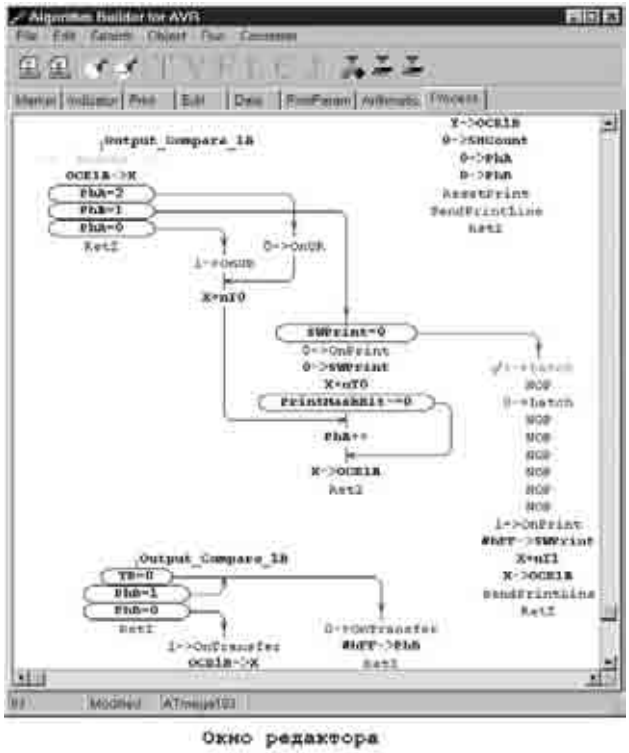
Графические технологии разработки программного обеспечения для микроконтроллеров

Программное обеспечение вдыхает жизнь в систему, основанную на микроконтроллере или микропроцессоре. Разработка его является центральным моментом общего процесса проектирования. Центр тяжести функциональных свойств таких систем ложится именно на программное обеспечение

Геннадий Громов

algrom@tula.net

Основным инструментом для профессиональной разработки программ является ассемблер, предполагающий детализацию на уровне команд микроконтроллера или микропроцессора. Только ассемблер позволяет максимально использовать ресурсы кристалла. Но очень удобной работу в ассемблере еще никто не называл. Существенным барьером между программистом и предметом программирования является текстовый редактор с его ограниченными возможностями.



Окно редактора

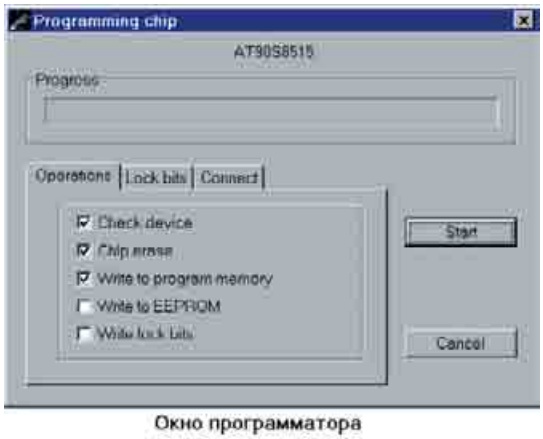
В данной публикации я хотел поговорить о технологии графического проектирования программного обеспечения для микроконтроллеров и микропроцессоров. Такая технология освобождает программиста от целого ряда неудобств, свойственных классическому ассемблеру. Использование ее предполагает существенное упрощение работы при возможности сохранения уровня детализации ассемблера. Примером среды, в которой реализованы эти технологии, является пакет Algorithm Builder, в данный момент адаптированный под микроконтроллеры фирмы ATMEL с архитектурой AVR.

Вначале немного о мнемонике.

Мнемоника представляет собой короткий набор букв латинского алфавита, однозначно определяющий соответствующую ему операцию в ядре микропроцессора. Конечно, по возможности разработчики старались вложить в этот набор букв смысл выполняемой операции, чтобы максимально облегчить жизнь программисту. И в ряде случаев им это вполне удалось, в таких как, например, MOV, CALL, ADD, JMP и т. д. Но таких ситуаций отнюдь не большинство. Затем появляются такие, как SBI, XTHL, BRBC, ADIW, XCHD и пр., которые начинают соперничать с китайской письменностью. Разумеется, все это можно запомнить и в конечном счете привыкнуть к такой форме, но все равно ничего хорошего в этом нет. Иных проблем достаточно.

Кроме того, те, кто имел дело с разными микропроцессорами, тот, конечно, обратил внимание, что общность мнемоник их ассемблеров составляет не более трети от общего количества. Среди тех, которые присутствуют практически во всех ассемблерах, можно перечислить MOV, ADD, SUB, CALL, AND, OR, INC, DEC, PUSH, POP и еще некоторые. Основная же масса представлений уникальна, даже для ряда совершенно одинаковых операций. Например, в одном случае операция «исключающее ИЛИ» записывается как XOR, в других — EOR. В одном случае обозначение операций переходов построено на основе слова JUMP, в дру-

гом — на основе слова BRANCH. И таких примеров можно привести множество. Но ведь редко кому удается пользоваться всегда только одним ассемблером. По жизни «коней» приходится менять, и с таким нежелательным разнообразием неизбежно приходится сталкиваться.



Окно программатора

С учетом всего этого в среде Algorithm Builder форма представления операций микроконтроллера построена иначе, в более удобочитаемом виде. Здесь использован визуально-функциональный принцип. Запись операции содержит образ выполняемого действия.

Например:

вместо «`MOV R0,R1`» записывается «`R1->R0`»,

вместо «`LDI R16,63`» — «`63->R16`»,

вместо «`ST X,R2`» — «`R2->[X]`»,

вместо «`LSR r7`» — «`r7>>`»,

вместо «`SBI PortB,3`» — «`1->PortB.3`» и т. д.

В результате время освоения системы команд сокращается до минимума, а смысл выполняемой операции становится понятным даже неподготовленному человеку.

Но не о мнемонике я хотел поговорить в этой статье. Эти изменения в форму записи операторов были внесены как бы заодно. В конце концов, если очень надо, то запомнить ее не так уж и сложно.

Одно из основных неудобств ассемблера (и не только его) состоит в том, что в редакторе программа записывается и отображается в виде одной сплошной вертикальной колонки. И логическая структура тех или иных фрагментов может быть построена только в воображении программиста. В особо сложных случаях приходится прибегать к бумаге, предварительно прорисовывая необходимую структуру. Это дополнительно загружает и без того, как правило, перегруженный мозг программиста совершенно ненужной работой.

Главное предназначение графических сред — максимальное приведение интерфейса разработки в соответствие с природой человеческого восприятия, освобождение пользователя для чисто творческого процесса. Элементы этих технологий мы наблюдаем в таких языках программирования, как Visual Basic, C-Builder, Delphi и др. Но там этот процесс ограничивается в основном конструированием содержимого окна. А с полным правом к этой категории можно, пожалуй, отнести такие среды, как PCAD, OrCAD и др. Можно, конечно, электрическую схему описать в текстовом редакторе, вводя список соединений, но бесконечно удобнее это делать в специализированном графическом редакторе. Но эти пакеты из другой области.

Любая программа, написанная на уровне ассемблера, состоит из ряда законченных монотонных (или условно монотонных) цепочек, в которых ее исполнение возможно без ветвлений. Такие фрагменты всегда начинаются с метки (если это не само начало программы), а завершаются либо оператором безусловного перехода, либо оператором возврата из подпрограммы (RET или RETI), то есть оператором, который безо всяких условий перемещает ход исполнения программы в другое место. Такие участки программы представляют собой первичные логически завершенные блоки.

Например, приведенная ниже подпрограмма содержит три таких блока.

```
SubName   LDI   XL,96
          LDI   XH,00
Label0:   LD    R16,X
          CPI   R16,1
          BREQ  Label1
          CPI   R16,2
          BRNE  Label2
          SBIW  X,2
          RJMP  Label0
Label1:   SBI   PortA,0
          CBI   PortA,1
          RET
Label2:   SBI   PortA,2
          CBI   PortA,3
          RET
```

Первый блок начинается с оператора LDI XL,96 с меткой SubName, а завершается оператором безусловного перехода RJMP, два других — начинаются с оператора SBI с метками Label1 и Label2 соответственно, а заканчиваются оператором RET.

Визуальное разделение таких блоков на плоскости является одним из принципов графических методов. Другим принципом является возможность графического отображения ветвления в виде вектора, со стрелкой на конце, до необходимой точки в программе. Благодаря этому обеспечивается возможность ввода программы на плоскости, в двух измерениях, в виде алгоритма с древовидной структурой, с визуальным отображением паутины условных и безусловных переходов. В результате — вся логическая структура как на ладони.

На рис. 1 приведено отображение вышеприведенной программы в графической среде Algorithm Builder.

Поскольку основная масса условных и безусловных переходов вводится и отображаются графически, то программа освобождается от бесчисленных имен меток, которые в ассемблере являются неизбежным балластом, загромождающим текст программы. Необходимость имен для меток остается только для входов в подпрограммы.

Графическая технология ассемблера в среде Algorithm Builder реализуется посредством нескольких базовых объектов, из которых выстраивается конструкция алгоритма. Среди них:

- Объект Label (метка) — отображается в виде вертикального штриха, расположенного на оси блока операторов. Метка может иметь необязательное имя, которое располагается слева или справа от штри-

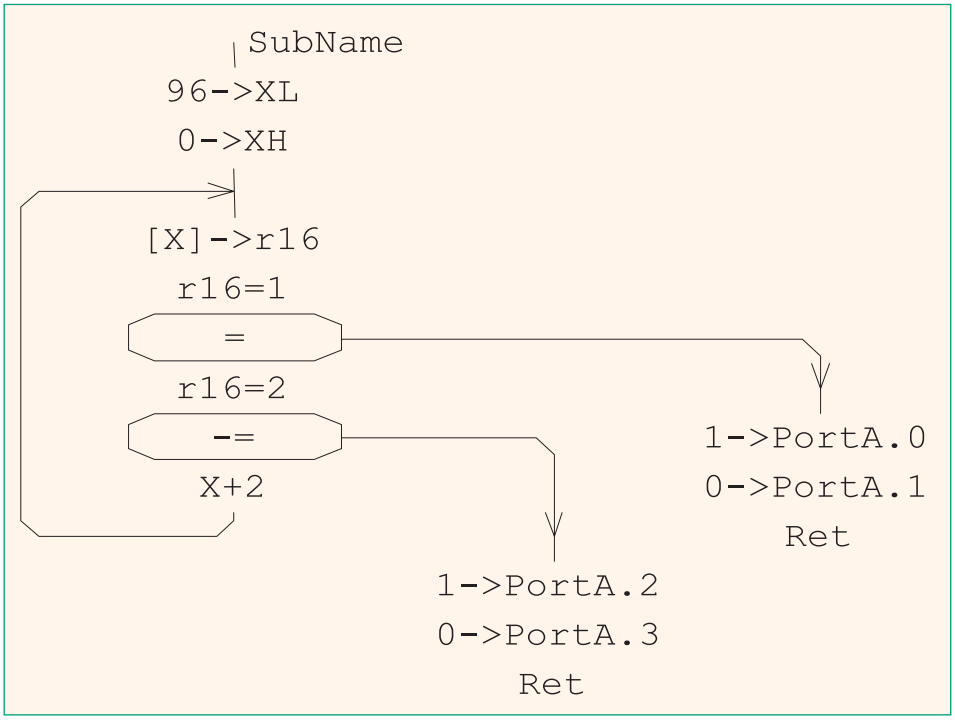


Рис. 1

```
Send:
LDI YL,$FF      Label3:
LDI YH,$01      SBI PortB,1
Label0:         RJMP Label5
RCALL Exchange  Label4:
CPI YH,0        SBI PortB,2
BRNE Label1     RJMP Label5
CPI YL,$2D
BREQ Label2     Exchange:
CPI YL,$2A      LDI XH,8
BREQ Label2     Label10:
CPI YL,$41      SBI PortB,0
BREQ Label2     SBIC PinB,1
CPI YL,$1D      RJMP Label10
BREQ Label2     SBRC XL,7
Label1:         SBI PortD,1
ST Y,XL         CLC
Label2:         SBIC PinD,0
SBIW Y,1        SEC
BRCC Label0     ROL XL
CPI XL,1        Label11:
BREQ Label3     CBI PortB,0
CPI XL,2        SBIS PinB,1
BREQ Label4     RJMP Label11
SBI PortB,0     SBI PortB,0
CPI XL,3        DEC XH
BRE4Q Label3    BRNE Label10
Label5:         RET
LDI XL,$2B
RET
```

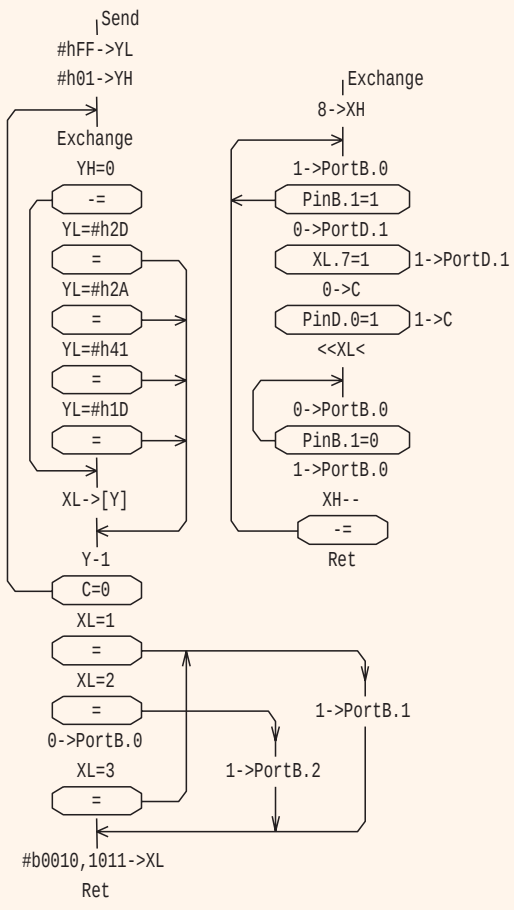


Рис. 2

ха. Метки предназначены для подвода концов векторов переходов.

- Объект Vertex (вершина) используется в качестве начала блока, а по своему отображению и назначению аналогичен метке.
- Объект Field (поле) предназначен для записи операторов алгоритма и представляет собой отцентрированную строку в блоке.
- Объект Condition (условие) предназначен для обозначения операторов условных переходов.

Конструктивно наиболее сложный.

Графически представляет собой контур, располагающийся посередине блока, внутри которого вписывается текст с условием перехода, и возможный вектор перехода в виде ломаной линии со стрелкой на конце, исходящей от одного из краев контура, который должн заканчиваться на метке или вершине. Действие интерпретируется как ветвление, если вписанное условие выполняется.

- Объект JMP Vector предназначен для представления безусловного перехода. Графически представляет собой ломаную линию, исходящую из середины блока операторов, аналогичную вектору объекта Condition.

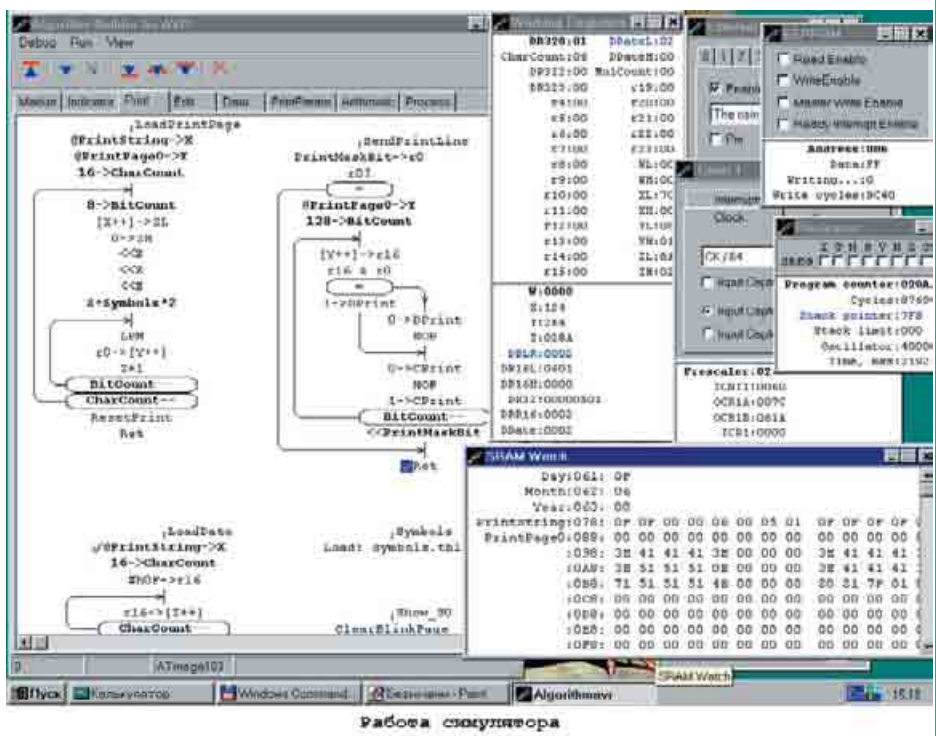
Редактор среды позволяет свободно вносить на рабочее поле любые необходимые объекты, модифицировать их, перемещать, вставлять и т. д. Принципы редактирования аналогичны с прочими графическими редакторами.

Для удобства допускается конец вектора ветвления останавливать на отрезке другого вектора, конец которого определен, если конечный адрес у них общий. Но при большом нежелании рисовать вектор ветвления, если это ветвление слишком длинное, всегда остается возможность адресовать переход классическим способом, на имя метки.

В целом данная среда программирования является самодостаточной системой. Она содержит в себе компилятор алгоритма, симулятор микроконтроллера и внутрисхемный программатор, обеспечивающий загрузку откомпилированного алгоритма в кристалл. Кроме того, обеспечивается режим мониторной отладки, при котором к откомпилированному коду программы добавляется скрытый код, обеспечивающий вывод всего внутреннего состояния реального микроконтроллера в заданных точках останова в соответствующие окна, как при работе в симуляторе. При работе мониторного отладчика и программатора микроконтроллер непосредственно подключается к параллельному порту LPT несколькими цепями.

Но возможности среды не ограничены программированием на уровне элементарных команд микроконтроллера. Компилятором поддерживается система макрооператоров, поглощающих в себя наиболее распространенные комбинации команд. Программирование с использованием макрооператоров предполагает операции с многобайтными величинами со знаком, что приближает данную среду к категории языка высокого уровня. В алгоритме, например, возможна запись таких операторов:

«`-12500->Y`»
«`#h025F->SP`»
«`#h2E40->Svar`»
«`Svar+Y`», где `Svar` — объявленная двухбайтная ячейка оперативной памяти данных (SRAM). Очевидно, нет необходимости расшифровывать смысл выполняемых действий. А в качестве условия перехода записать, например:

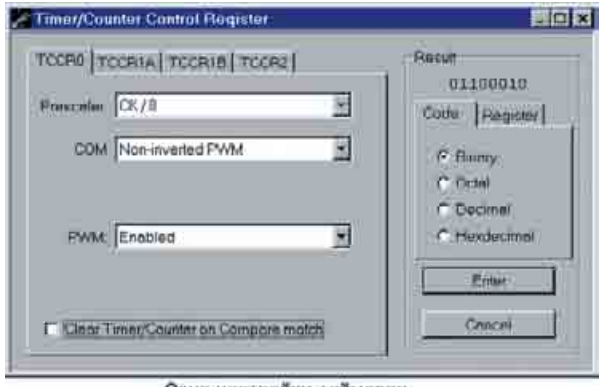


Работа симулятора

«X<2049».

Кроме того, поддерживаются метки со стандартным именем прерывания. Встретив такую метку, компилятор автоматически вставит в соответствующее вектору прерывания место программы код необходимого безусловного перехода на нее. Имя таких меток можно выбрать через соответствующий пункт меню. Реализован также удобный интерфейс настройки таймеров, который избавляет от необходимости помнить, какой бит управляющего регистра что означает и ряд других сервисных возможностей.

Среда предназначена для работы в операционной системе Windows 95/98.



Окно настройки таймеров

На рис. 2 приведен более сложный пример фрагмента программы. Слева — в классическом ассемблере, а справа — полная его копия в Algorithm Builder.

Конечно, в рамках одной статьи невозможно достаточно подробно изложить обо всех особенностях работы в графической среде.

Графические технологии проектирования раскрывают новые возможности для разработчиков. Визуальность логической структуры программы уменьшает вероятность ошибок и сокращает сроки разработки. Появляется такое понятие, как дизайн алгоритма, предполагающее определенный эстетический вкус программиста.

Психологически переход к использованию такой среды требует определенного усилия. В голове многих проскользнет мысль: «может ассемблер и не располагает этими возможностями, но зато я к нему очень привык, мне и с ним хорошо». Это чем-то сродни переходу от командной DOS к графической оболочке Windows. Однако освоение этого инструмента и последующая работа с ним заметно проще классического ассемблера. Во всяком случае, те, кто его уже использует, назад пути не ищет.

Подробнее познакомиться с Algorithm Builder вы можете посетив сайт **ht tp://w ww . atmel. ru** в разделе «Software».