Digital System Design

by
Dr. Lesley Shannon
Email: lshannon@ensc.sfu.ca
Course Website: http://www.ensc.sfu.ca/~lshannon/courses/ensc350

Simon Fraser University

Slide Set: 10
Date: March 4, 2009

Slide Set Overview

- Midterm
  - Results
  - Answers

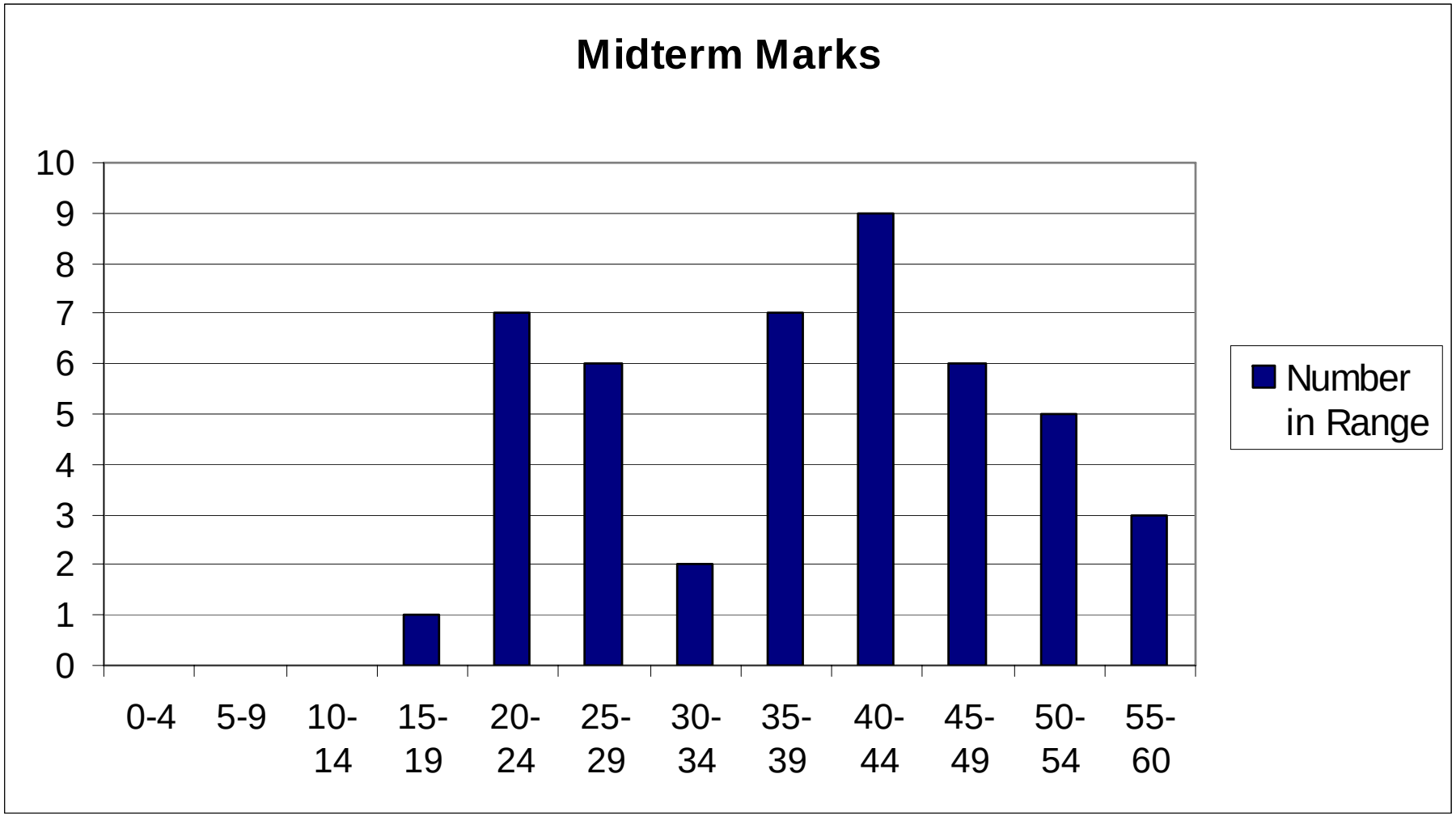
What your marks mean in terms of your final mark

• 53.5	18%
• 47.5	16%
• 41.5	14%
• 35.5	12%
• 29.5	10%
• 24	8%
• 18	6%
• 12	4%
• 6	2%

How did everyone do

- The “bad” questions:
  - 1. & 4. were the worst
  - 2. & 3. weren’t great =(
- Average
  - Mean: 38.22 (Std Dev: 11.53)
  - Median: 40
  - Mode: 41
  - Maximum: 60.5
  - Minimum: 15

How did everyone do

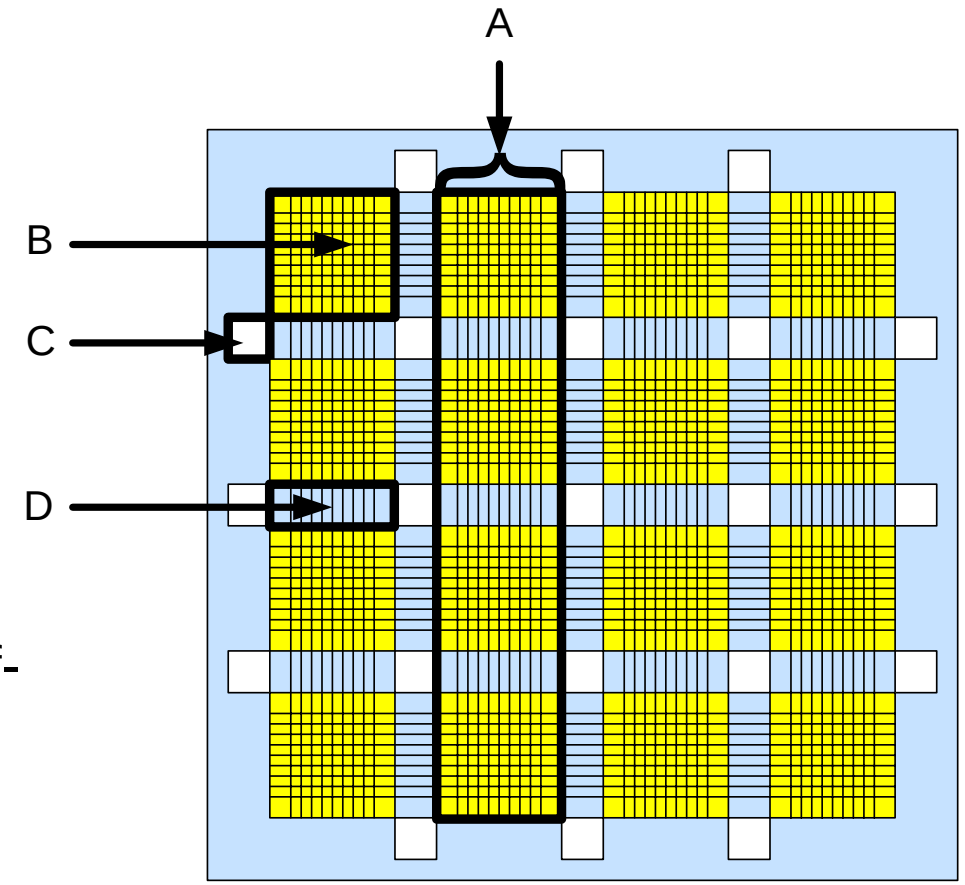


Answers 1.

- Simulated Annealing:
  - A Placement algorithm
  - Minimizes: WIRE LENGTH (accepted cost and area this time only)
  - A Heuristic; Completes in non-polynomial time
- Flowmap:
  - A Technology Mapping algorithm
  - Minimizes: DELAY
  - An Optimal Algorithm; Completes in polynomial time
- Pathfinder:
  - A Routing algorithm

Answers 2.

- A: Routing Channel
- B: Switch Block
  - Enables between vertical and horizontal tracks
- C: CLB/ I/O Pad
  - Implements circuit logic (LUTs and Flipflops)
  - Provides connection with off-chip I/O supporting different transmission standards
- D: Connection Block
  - Enables I/O connections with CLBs to routing tracks



Answers 2. (cont'd)

- Other FPGA components
  - Embedded memory, Multipliers/DSP, Dedicated Clock circuitry/DCMs/PLLs, CPUs, High Speed I/Os
- Specific Component consuming the most silicon area
  - Switch Blocks: 31% of Area

Answers 3.

- When do processes execute?
  - Processes don't 'execute' in hardware: they are only a behavioural description of the circuit to be implemented on the chip

    OR

  - Processes execute in a simulator when an event occurs on one of the signals in the sensitivity list; if there is no sensitivity list, a process executes continuously
- Different statements that specify a rising clock edge
  - (clk = 1) and (clk'event)
  - (clk = 1) and (not (clk'stable))
  - rising_edge(clk) *** This may not be synthesizable depending on the tools***

Answers 3.

- The difference between combinational and sequential logic:
  - Combinational logic: Has no memory; Outputs are only a function of the current inputs
  - Sequential logic: Has memory (aka “state”) such that the outputs are a function of historical information

Answers 4.(i)

```
entity adder_subtractor is
    port(in1, in2:          IN      STD_LOGIC_VECTOR (3 DOWNTO 0);
         sub_not_add:       IN      STD_LOGIC;
         result:            OUT     STD_LOGIC_VECTOR (4 DOWNTO 0));
end unknown;

architecture structural of adder_subtractor is
  signal cin:    STD_LOGIC_VECTOR (4 DOWNTO 0); -- ripple carry
                                                -- signal
  in2_m:         STD_LOGIC_VECTOR (3 DOWNTO 0); -- “muxed”
                                                -- in2/in2_bar

  component full_adder is
      port(a, b, cin:       IN      STD_LOGIC;
           cout, sum:       OUT     STD_LOGIC);
  end component;
```

Answers 4.(i)

```
begin
    in2_m(0) <= in2(0) xor sub_not_add; --Use sub_not_add signal to
    in2_m(1) <= in2(1) xor sub_not_add; --decide whether the in2 vector
    in2_m(2) <= in2(2) xor sub_not_add; --should or should not be inverted
    in2_m(3) <= in2(3) xor sub_not_add; --Invert when sub_not_add = 1
--Generate the Full Adder chain using your Full Adder component:
FA0:  full_adder port map (in1(0), in2_m(0), cin(0), cin(1), result(0));
FA1:  full_adder port map (in1(1), in2_m(1), cin(1), cin(2), result(1));
FA2:  full_adder port map (in1(2), in2_m(2), cin(2), cin(3), result(2));
FA3:  full_adder port map (in1(3), in2_m(3), cin(3), cin(4), result(3));
-- Initialize the cin to the adder/subtractor; provides the additional “plus one”
-- required for twos complement
    cin(0) <= sub_not_add;
-- The final carry out bit provides the “sign bit” for the 2’s complement
-- number:
    result <= cin(4);
end architecture;
```

Answers 4.(i)

```
entity full_adder is
    port(a, b, cin:         IN      STD_LOGIC;
         cout, sum:         OUT     STD_LOGIC);
end full_adder;

architecture structural of adder_subtractor is
  begin
    sum <= a xor b xor cin;  --sum is high for an odd number of '1's
    cout <= (a and b) or (a and cin) or (b and cin); -- carry is high when
                                                     -- more than 1 (>1)
                                                     -- input is high
end structural;
```

Answers 4.(i) Marking Scheme

-1 adder entity
-1 sum
-1 carry
-1 architecture
-1 COMMENTS

-1 component
-1 cin assignment for bit 0
-4 Full Adder instantiations
-1 in2 inversion
-1 in2 multiplexing
-2 comments

Answers 4.(ii) Marking Scheme

-1 sum
-2 carry
-1 a's connection
-1 b's connection

-1 in1
-1 invert in2
-2 mux in2/inv_in2
-1 mux select
-1 cin(0) = sub_not_add
-1 result
-1 cout = cin chain
-1 replication
-1 labels

+1 result(4) = cout(3)

Answers 4.(ii)

SEE BOARD WORK FOR CIRCUIT SOLUTION