EECS 373
Exam 1

Winter 2005

February 24, 2005

SOLUTION KEY

Name:

University of Michigan uniqname:

Open book, open notes. Calculators are permitted, but no laptops, PDAs, cell phones, etc. This exam has four problems in this packet, plus one bus interface design problem in a separate packet. Please be sure to put your name and uniqname on BOTH packets. Problems vary in difficulty; it is strongly recommended that you do not spend too much time on any one problem.

The rules of the Honor Code of the University of Michigan College of Engineering apply for this exam.

Honor code pledge: I have neither given nor received aid on this examination, nor have I concealed any violations of the Honor Code.

Signature:
(examinations without a signed honor pledge will not be graded)

1. Memory types *(12 points)*:

You are given the task of designing a portable scientific instrument that will be used to acquire sensor data and process it. Sensor data is acquired at very high speed into a fast memory; this is controlled with specialized logic implemented in an FPGA. After acquisition, the processing is done with a conventional microprocessor using a program that is stored on the instrument. This processing requires a fairly large memory, but extreme speed is not required. The system is exposed to severe mechanical shock and vibration, so you cannot use a hard drive. All storage must be implemented with electronic memory components. The system has the following storage requirements:

- 64 megabyte filesystem to store the program and data processing results
- 512 megabyte main memory for the microprocessor (moderate speed)
- 4 megabyte buffer for sensor data acquisition (very high speed)
- 2 megabyte storage for the bit file used to program the FPGA

The system will have the program and FPGA bit file installed in the lab. It will be taken into the field, used to collect results, and then returned to the lab. Back at the lab, processed results will be retrieved and updated processing programs and FPGA bit files will be installed before the instrument is returned to the field. Note that the memory internal to the FPGA chip is volatile; it must be programmed from the bit file at each system power up.

From the following choices select the one MOST APPROPRIATE type of memory for each system function:

- mask programmable read only memory (ROM)
- fuse programmable read only memory (PROM)
- diode programmable write only memory (DWOM)
- dynamic random access memory (DRAM)
- static random access memory (SRAM)
- flash memory (FLASH)

SOLUTION:

FLASH	64 megabyte filesystem
DRAM	512 megabyte main memory
SRAM	4 megabyte high speed sensor data buffer
FLASH	2 megabyte FPGA bit file storage

2. Asynchronous serial communication *(5 points)*:

Consider an asynchronous serial data connection with 1 start bit, 8 data bits, and 1 stop bit. This is a vary basic asynchronous connection; there is no bit stuffing, no adaptive clocks, no parity bit, absolutely nothing fancy. The receiver has been designed so that it attempts to sample the received data near the center of the appropriate time for each bit.

Approximately how much total clock frequency mismatch between the transmitter and the receiver can be tolerated without causing errors?

(Example: if the transmitter ran at 101 KHz and the receiver ran at 99 KHz, this would be about a 2% mismatch.)

Select the one best answer from the choices below:

(a) 0%; the transmitter and receiver clocks must match exactly or there will eventually be errors if a long enough message is sent.

(b) about a 5% mismatch is tolerable

(c) about a 12% mismatch is tolerable

(d) about a 50% mismatch is tolerable

(e) there is no limit to the clock frequency mismatch; with asynchronous serial communication, accurate data transmission does not depend at all on having matched clock frequencies.

Solution: each character takes 10 bits to transmit (1 start + 8 data + 1 stop). In this time the maximum drift between the transmitter must be less than half a bit time. Thus, the error must be less than half a bit in 10 bits, or one part in 20, or 5%.

The correct answer is (b), about 5%.

3. USB communications *(18 points)*:

The basic clock rate of a high speed USB bus is 480 MHz. Each bit transmitted takes one clock period. Thus, the raw data rate on the wires is 480 megabits per second. A certain, unknown data pattern is being transmitted. Standard USB signalling (bit stuffing and encoding) is being used. You connect an oscilloscope to the USB bus and discover that the transmitted data is a symmetric 240 MHz square wave.

What bit pattern is being sent, before bit stuffing and encoding has been applied? (In other words, what bit pattern, after bit stuffing and encoding, will turn into a symmetric 240 MHz square wave?)

Solution: a symmetric 240 MHz square wave is just alternating zero and one bits:

01010101010101010101

To get back to the data being sent, we must NZRI decode and remove any stuffed bits. After NZRI decoding:

000000000000000000000000

Unstuffing is trivial since there are no 1 bits at all (and certainly no runs of six 1 bits).

Thus, the data being sent is just a solid run of 0 bits.

000000000000000000000000

What is the lowest frequency symmetric square wave you might observe on the wires? (calculator optional; a correct answer given as a fraction or other expression will be accepted for full credit).

What bit pattern would have to be sent to produce this?

Solution:

For a low frequency, we want as few transitions as possible in the bit stuffed, NZRI encoded data. A 1 bit in the unencoded data corresponds to no transition in the NZRI data. Thus, we want runs of 1 bits that are as long as possible. Bit stuffing imposes a limit of six consecutive 1 bits before a 0 bit is introduced. Thus, the best we can do is six 1 bits and a zero bit, six 1 bits and a zero bit, etc. The data sent to produce this would be a solid stream of 1 bits:

111111111111111111111111

Bit stuffing would introduce a 0 bit after every sixth 1 bit:

1111110111111011111101111110

NZRI encoding would produce this:

0000001111111000000011111110

The steady repeating pattern is seven 0 bits, seven 1 bits. Thus, the pattern is periodic with a period of 14 bit times.

The resulting square wave would be at 480 MHz / 14 = 34.3 MHz

4. PowerPC assembly language and EABI *(30 points)*:

Below is a small function and an attempt to implement it as a PowerPC EABI compliant assemply language program. All variables are unsigned integers. The assembly language program contains numerous errors; find them. Don't worry about wasteful or inefficient code. Only report errors that will cause this to fail to assemble, fail to implement the specified function correctly, or violate the EABI interface. For each line on which you find an error, give the line number and a very brief description of the error. It is possible for a line to have more than one error. For errors without specific line numbers (such as if something important is left out), write "missing" instead of the line number. Three errors are given as examples.

```
fun(w,x,y,z) {
  w = ggg(w)
  a = w + 67890;
  b = (x + y) & -4
  c = hhh(987654321, 555555, z, 7)
  d = z + c
  e = (b - a) + 5
  return(e)
}

fun: stwu   r1,-30(r1)       #line 1   breaks stack alignment
     stw    r29,8(r1)        #line 2
     stw    r30,12(r1)       #line 3   didn't save regs contig through r31
     mfspr  r0,LR            #line 4
     stw    r0,4(r1)         #line 5   LR saved in wrong place on stack
     addi   r29,r3,67890     #line 6   immediate too large
     lis    r3,big@h         #line 7
     ori    r3,r3,big@l      #line 8   loads address, not value
     add    r14,r4,r5        #line 9   clobbers nonvolatile register r14
     andi.  r30,r14,-4       #line 10  andi. will zero top 16 bits
     ori    r4,r0,7          #line 11  r0 may have garbage in it, not taken as 0
     ori    r5,r5,555555@l   #line 12
     lis    r5,555555@h      #line 13  halves loaded in wrong order
     bl     hhh              #line 14  called with parameters in wrong registers
     add    r11,r6,r3        #line 15  z value in r6 may have been clobbered
     sub    r0,r29,r30       #line 16  arguments to subtract reversed
     addi   r12,r0,5         #line 17  can't addi using value in r0
     add    r5,r11,r12       #line 18  no return value in r3
     lwz    r30,8(r1)        #line 19
     lwz    r29,12(r1)       #line 20  registers restored swapped
     blr                     #line 21  LR and stack pointer not restored
     crash  r56              #line 22  EXAMPLE, bad instruction and register

big  .word  987654321
```

(continued from previous page)

List of errors found:

22: nonexistent instruction “crash”

22: invalid register number “r56”

missing: function “w = ggg(w)” is never called

(find as many more errors as you can, full credit for 15 additional errors)