

**University of California at Berkeley**  
**College of Engineering**  
**Department of Electrical Engineering and Computer Science**

EECS 150  
 Fall 2000

R. H. Katz

**Problem Set # 8/9 (Assigned 2 November, Due 17 November)**

1. Recall the sequence detector FSM described in lecture: the machine outputs a 1 whenever the input sequence ...010... has been detected, as long as the string 100 has not been encountered. Implement this state machine with a trickier datapath: a shift register for the state register and a comparator to detect the critical sequences.
  - (a) Design a datapath down to the block diagram level of its functional units to support this state machine. Define your register transfer operations, control signals, bus architecture, and timing behavior.
  - (b) Design the state machine that implements the behavior of the specification above in terms of how it orchestrates the execution of the datapath you designed in part (a).
2. Register files are pretty useful components of a datapath. In this question, you will explore a variety of ways of implementing a four word by 4-bit register file.
  - (a) A single port register file can read or write a single word at a time (but can't do both). Show the block diagram for such a register file, and indicate how a single storage bit would be implemented down to the logic level (i.e., gates and flipflops). Include selection and control inputs, such as Read and Write. Make your assumptions explicit.
  - (b) A dual port register file can read two words at a time, or write one word at a time (but can't do both). Modify your design to support a dual-read capability. Is there any change to the timing behavior of your design?
  - (c) A dual port read/single port write register file allows double register reading and single register writing at the same time. Your implementation in this case must be able to handle the case where a register being read can be identical to the register being written. This is usually handled by adding special write buffer registers to the register file that update the register automatically only after it has been successfully read. Redesign your register file to implement this special kind of datapath "interlock," and describe read and write control state machine fragments that illustrate how it behaves.
3. Design an Arithmetic Logic Unit to the following specification. The ALU has 4 control inputs: M, S<sub>2</sub>, S<sub>1</sub>, S<sub>0</sub>, a carry-in and carry-out, and bit slice data inputs A<sub>i</sub> and B<sub>i</sub> (for each bit of the ALU). When M=1, the ALU is in arithmetic mode and when M=0, it is in logic mode. The functions work as follows:

M = 0	<table style="border-collapse: collapse;"> <tr><td style="padding: 2px 5px;">S2</td><td style="padding: 2px 5px;">S1</td><td style="padding: 2px 5px;">S0</td></tr> <tr><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td></tr> <tr><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">1</td></tr> <tr><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">0</td></tr> <tr><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">1</td></tr> <tr><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td></tr> <tr><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">1</td></tr> <tr><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">0</td></tr> <tr><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">1</td></tr> </table>	S2	S1	S0	0	0	0	0	0	1	0	1	0	0	1	1	1	0	0	1	0	1	1	1	0	1	1	1	<table style="border-collapse: collapse;"> <tr><td style="padding: 2px 5px;">F</td></tr> <tr><td style="padding: 2px 5px;">0</td></tr> <tr><td style="padding: 2px 5px;">A or B</td></tr> <tr><td style="padding: 2px 5px;">A and B</td></tr> <tr><td style="padding: 2px 5px;">Not A</td></tr> <tr><td style="padding: 2px 5px;">Not B</td></tr> <tr><td style="padding: 2px 5px;">A nand B</td></tr> <tr><td style="padding: 2px 5px;">A nor B</td></tr> <tr><td style="padding: 2px 5px;">1</td></tr> </table>	F	0	A or B	A and B	Not A	Not B	A nand B	A nor B	1	M = 1	<table style="border-collapse: collapse;"> <tr><td style="padding: 2px 5px;">S1</td><td style="padding: 2px 5px;">S0</td></tr> <tr><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td></tr> <tr><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">1</td></tr> <tr><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">0</td></tr> <tr><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">1</td></tr> </table>	S1	S0	0	0	0	1	1	0	1	1	<table style="border-collapse: collapse;"> <tr><td style="padding: 2px 5px;">F</td></tr> <tr><td style="padding: 2px 5px;">A plus 1</td></tr> <tr><td style="padding: 2px 5px;">A plus B</td></tr> <tr><td style="padding: 2px 5px;">A minus B</td></tr> <tr><td style="padding: 2px 5px;">A minus 1</td></tr> </table>	F	A plus 1	A plus B	A minus B	A minus 1
S2	S1	S0																																																						
0	0	0																																																						
0	0	1																																																						
0	1	0																																																						
0	1	1																																																						
1	0	0																																																						
1	0	1																																																						
1	1	0																																																						
1	1	1																																																						
F																																																								
0																																																								
A or B																																																								
A and B																																																								
Not A																																																								
Not B																																																								
A nand B																																																								
A nor B																																																								
1																																																								
S1	S0																																																							
0	0																																																							
0	1																																																							
1	0																																																							
1	1																																																							
F																																																								
A plus 1																																																								
A plus B																																																								
A minus B																																																								
A minus 1																																																								

Develop a gate level implementation for a single bit of the ALU.

4. A divider can be implemented by the method of successive subtraction. Let's illustrate this with the following example of dividing 13 by 3 (Quotient = 4, Remainder = 1):

Quotient (Q) is initially 0 and Remainder (R) is initially 0

$$13 - 3 = 10; \quad Q = Q + 1 = 1$$

$$10 - 3 = 7; \quad Q = Q + 1 = 2$$

$$7 - 3 = 4; \quad Q = Q + 1 = 3$$

$$4 - 3 = 1; \quad Q = Q + 1 = 4$$

Since the Dividend is smaller than the Divisor, R = Dividend = 1

Design a datapath to the block level, with all necessary control signals, and a controller finite state machine, to implement this iterative divider system.

5. Your textbook describes a design for a combinational multiplier, but its implementation, though fast, uses up a huge amount of logic. An alternative is to build a sequential binary multiplier whose operation can be motivated by the following example of multiplying 100 (multiplicand) by 11 (multiplier) to obtain 1100 (product):

(i) Product is initially 0.

(ii) If the low order bit of the multiplier is 1, add the multiplicand to the accumulating product.

(iii) Shift the multiplicand one place to the left and the multiplier one place to the right.

(iv) Repeat this sequence of steps until the multiplier is zero.

Multiplicand	Multiplier	Product
0000 0100	001 <u>1</u>	0000 0100
0000 1000	000 <u>1</u>	0000 1100
0001 0000	<u>0000</u>	0000 1100

Design a datapath to the block diagram level, identify its control signals, and develop a controller finite state machine that implements this sequential multiplier function.