EEE4084F Final Exam 2010

SAMPLE SOLUTIONS

Section 1

Question 1
There are three general classes of system for carrying out electronic computation: 1) purely hardware (e.g., the computation is fixed); 2) reconfigurable computer systems; and 3) software processor systems.

(a) [6]

- A purely hardware system (e.g., an ASIC) typically provides the highest performance, best efficiency (e.g., in terms of operators / WATT), and the highest level of parallelism. However, a hardware only solution tends to be the most expensive (for complex computation tasks) and is static (cannot be changed).
- The software processor solution is considerably more flexible than the hardware only approach, and can be significantly cheaper to develop applications for (assuming that the underlying microprocessor itself is not developed from scratch). The software approach has limitations such as clock speed and available interconnects / supported peripheral interfaces.
- The reconfigurable computer (RC) approach is somewhat in the middle between a fully hardware system and a software system. The RC has a high level of flexibility (e.g., ability to do much in software and also change the hardware structure on the fly), while still allowing for varying degrees of parallelism. However, a major disadvantage to RC is the difficulty of using these systems (i.e., developing and deploying user applications).

(b) [2]

The determining characteristic of a reconfigurable computer is its ability to change its interconnections between processing elements. An FPGA-based RC computer is somewhat of an extreme example, as it can change interconnects on a gate level (or at least programmable element level, which is a collection of logic gates). Non-reconfigurable computers cannot change the interconnections between their processing elements (for example, most SMP processors have a fixed internal structure and cannot dynamically reroute the way in which their cores are linked).

(c) [2]

Communication Latency is the amount of time it takes to send a *minimal length* (i.e., zero byte) message from one task to another. This usual unit (nowadays) is microseconds.

Effective bandwidth is the *actual* (or *effective*) speed that data is sent over a connection. This is in contrast to the theoretical maximum that the connection can carry (which is often termed the maximum bandwidth). The calculation of effective bandwidth involves:

1. The sending overhead (the latency in sending bits to the network card / modem / talking to the DCE);
2. The transport latency (time to send the bits of the message + time it takes a bit to travel along the line); and
3. The receive overhead (the latency it takes the the network card/modem/DTE side to demodulate and validate the incoming bits)

Question 2

(a) Common examples of applications that run on these large-scale computer clusters include: Databases, web servers, internet commerce / OLTP (online transaction processing), technical computing, treat analysis, credit card fraud detection.

[2]

(b) A major difference between the design of an SMP processor and that of a Cell processor, is that the SMP processor has the same processor cores (e.g., the Intel Core2 Duo has two cores that are the same). The Cell processor, on the other hand, has a non-uniform collection of cores: it has a PowerPC (or 'power processing element') plus eight SPEs (Synergistic Processing Elements). The Cell processor also has a different memory subsystem internal to the processor, controlled by the MIC (memory interface controller), and it also has a specialized I/O system, namely the Rambus XIO system. SMPs generally has a simpler I/O and memory system, using two levels of cache, a small L1 cached private to each core, and a shared L2 cache shared between cores. The is no specialized I/O system; the cores share the same I/O lines and needs to collaborate with one another to avoid contention.

[3]

(c) While SMP processors may not be as high-end as Cell processors, they do tend to be easier to program. These architectures have seen much wider use, and have a greater deal of support, both in terms of free forums and the open-source than can be found for the Cell processor. Furthermore, it can be easier, and more intuitive, to conceptulize parallel designs for a SMP (i.e., the set of exact processors) rather than determining how to apply a solution using the more complex cell processor structure (this difficulty in may be partly responsiblity to radically change the design the new cell processor architecture). SMP-based computers, especially ones with only a few cores, are usually less expensive than Cell processors, which can lead to the cluster having more processors.

[3]

(d) While it may be easier to use either PThreads or MPI, and also make more readable code, there may be performance losses. MPI generally involves specifying which nodes to communicate with, and passing data between the nodes. This passing data between nodes might involve sending data from one PC to itself over the ethernet loopback - this is usually slower that just using shared memory, which you can do with Pthreads if the two threads are running on the same machine.

[4]

Question 3

(a) 1950 [1]

(b)

Database processing (D)	Back-end
Sampling ADCs (S)	Front-end
Pulse compression (P)	Front-end
Convolvers or FIR filtering (C)	Front-end
Clutter mitigation (M)	Back-end

[5]

(c) A measure for complexity is measuring the lines of code that is developed for a HPEC system; the usual unit is SLOC (source lines of code). Other measures of complexity are the number of software modules. The size of a FPGA bit image. The number of interconnects between hardware modules, or between software modules.

[4]

Measures of performance include Dhrystone, which describes how fast a processor can loops though a specific set of signal processing operations; the unit of measure is the DMIPS (Dhrystone MIPS). Whetstone is another performance measure like Dhrystone. This question has been left fairly open because the student should be able to think of many ways to measure performance and complexity as these issues were discussed in detail in the lectures and seminars.

(d) GPUs are suited to applications that cam be implemented following the Flynn's SIMD model, i.e., where all threads run the same set of instructions but many times in parallel. Examples applications include matrix operations such as matrix multiple, and image processing algorithms like edge detect or blur.

[2]

Section 2

---

1. (a) [4]
2. (c)
   1000ms for 1 thread
   48ms to initialize 1 thread
   Leaves 952ms for the work done by the thread
   For 4 threads, initialization time is 48x4 = 192ms
   The 952ms is divided into 4 = 238ms per thread in parallel
   The total time is then 192 + 238 = 430ms to complete for 4x threads
   Therefore speedup is 1000/430 = 2.3ms (to once decimal palce).
3. (d) - it's a slightly trick question: obiously any modern SMP is also a MIMD; however the code looks suited to a SIMD because all 4 threads run the same set of instructions but on different data. [4]
4. (b) [4]
5. (i) F (it returns 1)
   (ii) T
   (iii) F (the book says COTS are becoming more popular, but used in a custom design)
   (iv) F (SWAP means Size, Weight, And Power

[1 x 4 = 4]

Section 3

Q1 (a) The configuration architecture is the part of the RC platform design that relates to configuring the rest of the system, in particular how the data paths are configured and curcuitry needed for programming the FPGAs / PLDs (if used).

[2]

Q1 (b) LUT = look-up table. The main reason for choosing to use LUTs in the design of FPGA PBs is their ease of use (i.e., simply describing the truthtable instead of trying to design a circuit using gates). A second (but less significant) reason is that LUTs can be easier to implement that various gates (e.g., using blocks of SRAM).

[2]

Q1 (c) Xilinx's was probably trying to improve overall performance, while also making compile/routing time faster by allowing bigger blocks to be connected. These 'bigger blocks' also contain I/O optimizations and other features that are not found in the standard blocks. These 'bigger blocks' are likely based on commonly occurring design patterns , providing a means by which parts of VHDL programs can be efficiently implemented (in terms of both energy and use of programmable interconnects) using them. However, the Xilinx designers also did not want to limit the possible applications for their FPGAs, so only some of the PBs are these 'bigger blocks'; the rest are more fundamental logic that suites a broader range of applications and other 'pieces' that may need to be added into a design to make it work. The reason for the SLIDEMs and SLIDEL having similar interfaces are probably to simplify the design and routing algorithms.

[5]

Q1 (d) *See last page*

[13]

Q2 (a) Slow clock can be implemented using a down-counter that outputs a logic 1 result when zero is reached. The down-counter is clocked by the master clock. Initially the counter is set to the value 200 (this can be done by linking its reset line to its output). Each clock causes the counter to subtract 1 from its value until 0 is reached. The down-counter output can be used as a 1MHz clock.

Arguably, a sequence of toggle (T-type) flip-flops, with the Q of one flip-flop inverted and fed back into the same flip-flop's T input, could be used (note the first flip-flop would be clocked by the master clock). But this approach is only viable if it's a division that is a multiple of 2 (e.g., divide by 2, 4, 8, etc). But if it's a big division (e.g., divide by 128 or more), then it may start taking excessively many logic elements to implement the divide. Regardless, 200MHz down to 1MHz is obviously a divide by 200, so this approach wouldn't work in this particular instance.

[6]

Q2 (b) Handle-C code:

```
void PPG ( _in unsigned short L, _in unsigned short H, _in bit start,
           _in bit enable, _out bit out, _in bit SCLK
         )
{
   static bit prev_start = 0;
   static unsigned short saved_L;
   static unsigned short saved_H;
   static bit prev_out = 0;
   static bit started = 0;
   static bit prev_sclk = 0;
   static unsigned short cnt_L;
   static unsigned short cnt_H;
   // the entity is only active when enable is high
   if (enable) {
      if (prev_start != start) {
         // start pulse:
         prev_start = start; // save previous value
         if (start) { // positive edge:
             saved_L = L; // save L input
             saved_H = H; // save H input
             cnt_L = L;
             cnt_H = H;
             started = 1;
                 prev_sclk = SCLK;
         }
        }

      // check if SCLK clocked
      if (prev_sclk != SCLK) {
         prev_sclk = SCLK;
         // if positive edge...
         if (SCLK) { // 1us has elapsed, so count down either cnt_H or cnt_L
```

```
            if (prev_out) {
               if (cnt_H == 0) { // have we waited for H us?
                  prev_out = !prev_out; // yes time to toggle out
                  cnt_H = saved_H;
                } else cnt_H = cnt_H-1;
               out=prev_out;
               } // end if prev_out
            else {
               if (cnt_L == 0) { // have we waited for L us?
                  prev_out = !prev_out; // yes time to toggle out
                  cnt_L = saved_L;
                } else cnt_L = cnt_L-1;
               out=prev_out;
               } // end else
            } // end if (SCLK)
        } // end if

    } else {
     // when disabled just continue making sure started is low
     started = 0;
    }
}
```

[2 marks allocated to suitable use of comments]

[12]

Q2 (c) How to handle one input instead of two.
A second control bit could be added, call it for example choose_HL, and the H and L could be replaced by a 16-bit D. The use could initially set enable low, and choose_HL to low. The use could then set the enable high (but keep start low), and then set D to the H value to send, then then send a positive edge over choose_HL, and the PPG could then set its saved_H to D. Then D could be set to the L value, and a negative edge sent over choose_HL, and a PPG could save D into saved_L. Later, start pulse could be sent (but the start-pulse would not cause saved_L, saved_H to be changed, it would simply use what was set previously using choose_HL).

[4]