Parallax Propeller.
Заметки начинающего пропеллиста.

igorkov / fsp@igorkov.org / 2009.10.25

Среди множества представленных на рынке микроконтроллеров продукт компании Parallax под названием Propeller сильно отличается от всех остальных. Почти все микроконтроллеры содержат одно вычислительное ядро какой-либо архитектуры и разрядности, функциональную систему прерываний, память программ, память данных и некий набор периферийных устройств. Propeller нарушает почти все эти аксиомы на рынке микроконтроллеров.

Первое и основное отличие Propeller-а – это его многоядерность. В свое распоряжение мы получаем 8 абсолютно симметричных 32 битных ядер! Вероятнее всего благодаря этому Propeller является лидером попроизводительности среди небольших микроконтроллеров (ARM9/11/A8 не в счет, так как это все-таки иная весовая категория), она составляет целых 160MIPS (8 ядер, работающих на тактовой частоте 80МГц и дающих по 20MIPS каждое). Причем это на 32 битных операциях (а не 8ми битных, как в том же AVR). Обидно, конечно, что ядро Parallax Propeller четырехтактное, а не 1-тактное, как, к примеру, в ARM. В последнем случае теоретическая производительность вообще была бы заоблачной для чипа такого класса: 640MIPS, мечты-мечты...

Следующая особенность: очень гибкая система таймеров. В чипе по 2 счетчика-таймера на ядро (!) и каждый таймер имеет свой PLL (!!!), позволяющий работать последнему на частотах до 128МГц. Кроме обычных режимов счета, генерации ШИМ и управления пинами, у таймеров есть довольно специфичные режимы, например, генерация TV сигнала.

Интересно и построение памяти. В пропеллере 2 вида памяти: это индивидуальная память ядра – 2кБ (имеется у каждого ядра) и общая память на 64кБ. Последняя разделена на 2 одинаковых участка: ПЗУ и ОЗУ. ПЗУ не FLASH (если верить даташитам), а масочное, то есть его содержимое фиксировано и меняться не может. В нем содержится интерпретатор языка Spin (о нем далее) и вспомогательные таблицы, такие как: таблица знакогенератора, таблицы логарифмов и синусов. Последние, упрощают вычисления и позволяют реализовать относительно быстрые операции умножения, деления, возведения в степень и вычисления корня. Сама программа пользователя располагается в ОЗУ контроллера. Ее загрузка поддерживается как по последовательному интерфейсу, так и с внешней микросхемы EEPROM с интерфейсом I²C.

Благодаря таким возможностям Parallax Propeller в некоторых случаях может быть применим в задачах, где казалось бы без ПЛИС не обойтись, или там, где обычно требуется DSP-процессор. Если смотреть конструкции на нем, иногда он применяется в областях, где цифровая техника кажется неприменимой. К примеру, логический анализатор Propanalyser[1] или AM-приемник[2] БЕЗ внешних компонентов?

При изучении возможностей ядра в тупик может поставить отсутсвие описания системы прерываний, ведь последней просто НЕТ! Это кажется в начале поразительным, но все становится на места, когда смотришь на систему команд ассемблера и помнишь, что ядро не одно, а целых 8. Особенность системы команд, упрощающая жизнь без прерываний – это наличие инструкций типа: ожидание событий таймера, ожидание изменения на пине. Дополнительное преимущество такого

[1] http://forums.parallax.com/forums/default.aspx?f=25&m=331847

[2] http://forums.parallax.com/forums/default.aspx?f=25&m=285351

подхода: нет потери времени на переключение контекста, нет проблем при пересекании событий, получается, что построить систему реального времени на Propeller намного проще, чем на любом другом микроконтроллере.

Вообще, имея несколько вычислительных ядер принципы программирования под такой микроконтроллер меняются. Можно не заводить отдельное прерывание по таймеру для опроса внешней клавиатуры и обновления данных на экране, а выделить отдельное ядро и заниматься в нем только одним, элементарным действием. В некоторых случаях отпадает надобность в операционной системе: множество ее функций уже реализовано «в железе».

Сам по себе Parallax Propeller имеет довольно компактное описание, всего 37 страниц. Оно включает описание почти всех возможностей, ассемблера и даже краткий обзор Spin-а. Другое дело, что по данному описанию непонятным остается множество моментов.

Что такое Spin, еще сказано не было. У нашего вентилятора, очень специфичная архитектура. Понятное дело, что программировать можно на ассемблере для этой архитектуры, однако, компилятор языка Си, хоть и возможен, но все-таки не очень эффективен (а точнее совсем не эффективен). Инженеры из Parallax не стали мучаться и писать тормознутый Си-компилятор (если будет Си, все-равно начнут на нем писать все подряд и в итоге проклянут за медлительность, а что еще хуже откажутся от использования самого контроллера, так и не разобравшись), поэтому был создан специальный язык под названием Spin, по своей кривизне, напоминающий бейсик (может не кривой, но уж точно довольно противоречивый).

Язык Spin компилируется не в ассемблерный код, а в байт код, который исполняется в виртуальной машине. Технология похожа на Java и всякие # (C#, J#, ...). Преимущество: язык относительно высокоуровневый, недостаток: опять-таки скорость работы. Виртуальная машина для исполнения Spin-кода встроена в ПЗУ микроконтроллера, то есть при его использовании мы ничего не теряем. Мало того, в применении Spin пошли еще дальше: исполнение пользовательского кода начинается именно в Spin-режиме и только потом при желании мы можем переключиться в нативный, ассемблерный режим. А будет ли желание, зависит от нас и от задачи. Все-таки возможностей данного языка достаточно, кроме тех случаев, когда требуется по-настоящему высокое быстродействие и оптимизация. Маньяки же могут всегда ставить маленькую заглушку и сразу переходить в режим ассемблера.

Кратко с архитектурой пропеллера познакомились. Теперь можно поговорить о граблях. Их все-таки довольно много, как и во всем новаторском и задающем новые концепции, отличные от всего существующего:

1. Малое количество памяти для программы на ассемблере: всего 2к для программы и данных, выполняющихся на ядре. В данный объем должна влезть программа и все требуетмые переменные.
2. Отсутствие аппаратного стека и инструкций, позволяющих легко организовать стек. Вызов функций поддерживается системой команд, однако, нельзя осуществлять рекурсивные вызовы и довольно хитро ядро оперирует с адресом возврата.
3. Как было сказано: отсутствие прерываний. Сомнительные грабли, просто заставляют изменить принципы построения приложения.
4. Отсутствие периферийных устройств, кроме портов ввода-вывода и таймеров с функцией ШИМ и генерацией видеосигнала. Всю периферию приходится реализовывать программно. Однако, библиотеки, поставляемые со средой разработки, имеют довольно много готовых периферийных устройств, таких как

ADC, I2C, SPI и даже интерфейсы к некоторым устройствам, таким как LCD экраны и различные драйверы.

5. Отсутствие операций умножения и деления в железе. В мануале есть упоминание об умножении, но инструкция помечена как «Future». Возможно, в новых ревизиях чипа она будет добавлена, но в настроящий момент, видимо, не функционирует.
6. Отсутствие встроенной памяти. Программа хранится в ОЗУ, после цикла питания она пропадает и требуется повторная загрузка. Однако пропеллер умеет грузиться с внешней EEPROM по I²C. Но в любом случае, из этого следует, что микропрограмма не может быть скрыта от посторонних глаз.

Теперь перейдем к практической части. Parallax Propeller удобен для быстрого старта. Официальная среда для разработки называется Propeller Tool. Не такая навороченная, как тот же AVR Studio, но достаточная для работы. Для запуска первых программ достаточно иметь только сам чип и последовательный порт (COM/UART/RS-232) с TTL уровнями на выходе. В описании пропеллера есть простая схема преобразователя с парочкой транзисторов, но по-моему мнению лучше и надежнее использовать преобразователь с MAX232 или конверторы USB-UART.

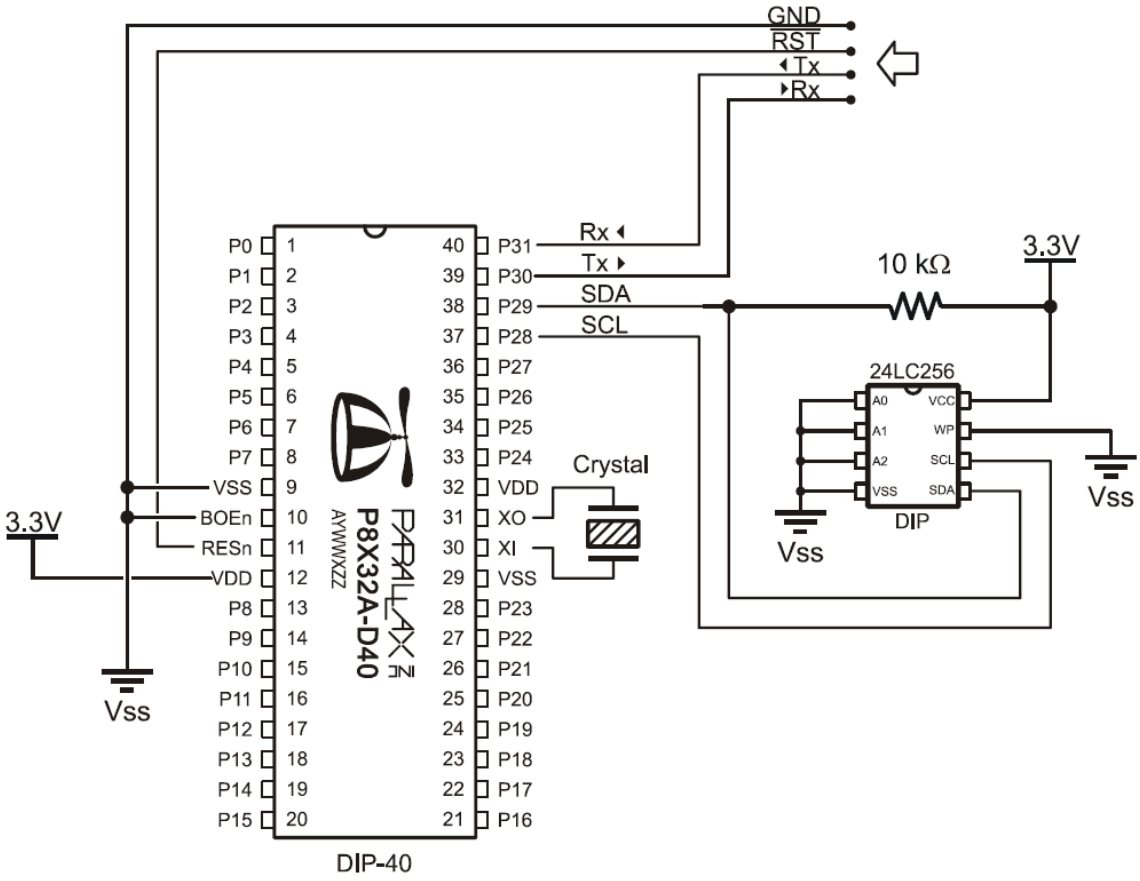

Рис. 1

Полная схема подключения представлена на рис.1. На ней схематично показано подключение контроллера Propeller в корпусе DIP-40. Кварц лучше использовать на 5МГц. С ним удобнее всего получать максимальную частоту тактирочания ядра 80МГц, однако подойдет любой на частоту 4-20МГц, но Propeller имеет 2 внутренних генератора, поэтому внешний кварц на первых порах тоже не обязателен. На схеме так же фигурирует внешняя EEPROM для хранения конфигурации, однако для первых экспериментов можно обойтись и без нее.

Для программирования требуется 3 линии: прием, передача и сброс. Первые 2 – это RXD, TXD, а 3я может быть как DTR, так и RST в ком-порте. (можно настроить в среде Propeller Tool). Я для удобства стал использовать переходник USB-COM. В начале взял относительно самодельное, работающее с стандартным драйвером USBSerial.sys. У меня оно отлично работает со всеми терминальными программами и используется для прошивки ARM-ов. Propeller Tool однако, работать с ним отказался, выдавая ошибку: «Cann't write to COM port». Пришлось откапывать переходник на FT232. Он заработал как надо.

Основные операции в среде:

F8	Собрать приложение и посмотреть информацию.
F10	Загрузить приложение в ОЗУ.
F11	Загрузить приложение во внешнее EEPROM.

Единственный существенный недостаток среды: полное непонимание кириллицы. Даже комментарии не написать... Чтож, будем привыкать комментировать на англицком.

Все готово для того, чтобы попробовать написать для пропеллера микроконтроллерный «Hello World» (мигание светодиодом). Итак, первый пример:

```
01| PUB main                        'Точка входа в приложение.
02|  dira[16]~~              'Настраиваем PIN16 на вывод.
03|  repeat                  'Задаем бесконечный цикл (аналог while (1))
04|    !outa[16]             'Инвертируем состояние пина 16.
05|    waitcnt(5_000 + cnt)    'Ожидаем 5000 циклов процессора.
```

Страшненько, не правда ли? Кострукция «XX|» в начале каждой строки не относится к синтаксису spin, а добавлена мною для удобства чтения листинга.

Частота переключения пина в примере более 1КГц, светодиод не подцепить, можно либо увеличить задержку до 5_000_000, либо подключать осциллограф, чтобы понаблюдать сигнал.

Разберем по строчкам:

01. Объявление функции с именем main, является точкой входа. PUB говорит о том, что имя будет видно из прочих модулей. Проект на Spin в среде разработки Propeller представлен деревом модулей, каждый из которых выполняет определенную функцию. И есть один модуль, являющийся головным, в котором присутствует точка входа. Для того, чтобы иметь возможность вызывать функции между модулями используется данная конструкция. Так же допустимо задавать PRI, в данном случае функция является внутренняя для модуля (аналог static в Си или приватного метода класса в C++).

02. Настраиваем порт на выход, выбирается конкретная линия в порте DIRA.

03. Бесконечный цикл. Все что идет ниже с отступами попадает в данный цикл.

04. Инвертирование состояния PIN16.

05. Ожидание совпадения счетчика с числом (cnt + 5000), где cnt – текущее значение счетчика. Частота счета около 12МГц, получаемая задержка около 0.4мс. Прочерки в числе являются декоративными и разделяют порядки числа. Довольно удобно, сразу видно – миллион или 10 миллионов.

Важно замечание по коду: имена переменных и все прочее не чувствительно к регистру, так что можно писать как dira (в примере), так и DIRA и даже DiRa, правда для себя лучше с самого начала определить правила написания (к примеру, маленькими буквами), так код будет лучше читаться.

Теперь второй пример, немного более интересный: попытки генерации более сложного сигнала. Для экспериментов припасена сервомашинка Pilotage C-02CT (аналог Hitec HS-55), которая для управления требует некую посылку импульсов определенной длительности, задача близкая к генерации ШИМ-а. Для ее решения воспользуемся распределением по ядрам: одно ядро будет опрашивать кнопку, которой можно будет менять позицию, другое – генерировать сигнал для сервомашинки.

```
01| CON
02|   _clkmode = xtal1 + pll16x
03|   _xinfreq = 5_000_000
04|
05| VAR
06|   long servo_width            'shared variable
07|   long stack[10]
08|
09| PUB main
10|   servo_width := 15           'pulse def width - 1.5ms (center)
11|   cognew(servogen, @stack[0])
12|   repeat
13|     waitpne(%01, %01, 0)      'wait PIN0 low (push button)
14|     servo_width++             '+0.1ms
15|     if servo_width == 22
16|       servo_width := 9        'back
17|     waitcnt(clkfreq/2 + cnt)  'wait 500ms
18|
19| PUB servogen
20|   dira[16]~~
21|   repeat
17|     outa[16] := 1  'gen pulse
18|     waitcnt(clkfreq/10_000*servo_width + cnt)
19|     outa[16] := 0  'wait for next pulse
20|     waitcnt(clkfreq/50 + cnt)
```

По сравнению с первым примером здесь стоит особенно выделить первичную инициализацию в секции CON. Настраивается источник тактирования (выбирается внешний кварц) и устанавливается множитель PLL: 16, итого, имея 5Мгц кварц, получаем частоту работы – 80МГц.

Конечно, пример можно несколько оптимизировать и улучшить (например, операцию умножения вынести в основной поток, чтобы она не влияла на генерацию импульсов, да и частота следования импульсов получается не 20мс, а 20мс плюс длительность импульса управления), но разберем по строкам:

01. Секция с константами. Здесь же можно определить значения каких-либо констант или макросов, макросы – это могут быть настройки проекта и железа Propeller-а.

02-03. Собственно константы. Настраивается режим тактирования и задается частота внешнего кварца, которая используется при вычислении задержек в программе.

05. Глобальные переменные. Видны всем ядрам, так как расположены в общей памяти. Стек требуется для инициализации потока на другом ядре, данный стек использует поток и копия виртуальной машины Spin. Как вычислять размер стека пока не понимаю.

09. Точка входа в приложение.

10. Устанавливаем начальное положение сервопривода, центральное, длительность импульса 1.5мс.

11. Запускаем функций servogen на новом ядре. Начиная с этой строчки параллельно функции main начинает работать функция servogen.

В данном примере работает 2 потока и оба потока осуществляют доступ к порту ввода-вывода. Есть и еще один общий ресурс: это глобальная переменная servo_width.

Каждое ядро имеет свою копию регистров управления портом: DIRA, INA, OUTA. Именно поэтому инициализация пина вывода PIN16 осуществляется в процедуре servogen (если инициализировать в main-е, то ничего работать не будет). Как же тогда осуществляется доступ к портам? В пропеллере действует следующие 3 правила:

1. По умолчанию все пины являются входами (т.е. находятся в 3ем состоянии).
2. Если хотя бы одно ядро настроило пин как выход, он становится выходом (логическое или).
3. Если несколько ядер управляют одним пином, настроенным как выход, низкий уровень на пине будет только в том случае, когда все ядра установили на нем низкий уровень (логическое и).

Для разграничения доступа к общим переменным предусмотрены 8 аппаратных семафоров (скорее мьютексов). Одно ядро может занять семафор. После этого второе это сделать уже не сможет. Для работы с семаформами существуют специальные инструкции ассемблера и железо гарантирует, что не будет одновременно занят один и тот же семафор. В данном примере семафоры не требуются, так как один поток записывает данные, а другой читает их и нет опасных ситуаций, когда из-за одновременного доступа значение переменной будет испорчено.

Здесь был предложен Spin-вариант программы. Можно продемонстрировать на данном примере использование ассемблера, переписав на него функцию генерации сигнала сервомашинке (и добавив еще пару оптимизаций предыдущего кода):

```
01| CON
02|   _clkmode = xtal1 + pll16x
03|   _xinfreq = 5_000_000
04|
05| VAR
06|   long servo_width           'pulse def width - 1.5ms (center)
07|   long servo_width_ticks     '
08|
09| PUB main
10|   servo_width       := 15                      '
11|   calc_width(servo_width)                      '
12|
13|   cognew(@servogen, @servo_width_ticks)        '
14|
15|   repeat
16|     waitpne(%01, %01, 0)                       'wait PIN0 low (push
button)
17|     servo_width++                              '+0.1ms
18|     if servo_width == 22                       'max 2.1ms
19|       servo_width := 9                         'back
20|     calc_width(servo_width)                    'calc width ticks
21|     waitcnt(clkfreq/2 + cnt)                   'wait 500ms
22|
23| PRI calc_width(w)
24|   servo_width_ticks := clkfreq/10_000 * w       'calc width
25|
26| DAT
27|               org
28| servogen
29|               mov     DIRA, diraval_1        'pin 16 out
30|               mov     time_loop, CNT         'main loop parameter
31|               add     time_loop, period_20ms '
32| :loop
33|               mov     OUTA, diraval_1
34|               rdlong  width, PAR 'get pulse width from shared memory
35|               mov     time_pulse, CNT        '
36|               add     time_pulse, width      'calculation wait value
37|               waitcnt time_pulse, width NR   '
```

```
38|                mov     OUTA, diraval_0         '
39|                waitcnt time_loop, period_20ms 'wait until next period
40|                jmp     #:loop
41|
42| time_loop      long      0
43| time_pulse     long      0
44| width          long      0
45| diraval_1      long      |<16
46| diraval_0      long      |<0
47| period_20ms    long      (80_000_000/50)
```

Конечно, в данном случае особого преимущества от ассемблера нет, но понятны основные принципы программирования на нем. Первый принцип: ассеблерный код объявляется в секции DAT (данных) Spin-кода, оно понятно: для него ассемблерная вставка является не более чем данными.

Операция cognew() так же задействует новое ядро, но в данном случае функция получает на вход не указатель на процедуру Spin, которую требуется выполнить, а указатель на данные с ассемблерным кодом. Дальнейшие действия можно предсказать, секция с данными и ассемблерной процедурой загружается на свободное ядро и происходит запуск.

Как можно заметить, в примере была несколько изменена логика работы: вычисление длительности импульса в тиках было вынесено в основной цикл, чтобы не производить рассчет с уможением на каждой итерации. Расчет расположен в отдельной функции (строка 23 листинга).

Разберем ассемблерный код:

26. Объявляется, что сейчас идет секция с данными. Именно в нее кладется ассемблерный листинг. Согласен, криво.

27. Обычно этой директивой задается смещение при компоновке кода, здесь же судя по всему оно означает начало ассемблерного листинга.

28. Метка с именем функции.

29. Настраиваем порт на выход. Так как требуется 32х битное значение, а команда MOV может кодировать внутри себя только 9 бит значения, приходится использовать перемещение отдельной константы, объявленной ниже на строке 45.

30. Заносим в переменную цикла текущее значение счетчика.

31. И высчитываем его значение через 20мс.

32. Объявление метки. Начало цикла.

33. Порт в 1.

34. Загружаем из общей памяти значение длины импульса.

35-36. Считаем, какое значение должно быть у счетчика в момент окончания импульса.

37. Инструкция ожидаения. Находимся в ней пока не будет совпадения time_pulse с CNT.

38. Сбрасываем пин в 0.

39. Опять ожидание. Теперь ожидание окончания 20мс окна.

40. Загрузить приложение во внешнее EEPROM.

42-47. Объявление локальных переменных, используемых в ассемблерной программе.

Для тех кто не особо понял, что за сигнал генерится, поясню подробнее: для управления сервомашинкой требуются импульсы определенной длительности, повторяющиеся через равные промежутки времени. Для Hitec HS-55 (аналога, тестовой Pilotage C-02CT) длительность этих импульсов должна быть в пределах 0.9-2.1мс. Длительность импульса задает угол сервомашинки. Рекомендованная по документации частота следования импульсов должна быть 20мс.

Вид сигнала, который генерирует последняя программа сразу после старта показан на рис. 2. Соответственно замыкая кнопку на PIN0 длительность импульса увеличивается до 2.1мс и затем возвращается к исходному значению 0.9мс. Сервомашинка при этом совершает поворот с одного крайнего положения до другого (на самом деле положения не совсем крайние, но близки к ним).

Рис. 2 – Управляющие импульсы для сервомашинки

Для первого раза это все. Продолжения не обещаю, но эксперименты в любом случае будут продолжаться.

Ссылки

1. Актуальная версия данного документа – igorkov.org/pdf/propeller1.pdf
2. Файлы листингов – igorkov.org/lst/propeller1_1.spin, igorkov.org/lst/propeller1_2.spin, igorkov.org/lst/propeller1_2n.spin, igorkov.org/lst/propeller1_3.spin.
3. Parallax Home Page – www.parallax.com
4. Полный мануал на русском языке – www.parallax.com/dl/docs/prod/prop/PM-v1.0-RUS-v1.0.pdf