



High Performance Embedded Systems EEE4120F



FINAL EXAM MEMO!

30 May 2023

Venue: LS2B Time starting: 12h30

3 hours MEMO!!!

Examination Prepared by:
Simon Winberg

Last Modified: 05-Jun-2023

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

<p><u>Section 1</u></p> <p>Short Answers (2 x 10-mark questions) [20 marks]</p> <p>pg 2</p>	<p><u>Section 2</u></p> <p>Multiple choice (10 x 5-mark questions) [50 marks]</p> <p>pg 4</p>	<p><u>Section 3</u></p> <p>Long Answers (2x question design/coding) [30 marks]</p> <p>pg 7</p>	<p><u>Appendices</u></p> <p>A: Verilog cheatsheet</p> <p>pg 9</p>
---	---	--	---

(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.
- 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]

These questions relates to some HPES terminology and concepts.

- (a) Explain what is meant by the concept of speed-up, and the basic formula by which this is calculated in relation to T_{p1} (which we can call the non-optimized program) and T_{p2} (the optimized program) {2 marks}. There was also talk about a warmed-up performance comparison and a non-warmed-up performance comparison – but surely this was just for unusual or problematic situations, like trying to make tea from a cold kettle during loadshedding. Explain whether or not there is actually any point in doing performance comparisons for a non-warmed-up situation. {3 marks}

[5]

Speedup is a number for expressing the relative performance of two systems processing the same problem or implementing the same processing solution (either ‘same problem’ or ‘same processing solution’ would be marked as correct, although generally, and I know it’s a somewhat philosophical view, it would be better saying the same problem or solution, for which we want to get similarly correct results out, but we don’t mind what the specific processing is; for example sequential and parallel processing implementations that solve a given problem would likely be different, thus it is more correct to “say solving the same problem” as apposed to “doing the same processing”. Speed-up is thus a value T_{p1}/T_{p2} indicating how much faster an adjusted or optimized solution, called p1 which takes T_{p1} seconds to run, is compared to the non-optimized solution called p2 which takes T_{p2} seconds to run. [2]

(marking note: such a long and discursive answer is not expected, just the final sentence would do.)

A warmed-up comparison is one in which a program p is run on the data d one or more times before the run that is measured to calculate the T_p time for use in speed comparisons – this is done so that data d gets into memory from the harddrive and into cache, thus reducing this time from the second run. In the case of a non-warmed-up testing there is no prior run of p on d ; rather the testing will include the time of getting data from the input sources and into cache for being worked on. [3]

- (b) Benchmarking has been discussed frequently in this course. Indeed, often mention the saying “don’t lose sight of the forest for the trees...”. At first glance that may seem confusing, we’re not usually working on computers in a forest. But as you know now this is of particular relevance to development focus and effort, what benchmarking objectives have been set and what is being benchmarked. Elaborate further on this point, for example being ecstatically pleased about spending months having achieved a few percent speedup; but how speedup is not necessarily the only thing that is benchmarked and worth development effort.

[5]

There are multiple aspects to benchmark, and this question aims to check that the student is aware of this factor, that it is not only the speed that is of consideration. For example, items to benchmark include:

- Compiler (e.g. efficiency of converting high level language to assembly or machine code).
- The Processor (e.g. efficiency of instructions)
- Operating System (e.g. interrupt latencies, overhead of OS calls, limit of number drivers, etc.)
- Platform (e.g. scalability of memory. peripheral limits, supported interfacing standards)
- Power use (e.g. power saving features)
- Applications (e.g. running DRYSTONE and other software benchmarks)

[5]

Question 1.2 [10 marks]

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

- (a) You hopefully remember Flynn's taxonomy. He writes about four forms of computer architecture, the first three having the acronyms: SISD, SIMD and MIMD. But what is the acronym and full name of that missing fourth form? {2 mark}. What does SISD stand for and why is it commonly considered that this SISD form is the classic and also simpler computer architecture design {2 marks}.

[4]

The missing one is MISD for "Multiple Instructions Single Data". Which is the rather unusual style of architecture design where all the cores run different operations but work on the same data (as in the data that is sent or received from the processor). [2]

SIMD stands for "Single Instruction Single Data", this is a simpler type of processor to design as it has a single item of data (e.g. accumulator) that is worked on by the processor in each clock. In such processors there is only one CPU, no need to design around inter-processor coordination and data transfer, or sharing memory among multiple processors and dealing with consequential challenges such as coordinating access to shared memory address and data busses. [2]

- (b) The concept of "maximum effective parallelism" was discussed in term 1 in regards to parallel processing of multi-core processors, where each core is a processor that runs instructions (as opposed to specialized or dynamically reconfigurable accelerator hardware). Explain what is meant by "maximum effective parallelism" in this case and its relevance to designing programs and to selecting multi-core computing platforms.

[4]

The concept "maximum effective parallelism" refers to the point of the number of cores / amount of parallelism beyond which additional parallelism provides no further benefit or (worse) performance may start to reduce as further processing cores are added. Its relevance to designing programs is useful in deciding either the *viability* or *limit* of parallelizing a solution (e.g. if the problem limits the parallelism of a solution, for example like the Fibonacci series, you can't really have a parallel solution due to each step having a data dependence the previous step) which would also help in deciding if there is benefit to a multi-core solution or the limit to the number of cores for a multi-core solution. If the parallelism is low, this may suggest the algorithm in consideration may not be worth the effort to develop a parallel solution or the expense of using a parallel compared to sequential processing platform. [4]

- (c) The performance metrics FLOPS and MOPS have various similarities; they both measure a count of operations that a processor can complete within a certain period. Generally, a MOPS measure is useful for characterizing a processor. FLOPS is sometimes of limited use to selecting a processor, but at other times it may be the crucial factor for choosing a processor. Briefly explain why this is; and also why we can't simply just rely on MOPS measures for processor selection.

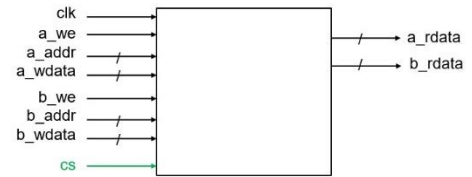
[2]

FLOPS stands for "Floating Point Operations Per Second" and MOPS for "Million Operations per Second". So, first off, the one metric (MOPS) refers to millions of operations per second whereas the other is not in terms of millions, but that is not really a relevant issue besides saving some numbers in discussing speed. FLOPS may be of limited benefit if the processing needs do not involve much floating point calculations, in which case it may be more important to get an measure of how fast general operations that the processor can handle are executed at; this become more useful if you are comparing processors that might not all have floating point processors and the speed at which the processor can run general (non-floating point) instructions is of greatest relevance in benchmarking. But some applications are very dependent on floating point operations, and a processor that does not support floating point tends to carry out these operations much more slowly than one which does, hence relying only on MOPS can be a poor measure of comparison between processors for applications in which the fast execution of floating point operations is imperative. [2]

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 Consider the block diagram of a memory module on the right that can be access in Verilog. It has separate ports for the read and for write ports, in addition to separate read and write data ports, which together allow reading and writing of different addresses in the same clock. But what if we want to refer to something like this quickly, as it takes long to explaining this in text. What is the standard term we use to refer to this type of memory?



Explanation of ports:

clk : clock input
a_we, b_we: write enable/read enable for a,b
a_addr, b_addr : address for channel a and b
a_wdata, b_wdata : data to write for a,b if we=1
a_rdata, b_rdata : data read for a,b if we=0
cs : chip select (i.e. chip ignores inputs if cs=0)

Select only one option below:

- a) Flash memory with bidirectional access.
- b) Duel memory with full-duplex access.
- c) Half-duplex double port access memory.
- d) Full-duplex dual-port memory.**

[5 marks]

Q2.2 You might remember some discussion about the term used to measure the symbol rate of communication. Which of these terms refs to symbol rate, as opposed amounts of data transferred?

- a) Data rate
- b) Signaling rate
- c) Baud width
- d) Baud rate**

[5 marks]

Q2.3 Which one of the following algorithms is the most fine-grained when parallel processing a NxN pixels 8-bit grayscale image that is sitting in shared memory accessible by multiple processors?

- a) Reduce the brightness of each pixel in an image that is brighter than pixel value 120.
- 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) 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;).

[5 marks]

Q2.4 Consider you are working at a development company and you were asked to look over some legacy code to make sense of the type of communication is used. You consistently find that any task sending a message to another task will go ahead and send the message and immediately proceed with other work. Accordingly, what type of communication would you say is being used in the system (select one option):

- (a) Through
- (b) Synchronous
- (c) Asynchronous
- (d) Blocking

[5 marks]

Q2.5 When planning the design of a state machine, it may be decided to use one-hot encoding for the state index? What does this imply? Select the best option below that indicates what this encoding involves.

- (a) A coding scheme by which the state index will not have more than one bit change when changing from one state to another state.
- (b) A coding scheme by which each state index has only one bit set and all other bits clear.
- (c) A coding scheme by which all valid state index values are odd (e.g. the index 01011_2 is valid but the index 01001_2 is invalid).
- (d) A coding scheme by which all valid state index values are even (e.g. the index 01011_2 is invalid but the index 01001_2 is valid).

[5 marks]

Q2.6 The term “configuration architecture” was mentioned in relation to FPGA-based systems? What is meant by this term? Select the most accurate answer below.

- (a) The mapping of FPGA internal chip circuitry to the pins that will connect to the PCB.
- (b) The circuitry connected to the FPGA that loads configuration data into it at the correct locations.
- (c) The structuring of the FPGA that specified how LUTs and PLBs are arranged within the chip.
- (d) The way that data is to be clocked in or out of the FPGA when running an application.

[5 marks]

Q2.7 Bus structures for connecting digital logic sub-systems provides multiple advantages, for instance a bus allows for potentially multiple bus masters and allows extensibility, adding additional nodes to the bus without needing to make significant if any changes to the bus access protocol. However, a hierarchical bus incorporates both additional advantages but potential drawbacks as well. A significant advantage of the hierarchical bus model is that ... (select only one option below):

- (a) The root, the node at the top of the hierarchy, has access to all data transfers between nodes connected by the bus hierarchy which allows for additional safety and security facilities.
- (b) Data can only be transferred between nodes on the same or lower branch or the hierarchy, which provides a means to use node placement to avoid communication between nodes that should not exchange data.
- (c) There is decreased loading per bus hierarchy segment (compared to using a common connected bus) that enables potentially greater volumes of overall data transfer at a time within the network.
- (d) A bus hierarchy allows for only two nodes to communicate at a time, which both simplifies digital designs using this topology and increases reliability of data transfer.

[5 marks]

Q2.8 What does the acronym SDRAM stand for? This is a term used commonly in selecting components for the design of an embedded platform.

- (a) Synchronous Dynamic Random Access Memory.
- (b) Standardized Dual-port Random Access Memory.
- (c) Static Double-rate Random Access Memory.
- (d) Secured Digital Random Addressable Memory.

[5 marks]

Q2.9 An embedded system development company typically encounters “non-recurring costs” during the development of a system? Select one of the options below that gives an example of “non-recurring costs” (choose only one option below) ...

- (a) Research and development costs.
- (b) Components and wiring costs.
- (c) Postage on returns.
- (e) Batteries to run the platforms.

[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	FPGAs incorporate a routing network for connecting up PLBs.	Y
ii	Load balancing for parallel systems focuses on distributing work among tasks so that all tasks process the same amount of data.	N (no, it is about minimizing idle time, distributing the load)
iii	Software profiling is a process of dynamic program analysis that investigates a program’s behavior by gathering data related to its execution.	Y
iv	Access to network resources (e.g. to a remote file store) is considered a software bottleneck for many modern high-performance systems.	Y
v	The term ‘embarrassingly parallel’ is used to refer to a design that is excessively complex (has many more interrelations between parts than is necessary, hence alluding to a sense of embarrassment for the designer).	N (it is rather about many tasks done similar things will little coordination between them).

[5 marks]

Section 3: Long Answers [30 marks]

This section involves a scenario and two rather questions that have rather lengthy answers

ACME Corporate Gift WeVote Innovation – a design scenario

(an assessment of your thinking of specialized parallel systems development ... with minimal programming)

Consider that you are an embedded engineer working at the ACME Corporate Gifts development division... where ACME in this case is the acronym and registered trademark of the “Amalgamated Computing Merchandise Enterprise”, a hypothetical business. The CEO has had a major brainwave and wants to expand the usual range of electronic corporate gifts, such as bulky calculators and flashlights with short-lived

batteries, to include products that could be used to assist teamwork or business meetings. This latest concept is the WeVote innovation, which he is incredibly eager about and wants produced asap. The WeVote is planned around being used during in-person meetings; although the engineers in the corporation did try to warn him about the trends for remote work and online meetings, but those suggestions were not appreciated. A concept image of the WeVote is shown on the right. The objective of the WeVote is essentially to help a team make a decision by being able to vote Yes or No to questions or proposed statements. Or to indicate confusion

by pressing both Yes and No together. The engineers suspect the CEO has been watching too many ‘Would I Lie To You?’ shows, which is not necessarily an effective model by which organizations should have meetings or make decisions. Nevertheless, and rather ironically since there is no team for this project, you have been tasked with making design decisions on your own for this product.

The VoteCentral is the main component of the system, which is the box shown on the middle of the table in Figure 1. This component does the I/O processing using an FPGA (a rather cheap one with limited LEs, but clients don’t need to know that). The VoteCentral and other parts and operation of the system is further refined in the points below; read over this information and respond to the queries that follow.

Parts of the ACME WeVote system:

- VoteCentral unit parts
 - Switches x2: Power-on switch (you can call it *on_switch*). Enable/Disable confusion response (*conf_switch*) which enables/disables response to pushing both Yes and No buttons together.
 - VotePad_connector x 8: these are three-wire interfaces (lines *vpady[0] ... vpady[7]* and *vpadn[0] ... vpadn[7]*) to which a VotePad can be connected. There are 8 of these ports on the central unit. (It happens that only three VotePads are in the box, although additional ones can be ordered to supplement these.)
 - Side lights x 4: These are the rectangles on the sides of the WeVote indicated in Fig. 1. Behind these there is a red, a blue and a green LED. These are controlled by an array of ports *sl_red[3:0]*, *sl_blue[3:0]*, *sl_green[3:0]*. Sure, you can activate more than one LED in a side light at the same time, e.g. setting *sl_red[0]* and *sl_green[0]* to high and *sl_blue[0]* to low will cause the first side light to glow yellow.
 - Speaker out (*spk*). This is just a single bit that goes via an opamp to a speaker (i.e. provides a 1-bit DAC). It can be used for generating sounds (e.g. a beep).
- VotePad: this has two buttons (essentially pushbuttons each linked to a nice big button)
 - YES button (*pb_yes*) and NO button (*pb_no*).
 - Pad connector (*pad*) which is tristate line. Both the VoteCentral and the VotePad can write to that wire, but it is imperative that they do not write at the same time, especially never the one writing a 0 and the other writing a 1, as that will cause a short and damaged the system.

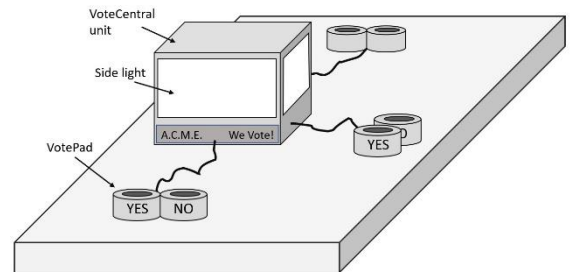


Figure 1: WeVote concept image.

The block diagram of the VotePad and how it connects to the VoteCentral unit is shown in Figure 2 below. As you can see, the VotePad has a little CPLD on it, which can also be programmed using Verilog.

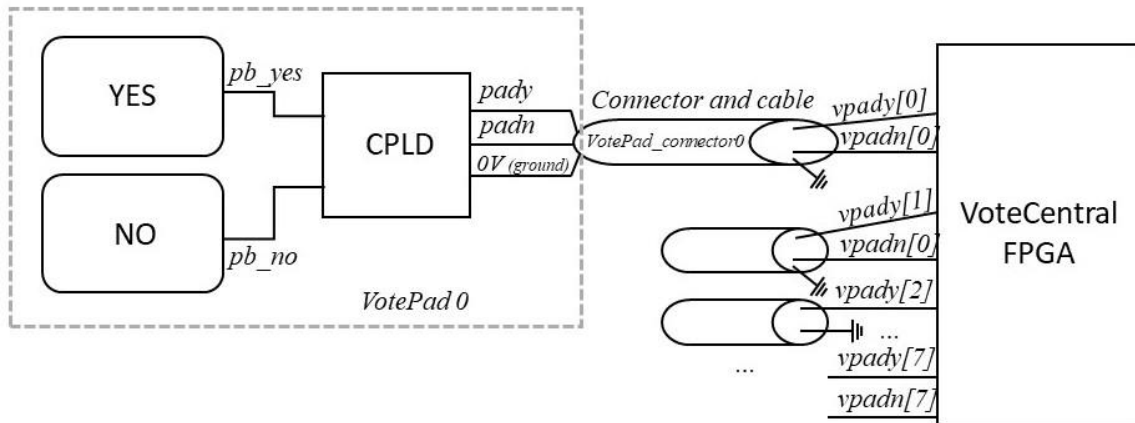


Figure 2: How a VotePad connects to the VoteCentral FPGA.

The VoteCentral FPGA runs two state machines that are described below. The first state machine, called *governor* machine just responds to reset lines and checks signals from the *votecheck* state machine.

Operation of the governor state machine:

Assume there is a 8-bit register called *state* which gets set to state values (the names given in all caps). The operation is as follows:

- On startup (when power button turned on) the system enters STARTUP state. The STARTUP will also be change to on reset or for however long reset is held in (reset line is high when reset pressed).
- In STARTUP, all four *side lights* are turn on to green for 1s and then all lights turn off and the system changes to WAITING state. The downcounter module's downcount enable line is set to 0.
- In WAITING state the system is simply waiting for any one of the VotePads to signal that a YES, NO or both YES and NO button has been pressed on it. When a VotePad button is pressed, the *pending* bit flag is set to 1 (to indicate a group decision has started), the downcount counter is set to 60 000 (for counting down one minute, i.e. by setting counter lines set<=0; limit<=60000; set<=1;) and then downcount enable line is set to 1. The *side lights* are all turned to blue. State changes to PENDING.
- In PENDING state, the system is waits for the downcount counter to reach 0, when this happens it asks *votecheck* for a vote result by sending a positive edge on the *ask* line to *votecheck* and then changes to DISPLAY state.
- In DISPLAY state the 2-bit *majority* line from *votecheck* is used to show the result of the vote:
 - If majority == 3 the side lights are set to yellow.
 - If majority == 2 the side lights are set to green.
 - If majority == 1 the side lights are set to red.
 - If majority == 0 the side lights are set to cyan (or indigo if you prefer that name).
 - The system then sets downcount counter is to 120 000 (for counting down two minute) abd the downcount enable line is set to 1, and state changes to DELAY state.
- In DELAY state, the system just waits for two minutes (i.e. for downcount counter to reach 0) before returning to the STARTUP state.

For your information, the downcount module interface is as follows (*clk_ms* is a millisecond clock):

```
// Verilog code for down counter
module downcounter (input clk_ms, enable, set, [23:0] limit, output [23:0] counter );
```

The operation of the *votecheck* state machine is given on the next page...

Operation of the votecheck state machine:

This description describes the operation that this state machine does, not the specific states needed – that is something you need to decide on (see question below). The operation is as follows:

- On reset (call it state CHKWAIT) the 8-bit arrays *yeses* and *nos* have all their bits set to 0 i.e. in C this would be

```
for (i=0; i<8; i++) {yeses[i] = 0; nos[i]=0;}
```
- On each positive edge of the clk, do for $i=0..7$:
 - if ($vpadsy[i]==1$) $yeses[i]=1$;
 - if ($vpadsn[i]==1$) $nos[i]=1$;
- If there is a positive edge on input line *ask* then the two-bit outline line *majority* is set as follows:
 - $nyes = \sum_0^7 yeses[i]$
 - $nno = \sum_0^7 nos[i]$
 - $nconf = \sum_0^7 (nos[i] == yeses[i])$ (i.e. count number pads with both yes and no set)
 - if ($nconf \geq (nyes+nno)$) $majority = 3$;
 - else if ($nyes > nno$) $majority = 2$;
 - else if ($nyes < nno$) $majority = 1$; else $majority = 0$;

Question 3.1 Verilog Coding [18 marks]

Answer the following sub-questions that consider aspects of further refining the WeVote product...

- (a) You need to attempt completion of the Verilog module for the *votecheck* state machine, as per the description of its operation given above. A starting point for its module is given below, you can change the interface and its implementation as needed. To save time, you don't have to repeat writing out the interface and instantiation of ports and registers in your answer. [12 marks]

```
// Module for hypothetical ACME Corporate Gift WeVote Innovation
// This module is used to keep track of number of votes for Yes, No and Confused
module votecheck ( reset, clk, vpadsy, vpadsn, ask, majority );
  // ports:
  input reset;           // reset line
  input clk;            // 1MHz clock line
  input [7:0]vpadsy;    // yes lines from 8 votepads
  input [7:0]vpadsn;    // no lines from 8 votepads
  input ask;           // request majority be calculated
  output reg[1:0] majority; // determine majority
  // internal registers
  reg [7:0] state;
  reg [7:0] yeses;
  reg [7:0] nos;

  // ADD YOUR CODE HERE!!!

endmodule
```

Marking notes / hints: higher marks will be given for efficiency of resource utilization. Keep in mind that the clock is 1MHz, which is fast compared to human activities, so that could be a means to save on logic utilization.

ANSWER 3.1(a): The marks will be awarded in regard to the accuracy as well as resource efficiency of the solution, the efficiency being in terms of the digital logic utilization as per the *hint* that was added at the end of the question. A simple, but resource inefficient solution would be making use of a Verilog for loops, largely just putting in place a valid Verilog syntax of for $i=0..7$ description of the processing operations indicated. This is inefficient as it will just be duplicating code with changes in bus indices used. However, there is a reasonably fast clock, *clk*, which can make the checking of the lines sequential rather than parallel, in other words fast polling of button presses rather than being triggered by a signal change. The sequential polling would effectively around utilize 1/8 of the resources compared to one doing all 8 checks simultaneously in parallel (which a for loop in Verilog would expanded out to be).

Comments: No comments will lose 2 marks (implying a max. mark of 10 for a solution without comments). The comments help with the understanding and structure of a solution thus effective use of comments to help the explanation and clarity of the solution will be awarded even if the solution is not highly efficient.

See sample solution at: <https://www.edaplayground.com/x/gZyQ>

```
// EEE4120F 2023 Verilog Coding sample solution
// See EEE4120F Exam 2023 Section 3 Question 3.1
// Exam at: http://ocw.ee.uct.ac.za/courses/EEE4120F/Archive.html
// S. Winberg June-2023

// Module for hypothetical ACME Corporate Gift WeVote Innovation
// This module is used to keep track of number of votes for Yes, No and Confused

module votecheck ( reset, clk, vpadsy, vpadsn, ask, majority );
  // ports:
  input reset;           // reset line
  input clk;             // 1MHz clock line
  input [7:0]vpadsy;     // yes lines from 8 votepads
  input [7:0]vpadsn;     // no lines from 8 votepads
  input ask;             // request majority be calculated
  output reg[1:0] majority; // determine majority
  // internal registers
  reg [7:0] state;       // state of the state machine
  reg [7:0] yeses;       // log of yes votes
  reg [7:0] nos;         // log of no votes
  // additional registers:
  reg [3:0] count;       // number of no votes
  reg [3:0] nyes;        // number of yes votes
  reg [3:0] nno;         // number of no votes
  reg [3:0] nconf;       // number of no confused votes

  // additional variables
  integer i; // used in generate loop for CHKWAIT

  // ADD YOUR CODE HERE!!! -- here is the sample solution:

  // Define some parameters for state machine states
  parameter CHKWAIT      = 1;
  parameter CHECKING     = 2;
  parameter CHECKING_NEXT= 3;
  parameter SET_MAJORITY = 4;

  // Define majority output values
  parameter MAJORITY_NOVOTES = 0;
  parameter MAJORITY_NO      = 1;
  parameter MAJORITY_YES     = 2;
  parameter MAJORITY_CONFUSE = 3;

  // This always loop implements the statemachine, which
  // is a clock-triggered (or 'hot-running') FMS.
  always@ (posedge clk)
  begin

    // check for reset (clock must be running to do reset)
    if (reset == 1)
      begin
        state = CHKWAIT;
        majority= MAJORITY_NOVOTES;
        yeses = 0;
        nos = 0;
        $display ("Reset! yeses=%b nos=%b", yeses, nos);
      end
  end
endmodule
```

```

// state operations
case (state)
  CHKWAIT:
    begin
      // keep checking all the lines to see button presses
      for (i=0; i<=7; i++)
        begin
          if (vpadsy[i] == 1) yeses[i]=1;
          if (vpadsn[i] == 1) nos[i]=1;
        end
      // if ask line is taised, start checking majority
      if (ask) state = CHECKING;
    end

  CHECKING:
    begin
      // goes from count=0 to 7 adding up number of yesses and nos
      // using clk to increment the counter so as to save logic
      count = 0;
      nyes = 0;
      nno = 0;
      nconf = 0;
      state = CHECKING_NEXT;
    end

  CHECKING_NEXT:
    begin
      $display ("count=%d", count);
      // count number of yes,no,confused
      if ((yeses[count] == 1) &&
          (nos[count] == 1)) nconf= nconf+ 1;
      else
        if (yeses[count] == 1) nyes = nyes + 1;
        else
          if (nos[count] == 1) nno = nno + 1;
          // increment the counter
          count = count + 1;
          // if counter exceeds 7 go to SET_MAJORITY
          if (count == 8)
            begin
              count = 0;
              state = SET_MAJORITY;
            end
    end

  SET_MAJORITY:
    begin
      // calculates the majority vote
      $display ("set_majority yeses=%b (%d) nos=%b (%d) ",
              yeses,nyes,nos,nno);
      if (nconf >= (nyes + nno)) majority = MAJORITY_CONFUSE;
      else if (nyes > nno) majority = MAJORITY_YES;
      else if (nyes < nno) majority = MAJORITY_NO;
      else majority = MAJORITY_NOVOTES;
      // next state will be back to wait state
      state = CHKWAIT;
      // clear the logs to start again
      yeses = 0;
      nos = 0;
    end

  default:
    // if in undefined state, jump to CHKWAIT
    state = CHKWAIT;

endcase
end

endmodule

```

[Total marks: 12 marks].

- (b) In part (a) you implemented code for the VoteCentral FPGA. It happens that the VotePad has a small CPLD which can also be programmed using Verilog. Now we need to think about Verilog for the VotePad. All it needs to do is see if there is a *pb_yes* or *pb_no* that got pressed. If *pb_yes* is pressed, then it needs to raise its *pady* output to 1 for a few milliseconds (say at least 4ms), if *pd_no* is raised then *padn* needs to raise for a few milliseconds. Of course if both are pressed together, then both *pady* and *padn* must both be raised for a few milliseconds. VotePad also has its own millisecond clock *clk_ms* line. Attempt to write a Verilog code module (call it *VotePadCtr*) that provides this functionality. [6 marks]

(*hint*: keep in mind debounce you may want to argue why it is unnecessary or incorporate button debouncing in your solution.)

ANSWER 3.1(b): A simple solution would be adequate. The student can reason that debounce is unnecessary in this application because, as per the description of the *votecheck* module, the yes and no vote lines will be polled and there is no counting of the number of times a VotePad button is pressed thus eliminating the need to remove debounce. The buttons are just registered as having been pressed or not pressed. Similarly, pressing both together would be picked up easily by the *votecheck* module (there is no description indicating specifically that a vote can be changed, as if one button is pressed and then in the same voting period the other button is pressed, this will be recorded as confused as apposed to either a Yes or No response). The *clk_ms* clock could be used to ensure a that the *pady* or *padn* line is held high for long enough, But, actually, this was just put in as a potentially extraneous facility because, if the student thinks about it a bit, the human pressing the button is probably going to hold it pressed in for more than a millisecond or so, thus it is highly unlikely that any of the buttons would be pressed and let go so very quickly as not to get detected by the *votecheck* module, even if it is polling each button once every 8us (i.e. if it cycles between checking each button for each 1MHz clock edge).

Comments: A solution with no comments will be limited to a maximum of 5 marks. As mentioned above, effective use of comments to help the explanation and clarity of the code will be acknowledged in the mark.

Question 3.2 Design Considerations [12 marks]

Answer the following two questions which don't require Verilog coding. These relate to the WeVote product described earlier.

The WeVote allows for up to 8 VotePads, each of which has its own connector that plugs in to the VoteCentral. There are of course 8 VotePad connector ports on the VoteCentral. Indeed, this ends up using $2 \times 8 = 16$ pads of the FPGA. It would be nice to use a bus instead, essentially having an expandable number of VotePad connectors, allowing many (e.g. 20) to be connected up.

- (a) Propose a solution by which the two-wire connectors could instead be a bus, and adding some sort of protocol by which each VotePad can communicate with the VoteCentral to inform it of votes. For your answer you should provide a block diagram as well as a written explanation of how this would work. Note that you do not need to write any code for your answer. [8 marks]

The plan here is to assess the student's competency in proposing a digital design solution. For the answer the student could discuss utilizing a simple protocol, in which there may be one or more bus master. Although, a fairly simple and effective approach would just be allocating each VotePad a number (e.g. using a DIP switch) and then the VoteCentral can be the bus master which simply poll each VotePad by sending out its ID which asks the VotePad to either write back the state of its buttons. The one line (e.g. *vpady* as per Fig. 2) could become the line from VoteCentral to all VotePads over which an ID is sent requesting a VotePad to reply. The other line (e.g. *vpadn*) could be written to by the VotePad in order to return the state of its buttons. For such a protocol, there would still be a limitation on the number of VotePads that can be connected, e.g. a 8-bit DIP switch implying a maximum of 256 vote pads attached. Furthermore, in allowing expandability to so many VotePads, the CPLD code on the VotePad would need to be expanded, for instance to record what buttons were pressed and only send this information when it receives a request from the VoteCentral.

- (b) The VotePad only has a CPLD on it. Discuss the viability of switching over to using communication via a two-wire bus to implement this more expandable version of the WeVote system. You can motivate for or against the relevance of a CPLD, e.g. if it is advisable or not to switch to a processor and if so what sort of processor. Also briefly indicate potential pros and cons of such a switch. [4 marks]

The viability of switching over to a two-wire bus implementation would be entirely doable, there would likely be no need to change the hardware besides the gateway running on the CPLD. While CPLDs have much fewer logic resources than a FPGA, it would probably be adequate for implementing a simple protocol of detecting a ID string of pulses being sent over one of the wires. Since there are two lines, *vpad_y* that could become used for pulsed messages broadcast from VoteCentral to all VotePads, and *vpad_n* that a VotePad uses for sending a replying, the needed hardware would be in place, and the protocol would be sufficiently simple so as not to necessitate a processor. A low-cost CPLD still with a few 100K of logic costs around R100, but a low-cost microcontroller, like a PIC16 costs a fair bit less, around R20 (at least for a very basic one). So, for cost saving it may well be more viable using a cheap microcontroller rather than a CPLD, and additionally a microcontroller would be easier to program and support a certainly be easier for supporting more sophisticated communication protocols that using a CPLD for such purposes. So, in summary a benefit of switching to a cheap microcontroller would be savings on hardware costs over a CPLD and ease of programming and ability to more easily supporting more complex communication protocols. But the drawback of switch to microprocessor would be potential restrains in processing speed, convenience of access to many PIOs and configurability of these.

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
```

