18. Системные соглашения UltraLogik32 для контроллеров на базе процессоров семейства 80x86

18.1 Введение

Система программирования **UltraLogik 2.0** может использовать произвольные модели памяти и порождать исполняемый код для любого типа процессора. Но следует учитывать, что эту работу выполняет драйвер, указанный в разделе [**Компилятор**] у элемента проекта **Контроллер**.

В комплекте поставки могут быть следующие драйверы:

1) Emulator
2) 80x86-compatible simple compiler
3) 80x86-compatible compiler with optimizations

Emulator служит для отладки программы на компьютере, где ведется программирование без использования дополнительного оборудования.

Остальные компиляторы служат для создания исполняемого кода процессоров типа 8086/8088/8087, 80186/80188/80187, 80286/80287, 80386/i387, Intel486, Pentium и выше, а также их модификаций и совместимых с ними.

18.2 Модели памяти

В отличие от **Ultralogik 1.0**, новая версия не имеет **опции "Far code generation**". Переход к модели памяти с несколькими сегментами кода делается автоматически при достижении программы некоторого размера. При этом пользователь может не учитывать этого факта, даже если он производит вызовы внешних подпрограмм из проекта **Ultralogik** и наоборот, вызывает программы **Ultralogik** из внешних подпрограмм.

Не зависимо от количества сегментов кода, программа содержит только один сегмент данных (группа сегментов **DGROUP**). Размер этого сегмента не может превышать 64 килобайта.

Не зависимо от модели памяти, внешние подпрограммы можно вызывать вызовами типа NEAR или FAR. Причем одновременно можно использовать вызовы разного типа, но тип вызова должен соответствовать типу подпрограммы.

Независимо от модели памяти, программы **Ultralogik** вызываются всегда вызовом типа NEAR, а вызывающий код всегда должен находиться в сегменте с именем "**_TEXT**"

Если конечный файл имеет расширение **EXE**, то **Ultralogik** создает исполняемый файл не используя внешнего сборщика(линкера). При этом можно использовать внешние объектные файлы и библиотеки, за исключением объектного файла **c0*.obj**(стартовый код СИ/СИ++) и системных библиотек СИ/СИ++.

Если конечный файл имеет расширение **OBJ**, то **Ultralogik** создает только объектный файл. Для получения исполняемого файла требуется внешний сборщик(ликер). При сборке стандартным линкером первым объектным файлом требуется указать стартовый код СИ/СИ++(файл **c0*.obj**). Модель памяти СИ/СИ++ может быть любой кроме **TINY**, но если используются вызовы с передачей параметров, то все параметры от **Ultralogik** передаются по ссылке типа NEAR, а соглашения по использованию стека применяются такие же как в язы-

ке Паскаль. Со стороны **Ultralogik** потребуется только объектная библиотека **PLCRTL.LIB**, которая находится в директории **<Ultralogik_ROOT>\LIB**

18.3 Используемые сегменты и группы сегментов

В созданном системой объектном файле содержатся следующие сегменты и группы сегментов:

```
_TEXT   segment byte use16 public 'CODE'
CONST   segment byte use16 public 'DATA'
DATA    segment byte use16 public 'DATA'
STACK   segment word use16 stack  'STACK'
DGROUP  group   CONST,DATA,STACK
```

Кроме этого, могут дополнительно содержаться сегменты с исполняемым кодом, который не уместился в сегменте _TEXT.

В сегменте _TEXT содержится исполняемый код программы и системной библиотеки. Все внешне подпрограммы, которые Вы намереваетесь вызывать вызовами типа NEAR, должны располагаться в этом сегменте. Аналогично и подпрограммы, которые будут вызывать программы **Ultralogik** должны располагаться в этом сегменте.

Группа сегментов DGROUP состоит из нескольких сегментов. Ниже приведено описание сегментов используемых в **Ultralogik**. Внешние модули могут содержать другие сегменты, относящиеся к этой группе.

Сегмент CONST содержит константы или другие данные, не подлежащие изменению. Если программа создается на ассемблере, то может быть получено предупреждение (warning), об использовании зарезервированного слова в качестве идентификатора. Это не является ошибкой в программе. В сегменте CONST так же помещается информация для сетевого драйвера о переменных проекта.

Сегмент DATA содержит переменные и некоторые другие данные (состояние программ, детекторов фронта и т.д.).

Сегмент STACK используется по прямому назначению. Обратите внимание, что он входит в группу DGROUP. По умолчанию размер стека равен 4 Кбайт, но если внешним подпрограммам требуется стек большего размера, то необходимый размер стека должен быть явно указан компоновщику (Link) либо переопределен в объектном модуле внешней процедуры.

18.4 Имена переменных

Все имена глобальных переменных, объявленных в проекте, сначала переводятся в верхний регистр, например, имя ’InpVal’ в ’INPVAL’. Потом к полученному слову прибавляется суффикс ‘_VAR’, например, INPVAL_VAR. К локальным переменным программ обращаться из внешних подпрограмм нельзя. Поэтому все переменные, которые Вам понадобятся во внешних подпрограммах, объявляйте как глобальные. Это замечание не относится к процедурам, которые вызываются как внешние функциональные блоки. В этом случае можно использовать локальные переменные, которые будут передаваться в Вашу процедуру как параметры по ссылке.

18.5 Типы переменных

Ultralogik поддерживает следующие типы переменных:

BOOLEAN - аналогичен одноименному типу в языках программирования Turbo Pascal и Delphi фирмы Borland. Занимает один байт памяти и имеет два значения - 0(False) и 1(True).

BYTE – 8-ми разрядное целое число без знака. Занимает один байт памяти и может принимать значения в диапазоне 0..+255.

SHORTINT - 8-ми разрядное целое число со знаком. Занимает один байт памяти и может принимать значения в диапазоне -128..+127.

INTEGER - 16-ти разрядное целое число со знаком. Занимает два байта памяти. Может принимать значения в диапазоне -32768..+32767.

WORD - 16-ти разрядное целое число без знака. Занимает два байта памяти и может принимать значения в диапазоне 0..+65535.

LONGINT - 32-х разрядное целое число со знаком. Занимает четыре байта памяти. Может принимать значения в диапазоне - 2147483648..+ 2147483647.

DWORD - 32-х разрядное целое число без знака. Занимает четыре байта памяти. Может принимать значения в диапазоне 0..+ 4294967295.

FLOAT - 32-х разрядное число с плавающей точкой одинарной точности. Занимает 4 байта, имеет 6 значащих цифр. Может принимать значения в диапазоне ±1.18E-38..±3.40E+38.

TIMER - таймерный тип переменной. В Паскале и Си/Си++ аналогов не имеется. Занимает 4 байта. Может принимать значение 0..248d13h13m56s47. Старший бит указывает, была ли выполнена команда TSTART для данной переменной, остальные биты содержат значение переменной в сотых долях секунды(10 мс).

18.6 Используемые регистры процессора

Когда пользовательские программы получают управление, регистры процессора установлены следующим образом: DS = DGROUP, SS = DGROUP, DF = 0 (флаг направления сброшен). Перед возвратом из подпрограммы требуется восстановить в этих регистрах исходные значения. Кроме этих регистров также должны быть сохранены значения регистров SP и BP. Все остальные регистры процессора сохранять не требуется.

18.7 Статус программ

Как описывалось в предыдущих главах, каждая программа может исполняться в цикле контроллера. Информацию об этом можно получить (и при необходимости изменить) через специальные переменные. Имя переменной образуется следующим образом: имя программы преобразуется в верхний регистр, к получившемуся слову добавляется суффикс '_STATUS', например, PR1_STATUS. Под статус программы отводится один байт, который может принимать следующие значения: 0 (Stop) - программа не выполняется в цикле контроллера, 1 (Start) - программа выполняется в цикле, все остальные значения зарезервированы для расширения системы.

18.8 Вызов программ из внешних модулей

Внешние модули могут вызывать для исполнения программы проекта **UltraLogik**. Вы должны использовать ближний (near) вызов. Управление надо

передавать на метку, имя которой образуется следующим образом: имя программы преобразовывается в верхний регистр, к получившемуся слову добавляется суффикс '_PROG', например PR1_PROG. Перед вызовом программы убедитесь, что регистры DS, SS, CS и DF установлены как описано в п.18.6.

18.9 Сохраняемые переменные

Все переменные, имеющие признак **Stored**, при компиляции помещаются рядом, образуя область памяти, состоящую только из переменных с атрибутом **Stored**. Начало этой области памяти в группе DGROUP определяет метка с именем 'StoredMem', а конец помечен меткой 'StoredEnd'. Вы можете использовать свои программы сохранения/восстановления этой области памяти, но следует придерживаться приведенного ниже алгоритма работы и рекомендаций.

а) Поддержка переменных типа **Stored** требует двух программ, которые сохраняют и восстанавливают область памяти между метками **StoredMem** и **StoredEnd**. Программу восстановления обычно требуется вызвать один раз при запуске контроллера. Программа сохранения вызывается во время работы постоянно.

б) При восстановлении сохраненных значений переменных следует помнить, что на момент запуска программы эта область данных содержит значения переменных по умолчанию, поэтому вначале требуется проверить целостность сохраненных данных, а уже потом восстанавливать содержимое памяти, иначе переменным присвоятся случайные значения, что только усугубит ситуацию (система может потерять работоспособность).

в) Программу сохранения можно вызывать в каждом цикле контроллера, но это может занять у процессора значительное время, поэтому можно прийти к компромиссу, вызывая программу один раз за несколько циклов или один раз за определенное время.

г) Для сохранения данных лучше использовать статическую память (SRAM) с батарейным питанием. Электрически перепрограммируемая память (FLASH) для этой цели подходит меньше, так как она требует значительно большего времени на запись данных и имеет ограничение на количество перезаписей. Для большинства микросхем производители гарантируют 10000 циклов записи. Если сохранение переменных делать раз в секунду, то этот лимит будет исчерпан менее чем за три часа. На современных контроллерах применяются микросхемы памяти с блочной структурой, у которых можно независимо переписывать каждый блок до 100000 раз. Но можно предположить, что запись будет вестись всегда в один и тот же блок, а в таком случае лимит будет исчерпан за двое суток. Попытка записи в разные блоки может увеличить этот срок в 2-16 раз (в зависимости от емкости памяти и везения), что все равно представляется неудовлетворительным. Сейчас существуют микросхемы с числом перезаписей до одного миллиона на блок при использовании специальных алгоритмов, но при этом появляются ограничения на диапазон рабочих температур и общее количество перезаписей, которое допускает одна микросхема. Поэтому рекомендуется проконсультироваться у производителя оборудования, если Вы захотите использовать Flash-память. Но если сохраняемые переменные меняют свое значение относительно редко, то при добавлении в алгоритм сохранения проверки на изменение значения переменных, можно использовать Flash-память не зависимо от допустимого количества перезаписей.

д) Процесс сохранения данных, как показывает опыт, занимает продолжительное время, и часто наблюдаются случаи, когда сбои по питанию приходятся на момент выполнения процедуры сохранения данных. Поэтому рекомендуется иметь несколько копий(областей данных), которые обновляются по очереди или одновременно, в зависимости от требований к системе.

18.10 Обработка прерываний

При запуске система исполнения берет на себя следующие прерывания:

1) Системный таймер (программируется на 1ссек=0.01сек=10мсек). Исходный обработчик прерывания вызывается два раза за 110 мсек (сначала через 60, а потом через 50 мсек), то есть системное время остается корректным.
2) Деление на ноль.
3) Задействованных в системе исполнения последовательных портов.

Во время работы система исполнения не запрещает никаких прерываний, т.е. пользовательские программы, включенные в систему, могут использовать любые прерывания при условии, что:

1) Обработчик прерывания от системного таймера системы исполнения должен вызываться каждые 10 мсек.
2) По завершению работы системы исполнения необходимо восстановить старые обработчики прерываний. Вызов этих процедур можно поместить в программу STOP.

18.11 Передача параметров во внешние функциональные блоки

Все параметры (входные и выходные) передаются по ссылке. На каждый параметр передается только его смещение от начала DGROUP. Первым в стек заносится адрес первого входа и далее все входа по порядку, потом адрес первого выхода (если есть) и далее все выхода по порядку. Потом делается вызов процедуры с помощью calln или callf - в зависимости от установленной галочки. Параметры из стека должна удалить вызываемая процедура, например, командой “ret XXXX”, где XXXX = <количество параметров> * 2.