Digital Systems
EEE4084F

FINAL EXAM

3 June 2016

3 hours

REGULATIONS

This is a closed-book exam. Scan through the questions quickly before starting, so that you can plan your strategy for answering the questions. If you are caught cheating, you will be referred to University Court for expulsion procedures. Answer on the answer sheets provided. Make sure that you put your **student name and student number**, the course code **EEE4084F** and a title **Final Exam** on your answer sheet(s).

DO NOT TURN OVER UNTIL YOU ARE TOLD TO

Exam Structure
Marked out of 120 marks / 180 minutes. Time per mark = 1 min 30 sec

RULES

- You must write your name and student number on each answer book.
- Write the question numbers attempted on the cover of each book.
- **Start each section on a new page.**
- Make sure that you cross out material you do not want marked. Your first attempt at any question will be marked if two answers are found.
- Use a part of your script to plan the facts for your written replies to questions, so that you produce carefully constructed responses.
- Answer all questions, and note that the time for each question relates to the marks allocated.

Section 1: Short Answers [48 marks]

Question 1.1 [12 marks]

The following questions concern computation approaches.

1.1 (a) Briefly explain the difference between the concepts of temporal and spatial computation? You can use a diagram to elaborate if you want. **[4]**

1.1 (b) A particular platform can be supported by multiple ABIs. For example, the IBM Cell processor is supported by both the commercial IBM SPE ABI and the open-source Linux Cell ABI; yet the two ABIs are not identical. Explain what the acronym ABI stands for [1 mark] and what an ABI provides. In particular, how is an ABI is different from an API [3 marks]? **[4]**

1.1 (c) The processing done on many reconfigurable computing platforms are centred around high-performance FPGAs. Provide a brief argument motivating why a CPLD is sometimes included as a means for configuration and additional I/O routing, with the CPLD sometimes replaced or supplemented by a micro-controller to provide further facilities. [4 marks] **[4]**

Question 1.2 [12 marks]

This question concerns parallel and high performance computing.

1.2 (a) In an important business meeting with clients and your senior engineering colleagues, you bravely suggest, despite your inner voice berating you for being a show-off in your new job, that a parallel language should be considered as a way for quickly building, or at least piloting, a parallel computing solution. A colleague responds to you saying: “No way, that is too risky! Parallel computing languages don’t really exist and automatic parallelisation is still useless”. Present an argument to stand by your statement and illustrate examples confirming that parallel languages indeed exists – also (to be more of a smart-ass) clarify the current situation of automatic parallelisation techniques. **[6]**

1.2 (b) When porting a legacy sequential program into a parallel version, why is it that much of the code tends to remain sequential, with only a small part of the program actually made into a parallel form, and this part typically being meticulously hand-crafted instead of largely automatically generated (e.g. by using pragmas understood by the compiler)? **[4]**

1.2 (c) Nowadays, server class machines tend to use SMP architectures. Briefly explain what an SMP is and what the acronym SMP stands for. **[2]**

Question 1.3 [12 marks]

The question concerns parallel computing architectures and networking/interfacing associated with their use for high performance embedded system processing.

1.3 (a) What does the Von Neumann bottleneck refer to? **[3]**

1.3 (b) What is meant by synchronous communications? Does the concept 'blocking communications' refer to the same issue? **[3]**

1.3 (c) What does the acronym ENOB stand for in relation to ADCs? If two 12-bit ADCs have the same speed, but the one has higher ENOB and the other lower ENOB, which one would likely be the more accurate and also more expensive one to purchase – provide a brief argument to support your answer. **[6]**

Question 1.4 [12 marks]

This question relates to distributed memory and Message Passing. Please see Appendix B.

1.4 (a) What are some of the advantages of distributed memory over shared memory? Name at least two benefits. **[2]**

1.4 (b) A starting point MPI program code is given in Appendix C together with a list of MPI commands in Appendix B for sending messages. Please complete the program given in order to satisfy the following requirements: **[10]**

i The master needs to ask the user (in the console) if it must be in mode A or B. In mode A, it prints out computed x:$y(x)$ pairs. In mode B, it prints each x value paired with '+', '0' or '-' to indicate if the $y(x)$ values returned is positive, zero or negative. The master must determine how many processes (`world_size`) are active. If `world_size` is less than two, the program exits.

ii Each process will calculate the result $y(x)$, where x is the node rank number and sends the x value and $y(x)$ result back to the master (node 0). The master does not compute a value. The equation for $y(x)$ is given below:

$$y(x) = x^2 - 12x + 20$$

iii The master must print the x:$y(x)$ pairs sent to it from the other nodes in the manner described above, depending if it is in mode A or B. (Hint: the x values don't need to be in order.)

Example: If the world size is 8 (i.e. 7 threads are spawned), these x values will be used by threads for $y(x)$:

```
x: 1,2,3,4,5,6,7
```

In mode A the following will be displayed (not necessarily in this sequence):

```
1:9 2:0 3:-7 4:-12 5:-15 6:-16 7:-15
```

In mode B the following characters will be displayed:

```
1:+ 2:0 3:- 4:- 5:- 6:- 7:-
```

Besides indicating local or global variables needed, you only need to write the lines of text you would add at the point indicated as:

```
//// CODE BLOCK 1: ADD ADDITIONAL CODE GOES HERE ////
```

Section 2: Multiple Choice [30 marks]

Choose one option for each of the questions 2.1 to 2.4 in this section.

2.1 It is important to design OpenCL programs around the notion of workers and groups. Which option below accurately describes a difference between workers and groups in OpenCL? **[5]**

(a) In OpenCL, all the workers can run different program instructions, but they are all restricted to accessing the same "work-group" of memory – i.e. the 'group' refers to the specific group of memory blocks accessible to the workers.

(b) In OpenCL, all the workers run the same code, but they have different IDs (accessible by means of a special function for each worker); each group is an array of workers, a subset of the total workers on the GPU chip.

(c) In OpenCL, workers run one at a time, and each one can access the same or different group of memory blocks but cannot access memory outside that group.

(d) In OpenCL, 'workers' refer to the processes that run in lock-step, and 'group' simply refers to the group of workers that are involved in the same mutex-synchronisation operation.

(e) None of the above.

2.2 Consider the Verilog code for moduleX (see next page). Select the option that best describes what this module does. **[5]**

(a) This module returns active 1 or active 0 respectively on whether the module has been tasked to count up or to count down.

(b) This is an up/down counter with reset, it counts up on every positive clock when active is high or counts down when active is low.

(c) This is an up counter with reset, it continuously counts up on every positive clock when rst is low.

(d) This is a down counter, on reset high the initial count is set and held; when reset is low it starts counting down with every clock until zero is reached at which point it sets active high.

(e) This module initially sets active high, then counts up from 0 and when the maximum value is reached it sets active to low.

```
// Code for Question 2.2
module moduleX  (
  output reg [7:0]out,    // an output
  output          active, // another output
  input           clk,    // clock input
  input           rst     // reset input
);

// ------ Implementation -------
always @(posedge clk)
  if (rst) begin
    out    <= 8'b0;
    active <= 1'b0;
  end else begin
    out    <= out + 1;
    active <= 1'b1;
  end
endmodule
```

2.3 An SMP computer would, according to Flynn's taxonomy, be classified as which one of these computer architecture types? **[5]**

(a) SISD

(b) SIMD

(c) MISD

(d) DIMS

(e) MIMD

2.4 What is meant by handshaking in a communications system? **[5]**

(a) This is a type of error detection scheme used in high-speed networks.

(b) This refers to initiating an asynchronous data transfer agreement between machine and the user.

(c) This refers to how two devices initiate communication and ensure successful exchange of desired information.

(d) This refers to the parts of a computer design where there are bidirectional data connections.

(e) It merely refers to part of the boot sequence of an operating system whereby the CPU greets and introduces itself to each peripheral in the computer.

2.5 Answer **true** or **false** to each question below (each answer is 2 marks). **[10]**

(a) DeepQA is a natural language processing system.

(b) A GPU is only able to process graphics data, in particular two dimensional matrices, vectors and pixels.

(c) The real-world performance measure is a major telling factor in determining whether or not it was worth the effort to develop a parallel solution.

(d) A GPU tends to support more threads than a CPU.

(e) Amdahl's law states that processing speed always doubles when you double the number of processors.

Section 3: Long Answers [42 marks]

You are tasked to implement a system with the following characteristics:

1. There are 7 input channels of fast analogue to digital (ADC) data, which are handled separately at this point. This data comes from a data-acquisition card that slots into the PCI bus of the computer. This data must be split into frames of constant length (512 samples, in this case). There are some component of house-keeping that must be performed to control the ADC hardware.
2. For each of the input channels, each frame must pass through an FFT.
3. The results from these FFTs are analysed by comparing the relative phases (i.e. subtract the phases element-wise) and comparing the amplitudes (also element-wise) with some constant threshold.
4. The results are displayed to the user by means of a GUI.
5. The GUI contains an interactive graph.

Question 3.1 [22 marks]

Speed is of utmost importance, but your hardware is limited to an 8-core Intel CPU and a single nVidia GPU. For each of the questions below, explain your reasoning.

3.1 (a) How would you split the work across the hardware in order to obtain the optimal performance? **[10]**

3.1 (b) For each of the processing stages, which synchronisation method would you use (for example 'barrier', 'mutex', etc.)? **[8]**

3.1 (c) What programming model (for example 'message-passing', 'shared memory', etc.) would you use for each of the processing stages? **[4]**

Question 3.2 [20 marks]

Say, for argument sake, that the initial prototype (on the hardware above) proves the principle well enough that you are provided with a much larger budget (money, time and man-power) in order to implement the next generation of the system. You want to maximise profits for your company, so you'll need to make some engineering compromises (between performance, ease-of-maintenance and flexibility of the hardware, for instance). In each of the following questions, provide an argument for your design decisions.

3.2 (a) What hardware would you use? Would you consider an FPGA or ASIC, for instance, or even a Xeon Phi? Or would you stick to the CPU / GPU above? Would you make the system free-standing and feed results to a PC over the network, or keep everything in the PC (as above). **[10]**

3.2 (b) Provide a rough overview of how you would implement the system on your choice of hardware. A block diagram would help. **[10]**

END OF EXAMINATION

Appendix A: Formulae

Formulae for Crossbar Tree Networks

	Uniform	General
Processors	d^q	$\prod_{i=1}^{q} d_i$
Bisection Width (links)	$\frac{du^{q-1}}{2}$	$\frac{d_q}{2}\prod_{i=1}^{q-1} u_i$
I/O Links	u^q	$\prod_{i=1}^{q} u_i$
Switches	$\frac{d^q - u^q}{d-u}$	$\sum_{k=1}^{q} U_{q-k-1} D_k,$ $U_j = \frac{\prod_{i=1}^{j} u_i}{u_j}, \quad D_j = \frac{\prod_{i=1}^{j} d_i}{u_j}$

Adapted from page 291 of Martinez, Bond and Vai 2008

Appendix B: MPI Reference Sheet

Summary of Message Passing Functions - Interface Quick Reference in C

```
#include <mpi.h>

// The MPI_Status struct is defined as follows
typedef struct MPI_Status {
  int count;      //
  int cancelled;
  int MPI_SOURCE; // the sender
  int MPI_TAG;    // tag that can be set to indicate message type ID
  int MPI_ERROR;  // non-zero if error
} MPI_Status;

// Note: For MPI_Comm you can just use MPI_COMM_WORLD.
//       For datatype you can simply use the most generic option MPI_BYTE for elements of buf

// Blocking Point-to-Point message passing functions

// Send a message to one process
int MPI_Send (void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm);

// Receive a message from one process
int MPI_Recv (void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status);

// Count received data elements
int MPI_Get_count (MPI_Status *status, MPI_Datatype datatype, int *count);

// Wait for message arrival.
int MPI_Probe (int source, int tag, MPI_Comm comm, MPI_Status *status);

// Non-blocking Point-to-Point message passing functions

// Begin to receive a message
int MPI_Irecv (void *buf, int count, MPI\_Datatype, int source, int tag, MPI_Comm comm, MPI_Request *request);

// Complete a non-blocking operation
int MPI_Wait (MPI_Request *request, MPI_Status *status);

// Check or complete a non-blocking operation
int MPI_Test (MPI_Request *request, int *flag, MPI_Status *status);

// Check message arrival
int MPI_Iprobe (int source, int tag, MPI_Comm comm, int *flag, MPI_Status *status);
```

Based on full reference sheet available from

http://www.netlib.org/utk/people/JackDongarra/WEB-PAGES/SPRING-2006/mpi-quick-ref.pdf

Appendix C: MPI Starting Point Code

```
/*************************************************************
   Hello MPI - a simple starting point for MPI programmes
*************************************************************/

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

/*******************************************/

int main(int argc, char** argv) {
  // Initialize MPI environment
  MPI_Init(NULL, NULL);

  // Determine number of processes started
  int world_size;
  MPI_Comm_size(MPI_COMM_WORLD, &world_size);

  // Get the rank of this process
  int world_rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

  // Get the name of this processor
  char processor_name[MPI_MAX_PROCESSOR_NAME];
  int name_len;
  MPI_Get_processor_name(processor_name, &name_len);

  //// CODE BLOCK: ADD ADDITIONAL CODE GOES HERE ////

  // Finalize the MPI environment.
  MPI_Finalize();
}
```

Appendix D: Verilog Reference

Comments

```
// One-liner
/* Multiple
   lines */
```

Numeric Constants

```
// The 8-bit decimal number 106:
8'b_0110_1010 // Binary
8'o_152       // Octal
8'd_106       // Decimal
8'h_6A        // Hexadecimal
"j"           // ASCII

78'bZ         // 78-bit high-impedance
```

Too short constants are padded with zeros on the left. Too long constants are truncated from the left.

Nets and Variables

```
wire [3:0]w; // Assign outside always blocks
reg  [1:7]r; // Assign inside always blocks
reg  [7:0]mem[31:0];

integer j; // Compile-time variable
genvar  k; // Generate variable
```

Parameters

```
parameter  N     = 8;
localparam State = 2'd3;
```

Assignments

```
assign Output = A * B;
assign {C, D} = {D[5:2], C[1:9], E};
```

Operators

```
// These are in order of precedence...
// Select
A[N] A[N:M]
// Reduction
&A ~&A |A ~|A ^A ~^A
// Compliment
!A ~A
// Unary
+A -A
// Concatenate
{A, ..., B}
// Replicate
{N{A}}
// Arithmetic
A*B A/B A%B
A+B A-B
// Shift
A<<B A>>B
// Relational
A>B A<B A>=B A<=B
A==B A!=B
// Bit-wise
A&B
A^B A~^B
A|B
// Logical
A&&B
A||B
// Conditional
A ? B : C
```

Module

```
module MyModule
#(parameter N = 8) // Optional parameter
 (input  Reset, Clk,
  output [N-1:0]Output);
 // Module implementation
endmodule
```

Module Instantiation

```
// Override default parameter: setting N = 13
MyModule #(13) MyModule1(Reset, Clk, Result);
```

Case

```
always @(*) begin
 case(Mux)
  2'd0: A = 8'd9;
  2'd1,
  2'd3: A = 8'd103;
  2'd2: A = 8'd2;
  default:;
 endcase
end

always @(*) begin
 casex(Decoded)
  4'b1xxx: Encoded = 2'd0;
  4'b01xx: Encoded = 2'd1;
  4'b001x: Encoded = 2'd2;
  4'b0001: Encoded = 2'd3;
  default: Encoded = 2'd0;
 endcase
end
```

Synchronous

```
always @(posedge Clk) begin
 if(Reset) B <= 0;
 else      B <= B + 1'b1;
end
```

Loop

```
always @(*) begin
 Count = 0;
 for(j = 0; j < 8; j = j+1)
  Count = Count + Input[j];
end
```

Function

```
function [6:0]F;
 input [3:0]A;
 input [2:0]B;
 begin
  F = {A+1'b1, B+2'd2};
 end
endfunction
```

Generate

```
genvar j;
wire [12:0]Output[19:0];

generate
 for(j = 0; j < 20; j = j+1)
 begin: Gen_Modules
  MyModule #(13) MyModule_Instance(
   Reset, Clk,
   Output[j]
  );
 end
endgenerate
```

State Machine

```
reg   [1:0]State;
localparam Start = 2'b00;
localparam Idle  = 2'b01;
localparam Work  = 2'b11;
localparam Done  = 2'b10;

reg tReset;

always @(posedge Clk) begin
 tReset <= Reset;

 if(tReset) begin
  State <= Start;

 end else begin
  case(State)
   Start: begin
    State <= Idle;
   end
   Idle: begin
    State <= Work;
   end
   Work: begin
    State <= Done;
   end
   Done: begin
    State <= Idle;
   end
   default:;
  endcase
 end
end
```