-18 *)
END_PROGRAM
```

15.40 SDT_TO_DT

Тип модуля: функция	Переменная	Тип	Описание
Входы	DTI	SDT	Дата и время в формате SDT
Выходы	SDT_TO_DT	DT	Дата и время в формате DT
Используемые модули	SET_DATE		

Функция **SDT_TO_DT** конвертирует исходные дату и время **DTI** в формате **SDT** в переменную типа **DT**.

Пример

```

VAR
    sdt : OSCAT_BASIC.SDT;    (* дата и время в формате SDT *)
    d   : DT;                (* результат DT *)
END_VAR

(* Заполняем только разряды даты и времени, необходимые для конвертации *)
sdt.YEAR      := 2026;
sdt.MONTH     := 6;
sdt.DAY       := 18;
sdt.HOUR      := 13;
sdt.MINUTE    := 45;
sdt.SECOND    := 15;

d := OSCAT_BASIC.SDT_TO_DT(DTI := sdt);  (* вернёт DT#2026-06-18-13:45:15 *)

END_PROGRAM

```

15.41 SDT_TO_TOD

Тип модуля: функция	Переменная	Тип	Описание
Входы	DTI	SDT	Время в формате SDT
Выходы	SDT_TO_TOD	TOD	Время в формате TOD
Используемые модули	SET_DATE		

Функция **SDT_TO_TOD** конвертирует исходное время суток **DTI** в формате **SDT** в переменную типа **TOD**.

Пример

```

VAR
    sdt : OSCAT_BASIC.SDT;    (* время в формате SDT *)
    d   : TOD;                (* результат TOD *)
END_VAR

(* Заполняем только разряды времени, необходимые для конвертации *)
sdt.HOUR      := 13;
sdt.MINUTE    := 45;
sdt.SECOND    := 15;
sdt.MS        := 33;

d := OSCAT_BASIC.SDT_TO_TOD(DTI := sdt);  (* вернёт TOD#13:45:15.033 *)

END_PROGRAM

```

15.42 SECOND

Тип модуля: функция	Переменная	Тип	Описание
Входы	itod	TOD	Заданное время
Выходы	SECOND	REAL	Число секунд и миллисекунд

Функция **SECOND** возвращает число секунд и миллисекунд, вырезанное из исходного значения времени суток **itod** типа **TOD**.

Пример

```

VAR
  t      : TOD := TOD#13:45:20.200; (* заданное время суток *)
  s      : REAL;                      (* число минут *)
END_VAR

s := OSCAT_BASIC.SECOND(itod := t); (* вернёт 20.2 *)

END_PROGRAM

```

15.43 SECOND_OF_DT

Тип модуля: функция	Переменная	Тип	Описание
Входы	XDT	DT	Заданные дата и время
Выходы	SECOND_OF_DT	INT	Число секунд

Функция **SECOND_OF_DT** возвращает число секунд, вырезанное из исходного значения даты и времени **XDT** типа **DT**.

Пример

```

VAR
  ndt : DT := DT#2026-06-18-13:45:20; (* заданные дата и время *)
  s    : INT;                          (* число секунд *)
END_VAR

s := OSCAT_BASIC.SECOND_OF_DT(XDT := ndt); (* вернёт 20 *)

END_PROGRAM

```

15.44 SECOND_TO_TIME

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	REAL	Количество секунд и миллисекунд
Выходы	SECOND_TO_TIME	TIME	Время

Функция **SECOND_TO_TIME** конвертирует заданное количество секунд **IN** типа **REAL** (в виде числа с плавающей точкой) в значение переменной типа **TIME**.

Пример

```

VAR
  seconds : REAL := 36; (* количество секунд *)
  t       : TIME;      (* результат *)
END_VAR

t := OSCAT_BASIC.SECOND_TO_TIME(IN := seconds); (* вернёт T#36s *)

END_PROGRAM

```

15.45 SET_DATE

Тип модуля: функция	Переменная	Тип	Описание
Входы	YEAR	INT	Год
	MONTH	INT	Месяц
	DAY	INT	День
Выходы	SET_DATE	DATE	Дата в формате DATE

Функция **SET_DATE** возвращает дату типа **DATE**, собранную из значений года **YEAR**, месяца **MONTH** и дня **DAY**. Функция не проверяет корректность входных данных, так что пользователь должен ограничить диапазон их возможных значений (например, 1-12 для **MONTH** и т.д.).

Пример

```
VAR
  y : INT := 2026;  (* YEAR *)
  m : INT := 6;     (* MONTH *)
  d : INT := 18;    (* DAY *)
  dt: DATE;         (* результат *)
END_VAR

dt := OSCAT_BASIC.SET_DATE(
  YEAR := y,
  MONTH := m,
  DAY  := d
);  (* вернёт D#2026-06-18 *)

END_PROGRAM
```

15.46 SET_DT

Тип модуля: функция	Переменная	Тип	Описание
Входы	YEAR	INT	Год
	MONTH	INT	Месяц
	DAY	INT	День
	hour	INT	Часы
	minute	INT	Минуты
	second	INT	Секунды
Выходы	SET_DT	DT	Дата и время в формате DT

Функция **SET_DT** возвращает дату и время типа **DT**, собранную из значений года **YEAR**, месяца **MONTH** и дня **DAY**, часов **hour**, минут **minute** и секунд **second**. Функция не проверяет корректность входных данных, так что пользователь должен ограничить диапазон их возможных значений (например, 1-12 для **MONTH** и т.д.).

Пример

```
VAR
  y : INT := 2026;  (* YEAR *)
  mon : INT := 6;   (* MONTH *)
  d : INT := 18;    (* DAY *)
  h : INT := 13;    (* HOUR *)
  m : INT := 45;    (* MINUTE *)
  s : INT := 20;    (* SECOND *)
  dt: DT;           (* результат *)
END_VAR

dt := OSCAT_BASIC.SET_DT(
  YEAR := y,
  MONTH := mon,
  DAY  := d,
  HOUR := h,
  MINUTE := m,
  SECOND := s
);  (* вернёт DT#2026-06-18-13:45:20 *)

END_PROGRAM
```

15.47 SET_TOD

Тип модуля: функция	Переменная	Тип	Описание
Выходы	hour	INT	Часы
	minute	INT	Минуты
	second	INT	Секунды
Выходы	SET_TOD	TOD	Дата и время в формате TOD

Функция **SET_TOD** возвращает время суток типа **TOD**, собранную из значений часов **hour**, минут **minute** и секунд **second**. Функция не проверяет корректность входных данных, так что пользователь должен ограничить диапазон их возможных значений (например, 1-24 для **hour** и т.д.).

Пример

```

VAR
  h : INT := 13;    (* HOUR *)
  m : INT := 45;    (* MINUTE *)
  s : INT := 20;    (* SECOND *)
  dt: TOD;          (* результат *)
END_VAR

dt := OSCAT_BASIC.SET_TOD(
  hour   := h,
  minute := m,
  second := s
); (* вернёт TOD#13:45:20 *)

END_PROGRAM

```

15.48 SUN_MIDDAY

Тип модуля: функция	Переменная	Тип	Описание
Входы	LON	REAL	Долгота (в градусах)
	UTC	DATE	Дата в UTC
Выходы	SUN_MIDDAY	TOD	Время истинного полудня
Используемые модули	DAY_OF_YEAR		

Функция **SUN_MIDDAY** вычисляет время истинного полудня (в UTC) для даты **UTC** на долготе **LON**.

Пример

```

VAR
  lon   : REAL := 37.6155; (* LON *)
  dUtc  : DATE := D#2016-02-17; (* UTC *)
  midday: TOD;  (* результат *)
END_VAR

midday := OSCAT_BASIC.SUN_MIDDAY(
  LON := lon,
  UTC := dUtc
); (* вернёт TOD#9:43:23:252 *)

END_PROGRAM

```

15.49 SUN_POS

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	latitude	REAL	Широта (в градусах)
	longitude	REAL	Долгота (в градусах)
	utc	DT	Дата и время в UTC

Тип модуля: ФБ	Переменная	Тип	Описание
Выходы	B	REAL	Азимут (в градусах)
	H	REAL	Высота солнца над горизонтом (в градусах)
	HR	REAL	Высота солнца над горизонтом (в градусах) с учетом рефракции
Используемые модули	MODR, RAD, DEG, REFRACTION		

Функциональный блок **SUN_POS** вычисляет азимут **B**, высоту солнца над горизонтом **H** и высоту солнца над горизонтом с учетом рефракции **HR** для заданной широты **latitude** и долготы **longitude** в момент **utc** (дата и время по UTC). Расчет рефракции производится для нормального атмосферного давления (101325 Па) и температуры 10 °C. Погрешность расчета в диапазоне дат 2000...2050 не превышает 0.1 градуса.

Пример

```

VAR
  fbSunPos : OSCAT_BASIC .SUN_POS;

  lat : REAL := 55.7522;
  lon : REAL := 37.6155;
  utc : DT   := DT#2016-02-17-5:35:22;

  B : REAL; (* азимут, градусы *)
  H : REAL; (* высота, градусы *)
  HR : REAL; (* высота с рефракцией, градусы *)
END_VAR

fbSunPos(
  latitude := lat,
  longitude := lon,
  utc      := utc
);

B := fbSunPos.B; (* 120 *)
H := fbSunPos.H; (* 4.78 *)
HR := fbSunPos.HR; (* 4.95 *)

END_PROGRAM

```

15.50 SUN_TIME

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	latitude	REAL	Широта (в градусах)
	longitude	REAL	Долгота (в градусах)
	utc	DATE	Дата в UTC
	H	REAL	Высота солнца над горизонтом (в градусах) на восходе
Выходы	MIDDAY	TOD	Время солнечного полудня (в UTC)
	SUN_RISE	TOD	Время восхода солнца (в UTC)
	SUN_SET	TOD	Время заката солнца (в UTC)
	SUN_DECLINATION	REAL	Склонение солнца над горизонтом в полдень (в градусах)
Используемые модули	SUN_MIDDAY, DAY_OF_YEAR, DEG, HOUR_TO_TIME, RAD,		

Функциональный блок **SUN_TIME** вычисляет для заданной широты **latitude**, долготы **longitude** и даты **utc** (по UTC) время восхода солнца **SUN_RISE**, время заката солнца **SUN_SET**, время солнечного полудня **MIDDAY**, а также склонение солнца над горизонтом в полдень **SUN_DECLINATION**. Функциональный блок использует алгоритм, основанный на уравнении восхода, что минимизирует нагрузку на ПЛК. Не имеет смысла вызывать данный ФБ более одного раза в день – для расчета позиции солнца в течение дня должен использоваться ФБ **SUN_POS**. ФБ возвращает корректные результаты в пределах 65° Южной широты ... 65° Северной широты ($-65 < \text{latitude} < 65$).

Вход **H** используется для задания высоты солнечного диска. По умолчанию он имеет значение -0.83 (3), которое используется при наблюдениях на земле. При наблюдениях за светилом на морском горизонте или возвышении это значение должно быть скорректировано (см. ссылку на информацию об уравнении восхода).

Пример

```

VAR
  fbSunTime : OSCAT_BASIC.SUN_TIME;
  lat       : REAL := 55.7522;
  lon       : REAL := 37.6155;
  xutc      : DATE  := D#2016-02-17;
  h         : REAL := -0.833;
  MIDDAY    : TOD;      (* время солнечного полудня *)
  SUN_RISE  : TOD;      (* время восхода солнца *)
  SUN_SET   : TOD;      (* время заката солнца *)
  SUN_DECLINATION : REAL; (*склонение солнца над горизонтом в полдень в градусах*)
END_VAR

fbSunTime(
  latitude := lat,
  longitude := lon,
  utc      := xutc,
  h        := h
);
MIDDAY      := fbSunTime.MIDDAY;      (* TOD#9:43:23.252 *)
SUN_RISE    := fbSunTime.SUN_RISE;    (* TOD#4:50:7.228 *)
SUN_SET     := fbSunTime.SUN_SET;     (* TOD#14:36:39.276 *)
SUN_DECLINATION := fbSunTime.SUN_DECLINATION; (* 22.2 *)
END_PROGRAM

```

15.51 TIMECHECK

Тип модуля: функция	Переменная	Тип	Описание
Входы	TD	TOD	Заданное время суток
	START	TOD	Нижний предел контролируемого диапазона
	STOP	TOD	Верхний предел контролируемого диапазона
Выходы	TIMECHECK	BOOL	Флаг «TD принадлежит интервалу»

Функция **TIMECHECK** возвращает **TRUE**, если заданное время суток **TD** принадлежит интервалу $\text{START} < \text{TD} < \text{STOP}$. При этом значение **START** может превышать значение **STOP** (в том случае, если в контролируемый диапазон входит часть следующих суток).

Пример


```

VAR
  td      : TOD := TOD#23:30:00;  (* проверяемое время *)

  start1  : TOD := TOD#08:00:00;
  stop1   : TOD := TOD#17:00:00;
  x1      : BOOL;                  (* результат 1 *)

  start2  : TOD := TOD#22:00:00;
  stop2   : TOD := TOD#06:00:00;
  x2      : BOOL;                  (* результат 2 *)
END_VAR

x1 := OSCAT_BASIC.TIMECHECK(
  TD      := td,
  START   := start1,
  STOP    := stop1
); (* вернёт FALSE: 23:30 не входит в 08:00..17:00 *)

x2 := OSCAT_BASIC.TIMECHECK(
  TD      := td,
  START   := start2,
  STOP    := stop2
); (* вернёт TRUE: интервал 22:00..06:00 проходит через полночь, 23:30 входит *)

END_PROGRAM

```

15.52 UTC_TO_LTIME

Тип модуля: функция	Переменная	Тип	Описание
Входы	UTC	DT	Дата и время в UTC
	DST_ENABLE	BOOL	Сигнал включения учета летнего времени
	TIME_ZONE_OFFSET	INT	Смещение местного времени относительно UTC в минутах
Выходы	UTC_TO_LTIME	DT	Местные дата и время
Используемые модули	DST		

Функция **UTC_TO_LTIME** конвертирует дату и время из UTC в местное время. Вход **TIME_ZONE_OFFSET** используется для задания значения смещения местного времени относительно **UTC** в минутах. Если вход **DST_ENABLE** имеет значение **TRUE**, то функция учитывает переход на летнее время.

Пример

```

VAR
  utc      : DT := DT#2026-06-18-09:00:00; (* дата/время в UTC *)
  dst      : BOOL := FALSE;                (* учет летнего времени *)
  tzOff    : INT := 180;                    (* UTC+3 = 180 минут *)
  lt       : DT;                            (* местное дата/время *)
END_VAR

lt := OSCAT_BASIC.UTC_TO_LTIME(
  UTC := utc,
  DST_ENABLE := FALSE,
  TIME_ZONE_OFFSET := tzOff
); (* вернёт DT#2026-06-18-12:00:00 *)

END_PROGRAM

```

15.53 WORK_WEEK

Тип модуля: функция	Переменная	Тип	Описание
Входы	idate	DATE	Заданная дата
Выходы	WORK_WEEK	INT	Номер недели в году
Используемые модули	YEAR_OF_DATE, YEAR_BEGIN, DAY_OF_WEEK, LEAP_YEAR		

Функция **WORK_WEEK** возвращает номер недели в году для заданной даты **idate**. Год начинается с 1-ой недели.

Пример

```
VAR
  d      : DATE := D#2026-06-18;  (* заданная дата idate *)
  ww     : INT;                    (* номер недели *)
END_VAR

ww := OSCAT_BASIC.WORK_WEEK(idate := d);  (* вернёт 24 *)

END_PROGRAM
```

15.54 YEAR_BEGIN

Тип модуля: функция	Переменная	Тип	Описание
Входы	y	INT	Заданный год
Выходы	YEAR_BEGIN	DATE	Дата первого дня года

Функция **YEAR_BEGIN** возвращает дату первого дня для заданного года **y**.

Пример

```
VAR
  y      : INT := 2026;           (* заданный год *)
  dBegin : DATE;                  (* результат *)
END_VAR

dBegin := OSCAT_BASIC.YEAR_BEGIN(y);  (* вернёт D#2026-01-01 *)

END_PROGRAM
```

15.55 YEAR_END

Тип модуля: функция	Переменная	Тип	Описание
Входы	y	INT	Заданный год
Выходы	YEAR_END	DATE	Дата последнего дня года

Функция **YEAR_END** возвращает дату последнего дня для заданного года **y**.

Пример

```
VAR
  y      : INT := 2026;           (* заданный год *)
  dEnd   : DATE;                  (* результат *)
END_VAR

dEnd := OSCAT_BASIC.YEAR_END(y);  (* вернёт D#2026-12-31 *)

END_PROGRAM
```

15.56 YEAR_OF_DATE

Тип модуля: функция	Переменная	Тип	Описание
Входы	IDATE	DATE	Исходная дата
Выходы	YEAR_OF_DATE	INT	Год

Функция **YEAR_OF_DATE** возвращает значение года, вырезанное из исходной даты **IDATE** типа **DATE**.

Пример

```

VAR
  d      : DATE := D#2026-06-18;  (* исходная дата IDATE *)
  y      : INT;                    (* год *)
END_VAR

y := OSCAT_BASIC.YEAR_OF_DATE(IDATE := d);  (* вернёт 2026 *)

END_PROGRAM

```


16.4 BYTE_TO_STRH

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	BYTE	Строка бит (BYTE)
Выходы	BYTE_TO_STRH	STRING(2)	Строка бит в HEX (STRING)

Функция **BYTE_TO_STRH** конвертирует значение **IN** типа **BYTE** в строку, содержащую это значение в 16-ричной системе счисления.

Пример

```
VAR
  b      : BYTE := 16#AA;
  sHex   : STRING(2);
END_VAR

sHex := OSCAT_BASIC.BYTE_TO_STRH(IN := b);  (* sHex = 'AA' *)

END_PROGRAM
```

16.5 CAPITALIZE

Тип модуля: функция	Переменная	Тип	Описание
Входы	str	STRING	Исходная строка
Выходы	CAPITALIZE	STRING	Обработанная строка
Используемые модули	TO_UPPER		

Функция **CAPITALIZE** переводит первые символы слов из строки **str** в верхний регистр. Слова должны быть разделены пробелами. Для работы с символами верхней половины таблицы ASCII глобальная переменная **EXTENDED_ASCII** должна иметь значение **TRUE**.

Пример

```
VAR
  sIn    : STRING := 'hello world';
  sOut   : STRING;
END_VAR

sOut := OSCAT_BASIC.CAPITALIZE(str := sIn);  (* sOut = 'Hello World' *)

END_PROGRAM
```

16.6 CHARCODE

Тип модуля: функция	Переменная	Тип	Описание
Входы	STR	STRING(10)	Символ/идентификатор
Выходы	CHARCODE	BYTE	Код символа/идентификатора
Используемые модули	CODE		

Функция **CHARCODE** возвращает код для заданного ASCII-символа или идентификатора **STR**. Список идентификаторов приведен в таблице ниже. В случае получения неизвестного символа или идентификатора функция возвращает **0**. Работа функции основана на использовании глобальной переменной **CHARNAMES** должна иметь значение **TRUE**.

Пример

```

VAR
  s : STRING(10) := '&'; (* одиночный символ *)
  c : BYTE;           (* код символа *)
END_VAR

c := OSCAT_BASIC.CHARCODE(STR := s); (* c = 38 (16#26) *)

END_PROGRAM

```

Обозначение	Идентификатор	Код		Обозначение	Идентификатор	Код
"	quot	34		î	Icirc	206
&	amp	38		ï	Iuml	207
	lt	60		Ð	ETH	208
<	gt	62		Ñ	Ntilde	209
€	euro	136		Ò	Ograve	210
	nbsp	160		Ó	Oacute	211
ı	iexcl	161		Ô	Ocirc	212
¢	cent	162		Õ	Otilde	213
£	pound	163		Ö	Ouml	214
¤	curren	164		×	times	215
¥	yen	165		Ø	Oslash	216
	brvbar	166		Ù	Ugrave	217
§	sect	167		Ú	Uacute	218
¨	uml	168		Û	Ucirc	219
©	copy	169		Ü	Uuml	220
ª	ordf	170		Ý	Yacute	221
«	laquo	171		ß	szlig	223
¬	not	172		à	agrave	224
-	shy	173		á	aacute	225
®	reg	174		â	acirc	226
—	macr	175		ã	atilde	227
°	deg	176		ä	auml	228
±	plusmn	177		å	aring	229
²	sup2	178		æ	aelig	230
³	sup3	179		ç	cedil	231
´	acute	180		è	egrave	232
µ	micro	181		é	eacute	233
¶	para	182		ê	ecirc	234
·	middot	183		ë	euml	235
¸	cedil	184		ì	igrave	236
¹	sup1	185		í	iacute	237
º	ordm	186		î	icirc	238
»	raquo	187		ï	iuml	239
½	frac12	189		ð	eth	240
¾	frac34	190		ñ	ntilde	241
¿	iquest	191		ò	ograve	242
À	Agrave	192		ó	oacute	243
Á	Aacute	193		ô	ocirc	244

Â	Acirc	194	õ	otilde	245
Ã	Atilde	195	ö	ouml	246
Ä	Auml	196	÷	divide	247
Å	Aring	197	ø	oslash	248
Æ	AElig	198	ù	ugrave	249
Ç	Ccedil	199	ú	uacute	250
È	Egrave	200	û	ucirc	251
É	Eacute	201	ü	uuml	252
Ê	Ecirc	202	ý	yacute	253
Ë	Euml	203	þ	thorn	254
Ì	Igrave	204	ÿ	yuml	255
Í	Iacute	205			

16.7 CHARNAME

Тип модуля: функция	Переменная	Тип	Описание
Входы	C	BYTE	Код идентификатора
Выходы	CHARNAME	STRING(10)	Идентификатор
Используемые модули	CHR_TO_STRING		

Функция **CHARNAME** возвращает идентификатор по заданному коду **C**. Список идентификаторов приведен в таблице в разделе [CHARCODE](#). Работа функции основана на использовании глобальной переменной **CHARNAMES**.

Пример

```

VAR
  c      : BYTE := 38;      (* '&' -> amp *)
  name   : STRING(10);     (* результат *)
END_VAR

name := OSCAT_BASIC.CHARNAME(C := c);  (* вернёт 'amp' *)

END_PROGRAM

```

16.8 CHR_TO_STRING

Тип модуля: функция	Переменная	Тип	Описание
Входы	C	BYTE	Код ASCII-символа
Выходы	CHR_TO_STRING	STRING(1)	ASCII-символ

Функция **CHR_TO_STRING** возвращает символ заданного ASCII-кода **C**.

Пример

```

VAR
  c      : BYTE := 38;      (* ASCII-код символа '&' *)
  s      : STRING(1);      (* результат *)
END_VAR

s := OSCAT_BASIC.CHR_TO_STRING(C := c);  (* s = '&' *)

END_PROGRAM

```

16.9 CLEAN

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	STRING	Исходная строка
	CX	STRING(80)	Допустимые символы
Выходы	CLEAN	STRING	Обработанная строка

Функция **CLEAN** вырезает из строки **IN** все символы, не входящие в строку **CX**.

Пример

```

VAR
  sIn   : STRING := 'Tel:+7(999)12-34.56';
  cx    : STRING(80) := '0123456789.'; (* допустимые символы *)
  sOut  : STRING;
END_VAR

sOut := OSCAT_BASIC.CLEAN(
  IN := sIn,
  CX := cx
); (* sOut = '79991234.56' *)

END_PROGRAM

```

16.10 CODE

Тип модуля: функция	Переменная	Тип	Описание
Входы	STR	STRING	Исходная строка
	POS	INT	Позиция символа
Выходы	CODE	BYTE	ASCII-код символа

Функция **CODE** возвращает код ASCII-символа из строки **STR** с позиции **POS**. Если POS = 0 или POS > длины строки, то функция возвращает 0.

Пример

```

VAR
  s      : STRING := 'ABC';
  pos    : INT := 2;
  c1     : BYTE; (* код символа *)
  c0     : BYTE; (* код при POS=0 *)
END_VAR

c1 := OSCAT_BASIC.CODE(
  STR := s,
  POS := pos
); (* c1 = 66 (16#42) *)

c0 := OSCAT_BASIC.CODE(
  STR := s,
  POS := 0
); (* c0 = 0 (16#00) *)

END_PROGRAM

```

16.11 COUNT_CHAR

Тип модуля: функция	Переменная	Тип	Описание
Входы	STR	STRING	Исходная строка
	chr	BYTE	ASCII-код символа
Выходы	COUNT_CHAR	INT	Кол-во символов в строке

Функция **COUNT_CHAR** возвращает число символов с ASCII-кодом **chr** в строке **STR**.

Пример

```
VAR
  s      : STRING := 'A-B-C';
  ch     : BYTE  := 16#2D;  (* '-' *)
  cnt    : INT;           (* количество вхождений *)
END_VAR

cnt := OSCAT_BASIC.COUNT_CHAR(
  STR := s,
  chr := ch
); (* cnt = 2 *)

END_PROGRAM
```

16.12 COUNT_SUBSTRING

Тип модуля: функция	Переменная	Тип	Описание
Входы	SEARCH	STRING	Искомый фрагмент
	STR	STRING	Исходная строка
Выходы	COUNT_SUBSTRING	INT	Кол-во вхождений фрагмента в строку

Функция **COUNT_SUBSTRING** возвращает число вхождений строки **SEARCH** в строку **STR**.

Пример

```
VAR
  s      : STRING := 'one two one one';
  search : STRING := 'one';
  cnt    : INT;           (* количество вхождений *)
END_VAR

cnt := OSCAT_BASIC.COUNT_SUBSTRING(
  SEARCH := search,
  STR    := s
); (* cnt = 3 *)

END_PROGRAM
```

16.13 DEC_TO_BYTE

Тип модуля: функция	Переменная	Тип	Описание
Входы	DEC	STRING(10)	Значение в символьном виде
Выходы	DEC_TO_BYTE	BYTE	Значение в целочисленном виде

Функция **DEC_TO_BYTE** конвертирует символьное значение **DEC** в целочисленное значение типа **BYTE**. Учитываются только символы '0' – '9' в пределах допустимого для типа **BYTE** диапазона '0' – '255'.

Пример

```
VAR
  strInput : STRING(10) := '255';
  bResult  : BYTE;
END_VAR

bResult := OSCAT_BASIC.DEC_TO_BYTE(strInput); (* bResult = 255 (0xFF) *)

END_PROGRAM
```

16.14 DEC_TO_DWORD

Тип модуля: функция	Переменная	Тип	Описание
Входы	DEC	STRING(10)	Значение в символьном виде
Выходы	DEC_TO_DWORD	DWORD	Значение в целочисленном виде

Функция **DEC_TO_DWORD** конвертирует символьное значение **DEC** в целочисленное значение типа **DWORD**. Учитываются только символы '0' – '9' в пределах допустимого для типа DWORD диапазона '0' – '4294967295'.

Пример

```
VAR
    strInput : STRING(10) := '4096';
    dResult  : DWORD;
END_VAR

dResult := OSCAT_BASIC.DEC_TO_DWORD(strInput); (* dResult = 4096 *)

END_PROGRAM
```

16.15 DEC_TO_INT

Тип модуля: функция	Переменная	Тип	Описание
Входы	DEC	STRING(10)	Значение в символьном виде
Выходы	DEC_TO_INT	INT	Значение в целочисленном виде

Функция **DEC_TO_DWORD** конвертирует символьное значение **DEC** в целочисленное значение типа **INT**. Учитываются только символы '0' – '9' и '-' в пределах допустимого для типа INT диапазона '-32768' – '32767'.

Пример

```
VAR
    strInput : STRING(10) := '-1024';
    iResult  : INT;
END_VAR

iResult := OSCAT_BASIC.DEC_TO_INT(strInput); (* iResult = -1024 *)

END_PROGRAM
```

16.16 DEL_CHARS

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	STRING	Исходная строка
	CX	STRING(80)	Запрещенные символы
Выходы	DEL_CHARS	STRING	Обработанная строка

Функция **DEL_CHARS** вырезает из строки **IN** все символы, входящие в строку **CX**.

Пример

```

VAR
  strIN  : STRING(80) := 'Hello, World! 123';
  strCX  : STRING(80) := 'o,';  (* что удалить*)
  strRes : STRING;
END_VAR

strRes := OSCAT_BASIC.DEL_CHARS(IN := strIN, CX := strCX);
(* результат: 'Hell Wrld' *)
END_PROGRAM

```

16.17 DT_TO_STRF

Тип модуля: функция	Переменная	Тип	Описание
Входы	DTI	DT	Дата и время
	MS	INT	Число миллисекунд
	FMT	STRING	Формат строкового представления даты и времени
	LANG	INT	Язык приложения
Выходы	DT_TO_STRF	STRING	Строка с форматированной меткой времени
Используемые модули	YEAR_OF_DATE, MONTH_OF_DATE, MONTH_TO_STRING, DAY_OF_MONTH, DAY_OF_WEEK, WEEKDAY_TO_STRING, HOUR, MINUTE, SECOND		

Функция **DT_TO_STRF** конвертирует значение даты и времени **DTI** типа **DT** в форматированную строку. Вход **MS** позволяет указать число миллисекунд. Вход **LANG** определяет язык приложения (см. глобальную переменную **LANGUAGE**). Вход **FMT** представляет собой строку, состоящую из текста и заполнителей, которая определяет форматирование даты и времени. Список заполнителей приведен ниже.

Заполнитель	Описание
#A	Полное значение года (2008)
#B	Сокращенное значение года (08)
#C	Значение месяца без ведущего нуля (5)
#D	Значение месяца с ведущим нулем (05)
#E	Сокращенное название месяца (Jan)
#F	Полное название месяца (January)
#G	Значение дня без ведущего нуля (5)
#H	Значение дня с ведущим нулем (05)
#I	Номер дня недели (1 – понедельник, 7 – воскресенье)
#J	Сокращенное название дня (Mo)
#K	Полное название дня (Monday)
#L	Утро/вечер (AM/PM)
#M	Значение часа в 24-часовом формате без ведущего нуля (2, 22)
#N	Значение часа в 24-часовом формате с ведущим нулем (02, 22)
#O	Значение часа в 12-часовом формате без ведущего нуля (1, 11)
#P	Значение часа в 12-часовом формате с ведущим нулем (01, 11)
#Q	Значение минут без ведущего нуля (5)
#R	Значение минут с ведущим нулем (05)
#S	Значение секунд без ведущего нуля (5)
#T	Значение секунд с ведущим нулем (05)
#U	Значение миллисекунд без ведущего нуля (5)

Заполнитель	Описание
#V	Значение миллисекунд с ведущим нулем (005)
#W	Значение дня с ведущим нулем и пробелом (' 05')
#X	Значение месяца с ведущим нулем и пробелом (' 05')

Пример

```

VAR
  dt      : DT := DT#2026-06-18-13:45:20;
  ms      : INT := 250;
  fmt     : STRING := '#A-#D-#H #N:#R:#T.#V'; (* YYYY-MM-DD HH:MM:SS.mmm *)
  lang    : INT := 1; (* язык EN *)
  s       : STRING;
END_VAR

s := OSCAT_BASIC.DT_TO_STRF(
  DTI := dt,
  MS  := ms,
  FMT := fmt,
  LANG := lang
);

(* s = '2026-06-18 13:45:20.250' *)

END_PROGRAM

```

16.18 DWORD_TO_STRB

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	DWORD	Набор бит
Выходы	DWORD_TO_STRB	STRING(32)	Строка бит

Функция **DWORD_TO_STRB** конвертирует переменную **IN** типа **DWORD**, представляющую собой набор бит, в строку фиксированной длины, состоящую из символов '0' и '1'. При необходимости можно подавать на вход функции переменные типа **BYTE** и **WORD**, обрезая строку до нужных размеров с помощью функции **RIGHT** из библиотеки **Standard**.

Пример

```

VAR
  dwValue   : DWORD := 16#FF00FF00; (* Шестнадцатеричное значение *)
  strBinary : STRING(32);
END_VAR

strBinary := OSCAT_BASIC.DWORD_TO_STRB(IN := dwValue); (* Результат:
                                                         '11111111000000001111111100000000'
                                                         битовое представление 16#FF00FF00*)

END_PROGRAM

```

16.19 DWORD_TO_STRF

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	DWORD	Заданное значение
	N	INT	Длина строки
Выходы	DWORD_TO_STRF	STRING(20)	Значение в виде строки заданной длины
Используемые модули	FIX		

Функция **DWORD_TO_STRF** конвертирует значение переменной **IN** типа **DWORD** в строку с заданной длиной **N**. Если **N** > кол-ва разрядов **IN**, то строка дополняется ведущими нулями. Если **N** < кол-ва разрядов **IN**, то старшие разряды обрезаются.

Пример

```
VAR
    dwValue : DWORD := 12345;
    strRes   : STRING(20);
    strRes2  : STRING(20);
END_VAR

strRes := OSCAT_BASIC.DWORD_TO_STRF(IN := dwValue, N := 8 );
(* вернет '00012345' (дополнено ведущими нулями до 8 символов) *)

strRes2 := OSCAT_BASIC.DWORD_TO_STRF(IN := dwValue, N := 3 );
(* вернет '345' (обрезаны старшие разряды) *)

END_PROGRAM
```

16.20 DWORD_TO_STRH

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	DWORD	Заданное значение
Выходы	DWORD_TO_STRH	STRING(8)	Значение в виде HEX-форматированной строки

Функция **DWORD_TO_STRH** конвертирует значение переменной **IN** типа **DWORD** в строку фиксированной длины, содержащую это значение в 16-ричной системе счисления.

Пример

```
VAR
    dwIN      : DWORD := 16#CAFEBABE;
    hexStr    : STRING(8);
END_VAR

hexStr := OSCAT_BASIC.DWORD_TO_STRH(dwIN); (* результат 'CAFEBABE' *)

END_PROGRAM
```

16.21 EXEC

Тип модуля: функция	Переменная	Тип	Описание
Входы	str	STRING	Математическое выражение
Выходы	EXEC	STRING	Результат
Используемые модули	UPPERCASE, TRIM, FIND_NONUM		

Функция **EXEC** вычисляет математическое выражение, заданное в виде строки **str**, и возвращает его результат (также в виде строки). Выражение может включать только два операнда и не должно содержать скобки. В случае некорректной операции (например, деления на ноль) функция возвращает значение **'ERROR'**.

Список возможных операторов: +, -, *, /, ^, SIN, COS, TAN, SQRT.

Пример

```

VAR
  strExp   : STRING := '15 + 27';
  strExp1  : STRING := '15/0';
  strRes   : STRING;
  strRes1  : STRING;
END_VAR

strRes := OSCAT_BASIC.EXEC(strExp); (* результат '42' *)
strRes1 := OSCAT_BASIC.EXEC(strExp1); (* результат 'ERROR' *)
END_PROGRAM

```

16.22 FILL

Тип модуля: функция	Переменная	Тип	Описание
Входы	C	BYTE	ASCII-код символа
	L	INT	Длина строки
Выходы	FILL	STRING	Строка заданной длины, заполненная заданным символом
Используемые модули	CHR_TO_STRING		

Функция **FILL** формирует строку длиной **L**, заполненную символом с ASCII-кодом **C**.

Пример

```

VAR
  fChar : BYTE := 65; (* ASCII-код 'A' *)
  len    : INT  := 5;
  res    : STRING;
END_VAR

res := OSCAT_BASIC.FILL(C := fChar, L := len); (* результат: result = "AAAAA" *)
END_PROGRAM

```

16.23 FIND_CHAR

Тип модуля: функция	Переменная	Тип	Описание
Входы	str	STRING	Исходная строка
	pos	INT	Начальная позиция для поиска
Выходы	FIND_CHAR	INT	Позиция первого символа, который не является управляющим

Функция **FIND_CHAR** анализирует исходную строку **str** с позиции **pos** и возвращает позицию первого символа, который не является управляющим. Для работы с символами верхней половины таблицы ASCII глобальная переменная **EXTENDED_ASCII** должна иметь значение **TRUE**. Управляющими считаются символы с ASCII-кодом **< 27** и **127**.

Пример

```

VAR
  testStr : STRING := '$1812';
  position : INT;
  startPos : INT := 1;
END_VAR

position := OSCAT_BASIC.FIND_CHAR(str := testStr, pos := startPos);
(* вернет 2, т.к. $18 - управляющий символ *)
END_PROGRAM

```

16.24 FIND_CTRL

Тип модуля: функция	Переменная	Тип	Описание
Входы	str	STRING	Исходная строка
	pos	INT	Начальная позиция для поиска
Выходы	FIND_CTRL	INT	Позиция первого управляющего символа в строке

Функция **FIND_CTRL** анализирует исходную строку **str** с позиции **pos** и возвращает позицию первого управляющего символа. Для работы с символами верхней половины таблицы ASCII глобальная переменная **EXTENDED_ASCII** должна иметь значение **TRUE**. Управляющими считаются символы с ASCII-кодом **< 27** и **127**.

Пример

```
VAR
  testStr  : STRING := '12$18';
  position : INT;
  startPos : INT := 1;
END_VAR

position := OSCAT_BASIC.FIND_CTRL(str := testStr, pos := startPos);
(* вернет 3, т.к. $18 - управляющий символ *)

END_PROGRAM
```

16.25 FIND_NONUM

Тип модуля: функция	Переменная	Тип	Описание
Входы	str	STRING	Исходная строка
	pos	INT	Начальная позиция для поиска
Выходы	FIND_NONUM	INT	Позиция первого символа, который не является числом

Функция **FIND_NONUM** анализирует исходную строку **str** с позиции **pos** и возвращает позицию первого символа, который не является числом (символом '1' – '9' или '.').

Пример

```
VAR
  testStr  : STRING := '+7495-727-30-16';
  position : INT;
  startPos : INT := 2;
END_VAR

position := OSCAT_BASIC.FIND_NONUM(str := testStr, pos := startPos);
(* вернет 6, т.к проверка со второго символа *)

END_PROGRAM
```

16.26 FIND_NUM

Тип модуля: функция	Переменная	Тип	Описание
Входы	str	STRING	Исходная строка
	pos	INT	Начальная позиция для поиска
Выходы	FIND_NUM	INT	Позиция первого символа, который является числом

Функция **FIND_NUM** анализирует исходную строку **str** с позиции **pos** и возвращает позицию первого символа, который является числом (символом '1' – '9' или '.').

Пример

```
VAR
    testStr : STRING := '+7495-727-30-16';
    position : INT;
    startPos : INT := 1;
END_VAR

position := OSCAT_BASIC.FIND_NUM(str := testStr, pos := startPos);
(* вернет 2 *)

END_PROGRAM
```

16.27 FINDB

Тип модуля: функция	Переменная	Тип	Описание
Входы	str1	STRING	Анализируемая строка
	str2	STRING	Искомая строка
Выходы	FINDB	INT	Позиция последнего вхождения строки

Функция **FINDB** возвращает позицию последнего вхождения строки **str2** в строку **str1**.

Пример

```
VAR
    s1 : STRING := 'one two one one';
    s2 : STRING := 'one';
    p : INT;
END_VAR

p := OSCAT_BASIC.FINDB(str1 := s1, str2 := s2); (* p = 13 *)

END_PROGRAM
```

16.28 FINDB_NONUM

Тип модуля: функция	Переменная	Тип	Описание
Входы	str	STRING	Исходная строка
Выходы	FINDB_NONUM	INT	Позиция последнего символа, который не является числом

Функция **FINDB_NONUM** возвращает позицию последнего символа строки **str**, который не является числом (символом '1' – '9' или '.').

Пример

```
VAR
    testStr : STRING := '+7495-727-30-16';
    position : INT;
END_VAR

position := OSCAT_BASIC.FINDB_NONUM(str := testStr);
(* 13 *)

END_PROGRAM
```


16.29 FINDB_NUM

Тип модуля: функция	Переменная	Тип	Описание
Входы	str	STRING	Исходная строка
Выходы	FINDB_NUM	INT	Позиция последнего символа, который является числом

Функция **FINDB_NUM** возвращает позицию последнего символа строки **str**, который является числом (символом '1' – '9' или '.').

Пример

```
VAR
    testStr : STRING := '+7495-727-30-16';
    position : INT;
END_VAR

position := OSCAT_BASIC.FINDB_NUM(str := testStr );
(* вернет 15 *)

END_PROGRAM
```

16.30 FINDP

Тип модуля: функция	Переменная	Тип	Описание
Входы	str	STRING	Анализируемая строка
	src	STRING	Искомая строка
	pos	INT	Начальная позиция для поиска
Выходы	FINDP	INT	Позиция первого вхождения искомой строки

Функция **FINDP** анализирует исходную строку **str** с позиции **pos** и возвращает позицию первого вхождения в нее строки **src**.

Пример

```
VAR
    testStr : STRING := 'hello hell';
    testStc : STRING := 'hel';
    position : INT := 2;
    res : INT;
END_VAR

res := OSCAT_BASIC.FINDP(str := testStr, src := testStc, pos := position);
(* вернет 7, т.к. проверка со второй позиции *)

END_PROGRAM
```

16.31 FIX

Тип модуля: функция	Переменная	Тип	Описание
Входы	str	STRING	Исходная строка
	C	BYTE	ASCII-код символа-заполнителя
	L	INT	Длина дополненной строки
	M	INT	Режим заполнения
Выходы	FIX	STRING	Дополненная строка
Используемые модули	FILL		

Функция **FIX** дополняет исходную строку **str** до длины **L** с помощью символа-заполнителя с ASCII-кодом **C**. Переменная **M** определяет режим дополнения:

- M=0 – строка дополняется справа;
- M=1 – строка дополняется слева;
- M=2 – строка дополняется с обеих сторон.

Если длина **str** является нечетной, то правая часть дополненной строки будет больше левой.

Если $L \leq$ длина **str**, то исходная строка не дополняется, а обрезается слева (для M=0 и M=2) или справа (для M=1).

Пример

```
VAR
  inStr : STRING := '1';
  ascii : BYTE := 16#61;
  lon   : INT := 4;
  varM  : INT;
  f0    : STRING;
  f1    : STRING;
  f2    : STRING;
END_VAR

varM := 0;

f0 := OSCAT_BASIC.FIX(str := inStr, C := ascii, L := lon, M := varM);
(* вернет '1aaa' *)

varM := 1;

f1 := OSCAT_BASIC.FIX(str := inStr, C := ascii, L := lon, M := varM);
(* вернет 'aaa1' *)

varM := 2;

f2 := OSCAT_BASIC.FIX(str := inStr, C := ascii, L := lon, M := varM);
(* вернет 'a1aa' *)

END_PROGRAM
```

16.32 FLOAT_TO_REAL

Тип модуля: функция	Переменная	Тип	Описание
Входы	FLT	STRING(20)	Значение с плавающей точкой в формате STRING
Выходы	FLOAT_TO_REAL	REAL	Значение с плавающей точкой в формате REAL
Используемые модули	DEC_TO_INT, EXPN		

Функция **FLOAT_TO_REAL** конвертирует строку **FLT**, содержащую значение с плавающей точкой, в переменную типа **REAL**.

Строка **FLT** может содержать цифры и символы «.», «,», «-», «E», «e».

Пример

```

VAR
  str1 : STRING(20) := '123.45';
  str2 : STRING(20) := '-67.89';
  str3 : STRING(20) := '1.23E-4';
  str4 : STRING(20) := '-5.67E+3';

  real1 : REAL;
  real2 : REAL;
  real3 : REAL;
  real4 : REAL;
END_VAR

real1 := OSCAT_BASIC.FLOAT_TO_REAL(str1); (* 123.45 *)
real2 := OSCAT_BASIC.FLOAT_TO_REAL(str2); (* -67.89 *)
real3 := OSCAT_BASIC.FLOAT_TO_REAL(str3); (* 0.000123 *)
real4 := OSCAT_BASIC.FLOAT_TO_REAL(str4); (* -5670.0 *)

END_PROGRAM

```

16.33 FSTRING_TO_BYTE

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	STRING(12)	Значение байта в виде форматированной строки
Выходы	FSTRING_TO_BYTE	BYTE	Значение байта
Используемые модули	BIN_TO_BYTE, OCT_TO_BYTE, HEX_TO_BYTE, DEC_TO_BYTE		

Функция **FSTRING_TO_BYTE** конвертирует форматированную строку **IN**, содержащую значение байта, в переменную типа **BYTE**. Поддерживаются следующие форматы:

- двоичный ('2#01100100');
- восьмеричный ('8#144');
- шестнадцатеричный ('16#64FS');
- десятичный ('100').

Пример

```

VAR
  str1 : STRING(12) := '2#01100100';
  str2 : STRING(12) := '8#144';
  str3 : STRING(12) := '16#64';
  str4 : STRING(12) := '100';

  real1 : BYTE;
  real2 : BYTE;
  real3 : BYTE;
  real4 : BYTE;
END_VAR

real1 := OSCAT_BASIC.FSTRING_TO_BYTE(str1); (* 100 *)
real2 := OSCAT_BASIC.FSTRING_TO_BYTE(str2); (* 100 *)
real3 := OSCAT_BASIC.FSTRING_TO_BYTE(str3); (* 100 *)
real4 := OSCAT_BASIC.FSTRING_TO_BYTE(str4); (* 100 *)

END_PROGRAM

```

16.34 FSTRING_TO_DT

Тип модуля: функция	Переменная	Тип	Описание
Входы	SDT	STRING(60)	Значение даты и времени в виде строки

Тип модуля: функция	Переменная	Тип	Описание
	FMT	STRING(60)	Формат строки
Выходы	FSTRING_TO_DT	DT	Дата и время
Используемые модули	SET_DT		

Функция **FSTRING_TO_DT** конвертирует форматированную строку **SDT**, содержащую значение даты и времени, в переменную типа **DT**. Вход **FMT** определяет используемое форматирование, т. е. представляет строку **SDT** в виде последовательности заполнителей. Список заполнителей приведен ниже.

Заполнитель	Описание
#Y	Значение года (08 или 2008).
#M	Значение месяца в виде числа (1 или 01).
#N	Значение месяца в виде названия (Jan или January). Регистр не учитывается.
#D	Значение дня (1 или 01)
#h	Значение часов (1 или 01)
#m	Значение минут (1 или 01)
#s	Значение секунд (1 или 01)
*	Заполнитель для статического текста

Пример

```

VAR
  dt1      : DT;
  dateStr1 : STRING(60) := '2026 Jul 09 at 14:30:45';
  format1  : STRING(60) := '#Y #N #D ** #h:#m:#s';
END_VAR

dt1 := OSCAT_BASIC.FSTRING_TO_DT(SDT := dateStr1, FMT := format1);
// dt1 = 2026-07-09-14:30:45

END_PROGRAM

```

16.35 FSTRING_TO_DWORD

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	STRING(40)	Значение даты и времени в виде форматированной строки
Выходы	FSTRING_TO_DWORD	DWORD	Значение DWORD
Используемые модули	BIN_TO_DWORD, OCT_TO_DWORD, HEX_TO_DWORD, DEC_TO_DWORD		

Функция **FSTRING_TO_DWORD** конвертирует форматированную строку **IN**, содержащую значение **DWORD**, в переменную типа **DWORD**. Поддерживаются следующие форматы:

- двоичный ('2#01100100');
- восьмеричный ('8#144');
- шестнадцатеричный ('16#64FS');
- десятичный ('100').

Пример

```

VAR
  sBin : STRING(40) := '2#01100100';
  sOct : STRING(40) := '8#144';
  sHex : STRING(40) := '16#64';
  sDec : STRING(40) := '100';

  dw1 : DWORD;
  dw2 : DWORD;
  dw3 : DWORD;
  dw4 : DWORD;
END_VAR

dw1 := OSCAT_BASIC.FSTRING_TO_DWORD(IN := sBin); (* dw1 = 100 (16#00000064) *)
dw2 := OSCAT_BASIC.FSTRING_TO_DWORD(IN := sOct); (* dw2 = 100 (16#00000064) *)
dw3 := OSCAT_BASIC.FSTRING_TO_DWORD(IN := sHex); (* dw3 = 100 (16#00000064) *)
dw4 := OSCAT_BASIC.FSTRING_TO_DWORD(IN := sDec); (* dw4 = 100 (16#00000064) *)

END_PROGRAM

```

16.36 FSTRING_TO_MONTH

Тип модуля: функция	Переменная	Тип	Описание
Входы	MTH	STRING(20)	Значение месяца в виде строки
	LANG	INT	Язык приложения
Выходы	FSTRING_TO_MONTH	INT	Значение месяца в виде числа
Используемые модули	TRIM, CAPITALIZE, LOWERCASE		

Функция **FSTRING_TO_MONTH** конвертирует строку, содержащую значение месяца в виде числа или названия, в переменную типа **INT**, содержащую номер месяца. Вход **LANG** определяет язык приложения (см. глобальную переменную **LANGUAGE**).

Пример

```

VAR
  month1 : INT;
  month2 : INT;
END_VAR

month1 := OSCAT_BASIC.FSTRING_TO_MONTH(MTH := 'January', LANG := 1); // 1
month2 := OSCAT_BASIC.FSTRING_TO_MONTH(MTH := 'März', LANG := 2); // 3

END_PROGRAM

```

16.37 FSTRING_TO_WEEK

Тип модуля: функция	Переменная	Тип	Описание
Входы	WEEK	STRING(20)	Название или номер для недели
	LANG	INT	Язык приложения
Выходы	FSTRING_TO_WEEK	BYTE	Номер дня недели в виде битовой маски
Используемые модули	FSTRING_TO_WEEKDAY		

Функция **FSTRING_TO_WEEK** конвертирует строку **WEEK**, содержащую номер или название (полное или сокращенное, регистронезависимое) дня недели, в битовую маску (типа **BYTE**), где бит, имеющий значение **TRUE**, определяет текущий день недели (**бит 6** – понедельник, **бит 0** – воскресенье). Вход **LANG** определяет язык приложения (см. глобальную переменную **LANGUAGE**).

Пример

```

VAR
    weekMask1 : BYTE;
    weekMask2 : BYTE;
END_VAR

weekMask1 := OSCAT_BASIC.FSTRING_TO_WEEK(WEEK := 'Monday', LANG := 1);
(* 2#00000010 Понедельник*)

weekMask2 := OSCAT_BASIC.FSTRING_TO_WEEK(WEEK := 'Sonntag', LANG := 2);
(* 2#00000001 Воскресенье*)

END_PROGRAM

```

16.38 FSTRING_TO_WEEKDAY

Тип модуля: функция	Переменная	Тип	Описание
Входы	WEEK	STRING(20)	Название или номер для недели
	LANG	INT	Язык приложения
Выходы	FSTRING_TO_WEEKDAY	INT	Номер дня недели
Используемые модули	TRIM, CAPITALIZE, LOWERCASE		

Функция **FSTRING_TO_WEEKDAY** конвертирует строку **WEEK**, содержащую номер или название (полное или сокращенное, регистронезависимое) дня недели в переменную типа **INT**, содержащую номер дня недели (1 – понедельник, 7 – воскресенье). Вход **LANG** определяет язык приложения (см. глобальную переменную **LANGUAGE**).

Пример

```

VAR
    wDay1 : INT;
    wDay2 : INT;
END_VAR

wDay1 := OSCAT_BASIC.FSTRING_TO_WEEKDAY(WDAY := 'Monday', LANG := 1);
(* результат 1*)

wDay2 := OSCAT_BASIC.FSTRING_TO_WEEKDAY(WDAY := 'Sonntag', LANG := 2);
(* результат 7 *)

END_PROGRAM

```

16.39 HEX_TO_BYTE

Тип модуля: функция	Переменная	Тип	Описание
Входы	HEX	STRING(5)	HEX-значение в виде строки
Выходы	HEX_TO_BYTE	BYTE	HEX-значение в виде байта

Функция **HEX_TO_BYTE** конвертирует строку **HEX**, содержащую значение в 16-ричной системе счисления, в соответствующее значение типа **BYTE**. Исходная строка должна содержать только числа и символы от 'A' до 'F' (в любом регистре).

Пример

```

VAR
  str1 : STRING(5) := '1A';
  str2 : STRING(5) := 'ED';

  res1 : BYTE;
  res2 : BYTE;
END_VAR

res1 := OSCAT_BASIC.HEX_TO_BYTE(str1);  (* 16#1A *)
res2 := OSCAT_BASIC.HEX_TO_BYTE(str2);  (* 16#ED *)

END_PROGRAM

```

16.40 HEX_TO_DWORD

Тип модуля: функция	Переменная	Тип	Описание
Входы	HEX	STRING(20)	HEX-значение в виде строки
Выходы	HEX_TO_DWORD	DWORD	HEX-значение в виде DWORD

Функция **HEX_TO_DWORD** конвертирует строку **HEX**, содержащую значение в 16-ричной системе счисления, в соответствующее значение типа **DWORD**. Исходная строка должна содержать только числа и символы от 'A' до 'F' (в любом регистре).

Пример

```

VAR
  str1 : STRING(20) := '1A1A2B3C';
  str2 : STRING(20) := 'BABE';

  res1 : DWORD;
  res2 : DWORD;
END_VAR

res1 := OSCAT_BASIC.HEX_TO_DWORD(str1);  (* 16#1A1A2B3C *)
res2 := OSCAT_BASIC.HEX_TO_DWORD(str2);  (* 16#BABE *)

END_PROGRAM

```

16.41 IS_ALNUM

Тип модуля: функция	Переменная	Тип	Описание
Входы	str	STRING	Исходная строка
Выходы	IS_ALNUM	BOOL	Флаг «только буквы и цифры»
Используемые модули	ISC_ALPHA, ISC_NUM		

Функция **IS_ALNUM** возвращает **TRUE**, если исходная строка **str** состоит только из букв и цифр. Если в строку входит хотя бы один символ, не попадающий в эти категории (например, пробел), то функция возвращает **FALSE**.

Пример

```

VAR
  test1 : STRING := 'Hello123';
  test2 : STRING := 'Test@123';

  result1 : BOOL;
  result2 : BOOL;
END_VAR

result1 := OSCAT_BASIC.IS_ALNUM(str := test1); (* TRUE - только буквы и цифры *)
result2 := OSCAT_BASIC.IS_ALNUM(str := test2); (* FALSE - содержит @ *)

END_PROGRAM

```

16.42 IS_ALPHA

Тип модуля: функция	Переменная	Тип	Описание
Входы	str	STRING	Исходная строка
Выходы	IS_ALPHA	BOOL	Флаг «только буквы»
Используемые модули	ISC_ALPHA		

Функция **IS_ALPHA** возвращает **TRUE**, если исходная строка **str** состоит только из букв. Во всех остальных случаях функция возвращает **FALSE**. Для работы с символами верхней половины таблицы ASCII глобальная переменная **EXTENDED_ASCII** должна иметь значение **TRUE**.

Пример

```

VAR
  test1 : STRING := 'Hello';
  test2 : STRING := 'Hello123';

  result1 : BOOL;
  result2 : BOOL;
END_VAR

result1 := OSCAT_BASIC.IS_ALPHA(test1); (* TRUE *)
result2 := OSCAT_BASIC.IS_ALPHA(test2); (* FALSE *)

END_PROGRAM

```

16.43 IS_CC

Тип модуля: функция	Переменная	Тип	Описание
Входы	str	STRING	Исходная строка
	cmp	STRING	Строка с разрешенными символами
Выходы	IS_CC	BOOL	Флаг «только разрешенные символы»

Функция **IS_CC** возвращает **TRUE**, если исходная строка **str** состоит только из символов, включенных в строку **cmp**. Во всех остальных случаях функция возвращает **FALSE**.

Пример


```

VAR
  test1 : STRING := 'test';
  test2 : STRING := 'Test';
  cmp0  : STRING := 'tse';

  result1 : BOOL;
  result2 : BOOL;
END_VAR

result1 := OSCAT_BASIC.IS_CC(str := test1, cmp := cmp0); (* TRUE *)
result2 := OSCAT_BASIC.IS_CC(str := test2, cmp := cmp0); (* FALSE *)

END_PROGRAM

```

16.44 IS_CTRL

Тип модуля: функция	Переменная	Тип	Описание
Входы	str	STRING	Исходная строка
Выходы	IS_CTRL	BOOL	Флаг «только управляющие символы»
Используемые модули	ISC_CTRL		

Функция **IS_CTRL** возвращает **TRUE**, если исходная строка **str** состоит только из управляющих символов. Управляющими считаются символы с ASCII-кодом **< 27** и **127**.

Пример

```

VAR
  str1 : STRING := '$18';
  str2 : STRING := '11$18';

  res1 : BOOL;
  res2 : BOOL;
END_VAR

res1 := OSCAT_BASIC.IS_CTRL(str := str1);
(* TRUE *)
res2 := OSCAT_BASIC.IS_CTRL(str := str2);
(* FALSE *)

END_PROGRAM

```

16.45 IS_HEX

Тип модуля: функция	Переменная	Тип	Описание
Входы	str	STRING	Исходная строка
Выходы	IS_HEX	BOOL	Флаг «только HEX-символы»
Используемые модули	ISC_HEX		

Функция **IS_HEX** возвращает **TRUE**, если исходная строка **str** состоит только из HEX-символов ('0' – '9', 'a' – 'f', 'A' – 'F'). Во всех остальных случаях функция возвращает **FALSE**.

Пример

```

VAR
  str1 : STRING := '3AEB1C';
  str2 : STRING := '2XL';

  res1 : BOOL;
  res2 : BOOL;
END_VAR

res1 := OSCAT_BASIC.IS_HEX(str := str1);
(* TRUE *)
res2 := OSCAT_BASIC.IS_HEX(str := str2);
(* FALSE *)

END_PROGRAM

```

16.46 IS_LOWER

Тип модуля: функция	Переменная	Тип	Описание
Входы	str	STRING	Исходная строка
Выходы	IS_LOWER	BOOL	Флаг «все символы в нижнем регистре»
Используемые модули	ISC_LOWER		

Функция **IS_LOWER** возвращает **TRUE**, если исходная строка **str** состоит только из букв в нижнем регистре. Во всех остальных случаях функция возвращает **FALSE**. Для работы с символами верхней половины таблицы ASCII глобальная переменная **EXTENDED_ASCII** должна иметь значение **TRUE**.

Пример

```

VAR
  str1 : STRING := 'test';
  str2 : STRING := 'Test';

  str3 : STRING := 'test1';

  res1 : BOOL;
  res2 : BOOL;
  res3 : BOOL;
END_VAR

res1 := OSCAT_BASIC.IS_LOWER(str := str1);
(* TRUE *)
res2 := OSCAT_BASIC.IS_LOWER(str := str2);
(* FALSE *)
res3 := OSCAT_BASIC.IS_LOWER(str := str3);
(* FALSE *)

END_PROGRAM

```

16.47 IS_NCC

Тип модуля: функция	Переменная	Тип	Описание
Входы	str	STRING	Исходная строка
	cmp	STRING	Строка с запрещенными символами.
Выходы	IS_NCC	BOOL	Флаг «без запрещенных символов».

Функция **IS_NCC** возвращает **TRUE**, если исходная строка **str** не включает в себя ни один символ из строки **cmp**. Во всех остальных случаях функция возвращает **FALSE**.

Пример

```

VAR
  str0 : STRING := 'test';
  cmp1 : STRING := 'abcd';

  cmp2 : STRING := 'efgd';

  res1 : BOOL;
  res2 : BOOL;

END_VAR

res1 := OSCAT_BASIC.IS_NCC(str := str0, cmp := cmp1);
(* TRUE *)
res2 := OSCAT_BASIC.IS_NCC(str := str0, cmp := cmp2);
(* FALSE *)
END_PROGRAM

```

16.48 IS_NUM

Тип модуля: функция	Переменная	Тип	Описание
Входы	str	STRING	Исходная строка
Выходы	IS_NUM	BOOL	Флаг «только числа»
	ISC_NUM		

Функция **IS_NUM** возвращает **TRUE**, если исходная строка **str** состоит только из чисел. Во всех остальных случаях функция возвращает **FALSE**.

Пример

```

VAR
  str1 : STRING := '123';
  str2 : STRING := 'a123';

  res1 : BOOL;
  res2 : BOOL;

END_VAR

res1 := OSCAT_BASIC.IS_NUM(str := str1);
(* TRUE *)
res2 := OSCAT_BASIC.IS_NUM(str := str2);
(* FALSE *)
END_PROGRAM

```

16.49 IS_UPPER

Тип модуля: функция	Переменная	Тип	Описание
Входы	str	STRING	Исходная строка
Выходы	IS_UPPER	BOOL	Флаг «все символы в верхнем регистре»
Используемые модули	ISC_UPPER		

Функция **IS_UPPER** возвращает **TRUE**, если исходная строка **str** состоит только из букв в верхнем регистре. Во всех остальных случаях функция возвращает **FALSE**. Для работы с символами верхней половины таблицы ASCII глобальная переменная **EXTENDED_ASCII** должна иметь значение **TRUE**.

Пример

```

VAR
  str1 : STRING := 'TEST';
  str2 : STRING := 'Test';

  str3 : STRING := 'TEST1';

  res1 : BOOL;
  res2 : BOOL;
  res3 : BOOL;
END_VAR

res1 := OSCAT_BASIC.IS_UPPER(str := str1);
(* TRUE *)
res2 := OSCAT_BASIC.IS_UPPER(str := str2);
(* FALSE *)
res3 := OSCAT_BASIC.IS_UPPER(str := str3);
(* FALSE *)

END_PROGRAM

```

16.50 ISC_ALPHA

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	BYTE	ASCII-код символа
Выходы	ISC_ALPHA	BOOL	Флаг «только буквы»

Функция **ISC_ALPHA** возвращает **TRUE**, если символ с ASCII-кодом **IN** является буквой. Во всех остальных случаях функция возвращает **FALSE**. Для работы с символами верхней половины таблицы ASCII глобальная переменная **EXTENDED_ASCII** должна иметь значение **TRUE**.

Пример

```

VAR
  test1 : BYTE := 16#61; (* английский язык *)
  test2 : BYTE := 16#E0; (* русский язык *)
  test3 : BYTE := 16#21;

  res1 : BOOL;
  res2 : BOOL;
  res3 : BOOL;
END_VAR

res1 := OSCAT_BASIC.ISC_ALPHA(IN := test1); (* TRUE *)
res2 := OSCAT_BASIC.ISC_ALPHA(IN := test2); (* TRUE *)
res3 := OSCAT_BASIC.ISC_ALPHA(IN := test3); (* FALSE *)

END_PROGRAM

```

16.51 ISC_CTRL

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	BYTE	ASCII-код символа
Выходы	ISC_CTRL	BOOL	Флаг «управляющий символ»

Функция **ISC_CTRL** возвращает **TRUE**, если символ с ASCII-кодом **IN** является управляющим символом. Управляющими считаются символы с ASCII-кодом < 27 (16#1B) и 127 (16#7F). Во всех остальных случаях функция возвращает **FALSE**.

Пример

```

VAR
  test1 : BYTE := 16#61;
  test2 : BYTE := 16#14;

  res1 : BOOL;
  res2 : BOOL;
END_VAR

res1 := OSCAT_BASIC.ISC_CTRL(IN := test1); (* FALSE *)
res2 := OSCAT_BASIC.ISC_CTRL(IN := test2); (* TRUE *)

END_PROGRAM

```

16.52 ISC_HEX

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	BYTE	ASCII-код символа
Выходы	ISC_HEX	BOOL	Флаг «HEX-символ»

Функция **ISC_HEX** возвращает **TRUE**, если символ с ASCII-кодом **IN** является HEX-символом ('0' – '9', 'a' – 'f', 'A' – 'F'). Во всех остальных случаях функция возвращает **FALSE**.

Пример

```

VAR
  str1 : BYTE := 16#46; (* F *)
  str2 : BYTE := 16#47; (* G *)

  res1 : BOOL;
  res2 : BOOL;
END_VAR

res1 := OSCAT_BASIC.ISC_HEX(IN := str1);
(* TRUE *)
res2 := OSCAT_BASIC.ISC_HEX(IN := str2);
(* FALSE *)

END_PROGRAM

```

16.53 ISC_LOWER

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	BYTE	ASCII-код символа
Выходы	ISC_LOWER	BOOL	Флаг «буква в нижнем регистре»

Функция **ISC_LOWER** возвращает **TRUE**, если символ с ASCII-кодом **IN** является буквой в нижнем регистре. Во всех остальных случаях функция возвращает **FALSE**. Для работы с символами верхней половины таблицы ASCII глобальная переменная **EXTENDED_ASCII** должна иметь значение **TRUE**.

Пример

```

VAR
  str1  : BYTE := 16#46; (* F *)
  str2  : BYTE := 16#66; (* f *)

  res1  : BOOL;
  res2  : BOOL;
END_VAR

res1 := OSCAT_BASIC.ISC_LOWER(IN := str1);
(* FALSE *)
res2 := OSCAT_BASIC.ISC_LOWER(IN := str2);
(* TRUE *)

END_PROGRAM

```

16.54 ISC_NUM

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	BYTE	ASCII-код символа
Выходы	ISC_NUM	BOOL	Флаг «цифра»

Функция **ISC_NUM** возвращает **TRUE**, если символ с ASCII-кодом **IN** является цифрой. Во всех остальных случаях функция возвращает **FALSE**.

Пример

```

VAR
  str1  : BYTE := 16#31; (* 1 *)
  str2  : BYTE := 16#46; (* F *)

  res1  : BOOL;
  res2  : BOOL;
END_VAR

res1 := OSCAT_BASIC.ISC_NUM(IN := str1);
(* TRUE *)
res2 := OSCAT_BASIC.ISC_NUM(IN := str2);
(* FALSE *)

END_PROGRAM

```

16.55 ISC_UPPER

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	BYTE	ASCII-код символа
Выходы	ISC_UPPER	BOOL	Флаг «буква в верхнем регистре»

Функция **ISC_UPPER** возвращает **TRUE**, если символ с ASCII-кодом **IN** является буквой в верхнем регистре. Во всех остальных случаях функция возвращает **FALSE**. Для работы с символами верхней половины таблицы ASCII глобальная переменная **EXTENDED_ASCII** должна иметь значение **TRUE**.

Пример

```

VAR
  str1  : BYTE := 16#46; (* F *)
  str2  : BYTE := 16#66; (* f *)

  res1  : BOOL;
  res2  : BOOL;
END_VAR

res1 := OSCAT_BASIC.ISC_UPPER(IN := str1);
(* TRUE *)
res2 := OSCAT_BASIC.ISC_UPPER(IN := str2);
(* FALSE *)

END_PROGRAM

```

16.56 LOWERCASE

Тип модуля: функция	Переменная	Тип	Описание
Входы	str	STRING	Исходная строка
Выходы	LOWERCASE	STRING	Строка в нижнем регистре

Функция **LOWERCASE** конвертирует исходную строку **str** в соответствующую строку нижнего регистра. Для работы с символами верхней половины таблицы ASCII глобальная переменная **EXTENDED_ASCII** должна иметь значение **TRUE**.

Пример

```

VAR
  TestVar  : STRING := ' tEsT '; // исходная строка
  ResVar   : STRING;
END_VAR

ResVar := OSCAT_BASIC.LOWERCASE( str := TestVar );
(* результат 'test' *)

END_PROGRAM

```

16.57 MESSAGE_8

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN1...IN8	BOOL	Сигналы генерации сообщений
Выходы	M	STRING	Текущее сообщение
Параметры	S1...S8	STRING	Сообщения 1...8

Функциональный блок **MESSAGE_8** используется для генерации на выходе одного из 8-ми заданных сообщений по фронту соответствующего логического сигнала. Если логический сигнал **IN1... IN8** имеет значение **TRUE**, то на выход **M** поступает соответствующее ему сообщение. Если несколько сигналов имеют значение **TRUE**, то на выход поступает сообщение с наименьшим номером (т.е. если **IN1=IN2=IN3=TRUE**, то на выходе будет сообщение **S1**; если **S3=S5=S8=TRUE**, то на выходе будет сообщение **S3** и т.д.). Если все сигналы имеют значение **FALSE**, то на выходе – пустая строка.

16.58 MESSAGE_4R

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	M0...M3	STRING	Сообщения 0...3
	MM	INT	Число используемых сообщений
	ENQ	BOOL	Сигнал запуска ФБ
	CLK	BOOL	Сигнал переключения сообщений

Тип модуля: ФБ	Переменная	Тип	Описание
	T1	TIME	Время автоматического переключения сообщений
Выходы	MX	STRING	Текущее сообщение
	MN	INT	Номер текущего сообщения (0...4)
	TR	BOOL	Флаг «новое сообщение»
Используемые модули	INC1		

Функциональный блок **MESSAGE_4R** используется для последовательной циклической генерации на выходе 4-х заданных информационных сообщений по фронту логического сигнала. Когда на входе **CLK** проходит импульс по переднему фронту, то на выход **MX** поступает сообщение **M0**. При следующем импульсе на входе **CLK** на выход попадет сообщение **M1** и т.д. Вход **MM** определяет число используемых сообщений (например, при MM=2 на выход будут поступать только сообщения M0 и M1). Вход **ENQ** используется для запуска/остановки работы блока (блок работает только при ENQ=TRUE). Вход **T1** используется для автоматического переключения сообщений на выходе **MX** – если в течение времени **T1** вход **CLK** сохраняет значение **TRUE**, то на выход **MX** поступает следующее сообщение. Выход **MN** содержит номер текущего сообщения на выходе (**0** – M0, **3** – M3). На выходе **TR** генерируется единичный импульс при смене сообщения.

16.59 MIRROR

Тип модуля: функция	Переменная	Тип	Описание
Входы	str	STRING	Исходная строка
Выходы	MIRROR	STRING	Зеркальная строка

Функция **MIRROR** конвертирует исходную строку **str** в строку с зеркальным расположением символов.

Пример

```
VAR
    str1 : STRING := 'this is test';
    res  : STRING;
;
END_VAR

res := OSCAT_BASIC.MIRROR(str := str1);
(* 'tset si siht' *)

END_PROGRAM
```

16.60 MONTH_TO_STRING

Тип модуля: функция	Переменная	Тип	Описание
Входы	MTH	INT	Номер месяца
	LANG	INT	Язык приложения
	LX	INT	Тип названия (полное/сокращенное)
Выходы	MONTH_TO_STRING	STRING(10)	Название месяца

Функция **MONTH_TO_STRING** возвращает название месяца с номером **MTH**. Вход **LANG** определяет язык приложения (см. глобальную переменную [LANGUAGE](#)). Вход **LX** определяет тип названия месяца: **0** – полное (January), **3** – сокращенное (Jan).

Пример


```

VAR
  month : INT := 3;
  res1  : STRING(10);
  res2  : STRING(10);
END_VAR

res1 := OSCAT_BASIC.MONTH_TO_STRING(MTH := month, LANG := 1, LX := 0); (* March *)
res2 := OSCAT_BASIC.MONTH_TO_STRING(MTH := month, LANG := 2, LX := 3); (* Mrz *)

END_PROGRAM

```

16.61 OCT_TO_BYTE

Тип модуля: функция	Переменная	Тип	Описание
Входы	OCT	STRING(10)	Восьмеричное значение в символьном виде
Выходы	OCT_TO_BYTE	BYTE	Восьмеричное значение в целочисленном виде

Функция **OCT_TO_BYTE** конвертирует восьмеричное символьное значение **OCT** в целочисленное восьмеричное значение типа **BYTE**. Учитываются только символы '0' – '7' в пределах допустимого для типа **BYTE** диапазона '0' – '255'.

Пример

```

VAR
  oct1 : STRING(10) := '11';
  oct2 : STRING(10) := '400';
  byte1 : BYTE;
  byte2 : BYTE;
END_VAR

byte1 := OSCAT_BASIC.OCT_TO_BYTE(OCT := oct1); (* 9 *)
byte2 := OSCAT_BASIC.OCT_TO_BYTE(OCT := oct2); (* 0 - переполнение *)

END_PROGRAM

```

16.62 OCT_TO_DWORD

Тип модуля: функция	Переменная	Тип	Описание
Входы	OCT	STRING(20)	Восьмеричное значение в символьном виде
Выходы	OCT_TO_DWORD	DWORD	Восьмеричное значение в целочисленном виде

Функция **OCT_TO_DWORD** конвертирует восьмеричное символьное значение **OCT** в целочисленное восьмеричное значение типа **BYTE**. Учитываются только символы '0' – '7' в пределах допустимого для типа **DWORD** диапазона '0' – '4294967295'.

Пример

```

VAR
  oct1 : STRING(20) := '1265';
  oct2 : STRING(20) := '8989';
  res1 : DWORD;
  res2 : DWORD;
END_VAR

res1 := OSCAT_BASIC.OCT_TO_DWORD(OCT := oct1); (* 693 *)
res2 := OSCAT_BASIC.OCT_TO_DWORD(OCT := oct2); (* 0 - недопустимые символы *)

END_PROGRAM

```

16.63 REAL_TO_STRF

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	REAL	Значение с плавающей точкой в формате REAL
	N	INT	Кол-во знаков после запятой
	D	STRING(1)	Символ-разделитель
Выходы	REAL_TO_STRF	STRING(20)	Значение с плавающей точкой в формате STRING
Используемые модули	EXP10		

Функция **REAL_TO_STRF** конвертирует переменную **IN** типа **REAL** в строку, содержащую соответствующее значение с плавающей точкой. Вход **N** определяет количество знаков после запятой, до которого будет округлено значение **IN**; он может принимать значения из диапазона **0...7** – это связано с максимальной возможной точностью для типа **REAL**. Вход **D** определяет символ-разделитель для целой и дробной части.

Пример

```

VAR
  rVal  : REAL := 3.1415;
  rD1   : STRING(1) := '.';
  rD2   : STRING(1) := '_';
  res1  : STRING(20);
  res2  : STRING(20);
END_VAR

res1 := OSCAT_BASIC.REAL_TO_STRF(IN := rVal, N := 2, D := rD1); (* 3.14 *)
res2 := OSCAT_BASIC.REAL_TO_STRF(IN := rVal, N := 5, D := rD2); (* 3_14150 *)

END_PROGRAM

```

16.64 REPLACE_ALL

Тип модуля: функция	Переменная	Тип	Описание
Входы	str	STRING	Исходная строка
	src	STRING	Искомая строка
	rep	STRING	Замещающая строка
Выходы	REPLACE_ALL	STRING	Обработанная строка
Используемые модули	FINDP		

Функция **REPLACE_ALL** осуществляет поиск в исходной строке **str** всех текстовых фрагментов **src** и заменяет их на **rep**.

Пример

```

VAR
  InputText  : STRING := 'Hello, world! Hello, everyone!'; (*Исходная строка*)
  SearchStr  : STRING := 'Hello';                          (*Искомая строка*)
  ReplaceStr  : STRING := 'Hi';                             (*Замещающая строка*)
  ResultText : STRING;
END_VAR

ResultText := OSCAT_BASIC.REPLACE_ALL( str := InputText,
                                       src := SearchStr,
                                       rep := ReplaceStr );

(* результат 'Hi, world! Hi, everyone!')

END_PROGRAM

```

16.65 REPLACE_CHARS

Тип модуля: функция	Переменная	Тип	Описание
Входы	str	STRING	Исходная строка
	SRC	STRING	Строка с набором искомых символов
	REP	STRING	Строка с набором замещающих символов
Выходы	REPLACE_CHARS	STRING	Обработанная строка

Функция **REPLACE_CHARS** осуществляет поиск в исходной строке **str** всех символов, входящих в строку **SRC**, после чего замещает их символами с идентичной (для строки **SRC**) позицией из строки **REP**.

Пример

```

VAR
  InputText   : STRING := '29738956'; (*Исходная строка*)
  SearchStr   : STRING := '123456789'; (*Искомые символы*)
  ReplaceStr   : STRING := 'abcdefghi'; (*Замещающие символы*)
  ResultText  : STRING;
END_VAR

ResultText := OSCAT_BASIC.REPLACE_CHARS( str := InputText,
                                          SRC := SearchStr,
                                          REP := ReplaceStr );

(* результат 'bigchief' *)
END_PROGRAM

```

16.66 REPLACE_UML

Тип модуля: функция	Переменная	Тип	Описание
Входы	str	STRING	Исходная строка
Выходы	REPLACE_UML	STRING	Обработанная строка
Используемые модули	TO_UML		

Функция **REPLACE_UML** заменяет все умляуты в исходной строке **str** на соответствующие сочетания латинских букв:

Умляут	Транслитерация умляута
Ä	Ae
Ö	Oe
Ü	Ue
ä	ae
ö	oe
ü	ue
ß	Ss

Если строка состоит из умляутов в верхнем регистре, но предварительно необходимо перевести ее в нижний регистр с помощью функции **LOWERCASE**, после чего обработать функцией **REPLACE_UML**.

Пример

```

VAR
  InputText   : STRING := 'äöü';      (*Исходная строка*)
  ResultText  : STRING;
END_VAR

ResultText := OSCAT_BASIC.REPLACE_UML( str := InputText);
(* результат 'aeoue'*)

END_PROGRAM

```

16.67 TICKER

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	N	INT	Длина фрагмента сообщения
	PT	TIME	Время переключения фрагментов
Выходы	Display	STRING	Фрагмент сообщения
Входы-выходы	Text	STRING	Сообщение

Функциональный блок **TICKER** используется для реализации бегущей строки. Фрагмент исходного сообщения **Text** длиной **N** передается на выход **Display**, причем с интервалом **PT** происходит смещение фрагмента на один символ вправо относительно полного текста сообщения. Блок работает корректно только при $N < \text{Длина (Text)}$. На рисунке ниже приведен пример работы с блоком.

T#5s, Text='hello', длина N=3

Время от начала работы ФБ, с	0	5	10	15	20	25	...
Значение выхода Display	hel	ell	llo	lo	o	hel	...

16.68 TO_LOWER

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	BYTE	ASCII-код символа верхнего регистра
Выходы	TO_LOWER	BYTE	ASCII-код символа нижнего регистра

Функция **TO_LOWER** конвертирует ASCII-код символа верхнего регистра в ASCII-код аналогичного символа нижнего регистра. Для работы с символами верхней половины таблицы ASCII глобальная переменная **EXTENDED_ASCII** должна иметь значение **TRUE**.

Пример

```

VAR
  UppVar      : BYTE := 16#41;      // ASCII-код 'A'
  LowVar      : BYTE;
END_VAR

LowVar := OSCAT_BASIC.TO_LOWER( IN := UppVar );
(* результат = 16#61 ASCII-код 'a'*)

END_PROGRAM

```

16.69 TO_UML

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	BYTE	ASCII-код умляута
Выходы	TO_UML	STRING(2)	Транслитерация умляута
Используемые модули	CHR_TO_STRING		

Функция **TO_UML** конвертирует ASCII-код умляута в строку, содержащую его транслитерацию.

Умляут	ASCII-код	Транслитерация умляута
Ä	16#C4	Ae
Ö	16#D6	Oe
Ü	16#DC	Ue
ä	16#E4	ae
ö	16#F6	oe
ü	16#FC	ue
ß	16#DF	Ss

Пример

```

VAR
  UmlVar   : BYTE := 16#C4;    // Ä
  ResVar   : STRING(2);
END_VAR

ResVar := OSCAT_BASIC.TO_UML( IN := UmlVar );
(* результат 'Ae' *)

END_PROGRAM

```

16.70 TO_UPPER

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	BYTE	ASCII-код символа нижнего регистра
Выходы	TO_UPPER	BYTE	ASCII-код символа верхнего регистра

Функция **TO_UPPER** конвертирует ASCII-код символа нижнего регистра в ASCII-код аналогичного символа верхнего регистра. Для работы с символами верхней половины таблицы ASCII глобальная переменная **EXTENDED_ASCII** должна иметь значение **TRUE**.

Пример

```

VAR
  LowVar   : BYTE := 16#61;    // ASCII-код 'a'
  UppVar   : BYTE;
END_VAR

UppVar := OSCAT_BASIC.TO_UPPER( IN := LowVar );
(* результат = 16#41 ASCII-код 'A' *)

END_PROGRAM

```

16.71 TRIM

Тип модуля: функция	Переменная	Тип	Описание
Входы	str	STRING	Исходная строка
Выходы	TRIM	STRING	Строка без пробелов

Функция **TRIM** удаляет из строки **str** пробелы и возвращает обработанную строку.

Пример

```
VAR
  TestVar   : STRING := 'hello world !';    // исходная строка
  ResVar    : STRING;
END_VAR

ResVar := OSCAT_BASIC.TRIM( str := TestVar );
(* результат 'helloworld!')

END_PROGRAM
```

16.72 TRIM1

Тип модуля: функция	Переменная	Тип	Описание
Входы	str	STRING	Исходная строка
Выходы	TRIM1	STRING	Строка без последовательных пробелов

Функция **TRIM1** заменяет в строке **str** последовательные пробелы на один пробел, а также удаляет все пробелы в начале и конце строки.

Пример

```
VAR
  TestVar   : STRING := ' hello world! ';    // исходная строка
  ResVar    : STRING;
END_VAR

ResVar := OSCAT_BASIC.TRIM1( str := TestVar );
(* результат 'hello world!')

END_PROGRAM
```

16.73 TRIME

Тип модуля: функция	Переменная	Тип	Описание
Входы	str	STRING	Исходная строка
Выходы	TRIME	STRING	Строка без пробелов в начале и конце

Функция **TRIME** удаляет пробелы в начале и конце строки **str**, после чего возвращает обработанную строку.

Пример

```
VAR
  TestVar   : STRING := ' hello world! ';    // исходная строка
  ResVar    : STRING;
END_VAR

ResVar := OSCAT_BASIC.TRIME( str := TestVar );
(* результат 'hello world!')

END_PROGRAM
```

16.74 UPPERCASE

Тип модуля: функция	Переменная	Тип	Описание
Входы	str	STRING	Исходная строка
Выходы	UPPERCASE	STRING	Строка в верхнем регистре
Используемые модули	TO_UPPER		

Функция **UPPERCASE** конвертирует исходную строку **str** в соответствующую строку верхнего регистра. Для работы с символами верхней половины таблицы ASCII глобальная переменная **EXTENDED_ASCII** должна иметь значение **TRUE**.

Пример

```
VAR
  TestVar   : STRING := ' tEsT ';    // исходная строка
  ResVar    : STRING;
END_VAR

ResVar := OSCAT_BASIC.UPPERCASE( str := TestVar );
(* результат 'TEST'*)

END_PROGRAM
```

16.75 WEEKDAY_TO_STRING

Тип модуля: функция	Переменная	Тип	Описание
Входы	WDAY	INT	Номер дня недели
	LANG	INT	Язык приложения
	LX	INT	Тип названия (полное/сокращенное)
Выходы	WEEKDAY_TO_STRING	STRING(10)	Название дня недели

Функция **WEEKDAY_TO_STRING** возвращает название дня недели с номером **WDAY**. Вход **LANG** определяет язык приложения (см. глобальную переменную **LANGUAGE**). Вход **LX** определяет тип названия месяца: **0** – полное (Monday), **2** – сокращенное (Mo).

Пример

```
VAR
  DayNumber : INT := 5;           // 1 = Понедельник, 7 = Воскресенье
  Language  : INT := 1;           // 1 = Английский
  NameType  : INT := 0;           // 0 = Полное название
  DayName   : STRING(10);
END_VAR

DayName := OSCAT_BASIC.WEEKDAY_TO_STRING( WDAY := DayNumber,
                                           LANG := Language,
                                           LX   := NameType );

// Результат: 'Friday'

END_PROGRAM
```

17 Модули памяти

17.1 FIFO_16

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	Din	DWORD	Данные, записываемые в буфер
	E	BOOL	Сигнал управления блоком (вкл./откл.)
	RD	BOOL	Сигнал чтения данных из буфера
	WD	BOOL	Сигнал записи данных в буфер
	RST	BOOL	Сигнал очистки буфера.
Выходы	Dout	DWORD	Данные, считанные из буфера.
	EMPTY	BOOL	Флаг «буфер пуст»
	FULL	BOOL	Флаг «буфер полон»
Используемые модули	INC1		

Функциональный блок **FIFO_16** представляет собой буфер размером 16 значений типа **DWORD**, организованный по принципу **First In – First Out** (FIFO). Если вход **E=TRUE**, то блок находится в работе. Если **WD=TRUE**, то каждый цикл ПЛК значение входа **Din** записывается в буфер. Если **RD=TRUE**, то каждый цикл ПЛК значения из буфера подаются на выход **Dout** (в том порядке, в котором они были записаны в буфер). Чтобы записать/считать один элемент, необходимо подать на вход **RD/WD** единичный импульс. Выход **EMPTY** принимает значение **TRUE**, если буфер пуст (например, при попытке прочитать четвертое значение из буфера, в который было записано только 3 значения, этот флаг станет активным). Выход **FULL** принимает значение **TRUE**, если буфер заполнен. По переднему фронту на входе **RST** происходит очистка буфера.

17.2 FIFO_32

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	Din	DWORD	Данные, записываемые в буфер
	E	BOOL	Сигнал управления блоком (вкл./откл.)
	RD	BOOL	Сигнал чтения данных из буфера
	WD	BOOL	Сигнал записи данных в буфер
	RST	BOOL	Сигнал очистки буфера.
Выходы	Dout	DWORD	Данные, считанные из буфера.
	EMPTY	BOOL	Флаг «буфер пуст»
	FULL	BOOL	Флаг «буфер полон»
Используемые модули	INC1		

Функциональный блок **FIFO_32** представляет собой буфер размером 32 значений типа **DWORD**. Принцип работы полностью соответствует ФБ **FIFO_16**. Можно создать буфер любого размера, изменив константу **n** в разделе объявления переменных блока.

17.3 STACK_16

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	Din	DWORD	Данные, записываемые в буфер

Тип модуля: ФБ	Переменная	Тип	Описание
	E	BOOL	Сигнал управления блоком (вкл./откл.)
	RD	BOOL	Сигнал чтения данных из буфера
	WD	BOOL	Сигнал записи данных в буфер
	RST	BOOL	Сигнал очистки буфера.
Выходы	Dout	DWORD	Данные, считанные из буфера.
	EMPTY	BOOL	Флаг «буфер пуст»
	FULL	BOOL	Флаг «буфер полон»

Функциональный блок **STACK_16** представляет собой буфер размером 16 значений типа **DWORD**, организованный по принципу **Last In – First Out** (LIFO). Если вход **E=TRUE**, то блок находится в работе. Если **WD=TRUE**, то каждый цикл ПЛК значение входа **Din** записывается в буфер. Если **RD=TRUE**, то каждый цикл ПЛК значения из буфера подаются на выход **Dout** (в том порядке, обратном порядку их записи в буфер). Чтобы записать/считать один элемент, необходимо подать на вход **RD/WD** единичный импульс. Выход **EMPTY** принимает значение **TRUE**, если буфер пуст (например, при попытке прочитать четвертое значение из буфера, в который было записано только 3 значения, этот флаг станет активным). Выход **FULL** принимает значение **TRUE**, если буфер заполнен. По переднему фронту на входе **RST** происходит очистка буфера.

17.4 STACK_32

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	Din	DWORD	Данные, записываемые в буфер
	E	BOOL	Сигнал управления блоком (вкл./откл.)
	RD	BOOL	Сигнал чтения данных из буфера
	WD	BOOL	Сигнал записи данных в буфер
	RST	BOOL	Сигнал очистки буфера.
Выходы	Dout	DWORD	Данные, считанные из буфера.
	EMPTY	BOOL	Флаг «буфер пуст»
	FULL	BOOL	Флаг «буфер полон»

Функциональный блок **STACK_32** представляет собой буфер размером 32 значения типа **DWORD**. Принцип работы полностью соответствует ФБ [STACK_16](#). Можно создать буфер любого размера, изменив константу **n** в разделе объявления переменных блока.

18 Генераторы импульсов

18.1 A_TRIG

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	REAL	Контролируемое значение
	RES	REAL	Допустимое изменение
Выходы	Q	BOOL	Импульс превышения допустимого значения
	D	REAL	Разность между текущим и предыдущим недопустимым значением

Функциональный блок **A_TRIG** контролирует входное значение **IN** и при изменении его относительно последнего сохраненного значения на величину, превышающую **RES**, генерирует единичный импульс на выходе **Q** и сохраняет текущее значение **IN**. Разность между последним сохраненным значением **IN** и текущим значением **IN** подается на выход **D**. В момент импульса на выходе **Q** значение **D** обнуляется.

Пример работы блока:

Номер цикла	IN	RES	Q	D
1	0	5	FALSE	0
2	2		FALSE	2 (2-0)
3	10		TRUE (импульс на цикл) [(10-2)>5]	0
4	7		FALSE	-3 (7-10)
5	4		TRUE (импульс на цикл) [(10-4)>5]	0
6	0		FALSE	-4 (0-4)

18.2 B_TRIG

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	CLK	BOOL	Контролируемый сигнал
Выходы	Q	BOOL	Выход триггера

Функциональный блок **B_TRIG** генерирует единичный импульс на выходе **Q** при изменении входного сигнала **CLK**. В отличие от блоков **R_TRIG** и **F_TRIG** из библиотеки Standard, которые детектируют только импульсы переднего/заднего фронта соответственно, данный блок детектирует импульсы обоих фронтов.

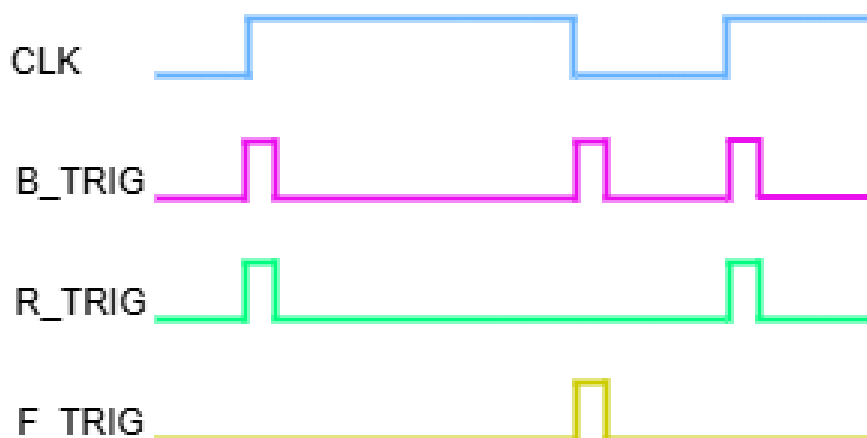


Рисунок 18.1 – Трассировка работы ФБ B_TRIG

18.3 CLICK_CNT

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	BOOL	Контролируемый сигнал
	N	INT	Ожидаемое число импульсов
	TC	TIME	Период времени
Выходы	Q	BOOL	Выход счетчика

Функциональный блок **CLICK_CNT** генерирует единичный импульс на выходе **Q**, если логический сигнал **IN** в течение заданного времени **TC** получает ровно **N** импульсов по заднему фронту. Генерация единичного импульса на выходе осуществляется по истечении времени **TC**. Работа блока (отсчет времени) начинается по импульсу переднего фронта на входе **IN**. Если **N=0**, то единичный импульс на выходе генерируется в том случае, если входной логический сигнал **IN** в течение времени **TC** не получает импульсов по заднему фронту.

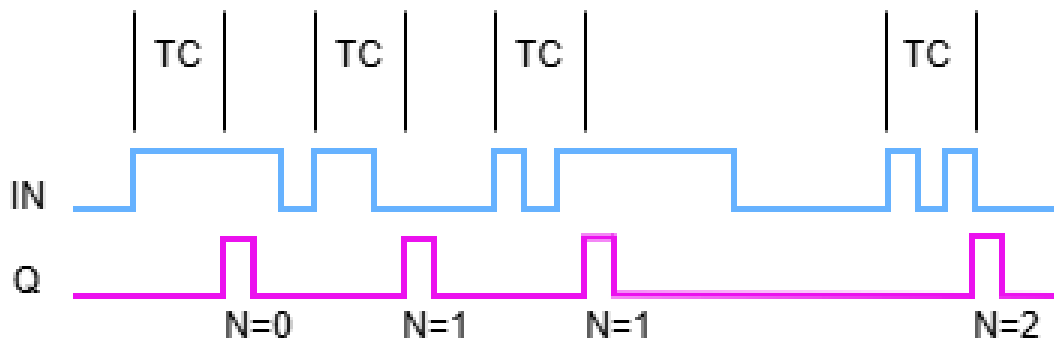


Рисунок 18.2 – Трассировка работы ФБ CLICK_CNT

18.4 CLICK_DEC

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	BOOL	Контролируемый сигнал
	TC	TIME	Период
Выходы	Q0	BOOL	Флаг «импульсов не было»
	Q1	BOOL	Флаг «был один импульс»
	Q2	BOOL	Флаг «было два импульса»
	Q3	BOOL	Флаг «было три импульса»

Функциональный блок **CLICK_DEC** генерирует единичный импульс на одном из выходов блока, если логический сигнал **IN** в течение заданного времени **TC** получает соответствующее количество импульсов по заднему фронту (см. таблицу ниже). Генерация единичного импульса на выходе осуществляется по истечении времени **TC**. Работа блока (отсчет времени) начинается по импульсу переднего фронта на входе **IN**.

Число импульсов по заднему фронту за время TC	Единичный импульс на выходе:
0	Q0
1	Q1
2	Q2
3	Q3
>3	-

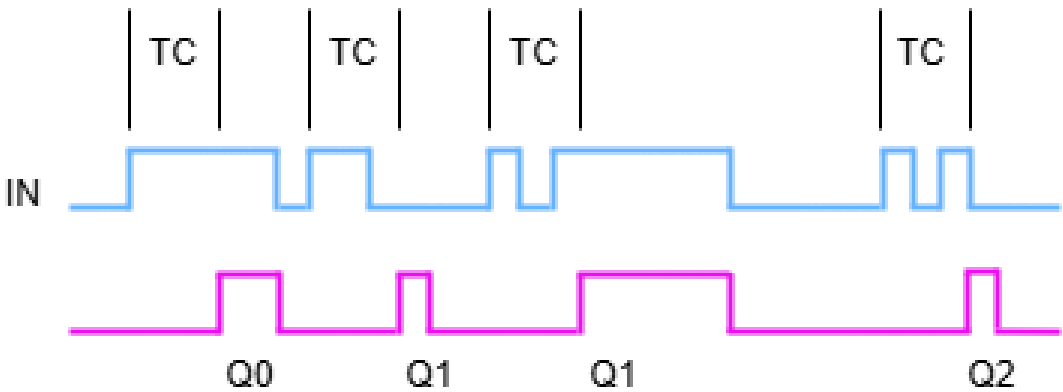


Рисунок 18.3 – Трассировка работы ФБ CLICK_DEC

18.5 CLK_DIV

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	clk	BOOL	Исходный сигнал
	rst	BOOL	Сброс выходов
Выходы	Q0...Q7	BOOL	Модулированный сигнал с частотой 1/2, 1/4...1/256 от исходного

Функциональный блок **CLK_DIV** модулирует единичные импульсы, поступающие на вход **clk**, уменьшая их частоту в 1/2, 1/4...1/256 раз и подает обработанные сигналы на выходы **Q0...Q7**. Вход **rst** используется для обнуления выходов блока. Обратите внимание, что на вход **clk** должны подаваться только единичные импульсы; если исходный сигнал не является таковым, то можно обработать его с помощью ФБ [TP_X](#).

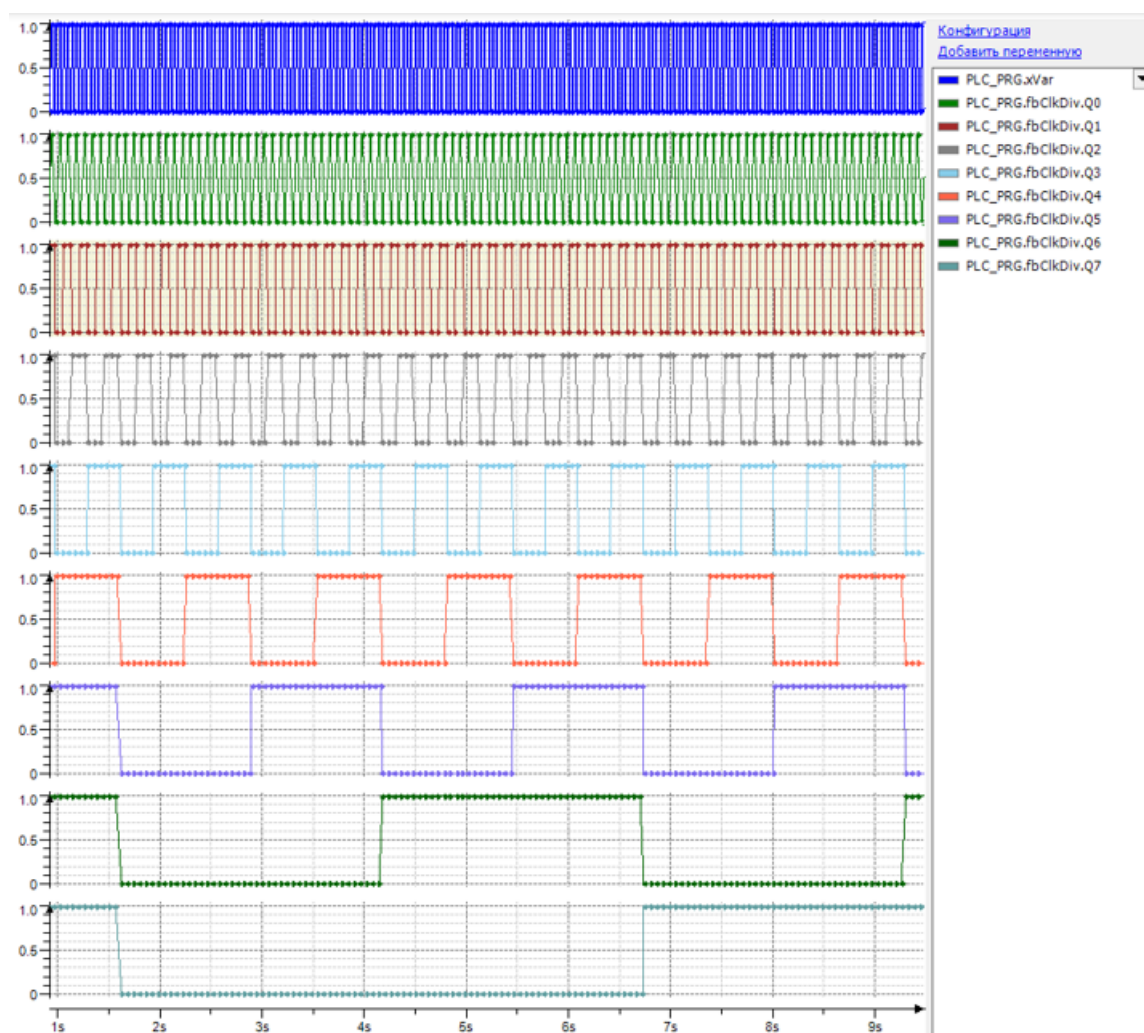


Рисунок 18.4 – Трассировка работы ФБ CLK_DIV

18.6 CLK_N

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	N	INT	Период между импульсами
Выходы	Q	BOOL	Выходной сигнал
Используемые модули	T_PLC_MS		

Функциональный блок **CLK_N** генерирует единичные импульсы с периодом 2^N мс. На рисунке ниже приведена трассировка работы блока в случае:

- N=3 (период между импульсами – 8 мс);
- N=4 (период между импульсами – 16 мс);
- N=5 (период между импульсами – 32 мс);
- N=6 (период между импульсами – 64 мс).

Время цикла ПЛК – 1 мс. По очевидным причинам период между импульсами не может быть меньше реального цикла ПЛК.

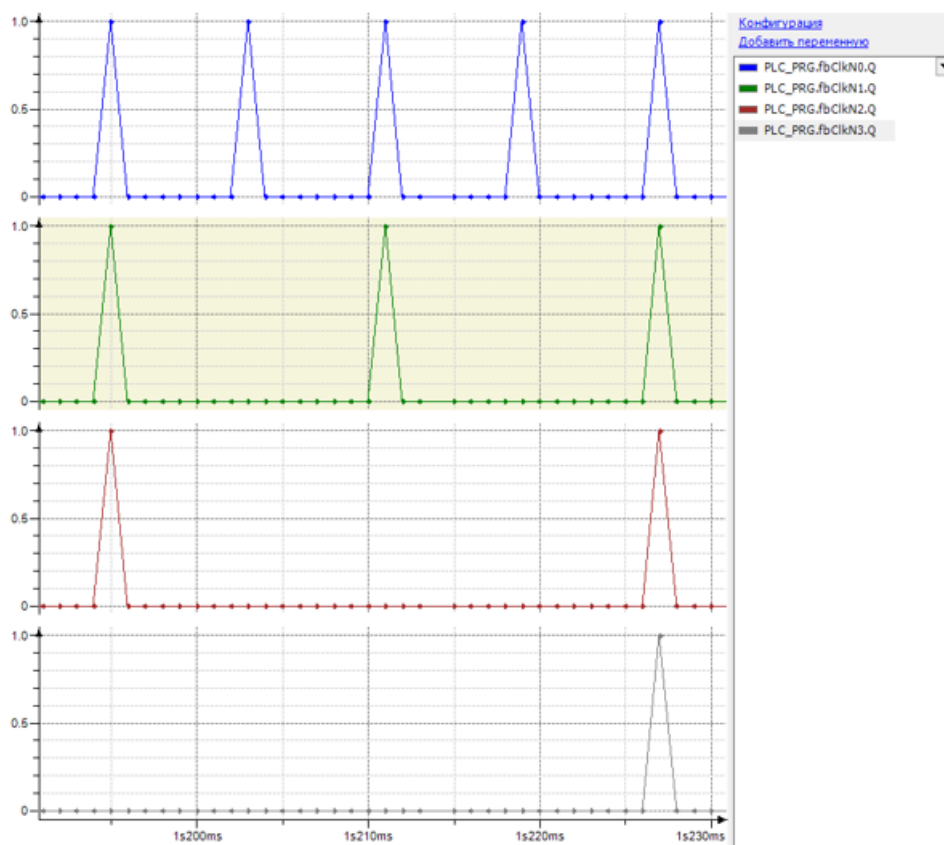


Рисунок 18.5 – Трассировка работы ФБ CLK_N

18.7 CLK_PRG

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	PT	TIME	Период между импульсами
Выходы	Q	BOOL	Выходной сигнал
Используемые модули	T_PLC_MS		

Функциональный блок **CLK_PRG** генерирует единичные импульсы с периодом **PT**. По очевидным причинам период между импульсами не может быть меньше реального цикла ПЛК.

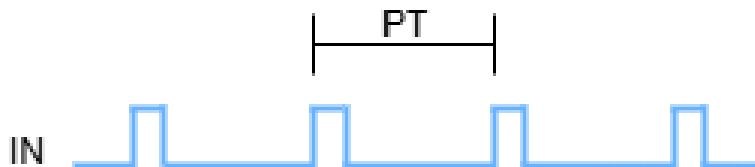


Рисунок 18.6 – Трассировка работы ФБ CLK_PRG

18.8 CLK_PULSE

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	PT	TIME	Период между импульсами
	N	INT	Кол-во импульсов за рабочий цикл
	rst	BOOL	Сигнал запуска блока

Тип модуля: ФБ	Переменная	Тип	Описание
Выходы	Q	BOOL	Выходной сигнал
	cnt	INT	Кол-во сгенерированных импульсов
	run	BOOL	Флаг «блок в работе»
Используемые модули	T_PLC_MS		

Функциональный блок **CLK_PULSE** генерирует **N** единичных импульсов с заданным периодом **PT**. Импульс по переднему фронту на входе **rst** запускает блок в работу. На выходе **cnt** отображается текущее число сгенерированных импульсов. Выход **run** имеет значение **TRUE**, пока работа блока не завершена. После завершения работы блока выход **run** принимает значение **FALSE**. По очевидным причинам период между импульсами не может быть меньше реального цикла ПЛК. Ниже приведена трассировка ФБ для **PT=T#1s** и **N=5**:

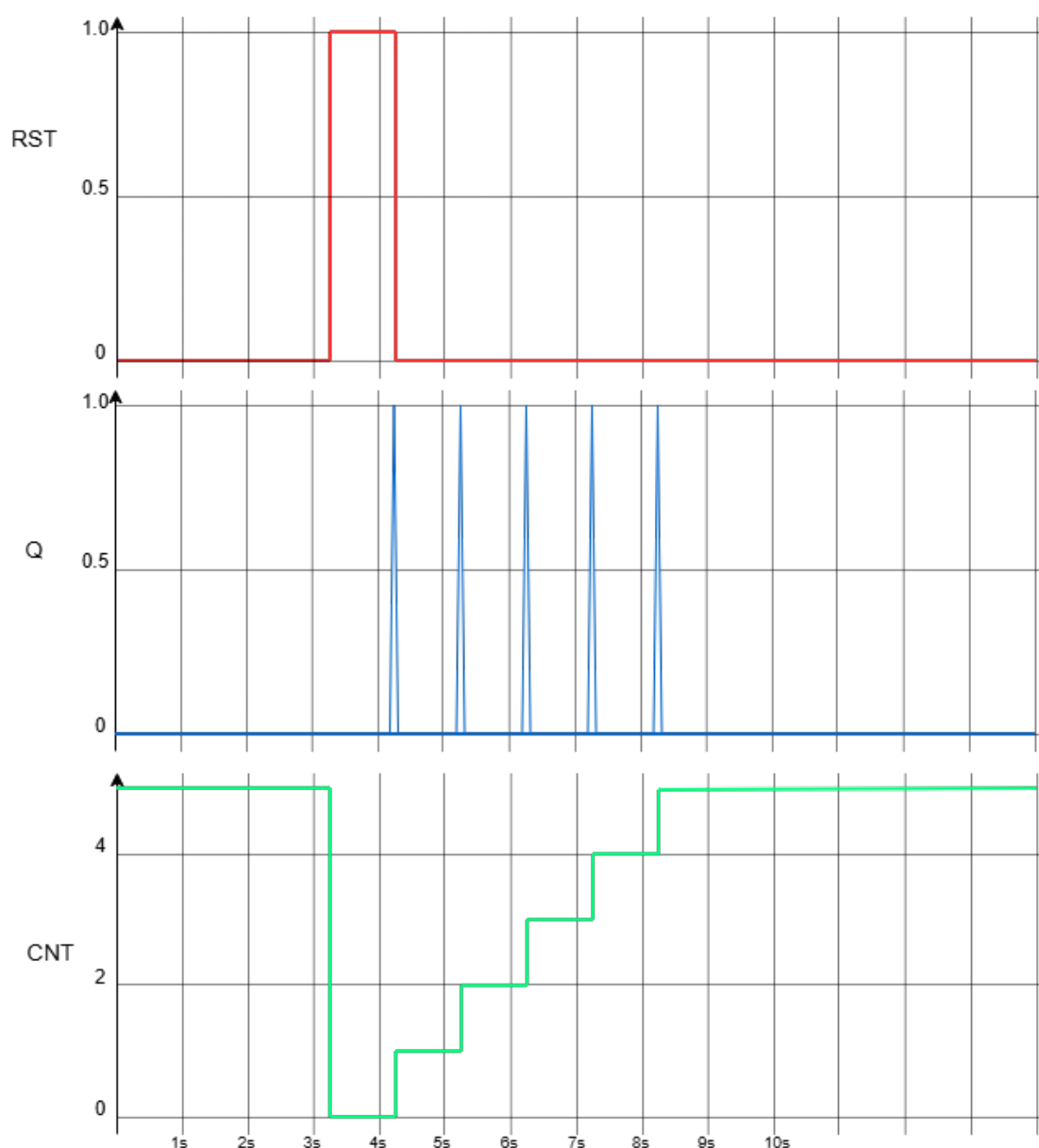


Рисунок 18.7 – Трассировка работы ФБ CLK_PULSE

18.9 CYCLE_4

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	E	BOOL	Вход управления блоком
	T0	TIME	Время в состоянии 0

Тип модуля: ФБ	Переменная	Тип	Описание
	T1	TIME	Время в состоянии 1
	T2	TIME	Время в состоянии 2
	T3	TIME	Время в состоянии 3
	S0	BOOL	Режим работы блока
	SX	INT	Номер состояния сброса
	SL	BOOL	Сброс в состояние SX
Выходы	STATE	INT	Номер текущего состояния
Используемые модули	T_PLC_MS		

Функциональный блок **CYCLE_4** является переключателем для четырех состояний (0-3). Если вход **E= TRUE**, то блок находится в работе, и на выход **STATE** подаются номера состояний (0-3), причем время, проводимое в каждом из состояний, задается переменными **T0... T3**. Вход **S0** определяет режим работы блока:

- Если **S0=TRUE**, то переключение состояний происходит циклически, пока **E=TRUE**.
- Если **S0=FALSE**, то происходит однократное переключение всех состояний, и по достижению состояния 3 блок прекращает работу.

По переднему фронту на входе **SL** происходит переход в состояние с номером **SX**, после чего автоматически начинается очередная смена состояний.

Если **E=FALSE**, то блок прекращает работу и переходит в состояние 0.

На рисунке ниже приведена трассировка работы блока для **S0=TRUE**, **T0=1мс**, **T1=2мс**, **T2=4мс**, **T3=2мс**:

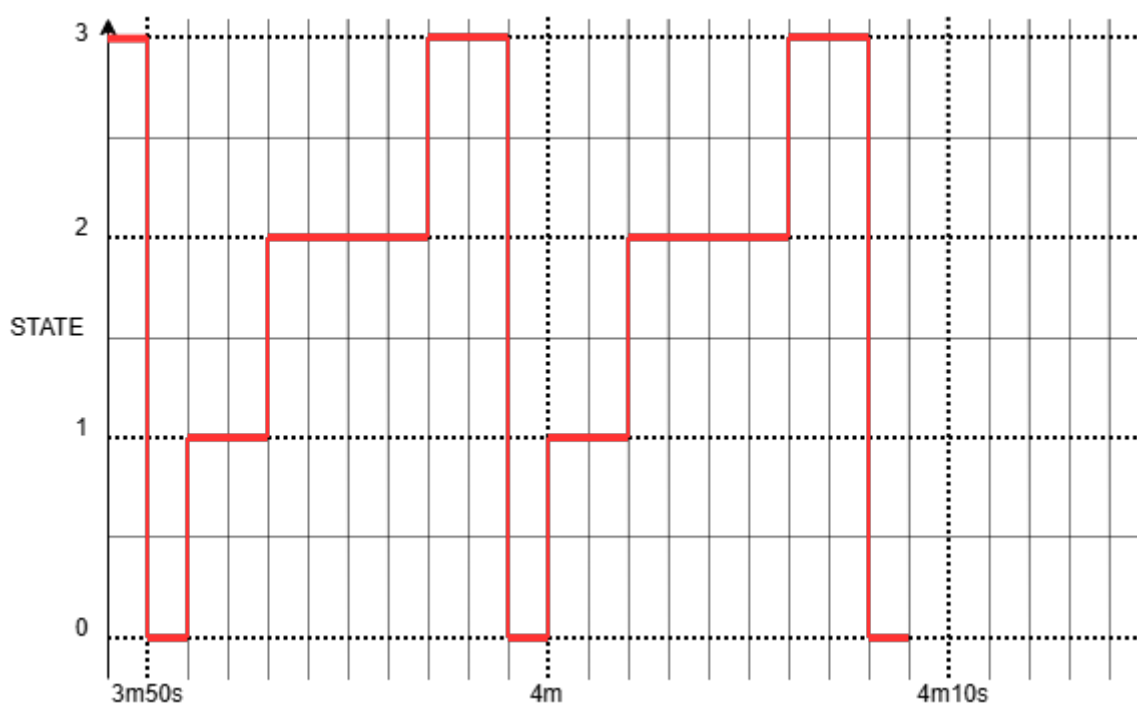


Рисунок 18.8 – Трассировка работы ФБ CYCLE_4

18.10 D_TRIG

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	PT	DWORD	Контролируемое значение
Выходы	Q	BOOL	Флаг «значение изменилось»
	X	DWORD	Разность между текущим и предыдущим значением

Функциональный блок **D_TRIG** генерирует единичный импульс при изменении входного значения **in**. На выход **X** подается разность между значениями **in** текущего и предыдущего цикла. На рисунке ниже приведен пример работы с блоком. Начальное значение **in** – 0, далее оно изменяется до 12, после чего – до 6.

Поскольку выход **X** имеет тип **DWORD**, а значение **in** может уменьшиться по сравнению с предыдущим циклом (т.е. разность будет отрицательной), то необходимо преобразовать его с помощью стандартного оператора **DWORD_TO_DINT**.

18.11 GEN_BIT

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	in0...in3	DWORD	Генерируемые последовательности бит
	clk	BOOL	Вход управления блоком
	steps	INT	Кол-во используемых бит последовательностей
	rep	INT	Число повторений последовательностей
	rst	BOOL	Вход сброса блока
Выходы	Q0...Q3	BOOL	Выходы генератора
	cnt	INT	Номер текущего бита последовательностей
	run	BOOL	Флаг «блок в работе»

Функциональный блок **GEN_BIT** используется для генерации битовых последовательностей на выходах **Q0...Q3** по заданным паттернам **in0... in3**. Если вход **clk=TRUE**, то блок находится в работе, и на выходы **Q0... Q3** каждый цикл ПЛК последовательно подаются значения бит входов **in0... in3** (от младшего бита **DWORD** к старшему). Вход **steps** определяет число используемых бит из паттернов **in0... in3**. Вход **rep** определяет число повторов битовых последовательностей на выходе; если **rep=0**, то последовательность повторяется циклически. На выход **cnt** поступает номер битов паттернов, в данный момент поданных на выход. Выход **run** имеет значение **TRUE**, пока работа блока не завершена. Импульс по переднему фронту на входе **rst** обнуляет все выходы блока.

На рисунке ниже приведен пример работы с блоком и его трассировка для ПЛК с циклом= 1 с.

В $t=11$ с на вход **xEnable** было подано значение **TRUE**, после чего блок два раза подряд (**rep=2**) сгенерировал последовательность размером 8 бит (**steps=8**), созданную на основе паттерна **in0** (2#0000_1111).

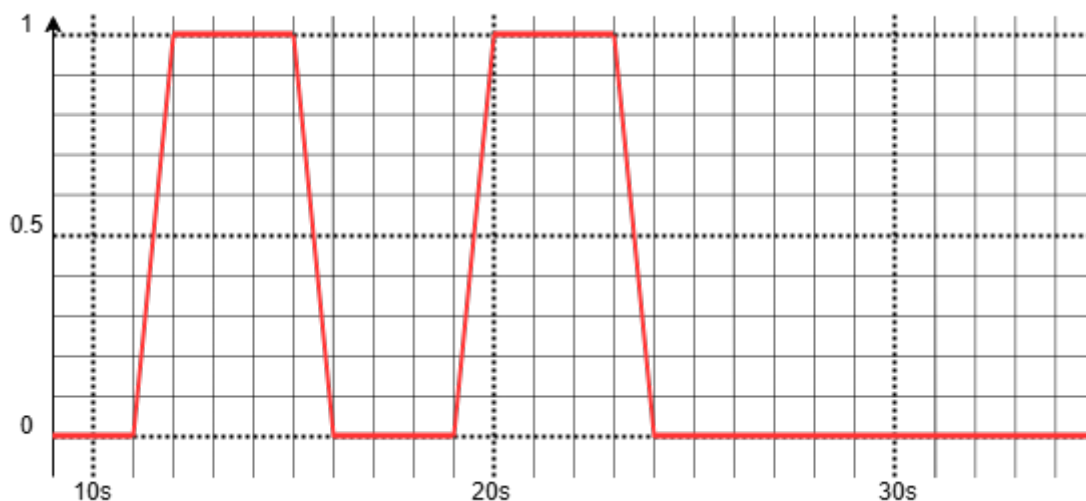


Рисунок 18.9 – Трассировка работы ФБ GEN_BIT

18.12 GEN_SQ

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	PT	TIME	Период между импульсами
Выходы	Q	BOOL	Выход генератора
Используемые модули	T_PLC_MS		

Функциональный блок **GEN_SQ** с заданным периодом **PT** генерирует импульсы длиной **0.5 PT** (соответственно, пауза между импульсами также составляет 0.5 PT).

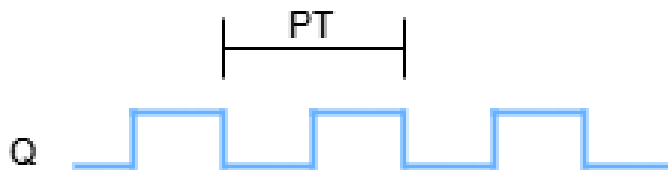


Рисунок 18.10 – Трассировка работы ФБ GEN_SQ

18.13 SCHEDULER

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	E0...E3	BOOL	Вход управления каналом генератора
Выходы	Q0...Q3	BOOL	Выход канала генератора
Параметры	T0...T3	TIME	Время между импульсами
Используемые модули	T_PLC_MS		

Функциональный блок **SCHEDULER** представлял собой четырехканальный генератор единичных импульсов. Если один из входов **E** принимает значение **TRUE**, то на соответствующем выходе **Q** начинают генерироваться единичные импульсы с периодом **T**. В каждом цикле ПЛК импульс генерируется только в одном из каналов.

На рисунке ниже приведена трассировка работы блока для T0=1мс, T1=2мс, T2=3мс, T3=4мс.

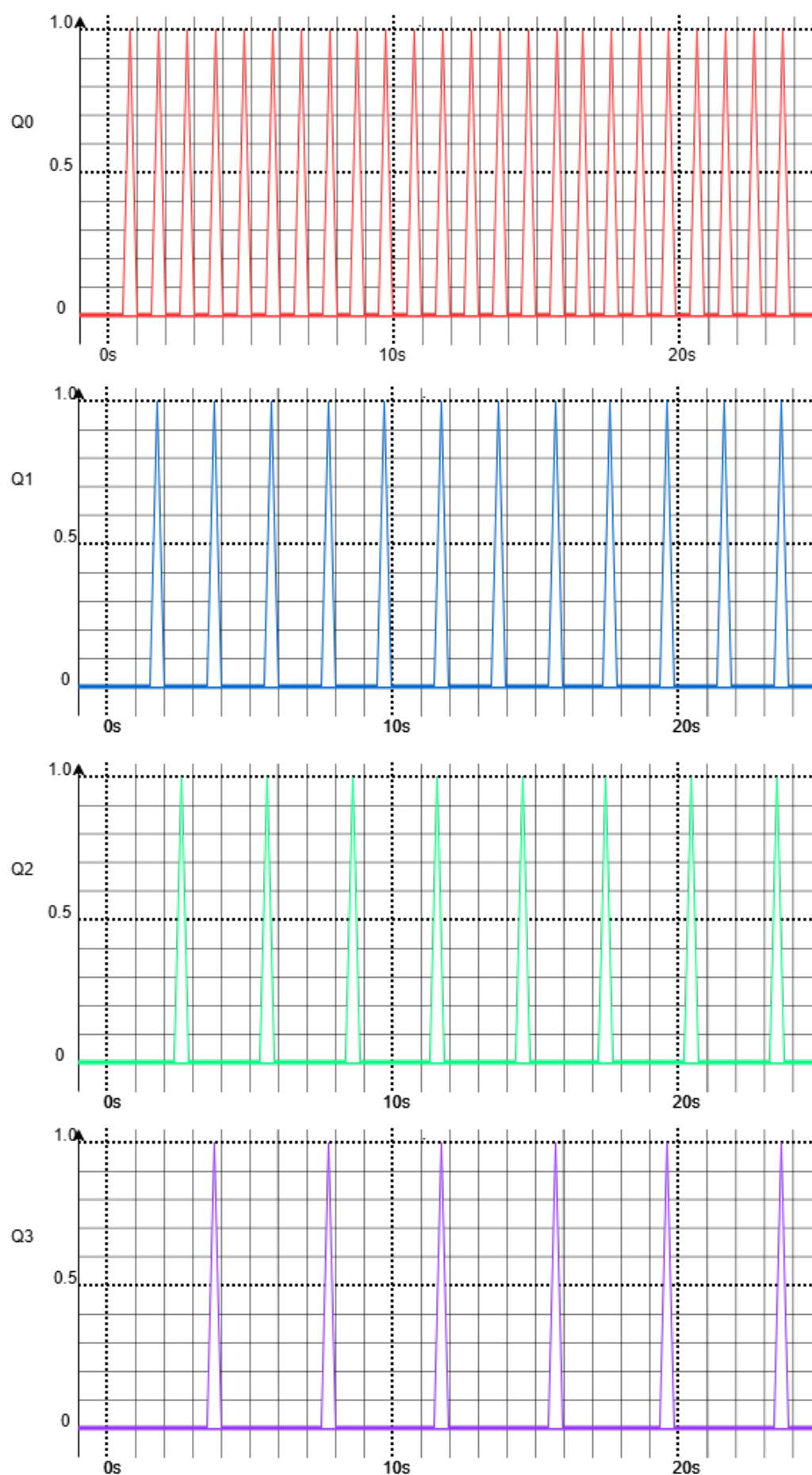


Рисунок 18.11 – Трассировка работы ФБ SCHEDULER

18.14 SCHEDULER_2

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	E0...E3	BOOL	Вход управления каналом генератора
Выходы	Q0...Q3	BOOL	Выход канала генератора
Параметры	C0...C3	UINT	Кол-во циклов между импульсами
	O0...O3	UINT	Кол-во циклов начальной задержки
Используемые модули	T_PLC_MS		

Функциональный блок **SCHEDULER_2** представляет собой четырехканальный генератор единичных импульсов. Если один из входов **E** принимает значение **TRUE**, то на соответствующем выходе **Q** начинают генерироваться единичные импульсы, пауза между которыми составляет **C** циклов ПЛК. Параметр **O** позволяет настроить задержку (в циклах ПЛК) перед первым импульсом.

На рисунке ниже приведена трассировка работы блока для ПЛК с циклом 100 мс со следующими значениями параметров: C0=10, C1=20, C2=30, C3=40, O0=O1=5, O2=O3=20. Все каналы генератора были запущены в работу одновременно.

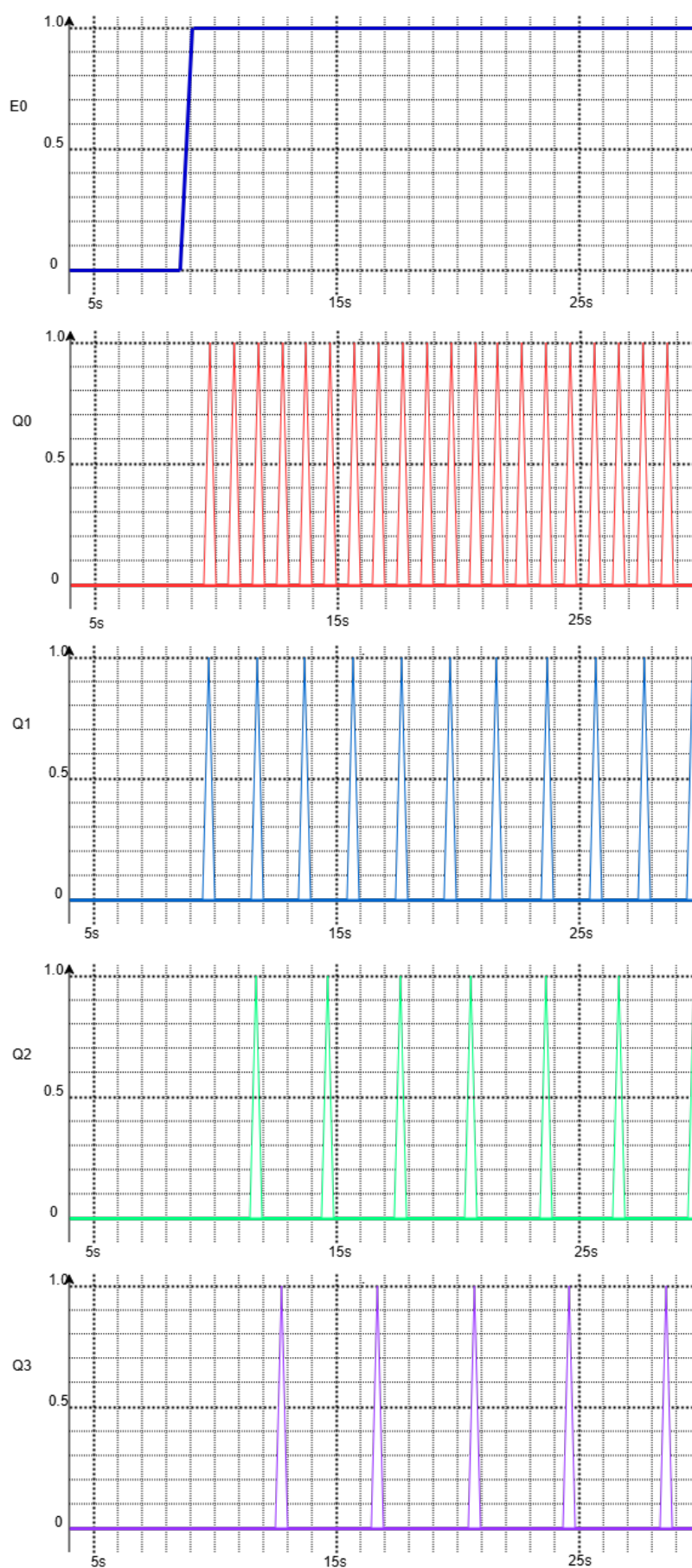


Рисунок 18.12 – Трассировка работы ФБ SCHEDULER_2

18.15 SEQUENCE_4

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	in0...in3	BOOL	Контролируемый сигнал
	start	BOOL	Вход управления блоком
	rst	BOOL	Вход сброса блока
	wait0...wait3	TIME	Время ожидания сигнала
	delay0...delay3	TIME	Задержка
Выходы	Q0...Q3	BOOL	Выход секвенсора
	QX	BOOL	Флаг «один из выходов активен»
	run	BOOL	Флаг «блок в работе».
	step	INT	Номер обрабатываемого канала
	status	BYTE	Код ошибки/сообщения
Параметры	stop_on_error	BOOL	Режим обработки ошибок
Используемые модули	T_PLC_MS		

Функциональный блок **SEQUENCE_4** представляет собой четырехбитный секвенсор. Блок запускается в работу импульсом по переднему фронту на входе **start** и в течение времени **wait0** ожидает сигнала **TRUE** на входе **in0**. Когда **in0** принимает значение **TRUE**, выход **Q0** устанавливается в **TRUE** и блок делает задержку на время **delay0**, после чего переходит к ожиданию сигнала **TRUE** на входе **in1** в течение времени **wait1** и т.д. После задержки **delay3** блок прекращает работу. Вход **rst** используется для сброса выходных переменных блока (по переднему фронту). Выход **QX** принимает значение **TRUE**, если хотя бы один из выходов **Q0...Q3** имеет значение **TRUE**. Выход **run** принимает значение **TRUE**, если блок находится в работе. Выход **step** сигнализирует о номере текущего обрабатываемого входа (**-1** – блок не в работе, **0** – **in0**, ..., **3** – **in3**).

Выход **status** возвращает код ошибки:

Код ошибки	Описание
1	Вход in0 не стал активным в течение времени wait0
2	Вход in1 не стал активным в течение времени wait1
3	Вход in2 не стал активным в течение времени wait2
4	Вход in3 не стал активным в течение времени wait3
110	Блок не запущен
111	Блок в работе

Параметр **stop_on_error** определяет поведение блока при возникновении ошибок с кодом 1-4: **FALSE** – блок продолжает работу, **TRUE** – блок прекращает работу.

На рисунке ниже приведена трассировка работы блока в случае, когда **delay0=delay2=10 мс**, **delay1=delay3=20 мс**. На входы **in0... in3** не заведены переменные – и по умолчанию они имеют значения **TRUE**. Из-за этого времена **wait0...wait3** не обрабатываются блоком (т.к. сигнал изначально =**TRUE**, то время ожидания =**0**). Иными словами, в данном случае выходы **Q0...Q3** последовательно принимают значение **TRUE** на время **delay0...delay3**.

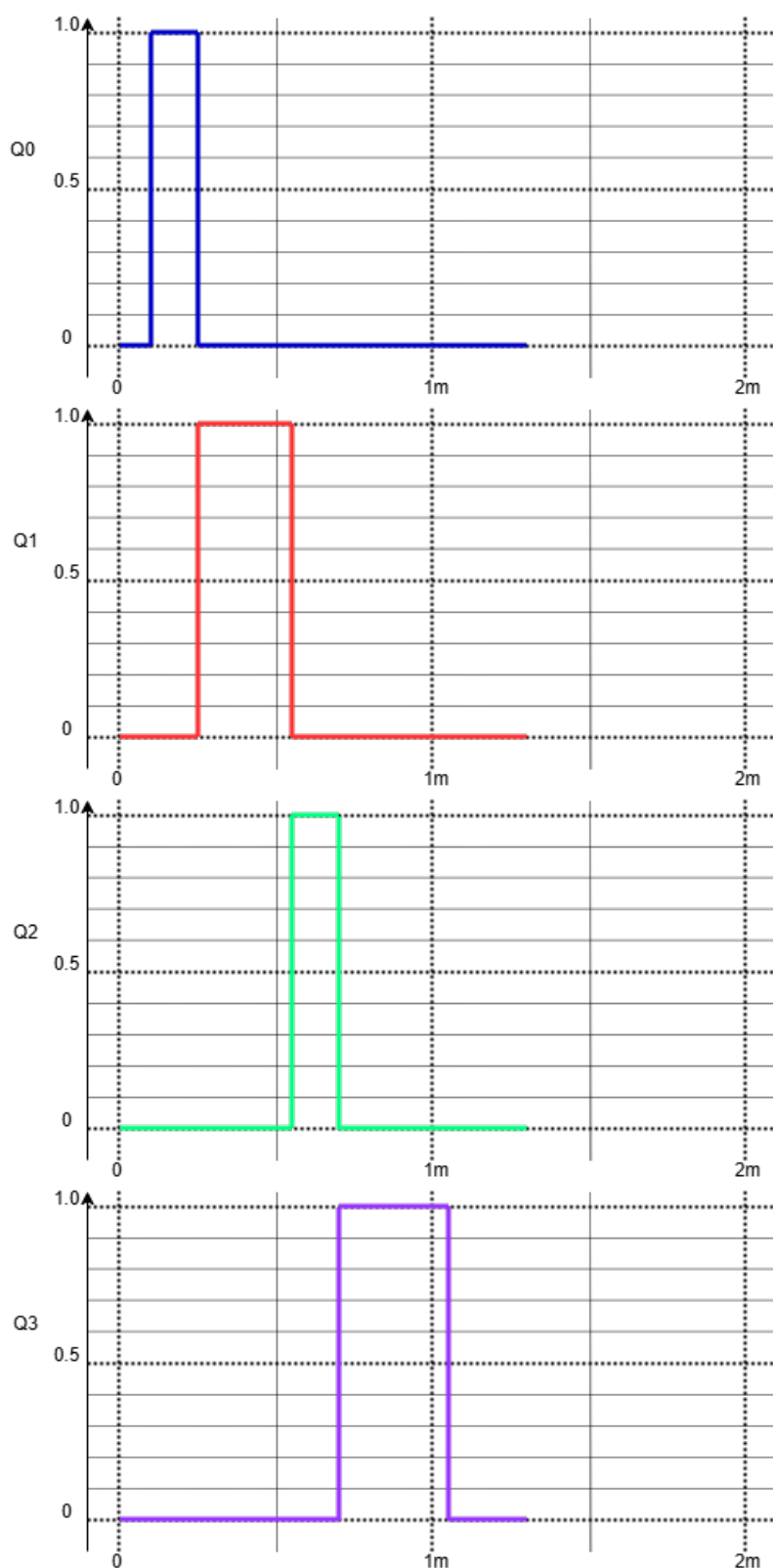


Рисунок 18.13 – Трассировка работы ФБ SEQUENCE_4

18.16 SEQUENCE_8

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	in0...in7	BOOL	Контролируемый сигнал
	Start	BOOL	Вход управления блоком
	rst	BOOL	Вход сброса блока

Тип модуля: ФБ	Переменная	Тип	Описание
Выходы	wait0...wait7	TIME	Время ожидания сигнала
	delay0...delay7	TIME	Задержка
	Q0...Q7	BOOL	Выход секвенсора
	QX	BOOL	Флаг «один из выходов активен»
	run	BOOL	Флаг «блок в работе»
Параметры	step	INT	Номер обрабатываемого канала
	status	BYTE	Код ошибки/сообщения
Используемые модули	stop_on_error	BOOL	Режим обработки ошибок
Используемые модули T_PLC_MS			

Функциональный блок **SEQUENCE_8** представляет собой восьмибитный секвенсор. Принцип работы полностью соответствует **SEQUENCE_4**.

18.17 SEQUENCE_64

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	START	BOOL	Вход управления блоком
	SMAX	INT	Кол-во состояний последовательности
	PROG	ARRAY [0..63] OF TIME	Периоды состояний последовательности
	RST	BOOL	Вход сброса блока
Выходы	STATE	INT	Выход секвенсора
	TRIG	BOOL	Сигнал смены состояний
Используемые модули T_PLC_MS, INC2			

Функциональный блок **SEQUENCE_64** представляет собой 64-х битный секвенсор. Пока блок не запущен, выход **STATE** = -1. Блок запускается в работу импульсом по переднему фронту на входе **START**, после чего выход **STATE** имеет значение **0** в течение времени **PROG [0]**, затем **1** в течение времени **PROG [1]** и т.д. вплоть до **STATE=(SMAX-1)** в течение времени **PROG[SMAX-1]**, после чего **STATE** снова принимает значение **-1** и блок прекращает работу. Вход **RST** используется для принудительного прекращения работы блока. На выходе **TRIG** генерируется единичный импульс при каждом изменении выхода **STATE**.

18.18 TMAX

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	BOOL	Вход таймера
	PT	TIME	Максимальное время работы
Выходы	Q	BOOL	Выход таймера
	Z	BOOL	Флаг «завершение работы»
Используемые модули T_PLC_MS			

Функциональный блок **TMAX** представляет собой таймер с ограничением на максимальное время включения. Пока вход **IN** имеет значение **TRUE**, таймер находится в работе и выход **Q** имеет значение **TRUE**. По истечении времени **PT** выход **Q** принимает значение **FALSE**, а на выходе **Z** генерируется единичный импульс. Если вход **IN** принимает значение **FALSE** до истечения времени **PT**, то таймер прекращает работу и обнуляет выходы.

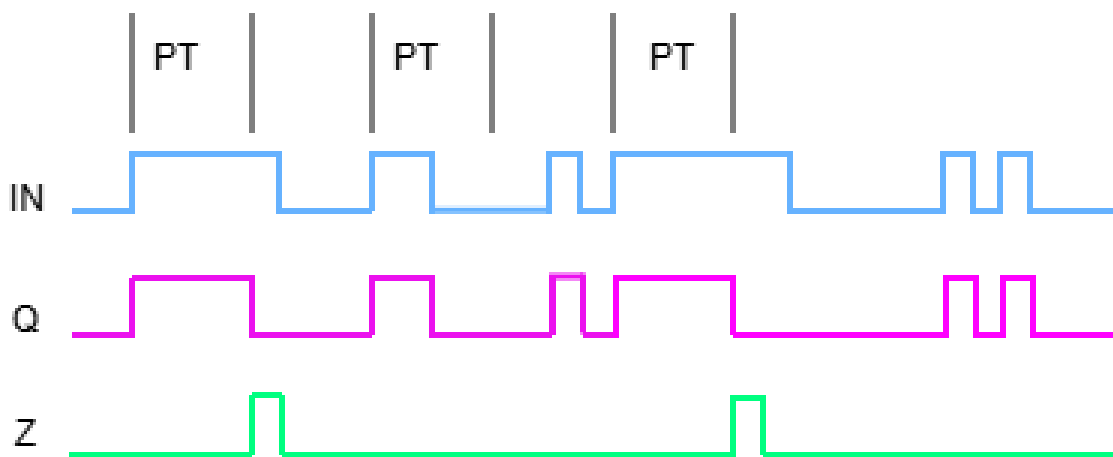


Рисунок 18.14 – Трассировка работы ФБ TMAX

18.19 TMIN

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	BOOL	Вход таймера
	PT	TIME	Минимальное время работы
Выходы	Q	BOOL	Выход таймера

Функциональный блок **TMIN** представляет собой таймер с ограничением на минимальное время включения. Пока вход **IN** имеет значение **TRUE**, таймер находится в работе и выход **Q** имеет значение **TRUE**. Если вход **IN** принимает значение **FALSE** до истечения времени **PT**, то по истечении времени **PT** выход **Q** принимает значение **FALSE**. Если вход **IN** принимает значение **FALSE** после истечения времени **PT**, то выход **Q** сразу принимает значение **FALSE**.

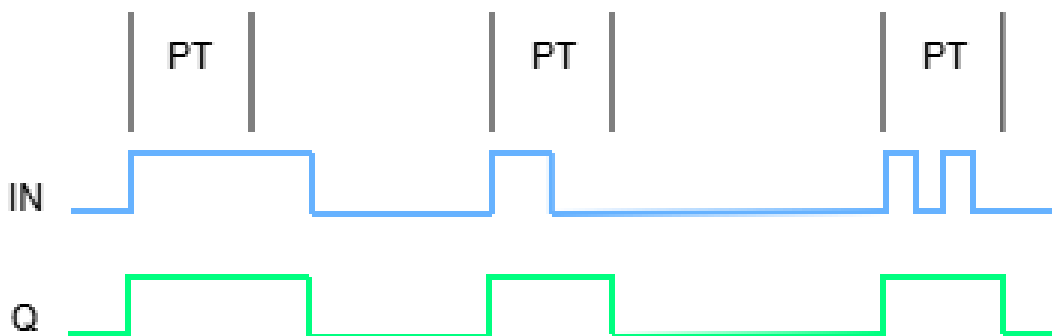


Рисунок 18.15 – Трассировка работы ФБ TMIN

18.20 TOF_1

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	BOOL	Вход таймера
	PT	TIME	Время работы
	RST	BOOL	Сброс таймера
Выходы	Q	BOOL	Выход таймера
Используемые модули	T_PLC_MS		

Функциональный блок **TOF1** представляет собой таймер с задержкой отключения. Блок запускается в работу по переднему фронту на входе **IN**, после чего выход **Q** принимает значение **TRUE**. По истечении времени **PT** выход **Q** принимает значение **FALSE**. По переднему фронту на входе **RST** блок прекращает работу и обнуляет выход.

Как можно заметить, в отличие от таймера **TOF** из библиотеки **Standard**, данный ФБ запускается по фронту (а не по уровню) и имеет дополнительный вход для сброса.

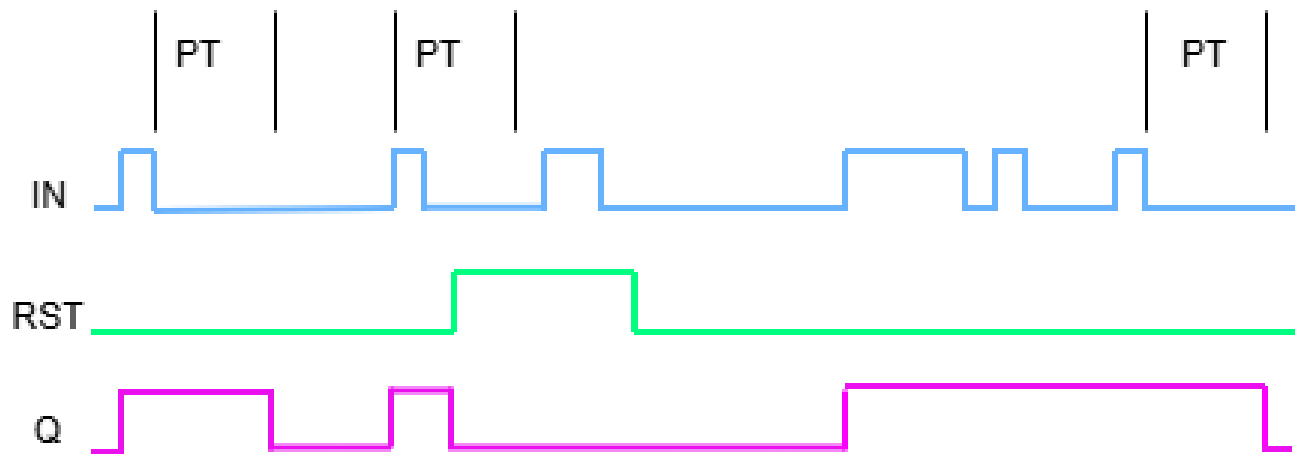


Рисунок 18.16 – Трассировка работы ФБ TOF1

18.21 TONOF

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	BOOL	Вход таймера
	T_ON	TIME	Задержка включения
	T_OFF	TIME	Задержка отключения
Выходы	Q	BOOL	Выход таймера

Функциональный блок **TONOF** представляет собой таймер с настраиваемыми задержками включения и отключения. Если вход **IN** принимает значение **TRUE**, то блок начинает работу. Спустя время **T_ON** выход **Q** принимает значение **TRUE**. Если вход **IN** принимает значение **FALSE**, то спустя время **T_OFF** выход **Q** принимает значение **FALSE**.

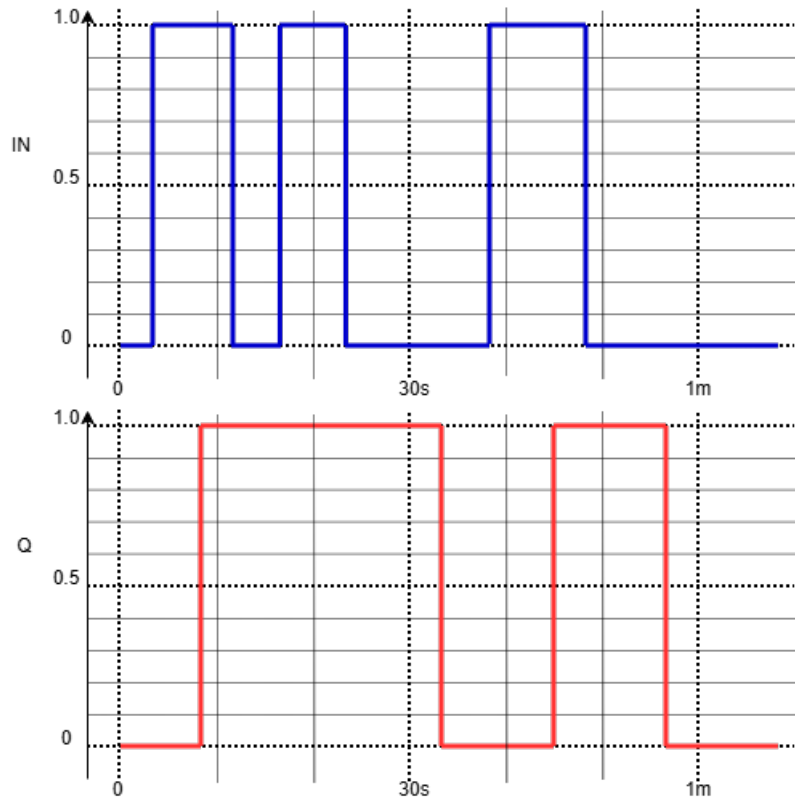


Рисунок 18.17 – Трассировка работы ФБ TONOF (T_ON=T#5s, T_OFF=T#10s)

18.22 TP_1

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	BOOL	Вход таймера
	PT	TIME	Время импульса
	RST	BOOL	Сброс таймера
Выходы	Q	BOOL	Выход таймера
Используемые модули	T_PLC_MS		

Функциональный блок **TP_1** представляет собой таймер. Блок запускается в работу по переднему фронту на входе **IN**, после чего выход **Q** принимает значение **TRUE**. Каждый импульс по переднему фронту на входе **IN** обнуляет счетчик таймера. По истечении времени **PT** выход **Q** принимает значение **FALSE**. По переднему фронту на входе **RST** блок прекращает работу и обнуляет выход.

Как можно заметить, в отличие от таймера **TP** из библиотеки **Standard**, данный ФБ перезапускается после каждого импульса на входе **IN** (в свою очередь **TP** детектирует первый импульс и обрабатывает в независимости от состояния **IN**, после чего может детектировать следующий импульс).

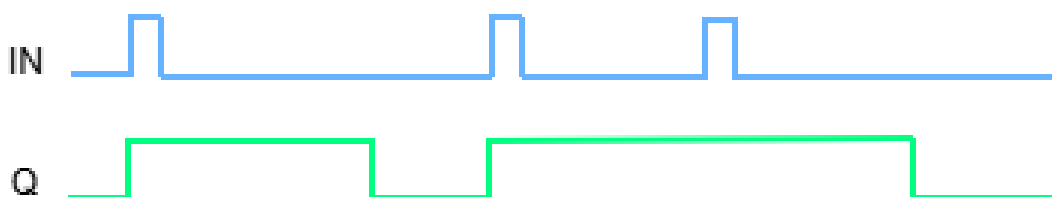


Рисунок 18.18 – Трассировка работы ФБ TP_1

18.23 TP_1D

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	BOOL	Вход таймера
	PT1	TIME	Время импульса
	PTD1	TIME	Время задержки
	RST	BOOL	Сброс таймера
Выходы	Q	BOOL	Выход таймера
	W	BOOL	Флаг «задержка»
Используемые модули	T_PLC_MS		

Функциональный блок **TP_1D** представляет собой таймер с задержкой. Блок запускается в работу по переднему фронту на входе **IN**, после чего выход **Q** принимает значение **TRUE**. Каждый импульс по переднему фронту на входе **IN** обнуляет счетчик таймера. По истечении времени **PT1** выход **Q** принимает значение **FALSE**. После этого выход **W** принимает значение **TRUE** и таймер делает задержку на время **PTD1**, во время которой не детектирует импульсы на входе **IN**. По переднему фронту на входе **RST** блок прекращает работу и обнуляет выход.

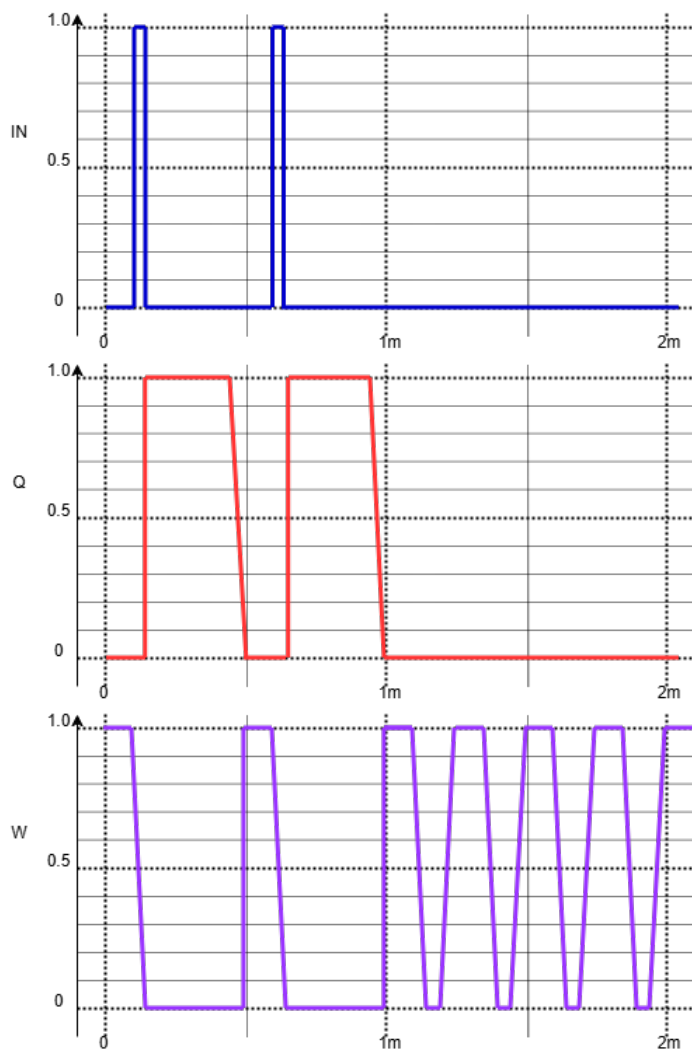


Рисунок 18.19 – Трассировка работы ФБ TP_1D

18.24 TP_X

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	BOOL	Вход таймера
	PT	TIME	Время импульса
Выходы	Q	BOOL	Выход таймера
	ET	TIME	Текущее время таймера
Используемые модули	T_PLC_MS		

Функциональный блок **TP_X** представляет собой таймер. Блок запускается в работу по переднему фронту на входе **IN**, после чего выход **Q** принимает значение **TRUE**. Каждый импульс по переднему фронту на входе **IN** обнуляет счетчик таймера. По истечении времени **PT** выход **Q** принимает значение **FALSE**. На выходе **ET** отображается текущее время таймера.

Как можно заметить, в отличие от таймера **TP** из библиотеки **Standard**, данный ФБ перезапускается после каждого импульса на входе **IN** (в свою очередь **TP** детектирует первый импульс и обрабатывает в независимости от состояния **IN**, после чего может детектировать следующий импульс).

Как можно заметить, в отличие от таймера **TP** из библиотеки **Standard**, данный ФБ перезапускается после каждого импульса на входе **IN** (в свою очередь **TP** детектирует первый импульс и обрабатывает в независимости от состояния **IN**, после чего может детектировать следующий импульс).

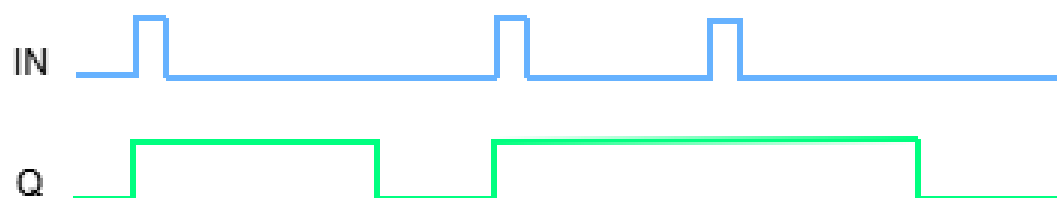


Рисунок 18.20 – Трассировка работы ФБ TP_X

19 Логические модули

19.1 BCDC_TO_INT

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	BYTE	Двухзначное число в формате BCD
Выходы	BCDC_TO_INT	INT	Двухзначное число в формате DEC

Функция **BCDC_TO_INT** конвертирует двухзначное число из двоично-десятичного формата в десятичный. См. также обратную функцию **INT_TO_BCD**.

Пример

```
VAR
    bcdInput  : BYTE := 2#01000010;
    decOutput : INT;
END_VAR

decOutput := OSCAT_BASIC.BCDC_TO_INT(bcdInput); // результат 42

END_PROGRAM
```

19.2 BIT_COUNT

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	DWORD	Строка бит
Выходы	BIT_COUNT	INT	Число бит со значением TRUE

Функция **BIT_COUNT** возвращает число бит со значением **TRUE** в переменной **IN** типа **DWORD**.

Пример

```
VAR
    InputVal : DWORD := 2#0011_0001;
    result   : INT;
END_VAR

result := OSCAT_BASIC.BIT_COUNT(InputVal); // результат 3

END_PROGRAM
```

19.3 BIT_LOAD_B

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	BYTE	Строка бит
	VAL	BOOL	Значение бита
	POS	INT	Номер бита (0...7)
Выходы	BIT_LOAD_B	BYTE	Строка бит с записанным битом

Функция **BIT_LOAD_B** устанавливает бит номер **POS** переменной **IN** типа **BYTE** в значение **VAL**.

Пример

```

VAR
  InputVal  : BYTE := 2#0011_0001;
  NewVal    : BOOL := TRUE;
  PosVal    : INT  := 1;
  result    : BYTE;
END_VAR

result := OSCAT_BASIC.BIT_LOAD_B(IN:=InputVal, VAL := NewVal, POS := PosVal);
// результат 2#00110011

END_PROGRAM

```

19.4 BIT_LOAD_B2

Тип модуля: функция	Переменная	Тип	Описание
Входы	I	BYTE	Строка бит
	D	BOOL	Значение бита
	P	INT	Номер начального бита для записи (0...7)
	N	INT	Число записываемых бит
Выходы	BIT_LOAD_B2	BYTE	Строка бит с записанными битами

Функция **BIT_LOAD_B2** устанавливает **N** бит переменной **I** типа **BYTE**, начиная с **P**-го, в значение **D**.

Пример

```

VAR
  InputVal  : BYTE := 2#0011_0001;
  NewVal    : BOOL := TRUE;
  PosVal    : INT  := 1;
  AmVal     : INT  := 3;
  result    : BYTE;
END_VAR

result := OSCAT_BASIC.BIT_LOAD_B2(I:=InputVal, D := NewVal, P := PosVal, N := AmVal);
// результат 2#00111111

END_PROGRAM

```

19.5 BIT_LOAD_DW

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	DWORD	Строка бит
	VAL	BOOL	Значение бита
	POS	INT	Номер бита (0...31)
Выходы	BIT_LOAD_DW	DWORD	Строка бит с записанным битом

Функция **BIT_LOAD_DW** устанавливает бит номер **POS** переменной **IN** типа **DWORD** в значение **VAL**.

Пример

```
VAR
    InputVal  : DWORD := 2#0000_0000_0000_0000_0000_0000_0000;
    NewVal    : BOOL  := TRUE;
    PosVal    : INT   := 2;
    result    : DWORD;
END_VAR

result := OSCAT_BASIC.BIT_LOAD_DW(IN := InputVal, VAL := NewVal, POS := PosVal);
(* результат 2#00000000000000000000000000000000100*)

END_PROGRAM
```

19.6 BIT_LOAD_DW2

Тип модуля: функция	Переменная	Тип	Описание
Входы	I	DWORD	Строка бит
	D	BOOL	Значение бита
	P	INT	Номер начального бита для записи (0...31)
	N	INT	Число записываемых бит
Выходы	BIT_LOAD_DW2	DWORD	Строка бит с записанными битами

Функция **BIT_LOAD_DW2** устанавливает **N** бит переменной **I** типа **DWORD**, начиная с **P**-го, в значение **D**.

Пример

[illegible]

19.7 BIT LOAD W

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	WORD	Строка бит
	VAL	BOOL	Значение бита
	POS	INT	Номер бита (0...15)
Выходы	BIT_LOAD_W	WORD	Строка бит с записанным битом

Функция **BIT_LOAD_W** устанавливает бит номер **POS** переменной **IN** типа **WORD** в значение **VAL**.

Пример


```

VAR
  InputVal : WORD := 2#0000_0000_0000_0000;
  NewVal   : BOOL := TRUE;
  PosVal   : INT  := 2;
  result    : WORD;
END_VAR

result := OSCAT_BASIC.BIT_LOAD_W(IN := InputVal, VAL := NewVal, POS := PosVal);
(* результат 2#00000000000000100*)

END_PROGRAM

```

19.8 BIT_LOAD_W2

Тип модуля: функция	Переменная	Тип	Описание
Входы	I	WORD	Строка бит
	D	BOOL	Значение бита
	P	INT	Номер начального бита для записи (0...15)
	N	INT	Число записываемых бит
Выходы	BIT_LOAD_W2	WORD	Строка бит с записанными битами

Функция **BIT_LOAD_W2** устанавливает **N** бит переменной **I** типа **WORD**, начиная с **P**-го, в значение **D**.

Пример

```

VAR
  InputVal : WORD := 2#0000_0000_0000_0000;
  NewVal   : BOOL := TRUE;
  PosVal   : INT  := 1;
  AmVal    : INT  := 3;
  result    : WORD;
END_VAR

result := OSCAT_BASIC.BIT_LOAD_W2(I:=InputVal,
                                   D := NewVal, P := PosVal, N := AmVal);
// результат 2#00000000000001110

END_PROGRAM

```

19.9 BIT_OF_DWORD

Тип модуля: функция	Переменная	Тип	Описание
Входы	in	DWORD	Строка бит
	N	INT	Номер бита (0...31)
Выходы	BIT_OF_DWORD	BOOL	Значение бита

Функция **BIT_OF_DWORD** возвращает значение бита номер **N** переменной **in**.

Пример

```

VAR
  InputVal : DWORD := 2#0000_1111_0000_1100_0011_1001_1111_0000;
  NumberVal : INT := 4;
  result    : BOOL;
END_VAR

result := OSCAT_BASIC.BIT_OF_DWORD(in := InputVal, N := NumberVal); // результат TRUE

END_PROGRAM

```

19.10 BIT_TOGGLE_B

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	BYTE	Строка бит
	POS	INT	Номер бита (0...7)
Выходы	BIT_TOGGLE_B	BYTE	Строка бит с инвертированным битом

Функция **BIT_TOGGLE_B** инвертирует бит номер **POS** переменной **in**.

Пример

```
VAR
    InputVal  : BYTE := 2#111_0000;
    PosVal    : INT  := 3;
    result    : BYTE;
END_VAR

result := OSCAT_BASIC.BIT_TOGGLE_B(IN:=InputVal, POS := PosVal);
// результат 2#11111000

END_PROGRAM
```

19.11 BIT_TOGGLE_DW

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	DWORD	Строка бит
	POS	INT	Номер бита (0...31)
Выходы	BIT_TOGGLE_DW	DWORD	Строка бит с инвертированным битом

Функция **BIT_TOGGLE_DW** инвертирует бит номер **POS** переменной **in**.

Пример

```
VAR
    InputVal  : DWORD := 2#0000_0000_0000_0000_0000_0000_1111_0000;
    PosVal    : INT  := 3;
    result    : DWORD;
END_VAR

result := OSCAT_BASIC.BIT_TOGGLE_DW(IN:=InputVal, POS := PosVal);
// результат 2#0000000000000000000000000000000011111000

END_PROGRAM
```

19.12 BIT_TOGGLE_W

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	WORD	Строка бит
	POS	INT	Номер бита (0...15)
Выходы	BIT_TOGGLE_W	WORD	Строка бит с инвертированным битом

Функция **BIT_TOGGLE_W** инвертирует бит номер **POS** переменной **in**.

Пример

```

VAR
  InputVal  : WORD := 2#0000_0000_1111_0000;
  PosVal    : INT  := 3;
  result    : WORD;
END_VAR

result := OSCAT_BASIC.BIT_TOGGLE_W(IN:=InputVal, POS := PosVal);
// результат 2#0000000011111000

END_PROGRAM

```

19.13 BYTE_OF_BIT

Тип модуля: функция	Переменная	Тип	Описание
Входы	B0...B7	BOOL	Отдельные биты (B0 – младший)
Выходы	BYTE_OF_BIT	BYTE	Байт, собранный из отдельных бит

Функция **BYTE_OF_BIT** собирает 8 **BOOL** переменных в переменную типа **BYTE**.

Пример

```

VAR
  bit0 : BOOL := TRUE;    // младший бит
  bit1 : BOOL := FALSE;
  bit2 : BOOL := TRUE;
  bit3 : BOOL := FALSE;
  bit4 : BOOL := TRUE;
  bit5 : BOOL := FALSE;
  bit6 : BOOL := FALSE;
  bit7 : BOOL := TRUE;    // старший бит

  resultByte : BYTE;
END_VAR

resultByte := OSCAT_BASIC.BYTE_OF_BIT(
  B0 := bit0, B1 := bit1, B2 := bit2, B3 := bit3,
  B4 := bit4, B5 := bit5, B6 := bit6, B7 := bit7);
// результат 10010101 (старший бит слева)

END_PROGRAM

```

19.14 BYTE_OF_DWORD

Тип модуля: функция	Переменная	Тип	Описание
Входы	in	DWORD	Исходная переменная
	N	BYTE	Номер считываемого байта (0...3)
Выходы	BYTE_OF_DWORD	BYTE	Выделенный байт

Функция **BYTE_OF_DWORD** свдечляет из переменной **in** типа **DWORD** байт с номер **N**. См. также обратную функцию **DWORD_OF_BYTE**.

Пример

```

VAR
    InputVar    : DWORD := 16#AABBCCDD;
    NumberVar   : BYTE  := 1;
    resultByte  : BYTE;
END_VAR

resultByte := OSCAT_BASIC.BYTE_OF_DWORD(in := InputVar, N := NumberVar);
// результат 16#CC

END_PROGRAM

```

19.15 BYTE_TO_BITS

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	in	BYTE	Исходная переменная
Выходы	B0...B7	BOOL	Выделенные биты (B0 – младший)

ФБ **BYTE_TO_BITS** извлекает отдельные биты из переменной **in** типа **BYTE**.

Пример

```

VAR
    inputByte : BYTE := 16#95;           // 2#1001_0101
    ByteToBitsInst : OSCAT_BASIC.BYTE_TO_BITS; // экземпляр ФБ

    bit0, bit1, bit2, bit3 : BOOL;        // выходные биты
    bit4, bit5, bit6, bit7 : BOOL;        // выходные биты
END_VAR

ByteToBitsInst(
    in := inputByte,
    B0 => bit0,      // bit0 = TRUE
    B1 => bit1,      // bit1 = FALSE
    B2 => bit2,      // bit2 = TRUE
    B3 => bit3,      // bit3 = FALSE
    B4 => bit4,      // bit4 = TRUE
    B5 => bit5,      // bit5 = FALSE
    B6 => bit6,      // bit6 = FALSE
    B7 => bit7,      // bit7 = TRUE
);

END_PROGRAM

```

19.16 BYTE_TO_GRAY

Тип модуля: функция	Переменная	Тип	Описание
Входы	in	BYTE	Число в двоичном коде
Выходы	BYTE_TO_GRAY	BYTE	Число в коде Грея

Функция **BYTE_TO_GRAY** преобразует число в двоичном коде **in** в число в коде Грея. См. также обратную функцию [GRAY_TO_BYTE](#).

Пример

```

VAR
    InputVal : BYTE := 2#0001_0110;
    result   : BYTE;
END_VAR

result := OSCAT_BASIC.BYTE_TO_GRAY(InputVal); // результат 2#0001_1101

END_PROGRAM

```

19.17 CHECK_PARITY

Тип модуля: функция	Переменная	Тип	Описание
Входы	in	DWORD	Проверяемая переменная
	p	BOOL	Бит четности
Выходы	CHECK_PARITY	BOOL	Результат проверки

Функция **CHECK_PARITY** проверяет переменную **in** типа **DWORD** с битом четности **p** на четность и возвращает **TRUE** в случае, если общее число битов со значением **TRUE** (включая значение бита четности) является четным числом. См. также функцию **PARITY**.

Пример

```
VAR
  InputVar   : DWORD := 16#1110_0000;
  ParityVar  : BOOL  := TRUE;
  result     : BOOL;
END_VAR

result := OSCAT_BASIC.CHECK_PARITY(in := InputVar, p := ParityVar); // результат TRUE

END_PROGRAM
```

19.18 CHK_REAL

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Проверяемое число с плавающей точкой
Выходы	CHK_REAL	BYTE	Код состояния числа

Функция **CHK_REAL** проверяет состояние числа с плавающей точкой **X** и возвращает код состояния согласно стандарту IEEE 754-2008:

- 16#00 – число с плавающей точкой;
- 16#20 – $+\infty$;
- 16#40 – $-\infty$;
- 16#80 – NaN*.

* — не число.

Пример

```
VAR
  InputVal : REAL := 11.22;
  result   : BYTE;
END_VAR

result := OSCAT_BASIC.CHK_REAL(InputVal); // результат 16#00

END_PROGRAM
```

19.19 CRC_GEN

Тип модуля: функция	Переменная	Тип	Описание
Входы	PT	POINTER TO ARRAY [0..32000] OF BYTE	Пакет данных, для которого будет сгенерирована контрольная сумма
	SIZE	INT	Размер пакета данных (в байтах)
	PL	INT	Степень порождающего полинома
	PN	DWORD	Порождающий полином

Тип модуля: функция	Переменная	Тип	Описание
	INIT	DWORD	Стартовые данные
	REV_IN	BOOL	Направление вычислений
	REV_OUT	BOOL	Инверсия порядка битов
	XOR_OUT	DWORD	Значение для XOR
Выходы	CRG_GEN	DWORD	Сгенерированная контрольная сумма
Используемые модули	REVERSE , REFLECT		

Функция **CRC_GEN** генерирует контрольную сумму (по алгоритму CRC) для пакета данных размером **SIZE** байт, расположенных по указателю **PT**. В большинстве случаев при вызове функции используются операторы **ADR()** и **SIZEOF()** – первый возвращает указатель на заданную переменную (массив байт), а второй – ее размер в байтах.

Входная переменная **PN** определяет производящий полином для функции, **PL** – его степень. Максимально допустимая степень полинома, используемого в функции – **32**. Вход **INIT** определяет стартовые данные для генерации (обычно **0x0000** или **0xFFFF**). Вход **REV_IN** определяет порядок вычислений: **FALSE** – со старшего значащего бита, **TRUE** – с младшего. Вход **REV_OUT** определяет порядок бит при входе на элемент **XOR**: **FALSE** – прямой, **TRUE** – инвертированный. Вход **XOR_OUT** определяет число, которое используется при операции **XOR**.

Функция способна работать с пакетами данных с длиной **не менее 4 байта**.

Ниже приведены значения входов функции для некоторых алгоритмов **CRC**. Более подробная информация может быть найдена в сети. Приведем несколько полезных ссылок:

<http://reveng.sourceforge.net/crc-catalogue/> (каталог алгоритмов CRC)

<http://zorc.breitbandkatze.de/crc.html> (онлайн-генератор контрольных сумм)

<http://www.tahapaksu.com/crc/> (еще один онлайн-генератор, удобен при работе с Modbus)

Таблица 19.1 – Таблица аргументов функции CRC_GEN для некоторых алгоритмов CRC

CRC	PL (DEC)	PN (HEX)	INIT (HEX)	REV_IN	REV_OUT	XOR_OUT (HEX)
CRC-3/ROHC	3	0x3	0x7	TRUE	TRUE	0x0
CRC-4/ITU	4	0x3	0x0	TRUE	TRUE	0x0
CRC5/EPC	5	0x9	0x9	FALSE	FALSE	0x0
CRC-5/ITU	5	0x15	0x0	TRUE	TRUE	0x0
CRC-5/USB	5	0x5	0x1F	TRUE	TRUE	0x1F
CRC-6/DARC	6	0x19	0x0	TRUE	FALSE	0x0
CRC-6/ITU	6	0x3	0x0	TRUE	TRUE	0x0
CRC-7	7	0x9	0x0	FALSE	FALSE	0x0
CRC-7/ROHC	7	0x4F	0x7F	TRUE	TRUE	0x0
CRC-8	8	0x7	0x0	FALSE	FALSE	0x0
CRC-8/DARC	8	0x39	0x0	TRUE	TRUE	0x0
CRC-8/I-CODE	8	0x1D	0xFD	FALSE	FALSE	0x0
CRC-8/ITU	8	0x7	0x0	FALSE	FALSE	0x55
CRC-8/MAXIM	8	0x31	0x0	TRUE	TRUE	0x0
CRC-8/ROHC	8	0x7	0xFF	TRUE	TRUE	0x0
CRC-8/WCDNA	8	0x9B	0x0	TRUE	TRUE	0x0
CRC-10	10	0x233	0x0	FALSE	FALSE	0x0
CRC-11	11	0x385	0x1A	FALSE	FALSE	0x0
CRC-12/3GPP	12	0x80F	0x0	FALSE	TRUE	0x0
CRC-12/DECT	12	0x80F	0x0	FALSE	FALSE	0x0
CRC-14/DARC	14	0x805	0x0	TRUE	TRUE	0x0

Продолжение таблицы 19.1

CRC	PL (DEC)	PN (HEX)	INIT (HEX)	REV_IN	REV_OUT	XOR_OUT (HEX)
CRC-15	15	0x4599	0x0	FALSE	FALSE	0x0
CRC-16/LHA	16	0x8005	0x0	TRUE	TRUE	0x0
CRC-16/ CCITT-AUG	16	0x1021	0x1D0F	FALSE	FALSE	0x0
CRC-16/ BUYPASS	16	0x8005	0x0	FALSE	FALSE	0x0
CRC-16/ CCITT-FALSE	16	0x1021	0xFFFF	FALSE	FALSE	0x0
CRC-16/DDS	16	0x8005	0x800D	FALSE	FALSE	0x0
CRC-16/ DECT-R	16	0x589	0x0	FALSE	FALSE	0x1
CRC	PL (DEC)	PN (HEX)	INIT (HEX)	REV_IN	REV_OUT	XOR_OUT (HEX)
CRC-16/ DECT-X	16	0x589	0x0	FALSE	FALSE	0x0
CRC-16/DNP	16	0x3D65	0x0	TRUE	TRUE	0xFFFF
CRC-16/ EN13757	16	0x3D65	0x0	FALSE	FALSE	0xFFFF
CRC-16/ GENIBUS	16	0x1021	0xFFFF	FALSE	FALSE	0xFFFF
CRC-16/ MAXIM	16	0x8005	0x0	TRUE	TRUE	0xFFFF
CRC-16/ MCRF4XX	16	0x1021	0xFFFF	TRUE	TRUE	0x0
CRC-16/ RIELLO	16	0x1021	0xB2AA	TRUE	TRUE	0x0
CRC-16/T10- DIF	16	0x8BB7	0x0	FALSE	FALSE	0x0
CRC-16/ TELEDISK	16	0xA097	0x0	FALSE	FALSE	0x0
CRC-16/USB	16	0x8005	0xFFFF	TRUE	TRUE	0xFFFF
CRC-16/ CCITT-TRUE	16	0x1021	0x0	TRUE	TRUE	0x0
CRC-16/ MODBUS	16	0x8005	0x0	TRUE	TRUE	0x0
CRC-16/X-25	16	0x1021	0xFFFF	TRUE	TRUE	0xFFFF
CRC-16/ XMODEM	16	0x1021	0x0	FALSE	FALSE	0x0
CRC-24/ OPENPGP	24	0x864CFB	0xB704CE	FALSE	FALSE	0x0
CRC-24/ FLEXRAY-A	24	0x5D6DCB	0xFEDCBA	FALSE	FALSE	0x0
CRC-24/ FLEXRAY-B	24	0x5D6DCB	0xABCDEF	FALSE	FALSE	0x0
CRC-32/ PKZIP	32	0x04C11DB7	0xFFFFFFFF	TRUE	TRUE	0xFFFFFFFF
CRC-32/BZIP2	32	0x04C11DB7	0xFFFFFFFF	FALSE	FALSE	0xFFFFFFFF
CRC-32/ CASTAGNOLI	32	0x1EDC6F41	0xFFFFFFFF	TRUE	TRUE	0xFFFFFFFF
CRC-32/D	32	0xA833982B	0xFFFFFFFF	TRUE	TRUE	0xFFFFFFFF
CRC-32/ MPEG2	32	0x04C11DB7	0xFFFFFFFF	FALSE	FALSE	0x0

Продолжение таблицы 19.1

CRC	PL (DEC)	PN (HEX)	INIT (HEX)	REV_IN	REV_OUT	XOR_OUT (HEX)
CRC-32/POSIX	32	0x04C11DB7	0x0	FALSE	FALSE	0xFFFFFFFF
CRC-32/Q	32	0x814141AB	0x0	FALSE	FALSE	0x0
CRC-32/JAM	32	0x04C11DB7	0xFFFFFFFF	TRUE	TRUE	0x0
CRC-32/XFER	32	0xAF	0x0	FALSE	FALSE	0x0

Ниже приведен пример использования функции **CRC_GEN** для расчета контрольной суммы пакета данных протокола Modbus.

**ПРИМЕЧАНИЕ**

Рассчитанная контрольная сумма представляет собой переменную типа **DWORD**, младший байт которой соответствует старшему байту контрольной суммы, прикладываемой к пакету данных.

Пример

19.20 DEC_2

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	D	BOOL	Вход декодера
	A	BOOL	Бит разрешения
Выходы	Q0	BOOL	Выход 0
	Q1	BOOL	Выход 1

Функциональный блок **DEC_2** представляет собой двухбитный декодер. Принцип работы блока:

- если A=FALSE, то Q0:=D;
- если A=TRUE, то Q1:=D.

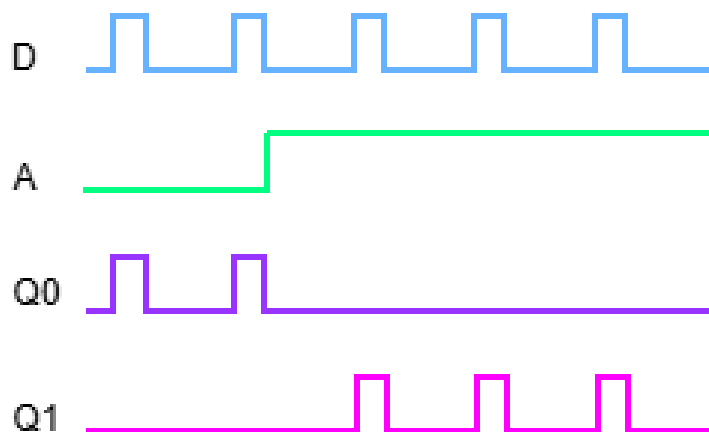


Рисунок 19.1 – Трассировка работы ФБ DEC_2

19.21 DEC_4

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	D	BOOL	Вход декодера
	A0	BOOL	Бит разрешения 0
	A1	BOOL	Бит разрешения 1
Выходы	Q0	BOOL	Выход 0
	Q1	BOOL	Выход 1
	Q2	BOOL	Выход 2
	Q3	BOOL	Выход 3

Функциональный блок **DEC_4** представляет собой четырехбитный декодер. Принцип работы блока:

- если A0=FALSE и A1=FALSE, то Q0:=D;
- если A0=TRUE и A1=FALSE, то Q1:=D;
- если A0=FALSE и A1=TRUE, то Q2:=D;
- если A0=TRUE и A1=TRUE, то Q3:=D.

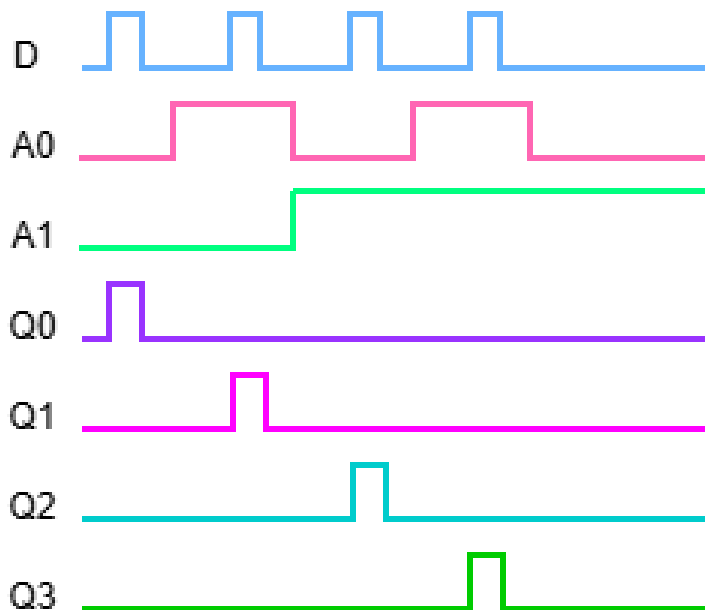


Рисунок 19.2 – Трассировка работы ФБ DEC_4

19.22 DEC_8

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	D	BOOL	Вход декодера
	A0	BOOL	Бит разрешения 0
	A1	BOOL	Бит разрешения 1
	A2	BOOL	Бит разрешения 2
Выходы	Q0	BOOL	Выход 0
	Q1	BOOL	Выход 1
	Q2	BOOL	Выход 2
	Q3	BOOL	Выход 3
	Q4	BOOL	Выход 4
	Q5	BOOL	Выход 5
	Q6	BOOL	Выход 6
	Q7	BOOL	Выход 7

Функциональный блок **DEC_8** представляет собой восьмибитный декодер. Принцип работы блока:

- если A0=FALSE и A1=FALSE и A2=FALSE, то Q0:=D;
- если A0=TRUE и A1=FALSE и A2=FALSE, то Q1:=D;
- если A0=FALSE и A1=TRUE и A2=FALSE, то Q2:=D;
- если A0=TRUE и A1=TRUE и A2=FALSE, то Q3:=D;
- если A0=FALSE и A1=FALSE и A2=TRUE, то Q4:=D;
- если A0=TRUE и A1=FALSE и A2=TRUE, то Q5:=D;
- если A0=FALSE и A1=TRUE и A2=TRUE, то Q6:=D;
- если A0=TRUE и A1=TRUE и A2=TRUE, то Q7:=D.

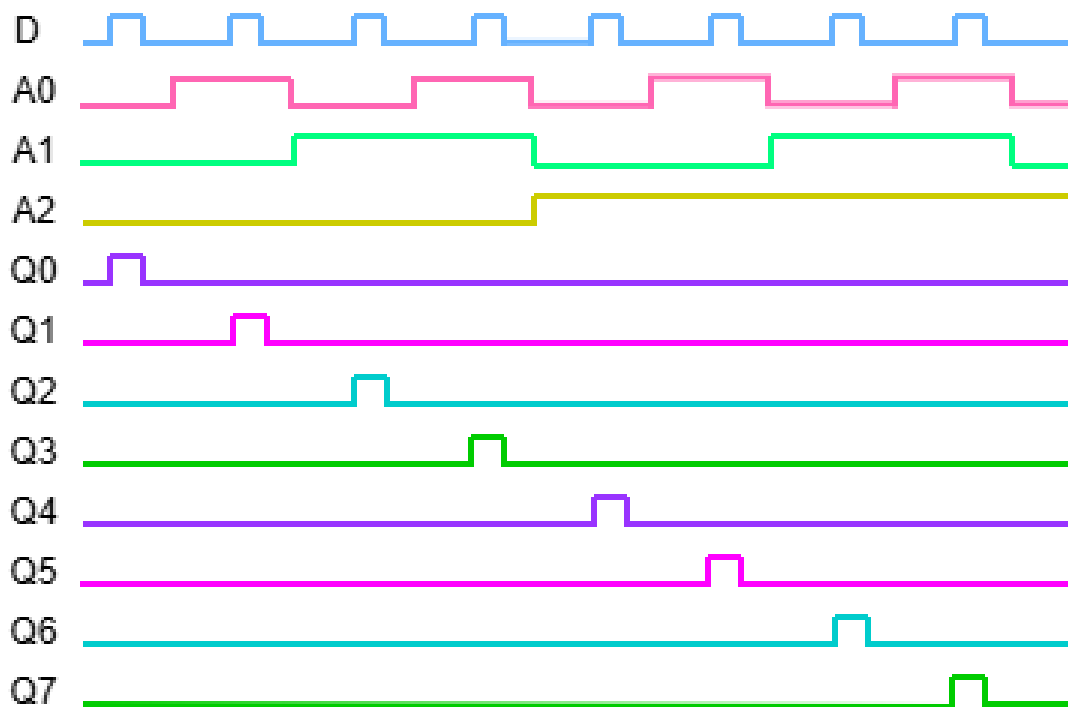


Рисунок 19.3 – Трассировка работы ФБ DEC_8

19.23 DW_TO_REAL

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	DWORD	Входное значение
Выходы	DW_TO_REAL	REAL	Представление входного значения как REAL

Функция **DW_TO_REAL** интерпретирует входную переменную **X** типа **DWORD** как значение с плавающей точкой, представленное в формате IEEE 754. В отличие от стандартного оператора конверсии **DWORD_TO_REAL**, который преобразует десятичное представление переменной типа **DWORD** в соответствующее число с плавающей точкой (например, 5->5.0), данная функция не меняет исходное значение, а только представляет его в другом формате. В частности, это необходимо при передаче переменных типа **REAL** по протоколу Modbus, который не стандартизирует передаваемые типы данных.

См. также обратную функцию [REAL_TO_DW](#).

Пример

```

VAR
    InputVar : DWORD := 16#4133851F;
    result   : REAL;
END_VAR

result := OSCAT_BASIC.DW_TO_REAL(X := InputVar); // результат 11.22

END_PROGRAM

```

19.24 DWORD_OF_BYTE

Тип модуля: функция	Переменная	Тип	Описание
Входы	B3...B0	BYTE	Исходные байты (B0 – младший)
Выходы	DWORD_OF_BYTE	DWORD	Собранное значение

Функция **DWORD_OF_BYTE** возвращает значение типа **DWORD**, собранное из отдельных байтов **B3...B0**, где **B0** является младшим байтом.

См. также обратную функцию [BYTE_OF_DWORD](#).

Пример

```
VAR
  bt0 : BYTE := 16#AA;    // младший байт
  bt1 : BYTE := 16#BB;
  bt2 : BYTE := 16#CC;
  bt3 : BYTE := 16#DD;
  resultDW : DWORD;
END_VAR

resultDW := OSCAT_BASIC.DWORD_OF_BYTE(
  B0 := bt0, B1 := bt1, B2 := bt2, B3 := bt3
);
// результат 16#DDCCBBAA

END_PROGRAM
```

19.25 DWORD_OF_WORD

Тип модуля: функция	Переменная	Тип	Описание
Входы	W1...W0	WORD	Исходные слова (W0 – младшее)
Выходы	DWORD_OF_WORD	DWORD	Собранное значение

Функция **DWORD_OF_WORD** возвращает значение типа **DWORD**, собранное из отдельных слов **W1** и **W0**, где **W0** является младшим словом.

См. также обратную функцию [WORD_OF_DWORD](#).

Пример

```
VAR
  word0 : WORD := 16#BBAA; //младшее слово
  word1 : WORD := 16#DDCC;
  resultDW : DWORD;
END_VAR

resultDW := OSCAT_BASIC.DWORD_OF_WORD(
  W0 := word0, W1 := word1);
// результат 16#DDCCBBAA

END_PROGRAM
```

19.26 GRAY_TO_BYTE

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	BYTE	Число в коде Грея
Выходы	GRAY_TO_BYTE	BYTE	Число в двоичном коде

Функция **GRAY_TO_BYTE** преобразует число в коде Грея **IN** в число в двоичном коде.

См. также обратную функцию [BYTE_TO_GRAY](#).

Пример

```

VAR
  inVar  : BYTE := 2#0001_1101;
  resVar : BYTE;
END_VAR

resVar := OSCAT_BASIC.GRAY_TO_BYTE(IN := inVar); (* результат 2#0001_0110 *)

END_PROGRAM

```

19.27 INT_TO_BCD

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	INT	Двухзначное число в формате DEC
Выходы	INT_TO_BCD	BYTE	Двухзначное число в формате BCD

Функция **INT_TO_BCD** конвертирует двухзначное число из десятичного формата в двоично-десятичный.

См. также обратную функцию [BCDC_TO_INT](#).

Пример

```

VAR
  inVar  : INT := 31;
  resVar : BYTE;
END_VAR

resVar := OSCAT_BASIC.INT_TO_BCD(IN := inVar); (* результат 2#0011_0001 *)

END_PROGRAM

```

19.28 MATRIX

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	x1...x5	BOOL	Сигналы строк
Выходы	code	BYTE	Код нажатия
	TP	BOOL	Флаг «произошло нажатие»
	y1...y4	BOOL	Сигналы опроса столбцов
Параметры	Release	BOOL	Обработка отпускания кнопки

Функциональный блок **MATRIX** используется для опроса матричной клавиатуры размером **5x4**. На входы **x1...5** заводятся сигналы от строк. ФБ последовательно генерирует импульсы на выходах **y1...y4**, которые должны использоваться для опроса соответствующих столбцов. Если один из входов имеет значение **TRUE**, то на выход **code** поступает код нажатой кнопки, содержащий номер ее строки и столбца, а на выходе **TP** генерируется единичный импульс. Если параметр **Release** имеет значение **TRUE**, то обрабатывается также отпускание кнопки.

Расшифровка битов кода		
Бит 7	TRUE – кнопка нажата, FALSE – кнопка отпущена	
Бит 6	Номер столбца (бит 2)	От 0 до 3
Бит 5	Номер столбца (бит 1)	
Бит 4	Номер столбца (бит 0)	
Бит 3	Всегда FALSE	
Бит 2	Номер строки (бит 2)	От 1 до 5
Бит 1	Номер строки (бит 1)	
Бит 0	Номер строки (бит 0)	

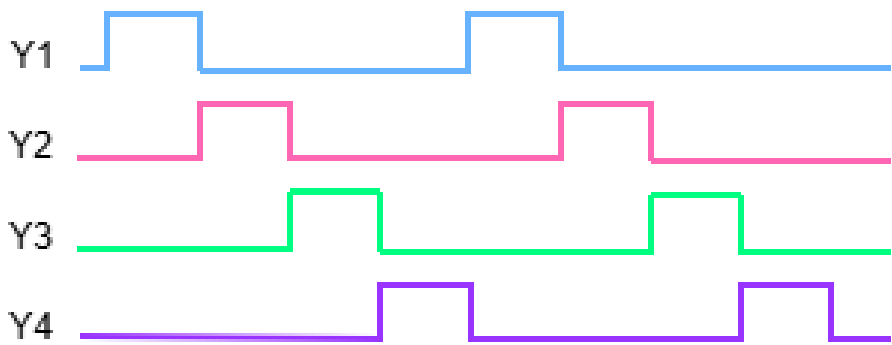


Рисунок 19.4 – Диаграмма опроса столбцов

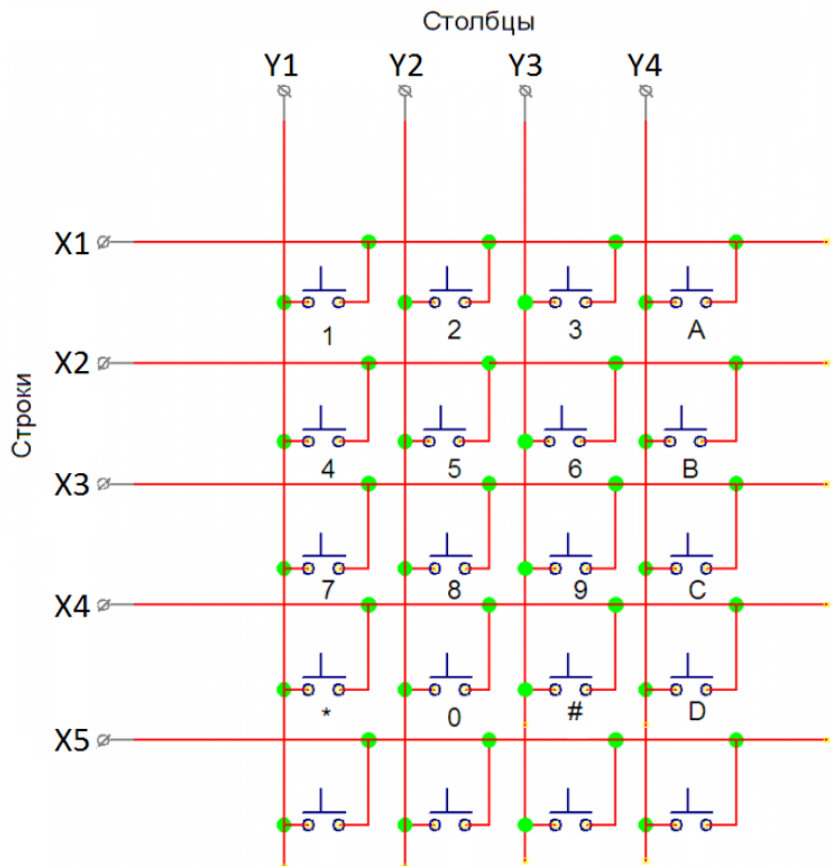


Рисунок 19.5 – Принципиальная схема матричной клавиатуры

19.29 MUX_2

Тип модуля: функция	Переменная	Тип	Описание
Входы	D0	BOOL	Информационный вход 0
	D1	BOOL	Информационный вход 1
	A0	BOOL	Адресный вход
Выходы	MUX_2	BOOL	Выход мультиплексора

Функция **MUX_2** реализует двухбитный мультиплексор. Принцип работы функции:

- если A0=FALSE, то MUX_2:=D0;
- если A0=TRUE, то MUX_2:=D1.

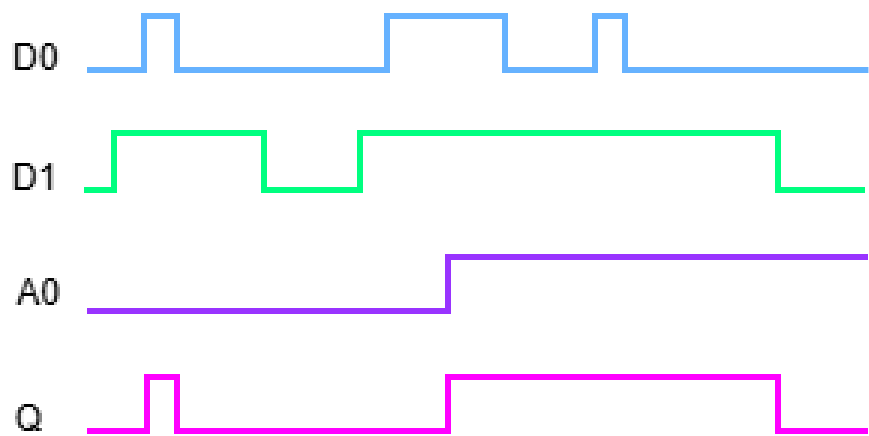


Рисунок 19.6 – Трассировка работы функции MUX_2

19.30 MUX_4

Тип модуля: функция	Переменная	Тип	Описание
Входы	D0	BOOL	Информационный вход 0
	D1	BOOL	Информационный вход 1
	D2	BOOL	Информационный вход 2
	D3	BOOL	Информационный вход 3
	A0	BOOL	Адресный вход 0
	A1	BOOL	Адресный вход 1
Выходы	MUX_4	BOOL	Выход мультиплексора

Функция **MUX_4** реализует четырехбитный мультиплексор. Принцип работы функции:

- если A0=FALSE и A1=FALSE, то MUX_4:=D0;
- если A0=TRUE и A1=FALSE, то MUX_4:=D1;
- если A0=FALSE и A1=TRUE, то MUX_4:=D2;
- если A0=TRUE и A1=TRUE, то MUX_4:=D3.

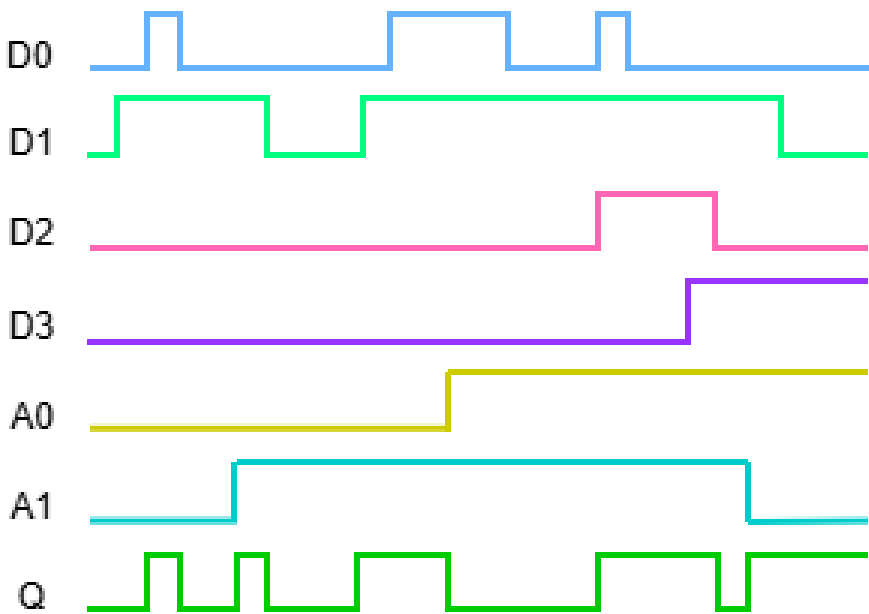


Рисунок 19.7 – Трассировка работы функции MUX_4

19.31 PARITY

Тип модуля: функция	Переменная	Тип	Описание
Входы	in	BYTE	Исходный байт
Выходы	PARITY	BOOL	Бит четности

Функция **PARITY** вычисляет бит четности для байта **in**. Соответственно, если число битов в состоянии **TRUE** в байте является нечетным, то бит четности = **TRUE**. В противном случае бит четности = **FALSE**.

См. также функцию [CHECK_PARITY](#).

Пример

```
VAR
  InputVar   : BYTE := 2#1111_1000;
  result     : BOOL;
END_VAR

result := OSCAT_BASIC.PARITY(in := InputVar); // результат TRUE

END_PROGRAM
```

19.32 REAL_TO_DW

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Входное значение
Выходы	REAL_TO_DW	DWORD	Представление входного значения как DWORD

Функция **REAL_TO_DW** интерпретирует входную переменную **X** типа **REAL** как значение в шестнадцатеричной системе. В отличие от стандартного оператора конверсии **REAL_TO_DWORD**, который округляет значение с плавающей точкой до ближайшего меньшего целого (например, 5.7->5), данная функция не меняет исходное значение, а только представляет его в другом формате. В частности, это необходимо при передаче переменных типа **REAL** по протоколу Modbus, который не стандартизирует передаваемые типы данных.

См. также обратную функцию [DW_TO_REAL](#).

Пример

```
VAR
  inVar   : REAL := 11.22;
  resVar  : DWORD;
END_VAR

resVar := OSCAT_BASIC.REAL_TO_DW(X := inVar); (* результат 16#4133851F *)

END_PROGRAM
```

19.33 REFLECT

Тип модуля: функция	Переменная	Тип	Описание
Входы	D	DWORD	Строка бит
	L	INT	Длина отражаемой части
Выходы	REFLECT	DWORD	Строка бит с обратным порядком бит

Функция **REFLECT** меняет порядок **L** младших бит переменной **D** на обратный.

Пример

```

VAR
  inVar  : DWORD := 2#1100_0001;
  LVar   : INT  := 4;
  resVar : DWORD;
END_VAR

resVar := OSCAT_BASIC.REFLECT(D := inVar, L := LVar);
(* результат 2#0000_0000_0000_0000_0000_0000_1100_1000 *)

END_PROGRAM

```

19.34 REVERSE

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	BYTE	Строка бит
Выходы	REVERSE	BYTE	Строка бит с обратным порядком бит

Функция **REVERSE** меняет порядок бит переменной **IN** на обратный.

Пример

```

VAR
  inVar  : BYTE := 2#1100_0001;
  resVar : BYTE;
END_VAR

resVar := OSCAT_BASIC.REVERSE(IN := inVar); (* результат 2#1000_0011 *)

END_PROGRAM

```

19.35 SHL1

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	DWORD	Строка бит
	N	INT	Кол-во сдвигаемых бит
Выходы	SHL1	DWORD	Строка бит, сдвинутая влево

Функция **SHL1** выполняет побитный сдвиг переменной **IN** влево на **N** бит с дополнением единицами справа (в отличие от стандартного оператора **SHL**, который производит дополнение нулями).

Пример

```

VAR
  inVar  : DWORD := 2#0000_0000_1100_0001;
  NVar   : INT  := 8;
  resVar : DWORD;
END_VAR

resVar := OSCAT_BASIC.SHL1(IN := inVar, N := NVar);
(* результат 2#0000_0000_0000_0000_1100_0001_1111_1111 *)

END_PROGRAM

```

19.36 SHR1

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	DWORD	Строка бит
	N	INT	Кол-во сдвигаемых бит
Выходы	SHR1	DWORD	Строка бит, сдвинутая вправо

Функция **SHR1** выполняет побитный сдвиг переменной **IN** вправо на **N** бит с дополнением единицами слева (в отличие от стандартного оператора **SHR**, который производит дополнение нулями).

Пример

```
VAR
  inVar  : DWORD := 2#0000_0000_0000_0000_0000_0000_1100_0001;
  NVar   : INT  := 4;
  resVar : DWORD;
END_VAR

resVar := OSCAT_BASIC.SHR1(IN := inVar, N := NVar);
(* результат 2#1111_0000_0000_0000_0000_0000_0000_1100 *)

END_PROGRAM
```

19.37 SWAP_BYTE

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	WORD	Исходная переменная
Выходы	SWAP_BYTE	WORD	Переменная с переставленными байтами

Функция **SWAP_BYTE** меняет порядок байт переменной **IN**.

Пример

```
VAR
  inVar  : WORD := 16#ADDE;
  resVar : WORD;
END_VAR

resVar := OSCAT_BASIC.SWAP_BYTE(IN := inVar); (* результат 16#DEAD *)

END_PROGRAM
```

19.38 SWAP_BYTE2

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	DWORD	Исходная переменная
Выходы	SWAP_BYTE	DWORD	Переменная с обратным порядком байт

Функция **SWAP_BYTE2** меняет порядок байт переменной **IN** на обратный.

Пример

```
VAR
  inVar  : DWORD := 16#AABBCCDD;
  resVar : DWORD;
END_VAR

resVar := OSCAT_BASIC.SWAP_BYTE2(IN := inVar); (* результат 16#DDCCBBAA *)

END_PROGRAM
```

19.39 WORD_OF_BYTE

Тип модуля: функция	Переменная	Тип	Описание
Входы	B1...B0	BYTE	Исходные байты (B0 – младший)
Выходы	WORD_OF_BYTE	WORD	Собранное значение

Функция **WORD_OF_BYTE** возвращает значение типа **DWORD**, собранное из отдельных байтов **B1...B0**, где **B0** является младшим байтом.

Пример

```
VAR
    bt0 : BYTE := 16#AD;    // младший байт
    bt1 : BYTE := 16#DE;
    resW : WORD;
END_VAR

resW := OSCAT_BASIC.WORD_OF_BYTE(B0 := bt0, B1 := bt1);
// результат 16#DEAD

END_PROGRAM
```

19.40 WORD_OF_DWORD

Тип модуля: функция	Переменная	Тип	Описание
Входы	in	DWORD	Исходная переменная
	N	BYTE	Номер считываемого слова (0...1)
Выходы	WORD_OF_DWORD	WORD	Выделенное слово

Функция **WORD_OF_DWORD** выделяет из переменной **in** типа **DWORD** слово с номером **N**.

См. также обратную функцию [DWORD_OF_WORD](#).

Пример

```
VAR
    dwVar : DWORD := 16#AABBCCDD; //младшее слово
    num : BYTE := 0;
    resultW : WORD;
END_VAR

resultW := OSCAT_BASIC.WORD_OF_DWORD(in := dwVar, N := num);
// результат 16#CCDD

END_PROGRAM
```

20 Триггеры, элементы хранения и регистры сдвига

20.1 COUNT_BR

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	SET	BOOL	Сигнал установки начального значения.
	in	BYTE	Начальное значение.
	UP	BOOL	Импульс «добавить»
	DN	BOOL	Импульс «отнять».
	STEP	BYTE	Шаг счетчика.
	MX	BYTE	Максимальное значение счетчика.
	RST	BOOL	Сигнал сброса блока.
Выходы	CNT	BYTE	Текущее значение счетчика.
Используемые модули	INC		

Функциональный блок **COUNT_BR** представляет собой счетчик импульсов. По переднему фронту входа **SET** на выход счетчика **CNT** устанавливается начальное значение **in**. По переднему фронту входов **UP/DN** происходит увеличение/уменьшение счетчика на величину **STEP**. Вход **MX** определяет максимальное значение счетчика. При достижении этого значения счетчик переходит через ноль. Иными словами, при значениях **in=0**, **STEP=100** и **MX=255** во время счета в положительном направлении счетчик будет принимать следующие значения: 0, 100, 200, 44, 144, 244, 88 и т.д. По переднему фронту входа **RST** происходит сброс счетчика в **0** в независимости от состояния остальных входов.

Пример

```

VAR
  (* Экземпляр ФБ *)
  Cnt : OSCAT_BASIC.COUNT_BR;

  (* Управление *)
  xTrig : BOOL := FALSE;
  xMode : BOOL := TRUE;  (* TRUE = вверх, FALSE = вниз *)

  (* Для фронта *)
  xTrig_Edge : BOOL;
  xTrig_Old : BOOL;

  (* Результат *)
  Value : BYTE;
END_VAR

(* Инициализация настроек (однократно) *)
Cnt(in := 0 , STEP := 100 , MX := 255);
Cnt(SET := TRUE);  (* Устанавливаем начальное значение 0 *)
Cnt();
Cnt(SET := FALSE);

(* Основной цикл *)
(* Детектор фронта *)
xTrig_Edge := xTrig AND NOT xTrig_Old;
xTrig_Old := xTrig;

(* По фронту xTrig выполняем счет *)
IF xTrig_Edge THEN
  IF xMode THEN
    Cnt(UP := TRUE);  (* Увеличение *)
  ELSE
    Cnt(DN := TRUE);  (* Уменьшение *)
  END_IF;

  Cnt();  (* Выполняем ФБ *)

  (* Сбрасываем управляющие сигналы *)
  Cnt(UP := FALSE, DN := FALSE);
END_IF;

(* Сохраняем текущее значение *)
Value := Cnt.CNT;

END_PROGRAM

```

20.2 COUNT_DR

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	SET	BOOL	Сигнал установки начального значения.
	in	DWORD	Начальное значение.
	UP	BOOL	Импульс «добавить»
	DN	BOOL	Импульс «отнять».
	STEP	DWORD	Шаг счетчика.
	MX	DWORD	Максимальное значение счетчика.
	RST	BOOL	Сигнал сброса блока.
Выходы	CNT	DWORD	Текущее значение счетчика.

Функциональный блок **COUNT_DR** представляет собой счетчик импульсов. Принцип работы полностью соответствует ФБ **COUNT_BR**, единственным отличием является тип используемых переменных – **DWORD**.

20.3 FF_D2E

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	D0	BOOL	Контролируемое значение 0
	D1	BOOL	Контролируемое значение 1
	CLK	BOOL	Сигнал сохранения значений
	RST	BOOL	Сигнал сброса блока
Выходы	Q0	BOOL	Сохраненное значение 0
	Q1	BOOL	Сохраненное значение 1

Функциональный блок **FF_D2E** представляет собой двухбитный D-триггер. По переднему фронту входа **CLK** значения входов **D0** и **D1** присваиваются выходам **Q0** и **Q1**. По переднему фронту входа **RST** значения выходов сбрасываются в **FALSE** в независимости от состояния остальных входов.

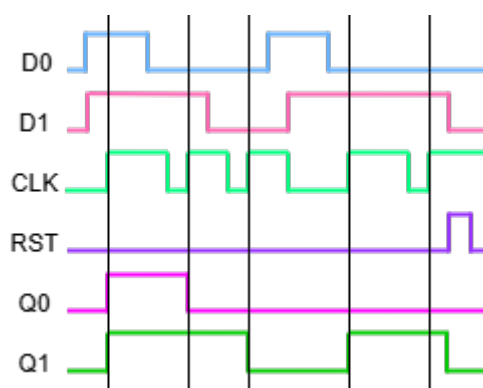


Рисунок 20.1

20.4 FF_D4E

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	D0	BOOL	Контролируемое значение 0
	D1	BOOL	Контролируемое значение 1
	D2	BOOL	Контролируемое значение 2
	D3	BOOL	Контролируемое значение 3
	CLK	BOOL	Сигнал сохранения значений
	RST	BOOL	Сигнал сброса блока
Выходы	Q0	BOOL	Сохраненное значение 0
	Q1	BOOL	Сохраненное значение 1
	Q2	BOOL	Сохраненное значение 2
	Q3	BOOL	Сохраненное значение 3

Функциональный блок **FF_D4E** представляет собой четырехбитный D-триггер. По переднему фронту входа **CLK** значения входов **D0**, **D1**, **D2** и **D3** присваиваются выходам **Q0**, **Q1**, **Q2** и **Q3** соответственно. По переднему фронту входа **RST** значения выходов сбрасываются в **FALSE** в независимости от состояния остальных входов. См. также описание ФБ [FF_D2E](#).

20.5 FF_DRE

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	SET	BOOL	Сигнал активации выхода
	D	BOOL	Контролируемое значение
	CLK	BOOL	Управляющий сигнал
	RST	BOOL	Сигнал сброса блока
Выходы	Q	BOOL	Сохраненное значение

Функциональный блок **FF_DRE** представляет собой D-триггер с асинхронным управлением выходом. По переднему фронту входа **CLK** значения входа **D** присваивается выходу **Q**. По переднему фронту входа **SET** выход принимает значение **TRUE**. По переднему фронту входа **RST** выход сбрасывается в **FALSE** в независимости от состояния остальных входов. Вход **RST** имеет приоритет над входом **SET**, и при их одновременном срабатывании сигнал на входе **SET** не обрабатывается. Вход **SET** имеет приоритет над входом **CLK**.

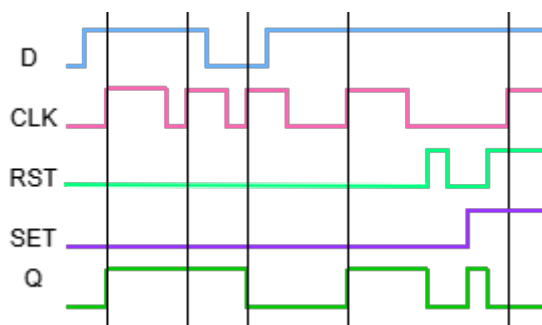


Рисунок 20.2

20.6 FF_JKE

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	SET	BOOL	Сигнал активации выхода
	J	BOOL	Вход «Jump»
	CLK	BOOL	Управляющий сигнал
	K	BOOL	Вход «Kill»
	RST	BOOL	Сигнал сброса блока
Выходы	Q	BOOL	Выход триггера

Функциональный блок **FF_JKE** представляет собой JK-триггер. По переднему фронту входа **SET** выход принимает значение **TRUE**. По переднему фронту входа **RST** выход сбрасывается в **FALSE** в независимости от состояния остальных входов. По переднему фронту входа **CLK** выход **Q** меняет свое значение в зависимости от состояний входов **J** и **K**:

- если J=TRUE, то Q:=TRUE;
- если K=TRUE, то Q:=FALSE;
- если J=TRUE и K=TRUE, то Q:=NOT(Q) [значение выхода инвертируется].

Вход **SET** имеет приоритет над входом **CLK**.

Рисунок 20.3 – СХЕМА НЕ СООТВЕТСТВУЕТ ФБ

20.7 FF_RSE

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	CS	BOOL	Вход «Set»
	CR	BOOL	Вход «Reset»
	RST	BOOL	Сигнал сброса блока
Выходы	Q	BOOL	Выход триггера

Функциональный блок **FF_RSE** представляет собой RS-триггер с асинхронным сбросом. По переднему фронту входа **CS** выход **Q** принимает значение **TRUE**. По переднему фронту входа **CR** выход **Q** принимает значение **FALSE**. Вход **CR** имеет приоритет над входом **CS**, и при их одновременном срабатывании сигнал на входе **CS** не обрабатывается. По переднему фронту входа **RST** выход сбрасывается в **FALSE** в независимости от состояния остальных входов.

20.8 LTCH

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	D	BOOL	Контролируемое значение
	L	BOOL	Режим «сохранение значений»
	RST	BOOL	Сигнал сброса блока
Выходы	Q	BOOL	Сохраненное значение

Функциональный блок **LTCH** представляет собой одноступенчатый триггер («защелку»). Пока вход **L** имеет значение **TRUE**, значение на выходе **Q** равно значению входа **D**. По заднему фронту входа **L** выход **Q** фиксируется в своем текущем состоянии. По переднему фронту входа **RST** выход сбрасывается в **FALSE** в независимости от состояния остальных входов.

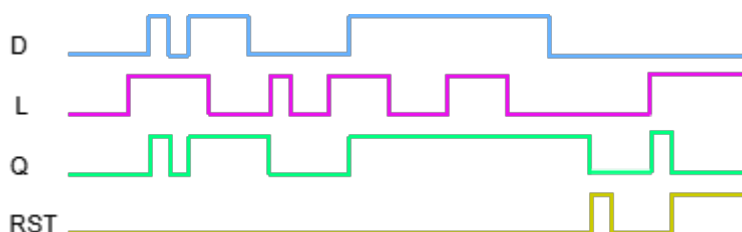


Рисунок 20.4

20.9 LTCH_4

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	D0...D3	BOOL	Контролируемые значения
	L	BOOL	Режим «сохранение значений»
	RST	BOOL	Сигнал сброса блока
Выходы	Q	BOOL	Сохраненные значения

Функциональный блок **LTCH_4** представляет собой четырехбитный одноступенчатый триггер («защелку»). Пока вход **L** имеет значение **TRUE**, значения на выходах **Q0... Q3** равны значениям входов **D0... D3**. По заднему фронту входа **L** выходы фиксируются в своих текущих состояниях. По переднему фронту входа **RST** выходы сбрасываются в **FALSE** в независимости от состояния остальных входов.

20.10 SELECT_8

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	E	BOOL	Вход управления блоком
	SET	BOOL	Сигнал установки выхода номер IN
	IN	BYTE	Номер устанавливаемого выхода
	UP	BOOL	Включение следующего выхода
	DN	BOOL	Включение предыдущего выхода
	RST	BOOL	Сигнал сброса блока
Выходы	Q0...Q7	BOOL	Выходы блока

Тип модуля: ФБ	Переменная	Тип	Описание
	STATE	INT	Номер активного выхода (0...7)
Используемые модули	INC		

Функциональный блок **SELECT_8** представляет собой восьмибитный селектор - в каждый момент времени только один из выходов **Q0...Q7** является активным. Пока вход **E** имеет значение **TRUE** блок находится в работе. По переднему фронту на входе **SET** выход с номером **IN** принимает значение **TRUE**, а все остальные выходы сбрасываются в **FALSE**. По переднему фронту на входе **UP** активным становится следующий выход, по переднему фронту на входе **DN** – предыдущий. Соответственно, если активным является **Q7**, то после импульса на входе **UP** активным становится **Q0**; если же активным является **Q0**, то после импульса на входе **DN** активным становится **Q7**. Выход **STATE** содержит номер активного выхода. По переднему фронту входа **RST** выход сбрасывается в **FALSE** в независимости от состояния остальных входов.

20.11 SHR_4E

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	SET	BOOL	Сигнал активации выходов
	D0	BOOL	Вход данных
	CLK	BOOL	Управляющий сигнал
	RST	BOOL	Сигнал сброса блока
Выходы	Q0...Q3	BOOL	Выходы блока

Функциональный блок **SHR_4E** представляет собой четырехбитный регистр сдвига. По переднему фронту входа **SET** выходы **Q0...Q3** принимают значение **TRUE**. По переднему фронту входа **RST** выходы **Q0...Q3** сбрасываются в **FALSE** в независимости от состояния остальных входов. По переднему фронту входа **CLK** значения на выходах **Q0...Q3** сдвигаются по следующему алгоритму:

- Q2 —> Q3
- Q1 —> Q2
- Q0 —> Q1
- D0 —> Q0

Вход **SET** имеет приоритет над входом **CLK**.

Обратите внимание, что сдвиг не является циклическим.

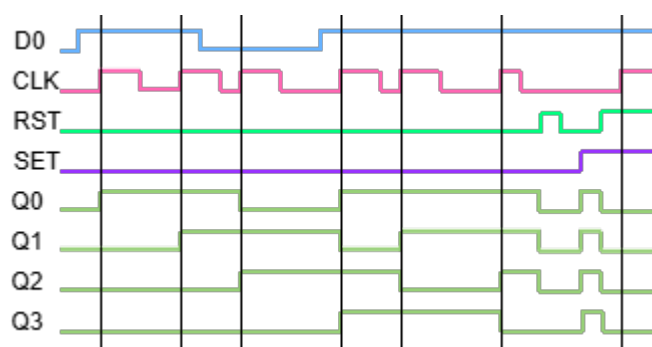


Рисунок 20.5 – Трассировка работы ФБ SHR_4E

20.12 SHR_4UDE

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	SET	BOOL	Сигнал активации выходов
	D0	BOOL	Вход данных (при счете вверх)
	D3	BOOL	Вход данных (при счете вниз)
	CLK	BOOL	Управляющий сигнал
	DN	BOOL	Направление счета (TRUE – вниз, FALSE – вверх)
	RST	BOOL	Сигнал сброса блока
Выходы	Q0...Q3	BOOL	Выходы блока

Функциональный блок **SHR_4UDE** представляет собой четырехбитный регистр сдвига с выбором направления сдвига. По переднему фронту входа **SET** выходы **Q0... Q3** принимают значение **TRUE**. По переднему фронту входа **RST** выходы **Q0... Q3** сбрасываются в **FALSE** в независимости от состояния остальных входов. По переднему фронту входа **CLK** значения на выходах **Q0...Q3** сдвигаются по алгоритму, определяемому состоянием входа **DN**:

- Ø DN=FALSE (сдвиг вверх):
 - Q2 —> Q3;
 - Q1 —> Q2;
 - Q0 —> Q1;
 - D0 —> Q0.
- Ø DN=TRUE (сдвиг вниз):
 - Q1 —> Q0;
 - Q2 —> Q1;
 - Q3 —> Q2;
 - D3 —> Q3.

Вход **SET** имеет приоритет над входом **CLK**.

Обратите внимание, что сдвиг не является циклическим.

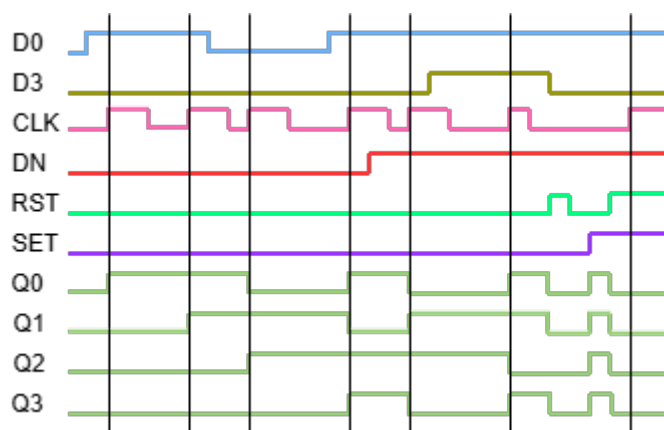


Рисунок 20.6 – Трассировка работы ФБ SHR_4UDE

20.13 SHR_8PLE

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	Din	BOOL	Значение для записи
	Dload	BYTE	Вход данных
	CLK	BOOL	Управляющий сигнал
	UP	BOOL	Направление счета (TRUE – вверх, FALSE – вниз)

Тип модуля: ФБ	Переменная	Тип	Описание
	load	BOOL	Режим загрузки данных в блок
	RST	BOOL	Сигнал сброса блока
Выходы	DOut	BOOL	Выход блока

Функциональный блок **SHR_8PLE** представляет собой восьмибитный регистр сдвига с выбором направления сдвига. Вход **UP** определяет направление сдвига: **TRUE** – вверх (при этом выход **DOut** соответствует старшему биту регистра), **FALSE** – вниз (при этом выход **DOut** соответствует младшему биту регистра). По переднему фронту на входе **CLK** происходит сдвиг регистра в соответствующем направлении, при этом младший (при **UP=TRUE**) или старший (при **UP=FALSE**) бит принимает значение **Din**. Если вход **load** имеет значение **TRUE**, то по переднему фронту на входе **CLK** происходит запись значения **Dload** в регистр.

Обратите внимание, что сдвиг не является циклическим.

20.14 SHR_8UDE

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	SET	BOOL	Сигнал активации выходов
	D0	BOOL	Вход данных (при счете вверх)
	D7	BOOL	Вход данных (при счете вниз)
	CLK	BOOL	Управляющий сигнал
	DN	BOOL	Направление счета (TRUE – вниз, FALSE – вверх)
	RST	BOOL	Сигнал сброса блока
Выходы	Q0...Q7	BOOL	Выходы блока

Функциональный блок **SHR_8UDE** представляет собой восьмибитный регистр сдвига с выбором направления сдвига. Принцип работы блока полностью соответствует ФБ [SHR_4UDE](#).

20.15 STORE_8

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	SET	BOOL	Сигнал активации выходов
	D0...D7	BOOL	Вход данных (при счете вверх)
	Clr	BOOL	Сигнал сброса выхода с наименьшим номером
	RST	BOOL	Сигнал сброса блока
Выходы	Q0...Q7	BOOL	Выходы блока

Функциональный блок **STORE_8** представляет собой восьмибитный элемент хранения. По переднему фронту входов **D0...D7** выходы **Q0...Q7** принимают значение **TRUE**. Пока вход **SET** имеет значение **TRUE**, выходы **Q0...Q7** принимают значение **TRUE** в независимости от состояния входов. По переднему фронту входа **RST** выходы **Q0...Q7** сбрасываются в **FALSE**. По переднему фронту входа **Clr** происходит сброс в **FALSE** активного выхода с наименьшим номером (только в том случае, если соответствующий вход имеет значение **FALSE**).

20.16 TOGGLE

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	CLK	BOOL	Управляющий сигнал
	RST	BOOL	Сигнал сброса блока
Выходы	Q	BOOL	Выход блока

Функциональный блок **TOGGLE** представляет собой переключатель со сбросом. По переднему фронту входа **CLK** значение выхода **Q** инвертируется. По переднему фронту входа **RST** выход **Q** принимает значение **FALSE** в независимости от состояния входа **CLK**.

21 Генераторы сигналов

21.1 _RMP_B

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	DIR	BOOL	Направление изменения выхода
	E	BOOL	Сигнал управления блоком
	TR	TIME	Полное время генерации (от 0 до 255)
Выходы	RMP	BYTE	Выход блока
Используемые модули	T_PLC_MS, FRMP_B		

Функциональный блок **_RMP_B** представляет собой генератор линейной функции. Пока вход **E** имеет значение **TRUE** блок находится в работе и значение переменной, связанной со входом-выходом **RMP**, линейно изменяется от текущего значения до максимального (при **DIR=TRUE**) или минимального (при **DIR=FALSE**). Минимальное и максимальное значения определяются диапазоном типа **BYTE** и соответственно составляют **0** и **255**. Скорость изменения выходного сигнала определяется отношением $(255/TR)$. Для повторного запуска блока после окончания его работы достаточно изменить значение **RMP** (например, обнулив его). Если вход **E** принимает значение **FALSE**, то блок прекращает работу, при этом **RMP** сохраняет свое последнее значение.

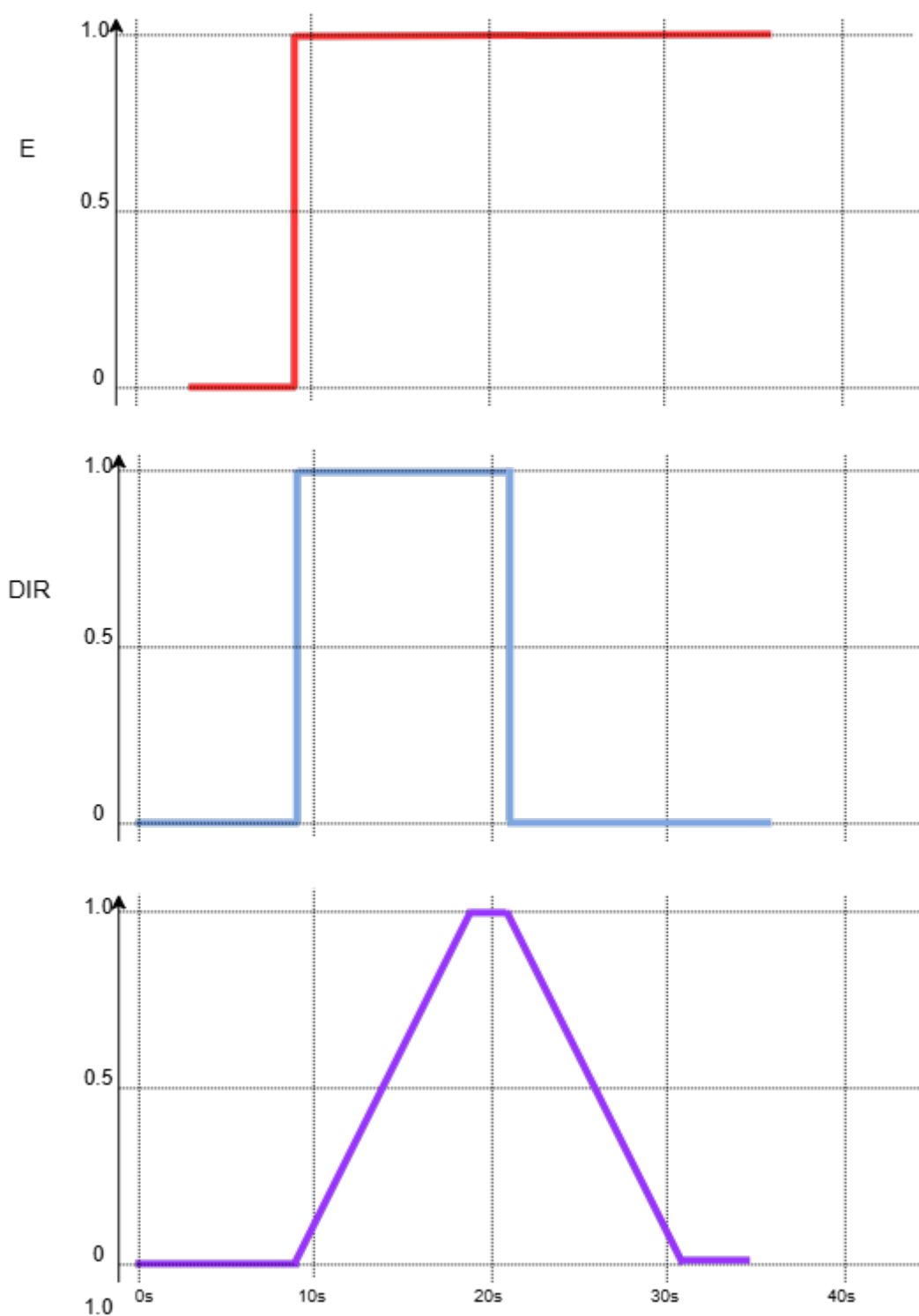


Рисунок 21.1 – Трассировка работы ФБ _RMP_V (TR=T#10s)

21.2 _RMP_NEXT

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	E	BOOL	Сигнал управления блоком
	IN	BYTE	Контролируемый сигнал
	TR	TIME	Время нарастания сигнала
	TF	TIME	Время спада сигнала
	TL	TIME	Время задержки

Тип модуля: ФБ	Переменная	Тип	Описание
Выходы	DIR	BOOL	Флаг «направление изменения»
	UP	BOOL	Импульс «начало нарастания»
	DN	BOOL	Импульс «начало спада»
Входы-выходы	OUT	BYTE	Выход блока
Используемые модули	_RMP_B , TREND_DW		

Функциональный блок **_RMP_NEXT** представляет собой генератор линейной функции. Пока вход **E** имеет значение **TRUE** блок находится в работе и значение переменной, связанной со входом-выходом **OUT**, линейно изменяется от текущего значения до ближайшего значения относительно входа **IN**. Время достижения этого значения определяется переменными **TR** (в случае нарастания сигнала) и **TF** (в случае спада сигнала). Время **TL** определяет задержку изменения выходного сигнала относительно изменения входного. Выход **DIR** определяет текущее изменение выхода (**TRUE** – нарастание, **FALSE** – спад); если изменений не происходит, то **DIR** сохраняет свое последнее значение. Выход **UP** принимает значение **TRUE** на время нарастания выходного сигнала, выход **DN** – на время его спада. В остальные периоды времени выходы имеют значение **FALSE**. Если вход **E** принимает значение **FALSE**, то блок прекращает работу, при этом **OUT** сохраняет свое последнее значение.

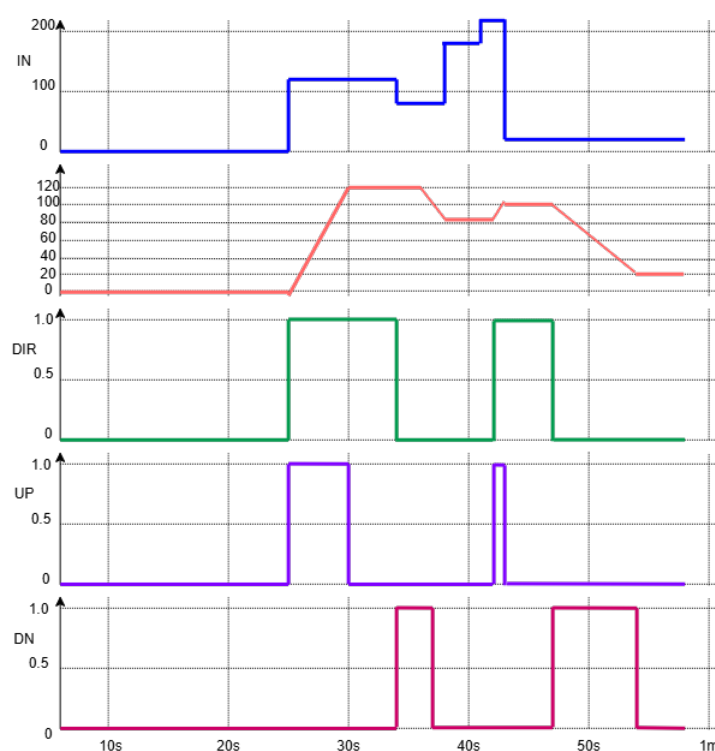


Рисунок 21.2 – Трассировка работы ФБ **_RMP_NEXT**(TR=T#10s, TF=T#20s, TL=T#5s)

21.3 **_RMP_W**

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	DIR	BOOL	Направление изменения выхода
	E	BOOL	Сигнал управления блоком
	TR	TIME	Полное время генерации (от 0 до 65535)
Входы-выходы	RMP	WORD	Выход блока
Используемые модули	T_PLC_MS		

Функциональный блок **_RMP_W** представляет собой генератор линейной функции. Принцип работы блока полностью соответствует блоку **_RMP_B**, единственным отличие является тип выхода генератора – у данного блока он имеет тип **WORD**.

21.4 GEN_PULSE

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	ENQ	BOOL	Сигнал управления блоком
	PTH	TIME	Время сигнала верхнего уровня
	PTL	TIME	Время сигнала нижнего уровня
Выходы	Q	BOOL	Выход генератора
Используемые модули	T_PLC_MS		

Функциональный блок **GEN_PULSE** представляет собой генератор прямоугольных импульсов. Когда вход **ENQ** принимает значение **TRUE**, блок запускается в работу, и на выходе **Q** начинают генерироваться прямоугольные импульсы с длительностью **PTL** (нижний уровень) и **PTH** (верхний уровень) соответственно. Блок начинает работу с импульса нижнего уровня. Если вход **ENQ** принимает значение **FALSE**, то блок прекращает работу; выход **Q** при этом принимает значение **FALSE**. Импульс с длительностью **0** соответствует импульсу длиной в цикл ПЛК.

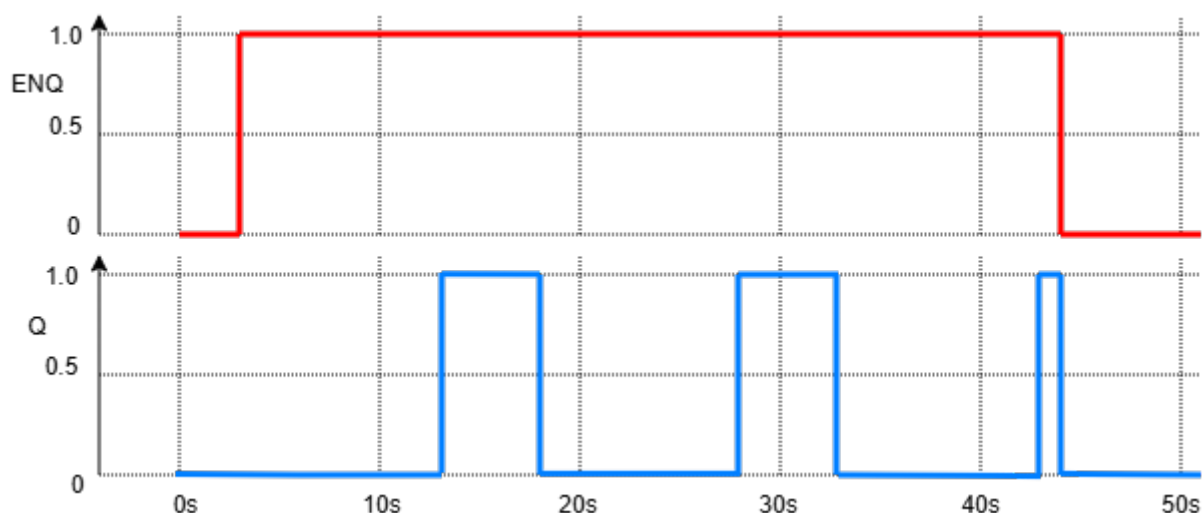


Рисунок 21.3 – Трассировка работы ФБ GEN_PULSE (PTH=T#5s, PTL=T#10s)

21.5 GEN_PW2

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	ENQ	BOOL	Сигнал управления блоком
	TH1	TIME	Время сигнала верхнего уровня (набор 1)
	TL1	TIME	Время сигнала нижнего уровня (набор 1)
	TH2	TIME	Время сигнала верхнего уровня (набор 2)
	TL2	TIME	Время сигнала нижнего уровня (набор 2)
	TS	BOOL	Сигнал переключения наборов
Выходы	Q	BOOL	Выход генератора
	TH	TIME	Текущее время импульса (верхний уровень)

Тип модуля: ФБ	Переменная	Тип	Описание
	TL	TIME	Текущее время импульса (нижний уровень)
Используемые модули	T_PLC_MS		

Функциональный блок **GEN_PW2** представляет собой генератор прямоугольных импульсов с двумя наборами параметров. Когда вход **ENQ** принимает значение **TRUE**, блок запускается в работу, и на выходе **Q** начинают генерироваться прямоугольные импульсы. Если вход **TS** имеет значение **FALSE**, то длительность генерируемых импульсов определяется значениями входов **TH1** (верхний уровень) и **TL1** (нижний уровень). Если вход **TS** имеет значение **TRUE**, то длительность генерируемых импульсов определяется значениями входов **TH2** (верхний уровень) и **TL2** (нижний уровень). Выходы **TH** и **TL** содержат время, прошедшее с начала генерации текущего импульса (верхнего и нижнего уровня соответственно). Блок начинает работу с импульса нижнего уровня. Если вход **ENQ** принимает значение **FALSE**, то блок прекращает работу; выход **Q** при этом принимает значение **FALSE**. Импульс с длительностью **0** соответствует импульсу длиной в цикл ПЛК.

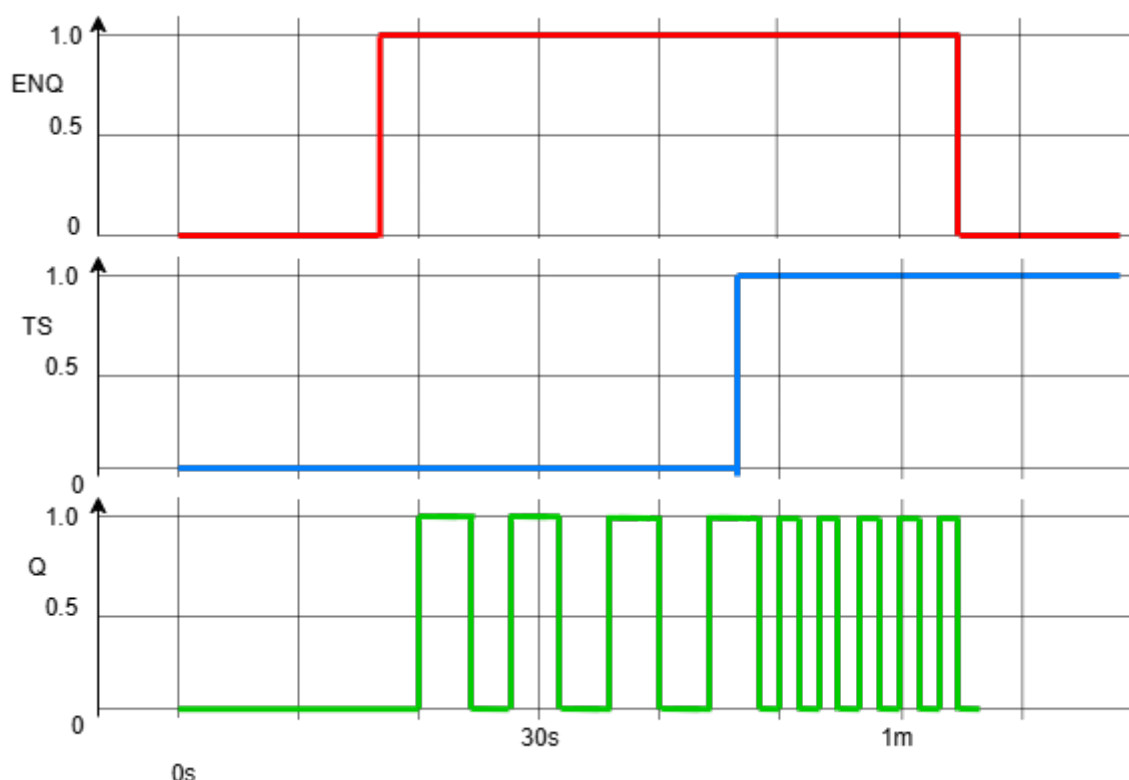


Рисунок 21.4 – Трассировка работы ФБ GEN_PW2 (TH1=TL1=T#4s, TH2=TL2=T#2s)

21.6 GEN_RDM

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	PT	BOOL	Период генерации значений
	AM	REAL	Амплитуда
	OS	REAL	Смещение
Выходы	Q	BOOL	Флаг «новое значение»
	Out	REAL	Выход генератора
Используемые модули	T_PLC_MS, RDM		

Функциональный блок **GEN_RDM** представляет собой генератор псевдослучайных значений. На выходе **Out** с интервалом **PT** генерируется псевдослучайное значение с амплитудой **AM** и смещением **OS**. При появлении нового значения на выходе **Q** генерируется единичный импульс.

Связь значения на выходе **Out** с амплитудой и смещением определяется формулой:

$$OUT = AM * (RDM - 0.5) + OS$$

, где

RDM – псевдослучайное число в диапазоне [0,1].

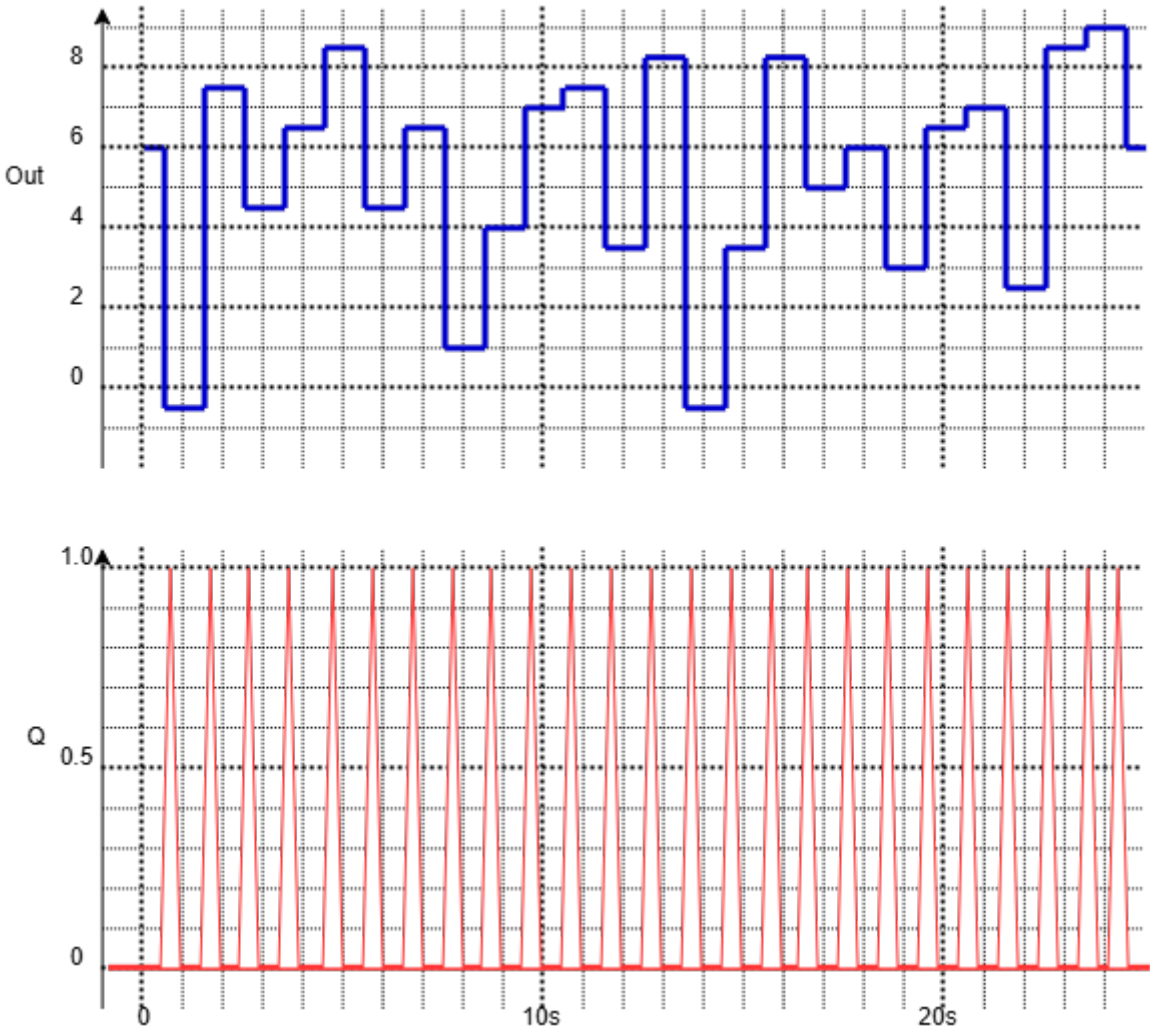


Рисунок 21.5 – Трассировка работы ФБ GEN_RDM (PT=T#1s, AM=10.0, OS=4.0)

21.7 GEN_RDT

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	Enable	BOOL	Сигнал управления блоком
	Min_Time_ms	TIME	Минимальная пауза между импульсами
	Max_Time_ms	TIME	Максимальная пауза между импульсами
	TP_Q	TIME	Длительность импульса
Выходы	xQ	BOOL	Выход генератора
Используемые модули	RDM		

Функциональный блок **GEN_RDT** представляет собой генератор импульсов заданной длины с псевдослучайным периодом. Когда вход **Enable** принимает значение **TRUE**, блок запускается в работу, и на выходе **xQ** начинают генерироваться импульсы длительностью **TP_Q**, причем пауза между импульсами является псевдослучайным числом из диапазона **Min_Time_ms...Max_Time_ms**. Импульс с длительностью **0** соответствует импульсу длиной в цикл ПЛК. Если вход **Enable** принимает значение **FALSE**, то блок прекращает работу; выход **xQ** при этом принимает значение **FALSE**.

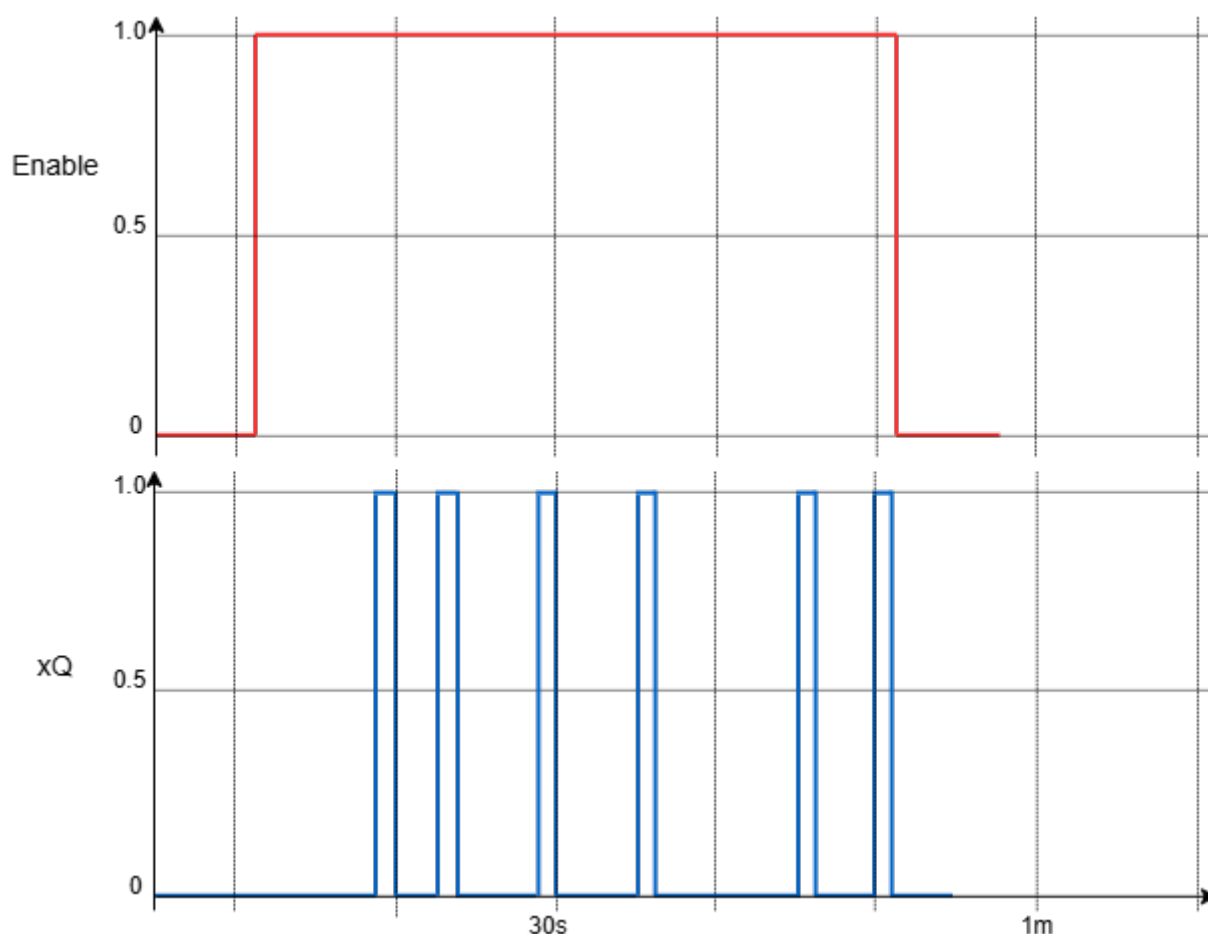


Рисунок 21.6 – Трассировка работы ФБ GEN_RDT (Min_Time_ms=T#5s, Max_Time_Ms=T#10s, TP_Q=T#1s)

21.8 GEN_RMP

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	PT	BOOL	Период генерации значений
	AM	REAL	Амплитуда
	OS	REAL	Смещение сигнала относительно нуля
	DL	REAL	Задержка включения в долях от PT (0...1)
Выходы	Q	BOOL	Флаг «новое значение»
	Out	REAL	Выход генератора
Используемые модули	T_PLC_MS, MODR, MULTIME		

Функциональный блок **GEN_RMP** представляет собой генератор пилообразного сигнала. На выходе блока **Out** циклически с периодом **PT** генерируется выходной пилообразный сигнал с амплитудой **AM**, смещением относительно нуля **OS** и задержкой включения **DL**. Задержка используется для генерации нескольких сигналов (с помощью нескольких экземпляров ФБ), смещенных относительно друг друга, и задается в долях от периода **PT**:

- DL=0 – задержки нет;
- DL=0.5 – задержка составляет 50% от времени **PT**;
- DL=1 – задержка равна времени **PT**.

На выходе **Q** генерируется единичный импульс в начале каждого цикла генерации.

На рисунке ниже приведена трассировка работы двух экземпляров блока для значений входных переменных:

	fbGEN_RMP_1	fbGEN_RMP_2
PT	T#10s	T#10s
AM	10	8
OS	4	2
DL	0.3	0.6

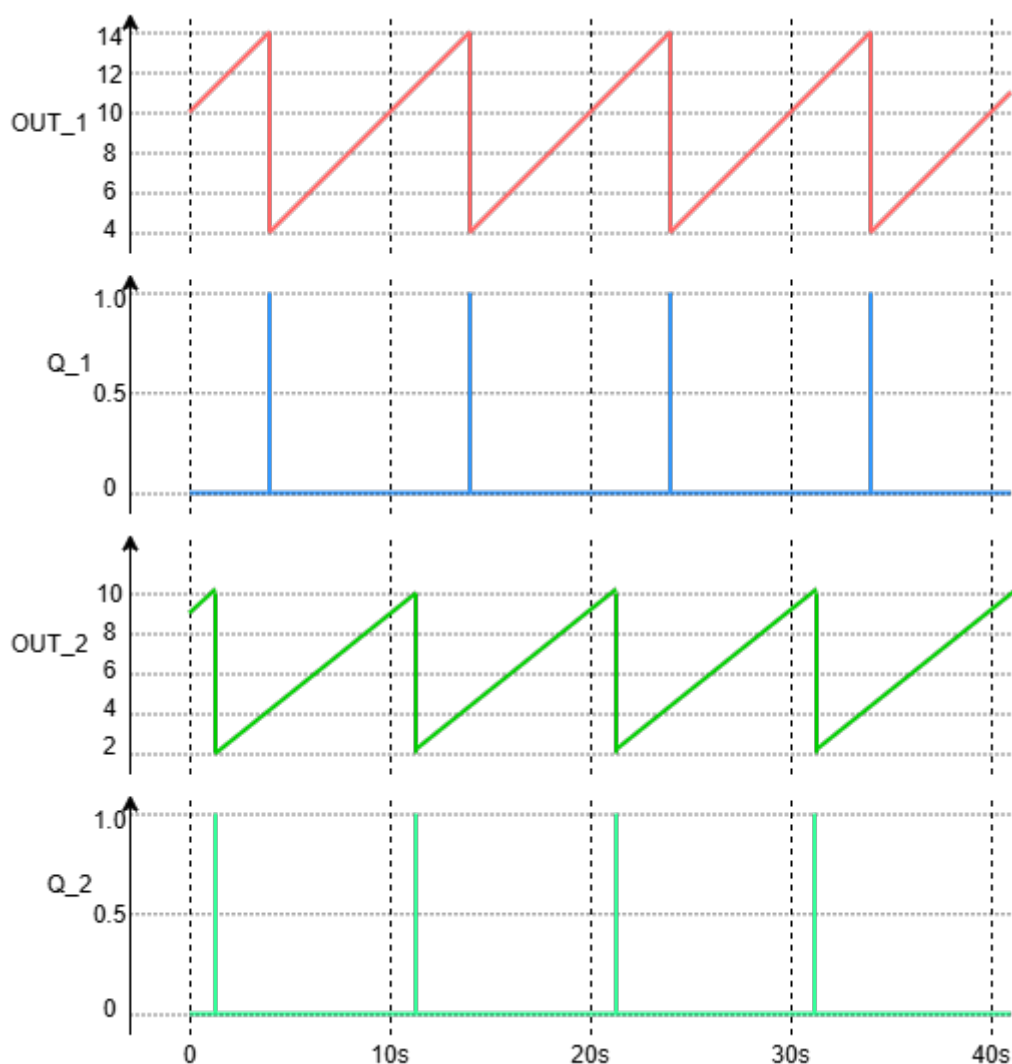


Рисунок 21.7 – Трассировка работы экземпляров ФБ GEN_RMP

21.9 GEN_SIN

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	PT	TIME	Период колебаний
	AM	REAL	Амплитуда
	OS	REAL	Смещение сигнала относительно нуля
	DL	REAL	Задержка включения в долях от PT (0...1)
Выходы	Q	BOOL	Флаг «знак полупериода»
	Out	REAL	Выход генератора
Используемые модули	T_PLC_MS, MODR, MULTIME, SIGN_R		

Функциональный блок **GEN_SIN** представляет собой генератор синусоидального сигнала. На выходе блока **Out** циклически с периодом **PT** генерируется выходной синусоидальный сигнал с амплитудой **AM**, смещением относительно нуля **OS** и задержкой включения **DL**. Задержка используется для генерации

нескольких сигналов (с помощью нескольких экземпляров ФБ), смещенных относительно друг друга, и задается в долях от периода **PT**:

- $DL=0$ – задержки нет;
- $DL=0.5$ – задержка составляет 50% от времени **PT**;
- $DL=1$ – задержка равна времени **PT**.

Выход **Q** имеет значение **TRUE** во время положительного полупериода синусоиды и **FALSE** – во время отрицательного.

На рисунке ниже приведена трассировка работы двух экземпляров блока для значений входных переменных:

	fbGEN_SIN_1	fbGEN_SIN_2
PT	T#10s	T#10s
AM	16	16
OS	0	8
DL	0.0	0.5

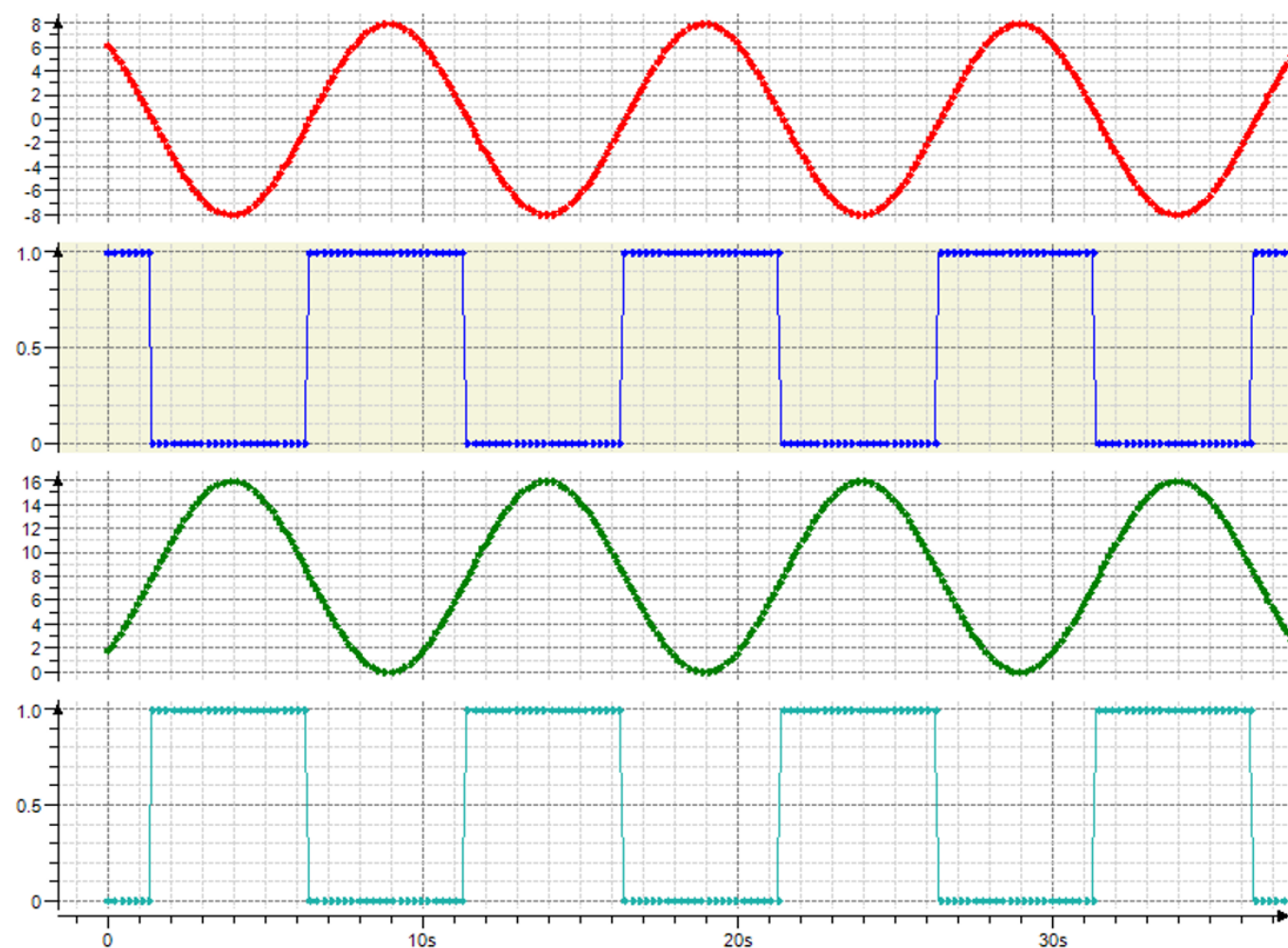


Рисунок 21.8 – Трассировка работы экземпляров ФБ GEN_SIN

21.10 GEN_SQR

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	PT	BOOL	Период
	AM	REAL	Амплитуда
	OS	REAL	Смещение сигнала относительно нуля
	DC	REAL	Время сигнала верхнего уровня в долях от PT (0...1).

Тип модуля: ФБ	Переменная	Тип	Описание
	DL	REAL	Задержка включения в долях от PT (0...1).
Выходы	Q	BOOL	Флаг «знак полупериода»
	Out	REAL	Выход генератора
Используемые модули	T_PLC_MS, MODR, MULTIME		

Функциональный блок **GEN_SQR** представляет собой генератор прямоугольного сигнала. На выходе блока **Out** циклически с периодом **PT** генерируется выходной прямоугольный сигнал с амплитудой **AM**, смещением относительно нуля **OS** и задержкой включения **DL**. Задержка используется для генерации нескольких сигналов (с помощью нескольких экземпляров ФБ), смещенных относительно друг друга, и задается в долях от периода **PT**:

- DL=0 – задержки нет;
- DL=0.5 – задержка составляет 50% от времени **PT**;
- DL=1 – задержка равна времени **PT**.

Вход **DC** определяет время сигнала верхнего уровня в долях от периода **PT**. Например, если **PT=T#1s** и **DC=0.5**, то на выходе блока в течение 500 мс будет сигнал верхнего уровня, затем в течение 500 мс – нижнего, после чего цикл повторится.

Выход **Q** имеет значение **TRUE** во время сигнала верхнего уровня на выходе генератора и **FALSE** – во время сигнала нижнего уровня.

На рисунке ниже приведена трассировка работы двух экземпляров блока для значений входных переменных:

	fbGEN_SQR_1	fbGEN_SQR_2
PT	T#1s	T#1s
AM	10	10
OS	0	5
DC	0.5	0.25
DL	0.0	0.0

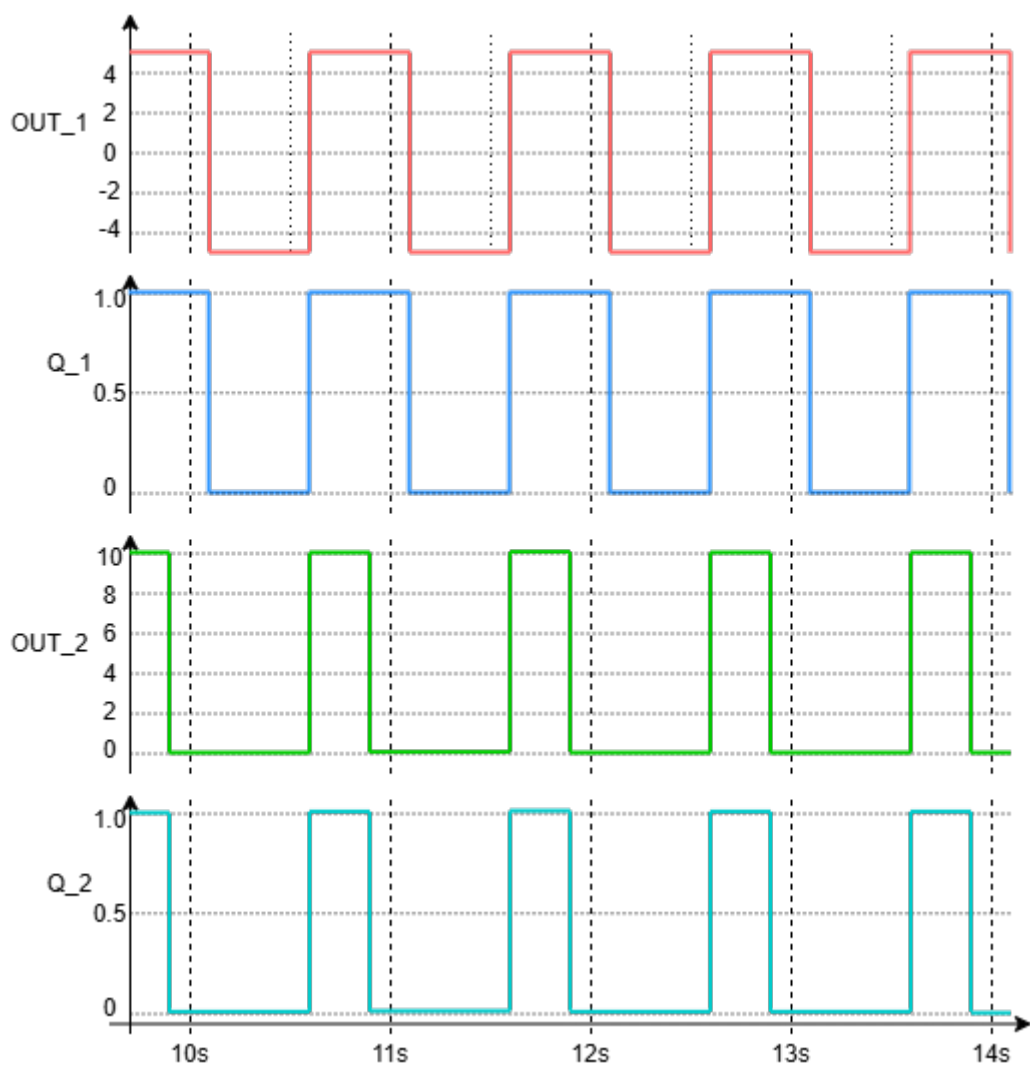


Рисунок 21.9 – Трассировка работы экземпляров ФБ GEN_SQR

21.11 PWM_DC

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	F	TIME	Частота генератора (в Гц)
	DC	REAL	Время сигнала верхнего уровня в долях от 1/F (0...1)
Выходы	Q	BOOL	Выход генератора
Используемые модули	CLK_PRG, TP_X		

Функциональный блок **PWM_DC** представляет собой генератор импульсов с задаваемой частотой **F**. Вход **DC** определяет время сигнала верхнего уровня в долях от периода **PT**. Например, если **F=20 Гц** и **DC=0.1**, то период генератора = $1/F = 50$ мс, причем время сигнала верхнего уровня = $0.1 \cdot 50 = 5$ мс, а время сигнала нижнего уровня – 45 мс.

Сгенерированный сигнал подается на выход **Q**.

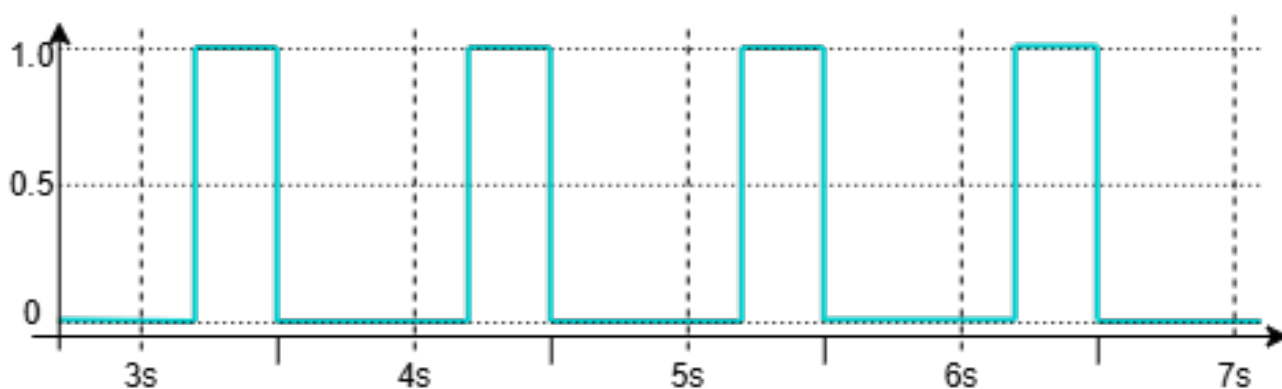


Рисунок 21.10 – Трассировка работы ФБ PWM_DC (F=1, DC=0.25)

21.12 PWM_PW

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	F	TIME	Частота генератора (в Гц)
	PW	REAL	Время сигнала верхнего уровня
Выходы	Q	BOOL	Выход генератора
Используемые модули	CLK_PRG, TP_X		

Функциональный блок **PWM_PW** представляет собой генератор импульсов с задаваемой частотой **F** и временем сигнала верхнего уровня **PW**. Сгенерированный сигнал подается на выход **Q**.

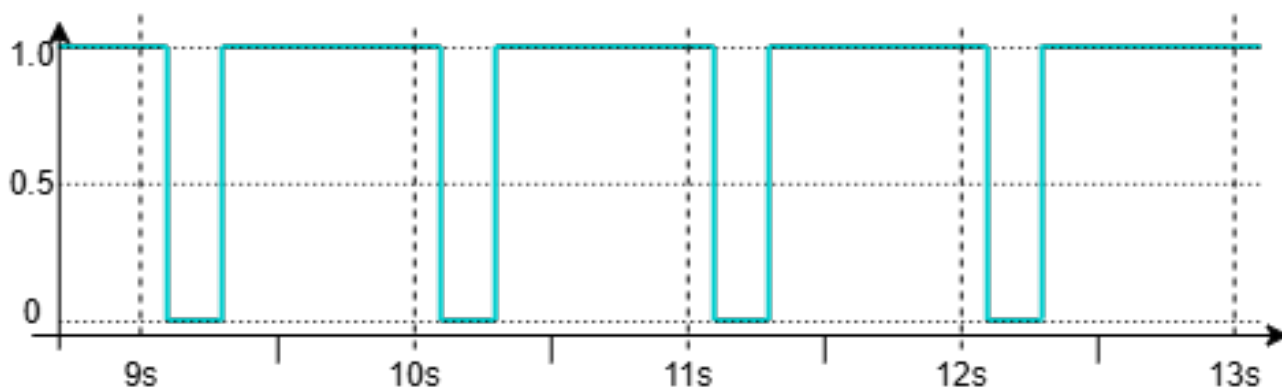


Рисунок 21.11 – Трассировка работы ФБ PWM_PW (F=1, PW=T#800ms)

21.13 RMP_B

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	SET	BOOL	Сигнал установки на выход макс. значения
	PT	TIME	Время генерации
	E	BOOL	Сигнал управления блоком
	UP	BOOL	Направление изменения выхода
	RST	BOOL	Сигнал установки на выход мин. значения
Выходы	OUT	BYTE	Выход блока
	BUSY	BOOL	Флаг «генератор в работе»
	HIGH	BOOL	Флаг «достигнуто макс. значение»

Тип модуля: ФБ	Переменная	Тип	Описание
	LOW	BOOL	Флаг «достигнуто мин. значение»
Используемые модули	_RMP_B		

Функциональный блок **RMP_B** представляет собой генератор линейной функции. Пока вход **E** имеет значение **TRUE** блок находится в работе и значение выхода **OUT** линейно изменяется от текущего значения до максимального (при **UP=TRUE**) или минимального (при **UP=FALSE**). Минимальное и максимальное значения определяются диапазоном типа **BYTE** и соответственно составляют **0** и **255**. Скорость изменения выходного сигнала определяется отношением $(255/PT)$. Сигнал по переднему фронту на входе **SET** устанавливает на выход максимальное значение (**255**), сигнал по переднему фронту на входе **RST** – минимальное (**0**). Выход **BUSY** имеет значение **TRUE**, пока выход меняет свое значение. Выходы **HIGH** и **LOW** принимают значение **TRUE** при достижении выходом **OUT** максимального и минимального значения соответственно.

Если вход **E** принимает значение **FALSE**, то блок прекращает работу, при этом выход **OUT** сохраняет свое последнее значение.

21.14 RMP_SOFT

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	BOOL	Сигнал управления блоком
	VAL	BYTE	Максимальное значение выхода
Входы-выходы	OUT	BYTE	Выход блока
Параметры	PT_OFF	TIME	Время спада сигнала
	PT_ON	TIME	Время нарастания сигнала
Используемые модули	_RMP_B		

Функциональный блок **RMP_SOFT** представляет собой генератор линейной функции с ограничением выходного сигнала. Пока вход **IN** имеет значение **TRUE**, значение выхода **OUT** линейно изменяется от нулевого значения до значения **VAL** за время **PT_ON**. Пока вход **IN** имеет значение **FALSE**, значение выхода **OUT** линейно изменяется от текущего значения до нулевого за время **PT_OFF**.

Ниже представлена трассировка работы ФБ со значениями:

VAL = 100

PT_ON=T#10s

PT_OFF=T#5s

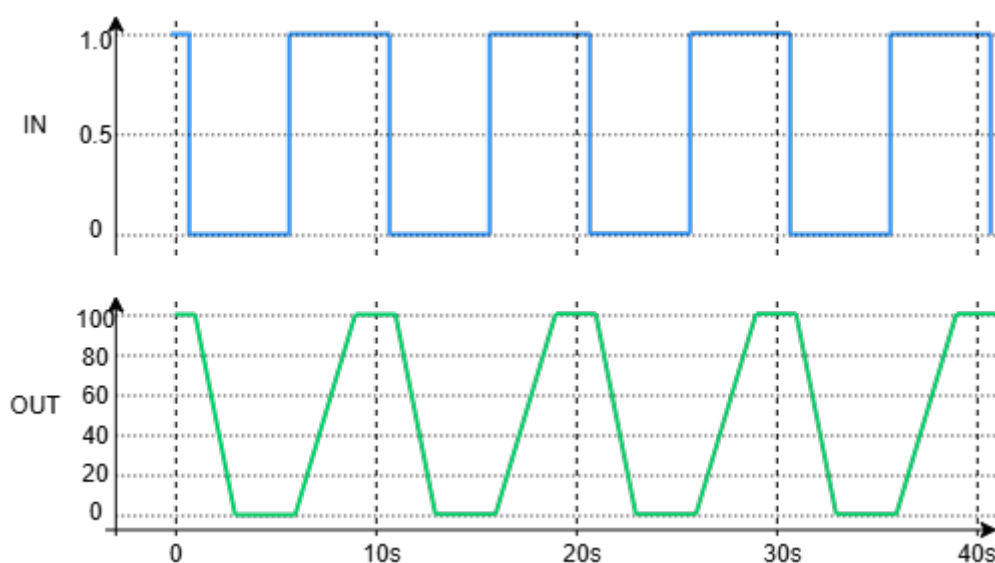


Рисунок 21.12 – Трассировка работы ФБ RMP_SOFT

21.15 RMP_W

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	SET	BOOL	Сигнал установки на выход макс. значения
	PT	TIME	Время генерации
	E	BOOL	Сигнал управления блоком
	UP	BOOL	Направление изменения выхода
	RST	BOOL	Сигнал установки на выход мин. значения
Выходы	OUT	WORD	Выход блока
	BUSY	BOOL	Флаг «генератор в работе»
	HIGH	BOOL	Флаг «достигнуто макс. значение»
	LOW	BOOL	Флаг «достигнуто мин. значение»
Используемые модули	_RMP_W		

Функциональный блок **RMP_W** представляет собой генератор линейной функции. Принцип работы блока полностью соответствует блоку [RMP_B](#), единственным отличием является тип выхода генератора – у данного блока он имеет тип **WORD**.

22 Обработка сигналов

22.1 AIN

Тип модуля: функция	Переменная	Тип	Описание
Входы	in	DWORD	Цифровой сигнал
	Bits	BYTE	Число значащих бит
	sign	BYTE	Номер бита, определяющего знак
	low	REAL	Нижний предел аналогового сигнала
	high	REAL	Верхний предел аналогового сигнала
Выходы	AIN	REAL	Аналоговый сигнал

Функция **AIN** используется для преобразования цифрового сигнала **in** в аналоговый **AIN**. Вход **Bits** определяет число значащих бит цифрового сигнала (максимальное значение – **32**). Вход **sign** определяет номер бита, содержащего знак (его значение **FALSE** соответствует знаку «+»); если такого бита нет, то на вход **sign** подается значение **16# FF**. Входы **low** и **high** определяют минимальное и максимальное значение аналогового сигнала. Нумерация бит ведется с нуля.

Ниже приведено несколько примеров настройки входов для разных сигналов:

- Аналоговый сигнал в диапазоне 0–10 представлен цифровым 12-битным сигналом: Bits=12, sign=16#255, low=0.0, high=10.0;
- Аналоговый сигнал в диапазоне -10–10 представлен цифровым 14-битным сигналом (биты 0-13) с битом знака (бит 14): Bits=14, sign=14, low=-10.0, high=10.0;
- Аналоговый сигнал в диапазоне -10–10 представлен цифровым 24-битным сигналом без бита знака: Bits=24, sign=16#255, low=-10.0, high=10.0.

Пример

```

VAR
  (* Входной цифровой сигнал (например, от АЦП) *)
  DigitalValue: DWORD := 1024;

  (* Настройки преобразования *)
  BitCount:    BYTE := 12;          (* 12 значащих бит *)
  SignBit:     BYTE := 16#FF;       (* Бит знака отсутствует *)
  LowLimit:    REAL := 0.0;         (* Нижний предел аналогового сигнала *)
  HighLimit:   REAL := 10.0;        (* Верхний предел аналогового сигнала *)

  (* Результат преобразования *)
  AnalogValue: REAL;
END_VAR

(* Вызов функции AIN *)
AnalogValue := OSCAT_BASIC.AIN(
  in := DigitalValue,
  Bits := BitCount,
  sign := SignBit,
  low := LowLimit,
  high := HighLimit
);

(* Результат: AnalogValue = 2.5 (В)
Пояснение:
- 1024 / 4095 = 0.25 (25% от максимального значения)
- 0.25 * 10.0 = 2.5 (В)
Таким образом, цифровой код 1024 соответствует напряжению 2.5 В*)
END_PROGRAM

```

22.2 AIN1

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	in	DWORD	Цифровой сигнал
Выходы	out	REAL	Аналоговый сигнал
	sign	BOOL	Знак сигнала
	error	BOOL	Флаг «ошибка»
	overflow	BOOL	Флаг «переполнение»
Параметры	sign_bit	INT	Номер бита, определяющего знак
	error_bit	INT	Номер бита ошибка
	error_code_en	BOOL	Бит обработки кода ошибки
	error_code	DWORD	Код ошибки
	overflow_bit	INT	Номер бита переполнения
	overflow_code_en	BOOL	Бит обработки кода переполнения
	overflow_code	DWORD	Код переполнения
	Bit_0	INT	Номер первого значащего бита
	Bit_N	INT	Номер последнего значащего бита
	out_min	REAL	Нижний предел аналогового сигнала
	out_max	REAL	Верхний предел аналогового сигнала
	code_min	DWORD	Нижний предел цифрового сигнала
	code_max	DWORD	Верхний предел цифрового сигнала
	error_output	REAL	Значение выхода в случае ошибки
	overflow_output	REAL	Значение выхода в случае переполнения

Функциональный блок **AIN1** используется для преобразования цифрового сигнала **in** в аналоговый **out**. Параметр **sign_bit** определяет номер бита, содержащего знак (его значение **FALSE** соответствует знаку «+»); если такого бита нет, то вход можно оставить пустым. Нумерация бит ведется с нуля.

Параметр **error_bit** определяет номера бита ошибки; если такого бита нет, то вход можно оставить пустым. Некоторые АЦП в случае ошибки транслируют на выход ее код. Этот код может быть задан в параметре **error_code**; параметр **error_code_en** в этом случае должен иметь значение **TRUE** для разрешения обработки кода. Параметр **error_output** определяет значение, которое будет подано на выход **out** в том случае, если диагностирована ошибка.

Параметр **overflow_bit** определяет номера бита переполнения (выхода за диапазон допустимых значений) если такого бита нет, то вход можно оставить пустым. Некоторые АЦП в случае ошибки переполнения транслируют на выход ее код. Этот код может быть задан в параметре **overflow_code**; параметр **overflow_code_en** в этом случае должен иметь значение **TRUE** для разрешения обработки кода. Параметры **code_min** и **code_max** определяют минимальное и максимальное значение цифрового сигнала. Параметр **overflow_output** определяет значение, которое будет подано на выход **out** в том случае, если выходной сигнал выходит за допустимый диапазон.

Параметры **out_min** и **out_max** определяют минимальное и максимальное значение аналогового сигнала.

Параметры **Bit_0** и **Bit_N** определяют номера первого и последнего значащих бит.

Выход **out** содержит полученное аналоговое значение. Выход **sign** определяет значение бита знака. Выходы **error** и **overflow** принимают значение **TRUE** при возникновении ошибки и переполнения соответственно.

22.3 AOUT

Тип модуля: функция	Переменная	Тип	Описание
Входы	in	REAL	Аналоговый сигнал
Выходы	AOUT	DWORD	Цифровой сигнал
Параметры	Bits	BYTE	Число значащих бит
	sign	BYTE	Номер бита, определяющего знак
	low	REAL	Нижний предел аналогового сигнала
	high	REAL	Верхний предел аналогового сигнала

Функция **AOUT** используется для преобразования аналогового сигнала **in** в цифровой **AOUT**. Вход **Bits** определяет число значащих бит цифрового сигнала (максимальное значение – 32). Вход **sign** определяет номер бита, содержащего знак (его значение **FALSE** соответствует знаку «+»); если такой бит не требуется, то на вход подается значение **16#FF**. Входы **low** и **high** определяют минимальное и максимальное значение аналогового сигнала. Нумерация бит ведется с нуля.

Ниже приведено несколько примеров настройки входов для разных сигналов.

- Аналоговый сигнал в диапазоне 0–10 преобразуется в цифровой 12-битный сигнал: Bits=12, sign=16#255, low=0.0, high=10.0;
- Аналоговый сигнал в диапазоне -10–10 преобразуется в цифровой 14-битный сигнал (биты 0-13) с битом знака (бит 14): Bits=14, sign=14, low=-10.0, high=10.0;
- Аналоговый сигнал в диапазоне -10–10 преобразуется в цифровой 24-битный сигнал без бита знака: Bits=24, sign=16#255, low=-10.0, high=10.0.

Пример

```

VAR
(* Входной аналоговый сигнал *)
AnalogValue: REAL := 7.5;      (*Напряжение 7.5 В*)

(* Настройки преобразования *)
BitCount:    BYTE := 12;      (*12 значащих бит*)
SignBit:     BYTE := 16#FF;   (*Бит знака отсутствует*)
LowLimit:    REAL := 0.0;     (*Нижний предел 0 В*)
HighLimit:   REAL := 10.0;    (*Верхний предел 10 В*)

(* Результат преобразования *)
DigitalValue: DWORD;
END_VAR

(* Вызов функции AOUT *)
DigitalValue := OSCAT_BASIC.AOUT(
  in    := AnalogValue,
  Bits  := BitCount,
  sign  := SignBit,
  low   := LowLimit,
  high  := HighLimit
);

(* Результат: DigitalValue = 3072 (десятичное) или 16#0C00 (шестнадцатеричное)
Пояснение:
7.5 / 10.0 = 0.75 (75% от диапазона)
0.75 * 4095 = 3071.25 ≈ 3072*)
END_PROGRAM

```

22.4 AOUT1

Тип модуля: функция	Переменная	Тип	Описание
Входы	in	REAL	Аналоговый сигнал
Выходы	AOUT	DWORD	Цифровой сигнал

Тип модуля: функция	Переменная	Тип	Описание
Параметры	Bits	BYTE	Число значащих бит
	sign	BYTE	Номер бита, определяющего знак
	low	REAL	Нижний предел аналогового сигнала
	high	REAL	Верхний предел аналогового сигнала
Используемые модули	SIGN_R		

Функция **AOUT1** используется для преобразования аналогового сигнала **in** в цифровой **AOUT1**. Параметр **sign_bit** определяет номер бита, содержащего знак (его значение **FALSE** соответствует знаку «+»); если такой бит не требуется, то вход можно оставить пустым. Параметры **Bit_0** и **Bit_N** определяют номера первого и последнего значащих бит. Входы **low** и **high** определяют минимальное и максимальное значение аналогового сигнала. Нумерация бит ведется с нуля.

Пример

```

VAR
  (* Входной аналоговый сигнал *)
  AnalogValue: REAL := 2.5;      (* 2.5 В *)

  (* Настройки преобразования *)
  FirstBit:    INT := 0;          (*Номер первого значащего бита*)
  LastBit:     INT := 11;         (*Номер последнего значащего бита
                                   (всего 12 бит: 0..11)*)
  SignBit:     INT := 31;         (*31-й бит — знак*)
  LowLimit:    REAL := -10.0;     (*Нижний предел -10 В*)
  HighLimit:   REAL := 10.0;     (*Верхний предел 10 В*)

  (* Результат преобразования *)
  DigitalValue: DWORD;
END_VAR

(* Вызов функции AOUT1 *)
DigitalValue := OSCAT_BASIC.AOUT1(
  in      := AnalogValue,
  Bit_0   := FirstBit,
  Bit_N   := LastBit,
  sign    := SignBit,
  low     := LowLimit,
  high    := HighLimit
);

(*Итоговое значение: 31-й бит = 0, младшие 12 бит = 2559*)
END_PROGRAM

```

22.5 BYTE_TO_RANGE

Тип модуля: функция	Переменная	Тип	Описание
Входы	x	BYTE	Исходное значение
	low	REAL	Нижний предел отмасштабированного значения
	high	REAL	Верхний предел отмасштабированного значения
Выходы	BYTE_TO_RANGE	REAL	Отмасштабированное значение

Функция **BYTE_TO_RANGE** линейно масштабирует входное целочисленное значение **x** из диапазона (**0...255**) в значение с плавающей точкой из диапазона (**low...high**). См. также обратную функцию **RANGE_TO_BYTE**.

Пример

```

VAR
  (* Входное значение *)
  RawValue:   BYTE := 25;           (* 25 из диапазона 0..255 *)
  (* Границы масштабирования *)
  LowLimit:   REAL := 0.0;          (* Нижний предел *)
  HighLimit:  REAL := 100.0;        (* Верхний предел *)
  (* Результат *)
  ScaledValue: REAL;
END_VAR

(* Вызов функции BYTE_TO_RANGE *)
ScaledValue := OSCAT_BASIC.BYTE_TO_RANGE(
  x      := RawValue,
  low    := LowLimit,
  high   := HighLimit
);

(* Результат: ScaledValue ≈ 9.803922, т.к.
  25 / 255 ≈ 0.09804 (9.8% от диапазона)
  0.09803922 * (100.0 - 0.0) = 9.803922
  То есть значение 25 из диапазона 0..255
  соответствует 9.8 из диапазона 0.0..100.0*)

END_PROGRAM

```

22.6 DELAY

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	REAL	Исходное значение
	N	INT	Кол-во циклов задержки
	RST	BOOL	Сигнал сброса блока
Выходы	OUT	REAL	Значение с задержкой
Используемые модули	INC1		

Функциональный блок **DELAY** транслирует на выход **OUT** значение входа **IN** спустя время задержки, равное **N** циклов ПЛК. По переднему фронту входа **RST** память блока очищается, и задержка начинает отсчитываться заново.

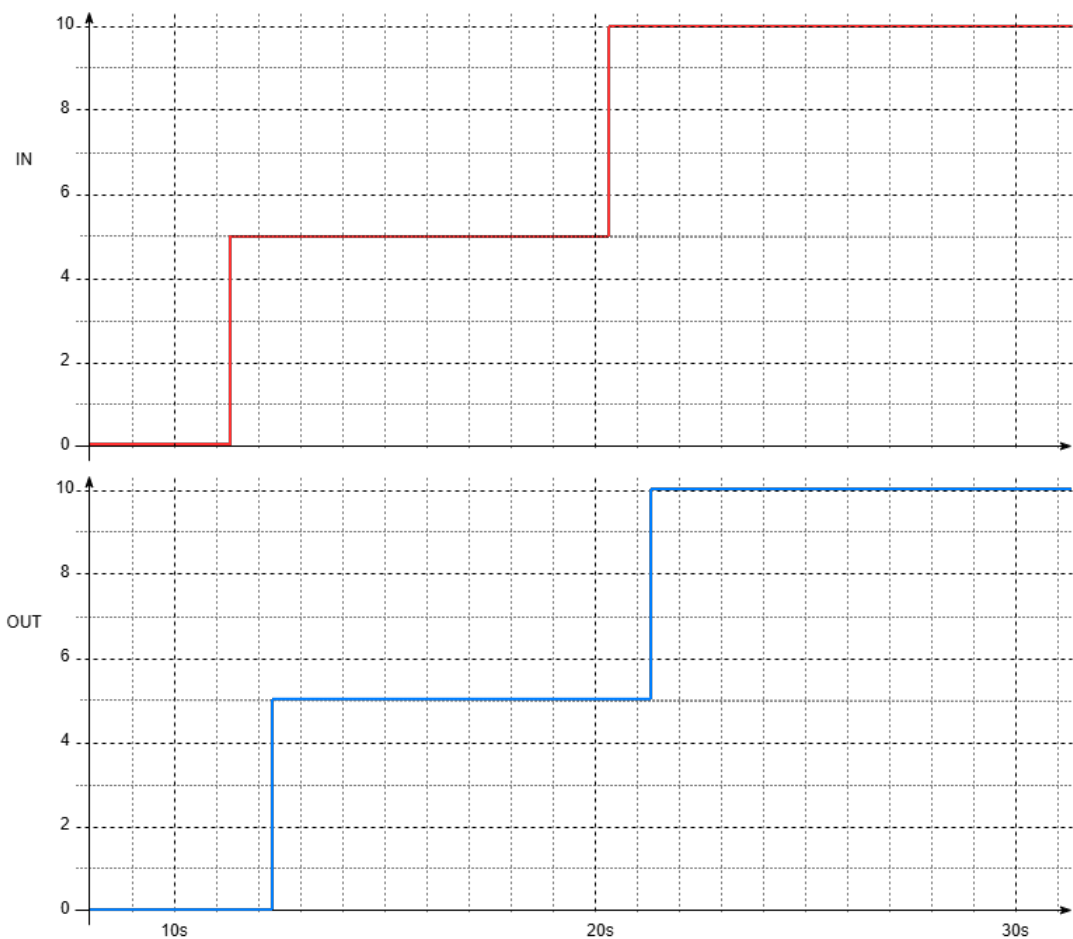


Рисунок 22.1 – Трассировка работы ФБ DELAY для ПЛК с временем цикла=100 мс, N=10

22.7 DELAY_4

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	REAL	Исходное значение
Выходы	OUT1	REAL	Значение с задержкой (1 цикл)
	OUT2	REAL	Значение с задержкой (2 цикла)
	OUT3	REAL	Значение с задержкой (3 цикла)
	OUT4	REAL	Значение с задержкой (4 цикла)

Функциональный блок **DELAY_4** транслирует на выходы **OUT1... OUT4** значение входа **IN** спустя время задержки, равное 1...4 цикла ПЛК.

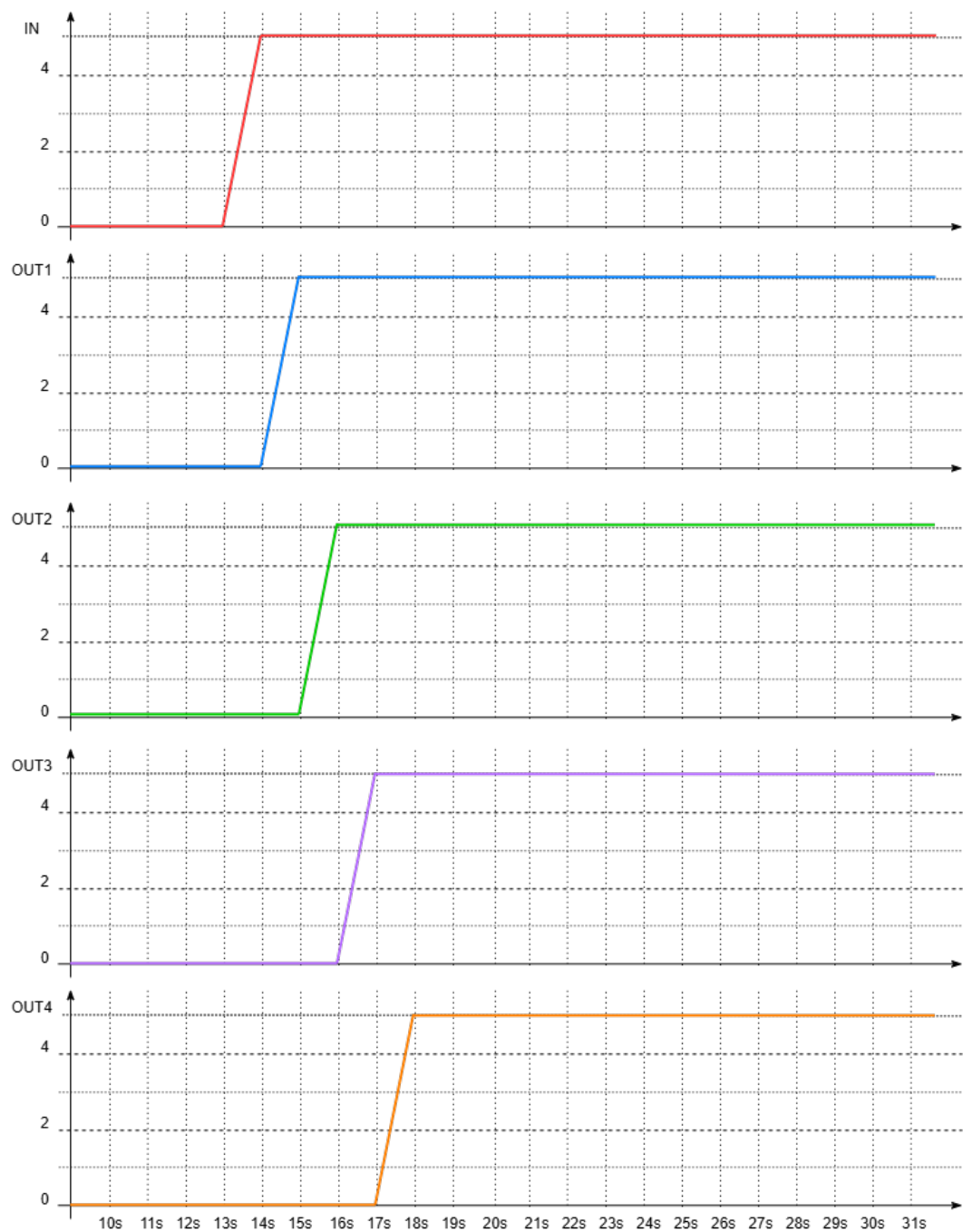


Рисунок 22.2 – Трассировка работы ФБ DELAY_4 для ПЛК с временем цикла=1 с

22.8 FADE

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN1	REAL	Значение 1
	IN2	REAL	Значение 2
	F	BOOL	Направление изменения
	TF	TIME	Время переключения значений
	rst	BOOL	Сигнал сброса блока
Выходы	Y	REAL	Выход блока
Используемые модули	RMP_W		

Функциональный блок **FADE** реализует плавное переключение входных значений **IN1** и **IN2** на выходе **Y** – от меньшего к большему, если вход **F** имеет значение **TRUE**, или от большего к меньшему, если вход **F** имеет значение **FALSE**.

Вход **TF** определяет время переключения (**T** – время с начала работы блока):

- $Y(T=0)=IN1$
- $Y(T=TF)=IN2$
- $Y(0<T<TF)=(T/TF) * IN1 + (1 - T/TF) * IN2$

По переднему фронту входа **RST** выход **Y** принимает значение **IN1** (если **F=FALSE**) или **IN2** (если **F=TRUE**).

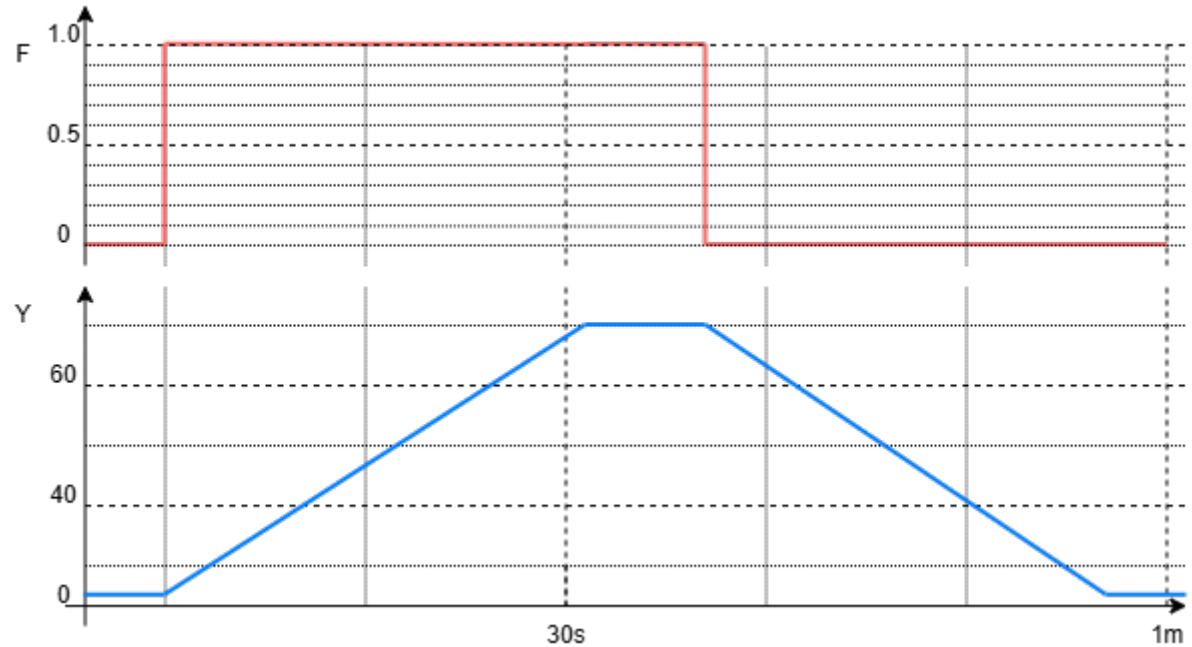


Рисунок 22.3 – Трассировка работы ФБ FADE (IN1=25.0, IN=70.0, TF=T#20s)

22.9 FILTER_DW

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	X	DWORD	Фильтруемое значение
	T	TIME	Постоянная времени фильтра
Выходы	Y	DWORD	Отфильтрованное значение
Используемые модули	T_PLC_MS		

Функциональный блок **FILTER_DW** представляет собой фильтр нижних частот для входного значения **X** типа **DWORD**. Это может быть полезным при фильтрации шума в сигналах от датчиков. Вход **T** определяет постоянную времени фильтра, которая связана с частотой пропускания следующей формулой:

$$f_{гр} = \frac{1}{2\pi T}$$

На выход **Y** подается отфильтрованное значение.

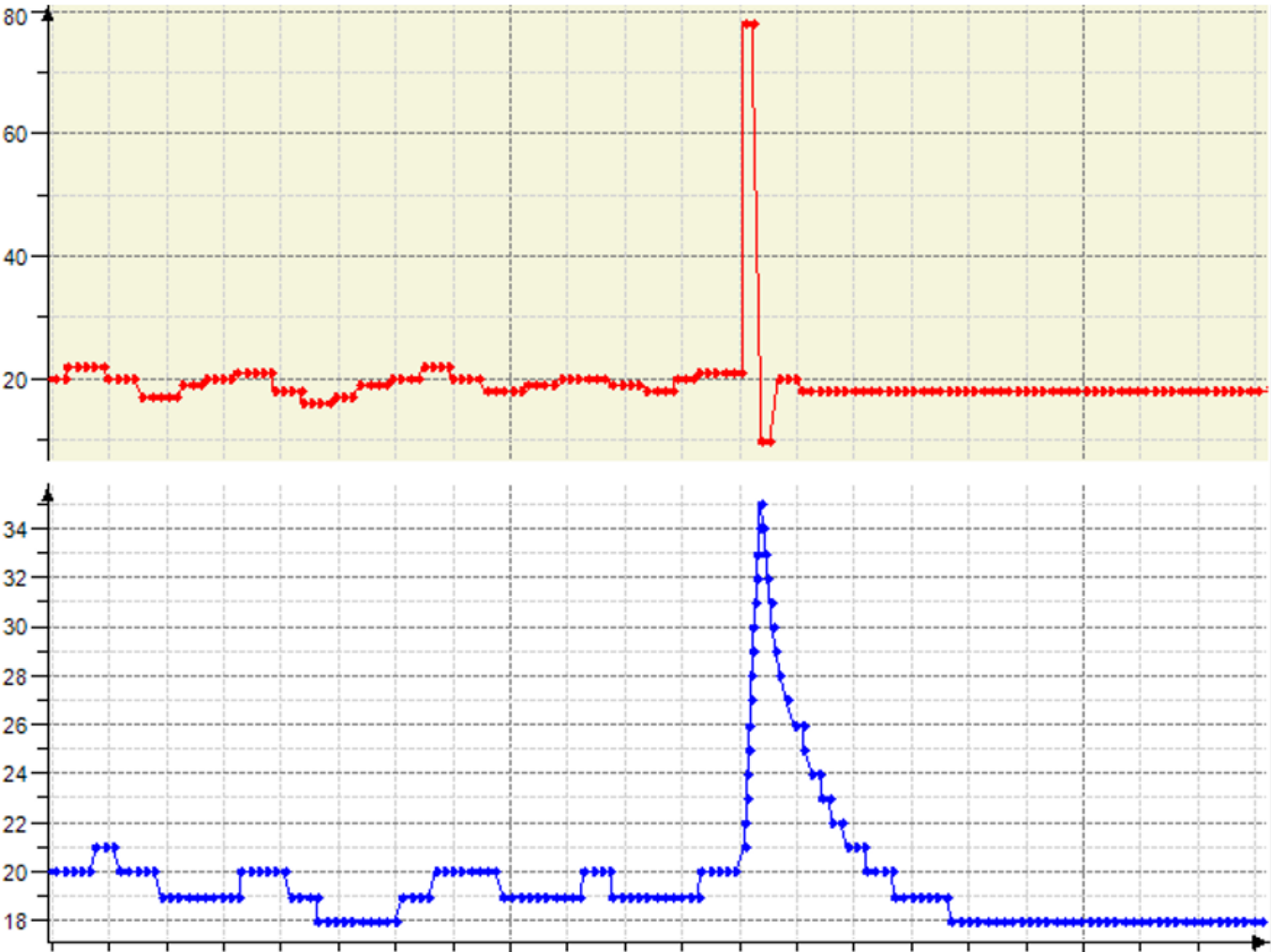


Рисунок 22.4 – Трассировка работы ФБ FILTER_DW (T=T#1s)

22.10 FILTER_I

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	X	INT	Фильтруемое значение
	T	TIME	Постоянная времени фильтра
Выходы	Y	INT	Отфильтрованное значение
Используемые модули	T_PLC_MS		

Функциональный блок **FILTER_I** представляет собой фильтр нижних частот для входного значения **X** типа **INT**. Это может быть полезным при фильтрации шума в сигналах от датчиков. Вход **T** определяет постоянную

$$f_{гр} = \frac{1}{2\pi T}$$

времени фильтра, которая связана с частотой пропускания следующей формулой:

На выход **Y** подается отфильтрованное значение.

Трассировка работы **FILTER_I** аналогичного по принципу блока **FILTER_DW** (который отличается только типом фильтруемой переменной).

22.11 FILTER_MAV_DW

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	X	DWORD	Фильтруемое значение
	N	UINT	Число точек усреднения (0..31)

Тип модуля: ФБ	Переменная	Тип	Описание
	RST	BOOL	Сигнал сброса блока
Выходы	Y	DWORD	Отфильтрованное значение
Используемые модули	INC1		

Функциональный блок **FILTER_MAV_DW** представляет собой фильтр скользящей средней для входного значения **X** типа **DWORD**. Это может быть полезным для исключения влияния на сигнал случайной составляющей. Вход **N** определяет сохраняемое число значений входа **X** (**N** принадлежит диапазону 0...31; **0** – только текущее значение, **1** – текущее и предыдущее и т.д.) для вычисления текущего отфильтрованного значения. По переднему фронту входа **RST** информация о предыдущих входных значениях удаляется.

На выход **Y** подается отфильтрованное значение, вычисленное по формуле (**K** – номер текущего цикла ПЛК):

$$Y_K = \frac{1}{N} \sum_{i=K-N}^K X_i$$

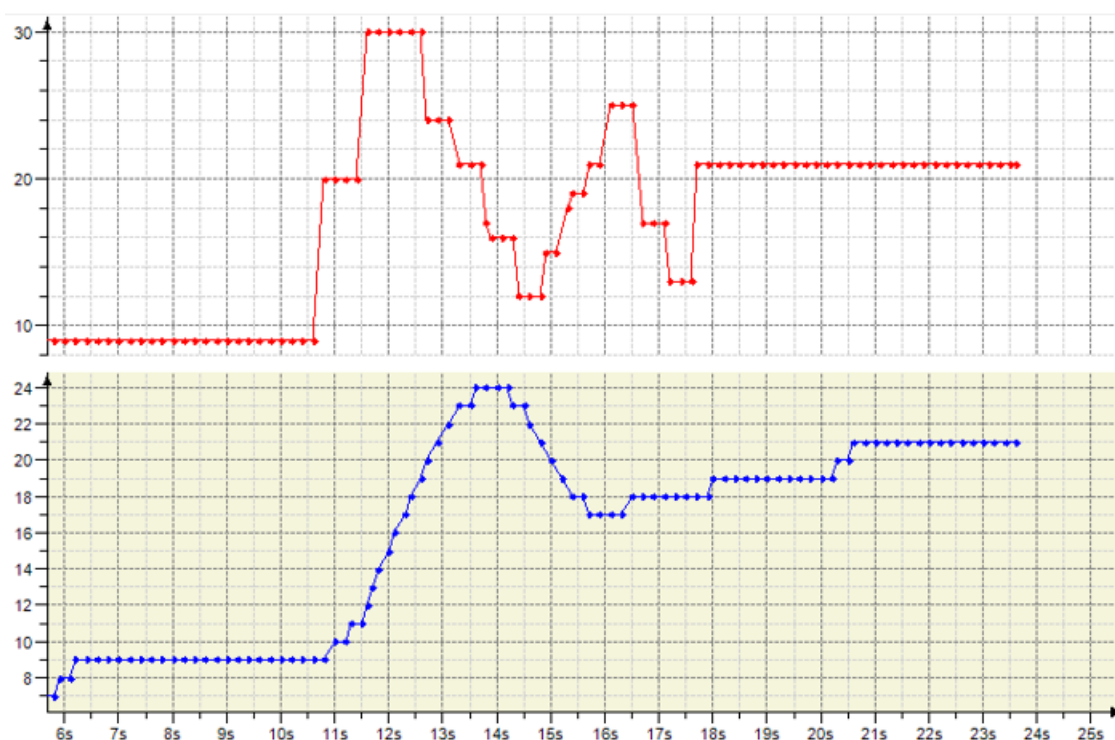


Рисунок 22.5 – Трассировка работы ФБ FILTER_MAV_DW (N=30)

22.12 FILTER_MAV_W

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	X	WORD	Фильтруемое значение
	N	UINT	Число точек усреднения (0..31)
	RST	BOOL	Сигнал сброса блока
Выходы	Y	WORD	Отфильтрованное значение
Используемые модули	INC1		

Функциональный блок **FILTER_MAV_W** представляет собой фильтр скользящей средней для входного значения **X** типа **WORD**. Принцип работы блока полностью соответствует блоку **FILTER_MAV_DW**, единственным отличием является тип фильтруемого значения.

22.13 FILTER_W

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	X	WORD	Фильтруемое значение
	T	TIME	Постоянная времени фильтра
Выходы	Y	WORD	Отфильтрованное значение
Используемые модули	T_PLC_MS		

Функциональный блок **FILTER_W** представляет собой **фильтр нижних частот** для входного значения **X** типа **WORD**. Это может быть полезным при фильтрации шума в сигналах от датчиков. Принцип работы блока полностью соответствует блоку **FILTER_DW**, единственным отличием является тип фильтруемого значения.

22.14 FILTER_WAV

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	X	REAL	Фильтруемое значение
	W	ARRAY [0..15] OF REAL	Массив коэффициентов
	RST	BOOL	Сигнал сброса блока
Выходы	Y	REAL	Отфильтрованное значение
Используемые модули	INC1, DEC1		

Функциональный блок **FILTER_WAV** представляет собой КИХ-фильтр для входного значения **X** типа **REAL**. Вход **W** определяет весовые коэффициенты для входных значений предыдущих 16-ти циклов ПЛК. На выход **Y** подается отфильтрованное значение, вычисленное по формуле (**K** – номер текущего цикла ПЛК):

$$Y_K = \sum_{i=K-16}^K W_i \cdot X_i$$

По переднему фронту входа **RST** информация о предыдущих входных значениях удаляется.

22.15 MIX

Тип модуля: функция	Переменная	Тип	Описание
Входы	A	REAL	Значение 1
	B	REAL	Значение 2
	M	REAL	Коэффициент смещения
Выходы	MIX	REAL	Вычисленное значение

Функция **MIX** возвращает значение, вычисленное по формуле:

$$MIX = (M - 1) * A + M * B$$

Подразумевается, что значение коэффициента смещения **M** должно принадлежать диапазону **[0,1]**.

Пример

```

VAR
  (* Исходные значения *)
  ValueA:    REAL := 6.0;      (* Первое значение *)
  ValueB:    REAL := 15.0;     (* Второе значение *)
  (* Коэффициент смещения *)
  MixFactor: REAL := 0.6666;   (* 66.66% от B, 33.34% от A *)
  (* Результат *)
  Result:    REAL;
END_VAR

(* Вызов функции MIX *)
Result := OSCAT_BASIC.MIX(
  A := ValueA,
  B := ValueB,
  M := MixFactor
);

(*Результат: Result ≈ 12.0*)
END_PROGRAM

```

22.16 MUX_R2

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN0	REAL	Информационный вход 0
	IN1	REAL	Информационный вход 1
	A	BOOL	Адресный вход
Выходы	MUX_R2	REAL	Выход мультиплексора

Функция **MUX_2R** реализует мультиплексор для двух **REAL** переменных. Принцип работы функции:

- если A=FALSE, то MUX_R2:=IN0;
- если A=TRUE, то MUX_R2:=IN1.

Пример

```

VAR
  (* Информационные входы *)
  Input0:    REAL := 10.0;    (* Значение при A = FALSE *)
  Input1:    REAL := 20.0;    (* Значение при A = TRUE *)
  (* Адресный вход *)
  Selector:  BOOL := FALSE;    (* Выбор входа *)
  (* Результат *)
  Output:    REAL;
END_VAR

(* Вызов функции MUX_R2 *)
Output := OSCAT_BASIC.MUX_R2(
  IN0 := Input0,
  IN1 := Input1,
  A   := Selector
);

(*Результат: Output = 10.0*)
END_PROGRAM

```

22.17 MUX_R4

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN0	BOOL	Информационный вход 0
	IN1	BOOL	Информационный вход 1
	IN2	BOOL	Информационный вход 2
	IN3	BOOL	Информационный вход 3

Тип модуля: функция	Переменная	Тип	Описание
	A0	BOOL	Адресный вход 0
	A1	BOOL	Адресный вход 1
Выходы	MUX_R4	REAL	Выход мультиплексора

Функция **MUX_4R** реализует мультиплексор для четырех **REAL** переменных. Принцип работы функции:

- если A0=FALSE и A1=FALSE, то MUX_4R:=IN0;
- если A0=TRUE и A1=FALSE, то MUX_4R:=IN1;
- если A0=FALSE и A1=TRUE, то MUX_4R:=IN2;
- если A0=TRUE и A1=TRUE, то MUX_4R:=IN3.

Пример

```

VAR
  (* Информационные входы (4 источника) *)
  Input0 : REAL := 10.0;      (* Выбирается при A0=FALSE, A1=FALSE *)
  Input1 : REAL := 20.0;      (* Выбирается при A0=TRUE, A1=FALSE *)
  Input2 : REAL := 30.0;      (* Выбирается при A0=FALSE, A1=TRUE *)
  Input3 : REAL := 40.0;      (* Выбирается при A0=TRUE, A1=TRUE *)

  (* Адресные входы *)
  Address0 : BOOL := FALSE;    (* Младший бит адреса *)
  Address1 : BOOL := FALSE;    (* Старший бит адреса *)

  (* Результат *)
  Output : REAL;
END_VAR

(* Вызов функции MUX_R4 *)
Output := OSCAT_BASIC.MUX_R4(
  IN0 := Input0,
  IN1 := Input1,
  IN2 := Input2,
  IN3 := Input3,
  A0  := Address0,
  A1  := Address1
);

(*Результат: Output = 10.0*)
END_PROGRAM

```

22.18 OFFSET

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Исходное значение
	O1	BOOL	Режим «смещение 1»
	O2	BOOL	Режим «смещение 2»
	O3	BOOL	Режим «смещение 3»
	O4	BOOL	Режим «смещение 4»
	D	BOOL	Режим «имитация входа»
	Offset_1	REAL	Коэффициент смещения 1
	Offset_2	REAL	Коэффициент смещения 2
	Offset_3	REAL	Коэффициент смещения 3
	Offset_4	REAL	Коэффициент смещения 4
	default	REAL	Значение входа в имитации
Выходы	OFFSET	REAL	Смещенное значение

Функция **OFFSET** возвращает смещенное значение входной переменной **X**. Входы **O1...O4** определяют, какие коэффициенты смещения будут прибавляться к входному значению, а входы **Offset_1...Offset_4** определяют

величины этих коэффициентов. Если одновременно будут активны несколько входов, то к входному значению будут прибавлены величины нескольких коэффициентов. Если вход **D** имеет значение **TRUE**, то функция вместо **X** использует значение **default**.

Пример

```

VAR
  (* Входное значение *)
  InputValue : REAL := 100.0;
  (* Режимы смещения *)
  EnableOffset1 : BOOL := TRUE;      (* Включаем смещение 1 *)
  EnableOffset2 : BOOL := FALSE;
  EnableOffset3 : BOOL := FALSE;
  EnableOffset4 : BOOL := FALSE;
  (* Коэффициенты смещения *)
  Offset_1 : REAL := 10.0;
  Offset_2 : REAL := 0.0;
  Offset_3 : REAL := 0.0;
  Offset_4 : REAL := 0.0;
  (* Режим имитации *)
  Simulate : BOOL := FALSE;
  DefaultValue : REAL := 50.0;
  (* Результат *)
  Result : REAL;
END_VAR

(* Вызов функции OFFSET *)
Result := OSCAT_BASIC.OFFSET(
  X := InputValue,
  O1 := EnableOffset1,
  O2 := EnableOffset2,
  O3 := EnableOffset3,
  O4 := EnableOffset4,
  D := Simulate,
  Offset_1 := Offset_1,
  Offset_2 := Offset_2,
  Offset_3 := Offset_3,
  Offset_4 := Offset_4,
  default := DefaultValue
);

(*Результат: Result = 110.0*)
END_PROGRAM

```

22.19 OFFSET2

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Исходное значение
	O1	BOOL	Режим «смещение 1»
	O2	BOOL	Режим «смещение 2»
	O3	BOOL	Режим «смещение 3»
	O4	BOOL	Режим «смещение 4»
	D	BOOL	Режим «имитация входа»
	Offset_1	REAL	Коэффициент смещения 1
	Offset_2	REAL	Коэффициент смещения 2
	Offset_3	REAL	Коэффициент смещения 3
	Offset_4	REAL	Коэффициент смещения 4
	default	REAL	Значение входа в имитации
Выходы	OFFSET2	REAL	Смещенное значение

Функция **OFFSET2** возвращает смещенное значение входной переменной **X**. Входы **O1...O4** определяют, какие коэффициенты смещения будут прибавляться к входному значению, а входы **Offset_1...Offset_4** определяют величины этих коэффициентов. Если одновременно будут активны несколько входов, то к входному значению будет прибавлен коэффициент входа с наибольшим номером (в отличие от функции **OFFSET**, в которой коэффициенты складываются). Если вход **D** имеет значение **TRUE**, то функция вместо **X** использует значение **default**.

Пример

```
VAR
  (* Входное значение *)
  InputValue : REAL := 100.0;
  (* Режимы смещения *)
  EnableOffset1 : BOOL := TRUE;      (* Включаем смещение 1 *)
  EnableOffset2 : BOOL := FALSE;
  EnableOffset3 : BOOL := FALSE;
  EnableOffset4 : BOOL := FALSE;
  (* Коэффициенты смещения *)
  Offset_1 : REAL := 10.0;
  Offset_2 : REAL := 20.0;
  Offset_3 : REAL := 30.0;
  Offset_4 : REAL := 40.0;
  (* Режим имитации *)
  Simulate : BOOL := FALSE;
  DefaultValue : REAL := 50.0;
  (* Результат *)
  Result : REAL;
END_VAR

(* Вызов функции OFFSET2 *)
Result := OSCAT_BASIC.OFFSET2(
  X := InputValue,
  O1 := EnableOffset1,
  O2 := EnableOffset2,
  O3 := EnableOffset3,
  O4 := EnableOffset4,
  D := Simulate,
  Offset_1 := Offset_1,
  Offset_2 := Offset_2,
  Offset_3 := Offset_3,
  Offset_4 := Offset_4,
  default := DefaultValue
);

(*Результат: Result = 110.0*)
END_PROGRAM
```

22.20 OVERRIDE

Тип модуля: функция	Переменная	Тип	Описание
Входы	X1	REAL	Исходное значение 1
	X2	REAL	Исходное значение 2
	X3	REAL	Исходное значение 3
	E1	BOOL	Активация значения 1
	E2	BOOL	Активация значения 2
	E3	BOOL	Активация значения 3
Выходы	OVERRIDE	REAL	Наибольшее активное значение

Функция **OVERRIDE** возвращает наибольшее из активных значений по модулю **X1...X3**. Активность значения определяется входным сигналом **E1...E3** (**TRUE** – значение активно). Если активных значений нет, то функция возвращает ноль.

Пример

```

VAR
  (* Исходные значения *)
  Value1 : REAL := 50.0;
  Value2 : REAL := -30.0;
  Value3 : REAL := 20.0;
  (* Флаги активности *)
  Enable1 : BOOL := TRUE;      (* Значение 1 активно *)
  Enable2 : BOOL := FALSE;
  Enable3 : BOOL := FALSE;
  (* Результат *)
  Result : REAL;
END_VAR

(* Вызов функции OVERRIDE *)
Result := OSCAT_BASIC.OVERRIDE(
  X1 := Value1,
  X2 := Value2,
  X3 := Value3,
  E1 := Enable1,
  E2 := Enable2,
  E3 := Enable3
);
(*Результат: Result = 50.0*)

END_PROGRAM

```

22.21 RANGE_TO_BYTE

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Исходное значение
	low	REAL	Нижний предел исходного значения
	high	REAL	Верхний предел исходного значения
Выходы	RANGE_TO_BYTE	BYTE	Отмасштабированное значение

Функция **RANGE_TO_BYTE** линейное масштабирует входное значение с плавающей точкой **X** из диапазона (**low... high**) в целочисленное значение типа **BYTE** из диапазона (**0... 255**). См. также обратную функцию [BYTE_TO_RANGE](#).

Пример

```

VAR
  (* Входное значение с плавающей точкой *)
  InputValue: REAL := 25.0;      (* 25.0 из диапазона 0.0...100.0 *)
  (* Границы исходного диапазона *)
  LowLimit:   REAL := 0.0;       (* Нижний предел *)
  HighLimit:  REAL := 100.0;     (* Верхний предел *)
  (* Результат масштабирования *)
  OutputByte: BYTE;
END_VAR

(* Вызов функции RANGE_TO_BYTE *)
OutputByte := OSCAT_BASIC.RANGE_TO_BYTE(
  X   := InputValue,
  low := LowLimit,
  high := HighLimit
);
(*Результат: OutputByte = 64*)

END_PROGRAM

```


22.22 RANGE_TO_WORD

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Исходное значение
	low	REAL	Нижний предел исходного значения
	high	REAL	Верхний предел исходного значения
Выходы	RANGE_TO_WORD	WORD	Отмасштабированное значение

Функция **RANGE_TO_WORD** линейно масштабирует входное значение с плавающей точкой **X** из диапазона (**low...high**) в целочисленное значение типа **WORD** из диапазона (**0...65535**). См. также обратную функцию [WORD_TO_RANGE](#).

Пример

```

VAR
  (* Входное значение с плавающей точкой *)
  InputValue: REAL := 25.0;      (* 25.0 из диапазона 0.0..100.0 *)
  (* Границы исходного диапазона *)
  LowLimit:   REAL := 0.0;       (* Нижний предел *)
  HighLimit:  REAL := 100.0;     (* Верхний предел *)
  (* Результат масштабирования *)
  OutputWord: WORD;
END_VAR

(* Вызов функции RANGE_TO_WORD *)
OutputWord := OSCAT_BASIC.RANGE_TO_WORD(
  X      := InputValue,
  low    := LowLimit,
  high   := HighLimit
);
(*Результат: OutputWord = 16383*)
END_PROGRAM

```

22.23 SCALE

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Исходное значение
	K	REAL	Коэффициент масштабирования
	O	REAL	Сдвиг
	MX	REAL	Верхний предел отмасштабированного значения
	MN	REAL	Нижний предел отмасштабированного значения
Выходы	SCALE	REAL	Отмасштабированное значение

Функция **SCALE** возвращает значение линейной функции с коэффициентом **K** и сдвигом **O** для аргумента **X**. Входы **MN** и **MX** определяют нижний и верхний предел возвращаемого значения.

$SCALE = LIMIT(MN, K * X + O, MX)$

Пример

```

VAR
  (* Входное значение *)
  InputValue: REAL := 25.0;      (* Исходное значение *)
  (* Коэффициенты масштабирования *)
  Gain:      REAL := 2.0;        (* Коэффициент масштабирования K *)
  Offset:    REAL := 10.0;       (* Сдвиг O *)
  (* Границы ограничения *)
  MinLimit:  REAL := 15.0;       (* Нижний предел MN *)
  MaxLimit:  REAL := 75.0;       (* Верхний предел MX *)
  (* Результат *)
  Result:    REAL;
END_VAR

(* Вызов функции SCALE *)
Result := OSCAT_BASIC.SCALE(
  X := InputValue,
  K := Gain,
  O := Offset,
  MX := MaxLimit,
  MN := MinLimit
);

(* Результат: Result = 60.0
Расчет:
1. Промежуточное значение:  $K * X + O = 2.0 * 25.0 + 10.0 = 60.0$ 
2. Проверка ограничений:  $15.0 \leq 60.0 \leq 75.0$ 
3. Значение в пределах диапазона, возвращается 60.0 *)
END_PROGRAM

```

22.24 SCALE_B

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	BYTE	Исходное значение
	I_LO	BYTE	Нижний предел исходного значения
	I_HI	BYTE	Верхний предел исходного значения
	O_LO	REAL	Нижний предел отмасштабированного значения
	O_HI	REAL	Верхний предел отмасштабированного значения
Выходы	SCALE_B	REAL	Отмасштабированное значение

Функция **SCALE_B** линейно масштабирует входное целочисленное значение **X** из диапазона (**I_LO...I_HI**) в значение с плавающей точкой из диапазона (**O_LO...O_HI**).

Пример

```

VAR
  (* Входное целочисленное значение *)
  InputValue: BYTE := 32;          (* 32 из диапазона 0...128 *)
  (* Границы исходного диапазона (целочисленные) *)
  InLow:      BYTE := 0;           (* Нижний предел *)
  InHigh:     BYTE := 128;        (* Верхний предел *)
  (* Границы выходного диапазона (вещественные) *)
  OutLow:     REAL := -10.0;       (* Нижний предел *)
  OutHigh:    REAL := 10.0;       (* Верхний предел *)
  (* Результат масштабирования *)
  OutputValue: REAL;
END_VAR

(* Вызов функции SCALE_B *)
OutputValue := OSCAT_BASIC.SCALE_B(
  X := InputValue,
  I_LO := InLow,
  I_HI := InHigh,
  O_LO := OutLow,
  O_HI := OutHigh
);
(*Результат: OutputValue = -5.0*)
END_PROGRAM

```

22.25 SCALE_B2

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN1	BYTE	Исходное значение 1
	IN2	BYTE	Исходное значение 2
	K	REAL	Коэффициент
	O	REAL	Сдвиг
Выходы	SCALE_B2	REAL	Рассчитанное значение
Параметры	IN1_MIN	REAL	Нижний предел отмасштабированного значения 1
	IN1_MAX	REAL	Верхний предел отмасштабированного значения 1
	IN2_MIN	REAL	Нижний предел отмасштабированного значения 2
	IN2_MAX	REAL	Верхний предел отмасштабированного значения 2

Функция **SCALE_B2** линейно масштабирует входное целочисленное значение **IN#** из диапазона (**0...255**) в значение с плавающей точкой из диапазона (**IN#_MIN...IN#_MAX**), суммирует их и возвращает для суммы значение линейного функции с коэффициентом **K** и сдвигом **O**.

$$SCALE_B2 = O + K \cdot \sum_{i=1}^2 IN_i$$

Это может быть использовано, например, для расчета общего количества воздуха в системе вентиляции.

Пример

```

VAR
  (* Входные целочисленные значения (0...255) *)
  Input1: BYTE := 32;          (* Первое значение *)
  Input2: BYTE := 64;          (* Второе значение *)
  (* Коэффициенты масштабирования результата *)
  Gain: REAL := 2.0;           (* Коэффициент К *)
  Offset: REAL := 4.0;         (* Сдвиг О *)
  (* Границы масштабирования для IN1 *)
  In1_Min: REAL := 0.0;        (* Нижний предел для IN1 *)
  In1_Max: REAL := 8.0;        (* Верхний предел для IN1 *)
  (* Границы масштабирования для IN2 *)
  In2_Min: REAL := 0.0;        (* Нижний предел для IN2 *)
  In2_Max: REAL := 8.0;        (* Верхний предел для IN2 *)
  (* Результат *)
  Result: REAL;
END_VAR

(* Вызов функции SCALE_B2 *)
Result := OSCAT_BASIC.SCALE_B2(
  IN1 := Input1,
  IN2 := Input2,
  K := Gain,
  O := Offset,
  IN1_MIN := In1_Min,
  IN1_MAX := In1_Max,
  IN2_MIN := In2_Min,
  IN2_MAX := In2_Max
);

(* Результат 10.0235294118*)
END_PROGRAM

```

22.26 SCALE_B4

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN1...IN4	BYTE	Исходное значение 1...4
	K	REAL	Коэффициент
	O	REAL	Сдвиг
Выходы	SCALE_B4	REAL	Рассчитанное значение
Параметры	IN1_MIN...IN4_MIN	REAL	Нижний предел отмасштабированного значения 1...4
	IN1_MAX...IN4_MAX	REAL	Верхний предел отмасштабированного значения 1...4

Функция **SCALE_B4** линейно масштабирует входное целочисленное значение **IN#** из диапазона (**0...255**) в значение с плавающей точкой из диапазона (**IN#_MIN...IN#_MAX**), суммирует их и возвращает для суммы значение линейного функции с коэффициентом **K** и сдвигом **O**.

$$SCALE_B4 = O + K \cdot \sum_{i=1}^4 IN_i$$

Это может быть использовано, например, для расчета общего количества воздуха в системе вентиляции.

22.27 SCALE_B8

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN1...IN8	BYTE	Исходное значение 1...8
	K	REAL	Коэффициент

Тип модуля: функция	Переменная	Тип	Описание
	O	REAL	Сдвиг
Выходы	SCALE_B8	REAL	Рассчитанное значение
Параметры	IN1_MIN...IN8_MIN	REAL	Нижний предел отмасштабированного значения 1...8
	IN1_MAX...IN8_MAX	REAL	Верхний предел отмасштабированного значения 1...8

Функция **SCALE_B8** линейно масштабирует входное целочисленное значение **IN#** из диапазона (**0...255**) в значение с плавающей точкой из диапазона (**IN#_MIN...IN#_MAX**), суммирует их и возвращает для суммы значение линейного функции с коэффициентом **K** и сдвигом **O**.

$$SCALE_B8 = O + K \cdot \sum_{i=1}^8 IN_i$$

Это может быть использовано, например, для расчета общего количества воздуха в системе вентиляции.

22.28 SCALE_D

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	DWORD	Исходное значение
	I_LO	DWORD	Нижний предел исходного значения
	I_HI	DWORD	Верхний предел исходного значения
	O_LO	REAL	Нижний предел отмасштабированного значения
	O_HI	REAL	Верхний предел отмасштабированного значения
Выходы	SCALE_D	REAL	Отмасштабированное значение

Функция **SCALE_D** линейно масштабирует входное целочисленное значение **X** из диапазона (**I_LO...I_HI**) в значение с плавающей точкой из диапазона (**O_LO...O_HI**).

Пример

```

VAR
  (* Входное целочисленное значение DWORD *)
  InputValue: DWORD := 75000;      (* 75000 из диапазона 0...100000 *)
  (* Границы исходного диапазона (DWORD) *)
  InLow:      DWORD := 0;          (* Нижний предел *)
  InHigh:     DWORD := 100000;     (* Верхний предел *)
  (* Границы выходного диапазона (REAL) *)
  OutLow:     REAL := -10.0;       (* Нижний предел *)
  OutHigh:    REAL := 10.0;        (* Верхний предел *)
  (* Результат масштабирования *)
  OutputValue: REAL;
END_VAR

(* Вызов функции SCALE_D *)
OutputValue := OSCAT_BASIC.SCALE_D(
  X := InputValue,
  I_LO := InLow,
  I_HI := InHigh,
  O_LO := OutLow,
  O_HI := OutHigh
);

(*Результат = 5.0*)
END_PROGRAM

```

22.29 SCALE_R

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Исходное значение
	I_LO	REAL	Нижний предел исходного значения
	I_HI	REAL	Верхний предел исходного значения
	O_LO	REAL	Нижний предел отмасштабированного значения
	O_HI	REAL	Верхний предел отмасштабированного значения
Выходы	SCALE_D	REAL	Отмасштабированное значение

Функция **SCALE_R** линейно масштабирует входное значение с плавающей точкой **X** из диапазона (**I_LO...I_HI**) в диапазон (**O_LO...O_HI**).

Пример

```

VAR
  (* Входное значение с плавающей точкой *)
  InputValue: REAL := 12.0;
  (* Границы исходного диапазона (REAL) *)
  InLow: REAL := 4.0; (* Нижний предел *)
  InHigh: REAL := 20.0; (* Верхний предел *)
  (* Границы выходного диапазона (REAL) *)
  OutLow: REAL := 0.0; (* Нижний предел *)
  OutHigh: REAL := 100.0; (* Верхний предел *)
  (* Результат масштабирования *)
  OutputValue: REAL;
END_VAR

(* Вызов функции SCALE_R *)
OutputValue := OSCAT_BASIC.SCALE_R(
  X := InputValue,
  I_LO := InLow,
  I_HI := InHigh,
  O_LO := OutLow,
  O_HI := OutHigh
);

(*Результат = 50.0*)
END_PROGRAM

```

22.30 SCALE_X2

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN1	BOOL	Исходное значение 1
	IN2	BOOL	Исходное значение 2
	K	REAL	Коэффициент
	O	REAL	Сдвиг
Выходы	SCALE_X2	REAL	Отмасштабированное значение
Параметры	IN1_MIN	REAL	Отмасштабированное значение 1 для IN1=FALSE
	IN1_MAX	REAL	Отмасштабированное значение 1 для IN1=TRUE
	IN2_MIN	REAL	Отмасштабированное значение 2 для IN2=FALSE
	IN2_MAX	REAL	Отмасштабированное значение 2 для IN2=TRUE

Функция **SCALE_X2** преобразует входное булевское значение **IN#** в значение с плавающей точкой (**IN#_MIN** для **IN#=FALSE** и **IN#_MAX** для **IN#=TRUE**), суммирует их и возвращает для суммы значение линейного функции с коэффициентом **K** и сдвигом **O**.

$$SCALE_X2 = O + K \cdot \sum_{i=1}^2 [IN_MIN_i + IN_i \cdot (IN_MAX_i - IN_MIN_i)]$$

Пример

```

VAR
  (* Входные булевые значения *)
  Input1: BOOL := TRUE;          (* Первое значение *)
  Input2: BOOL := FALSE;         (* Второе значение *)
  (* Коэффициенты масштабирования *)
  Gain: REAL := 2.5;             (* Коэффициент К *)
  Offset: REAL := 5.0;           (* Сдвиг О *)
  (* Границы масштабирования для IN1 *)
  In1_Min: REAL := 20.0;         (* Значение при IN1=FALSE *)
  In1_Max: REAL := 40.0;         (* Значение при IN1=TRUE *)
  (* Границы масштабирования для IN2 *)
  In2_Min: REAL := 30.0;         (* Значение при IN2=FALSE *)
  In2_Max: REAL := 40.0;         (* Значение при IN2=TRUE *)
  (* Результат *)
  Result: REAL;
END_VAR

(* Вызов функции SCALE_X2 *)
Result := OSCAT_BASIC.SCALE_X2(
  IN1 := Input1,
  IN2 := Input2,
  K := Gain,
  O := Offset,
  IN1_MIN := In1_Min,
  IN1_MAX := In1_Max,
  IN2_MIN := In2_Min,
  IN2_MAX := In2_Max
);

(*Результат 180.0*)
END_PROGRAM

```

22.31 SCALE_X4

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN1...IN4	BYTE	Исходное значение 1...4
	K	REAL	Коэффициент
	O	REAL	Сдвиг
Выходы	SCALE_X4	REAL	Рассчитанное значение.
Параметры	IN1_MIN...IN4_MIN	REAL	Отмасштабированное значение для IN1...IN4=FALSE
	IN1_MAX...IN4_MAX	REAL	Отмасштабированное значение для IN1...IN4=TRUE

Функция **SCALE_X4** преобразует входное булевское значение **IN#** в значение с плавающей точкой (**IN#_MIN** для **IN#=FALSE** и **IN#_MAX** для **IN#=TRUE**), суммирует их и возвращает для суммы значение линейного функции с коэффициентом **K** и сдвигом **O**.

$$SCALE_X4 = O + K \cdot \sum_{i=1}^4 [IN_MIN_i + IN_i \cdot (IN_MAX_i - IN_MIN_i)]$$

22.32 SCALE_X8

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN1...IN8	BYTE	Исходное значение 1...8
	K	REAL	Коэффициент
	O	REAL	Сдвиг

Тип модуля: функция	Переменная	Тип	Описание
Выходы	SCALE_X8	REAL	Рассчитанное значение
Параметры	IN1_MIN...IN8_MIN	REAL	Отмасштабированное значение для IN1...IN8=FALSE
	IN1_MAX...IN8_MAX	REAL	Отмасштабированное значение для IN1...IN8=TRUE

Функция **SCALE_X8** преобразует входное булевское значение **IN#** в значение с плавающей точкой (**IN#_MIN** для **IN#=FALSE** и **IN#_MAX** для **IN#=TRUE**), суммирует их и возвращает для суммы значение линейного функции с коэффициентом **K** и сдвигом **O**.

$$SCALE_X8 = O + K \cdot \sum_{i=1}^8 [IN_MIN_i + IN_i \cdot (IN_MAX_i - IN_MIN_i)]$$

22.33 SEL2_OF_3

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN1	REAL	Контролируемое значение 1
	IN2	REAL	Контролируемое значение 2
	IN3	REAL	Контролируемое значение 3
	D	REAL	Допустимое отклонение
Выходы	Y	REAL	Усредненное значение
	W	INT	Код состояния
	E	BOOL	Флаг ошибки

Функциональный блок **SEL2_OF_3** используется для контроля показаний датчиков по схеме «2 из 3». Если входные значения **IN1... IN3** отличаются друг от друга на величину, не превышающую **D**, то на выход **Y** поступает усредненное по трем сигналам значение, а выход **W=0**. Если одно из входных значений отличается от остальных на величину, превышающую **D**, то на выход **Y** поступает усредненное по двум другим сигналам значение, а на выходе **W** отображается номер входа с некорректным значением. Если все три входных значения отличаются на величину, превышающую **D**, то выход **Y** сохраняет свое последнее значение, выход **W=4**, а выход **E** принимает значение **TRUE**, сигнализируя о сбое в системе.

22.34 SEL2_OF_3B

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN1	BOOL	Контролируемый сигнал 1
	IN2	BOOL	Контролируемый сигнал 2
	IN3	BOOL	Контролируемый сигнал 3
	TD	TIME	Время задержки индикации ошибки
Выходы	Q	BOOL	Выходной сигнал
	W	BOOL	Флаг «ошибка»

Функциональный блок **SEL2_OF_3B** используется для контроля дискретных сигналов по схеме «2 из 3». Если входные значения **IN1... IN3** совпадают, то выход **Q** равен этому же значению. Если одно из входных значений отличается от остальных, то выход **Q** принимает значение, которое имеют два других входа, а выход **W** принимает значение **TRUE**, сигнализируя об ошибке. Вход **TD** определяет задержку между изменением значений входов и активацией выхода **W**.

22.35 SH

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	in	REAL	Контролируемое значение

Тип модуля: ФБ	Переменная	Тип	Описание
	CLK	BOOL	Сигнал сохранения данных
Выходы	out	REAL	Сохраненное значение
	trig	BOOL	Флаг «произошло сохранение»

Функциональный блок **SH** представляет собой элемент хранения. По переднему фронту на входе **CLK** выход **out** принимает и запоминает значение входа **in**. Выход **trig** принимает значение **TRUE** на один цикл при изменении значения на выходе **out**.

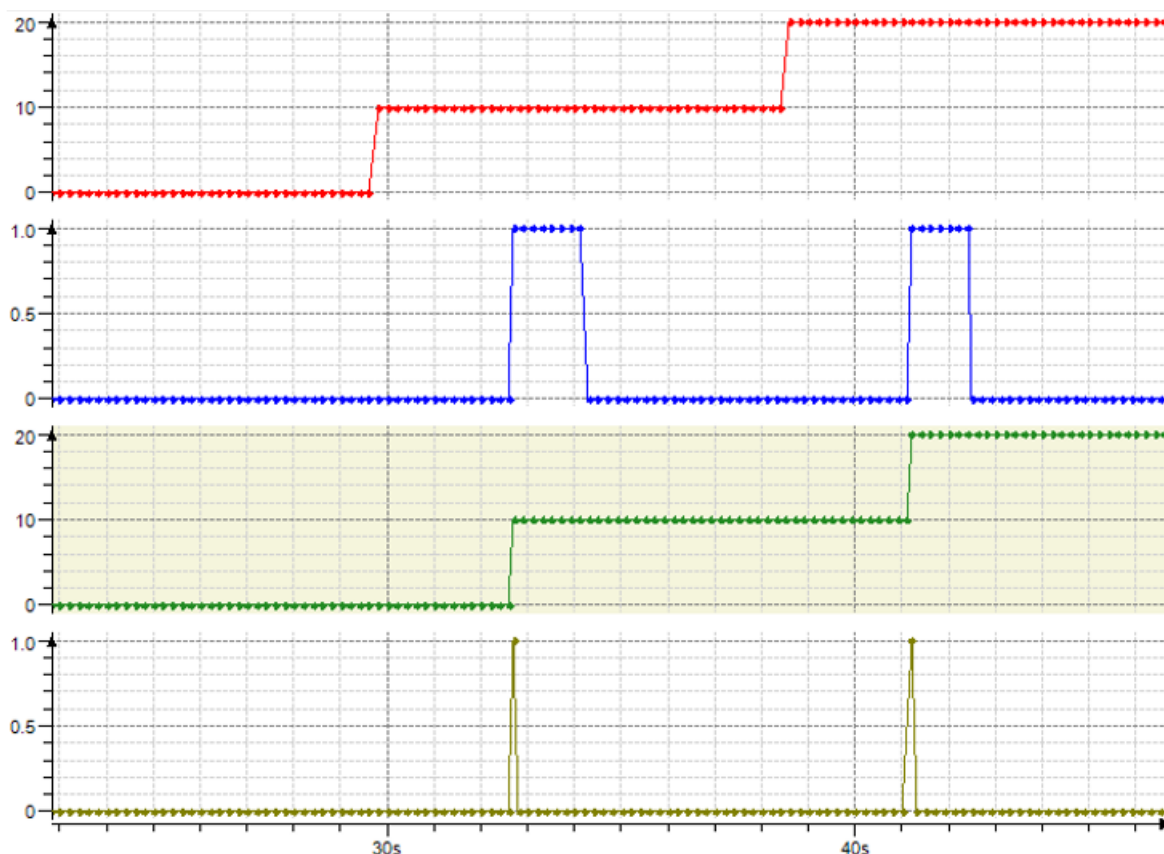


Рисунок 22.6 – Трассировка работы ФБ SH

22.36 SH_1

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	in	REAL	Контролируемое значение
	PT	TIME	Период сохранения
Выходы	out	REAL	Сохраненное значение
	trig	BOOL	Флаг «произошло сохранение»
Используемые модули	T_PLC_MS		

Функциональный блок **SH_1** представляет собой элемент хранения. Выход **out** принимает и запоминает значение входа **in** с заданным периодом **PT**. Выход **trig** принимает значение **TRUE** на один цикл при каждом сохранении данных.

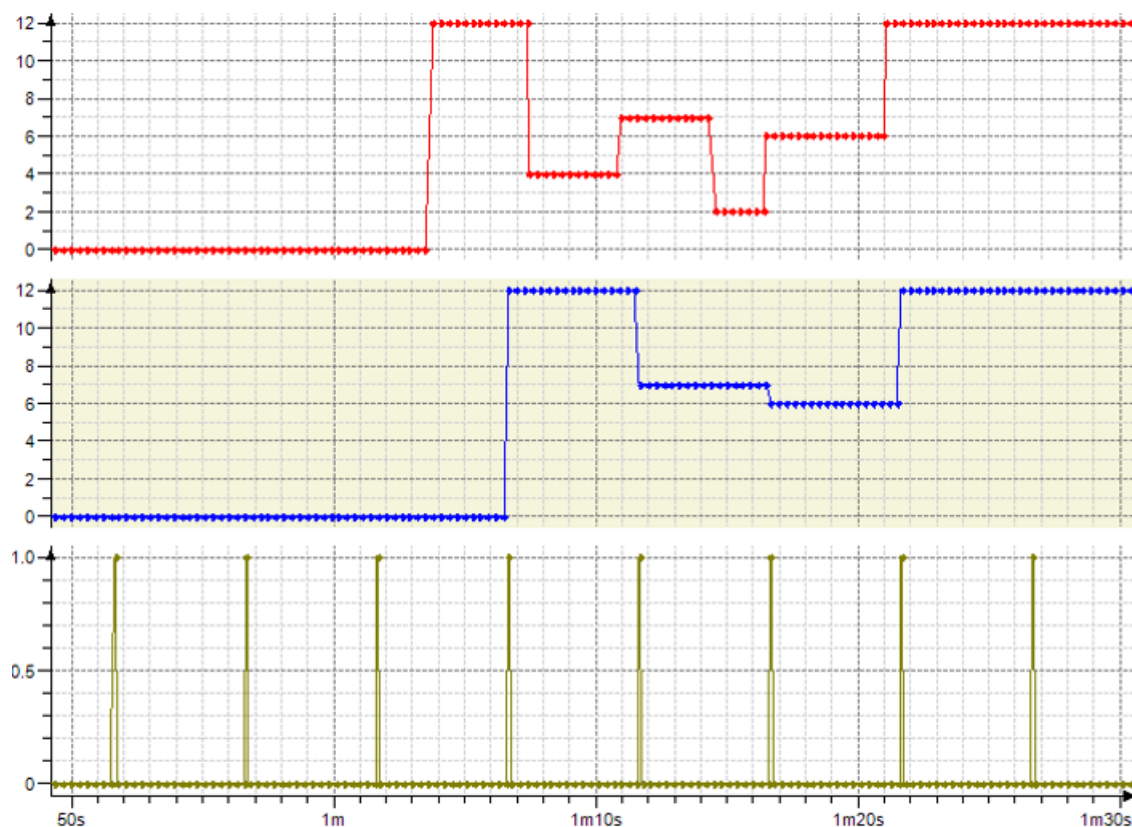


Рисунок 22.7 – Трассировка работы ФБ SH_1 (PT=T#5s)

22.37 SH_2

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	in	REAL	Контролируемое значение
	PT	TIME	Период сохранения
	N	INT	Число значений для статистики (1..16)
	disc	INT	Число отбрасываемых экстремумов
Выходы	out	REAL	Сохраненное значение
	trig	BOOL	Флаг «произошло сохранение»
	avg	REAL	Среднее значение
	high	REAL	Максимальное значение
	low	REAL	Минимальное значение
Используемые модули	T_PLC_MS, EVEN		

Функциональный блок **SH_2** представляет собой элемент хранения со сбором статистики. Выход **out** принимает и запоминает значение входа **in** с заданным периодом **PT**. Выход **trig** принимает значение **TRUE** на один цикл при каждом сохранении данных. Вход **N** определяет число срезов (до 16-ти) для сбора статистической информации – среднего (выход **avg**), максимального (выход **high**) и минимального значений (выход **low**). Вход **disc** позволяет определить число экстремумов, которые не будут учитываться при статистической обработке (это может быть полезно для фильтрации помех):

- **0** – в статистике учитываются все срезы;
- **1** – не учитывается наименьшее значение;
- **2** – не учитывается наибольшее значение;
- **3** – не учитываются два наименьших и одно наибольшее значение;
- **4** – не учитываются два наименьших и два наибольших значения;
- и т.д.

22.38 SH_T

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	in	REAL	Контролируемое значение
	E	BOOL	Режим сохранения значений
Выходы	out	REAL	Сохраненное значение

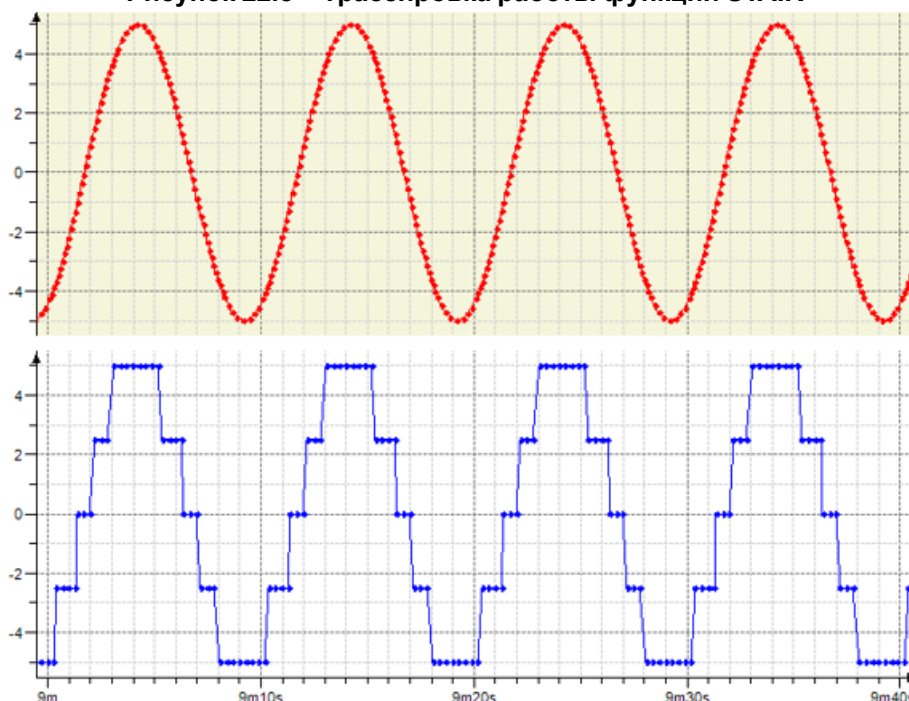
Функциональный блок **SH_T** представляет собой прозрачный элемент хранения. Пока вход **E** имеет значение **TRUE**, на выход **out** транслируется значение входа **in**. По заднему фронту на входе **E** выход **out** сохраняет свое последнее значение.

22.39 STAIR

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Входное значение
	D	REAL	Величина ступеньки
Выходы	STAIR	REAL	Ступенчатая функция

Функция **STAIR** возвращает значение ступенчатой функции для входного значения **X**. Вход **D** определяет величину ступеньки. Если **D=0**, то **STAIR=X**. Функция не производит фильтрацию входного значения; в случае ее необходимости следует использовать ФБ **STAIR2**, который учитывает гистерезис.

Рисунок 22.8 – Трассировка работы функции STAIR



22.40 STAIR2

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	X	REAL	Входное значение
	D	REAL	Величина ступеньки/значение гистерезиса
Выходы	Y	REAL	Ступенчатая функция
Используемые модули	FLOOR		

Функциональный блок **STAIR2** возвращает на выход **Y** значение ступенчатой функции для входного значения **X**. Вход **D** определяет величину ступеньки. Если **D=0**, то **Y=X**. **D** также определяет величину гистерезиса – если изменение входного значения не превышает **D**, то значение выхода не изменится. Таким образом осуществляется подавление шума исходного сигнала.

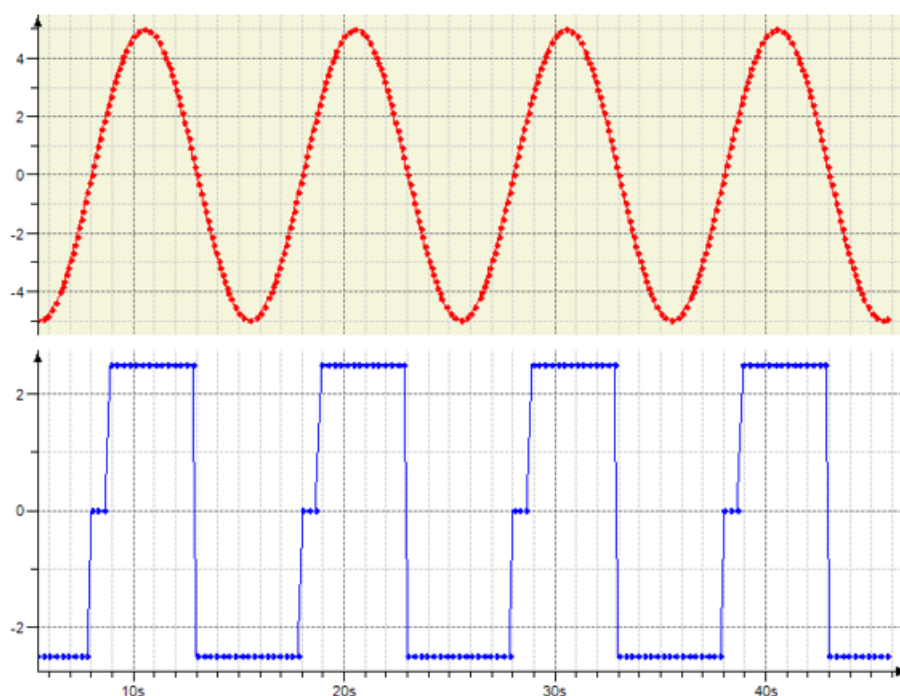


Рисунок 22.9 – Трассировка работы ФБ STAIR2

22.41 TREND

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	X	REAL	Входное значение
Выходы	Q	BOOL	Флаг “изменение значения”
	TU	BOOL	Флаг “увеличение значения”
	TD	BOOL	Флаг “уменьшение значения”
	D	REAL	Величина изменения

Функциональный блок **TREND** используется для контроля изменения значения входной переменной **X**. Если в текущем цикле значение **X** изменилось по сравнению с предыдущим, то выход **Q** принимает значение **TRUE**. Если значение увеличилось – то выход **TU** принимает значение **TRUE**, если уменьшилось – выход **TD** принимает значение **TRUE**. На выходе **D** отображается величина, на которую изменилось значение (со знаком) в текущем цикле. Если значение не изменилось по сравнению с предыдущим циклом, то выходы **Q**, **TU** и **TD** имеют значение **FALSE**, а **D=0**.

22.42 TREND_DW

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	X	DWORD	Входное значение
Выходы	Q	BOOL	Флаг “изменение значения”
	TU	BOOL	Флаг “увеличение значения”
	TD	BOOL	Флаг “уменьшение значения”
	D	DWORD	Величина изменения

Блок **TREND_DW** используется для контроля изменения значения входной переменной **X**. Принцип работы полностью соответствует ФБ **TREND**, единственным отличием является тип используемых переменных – **DWORD**.

22.43 WORD_TO_RANGE

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	WORD	Исходное значение

Тип модуля: функция	Переменная	Тип	Описание
	low	REAL	Нижний предел отмасштабированного значения
	high	REAL	Верхний предел отмасштабированного значения
Выходы	WORD_TO_RANGE	REAL	Отмасштабированное значение

Функция **WORD_TO_RANGE** линейно масштабирует входное целочисленное значение **X** из диапазона (**0...65535**) в значение с плавающей точкой из диапазона (**low...high**). См. также обратную функцию [RANGE_TO_WORD](#).

Пример

```

VAR
  (* Входное целочисленное значение WORD (0...65535) *)
  InputValue : WORD := 8192;          (* 8192 из диапазона 0...65535 *)
  (* Границы выходного диапазона (REAL) *)
  LowLimit   : REAL := -100.0;        (* Нижний предел *)
  HighLimit  : REAL := 100.0;         (* Верхний предел *)
  (* Результат масштабирования *)
  OutputValue : REAL;
END_VAR

(* Вызов функции WORD_TO_RANGE *)
OutputValue := OSCAT_BASIC.WORD_TO_RANGE(
  X      := InputValue,
  low    := LowLimit,
  high   := HighLimit
);
(* Результат: OutputValue = -75.0 *)
END_PROGRAM

```

23 Термометры сопротивления

23.1 MULTI_IN

Тип модуля: функция	Переменная	Тип	Описание
Входы	in_1	REAL	Значение 1
	in_2	REAL	Значение 2
	in_3	REAL	Значение 3
	Default	REAL	Значение по умолчанию
	in_min	REAL	Нижняя граница допустимых значений
	in_max	REAL	Верхняя граница допустимых значений
	mode	BYTE	Режим обработки значений
Выходы	MULTI_IN	REAL	Выходное значение

Функция **MULTI_IN** используется для обработки сигналов от датчиков **in_1...in_3** в режиме, определяемом входом **mode**. Входы **in_min** и **in_max** определяет нижний и верхний допустимый предел входных сигналов. Если значение не входит в заданный диапазон или совпадает с его границами, то в режимах **1-4** вместо него используется величина входа **default**, а в режимах **0** и **5-7** значение не обрабатывается. Описание режимов работы функции приведено в таблице:

Значение входа mode	Значение выхода MULTI_IN
0	Среднее арифметическое для in_1...in_3
1	in_1
2	in_2
3	in_3
4	default
5	Наименьшее из in_1...in_3
6	Наибольшее из in_1...in_3
7	Среднее из in_1...in_3
>7	0

Пример

```

VAR
  (* Исходные значения для обоих вызовов *)
  Value1 : REAL := 2.5;
  Value2 : REAL := 3.5;
  Value3 : REAL := 4.5;

  (* Результаты *)
  Result1 : REAL; (* Результат первой функции *)
  Result2 : REAL; (* Результат второй функции *)

  Default1 : REAL := 3.0;
  Default2 : REAL := 0.0;

  Min1 : REAL := 3.0;
  Max1 : REAL := 5.0;

  Min2 : REAL := 0.0;
  Max2 : REAL := 100.0;

  Mode1 : BYTE := 0;
  Mode2 : BYTE := 7;
END_VAR

Result1 := OSCAT_BASIC.MULTI_IN(
  in_1 := Value1,
  in_2 := Value2,
  in_3 := Value3,
  Default := Default1,
  in_min := Min1,
  in_max := Max1,
  mode := Mode1
);

(* Результат: Result1 = 4, т.к. значение in_1 выходит за диапазон (in_min...in_max),
поэтому функция возвращает среднее арифметическое для значений in_2 и in_3*)

Result2 := OSCAT_BASIC.MULTI_IN(
  in_1 := Value1,
  in_2 := Value2,
  in_3 := Value3,
  Default := Default2,
  in_min := Min2,
  in_max := Max2,
  mode := Mode2
);

(* Результат: Result2 = 3.5 - среднее из трех значений*)
END_PROGRAM

```

23.2 RES_NI

Тип модуля: функция	Переменная	Тип	Описание
Входы	T	REAL	Измеренная температура, °C
	R0	REAL	Номинальное сопротивление TC, Ом
Выходы	RES_NI	REAL	Сопротивление TC, Ом

Функция **RES_NI** возвращает значение сопротивления **никелевого** термометра сопротивления (ТС) с номинальным сопротивлением **R0** для измеренной температуры **T**, вычисленное по формуле:

$$RES_NI = R0 + A \cdot T + B \cdot T^2 + C \cdot T^4$$

, где

$$A = 0.5485 \quad B = 0.665 \cdot 10^{-3} \quad C = 2.805 \cdot 10^{-9}$$

Приемлемая точность значений функции ограничена диапазоном температур –60...+180 °C.

См. также обратную функцию [TEMP_NI](#)

Пример

```
VAR
  (*Измеренная температура, °C *)
  Temperature : REAL := 50.0;          (* 50 °C *)

  (*Номинальное сопротивление ТС при 0 °C, Ом *)
  R0_Nominal : REAL := 100.0;          (* TC 100 Ом *)

  (*Результат - сопротивление ТС, Ом *)
  Resistance : REAL;
END_VAR

(*Вызов функции RES_NI *)
Resistance := OSCAT_BASIC.RES_NI(
  T := Temperature,
  R0 := R0_Nominal
);

(*Результат ≈ 129.11 Ом*)
END_PROGRAM
```

23.3 RES_NTC

Тип модуля: функция	Переменная	Тип	Описание
Входы	T	REAL	Измеренная температура, °C
	RN	REAL	Номинальное сопротивление ТС, Ом при T=25 °C
	B	REAL	Коэффициент температурной чувствительности, К
Выходы	RES_NTC	REAL	Сопротивление NTC-термистора, Ом

Функция **RES_NTC** возвращает значение сопротивления NTC-термистора с номинальным сопротивлением **RN** (для T=25 °C) и коэффициентом температурной чувствительности **B** для измеренной температуры **T**, вычисленное по формуле:

$$RES_NTC = RN \cdot e^{(B \cdot (1/T - 1/(25+273.15)))}$$

Приемлемая точность значений функции ограничена диапазоном температур 0...100 °C.

См. также обратную функцию [TEMP_NTC](#)

Пример

```

VAR
  (* Измеренная температура, °C *)
  Temperature : REAL := 30.0;          (* 30 °C *)

  (* Номинальное сопротивление при 25 °C, Ом *)
  RN_Nominal : REAL := 10000.0;        (* 10 кОм при 25 °C *)

  (* Коэффициент температурной чувствительности, К *)
  B_Coeff : REAL := 3950.0;            (* B = 3950 K *)

  (* Результат - сопротивление NTC-термистора, Ом *)
  Resistance : REAL;
END_VAR

(* Вызов функции RES_NTC *)
Resistance := OSCAT_BASIC.RES_NTC(
  T := Temperature,
  RN := RN_Nominal,
  B := B_Coeff
);

(*Результат ≈ 8037 Ом*)
END_PROGRAM

```

23.4 RES_PT

Тип модуля: функция	Переменная	Тип	Описание
Входы	T	REAL	Измеренная температура, °C
	R0	REAL	Номинальное сопротивление TC, Ом
Выходы	RES_PT	REAL	Сопротивление TC, Ом

Функция **RES_PT** возвращает значение сопротивления платинового термометра сопротивления (ТС) с номинальным сопротивлением **R0** для измеренной температуры **T**, вычисленное по формуле:

$$\begin{cases} \text{RES_PT} = R0 \cdot (1 + A \cdot T + B \cdot T^2) & \text{при } T < 0 \\ \text{RES_PT} = R0 \cdot (1 + A \cdot T + B \cdot T^2 + C \cdot (T - 100) \cdot T^3) & \text{при } T > 0 \end{cases}$$

$$A = 3.90802 \cdot 10^{-3}$$

$$B = -5.802 \cdot 10^{-7}$$

$$C = -4.27350 \cdot 10^{-12}$$

Приемлемая точность значений функции ограничена диапазоном температур –200...+850 °C.

См. также обратную функцию [TEMP_PT](#)

Пример

```

VAR
  (* Измеренная температура, °C *)
  Temperature : REAL := 50.0;      (* 50 °C *)

  (* Номинальное сопротивление ТС при 0 °C, Ом *)
  R0_Nominal : REAL := 100.0;      (* Pt100 *)

  (* Результат - сопротивление ТС, Ом *)
  Resistance : REAL;
END_VAR

(* Вызов функции RES_PT *)
Resistance := OSCAT_BASIC.RES_PT(
  T := Temperature,
  R0 := R0_Nominal
);

(*Результат ≈ 119.40 Ом*)
END_PROGRAM

```

23.5 RES_SI

Тип модуля: функция	Переменная	Тип	Описание
Входы	T	REAL	Измеренная температура, °C
	RS	REAL	Номинальное сопротивление ТС, Ом
	TS	REAL	Температура при номинальном сопротивлении, °C
Выходы	RES_SI	REAL	Сопротивление ТС, Ом

Функция **RES_SI** возвращает значение сопротивления **кремниевого** термометра сопротивления (ТС) с номинальным сопротивлением **RS** при температуре **TS** для измеренной температуры **T**, вычисленное по формуле:

$$RES_SI = RS + A \cdot (T - TS) + B \cdot (T - TS)^2$$

, где

$$A = 7.64 \cdot 10^{-3} \quad B = 1.66 \cdot 10^{-5}$$

Приемлемая точность значений функции ограничена диапазоном температур –50...+150 °C.

См. также обратную функцию [TEMP_SI](#)

Пример

```
VAR
  (* Измеренная температура, °C *)
  Temperature : REAL := 50.0;          (* 50 °C *)

  (* Номинальное сопротивление при температуре TS, Ом *)
  RS_Nominal : REAL := 1000.0;         (* 1000 Ом при 25 °C *)

  (* Температура при номинальном сопротивлении, °C *)
  TS_Nominal : REAL := 25.0;          (* 25 °C *)

  (* Результат - сопротивление TC, Ом *)
  Resistance : REAL;
END_VAR

(* Вызов функции RES_SI *)
Resistance := OSCAT_BASIC.RES_SI(
  T := Temperature,
  RS := RS_Nominal,
  TS := TS_Nominal
);

(*Результат ≈ 1201.375 Ом*)
END_PROGRAM
```

23.6 SENSOR_INT

Тип модуля: функция	Переменная	Тип	Описание
Входы	Voltage	REAL	Напряжение, В
	Current	REAL	Сила тока, А
	RP	REAL	Параллельное паразитное сопротивление, Ом
	RS	REAL	Последовательное паразитное сопротивление, Ом
Выходы	SENSOR_INT	REAL	Сопротивление TC, Ом

Функция **SENSOR_INT** вычисляет значение сопротивления для термометра сопротивления (ТС) на основании измеренных значений напряжения **Voltage** и силы тока **Current** с учетом паразитных сопротивлений **RP** и **RS**. Схема измерения приведена на рисунке ниже. Вычисленное значение сопротивления может быть преобразовано в значение температуры с помощью функций TEMP_хх.

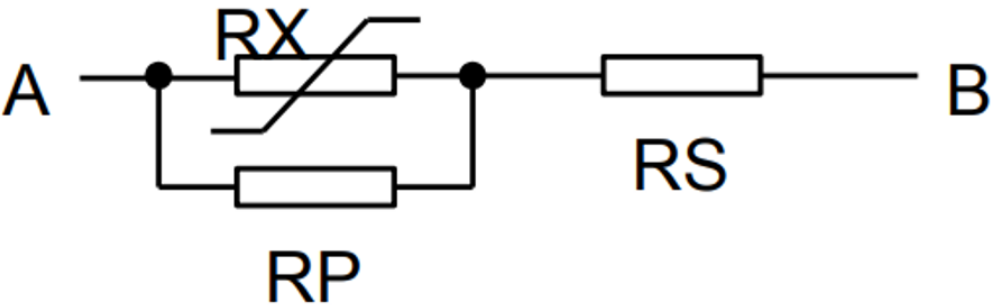


Рисунок 23.1 – Схема измерения для функции SENSOR_INT

23.7 TEMP_NI

Тип модуля: функция	Переменная	Тип	Описание
Входы	Res	REAL	Измеренное сопротивление, Ом
	R0	REAL	Номинальное сопротивление TC, Ом
Выходы	TEMP_NI	REAL	Температура TC, °C

Функция **TEMP_NI** возвращает значение температуры никелевого термометра сопротивления (TC) с номинальным сопротивлением **R0** для измеренного сопротивления **R**. См. также обратную функцию **RES_NI**.

Приемлемая точность значений функции ограничена диапазоном температур –60...+180 °C.

Пример

```

VAR      (* Измеренное сопротивление TC, Ом *)
Resistance : REAL := 129.1;      (* 129.1 Ом *)

      (* Номинальное сопротивление TC при 0 °C, Ом *)
R0_Nominal : REAL := 100.0;      (* TC 100 Ом *)

      (* Результат - температура, °C *)
Temperature : REAL;
END_VAR

(* Вызов функции TEMP_NI *)
Temperature := OSCAT_BASIC.TEMP_NI(
  Res := Resistance,
  R0 := R0_Nominal
);

(*Результат ≈ 50 °C*)
END_PROGRAM

```

23.8 TEMP_NTC

Тип модуля: функция	Переменная	Тип	Описание
Входы	Res	REAL	Измеренное сопротивление, Ом
	RN	REAL	Номинальное сопротивление TC, Ом при T=25 °C
	B	REAL	Коэффициент температурной чувствительности, К
Выходы	TEMP_NTC	REAL	Температура NTC-термистора, °C

Функция **TEMP_NTC** возвращает значение температуры NTC-термистора с номинальным сопротивлением **RN** (для T=25 °C) и коэффициентом температурной чувствительности **B** для измеренного сопротивления **Res**. См. также обратную функцию **RES_NTC**.

Приемлемая точность значений функции ограничена диапазоном температур 0...100 °C.

Пример

```

VAR      (* Измеренное сопротивление NTC-термистора, Ом *)
Resistance : REAL := 8037.0;      (* 8.037 кОм *)

      (* Номинальное сопротивление при 25 °С, Ом *)
RN_Nominal : REAL := 10000.0;    (* 10 кОм при 25 °С *)

      (* Коэффициент температурной чувствительности, К *)
B_Coeff : REAL := 3950.0;        (* B = 3950 К *)

      (* Результат - температура, °С *)
Temperature : REAL;
END_VAR

(* Вызов функции TEMP_NTC *)
Temperature := OSCAT_BASIC.TEMP_NTC(
  Res := Resistance,
  RN := RN_Nominal,
  B := B_Coeff
);

(*Результат ≈ 30.0 °С*)
END_PROGRAM

```

23.9 TEMP_PT

Тип модуля: функция	Переменная	Тип	Описание
Входы	Res	REAL	Измеренное сопротивление, Ом.
	R0	REAL	Номинальное сопротивление ТС, Ом.
Выходы	TEMP_PT	REAL	Температура ТС, °С.
Используемые модули	RES_PT		

Функция **TEMP_PT** возвращает значение температуры платинового термометра сопротивления (ТС) с номинальным сопротивлением **R0** для измеренного сопротивления **R**. См. также обратную функцию [RES_PT](#).

Приемлемая точность значений функции ограничена диапазоном температур –200...+850 °С.

Пример

```

VAR
  (* Измеренное сопротивление ТС, Ом *)
  Resistance : REAL := 119.4;    (* 119.4 Ом *)

  (* Номинальное сопротивление ТС при 0 °С, Ом *)
  R0_Nominal : REAL := 100.0;    (* Pt100 *)

  (* Результат - температура, °С *)
  Temperature : REAL;
END_VAR

(* Вызов функции TEMP_PT *)
Temperature := OSCAT_BASIC.TEMP_PT(
  Res := Resistance,
  R0 := R0_Nominal
);

(*Результат: Temperature ≈ 50.0 °С*)
END_PROGRAM

```

23.10 TEMP_SI

Тип модуля: функция	Переменная	Тип	Описание
Входы	Res	REAL	Измеренное сопротивление, Ом
	RS	REAL	Номинальное сопротивление TC, Ом
	TS	REAL	Температура при номинальном сопротивлении, °C
Выходы	TEMP_SI	REAL	Температура TC, °C

Функция **TEMP_SI** возвращает значение температуры кремниевого термометра сопротивления (TC) с номинальным сопротивлением **RS** при температуре **TS** для измеренного сопротивления **R**. См. также обратную функцию **RES_SI**.

Приемлемая точность значений функции ограничена диапазоном температур –50...+150 °C.

Пример

```

VAR
  (* Измеренное сопротивление, Ом *)
  Resistance : REAL := 1200.0;          (* 1200 Ом *)

  (* Номинальное сопротивление при температуре TS, Ом *)
  RS_Nominal : REAL := 1000.0;          (* 1000 Ом при 25 °C *)

  (* Температура при номинальном сопротивлении, °C *)
  TS_Nominal : REAL := 25.0;            (* 25 °C *)

  (* Результат - температура, °C *)
  Temperature : REAL;
END_VAR

(* Вызов функции TEMP_SI *)
Temperature := OSCAT_BASIC.TEMP_SI(
  Res := Resistance,
  RS := RS_Nominal,
  TS := TS_Nominal
);

(*Результат ≈ 50 °C*)
END_PROGRAM

```

24 Модули измерения и отсчета времени

24.1 ALARM_2

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	X	REAL	Контролируемое значение
	LO_1	REAL	Нижний предел 1
	HI_1	REAL	Верхний предел 1
	LO_2	REAL	Нижний предел 2
	HI_2	REAL	Верхний предел 2
	HYS	REAL	Значение гистерезиса
Выходы	Q1_LO	BOOL	Флаг «значение меньше нижнего предела 1»
	Q1_HI	BOOL	Флаг «значение больше верхнего предела 1»
	Q2_LO	BOOL	Флаг «значение меньше нижнего предела 2»
	Q2_HI	BOOL	Флаг «значение больше верхнего предела 2»

Функциональный блок **ALARM_2** проверяет входное значение **X** на принадлежность диапазонам **[LO_1...HI_1]** и **[LO_2...HI_2]**. Вход **HYS** определяет величину гистерезиса. Выходы блока становятся активными в следующих случаях:

$$\left\{ \begin{array}{l} Q1_LO = \text{TRUE} \text{ если } X < LO_1 \pm \frac{HYS}{2} \\ Q1_HI = \text{TRUE} \text{ если } X > HI_1 \pm \frac{HYS}{2} \\ Q2_LO = \text{TRUE} \text{ если } X < LO_2 \pm \frac{HYS}{2} \\ Q2_HI = \text{TRUE} \text{ если } X > HI_2 \pm \frac{HYS}{2} \end{array} \right.$$

Обычно с помощью диапазона **[LO_1...HI_1]** задаются пороговые значения для сигнала тревоги, с помощью **[LO_2...HI_2]** – для сигнала аварии.

Пример


```

VAR
  (* Экземпляр функционального блока ALARM_2 *)
  Alarm : OSCAT_BASIC.ALARM_2;

  (* Контролируемое значение (температура) *)
  Temperature : REAL := 85.0;      (* 85 °C *)

  (* Настройки пределов *)
  LO_1 : REAL := 60.0;              (* Нижний предел 1 (тревога) *)
  HI_1 : REAL := 80.0;              (* Верхний предел 1 (тревога) *)
  LO_2 : REAL := 50.0;              (* Нижний предел 2 (авария) *)
  HI_2 : REAL := 90.0;              (* Верхний предел 2 (авария) *)
  HYS : REAL := 2.0;                (* Гистерезис 2 °C *)

  (* Выходные флаги *)
  AlarmLO1 : BOOL;                  (* Менше нижнего предела 1 *)
  AlarmHI1 : BOOL;                  (* Больше верхнего предела 1 *)
  AlarmLO2 : BOOL;                  (* Менше нижнего предела 2 *)
  AlarmHI2 : BOOL;                  (* Больше верхнего предела 2 *)

  (* Состояние системы *)
  StatusMessage : STRING[50];
END_VAR

(* Настройка параметров блока *)
Alarm(
  LO_1 := LO_1,
  HI_1 := HI_1,
  LO_2 := LO_2,
  HI_2 := HI_2,
  HYS := HYS
);

(* Циклический вызов блока с текущим значением *)
Alarm(X := Temperature);

(* Чтение выходных флагов *)
AlarmLO1 := Alarm.Q1_LO;
AlarmHI1 := Alarm.Q1_HI;
AlarmLO2 := Alarm.Q2_LO;
AlarmHI2 := Alarm.Q2_HI;

(* Логика обработки *)
IF AlarmHI2 THEN
  StatusMessage := 'АВАРИЯ: превышение верхнего предела 90 °C!';
ELSIF AlarmLO2 THEN
  StatusMessage := 'АВАРИЯ: падение ниже нижнего предела 50 °C!';
ELSIF AlarmHI1 THEN
  StatusMessage := 'ПРЕДУПРЕЖДЕНИЕ: температура выше 80 °C';
ELSIF AlarmLO1 THEN
  StatusMessage := 'ПРЕДУПРЕЖДЕНИЕ: температура ниже 60 °C';
ELSE
  StatusMessage := 'Нормальный режим';
END_IF;

(*Результат при Temperature = 85 °C:
- AlarmHI1 = TRUE (85 > 80)
- AlarmHI2 = FALSE (85 < 90)
- StatusMessage = 'ПРЕДУПРЕЖДЕНИЕ: температура выше 80 °C'*)
END_PROGRAM (*Объявление конца программы*)

```

24.2 BAR_GRAPH

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	X	REAL	Контролируемое значение
	rst	REAL	Сигнал сброса тревоги
Выходы	LOW	BOOL	Флаг «значение меньше нижнего предела»
	Q1...Q6	BOOL	Флаги «значение в зоне 1...6»
	HIGH	BOOL	Флаг «значение больше верхнего предела»
	Alarm	BOOL	Флаг «тревога»
	Status	BYTE	ESR-код
Параметры	trigger_Low	REAL	Нижний предел
	trigger_High	REAL	Верхний предел
	Alarm_low	BOOL	Режим «обработка нижнего предела тревоги»
	Alarm_high	BOOL	Режим «обработка верхнего предела тревоги»
	log_scale	BOOL	Режим «логарифмирование интервалов»

Функциональный блок **BAR_GRAPH** проверяет входное значение **X** на принадлежность диапазону [**trigger_Low**... **trigger_High**]. Выход **LOW** принимает значение **TRUE**, если значение **X** меньше нижней границы диапазона, выход **HIGH** – если больше верхней границы. Если параметры **Alarm_low** и **Alarm_high** имеют значение **TRUE**, то при выходе значения **X** за границы допустимого диапазона выход **Alarm** принимает значение **TRUE**. Выход **Alarm** остается активным до тех пор, пока не будет сброшен с помощью сигнала по переднему фронту на входе **rst**.

Если значение **X** находится в пределах диапазона, то активен один из выходов **Q1...Q6**, характеризующий интервал, в котором находится значение. По умолчанию интервалы являются равными:

$$\begin{cases} Q1 = \text{TRUE} \text{ если } \text{trigger_Low} < X < \text{trigger_Low} + 1 \cdot \frac{\text{trigger_High} - \text{trigger_Low}}{6} \\ Q2 = \text{TRUE} \text{ если } \text{trigger_Low} < X < \text{trigger_Low} + 2 \cdot \frac{\text{trigger_High} - \text{trigger_Low}}{6} \\ \dots \\ Q6 = \text{TRUE} \text{ если } \text{trigger_Low} < X < \text{trigger_Low} + 6 \cdot \frac{\text{trigger_High} - \text{trigger_Low}}{6} \end{cases}$$

Если параметр **log_scale** имеет значение **TRUE**, то размеры интервалов связаны через логарифм:

$$\begin{cases} Q1 = \text{TRUE} \text{ если } \text{trigger_Low} < X < \text{trigger_Low} \cdot \frac{1}{6} \cdot \left[e^{\ln\left(\frac{\text{trigger_High}}{\text{trigger_Low}}\right)} \right]^1 \\ Q2 = \text{TRUE} \text{ если } \text{trigger_Low} < X < \text{trigger_Low} \cdot \frac{1}{6} \cdot \left[e^{\ln\left(\frac{\text{trigger_High}}{\text{trigger_Low}}\right)} \right]^2 \\ \dots \\ Q6 = \text{TRUE} \text{ если } \text{trigger_Low} < X < \text{trigger_Low} \cdot \frac{1}{6} \cdot \left[e^{\ln\left(\frac{\text{trigger_High}}{\text{trigger_Low}}\right)} \right]^6 \end{cases}$$

Выход **Status** определяет состояние блока и совместим с [ESR-модулями](#):

Значение выхода Status	Описание
110	Значение X находится в диапазоне [trigger_Low... trigger_High]

111	Значение X меньше trigger_Low, выход LOW равен TRUE
112	Значение X больше trigger_High выход HIGH равен TRUE
1	Значение X меньше trigger_Low, выход ALARM равен TRUE
2	Значение X больше trigger_High выход ALARM равен TRUE

24.3 CALIBRATE

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	X	REAL	Сигнал датчика
	CO	BOOL	Сигнал датчика для нижнего предела
	CS	BOOL	Сигнал датчика для верхнего предела
Выходы	Y	REAL	Измеренное значение
Параметры	Y_Scale	REAL	Верхний предел измеренного значения
	Y_Offset	REAL	Нижний предел измеренного значения

Функциональный блок **CALIBRATE** используется для калибровки сигнала аналогового датчика **X**. Параметры **Y_Offset** и **Y_Scale** определяют нижний и верхний предел измеряемого значения. По переднему фронту на входе **CO** в блок записывается измеренное значение для нижнего предела, по переднему фронту на входе **CS** – для верхнего. После этого значение входа **X** будет линейно масштабироваться и подаваться на выход **Y** с учетом рассчитанных коэффициентов. Коэффициенты являются энергонезависимыми (**VAR RETAIN**).

Поясним вышесказанное на примере. Предположим, мы получаем от датчика температуры сигнал **4-20 мА**, который соответствует диапазону **0-100 °С**. Зададим **Y_Offset=0** и **Y_Scale=100**. Поместим датчик в среду с температурой **0 °С** (значение температуры должно быть измерено с помощью эталонного термометра) и подадим импульс по переднему фронту на входе **CO**. Блок вычислит значение смещения сигнала. Нагреем датчик до **100 °С** и подадим импульс по переднему фронту на входе **CS**. Блок вычислит коэффициент масштабирования для входного сигнала. В случае, если датчик изначально откалиброван, смещение должно составить

$$\text{Offset} = Y_OFFSET - x = 0 - 4 = -4$$

а коэффициент масштабирования –

$$\text{Scale} = \frac{Y_SCALE}{X + \text{Offset}} = \frac{100}{20 + (-4)} = 6.5$$

Теперь если подать на вход **X** измеренное датчиком значение (например, **12 мА**), то на выходе **Y** будет получено значение температуры – **50 °С**. Поскольку в реальной жизни по мере эксплуатации характеристики датчиков могут изменяться, использование данного блока позволит поддерживать точность измеренного значения на приемлемом уровне.

24.4 CYCLE_TIME

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	RST	BOOL	Сигнал сброса блока
Выходы	ct_min	TIME	Минимальное время цикла
	ct_max	TIME	Максимальное время цикла
	ct_last	TIME	Последнее измеренное время цикла

Тип модуля: ФБ	Переменная	Тип	Описание
	sysstime	TIME	Время, прошедшее со старта ПЛК
	sysdays	INT	Число дней, прошедшее со старта ПЛК
	cycles	DWORD	Число циклов ПЛК с момента старта
Используемые модули	T_PLC_MS		

Функциональный блок **CYCLE_TIME** используется для получения статистической информации о времени цикла ПЛК. На выходы **ct_min**, **ct_max** и **ct_last** подается минимальное, максимальное и последнее измеренное время цикла. Выходы **sysstime** и **sysdays** содержат информацию о времени и числе дней, прошедших со старта ПЛК. Выход **cycles** содержит число циклов ПЛК, прошедших с момента его старта. По переднему фронту на входе **RST** выходы блока обнуляются и отсчет начинается заново.

24.5 DT_SIMU

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	START	DT	Начальное время часов (точка отсчета)
	SPEED	REAL	Коэффициент масштаба времени
Выходы	DTS	DT	Текущее время
Используемые модули	T_PLC_MS		

Функциональный блок **DT_SIMU** представляет собой программный модуль **RTC** с задаваемым масштабом времени. Вход **START** определяет начальное время, которое будет являться точкой отсчета. Выход **DTS** отображает текущее время. Вход **SPEED** определяет масштаб времени относительно стандартного. Например, если **SPEED=5.0**, то каждую секунду к текущему значению времени на выходе **DTS** будет прибавляться 5 секунд – т.е. время будет идти в 5 раз быстрее, чем в реальном мире. Это может быть полезным при отладке проектов, связанных с медленно протекающими процессами.

24.6 FLOW_METER

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	VX	REAL	Измеренное значение расхода (ед./час)
	E	BOOL	Сигнал управления блоком
	RST	BOOL	Сигнал сброса блока
Выходы	F	REAL	Мгновенное значение расхода (ед./час)
Входы - выходы	X	REAL	Суммарное значение расхода (дробная часть)
	Y	UDINT	Суммарное значение расхода (целая часть)
Параметры	PULSE_MODE	BOOL	Режим работы блока
	UPDATE_TIME	TIME	Время опроса входа VX
Используемые модули	INTEGRATE, FLOOR, T_PLC_MS		

Функциональный блок **FLOW_METER** используется для определения суммарного расхода. На вход **VX** подается измеренное значение расхода. Параметр **PULSE_MODE** определяет режим работы блока. Если **PULSE_MODE** имеет значение **FALSE**, то расходомер работает в режиме интегрирования и пока вход **E** имеет значение **TRUE**, значение входа **VX** интерпретируется как расход за час, а на входах-выходах **Y** и **X** отображается целая и дробная часть суммарного расхода за прошедшее с начала работы блока время. На выходе **F** отображается текущее значение расхода. Параметр **UPDATE_TIME** частоту опроса входа **VX**. Если **PULSE_MODE** имеет значение **TRUE**, то расходомер работает в импульсном режиме и при импульсе по переднему фронту входа **E** прибавляет значение **VX** к суммарному значению расхода **Y** и **X** (целая часть/

дробная часть). По переднему фронту входа **RST** происходит сброс памяти блока, после чего отсчет начинается заново.

Суммарное значение расхода представлено в виде двух величин – **X** и **Y**. Если суммарный расход = 210.115 л/ч, то **Y**=210 (целая часть), **X**=0.115 (дробная часть).

Поясним вышесказанное несколькими примерами.

1. Пусть **VX**=3600 л/ч, **PULSE_MODE**=FALSE, **UPDATE_TIME**=T#100ms.

Расходомер работает в режиме интегратора, вход **VX** опрашивается каждые 100 мс (т.е. если значение входа изменится, то блок начнет использовать его вместо предыдущего с задержкой до 100 мс). Значение **Y** каждую секунду увеличивается на +1 (3600 л/ч = 1 л/с). Значение **X** содержит дробную часть значения и изменяется с частотой цикла ПЛК.

2. Пусть **VX**=10 л/ч, **PULSE_MODE**=TRUE, **UPDATE_TIME**=T#100ms

Расходомер работает в импульсном режиме, вход **VX** опрашивается каждые 100 мс (т.е. если значение входа изменится, то блок начнет использовать его вместо предыдущего с задержкой до 100 мс). Значение **Y** увеличивается на +10 по переднему фронту на входе **E**.

24.7 M_D

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	start	BOOL	Сигнала начала отсчета
	stop	BOOL	Сигнал окончания отсчета
	tmax	TIME	Максимальное время отсчета
	RST	BOOL	Сигнал сброса блока
Выходы	PT	TIME	Время между сигналом старта и стопа
	ET	TIME	Прошедшее время
	run	BOOL	Флаг «идет отсчет времени»
Используемые модули	T_PLC_MS		

Функциональный блок **M_D** представляет собой модуль измерения времени между двумя импульсами переднего фронта. По переднему фронту на входе **start** блок начинает отсчет времени, при этом на выходе **ET** отображается время, прошедшее с начала отсчета, а выход **RUN** имеет значение **TRUE**. По переднему фронту на входе **stop** отсчет времени прекращается, при этом на выходе **PT** отображается время окончания отсчета (совпадающее с текущим значением выхода **ET**), а выход **RUN** принимает значение **FALSE**. Вход **tmax** используется для ограничения продолжительности отсчета – если **ET** > **tmax**, то блок прекращает работу, а его выходы обнуляются. По переднему фронту на входе **RST** происходит принудительная остановка блока с обнулением выходов.

Обратите внимание, старт начала отсчета возможен только в том случае, если входы **stop** и **RST** имеют значение **FALSE**.

24.8 M_T

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	BOOL	Контролируемый сигнал
	tmax	TIME	Максимальное время отсчета
	RST	BOOL	Сигнал сброса блока
Выходы	PT	TIME	Время, которое сигнал был в TRUE
	ET	TIME	Прошедшее время
Используемые модули	T_PLC_MS		

Функциональный блок **M_T** представляет собой модуль измерения времени между импульсами переднего и заднего фронта. По переднему фронту на входе **IN** блок начинает отсчет времени, при этом на выходе **ET** отображается время, прошедшее с начала отсчета. По заднему фронту на входе **IN** отсчет времени прекращается, при этом на выходе **PT** отображается время окончания отсчета (совпадающее с текущим значением выхода **ET**). Таким образом, блок измеряет время, в течение которого вход **IN** находился в

значении **TRUE**. Вход **tmax** используется для ограничения продолжительности отсчета – если **ET > tmax**, то блок прекращает работу, а его выходы обнуляются. По переднему фронту на входе **RST** происходит принудительная остановка блока с обнулением выходов.

Обратите внимание, старт начала отсчета возможен только в том случае, если вход **RST** имеет значение **FALSE**.

24.9 M_TX

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	BOOL	Контролируемый сигнал
	tmax	TIME	Максимальное время отсчета
	RST	BOOL	Сигнал сброса блока
Выходы	TH	TIME	Время, которое IN был в TRUE
	TL	TIME	Время, которое IN был в FALSE
	DC	REAL	Процент времени, которое IN был в TRUE (0...1)
	F	REAL	Частота изменения сигнала на входе IN
	ET	TIME	Прошедшее время
Используемые модули	T_PLC_MS		

Функциональный блок **M_TX** представляет собой модуль сбора статистики о состоянии входа **IN**. По переднему фронту на входе **IN** блок начинает отсчет времени, при этом на выходе **ET** отображается время, прошедшее с начала отсчета. По заднему фронту на входе **IN** **отсчет не прекращается**; для остановки работы блока необходим импульс по переднему фронту на входе **RST**, который также обнулит его выходы. На выходах **TH** и **TL** отображается время, которое сигнал **IN** находится в состоянии **TRUE** и **FALSE** соответственно. Выход **DC** сообщает, какой процент времени сигнал находился в состоянии **TRUE** (например, если **DC=0.4**, то в течение 40% времени вход **IN** имел значение **TRUE**, в течение 60% - значение **FALSE**). На выходе **F** отображается частота изменения входа **IN**. Вход **tmax** используется для ограничения продолжительности отсчета – если **ET > tmax**, то блок прекращает работу, а его выходы обнуляются.

Обратите внимание, обновление значений на выходах **TH**, **TL**, **DC** и **F** происходит после очередного импульса на входе **IN**. В первый раз значения на этих выходах появятся после **второго** импульса по переднему фронту на входе **IN** с начала работы блока.

24.10 METER

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	M1	REAL	Контролируемое значение 1
	M2	REAL	Контролируемое значение 2
	I1	BOOL	Режим контроля значения 1
	I2	BOOL	Режим контроля значения 2
	D	REAL	Делитель для выхода
	RST	BOOL	Сигнал сброса блока
Входы – выходы	MX	REAL	Интегральное значение M1 и M2
Используемые модули	T_PLC_MS, R2_ADD		

Функциональный блок **METER** используется для определения интегрального значения двух величин за заданный период времени. Пока вход **I1** имеет значение **TRUE**, значение **M1** каждый цикл ПЛК прибавляется к последнему сохраненному значению. Пока вход **I2** имеет значение **TRUE**, значение **M2** каждый цикл ПЛК прибавляется к последнему сохраненному значению. Вход **D** определяет значение делителя для сохраненного значения. Выход **MX** содержит сумму всех предыдущих значений **M1** и **M2** за периоды, во время которых **I1** и **I2** имели значение **TRUE**, поделенную на **D**:

$$MX = \frac{1}{D} \cdot \sum (I1 \cdot M1 + I2 \cdot M2)$$

По переднему фронту на входе **RST** значение **MX** обнуляется, после чего суммирование начинается заново.

На рисунке ниже приведена трассировка работы блока, используемого для подсчета суммарной потребляемой электроэнергии для двух нагревателей с мощностью 60 и 85 кВт соответственно. Переменные **x11** и **x12** определяют состояние соответствующего нагревателя (**TRUE** – включен, **FALSE** – отключен).

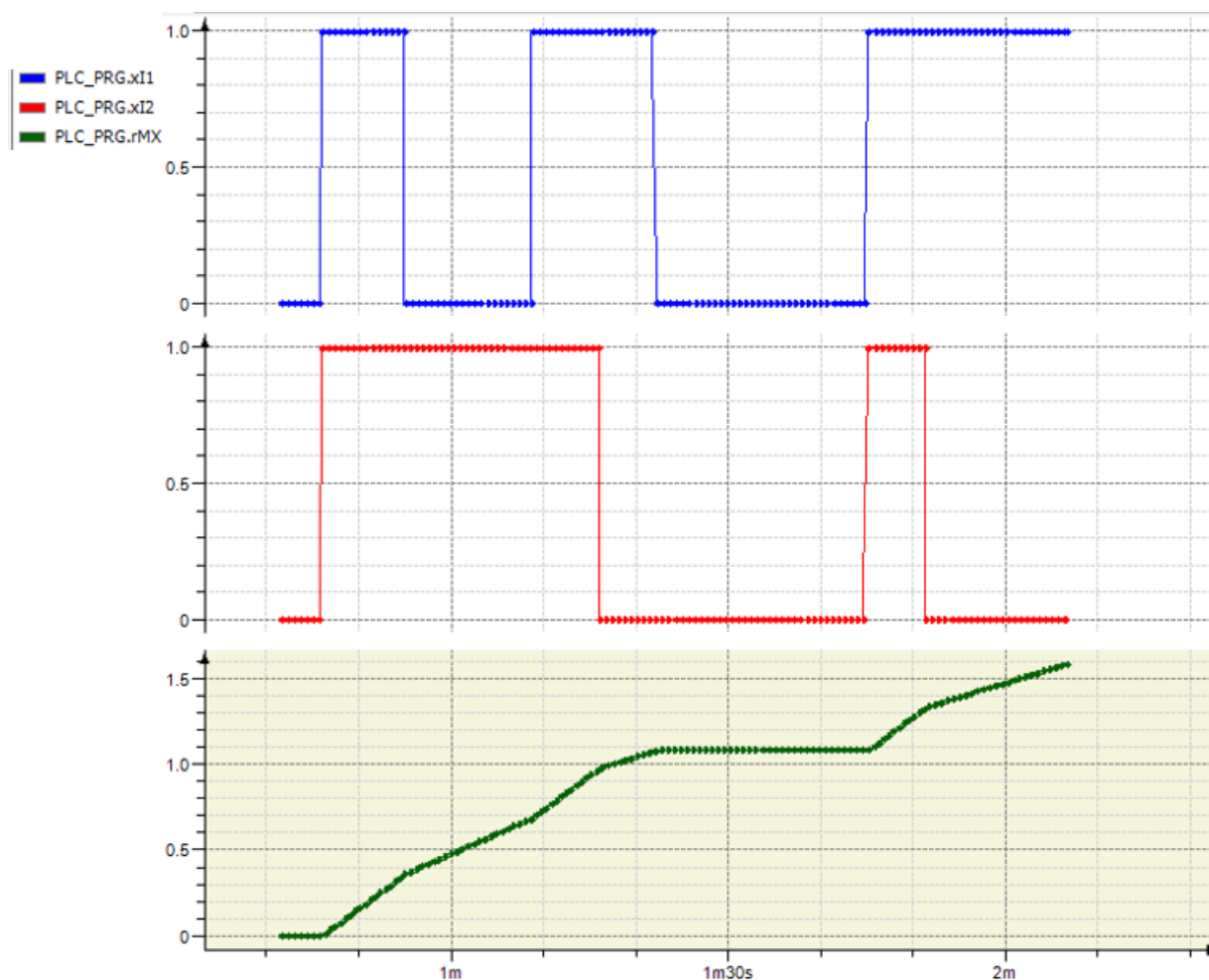


Рисунок 24.1 – Трассировка работы ФБ METER

24.11 METER_STAT

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	REAL	Контролируемое значение
	DI	DATE	Текущая дата
	RST	BOOL	Сигнал сброса блока

Тип модуля: ФБ	Переменная	Тип	Описание
Входы – выходы	Last_Day	REAL	Расход за вчерашний день
	Current_Day	REAL	Расход за текущий день
	Last_Week	REAL	Расход за прошлую неделю
	Current_Week	REAL	Расход за текущую неделю
	Last_Month	REAL	Расход за прошлый месяц
	Current_Month	REAL	Расход за текущий месяц
	Last_Year	REAL	Расход за прошлый год
	Current_Year	REAL	Расход за текущий год
Используемые модули	YEAR_OF_DATE, MONTH_OF_DATE, DAY_OF_YEAR, DAY_OF_WEEK		

Функциональный блок **METER_STAT** используется для определения суммарного расхода величины **IN** за прошлые день (**Last_Day**), неделю (**Last_Week**), месяц (**Last_Month**) и год (**Last_Year**), а также за текущие день (**Current_Day**), неделю (**Current_Week**), месяц (**Current_Month**) и год (**Current_Year**). На вход **DI** подается текущая дата. По переднему фронту на входе **RST** выходы блока обнуляются, после чего сбор статистики начинается заново.

Логика работы блока подразумевает, что **IN** является возрастающей с течением времени величиной, характеризующий расход какого-либо вещества, потребление электроэнергии и т.п.

24.12 ONTIME

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	BOOL	Контролируемый сигнал
	RST	BOOL	Сигнал сброса блока
Входы – выходы	SECONDS	UDINT	Время активного сигнала
	CYCLES	UDINT	Число периодов активности сигнала
Используемые модули	T_PLC_MS		

Функциональный блок **ONTIME** представляет собой модуль измерения периода активности сигнала. Если вход **IN** имеет значение **TRUE**, то на входе-выходе **SECONDS** отображается число секунд активности сигнала. Если вход **IN** принимает значение **FALSE**, то отсчет времени останавливается, но будет продолжен при следующем переходе в состояние **TRUE**. По переднему фронту входа **IN** значение входа-выхода **CYCLES** увеличивается на +1 – таким образом, эта величина характеризует число периодов активности сигнала. По переднему фронту на входе **RST** значения **SECONDS** и **CYCLES** обнуляются, после чего подсчет начинается заново.

24.13 T_PLC_MS

Тип модуля: функция	Переменная	Тип	Описание
Выходы	T_PLC_MS	DWORD	Значение системного таймера ПЛК
Константы	debug	BOOL	Управление режимом отладки
	N	INT	Коэффициент масштаба времени
	offset	DWORD	Смещение времени таймера, мс

Функция **T_PLC_MS** возвращает значение системного таймера (время, прошедшее со старта ПЛК) в миллисекундах. Функция **T_PLC_MS** представляет собой обвязку вокруг системной функции **TIME ()**, но помимо этого позволяет тестировать системный таймер на переполнение. Для этого надо внести изменения в область объявления переменных функции: присвоить константе **debug** значение **TRUE** и задать значения для констант **N** (коэффициент масштаба времени, каждую миллисекунду к значению таймера будет прибавляться 2^N миллисекунд) и **offset** (смещение значения таймера).

Следует помнить, что реализация программной обвязки для системного таймера может отличаться для разных сред программирования.

24.14 T_PLC_US

Тип модуля: функция	Переменная	Тип	Описание
Выходы	T_PLC_US	DWORD	Значение системного таймера ПЛК
Константы	debug	BOOL	Управление режимом отладки
	N	INT	Коэффициент масштаба времени
	offset	DWORD	Смещение времени таймера, мс

Функция **T_PLC_US** возвращает значение системного таймера (время, прошедшее со старта ПЛК) в микросекундах. Функция **T_PLC_US** фактически возвращает значение системной функции **TIME ()**, умноженное на 1000. При необходимости эта реализация может быть заменена на вызов системной функции **LTIME ()**, если последняя поддерживается на ПЛК. Функция **T_PLC_US** также содержит режим отладки; см. подробности в описании функции [T_PLC_MS](#).

24.15 TC_MS

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	TC	DWORD	Время системного таймера, мс
Используемые модули	T_PLC_MS		

Функциональный блок **TC_MS** возвращает значение длительности (в миллисекундах) прошедшего цикла задачи ПЛК, в которой осуществляется вызов блока. Блок представляет собой обвязку для функции [T_PLC_MS](#); если блок возвращает некорректное значение, то следует проверить, не используется ли данная функция в режиме отладки.

24.16 TC_S

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	TC	REAL	Время системного таймера
Используемые модули	T_PLC_US		

Функциональный блок **TC_S** возвращает значение длительности прошедшего цикла задачи ПЛК, в которой осуществляется вызов блока, в виде переменной с плавающей точкой с разрядностью до миллисекунд (например, 0.01 для T=10 мс). Блок представляет собой обвязку для функции [T_PLC_US](#); если блок возвращает некорректное значение, то следует проверить, не используется ли данная функция в режиме отладки. Более подробную информацию (в т.ч. сведения о точности) см. в описании функции [T_PLC_US](#).

24.17 TC_US

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	TC	DWORD	Время системного таймера, мкс
Используемые модули	T_PLC_US		

Функциональный блок **TC_US** (в микросекундах) прошедшего цикла задачи ПЛК, в которой осуществляется вызов блока. Блок представляет собой обвязку для функции [T_PLC_US](#); если блок возвращает некорректное значение, то следует проверить, не используется ли данная функция в режиме отладки. Более подробную информацию (в т.ч. сведения о точности) см. в описании функции [T_PLC_US](#).

25 Конвертация величин

25.1 ASTRO

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	m	REAL	Исходное расстояние, м
	AE	REAL	Исходное расстояние, а.е
	PC	REAL	Исходное расстояние, парсеки
	LJ	REAL	Исходное расстояние, световые года
Выходы	Ym	REAL	Конвертированное расстояние, м
	YAE	REAL	Конвертированное расстояние, а.е.
	YPC	REAL	Конвертированное расстояние, парсеки
	YLJ	REAL	Конвертированное расстояние, световые года

Функциональный блок **ASTRO** конвертирует значение расстояния из одних астрономических единиц измерения в другие. Подразумевается, что при каждом вызове блока используется только один вход, а остальные имеют нулевые значения. При этом на выходах отображаются конвертированные значения в соответствующих единицах измерения. Если при вызове ФБ есть значения на нескольких входах, то будет произведена конвертация суммы этих значений.

Соотношения между единицами:

- 1 а.е. = 149 597 870 700 м;
- 1 парсек $\approx 206\,264,8$ а. е. = $3,0856776 \cdot 10^{16}$ м = 3,2616 светового года;
- 1 световой год = 9 460 730 472 580 800 м $\approx 63\,241,077$ а.е.

Пример

```

VAR      AstroConv: OSCAT_BASIC.ASTRO;

(* Расстояние в световых годах *)
DistanceLJ: REAL := 4.24;          (* 4.24 световых года до Проксимы Центавра *)

(* Результаты *)
Meters:    REAL;
AE:        REAL;
Parsecs:   REAL;
LightYears: REAL;
END_VAR

(* Вызов ФБ с входом LJ *)
AstroConv(LJ := DistanceLJ);

(* Чтение результатов *)
Meters := AstroConv.Ym;      (*  $\approx 4.0113E16$  м *)
AE := AstroConv.YAE;         (*  $\approx 268144.6$  а.е. *)
Parsecs := AstroConv.YPC;    (*  $\approx 1.3$  парсека *)
LightYears := AstroConv.YLJ; (* 4.24 световых года *)

(* Результат:
- Ym  $\approx 4.0113E16$  м
- YAE  $\approx 268144.6$  а.е.
- YPC  $\approx 1.3$  парсека
- YLJ = 4.24 световых года*)
END_PROGRAM

```

25.2 BFT_TO_MS

Тип модуля: функция	Переменная	Тип	Описание
Входы	BFT	INT	Скорость ветра в баллах Бофорта (1..12)
Выходы	BFT_TO_MS	REAL	Скорость ветра в м/с

Функция **BFT_TO_MS** конвертирует значение скорости ветра из шкалы Бофорта в метры в секунду. Преобразование выполняется с помощью эмпирической формулы

$$\text{BFT_TO_MS} = 0.837 \cdot \text{BFT}^{3/2}$$

См. также обратную функцию [MS_TO_BFT](#).

Пример

```
VAR
  (* Скорость ветра в баллах Бофорта *)
  WindForce : INT := 4;          (* 4 балла - «Умеренный ветер» (5.5–7.9 м/с) *)

  (* Результат конвертации *)
  WindSpeed : REAL;
END_VAR

(* Вызов функции BFT_TO_MS *)
WindSpeed := OSCAT_BASIC.BFT_TO_MS(
  BFT := WindForce
);

(*Результат: WindSpeed ≈ 6.696 м/с*)
END_PROGRAM
```

25.3 C_TO_F

Тип модуля: функция	Переменная	Тип	Описание
Входы	celsius	REAL	Температура в градусах Цельсия
Выходы	C_TO_F	REAL	Температура в градусах по Фаренгейту

Функция **C_TO_F** конвертирует значение температуры из градусов Цельсия в градусы Фаренгейта. Преобразование выполняется с помощью формулы

$$\text{C_TO_F} = (9/5) \cdot \text{celsius} + 32$$

См. также обратную функцию [F_TO_C](#).

Пример

```
VAR
  (* Температура в градусах Цельсия *)
  TempCelsius : REAL := 25.0;    (* 25 °C *)

  (* Результат конвертации *)
  TempFahrenheit : REAL;
END_VAR

(* Вызов функции C_TO_F *)
TempFahrenheit := OSCAT_BASIC.C_TO_F(
  celsius := TempCelsius
);

(*Результат: TempFahrenheit = 77.0 °F*)
END_PROGRAM
```

25.4 C_TO_K

Тип модуля: функция	Переменная	Тип	Описание
Входы	celsius	REAL	Температура в градусах Цельсия
Выходы	C_TO_K	REAL	Температура в градусах Кельвина

Функция **C_TO_K** конвертирует значение температуры из градусов Цельсия в кельвины. Преобразование выполняется с помощью формулы

$$C_TO_K = celsius + 273.15$$

См. также обратную функцию [K_TO_C](#).

Пример

```

VAR
  (* Температура в градусах Цельсия *)
  TempCelsius : REAL := 30.0;          (* 25 °C *)

  (* Результат конвертации *)
  TempKelvin   : REAL;
END_VAR

(* Вызов функции C_TO_K *)
TempKelvin := OSCAT_BASIC.C_TO_K(
  celsius := TempCelsius
);

(*Результат: TempKelvin = 303*)
END_PROGRAM

```

25.5 DEG_TO_DIR

Тип модуля: функция	Переменная	Тип	Описание
Входы	DEG	INT	Азимут, градусы
	N	INT	Коэффициент кол-ва лучей розы румбов (1..3)
	L	INT	Язык проекта
Выходы	DEG_TO_DIR	STRING(3)	Обозначение румба

Функция **DEG_TO_DIR** конвертирует значения азимута **DEG** (в градусах) в обозначение соответствующего румба. Вход **N** определяет кол-во лучей розы румбов (см. рисунок ниже):

- N=1 – используются 4 луча;
- N=2 – используются 8 лучей;
- N=3 – используются 16 лучей.

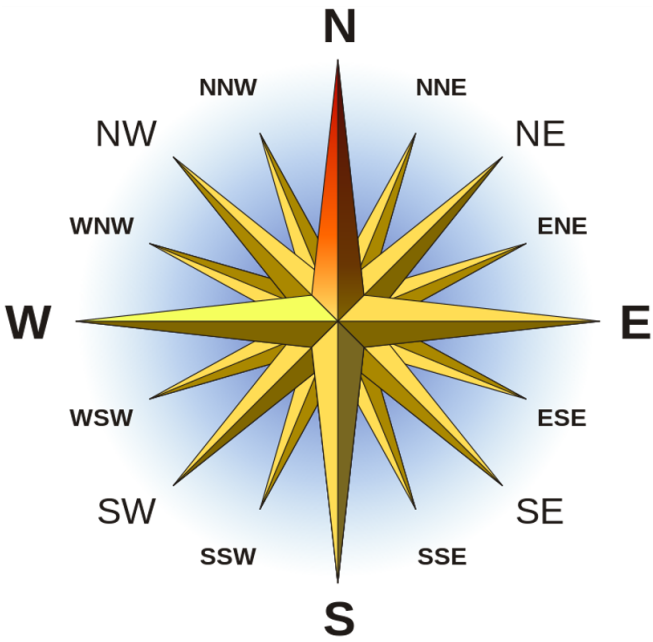


Рисунок 25.1 – Шестнадцатилучевая роза румбов

Вход **LANG** определяет язык приложения (см. глобальную переменную [LANGUAGE](#); в ней же содержатся и обозначения румбов). См. также обратную функцию [DIR_TO_DEG](#).

Пример

```
VAR      (* Азимут в градусах *)
Azimuth : INT := 120;                (* 120 градусов *)

(* Количество лучей розы румбов *)
Rays : INT := 2;                     (* 8 лучей (N=2) *)

(* Язык проекта (0 - английский) *)
Lang : INT := 0;                     (* Английский язык *)

(* Результат - обозначение румба *)
Direction : STRING(3);
END_VAR

(* Вызов функции DEG_TO_DIR *)
Direction := OSCAT_BASIC.DEG_TO_DIR(
    DEG := Azimuth,
    N := Rays,
    L := Lang
);

(*Результат: Direction = 'SE'*)
END_PROGRAM
```

25.6 DIR_TO_DEG

Тип модуля: функция	Переменная	Тип	Описание
Входы	DIR	STRING(3)	Обозначение румба
	L	INT	Язык проекта
Выходы	DIR_TO_DEG	INT	Азимут, градусы

Функция **DIR_TO_DEG** конвертирует обозначения румба в значение азимута в градусах. Функция работает с 16-лучевой розой румбов (см. рисунок выше). Вход **L** определяет язык приложения (см. глобальную переменную [LANGUAGE](#); в ней же содержатся и названия румбов). См. также обратную функцию [DEG_TO_DIR](#).

Пример

```

VAR
  (* Обозначение румба *)
  Direction : STRING(3) := 'SE';      (* Юго-восток *)

  (* Язык проекта (0 - английский) *)
  Lang : INT := 0;                    (* Английский язык *)

  (* Результат - азимут в градусах *)
  Azimuth : INT;
END_VAR

(* Вызов функции DIR_TO_DEG *)
Azimuth := OSCAT_BASIC.DIR_TO_DEG(
  DIR := Direction,
  L := Lang
);

(*Результат: Azimuth = 135*)
END_PROGRAM

```

25.7 ENERGY

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	J	REAL	Исходное значение энергии, Дж
	C	REAL	Исходное значение энергии, кал
	WH	REAL	Исходное значение энергии, Вт·ч
Выходы	YJ	REAL	Конвертированное значение энергии, Дж
	YC	REAL	Конвертированное значение энергии, кал
	YWH	REAL	Конвертированное значение энергии, Вт·ч

Функциональный блок **ENERGY** конвертирует значение энергии из одних единиц измерения в другие. Предполагается, что при каждом вызове блока используется только один вход, а остальные имеют нулевые значения. При этом на выходах отображаются конвертированные значения в соответствующих единицах измерения. Если при вызове ФБ есть значения на нескольких входах, то будет произведена конвертация суммы этих значений.

- 1 кал = 4.1868 Дж = $1,163 \cdot 10^{-3}$ Вт·ч;
- 1 Вт·ч = $3.6 \cdot 10^{-3}$ Дж = 860 кал.

Пример

```

VAR
  (* Экземпляр ФБ ENERGY *)
  EnergyConv : OSCAT_BASIC.ENERGY;

  (* Исходное значение энергии в джоулях *)
  EnergyJ : REAL := 4186.8; (* 4186.8 Дж = 1000 кал *)
  (* Результаты конвертации *)
  ResultJ : REAL; (* В джоулях *)
  ResultC : REAL; (* В калориях *)
  ResultWH : REAL; (* В ватт-часах *)

END_VAR

(* Вызов ФБ с заполненными входами J*)
EnergyConv(J := EnergyJ);
(* Чтение результатов *)
ResultJ := EnergyConv.YJ; (*4186.8 Дж *)
ResultC := EnergyConv.YC; (*1000.0 кал*)
ResultWH := EnergyConv.YWH; (* ≈ 1.163 Вт·ч*)

END_PROGRAM

```

25.8 F_TO_C

Тип модуля: функция	Переменная	Тип	Описание
Входы	fahrenheit	REAL	Температура в градусах по Фаренгейту
Выходы	F_TO_C	REAL	Температура в градусах Цельсия

Функция **C_TO_F** конвертирует значение температуры из градусов Фаренгейта в градусы Цельсия. Преобразование выполняется с помощью формулы

$$F_TO_C = (5/9) \cdot (fahrenheit - 32)$$

См. также обратную функцию [C_TO_F](#)

Пример

```

VAR
  (* Температура в градусах Фаренгейта *)
  TempFahrenheit : REAL := 77.0; (* 77 °F *)

  (* Результат конвертации *)
  TempCelsius : REAL;
END_VAR

(* Вызов функции F_TO_C *)
TempCelsius := OSCAT_BASIC.F_TO_C(
  fahrenheit := TempFahrenheit
);

(*Результат: TempCelsius = 25.0 °C*)

```

25.9 F_TO_OM

Тип модуля: функция	Переменная	Тип	Описание
Входы	F	REAL	Частота, Гц
Выходы	F_TO_OM	REAL	Угловая частота, рад/с

Функция **F_TO_OM** возвращает значение угловой частоты для заданной частоты **F**. Вычисление выполняется по формуле

$$F_TO_OM = 2\pi F$$

См. также обратную функцию [OM_TO_F](#)

Пример

```

VAR
  (* Частота, Гц *)
  Frequency : REAL := 50.0;          (* 50 Гц (промышленная частота) *)

  (* Результат конвертации *)
  AngularFrequency : REAL;          (* Угловая частота, рад/с *)
END_VAR

(* Вызов функции F_TO_OM *)
AngularFrequency := OSCAT_BASIC.F_TO_OM(
  F := Frequency
);

(*Результат: AngularFrequency ≈ 314.159 рад/с*)

END_PROGRAM

```

25.10 F_TO_PT

Тип модуля: функция	Переменная	Тип	Описание
Входы	F	REAL	Частота колебаний, Гц
Выходы	F_TO_PT	TIME	Период колебаний, с

Функция **F_TO_PT** возвращает значение периода колебаний для заданной частоты **F**. Вычисление выполняется по формуле

$$F_TO_PT = 1/F$$

См. также обратную функцию [PT_TO_F](#)

Пример

```

VAR
  (* Частота колебаний, Гц *)
  Frequency : REAL := 50.0;          (* 50 Гц (промышленная частота) *)

  (* Результат конвертации *)
  Period : TIME;                    (* Период колебаний, с *)
END_VAR

(* Вызов функции F_TO_PT *)
Period := OSCAT_BASIC.F_TO_PT(
  F := Frequency
);

(*Результат: Period = T#20ms*)

END_PROGRAM

```

25.11 GEO_TO_DEG

Тип модуля: функция	Переменная	Тип	Описание
Входы	D	INT	Градусы
	M	INT	Минуты
	SEC	REAL	Секунды
Выходы	GEO_TO_DEG	REAL	Десятичные градусы

Функция **GEO_TO_DEG** конвертирует географические координаты из формата DMS (градусы **D**/минуты **M**/секунды **SEC**) в формат DD (десятичные градусы). Вычисление выполняется по формуле

$$GEO_TO_DEG = D + M/60 + SEC/3600$$

Пример


```

VAR
  (* Географические координаты в формате DMS *)
  Degrees : INT := 55;          (* 55 градусов *)
  Minutes  : INT := 45;          (* 45 минут *)
  Seconds  : REAL := 30.0;       (* 30 секунд *)

  (* Результат конвертации *)
  DecimalDegrees : REAL;        (* Десятичные градусы *)
END_VAR

(* Вызов функции GEO_TO_DEG *)
DecimalDegrees := OSCAT_BASIC.GEO_TO_DEG(
  D := Degrees,
  M := Minutes,
  SEC := Seconds
);

(* Результат: DecimalDegrees ≈ 55.758333° *)
END_PROGRAM

```

25.12 K_TO_C

Тип модуля: функция	Переменная	Тип	Описание
Входы	Kelvin	REAL	Температура в градусах Кельвина
Выходы	K_TO_C	REAL	Температура в градусах Цельсия

Функция **K_TO_C** конвертирует значение температуры из кельвинов в градусы Цельсия. Преобразование выполняется с помощью формулы

$$C_TO_K = Kelvin - 273.15$$

См. также обратную функцию [C_TO_K](#).

Пример

```

VAR
  (* Температура в градусах Цельсия *)
  TempKelvin : REAL := 303.15;

  (* Результат конвертации *)
  TempCelsius : REAL;
END_VAR

(* Вызов функции K_TO_C *)
TempCelsius := OSCAT_BASIC.K_TO_C(
  Kelvin := TempKelvin
);

(*Результат: TempCelsius = 30*)
END_PROGRAM

```

25.13 KMH_TO_MS

Тип модуля: функция	Переменная	Тип	Описание
Входы	kmh	REAL	Скорость, км/ч
Выходы	KMH_TO_MS	REAL	Скорость, м/с

Функция **KMH_TO_MS** конвертирует значение скорости из км/ч в м/с. См. также обратную функцию [MS_TO_KMH](#).

Пример

```

VAR
  (* Скорость в км/ч *)
  SpeedKmh : REAL := 180;

  (* Результат конвертации *)
  SpeedMS : REAL;
END_VAR

(* Вызов функции KMH_TO_MS *)
SpeedMS := OSCAT_BASIC.KMH_TO_MS(
  kmh := SpeedKmh
);

(*Результат: SpeedMS = 50.0000038*)
END_PROGRAM

```

25.14 LENGHT

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	m	REAL	Исходное значение длины, м
	p	REAL	Исходное значение длины, типографские пункты
	in	REAL	Исходное значение длины, дюймы
	ft	REAL	Исходное значение длины, футы
	yd	REAL	Исходное значение длины, ярды
	mile	REAL	Исходное значение длины, мили
	sm	REAL	Исходное значение длины, морские мили
	fm	REAL	Исходное значение длины, морские сажени
Выходы	Ym	REAL	Конвертированное значение длины, м
	Yp	REAL	Конвертированное значение длины, типограф. пункты
	Yin	REAL	Конвертированное значение длины, дюймы
	Yft	REAL	Конвертированное значение длины, футы
	Yyd	REAL	Конвертированное значение длины, ярды
	Ymile	REAL	Конвертированное значение длины, мили
	Ysm	REAL	Конвертированное значение длины, морские мили
	Yfm	REAL	Конвертированное значение длины, морские сажени

Функциональный блок **LENGTH** конвертирует значение длины из одних единиц измерения в другие. Предполагается, что при каждом вызове блока используется только один вход, а остальные имеют нулевые значения. При этом на выходах отображаются конвертированные значения в соответствующих единицах измерения. Если при вызове ФБ есть значения на нескольких входах, то будет произведена конвертация суммы этих значений.

Соотношения между единицами:

- 1 типографский пункт = 0,376065 мм;

- 1 дюйм = 25.4 мм;
- 1 фут = 0,3048 м;
- 1 ярд = 0,9144 м;
- 1 миля = 1609,344 м;
- 1 морская миля = 1852 м;
- 1 морская сажень = 1,829 м.

Пример “Конвертация одного значения”

```

VAR
  (* Экземпляр ФБ LENGTH *)
  LengthConv : OSCAT_BASIC.LENGTH;

  (* Исходное значение длины в метрах *)
  LengthM : REAL := 1000.0;      (* 1000 м = 1 км *)

  (* Результаты конвертации *)
  ResultM : REAL;                (* В метрах *)
  ResultIn : REAL;               (* В дюймах *)
  ResultFt : REAL;               (* В футах *)
  ResultMile : REAL;             (* В милях *)
  ResultSm : REAL;               (* В морских милях *)
END_VAR

(* Вызов ФБ с заполненным входом m *)
LengthConv(m := LengthM);

(* Чтение результатов *)
ResultM := LengthConv.Ym;        (* 1000.0 м *)
ResultIn := LengthConv.Yin;     (* ≈ 39370.08 дюймов *)
ResultFt := LengthConv.Yft;     (* ≈ 3280.84 футов *)
ResultMile := LengthConv.Ymile; (* ≈ 0.6214 мили *)
ResultSm := LengthConv.Ysm;     (* ≈ 0.5400 морской мили *)

END_PROGRAM

```

25.15 MS_TO_BFT

Тип модуля: функция	Переменная	Тип	Описание
Входы	MS	REAL	Скорость ветра в м/с
Выходы	MS_TO_BFT	INT	Скорость ветра в баллах Бофорта (1..12)

Функция **MS_TO_BFT** конвертирует значение скорости ветра из метров в секунду в баллы по шкале Бофорта. Преобразование выполняется с помощью эмпирической формулы

$$MS_TO_BFT = (1.196172 \cdot MS)^{2/3}$$

См. также обратную функцию [BFT_TO_MS](#).

Пример

```

VAR
  (* Скорость ветра в м/с *)
  WindSpeed : REAL := 6.69; (* 6.69 м/с *)

  (* Результат конвертации *)
  WindForce : INT;
END_VAR

(* Вызов функции MS_TO_BFT *)
WindForce := OSCAT_BASIC.MS_TO_BFT(
  MS := WindSpeed
);

(*Результат: 4 балла "Умеренный ветер"*)
END_PROGRAM

```

25.16 MS_TO_KMH

Тип модуля: функция	Переменная	Тип	Описание
Входы	MS	REAL	Скорость, м/с
Выходы	MS_TO_KMH	REAL	Скорость, км/ч

Функция **MS_TO_KMH** конвертирует значение скорости из м/с в км/ч.

$$MS_TO_BFT = (1.196172 \cdot MS)^{2/3}$$

См. также обратную функцию [KMH_TO_MS](#).

Пример

```

VAR
  (* Скорость в мс *)
  SpeedMS : REAL := 50.0000038;

  (* Результат конвертации *)
  SpeedKmh : REAL;
END_VAR

(* Вызов функции MS_TO_KMH *)
SpeedKmh := OSCAT_BASIC.MS_TO_KMH(
  MS := SpeedMS
);

(*Результат: SpeedKmh = 180 км/ч*)
END_PROGRAM

```

25.17 OM_TO_F

Тип модуля: функция	Переменная	Тип	Описание
Входы	OM	REAL	Угловая частота, рад/с
Выходы	OM_TO_F	REAL	Частота, Гц

Функция **OM_TO_F** возвращает значение частоты для заданной угловой частоты **OM**. Вычисление выполняется по формуле

$$OM_TO_F = OM / 2\pi$$

См. также обратную функцию [F_TO_OM](#)

Пример

```

VAR
  (* угловая частота, рад/с *)
  AngularFrequency : REAL := 314.15;

  (* Результат конвертации *)
  Frequency : REAL;          (* Частота, Гц *)
END_VAR

(* Вызов функции OM_TO_F *)
Frequency := OSCAT_BASIC.OM_TO_F(
  OM := AngularFrequency
);

(*Результат: Frequency = 50 Гц*)
END_PROGRAM

```

25.18 PRESSURE

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	mws	REAL	Исходное значение давления, метры водяного столба
	torr	REAL	Исходное значение давления, мм ртутного столба
	att	REAL	Исходное значение давления, технические атмосферы
	atm	REAL	Исходное значение давления, физические атмосферы
	pa	REAL	Исходное значение давления, Паскали
	bar	REAL	Исходное значение давления, бары
Выходы	Ymws	REAL	Конвертированное значение, метры водяного столба
	Ytorr	REAL	Конвертированное значение, мм ртутного столба
	Yatt	REAL	Конвертированное значение, технические атмосферы
	Yatm	REAL	Конвертированное значение, физические атмосферы
	Ypa	REAL	Конвертированное значение, Паскали
	Ybar	REAL	Конвертированное значение, бары

Функциональный блок **PRESSURE** конвертирует значение давления из одних единиц измерения в другие. Подразумевается, что при каждом вызове блока используется только один вход, а остальные имеют нулевые значения. При этом на выходах отображаются конвертированные значения в соответствующих единицах измерения. Если при вызове ФБ есть значения на нескольких входах, то будет произведена конвертация суммы этих значений.

Соотношения между единицами приведены в [этой статье](#).

Пример “Конвертация одного значения”

```

VAR      (* Экземпляр ФБ PRESSURE *)
PressureConv : OSCAT_BASIC.PRESSURE;

(* Исходное значение давления в барах *)
PressureBar : REAL := 1.0;          (* 1 бар *)

(* Результаты конвертации *)
ResultBar : REAL;                  (* В барах *)
ResultPa : REAL;                   (* В паскалях *)
ResultAtm : REAL;                  (* В физических атмосферах *)
ResultAtt : REAL;                  (* В технических атмосферах *)
ResultTorr : REAL;                 (* В мм ртутного столба *)
ResultMws : REAL;                  (* В метрах водяного столба *)
END_VAR

(* Вызов ФБ с заполненным входом bar *)
PressureConv(bar := PressureBar);

(* Чтение результатов *)
ResultBar := PressureConv.Ybar;    (* 1.0 бар *)
ResultPa := PressureConv.Ypa;      (* 100000.0 Па *)
ResultAtm := PressureConv.Yatm;    (* ≈ 0.9869 атм *)
ResultAtt := PressureConv.Yatt;    (* ≈ 1.0197 ат *)
ResultTorr := PressureConv.Ytorr;  (* ≈ 750.06 мм рт. ст. *)
ResultMws := PressureConv.Ymws;    (* ≈ 10.197 м вод. ст. *)

END_PROGRAM

```

25.19 PT_TO_F

Тип модуля: функция	Переменная	Тип	Описание
Входы	PT	TIME	Период колебаний, с
Выходы	PT_TO_F	REAL	Частота колебаний, Гц

Функция **PT_TO_F** возвращает значение частоты для заданного периода колебаний **PT**. Вычисление выполняется по формуле

$$PT_TO_F = 1 / PT$$

См. также обратную функцию [F_TO_PT](#).

Пример

```

VAR
(* Период колебаний, с *)
Period : TIME := T#20ms;

(* Результат конвертации *)
Frequency : REAL;          (* Частота колебаний, Гц *)
END_VAR

(* Вызов функции PT_TO_F *)
Frequency := OSCAT_BASIC.PT_TO_F(
    PT := Period
);

(*Результат: Frequency = 50 Гц*)

END_PROGRAM

```

25.20 SPEED

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	ms	REAL	Исходное значение скорости, м/с

Тип модуля: ФБ	Переменная	Тип	Описание
	kmh	REAL	Исходное значение скорости, км/ч
	kn	REAL	Исходное значение скорости, узлов
	mh	REAL	Исходное значение скорости, миль/ч
Выходы	Yms	REAL	Конвертированное значение скорости, м/с
	Ykmh	REAL	Конвертированное значение скорости, км/ч
	Ykn	REAL	Конвертированное значение скорости, узлов
	Ymh	REAL	Конвертированное значение скорости, миль/ч

Функциональный блок **SPEED** конвертирует значение скорости из одних единиц измерения в другие. Предполагается, что при каждом вызове блока используется только один вход, а остальные имеют нулевые значения. При этом на выходах отображаются конвертированные значения в соответствующих единицах измерения. Если при вызове ФБ есть значения на нескольких входах, то будет произведена конвертация суммы этих значений.

Соотношения между единицами:

- 1 км/ч = 0,278 м/с;
- 1 узел = 1 морская миля/ч = 1.852 км/ч = 0.5144 м/с;
- 1 миля/ч = 1.609 км/ч = 0.447 м/с

Пример

```

VAR
  (* Экземпляр ФБ SPEED *)
  SpeedConv : OSCAT_BASIC.SPEED;

  (* Исходное значение скорости в м/с *)
  SpeedMs : REAL := 10.0;          (* 10 м/с *)

  (* Результаты конвертации *)
  ResultMs : REAL;                  (* В м/с *)
  ResultKmh : REAL;                 (* В км/ч *)
  ResultKn : REAL;                  (* В узлах *)
  ResultMh : REAL;                  (* В милях/ч *)
END_VAR

(* Вызов ФБ с заполненным входом ms *)
SpeedConv(ms := SpeedMs);

(* Чтение результатов *)
ResultMs := SpeedConv.Yms;          (* 10.0 м/с *)
ResultKmh := SpeedConv.Ykmh;        (* 36.0 км/ч *)
ResultKn := SpeedConv.Ykn;          (* ≈ 19.44 узла *)
ResultMh := SpeedConv.Ymh;          (* ≈ 22.37 миль/ч *)
END_PROGRAM

```

25.21 TEMPERATURE

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	K	REAL	Исходное значение температуры, кельвины
	C	REAL	Исходное значение температуры, градусы Цельсия

Тип модуля: ФБ	Переменная	Тип	Описание
	F	REAL	Исходное значение температуры, градусы Фаренгейта
	Re	REAL	Исходное значение температуры, градусы Реомюра
	Ra	REAL	Исходное значение температуры, градусы Ранкина
Выходы	YK	REAL	Конвертированное значение, кельвины
	YC	REAL	Конвертированное значение, градусы Цельсия
	YF	REAL	Конвертированное значение, градусы Фаренгейта
	YRe	REAL	Конвертированное значение, градусы Реомюра
	YRa	REAL	Конвертированное значение, градусы Ранкина

Функциональный блок **TEMPERATURE** конвертирует значение температуры из одних единиц измерения в другие. Подразумевается, что при каждом вызове блока используется только один вход, а остальные имеют нулевые значения. При этом на выходах отображаются конвертированные значения в соответствующих единицах измерения. Если при вызове ФБ есть значения на нескольких входах, то будет произведена конвертация суммы этих значений.

Соотношения между единицами:

- $T_K = T_C + 273,15;$
- $T_C = T_K - 273,15;$
- $T_F = 9/5 \cdot T_C + 32;$
- $T_{RE} = (T_K - 273,15) \cdot 0,8;$
- $T_{RA} = 1,8 \cdot T_K$

Следует помнить, что разные температурные шкалы имеют разные точки отсчета. По умолчанию входы блока имеют следующие значения:

- K= 0.0;
- C= -273,15;
- F= -459,67;
- Re= -218,52;
- Ra= 0.0

Пример


```
VAR
  (* Экземпляр ФБ TEMPERATURE *)
  TempConv : OSCAT_BASIC.TEMPERATURE;

  (* Исходное значение температуры в градусах Цельсия *)
  TempCelsius : REAL := 100.0;      (* 100 °C (точка кипения воды) *)

  (* Результаты конвертации *)
  ResultK : REAL;                    (* В кельвинах *)
  ResultC : REAL;                    (* В градусах Цельсия *)
  ResultF : REAL;                    (* В градусах Фаренгейта *)
  ResultRe : REAL;                   (* В градусах Реомюра *)
  ResultRa : REAL;                   (* В градусах Ранкина *)
END_VAR

(* Вызов ФБ с заполненным входом C *)
TempConv(C := TempCelsius);

(* Чтение результатов *)
ResultK := TempConv.YK;              (*YK = 373.15 K*)
ResultC := TempConv.YC;              (*YC = 100.0 °C*)
ResultF := TempConv.YF;              (*YF = 212.0 °F*)
ResultRe := TempConv.YRe;            (*YRe = 80.0 °Re*)
ResultRa := TempConv.YRa;            (*YRa = 671.67 °Ra*)

END_PROGRAM
```

26 Регуляторы

Данная глава содержит описание модулей управления (регуляторов), содержащихся в библиотеке **OSCAT**. Регулятором называется устройство, которое следит за функционированием объекта управления и, постоянно анализируя его состояние, вырабатывает определенное управляющее воздействие (сигнал управления). В тех случаях, когда это возможно, модули управления должны вызываться каждый цикл ПЛК. Преимуществом такого подхода является возможность в пределах одной задачи контролировать процессы, протекающие с разной скоростью. Альтернативный подход подразумевает вызов модулей с заданной частотой (например, по таймеру); его недостатком является возможное нарушение частоты вызова блоков в задачах с низким приоритетом. Величину цикла ПЛК должен определить программист, проанализировав требования и характеристики конкретного технологического процесса.

Ниже описана структура и взаимосвязь основных модулей данной главы:

Модуль	Основан на	Описание
INTEGRATE	-	Блок интегрирования
FT_DERIV	-	Блок дифференцирования
CTRL_IN	-	Блок вычисления сигнала рассогласования
CTRL_OUT	-	Блок вычисления управляющего воздействия
FT_INT	INTEGRATE	Блок интегрирования с ограничением выхода
FT_PI	FT_INT	Блок ПИ-регулирования
FT_PIW	FT_INT	Блок ПИ-регулирования с компенсацией эффекта интегрального насыщения
FT_PIWL	-	Блок ПИ-регулирования с компенсацией эффекта интегрального насыщения
CTRL_PI	FT_PIWL, CTRL_IN, CTRL_OUT	ПИ-регулятор с обвязкой
FT_PID	FT_INT, FT_DERIV	Блок ПИД-регулирования
FT_PIDW	INTEGRATE, FT_DERIV	Блок ПИД-регулирования с компенсацией эффекта интегрального насыщения
FT_PIDWL	FT_PIWL, FT_DERIV	Блок ПИД-регулирования с компенсацией эффекта интегрального насыщения
CTRL_PID	FT_PIDWL, CTRL_IN, CTRL_OUT	ПИД-регулятор с обвязкой
FT_PD	FT_DERIV	ПД-регулятор
FT_PDT1	FT_DERIV, FT_PT1	ПД-регулятор с А-звеном

26.1 BAND_B

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	BYTE	Входное значение
	B	BYTE	Область ограничения
Выходы	BAND_B	BYTE	Выходное значение

Функция **BAND_B** ограничивает диапазон входного сигнала **X** по следующему принципу:

- $BAND_B = 0$, если $X < B$;
- $BAND_B = X$, если $B \leq X \leq (255-B)$;
- $BAND_B = 255$, если $X > (255-B)$.

26.2 CONTROL_SET1

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	Kt	REAL	Коэффициент усиления на границе устойчивости.
	Tt	REAL	Период установившихся колебаний
	PI	BOOL	Режим «Настройки ПИ-регулятора»
	PID	BOOL	Режим «Настройки ПИД-регулятора»
Выходы	KP	REAL	Рассчитанный коэффициент усиления
	TN	REAL	Постоянная времени интегрирования
	TV	REAL	Постоянная времени дифференцирования
	KI	REAL	Рассчитанный коэффициент интегрирования
	KD	REAL	Рассчитанный коэффициент дифференцирования
Параметры	P_K	REAL	Эмпирический коэффициент (0.5)
	PI_K	REAL	Эмпирический коэффициент (0.45)
	PI_TN	REAL	Эмпирический коэффициент (0.83)
	PID_K	REAL	Эмпирический коэффициент (0.6)
	PID_TN	REAL	Эмпирический коэффициент (0.5)
	PID_TV	REAL	Эмпирический коэффициент (0.125)

Функциональный блок **CONTROL_SET1** используется для настройки регулятора методом Циглера-Никольса (по частотным параметрам). Для этого регулятор переводится в режим П-регулятора, коэффициент усиления которого увеличивается до тех пор, пока на выходе системы не установятся колебания с постоянной амплитудой (т. е. система выводится на границу устойчивости). Коэффициент усиления на границе устойчивости и период незатухающих колебаний передаются на входы блока **Kt** и **Tt**. Если вход **PI** имеет значение **TRUE**, то будет проведена настройка ПИ-регулятора; если вход **PID** имеет значение **TRUE**, то будет проведена настройка ПИД-регулятора. Если оба входа имеют значение **FALSE**, то будет проведена настройка П-регулятора. Если оба входа имеют значение **TRUE**, то настройка проведена не будет. Рассчитанные параметры подаются на выходы блока. Они вычисляются по следующим эмпирическим соотношениям:

Регулятор	PI	PID	KP	TN	TV	KI	KD
П	FALSE	FALSE	$P_K \cdot Kt$				
ПИ	TRUE	FALSE	$PI_K \cdot Kt$	$PI_TN \cdot Tt$		KP/TN	
ПИД	FALSE	TRUE	$PID_K \cdot Kt$	$PID_TN \cdot Tt$	$PID_TV \cdot Tt$		$KP \cdot TV$

26.3 CONTROL_SET2

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	KS	REAL	Коэффициент усиления
	TU	REAL	Время запаздывания

Тип модуля: ФБ	Переменная	Тип	Описание
	TG	REAL	Постоянная времени
	PI	BOOL	Режим «Настройки ПИ-регулятора»
	PID	BOOL	Режим «Настройки ПИД-регулятора»
Выходы	KP	REAL	Рассчитанный коэффициент усиления
	TN	REAL	Постоянная времени интегрирования
	TV	REAL	Постоянная времени дифференцирования
	KI	REAL	Рассчитанный коэффициент интегрирования
	KD	REAL	Рассчитанный коэффициент дифференцирования
Параметры	P_K	REAL	Эмпирический коэффициент (1.0)
	PI_K	REAL	Эмпирический коэффициент (0.9)
	PI_TN	REAL	Эмпирический коэффициент (3.33)
	PID_K	REAL	Эмпирический коэффициент (1.2)
	PID_TN	REAL	Эмпирический коэффициент (2.0)
	PID_TV	REAL	Эмпирический коэффициент (0.5)

Функциональный блок **CONTROL_SET2** используется для настройки регулятора методом Циглера-Никольса (по кривой разгона). Для этого на объект управления подается ступенчатое воздействие и снимается кривая разгона (переходная характеристика). Графическим методом определяются параметры **KS** (коэффициент усиления), **TU** (время запаздывания) и **TG** (постоянная времени), которые подаются на соответствующие входы блока. Если вход **PI** имеет значение **TRUE**, то будет проведена настройка ПИ-регулятора; если вход **PID** имеет значение **TRUE**, то будет проведена настройка ПИД-регулятора. Если оба входа имеют значение **FALSE**, то будет проведена настройка П-регулятора. Если оба входа имеют значение **TRUE**, то настройка проведена не будет. Рассчитанные параметры подаются на выходы блока. Они вычисляются по следующим эмпирическим соотношениям:

Регулятор	PI	PID	KP	TN	TV	KI	KD
П	FALSE	FALSE	$(P_K \cdot K_S \cdot TG) / TU$				
ПИ	TRUE	FALSE	$(PI_K \cdot K_S \cdot TG) / TU$	$PI_TN \cdot TU$		KP / TN	
ПИД	FALSE	TRUE	$(PID_K \cdot K_S \cdot TG) / TU$	$PID_TN \cdot TU$	$PID_TV \cdot TU$	KP / TN	$KP \cdot TV$

26.4 CTRL_IN

Тип модуля: функция	Переменная	Тип	Описание
Входы	SET_POINT	REAL	Значение уставки
	ACTUAL	REAL	Значение регулируемой величины

Тип модуля: функция	Переменная	Тип	Описание
	NOISE	REAL	Зона нечувствительности
Выходы	CTRL_IN	REAL	Сигнал рассогласования
Используемые модули	DEAD_ZONE		

Функция **CTRL_IN** возвращает сигнал рассогласования между значением уставки **SET_POINT** и измеренным значением регулируемой величины **ACTUAL** с учетом зоны нечувствительности **NOISE**. Если $|\text{SET_POINT} - \text{ACTUAL}| < \text{NOISE}$, то функция возвращает **0**.

Пример

```

VAR
  (* Значение уставки *)
  SetPoint : REAL := 100.0;          (* Заданная температура 100 °C *)

  (* Измеренное значение регулируемой величины *)
  ActualValue : REAL := 98.5;        (* Фактическая температура 98.5 °C *)

  (* Зона нечувствительности *)
  Noise : REAL := 2.0;              (* ±2 °C *)

  (* Результат - сигнал рассогласования *)
  ControlError : REAL;
END_VAR

(* Вызов функции CTRL_IN *)
ControlError := OSCAT_BASIC.CTRL_IN(
  SET_POINT := SetPoint,
  ACTUAL := ActualValue,
  NOISE := Noise
);

(*
  Результат: ControlError = 0.0

  Пояснение:
  |SET_POINT - ACTUAL| = |100.0 - 98.5| = 1.5
  |1.5| < NOISE (2.0) → функция возвращает 0

  Это означает, что отклонение находится в пределах зоны
  нечувствительности, и регулятор не должен реагировать.
*)
END_PROGRAM

```

26.5 CTRL_OUT

Тип модуля: функция	Переменная	Тип	Описание
Входы	CI	REAL	Выход регулятора
	OFFSET	REAL	Смещение
	MAN_IN	REAL	Сигнал ручного ввода
	LIM_L	REAL	Нижний предел управляющего сигнала
	LIM_H	REAL	Верхний предел управляющего сигнала
	MANUAL	BOOL	Режим управления
Выходы	Y	REAL	Величина управляющего воздействия
	LIM	BOOL	Флаг «достигнут предел для Y»

Функциональный блок **CTRL_OUT** используется для генерации управляющего воздействия. Если вход **MANUAL** имеет значение **FALSE**, система работает в автоматическом режиме управления, и на выход **Y** подается сумма значения выходного сигнала регулятора **CI** и смещения **OFFSET**; если вход **MANUAL** имеет значение **TRUE**, то система работает в режиме ручного управления и на выход **Y** подается сумма значения сигнала ручного ввода **MAN_IN** и смещения **OFFSET**. В обоих случаях значение выхода **Y** ограничивается значениями входов **LIM_L** (минимум) и **LIM_H** (максимум). При достижении любой из границ выход **LIM** принимает значение **TRUE**.

Пример

```

VAR      (* Экземпляр ФБ CTRL_OUT *)
  CtrlOutput : OSCAT_BASIC.CTRL_OUT;

  (* Входы *)
  RegulatorOutput : REAL := 75.0;      (* Выход регулятора (CI) *)
  Offset : REAL := 5.0;                (* Смещение (OFFSET) *)
  ManualInput : REAL := 50.0;          (* Ручной ввод (MAN_IN) *)
  LimitLow : REAL := 0.0;              (* Нижний предел (LIM_L) *)
  LimitHigh : REAL := 100.0;          (* Верхний предел (LIM_H) *)
  ManualMode : BOOL := FALSE;         (* Автоматический режим *)

  (* Выходы *)
  ControlValue : REAL;                (* Управляющее воздействие *)
  LimitReached : BOOL;               (* Флаг достижения предела *)
END_VAR

(* Вызов ФБ CTRL_OUT *)
CtrlOutput(
  CI := RegulatorOutput,
  OFFSET := Offset,
  MAN_IN := ManualInput,
  LIM_L := LimitLow,
  LIM_H := LimitHigh,
  MANUAL := ManualMode
);

(* Чтение выходов *)
ControlValue := CtrlOutput.Y;
LimitReached := CtrlOutput.LIM;

(*
  Результат:
  - MANUAL = FALSE → автоматический режим
  - Y = CI + OFFSET = 75.0 + 5.0 = 80.0
  - 80.0 находится в пределах [0.0, 100.0]
  - LIM = FALSE (предел не достигнут)

  Ответ:
  - ControlValue = 80.0
  - LimitReached = FALSE
*)
END_PROGRAM

```

26.6 CTRL_PI

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	ACT	REAL	Значение регулируемой величины
	SET	REAL	Значение уставки
	SUP	REAL	Зона нечувствительности
	OFS	REAL	Смещение.

Тип модуля: ФБ	Переменная	Тип	Описание
	M_I	REAL	Сигнал ручного ввода
	MAN	BOOL	Режим управления
	RST	BOOL	Сигнал обнуления И-составляющей
	KP	REAL	Коэффициент усиления П-составляющей
	KI	REAL	Коэффициент усиления И-составляющей
	LL	REAL	Нижний предел управляющего сигнала
	LH	REAL	Верхний предел управляющего сигнала
Выходы	Y	REAL	Величина управляющего воздействия
	DIFF	REAL	Сигнал рассогласования
	LIM	BOOL	Флаг «достигнут предел для Y»
Используемые модули	CTRL_IN, FT_PIWL, CTRL_OUT		

Функциональный блок **CTRL_PI** представляет собой ПИ-регулятор с автоматической компенсацией эффекта интегрального насыщения и возможностью перехода на ручное управление. Значение управляющего воздействия **Y** вычисляется по формуле:

$$Y = OFS + KP \cdot DIFF + KI \cdot \sum_{i=0}^n DIFF_i \cdot \Delta t_{изм}$$

, где

$DIFF = SET - ACT$;

$\Delta t_{изм}$ — время между двумя измерениями.

На вход **ACT** подается измеренное значение регулируемой величины, на вход **SET** — уставка регулятора. Вход **SUP** определяет зону нечувствительности для регулятора, которая используется, чтобы избежать перерегулирования. Вход **OFS** задает смещение управляющего сигнала. Входы **KP** и **KI** определяют коэффициенты усиления пропорциональной (П) и интегральной (И) составляющей регулятора. Входы **LL** и **LH** определяют нижний и верхний пределы управляющего сигнала **Y**. При достижении любой из границ выход **LIM** принимает значение **TRUE**, и значение выхода **Y** фиксируется. На выход **DIFF** подается сигнал рассогласования, вычисленный с помощью ФБ **CTRL_OUT**. По переднему фронту на входе **RST** происходит обнуление накопленной интегральной составляющей блока.

Если вход **MAN** имеет значение **TRUE**, то система работает в режиме ручного управления и на выход **Y** подается сумма значения сигнала ручного ввода **M_I** и смещения **OFS**.

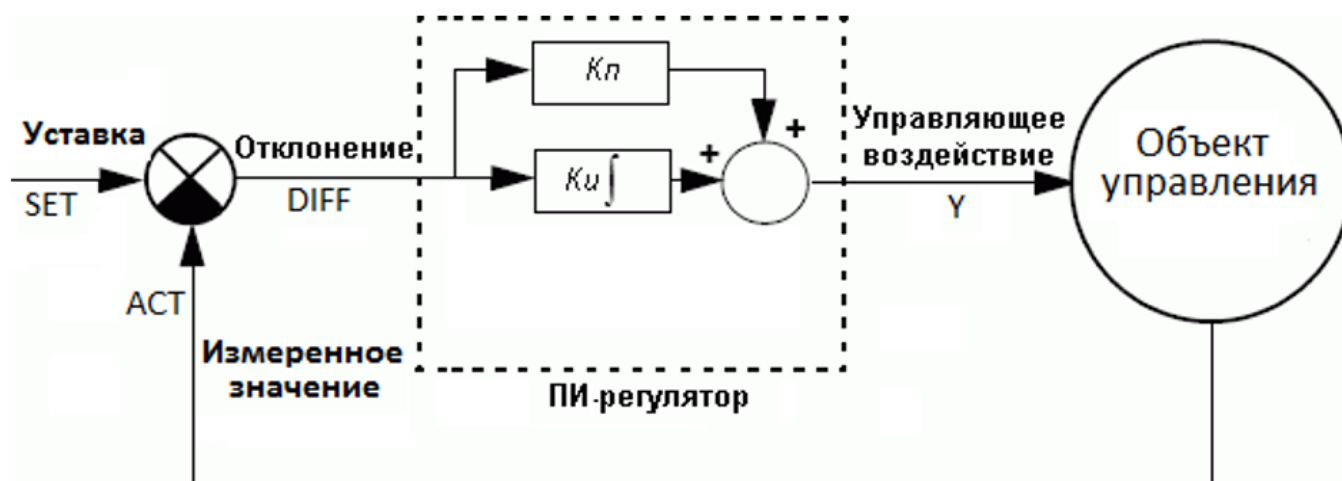


Рисунок 26.1 – Структурная схема ПИ-регулятора

26.7 CTRL_PID

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	ACT	REAL	Значение регулируемой величины
	SET	REAL	Значение уставки
	SUP	REAL	Зона нечувствительности
	OFS	REAL	Смещение
	M_I	REAL	Сигнал ручного ввода
	MAN	BOOL	Режим управления
	RST	BOOL	Сигнал обнуления И-составляющей
	KP	REAL	Коэффициент усиления П-составляющей
	TN	REAL	Постоянная времени интегрирования
	TV	REAL	Постоянная времени дифференцирования
	LL	REAL	Нижний предел управляющего сигнала
	LH	REAL	Верхний предел управляющего сигнала
Выходы	Y	REAL	Величина управляющего воздействия
	DIFF	REAL	Сигнал рассогласования
	LIM	BOOL	Флаг «достигнут предел для Y»
Используемые модули	CTRL_IN, FT_PIWL, CTRL_OUT		

Функциональный блок **CTRL_PID** представляет собой ПИД-регулятор с компенсацией эффекта интегрального насыщения и возможностью перехода на ручное управление. Значение управляющего воздействия Y вычисляется по формуле:

$$Y = OFS + KP \cdot DIFF + \frac{1}{TN} \cdot \sum_{i=0}^n DIFF_i \cdot \Delta t_{изм} + TV \cdot \frac{\Delta DIFF_i}{\Delta t_{изм}}$$

, где

$$\text{DIFF} = \text{SET} - \text{ACT};$$

$\Delta t_{\text{изм}}$ — время между двумя измерениями.

На вход **ACT** подается измеренное значение регулируемой величины, на вход **SET** — уставка регулятора. Вход **SUP** определяет зону нечувствительности для регулятора, которая используется, чтобы избежать перерегулирования. Вход **OFS** задает смещение управляющего сигнала. Вход **KP** и определяет коэффициент усиления пропорциональной (П) составляющей регулятора, входы **TN** и **TV** — постоянные времени интегрирования и дифференцирования (*обратите внимание*, они не могут быть равны 0 — это приведет к исключению в работе ПЛК). Входы **LL** и **LH** определяют нижний и верхний пределы управляющего сигнала **Y**. При достижении любой из границ выход **LIM** принимает значение **TRUE**, и значение выхода **Y** фиксируется. На выход **DIFF** подается сигнал рассогласования, вычисленный с помощью ФБ **CTRL_OUT**. По переднему фронту на входе **RST** происходит обнуление накопленной интегральной составляющей блока.

Если вход **MAN** имеет значение **TRUE**, то система работает в режиме ручного управления и на выход **Y** подается сумма значения сигнала ручного ввода **M_I** и смещения **OFS**.

Блок использует в качестве параметров постоянные времени интегрирования и дифференцирования **TV** и **TN**. При необходимости можно перейти от них к коэффициентам усиления:

$$KI = KP / TN$$

$$KD = KP \cdot TV$$

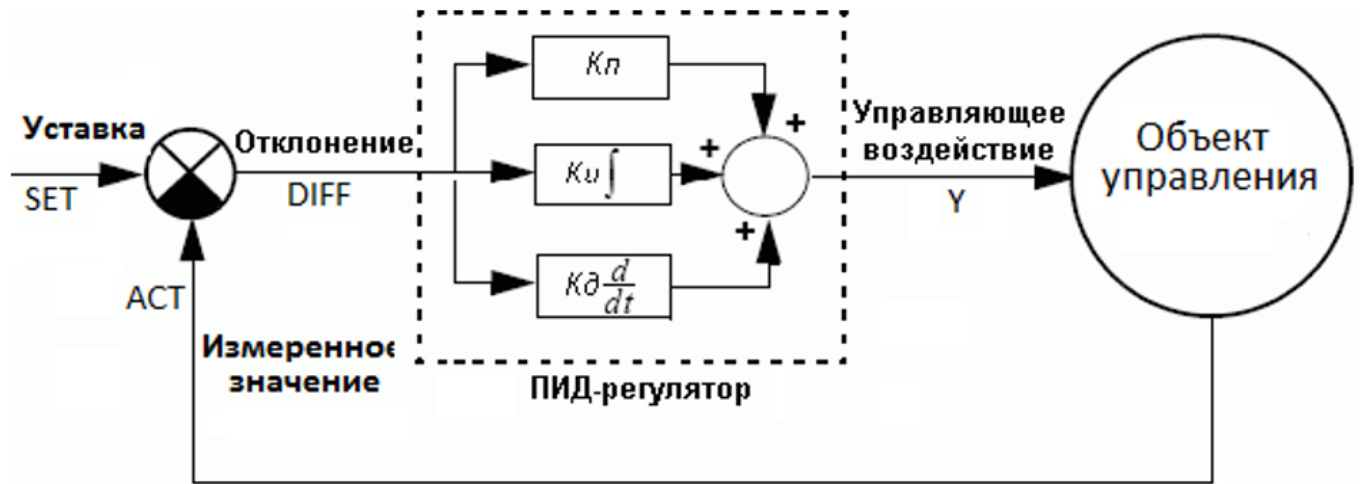


Рисунок 26.2 – Структурная схема ПИД-регулятора

26.8 CTRL_PWM

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	ACT	REAL	Значение регулируемой величины
	SET	REAL	Значение уставки
	SUP	REAL	Зона нечувствительности
	OFS	REAL	Смещение
Выходы	Y	REAL	Величина управляющего воздействия
Используемые модули	PWM_DC		

Функциональный блок **CTRL_PWM** представляем собой широтно-импульсный модулятор (ШИМ) с возможностью ручного управления. Если вход **MANUAL** имеет значение **FALSE**, то ШИМ находится в работе и на выход **Q** подается широтно-модулированный сигнал с частотой **F** и скважностью **CI** (0...1.0); если вход **MANUAL** имеет значение **TRUE**, то блок работает в режиме ручного управления и на выход **Q** подается значение входа **MAN_IN**.

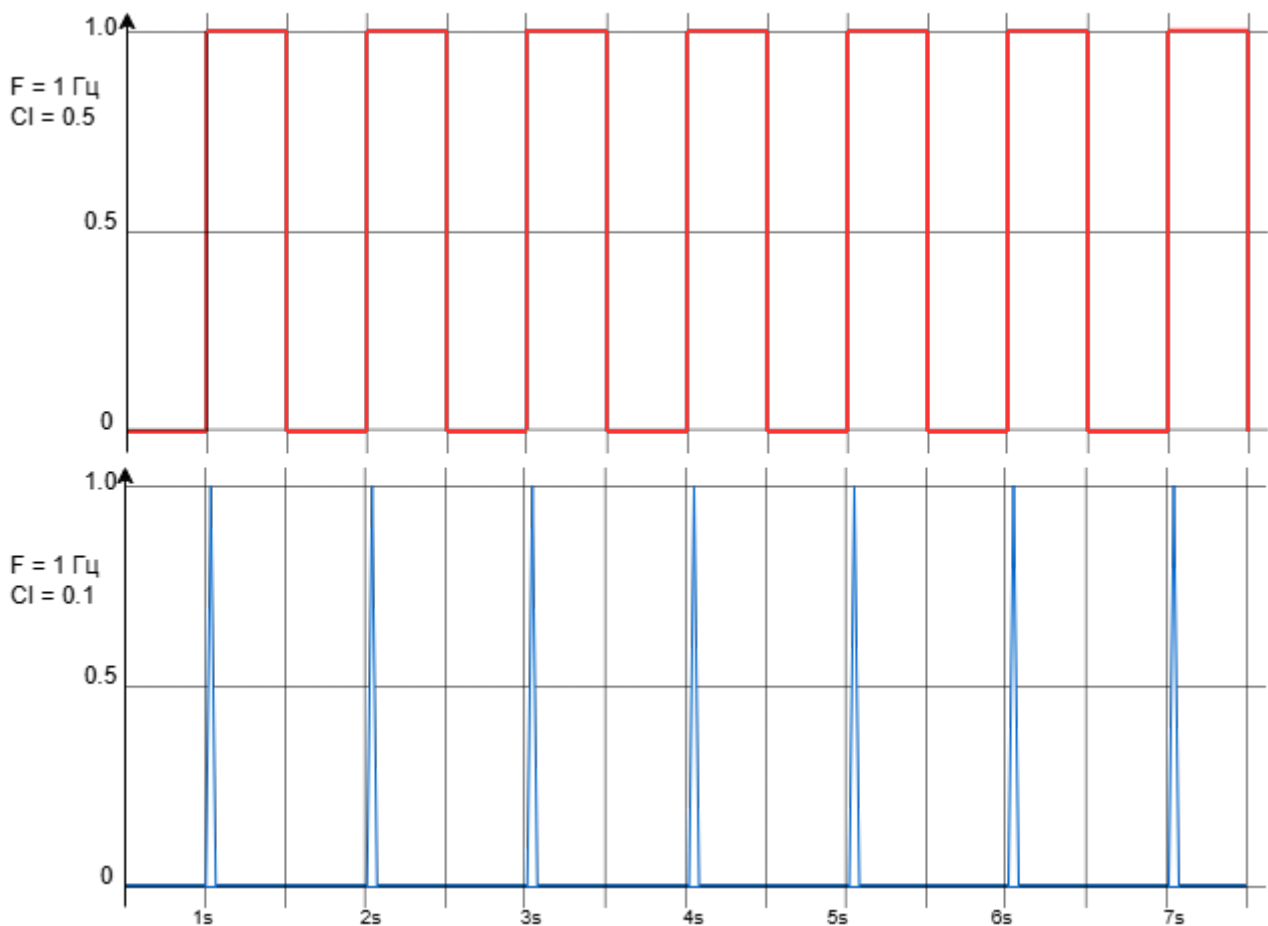


Рисунок 26.3 – Трассировка работы ФБ CTRL_PWM

26.9 DEAD_BAND

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Входное значение
	L	REAL	Смещение
Выходы	DEAD_BAND	REAL	Выходное значение

Функция **DEAD_BAND** ограничивает значение входа X по следующему принципу:

- DEAD_BAND = (X-L), если L < X;
- DEAD_BAND = (X+L), если X < (-L);
- DEAD_BAND = 0.0, если -L ≤ X ≤ L.

Пример

```

VAR
  (* Входное значение *)
  InputValue : REAL := 15.0;          (* 15.0 *)

  (* Смещение (зона нечувствительности) *)
  Offset : REAL := 5.0;              (* ±5.0 *)

  (* Результат *)
  Result : REAL;
END_VAR

(* Вызов функции DEAD_BAND *)
Result := OSCAT_BASIC.DEAD_BAND(
  X := InputValue,
  L := Offset
);

(*Результат: Result = 10.0*)
END_PROGRAM

```

26.10 DEAD_BAND_A

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	X	REAL	Входное значение
	T	TIME	Постоянная времени фильтра
	KL	REAL	Коэффициент усиления фильтра
	LM	REAL	Максимальная амплитуда высокочастотной составляющей сигнала
Выходы	Y	REAL	Отфильтрованное значение
	L	REAL	Смещение сигнала
Используемые модули	FT_PT1, MULTIME		

Функция **DEAD_BAND_A** ограничивает значение входа X по следующему принципу:

- $DEAD_BAND_A = (X - L)$, если $L < X$;
- $DEAD_BAND_A = (X + L)$, если $X < (-L)$;
- $DEAD_BAND_A = 0.0$, если $-L \leq X \leq L$.

Блок содержит фильтр нижних частот для входного сигнала **X** с постоянной времени **T**. Значение **L** определяется как произведение коэффициента усиления фильтра **KL** и текущей амплитуды высокочастотной составляющей входного сигнала; при этом максимальное значение **L** ограничено значением **LM**. Коэффициент **KL** определяет чувствительность фильтра; его значение рекомендуется выбирать из интервала (1.0...5.0).

Блок может использоваться в системах регулирования для фильтрации шума входного сигнала, что может являться причиной постоянной работы и как следствие исполнительных механизмов.

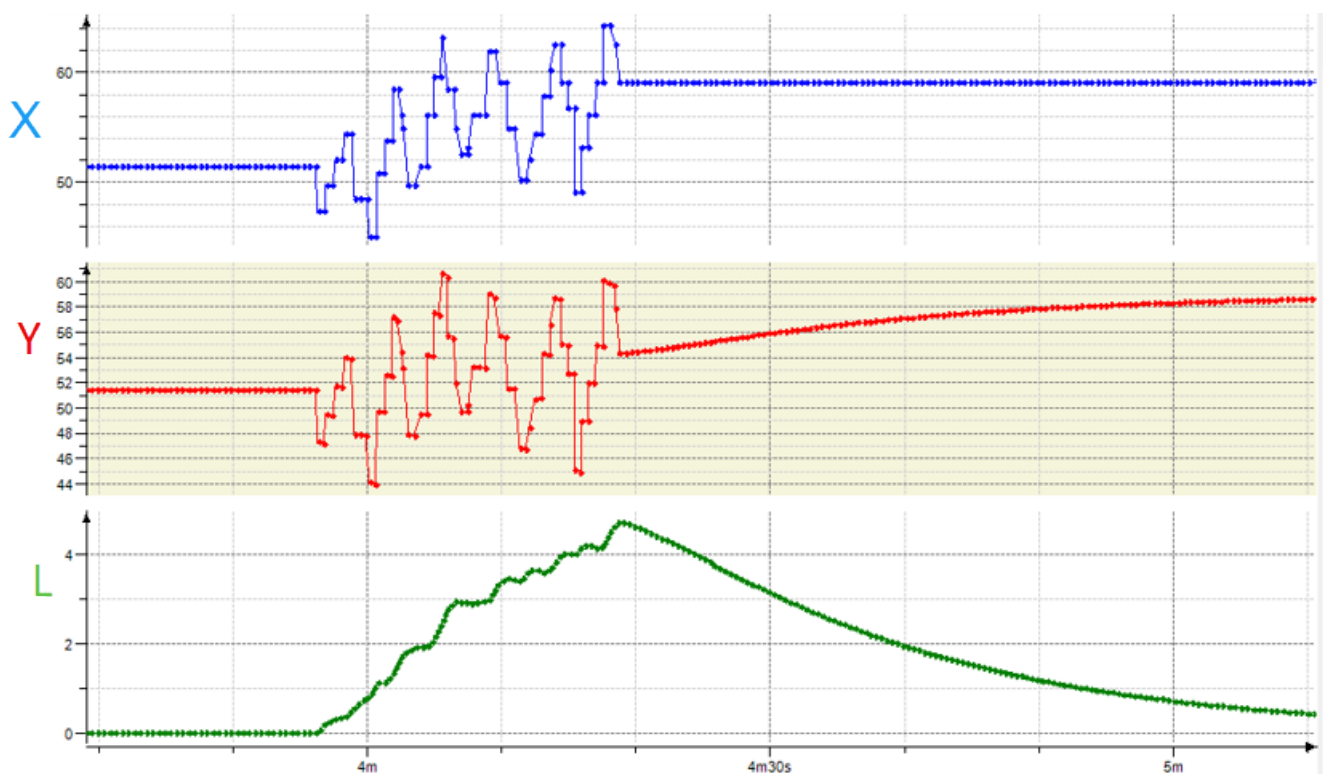


Рисунок 26.4 – Трассировка работы ФБ DEAD_BAND_A (T=T#5s, KL=2.0, LM=5.0)

26.11 DEAD_ZONE

Тип модуля: функция	Переменная	Тип	Описание
Входы	X	REAL	Входное значение
	L	REAL	Зона нечувствительности
Выходы	DEAD_ZONE	REAL	Выходное значение

Функция **DEAD_ZONE** представляет собой передаточную функцию с зоной нечувствительности. Значение входа **X** ограничивается по следующему принципу:

- $DEAD_ZONE = X$, если $L < |X|$;
- $DEAD_ZONE = 0.0$, если $|X| \leq L$.

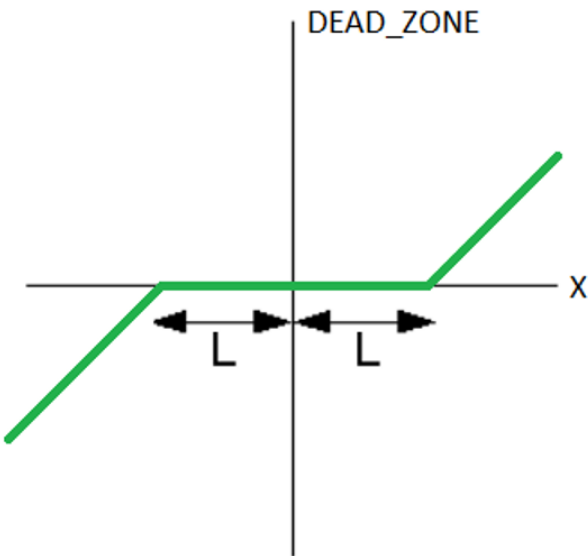


Рисунок 26.5 – Принцип работы функции DEAD_ZONE

26.12 DEAD_ZONE2

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	X	REAL	Входное значение
	L	REAL	Зона нечувствительности
Выходы	Y	REAL	Выходное значение

Функциональный блок **DEAD_ZONE2** представляет собой передаточную функцию с зоной нечувствительности и гистерезисом. Значение входа **X** ограничивается по следующему принципу:

- $DEAD_ZONE2 = X$, если $L < |X|$;
- $DEAD_ZONE2 = L$, если $0 \leq |X| \leq L$;
- $DEAD_ZONE2 = -L$, если $-L \leq |X| \leq 0$.

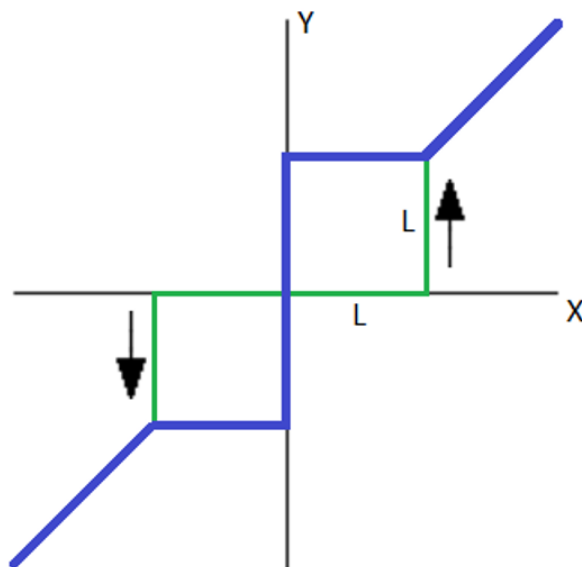


Рисунок 26.6 – Принцип работы ФБ DEAD_ZONE2

26.13 FT_DERIV

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	in	REAL	Входное значение
	K	REAL	Коэффициент передачи
	run	BOOL	Сигнал управление блоком
Выходы	out	REAL	Производная входного сигнала
Используемые модули	T_PLC_US		

Функциональный блок **FT_DERIV** представляет собой Д-звено. Если вход **run** имеет значение **TRUE**, то на выход **out** поступает производная входного сигнала **X** с коэффициентом усиления **K**.

$$out = K \cdot (\Delta in / \Delta T)$$

, где ΔT — время цикла ПЛК секундах.

26.14 FT_IMP

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	in	REAL	Входное значение
	T	TIME	Постоянная времени
	K	REAL	Коэффициент усиления
Выходы	out	REAL	Отфильтрованное значение
Используемые модули	FT_PT1		

Функциональный блок **FT_IMP** представляет собой фильтр верхних частот с коэффициентом усиления **K** и постоянной времени **T**.

После изменения входного значения **IN**, выход **out** изменяется по следующему принципу:

- $out(0)=K \cdot IN$;
- $out(T) \approx 0.37 \cdot K \cdot IN$;
- $out(3T) \approx 0.05 \cdot K \cdot IN$;
- $out(\dots)=0$.

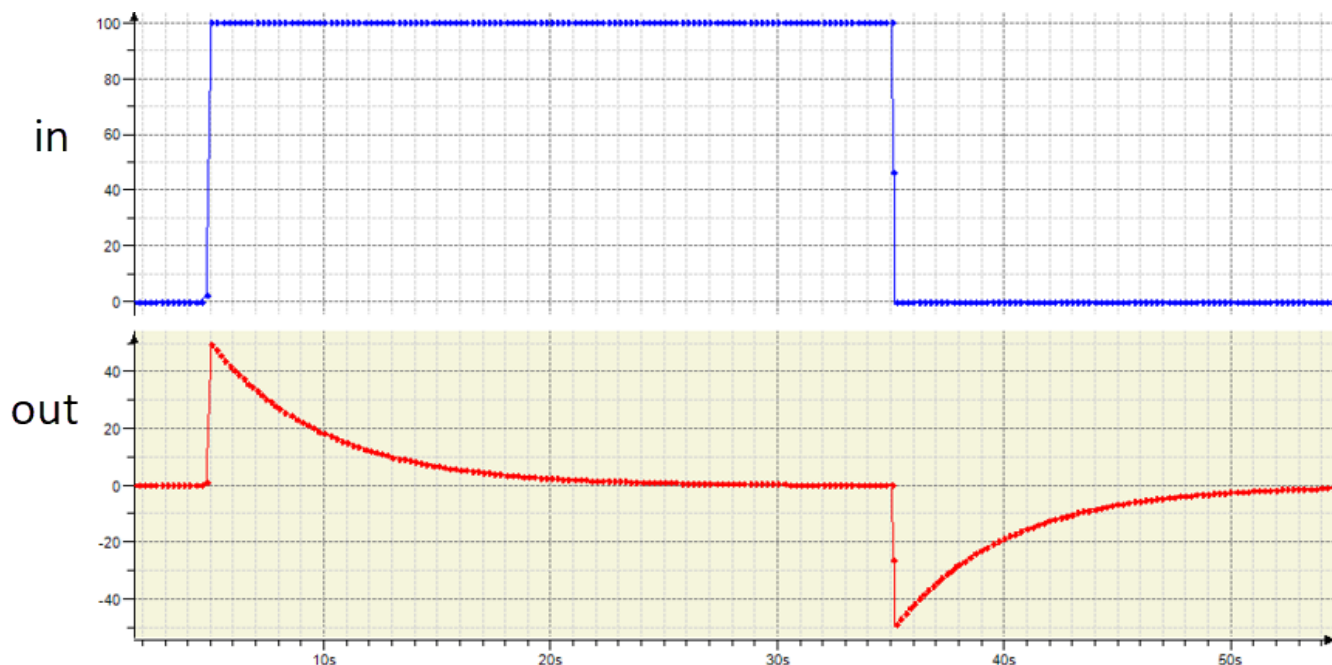


Рисунок 26.7 – Трассировка работы ФБ FT_IMP ($K=0.5$, $T=T\#5s$)

26.15 FT_INT

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	REAL	Входное значение
	K	REAL	Коэффициент передачи
	RUN	BOOL	Сигнал управление блоком
	RST	BOOL	Сигнал сброса блока
	OUT_MIN	REAL	Нижний предел выходного сигнала
	OUT_MAX	REAL	Верхний предел выходного сигнала
Выходы	OUT	REAL	Значение интеграла
	LIM	BOOL	Флаг «достигнут предел»
Используемые модули	INTEGRATE		

Функциональный блок **FT_INT** представляет собой И-звено. Если вход **RUN** имеет значение **TRUE**, то на выход **out** поступает интеграл входного сигнала **IN** с коэффициентом усиления **K**. Для расчета интеграла используется метод трапеций:

$$OUT(n) = OUT(n-1) + K \cdot (IN(n) + IN(n-1)) / 2 \cdot \Delta T$$

, где n – номер текущего цикла ПЛК, ΔT – время цикла ПЛК секундах.

Входы **OUT_MIN** и **OUT_MAX** определяют нижний и верхний пределы выхода **OUT**. При достижении любой из границ выход **LIM** принимает значение **TRUE**. По переднему фронту на входе **RST** происходит обнуление выходов блока.

Типичной проблемой интегратора является его точность. Точность переменных типа **REAL**, используемых в данном блоке, ограничена 7-8 знаками после запятой; в результате, если среднее значение входа **X** и его изменение отличаются более чем на 7 порядков (например, $X=1000000$, $\Delta X=0.0001$), значение интеграла не будет меняться. См. блок **FT_INT2**, использующий числа двойной точности.

26.16 FT_INT2

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	REAL	Входное значение
	K	REAL	Коэффициент передачи
	RUN	BOOL	Сигнал управление блоком
	RST	BOOL	Сигнал сброса блока
	OUT_MIN	REAL	Нижний предел выходного сигнала
	OUT_MAX	REAL	Верхний предел выходного сигнала
Выходы	OUT	REAL	Значение интеграла
	LIM	BOOL	Флаг «достигнут предел»
Используемые модули	INTEGRATE, R2_SET, R2_ADD		

Функциональный блок **FT_INT2** представляет собой И-звено. Принцип работы блока полностью соответствует блоку **FT_INT**, единственным отличием является использование чисел двойной точности для промежуточных вычислений (более подробно см. в описании блока **FT_INT**).

26.17 FT_PD

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	REAL	Сигнал рассогласования
	KP	REAL	Коэффициент передачи
	TV	REAL	Постоянная времени дифференцирования
Выходы	Y	REAL	Величина управляющего воздействия
Используемые модули	FT_DERIV		

Функциональный блок **FT_PD** представляет собой блок ПД-регулирования. Значение управляющего воздействия **Y** вычисляется по формуле:

$$Y = KP \cdot (IN + FT_DERIV.OUT(IN, TV))$$

Блок может использоваться совместно с модулями **CTRL_IN** и **CTRL_OUT** для реализации ПД-регулятора.

26.18 FT_PDT1

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	REAL	Сигнал рассогласования
	KP	REAL	Коэффициент передачи
	TV	REAL	Постоянная времени дифференцирования
	T1	REAL	Запаздывание
Выходы	Y	REAL	Величина управляющего воздействия
Используемые модули	FT_DERIV, FT_PT1		

Функциональный блок **FT_PDT1** представляет собой блок ПД-регулирования с апериодическим звеном 1-го порядка. Значение управляющего воздействия **Y** вычисляется по формуле:

$$Y = KP \cdot (IN + FT_PT1.OUT(FT_DERIV.OUT(IN, TV), T1))$$

Блок может использоваться совместно с модулями **CTRL_IN** и **CTRL_OUT** для реализации ПД-регулятора.

26.19 FT_PI

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	REAL	Сигнал рассогласования
	KP	REAL	Коэффициент усиления П-составляющей
	KI	REAL	Коэффициент усиления И-составляющей
	ILIM_L	REAL	Нижний предел зоны накопления интеграла
	ILIM_H	REAL	Верхний предел зоны накопления интеграла
	IEN	BOOL	Сигнал управления интегратором
	RST	BOOL	Сигнал обнуления И-составляющей
Выходы	Y	REAL	Величина управляющего воздействия
	LIM	BOOL	Флаг «достигнут предел»
Используемые модули	FT_INT		

Функциональный блок **FT_PI** представляет собой блок ПИ-регулирования. Значение управляющего воздействия **Y** вычисляется по формуле:

$$Y = KP \cdot IN + KI \cdot \sum_{i=0}^n IN_i \cdot \Delta t_{изм}$$

, где $\Delta t_{изм}$ время между двумя измерениями.

На вход **IN** подается сигнал рассогласования. Входы **KP** и **KI** определяют коэффициенты усиления пропорциональной (П) и интегральной (И) составляющей регулятора. Входы **ILIM_L** и **ILIM_H** определяют нижний и верхний пределы зоны накопления интеграла. При достижении любой из границ выход **LIM** принимает значение **TRUE**, и значение интегратора перестает увеличиваться. Кроме того, интегратор в любой момент может быть отключен с помощью присвоения значения **FALSE** входу **IEN**. По переднему фронту на входе **RST** происходит обнуление накопленной интегральной составляющей блока.

26.20 FT_PID

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	REAL	Сигнал рассогласования
	KP	REAL	Коэффициент усиления П-составляющей
	TN	REAL	Постоянная времени интегрирования
	TV	REAL	Постоянная времени дифференцирования
	ILIM_L	REAL	Нижний предел зоны накопления интеграла
	ILIM_H	REAL	Верхний предел зоны накопления интеграла
	IEN	BOOL	Сигнал управления интегратором
	RST	BOOL	Сигнал обнуления И-составляющей

Тип модуля: ФБ	Переменная	Тип	Описание
Выходы	Y	REAL	Величина управляющего воздействия
	LIM	BOOL	Флаг «достигнут предел»
Используемые модули	FT_INT, FT_DERIV		

НАДО ЛИ ПИСАТЬ ЧТО БЛОК РАБОТАЕТ НЕКОРРЕКТНО, КАК В ПЕРВОИСТОЧНИКЕ? ИЛИ У НАС — КОРРЕКТНО?

Функциональный блок **FT_PID** представляет собой блок ПИД-регулирования. Значение управляющего воздействия **Y** вычисляется по формуле:

$$Y = KP \cdot IN + \frac{1}{TN} \cdot \sum_{i=0}^n IN_i \cdot \Delta t_{изм} + TV \cdot \frac{\Delta IN_i}{\Delta t_{изм}}$$

, где $\Delta t_{изм}$ — время между двумя измерениями.

На вход **IN** подается сигнал рассогласования. Вход **KP** и определяет коэффициент усиления пропорциональной (П) составляющей регулятора, входы **TN** и **TV** – постоянные времени интегрирования и дифференцирования (**обратите внимание**, они не могут быть равны **0** – это приведет к исключению в работе ПЛК). Входы **ILIM_L** и **ILIM_H** определяют нижний и верхний пределы зоны накопления интеграла. При достижении любой из границ выход **LIM** принимает значение **TRUE**, и значение интегратора перестает увеличиваться. Кроме того, интегратор в любой момент может быть отключен с помощью присвоения значения **FALSE** входу **IEN**. По переднему фронту на входе **RST** происходит обнуление накопленной интегральной составляющей блока.

26.21 FT_PIDW

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	REAL	Сигнал рассогласования
	KP	REAL	Коэффициент усиления П-составляющей
	TN	REAL	Постоянная времени интегрирования
	TV	REAL	Постоянная времени дифференцирования
	LIM_L	REAL	Нижний предел управляющего сигнала
	LIM_H	REAL	Верхний предел управляющего сигнала
	RST	BOOL	Сигнал обнуления И-составляющей
Выходы	Y	REAL	Величина управляющего воздействия
	LIM	BOOL	Флаг «достигнут предел»
Используемые модули	INTEGRATE, FT_DERIV		

Функциональный блок **FT_PIDW** представляет собой блок ПИД-регулирования с компенсацией эффекта интегрального насыщения. Значение управляющего воздействия **Y** вычисляется по формуле:

$$Y = KP \cdot IN + \frac{1}{TN} \cdot \sum_{i=0}^n IN_i \cdot \Delta t_{изм} + TV \cdot \frac{\Delta IN_i}{\Delta t_{изм}}$$

, где $\Delta t_{изм}$ — время между двумя измерениями.

На вход **IN** подается сигнал рассогласования. Вход **KP** и определяет коэффициент усиления пропорциональной (П) составляющей регулятора, входы **TN** и **TV** – постоянные времени интегрирования и дифференцирования (**обратите внимание**, они не могут быть равны **0** – это приведет к исключению в работе ПЛК). Входы **LIM_L** и **LIM_H** определяют нижний и верхний пределы управляющего сигнала. При достижении любой из границ выход **LIM** принимает значение **TRUE**, и значение выхода **Y** фиксируется. По переднему фронту на входе **RST** происходит обнуление накопленной интегральной составляющей блока.

26.22 FT_PIDWL

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	REAL	Сигнал рассогласования
	KP	REAL	Коэффициент усиления П-составляющей
	TN	REAL	Постоянная времени интегрирования.
	TV	REAL	Постоянная времени дифференцирования
	LIM_L	REAL	Нижний предел управляющего сигнала
	LIM_H	REAL	Верхний предел управляющего сигнала
	RST	BOOL	Сигнал обнуления И-составляющей
Выходы	Y	REAL	Величина управляющего воздействия
	LIM	BOOL	Флаг «достигнут предел»
Используемые модули	FT_PIWL, FT_DERIV		

Функциональный блок **FT_PIDWL** представляет собой блок ПИД-регулирования с компенсацией эффекта интегрального насыщения. Значение управляющего воздействия **Y** вычисляется по формуле:

$$Y = KP \cdot IN + \frac{1}{TN} \cdot \sum_{i=0}^n IN_i \cdot \Delta t_{изм} + TV \cdot \frac{\Delta IN_i}{\Delta t_{изм}}$$

, где $\Delta t_{изм}$ — время между двумя измерениями.

Принцип работы блока полностью соответствует блоку **FT_PIDW**. Единственным отличием является набор модулей, использованных для реализации блока.

26.23 FT_PIW

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	REAL	Сигнал рассогласования
	KP	REAL	Коэффициент усиления П-составляющей
	KI	REAL	Коэффициент усиления И-составляющей
	LIM_L	REAL	Нижний предел управляющего сигнала
	LIM_H	REAL	Верхний предел управляющего сигнала
	RST	BOOL	Сигнал обнуления И-составляющей
Выходы	Y	REAL	Величина управляющего воздействия

Тип модуля: ФБ	Переменная	Тип	Описание
	LIM	BOOL	Флаг «достигнут предел»
Используемые модули	FT_INT		

Функциональный блок **FT_PIW** представляет собой блок ПИ-регулирования с компенсацией эффекта интегрального насыщения. Значение управляющего воздействия **Y** вычисляется по формуле:

$$Y = KP \cdot IN + KI \cdot \sum_{i=0}^n IN_i \cdot \Delta t_{изм}$$

, где $\Delta t_{изм}$ — время между двумя измерениями.

На вход **IN** подается сигнал рассогласования. Входы **KP** и **KI** определяют коэффициенты усиления пропорциональной (П) и интегральной (И) составляющей регулятора. Входы **LIM_L** и **LIM_H** определяют нижний и верхний пределы управляющего сигнала. При достижении любой из границ выход **LIM** принимает значение **TRUE**, и значение выхода **Y** фиксируется. По переднему фронту на входе **RST** происходит обнуление накопленной интегральной составляющей блока.

26.24 FT_PIWL

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	REAL	Сигнал рассогласования
	KP	REAL	Коэффициент усиления П-составляющей.
	KI	REAL	Коэффициент усиления И-составляющей.
	LIM_L	REAL	Нижний предел управляющего сигнала.
	LIM_H	REAL	Верхний предел управляющего сигнала.
	RST	BOOL	Сигнал обнуления И-составляющей.
Выходы	Y	REAL	Величина управляющего воздействия
	LIM	BOOL	Флаг «достигнут предел»
Используемые модули	T_PLC_US		

Функциональный блок **FT_PIWL** представляет собой блок ПИ-регулирования с компенсацией эффекта интегрального насыщения. Значение управляющего воздействия **Y** вычисляется по формуле:

$$Y = KP \cdot IN + KI \cdot \sum_{i=0}^n IN_i \cdot \Delta t_{изм}$$

, где $\Delta t_{изм}$ — время между двумя измерениями.

Принцип работы блока полностью соответствует блоку **FT_PIW**. Отличием является внутренняя реализация блока, в которой не используются другие модули регуляторов.

26.25 FT_PT1

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	REAL	Входное значение
	T	TIME	Постоянная времени
	K	REAL	Коэффициент усиления

Тип модуля: ФБ	Переменная	Тип	Описание
Выходы	out	REAL	Выходное значение
Используемые модули	T_PLC_US		

Функциональный блок **FT_PT1** представляет собой апериодическое звено 1-го порядка с коэффициентом усиления **K** и постоянной времени **T**.

После изменения входного значения **IN**, выход **out** изменяется по следующему принципу:

- $out(0)=0$;
- $out(T) \approx 0.63 \cdot K \cdot IN$;
- $out(3T) \approx 0.95 \cdot K \cdot IN$;
- $out(\dots) = K \cdot IN$.

26.26 FT_PT2

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	REAL	Входное значение
	T	TIME	Постоянная времени
	D	REAL	Коэффициент затухания
	K	REAL	Коэффициент усиления
Выходы	out	REAL	Выходное значение
Используемые модули	INTEGRATE		

Функциональный блок **FT_PT2** представляет собой апериодическое звено 2-го порядка с коэффициентом усиления **K**, коэффициентом затухания **D** и постоянной времени **T**.

После изменения входного значения **IN**, выход **out** изменяется по следующему принципу:

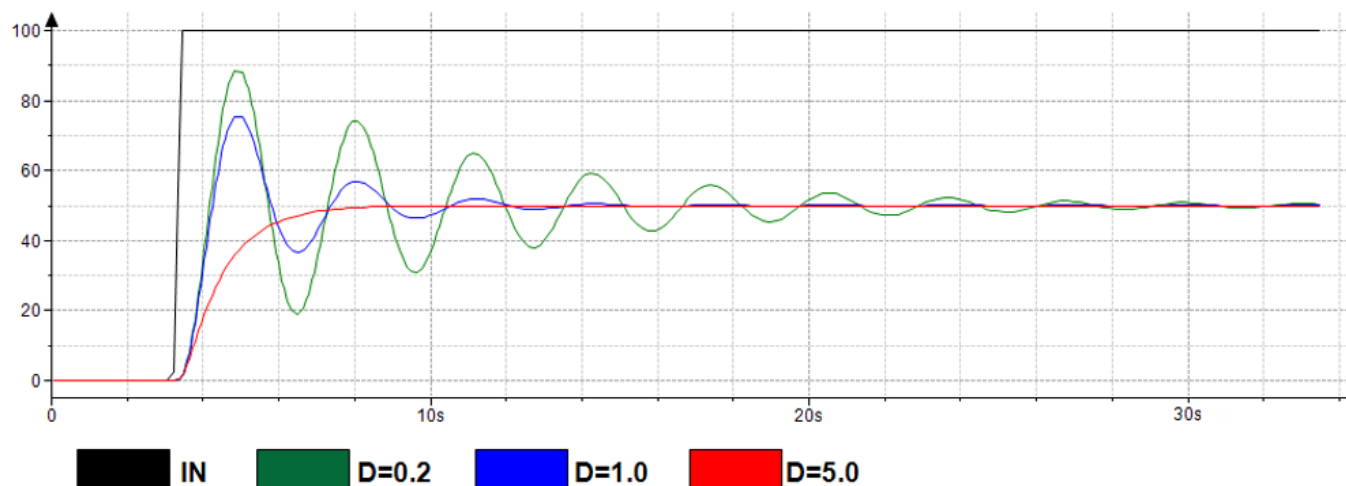


Рисунок 26.8 – Трассировка работы ФБ FT_PT2 (K=0.5, T=T#500ms)

26.27 FT_TN8

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	REAL	Входное значение
	T	TIME	Время задержки
Выходы	out	REAL	Выходное значение
	trig	BOOL	Флаг «сохранение значения»
Используемые модули	T_PLC_MS		

Функциональный блок **FT_TN8** представляет собой модуль задержки сигнала. Входное значение **IN** сохраняется с интервалом $T/8$, при каждом сохранении выход **out** принимает значение **TRUE** на один цикл ПЛК. Блок сохраняет 8 значений переменной **IN**. Эти значения подаются на выход **out** спустя время задержки **T**. См. также блоки **FT_TN16** и **FT_TN64**, которые отличаются частотой дискретизации входного значения.

26.28 FT_TN16

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	REAL	Входное значение
	T	TIME	Время задержки
Выходы	out	REAL	Выходное значение
	trig	BOOL	Флаг «сохранение значения»
Используемые модули	T_PLC_MS		

Функциональный блок **FT_TN16** представляет собой модуль задержки сигнала. Входное значение **IN** сохраняется с интервалом **T/16**, при каждом сохранении выход **out** принимает значение **TRUE** на один цикл ПЛК. Блок сохраняет 16 значений переменной **IN**. Эти значения подаются на выход **out** спустя время задержки **T**. См. также блоки **FT_TN64** и **FT_TN8**, которые отличаются частотой дискретизации входного значения.

26.29 FT_TN64

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	REAL	Входное значение
	T	TIME	Время задержки
Выходы	out	REAL	Выходное значение
	trig	BOOL	Флаг «сохранение значения»
Используемые модули	T_PLC_MS		

Функциональный блок **FT_TN64** представляет собой модуль задержки сигнала. Входное значение **IN** сохраняется с интервалом **T/64**, при каждом сохранении выход **out** принимает значение **TRUE** на один цикл ПЛК. Блок сохраняет 64 значения переменной **IN**. Эти значения подаются на выход **out** спустя время задержки **T**. См. также блоки **FT_TN16** и **FT_TN8**, которые отличаются частотой дискретизации входного значения.

26.30 HYST

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	REAL	Входное значение
	ON	REAL	Порог включения ИМ
	OFF	REAL	Порог отключения ИМ
Выходы	Q	BOOL	Сигнал управления
	win	BOOL	Флаг «значение в допустимых границах»

Функциональный блок **HYST** представляет собой двухпозиционный регулятор. На вход **IN** подается измеренное значение, входы **ON** и **OFF** определяют порог включения и отключения исполнительного механизма (ИМ). Значение выхода **Q** определяется по следующему принципу:

1. $ON \geq OFF$
 - Q=FALSE, если $IN < OFF$;
 - Q=TRUE, если $IN > ON$.
2. $ON < OFF$
3. Q=FALSE, если $IN > OFF$;
4. Q=TRUE, если $IN < ON$.

Выход **WIN** принимает значение **TRUE**, если **IN** принадлежит интервалу **[OFF, ON]**.

26.31 HYST_1

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	REAL	Входное значение

Тип модуля: ФБ	Переменная	Тип	Описание
	high	REAL	Порог включения ИМ
	low	REAL	Порог отключения ИМ
Выходы	Q	BOOL	Сигнал управления
	win	BOOL	Флаг «значение в допустимых границах»

Функциональный блок **HYST_1** представляет собой двухпозиционный регулятор. На вход **IN** подается измеренное значение, входы **high** и **low** определяют порог включения и отключения исполнительного механизма (при этом значение **high** должно быть больше значения **low**). Значение выхода **Q** определяется по следующему принципу:

- $Q = \text{FALSE}$, если $IN < \text{low}$;
- $Q = \text{TRUE}$, если $IN > \text{high}$;
- Q сохраняет предыдущее значение, если $\text{low} \leq IN \leq \text{high}$.

Выход **WIN** принимает значение **TRUE**, если **IN** принадлежит интервалу [**OFF**, **ON**].

26.32 HYST_2

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	REAL	Входное значение
	VAL	REAL	Граница гистерезиса
	HYS	REAL	Ширина гистерезиса
Выходы	Q	BOOL	Сигнал управления
	win	BOOL	Флаг «значение в допустимых границах»

Функциональный блок **HYST_2** представляет собой двухпозиционный регулятор с зоной гистерезиса. На вход **IN** подается измеренное значение, входы **VAL** и **HYS** определяют границу гистерезиса и его ширину. Значение выхода **Q** определяется по следующему принципу:

- $Q = \text{FALSE}$, если $IN < (VAL - 0.5 \cdot HYS)$;
- $Q = \text{TRUE}$, если $IN > (VAL + 0.5 \cdot HYS)$;
- Q сохраняет предыдущее значение, если $(VAL - 0.5 \cdot HYS) \leq IN \leq (VAL + 0.5 \cdot HYS)$.

Выход **WIN** принимает значение **TRUE**, если **IN** принадлежит интервалу [**VAL** – 0.5·HYS, **VAL** + 0.5·HYS].

26.33 HYST_3

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	REAL	Входное значение
	hyst	REAL	Ширина гистерезиса
	val1	REAL	Граница гистерезиса 1
	val2	REAL	Граница гистерезиса 2
Выходы	Q1	BOOL	Сигнал управления «ход вправо»
	Q2	BOOL	Сигнал управления «ход влево»

Функциональный блок **HYST_3** представляет собой трехпозиционный регулятор с двумя зонами гистерезиса. На вход **IN** подается измеренное значение, входы **val1**, **val2** и **hyst** определяют границы гистерезиса и его ширину. Значение выходов **Q1** и **Q2** определяется по следующему принципу:

- $Q1 = \text{TRUE}$, если $IN < (val1 - 0.5 \cdot hyst)$;
- $Q1 = \text{FALSE}$, если $IN > (val1 + 0.5 \cdot hyst)$;
- $Q2 = \text{FALSE}$, если $IN < (val2 - 0.5 \cdot hyst)$;
- $Q2 = \text{TRUE}$, если $IN > (val2 + 0.5 \cdot hyst)$.

Пояснить принцип работы блока можно на следующем примере. Пусть **IN** – измеренное значение температуры, **U1** и **U2** – нижняя и верхняя уставки температуры, **Q1** и **Q2** – реле управления задвижкой (**Q1** – открытие, **Q2** – закрытие). Значения **U1** и **U2** определяются следующим образом:

- $U1 = (val1 - 0.5 \cdot hyst)$;
- $U2 = (val2 + 0.5 \cdot hyst)$.

Если текущее значение температуры **IN** меньше уставки **U1**, то замыкается реле **Q1**, заставляя задвижку открываться и увеличивать количество теплоносителя. Как только значение температуры превысит (**U1 + hyst**), реле **Q1** разомкнется и задвижка останется в том промежуточном положении, которое успело достигнуть за время **t1**. Если затем значение температуры по какой-либо причине превысит уставку **U2**, сработает реле **U2** и задвижка начнет закрываться до тех пор, пока температура не вернется в диапазон между значениями [**U1+hyst, U2-hyst**].

26.34 INTEGRATE

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	E	BOOL	Сигнал управление блоком
	X	REAL	Входное значение
	K	REAL	Коэффициент передачи
Выходы	Y	REAL	Значение интеграла
Используемые модули	T_PLC_MS		

Функциональный блок **INTEGRATE** представляет собой И-звено. Если вход **E** имеет значение **TRUE**, то на выход **Y** поступает интеграл входного сигнала **X** с коэффициентом усиления **K**. Для расчета интеграла используется метод трапеций:

$$OUT(n) = OUT(n - 1) + K \cdot \frac{IN(n) + IN(n - 1)}{2} \cdot \Delta T,$$

, где n-номер текущего цикла ПЛК, ΔT-время цикла ПЛК секундах.

27 Модули управления

27.1 DRIVER_1

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	SET	BOOL	Ручное управление выходом
	IN	BOOL	Сигнал нажатия на кнопку
	RST	BOOL	Переключение выхода в FALSE
Выходы	Q	BOOL	Выход кнопки
Параметры	Toggle_Mode	BOOL	Режим работы кнопки
	Timeout	TIME	Таймаут сброса выхода

Функциональный блок **DRIVER_1** представляет собой модуль обработки нажатия кнопки. Если параметр **Toggle_Mode** имеет значение **FALSE**, то по переднему фронту на входе **IN** выход **Q** принимает значение **TRUE** на время **Timeout**, после чего выход будет автоматически сброшен в состояние **FALSE**. В течение этого времени вход **IN** не обрабатывается. Если параметр **Toggle_Mode** имеет значение **TRUE**, то по переднему фронту на входе **IN** значение выхода **Q** инвертируется. Параметр **Timeout** определяет время, через которое выход **Q** будет автоматически сброшен в состояние **FALSE** (если выход имеет значение **TRUE**). Для отключения таймаута необходимо задать **Timeout= T#0ms**.

Вход **SET** позволяет управлять выходом **Q** независимо от состояния входа **IN**. По переднему фронту на входе **RST** выход **Q** принимает значение **FALSE** независимо от состояния входов **SET** и **IN**. См. также ФБ **DRIVER_4**, позволяющий обрабатывать нажатия 4-х кнопок.

27.2 DRIVER_4

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	SET	BOOL	Ручное управление выходом
	IN0...IN3	BOOL	Сигнал нажатия на кнопку 0...3
	RST	BOOL	Переключение выхода в FALSE
Выходы	Q0...Q3	BOOL	Выход кнопки 0...3
Параметры	Toggle_Mode	BOOL	Режим работы кнопки
	Timeout	TIME	Таймаут сброса выходов

Функциональный блок **DRIVER_4** представляет собой модуль обработки нажатий для 4-х кнопок. Принцип работы блока полностью соответствует ФБ **DRIVER_1**, единственным отличием является число обрабатываемых кнопок.

27.3 DRIVER_4C

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	BOOL	Сигнал переключения выходов
	RST	BOOL	Сигнал сброса блока
Выходы	SN	INT	Номер текущего состояния
	Q0...Q3	BOOL	Выходы блока
Параметры	Timeout	TIME	Таймаут переключения в состояние 0
	SX	ARRAY [1..7] OF BYTE	Массив битовых масок выходов

Функциональный блок **DRIVER_4C** представляет собой модуль управления с 7-ю заданными состояниями выходов, которые устанавливаются в параметре **SX** в виде битовых масок. Выход **SN** определяет номер текущего состояния. По умолчанию модуль находится в состоянии **0**, при этом **SN=0**, а битовая маска выходов = **2#0000**. По переднему фронту на входе **IN** происходит переключение выходов в состояние **SN=1**, при котором битовая маска выходов = **SX[1]** (где **Q0** – младший бит битовой маски). При последующих передних фронтах на входе **IN** модуль будет последовательно переключаться в состояния **2...7**, и после 7-ого состояния вернется в нулевое. Если одна из битовых масок массива **SX** равна **2#0000**, то по ее достижению модуль вернется в состояние **SN=0**. Значение параметра **Timeout** ограничивает время, которое модуль может провести в состояниях, отличных от нулевого. По истечении этого времени модуль вернется в состояние **SN=0**. Для отключения таймаута необходимо задать **Timeout=T#0ms**. По переднему фронту на входе **RST** происходит принудительное переключение модуля в состояние **SN=0**.

Рассмотрим пример работы с модулем. Пусть **Timeout=T#0ms**, **SX=[2#0001, 2#0011, 2#0111, 2#0000, 2#1111, 2#1100, 2#1110]**.

Сигнал управления	SN	Q0	Q1	Q2	Q3
-	0	FALSE	FALSE	FALSE	FALSE
Передний фронт по IN	1	TRUE	FALSE	FALSE	FALSE
Передний фронт по IN	2	TRUE	TRUE	FALSE	FALSE
Передний фронт по IN	3	TRUE	TRUE	TRUE	FALSE
Передний фронт по IN	0	FALSE	FALSE	FALSE	FALSE
Передний фронт по IN	1	TRUE	FALSE	FALSE	FALSE
Передний фронт по RST	0	FALSE	FALSE	FALSE	FALSE

27.4 FLOW_CONTROL

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	BOOL	Сигнал ручного управления
	REQ	BOOL	Сигнал автоматического управления
	ENQ	BOOL	Разрешение управления
	RST	BOOL	Сигнал сброса
Выходы	Q	BOOL	Сигнал управления клапаном
	STATUS	BYTE	ESR-код
Параметры	T_AUTO	TIME	Время активности выхода Q
	T_DELAY	TIME	Время блокировки управления REQ
Используемые модули	TP_1D		

Функциональный блок **FLOW_CONTROL** используется для управления клапаном. Если вход **ENQ** имеет значение **TRUE**, то блок находится в работе и разрешает управление выходом **Q** через входы **IN** и **REQ**. Вход **IN** позволяет вручную управлять выходом **Q**. По переднему фронту на входе **REQ** выход **Q** принимает значение **TRUE** на время **T_AUTO**, после чего выход сбрасывается **FALSE** и управление через вход **REQ** блокируется на время **T_DELAY**. В течение времени блокировки можно управлять выходом через вход **IN**. В течение времени **T_AUTO** выход может быть сброшен в состояние **FALSE** с помощью переднего фронта на входе **RST**.

Выход **Status** определяет состояние блока и совместим с ESR-модулями:

Значение выхода Status	Описание
100	Блок готов к работе
101	Выход включен с помощью входа IN
102	Выход включен с помощью входа REQ
103	Выход отключен с помощью входа RST

27.5 FT_PROFILE

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	K	REAL	Коэффициент масштабирования
	O	REAL	Смещение
	M	REAL	Коэффициент масштаба времени
	E	BOOL	Сигнал запуска блока
Выходы	Y	REAL	Выход генератора
	RUN	BOOL	Флаг «блок в работе»
	ET	TIME	Время, прошедшее с начала работы
Параметры	value_0, value_1, value_2, value_3, value_10, value_11, value_12, value_13	REAL	Значение уставки
	time_1, time_2, time_3, time_10, time_11, time_12, time_13	TIME	Время достижения уставки
Используемые модули	T_PLC_MS, MULTIME		

Функциональный блок **FT_PROFILE** используется для изменения величины по заданной зависимости «уставка – время достижения» (например, для температурного профиля печи). Профиль разбивается на 7 участков (с номерами 1, 2, 3, 10, 11, 12, 13), продолжительности которых определяют соответствующие параметры **time_x**. Следует заметить, что все интервалы **time_x** отсчитываются относительно момента запуска блока, поэтому каждое последующее значение времени должно быть больше предыдущего. Значения крайних точек участков определяются параметрами **value_x**. Переходы между участками происходят по линейной зависимости. Входы **K** и **O** определяют коэффициент масштабирования и смещения выхода **Y**; вход **M** определяет коэффициент масштаба времени. Значение выхода **Y** на участке **b** определяется следующей формулой:

$$Y(b) = K \cdot (value_b - value_a) \cdot \frac{\text{время с начала отработки участка } b}{M \cdot (\text{time}_b - \text{время с начала отработки участка } a)} + O, \text{ где } a=b-1$$

Блок запускается в работу импульсом по переднему фронту на входе **E**; при этом участки 11-13 будут обработаны только в том случае, если к началу времени их обработки **E** уже имеет значение **FALSE**. Выход **RUN** имеет значение **TRUE**, пока блок находится в работе; выход **ET** отображает время, прошедшее с начала работы блока. После завершения обработки всех участков выход **RUN** принимает значение **FALSE**, а на выходе **ET** сохраняется общее время работы блока.

Каждый импульс по переднему фронту на входе **IN** запускает работу блока заново.

27.6 INC_DEC

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	CHa	BOOL	Канал А
	CHb	BOOL	Канал В
	RST	BOOL	Сигнал сброса
Выходы	dir	BOOL	Направление вращения
	cnt	INT	Счетчик фронтов

Функциональный блок **INT_DEC** представляет собой модуль инкрементального квадратурного энкодера. На входы **CHa** и **CHb** должны быть поданы сигналы каналов **A** и **B** энкодера. Выход **dir** определяет направление вращения (**TRUE** – прямое, **FALSE** – обратное). Выход **cnt** содержит сумму подсчитанных фронтов (как передних, так и задних) по обоим каналам. По переднему фронту на входе **RST** счетчик обнуляется.

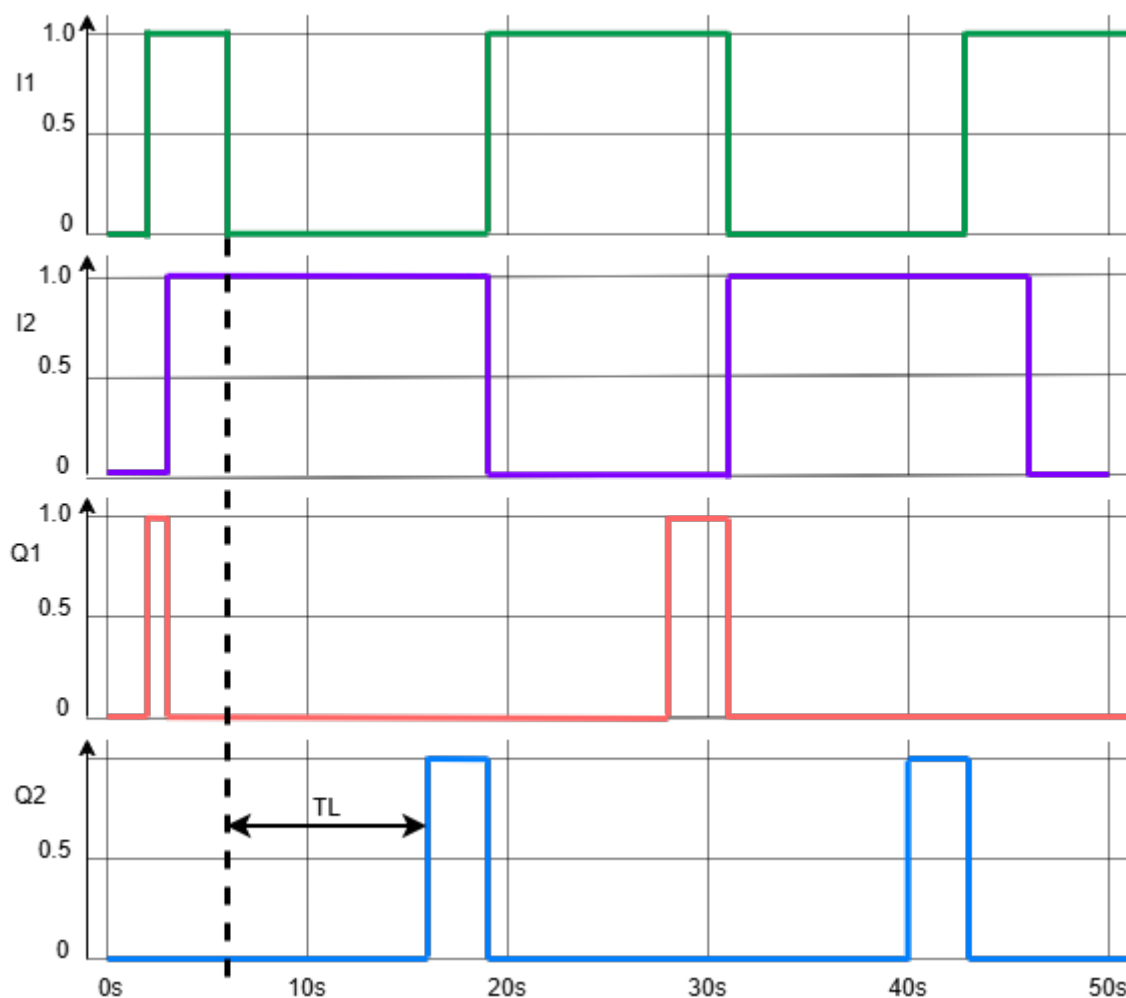
27.7 INTERLOCK

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	I1	BOOL	Вход 1
	I2	BOOL	Вход 2
	TL	TIME	Время блокировки
Выходы	Q1	BOOL	Выход 1
	Q2	BOOL	Выход 2

Функциональный блок **INTERLOCK** содержит два входа (**I1** и **I2**), которые управляют соответствующими выходами (**Q1** и **Q2**). В каждый момент времени только один из выходов может быть активен. Ниже приведена таблица состояний входов и выходов.

I1	I2	Q1	Q2
FALSE	FALSE	FALSE	FALSE
FALSE	TRUE	FALSE	TRUE
TRUE	FALSE	TRUE	FALSE
TRUE	TRUE	FALSE	FALSE

Вход **TL** определяет время блокировки между отключением одного выхода и включением другого (т.е. после того, как выход **Q1** примет значение **FALSE**, выход **Q2** не сможет принять значение **TRUE** до истечения времени **TL**).

Рисунок 27.1 – Трассировка ФБ INTERLOCK ($TL=T\#10s$)

27.8 INTERLOCK_4

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	I0	BOOL	Вход 0
	I1	BOOL	Вход 1
	I2	BOOL	Вход 2
	I3	BOOL	Вход 3
	E	BOOL	Сигнал управления блоком
	MODE	INT	Режим работы блока
Выходы	OUT	BYTE	Выход блока
	TP	BOOL	Флаг «значение выхода изменилось»

Функциональный блок **INTERLOCK_4** содержит четыре входа типа BOOL (**I0... I3**), которые управляют выходом типа BYTE (**OUT**). Если вход **E** имеет значение **FALSE**, то выход **OUT** имеет значение **0**. Если вход **E** имеет значение **TRUE**, то значение выхода **OUT** определяется значениями входов **I0...I3** и режимом работы **MODE**:

Mode	Описание
0	Все входы транслируются на выход (I0 – младший бит)
1	На выход транслируется только старший активный бит

Mode	Описание
2	На выход транслируется активный бит, изменившийся последним
3	На выход транслируется активный бит, причем до его отключения остальные входы не обрабатываются

Выход **TP** принимает значение **TRUE** на 1 цикл ПЛК при каждом изменении выхода **OUT**.

27.9 MANUAL

Тип модуля: функция	Переменная	Тип	Описание
Входы	IN	BOOL	Сигнал дистанционного управления
	ON	BOOL	Сигнал местного управления «Включить»
	OFF	BOOL	Сигнал местного управления «Выключить»
Выходы	MANUAL	BOOL	Выход функции

Функция **MANUAL** представляет собой модуль местного/дистанционного управления. Выход функции определяется значением входа дистанционного управления **IN** и входами локального управления **ON/OFF** («включить» / «выключить»), причем сигналы управления имеют следующие приоритеты:

IN	ON	OFF	MANUAL
FALSE	FALSE	FALSE	FALSE
TRUE	FALSE	FALSE	TRUE
не имеет значения	не имеет значения	TRUE	FALSE
не имеет значения	TRUE	FALSE	TRUE

Таким образом, местное управление имеет приоритет по сравнению с дистанционным, а сигнал местного управления «Выключить» - над сигналом «Включить».

27.10 MANUAL_1

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	BOOL	Сигнал дистанционного управления
	MAN	BOOL	Режим управления
	M_I	BOOL	Сигнал местного управления
	SET	BOOL	Сигнал принудительного включения
	RST	BOOL	Сигнал принудительного отключения
Выходы	Q	BOOL	Выход блока
	STATUS	BYTE	ESR-код

Функциональный блок **MANUAL_1** представляет собой модуль местного/дистанционного управления. Если вход **MAN** имеет значение **FALSE**, то значение выхода **Q** определяется значением входа **IN**. Если вход **MAN** имеет значение **TRUE**, то значение выхода **Q** определяется значением входа **M_I** и, кроме того, может быть принудительно изменено на **TRUE/FALSE** по переднему фронту входа **SET/RST**. После импульса на входе **SET** или **RST** значение входа **M_I** перестает влиять на значение выхода; чтобы вернуться к управлению через вход **M_I** необходимо переключить вход **MAN** в состояние **FALSE**, после чего опять вернуть его в **TRUE**.

Выход **Status** определяет состояние блока и совместим с [ESR-модулями](#):

Значение выхода Status	Описание
100	Блок в режиме дистанционного управления (MAN=FALSE).
101	Выход принудительно взведен в TRUE (через SET).
102	Выход принудительно сброшен в FALSE (через RST).
103	Блок в режиме местного управления (MAN=TRUE).

27.11 MANUAL_2

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	BOOL	Сигнал дистанционного управления
	ENA	BOOL	Сигнал управления блоком
	ON	BOOL	Сигнал принудительного включения
	OFF	BOOL	Сигнал принудительного отключения
	MAN	BOOL	Сигнал местного управления
Выходы	Q	BOOL	Выход блока
	STATUS	BYTE	ESR-код

Функциональный блок **MANUAL_2** представляет собой модуль местного/дистанционного управления. Если вход **ENA** имеет значение **FALSE**, то блок отключен и выход **Q** имеет значение **FALSE**. Если вход **ENA** имеет значение **TRUE**, то блок находится в работе и значение выхода **Q** определяется состояниями входов **IN**, **ON**, **OFF** и **MAN** («-» – состояние входа не имеет значения, «х» – совпадающие состояния входов и выходов).

Выход **Status** определяет состояние блока и совместим с [ESR-модулями](#).

ENA	IN	ON	OFF	MAN	Q	STA-TUS	Описание
FALSE	-	-	-	-	FALSE	104	Блок отключен
TRUE	x	FALSE	FALSE	-	x	100	Блок в режиме дист. управления
TRUE	-	TRUE	FALSE	-	TRUE	101	Выход принудительно взведен в TRUE
TRUE	-	FALSE	TRUE	-	FALSE	102	Выход принудительно сброшен в FALSE
TRUE	-	TRUE	TRUE	x	x	103	Блок в режиме местного управления

27.12 MANUAL_4

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	I0...I3	BOOL	Сигналы дистанционного управления
	MAN	BOOL	Режим управления
	STP	BOOL	Сигнал пошагового переключения
	M0...M3	BOOL	Сигналы местного управления
Выходы	Q0...Q3	BOOL	Выходы блока
	STATUS	BYTE	ESR-код
Используемые модули	INC		

Функциональный блок **MANUAL_4** представляет собой 4-х каналный модуль местного/дистанционного управления. Если вход **MAN** имеет значение **FALSE**, то значения выходов **Q0...Q3** определяются значениями входов **I0... I3**. Если вход **MAN** имеет значение **TRUE**, то значения выходов **Q0... Q3** определяются значениями входов **M0...M3**. Если вход **MAN** имеет значение **TRUE**, и на входе **STP** детектируется импульс по переднему фронту, то блок переходит в режим пошагового переключения выходов – при этом выход **Q0** принимает значение **TRUE**, а все остальные выходы – **FALSE**. Каждый следующий импульс на входе **STP** будет переключать активный выход блока:

Действие	Q0	Q1	Q2	Q3
Импульс по переднему фронту на входе STP	TRUE	FALSE	FALSE	FALSE
Импульс по переднему фронту на входе STP	FALSE	TRUE	FALSE	FALSE
Импульс по переднему фронту на входе STP	FALSE	FALSE	TRUE	FALSE
Импульс по переднему фронту на входе STP	FALSE	FALSE	FALSE	TRUE
Импульс по переднему фронту на входе STP	TRUE	FALSE	FALSE	FALSE
...	...	...	...	...

Выход **Status** определяет состояние блока и совместим с [ESR-модулями](#):

STATUS	Описание
100	Блок в режиме дистанционного управления (MAN=FALSE)
101	Блок в режиме местного управления (MAN=TRUE)
110/111/112/113	Блок в режиме пошагового переключения, активен выход Q0/Q1/Q2/Q3

27.13 PARSET

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	A0	BOOL	Сигнал переключения выходов 0
	A1	BOOL	Сигнал переключения выходов 1
Выходы	P1...P4	REAL	Выходы блока
Параметры	X01...X04, X11...X14, X21...X24, X31...X34	REAL	Возможные значения выходов
	TC	TIME	Время перехода к новому значению
Используемые модули	T_PLC_MS		

Функциональный блок **PARSET** передает на выходы **P1... P4** одну из 4-х заранее подготовленных групп значения (X01...X04/X11...X14/X21...X24/X31...X34) в зависимости от состояний входов **A0** и **A1**:

A0	A1	P1	P2	P3	P4
FALSE	FALSE	X01	X02	X03	X03
FALSE	TRUE	X11	X12	X13	X14
TRUE	FALSE	X21	X22	X23	X24
TRUE	TRUE	X31	X32	X33	X34

Если параметр **TC** имеет значение **T#0ms**, то при переключении выходов значения меняются мгновенно; если значение **TC** отлично от нуля, то изменение значения выходов происходит плавно, по линейной зависимости за время **TC**.

27.14 PARSET2

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	X	REAL	Входное значение
Выходы	P1...P4	REAL	Выходы блока
Параметры	X01...X04, X11...X14, X21...X24, X31...X34	REAL	Возможные значения выходов
	L1...L3	REAL	Граничные значения для X
	TC	TIME	Время перехода к новому значению
Используемые модули	PARSET		

Функциональный блок **PARSET2** передает на выходы **P1...P4** одну из 4-х заранее подготовленных групп значения (X01...X04/X11...X14/X21...X24/X31...X34) в зависимости от значения входа **X** и заданных для него граничных значений **L1...L3**:

X	P1	P2	P3	P4
$X < L1$	X01	X02	X03	X03
$L1 < X < L2$	X11	X12	X13	X14
$L2 < X < L3$	X21	X22	X23	X24
$L3 \leq X$	X31	X32	X33	X34

Если параметр **TC** имеет значение **T#0ms**, то при переключении выходов значения меняются мгновенно; если значение **TC** отлично от нуля, то изменение значения выходов происходит плавно, по линейной зависимости за время **TC**.

27.15 SIGNAL

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN	BOOL	Сигнал управления блоком
	SIG	BYTE	Последовательность бит
	TS	TIME	Период переключения выхода
Выходы	Q	BOOL	Выход генерации последовательности бит
Используемые модули	T_PLC_MS		

Функциональный блок **SIGNAL** используется для циклической генерации заданной последовательности бит. Пока вход **IN** имеет значение **TRUE**, на выходе **Q** генерируется последовательность бит, определенная значением входа **SIG**, при этом изменение выхода происходит с периодичностью **TS**. Генерация сигнала начинается с произвольного бита последовательности.

27.16 SIGNAL_4

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	IN1...IN4	BOOL	Сигнал выбора последовательности
	TS	TIME	Период переключения выхода
Выходы	Q	BOOL	Выход генерации последовательности бит
Параметры	S1...S4	BYTE	Заданные последовательности бит
Используемые модули	SIGNAL		

Функциональный блок **SIGNAL_4** используется для циклической генерации одной из 4-х заданных последовательностей бит. Пока один из входов **IN1...IN4** имеет значение **TRUE**, на выходе **Q** генерируется соответствующая последовательность бит (**S1... S4**), при этом изменение выхода происходит с периодичностью **TS**. Генерация сигнала начинается с произвольного бита последовательности. Если одновременно активны несколько входов, то на выходе генерируется последовательность с наименьшим номером.

См. также описание ФБ [SIGNAL](#).

27.17 SRAMP

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	X	REAL	Входной сигнал
	A_UP	REAL	Ускорение нарастания выходного сигнала
	A_DN	REAL	Ускорение спада выходного сигнала
	VU_MAX	REAL	Макс. скорость нарастания выходного сигнала
	VD_MAX	REAL	Макс. скорость спада выходного сигнала
	LIMIT_HIGH	REAL	Верхний предел выходного сигнала
	LIMIT_LOW	REAL	Нижний предел выходного сигнала
	RST	BOOL	Сигнал сброса блока
Выходы	Y	REAL	Выходной сигнал
	V	REAL	Скорость изменения выходного сигнала
Используемые модули	TC_S		

Функциональный блок **SRAMP** используется для плавного изменения выходного сигнала при изменении входного. Значения входов блока ограничивают скорость нарастания и спада выходного сигнала **Y** (**VU_MAX**, **VD_MAX**), ускорение его нарастания и спада (**A_UP**, **A_DN**) и максимальное/минимальное допустимое значение (**LIMIT_HIGH**, **LIMIT_LOW**). Значения **VU_MAX** и **A_UP** должны быть положительными, а **VD_MAX** и **A_DN** – отрицательными. На выход **V** подается текущая скорость изменения выходного сигнала. По переднему фронту на входе **RST** происходит обнуление выходов блока.

27.18 TUNE

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	SET	BOOL	Сигнал установки значения SET_val
	SU	BOOL	Сигнал увеличения значения
	SD	BOOL	Сигнал уменьшения значения
	RST	BOOL	Сигнал установки значения RST_val
Выходы	Y	REAL	Изменяемое значение
Параметры	SS	REAL	Шаг изменения значения по нажатию кнопок SU/SD
	Limit_H	REAL	Верхний предел значения выхода
	Limit_L	REAL	Нижний предел значения выхода

Тип модуля: ФБ	Переменная	Тип	Описание
	SET_val	REAL	Значение для сигнала SET
	RST_val	REAL	Значение для сигнала RST
	T1	TIME	Время зажатия кнопки для S2
	T2	TIME	Время зажатия кнопки для S2
	S1	REAL	Шаг изменения значения при зажатии кнопок SU/SD на время T1, ед/с
	S2	REAL	Шаг изменения значения при зажатии кнопок SU/SD на время T2, ед/с
Используемые модули	T_PLC_MS		

Функциональный блок **TUNE** представляет собой модуль управления значением. По переднему фронту на входе **SU/SD** значение выхода **Y** увеличивается/уменьшается на величину **SS**. Таким образом, входы **SU** и **SD** могут быть привязаны к кнопкам «больше»/ «меньше». Если вход **SU/SD** сохраняет значение **TRUE** в течение времени **T1/T2** (т.е. кнопка зажата), то значение **Y** начинается каждую секунду увеличиваться/уменьшаться на величину **S1/S2**. По переднему фронту на входе **SET/ RST** выход **Y** принимает значение **SET_val/RST_val**. Значение **Y** ограничено диапазоном [**Limit_L...Limit_H**].

27.19 TUNE2

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	SET	BOOL	Сигнал установки значения SET_val
	SU	BOOL	Сигнал увеличения значения (SS)
	SD	BOOL	Сигнал уменьшения значения (SS)
	FU	BOOL	Сигнал увеличения значения (FS)
	FD	BOOL	Сигнал уменьшения значения (FS)
	RST	BOOL	Сигнал установки значения RST_val
Выходы	Y	REAL	Изменяемое значение
Параметры	SS	REAL	Шаг изменения значения по нажатию кнопок SU/SD
	FS	REAL	Шаг изменения значения по нажатию кнопок FU/FD
	Limit_H	REAL	Верхний предел значения выхода
	Limit_L	REAL	Нижний предел значения выхода
	SET_val	REAL	Значение для сигнала SET
	RST_val	REAL	Значение для сигнала RST
	TR	TIME	Время зажатия кнопки для S2
	S1	REAL	Шаг изменения значения при зажатии кнопок SU/SD на время TR, ед/с

Тип модуля: ФБ	Переменная	Тип	Описание
	S2	REAL	Шаг изменения значения при зажатии кнопок FU/FD на время TR, ед/с
Используемые модули	T_PLC_MS		

Функциональный блок **TUNE2** представляет собой модуль управления значением. По переднему фронту на входе **SU/SD** значение выхода **Y** увеличивается/уменьшается на величину **SS**. По переднему фронту на входе **FU/FD** значение выхода **Y** увеличивается/уменьшается на величину **FS**. Таким образом, входы **SU** и **SD** могут быть привязаны к кнопкам «больше»/ «меньше», а входы **FD/ FS** – к кнопкам «значительно больше»/ «значительно меньше». Если вход **SU/SD** сохраняет значение **TRUE** в течение времени **TR** (т. е. кнопка зажата), то значение **Y** начинается каждую секунду увеличиваться/уменьшаться на величину **S1**. Если вход **FU/FD** сохраняет значение **TRUE** в течение времени **TR**, то значение **Y** начинается каждую секунду увеличиваться/уменьшаться на величину **S2**. По переднему фронту на входе **SET/RST** выход **Y** принимает значение **SET_val/RST_val**. Значение **Y** ограничено диапазоном [**Limit_L...Limit_H**].

28 Работа с буфером

Функции, описанные в данной главе, используются для операций с буфером. Буфер представляет собой массив байт. Входными переменными для каждой функции являются указатель на массив (**PT**) и размер массива (**SIZE**). В большинстве случаев представляется удобным использовать операторы **ADR** и **SIZEOF**, которые возвращают адрес и размер массива в байтах соответственно.

Обработка массива, полученного по указателю, производится путем прямых манипуляций с памятью ПЛК. Этот тип обработки является крайне эффективным, так как не требует копирования содержимого массива.

28.1 _BUFFER_CLEAR

Тип модуля: функция	Переменная	Тип	Описание
Входы	PT	POINTER TO BYTE	Указатель на буфер
	SIZE	UINT	Размер буфера
Выходы	_BUFFER_CLEAR	BOOL	Флаг окончания обработки буфера

Функция **_BUFFER_CLEAR** присваивает каждому из элементов буфера значение **0**. Результаты записываются в тот же буфер по указателю. После окончания обработки выход функции принимает значение **TRUE**.

Пример: буфер [2, 12, 4, 8, 6] после обработки функцией примет вид [0, 0, 0, 0, 0].

Пример

```

VAR
  (* Буфер в виде массива байт *)
  Buffer : ARRAY[1..5] OF BYTE := [2, 12, 4, 8, 6];

  (* Флаг окончания обработки *)
  Done : BOOL;

  (* Счетчик для отображения *)
  i : INT;
END_VAR

(* Вызов функции _BUFFER_CLEAR *)
Done := OSCAT_BASIC._BUFFER_CLEAR(
  PT := ADR(Buffer),
  SIZE := SIZEOF(Buffer)
);

(*
Результат:
- Buffer = [0, 0, 0, 0, 0]
- Done = TRUE

Пояснение:
- ADR(Buffer) возвращает адрес массива в памяти
- SIZEOF(Buffer) возвращает размер массива в байтах (5 байт)
- Функция присваивает каждому элементу буфера значение 0
*)
END_PROGRAM

```

28.2 _BUFFER_INIT

Тип модуля: функция	Переменная	Тип	Описание
Входы	PT	POINTER TO BYTE	Указатель на буфер
	SIZE	UINT	Размер буфера
	INIT	BYTE	Присваиваемое значение
Выходы	_BUFFER_INIT	BOOL	Флаг окончания обработки буфера

Функция **_BUFFER_INIT** присваивает каждому из элементов буфера значение **INIT**. Результаты записываются в тот же буфер по указателю. После окончания обработки выход функции принимает значение **TRUE**.

Пример: буфер [2, 12, 4, 8, 6] после обработки функцией примет вид [3, 3, 3, 3, 3].

Пример

```

VAR
  (* Буфер в виде массива байт *)
  Buffer : ARRAY[1..5] OF BYTE := [2, 12, 4, 8, 6];

  (* Присваиваемое значение *)
  InitValue : BYTE := 3;

  (* Флаг окончания обработки *)
  Done : BOOL;
END_VAR

(* Вызов функции _BUFFER_INIT *)
Done := OSCAT_BASIC._BUFFER_INIT(
  PT := ADR(Buffer),
  SIZE := SIZEOF(Buffer),
  INIT := InitValue
);

(*
  Результат:
  - Buffer = [3, 3, 3, 3, 3]
  - Done = TRUE

  Пояснение:
  - ADR(Buffer) возвращает адрес массива в памяти
  - SIZEOF(Buffer) возвращает размер массива в байтах (5 байт)
  - Функция присваивает каждому элементу буфера значение INIT (3)
*)
END_PROGRAM

```

28.3 _BUFFER_INSERT

Тип модуля: функция	Переменная	Тип	Описание
Входы	STR	STRING	Записываемая строка
	POS	INT	Позиция для записи
	PT	POINTER TO ARRAY [0..32767] OF BYTE	Указатель на буфер
	SIZE	UINT	Размер буфера
Выходы	_BUFFER_INSERT	INT	Позиция в буфере после записи
Используемые модули	_STRING_TO_BUFFER		

Функция **_BUFFER_INSERT** записывает строку **STR** (в виде ASCII-кодов) в буфер с позиции **POS** (где **0** – позиция первого элемента буфера), смещая оставшуюся часть данных в буфере на длину строки. После окончания операции на выход функции транслируется позиция буфера после вставленной строки.

Пример

```

VAR
  (* Буфер в виде массива байт *)
  Buffer : ARRAY[0..9] OF BYTE := [2, 12, 4, 8, 6, 0, 0, 0, 0, 0];

  (* Записываемая строка *)
  TextToInsert : STRING := 'test';

  (* Позиция для вставки (0 — начало буфера) *)
  InsertPosition : INT := 2;

  (* Результат — позиция после вставки *)
  NewPosition : INT;
END_VAR

(* Вызов функции _BUFFER_INSERT *)
NewPosition := OSCAT_BASIC._BUFFER_INSERT(
  STR := TextToInsert,
  POS := InsertPosition,
  PT := ADR(Buffer),
  SIZE := SIZEOF(Buffer)
);

(*
  Результат:
  - Buffer = [2, 12, 116, 101, 115, 116, 4, 8, 6, 0]
  - NewPosition = 6

  Пояснение:
  - Исходный буфер: [2, 12, 4, 8, 6, 0, 0, 0, 0, 0]
  - Вставляем 'test' с позиции 2 (третья ячейка)
  - ASCII-коды: 't'=116, 'e'=101, 's'=115, 't'=116
  - Данные после позиции 2 (4, 8, 6, 0, 0, 0, 0) сдвигаются вправо на 4 байта
  - Итоговый буфер: [2, 12, 116, 101, 115, 116, 4, 8, 6, 0]
  - NewPosition = 2 + 4 = 6 (позиция сразу после вставленной строки)
*)
END_PROGRAM

```

См. также функцию [__STRING_TO_BUFFER](#), которая перезаписывает данные буфера, а не смещает их.

28.4 _BUFFER_UPPERCASE

Тип модуля: функция	Переменная	Тип	Описание
Входы	PT	POINTER TO BYTE	Указатель на буфер
	SIZE	INT	Размер буфера
Выходы	_BUFFER_UPPERCASE	BOOL	Флаг окончания обработки буфера
Используемые модули	TO_UPPER		

Функция **_BUFFER_UPPERCASE** интерпретирует буфер как набор ASCII-кодов, и конвертирует ASCII-коды букв в коды тех же букв в верхнем регистре.

Пример

```

VAR      (* Буфер в виде массива байт *)
  Buffer : ARRAY[0..3] OF BYTE := [242, 229, 241, 242];

  (* Флаг окончания обработки *)
  Done : BOOL;
END_VAR

(* Вызов функции _BUFFER_UPPERCASE *)
Done := OSCAT_BASIC._BUFFER_UPPERCASE(
  PT := ADR(Buffer),
  SIZE := SIZEOF(Buffer)
);

(*
  Результат:
  - Buffer = [210, 197, 209, 210]
  - Done = TRUE

  Пояснение:
  - Исходные ASCII-коды: 242, 229, 241, 242 (буквы "т", "е", "с", "т")
  - После конвертации: 210, 197, 209, 210 (буквы "Т", "Е", "С", "Т")
  - Все строчные буквы заменены на заглавные
*)
END_PROGRAM

```

См. также функцию `_STRING_TO_BUFFER`, которая перезаписывает данные буфера, а не смещает их.

28.5 _STRING_TO_BUFFER

Тип модуля: функция	Переменная	Тип	Описание
Входы	STR	STRING	Записываемая строка
	POS	INT	Позиция для записи
	PT	POINTER TO ARRAY [0..32767] OF BYTE	Указатель на буфер
	SIZE	UINT	Размер буфера
Выходы	_STRING_TO_BUFFER	INT	Позиция в буфере после записи

Функция `_STRING_TO_BUFFER` записывает строку **STR** (в виде ASCII-кодов) в буфер с позиции **POS** (где **0** – позиция первого элемента буфера), перезаписывая хранящиеся в буфере данные. После окончания операции на выход функции транслируется позиция буфера после вставленной строки.

Пример

```

VAR
  (* Буфер в виде массива байт *)
  Buffer : ARRAY[0..9] OF BYTE := [2, 12, 4, 8, 6, 0, 0, 0, 0, 0];

  (* Записываемая строка *)
  TextToWrite : STRING := 'test';

  (* Позиция для записи (0 — начало буфера) *)
  WritePosition : INT := 2;

  (* Результат — позиция после записи *)
  NewPosition : INT;
END_VAR

(* Вызов функции _STRING_TO_BUFFER *)
NewPosition := OSCAT_BASIC._STRING_TO_BUFFER(
  STR := TextToWrite,
  POS := WritePosition,
  PT := ADR(Buffer),
  SIZE := SIZEOF(Buffer)
);

(*
  Результат:
  - Buffer = [2, 12, 116, 101, 115, 116, 0, 0, 0, 0]
  - NewPosition = 6

  Пояснение:
  - Исходный буфер: [2, 12, 4, 8, 6, 0, 0, 0, 0, 0]
  - Записываем 'test' с позиции 2 (третья ячейка)
  - ASCII-коды: 't'=116, 'e'=101, 's'=115, 't'=116
  - Данные в позициях 2, 3, 4, 5 перезаписываются
  - Данные после позиции 5 (0, 0, 0, 0) остаются без изменений
  - Итоговый буфер: [2, 12, 116, 101, 115, 116, 0, 0, 0, 0]
  - NewPosition = 2 + 4 = 6 (позиция сразу после записанной строки)
*)
END_PROGRAM

```

См. также функцию [_BUFFER_INSERT](#), которая перезаписывает данные буфера, а не смещает их.

28.6 BUFFER_COMP

Тип модуля: функция	Переменная	Тип	Описание
Входы	PT1	POINTER TO ARRAY [0..32767] OF BYTE	Указатель на буфер 1
	SIZE1	INT	Размер буфера 1
	PT2	POINTER TO ARRAY [0..32767] OF BYTE	Указатель на буфер 2
	SIZE2	INT	Размер буфера 2
	START	INT	Начальная позиция для поиска буфера 2 в буфере 1
Выходы	BUFFER_COMP	INT	Начальная позиция буфера 2 в буфере 1

Функция **BUFFER_COMP** производит поиск содержимого буфера, расположенного по адресу **PT2**, в буфере, расположенном по адресу **PT1**, начиная с элемента номер **START** (нулевому элементу буфера **1** соответствует **START=0**). Функция возвращает номер элемента буфера **1**, начиная с которого в нем было найдено первое вхождение буфера **2**. Если буфер не был найден, то функция возвращает **-1**.

Пример


```

VAR
  (* Буфер 1 – исходный массив *)
  Buffer1 : ARRAY[0..9] OF BYTE := [2, 12, 4, 8, 6, 2, 12, 4, 8, 6];

  (* Буфер 2 – искомый шаблон *)
  Buffer2 : ARRAY[0..1] OF BYTE := [12, 4];

  (* Начальная позиция для поиска *)
  StartPos : INT := 0;

  (* Результат – позиция первого вхождения *)
  FoundPos : INT;
END_VAR

(* Вызов функции BUFFER_COMP *)
FoundPos := OSCAT_BASIC.BUFFER_COMP(
  PT1 := ADR(Buffer1),
  SIZE1 := SIZEOF(Buffer1),
  PT2 := ADR(Buffer2),
  SIZE2 := SIZEOF(Buffer2),
  START := StartPos
);

(*
  Результат: FoundPos = 1

  Пояснение:
  - Buffer1 = [2, 12, 4, 8, 6, 2, 12, 4, 8, 6]
  - Buffer2 = [12, 4]
  - Поиск начинается с позиции 0
  - Первое вхождение [12, 4] – позиции 1 и 2
  - Функция возвращает номер начального элемента: 1

  Второе вхождение: позиции 6 и 7 (не возвращается, так как найдено первое)
*)
END_PROGRAM

```

28.7 BUFFER_SEARCH

Тип модуля: функция	Переменная	Тип	Описание
Входы	PT	POINTER TO ARRAY [0..32767] OF BYTE	Указатель на буфер
	SIZE	INT	Размер буфера
	STR	STRING	Искомая строка
	POS	INT	Начальная позиция для поиска
	IGN	BOOL	Регистрозависимость поиска
Выходы	BUFFER_SEARCH	BOOL	Начальная позиция найденной строки
Используемые модули	TO_UPPER		

Функция **BUFFER_SEARCH** интерпретирует буфер как набор ASCII-кодов, и производит в нем поиск строки **STR** начиная с позиции **POS** (где **0** – позиция первого элемента буфера). Функция возвращает позицию первого вхождения строки в буфер. Если строка не найдена, то функция возвращает **-1**. Если вход **IGN** имеет значение **TRUE**, то поиск будет регистронезависимым, при этом искомая строка **STR** должна состоять только из заглавных букв (т.е. при **STR='AB'** в буфере также будет производиться поиск строк 'AB', 'ab', 'Ab' и 'aB'). Если вход **IGN** имеет значение **FALSE**, то поиск будет регистрозависимым.

Пример

```

VAR
  (* Буфер с двумя строками: "тест" и "ТЕСТ" *)
  Buffer : ARRAY[0..9] OF BYTE := [242, 229, 241, 242, 0, 210, 197, 209, 210, 0];
  (* Искомая строка (только заглавные буквы) *)
  SearchStr : STRING := 'TECT';
  (* Начальная позиция поиска *)
  StartPos : INT := 0;
  (* Результаты *)
  PosIgnoreCase : INT;      (* IGN = TRUE — регистронезависимый *)
  PosCaseSensitive : INT;   (* IGN = FALSE — регистрозависимый *)
END_VAR

(* Регистронезависимый поиск *)
PosIgnoreCase := OSCAT_BASIC.BUFFER_SEARCH(
  PT := ADR(Buffer),
  SIZE := SIZEOF(Buffer),
  STR := SearchStr,
  POS := StartPos,
  IGN := TRUE
);
(* Регистрозависимый поиск *)
PosCaseSensitive := OSCAT_BASIC.BUFFER_SEARCH(
  PT := ADR(Buffer),
  SIZE := SIZEOF(Buffer),
  STR := SearchStr,
  POS := StartPos,
  IGN := FALSE
);

(*
  Результаты:
  - PosIgnoreCase = 0
  - PosCaseSensitive = 5

  Пояснение:
  Буфер: [242, 229, 241, 242, 0, 210, 197, 209, 210, 0]
  - Позиции 0-3: 242/229/241/242 = "тест" (строчные)
  - Позиция 4: 0 = разделитель
  - Позиции 5-8: 210/197/209/210 = "ТЕСТ" (заглавные)
  - Позиция 9: 0 = разделитель

  При IGN = TRUE:
  - Поиск не учитывает регистр
  - Находит "тест" с позиции 0 → возвращает 0

  При IGN = FALSE:
  - Поиск учитывает регистр
  - Строчные "тест" не подходят
  - Находит "ТЕСТ" с позиции 5 → возвращает 5
*)
END_PROGRAM

```

28.8 BUFFER_TO_STRING

Тип модуля: функция	Переменная	Тип	Описание
Входы	PT	POINTER TO ARRAY [0..32767] OF BYTE	Указатель на буфер
	SIZE	UINT	Размер буфера
	START	START	Начальная позиция
	STOP	STOP	Конечная позиция
Выходы	BUFFER_TO_STRING	STRING	Извлеченная строка

Функция **BUFFER_TO_STRING** интерпретирует буфер как набор ASCII-кодов, и возвращает текст, размещенный в нем с позиции **START** по позицию **STOP** в виде строки (где **0** – позиция первого элемента буфера).

Пример

```

VAR
  (* Буфер с ASCII-кодами *)
  Buffer : ARRAY[0..9] OF BYTE := [2, 12, 116, 101, 115, 116, 0, 0, 0, 0];

  (* Позиции для извлечения *)
  StartPos : INT := 2;          (* Начало строки *)
  StopPos : INT := 5;          (* Конец строки *)

  (* Результат – извлечённая строка *)
  ExtractedString : STRING;
END_VAR

(* Вызов функции BUFFER_TO_STRING *)
ExtractedString := OSCAT_BASIC.BUFFER_TO_STRING(
  PT := ADR(Buffer),
  SIZE := SIZEOF(Buffer),
  START := StartPos,
  STOP := StopPos
);

(*
  Результат: ExtractedString = 'test'

  Пояснение:
  - Buffer = [2, 12, 116, 101, 115, 116, 0, 0, 0, 0]
  - Извлекаем с позиции 2 по позицию 5
  - ASCII-коды 116/101/115/116 = 't'/'e'/'s'/'t'
  - Результат: 'test'
*)
END_PROGRAM

```

29 Работа со списками

Функции, описанные в данной главе, используются для операций над списками. Список представляет собой строковую переменную (типа **STRING**), содержащую записи, каждая из которых начинается со специального символа (этот символ определяется пользователем). Максимальный размер списка определяется **глобальной константой** **LIST_LENGTH**.

Функции обрабатывают списки через **VAR_IN_OUT** переменные (входы-выходы) – т.е. на вход функции подается исходный список, и на этот же вход возвращается результат его обработки. Такой подход позволяет обрабатывать списки путем прямых манипуляций с памятью ПЛК и является крайне эффективным, так как не требует копирования содержимого списков.

В рамках рассматриваемых примерах в качестве разделителя используется символ **'&'** (ASCII-код – 16#26).

Примеры списков:

- **'&12&345'** – список с двумя записями ('12' и '345');
- **''** – список без записей;
- **'&&&'** – список с тремя пустыми записями;
- **'&12&345&&ABC'** – список с четырьмя записями ('12', '345', пустая запись, 'ABC').

Номера записей отсчитываются с единицы (т.е. первая запись списка имеет номер 1).

29.1 LIST_ADD

Тип модуля: функция	Переменная	Тип	Описание
Входы	SEP	BYTE	ASCII-код символа-разделителя
	INS	STRING	Добавляемая запись
Входы-выходы	LIST	STRING(LIST_LENGTH)	Список
Выходы	LIST_ADD	BOOL	Флаг завершения операции
Используемые модули	CHR_TO_STRING		

Функция **LIST_ADD** добавляет в конец списка **LIST** с разделителем **SEP** запись **INS**. После выполнения операции выход функции принимает значение **TRUE**.

Пример

```

VAR
  (* Список с разделителем '&' (ASCII 16#26) *)
  List : STRING(OSCAT_BASIC.LIST_LENGTH) := '&12&345';
  (* Разделитель *)
  Separator : BYTE := 16#26;          (* '&' *)
  (* Добавляемая запись *)
  NewRecord : STRING := 'ABC';
  (* Флаг завершения операции *)
  Done : BOOL;
END_VAR

(* Вызов функции LIST_ADD *)
Done := OSCAT_BASIC.LIST_ADD(
  SEP := Separator,
  INS := NewRecord,
  LIST := List
);

(*
  Результат:
  - List = '&12&345&ABC'
  - Done = TRUE

  Пояснение:
  - Исходный список: '&12&345' (записи '12' и '345')
  - Добавляем запись 'ABC' в конец
  - Новая запись отделяется разделителем '&'
  - Итоговый список: '&12&345&ABC' (записи '12', '345', 'ABC')
*)
END_PROGRAM

```

29.2 LIST_CLEAN

Тип модуля: функция	Переменная	Тип	Описание
Входы	SEP	BYTE	ASCII-код символа-разделителя
Входы-выходы	LIST	STRING(LIST_LENGTH)	Список
Выходы	LIST_CLEAN	BOOL	Флаг завершения операции

Функция **LIST_CLEAN** очищает список от пустых записей. После выполнения операции выход функции принимает значение **TRUE**.

Пример

```

VAR
  (* Список с пустыми записями *)
  List : STRING(OSCAT_BASIC.LIST_LENGTH) := '&12&345&&ABC';
  (* Разделитель *)
  Separator : BYTE := 16#26;          (* '&' *)
  (* Флаг завершения операции *)
  Done : BOOL;
END_VAR

(* Вызов функции LIST_CLEAN *)
Done := OSCAT_BASIC.LIST_CLEAN(
  SEP := Separator,
  LIST := List
);

(*
Результат:
- List = '&12&345&ABC'
- Done = TRUE

Пояснение:
- Исходный список: '&12&345&&ABC' (4 записи: '12', '345', '', 'ABC')
- Пустая запись между '&&' удаляется
- Итоговый список: '&12&345&ABC' (3 записи: '12', '345', 'ABC')
*)
END_PROGRAM

```

29.3 LIST_GET

Тип модуля: функция	Переменная	Тип	Описание
Входы	SEP	BYTE	ASCII-код символа-разделителя
	POS	INT	Номер читаемой записи
Входы-выходы	LIST	STRING(LIST_LENGTH)	Список
Выходы	LIST_GET	STRING(LIST_LENGTH)	Прочитанная запись

Функция **LIST_GET** возвращает из списка **LIST** с разделителем **SEP** запись с номером **POS**. Нумерация записей начинается с 1.

Пример

```

VAR
  (* Список с записями *)
  List : STRING(OSCAT_BASIC.LIST_LENGTH) := '&12&345&ABC&XYZ';
  (* Разделитель *)
  Separator : BYTE := 16#26;          (* '&' *)
  (* Номер читаемой записи *)
  RecordNumber : INT := 2;
  (* Прочитанная запись *)
  Record : STRING(OSCAT_BASIC.LIST_LENGTH);
END_VAR

(* Вызов функции LIST_GET *)
Record := OSCAT_BASIC.LIST_GET(
  SEP := Separator,
  POS := RecordNumber,
  LIST := List
);

(*
  Результат: Record = '345'

  Пояснение:
  - Список: '&12&345&ABC&XYZ'
  - Записи: 1='12', 2='345', 3='ABC', 4='XYZ'
  - POS = 2 → возвращается вторая запись '345'
*)
END_PROGRAM

```

29.4 LIST_INSERT

Тип модуля: функция	Переменная	Тип	Описание
Входы	SEP	BYTE	ASCII-код символа-разделителя
	POS	INT	Позиция для вставки записи
	INS	STRING	Добавляемая запись
Входы-выходы	LIST	STRING(LIST_LENGTH)	Список
Выходы	LIST_INSERT	BOOL	Флаг завершения операции
Используемые модули	CHR_TO_STRING		

Функция **LIST_INSERT** добавляет в список **LIST** с разделителем **SEP** запись **INS**. Запись заменяет существующую запись номер **POS**, при этом существующая и все последующие записи смещаются. Если **POS=0**, то запись добавляется в начало списка (так же, как и при **POS=1**). Если значение **POS** превышает число существующих записей списка, то список дополняется пустыми записями.

После выполнения операции выход функции принимает значение **TRUE**.

Пример

```

VAR
  (* Список с записями *)
  List : STRING(OSCAT_BASIC.LIST_LENGTH) := '&12&345&ABC';

  (* Разделитель *)
  Separator : BYTE := 16#26;          (* '&' *)

  (* Позиция для вставки *)
  Position : INT := 2;

  (* Вставляемая запись *)
  NewRecord : STRING := 'NEW';

  (* Флаг завершения операции *)
  Done : BOOL;
END_VAR

(* Вызов функции LIST_INSERT *)
Done := OSCAT_BASIC.LIST_INSERT(
  SEP := Separator,
  POS := Position,
  INS := NewRecord,
  LIST := List
);

(*
Результат:
- List = '&12&NEW&345&ABC'
- Done = TRUE

Пояснение:
- Исходный список: '&12&345&ABC' (записи '12', '345', 'ABC')
- Вставляем 'NEW' в позицию 2
- Запись '345' и последующие сдвигаются вправо
- Итоговый список: '&12&NEW&345&ABC'
*)
END_PROGRAM

```

29.5 LIST_LEN

Тип модуля: функция	Переменная	Тип	Описание
Входы	SEP	BYTE	ASCII-код символа-разделителя
Входы-выходы	LIST	STRING(LIST_LENGTH)	Список
Выходы	LIST_LEN	INT	Число записей в списке

Функция **LIST_LEN** возвращает число записей в списке **LIST** с разделителем **SEP**.

Пример


```

VAR
  (* Список с записями *)
  List : STRING(OSCAT_BASIC.LIST_LENGTH) := '&12&345&ABC&XYZ';
  (* Разделитель *)
  Separator : BYTE := 16#26;          (* '&' *)
  (* Число записей в списке *)
  RecordCount : INT;
END_VAR

(* Вызов функции LIST_LEN *)
RecordCount := OSCAT_BASIC.LIST_LEN(
  SEP := Separator,
  LIST := List
);

(*
  Результат: RecordCount = 4

  Пояснение:
  - Список: '&12&345&ABC&XYZ'
  - Записи: '12', '345', 'ABC', 'XYZ'
  - Всего 4 записи
*)
END_PROGRAM

```

29.6 LIST_NEXT

Тип модуля: ФБ	Переменная	Тип	Описание
Входы	SEP	BYTE	ASCII-код символа-разделителя
	RST	BOOL	Сигнал перезапуска блока
Входы-выходы	LIST	STRING(LIST_LENGTH)	Список
Выходы	LEL	STRING(LIST_LENGTH)	Запись списка
	NUL	BOOL	Флаг «прочитаны все записи»

Функциональный блок **LIST_NEXT** используется для чтения записей из списка **LIST** с разделителем **SEP**. При каждом последующем вызове блока на выход **LEL** транслируется следующая запись списка. Когда прочитаны все записи (или в списке их нет), выход **NUL** принимает значение **TRUE**. По переднему фронту на входе **RST** происходит перезапуск блока.

Пример

```

VAR
  fbListNext:      OSCAT_BASIC.LIST_NEXT;
  sList:           STRING(OSCAT_BASIC.LIST_LENGTH) := 's34&sAB&1122sTest';
  asListElements:  ARRAY [1..10] OF STRING(OSCAT_BASIC.LIST_LENGTH);
  iPos:            INT;
END_VAR

iPos := 0;
fbListNext(LIST := sList, SEP := 16#26);

WHILE NOT fbListNext.NUL AND iPos <= 10 DO
  asListElements[iPos] := fbListNext.LEL;
  fbListNext(list := sList);
  iPos := iPos + 1;
END_WHILE
END_PROGRAM

```

29.7 LIST_RETRIEVE

Тип модуля: функция	Переменная	Тип	Описание
Входы	SEP	BYTE	ASCII-код символа-разделителя
	POS	INT	Номер вырезаемой записи
Входы-выходы	LIST	STRING(LIST_LENGTH)	Список
Выходы	LIST_RETRIEVE	STRING(LIST_LENGTH)	Вырезанная запись

Функция **LIST_RETRIEVE** вырезает из списка **LIST** с разделителем **SEP** запись с номером **POS**. Нумерация запись начинается с 1. Если элемент с данным номером отсутствует, то функция возвращает пустую строку. См. также функцию [LIST_RETRIEVE_LAST](#), которая используется для вырезания последней записи списка.

Пример

```

VAR
  (* Список с записями *)
  List : STRING(OSCAT_BASIC.LIST_LENGTH) := '&12&345&ABC&XYZ';
  (* Разделитель *)
  Separator : BYTE := 16#26;          (* '&' *)
  (* Номер вырезаемой записи *)
  Position : INT := 2;
  (* Вырезанная запись *)
  RetrievedRecord : STRING(OSCAT_BASIC.LIST_LENGTH);
END_VAR

(* Вызов функции LIST_RETRIEVE *)
RetrievedRecord := OSCAT_BASIC.LIST_RETRIEVE(
  SEP := Separator,
  POS := Position,
  LIST := List
);

(*
  Результат:
  - RetrievedRecord = '345'
  - List = '&12&ABC&XYZ'

  Пояснение:
  - Исходный список: '&12&345&ABC&XYZ' (записи '12', '345', 'ABC', 'XYZ')
  - Вырезаем запись с номером 2 ('345')
  - Запись удаляется из списка
  - Итоговый список: '&12&ABC&XYZ' (записи '12', 'ABC', 'XYZ')
*)
END_PROGRAM

```

29.8 LIST_RETRIEVE_LAST

Тип модуля: функция	Переменная	Тип	Описание
Входы	SEP	BYTE	ASCII-код символа-разделителя
Входы-выходы	LIST	STRING(LIST_LENGTH)	Список
Выходы	LIST_RETRIEVE_LAST	STRING(LIST_LENGTH)	Вырезанная запись

Функция **LIST_RETRIEVE_LAST** вырезает из списка **LIST** с разделителем **SEP** последнюю запись. См. также функцию [LIST_RETRIEVE](#), которая используется для вырезания заданной записи списка.

Пример

```
VAR
  (* Список с записями *)
  List : STRING(OSCAT_BASIC.LIST_LENGTH) := '&12&345&ABC&XYZ';
  (* Разделитель *)
  Separator : BYTE := 16#26;          (* '&' *)
  (* Вырезанная запись *)
  RetrievedRecord : STRING(OSCAT_BASIC.LIST_LENGTH);
END_VAR

(* Вызов функции LIST_RETRIEVE_LAST *)
RetrievedRecord := OSCAT_BASIC.LIST_RETRIEVE_LAST(
  SEP := Separator,
  LIST := List
);

(*
  Результат:
  - RetrievedRecord = 'XYZ'
  - List = '&12&345&ABC'
*)
END_PROGRAM
```



ЦИФРОВЫЕ
РЕШЕНИЯ

ООО "Овен Цифровые решения"

Россия, г. Москва, пл. Семёновская, д. 1А, помещ. 3/1

support@owendigital.ru

www.owendigital.ru

пер.:1-RU-165227-1.1