Digital Systems
EEE4084F

FINAL EXAM

14 July 2017

3 hours

Examination Prepared by:
Simon Winberg

Last Modified: 12-Jul-2017

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 that you **put** your **student name and student number,** the course code **EEE4084F** and a title **Final Exam** on your answer sheet(s). Answer each section on a separate page.

DO NOT TURN OVER UNTIL YOU ARE TOLD TO

Exam Structure

Marked out of 120 marks / 180 minutes. Time per mark = 1min 30sec

Section 1	Section 2	Section 3	Appendices
Short Answers *(4 x 12 mark questions)* [48 marks] pg 2	Multiple choice *(6x4-mark questions + 3x2-mark true/false q's)* [30 marks] pg 4	Long Answers *(2x question HDL & MPI)* [42 marks] pg 6	A: Formulae B: Verilog cheatsheet C: OIC Instructions D: MPI cheatsheet pg 8

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 [50 marks]

Question 1.1   [16 marks]

The OIC processor is an 8-bit processor that can address only 256 bytes of internal memory and does input/output via port access to 256 port addresses. See Appendix C for the processor's instruction set.

Assume you've written the code below that reads two byte values (into variables X and Y), each from a different ADC. A trigger output needs to be set to 1 if either X or Y is above a trigger level. If neither X nor Y are above the trigger level then the trigger output needs to be 0. The program is shown below.

```
/* Trigger generator application runs on the OIC */
typedef unsigned char BYTE;
/* define the port interfaces */
#define ADC1    0x10
#define ADC2    0x11
#define TRIGGER 0x12

int main () {
  BYTE X,Y,                 /* Latest ADC values */
       trigger_level = 10;  /* trigger level */
  while (1) {
    /* read from ADCs */
    X = IN(ADC1);
    Y = IN(ADC2);
    /* check if level exceeded on both ADCs, and if so
       output 1 to the trigger port. */
    if ((X>trigger_level) || (Y>trigger_level)) OUT(TRIGGER,0x1);
        else OUT(TRIGGER,0x0);
  }
  return 0;
}
```

Please answer these questions that relate to the above code …

(a) What type of a processor is the OIC, Van Neumann or Harvard? [1 marks]

(b) Convert the C code above into OIC assembly code [10 marks]. (*Hint:* end of Appendix C shows an example of converting a for-loop into OIC code.)

(c) At what rate (worst case, i.e., slowest rate) does your OIC program update the TRIGGER? The OIC is clocked at 100MHz, and completes one instruction (including jump) in one clock period. [3 marks]

(d) Given the combinational circuit in Figure 1, what is the speedup of this circuit in updating the TRIGGER output in comparison to the speed the OIC is updating its TRIGGER (as worked out in (c) above). The comparators (CMP) take 50ns to update, the OR gate takes 10ns, the ADC updates whenever it wants to (i.e. assume the ADC and REG are inputs and no delays in getting their data). [2 marks]

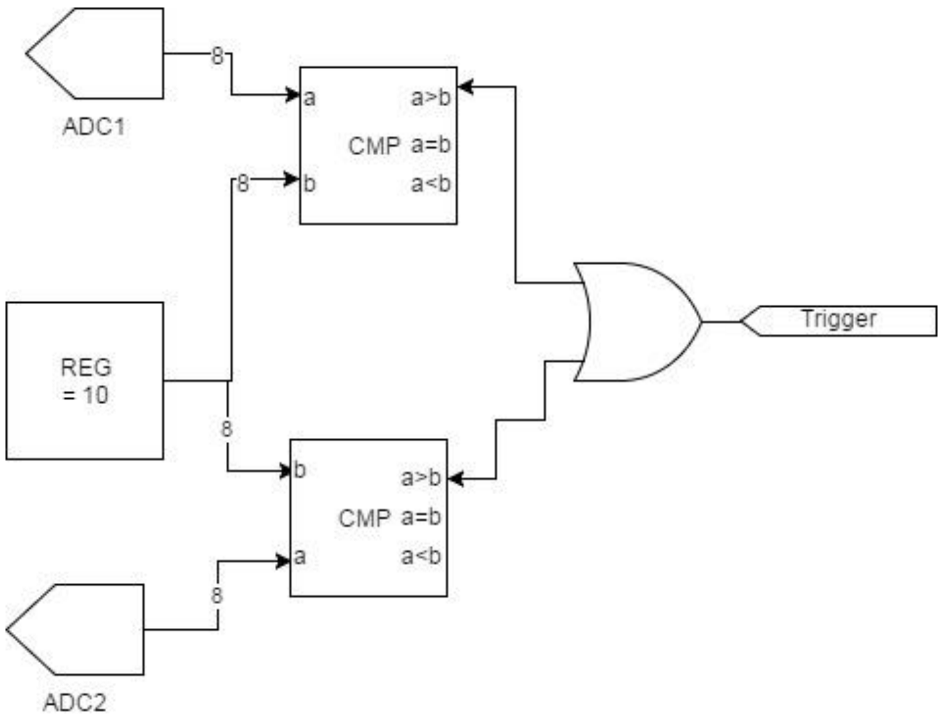


Figure 1 Cluster Topology for Question 1

Question 1.2 [12 marks]

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

(a) Many RC platforms are built using FPGAs. But they often also include a CPLD or PLA as well in the platform design. Explain the following…
i. What is the main difference between an FPGA and a PLA? [2 marks]
ii. Explain why a PLA (or a CPLD) in a reconfigurable computing system is often needed as configuration / glue logic whereas FPGA(s) are more commonly used for the actual computation (e.g., digital filtering) operations. [3 marks]

(b) Developing FPGA code can get you part of the way to an ASIC. What are some of the further difficulties (name at least one) and risks (name at least two) in taking an FPGA design further to become an ASIC? [4 marks]

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

Question 1.3 [12 marks]

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

(a) Describe what is meant by the acronym *SWAP* for a high performance embedded computer (HPEC) system, and one *performance metric* (relating to any aspect of SWAP) of a HPEC system. Mention units of quantification (for example, the power of a system can be measured by how quickly the system converts energy into work; the unit for power is the WATT). [3 marks].

(b) Granularity of a computing solution can be characterized by the ratio of the amount of computation (processing instructions per datum) to the amount of communication (transfer cost per datum).

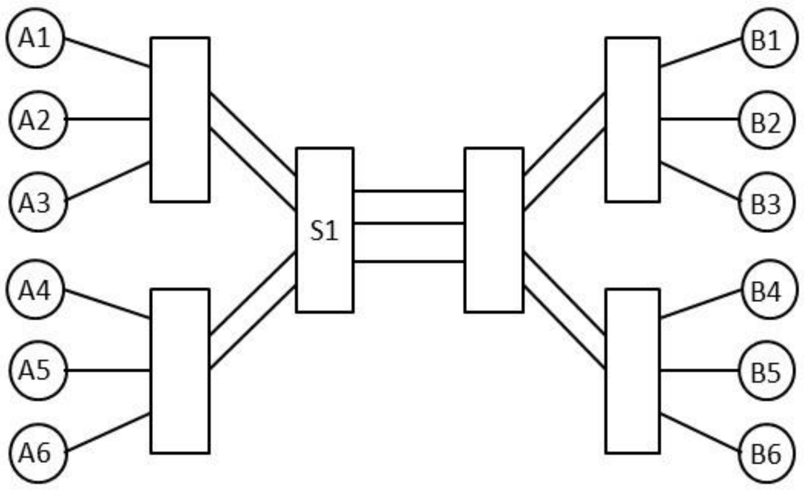
i. Explain the difference between coarse-grained and fine-grained computation and use this computation:communication ratio to assist your answer. [3 marks]

ii. Explain in a sentence what is meant by embarrassingly parallel. Indicate if it is fine-grained or course-grained computation. [2 marks]

iii. Provide two code snippets (e.g. examples of a loop), one to illustrate a course-grained problem and one to indicate a fine-grained problem. [4 marks]

Question 1.4 [10 marks]

The network structure below shows a network comprising 12 processing nodes shown as circles (labeled A1-A6 on the left and B1-B6 on the right) and crossbars shown as rectangles. Each network link is shown as a single line and these are each full duplex supporting 1Gbps simultaneously in either direction.



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

(a) The concept of bisection bandwidth is a measure of the networking performance of a system. Explain the concept of bisectional bandwidth. Indicate how it is useful in describing the anticipated performance of a HPEC system composing multiple processing nodes. Motivate for or against the bisection bandwidth of the network in the above figure being 1Gbps. [4 marks]

(b) If each node contains 100MBytes of data and a complete exchange is performed[1] (i.e. data is swapped between A1←→B6, A2←→B5, A3←→B4, A4←→B3, A5←→B2 and A6←→B1), then explain the following…
(i) how much data is going to pass through crossbar S1 (i.e. through all ports in any directions, assume 1Mb travelling from a port on one side to a port on the other side counts as 1Mb. Assume each node has 1GBytes of RAM)? [1 mark]
(ii) Describe an effective (as optimal as possible) scheduling of transfers that will do the complete exchange in minimum time and indicate what this minimum time would be (assume no delays for acknowledging transfers, reading/writing to memory etc.). [5 marks]

[1] Assume that for the network each byte is simply sent as 8 bits, not worrying about stop and start or other encoding bits.

Section 2: Multiple Choice [30 marks]

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

Q2.1 A FLASH ADC tends to need a lot of comparators, but it is quick. To illustrate this, calculate how many comparators are needed for an 8-bit FLASH ADC? (using the classic FLASH ADC design).

(a) 2.

(b) 8.

(c) 31.

(d) 127.

(e) 255.

[5 marks]

Q2.2 Which option below best describes the notion of peak computation rate (PCR)?

(a) PCR is a multiplication of the number of operations per second the processor can do by the clock rate in Hz.

(b) PCR is determined by calculating the arithmetic operations the processor can do per clock cycle multiplied by the maximum clock rate of the processor.

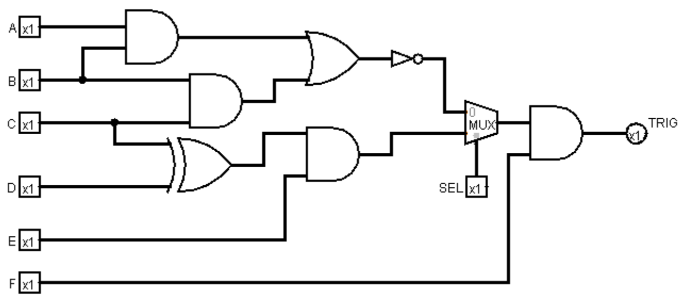
(c) PCR involves running a benchmark and then calculating the number of arithmetic operations performed by the benchmark and dividing by the time it took to complete the benchmark.

(d) PCR is equivalent to the maximum clock rate of the processor multiplied by the time it takes to do the longest arithmetic operation.

(e) PCR is equivalent to the maximum clock rate of the processor divided by the number of cycles it takes to complete one instruction.

[5 marks]

Q2.3 A simple controller running on a PIC processor, clocked at 40MHz, continuously runs a control loop comprising 50 instructions. Each instruction is completed in one clock cycle. A PLD design is under progress to replace this, which could save (a tiny bit) of cost and maybe be more responsive. The logic design shown below shows a start, which would be representative of the logic circuits to replace other pieces of the program. Assuming a propagation delay for each gate is at most 40ns, what would be the expected speedup of the PLD-based system over the PIC system? Calculate the propagation delays as needed (the other parts of the code will be replaced with similar circuits that would run in parallel).



(a) The speedup is less than 2 (could even be a slowdown)

(b) The speedup is between 2 and 9

(c) The speedup is greater than 9 but less than 12.

(d) The speedup is greater than 12 but less than 15

(e) The speedup is equal to or greater than 15

[5 marks]

Q2.4 Differential nonlinearity (DNL) of a non-ideal ADC, after compensating for any offset and gain errors, is specified as (assume non-dynamic DNL)…

(a) The deviation between the minimum and maximum voltage range that the ADC samples divided by the number of codes (e.g. an 8-bit ADC measuring -1V to 1V has a DNL of 2/256 (i.e. 0.0078V).

(b) The deviation of any code in the transfer function from the ideal code width of one LSB.

(c) The average deviation between the ideal transfer function and the actual transfer function.

(d) The sum of all deviations between the actual and ideal transfer function at the midpoint of each code.

(e) The absolute deviation in transfer function between any two unique codes, A and B, divided by the number of codes between A and B.

[5 marks]

Q2.5 The AMBA bus standard was developed by, and is used by, which company for connecting peripherals and processors?

(a) Achronix.

(b) Actel.

(c) Altera/Intel.

(d) ARM.

(e) Xilinx.

[5 marks]

Q2.6 The OpenMP stands for … (choose the best description below where the underlining explains the abbreviation)

(a) Open standard for Multi-Processing.

(b) Open standard for Message Passing.

(c) Open standard for Message-based Programming.

(d) Open Multi-platform Programming framework.

(e) Open MicroProcessor architecture (like the Harvard but more flexible).

[5 marks]

Section 3: Long Answers [42 marks]

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

Luke is frustrated that R2D2 has become a bit of a lumpen clunky device which too many compatibility difficulties. It takes hours sometimes to reset and adjust the droid to connect with new technologies, and most of the original code is long gone. R2D2 has decided to go to Droid School in an attempt to alleviate some of these problems and to better serve the Rebellion. Luke particularly wants the droid to go to TIE Fighter class and become able to control these craft – largely for business purposes naturally (although R2 is excited about the prospect of flying something smaller, if more economical that an X-Wing). But droids applying need to pass the TO8AFL (Test Of 8-bit ASCII as a Foreign Language) entrance examination. R2D2, being a bit clunky and verging on obsolete (Luke would never say that aloud), supports only 4-bit ESQM (Encoding for SQueak Modulation).

Luke is pretty useless at modern electronics, or programming for that matter (being something of a veteran now), and is seeking your assistance to help develop an ESQM ←→ ASCII8 encoding module for R2D2. An off-the-shelf mini FPGA kit with lots of PIO is to be used for this upgrade.

Question 3.1 [26 marks]

The ESQM ←→ 8bit ASCII encoding module (let's call it the E2A) is going to be provided as a FPGA-based add-on, and the coding needs to be developed in Verilog (or VHDL) (if you choose Verilog, you can use Verilog95, Verilog2001 or SystemVerilog).

An architectural diagram of the encoder is shown below. As you can see, there are three interfaces for the E2A module. These are described as follows:

- ANM Interface (ports on the left): these connect to R2D2's existing annunciation module (ANM), which has signal lines of RESET, CLK, WE, STROBE, etc. These provide data that R2D2 wants to send out. Normally these would connect to the SQSG (Squeak Sound Generator), shown at the top of the diagram, which would generate modulated audible sound transmission in the sub 4KHz range. ADATA is a 4-bit bus.
- SQSG Interface (ports on the top): these link ANM lines to the SQSG when EN_CONV is low (see below). DATAI is a 4-bit bus that simply links to ADATA when not in translate mode.
- AIO Interface (ports on the right): when EN_CONV is high input from ANM needs to be translated and send on to the AIO module to send to ASCII devices. DO and DI are 8-bit busses.

The E2A needs to run state machine(s) that converts incoming 4-bit data sequences into 8-bit bytes (ASCII) following the procedure described in pseudocode below. R2D2's system clock (running at 100MHz) is fed into the E2A as well, this can be useful in ensuring data gets sent out the AIO interface speedily.

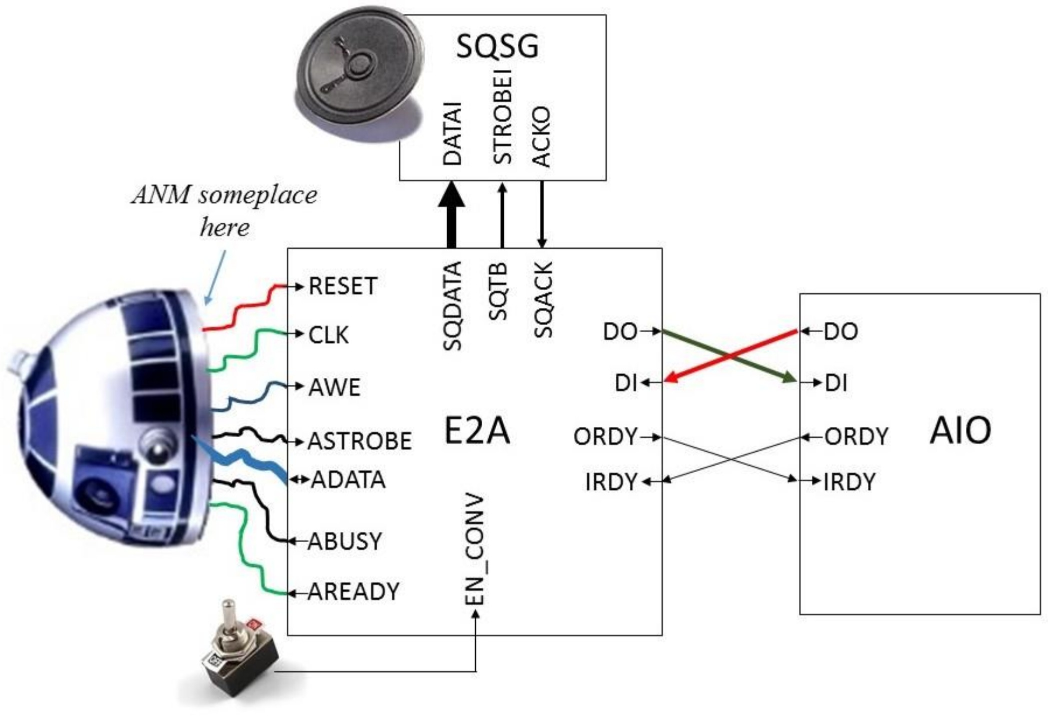


Figure showing connections to the E2A module.

The E2A works as follows:

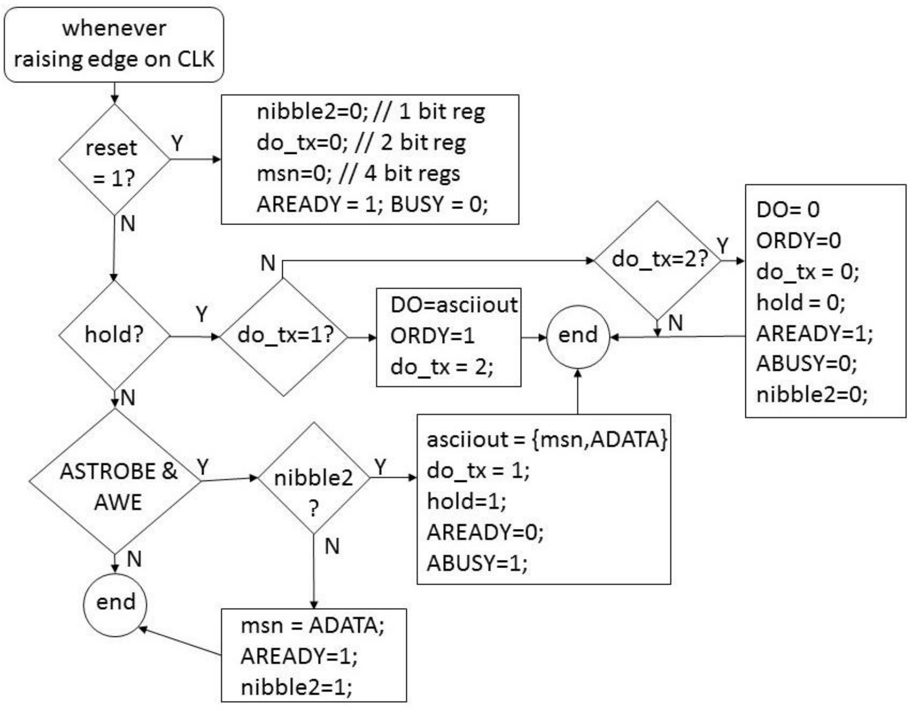
NB: All the inputs to the E2A are synchronized to CLK (although inputs from AIO technically are not synchronized to CLK they only need to be checked on CLK changes since CLK is fast enough).

- When RESET is high
    - ABUSY, SQTB, ORDY set to 0 is set to 0 and AREADY set to 1.
    - All bits of SQDATA and DO set to 0
- When EN_CONV is low
    - DO and ORDY set low
    - When AWE and ASTROBE are high
        - SQDATA output is assigned to ADATA input values
        - SQTB is assigned to ASTROBE, and set back to 0 in next clock.
    - The SQACK input of E2A is ignored.
- When EN_CONV is high
    - Run the ESQM2ASCII operation (see flowchart below)
    - [We will not bother to handled incoming ASCI i.e. not doing ASCII2ESQM for this question]

<u>ESQM2ASCII operation: Flowchart for converting ESQM to ASCII.</u>

This operation involves converting input from the ANM to output sent to the AIO module. The operation involves a state machine that has been presented as flow charts below. Data from ANM are 4-bit values (i.e. nibbles) send on the ADATA line; two nibbles are sent in sequence, the MSN (most significant nibble) first and then the LSN. On receiving the second nibble the MSN and LSN are placed into the 8-bit asciiout register for transmission to the DO output on the next CLK pulse. The DO output is kept for only one clock period, with ORDY high, and then DO returned to low and ORDY set low. Clearly there is only one clock domain, CLK which simplifies the operation[2].

[2] After much deliberation I decided not to make ASTOBE another clock domain.



TODO: So in summary what you need to do is implement the Verilog or VHDL code for the E2A module which must implement the operation described in the bullet points list above and the flowcharts above. If you feel any additional modules are needed (e.g. a separate module to implement ASCII2ESQM if you feel inspired, you can choose to do so).

NOTE: To save you time you can assume the following Verilog (and equivalent VHDL) starting point is given so you don't have to write out the module interface.

```
module E2A (
    // Inputs
    RESET,CLK,AWE,ASTROBE,EN_CONV,SQACK,DI,IRDY,
    // Outputs
    ADATA, ABUSY,AREADY,SQDATA,SQTB,DO,ORDY
    );
  // Declare the directions of size of each input
  input RESET,CLK,AWE,ASTROBE,EN_CONV,SQACK, IRDY;
  input [7:0] DI;
  // Declare the directions of size of each input
  output reg ABUSY,AREADY,SQTB,ORDY;
  output reg [3:0] SQDATA;
  output reg [7:0] DO;
  // Declare the directions of size of each tristate (inout)
  inout [3:0] ADATA;
  // Internal registers for ESQM2ASCII state machine
  reg nibble2;
  reg [1:0]do_tx;
  reg msn;
  reg hold;
  reg [7:0] asciiout;

  // Start of code    🡸 PUT YOUR CODE STARTING FROM HERE!!

endmodule
```

[36 marks]

Question 3.2 MPI [16 marks]

This is a short-ish (or rather let's say not excruciatingly long) question concerning MPI coding…

The client wants an application that can find which Fibonacci[3] numbers between F_{n1} and F_{n2} are divisible by a given integer *D*. Assume F_n represents the *n*th Fibonacci number counting from 0. The user needs to input the integer numbers *n1* and *n2* (i.e. the user doesn't type in the Fibonbacci numbers themselves but rather their Fibonacci index *n* for F_n).

For example, given *n1* = 0 and *n2* = 6 and *D* = 2, we want to know which of these Fibonacci numbers in the range, i.e. F_0, F_1, F_2, F_3, F_4, F_5, F_6, are divisible by 2. To solving this: The Fibonacci numbers to consider in this case are {0,1,1,2,3,5,8} so the answer to this example is: $F_3 = 2$ and $F_6 = 8$.

Implement an MPI program in C / C++ that does this (note that C++ cin and cout often doesn't work with MPI so it is safer to use scanf and printf).

To start with:

- **Assume *n1*, *n2* and *D* are all constant values** in the program (no need to input these).
- Display some **cool welcome message** for good measure (just one node to do this!).
- The **Fibonacci function is provided** that returns the Fibonacci number F_x for a given integer *x* (see below, you don't need to write it out again).

Now get into writing some MPI code for an application that will run on *num_procs* nodes, such that:

1. All processors are to check '*todo*' Fibonacci numbers where *todo = (N2-N1+1)/num_procs* Fibonacci numbers. Each processor searches its subgroup starting from my_rank * todo (hint use: MPI_Comm_rank (MPI_COMM_WORLD, &my_rank) to get the processor's rank or ID, where rank=0 is the master).
2. Each processor (including the master) needs to simply print each Fibonacci numbers in its subrange that is divisible by D.
3. The master has to check any remaining Fibonacci numbers in the range that have not been checked.

```
int fibonacci(int x)
{  /* Complimentary implementation of Fibonacci function to save you time */
   if ( x == 0 ) return 0;
   else if ( x == 1 ) return 1;
   else return ( Fibonacci(x-1) + Fibonacci(x-2) );
}
```

For this this task you only need to provide the code. Having some comments will be favorable and improve the mark for your solution. No #includes are needed.

END OF EXAMINATION

[3] In the unlikely event that you have forgotten about dear old Fibonacci, the hint is: $F_n=F_{n-1}+F_{n-2}$ with $F_0=0$, $F_1=1$

Appendix A: Formulae

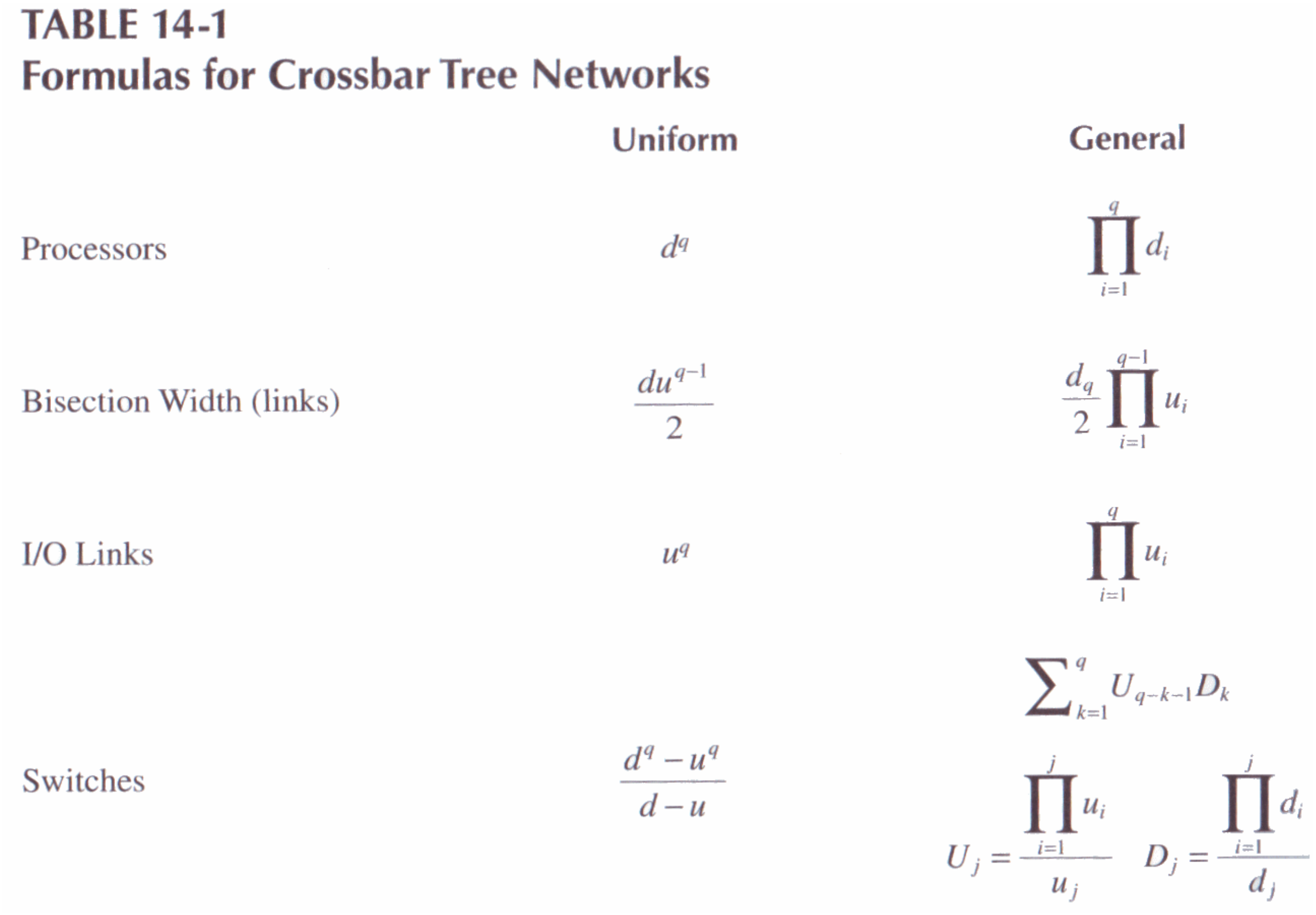


TABLE 14-1
Formulas for Crossbar Tree Networks

	Uniform	General
Processors	d^q	$\prod_{i=1}^{q} d_i$
Bisection Width (links)	$\frac{du^{q-1}}{2}$	$\frac{d_q}{2}\prod_{i=1}^{q-1} u_i$
I/O Links	u^q	$\prod_{i=1}^{q} u_i$
Switches	$\frac{d^q - u^q}{d-u}$	$\sum_{k=1}^{q} U_{q-k-1} D_k$ $U_j = \frac{\prod_{i=1}^{j} u_i}{u_j}$ $D_j = \frac{\prod_{i=1}^{j} d_i}{d_j}$

Table from pg. 291 of Martinez, Bond and Vai 2008

Appendix B: 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 (ˆ), 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
```

Appendix C: OIC Instructions

One Instruction per Clock (OIC) Instruction Set

The OIC is an 8-bit processor that can address only 256 bytes of memory and get input/output via port access to 256 port addresses. Each register-only instruction takes one byte to store; each instruction using a constant takes two bytes, one for the instruction the other for the 256 byte constant value.

OIC Instructions

Instruction	***Description***
IN X, *reg / const*	Input value into X from port *reg* or from construct *const*
IN Y, *reg / const*	Input value into Y from port *reg* or from construct *const*
OUT X, *reg / const*	Output accumulator to port *reg* or constant *const*.
LD X, [*reg*]	X = mem[*reg*]
ST X,[*reg*]	mem[*reg*] = X
CLR [*reg*]	*reg* = 0
ADD X, *reg / const*	X = (X + *reg*) *or* X = (X + *const*)
ADD Y, *reg / const*	Y = (X + *reg*) *or* Y = (X + *const*)
AND X, *reg / const*	X = (X and *reg*) *or* X = (X and *const*)
AND Y, *reg / const*	Y = (X and *reg*) *or* Y = (X and *const*)
OR X, *reg / const*	X = (X or *reg*) *or* X = (X or *const*)
OR Y, *reg / const*	Y = (X or *reg*) *or* Y = (X or *const*)
SWP *reg1*, *reg2*	Swap values stored in reg1 and reg2
NOT X, *reg*	X = not *reg*
ROR reg, *n*	X = rotate right *reg* by *n* bits
ROL reg, *n*	X = rotate left *reg* by *n* bits
NEG X	X = -X
NEG Y	Y = -X
CMP X, *reg*	Compare X and *reg* *(reg can be any reg A-H or X or Y)*
JMP *reg*	Jump to address pointed to by *reg* (JMP X is the same as SI)
JMP *const*	Jump to address *const*, i.e. I = *const*
JMP*f* *reg*	Jump to address pointed to by *reg* if flag *f* set
JMP*f* *const*	Jump to address pointed to by *const* if flag *f* set
SKIP	Skip the next instruction unconditionally
SKIP*f*	Skip the next instruction only if conduction flag f is set…
LI	X = I (i.e. load instruction pointer into X)
SI	I = X (i.e. jump to address X)

OIC Registers

Register Name	***Register Description***
X	Accumulator
Y	General purpose 8-bit register, can also store results from some ALU operations
A – F	General purpose 8-bit register
H	Flag register (can also be used as input to ALU operations)
I	Instruction pointer – cannot be used with ALU operations beside LI or SI

OIC Condition Flags (e.g. "JMPz label" will jump to label if zero flag is set)

Flag Name	***Flag Description***
z	Zero / accumulator result was 0 or comparison was equal
eq	Comparison was equal or accumulator result was equal to 0
gt	Set true if *reg* > A for CMP A, *reg* or accumulator result was >0
lt	Set true if *reg* < A for CMP A, *reg* or accumulator result was <0
gte	Set true if *reg* >= A for CMP A, *reg* or accumulator result was >=0
lte	Set true if *reg* <= A for CMP A, *reg* or accumulator result was <=0
c	Set true if carry occurred in accumulator

Examples of converting C into OIC

For loop:

C code	OIC assembly
`void test ()`	`; OIC for loop`
`{`	`AND X,0`
`   BYTE i;`	`ADD X,10`
`   for (i=0; i<10; i++) OUT(1,i);`	`SWP X,A   ; A = 10`
`}`	`AND X,0`
	`ADD X,1`
	`NEG X`
	`SWP A,B   ; B = -1`
	`AND X,0   ; X = 0`
	`Loop:`
	`CMP X,A   ; X<A ?`
	`JMPgte Exit  ; if (X>=A) return`
	`OUT X,1   ; output X to port 1`
	`ADD X,B   ; X = X - 1`
	`JMP Loop`
	`Exit:`

Appendix D: MPI Cheat sheet

Initialize & Finalize	
MPI_Init(&argc, &argv);	Called at start of program before using any other MPI functions
MPI_Finalize();	Called once done with using MPI
Information about the collaborating nodes	
MPI_Comm_rank (MPI_COMM_WORLD, &my_rank);	Returns the process ID of the current process (a value between 0 and number_processors-1 inclusive)
MPI_Comm_size (MPI_COMM_WORLD, &num_procs);	Returns number of processes available, typically called after MPI_Init.
MPI_Get_processor_name (&name, &result_length)	Returns the name of the host running on (generally not used)
Passing Messages	
MPI_Send (message, message_size, MPI_CHAR, destination, tag, MPI_COMM_WORLD);	Send a message from the current process to another process (indicated by the destination ID). If message is bytes (MPI_CHAR) message_size is size in bytes. MPI_CHAR can be replaced with other type e.g. MPI_INT but I put char because that's more general.
MPI_Recv (message, max_message_size_in_byes+1, MPI_CHAR, source, tag, MPI_COMM_WORLD, &status)	Receives a message on the current process from another process. Note re source parameter: Set source = MPI_ANY_SOURCE to wait on and receive a message from any source. Otherwise source is set to the ID of the processor to receive a packet from. MPI_CHAR can be replaced with other type e.g. MPI_INT but I put char because that's more general.
MPI_Bcast(message, message_size, MPI_CHAR, source, MPI_COMM_WORLD);	Broadcasts message from current process to all other processors.
MPI_Reduce(&value, &value_collect, count, type, MPI_Op, server, MPI_COMM_WORLD);	Performs a reduction (e.g., summation or find min) of a variable on all processes, sending the result to a single process. Type could be e.g. MPI_CHAR. Built in ops: PI_MAX, MPI_MIN, MPI_PROD, MPI_SUM, MPI_LAND, MPI_LOR, MPI_LXOR, MPI_BAND, MPI_BOR, MPI_BXOR, MPI_MAXLOC, MPI_MINLOC in place of MPI_Op
MPI_Allreduce(&value, &value_collect, count, type, MPI_Op, MPI_COMM_WORLD);	Performs a reduction of a variable on all processes, and sends result to all processes (takes longer)
MPI Data Types	
MPI_CHAR	char
MPI_INT	int
MPI_FLOAT	float
MPI_DOUBLE	double