EECS 373-F10
Homework #1
Sep 28, 2010
Due: Oct 7, 2010 at the beginning of class.

Problem 1: Assembly (40 pts). On a machine with the ARM GNU tools installed, type the following into a file named `main.c`:

```
#include <inttypes.h>

volatile int32_t a = 5, b = 6;

int32_t add(int32_t x, int32_t y) {
  return x + y;
}

main() {
  uint32_t c;
  c = add(a, b);
  return c;
}
```

Type the following into a file named `Makefile` (note: all but the first line begins with a TAB character):

```
all:
      arm-none-eabi-gcc -mcpu=cortex-m3 -mthumb main.c \
        -T generic-hosted.ld -o main.out -g
      arm-none-eabi-objdump -S main.out > main.list
```

(a) (10 pts) Explain what each tool and command line option that is used in the `Makefile` actually does. After entering this into the `Makefile`, type `make` at the command line.

(b) (30 pts) Find the assembly code for the `add()` function in the `main.list` file. Annotate each line of disassembled listing with a detailed description of what the assembly code actually does.

(c) **Bonus (10 pts).** Find, download, and install the `qemu-arm` tool (on a non-Linux machine, this may requires installing a virtual machine). Show us `qemu` running this on your machine for credit. Modify `main.c` to add the following lines just before "`#include <inttypes.h>`" and just before the "`return c`" statement in `main()`, respectively:

```
#include <stdio.h>
...
  printf("add(%d, %d) = %d\n", a, b, c);
```

Finally, recompile the code (type `make`), run the code in the simulator, and verify the output:

```
$ qemu-arm -cpu cortex-m3 ./main.out
add(5, 6) = 11
```

Problem 2: Memory Interfacing (30 pts). Imagine that you have a SmartFusion system configured with four asynchronous SRAMs wired up in the following configuration. Assume that the hardware is configured to assert the chip select lines LOW for memory access in the regions show in the table below.

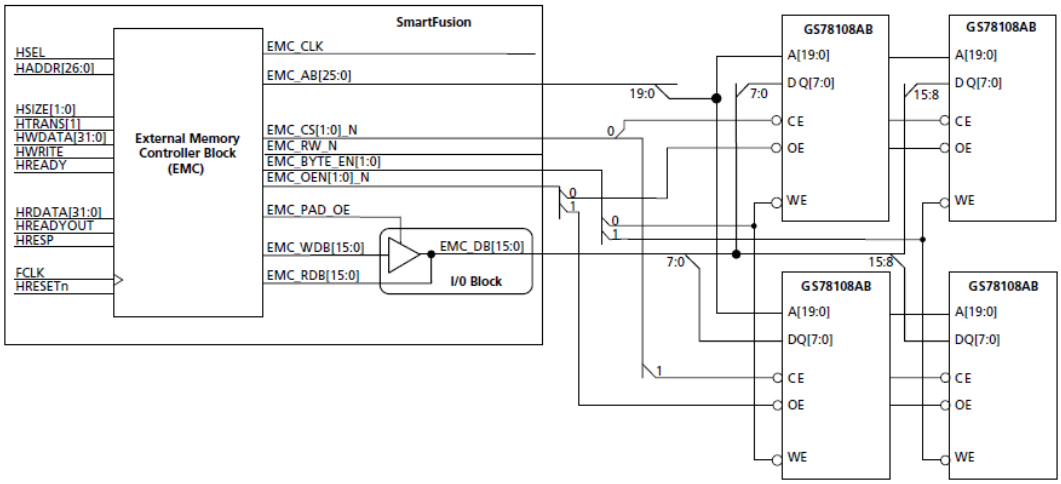

Table 7-1 • **External Memory Controller Memory Regions**

Chip Select	Starting Address	Ending Address
EMC_CS0_N	0x70000000	0x73FFFFFF
EMC_CS1_N	0x74000000	0x77FFFFFF

(a) (5 pts) How many **bytes** can each GS78108 store (you'll need to find the chip's datasheet for this)? How is the memory organized within each chip (# of data bits X size address space)?

(b) (10 pts) Assume that memory is initialized to zero and the following code executes:

```
BASE_EMC = 0x74000000;
uint32_t *a = (uint32_t*)BASE_EMC;
*a      = 0x76543210;
*(a-1) = 0xfedcba98;
```

Fill out the following table with the memory contents after executing the prior lines of code:

```
Mem. Addr. |31..24|23..16|15...8|7....0|
=========================================
0x74000004 |      |      |      |      |
0x74000000 |      |      |      |      |
0x73FFFFFC |      |      |      |      |
0x73FFFFF8 |      |      |      |      |
```

(c) (5 pts) Assume that the memory chips are numbered as follow: upper left (#1), upper right (#2), lower left (#3), and lower right (#4). Fill out a table that maps each of the following system addresses to one of the four chips: `0x74000000, 0x74000001, 0x74000002, 0x74000003`, and explain in detail how you determined this mapping from the schematic.

(d) (10 pts) Draw a timing diagram that shows both the AHB-Lite and EMC signals for the following write operation (note: this writes a 32-bit word onto a 16-bit bus): `*a = 0x76543210;`

Problem 3: Library/ABI (40 pts). Write a library fully in assembly (i.e. no C code or inline assembly is permitted, but you will need to provide a C header file). The library should provide the following function signature and should compile using the Actel SoftConsole tools and be able to run on the SmartFusion board:

```
/**
 * Changes the state of LED number "num" to "state".
 * @param   num the index number of the LED to change (0 to 7).
 * @param   state the desired state of the LED (s/b 0 or 1).
 * @return  the value that LED num was set to (0 or 1).
 */
uint8_t setled(uint8_t num, uint8_t state);
```

(a) Write a C header file ("`leds.h`") that includes this function signature.

(b) (10 pts) Write a C program that uses this function to implement a Knight Rider-style display with an approximately 100 ms delay between LED state changes (you can use a busy loop if you like). Unlike Knight Rider, one LED can turn off completely before the next LED turns on. If you're not totally sure what this means, see the following URL, which has an animated GIF: http://www.coilgun.info/kitt_car/home.htm

(c) (25 pts) Write an ARM Thumb-2 compliant assembly language program ("`leds.s`") that uses the ARM Architecture Procedure Call Standard (ARM AAPCS or EABI) to implement the `setled()` function. Make sure that you use the proper ABI to pass variable into the function and that you save any registers that you need to use on the stack.

(d) (5 pts) Create these files within a SoftConsole project. Make sure that the project compiles using the Debug target.

(e) **Deliverables:** To receive points for this problem, you need to ZIP up the contents of your SoftConsole project and email it to eecs373@gmail.com with the subject line: "f10-hw1-p3" one before the homework due date and time.