EXT — язык расширения CCU

Руководство пользователя

8 апреля 2016 г.

Содержание

1 Элементы языка

1.1 Числовые литералы

Числовой литерал — это десятичное, шестнадцатеричное, восьмеричное или двоичное число, которое представляет собой целое значение (опционально может предшествовать знак минус). По умолчанию число является десятичным (основание 10). Систему счисления можно задавать с помощью префикса.

Префикс	Система счисления
0x, 0X	Шестнадцатеричная (основание 16)
0o, 0O	Восьмеричная (основание 8)
0b, 0B	Двоичная (основание 2)

Таблица 1: Префиксы систем счисления

```
12345
-1
0xabcd
-0xEF
0o123
-0o777
0b0101
-0b1010
```

Для читабельности можно использовать нижнее подчеркивание (_) для разделения групп в составе числа.

```
1_048_576
0x1234_ABCD
0b1111_1010_0101_0000
```

1.2 Идентификаторы

Идентификаторы — это имена, присваиваемые константам, переменным и подпрограммам. Идентификатор должен начинаться с символа из множества [A–Z a–z _]. Последующие символы должны быть из множества [A–Z a–z _ ' 0–9]. Идентификаторы являются регистрозависимыми. Ниже приведены примеры правильных идентификаторов.

```
foo
_foo
Foo
foo_bar123'
```

1.3 Ключевые слова

Зарезервированные слова, которые не могут использоваться в качестве идентификаторов.

```
const var fun proc if else while break continue return
```

1.4 Комментарии

Комментарии однострочные. Комментарий отмечается специальным символом # в начале и продолжается до конца строки.

```
# функция сложения

fun add(x, y)
{
    return x + y; # вернуть результат
}
```

2 Типы данных

В языке существует единственный тип данных: 32-битное знаковое целое. Диапазон значений от -2147483648 (-2^{31}) до 2147483647 ($2^{31} - 1$).

Логические (булевы) значения задаются целым: *ложь* (*false*) — ноль, *истина* (*true*) — отличное от нуля значение.

3 Объявления

3.1 Константы

Константы объявляются на верхнем уровне (уровне глобальных переменных, вне подпрограмм) с помощью ключевого слова *const* и обязательно инициализируются константным выражением, которое должно вычисляться на этапе компиляции. Одновременно можно объявить несколько констант, разделяя их запятой. Объявление должно завершаться точкой с запятой.

```
const kilo = 1000, mega = kilo * 1000;

fun S(r)
{
    return PI * r * r / 10000;
}

const PI = 31415;
```

3.2 Переменные

Переменные объявляются с помощью ключевого слова *var*. Неинициализированные переменные по умолчанию инициализируются нулем. Одновременно можно объявить несколько переменных, разделяя их запятой. Объявление должно завершаться точкой с запятой.

```
var x;
var y, z = 5;
```

3.2.1 Глобальные переменные

Глобальные переменные объявляются вне всех подпрограмм (порядок объявлений не имеет значения) и могут инициализироваться константным выражением, которое должно вычисляться на этапе компиляции. Областью видимости глобальных переменных является вся программа.

```
var a = 1;

fun d(c)
{
    return b * b - 4 * a * c;
}

var b = (2 + 3) * 4;
```

3.2.2 Локальные переменные

Локальные переменные объявляются внутри блока подпрограммы (*fun* и *proc*), условного оператора (*if-else*), цикла с предусловием (*while*) и могут инициализироваться выражением. Локальные переменные имеют лексическую область видимости (видимость ограничена текстом определения подпрограммы, условного оператора, цикла с предусловием) и могут объявляться в произвольном порядке, но до места их использования.

```
fun roots(a, b, c)
{
    var d = b * b - 4 * a * c;

    if d > 0 {
        return 2;
    } else if d == 0 {
        return 1;
    } else {
        return 0;
```

```
    }
}
```

3.2.3 Параметры подпрограмм

Параметры (аргументы) подпрограммы задаются в круглых скобках через запятую в момент определения подпрограммы и имеют лексическую область видимости, аналогично локальным переменным. При определении подрограммы без параметров используются пустые круглые скобки ().

3.3 Подпрограммы

Подпрограмма — это именованная часть программы, содержащая описание определенного набора действий. В языке используется два типа подпрограмм: функции и процедуры.

3.3.1 Функции

Функция — это подпрограмма, которая обязательно возвращает результат. Если функция не возвращает результат явно с помощью *return*, то по умолчанию возвращается ноль. Функция определяется с помощью ключевого слова *fun*.

```
const c = 299_792_458;

fun E(m)
{
    return m * c * c;
}
```

3.3.2 Процедуры

Процедура — это подпрограмма, которая не возвращает результат. Процедура определяется с помощью ключевого слова *proc*.

```
proc hyper_jump(where)
{
    set_destination(where);
    turn_warp_drive_on();
}
```

4 Операторы

Блок — это последовательность операторов, заключенная в фигурные скобки {}. Операторы должны завершаться точкой с запятой, кроме условного оператора и

цикла с предусловием (они завершаются фигурными скобками).

4.1 Присваивание

Присваивание позволяет изменить значение ранее определенной переменной. Новое значение является результатом вычисления выражения правой части оператора присваивания.

```
x = x + 1;
```

4.2 Вызов процедуры

Вызов процедуры является оператором. Так как процедура не возвращает значение, то ее вызов может использоваться только в составе блока (последовательности операторов).

```
p(x, y);
```

4.3 Возврат из подпрограммы

4.3.1 Возврат из процедуры

Возврат из процедуры используется для ее явного завершения (например, по определенному условию).

```
proc p(x)
{
    if x > 10 {
        return;
    }
    do_something();
}
```

Если процедура не имеет *return*, то она завершается после выполнения последнего оператора.

```
proc p()
{
    do_first();
    do_next();
    do_last();
}
```

4.3.2 Возврат из функции.

Результатом работы функции является результат вычисления выражения, заданного в операторе *return*.

```
fun f(x)
{
    return x + 1;
}
```

Если функция не возвращает результат явно с помощью *return*, то по умолчанию возвращается ноль.

```
fun should_return_zero(x)
{
}
```

4.4 Управляющие конструкции

4.4.1 Условный оператор

Оператор *if-else* является стандартным условным оператором. Фигурные скобки {} обязательны, даже если ветвь содержит один оператор.

```
fun cmp(a, b)
{
    if a > b {
        return 1;
    } else if a < b {
        return -1;
    } else {
        return 0;
    }
}
```

4.4.2 Цикл с предусловием

Оператор *while* является стандартным циклом с предусловием. Фигурные скобки {} обязательны, даже если цикл содержит один оператор.

```
fun gcd(a, b)
{
    var t;
    while b != 0 {
        t = a;
        a = b;
```

```
        b = t % b;
    }
    return a;
}
```

Оператор *break* является оператором завершения цикла. Оператор *continue* является оператором пропуска итерации цикла.

```
while i < n {
    if i == m {
        break;
    }
    if i == k {
        continue;
    }
    i = i + 1;
}
```

5 Выражения

5.1 Операции

Унарные операции — это операции, содержащие единственный операнд, перед которым расположен знак операции. Бинарные операции — это операции, содержащие два операнда, между которыми расположен знак операции.

В следующей таблице отражены приоритет и ассоциативность операций. Операции перечислены сверху вниз в порядке убывания приоритета.

Знак операции	Описание	Ассоциативность
Унарные операции		
-	Унарный минус	Нет
~	Побитовое НЕ	
!	Логическое НЕ	
Бинарные операции		
*	Умножение	Слева-направо
/	Деление	
%	Остаток	
+	Сложение	
-	Вычитание	
<<	Побитовый сдвиг влево	
>>	Побитовый сдвиг вправо	
<	Меньше	
<=	Меньше или равно	
>	Больше	
>=	Больше или равно	
==	Равно	
!=	Не равно	
&	Побитовое И	
^	Побитовое исключающее ИЛИ	
|	Побитовое ИЛИ	
&&	Логическое И	
||	Логическое ИЛИ	

Таблица 2: Приоритет и ассоциативность операций

Для задания приоритета используются круглые скобки.

```
var x = (y + z) / 2;
```

5.2 Вызов функции

Вызов функции является выражением. Он должен использоваться в других выражениях или в качестве правой части присваивания.

```
z = f(g(x) + y);
```

6 Встроенные константы и подпрограммы

Идентификаторы встроенных констант и подпрограмм начинаются с символа $. Данные идентификаторы встроены в язык и не могут быть определены или переопределены пользователем.

6.1 Функция $get_event_id

Получение идентификатора текущего события.

```
var e = $get_event_id();
```

Возможные идентификаторы событий перечислены в таблицах 3, 4, 5, 6, 7, 8.

Имя	Описание
$EVT_POWER_FAULT	Отключение внешнего питания
$EVT_POWER_RECOVERY	Восстановление внешнего питания
$EVT_BATTERY_LOW1	Разряд батареи до 1 уровня
$EVT_BATTERY_LOW2	Разряд батареи до 2 уровня
$EVT_TEMP_LOW	Температура платы упала до нижней границы
$EVT_TEMP_NORMAL	Температура платы вернулась в допустимый диапазон
$EVT_TEMP_HIGH	Температура платы поднялась до верхней границы
$EVT_CASE_OPEN	Вскрытие корпуса контроллера
$EVT_BALANCE_LOW	Баланс снизился до минимального значения
$EVT_DEVICE_ON	Контроллер включен
$EVT_DEVICE_RESTART	Контроллер перезапущен
$EVT_FIRMWARE_UPGRADE	Прошивка обновлена
$EVT_RELAY_CONN_FAILED	Контроллер не смог подключиться к серверу-ретранслятору
$EVT_GPRS_CONN_FAILED	Контроллер не смог подключиться к сервису GPRS

Таблица 3: Системные события

Имя	Описание
$EVT_INPUT1_ACTIVE	Вход 1 активен
$EVT_INPUT2_ACTIVE	Вход 2 активен
$EVT_INPUT3_ACTIVE	Вход 3 активен
$EVT_INPUT4_ACTIVE	Вход 4 активен
$EVT_INPUT5_ACTIVE	Вход 5 активен
$EVT_INPUT6_ACTIVE	Вход 6 активен
$EVT_INPUT7_ACTIVE	Вход 7 активен
$EVT_INPUT8_ACTIVE	Вход 8 активен
$EVT_INPUT9_ACTIVE	Вход 9 активен
$EVT_INPUT10_ACTIVE	Вход 10 активен
$EVT_INPUT11_ACTIVE	Вход 11 активен
$EVT_INPUT12_ACTIVE	Вход 12 активен
$EVT_INPUT13_ACTIVE	Вход 13 активен
$EVT_INPUT14_ACTIVE	Вход 14 активен
$EVT_INPUT15_ACTIVE	Вход 15 активен
$EVT_INPUT16_ACTIVE	Вход 16 активен
$EVT_INPUT1_PASSIVE	Вход 1 пассивен
$EVT_INPUT2_PASSIVE	Вход 2 пассивен
$EVT_INPUT3_PASSIVE	Вход 3 пассивен
$EVT_INPUT4_PASSIVE	Вход 4 пассивен
$EVT_INPUT5_PASSIVE	Вход 5 пассивен
$EVT_INPUT6_PASSIVE	Вход 6 пассивен
$EVT_INPUT7_PASSIVE	Вход 7 пассивен
$EVT_INPUT8_PASSIVE	Вход 8 пассивен
$EVT_INPUT9_PASSIVE	Вход 9 пассивен
$EVT_INPUT10_PASSIVE	Вход 10 пассивен
$EVT_INPUT11_PASSIVE	Вход 11 пассивен
$EVT_INPUT12_PASSIVE	Вход 12 пассивен
$EVT_INPUT13_PASSIVE	Вход 13 пассивен
$EVT_INPUT14_PASSIVE	Вход 14 пассивен
$EVT_INPUT15_PASSIVE	Вход 15 пассивен
$EVT_INPUT16_PASSIVE	Вход 16 пассивен

Таблица 4: События при изменении состояния входов

Имя	Описание
$EVT_ARM	Охрана
$EVT_DISARM	Наблюдение
$EVT_PROTECT	Защита
$EVT_ARM1	Раздел 1: Охрана
$EVT_DISARM1	Раздел 1: Наблюдение
$EVT_ARM2	Раздел 2: Охрана
$EVT_DISARM2	Раздел 2: Наблюдение
$EVT_ARM3	Раздел 3: Охрана
$EVT_DISARM3	Раздел 3: Наблюдение
$EVT_ARM4	Раздел 4: Охрана
$EVT_DISARM4	Раздел 4: Наблюдение

Таблица 5: События при изменении режима охраны

Имя	Описание
$EVT_TEST	Тестовое сообщение
$EVT_INFO	Информационное сообщение

Таблица 6: Тестовые и информационные события

Имя	Описание
$EVT_PROFILE1_APPLIED	Применен профиль 1
$EVT_PROFILE2_APPLIED	Применен профиль 2
$EVT_PROFILE3_APPLIED	Применен профиль 3
$EVT_PROFILE4_APPLIED	Применен профиль 4
$EVT_PROFILE5_APPLIED	Применен профиль 5
$EVT_PROFILE6_APPLIED	Применен профиль 6
$EVT_PROFILE7_APPLIED	Применен профиль 7
$EVT_PROFILE8_APPLIED	Применен профиль 8

Таблица 7: События при применении профилей

Имя	Описание
$EVT_INIT	Инициализация
$EVT_TIMER1	Таймер 1
$EVT_TIMER2	Таймер 2
$EVT_TIMER3	Таймер 3
$EVT_TIMER4	Таймер 4
$EVT_ALARM1	Будильник 1
$EVT_ALARM2	Будильник 2
$EVT_ALARM3	Будильник 3
$EVT_ALARM4	Будильник 4

Таблица 8: События среды выполнения

6.2 Функция $get_input_state

Получение состояния входа n (1 — активен; 0 — пассивен). Значение параметра n должно быть от 1 до 8 или от 1 до 16 (в зависимости от модификации контроллера).

```
var s = $get_input_state(n);
```

6.3 Функция $get_input_value

Получение значения входа n (от 0 до 4095, диапазон измерения 10 вольт). Значение параметра n должно быть от 1 до 8 или от 1 до 16 (в зависимости от модификации контроллера).

```
var v = $get_input_value(n);
```

6.4 Функция $get_input_low_limit

Получение значения нижней границы входа n (от 0 до 4095, диапазон измерения 10 вольт). Значение параметра n должно быть от 1 до 8 или от 1 до 16 (в зависимости от модификации контроллера).

```
var v = $get_input_low_limit(n);
```

6.5 Функция $get_input_high_limit

Получение значения верхней границы входа n (от 0 до 4095, диапазон измерения 10 вольт). Значение параметра n должно быть от 1 до 8 или от 1 до 16 (в зависимости от модификации контроллера).

```
var v = $get_input_high_limit(n);
```

6.6 Функция $get_sensor_value

Получение значения датчика, подключенного ко входу n. Значение параметра n должно быть от 1 до 8 или от 1 до 16 (в зависимости от модификации контроллера). Результатом является 32-битное знаковое число с фиксированной точкой (fixed-point). Параметр p задает размер *дробной* части в битах, и его значение должно быть от 0 до 31.

```
var v = $get_sensor_value(n, p);
```

6.7 Функция $get_sensor_low_limit

Получение значения нижней границы датчика, подключенного ко входу n. Значение параметра n должно быть от 1 до 8 или от 1 до 16 (в зависимости от модификации контроллера). Результатом является 32-битное знаковое число с фиксированной точкой (fixed-point). Параметр p задает размер *дробной* части в битах, и его значение должно быть от 0 до 31.

```
var v = $get_sensor_low_limit(n, p);
```

6.8 Функция $get_sensor_high_limit

Получение значения верхней границы датчика, подключенного ко входу n. Значение параметра n должно быть от 1 до 8 или от 1 до 16 (в зависимости от модификации контроллера). Результатом является 32-битное знаковое число с фиксированной точкой (fixed-point). Параметр p задает размер *дробной* части в битах, и его значение должно быть от 0 до 31.

```
var v = $get_sensor_high_limit(n, p);
```

6.9 Функция $get_output_state

Получение состояния выхода n (таблица 12). Значение параметра n должно быть от 1 до 7 (1, 2 — реле; 3, 4, 5, 6, 7 — выходы).

```
var s = $get_output_state(n);
```

6.10 Функция $get_arm_mode

Получение текущего режима охраны для модификации контроллера с одним разделом.

```
var m = $get_arm_mode();
```

Имя	Описание
$ARM	Охрана
$DISARM	Наблюдение
$PROTECT	Защита

Таблица 9: Режимы охраны

6.11 Функция $get_part_arm_mode

Получение текущего режима охраны раздела n для модификации контроллера с несколькими разделами. Значение параметра n должно быть от 1 до 4.

```
var m = $get_part_arm_mode(n);
```

Имя	Описание
$ARM	Охрана
$DISARM	Наблюдение

Таблица 10: Режимы охраны раздела

6.12 Функция $is_power_on

Получение состояния внешнего питания (1 — в норме; 0 — отсутствует).

6.13 Функция $get_power_voltage

Получение напряжения внешнего питания (≥ 0 — напряжение в вольтах, умноженное на 10; -1 — внешнее питание отсутствует).

```
if $is_power_on() {
    var v = $get_power_voltage();
}
```

6.14 Функция $get_battery_state

Получение состояния батареи.

```
var s = $get_battery_state();
```

Имя	Описание
$BATTERY_NOT_USED	Не использовалась
$BATTERY_DISCONNECTED	Отключена
$BATTERY_LOW1	Разряжена до 1 уровня
$BATTERY_LOW2	Разряжена до 2 уровня
$BATTERY_OK	В норме

Таблица 11: Состояния батареи

6.15 Функция $get_battery_charge

Получение уровня заряда батареи (≥ 0 — заряд в процентах; -1 — батарея не использовалась или отключена).

```
var s = $get_battery_state();
if s != $BATTERY_NOT_USED && s != $BATTERY_DISCONNECTED {
    var c = $get_battery_charge();
}
```

6.16 Функция $is_temp_valid

Проверка достоверности значения температуры системной платы (1 — достоверно; 0 — недостоверно).

6.17 Функция $get_temp

Получение температуры системной платы (в градусах Цельсия).

```
if $is_temp_valid() {
    var t = $get_temp();
}
```

6.18 Функция $is_case_opened

Проверка состояния корпуса контроллера (1 — открыт; 0 — закрыт).

```
var s = $is_case_opened();
```

6.19 Функция $is_balance_valid

Проверка достоверности значения баланса (1 — достоверно; 0 — недостоверно).

6.20 Функция $get_balance

Получение баланса (значение, умноженное на 100).

```
if $is_balance_valid() {
    var b = $get_balance();
}
```

6.21 Функция $get_time

Получение текущего времени контроллера в секундах с 1 января 2000 года 00:00:00 (в установленном часовом поясе). Если значение меньше нуля, то время не определено по причине остановки часов реального времени (RTC). Максимальное значение времени соответствует 2147483647 ($2^{31} - 1$) секундам или 19 января 2068 03:14:07 (в установленном часовом поясе).

6.22 Функция $get_year

Получение года (YYYY) из текущего времени контроллера.

6.23 Функция $get_month

Получение месяца (1–12) из текущего времени контроллера.

6.24 Функция $get_day

Получение дня месяца (1–31) из текущего времени контроллера.

6.25 Функция $get_day_of_week

Получение дня недели (1–7) из текущего времени контроллера.

6.26 Функция $get_hour

Получение часов (0–23) из текущего времени контроллера.

6.27 Функция $get_minute

Получение минут (0–59) из текущего времени контроллера.

6.28 Функция $get_second

Получение секунд (0–59) из текущего времени контроллера.

```
var t = $get_time();
if t >= 0 {
    var y = $get_year(t);
    var m = $get_month(t);
    var d = $get_day(t);
    var dow = $get_day_of_week(t);
    var hh = $get_hour(t);
    var mm = $get_minute(t);
    var ss = $get_second(t);
}
```

6.29 Процедура $set_output_state

Изменение состояния выхода n. Значение параметра n должно быть от 1 до 7 (1, 2 — реле; 3, 4, 5, 6, 7 — выходы).

```
$set_output_state(n, s);
```

Имя	Описание
$OFF	Выключен
$ON	Включен
$SCENARIO1	Сценарий 1
$SCENARIO2	Сценарий 2
$SCENARIO3	Сценарий 3
$SCENARIO4	Сценарий 4
$SCENARIO5	Сценарий 5
$SCENARIO6	Сценарий 6
$SCENARIO7	Сценарий 7
$SCENARIO8	Сценарий 8
$SCENARIO9	Сценарий 9
$SCENARIO10	Сценарий 10
$SCENARIO11	Сценарий 11
$SCENARIO12	Сценарий 12
$SCENARIO13	Сценарий 13
$SCENARIO14	Сценарий 14

Таблица 12: Состояния выходов

6.30 Процедура $set_output_pulse

Формирование импульса: установка выхода n в состояние s ($ON или $OFF) на время t (в одной единице времени 100 миллисекунд; диапазон значений от 1 до

42949672). Значение параметра n должно быть от 1 до 7 (1, 2 — реле; 3, 4, 5, 6, 7 — выходы).

```
$set_output_pulse(n, s, t);
```

6.31 Процедура $set_arm_mode

Изменение текущего режима охраны для модификации контроллера с одним разделом. Возможные значения параметра m перечислены в таблице 9.

```
$set_arm_mode(m);
```

6.32 Процедура $set_part_arm_mode

Изменение текущего режима охраны раздела n для модификации контроллера с несколькими разделами. Значение параметра n должно быть от 1 до 4. Возможные значения параметра m перечислены в таблице 10.

```
$set_part_arm_mode(n, m);
```

6.33 Процедура $apply_profile

Применение профиля n. Значение параметра n должно быть от 1 до 8.

```
$apply_profile(n);
```

6.34 Процедура $set_event_mask

Установка маски событий. Маски событий могут объединяться с помощью операции *побитовое ИЛИ* (|).

```
var m = $EM_INPUT | $EM_SYSTEM;
$set_event_mask(m);
```

Имя	Описание
$EM_INPUT	События при изменении состояния входов
$EM_ARM_MODE	События при изменении режима охраны
$EM_SYSTEM	Системные события
$EM_TEST_INFO	Тестовые и информационные события
$EM_PROFILE	События при применении профилей
$EM_ALL	Все типы событий

Таблица 13: Маски событий

6.35 Процедура $reset_event_mask

Сброс маски событий (таблица 13).

```
$reset_event_mask(m);
```

6.36 Процедура $set_timer

Установка периодического таймера n с интервалом t (в одной единице времени 100 миллисекунд; диапазон значений от 1 до 42949672). По истечении времени t будет вызвано соответствующее событие таймера (таблица 8). Значение параметра n должно быть от 1 до 4.

```
$set_timer(n, t);
```

6.37 Процедура $reset_timer

Сброс (остановка) таймера n. Значение параметра n должно быть от 1 до 4.

```
$reset_timer(n);
```

6.38 Процедура $set_alarm

Установка периодического будильника n на время hh:mm. После сработки будет вызвано соответствующее событие будильника (таблица 8). Значение параметра n должно быть от 1 до 4. Значение параметра hh задает часы в диапазоне от 0 до 23. Значение параметра mm задает минуты в диапазоне от 0 до 59.

```
$set_alarm(n, hh, mm);
```

6.39 Процедура $reset_alarm

Сброс (остановка) будильника n. Значение параметра n должно быть от 1 до 4.

```
$reset_alarm(n);
```

6.40 Процедура $raise_event

Вызов события с идентификатором e. Возможные идентификаторы событий перечислены в таблицах 4 и 6. Данное событие вызовет заданное в конфигурации контроллера оповещение (сетевое, голосовое, SMS).

```
$raise_event(e);
```

6.41 Процедура $cancel_event

Отмена текущего события. Данное событие не будет вызывать заданное в конфигурации контроллера оповещение (сетевое, голосовое, SMS). Идентификаторы событий, которые могут быть отменены, перечислены в таблицах 3, 4, 5, 6, 7.

```
$cancel_event();
```

7 Событийно-ориентированная среда выполнения

7.1 Основные понятия

Точкой входа в программу является процедура *main* без параметров. Выполнение программы определяется событиями: при возникновении события среда выполнения передает управление в процедуру *main*. Идентификатор текущего события должен быть получен с помощью функции *$get_event_id*.

```
proc main()
{
    var e = $get_event_id();
    handle_event(e);
}
```

Среда выполнения однократно вызывает событие *$EVT_INIT* в следующих случаях:

- После загрузки программы в контроллер с помощью *CCU shell* (обновление программы).
- После включения или перезагрузки контроллера.

При обработке данного события должна быть задана необходимая маска событий с помощью процедуры *$set_event_mask* (таблица 13). После этого среда выполнения будет передавать управление в процедуру *main* при возникновении разрешенных типов событий. События среды выполнения (таблица 8) разрешены всегда и не могут быть запрещены с помощью процедуры *$reset_event_mask*.

```
proc main()
{
    var e = $get_event_id();
    if e == $EVT_INIT {
        $set_event_mask($EM_INPUT | $EM_SYSTEM);
    } else if e == $EVT_INPUT1_ACTIVE {
        handle_input1_active();
    } else if e == $EVT_POWER_FAULT {
        handle_power_fault();
    } else {
```

```
        handle_event(e);
    }
}
```

7.2 Ошибки выполнения программы

В следующей таблице перечислены ошибочные ситуации времени выполнения, которые приводят к остановке программы. В этом случае код ошибки будет отображен в *CCU shell* в разделе *Программируемая логика*, а также отправлен пользователям, у которых в конфигурации контроллера разрешены системные события.

Код ошибки	Описание
1	Неверный код инструкции
2	Неверный код встроенной подпрограммы
3	Деление на ноль
4	Переполнение стека сверху
5	Переполнение стека снизу
6	Переполнение кода сверху
7	Переполнение кода снизу
8	Переполнение данных сверху
9	Переполнение данных снизу
10	Ошибка чтения флеш-памяти
11	Превышено максимально допустимое время обработки события
1025	Неверный параметр встроенной подпрограммы *$get_input_state*
1026	Неверный параметр встроенной подпрограммы *$get_input_value*
1027	Неверный параметр встроенной подпрограммы *$get_sensor_value*
1028	Неверный параметр встроенной подпрограммы *$get_output_state*
1029	Недопустимый вызов встроенной подпрограммы *$get_arm_mode* для данной модификации контроллера
1030	Неверный параметр встроенной подпрограммы *$get_part_arm_mode* или недопустимый вызов для данной модификации контроллера
1041	Неверный параметр встроенной подпрограммы *$get_year*
1042	Неверный параметр встроенной подпрограммы *$get_month*

1043	Неверный параметр встроенной подпрограммы *$get_day*
1044	Неверный параметр встроенной подпрограммы *$get_day_of_week*
1045	Неверный параметр встроенной подпрограммы *$get_hour*
1046	Неверный параметр встроенной подпрограммы *$get_minute*
1047	Неверный параметр встроенной подпрограммы *$get_second*
1048	Неверный параметр встроенной подпрограммы *$set_output_state*
1049	Неверный параметр встроенной подпрограммы *$set_output_pulse*
1050	Неверный параметр встроенной подпрограммы *$set_arm_mode* или недопустимый вызов для данной модификации контроллера
1051	Неверный параметр встроенной подпрограммы *$set_part_arm_mode* или недопустимый вызов для данной модификации контроллера
1052	Неверный параметр встроенной подпрограммы *$apply_profile*
1053	Неверный параметр встроенной подпрограммы *$set_event_mask*
1054	Неверный параметр встроенной подпрограммы *$reset_event_mask*
1055	Неверный параметр встроенной подпрограммы *$set_timer*
1056	Неверный параметр встроенной подпрограммы *$reset_timer*
1057	Неверный параметр встроенной подпрограммы *$set_alarm*
1058	Неверный параметр встроенной подпрограммы *$reset_alarm*
1059	Неверный параметр встроенной подпрограммы *$raise_event*
1060	Неверный параметр встроенной подпрограммы *$get_input_low_limit*
1061	Неверный параметр встроенной подпрограммы *$get_input_high_limit*
1062	Неверный параметр встроенной подпрограммы *$get_sensor_low_limit*

1063	Неверный параметр встроенной подпрограммы *$get_sensor_high_limit*

Таблица 14: Ошибки выполнения программы

8 Интегрированная среда разработки

Интегрированная среда разработки состоит из компилятора и редактора программного кода, которые встроены в *CCU shell* в разделе *Программируемая логика.*

Клавиши	Описание
Стрелки	Управление курсором
Home	Перейти в начало строки
End	Перейти в конец строки
Ctrl+Home (PC), Cmd+Home (Mac)	Перейти в начало текста
Ctrl+End (PC), Cmd+Down (Mac)	Перейти в конец текста
Page Up	Перейти на страницу вверх
Page Down	Перейти на страницу вниз
Enter	Вставить новую строку
Backspace	Удалить символ слева от курсора
Delete	Удалить символ справа от курсора
Ctrl+D (PC), Cmd+D (Mac)	Удалить строку
Ctrl+A (PC), Cmd+A (Mac)	Выделить все
Ctrl+U (PC), Cmd+U (Mac)	Отменить выделение
Ctrl+F (PC), Cmd+F (Mac)	Найти
Ctrl+G (PC), Cmd+G (Mac)	Найти следующий
Shift+Ctrl+G (PC), Shift+Cmd+G (Mac)	Найти предыдущий
Shift+Ctrl+F (PC), Cmd+Alt+F (Mac)	Заменить
Shift+Ctrl+R (PC), Shift+Cmd+Alt+F (Mac)	Заменить все
Ctrl+Z (PC), Cmd+Z (Mac)	Отмена последнего действия
Ctrl+Y (PC), Cmd+Y (Mac)	Повтор последнего отмененного действия
Ctrl+Enter	Полноэкранный режим вкл./выкл.

Таблица 15: Основные сочетания клавиш редактора программного кода

Команда	Описание
Компилировать	Компилировать исходный код программы в редакторе
Загрузить и запустить	Загрузить в контроллер и запустить скомпилированную программу
Обновить из контроллера	Загрузить в редактор исходный код программы из контроллера
Скачать	Скачать из контроллера исходный код программы в файл *prog.ext*
Открыть файл	Загрузить в редактор исходный код программы из файла с расширением *ext*
Пустая программа	Загрузить в редактор исходный код пустой программы

Таблица 16: Команды интегрированной среды разработки