Digital Systems
EEE4084F

FINAL EXAM
14 July 2017

Out of 120 marks

SOLUTIONS!!!

Section 1: Short Answers [50 marks]

Q1. [16 marks]

(a) The OIC is a Harvard architecture as it have a separate memory bus to peripheral bus. [1]

(b) Implementation of the assembly code below: (ticks shows mark allocations)

```
   #define ADC1    0x10
   #define ADC2    0x11
   #define TRIGGER 0x12
   CLR A
   ADD A, 10  // A = trigger_level  = 10
loop:
   IN X,ADC1   // X = IN(ADC1)
   IN Y,ADC2   // Y = IN(ADC2)
   CMP X,A     // compare X to A
   JMPgt writet
   SWP X,Y
   CMP X,B     // compare X to B
   JMPgt writet
   OUT TRIGGER,0
   JMP loop
writet:
   OUT TRIGGER,1
   JMP loop
```

For comments

[10 marks]

Q1 (c) The total number of instructions in the loop are 9 (for the worst case where it needs to check both X and Y inputs). This implies the loop runs at 9 x 1/100x10^6 = 9 x 10^{-8} = 90ns [3]

Q1 (d) The circuit takes 50 + 10 ns = 60ns to complete. Speedup = Eu / Ep (Eu the uniprocessor, Ep parallel version). Therefore Speedup = 90/60 = 1.5. [2]

Q1.2. [12 marks]

Q1.2(a)

i. The main difference between a FPGA and a PLA is the: the architecture (how the system is configured and programmed), the number of logic elements available, and the programming speed. The FPGA has a more complex architecture, supports more complex designs, usually many types of logic elements. [2]

ii. There is usually a particular programing sequence needed for an FPGA. In particular, if a FPGA board needs to start up without being programmed from a host (e.g. attached PC), there needs to be some way to program the FPGA. This is where a configuration architecture, utilizing a statemachine implemented using a PLA or CPLD, is used in order to read the FPGA program from non-volatile memory (e.g. a EEPROM chip) and to program the FPGA. Furthermore, the PLD/CPLD may also include logic to support programming from a host, i.e. to receive a program sent from the host into a then exercise the necessary programming pins on the FPGA in order to program it. [3]

Q1.2 (b)

Difficulties associated with taking an FPGA design forward to an ASCI design include accounting for differences in propagation delays and operational speeds, different layouts of components; possibly different implementations of components or CLBs that are utilized. Changes in the interconnections and electrical properties of the material used for the ASIC. Futhermore, the tool chains may be quite different and require the designer to undergo a lengthy learning curve to learn how to use the tools effectively. There would also need to be more reliance on simulation, due to the expense of running of physical instances of ASICs; whereas for FPGAs it is just a matter of programming the FPGA and testing it on hardware, using a development kit prototyped board. Risks for ASIC include the potential for having a re-do designs and the expense of additional runs to compensate for design faults. Further there may be the risk of hiring consultants to assist with ASIC design and that it is difficult to predict how long it will take to achieve a final operational ASIC due to the complexity of this practice. [4]

Q1.2 (c) Advantages of parallel code are: potential for increased performance (by doing multiple operations in parallel as opposed to being limited to sequential operation), the potential for redundancy and fault tolerance (e.g. running the same operation on multiple different processors which could be used to work around interference or damage that could cause processors to fail temporarily or permanently). Improved responsiveness / decreased latency, the ability to respond to interrupts more quickly, without necessarily relying on one available processor to handle the request. [3]

Q1.3

Q1.3 (a) SWAP = Size Weight And Power. Some measures:

computational power efficiency : GOPS/W

Size or volume efficiency of computation : GOPS/L (giga operations per litre)

Weight efficiency of computation : GOPS/Kg (giga operations per kilogram) [3]

Q1.3 (b)

i. The granularity of a problem, in computing, indicates the extent that the problem can be decomposed into big or small parts, moreover how interrelated the sub-tasks (or fragments of the problem space) are. This can be expressed using the ratio instructions:communications. Where

1:N → fine grained problem (high interdependence, each result needs a lot of communication, much of the data needed for each result generated)

1:1 → medium grained e.g. moving average filter (where each result depends on only a few items of the source data)

N:1 -> course grained (low interdependence, each computation needs little or no other data)
e.g. X[1:10]=0 is very course grained, even embarrassingly parallel, since no data is needed for any result. [3]

ii. Embassasingly parallel means that there is no or very little communication needed to produce a result of the computation. It is very course-grained. [2]

Q1.4 [10 marks]

Q1.4(a) The Bisection Bandwidth of a network determine as follows: the network is bisected into two partitions, the bisection bandwidth of a network topology is the bandwidth available between the two partitions. The bisection should be done in such a way that the bandwidth between the two partitions is a minimum. The bisection bandwidth is a useful metric to determine potential performance of a system as it indicates the worstcase network delay between any two nodes in the system which could be a constraining factor in the overall performance of the system.

I motivate for the bisection bandwidth of the figure being above 1GBps because A1 and A2 can simultaneous send and receive data from B5 and B6. [4]

Q1.4(b)

i. Obviously all the data from A1-A6 needs to pass over to B1-B6, and similarly all the data from B1-B6 has to be sent over the other way. So that is 2 x 6 x 100 MB = 1.2GB [1]

ii. The schedule of transfers would be:

```
PARALLEL TX 1 {
 A1->B6
 A2->B5
 A6->B1
 A5->B2
 B6->A1
 B5->A2
 B2->A5
}
PARALLEL TX2 {
 A3->B4
 A4->B3
 B4->A3
 B3->A4
}
```

TX1: sends 100MBytes at 1Gbps takes 100*8/1000 s = 0.8s (7 transfers in parallel)

TX2: sends 100MBytes at 1Gbps, so that is 100*8/1000 s = 0.8s (4 transfers in parallel)

Total time = 0.8 x 2 s = 1.6s. [5]

SECTION 2 [each question worth 5 marks]

Q2.1 (e)

Q2.2 (b)

The PCR as per the definition is determined by calculating the **<u>arithmetic operations</u>** the processor can do per clock cycle multiplied by the maximum clock rate of the processor.

Q2.3 (b)

Processor speed:

40MHz -> $1/(40 \times 10^6)$ s per instruction

For 50 instructions → $50 \times 0.25 \times 10^{-7} = 12.5 \times 10^{-7} = 0.00000125$

Propagation delay for PLD:

5×40 ns = 200 ns = $200 \times 10^{-9} = 2 \times 10^{-7} = 0.0000002$

Speedup = Eu / Ep = $12.5 \times 10^{-7} / (2 \times 10^{-7}) = 6.25$

So, in conclusion there is quite a good speedup of 6.25, the answer is (b).

Q2.4 (b)

Q2.5 (d)

Q2.6 (a)

SECTION 3: Long Answers [42 marks]

Q3.1

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: UCT
// Engineer: S. Winberg
// Module Name:    E2A
// Project Name: E2A Solution for Q3.1 in EEE5085F 2017 Exam
//////////////////////////////////////////////////////////////////////////////////

module E2A (
    // Inputs
    RESET,CLK,AWE,ASTROBE,EN_CONV,SQACK,DI,IRDY,
    // Outputs
    ADATA, ABUSY,AREADY,SQDATA,SQTB,DO,ORDY
    );
  // Declare the directions of size of each input
  input RESET,CLK,AWE,ASTROBE,EN_CONV,SQACK, IRDY;
  input [7:0] DI;
  // Declare the directions of size of each input
  output reg ABUSY,AREADY,SQTB,ORDY;
```

```
output reg [3:0] SQDATA;
output reg [7:0] DO;
// Declare the directions of size of each tristate (inout)
inout [3:0] ADATA;
// Internal registers for ESQM2ASCII state machine
reg nibble2;
reg [1:0]do_tx;
reg msn;
reg hold;
reg [7:0] asciiout;

// Start of the Code
always @(posedge CLK)
 begin
   // ------------------ Handle reset ------------------
   if (RESET == 1)
    begin
     ABUSY  <= 1'b0;
     SQTB   <= 1'b0;
     ORDY   <= 1'b0;
     AREADY <= 1'b1;
     SQDATA <= 0;
     DO     <= 0;
     // ESQM2ASCII state machine registers
     hold   <= 0;
     nibble2<= 1'b0;
     do_tx  <= 2'd0;
     msn    <= 1'b0;
    end // if
   else
    begin
    // ------------------ OPERATION ----------------------
    if (EN_CONV == 0)
      // -------------- CONVERSION OFF -------------------
      begin
          ORDY   <= 0;
          DO     <= 0;
          if (AWE == 1 && ASTROBE == 1)
            begin
                SQDATA <= ADATA;
                SQTB   <= ASTROBE;
               end
        if (ASTROBE == 0)
          begin
            SQTB   <= 0;
          end
        end
        else
        // ------------------ CONVERSION ON -----------------
        begin
         if (hold)
           // -------------- IN HOLD ----------------
           begin
               if (do_tx == 1)
                 begin
                     DO    <= asciiout;
                     ORDY  <= 1;
                     do_tx <= 2;
                   end
               else
               if (do_tx == 2)
```

```
                    begin
                        DO     <= 0;
                        ORDY   <= 0;
                        do_tx  <= 0;
                        hold   <= 0;
                        AREADY <= 1;
                        ABUSY  <= 0;
                        nibble2<= 0;
                       end
                end else // hold
                // ----------- OUT OF HOLD ---------------
            begin
                if (ASTROBE && AWE)
                  begin
                   if (nibble2)
                        begin
                         asciiout<= {msn,ADATA};
                            do_tx   <= 1;
                            hold    <= 1;
                            AREADY  <= 0;
                            ABUSY   <= 0;
                        end
                        else
                        begin
                         msn    <= ADATA;
                            AREADY <= 1;
                            nibble2<= 1;
                        end
                  end
           end // end not hold
          end // of convert on
     end // not in RESET
 end // always@ CLK

endmodule
```

See implementation and testbench at https://www.edaplayground.com/x/29Q7

[26 marks]

Q 4.2

```
#include <stdio.h>
#include <mpi.h>

/*
 Process for compiling and running:
  Compile using:
  mpicc -o q32 main.cpp

  Run using:
  mpirun -n 4 ./q32

 */

int fibonacci(int n)
{
   if ( n == 0 ) return 0;
```

```
   else if ( n == 1 ) return 1;
   else return ( fibonacci(n-1) + fibonacci(n-2) );
}

int debugon = 0;
#define DB1 if (debugon>=1)
#define DB2 if (debugon>=2)

int main ( int argc, char** argv)
{
  int N1, N2, D;
  int my_rank, num_procs;
  int todo; // number of items this process is to do
  int startn,endn; // start and end values to do
  // Assume these are hard coded:
  N1 = 0;
  N2 = 10;
  D  = 2;
  // Perform calculation
  MPI_Init(&argc, &argv);
  MPI_Comm_rank (MPI_COMM_WORLD, &my_rank);
  MPI_Comm_size (MPI_COMM_WORLD, &num_procs);
  todo = (N2-N1+1)/num_procs;
  startn = my_rank * todo;
  endn   = startn + todo -1;
  if (my_rank == 0) {
    printf("Welcome to the Fibonacci checker!\n");
    printf("  n1 = %d ",N1);
    printf("n2 = %d ",N2);
    printf("D = %d\n",D);
    }
  printf("p%d doing %d -> %d\n",my_rank,startn,endn);
  for (int i=startn; i<=endn; i++) {
     DB2 printf("p%d test: %d\n",my_rank,i);
     int fib = fibonacci(i);
     if (fib%D == 0) printf("F%d = %d\n",i,fib);
  }
  if (my_rank == 0) {
     // master to finish up
     DB1 printf("Master todo any remaining ones: \n");
     int left = (N2-N1+1)%num_procs;
     startn = N2-left+1;
     endn   = N2;
     DB1 printf(" doing %d: %d->%d\n",left,startn,endn);
     for (int i=startn; i<=endn; i++) {
       DB2 printf("fill p%d test: %d\n",my_rank,i);
       int fib = fibonacci(i);
       if (fib%D == 0) printf("F%d = %d\n",i,fib);
     }
     DB2 printf("Finalize\n");
     }
  MPI_Finalize();
}
```

[16 marks]