High Performance Embedded Systems EEE4120F

FINAL EXAM

9 July 2021

Venue: New Lecture Theatre, Starting: 08:00

3 hours

Examination Prepared by:
Simon Winberg

Last Modified: 11-Aug-2021

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 to **put your student number on any separate answer sheets!** Also **put your name and student number** and course code **EEE4120F** and title **Final Exam** in your UCT exam answer book (this answer book is mainly for rough work).

FOLLOW THE EXAMINATION PROCEDURES ON THE **PROCEDURES PAGE** ON YOUR DESK

DO NOT TURN OVER UNTIL YOU ARE TOLD TO

Exam Structure

Marked out of 100 marks / 180 minutes. Time per mark = 1min 48sec

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 [30 marks]

Question 1.1 [10 marks]

Large clusters can include possibly hundreds of networked computers. It is now becoming more common to find clusters with heterogeneous nodes, for instance each node of the cluster may be a computer that also has FPGA-based accelerators, if not a graphics processing unit (GPU) as well. Programming such systems may use a combination of Pthreads and the Message Passing Interface (MPI), including specialized compilers and tools such as OpenCL for the GPU and Xilinx Vivado for the FPGA. Answer the following:

Q1.1(a) Why might it be useful to have a cluster with nodes comprising heterogeneous architectures? Why could this provide an overall performance better than nodes of homogeneous architectures? [2 marks]

Q1.1(b) What are some of the difficulties and expenses concerned in developing applications for heterogeneous architectures? Explain using comparisons to traditional clusters of SMP processor architectures, and also motivate reasons why SMP-only clusters are sometimes favored rather than heterogeneous architectures. [4 marks]

Q1.1(c) Pthreads provides a particular, almost intuitive, approach to developing a parallel program for deployment on a computer where the processors are largely all the same. Consider the following situation:

- We have global array variables: int x[N], y[N]; and global int sum=0, where N is a constant positive value under 1024.
- We spawn N threads where thread IDs, a variable *i* in the thread function, ranges from 0 to N-1
- Within each thread instance, i.e. in thread ID *i*, the operation sum+= x[i]*y[i] is performed

But there are problems with the above program design, which likely causes the wrong *sum* value to be calculated when run with more than one thread. Briefly explain what mistakes are being made [2 marks]. And explan why, when it is run using multiple threads, besides giving erroneous answers for *sum*, it is actually slower than the case where a for-loop is simply used to sequentially add up the x[i]*y[i] terms using just one thread [2 marks].

Question 1.2 [10 marks]

Recursion is a programming technique where you call a function inside the same function. You and a friend find this concept very intriguing and create a recursive function that generates the Fibonacci number at a specific position. Pseudo code generating the Recursive Fibonacci function is as follows:

```
function fib(position):
    if(position == 1):
          return 1;
    if else (position == 0):
          return 0;
    else:
          return fib(position - 1) + fib(position – 2)
```

After considering some parallel programming, your partner says that this function will be perfect to implement using parallel techniques. But what do you think? Please answer the following questions.

Q1.2. (a)
Can you implement this recursive function in parallel? If yes explain how and whether its a good idea. If no explain why not. [4]

Q1.2 (b)
After some practice with OpenCL you and your friend decide to take a look at recursion again but this time you are making a function that calculates the factorial and multiplies the factorials together.

Pseudo code for the factorial function:

```
func factorial(position, power):
    if(position == 1):
        return 1;
    else:
        return (position ^ power) x factorial(position - 1, power)
```

Example Output:

factorial(5, 3) => 1^3 x 2^3 x 3^3 x 4^3 x 5^3 = 5! x 5! x 5! = 1728000

Answer this question for Q1.2.2:

Is this algorithm highly *parallelizable*? [1]

Q1.2 (c)

After some careful thought your friend says that he can run this algorithm for multiple powers and sizes in parallel and add them together. As an example:

factorial(3, 1) + factorial(4, 2) + factorial(5, 3) = 6 + 576 + 1728000 = 1728582

If you were to implement this algorithm using openCL how would you setup the memory management, with you work items and work groups? No code is required, but emphasis should be placed on efficiency, and you can use the example above to aid in your explanation. [5]

Question 1.3 [10 marks]

Answer the following questions and sub-questions concerning computation approaches…

(a) Many RC platforms are built using FPGAs. But they often also include a PLA or CPLD in addition to the FPGA on the platform. Explain the following…
i. What is the main difference between an FPGA and a PLA? [1 marks]
ii. Explain why a PLA (or a CPLD) in a reconfigurable computing system is often needed in the configuration architecture / glue logic whereas FPGA(s) are more commonly used for the actual computation (e.g., signal processing) operations. [2 marks]

(b) They say that an FPGA gives around a tenth of the performance (if that) of an ASIC. What is an ASIC? [1 mark] Explain why an FPGA (e.g. a softcore RISC-V processor) is generally not going to be as fast as an ASIC implementation (e.g. a RISC-V processor in silicon)? [3 marks]

(c) Contrast some of the benefit offered by using parallel code over sequential code for an embedded microprocessor-based or microcontroller-based solution. [3 marks]

Section 2: Multiple Choice [35 marks]

NOTE: Choose only one option (i.e. either a, b, c, d or e) for each question in this section.
Questions are 5 marks each.

Q2.1 Which one of the following options is not an abstraction level?

(a) Gate Level
(b) Switch Level Abstraction
(c) Behavioral Level
(d) Dataflow Level
(e) Syntax Level

[5 marks]

Q2.2 An embedded system development company typically encounters two general types of costs related to their product manufacture. Which one of these is an example of "non-recurring costs"? (choose only one option) …

(a) Batteries for the platforms
(b) Maintenance costs
(c) Components and wiring costs
(d) Research and development costs
(e) Postage on returns

[5 marks]

Q2.3 Which component does the Harvard Architecture have that the von Neumann Architecture does not have?

(a) Arithmetic Logic Unit
(b) Input Ouput
(c) Instruction Memory
(d) Memory
(e) Control Unit

[5 marks]

Q2.4 Which of the following statements is false, with regards to the OpenCL programming model?

(a) A kernel is a basic unit of execution.
(b) Workgroups are work-items that execute together on one device.
(c) Work-items can communicate directly with each other.
(d) Each work-item has its own small amount of private memory.
(e) Workgroups can take place simultaneously or via a specific schedule.

[5 marks]

Q2.5 Which of the following properties does not apply to a standard PIC?

(a) Has some very low-cost versions
(b) Most are low power consumption
(c) Is a type of microcontroller
(d) Uses Harvard or modified Harvard architecture
(e) All models are 32-bit.

[5 marks]

Q2.6 Which of the following memory partitioning models are used in the program shown below? (just provide one of the letters corresponding to your choice). Assume an image gets successfully loaded into the image object.

[5 marks]

(a) Contiguous
(b) Partitioned
(c) Interlaced
(d) Haphazard
(e) Interleaved

```
global JPEG image;  // an image of image.sizex by image.sizey pixels
global nth = 4;
function threadfn ( int arg ) {
  int y = arg;
  while (y<image.sizey) {
    for int x=0 to image.sizex-1 {
     Pixel t = image.pixel[x,y];
     image.pixel[x,y] = image.pixel[image.sizex-x-1,y];
     image.pixel[image.sizex-x-1,y] = t;
    }
    y = y + nth;
  }
} // end function

function main () {
  image.load("happy.jpg");
  thread0 = create_thread(threadfn,0); thread1 = create_thread(threadfn,1);
  thread2 = create_thread(threadfn,2); thread3 = create_thread(threadfn,3);
  wait(thread0);wait(thread1); wait(thread2);wait(thread3);
  showimage(image);
}
```

Q2.7 These are **T**rue / **F**alse answer options, put in a T or a F using the Answer Sheet for your answer to each of these questions (each question worth 1 mark).

(i) The 'golden measure' refers to a trusted baseline implementation of an algorithm to be parallelized (or implemented on an specialized architecture).
(ii) A correlation between two identical data sets returns the value 0.
(iii) A DWARF relates to a type of common processing pattern found in HPEC systems.
(iv) The commonly used HPEC acronym “SWAP” means Stop Wait And Process.
(v) DRYSTONES is related to iterations of a performance algorithm rather than a measure of operations per second.

[5 marks]

Section 3: Long Answers [35 marks]

This section involves one rather lengthy answer and one not-so-lengthy answer

Question 3.1 : Huge Calculator With Tiny Battery Production Line [20 marks]

(an assessment of your thinking of parallel systems… without you needing to do any programming)

Imagine that you are an engineer working for a development company called Amalgamated Constructors of Meaningless Equipment (or just ACME for short). The company produces a variety of widgets – for which there is high demand, mainly as corporate gifts. These widgets include: huge desk calculator (with tiny non-standard built-in batteries designed to go flat after a year), colorful USB hubs (version 1.0 USB naturally), pens with chrome parts and dried out ink, and laptop carrier bags that are just a few millimeters too small for a full-size laptop. Business has been booming! It is imperative that the production lines are enhanced. You, with your knowledge of parallel programming and electronics (since they could not find a systems engineer), have been tasked to streamline part of the Huge Calculator (HC) with tiny non-standard battery production line.

The parts of the calculator and how they need to be put together are described below[1]. There are two types of molds[2] used in the HC production: *top mold* for the top of the calculator and *bottom mold* for the bottom of the calculator. The stages for producing the top and bottom mold are as follows:

Stage	**Duration**
Top mold process	
Clean mold and treat with anti-stick chemical.	2 min.
Wait for flammable ingredients to evaporate (safety requirement).	2 min
Heat mold (to avoid plastic cracking when liquid plastic poured onto cold surface).	4 min
Move mold to pourer.	1 min
Pour plastic into mold. (slow because of complicated shape).	3 min
Allow mold to set.	2 min
Extract shape from mold.	2 min
Bottom mold process	
Clean mold.	2 min
Wait for flammable chemicals to evaporate (safety requirement).	2 min
Heat mold.	4 min
Move mold to pourer (this station is further away).	2 min
Pour plastic into mold. (can be poured quickly as shape is simple).	2 min
Allow mold to set. (takes longer because shape has more plastic).	4 min
Extract shape from mold. (simpler than top mold).	1 min
Finishing (can only be done once the top and bottom molds are ready for joining*)*	
Assembling calculator: Once a pair of top and bottom molds is ready, the other components (preassembled PCB, LCD and tiny battery, that are immediately available in the store) are incorporated, and the top and bottom molds are screwed together. The product is then ready to be sent to the boxing station.	15 mins

There are 12 workspaces for assembling a calculator. Each workspace can assembly only one at a time.

[1] You can of course think of the stages of the factory line as steps in a program, and the stages each having a pool of limited worker threads (equivalent to resources) that can be used in each stage.

[2] I realize that 'mold' may be US spelling, assume it is 'mould' in UK spelling, i.e. meaning *a container* and <u>not</u> a fungus!

Resources: there is a large oven for the Heat Mold step that can take up to 8 molds of either type. Three top molds can be poured at a time and four bottom molds can be poured at a time. Top and bottom molds can be poured simultaneously but you cannot poor from a top mold pourer into a bottom mold (or vice versa). There is plenty of space for putting molds out to dry (to set solid) and for workers (or robots) to extract the molds. There are only 12 workstations for assembling the calculators. There are plenty of molds.

Clearly this is similar to a parallel programming problem. Answer the following:

(a) Plan an efficient pipeline-based production line for the Huge Calculator using the available resources indicated above and the timing constraints for the stage explained above, indicate the amount of parallelism that can be achieved in the different stages. [8 marks]

(b) With the existing resources (listed above) indicate what production rate can be achieved – indicate the maximum number of calculators per hour and the rate of top and bottom molds worked on in the different stages. Assume the number of employees is not constrained. [7 marks]

(c) If the pourer and ovens cost the most (and thus the company wants to add minimally to this), but the rest of the production line is cheap, propose how the production line could be optimized to get the best production for the lowest cost with bottlenecks minimized? [5 marks]

Question 3.2 Verilog Coding [15 marks]

The ACME Company is planning a prototype version of their Huge Calculator. This simplified form is not going to implement any of the fancy functions such as square-root or divide. It will just do basic operations using 32-bit integers (i.e. no decimals). How the system works is that one punches in a number (e.g. 10) and then presses one of the math function keys (only '+' '-' 'x' and '=' works). Each time you press a function key it displays the result of the previous operation. You need not worry about reset, assume regs 0 at start.

Example scenarios of operation and what is displayed are as follows:

Keypresses	*Display*
❶❼	17
+	17
❷	2
=	19

Thus, when a math key is pressed the last entered value (LEV register) and the accumulator (ACC register) are applied to the math function and the result is saved to the ACC and displayed on the screen, by copying the value of register ACC to the display register (DSP). (i.e. assume the DSP register is already connected to the LCD display so that the LCD will display whatever is in the DSP register).
The math operations are involved by pressing keys '+', '-', 'x', and '=' which respectively results in signal lines add, sub, mul or eq being raised high for a few microseconds and returning to 0V. Assume hardware debounce is already dealt with so you don't need to write code to accommodate this.

The function code provided in the Answer Sheet for this question gives you a starting point, please complete it to implement the above functionality given the operation of the inputs and outputs as explained above.

(you can assume this is the starting point of your solution, you don't have to rewrite this part, but if you want to add any input/output parameters remember to indicate which are added and where)

```
module hugecalc (lev, acc, dsp, add, sub, mul, eq);
  /* FPGA prototyped version of the Huge Calculator */
  input  lev[31:0]; /* last entered value */
  inout  acc[31:0]; /* accumulator        */
  output dsp[31:0]; /* display register   */
  input  add,sub,mul,eq; /* the math function buttons */

  /* add your code here: */









endmodule
```

END OF EXAMINATION

Appendix A: Verilog Cheat sheet

Numbers and constants

Example: 4-bit constant 10 in binary, hex and in decimal: 4'b1010 == 4'ha -- 4'd10

(numbers are unsigned by default)

Concatenation of bits using {}

4'b1011 == {2'b10 , 2'b11}

Constants are declared using parameter:

parameter myparam = 51

Operators

Arithmetic: and (+), subtract (-), multiply (*), divide (/) and modulus (%) all provided.

Shift: left (<<), shift right (>>)

Relational ops: equal (==), not-equal (!=), less-than (<), less-than or equal (<=), greater-than (>), greater-than or equal (>=).

Bitwise ops: and (&), or (|), xor (^), not (~)

Logical operators: and (&&) or (||) not (!) note that these work as in C, e.g. (2 && 1) == 1

Bit reduction operators: [n] n=bit to extract

Conditional operator: ? to multiplex result

Example: (a==1)? funcif1 : funcif0

The above is equivalent to:

```
((a==1) && funcif1)
|| ((a!=1) && funcif0)
```

Registers and wires

Declaring a 4 bit wire with index starting at 0:

```
wire [3:0] w;
```

Declaring an 8 bit register:

```
reg [7:0] r;
```

Declaring a 32 element memory 8 bits wide:

```
reg [7:0] mem [0:31]
```

Bit extract example:

r[5:2] returns 4 bits between pos 2 to 5 inclusive

Assignment

Assignment to wires uses the assign primitive outside an always block, e.g.:

```
assign mywire = a & b
```

Registers are assigned to inside an always block which specifies where the clock comes from, e.g.:

```
always@(posedge myclock)
   cnt = cnt + 1;
```

Blocking vs. unblocking assignment <= vs. =

The <= assignment operator is non-blocking (i.e. if use in an always@(posedge) it will be performed on every positive edge. If you have many non-blocking assignments they will all updated in parallel. The <= operator must be used inside an always block – you can't use it in an assign statement.

The blocking assignment operator = can be used in either an assign block or an always block. But it causes assignments to be performed in sequential order. This tends to result in slower circuits, so avoid using it (especially for synthesized circuits) unless you have to.

Case and if statements

Case and if statements are used inside an always block to conditionally update state. e.g.:

```
always @(posedge clock)
  if (add1 && add2) r <= r+3;
  else if (add2) r <= r+2;
  else if(add1) r <= r+1;
```

Note that we don't need to specify what happens when add1 and add2 are both false since the default behavior is that r will not be updated. Equivalent function using a case statement:

```
always @(posedge clock)
  case({add2,add1})
  2'b11  : r <= r+3;
  2'b10  : r <= r+2;
  2'b01  : r <= r+1;
  default: r <= r;
endcase
```

Module declarations

Modules pass inputs, outputs as wires by default.

```
module ModName (
  output reg [3:0] result,  // register output
  input [1:0] bitsin,  input clk, inout bidirectnl  );
 … code …
endmodule
```

Verilog Simulation / ISIM commands

```
$display ("a string to display");
$monitor ("like printf. Vals: %d %b", decv,bitv);
#100   // wait 100ns or simulation moments
$finish  // end simulation
```