# High Performance Embedded Systems EEE4120F

**FINAL EXAM MEMO!**

***10 June 2024***

***Venue: LS2B  Time starting: 08h00***

***3 hours  MEMO!!!***

*Examination Prepared by:*
*Simon Winberg*

*Last Modified: 01-Jun-2024*

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



*(hint: consider taking a quick look at Q3.1 and Q3.2 to help you plan your answering)*

**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 [20 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]

*Answer:* (a) Expected to discuss the affordances that a heterogeneous architecture provides: some processors that are suited to course-grained parallel solutions (e.g. SMP and GPUs, e.g. SIMD allowing the same operation to run on multiple data) and others suited to fine-grained parallel solutions, such as FPGAs doing many different computing tasks at the same time.

**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]

*Answer:* (b) Difficulties that the programmers are likely to need expertise in using a variety of programming tools, such as CUDA, pthreads, MPI, etc., which could make it more difficult and expensive obtaining staff to develop applications, and also could cause lengthier development times, such as developers needing to learn tools in order to leverage the available processing power of the cluster. Maintaining the cluster may be more challenging and expensive as well, for example being able to service the variety of architectures from SMP to FPGAs. Additionally there can also major difference between the application design approaches, and programming paradigms, for example the way of thinking about SMP-based application design and use of pthreads is quite different to the design thinking for Cell processors and FPGAs. Furthermore considerations for specialized I/O system and performing I/O on the different architectures could be quite different and more difficulty to accomplish than when using only standard programming languages such as C and Python.

**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 due to a race condition on the shared sum variable. Briefly explain what this problem is and how it can lead to incorrect results in this case [2 marks].

In addition, explain why, when run using multiple threads, this code actually ends up slower than using a single-threaded for-loop to sequentially add up the x[i]*y[i] terms. Consider factors such as thread management overhead, synchronization overhead, and memory contention in your explanation [2 marks].

*Answer:* (c) The program is not using adequate semaphores to protect the value of sum that is being written by multiple threads which are contending on the write access to sum. In some cases multiple threads may have read a same value from sum, ending up with stale values being overwritten and thus calculating the wrong final value. It is also slower due to the overhead of starting the threads and context switching.

## Question 1.2 [10 marks]

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

(a) Reconfigurable Computing (RC) platforms are constructed using FPGAs among other processing components. They often also include a PLA or CPLD in addition to an 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]

*Answers:* 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 marks]**
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 marks]**

(b) It is often stated that an FPGA gives around a tenth of the performance (if that) of an ASIC. But what is an ASIC? [1 mark] Explain why an FPGA is generally not going to be as fast as an ASIC implementation; you can consider a case of comparing a softcore RISC-V processor (implemented in HDL running on a FPGA) to a RISC-V processor implemented in silicon that are both running the same processing tasks. [3 marks]

*Answers: Q1.3 (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 marks]**

(c) Contrast the benefit of using parallel code over sequential code for an embedded microprocessor-based or microcontroller-based solution. [3 marks] (marking: 1 mark for explaining a significant benefit, and 2 marks for a clear motivation for why you consider it a significant benefit).

*Answers:* Q1.3 (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 marks]**

# Section 2: Multiple Choice [50 marks]

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

Q2.1 An embedded system development company often encounters two general types of costs related to their product manufacture. Which one of these is an example of "non-recurring costs"? (choose one option):

a) Battery purchase, as this has become a dominant factor in the production of new systems.

b) Maintenance costs.

c) Postage on returns.

d) Research and development costs. ← 

[5 marks]

Q2.2 You might remember some discussion about the term baud rate. Sentence below most accurately describes what baud rate is? Consider each sentence starts with The baud rate is …

a) … the number of bits transferred in the communication channel each second.

b) … the speed that a packet of data is encoded for sending over the communication channel.

c) … the symbol rate used for sending information over the communication channel. ← answer

d) … the time period after which particles of the conveying medium decay and lose information.

[5 marks]

Q2.3 In the first term, the paper "The Landscape of Parallel Computing Research: A view from Berkeley" was presented. It was highlighted that for processor cores "small is beautiful". Which of these statements accurately reflects the article's advice on multi-core processor architecture design?

a) A small number of big cores should be used as they are much better at doing many small tasks.

b) Many small cores tend to give better performance per unit area than a few big cores taking an equivalent area. ← answer

c) A small processor gives better performance, so processor design should always be kept simple.

d) A big cluster of small cores will almost always run faster, for high granularity problems, than a small cluster of big cores.

[5 marks]

Q2.4 Which one of the following memory partitioning models is used in the program shown below? (Choose just one letter for your choice). Assume an image gets successfully loaded into the image object.

(a) Partitioned
(b) Interleaved
(c) Haphazard
(d) Interlaced ← answer

[5 marks]

(… code on next page …)

```
#include <thread1>  // thread library
#include <Image>    // image library

using namespace std;

JPG_Image image;  // image object of image.sizex by image.sizey pixels
const int nth = 4;

void threadfn ( int arg ) {
  int y = arg; // arg indicates starting y coord
  while (y<image.sizey) {
    for (int x=0; x<(image.sizex-1)/2; x++) {
     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

int main () {
  int i;
  thread threads[nth]; // array of threads
  image.load("happy.jpg"); // load an image from file
  for (i = 0; i < nth; ++i)
     threads[i] = create_thread(threadfn,i);
  for (i = 0; i < nth; ++i) join_thread(threads[i]);
  showimage(image);
  return 0; // program completed
}
```

Q2.5 The concept of load balancing is often discussed in terms of parallel computing. But what is load balancing in relation to parallel computing? Select the best description of load balancing below:

(a) It is ensuring precise synchronization between tasks to ensure task order is maintained.
(b) It is distributing work to minimize idle time. ← 
(c) It is maintaining fair access to shared resources to avoid deadlock or excessive wait times.
(d) It is regulating equal processor allocation between threads.

[5 marks]

Q2.6 Consider that it has been decided to implement a state machine using one-hot encoding for the state index. But what does this mean? Select the best option below explains this encoding method.

(a) One-hot encoding is potentially wasteful on bits, but it can enhance robustness (e.g. tolerance to radiation) whereby valid state indexes have only one bit set and all other bits clear. ← answer

(b) It is an encoding scheme by which the state index will not have more than one bit change when changing from one state to another state.

(c) One-hot encoding is a scheme by which the sum of all bits in a state adds up to a prime number (if this is not the case, then the state is invalid).

(d) It is a state machine design by which only one state does any processing or result generation.

[5 marks]

Q2.7 Which one of the following algorithms is the most course-grained when parallel processing a NxN pixels 8-bit grayscale image that is sitting in shared memory accessible by multiple processors?

a) Perform a horizontal image flip (i.e. for x=1:N for y=1:N tmp = img[x,y]; img[x,y]=img[N-x+1,y]; img[N-x+1,y]=tmp; end; end;)

b) A mean image filter that, for each pixel, calculates the average grey scale value from that pixel from the surrounding 5x5 pixels.

c) Find the location of the brightest pixel on each row and see if that pixel is also the brightest pixel on the corresponding column.

d) Reduce the value of each pixel in an image if its value is above 10. ← answer

[5 marks]

Q2.8 The acronym DDR-SDRAM might have been used to describe a type of memory or memory module for an embedded system? What does DDR-SDRAM refer to if it is actually a valid term.

(a) This is not a valid term.

(b) Double-Data Rate Synchronous Dynamic Random Access Memory. ← answer

(c) Data-Debounced Regulated Standard Dynamic Random Access Memory.

(d) Dual-Data Regulated Symmetric Dynamic Random Access Memory

[5 marks]

Q2.9 Which one of the following statements is correct? (choose only one answer).

(a) A softcore processor cannot be placed on an FPGA, as the propagation delays on FPGAs are excessively lengthy and will lead to instructions not being able to complete on time.

(b) Daisy chaining is a bus architecture for capturing time of data transfers.

(c) Fine-grained (compared to course-grained) parallel tasks require a great deal of intercommunication relative to the amount of computation they do. ← answer

(d) It is necessary to use the same number of blocking assignments as nonblocking assignments within the same Verilog 'always block'.

[5 marks]

Q2.10 Select True (T) or False (F) for the following statements to indicate if the statement is true or false?

| # | Statement | ANSWER |
|---|---|---|
| i | We used OpenCL in the pracs; the acronym OpenCL stands for "Open Compile Language" | N (it actually stands for Open Computing Language) |
| ii | In Verilog, non-blocking assignments ('<=') are for sequential logic, ensuring all assignments within the same 'always@' block occur one after the other. | N |
| iii | FPGAs incorporate a routing network for connecting up PLBs. | Y |
| iv | Access to network resources (e.g. database on different machine) is considered a software bottleneck in many modern high-performance systems. | Y |
| v | The term 'embarrassingly parallel' is used to refer to a very course-grained processing operation that is easy to make parallel. | Y |

[5 marks]

# Section 3: Long Answers [30 marks]

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

## Huge Calculator With Tiny Battery

*(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.

You need to think about the Verilog coding and testing of it and respond to the questions that follow.

### Question 3.1 Verilog Module Coding [20 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. A reset is needed, that resets things, i.e. sets regs to 0.

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).
Note that you do not need to worry about capturing the number input, there is a separate module (called *levin*, which you need not worry about connecting up) that senses number key presses and updates the LEV register (so, if you press reset, LEV is set to 0; then pressing '1' LEV is set to 1, then pressing '2' causes LEV to be multiplied by 10 and have 2 added, resulting in LEV=12. And so on as per normal use.
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  reset;     /* reset the 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: */
```

SOLUTIONS:

*A solution such as the below would suffice:*

```
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: */
 always@ (posedge add or posedge mul or posedge eq)
 begin
   if (add == 1) begin
      acc = acc + lev;
      dsp = acc;
   end
   else if (sub == 1) begin
      acc = acc - lev;
      dsp = acc;
   end
   else if (mul == 1) begin
      acc = acc * lev;
      dsp = acc;
   end
 end // always@

endmodule
```

## Question 3.2 Verilog Test Bench Coding [10 marks]

Although it may seem utterly clear that the calculator isn't going to do anything fancy, and will surely work fine (especially considering that the keypad is designed to be clunky and unpleasing to use). But your team manager wants to be sure… and wants it tested in simulation using a reasonable test case. Hence, you need to put together a testbench for hugecalc (you can call it hugecalc_tb). Check with the scenario 10 + 2 * 5 = … and check that you get the anticipated result.

SAMPLE SOLUTION!!! (much thanks to ChatGPT for assisting with this code … I don't expect students to provie such a complete and detailed solution, they just need to test the interface to the hugecalc to see that the responses between +, * and = are all correct.. and not calculating 10+(2*5) = 20 but (10+2)*5 = 60 as would be the case for a basic calculator such as this scanario)

```
module hugecalc_tb;

// Inputs
reg [31:0] lev;
reg add, sub, mul, eq, rst;

// Outputs
wire [31:0] acc;
wire [31:0] dsp;

// Instantiate the hugecalc module
hugecalc uut (
    .lev(lev),
    .acc(acc),
    .dsp(dsp),
    .add(add),
    .sub(sub),
    .mul(mul),
    .eq(eq)
);

// Initialize the test
initial begin
    // Initialize inputs
    lev = 0;
    add = 0;
    sub = 0;
    mul = 0;
    eq = 0;
    rst = 0;

    // Wait 100 ns for global reset to finish
    #100;

    // Reset the calculator
    rst = 1;
    #10;
    rst = 0;

    // Scenario: 10 + 2 * 5 =
    // Step 1: Enter 10
    lev = 10;
    add = 1;
    #10;
    add = 0;
    #10;

    // Step 2: Enter 2
    lev = 2;
    add = 1;
```

```
        #10;
        add = 0;
        #10;

        // Step 3: Press '=' to update the accumulator
        eq = 1;
        #10;
        eq = 0;
        #10;

        // Step 4: Enter 5
        lev = 5;
        mul = 1;
        #10;
        mul = 0;
        #10;

        // Step 5: Press '=' to get the final result
        eq = 1;
        #10;
        eq = 0;
        #10;

        // Check the result
        $display("Final result: %d", dsp);

        // Stop simulation
        $stop;
    end

    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
```