EEE4084F Exam Sample Solution 2011

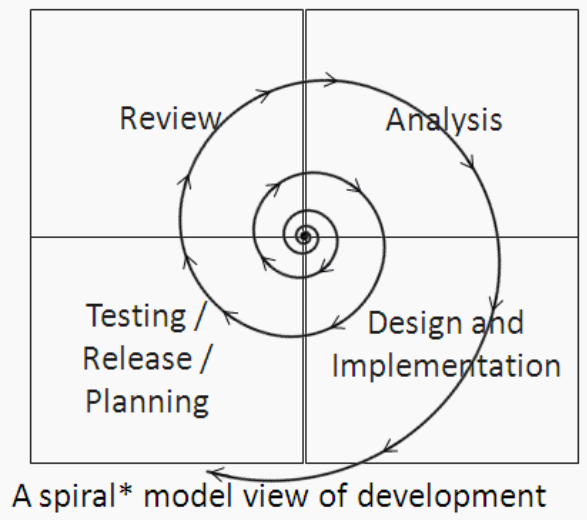
SECTION 1

Q1.1 (a)

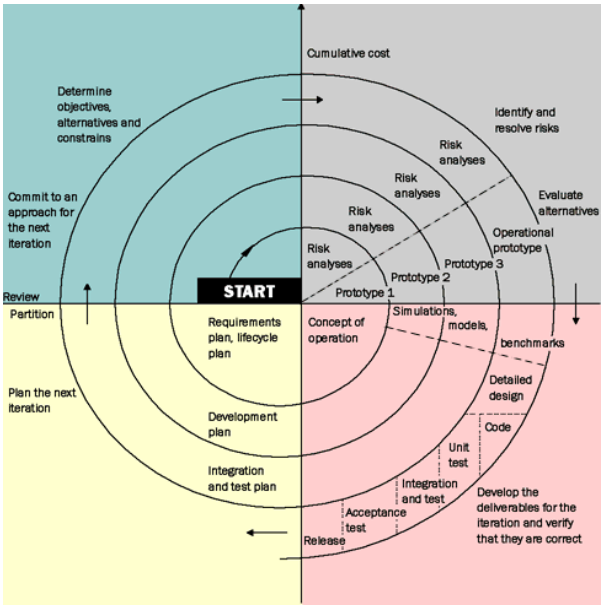
Major activities repeated:

- Analysis
- Design/implementation/prototyping
- Testing and planning for the next iteration
- Review

The spiral model tends to start small and 'wind' into an increasingly more complex and complete product. The diagrams below illustrates suitable spiral model. (note on marking: a level of detail as shown on the left is more expected in a student's answer, i.e., an indication of where the 4 activities above could be position would be sufficient – the more detailed diagram is just give as a reference for marking).



A spiral* model view of development



Q1.1 (b) The student will be expected to discuss how 'step 1 understanding the problem' would likely to involve the complete first cycle of the spiral model, in which no actual product prototype is developed. The second cycle would likely be doing partitioning, followed by a cycle that involves deciding the granularity, and so on.

Q1.1 (c) common design problems:

- Insufficient design and planning done
- No documents (or poorly formed)
- Inefficient data structures / file formats
- Infrequent or no design reviews
- Lack of consultation/input from experts and senior engineering staff

Q1.2

(a) Details on this is given in the textbook. Essentially, it is useful in establishing whether the interconnections between arbitrary processing nodes, that may need to communicate, have an adequate bandwidth in the system.

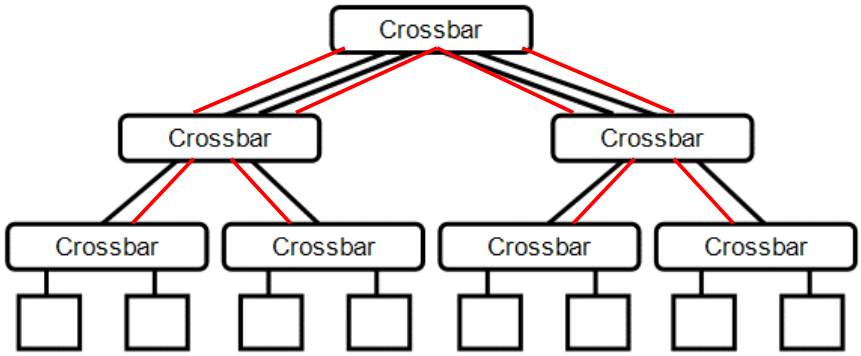
(b) Bisection bandwidth for this one is calculated using the general formula on row 2 in appendix B. The respective variables are as follows:

q = 3 d2 = 1 u1 = 1 d2 = 2 u2 = 2 d3 = 4 u3 = 0

d3/2 x (u1 x u2) = 4/2 x 2 x 1 = 2 x 2 = 4 x Bandwidth = 4 x 1Gbps = 4Gbps

(This may seem to be a high bisection bandwidth, but this is just a measure on the connection between parts; it doesn't mean to say that any one node can communicate with another at 4Gbps which is beyond any one downlink to a specific node).

(c) A fat has non-root nodes with multiple parents, increasing bandwidth towards the top of the tree. One approach is as follows:



(d) This question was a little bit design to trick students. The answer is the maximum speed is 1Gbps from P1-P8 and the same from P5 to P3 since there are adequate channels to do so without having to share them. Adding additional downlinks in level 2 (i.e. connections from level 2 crossbars to level 1 crossbars, a the fat tree would have) doesn't help in this case.

Q1.3

(a) A regonfigurable computer is a computer system that is able to change its hardware datapaths and control flows via software control.

(b) LUT = look up table, it is essentially a logic element that works much like ROM memory – it is given a set of logic inputs which are mapped to one or more logic outputs. An FPGA typically has a collection of LUTs each of which can represent a combinational logic circuit of multiple gates, which is easier to program and manage than the routing of, than the interconnections between individual logic gates.

(c) A combinational logic block is a combination of logic elements, such as multiple LUTs, BRAM, invertors and other gates, essentially a collection of useful building blocks composed into one block which can accommodate a range of combinational logic designs. The same CLB is then duplicated multiple times in a FPGA and this

simplifies the programming and routing of the device. SLICEL blocks are standard blocks that mostly have LUTs and commonly used combinational logic elements; SLICEMs are more fancy and tend to include less commonly used, but more powerful functionality, such as multipliers, dividers, larger memories and possibly even DSP processors as well.

(d) It is a synchronization problem. Entity 2 has no handshaking in place in order to use the correct average (avg output) value before outputting its DeltaN result. But this handshaking can't really be remedied without changing the code of the entities to add additional handshaking lines. Consequently, the most likely solution is to put a delay on the WStrobe, so that Entity 2 'sees' the strobe a few nanoseconds after Entiry 1 does and once that entity had processed its data. The DataIn line would be assumed not to change soon after WStrobe is pulsed.

(e)

SECTION 2

Q2.1. (d)

Q2.2. (c)

Q2.3. (b)

Q2.4. (b)

Q2.5. True or False answers:

(i) The commonly used HPEC acronym "SWAP" means Stop Wait And Process. **FALSE!**
(ii) VHDL can be used to program FPGAs, but it can't be used to program CPLDs. **FALSE!**
(iii) Front-end processors tend to be the highest-speed element of a HPEC system. **TRUE!**
(iv) A correlation between two identical data sets returns the value 1. **TRUE!**

SECTION 3

3.1 (a)

```
// Golden measure in C++:
#include <iostream>
Using namespace std;
#define LENGTH 16384
double a[LENGTH],b[LENGTH];
int main ()
{
  // populate a and b arrays
  double res = 0;  // set sum to 0
  for (int i=0; i<LENGTH; i++) sum+=a[i]*b[i];  // calc the vect product
  cout << "result = " << res << endl; // display the result
  return 0; // exit the program
}
```

3.1 (b)

```
Change struct as follows (at line 15):
typedef struct {  // Define struct for passing parameters to threadfn
   int id;
   double res;  // <= addded
   string text;
 } Params_threadfn;



void *threadfn (void *arg) {
   Params_threadfn& p= *((Params_threadfn *)arg);

    // the p.id provides the thread number. Somehow, one also needs to
    // know the total number of threads, which is stored in the global n
   int sz = LENGTH/n; // size of each segment to do in this thread
   if ((sz*n)<LENGTH) sz++; // have some overlap if needed
   int starti = p.id * sz;
   int endi = (p.id+1)*sz-1;
   while (endi>LENGTH) endi--; // back off incase overshot
   double res = 0.0; // just faster to declare a local
   for (int i=starti; i<=endi; i++) res+=a[i]*b[i];
   p.res = res;
   return 0; // The thread stops at this point
 }
```

Insert after line 58:

```
double sum = 0.0;
for (i=0; i<n; i++) sum+=p[i].res;
printf("result = %f\n",sum);
```

3.2

The solution would likely be along the lines of the following code:

```
// C prototype for pulse length calculator (PLC)
void PLC ( _in  byte 8 x,   // input data byte
     _in  bit newx,  // positive edge when new data available
     _in  bit reset, // set high to clear the
     _out bit ready, // set high when device is ready for more input
     _out bit valid, // high when output is valid
     _out unsigned 16 period, // num samples from one peaks to the next
     _out unsigned 8 peak, // the most recent peak value
     )  {
  static byte 3 goingup; // 0=down 1=up 2 = undefined
  static byte prevx; // previous value. A byte is default 8 bits
  static bit prev_newx; // the previous value of newx
  static unsigned 16 n;
  static unsigned 16 lastperiod; // storage needed for period value
  static byte lastpeak; // some storage needed for peak value
  // remember that the function essentially loops continuously
  if (reset) {
      valid = 0; period = 0; peak = 0; // set all the outputs to a neat 0
      lastpeak = 0;  // reset the past peak
      lastperiod = 0;
```

```
      goingup = 2; // going up is a 2-bit value
      prevx = 0;
      prev_newx = 0;
      n=0; // number of samples since previous peak
      return; // just loop back when in reset state
  }
  // check the signal in comparison to the last value
  if (newx & (newx != prevnewx)) { // positive edge on newx?
     n++; // this was possibly a local minimum so we ignore it
     if (x>prevx) { // going up...
        goingup=1;
        } else
     if (x<prevx) { // no, doing down...
        if (goingup) {
           // a change from gown up to going down, so a local max
           lastpeak = prevx;
           lastperiod = n-1; // i.e. from last max to previous value
           n = 0;
           }
        goingup = 0;
        }
     else {}  // staying the same so do nothing
     prevx=x;
     }
  // update the outputs
  prevnewx = newx;
  period = lastperiod;
  peak = lastpeak;
  if (goingup == 0 || goingup == 1) valid = 1; else valid = 0;
}
```

3.3 (a)

X <= ((A or B) and not (A or C)) or not ((A or B) and (B or C))

Y <= ((A or B) and not (A or C)) or ((A or C) and (B or C))

3.3 (b)

The maximum delay is if all the NOT gates at each level are connected up, which will give a propagation delay of:

First mux + first gate + first not + second mux + second gate + second not + third mux + third gate + last mux

= 4 x mux + 5 gates = 8 x 4 + 10 x 5 = 32 + 50 = **82 nanoseconds** is the maximum propagation delay

The least propagation delay will be if none of the NOT gates are used. This would work out as:

4 x mux + 3 x gates = 8 x 4 + 10 x 3 = 32 + 30 = **62 nanoseconds** is the minimum prop delay.