Exam \#:

# EECS 373 Practice Midterm \& Homework \#2 Fall 2011 

Name: $\qquad$ Uniquename: $\qquad$

Sign the honor code:
I have neither given nor received aid on this exam nor observed anyone else doing so.

Scores:

| Problem \# | Points |
| :--- | ---: |
| 1 | $/ 20$ |
| 2 | $/ 15$ |
| 3 | $/ 10$ |
| 4 | $/ 15$ |
| 5 | $/ 10$ |
| 6 | $/ 10$ |
| 7 | $/ 10$ |
| 8 | $/ 10$ |
| Total | $\mathbf{1 0 0}$ |

## NOTES:

- PUT YOU NAME/UNIQUENAME ON EVERY PAGE TO ENSURE CREDIT!
- Can refer to the ARM Assembly Quick Reference Guide
- Can use a basic/scientific calculator (but not a phone, PDA, or computer)
- Don't spend too much time on any one problem.
- You have 80 minutes for the exam.
- The exam is 7 pages long, including the cover sheet.
- Show your work and explain what you're doing. Partial credit w/o this is rare.

Name: $\qquad$ Uname: $\qquad$

1) Fill-in-the-blank or circle the best answer. [ 20 points, $\mathbf{- 2}$ per wrong or blank answer]

b) A DRAM / SRAM / PROM is non-volatile. DRAM most commonly uses a $\mathbf{~ I T ~ / ~ 4 T ~ / ~ 6 T ~}$ cell. SRAM most commonly uses a $\underline{\mathbf{1 T} / 4 T / \mathbf{6 T}}$ cell.
c) A 100 MHz clock with a duty cycle of $25 \%$ is high for $\qquad$ $n s$ per cycle.
d) An EABI-compliant procedure could use registers / stack / both / neither to receive arguments from the caller, and the callee should never/sometimes/alwavs save and restore the registers it uses.
e) The I2C bus is most commonly used to connect chips on a board / devices in a rack / computers and modems.
f) The SPI bus is most commonly used to connect one / two / many master device(s) to one / two / many slave device(s).
g) You would expect that $\underline{\boldsymbol{I 2 C} / \boldsymbol{S P I}}$ would be able to support the higher data rate.
h) Differential signaling over a twisted pair of wires is used in a number of modern buses to reduce the impact of external noise / make it easier to not have a shared clock/ implement Manchester encoding / increase the speed of the data lines.
i) Multiple devices can usually share an edge / level triggered interrupt line.
j) The RESET interrupt is maskable / non-maskable.

Name: $\qquad$ Uname: $\qquad$
2) Consider the following 3-bit ADC. Draw the conversion transfer function (binary output vs input voltage) on the top graph. Draw the quantization error transfer function (error voltage vs input voltage) on the bottom graph. Make sure the transition points are clear. Assume Vref is 5 V . [15]

3) Assume you have a 3-bit SAR ADC. The analog input is 0.35 V and the Vref is 1 V . Show how the SAR would approximate the analog input over three cycles. Label the cycles on the x -axis and show the approximation as a meandering stair-step line on the graph. [10]


Name: $\qquad$ Uname: $\qquad$
4) ARM assembly [15]
a) Write ARM assembly code that sets bits 4,5 , and 6 of register $r 0$ to " 1 s " without changing any other bits in that register. Use as few lines of assembly as you can. [5]
b) After running the following code snippet, the value of $r 2$ at done is $\qquad$ . Show how your arrived at this conclusion. [5]

```
•••
start:
    movs r0, #1
    movs rl, #1
        movs r2, #1
        subs r0, r1
        bne done
        movs r2, #2
done:
        b done
•••
```

c) Write an ARM assembly language procedure that implements the following function in an EABI-compliant manner and conforms to the following signature. Identify which register holds which argument. [5]

```
f(x,y,z) = x+y-z
int32_t func(int32_t x, int32_t y, int32_t z);
```

Name: $\qquad$ Uname: $\qquad$
5) Assume you have a memory-mapped register location REG_FOO on a 32-bit architecture.

Modify main to add 7 to REG_FOO's value. Your code should compile without any warnings. [10]

```
#include <stdio.h>
#include <inttypes.h>
#define REG_FOO 0x40000140
main () {
    // Your code here to declare reg and
    // add 7 to REG_FOO's value.
    printf("0x%x\n", *reg); // Prints out new value
}
```

6) Say your code writes the value $0 \times 76543210$ to memory location $0 \times 20000604$, on an ARM Cortex-M3. Assume this memory location sits in an SRAM with no access delay (i.e. a zero wait state). If you could attach a logic analyzer to the AHB bus, draw a timing diagram that shows what you would expect to see on the logic analyzer for this write operation. Make sure the actual values being transferred on the bus are clearly labeled. [10]

AHB Bus


Name: $\qquad$ Uname: $\qquad$
7) Imagine that you want to attach an LED to the ARM Advanced Peripheral Bus (APB). Sketch out the glue logic needed to interface a D flip-flop (DFF) to the APB. Assume that the PSEL line is the peripheral select (i.e. it goes high when the processor is addressing the DFF) and that the DFF will be attached to data bits (PWDATA[0] and PRDATA[0]), perhaps through tri-state drivers. You should be able to change the value of the DFF by writing the memory location corresponding to the PSEL and read the current value of the DFFs by reading the location. Assume that the DFF has an enable (ENA) line that, when not asserted, ignores any inputs (e.g. D and CLK) but continues to drive Q and $\mathrm{Q} \#$. The APB signals available to you are: PCLK, PADDR, PWRITE, PSEL, PENABLE, PWDATA, and PRDATA. [10]

Name: $\qquad$ Uname: $\qquad$
8) Write a $C$ function void enable_interrupts (int $\mathbf{x}$ ) that enables interrupt $\mathbf{x}$. You need not check to validate that x is a legal interrupt number. The table below might be useful. [10]

Table 6-1 NVIC registers

| Address | Name | Type | Reset | Description |
| :---: | :---: | :---: | :---: | :---: |
| 0xE000E004 | ICTR | RO | - | Interrupt Controller Type Register, ICTR |
| 0xE000E100 - <br> 0xE000E11C | NVIC_ISER0 NVIC_ISER7 | RW | 0x00000000 | Interrupt Set-Enable Registers |
| 0xE000E180 0E000xE19C | NVIC_ICER0 - <br> NVIC_ICER7 | RW | 0x00000000 | Interrupt Clear-Enable Registers |
| 0xE000E200 0xE000E21C | $\begin{aligned} & \text { NVIC_ISPR0 - } \\ & \text { NVIC_ISPRR } \end{aligned}$ | RW | 0x00000000 | Interrupt Set-Pending Registers |
| 0xE000E280 - <br> 0xE000E29C | NVIC_ICPR0 NVIC_ICPR7 | RW | 0x00000000 | Interrupt Clear-Pending Registers |
| 0xE000E300 - <br> 0xE000E31C | NVIC_IABR0- <br> NVIC_IABR7 | RO | 0x00000000 | Interrupt Active Bit Register |
| 0xE000E400 - <br> 0xE000E4EC | NVIC_IPR0 - <br> NVIC_IPR59 | RW | 0x00000000 | Interrupt Priority Register |

void enable_interrupts(int x) \{

