Advanced Embedded Microcontroller Seminar -- Day 3

Andrew Huang
bunnie@mit.edu
IAP 1999

Agenda

- Day 3: 1/22 -- everything else you need to know to do something with the SH-1
  - development environment and tools
  - code development procedure
  - example: digital I/O, light flasher
  - utilities provided
  - example: tachometer
  - example: DMA, ITU, TPC, and D/A converter
  - idea session, Q&A
  - prep for hands-on session next week

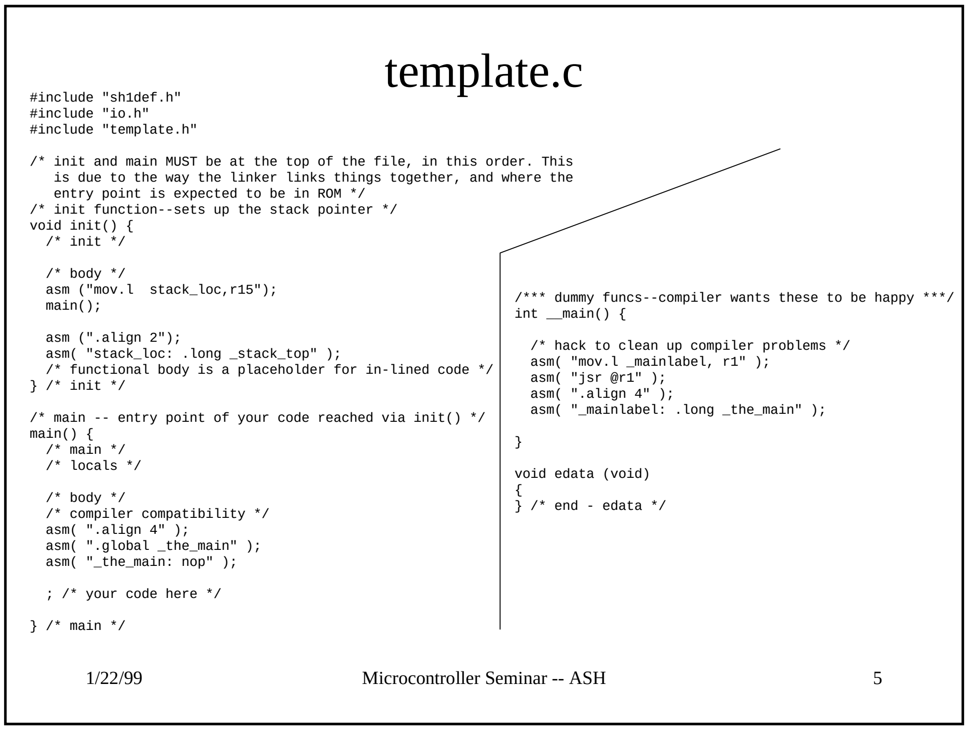
Development Procedure

- note that all examples in this presentation are for env. under linux; current indications show that env. under DOS will differ slightly, mostly in terms of constants and offsets

- required tools for development:
  - gcc, binutils (ld, ar, gas), plus gcc includes
  - can get via ftp from prep.ai.mit.edu and cross-compile
- common code directory
  - contains functions like mon_printf() which is version of printf() that prints to the serial port on the SH1WH
  - contains .h files such as
    - sh1def.h -- all your memory-mapped peripherals here
    - sram.h -- constants that delimit regions of SRAM
    - ascii.h -- a table of ASCII constants for convenience

Development Procedure

- Development starts with a template directory
  - makefile
    - contains commands for building target binary
  - template.c
    - contains your code
  - template.cmd
    - contains a command file describing to ld how to link everything together
  - readme
    - contains useage instructions
  - mktime.c
    - utility routine to generate timestamps (linux only)
- code you write runs at a very low level
  - no OS, no memory management, no nothing--just the machine



template.c

```
#include "sh1def.h"
#include "io.h"
#include "template.h"

/* init and main MUST be at the top of the file, in this order. This
   is due to the way the linker links things together, and where the
   entry point is expected to be in ROM */
/* init function--sets up the stack pointer */
void init() {
  /* init */

  /* body */
  asm ("mov.l  stack_loc,r15");
  main();

  asm (".align 2");
  asm( "stack_loc: .long _stack_top" );
  /* functional body is a placeholder for in-lined code */
} /* init */

/* main -- entry point of your code reached via init() */
main() {
  /* main */
  /* locals */

  /* body */
  /* compiler compatibility */
  asm( ".align 4" );
  asm( ".global _the_main" );
  asm( "_the_main: nop" );

  ; /* your code here */

} /* main */
```

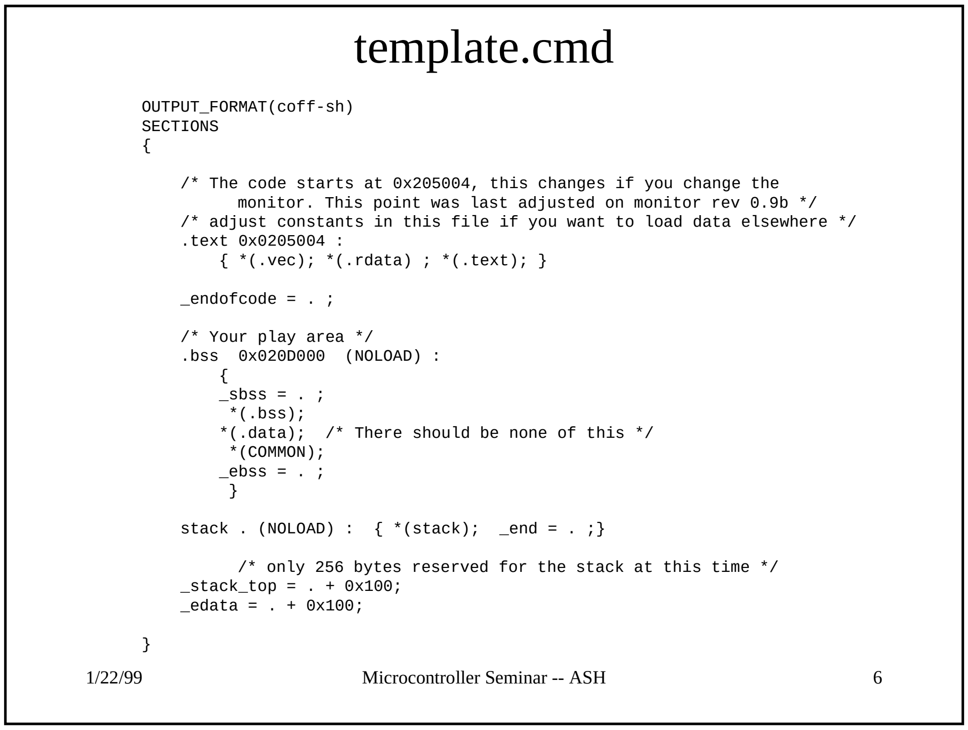
```
/*** dummy funcs--compiler wants these to be happy ***/
int __main() {

  /* hack to clean up compiler problems */
  asm( "mov.l _mainlabel, r1" );
  asm( "jsr @r1" );
  asm( ".align 4" );
  asm( "_mainlabel: .long _the_main" );

}

void edata (void)
{
} /* end - edata */
```

1/22/99 Microcontroller Seminar -- ASH 5



template.cmd

```
OUTPUT_FORMAT(coff-sh)
SECTIONS
{

    /* The code starts at 0x205004, this changes if you change the
          monitor. This point was last adjusted on monitor rev 0.9b */
    /* adjust constants in this file if you want to load data elsewhere */
    .text 0x0205004 :
        { *(.vec); *(.rdata) ; *(.text); }

    _endofcode = . ;

    /* Your play area */
    .bss  0x020D000  (NOLOAD) :
        {
        _sbss = . ;
         *(.bss);
        *(.data);  /* There should be none of this */
         *(COMMON);
        _ebss = . ;
         }

    stack . (NOLOAD) :  { *(stack);  _end = . ;}

          /* only 256 bytes reserved for the stack at this time */
    _stack_top = . + 0x100;
    _edata = . + 0x100;

}
```

1/22/99 Microcontroller Seminar -- ASH 6

.map file

- Compilation of program will yield a .map file
  - contains list of locations of all symbols
  - very useful for debugging, or determining if your code will fit in memory!

```
LOAD /home/bunnie/shtools/sh-coff/lib/crt0.o

.text           0x00201764      0x970
 *(.vec)
 *(.rdata)
 *(.text)
 *fill*         0x00201764        0xc
 .text          0x00201770       0x40 /home/bunnie/shtools/sh-coff/lib/crt0.o
                0x00201770               start
 .text          0x002017b0       0xd0 common.a(test.o)
                0x00201844               sleep
                0x002017b0               init
                0x00201820               dummy
                0x0020187c               __main
                0x00201834               somewhere
                0x00201830               edata
                0x002017cc               main
 .text          0x00201880       0x60 /home/bunnie/shtools/sh-coff/lib/libc.a(exit.o)
                0x00201880               exit
 .text          0x002018e0      0x110 /home/bunnie/shtools/sh-coff/lib/libc.a(reent.o)
```

1/22/99 Microcontroller Seminar -- ASH 7

light flashing code

- SH1WH has an LED tied to the “spare 1” pin on the SH-1 (pin 103, PB5)
- dummy() is needed because gcc -O2 will optimize out an empty for() loop sometimes
- compiled code will yield a file called test.bin (assuming .c file is called test.c)

```
#define PBIOR (*(volatile short int *)(0x5ffffc6))
#define PBCR1 (*(volatile short int *)(0x5ffffcc))
#define PBCR2 (*(volatile short int *)(0x5ffffce))
#define PBDR  (*(volatile short int *)(0x5ffffc2))
#define DELAY 20

main(){
  int i;
  int j = 0;

  PBIOR |= 0x0020;
  PBCR2 &= ~0x0C00;
  while (1){
    for (i = 0; i < 10; i++) {
      /* Toggle the light */
      PBDR ^= 0x0020;
      sleep (DELAY);
    } /* for */
  } /* while */
} /* main*/

sleep (int delay){
  int i;
  int j;
  for (j = 0; j < 1000; j++)
    for (i = 0; i < delay; i++)
      dummy();
}

void dummy() {}
```

1/22/99 Microcontroller Seminar -- ASH 8

Uploading and Running Your Code

- java utility called JTerm is used to upload code and interact with board
  - basic terminal program with upload ability
  - upload protocol is simple--just straight binary with some headers and footers wrapped around the code
- sample session:

```
[SH1WH mon Rev 1.0b bunnie - Fri Jan 22 00:37:51 EST 1999]
mon> lp

Ready to lead data at 2050004

command_kload(): 970 bytes successfully received.
mon> gp
```

Utilities

- A few utilities have been written for your convenience
  - RTC utility
    - set and get time of day
    - set and get arbitrary memory locations in the RTC
    - get current temperature in degrees C
  - FLASH utility
    - erase sectors
    - program data
    - erase boot block
    - program boot block
  - FPGA programming utility
    - upload bitfiles to breakout board FPGA
  - plus source code for monitor, in-class examples, etc.

Programming the Boot Block

- Procedure
  - make sure you have a cmon.bin (monitor binary image) that you have confidence in
  - upload to data memory using “ld” command
  - examine data to verify that its actually there
  - upload flash.bin FLASH utility using “lp” command
  - run “gp” to execute flash.bin
  - erase the bootblocks, erase until each sector is successfully erased
  - program the bootblock
  - quit FLASH utility or reset board, and if you did everything right--you should drop back into the monitor
- currently developing a routine that lets you modify interrupt vectors automagically