Описание языка IndustrialC

Разработчик

_____А. С. Розов

"____"_____2018 г.

2018

Описание языка IndustrialC

АЭ161.1705.10000-01 35

(CD ROM)

Листов 11

2018

Содержание

1. ОБЩИЕ СВЕДЕНИЯ

Язык предназначен для разработки ПО микроконтроллеров во встраиваемых системах с использованием методики процесс-ориентированного программирования. Синтаксис основан на языках Си и Reflex и расширен конструкциями для работы с прерываниями.

Программа состоит из одного .ic – файла (модуля) и набора заголовочных файлов .ih. Все файлы должны находиться в одной папке. IndustrialC поддерживает большую часть синтаксиса языка Си. Также предусмотрен механизм вставки Си кода в программу на industrialC.

2. ЭЛЕМЕНТЫ ЯЗЫКА

2.1. Программа – набор объявлений:

- переменных;
- векторов, регистров, бит;
- гиперпроцессов;
- процессов.

2.2. Объявление переменных

IndustrialC поддерживает объявление переменных в стиле языка Си. Возможно объявление списка переменных через запятую, объявление массивов постоянной длины, инициализация переменных и массивов таким же образом, как в языке Си. Указатели, структуры, союзы (union) и перечисления (enum) в текущей версии не поддерживаются.

Типы переменных: *void*, *char*, *int*, *short*, *long*, *float*, *signed*, *unsigned*, *bool*.

Классификаторы типа: *const* и *volatile*.

Примеры:

```
unsigned int a = 5, b, c = 1, blah;
float some_variable_name, other_var_name;
```

```
    float some_array[100];
    const char NAMES[3][5] = { "some", "stri", "ngs
",};
    char some_text[]="Process-Oriented Programming";
```

Переменные, объявленные вне состояний и функций транслируются в глобальные.

Локальные переменные в функциях и состояниях транслируются в локальные.

2.3. Объявления векторов, регистров и битов

Формат объявления векторов регистор и битов в языке IndustrialC:

```
vector имя_вектора;
register имя_регистра;
bit имя_бита;
```

Эти объявления необходимы для того, чтобы транслятор узнавал макроопределения имен векторов, регистров и бит языка Си и не выдавал ошибку. Для каждого микроконтроллера составляется файл .ih с набором таких объявлений и подшивается к тексту программы.

Примеры:

```
register UCSR0A;
vector TIMER2_COMPA_vect;
vector USART_UDRE_vect;
bit MPCM0;
bit U2X0;
```

2.4. Объявление гиперпроцесса

Формат объявления гиперпроцессов в языке IndustrialC:

```
hyperprocess имя_гиперпроцесса
{
vector = имя_вектора;
register = имя_регистра;
bit = имя_бита;
}
```

Имя гиперпроцесса задается произвольным образом. Имена вектора, регистра и бита должны быть объявлены в программе (п. 3). Вектор задает прерывание, которым будет активироваться гиперпроцесс, регистр и бит будут использоваться для включения/выключения прерывания операторами

start hyperprocess и stop hyperprocess. Предполагается что прерывание разрешено, когда бит в регистре выставлен в 1.

Пример:

```
hyperprocess Blink
{
vector = TIMER1_COMPA_vect;
register = TIMSK1;
bit = OCIE1A;
}
```

2.5. Объявление процесса

Формат объявления процессов в языке IndustrialC:

```
process имя_процесса : имя_гиперпроцесса { тело_процесса  }
```

Пример:

```
process BlinkLED: Blink
{
}
```

Тело процесса – набор объявлений:

- переменных
- состояний

2.6. Объявление состояния

Формат объявления состояний в языке IndustrialC:

```
state имя_состояния { тело_состояния }
```

Пример:

```
state WaitTransmit { … }
```

Тело состояния – набор:

- объявлений переменных
- утверждений
- определение таймаута (максимум одно)

2.7. Утверждения

Формат записи утверждений в языке IndustrialC:

```
set state имя_состояния; // переход в другое состояние
start process имя_процесса; // запуск процесса
stop process имя_процесса; // остановка процесса
stop process; // остановка процесса, в котором мы находимся
```

```
выражение; // Например, “a=5;”
start hyperprocess имя_гиперпроцесса; // разрешает прерывание
stop hyperprocess имя_гиперпроцесса; // запрещает прерывание
stop hyperprocess; // запрещает прерывание, в котором находимся
atomic утверждение // критическая секция
{список_утверждений} // compound statement – составное утверждение
if(выражение) утверждение
if(выражение) утверждение else утверждение
for(выражение;выражение;выражение) утверждение
for(объявление переменной;выражение;выражение) утверждение
return;
return выражение;
```

Важно отметить, что утверждение `set state` не прерывает исполнение текущего состояния – переход осуществится при следующей активации процесса.

Утверждение `atomic` задает критическую секцию - участок кода, исполняющийся атомарно. На время исполнения критической секции запрещаются все прерывания.

Пример :

```
if(a > b){
c++; // выражение;
a = b + c; // выражение;
unsigned int temp;
atomic temp = some_shared_variable;
if(c == d)
```

```
        stop process Proc1; // остановка процесса
    else if(TIFR0 & OCF0A){
        atomic{
            for(int i=0;i<10;i++){
                shared_var = i;
            }
        }
        }
    }
    else
    to state SomeState; // переход в другое состояние
```

2.7. Тайм-ауты

Формат определения тайм-аутов в языке IndustrialC:

```
    timeout ( время_в_миллисекундах ) {
набор_утверждений }
```

Пример:

```
timeout(1000)
{
stop hyperprocess Blink;
OCR1A = 2000;
start hyperprocess Blink;
to state SPEED2;
}
```

Код в теле тайм-аута срабатывает, если процесс находится в текущем состоянии дольше заданного времени. Определение нескольких тайм-аутов

в одном состоянии запрещено. Тело тайм-аута не должно быть пустым ("`timeout(10){}`" вызовет ошибку трансляции).

Для отслеживания времени используется таймер 0. Изменение регистров этого таймера или переопределение его функции-обработчика может привести к неопределенному поведению системы.

2.8. Выражения

Поддерживается основная часть выражений языка С, **кроме**:

- работы с указателями
- sizeof
- a?b:c
- оператора ',’

К этим выражениям добавлены проверки активности процессов

имя_процесса active

имя_процесса passive

Пример:

```
process SomeProc {. . .}
process SomeOtherProc {
    state FS_START {
        if(SomeProc active && a!=b)
        stop process SomeProc;
    }
    . . .
}
```

2.9. Константы

Поддерживаются десятеричные, шестнадцатеричные и двоичные целочисленные константы, а также десятеричные с плавающей запятой, символьные и строковые константы в стиле языка Си.

Примеры:

`666` – десятеричная целочисленная константа

`0xffee79` – шестнадцатеричная целочисленная константа

`0b11010` – двоичная целочисленная константа

`0.05` – десятеричная дробная константа

2.10. Комментарии и директивы препроцессора

Грамматика языка не поддерживает директивы препроцессора (кроме line marker'ов), но перед трансляцией код на industrialC обрабатывается препроцессором avr-gcc или avr-g++. При соблюдении этого условия в коде можно использовать, соответственно, комментарии и директивы, поддерживаемые этими препроцессорами.

2.11. Вставки на Си

В языке предусмотрена возможность вставлять код на языке Си двумя способами:

- Построчно, между $ и \n
- Внутри строки между двумя ограничителями $$ … $$

Внутри кода на Си можно использовать переменные, объявленные средствами industrialC. Для этого перед переменной ставиться знак $.

Необходимость такой конструкции обусловлена тем, что в сгенерированном Си - коде переменные из industrialC будут называться иначе, чем в изначальной программе – к их именам добавятся префиксы областей видимости (например, процесса или состояния).

Не рекомендуется использовать вставки на Си в тех случаях, где можно обойтись синтаксисом industrialC.

Пример:

```
int a, b, c;
float d;
$int a, blah;
$int d = $a + $b - $c;
```

В результирующем коде это будет выглядеть примерно так:

```
int program_0_a, program_0_b, program_0_c;
float program_0_d;
int a, blah;
int d = program_0_a + program_0_b - program_0_c;
```

Конструкция $$ … $$ предназначена для использования в выражениях, например:

```
if(a == b || $$ 0==strcmp(str, "blah") $$)
{
 …
}
```

Во всех остальных случаях следует использовать строчные вставки.

2.12. Запуск и исполнение программы

В языке предопределен фоновый гиперпроцесс background. Входящие в него процессы исполняются в основном цикле программы. Изначально во всей программе активен только процесс, описанный первым, и принадлежащий к гиперпроцессу background. Все остальные процессы изначально неактивны.

При запуске процесс оказывается в своем начальном состоянии, которое всегда называется FS_START. Такое состояние необходимо объявлять для каждого процесса.

3. СРЕДА РАЗРАБОТКИ

В качестве среды разработки используется Notepad++ с двумя дополнениями: файлы UserDefineLang.xml – подсветка синтаксиса и IndustrialC.dll – плагин для сборки и загрузки программ.

Установка:

- Notepad++ portable можно скачать с официального сайта
- в корень папки Notepad++ поместить файл UserDefineLang.xml
- в папке plugins разместить файл IndustrialC.dll
- добавить путь к файлу industrialc.exe в PATH
- установить пакет WinAVR, путь к его директории bin добавить в PATH
- после открытия .ic-файла, если подсветка синтаксиса отсутствует, нужно выбрать язык Language->industrialC
- при необходимости установить драйверы для программатора и FT232

Плагин имеет три команды: Build, Upload и Settings. Команда Build выполняет все шаги по трансляции и сборке программы в .hex – файл. Команда Upload собирает и загружает код в контроллер. Команда Settings выдает диалоговое окно с возможностью задания модели контроллера, способа прошивки и номера COM – порта. Плагин также реализует окно для вывода исполняемых команд и ошибок.

Команды плагина выполняются над тем файлом, который в данный момент выбран. При выполнении команд Build И Upload сохраняются все открытые в Notepad++ файлы.

Лист регистрации изменений									
Изм	Номера листов (страниц)				Всего листов (страниц) в докум.	N докумен-та	Входящий № сопроводи-тельного документа и дата	Подп.	Дата
	Измененных	Замененных	Новых	Аннулиро-ванных					
