EEE4120F

High Performance Digital Embedded Systems

FINAL EXAM

24 May 2019

120 marks, 3 hours

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 **EEE4120F** and a title **Final Exam** on your answer sheet(s). Answer each section on a separate page.

DO NOT TURN OVER UNTIL YOU ARE TOLD TO

Exam Structure

Section 1	Section 2	Section 3	Addenda
Short Answers	Multiple Choice True/False	Design Questions	A: Schematic
[44 marks]	[38 marks]	[38 marks]	B: Verilog Cheat Sheet

RULES

- You must write your name and student number on each answer book.
- Write the question numbers attempted on the cover of each book.
- NB **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 [44 marks]

Question 1.1 [10]

This question seeks to review your understanding of HPEC design processes...

Speech translation technology enables speakers of different languages to communicate. A speech translation system typically incorporates three main technologies: automatic speech recognition (ASR), machine translation (MT) and voice synthesis (TTS). The speaker of language A speaks into a microphone and the ASR module recognizes the utterance and converts it into a string of words, using dictionary and grammar of language A. The MT module then translates this string which is then sent to the TTS module for speech output generation and output.

There is a lack of low-cost wearable speech translation devices. Therefore, you have decided that it would be fun to prototype your own speech translation system that is capable of translating 20 languages in real-time. A friend has offered to take care of the aesthetic design of the final end device.

1.1.1) Would you classify your device as a high performance embedded computing system? Explain why. [3]

1.1.2) Write 2 requirements for each of the sensing and processing subsystems of the system. Use proper requirements language (use complete sentences, and words like 'should' and 'will' to express yourself confidently and clearly.) [4]

1.1.3) You have two main architectural options for the implementation of the translator system: distributed (i.e., networked) or non-distributed (i.e., stand-alone). Which one of these two architectural approaches would you use and why? [3]

Question 1.2 [12]

1.2.1) Engineers often confuse terms when describing the performance of a communications network. Explain the differences between Latency, Bandwidth and Baud Rate. [6]

1.2.2) Effective bandwidth, which takes total latency into account, is a more realistic metric than just the raw bandwidth of a communications channel. Calculate the effective bandwidth of a particular channel with the following parameters: [6]

- Distance = 1km
- Raw bandwidth = 12Mbits/s
- Message site = 10 000 Bytes
- Sending overhead = 100us
- Receiving overhead = 150us

Question 1.3 [12]

A high performance computing developer started a job to optimize a sequential computation for Pavement/Road engineering for the South African national road agency. The Pavement/Road damage model is a numerical integration to infinity problem, which iteratively approximates the pavement damage rate over a given time in years. The pavement damage model algorithm has an internal parallelism which is limited to only 16 possible threads which correspond to the 16 load points of a car tyre configuration. These load points are for one car tyre configuration, however, there exist 150 possible car tyre configurations, thus the damage model becomes an accumulative sum of all these 150 tyre configurations in one single day of a month for a given section of a road (i.e. M2 road, 2KM from Mowbray Shoprite towards Rondebosch PicknPay).

1.3.1) Given that the pavement damage model problem above requires the use of hundreds of cores for one day damage model calculation, long double floating point arithmetic and fused multiply/add operations, advanced vector processing, and shared memory, what kind of compute machine would you use and why? [2]

The sequential damage model algorithm takes 30 seconds to complete during execution. In order to speedup the computation, the developer designed a compute cluster with 150 Intel Xeon Gold processors with 16 cores each, each core running at 3.6GHz clock frequency.

1.3.2) Given that each instance of the pavement/road damage calculation requires 16 cores (i.e. Common industry practice of using processor physical cores not processor threads) and the road damage calculating function for one load is named **road_damage_calc(Input &in, int mpi_rank),** discuss how you would use a combination of OpenMP and MPI code in order to process 150 load configurations. [4]

1.3.3) Given that each of the load damage model (i.e. **road_damage_calc(Input &in, int mpi_rank)** function) takes 8 seconds when running in parallel in the 16-cores machine designed by the developer, how much efficiency is achieved? [4]

The damage model of a pavement/road in a given period in years is the sum of every monthly damage computed and accumulated over a given number of years. The damage model of each month is the input to the damage model of the next month until the last month of the last year (i.e. Damage value of January is the initial value of damage value for February, ...).

1.3.4) Given that the damage model of one month is equivalent to the daily sum of damages for 150 load configurations (i.e. damages are computed once every month), what parallel programming communication patterns can you use to implement the parallel pattern for calculating this pavement damage model in the compute cluster of Intel Xeon Gold Processors mentioned above? [2]

Question 1.4 [10]

The Hough Transform[1] is a well-known and popular algorithm for detecting lines in raster images. It is widely used in embedded applications including in autonomous vehicles for lane detection. Given an input edge detected input image, the transform computes the straight lines that passes through each given edge in the input image and for all the lines, a voting operation is used to count which ones have the same slope and intercept i.e. form a straight line through the edges. A pseudo-code description of the algorithm is shown in the code listing below.

```
**Input:** Input image I with pixel dimensions $I_w$, $I_h$, Hough space dimensions $H_\rho$, $H_\theta$
**Output:** Detected lines $L = \{(\theta_1, \rho_1), \ldots\}$
$H(\theta', \rho') \leftarrow 0, \forall \rho' \in \{1, \ldots, H_\rho\}, \theta \in \{1, \ldots, H_\theta\}$
 **for all** $x \in \{1, \ldots, I_w\}$, $y \in \{1, \ldots, I_h\}$ do
  **if** $I(x, y)$ **is edge then**
    increment $H(\rho'(\theta'), x, y), \theta')$, $\forall \theta' \in \{1, \ldots, H_\theta\}$
  **end if**
 **end for**
$L = \{(\theta(\theta'), \rho(\rho')) \mid \rho' \in \{1, \ldots, H_\rho\} \wedge \theta' \in \{1, \ldots H_\theta\} \wedge$ at $(\rho', \theta')$ is a high
     max. in H$\}$
```

1.4.1) A programmer has developed a C implementation of the algorithm which was found to be extremely slow when tested on a 3.06 GHz P4 platform. In order to improve response time, the programmer transferred the code to a platform that uses the latest multi-core ARM Cortex-A72 chip. However, a degradation in performance was observed instead. Explain the possible reasons for this behaviour. [3]

1.4.2) Analyse the worst-case execution time of the Hough Transform. You may use the Big-Oh notation for your analysis. [5]

1.4.3) Consider an embedded application of the algorithm for lane detection that uses a 4K video camera with a chip that captures the input video stream with 4096 H × 2160 V pixels resolution at 60 fps and 16 bits. What is the minimum size in bytes of the frame buffer required to store a frame? [2]

[1] https://en.wikipedia.org/wiki/Hough_transform

Section 2: Multiple Choice and True/False [38 marks]

Question 2.1 [30]

Answer the following with a letter (a – e)

2.1.1) In the OpenMP code segment shown below, how many iterations are executed if four threads execute this program? [5]

```
#pragma omp parallel private(i)
for (int i = 0; i < 100; i++) {
     a[i] = i;
}
```

a) 100
b) 50
c) 25
d) 0
e) None of the above

2.1.2) Given that the energy dissipation for load/save, add/sub/cmp/branch/mov, multiply/divide instructions are 40nJ and 20nJ and 60nJ respectively. The total energy dissipation for the following assembly program is: [5]

```
  MOV B, #0x0000 ; set B to constant value
LOOP: LD A, [B]  ; read memory
  MUL A, A, C    ; A = A * C
  ST A, [B]      ;save changed value of A back to [B]
  ADD B, #1      ;B = B + 1
  CMD B, #0xDDFF ; compare B to 0xDDFF
  JNE LOOP
```

a) 5242800
b) 3932120
c) 3932100
d) 13107020
e) 11366220

2.1.3) Which one of the following statements is correct: [5]

a) The Cell processor is considered as a homogeneous multicore processor because it has 1 power processing element and 8 synergistic processing elements.
b) A softcore processor is used for compiling software in a personal computer, whereas a hardcore processor is synthesized in FPGA hardware.
c) It is recommended to use blocking and nonblocking assignments within the same Verilog 'always block' to improve FPGA performance.
d) Fine-grained parallel tasks require a great deal of intercommunication relative to the amount of computing they do.
e) Wishbone, Altera's proprietary bus system, supports shared bus topologies and bus widths of different sizes.

2.1.4) Which one of the following is true about Cache coherence: [5]

a) It is when one processor writes a location in a shared variable, all other processors are hindered from accessing it at the same time.
b) It is when one processor writes a location in a shared variable, all other processors are updated.
c) It is when one processor writes a location in a shared memory and all other processors are updated.
d) It is implemented at the software level when using reconfigurable embedded processor.
e) It is only used in small memory capacity requirement.

2.1.5) A compute cluster is often build using three or more compute nodes with high-speed processors such as the Intel Xeon Gold processors with 16 cores at 3.6GHz clock frequencies. Which one of the following is the most accurate reason why a HPC developer for a small company would opt for an NVidia Titan Xp GPU even though it has a clock speed of 1.5GHz?

[5]

a) It is too complicated to parallel program compute clusters.
b) Communication latencies in compute clusters are too high and interconnect suppliers have not made a breakthrough in a while.
c) Compute clusters are very costly, can only be afforded by government funding and huge private companies.
d) This GPU has thousands of CUDA cores, when the parallel code is efficiently optimized to run on them, they can outperform high-speed processors like the Xeon Gold.
e) GPUs are very fast regardless of their clock speed.

2.1.6) The algorithm complexity of algorithm X is given by 5log2(N), for N=number of samples = 2048. Given that 1000 instances of this algorithms are executed in k=1000 processors in parallel, with a uniform communication latency of 10us between the master processor and all these 1000 processors.

How many operations per second (e.g. In GFLOPS) can be achieved by these 1000 processors machine if a single instance of algorithm X takes 0.1ms to execute and consists of adder and multiplier floating-point operations only. [5]

a) 500 GFLOPS
b) 100 MFLOPS
c) 1,000,000,000 GFLOPS
d) 1 GFLOPS
e) 0.5 GFLOPS

Question 2.2 [8]

Answer the following with either "True" or "False"

2.2.1) To benefit from a multiprocessor, an application must be concurrent. [2]

2.2.2) Like SMPs, message-passing computers rely on locks for synchronization. [2]

2.2.3) Three main constraints for embedded computers are power, size, and weight. [2]

2.2.4) There is no such thing as a 64-bit embedded processor. [2]

Section 3: Design Questions [38 marks]

Rick and Morty's Defence System Upgrade - A Design and Verilog Interfacing Challenge

Genius inventor, Rick, and his grandson, Morty, frequently travel through space on wild adventures. Recently the duo has had trouble with angry aliens trying to obliterate them. Being outgunned, the duo can only retreat and on many occasions have only escaped in the nick of time! The ever-confident Rick has had enough of fleeing and wants to upgrade their laser cannon on their spaceship to destroy the enemy aliens (in self-defence, obviously). Currently, their laser cannon is manually operated, inaccurate and frequently overheats. Rick has already created a Target Acquisition System (TAS), that when powered up, scans and locks onto enemy targets. All he needs now is a controller to interface both the TAS and the laser cannon. The always cautious Morty says that this new system needs a safety override module in case things go haywire. Rick reluctantly agrees to add this override system, he also thinks an FPGA will do the trick - but he is a bit rusty with his Verilog and needs your help.

The next paragraph is Rick's explanation of how he wants the controller to function, he has drawn up a schematic of the whole system (**see addendum A**) as a useful reference.

In order to not give their position away by constantly scanning the area, the controller needs to be idle until an 'arm system' switch is flicked. At this point, the controller needs to enable the TAS and wait for it to initialize. Once the TAS has initialized, the system is armed - the TAS scans around, identifies and locks onto enemy spaceships. Once an enemy is locked on, the controller needs to send a 'fire' signal to the cannon, which when fully charged, sends a maximum power laser blast at the enemy. Once a laser blast has been fired, the controller should halt for an adjustable count sequence, which will control the rate of fire. This rate is adjustable from slowest (1023ms) to fastest (1ms) based on the position of a 10-bit digital sliding potentiometer. After halting, the controller should return to the armed state, where it will either fire again depending on the lock on signal or returns to its initial state if the 'arm system' has been switched off. There is also a reset button which must put the controller back into an idle state and reset any signals to their initial values.

The explanation on the previous page indicates that a finite state machine would be the ideal way to implement Rick's Controller. A state diagram is a good place to start when designing FSMs.

3.1) Draw a state diagram for Rick's Controller. [12]

Using your state diagram you drew in Q3.1 as reference, you will now help Rick with his Verilog.

3.2) Implement a one-hot encoded finite state machine for Rick's Controller. [14]

Here is a starting point for your implementation:

```
module controller (clk_100MHz, rst, arm_sys, fire_rate,
                   tas_rdy, target_lock, tas_en, fire_sig);
  // inputs
  input clk_100MHz, rst, arm_sys, tas_rdy, target_lock;
  input [9:0] fire_rate;

  // outputs
  output reg tas_en, fire_sig;

  // TODO: internal register(s), parameter(s) and FSM code
  // *---answer for Q3.2---*

endmodule
```

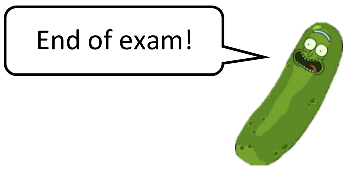
Morty explains that his Override module should allow the `fire_sig` to reach the cannon only if the cannon is not overheating and that the cannon enable switch is on. He says that for the quickest response time, the module should be purely combinational.

3.3) With reference to the above paragraph and addendum A, write the Verilog code for Morty's Override module. [6]

Whilst you were coding, Morty was reading up about FPGAs and how a logic function is mapped to a lookup table (LUT) consisting of memory and a multiplexer.

3.4) Draw a LUT that implements the combinational logic used in Morty's Override module. [6]

Hint: start with a truth table.



Addendum A: Schematic

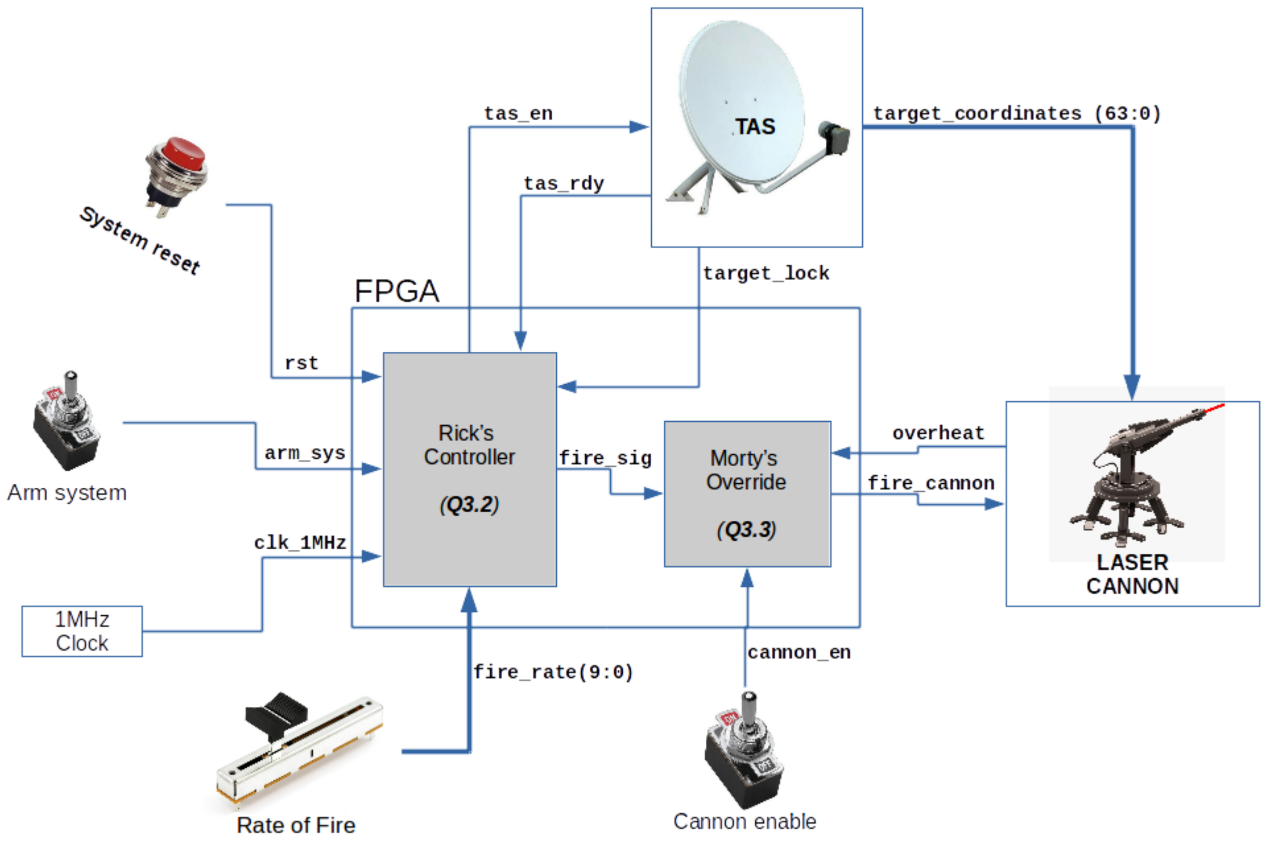


Table 1: Signal descriptions

Signal / Bus	Description
rst	Resets the system.
arm_sys	Signal to power up and arm the system.
clk_1MHz	1MHz clock for the FPGA.
fire_rate	Controls the rate of fire (rof). 10-bit signal: range from 0x001 (Highest rof) to 0x3FF (lowest rof).
fire_sig	Driven HIGH to signal that cannon must fire.
target_lock	Goes HIGH when TAS has locked onto a target.
tas_en	Driven HIGH to start TAS power up.
tas_rdy	Goes HIGH when TAS has completed its power up and initialization phase.
cannon_en	Driven HIGH to enable the cannon.
fire_cannon	When asserted, the cannon fires a laser pulse.
overheat	Goes HIGH when the cannon is overheating.

Addendum B

Verilog Cheat Sheet

S Winberg and J Taylor

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
```