SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING
Georgia Institute of Technology

Digital Hardware Lecture Manual

SPRING 2000

MICHAEL D. FURMAN AND TYSON S. HALL

All Rights Reserved, Copyright © 1999, M. D. Furman

Table of Contents

INTRODUCTION AND SYLLABUS 2
MIXED LOGIC 5
WRITING A LAB REPORT 11
STATE MACHINE REVIEW 26
VHDL I 32
TIMING AND LOGIC HAZARDS 38
OSCILLOSCOPE DEMONSTRATION 42
VHDL II 44
MIDTERM REVIEW 51
THE ELECTRIC TRAIN CONTROLLER 53
SIMPLE COMPUTER I 60
SIMPLE COMPUTER II 65
SIMPLE COMPUTER III 69
INTRODUCTION TO THE UP1-BOT 73
PEER REVIEW SURVEY 76

Introduction/Syllabus

ECE 2031: Digital Design Lab

Michael D. Furman
michael.furman@ece.gatech.edu
Phone: (404) 894-9483
Office: CoC 305

ECE 2031 Overview

- Lecture
  - Mandatory
  - Video – RETP
- Laboratory
  - Prelab Exercises
  - In-lab Exercises
  - Reports

Resources

- Tyson Hall, Lead TA
  - gte048g@prism.gatech.edu
  - Phone/Office: TBA
- In lab GTA
- Class Web Page
  - www.ece.gatech.edu/research/labs/diglab

ECE 2031 Supplies Needed

- *Rapid Prototyping of Digital Systems* by James O. Hamblen/Michael D. Furman
- The ECE 2031 Chipset
- A protoboard
- The Wire Kit
- 2 – 3.5" Floppy Disks
- 1 – PC-formatted ZIP disk
- The ECE NT Account

Lab Logistics

- Lab Quizzes
- Homework
- Mid-Term
- Lab Final

Grades

- A final grade will be assigned based on the following submitted work.
  - 30% - Lab Reports
  - 30% - Final
  - 20% - Prelab Quizzes and Homework
  - 10% - Midterm
  - 10% - TA Perspective

Other Ancillary Concerns

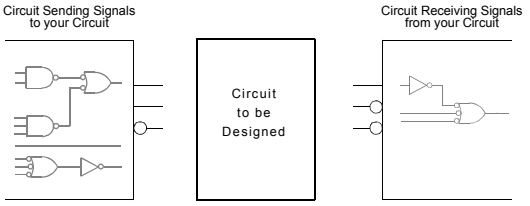
- 50% Check-off
- Students must remain in the lab during the three-hour period.
- No lab report will be accepted without proper check-offs.
- Failure to turn in an appropriate lab assignment will result in an F in the course.
- No make-up quizzes.

Final Comments

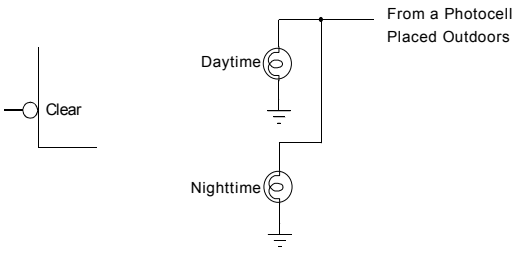
- See schedule on page vii of the lab manual.
- Lecture Homework
  - Read the Mixed Logic Tutorial (Appendix A in the lab manual)
- Video Demonstration

Mixed Logic

Motivation



Active High/Low



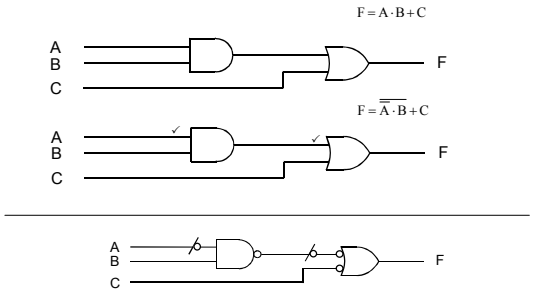
Mixed Logic Steps

1. Draw a circuit using only **AND** and **OR** gates exactly as specified by the equation. Where bars appear over signals or signal groups in the equation, place a checkmark as far to the right as possible and still over the signal line.
2. If there are gate restrictions, apply them next.
3. If there are signal restrictions, apply them next.
4. Balance all line segments by adding appropriate inverters.

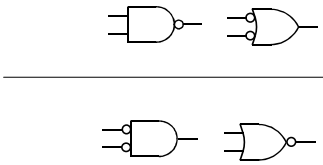
Problem

Equation:	$F = \overline{\overline{A} \cdot B} + C$
Constraints:	A, B: Active Low F: Active High C: Unspecified Use: 2-input NAND's and Inverters

Step 1:

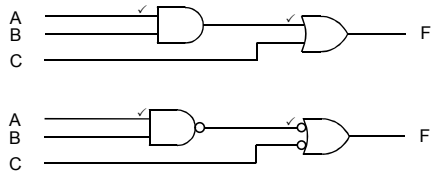


Equivalent Gates



Step 2: Add Gate Restrictions

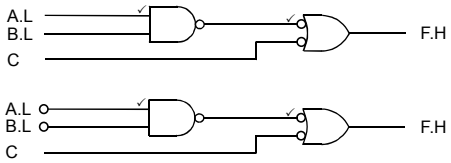
$$F = \overline{\overline{A} \cdot B} + C$$



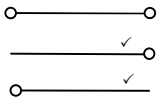
Step 3: Add Signal Restrictions

A, B: Active Low
F: Active High
C: Unspecified

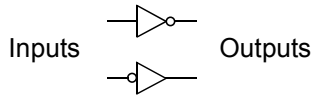
$$F = \overline{\overline{A} \cdot B} + C$$



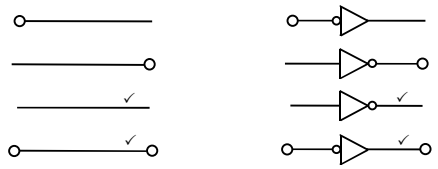
Balanced Line Segments



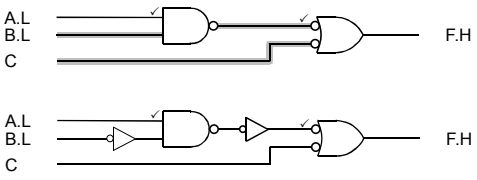
Add Inverters for Balance



Balancing Line Segments

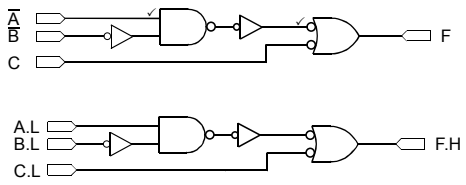


Unbalance Lines in Problem

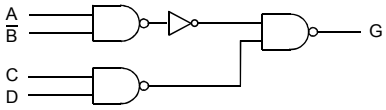


Solutions:

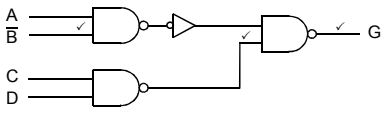
$$F = \overline{\overline{A} \cdot B} + C$$



Reading an Unknown:



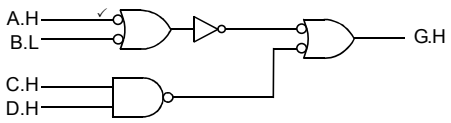
Change inverter and balance:



$$G = \overline{(A \cdot \overline{B}) \cdot \overline{(C \cdot D)}}$$

Just build the original equation

$$G = \overline{A} + B + C \cdot D$$



Lecture Homework

- Read the ASM tutorial (Appendix B in the lab manual)

Writing a Lab Report

Writing Requirement

- Requirement mandated by ABET
  - Establish a Process
  - Establish a Measurement Criteria
- Communications
  - Best ideas are worthless unless communicated.
  - Industry requires it.
  - (We require it too!)

2

Establishing Your Purpose

- To develop your ability to communicate in a technical environment.
- To establish and test that ability through well defined models.

3

Topics we will consider:

- Aligning your perspective.
- Considering your audience.
- Knowing why we are writing.
- Establish what your reports should be.
- Building the right tone.
- Using an acceptable format.
- Looking at some examples.

4

What is Your Purpose For Writing a Report?

- To demonstrate what you know?
- To document what you did?
- To educate your TA?
- To get a good grade?
- To relay a fascinating and personal experience?

5

Know Why You are Writing!

- This defines what is important and what is not.
- Describe things in a way that indicates that you understand what is important without saying, "… is important."
- Look for clues in the assignment – most assignments have a fundamental purpose.
- If all else fails, ask your TA!

6

Documenting What You Did?

- Avoid anthropomorphisms.
- Avoid the "What-you-did" perspective.
- Microsoft beta testing would be interested in what you did or how you felt about something – we're not.
- Document what was done without "we," "I," "the student," etc. (See examples.)
- Consumer's Reports perspective.

7

Anthropomorphisms

- Applying human feelings or actions to inanimate objects.
  - "The experiment demonstrated …."
  - "The flip-flop did not behave until the pull-up resistor was added to the Clear line."
  - "These exercises familiarized the student to the …."

8

What Your Report Should *Not* Be!

- A personal experience:
  - "It took three hours to get the program to work."
    (Unless this was not a function of the person.)
  - "After much trial and error, the student went home and came back later to finish the project."

9

More: What Your Report Should *Not* Be!

- A personal experience:
  - "The student found the project interesting."
  - "The exercises were very effective at keeping the student's interest."
  - "The student dealt with all the problems successfully."

10

Possible Substitutions

- An impersonal experience:
  - "The program worked successfully."
  - "The project was completed on time."
  - "The laboratory represents an interesting collection of exercises."
  - "The exercises form a strong pedagogical basis for teaching the material."
  - "The exercises were finished on schedule."

11

What we don't want:

- Talking about yourself in the third person. (i.e., "The student….")
- What happened ***TO*** you.
- How you "felt" about the outcome.
- Why you chose one way instead of another unless it affects what may have been the results.

12

When to Add Details

- When there is more than one way to do something (be practical).
- When HOW you did something is a choice (same idea).
- When what you did involves a process that is not intuitively obvious.
- When details are required for repeatability.

13

Be Reasonable

- Don't state the obvious: "The unit was plugged in and turned on before taking data."
- "Data was collected from the <state whatever it was> and the results are displayed in Figure 2."

14

More than One Way: Measuring Voltage

- If you've listed an oscilloscope AND a voltmeter in your list of equipment, then you need to tell which was used. If you listed only an oscilloscope then:
  - "Voltage measurements were taken while the circuit was under load, and the results are displayed in Figure 3."

15

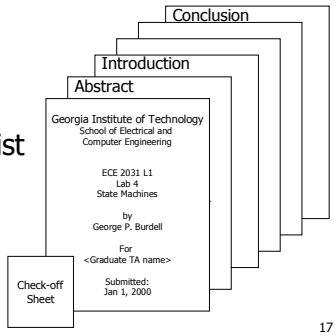
More than One Way: Measuring Rise Time

- Possibilities need to be specified:
  - Manually viewing grid separation on the oscilloscope,
  - Using the cursors to measure the difference on the oscilloscope, or
  - Using the automatic rise-time function on the oscilloscope.

16

The Formal Report

- Title Page
- Abstract
- Introduction
- Equipment List
- Method
- Results
- Conclusion

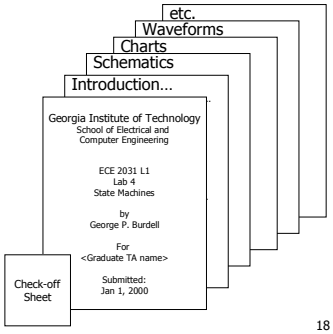


17

The Informal Report

- Title Page
- Introduction
- Equipment List
- Summary
- Conclusion

- Outputs
  - Schematics,
  - Waveforms, …

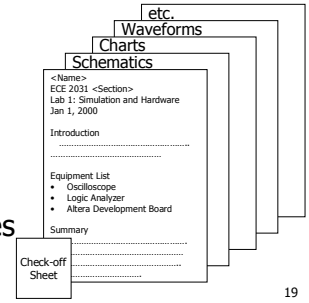


18

The Status Report

This report is interim to the next formal report.

- No Title Page
- Introduction
- Equipment List
- Summary
- Attached Figures



19

Abstract

- VERY brief and clear – summarizing everything done.
- No details. No reference to figures.
- Should be able to read this to a non-technical type.

20

Abstract – Example

"Development of state-machine synthesis is explored with Xilinx FPGA technology. Various comparisons with discrete circuitry indicate that complex multi-state processes can be designed, compiled, and simulated in less than half the time needed for comparable discrete circuits."

21

Abstract – Another Example

"Development of fundamental arithmetic processes are explored with multiplexer and MSI circuits. Performance comparisons are made with relative compromises delineated. MSI circuits significantly outperformed discrete attempts in almost every category."

22

Abstract – Another Example

"Microprocessors are used to control simple processes that do not need high-speed throughput. FPGAs, however, generally sacrifice versatility for faster propagation delays. These two devices are utilized and compared in an identical process to quantify cost, power, and propagation tradeoffs. Conclusions are mixed since application relevance becomes the deciding factor."

23

Abstract – Final Thoughts

- Write this AFTER the report is completed.
- The perspective is moreover, "The next few pages are …."
- All major points are covered without details of any of them.

24

Introduction (Objective)

- The first "abstraction."
- Purpose, objective, history (why), are all factors.
- Processes used to obtain the output in general terms can be included.
- Keep this very short.

25

Introduction – Example

"Multiplexers are utilized to implement full-adder circuits. These modules are then cascaded together to form a 4-bit adder. An adder/subtracter circuit is then created by using XORs as programmable inverters on the inputs of one of the input variables. Power, cost, and propagation delay is calculated and compared with TTL MSI-adder circuits."

26

Introduction – Another Example

"A 555 timer is utilized to generate a 32KHz square wave at a 50% duty cycle. This waveform is then applied to a 16-bit counter to generate a 1Hz signal at the most significant output pin. Utilizing 4-input NAND gates, various patterns from the counter are decoded to generate other intermediate frequencies."

27

Equipment List

- Label this as you would any other heading.
- No explanation, just list the equipment that was used during the given laboratory.

28

Methodology

- State what was done – this is where you avoid the obvious.
- Remember: If there is only one way to do something, leave it out.
- Must be sufficient to be completely reproducible.
- "Voltage was measured," "The Altera CPLD was programmed to …."

29

Methodology – Avoid Redundancies

- Don't say, "Karnaugh Maps were filled in and reduced to generate the simplified equations."
- Instead, "Karnaugh Maps were used to generate the equations."

30

Methodology – Avoid Redundancies

- Don't say, "The logic analyzer was hooked up to the circuit and a timing diagram was generated from the signals on the board."
- Instead, "The logic analyzer was used to generate the timing diagram in Figure 3." – And then only because a timing diagram can be generated by the software waveform generator.

31

Results

- This is final data, not intermediate data.
  - A timing diagram that is used to verify correct operation is not a result.
  - However, a timing diagram that IS the objective would be a result.
- Computed quantitative numbers go here.
- This is NOT an for Conclusions!

32

More on Results

- Calculations
- Power Consumptions
- Propagation Delays
- Costs
- This area should be short, to the point, and report any data that may be important to the Conclusion.

33

Summary

- Combination of Method and Results
- Utilized ONLY in the Informal and Informal Status Reports.
- State what was done, generally in one sentence. Then state the result of what was done, also in one sentence.

34

Summary – Example

- "Frequency and Rise Time of the loaded inverter was measured with the oscilloscope's built-in functions. The results are displayed in Figure 5."
- "A state diagram was developed to perform the collision-avoidance function in the robot. The ASM diagram is illustrated in Figure 6."

35

Conclusions

- Compare, contrast, compare, contrast!
- In simplest matter-of-fact language, state what was good and bad without getting personal.
- Should relate back to Introduction.

36

Conclusions – Example

- "VHDL is a significant improvement over classic synthesis. CPLD propagation delays were 4 times faster, power dissipation was 1/10th standard logic, and development time was relatively a two-orders-of-magnitude improvement. While real estate savings were better than 80%, cost was, however, roughly four times higher."

37

Figures

- Label figures consistently and consecutively.
- Recommended format:

  Figure<space>3.<space><space>Full descriptions taking up no more than one line.

38

Figures – Bad Examples

Figure 2. Karnaugh map.

Figure 3. Timing diagram.

Figure 4. Power dissipation.

39

Figures – Better Examples

Figure 2. Karnaugh map of XOR circuit.

Figure 3. Timing diagram of up-down counter.

Figure 4. Power dissipation of ripple counter.

40

Correct

- Do not use a bulleted format!
- No single spacing – use double spacing.
- Font for text should be 11 or 12 pt. Font for Title page should largest font used.
- Check-off sheet should be first page.
- Figures need to be specific and unambiguous.
- Do not start sections on separate pages.

41

Correct (Cont.)

- Software used (Max+PlusII) should be listed.
- All things MUST be checked off.
- Do not leave sections out.
- Put sections in indicated order.
- Summaries must be complete such that experiment is REPRODUCABLE.
- Log off the computer.

42

Sections

- Abstract
  - No Figure references here.
- Introduction
  - Make specific to what you are doing.
  - No Figure references here.

43

Summaries

- Summarize. Don't say, "Here it is."
- Make sure you describe what methods were used to obtain your results.
- All figures are referenced here.

44

Conclusions

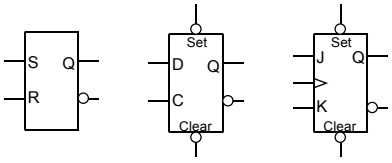
- If you have to build multiple designs, you should compare and contrast the multiple designs.
- What worked best? Which was simpler to design?

45

State Machine Review

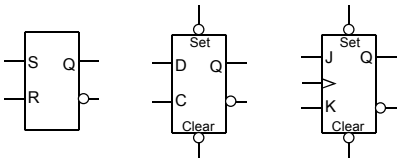
Flip-flops and Latches

- S-R (Set/Reset)
- D
- J-K



2

Flip-flops and Latches



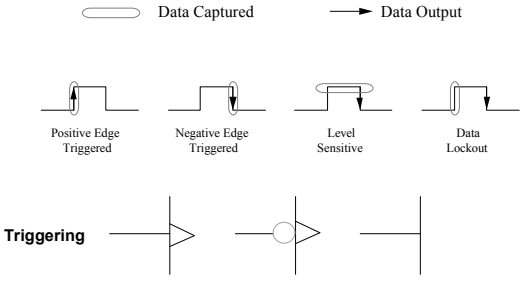
S	R	Q_{t+1}
0	0	Q_t
0	1	0
1	0	1
1	1	?

D	Q_{t+1}
0	0
1	1

J	K	Q_{t+1}
0	0	Q_t
0	1	0
1	0	1
1	1	$/Q_t$

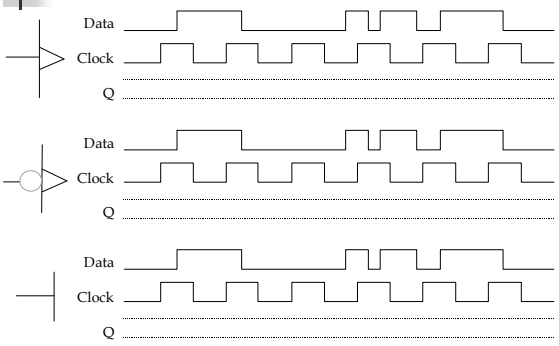
3

Clocking



4

Clocking Exercise



5

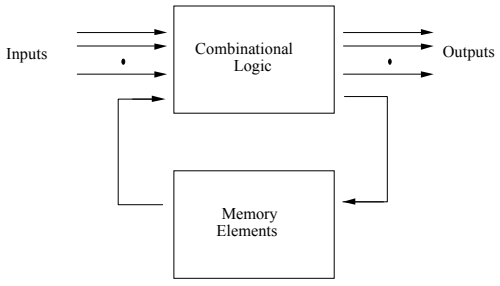
Transition Table *

Q → Q+	S R	D	J K
0 0	0 d	0	0 d
0 1	1 0	1	1 d
1 0	0 1	0	d 1
1 1	d 0	1	d 0

The transition table is the key table used in implementing state machines.

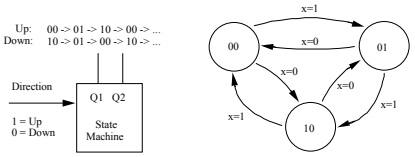
6

State Machines



7

State Machines – The Problem



8

General Truth-Table Implementation

Input	Present State		Next State		Required Input to Flip-Flop
X	Q1	Q0	Q1+	Q0+	
0	0	0	1	0	For a D-type Flip-Flop D1 = Q1+ and D0 = Q0+ (See the Flip-Flop Transition Table)
0	0	1	0	0	
0	1	0	0	1	
0	1	1	d	d	
1	0	0	0	1	
1	0	1	1	0	
1	1	0	0	0	
1	1	1	d	d	

Note: The circled areas are a case from the state diagram.
"If the input is 0 and the state = 2, count down to 1."

9

Implementation with D Flip-Flops

Q1Q0 \ X	0	1
00	1	0
01	0	1
11	d	d
10	0	0

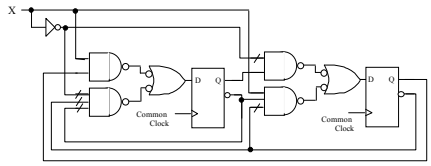
$D1 = X\,Q0 + X\,\overline{Q1}\,\overline{Q0}$

Q1Q0 \ X	0	1
00	0	1
01	0	0
11	d	d
10	1	0

$D0 = X\,\overline{Q1}\,\overline{Q0} + X\,\overline{Q1}$

10

Implementation with D Flip-Flops



Outputs are taken from Q1 and Q0.
Also note that Qn and $\overline{Qn}$ are available on the flip-flop.

11

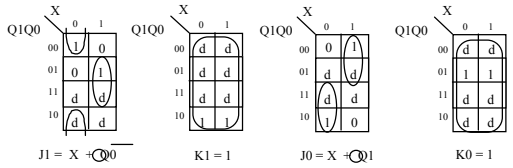
Implementation with J-K Flip-Flops

Input	Present State		Next State		Required Input to Flip-Flops			
					for Q1->Q1+		for Q0 ->Q0+	
X	Q1	Q0	Q1+	Q0+	J1	K1	J0	K0
0	0	0	1	0	1	d	0	d
0	0	1	0	0	0	d	d	1
0	1	0	0	1	d	1	1	d
0	1	1	d	d	d	d	d	d
1	0	0	0	1	0	d	1	d
1	0	1	1	0	1	d	d	1
1	1	0	0	0	d	1	0	d
1	1	1	d	d	d	d	d	d

12

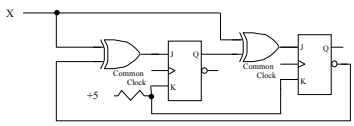
Implementation with J-K Flip-Flops

To implement the JK system, Karnaugh Maps are drawn for J1, K1, J0, and K0. The results are as follows:



13

Implementation with J-K Flip-Flops

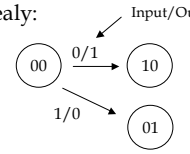


Note that the JK implements into a much simpler circuit. The output is also taken from Q1, Q0.

14

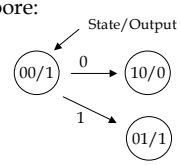
Mealy vs. Moore Implementation

Mealy:



X	Q1	Q0	Q1+	Q0+	Z
0	0	0	1	0	1
0	0	1	?	?	?
0	1	0	?	?	?
0	1	1	?	?	?
1	0	0	0	1	0
1	0	1	?	?	?
1	1	0	?	?	?
1	1	1	?	?	?

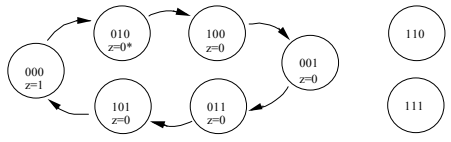
Moore:



X	Q1	Q0	Q1+	Q0+	Z
0	0	0	1	0	1
0	0	1	?	?	1
0	1	0	?	?	0
0	1	1	?	?	?
1	0	0	0	1	1
1	0	1	?	?	1
1	1	0	?	?	0
1	1	1	?	?	?

15

Exercise: Implementing with J-K



*Note that output (ie. "z=0") is a function of the given state and is therefore placed inside the circle. When z is a function of the given state and an input, it is placed on the transition line.

16

Exercise: Implementing with J-K

Present State			Next State			Required Input to Flip-Flops						Output
Q2	Q1	Q0	Q2+	Q1+	Q0+	J2	K2	J1	K1	J0	K0	Z
0	0	0	0	1	0	0	d	1	d	0	d	
0	0	1	0	1	1	0	d	1	d	d	0	
0	1	0										
0	1	1										
1	0	0										
1	0	1										
1	1	0										
1	1	1										

17

Lecture Homework

- Watch the online video tutorial on building ASM charts.

18

VHDL I

What is VHDL?

- VHSIC:
  - Very High Speed Integrated Circuit
- Hardware
- Description
- Language

2

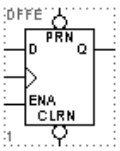
History of VHDL

- 1980: Developed by Dept. of Defense
- 1983: TI, IBM, and Intermetics
- 1987: IEEE ratified Standard 1076
- 1993: IEEE revised Standard 1076 '93

3

Need for Concept

- Suppose you want to build a circuit with VHDL
  - What does it look like from the outside?
  - What does it look like from the inside?



4

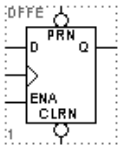
The outside view: "Entity"

```
ENTITY __entity_name IS

   PORT
   (
        __d_input_name     : IN    STD_LOGIC;
        __clk_input_name   : IN    STD_LOGIC;
        __clrn_input_name  : IN    STD_LOGIC;
        __ena_input_name   : IN    STD_LOGIC;
        __q_output_name    : OUT   STD_LOGIC
   );

END __entity_name;
```

5

The "Entity" Declaration

```
ENTITY my_DFF IS

   PORT
   (
        D      : IN    STD_LOGIC;
        CLK    : IN    STD_LOGIC;
        CLRN   : IN    STD_LOGIC;
        ENA    : IN    STD_LOGIC;
        Q      : OUT   STD_LOGIC
   );

END my_DFF;
```



6

The "Architecture" or Inside

```
ARCHITECTURE <identity> OF <the entity> IS
   SIGNAL      <declare "internal signals" here>;

BEGIN
   <Describe the behavior of the circuit.>
END <identity>;
```

7

Architecture Template

```
BEGIN
    PROCESS (__clk_input_name, __clrn_input_name)
    BEGIN
        IF __clrn_input_name = '0' THEN
                __q_signal_name <= '0';
        ELSIF (__clk_input_name'EVENT AND __clk_input_name = '1') THEN
                IF __ena_input_name = '1' THEN
                        __q_signal_name <= __d_input_name;
                ELSE
                        __q_signal_name <= __q_signal_name;
                END IF;
        END IF;
    END PROCESS;
    __q_output_name <= __q_signal_name;
END a;
```

8

Architecture (Continued)

```
ARCHITECTURE  behavior  OF  my_DFF  IS
   SIGNAL       internal_q  :  STD_LOGIC;

BEGIN
   <Describe the behavior of the circuit.>
END behavior;
```

9

Architecture (Continued)

```
ARCHITECTURE  behavior  OF  my_DFF  IS
   SIGNAL       internal_q  :  STD_LOGIC;
BEGIN

   PROCESS ( CLK, CLRN )
   BEGIN
        < Anytime CLK or CLRN changes, this happens>
   END PROCESS;

   Q <= internal_q;
END behavior;
```

10

Template after Search/Replace

```
PROCESS (CLK, CLRN)
BEGIN
    IF CLRN = '0' THEN
        internal_q <= '0';
    ELSIF ( CLK'EVENT AND CLK = '1' ) THEN
        IF ENA = '1' THEN
            internal_q <= D;
        ELSE
            internal_q <= internal_q;
        END IF;
    END IF;
END PROCESS;
```

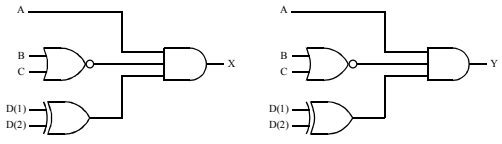
11

Libraries Used

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE ieee.std_logic_signed.all;
```

12

Building Two Circuits

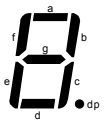


13

```
LIBRARY IEEE;                   -- Include Libraries for standard logic data types
USE  IEEE.STD_LOGIC_1164.ALL;
                                -- Entity name normally the same as file name
ENTITY gate_network IS          -- Ports: Declares module inputs and outputs
      PORT(   A, B, C       : IN   STD_LOGIC;
                              -- Standard Logic Vector ( Array of 4 Bits )
              D             : IN   STD_LOGIC_VECTOR( 3 DOWNTO 0 );
                              -- Output Signals
              X, Y          : OUT    STD_LOGIC );
END gate_network;

                              -- Defines internal module architecture
ARCHITECTURE behavior OF gate_network IS
BEGIN                         -- Concurrent assignment statements operate in parallel
                              -- D(1) selects bit 1 of standard logic vector D
  X <= A AND NOT( B OR C ) AND ( D( 1 ) XOR D( 2 ) );
                              -- Process must declare a sensitivity list,
                              -- In this case it is  ( A, B, C, D )
                              -- List includes all signals that can change the outputs
   PROCESS ( A, B, C, D )
     BEGIN                    -- Statements inside process execute sequentially
       Y <= A AND NOT( B OR C) AND ( D( 1) XOR D( 2 ) );
      END PROCESS;
END behavior;
```

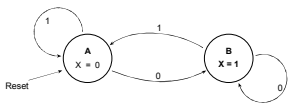


```
LED_MSD_DISPLAY:              -- BCD to 7 Segment Decoder for LED Displays
PROCESS  (MSD)
BEGIN                                 -- Case statement implements a logic truth table
   CASE MSD IS
        WHEN "0000" =>
                MSD_7SEG <= "1111110";
        WHEN "0001" =>
                MSD_7SEG <= "0110000";
        WHEN "0010" =>
                MSD_7SEG <= "1101101";
        WHEN "0011" =>
                MSD_7SEG <= "1111001";
        WHEN "0100" =>
                MSD_7SEG <= "0110011";
        WHEN "0101" =>
                MSD_7SEG <= "1011011";
        WHEN "0110" =>
                MSD_7SEG <= "1011111";
        WHEN "0111" =>
                MSD_7SEG <= "1110000";
        WHEN "1000" =>
                MSD_7SEG <= "1111111";
        WHEN "1001" =>
                MSD_7SEG <= "1111011";
        WHEN OTHERS =>
                MSD_7SEG <= "0111110";
   END CASE;
END PROCESS LED_MSD_DISPLAY;
```



State Machines in VHDL

- Describe the following state machine in VHDL.



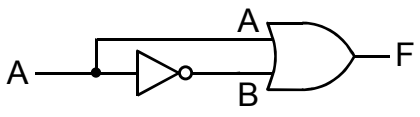
```
LIBRARY IEEE;
USE ieee.std_logic_1164.all;

ENTITY state_1_18 IS
    PORT(
        reset      : IN     STD_LOGIC;
        sm_input   : IN     STD_LOGIC;
        clock      : IN     STD_LOGIC;
        X          : OUT    STD_LOGIC );
END state_1_18 ;

ARCHITECTURE behavior OF state_1_18 IS
    TYPE STATE_TYPE IS ( state_A, state_B );
    SIGNAL state : STATE_TYPE;
BEGIN
  PROCESS ( reset, clock )
  BEGIN
    IF reset = '1' THEN
      state <= state_A;
    ELSIF clock'EVENT AND clock = '1' THEN
      CASE state IS
        WHEN state_A =>
          IF sm_input = '0' THEN
            state <= state_B;
          ELSE
            state <= state_A;
          END IF;
        WHEN state_B =>
          IF sm_input = '1' THEN
            state <= state_A;
          ELSE
            state <= state_B;
          END IF;
        WHEN OTHERS  =>
            state <= state_A;
      END CASE;
    END IF;
  END PROCESS;

  WITH state SELECT
    X <= '0' WHEN state_A,
         '1' WHEN state_B,
         '0' WHEN OTHERS;

END behavior;
```

17

Timing and Logic Hazards

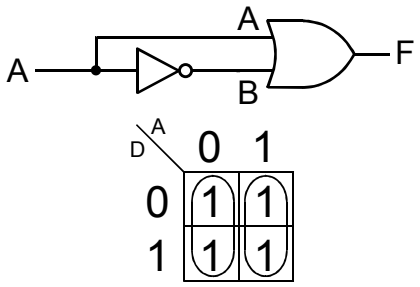
Consider: F = A + /A

- Output should always be High



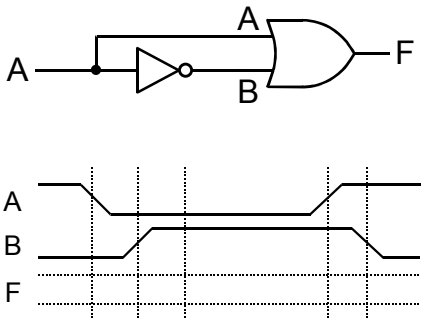
© MDF 1999

K-Map to Circuit looks like:



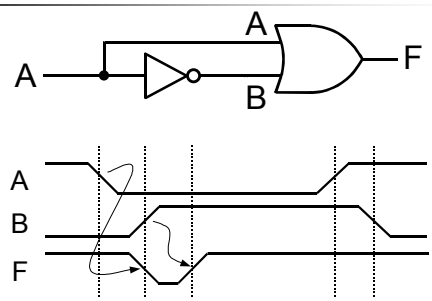
© MDF 1999

Generated Timing



© MDF 1999

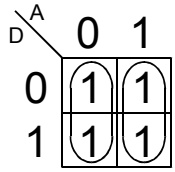
Generated Timing



© MDF 1999

Back to the K-Map

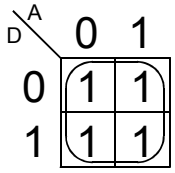
Note: Transition from adjacent groups not grouped together!



© MDF 1999

Solution:

- Combine adjacent groups



© MDF 1999

Function: F = 1

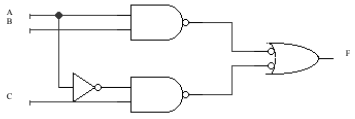
- Seems trivial, but works for more complex cases!

Vcc ——————— F

© MDF 1999

Real Problem:

- This problem has a static hazard. How do we fix it?



© MDF 1999

Karnaugh Map is the solution

- A static hazard shows up as two adjacent 1's that are not in the same group!

/A C ->
A B ->

BC \ A	0	1
00	0	0
01	1	0
11	1	1
10	0	1

© MDF 1999

Combine adjacent 1's

- If we add a redundant term:

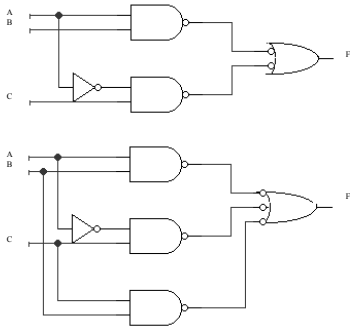
/A C ->
A B ->

+ B C ->

BC \ A	0	1
00	0	0
01	1	0
11	1	1
10	0	1

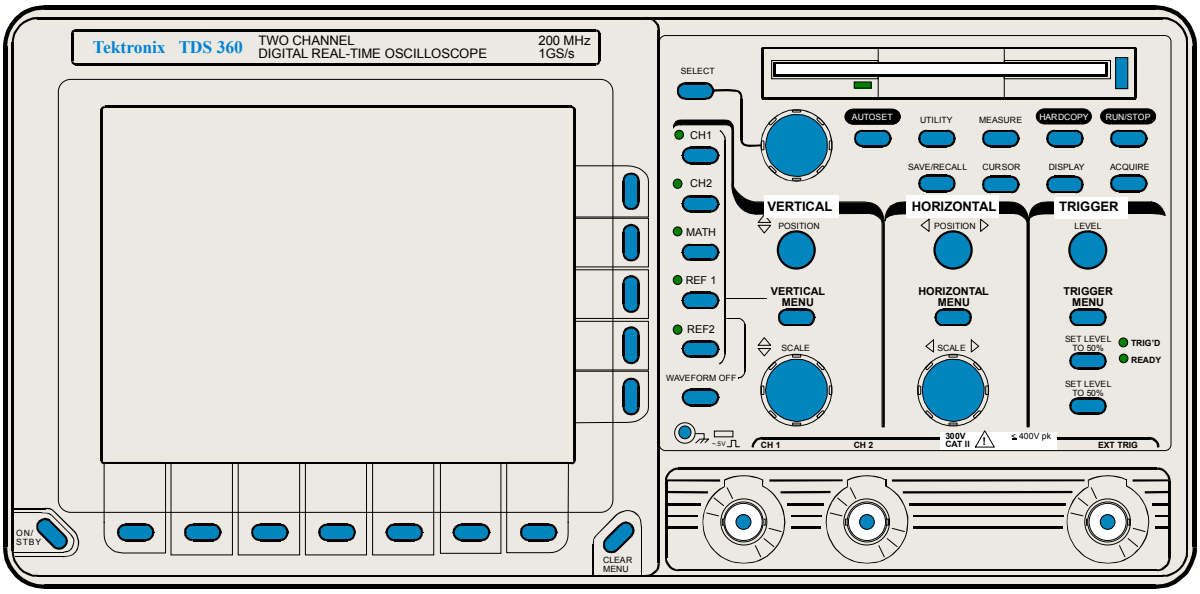
© MDF 1999

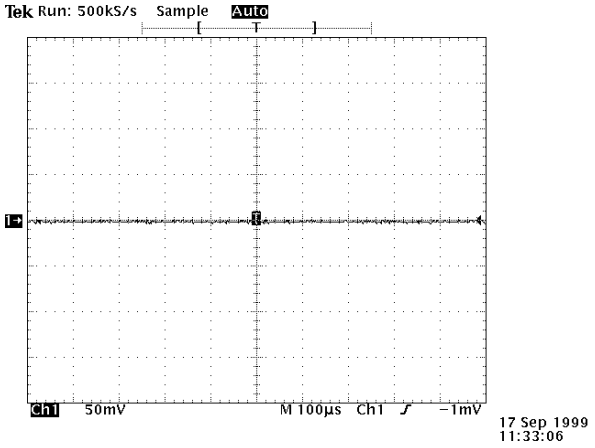
Problem/Solution:

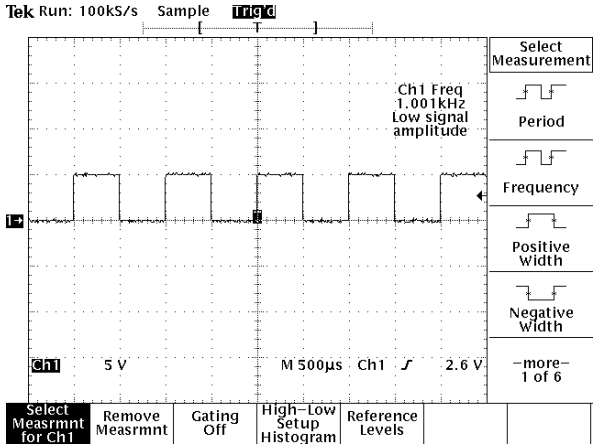


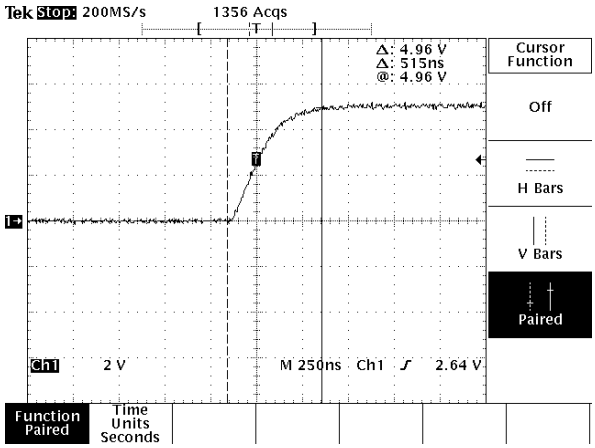
© MDF 1999

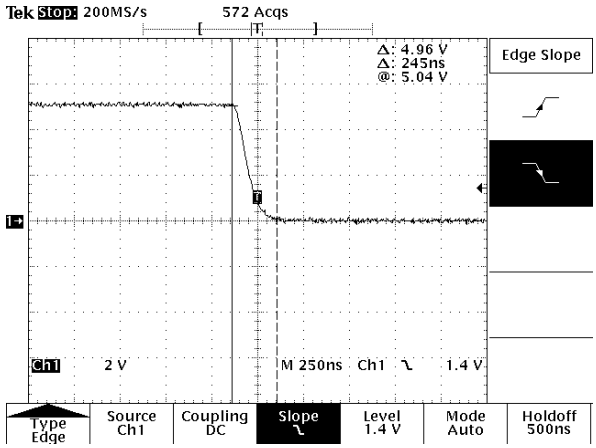
Oscilloscope Demonstration




Tek Run: 500kS/s Sample Auto
Ch1 50mV M 100µs Ch1 ƒ −1mV
17 Sep 1999
11:33:06


Tek Run: 100kS/s Sample Trig'd
Ch1 Freq
1.001kHz
Low signal
amplitude
Select Measurement
Period
Frequency
Positive Width
Negative Width
−more−
1 of 6
Ch1 5 V M 500µs Ch1 ƒ 2.6 V
Select Measrmnt for Ch1
Remove Measrmnt
Gating Off
High−Low Setup Histogram
Reference Levels


Tek Stop: 200MS/s 1356 Acqs
Δ: 4.96 V
Δ: 515ns
@: 4.96 V
Cursor Function
Off
H Bars
V Bars
Paired
Ch1 2 V M 250ns Ch1 ƒ 2.64 V
Function Paired
Time Units Seconds


Tek Stop: 200MS/s 572 Acqs
Δ: 4.96 V
Δ: 245ns
@: 5.04 V
Edge Slope
Ch1 2 V M 250ns Ch1 1.4 V
Type Edge
Source Ch1
Coupling DC
Slope
Level 1.4 V
Mode Auto
Holdoff 500ns

VHDL II

Data Types

- STD_LOGIC
  - A, B, C : IN STD_LOGIC;
- STD_LOGIC_VECTOR
  - D : IN STD_LOGIC_VECTOR(3 DOWNTO 0);

  INTEGER
  - D : IN INTEGER RANGE 0 TO 15;

18

STD_LOGIC

- 'U' -- Uninitialized
- 'X' -- Forcing Unknown
- '0' -- Forcing 0
- '1' -- Forcing 1
- 'Z' -- High Impedance
- 'W' -- Weak Unknown
- 'L' -- Weak 0
- 'H' -- Weak 1
- '-' -- Don't care

19

STD_LOGIC

- Under "ENTITY"
  - PORT(A, B, C : IN STD_LOGIC);
- Under "ARCHITECTURE"
  - SIGNAL A, B, C : STD_LOGIC;
- Under "PROCESS"
  - VARIABLE A, B, C : STD_LOGIC;

20

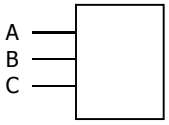
STD_LOGIC_VECTOR

- Under "ENTITY"
  - PORT(A, B, C : IN STD_LOGIC_VECTOR(3 DOWNTO 0));
- Under "ARCHITECTURE"
  - SIGNAL A, B, C : STD_LOGIC _VECTOR(3 DOWNTO 0);
  - TYPE Name IS ARRAY(0 to 7) OF STD_LOGIC _VECTOR(3 DOWNTO 0);
- Under "PROCESS"
  - VARIABLE A, B, C : STD_LOGIC _VECTOR(3 DOWNTO 0);

21

STD_LOGIC under "ENTITY"

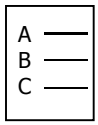
- PORT(A, B, C : IN STD_LOGIC);



22

STD_LOGIC under "ARCHITECTURE"

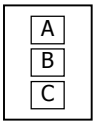
- SIGNAL A, B, C : STD_LOGIC;



23

STD_LOGIC under "PROCESS"

- VARIABLE A, B, C : STD_LOGIC;



24

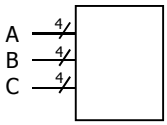
STD_LOGIC_VECTOR

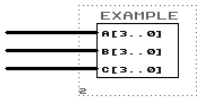
- Under "ENTITY"
  - PORT(A, B, C : IN STD_LOGIC_VECTOR(3 DOWNTO 0));
- Under "ARCHITECTURE"
  - SIGNAL A, B, C : STD_LOGIC _VECTOR(3 DOWNTO 0);
  - TYPE Name IS ARRAY(0 to 7) OF STD_LOGIC _VECTOR(3 DOWNTO 0);
- Under "PROCESS"
  - VARIABLE A, B, C : STD_LOGIC _VECTOR(3 DOWNTO 0);

25

STD_LOGIC_VECTOR under "ENTITY"

- PORT(A, B, C : IN STD_LOGIC_VECTOR(3 DOWNTO 0));

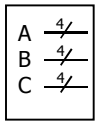




26

STD_LOGIC_VECTOR under "ARCHITECTURE"

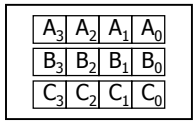
- SIGNAL A, B, C : STD_LOGIC_VECTOR(3 DOWNTO 0);



27

STD_LOGIC_VECTOR under "PROCESS"

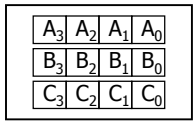
- VARIABLE A, B, C : STD_LOGIC_VECTOR(3 DOWNTO 0);



28

INTEGER

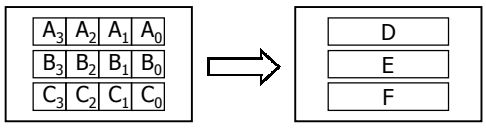
- Used just like a STD_LOGIC_VECTOR
- Defaults to a 32-bit STD_LOGIC_VECTOR
- Use the RANGE *x* TO *y* syntax to set the number of signal lines synthesized



29

INTEGER

- SIGNAL A, B, C : STD_LOGIC_VECTOR(3 DOWNTO 0);
- SIGNAL D, E, F : INTEGER RANGE 0 TO 15;



A <= "1001"; D <= 9;

30

STD_LOGIC_VECTOR in VHDL

- Definitions

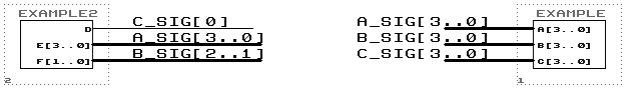
```
SIGNAL A, B : STD_LOGIC_VECTOR( 3 DOWNTO 0 );
SIGNAL C : STD_LOGIC_VECTOR( 2 DOWNTO 0 );
SIGNAL D : STD_LOGIC;
```

- Assignments

```
A <= B;    ( or A( 3 DOWNTO 0) <= B( 3 DOWNTO 0); )
A <= C( 2 DOWNTO 0 ) & D;
A(3) <= D;
A( 1 DOWNTO 0 ) <= B( 3 DOWNTO 2 );
```

31

STD_LOGIC_VECTOR in GDFs



Question: Why won't this schematic compile in MAX+plus II?

32

TYPE

- TYPE keyword is used to create enumerated types not already defined in VHDL.
- Good places to create TYPEs
  - STATE signals in a state machine
  - ARRAYs of STD_LOGIC VECTORs for synthesizing memory or registers

33

TYPE

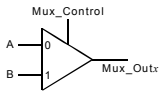
- Defining a new TYPE
- TYPE A_TYPE IS ARRAY(0 to 7) OF
  STD_LOGIC _VECTOR(3 DOWNTO 0);
- TYPE STATE_TYPE IS (state1, state2, state3);

- Declaring signals of a user-defined TYPE:
  - SIGNAL state : STATE_TYPE;
  - SIGNAL memory : A_TYPE;

34

Entity Statement

```
ENTITY multiplexer IS                -- Input Signals and Mux Control
        PORT(A, B, Mux_Control       : IN    STD_LOGIC;
             Mux_Out1, Mux_Out2,
             Mux_Out3, Mux_Out4      : OUT STD_LOGIC  );
END multiplexer;
```



35

Lecture Homework

- Watch the online video tutorial on using the logic analyzer.

20

Midterm Review

Midterm Review

While the Midterm test can include any material covered in lecture and/or lab up to this point, it will concentrate on the major topics outlined here.

Midterm Review

- Mixed Logic
  - Given an equation, draw the corresponding circuit using mixed logic.
  - Given a circuit, write the corresponding equation using mixed logic.
  - Always remember to put your result in simplest form and complete the equation, i.e. F = x not just x.

Midterm Review

- State Machines
  - Given a word description, generate an ASM chart for the state machine implementation.
  - Given an ASM chart for a state machine, generate the VHDL CASE statement to synthesize it.
- Discrete Circuit Design
  - You should be able to work with 74LS*x* chips (anything in your chipset) and be able to illustrate how to connect them on a breadboard.
  - You aren't expected to know pin numbers or even chip numbers. You'll be given the appropriate pin assignments or data sheets.

Midterm Review

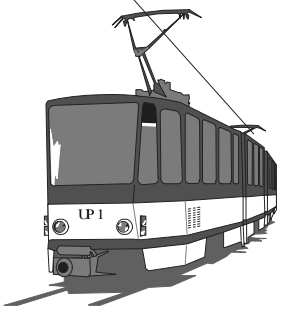
- VHDL
  - Given a section of VHDL code, describe its function and/or draw the circuit it describes. *Hint: Read chapter 6 of the textbook.*
  - Given a word description or circuit, write the VHDL code to synthesize it.
  - Be familiar with the basic VHDL syntax and characteristics.

Midterm Review

- Oscilloscope
  - Know how to use the oscilloscope to perform basic analysis on a circuit.
  - *Hint:* be able to specify where to place the probes on a circuit to measure different circuit characteristics.
- Miscellaneous
  - D Flip-flops
  - Propagation Delay, Rise/Fall Time, Duty Cycle, etc.
  - Counters, Decoders, MUXs

The Electric Train Controller

State Machine Design:
The Electric Train Controller



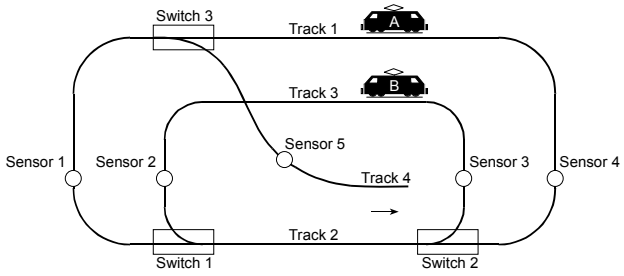


Figure 7.1 Track Layout with Input Sensors and Output Switches and Output Tracks.

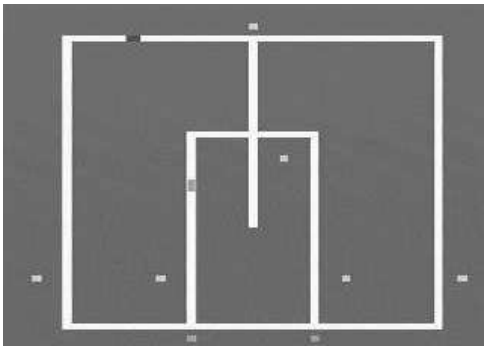
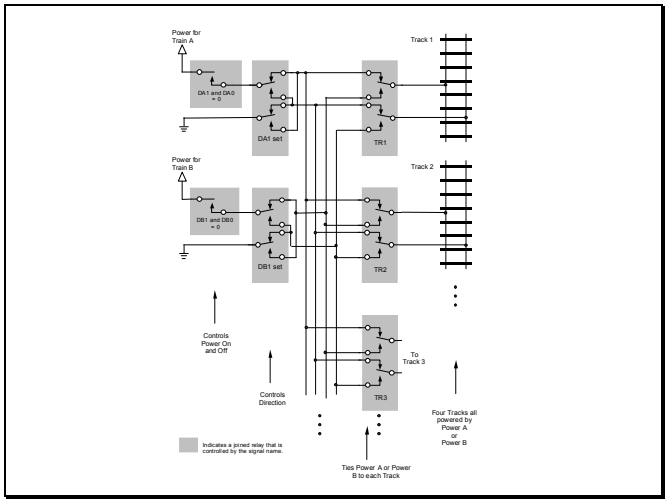


Figure 7.10 Video Image from Train System Simulation.



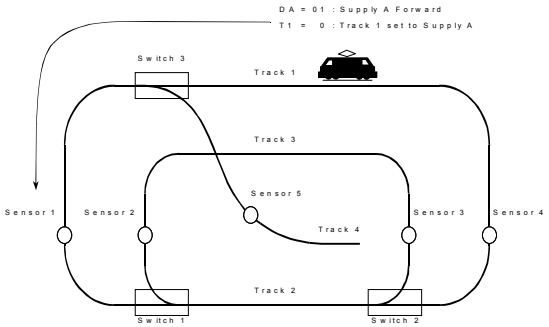


Figure 7.3 Track Power is connected to one of Two Power Sources: A and B.

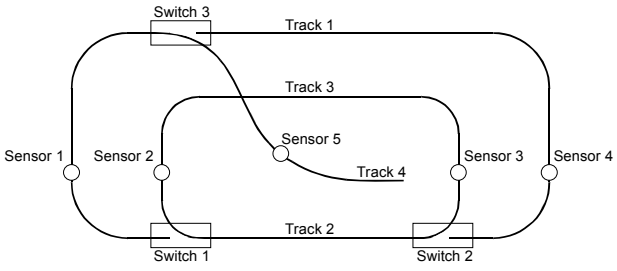
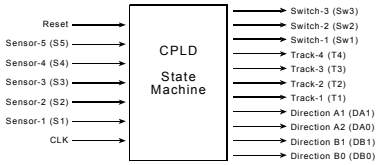


Figure 7.4 Track Direction if all Switches are Asserted (SW1 = SW2 = SW3 = 1)



Sensor (S1, S2, S3, S4, S5)	= 1	Train Present
	= 0	Train not Present
Switches (SW1, SW2, SW3)	= 0	Connected to Outside Track
	= 1	Connected to Inside Track
Track (T1, T2, T3, T4)	= 0	A Virtual Power on Track n
	= 1	B Virtual Power on Track n
Direction (DA1-DA0) and (DB1-DB0)	= 00	Stop
	= 01	Forward (Counterclockwise)
	= 10	Backward (Clockwise)

Figure 7.5 State Machine I/O Configuration

Description of States in Example State Machine

All States
- T3 Asserted: The B power supply is assigned to track 3.
- All signals that are not "Asserted" are zero and imply a logical result as described.

ABout: "Trains A and B Outside"
- DA0 Asserted: Train A is on the outside track and moving counter-clockwise (forward).
- DB0 Asserted: Train B is on the inner track (not the common track) and also moving forward.
- Note that by NOT Asserting DA1, it is automatically zero -- same for DB1. Hence, the outputs are DA = "01" and DB = "01".

Ain: "Train A moves to Common Track"
- Sensor 1 has fired either first or at the same time as Sensor 2.
- Either Train A is trying to move towards the common track, or
- Both trains are attempting to move towards the common track.
- Both trains are allowed to enter here; however, state Bstop will stop B if both have entered.
- DA0 Asserted: Train A is on the outside track and moving counter-clockwise (forward).
- DB0 Asserted: Train B is on the inner track (not the common track) and also moving forward.

Bstop: "Train B stopped at S2 waiting for Train A to clear common track"
- DA0 Asserted: Train A is moving from the outside track to the common track.
- Train B has arrived at Sensor 2 and is stopped and waits until Sensor 4 fires.
- SW1 and SW2 are NOT Asserted to allow the outside track to connect to common track.
- Note that T2 is not asserted making Track 2 tied to the A Power Supply.

Bin: "Train B has reached Sensor 2 before Train A reaches Sensor 1"
- Train B is allowed to enter the common track. Train A is approaching Sensor 1.
- DA0 Asserted: Train A is on the outside track and moving counterclockwise (forward).
- DB0 Asserted: Train B is on the inner track moving towards the common track.
- SW1 Asserted: Switch 1 is set to let the inner track connect to the common track.
- SW2 Asserted: Switch 2 is set to let the inner track connect to the common track.
- T2 Asserted: The B Power Supply is also assigned to the common track.

Astop: "Train A stopped at S1 waiting for Train B to clear the common track"
- DB0 Asserted: Train B is on the inner track moving towards the common track.
- SW1 and SW2 Asserted: Switches 1 and 2 are set to connect the inner track to the common track.
- T2 Asserted: The B Power Supply is also assigned to the common track.

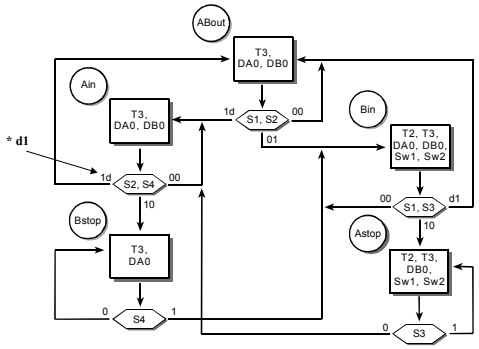


Figure 7.6 Example Train Controller ASM Chart.

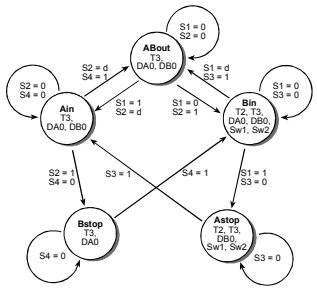


Figure 7.7 Example Train Controller State Diagram.

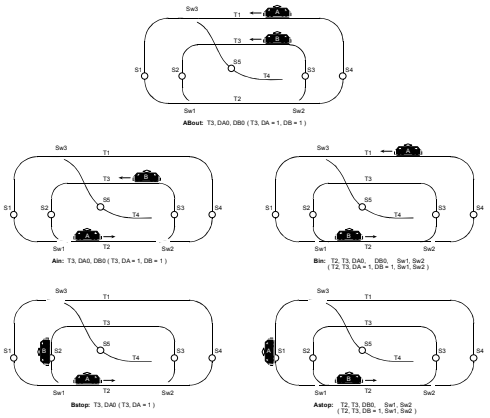


Figure 7.8 Working diagrams of train positions for each state.

Table 7.1 Outputs corresponding to states.

State	ABout	Ain	Astop	Bin	Bstop
Sw1	0	0	1	1	0
Sw2	0	0	1	1	0
Sw3	0	0	0	0	0
T1	0	0	0	0	0
T2	0	0	1	1	0
T3	1	1	1	1	1
T4	0	0	0	0	0
DA(1-0)	01	01	00	01	01
DB(1-0)	01	01	01	01	00

```
-- Example State machine to control trains-- File: Tcontrol.vhd
-- These libraries are required in all VHDL source files
LIBRARY IEEE;
USE  IEEE.STD_LOGIC_1164.ALL;
USE  IEEE.STD_LOGIC_ARITH.ALL;
USE  IEEE.STD_LOGIC_UNSIGNED.ALL;
                -- This section defines state machine inputs and outputs
                -- No modifications should be needed in this section
ENTITY Tcontrol IS
PORT(           reset, clock, sensor1, sensor2,
        sensor3, sensor4, sensor5 : IN      STD_LOGIC;
          switch1, switch2, switch3 : OUT   STD_LOGIC;
          track1, track2, track3, track4        : OUT     STD_LOGIC;
                -- dirA and dirB are 2-bit logic vectors(i.e. an array of 2 bits)
        dirA, dirB                      : OUT STD_LOGIC_VECTOR( 1 DOWNTO 0 ));
END Tcontrol;
                -- This code describes how the state machine operates
                -- This section will need changes for a different  state machine
ARCHITECTURE a OF Tcontrol IS
                -- Define local signals (i.e. non input or output signals) here
    TYPE STATE_TYPE IS ( ABout, Ain, Bin, Astop, Bstop );
    SIGNAL state: STATE_TYPE;
    SIGNAL sensor12, sensor13, sensor24   : STD_LOGIC_VECTOR(1 DOWNTO 0);
BEGIN
                -- This section describes how the state machine behaves
                -- this process runs once every time reset or the clock changes
    PROCESS ( reset, clock )
    BEGIN
                -- Reset to this state (i.e. asynchronous reset)
        IF reset = '1' THEN
                state <= ABout;
        ELSIF clock'EVENT AND clock = '1' THEN
```

```
                -- clock'EVENT means value of clock just changed
                --This section will execute once on each positive clock edge
                --Signal assignments in this section will generate D flip-flops
                -- Case statement to determine next state
                CASE state IS
                        WHEN ABout =>
                                -- This Case checks both sensor1 and sensor2 bits
                            CASE Sensor12 IS
                                -- Note: VHDL's use of double quote for bit vector versus
                                -- a single quote for only one bit!
                                WHEN "00" => state <= ABout;
                                WHEN "01" => state <= Bin;
                                WHEN "10" => state <= Ain;
                                WHEN "11" => state <= Ain;
                                -- Default case is always required
                                WHEN OTHERS => state <= ABout;
                            END CASE;
                        WHEN Ain =>
                            CASE Sensor24 IS
                                WHEN "00" => state <= Ain;
                                WHEN "01" => state <= ABout;
                                WHEN "10" => state <= Bstop;
                                WHEN "11" => state <= ABout;
                                WHEN OTHERS => state <= ABout;
                            END CASE;
                        WHEN Bin =>
                            CASE Sensor13 IS
                                WHEN "00" => state <= Bin;
                                WHEN "01" => state <= ABout;
                                WHEN "10" => state <= Astop;
                                WHEN "11" => state <= About;
                                WHEN OTHERS => state <= ABout;
                            END CASE;
                        WHEN Astop =>
                            IF Sensor3 = '1' THEN
                                state <= Ain;
                            ELSE
                                state <= Astop;
                            END IF;
                        WHEN Bstop =
                            IF Sensor4 = '1' THEN
                                state <= Bin;
                            ELSE
                                state <= Bstop;
                            END IF;
                END CASE;
        END IF;
    END PROCESS;
```

```
                    -- combine sensor bits for case statements above
                    -- "&" operator combines bits
sensor12 <= sensor1 & sensor2;
sensor13 <= sensor1 & sensor3;
sensor24 <= sensor2 & sensor4;

   Track1  <= '0';                  -- These outputs do not depend on the state
   Track4  <= '0';
   Switch3 <= '0';

                    -- Outputs that depend on state, use state to select value
                    -- Be sure to specify every output for every state
                    -- values will not default to zero!

   WITH state SELECT
            Track3    <=        '1'     WHEN ABout,
                                '1'     WHEN Ain,
                                '1'     WHEN Bin,
                                '1'     WHEN Astop,
                                '1'     WHEN Bstop;

   WITH state SELECT
            Track2    <=        '0'     WHEN ABout,
                                '0'     WHEN Ain,
                                '1'     WHEN Bin,
                                '1'     WHEN Astop,
                                '0'     WHEN Bstop;

   WITH state SELECT
            Switch1   <=        '0'     WHEN ABout,
                                '0'     WHEN Ain,
                                '1'     WHEN Bin,
                                '1'     WHEN Astop,
                                '0'     WHEN Bstop;

   WITH state SELECT
            Switch2   <=        '0'     WHEN ABout,
                                '0'     WHEN Ain,
                                '1'     WHEN Bin,
                                '1'     WHEN Astop,
                                '0'      WHEN Bstop;

   WITH state SELECT
            DirA      <=        "01"    WHEN ABout,
                                "01"    WHEN Ain,
                                "01"    WHEN Bin,
                                "00"    WHEN Astop,
                                "01"    WHEN Bstop;

   WITH state SELECT
            DirB      <=        "01"    WHEN ABout,
                                "01"    WHEN Ain,
                                "01"    WHEN Bin,
                                "01"    WHEN Astop,
                                "00"    WHEN Bstop;

END a;
```

```
% Simulation Test Vectors for Train Control State Machine %
%     - File: Tcontrol.vec %

                    % Simulation Start time, Stop time, and Interval%
        START 0us;
        STOP 1100ns;
        INTERVAL 20ns;

                    % Specify the values of all inputs in the simulation %
                    % Setup the System Clock %
        INPUTS Clock;
        PATTERN
        0 1;
                    % Force a Reset to Start %
        INPUTS Reset;
        PATTERN
                    % Pattern Format is time>value %
        0>1
        100>0;
                    % Cycle Sensor Inputs at following times %
                    %       to cycle through all states%

                    %Specify Sensor1 values%
        INPUTS Sensor1;
        PATTERN
        0>0
        100>0
        200>1
        300>0
        400>0
        500>0
        600>0
        700>0
        800>1;
```

```
                    %Specify Sensor1 values%
INPUTS Sensor1;
PATTERN
0>0
100>0
200>1
300>0
400>0
500>0
600>0
700>0
800>1;
                    %Specify Sensor2 values%
INPUTS Sensor2;
PATTERN
0>0
100>0
200>0
300>1
400>0
500>0
600>0
700>1;
```

```
                %Specify Sensor3 values%

INPUTS Sensor3;
PATTERN
0>0
100>0
200>0
300>0
400>0
500>0
600>1

700>0

800>0
900>1
1000>0;

                %Specify Sensor4 values%

INPUTS Sensor4;
PATTERN
0>0
100>0
200>0
300>0
400>1
500>0
600>0
700>0
800>0
900>0
1000>1;

        % Display these output signals on simulation timing
        diagram %
        % Buried means it is an internal signal - not an input
        or output %
BURIED state;
OUTPUTS switch1 switch2 switch3 track1 track2 track3 track4 dirA
dirB;
```

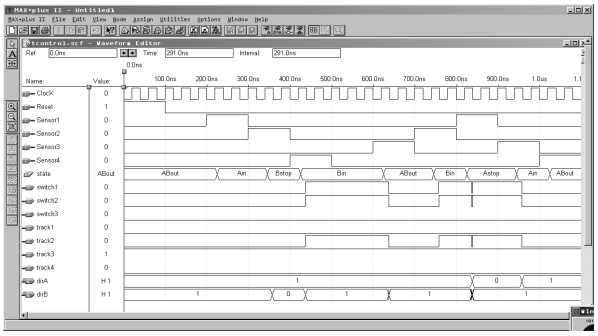
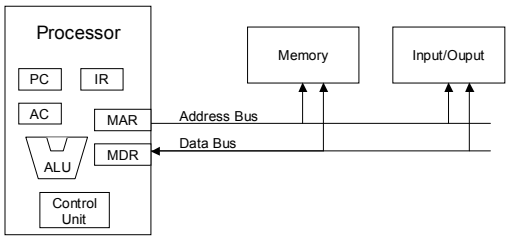


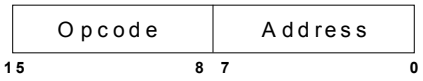
Figure 7.9 Simulation of Tcontrol.vhd using Tcontrol.vec vector file.

Simple Computer I

Architecture



Instruction Format



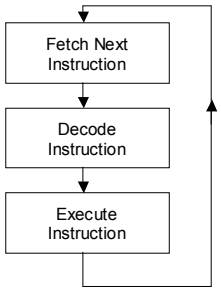
Computer Instructions

Instruction Mnemonic	Operation Preformed	Opcode Value
ADD *address*	AC <= AC + contents of memory address	00
STORE *address*	contents of memory address <= AC	01
LOAD *address*	AC <= contents of memory address	02
JUMP *address*	PC <= address	03
JNEG *address*	If AC < 0 Then PC <= address	04

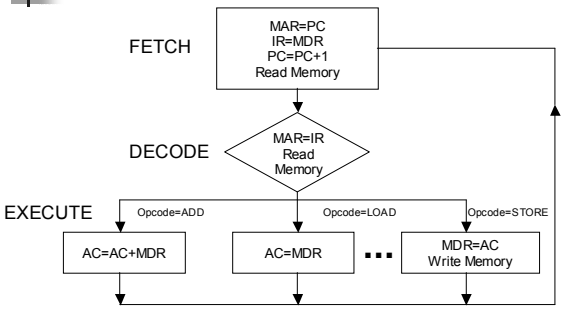
Example Program for A = B + C

Assembly Language	MachineLanguage
LOAD B	0211
ADD C	0012
STORE A	0110

Fetch, Decode and Execute Cycle



Detailed View



```
LIBRARY IEEE;
USE  IEEE.STD_LOGIC_1164.ALL;
USE  IEEE.STD_LOGIC_ARITH.ALL;
USE  IEEE.STD_LOGIC_UNSIGNED.ALL;
LIBRARY lpm;
USE lpm.lpm_components.ALL;
ENTITY SCOMP IS
PORT(   clock, reset                    : IN STD_LOGIC;
        program_counter_out             : OUT STD_LOGIC_VECTOR( 7 DOWNTO 0 );
        register_AC_out                 : OUT STD_LOGIC_VECTOR(15 DOWNTO 0 );
        memory_data_register_out        : OUT STD_LOGIC_VECTOR(15 DOWNTO 0 ));
END SCOMP;
ARCHITECTURE a OF scomp IS
TYPE STATE_TYPE IS ( reset_pc, fetch, decode, execute_add, execute_load, execute_store,
                     execute_store3, execute_store2, execute_jump );
SIGNAL state: STATE_TYPE;
SIGNAL instruction_register, memory_data_register   : STD_LOGIC_VECTOR(15 DOWNTO 0 );
SIGNAL register_AC                                  : STD_LOGIC_VECTOR(15 DOWNTO 0 );
SIGNAL program_counter                              : STD_LOGIC_VECTOR( 7 DOWNTO 0 );
SIGNAL memory_address_register                      : STD_LOGIC_VECTOR( 7 DOWNTO 0 );
SIGNAL memory_write                                 : STD_LOGIC;
BEGIN   -- Use LPM function for computer's memory (256 16-bit words)
  memory: lpm_ram_dq
    GENERIC MAP (
        lpm_widthad         => 8,
        lpm_outdata         => "UNREGISTERED",
        lpm_indata          => "REGISTERED",
        lpm_address_control => "UNREGISTERED",
                    -- Reads in mif file for initial program and data values
        lpm_file            => "program.mif",
        lpm_width           => 16 )
    PORT MAP ( data => Register_AC, address => memory_address_register,
               we => memory_write, inclock => clock, q => memory_data_register );

        program_counter_out     <= program_counter;
        register_AC_out         <= register_AC;
        memory_data_register_out <= memory_data_register;
PROCESS ( CLOCK, RESET )
    BEGIN
    IF reset = '1' THEN
        state <= reset_pc;
    ELSIF clock'EVENT AND clock = '1' THEN
```

```
ENTITY SCOMP IS
PORT(                clock, reset   : IN STD_LOGIC;
           program_counter_out      : OUT STD_LOGIC_VECTOR(  7 DOWNTO 0 );
                 register_AC_out    : OUT STD_LOGIC_VECTOR(15 DOWNTO 0 );
      memory_data_register_out      : OUT STD_LOGIC_VECTOR(15 DOWNTO 0 ));
END SCOMP;

ARCHITECTURE a OF scomp IS
TYPE STATE_TYPE IS ( reset_pc, fetch, decode, execute_add, execute_load,
                     execute_store, execute_store3, execute_store2, execute_jump );
SIGNAL state                                        : STATE_TYPE;
SIGNAL instruction_register, memory_data_register   : STD_LOGIC_VECTOR(15 DOWNTO 0 );
SIGNAL register_AC                                  : STD_LOGIC_VECTOR(15 DOWNTO 0 );
SIGNAL program_counter                              : STD_LOGIC_VECTOR(  7 DOWNTO 0 );
SIGNAL memory_address_register                      : STD_LOGIC_VECTOR(  7 DOWNTO 0 );
SIGNAL memory_write                                 : STD_LOGIC;
```

```
USE lpm.lpm_components.ALL; -- Library of Parameterized Modules

BEGIN   -- Use LPM function for computer's memory (256 16-bit words)
  memory: lpm_ram_dq

    GENERIC MAP (
      lpm_widthad           => 8,
      lpm_outdata           => "UNREGISTERED",
      lpm_indata            => "REGISTERED",
      lpm_address_control   => "UNREGISTERED",

      -- Reads in mif file for initial program and data values

      lpm_file              => "program.mif",
      lpm_width             => 16 )

    PORT MAP (  data  => Register_AC,   address => memory_address_register,
                we    => memory_write,  inclock => clock, q => memory_data_register );

    program_counter_out     <= program_counter;
    register_AC_out         <= register_AC;
    memory_data_register_out <= memory_data_register;
```

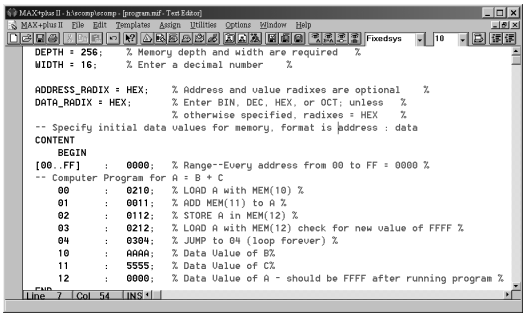
```
CASE state IS
        -- reset the computer, need to clear some registers
        WHEN reset_pc =>
                program_counter  <= "00000000";
                memory_address_register          <= "00000000";
                register_AC          <= "0000000000000000";
                memory_write       <= '0';
                state                <= fetch;
        -- Fetch instruction from memory and add 1 to PC
        WHEN fetch =>
                instruction_register          <= memory_data_register;
                program_counter  <= program_counter + 1;
                memory_write       <= '0';
                state              <= decode;
        -- Decode instruction and send out address of any data operands
        WHEN decode =>
                memory_address_register <= instruction_register( 7 DOWNTO 0);
                CASE instruction_register( 15 DOWNTO 8 ) IS
                        WHEN "00000000" =>
                            state <= execute_add;
                        WHEN "00000001" =>
                            state <= execute_store;
                        WHEN "00000010" =>
                            state <= execute_load;
                        WHEN "00000011" =>
                            state <= execute_jump;
                        WHEN OTHERS =>
                            state <= fetch;
                END CASE;
```

```
                                                -- Execute the ADD instruction
WHEN execute_add =>
    register_ac                 <= register_ac + memory_data_register;
    memory_address_register <= program_counter;
    state                       <= fetch;
                                                -- Execute the STORE instruction
                                                -- (needs three clock cycles for memory write)
WHEN execute_store =>
                                                -- write register_A to memory
    memory_write            <= '1';
    state                       <= execute_store2;
                                                -- This state ensures that the memory address is
                                                --  valid until after memory_write goes inactive
WHEN execute_store2 =>
    memory_write            <= '0';
    state                       <= execute_store3;
WHEN execute_store3 =>
    memory_address_register <= program_counter;
    state                       <= fetch;
                                                -- Execute the LOAD instruction
WHEN execute_load =>
    register_ac                 <=  memory_data_register;
    memory_address_register <= program_counter;
    state                       <= fetch;
                                                -- Execute the JUMP instruction
WHEN execute_jump =>
    memory_address_register <= instruction_register( 7 DOWNTO 0 );
    program_counter         <= instruction_register( 7 DOWNTO 0 );
    state                       <= fetch;
WHEN OTHERS =>
    memory_address_register <= program_counter;
    state                       <= fetch;
```

Asynchronous Reset

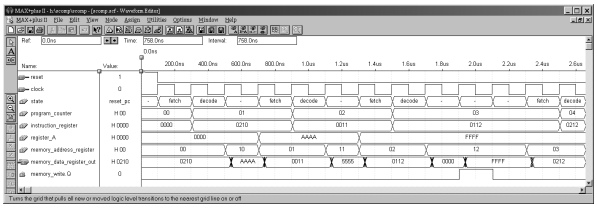
```
PROCESS ( CLOCK, RESET )
  BEGIN
  IF reset = '1' THEN
     state <= reset_pc;
  ELSIF clock'EVENT AND clock = '1'
THEN
```

MIF File Containing μP 1 Program



```
DEPTH = 256;    % Memory depth and width are required   %
WIDTH = 16;     % Enter a decimal number    %

ADDRESS_RADIX = HEX;    % Address and value radixes are optional    %
DATA_RADIX = HEX;       % Enter BIN, DEC, HEX, or OCT; unless   %
                        % otherwise specified, radixes = HEX    %
-- Specify initial data values for memory, format is address : data
CONTENT
    BEGIN
[00..FF]    :   0000;   % Range--Every address from 00 to FF = 0000 %
-- Computer Program for A = B + C
    00      :   0210;   % LOAD A with MEM(10) %
    01      :   0011;   % ADD MEM(11) to A %
    02      :   0112;   % STORE A in MEM(12) %
    03      :   0212;   % LOAD A with MEM(12) check for new value of FFFF %
    04      :   0304;   % JUMP to 04 (loop forever) %
    10      :   AAAA;   % Data Value of B%
    11      :   5555;   % Data Value of C%
    12      :   0000;   % Data Value of A - should be FFFF after running program %
END;
```

Simulation of the Simple μP 1 Program

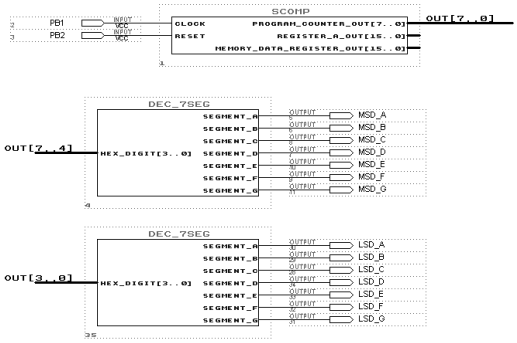


Simple Computer II

Lab 9

- Complete Lab Exercise 5 at the end of Chapter 8.
- Use the programs written for Exercises 8.2 and 8.3.
- Verify that their behavior matches the simulations you made in Lab 8.

Running the µp 1 – Exer. 8.5



I/O Instructions – Exer. 8.6

Instruction	Initial Function (i/o address = 0x00)	Opcode
IN *i/o address*	AC(7 downto 0) = FLEX DIP switches	OE
OUT *i/o address*	FLEX 7-Seg. Disp. = AC(7 downto 0)	OF

Multiple I/O Devices

Instruction	Function	Opcode
IN *i/o address*	AC(7 downto 0) = contents of *i/o addr.*	OE
OUT *i/o address*	contents of *i/o addr.* = AC(7 downto 0)	OF

Future Use of Multiple Inputs

- Inputs on the UP1-bot
  - 0x00: I/R Proximity Detector
  - 0x01: Digital Compass
  - 0x02: DIP switches
  - 0x03: Pushbuttons
  - 0x04: <etc>

Future Use of Multiple Outputs

- Multiple outputs on the UP1-bot
  - 0x00: Motor speed
  - 0x01: Motor direction
  - 0x02: 7-segment displays
  - 0x03: particle-beam weapon
  - 0x04: <etc>

ASM Program – Exer. 8.6

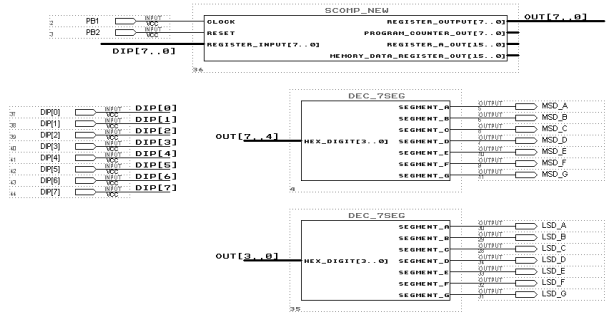
```
IN   00   ; Load DIP switches into AC[7..0]
ADDI 01   ; Add 1 to the AC
OUT  00   ; Output AC[7..0] to the 7-Seg Disp
JUMP 00   ; Infinite loop to beginning
```

Adding I/O Lines

NEW Signals

```
ENTITY SCOMP IS
PORT( clock, reset                : IN STD_LOGIC;
      register_input              : IN STD_LOGIC_VECTOR( 7 DOWNTO 0 );
      register_output             : OUT STD_LOGIC_VECTOR( 7 DOWNTO 0 );
      program_counter_out         : OUT STD_LOGIC_VECTOR( 7 DOWNTO 0 );
      register_AC_out             : OUT STD_LOGIC_VECTOR(15 DOWNTO 0 );
      memory_data_register_out    : OUT STD_LOGIC_VECTOR(15 DOWNTO 0 ));
END SCOMP;
```

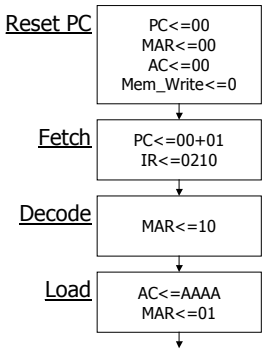
Running the µp 1 with I/O



Lab 9

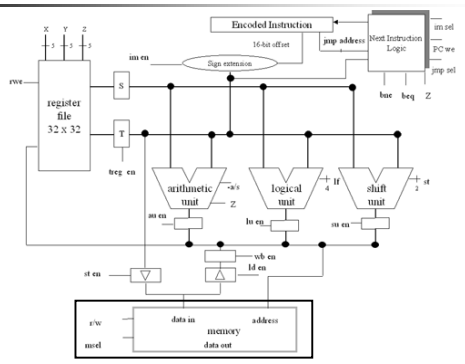
- Illustrate the simple computer's datapath by tracing the execution of the program from Exercise 8.2.
- Use an ASM-type chart like the one in Figure 8.6. Your chart will have only one path, because you are mapping a specific thread of execution.

Execution Path – Exercise 3

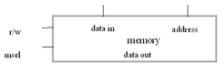


Simple Computer III

ECE 2030 View of Memory

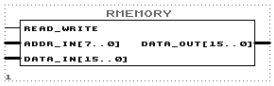


ECE 2030 View of Memory



- Memory Write (r/w)
- Data In
- Address
- Data Out
- Memory Select

ECE 2031 View of Memory



- Memory Write (r/w) ➔ read_write
- Data In ➔ data_in[7..0]
- Address ➔ addr_in[7..0]
- Data Out ➔ data_out[15..0]
- Memory Select ➔ <Not used>

Inside RMEMORY

```
library IEEE;
use  IEEE.STD_LOGIC_1164.all;
use  IEEE.STD_LOGIC_ARITH.all;
use  IEEE.STD_LOGIC_UNSIGNED.all;
library lpm;
USE lpm.lpm_components.all;

ENTITY rmemory IS
    PORT(
        read_write      : IN    STD_LOGIC;
        addr_in         : IN    STD_LOGIC_VECTOR(7 downto 0);
        data_in         : IN    STD_LOGIC_VECTOR(15 downto 0);
        data_out        : OUT   STD_LOGIC_VECTOR(15 downto 0));
END rmemory;
```

Inside RMEMORY – cont.

```
ARCHITECTURE a OF rmemory IS
BEGIN

data_memory: lpm_ram_dq
    GENERIC MAP (       lpm_widthad => 8,
                        lpm_outdata => "UNREGISTERED",
                        lpm_indata  => "UNREGISTERED",
                        lpm_address_control => "UNREGISTERED",
                        lpm_width   => 16)

    PORT MAP (data => data_in, address => addr_in,
                we => read_write, q => data_out);

END a;
```

Lab 10 – Exercise 1

- Modify your ENTITY declaration to **identically** match the one on the following slide (names and order of signal declarations should be the same).
- You will have to add the signals
  `instruction_out`
  `mem_write_out`
- You may have to change the names of the signals you added in Exercise 8.6 to
  register_in
  register_out.

Lab 10 – Exercise 1

```
ENTITY SCOMP IS
PORT( clock, reset            : IN  STD_LOGIC;
      register_in             : IN  STD_LOGIC_VECTOR( 7 DOWNTO 0 );
      register_out            : OUT STD_LOGIC_VECTOR( 7 DOWNTO 0 );
      program_counter_out     : OUT STD_LOGIC_VECTOR( 7 DOWNTO 0 );
      register_AC_out         : OUT STD_LOGIC_VECTOR(15 DOWNTO 0 );
      memory_data_register_out: OUT STD_LOGIC_VECTOR(15 DOWNTO 0 );
      instruction_out         : OUT STD_LOGIC_VECTOR(15 DOWNTO 0 );
      mem_write_out           : OUT STD_LOGIC );
END SCOMP;
```

Lab 10 – Exercise 1

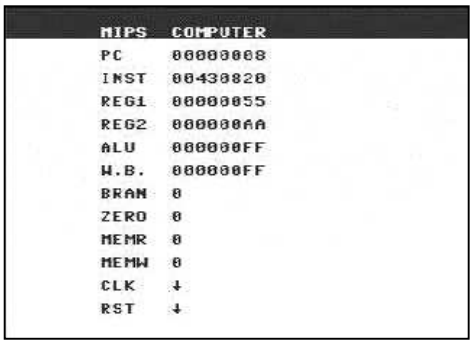
- Add the following lines of VHDL code to the ARCHITECTURE section (outside any processes) to drive the **mem_write_out** and **instruction_out** signals.

```
mem_write_out   <= memory_write;
instruction_out <= instruction_register;
```

Lab 10 – Exercise 8.12:

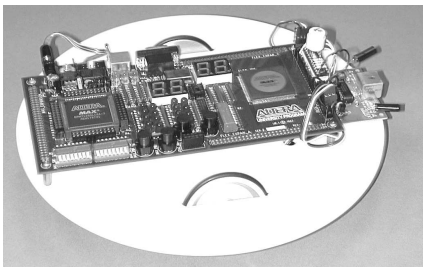
- Modify the μP 1 computer so that it uses two different memories—one for instructions and one for data values.
- The new data memory should be 256 locations of 16-bit data.

μp 1 Display Demo

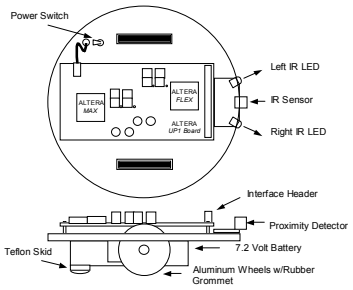


Introduction to the UP1-bot

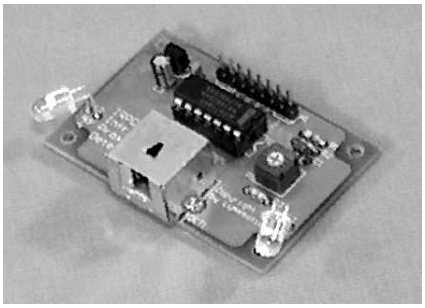
Intro to the UP1bot



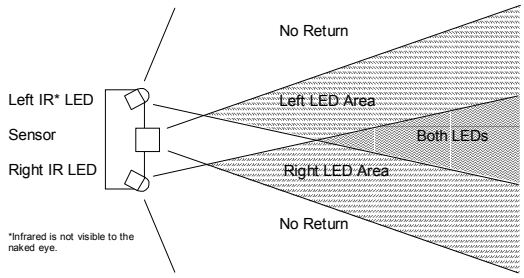
Layout of the UP1bot



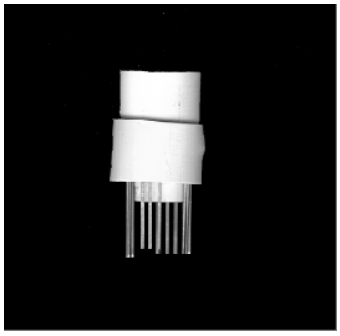
IR Proximity Sensor



IR Sensor Area



Digital Compass

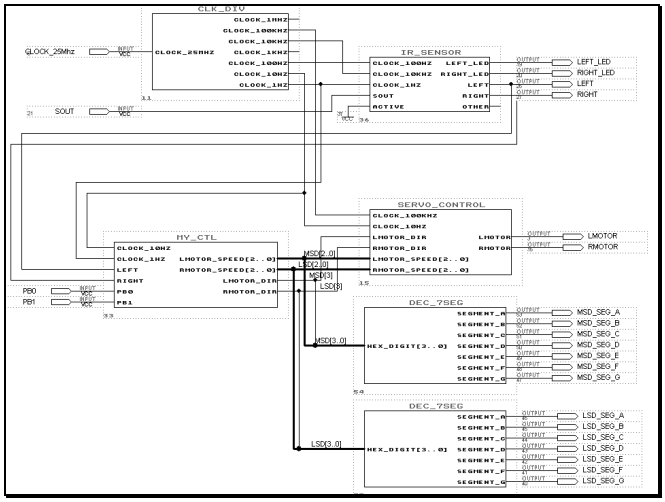


Digital Compass

- The compass outputs are active low.
- Pin Assignments
  - North: pin 163
  - South: pin 161
  - East: pin 158
  - West: pin 156

Digital Compass

- Use the compass outputs as direct inputs into your state machine or simple computer controller. No signal conditioning or separate "interface" is needed.



Peer Review Survey

Attendance

Member was expected to come to

- ALL group meetings.
- MOST group meetings.
- SOME group meetings.
- NONE group meetings.

Member came to

- ALL group meetings.
- MOST group meetings.
- SOME group meetings.
- NO group meetings.

Workload

Member was expected to complete _____% of this project.

Member completed _____% of this project.

Group Participation

Member's expected group participation was:

- MOST - facilitate group discussions/work.
- SOME - participate in group discussions/work.
- NONE - work was to be done individually.
- NEGATIVE - disrupt group discussions/work.

Member's group participation was:

- MOST - facilitated group discussions/work.
- SOME - participated in group discussions/work.
- NONE - work was done individually.
- NEGATIVE - disrupted group discussions/work.

Knowledge of Lab

Member's expected knowledge about the lab was:

- EXCEPTIONAL.
- WELL INFORMED.
- ADEQUATE.
- SOMEWHAT DEFICIENT.
- INADEQUATE.

Member's knowledge about the lab was:

- EXCEPTIONAL.
- WELL INFORMED.
- ADEQUATE.
- SOMEWHAT DEFICIENT.
- INADEQUATE.

Quality of Work

Member's expected work quality was:

- CONSISTENTLY HIGH.
- GOOD.
- SATISFACTORY.
- CARELESS.
- VERY POOR.

Member's work quality was:

- CONSISTENTLY HIGH.
- GOOD.
- SATISFACTORY.
- CARELESS.
- VERY POOR.

Overall

My expectation of Member's contribution was:

- VERY HIGH.
- HIGH.
- MODERATE.
- LOW.
- VERY LOW.

Member's contribution was:

- VERY HIGH.
- HIGH.
- MODERATE.
- LOW.
- VERY LOW.

]