



High Performance Embedded Systems EEE4120F



QUESTIONS & MEMO IN ONE !!!

Remove answers to generate question paper

FINAL EXAM

10 June 2022

Venue: Snape TS2B Time starting: 14h00

3 hours

*Examination Prepared by:
Simon Winberg*

Last Modified: 11-Jun-2022

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

<u>Section 1</u>	<u>Section 2</u>	<u>Section 3</u>	<u>Appendices</u>
Short Answers (3 x 10-mark questions) [30 marks] pg 2	Multiple choice (7 x 5-mark questions) [35 marks] pg 4	Long Answers (2x question design/coding) [35 marks] pg 7	A: Verilog cheatsheet pg 9

RULES

NB

- 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 [30 marks]

Question 1.1 [10 marks]

This question relates to some terminology and design strategies in regards to HPES systems. Briefly answer the sub-questions that follow.

Q1.1. (a)

You are probably aware of the increase popularity in edge computing. Indeed, to some extent this is a revival of the approach, as prior to the availability of high-speed networks, there may have been little option than to use this approach. Briefly explain what the notion of edge computing, and why might the availability of small, low-powered and fast processors possibly have any impact on the increase popularity of this despite there being such huge gains in connectivity and networks bandwidth today. [4]

ANSWER: The notation of edge computing refers to a distributed computing approach that brings computation, and possibly data storage as well, closer to the sources of data or sensing of the system. This approach can improve response time and reduce bandwidth utilization. IoT is an approach that often utilizes edge computing, where more 'intelligent' nodes close to the sampling filters the received data sending only small amount of data, e.g. alert signals, back to the central nodes that record or respond to the sensor input. The availability of more powerful, but smaller and low power, compute facilities for edge computing allows that much more processing and advanced operations to be performed at the edge, considerably more so and at less power use than was previously possible, thus the available response time and 'smarts' of the system is that much greater at the edges which offsets the dependence on high-speed networking and central processing as that data transfer becomes extraneous.

Q1.1. (b)

The following three terms are commonly used in very broadly describing a parallelization approach (e.g. an approach applied to go from a sequential or standard algorithm to a parallel version). The terms concerned are: (1) Embarrassingly Parallel; (2) Massively Parallel; and (3) Stupidly Parallel. Explain what is meant by the first two terms (to boost your points towards 4/4, and offset any grammatical faults, make mention of what is meant by the 3rd term which is not an official formal term). [4]

ANSWER: As per lecture 8:

Embarrassingly Parallel = Simultaneously performing many similar, independent tasks, with little to no coordination between tasks.

Massively Parallel = Hardware that has very many processors (execution of parallel tasks). Can consider this classification of 100 000+ parallel tasks.

Stupidly Parallel = While this isn't really an official term it typically relates to instances where a big (and possibly very complex) coding effort is put into developing a solution that in practice has negligible savings or worse is a whole lot slower (and possibly more erroneous/buggy) than if it was just left as a simpler sequential implementation.

Q1.1. (c)

Two common practices of divvying up memory among multiple processors in a shared memory system including having the memory 'partitioned' between cores or 'interleaved' between cores. Explain the difference between these two approaches. [2]

ANSWER: Also mentioned in 8:

Partitioned = is simply separating the memory into two or more blocks, these blocks might be the same size or follow some other sizing scheme. Technically interleaving can be considered as a form of partitioning (but the student does not need to explicitly indicate this).

Interleaved = To arrange data in a noncontiguous way, this may mean that vector element $x[i]$ are accessed

by threads in a step sequence, skipping indices that an other thread would work on, e.g. thread 1 works on $x[0]$ $x[2]$ $x[4]$... and thread 2 simultaneously works on $x[1]$ $x[3]$ $x[5]$. Interleaving is essentially swapping contiguous blocks of memory, within a larger memory area to be work on, between different handlers in a consistent fashion.

Question 1.2 [10 marks]

Consider that you have been provided the following MATLAB code that needs to be converted to a Verilog solution in order to execute on an FPGA. This MATLAB code is simply reading values from a vector. But in the system you are building, you need to read the values in from an input port (i.e. ADC_in) that is connected to an ADC. To get a value from the ADC, you first need to raise the $read_req$ line and then wait for the $read_rdy$ line to change from low to high. After the $read_rdy$ line has changed to high you need to make sure to set $read_req$ back to low so that you can, at some stage, request another sample. Now that you know how the interface to the ADC works, proceed with completing the Verilog starting point so that you can ready 128 values from the ADC. Count the number of samples above the threshold, and if this number exceeds N_{above} turns on the alert line.

MATLAB code to be converted into Verilog (but note this just uses a vector, not reading signals):

```
function alert = checkabove (samps, n, thresh, nabove)
% This function counts the number of samples in the byte array
% (i.e. 8-bit samples) in samps that are above the threshold thresh parameter.
% if there are more than nabove samples above thresh then alert is returned
% as true to indicate the altern state should be entered.

cnt_above = 0;
for i=1:n
    % check if next sample is above threshold
    if (samps(i)>thresh) % if it is then
        cnt_above = cnt_above + 1; % increment cnt_above
    end
end
% if there are more than nabove samples over thresh raise alert
alert = cnt_above > nabove;
end
```

Starting point for the Verilog code that you need to port the MATLAB code above to.

```
module checkabove (ADC_in, read_req, read_rdy, Nabove, clk);
// This function counts the number of samples read from ADC_in
// (i.e. 8-bit samples). Then needs to check how many of these are above
// the threshold thresh. there are more than nabove samples above thresh
// then raise alert.
// Use of the clock (clk) is optional. Assume it is high-speed, ~ 40MHz.

input [7:0] ADC_in;
output read_req;
input read_rdy;
input [7:0] Nabove;
input clk;

// ... add your code and any other registers here ...

endmodule
```

[10 marks]

ANSWER: Student is expected to write Verilog code for this. There is no single solution for this. Points awarded for structure and use of comments. (A sample solution to be provided later).

Question 1.3 [10 marks]

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

- (a) You may recall the use of the terms spatial computing and temporal computing. Briefly distinguish between these concepts [2 marks]. Explain one case where you recommend spatial over temporal computing as a means of thinking about a computing solution; and another case where you would recommend temporal over spatial computing for thinking about a solution. [2 x 2 marks].

[total of 6 marks for (a)]

ANSWER: See lecture 4...

Temporal = The traditional paradigm. Typical of sequential programs. Things done over time steps.

Spatial = Expression computing as happening in different spaces, rather than in different times. Suited to hardware. Emphasises things related in a space more than in time (although representations typically have mechanisms to represent temporal aspects as in what is done first, what results after a process).

- (b) In regards to thinking about solutions for deployment on an FPGA, it can be useful to follow a 'HDL Imitation' approach prior to settling on a specific HDL language (e.g. Verilog, System Verilog or VHDL) to utilize in the design. Briefly elaborate on what the HDL Imitation approach involves [2 marks], Why might it not be a means from which a bitfile can be made to run on a FPGA [2 marks].

[total of 4 marks for (b)]

ANSWER: See lecture 4...


HDL imitation is essentially developing an application in an executable – usually sequential – programming language such as Python or C that mimics the operation of what would be done by one or more modules (or entities) in an HDL design. It is circumventing the processing of writing actual HDL code that can be time-consuming to do if your immediate objective is just to do design or processing exploration rather than implementing executable (deployable as hardware) processing solution. Effectively, if bringing in some software terminology, HDL Imitation coding would typically become 'Throwaway code'... but not necessarily thrown away as it can still be useful design artefacts that can help maintenance engineers, that may need to upgrade or maintain systems, to better understand what operations are performed by potentially complicated, if not convoluted, HDL cores.

Clearly you can't usually generate bitfiles from HDL imitation code because you aren't using a valid HDL description language for writing it. Nevertheless, if you are using something like C for your HDL imitation code you may be able to directly convert it to executable HDL via a sophisticated tool like Xilinx Vivado HLS.

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. **QUESTIONS COMPLETED!**

Q2.1 Which one of the following algorithms is the most course grained when processing a $N \times N$ pixels 8-bit grayscale image that is sitting in shared memory accessible by multiple processors?

- a) A mean image filter, that calculates the average grey scale value from the surrounding 5×5 pixels.
- b) Add all the integer values in an array together, outputting one integer.
- c) Increment the brightness of each pixel in the image that is dimmer than pixel value 200. 
- d) For each pixel $IMG[x,y]$ set the value to $(IMG[x,y] + IMG[(x+1)\%N,y])/2$.


[5 marks]

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


- (a) Cost of maintaining or upgrading the system
- (b) Cost of the wiring and components
- (c) Cost of power to run the platforms
- (d) Cost of designing and prototyping the system 

[5 marks]

Q2.3 Which of the following options most accurately explains what a ‘half-duplex dual-port memory controller’ is, if you are considering to implement such a module in Verilog for an FPGA?


- (a) It is a module that has separate interfaces for a write address and read address, as well as read data line and write data lines, but you cannot write and read simultaneously to a particular address. 
- (b) It is a module where you can read any address and write any address in memory at the same time.
- (c) It is a module where you can only read one address, *or* write one address in memory at a time.
- (d) It is a module where you need to first enable the *cs* (chip select) line, then select if you are going to read or write (by setting either the *we* or *re* lines, and leaving the other low), and then setting or reading the data lines.

Q2.4 Which one of these clearly explains the difference between vertical and horizontal load balancing?

- (a) Vertical load balancing is all about bring more power in to make the individual CPUs in the system run faster; this may sacrifice the availability of shared memory; but in comparison horizontal load is all about sacrificing the speed of processors to boost the availability of shared memory.
- (b) Vertical load balancing is more about boosting the power of the centralized processor or closely coupled processors, making the faster and having less latency. Whereas horizontal load balancing is about achieving more distribution of the processing, running more applications at once and achieves greater scalability. 
- (c) Vertical load balancing and horizontal load balancing is exactly the same thing; load balancing is load balancing. There is no point to separate them into horizontal and vertical variants as both would have exactly the same limitation in terms of processing speed, access to memory, scalability and processing power.
- (d) Vertical load balancing is about aligning the power of the central processor(s) with availability of shared memory; but horizontal load balancing is aligning availability of distributed processors to distributed memory.


[5 marks]

Q2.5 Select which one of these is an accurate alternate term for synchronous communications?

- (a) On-time transfers.
- (b) Blocking communications. 
- (c) Serialized communications.
- (d) Non-blocking communications.






[5 marks]

Q2.6 What is meant by the “configuration architecture” of a FPGA-based system? (Select the most accurate answer below).

- (a) The structuring of the FPGA that specified how LUTs are arranged in the chip.
- (b) The mapping of FPGA internal chip circuitry to the pins that will connect to the PCB.
- (c) The circuitry connected to the FPGA that loads configuration data into it at the correct locations. 
- (d) The specification of binary data to be clocked to/from the FPGA during programming.

[5 marks]

Q2.7 These are True / False 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 the baseline speed performance for an algorithm for which you are intending to implement a parallelized or enhanced version on a specialized platform. 
- (ii) The last two letters in the acronym “SWAP”, in relation to HPES systems, represents ‘APplication’. 
- (iii) The classic Von Neumann architecture supports memory that can be shared between instructions and data memory. 
- (iv) A standard ADC, such as the classic MCP3008 that you might remember, is a type of programmable processor. 
- (v) Speedup is defined as execution time of the non-optimized version divided by the execution time of the optimized version. 

[5 marks]

Section 3: Long Answers [35 marks]

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

Smart Vacuum Cleaner with Air-speed & Blockage Detector (SVC-ABD) Scenario

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

Imagine that you are one of the engineers in a development company called Brilliantly Amazing Appliances for Domestic Use Corporation (BAADUC for short). The company produces a variety of consider gadgetry and other widgets... and, as they (somewhat poorly written) slogan goes: our appliances don't just waste space, they provide super time-saving functions too! Among their products, which have recently moved beyond USB version 1.0 compatibility, is a line of smart vacuum cleaners (SVCs). The company does much research in this area, having developed machine learning (ML) technology by which their vacuum cleaners can learn to orient and disappear in an indoor environment (i.e. among interleading rooms and tables).



The latest version of SVC, which you happen to be assigned to working on, has a myriad of detectors combining both airflow speed measurement and blockage detection. As you realized in your experience with this company, they do rather like to save development costs where possible. But not so much in this case. The system you are working with has a low-cost FPGA that connects with an ARM Cortex-M0 processor (yes, they splurged on that, but it isn't much pricier than an 16bit PIC).

As mentioned, the SVC-ABD has a FPGA on it. However, the design is such that only the ARM Cortex-M0 can 'talk' to and control it. The M0 programs the FPGA via bit-banging one of the PIO lines. And it can communicate with the FPGA, once it is programmed with an appropriate bitfile, via either PIO lines 1-12 (called PIO 1-12 on the M0 or U 11-22 on the FPGA) or via SPI (specifically SPI1 on the M0). This interface is illustrated on the right. As indicated, the M0 has a SPI controller that offloads the SPI communication from the processor.

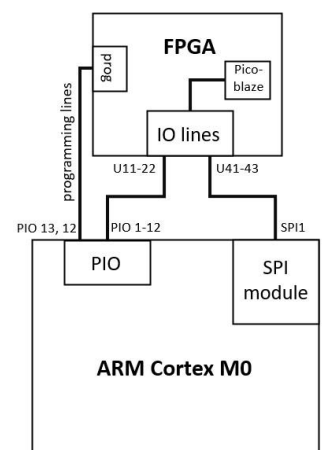


Figure 1: SVC controller block diagram

Question 3.1 Design Considerations [17 marks]

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

- (a) In this design, it is necessary for the microcontroller to start up first and perform programming of the FPGA. This can make for quite a few complications during development and debugging, but it can also save quite a bit of complications in circuit design. Discuss at least two pros and at least two cons to this approach of relying on the microprocessor to program the FPGA. [2x 2 = 5 marks]

ANSWER: The student is expected to discuss potential drawbacks, such as not being able to connect up with the Xilinx Vivado programmer and debugger (if Xilinx used, or other tools if not). This alone would be a significant slow-down for development and will likely necessitate the developers to utilize evaluation boards for much of the development and having to swap HDL between the actual platform that would need its own programming software and the evaluation board that would be pre-configured for interacting with Vivado. Major disadvantages would hence be having to develop custom programmers and debugging facilities for the SVC platform concerned.

In terms of pros... these will like be pretty limited compared to the potentially huge costs and time involved in developing custom programming and debugging software infrastructure. Nevertheless, the approach could

save a fair bit of configuration circuitry needs for the system, deploying all this essentially as software that will run on the M0. Indeed, this approach is not actually by any means unusual. But many times, to avoid the hassles of not being able to integrate the FPGA easily via to existing debugging tools, the designs are more likely to simply have a few additional pins available on the board that can connect up to a standard programming link... i.e. putting in to the PCB at least a few connections that standard programming connectors (like the Xilinx 'USB Platform Cable' ... which is not just a cable but essentially incorporates configuration circuitry as well that can be hook up to programming pins on a custom board to circumvent any on-platform programming methods, such as an on-platform CPU doing the programming).

- (b) For this system, consider that a simple softcore, in particular the Picoblaze, is being used on the FPGA (among other things). The Picoblaze simply captures samples, does simple filtering and then sends data to be kept and processed on to the M0. But two of your colleagues are having a bit of a verbal tiff. The one, let's call her Ann¹, is adamant that it is technically a heterogeneous computing system that uses message passing. But another – and more grouchy – colleague, let's call him Bob, says that's totally wrong, that what we're dealing with here is just a dual-processor system with shared memory. For your answer to this question provide a correct and well-motivated argument that would support either Ann's or Bob's view. [7 marks]

ANSWER: If you read the notes, it's pretty clear that grouchy Bob is mistaken. Ann is right: what we are actually dealing with here is a multi-processor heterogeneous system that is using (as far as the information provided indicates) a message passing system. If the Picoblaze and the M0 shared access to the same memory ... sorry I'm giggling at the thought, ha ha; an 8-bit processor sharing memory with a 32-bit processor, that is ludicrous, but nevermind ... you would need some sort of memory control unit that would allow for this, which would be a tiny bit of the available memory that the 32-bit M0 is able to access. The Picoblaze isn't really slow, but surely there would be no serious processing that it would be doing to merit sharing memory between it and a considerably more high-performance, hardcore, processor. Just to act suitably professionally in this case, if you were mediating this argument, you should nevertheless be courteous and considerate; nice as a pancake in correcting Bob in a suitable flat but pleasant manner.

- (c) If the FPGA (actually the Picoblaze on the FPGA) is to pass bytes to the M0, albeit fairly slowly around 1Kb per second, would you recommend use of one or more of the PIO lines (ie. PIO1 – 12) or use of SPI1? Provide a motivation for your choice. (Assume there's currently no other plans for using SPI1). [5 marks]

ANSWER: There is no single right answer for this one. The marking is more based on the logic and reasoning that the student provides. Using the bitbanging the PIO would be totally fine, considering the slow speeds that are needed, and this would leave the SPI, which can work at much higher rates, for other needs.

Question 3.2 Verilog Coding [18 marks]

The SVC-ABD vacuum cleaner measures inlet airflow speed and does blockage detection. Yes, you were probably – and not necessarily eagerly – expecting a task concerning this aspect. And you are right, that's what this last question covers... so, brace yourself and put a new bag in your fancy new SVC-ABD!

The SVC-ABD measures airflow and blockages pretty crudely. It has one spiral turbine, call it *IAF* (for 'Incoming AirFlow'), that simply spins according to the speed of incoming air². And a stretch sensor, that is woven into the cover of the exit vent, and either contracts (if little outgoing air, i.e. motor off or nozzle blocked) or expands (i.e., if lots of outgoing air, nozzle not blocked). The extent of stretch is used to produce an *OAF* (for 'Outgoing AirFlow') measurement.

¹ Obviously trying to think of short three letter names that could be easily enumerated 😊

² I know, not a great design as it could get clogged up and defuncted easily. Built-in-obsolence, or such a crude design, isn't something that should be encourage.

The AIF effectively spins once, produces one pulse, for air moving at 4cm/s. That's a pretty slow speed. The vacuum cleaner generally sucks in air, when not blocked up, at 40m/s.

The OAF works differently. It measures a resistance value that relates to how stretched the fabric is of the outgoing vent of the vacuum cleaner. It has a collection of 16 threads, which are resistors, that need to be read close together in time. The change in measured resistance of the threads between when the motor has been off (which is configured before vacuuming starts) and the resistance of the threads (e.g. while the motor is on) corresponds to the outgoing air flow (this calculation just uses a scaling factor, called Detected Resistance To Air Speed or DRTAS factor, to do this).

The following pseudocode shows how the OAF measurements are generated, to be done every 1s at least.

```
OUTPUT INT OAF; // outgoing air flow speed to be worked out at end of this state
REG BYTE BASER[16]; // used to calibrate the base resistance of the threads at startup
RESET: // when in reset mode the system needs to do this, the clock clk is still running:
    FOR I = 1 TO 16
        MUX = I; // select which thread to read resistance of
        BASER[I] = READ_ADC(); // read resistance of thread I
    NEXT
RUN: // this is done while
    INT DR = 0; // delta of resistance
    FOR I = 1 TO 16
        MUX = I; // select which thread resistance to read
        DR = DR + READ_ADC() - BASER[I]; // resistance increases if threads stretched
    NEXT
    DR = DR / 16; // average change in resistance
    OAF = DR / DRTAS // convert average resistance change to airspeed in cm/s
```

You Todo:

You're to attempt to develop a Verilog module with a suitable interface, aligning with the description above, that that will calculate the both the incoming airflow according to the operation of the IAF measurement and the outgoing airflow OFS measurements. You can call this the IAFOFS module. These calculations should each be sent to a 16-bit output port, called IAF_out and OFS_out respectively. Allow for a clk clock input, that runs at 10MHz.

- (a) Provide the module interface definition for the IAFOFS module. Include comments for any additional arguments that you may decide to add to the module. You can assume the select lines for the multiplexer needed for the OAF measurements is just a 3-bit output called *mux*. [6 marks]

ANSWER: The student needs to ensure ports are provided for the AIF pulse line (i.e. AIF_in) that indicates whenever the turbine has done one revolution. The clk input is needed, as well as the 3-bit mux output line. The ADC_input lines are also needed. The student may need to argue that the ADC lines can just be read at any time without handshaking (this is left as an open-ended design characteristic). Comments are expected but those could boost an imperfect 3/4 answer to 4/4.

- (b) Attempt to complete the IAFOFS Verilog module as fully as possible that that will calculate both the incoming airflow according to the operation of the IAF measurement and the outgoing airflow OFS measurements. (Decent comments count 2 marks.) [11 marks]

ANSWER: The student should put together needed statemachines or alternative methods to firstly calculate the speed of the inflow air, which can count the number of clk pulses between AIF_in pulses to determine the inflow air speed. Then the statemachine for the OAF will need to measure the resistances and do the

average calculation of these and the conversion calculation to convert to the outgoing airspeed based on how much the webbing of the outlet vent has been stretched based on the outflowing air. The IAF_out and OAF_out would need to be updated every second or quicker.

- (c) For those wanting to vacuum up a remaining mark, or to blow your own trumpet: explain how you'd detect that the SVC-ABD has got blocked. [1 mark]

ANSWER: This would probably be pretty simple: just checking to see if the AOF_out is at a low level for more than a few seconds, while the motor is on. This would probably be the case if the hose is blocked, but I'm not entirely sure, it's something that would need to be investigated experimentally. Nevertheless a reasonable guess such as this would get the student a 1 mark for the attempt.

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
```

