

---

# **Computer Aided Digital Systems Design - EE 4743/6743**

**Sherif Abdelwahed**

---

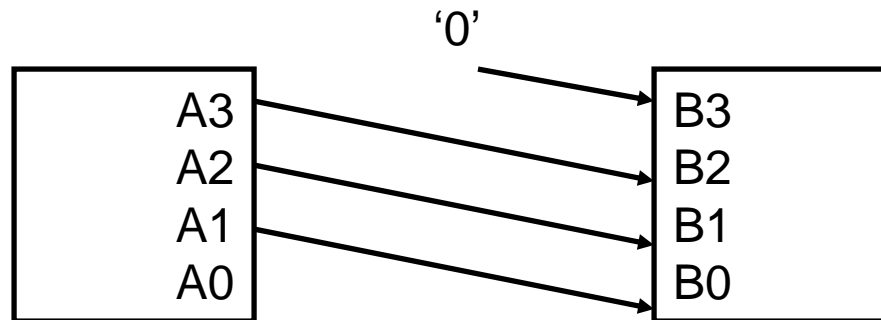
**Shift-add-3 Algorithm**

**September 5, 2007**

**Department of Electrical and Compute Engineering  
Mississippi State University**

# Multiplication and Division made easy

- To multiply by 2, shift left
- To divide by 2, shift right
- Don't need to use shifter though!
- $B = A / 2$



---

# Binary to BCD

- Seven-segment display on the Spartan 3 board
  - Currently is set up to display hexadecimal  
0-9, A-F
  - Lab 2 is set up to display input switches in hexadecimal  
on seven-segment display
  - Exercise is to build circuit to change to decimal
  - Input is 8 switches  $\rightarrow 2^8 = 256$
  - Three segments needed
-

# Binary to BCD

- Easy to change from four-bit binary to BCD
- Anything above 9 is added to the next position

Decimal Digit	Binary Representation
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

# Conversion options

- Given 4-bit number : 1100
  - Equivalent to decimal 12
- Options:
  - Subtract 10 from number to get 2
    - Then add one to the ten digit
  - Add 6, then check the remainder

Original number:	1100	12
Add 6	0110	6
Final number	0001 0010	12

# BCD Multiplication

- If we want to multiply by 2, we can keep the number in BCD by thinking ahead
- If starting number is 5 or larger, then we will end up with a number that must be converted
  - $4 * 2 = 8$  (1 BCD character)
  - $5 * 2 = 10$  (2 BCD characters)
  - $6 * 2 = 12$  (2 BCD characters)
- Add 3 if number is  $>4$ , then shift left for the multiply
- Like adding 6, but with no carry

Original number	0111	7
Add 3	1010	10
Shift left	0001 0100	14

---

# Why Use a Comparator?

- Use a recursive program to compute numbers without a comparator
  - For every bit, we will check & shift once
    - 4-bit number require 4 shifts
    - 4 shifts mean our final answer is “multiplied” by 16
    - Use 4 bits to the left to “divide” by 16
    - Ones must be checked for  $>4$ 
      - Add 3 if so
  - Called “Shift-Add-3” formula
  - Doesn’t necessarily need to be recursive...
-

# Shift-Add-3

<b>Operation</b>	<b>Tens</b>	<b>Ones</b>	<b>Binary</b>
Start	0000	0000	1111
Shift (1)	0000	0001	1110
Shift (2)	0000	0011	1100
Shift (3)	0000	0111	1000
Add-3	0000	1010	1000
Shift (4)	0001	0101	0000
Decimal	<b>1</b>	<b>5</b>	

- Note for the largest 4-bit number, only one Add-3 is ever needed
- Can shorten procedure by doing a 3-bit shift to start

# No Adders!

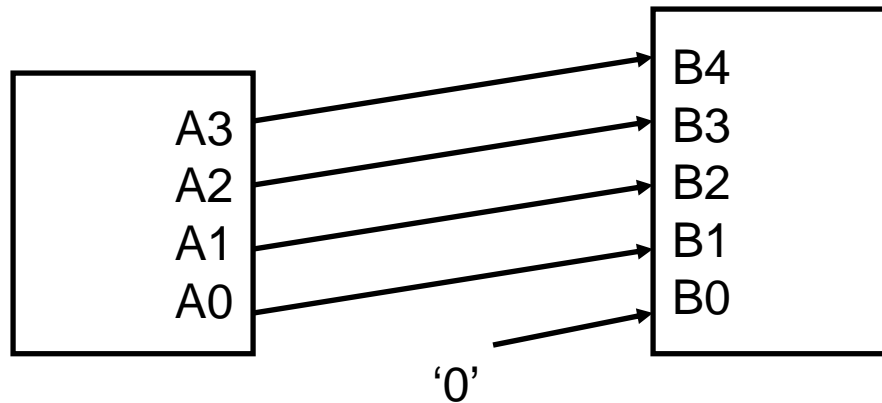
- Always adding by a constant (3)
- Instead of using an Adder, we can use a look-up table
- Only 4-bits input and output (since our value will always be in BCD)
- Verilog will want to make an adder out of

$S \leq A + 3;$

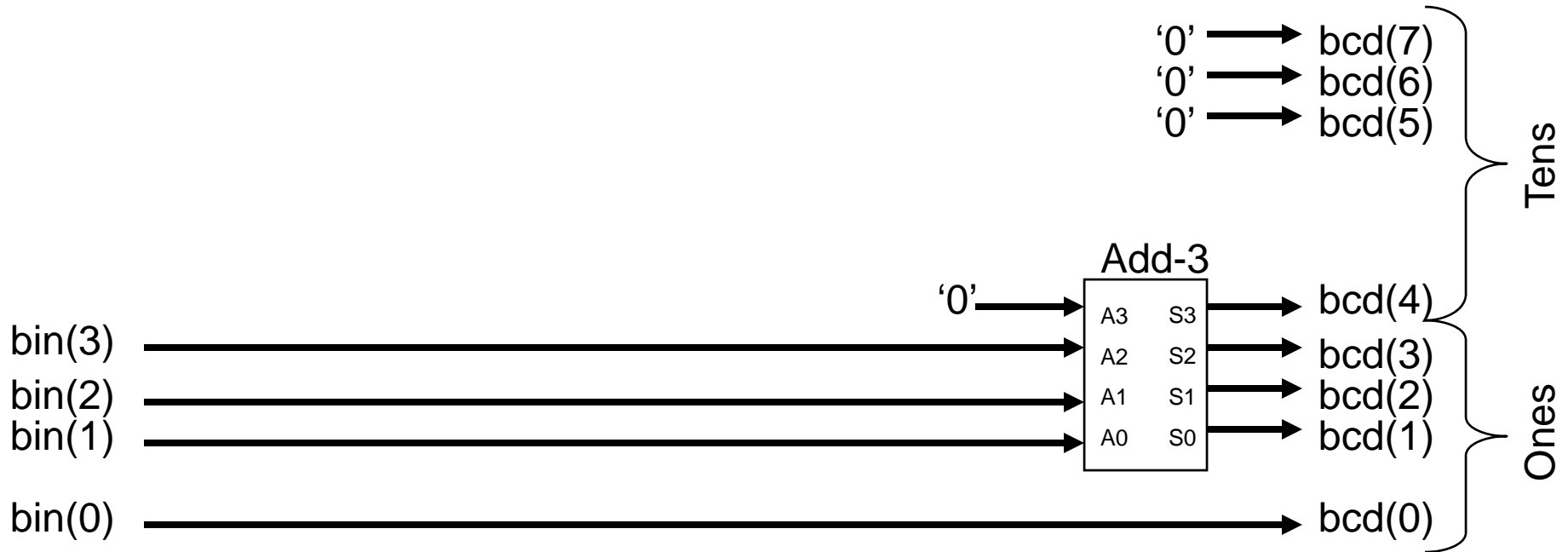
A3 A2 A1 A0	S3 S2 S1 S0
0000	0000
0001	0001
0010	0010
0011	0011
0100	0100
0101	1000
0110	1001
0111	1010
1000	1011
1001	1100
1010	XXXX
1011	XXXX
1100	XXXX
1101	XXXX
1110	XXXX
1111	XXXX

# No Shifters!

- Use multiply-by-2 trick
- Instead of shifting, we can just connect our output one place to the left
- $B = A * 2$



# No Recursion





---

# Lab Assignment

- First task is to build the Add-3 circuit
    - All combinational
    - Just implement truth table
    - Four equations:  $S_3$ ,  $S_2$ ,  $S_1$ ,  $S_0$
  - Put the Add-3 module in the top-level schematic
    - Use previous circuit diagram
  - For B grade: use all schematic
  - For A grade: use Verilog for Add-3 circuit
  - Use simulator to check Add-3 circuit
-