Разработка DSP системы

5

Обзор

Для разработки собственных программ можно воспользоваться программным обеспечением разработчика поставляемым с EZ-KIT Lite. При отсутствии опыта в разработке программ для систем, основанных на DSP, нужно следовать приведенным в этой главе действиям. Представленные шаги разработки служат руководящими указаниями для создания собственных программ. Принимается во внимание, что процесс разработки изменяется в зависимости от стиля конкретного разработчика. Приведенные действия следует рассматривать как отправную точку, и поэтому не стесняйтесь, если потребуется изменить их согласно собственному стилю работы.

Шаг 1: Системные требования

Первый шаг при создании DSP системы состоит в том, чтобы определить какие возможности системы потребуются. Эти возможности будут зависеть от типов применяемых алгоритмов, типов используемых сигналов и типов входных/выходных приборов, которые необходимо подключить к DSP процессору. Расчет размера требуемой памяти данных и памяти программ основан на количестве данных, требуемых системам ввода/вывода, и процессов обработки, которые должны быть выполнены. Предполагаемый размер программ, созданных в соответствии с используемыми алгоритмами, определяет требования к памяти программ.

Например, при создании системы обработки речевого сигнала выбирается подходящий алгоритм для сжатия речевого сигнала, допустим, LPC. Этот алгоритм требует сохранения некоторого количества данных и хранения некоторого числа программных инструкций. Конечно, пока этот алгоритм не выполняется, эти требования могут быть только приблизительными. Например, установим, что требуется 4K слов памяти данных и 1K слов памяти программ. Может потребоваться А/Ц или Ц/А преобразование. В этом случае к последовательному порту DSP подключается звуковой кодек. Необходимо, чтобы DSP процессор мог поддерживать скорость выборок данных. Частота 33MHz ADSP-2181 для этого более чем достаточна.

Шаг 2: Проектирование системы

После того, когда системные требования определены, необходимо спроектировать оборудование системы. EZ-KIT Lite использует для А/Ц и Ц/А преобразований звуковой кодек AD1847. AD1847 подключен к последовательному порту 0 (SPORT0). Используется только внутренняя память ADSP-2181 (16K слов памяти программ и 16K слов памяти данных), поэтому внешняя память не подключена. Для связи через интерфейс RS-232 используются сигналы последовательного порта 1 (SPORT1).

©АВТЭКС Санкт-Петербург. http://www.autex.spb.ru, E-mail: info@autex.spb.ru

Шаг 3: Файл описания архитектуры

При использовании средств разработки для процессоров семейства ADSP-2100, оборудование системы нужно перечислить в файле описания архитектуры. Поскольку система ADSP-2181 уже определена, этот шаг уже сделан. Описание содержится в файле `ADSP2181.ACH` (файл созданный системным конфигуратором), который включен в программное обеспечение для EZ-KIT Lite.

```
.system demo;
.adsp2181;
.mmap0;

.seg/pm/ram/abs=0/code/data          int_pm_lo[8192];
.seg/pm/ram/abs=8192/code/data       int_pm_hi[8192];
.seg/dm/ram/abs=0/data               int_dm_lo[8192];
.seg/dm/ram/abs=8192/data            int_dm_hi[8160];

.endsys;
```

Первые три строчки файла определяют имя системы, использующей ADSP-2181 с возможностью начальной загрузки из EPROM (на вывод MMAP подается низкий уровень).

16K слов внутренней памяти программ описаны как два сегмента по 8K слов, которые могут содержать как коды, так и данные. Первая секция 8K слов будет всегда внутри ADSP-2181. Вторая секция может быть запрограммирована с помощью инструкций ADSP-2181 (установка режимов `PMOVLAY`) как внутренний или внешний оверлей. Архитектура с двумя секциями упрощает управление памятью для приложений, которые могут использовать дополнительную внешнюю память. Память может быть описана и как единая 16K секция.

Память данных описана подобно памяти программ. 32 слова резервируют для регистров управления отображенных в памяти ADSP-2181 .

Этот текстовый файл (`.SYS`) используется как входные данные системного конфигуратора для создания `.ACH` файла. Файл `.ACH` используется редактором связей и программой моделирования. Файл `ADSP2181.ACH` поставляется с программным обеспечением для ADSP-2181 EZ-KIT Lite. Для определения отличной архитектуры системы, потребуется обратиться к программе системного конфигуратора.

Шаг 4: Разработка программного кода

Когда оборудование определено (или приближено к требованиям), можно начинать разработку программного кода. Во-первых, определите требования к размеру памяти, необходимого для хранения переменных и массивов и все используемые прерывания. Предусмотрите любое оборудование или регистры, которые нуждаются в инициализации.

Написание программы осуществляется вводом инструкций в текстовый файл, который впоследствии обрабатывается программой ассемблера. Ассемблер транслирует набор алгебраических инструкций процессора в бинарный объектный файл. Существует несколько вещей, которые следует учесть, в частности, о языке ассемблера ADSP-2100/

Алгебраический синтаксис использует символ «=» для представления переноса данных:

```
AX0 = MX0;
```

В приведенном примере данные из регистра `MX0` переносятся в регистр `AX0`. Символы «+», «-», «*» используются для обозначения арифметических операций. Например, инструкция

```
AR = AX0 + AY0;
```

добавляет содержимое регистра `AX0` к содержимому регистра `AY0` и размещает результат (сумму) в регистре `AR`.

Все инструкции должны заканчиваться символом «;». Многофункциональные инструкции отделяются символом «,». Следующий пример показывает многофункциональные инструкции процессоров ADSP-21xx.

```
MR = MR + MX1 * MY(SU), MX1 = DM(I0,M3), MY1 = PM(I4,M5);
```

Первая инструкция (до первой запятой) это операция умножения/накопления (MAC). Содержимое входных регистров `MX1` и `MY1` перемножается и добавляется к содержимому регистра результата умножителя. Вторая инструкция загружает регистр `MX1` из памяти данных (DM), а третья загружает регистр `MY1` из памяти программ (PM). Регистры I используются для хранения адресов, а регистры M хранят величину, используемую для модификации адреса. Все эти инструкции могут быть выполнены за один процессорный цикл.

Объявление любых переменных, массивов или констант, также как и указание включаемых файлов и портов ввода/вывода, делается в начале программы, используя директивы ассемблера.

После перезапуска системы, ADSP-2181 начнет выполнение кода из памяти программ со стартового адреса 0. Из приведенного ниже листинга видно, что там расположена инструкция `jump start`. Она используется для обхода таблицы векторов прерываний.

При прерывании исполнение программы останавливается и переходит к адресу, указанному в таблице векторов прерываний. Кодовый сегмент, показанный ниже, включает таблицу векторов прерываний и размещается в первых 48 адресах памяти программ.

Для удобства в поле комментария показаны шестнадцатеричные адреса памяти. Необходимо заполнить неиспользованные адреса в памяти таблицы векторов инструкцией `rti`. Это делается в целях безопасности. Инструкция `rti` обеспечит перенос исполнения обратно в основную программу.

Следующий пример показывает процесс инициализации аппаратной части EZ-KIT Lite, чтение значений с аналогового входа и их вывод.

```
. module/RAM/ABS=0   my_program;

(**** Объявление констант ****)
(управляющие регистры ADSP-2181 отображенные в памяти)
.const IDMA =                       0x3fe0;
.const BDMA_BIAD =                  0x3fe1;
.const BDMA_BEAD =                  0x3fe2;
.const BDMA_BDMA_Ctrl =             0x3fe3;
.const BDMA_BWCOUNT =               0x3fe4;
.const PFDATA =                     0x3fe5;
.const PFTYPE =                     0x3fe6;
.const SPORT1_Autobuf =             0x3fef;
.const SPORT1_RFSDIV =              0x3ff0;
.const SPORT1_SCLKDIV =             0x3ff1;
.const SPORT1_Control_Reg =         0x3ff2;
.const SPORT0_Autobuf =             0x3ff3;
.const SPORT0_RESDIV =              0x3ff4;
.const SPORT0_SCLKDIV =             0x3ff5;
.const SPORT0_Control_Reg =         0x3ff6;
.const SPORT0_TX_Channels0 =        0x3ff7;
.const SPORT0_TX_Channels1 =        0x3ff8;
.const SPORT0_RX_Channels0 =        0x3ff9;
.const SPORT0_RX_Channels1 =        0x3ffa;
.const TSCALE =                     0x3ffh;
.const TCOUNT =                     0x3ffc;
.const TRERIOD =                    0x3ffd;
.const DM_Wait_Reg =                0x3ffe;
.const System_Control_Reg =         0x3fff;

(**** Объявления переменных и буферов ****)
.var/dm/ram/circ  rx_buf[3];        /* приемный буфер AD1847 */
.var/dm/ram/circ  tx_buf[3];        /* передающий буфер AD1847 */
.var/dm/ram/circ  init_cmds[13];
.var/dm           stat_flag;

(**** Инициализация переменных и буферов ****)
.init  tx_buf:  0xc000, 0x0000, 0x0000;
.init  init_cmds:
      0xc003,     (регистр управления левого входа AD1847)
      0xc103,     (регистр управления правого входа AD1847)
      0xc288,     (регистр управления левого 1 внешнего входа AD1847)
      0xc388,     (регистр управления левого 1 внешнего входа AD1847)
      0xc488,     (регистр управления левого 2 внешнего входа AD1847)
      0xc588,     (регистр управления левого 2 внешнего входа AD1847)
      0xc680,     (регистр управления левого ЦАП AD1847)
      0xc780,     (регистр управления правого ЦАП AD1847)
      0xc85b,     (регистр формата данных AD1847)
      0xc909,     (регистр конфигурации интерфейса AD1847)
      0xca00,     (регистр управления выводом AD1847)
      0xcc40,     (регистр различной информации AD1847)
      0xcd00;     (регистр управления цифрового миксера AD1847)

(**** Таблица векторов прерываний ****)
      jump start;             (Адрес 0000: перезапуск)
       rti;
       rti;
       rti;

       rti;                   (Адрес 0004: IRQ2)
       rti;
       rti;
       rti;
```

```
       rti;                    (Адрес 0008: IRQL1)
       rti;
       rti;
       rti;

       rti;                    (Адрес 000C: IRQL0)
       rti;
       rti;
       rti;

      ar = dm(stat_flag);      {Адрес 0010: SPORT0 tx}
      ar = pass ar;
      if eq rti;
      jump next_cmd;

      jump input_samples;      {Адрес 0014: SPORT0 rx}
       rti;
       rti;
       rti;

       rti;                    (Адрес 0018: IRQE)
       rti;
       rti;
       rti;

       rti;                    (Адрес 001C: BDMA)
       rti;
       rti;
       rti;

       rti;                    (Адрес 0020: SPORT1 tx или IRQ1)
       rti;
       rti;
       rti;

       rti;                    (Адрес 0024: SPORT1 rx или IRQ0)
       rti;
       rti;
       rti;

       rti;                    (Адрес 0028: таймер)
       rti;
       rti;
       rti;

       rti;                    (Адрес 002C: power down)
       rti;
       rti;
       rti;

{***** Инициализация ADSP-2181 ****}
start:
      i0 = ^rx_buf;            {установка указателя на начало буфера}
      lo = %rx_buf;            {установка длины буфера}
      i1 = ^tx_buf;
      l1 = %tx_buf;
      i3 = ^init_cmds;
      l3 = %init_cmds;
      m1 = 1;
```

```
{**** Последовательный порт 0 (SPORT0) ****}
      ax0 = b#0000001010000111;
      dm(SPORT0_Autobuf) = ax0;

      ax0 = 0;
      dm(SPORT0_RFSDIV ) = ax0;
      dm(SPORT0_SCLKDIV) = ax0;
      ax0 = b#1000011000001111;
      dm(SPORT0_Control_Reg) = ax0;

      ax0 = b#0000000000000111;
      dm(SPORT0_TX_Channels0) = ax0;

      ax0 = b#0000000000000111;
      dm(SPORT0_TX_Channels1) = ax0;

      ax0 = b#0000000000000111;
      dm(SPORT0_RX_Channels0) = ax0;

      ax0 = b#0000000000000111;
      dm(SPORT0_RX_Channels1) = ax0;

{**** Последовательный порт 1 (SPORT1) ****}
      ax0 = 0;
      dm(SPORT1_Autobuf) = ax0;     {автобуферирование недоступно}
      dm(SPORT1_RFSDIV ) = ax0;     {RFSDIV не используется}
      dm(SPORT1_SCLKDIV) = ax0;     {SCLKDIV не используется}
      dm SPORT1_Control_Reg ) = ax0;{функции ctrl не установлены}

{**** Установка таймера ****}
      ax0 = 0;
      dm(TSCALE) = ax0;             {таймер не будет использоваться}
      dm(TCOUNT) = ax0;
      dm(TPERIOD) = ax0;

{**** Установка системы и памяти ****}
      ax0 = b#0000000000000000;
      dm(DM_Wait_Reg) = ax0;
      ax0 = b#0001000000000000;     {доступен SPORT0 }
      dm(System_Control_Reg) = ax0;

      ifc = b#0000001111111111;     {очистка очереди прерываний}
      nop;

      icntl = b#00000;
      mstat = b#1000000;

{**** Инициализация кодека AD1847 ****}
      ax0 = 1;
      dm(stat_flag) = ax0;          {очистка флага}
      imask = b#0001000000;         {доступны прерывания}

      ax0 = dm(i1, m1);
      tx0 = ax0;

check_init:
      ax0 = dm(stat_flag);          {ожидание до целой выборки}
      af = pass ax0;                {буфер переслать к кодеку}
      if ne jump check_init;

      ay0 = 2;
```

```
check_acih:
      ax0 = dm(rx_buf);             {ожидание инициализации пока}
      ar  = ax0 and ay0;            { кодек выйдет из автокалибровки}
      if eq jump check_acih;

check_acil:
      ax0 = dm(rx_buf);             {ожидание инициализации пока}
      ar  = ax0 and ay0;            {кодек выйдет из автокалибровки}
      if ne jump check_acil;
      idle;

      ay0 = 0xbf3f;                 {включение левого DAC}
      ax0 = dm(init_cmds + 6);
      ar  = ax0 AND ay0;
      dm(tx_buf) = ar;
      idle;
      ax0 = dm(init_cmds + 7);      {включение правого DAC}
      ar  = ax0 AND ay0 ;
      dm(tx_buf) = ar;
      idle;
      ax0 = 0xc901;                 {очистка требования автокалибровки}
      dm(tx_buf) = ax0;
      idle;
      ax1 = 0x8000; {управляющее слово очистки}
      dm(tx_buf) = ax1;

      ifc = b#0000001111111111;     {очистка очереди прерываний}
      nop;

      imask = b#0000100000;         {разрешение прерывания rx0}

{**** Ожидание прерывания ****}
talkthru: idle;
      jump talkthru;

{**** Подпрограммы обслуживания прерываний ****}
{**** прерывание по приему ****}
input_samples:
      ena sec_reg;                  {использование теневого банка регистров}
      ax1 = dm(rx_buf + 1);         {получение данных от кодека}
      mx = dm(rx_buf + 2);
      dm(tx_buf + 1) = ax1;         {пересылка данных в кодек}
      dm(tx_buf + 2) = mx1;
      rti;

{**** прерывание по передаче для инициализации кодека  ****}
next_cmd:
      ena sec_reg;
      ax0 = dm(i3, m1 );            {получение следующего управляющего слова и}
      dm(tx_buf) = ax0;             { размещение в передающий слот 0}
      ax0 = i3;
      ay0 = ^init_cmds;
      ar = ax0 - ay0;
      if gt rti;                    {rti если больше управляющих слов}
      ax0 = 0x8000;                 {еще установить флаг и }
      dm(tx_buf) = ax0;             {убрать MCE если завершена}
      ax0 = 0;
      dm(stat_flag) = ax0;          {переустановить флаг состояния }
      rti;
.endmod;
```

Шаг 5: Запуск ассемблера

После завершения написания текстового файла, который содержит программу на языке ассемблера можно запустить ассемблер следующей командой:

```
asm21  my_prog  -2181
```

В этом примере ассемблируется текстовый файл `my_prog.dsp`. Ключ `-2181` указывает ассемблеру на необходимость присоединить специальные инструкции процессора ADSP-2181. Ассемблер создает соответствующий объектный файл или файлы.

Существует некоторое число дополнительных ключей ассемблера. Например, для создания файла листинга `my_prog.lst` используется следующий ключ:

```
asm21  my_prog  -2181  -l
```

Ассемблер создает объектный файл с расширением `.obj`.

Шаг 6: Запуск редактора связей

Редактор связей создает исполняемый файл из объектного модуля созданного ассемблером. Следующий пример создает исполняемый файл с именем `demo.exe`:

```
ld21  my_prog  -a adsp2181  -e demo
```

Файл описания архитектуры `adsp2181.ach` определяется ключом `-a`, а имя исполняемого файла выбирается ключом `-e`. Существует дополнительные ключи, которые позволяют создавать файл карты распределения памяти карты, файл с таблицей символов и др.

Шаг 7: Запуск программы моделирования

Программа моделирования позволяет запускать код в моделируемом окружении, чтобы проверить ваше программное обеспечение без использования реального оборудования системы. Шаг моделирования программы используют, чтобы удостовериться в работоспособности программы.

В большинстве случаев он позволяет избежать ситуации, когда вы загружаете ваше программное обеспечение в аппарат, и оно не работает. Если вы проверили работу вашего программного обеспечения на программе моделирования, но программа в аппарате не запускается, это может быть вызвано дефектом аппаратной части. Программа моделирования вызывается следующим образом:

```
sim2181  -a adsp2181  -e demo
```

Эта команда запускает программу моделирования ADSP-2181 для исполняемого файла `demo.exe` и системы, описанной в файле архитектуры `adsp2181.ach`. Если программа моделирования не может найти какой-либо из этих файлов, необходимо проверить установки путей в `autoexec.bat` или определить полный путь:

```
sim2181  -a c:\adi_dsp\21xx\lib\adsp2181  -e demo
```

Шаг 8: Программирование EPROM

После проверки, что программное обеспечение работает, можно разделить программу для программирования EPROM. Микросхема EPROM может быть вставлена в кроватку на плате для запуска вашей программы. Преобразователь кодов для программирования ППЗУ может быть вызван следующей командой:

```
spl21 demo demoprom -loader -2181
```

Эта команда создает из исполняемого файла `demo.exe` PROM файл с именем `demoprom.bnm`. По умолчанию формат этого файла является записью Motorola S. При указании ключа `-i` можно задать запись формата Intel Hex.

После включения питания или когда вы нажали кнопку перезапуска на плате, содержимое EPROM автоматически загрузится во внутреннюю память программ и память данных процессора и начнется выполнение команд.

Шаг 9: Запуск платы ADSP-2181 EZ_KIT Lite

Плата EZ-KIT Lite имеет микросхему EPROM, которая содержит программу ADSP-2181. Когда на плату подают питание (или ее перезагружают), коды автоматически передаются из EPROM во внутреннюю память ADSP-2181. Коды загружаются вместе с включением программы монитора, которая позволяет ADSP-2181 связаться с ПК по интерфейсу RS-232.

Коды выполняют самотестирование, а затем посылают звуковой сигнал на выход кодека. Вам потребуется установить громкоговоритель, подключенный к разъему звукового выхода, чтобы услышать производимый звук. Сигнал на этом разъеме низкого уровня, таким образом, необходимо установить динамики, которые имеют встроенные усилители.

После установки программного обеспечения, можно запустить хост программу EZ-KIT Lite. Вы сможете загрузить исполняемый файл (`.exe` файл созданный редактором связей).

Шаг 10: Отладка

Обычно, если ваш EZ-KIT Lite не работает должным образом с поставленными программами, проблема чаще всего заключается в подключении питания, или подключении RS-232, или с компонентами на плате. Убедитесь, что все соединения сделаны плотно, и напряжение питания лежит в пределах от 8 В до 10 В выпрямленного тока. Убедитесь также, что EPROM должным образом установлена в кроватке, и нет предметов оставленных на или под платой, вызывающих короткое замыкание цепи.

Если сомнения позади, нажмите кнопку перезапуска.