The College of New Jersey

"A Verilog Overview"

by

Orlando J. Hernandez, Ph.D.

Electrical & Computer Engineering
School of Engineering
THE COLLEGE OF NEW JERSEY

Presentation Overview

- Introduction to Verilog – Part I
- Introduction to Verilog – Part II
  - AND, OR, HALF ADDER, FULL ADDER
- Introduction to Verilog – Part III
  - ALU Design
- Control and Data Path Organization
  - Finite State Machines, Digital Filter
- Q&A Sessions

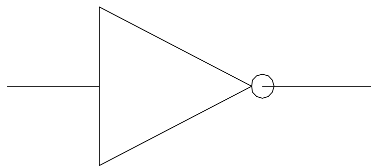
INTRODUCTION TO Verilog
PART I

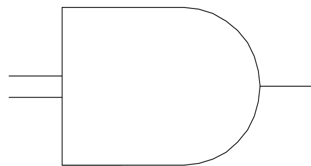
Design Automation

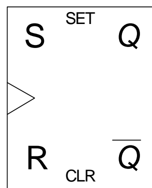
- Need To Keep With Rapid Changes, Electronic Products Have To Be Designed Extremely Quickly
- Electronic Design Automation (EDA)
  - Design Entry
  - Simulation
  - Synthesis
  - Design Validation & Test

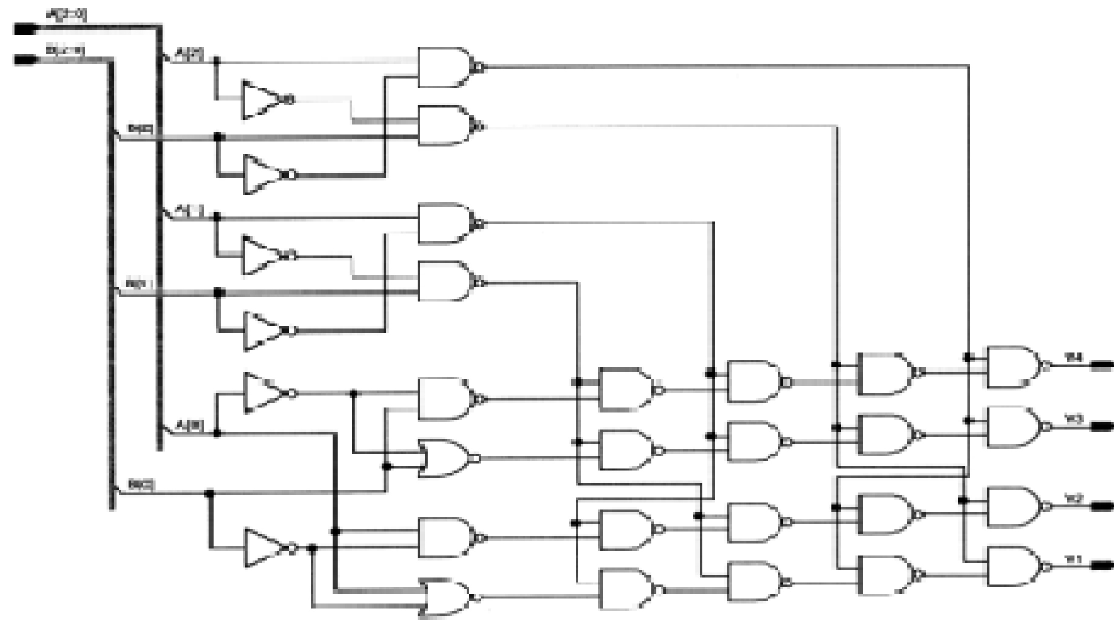
Design Automation. Cont…

- Design Entry
  - Schematic Capture









Design Automation. Cont...

- Design Entry - Textual Form:
  - **Verilog**
  - **VHDL** (**V**HSIC **H**ardware **D**escription **L**anguage)
  - **VHSIC** (**V**ery **H**igh **S**peed **I**ntegrated **C**ircuits)

Design Automation. Cont…

- Design Entry - Textual Form:

```
module and_2 (X, Y, Z);
        input X, Y;
        output Z;

        assign Z = X & Y;
endmodule
```

Introduction To Verilog

- Verilog Is an Industry Standard Language to Describe Hardware From the Abstract to Concrete Level.

The College of New Jersey

BRIEF HISTORY OF Verilog

- Began as a proprietary HDL promoted by Cadence Design Systems.
- Cadence transferred control of Verilog to a consortium of companies and universities known as Open Verilog International (OVI).
- Verilog is an IEEE Standard (IEEE Standard 1364-1995).
- Verilog continues to be extended and upgraded (IEEE Standard 1364-2000, System Verilog).

MOTIVATION

- Need a Method to Quickly Design, Implement, Test and Document Increasingly Complex Digital Systems.
- Schematic and Boolean Equations Inadequate for Million-Gate ICs.
- Design Portability

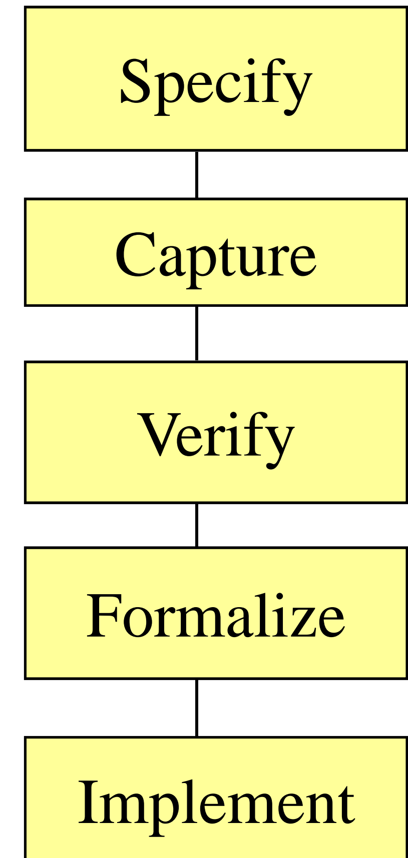
The College of New Jersey

What is Verilog?

- A Design entry language
- A Simulation modeling language.
- A Verification language.
- A Standard language.
- As simple or complex as required.

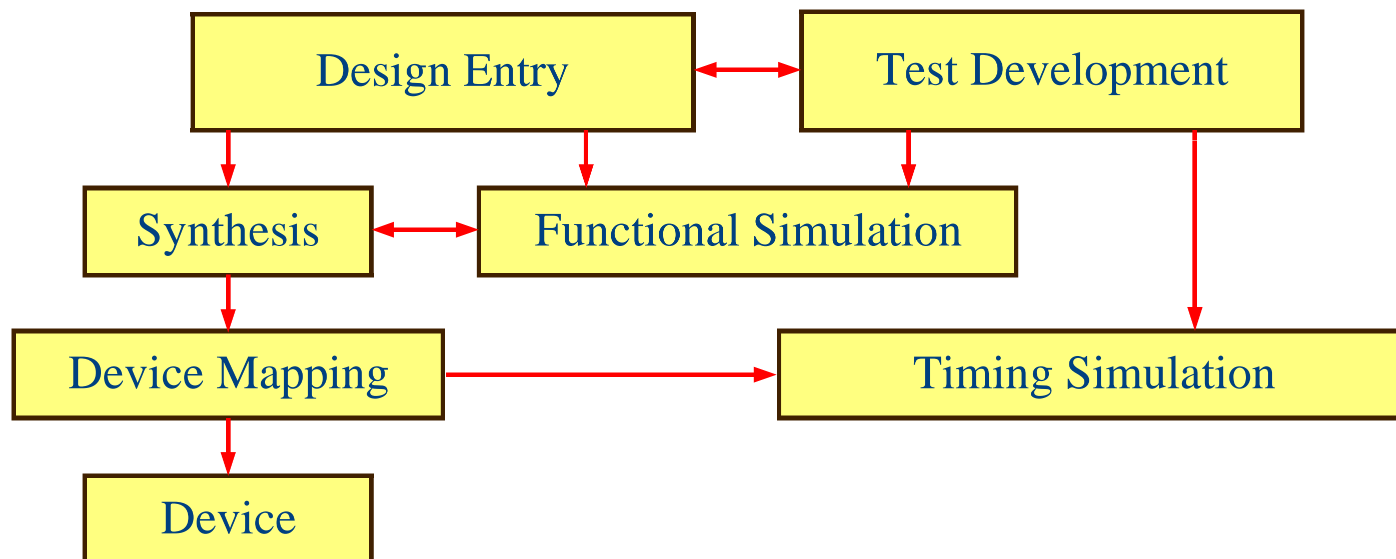
How is Verilog Used?

- For Design Specification ("Specify")
- For Design Entry ("Capture")
- For Design Simulation ("Verify")
- For Design Documentation ("Formalize")
- As an Alternate to Schematics



Design Process (e. g. for FPGAs)

- Verilog Can Be Used for Both Design and Test Development



When Should Verilog Be Used?

- Verilog is highly beneficial to use as a structured, top down approach to design.
- Verilog makes it easy to build, use, and reuse libraries of circuit elements.
- Verilog can greatly improve your chances of moving into more advanced tools and design flows.

The College of New Jersey

Advantages of Verilog

- The Ability to Code the Behavior and to Synthesize an Actual Circuit.
- Power and Flexibility
- Device (specific FPGA) Independent Design
- Technology (specific silicon process) Independent Design

Advantages of Verilog Cont…

- Portability Among Tools and Devices
- Fast Switch Level Simulations
- Quick Time to Market and Low Cost
- Industry Standard

Getting Started with Verilog

- Its Easy To Get Started With Verilog, But It Can Be Difficult To Master It.
- To Begin With, A Subset of The Language Can Be Learned To Write Useful Models.
- Later, More Complex Features Can Be Learned To Implement Complex Circuits, Libraries, And APIs.

A First look at Verilog

- Lets start with a simple Combinational circuit: an 8-bit Comparator.

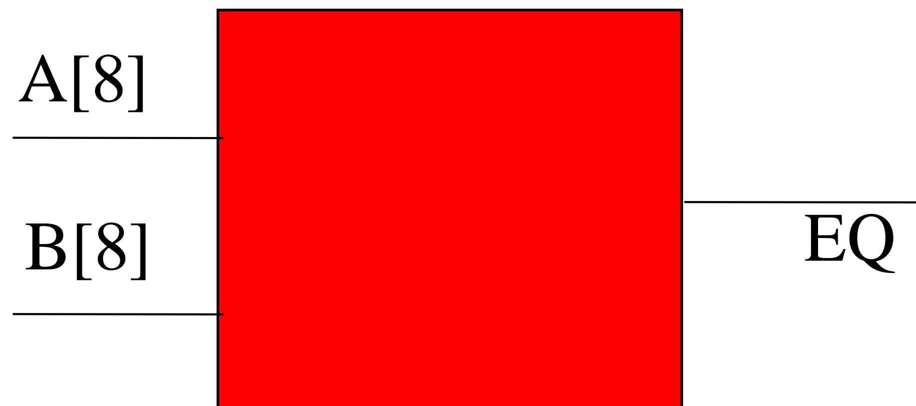
The College of New Jersey

An 8 Bit Comparator

- Comparator Specifications:
  - Two 8-bit inputs
  - 1-bit Output
  - Output is 1 if the inputs match or 0 if they differ.

The College of New Jersey

An 8 Bit Comparator



	0	1	2	3	4	5	6	7
A	1	0	1	1	0	0	1	1
B	1	0	1	1	0	0	1	1

The College of New Jersey

Comparator Verilog Source Code

```
// Eight-bit Comparator
  module compare (A, B, EQ)
      input [7:0] A, B;
      output EQ;

      assign EQ = (A == B);
  endmodule
```

- **Define the inputs and outputs - the ports of the circuit**
- **Define the function of the circuit**

The College of New Jersey

What is a module

- Every Verilog design description has at least one module construct.
- A large design has many modules and are connected to form the complete circuit.
- The module port declarations describe the circuit as it appears from “outside”- from perspective of its input and output interfaces.

What is a module?

```
module compare (A, B, EQ)
        input [7:0] A, B;
        output EQ;
                :
                :
```

- The module and port declarations includes a name, compare, and port direction statements defining all the inputs and outputs of the module.
- The Rest of the module Describes the Actual Function.

The College of New Jersey

What is a module?

```
:
:
assign EQ = (A == B);
endmodule
```

- Before the keyword endmodule is found the actual functional description of the comparator.

Data Types

- Verilog's high level data types allow data to be represented in much the same way as in high-level programming languages.
- A data type is an abstract representation of stored data.

Data Types

- These data types might represent individual wires in a circuit, or a collection of wires.

Data Types

- Basic Data Types
  - Nets
    - **wire**, **wand**, **tri**, **wor**
    - Continuously driven
    - Gets new value when driver changes
    - LHS of continuous assignment

```
tri [15:0] data;
    // unconditional
assign data[15:0] = data_in;
    // conditional
assign data[15:0] = enable ? data_in : 16'bz;
```

  - Registers
    - Reg
    - Represents storage
    - Always stores last assigned value
    - LHS of an assignment in a procedural block

```
reg signal;
@(posedge clock) signal = 1'b1;
    // possitive edge
@(reset) signal = 1'b0;  // event (both edges)
```

The College of New Jersey

Some Data Types

Data Type	Values	Examples
Bit	'1' , '0' , 'x' , 'z'	Q = 1’b1;
Array of bits	"101001"	Data[5:0] = 6’b101001;
Boolean	Use Bit	EQ = 1’b1; // True
Integer	-2, -1, 0, 1, 2, 3	C = c+2;
Real	1.0, -1.0E5	V1 − V2/5.3;
Time	‘timescale 1ns/1ps	#6 Q = 1’b1;
Register	Single or array of bits	
Character	Use 8-bit register	
String	Use register of length 8 x the # of characters	

Design Units

- Design units are a concept that provide advanced configuration management capabilities.

- Design units are modules of Verilog that can be compiled separately and stored in a library.

The College of New Jersey

Library Design unit

- A Library is a collection of commonly used modules to be used globally among different design units.
- Library is identified with compiler/simulator command line switches.

Levels of Abstraction (Styles)

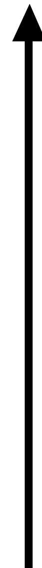
- Verilog supports many possible styles of design description.

- These styles differ primarily in how closely they relate to the underlying hardware.

Levels of Abstraction (Styles)

- Levels of Abstraction refers to how far your design description is from an actual hardware realization.
- The three main levels of abstraction are:
  - Behavior
  - Dataflow
  - Structure

Levels of Abstraction (Styles)

Behavior		Performance Specification Test Benches Sequential Description State Machines
Dataflow		Register Transfers Selected Assignments Arithmetic Operation Boolean Equations
Structure		Hierarchy Physical Information

Behavioral Modeling

- The Highest Level of Abstraction Supported in Verilog.

- The Behavior Approach Describes the Actual Behavior of Signals Inside the Component.

Verilog Timing Issues

- The Concept of Time Is the Critical Distinction Between Behavioral Descriptions and Low Level Descriptions.
- The Concept to Time May Be Expressed Precisely, With Actual Delays Between Related Events

An Example of Behavioral Modeling: A half adder

half_adder

- Half Adder
- Inputs a, b : 1 bit each.
- Output Sum, Carry : 1 bit each.

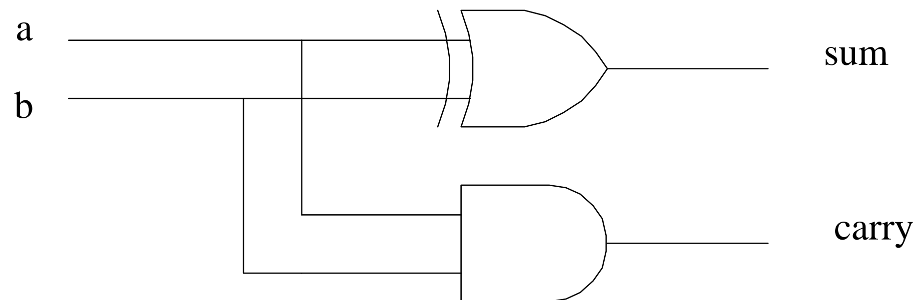


Figure 1-1 Half adder circuit

The College of New Jersey

Verilog Code for half_adder

```
// Half Adder
module half_adder (a, b, sum, carry);
    input a, b;
    output sum, carry;

    reg sum, carry;

    always @ (a or b) begin
        sum = a ^ b;
        carry = a & b;
    end
endmodule
```

INTRODUCTION TO Verilog PART II

The College of New Jersey

Dataflow Modeling

- The dataflow level of abstraction is often called Register Transfer Language (RTL).
- Some behavioral modeling can also be called RTL.
- The dataflow level of abstraction describes how information is passed between registers in the circuit.

Concurrent and Sequential Verilog

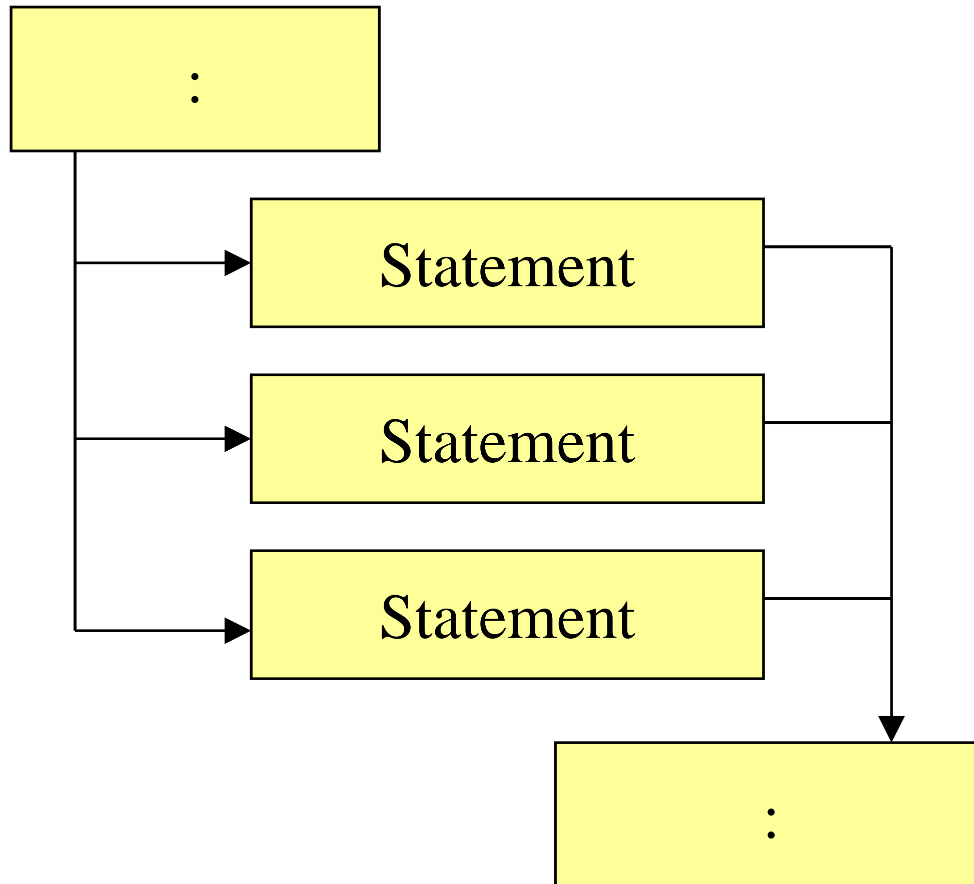
- Verilog Allows Both Concurrent and Sequential Statements to Be Entered.

- The Difference Between Concurrent and Sequential Statements Must Be Known for Effective Use of the Language.

Concurrent Verilog

- All Statements in the Concurrent Area Are Executed at the Same Time.
- There Is No Significance to the Order in Which Concurrent Statements Occur.

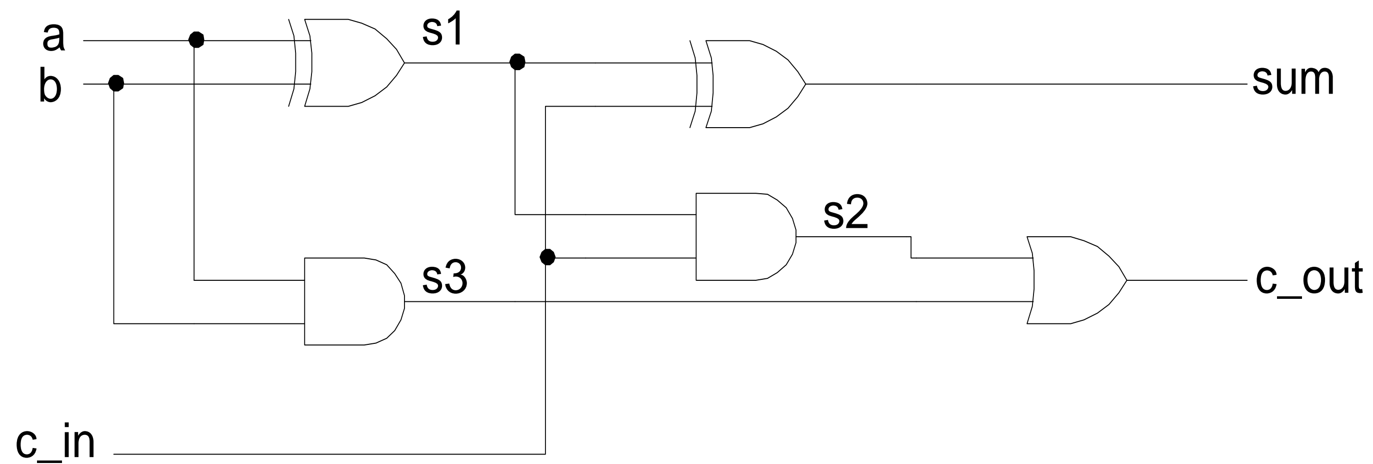
Concurrent Verilog



Example of Concurrent Verilog

Full Adder circuit

The College of New Jersey



The College of New Jersey

Verilog code for Full Adder

```
// Full Adder Using Signal Assignment Instructions
module full_adder (a, b, c_in, sum, c_out);
    input a, b, c_in;
    output sum, c_out;

    wire s1, s2, s3;

    assign s1 = a ^ b;
    assign s2 = s1 & c_in;
    assign s3 = a & b;
    assign sum = s1 ^ c_in;
    assign c_out = s2 | s3;
endmodule
```

Verilog code for Full Adder

- The **assign** expressions are all concurrent signal assignment statements. All the statements are executed at the same time.

```
assign s1 = a ^ b;
assign s2 = s1 & c_in;
assign s3 = a & b;
assign sum = s1 ^ c_in;
assign c_out = s2 | s3;
```

The College of New Jersey

Verilog code for Full Adder

- The simulator evaluates all the **assign** expressions, and then applies the results to the signals.
- Once the simulator has applied the results it waits for one of the signal to change and it reevaluates all the expressions again.

Verilog code for Full Adder

- This cycle will continue until the simulation is completed.
- This is called “event driven simulation”.
- It is more computationally efficient than time driven simulation.

The College of New Jersey

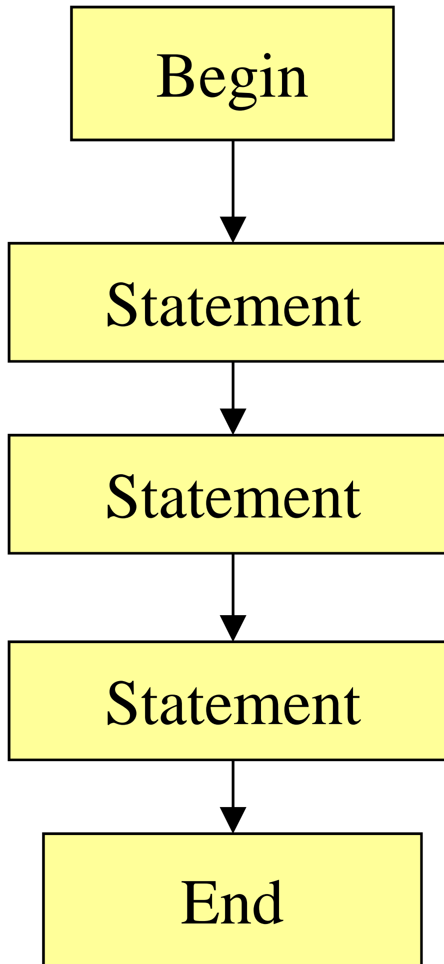
Wires

- In the full_adder Verilog code we came across “wire”.
- So what are “wires”?
  - Wires Are Used to Carry Data From Place to Place in a Verilog Design Description.
  - Wires in Verilog Are Similar to Wires in a Schematic.
  - Wires are internal to a module.

Sequential Verilog

- Sequential Statements Are Executed One After the Other in the Order That They Appear.
- Example of Sequential Statement: Always.

Sequential Verilog



The College of New Jersey

Always Construct

- The Always construct is the primary means to describe sequential operations.
- Always starts with the keyword *always*, then *begin*, and ends with the keyword *end*.
- The whole *always* construct itself is treated as a concurrent statement.

The College of New Jersey

Always Statement

- The *always* construct consists of three parts
  - Sensitivity List
  - Declaration Part
  - Statement Part

The College of New Jersey

Syntax of Always Statement

```
module module_name ( … ports … );
        :
  always @ (sensitivity_list)
        begin : block_name
                  local_declaration;
                  ……
                  sequential statement;
                  sequential statement;
                  ……
        end
endmodule
```

The College of New Jersey

Always Example

```
module nand2 (a, b, c);
    input a, b;
    output c;

    reg c;

    always @ (a or b)
        begin : nand2_always_block
            reg temp;
            temp = ~(a & b);
            if (temp == 1'b1) #5 c = temp;
            else if (temp == 1'b0) #6 c = temp;
        end
endmodule
```

Example Description

- The **always** sensitivity list enumerates exactly which signals causes the block to execute.

```
always @ (a or b)
```

The College of New Jersey

Example Description

- The declarative part is used to declare local variables or constants that can be used in the block.

```
reg temp;
```

Example Description

- Variables are temporary storage areas similar to variables in software programming languages.

```
reg temp;
```

Use of Sequential Statements

- Sequential Statements Exist Inside the Always Statements As Well As in Sub Programs.

- The Sequential Statements Are:

if	case	forever	repeat
while	for	wait	fork/join

if Statements

- The IF statement starts with the keyword ***if*** and ends with the keyword ***end***.

```
if (x < 10) begin
    a = b;
end
```

if Statements

- There are also two optional clauses
  - else if clause
  - else clause

```
if (day == Sunday) begin
    weekend = true;
end
else if (day == Saturday) begin
    weekend = true;
end
else begin
    weekday = true;
end
```

The College of New Jersey

if Statements

- The **if** statement can have multiple **else if** statement parts but only one **else** statement part.

The College of New Jersey

Case Statement

- The Case statement is used whenever a single expression value can be used to select between a number of actions.

- A Case statement consists of the keyword ***case*** followed by an operator expression, and ended with an ***endcase*** keyword.

The College of New Jersey

Case Statement

- The expression will either return a value that matches one of the ***choices*** in a statement part or match a ***default*** clause.

The College of New Jersey

Case Statement Example

```
reg [1:0] bit_vec;
……
case bit_vec
    2'b00 :
    return = 0;
    2'b01 :
    return = 1;
    2'b10 :
    return = 2;
    2'b11 :
    return = 3;
endcase
```

The College of New Jersey

Loop Statements

- The loop statement is used whenever an operation needs to be repeated.

- Loop statements are implemented in three ways
  - ***repeat*** condition loop statement
  - ***while*** condition loop statement
  - ***for*** condition loop statement

Loop Statements (*repeat*)

- The ***repeat*** condition Loop statement will loop as many times as the condition expression.

```
repeat (flag) begin
    day =  get_next_day (day);
end
```

Loop Statements (*while*)

- The ***while*** condition Loop statement will loop as long as the condition expression is TRUE.

```
while (day == weekday) begin
    day =  get_next_day (day);
end
```

The College of New Jersey

Loop Statements (*for loop*)

```
for (i = 1; i <= 10; i = i + 1) begin
    i_squared[i] = i*i;
end
```

- This loop will execute 10 times whenever execution begins and its function is to calculate squares from 1 to 10 and insert them into i_squared memory.

The College of New Jersey

Wait Statement

- The ***wait*** statement allows to suspend the sequential execution based on a conditional expression.

- ***wait*** until an expression is true.

```
wait (conditional expression)
```

Wait Statement

- The ***wait*** conditional expression clause will suspend execution of the process until the expression returns a true value.

```
initial
   begin
      wait (!oe)
      o = q;
   end
```

Structural Verilog

The College of New Jersey

- Structural-level design methods can be useful for managing the complexity of a large design description.
- Structure level of abstraction is used to combine multiple components to form a larger circuit.

Structural Verilog

- Structural Verilog Descriptions Are Quite Similar in Format to Schematic Netlists.

- Larger Circuits Can Be Constructed From Smaller Building Blocks.

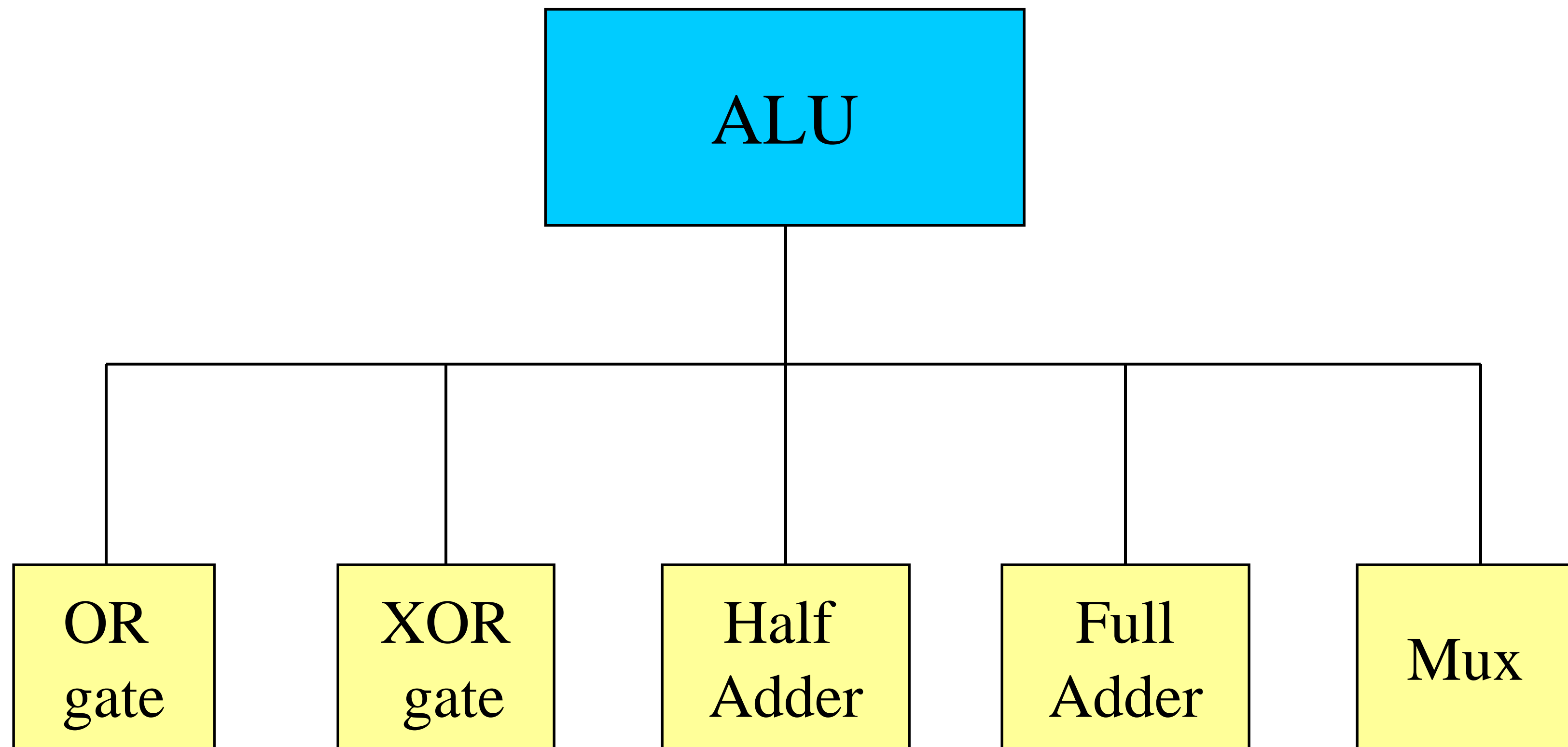
Example of Structural Verilog

Let us consider an ALU with

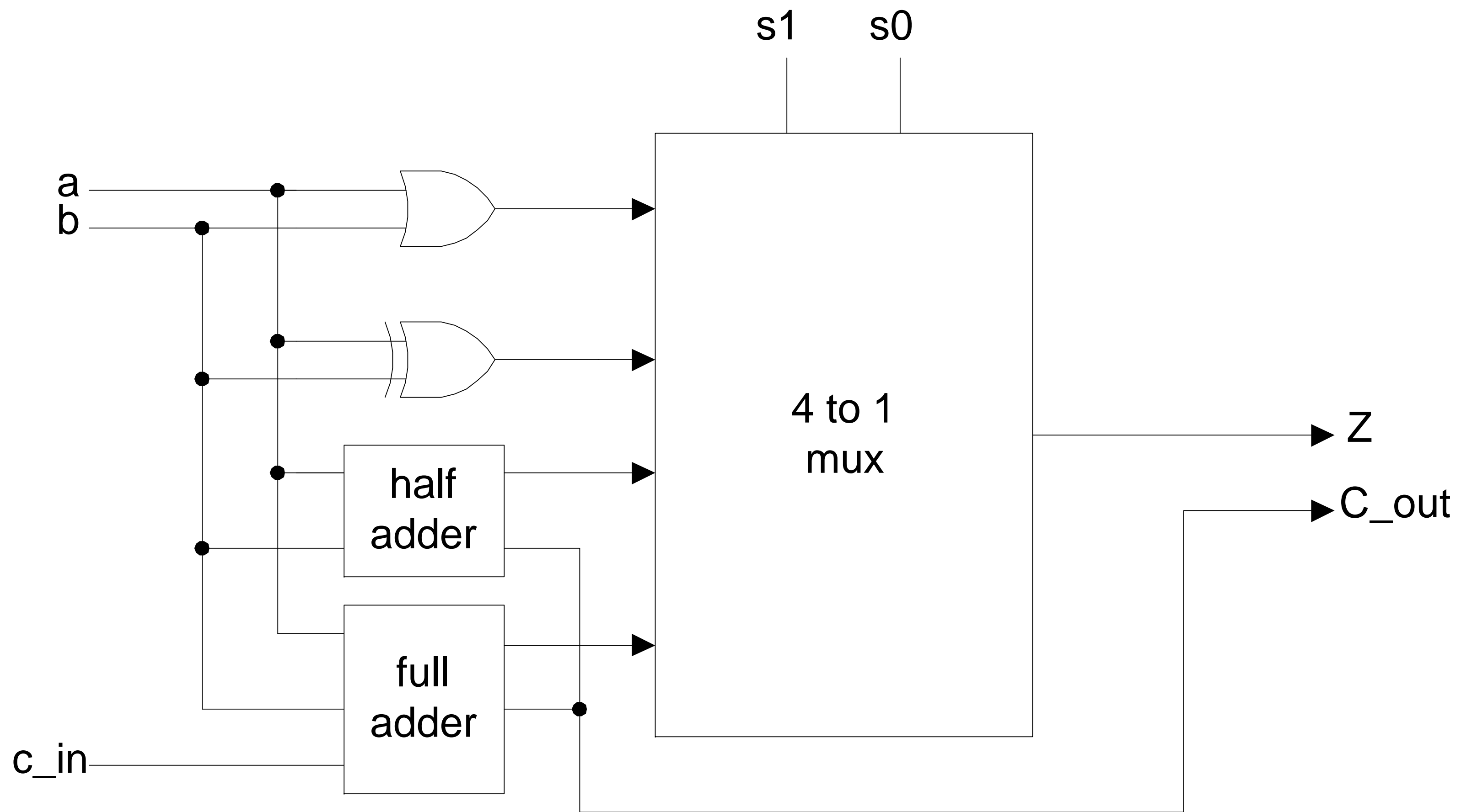
- An OR gate
- An XOR gate
- A Half Adder
- A Full Adder
- A Multiplexer

Example of Structural Verilog

The College of New Jersey

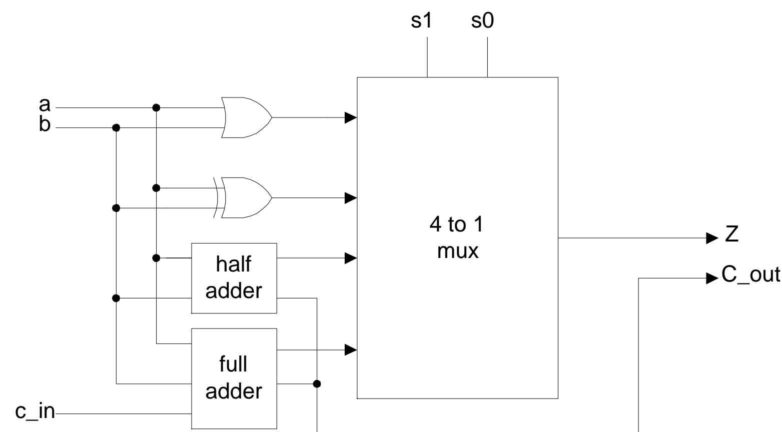


ALU – Block Diagram



ALU – Function Table

S1	S0	Z	C_out
0	0	a or b	0
0	1	a xor b	0
1	0	ha_sum	ha_c_out
1	1	fa_sum	fa_c_out



The College of New Jersey

Verilog code for OR gate

```
module t_or (a, b, ored);
   input a, b;
   output ored;

   assign ored = a | b;
endmodule
```

The College of New Jersey

Verilog code for XOR

```
module t_xor (a, b, xored);
  input a, b;
  output xored;

  assign xored = a ^ b;
endmodule
```

Verilog Code for half_adder

```
// Half Adder
module half_adder (a, b, sum, c_out);
  input a, b;          // declaring I/O ports
  output sum, c_out;

  assign sum = a ^ b;
  assign c_out = a & b;
endmodule
```

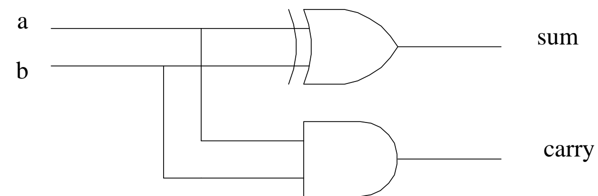
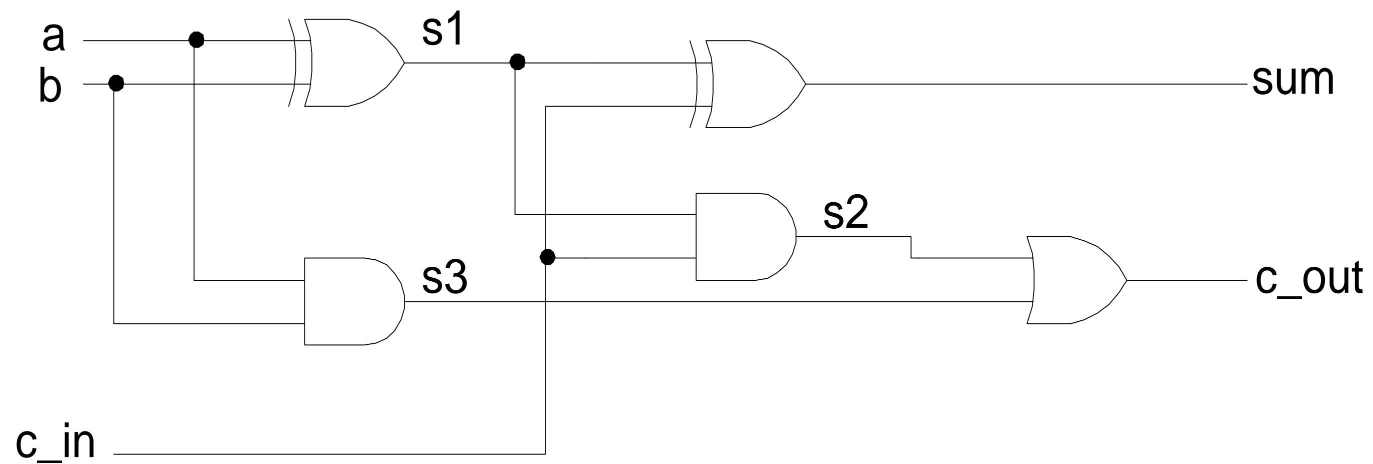


Figure 1-1 Half adder circuit

Full Adder circuit



The College of New Jersey

Verilog code for full_adder

```
// Full Adder
module full_adder (a, b, c_in, sum, c_out);
    input a, b, c_in;
    output sum, c_out;

    wire s1, s2, s3;

    assign s1 = a ^ b;
    assign s2 = c_in & s1;
    assign s3 = a & b;
    assign sum = a ^ b;
    assign c_out = s2 | s3;
endmodule

// Using Signal Assignment Instructions
```

The College of New Jersey

Main Code for ALU

```
module alu (a, b, c_in, s0, s1, z, c_out);
    input a, b, c_in, s0, s1;
    output z, c_out;

    reg z, c_out;

    wire ored, xored, ha_sum, ha_c_out, fa_sum, fa_c_out;

    t_or a1 (.a(a), .b(b), .ored(ored));
    t_xor x1 (.a(a), .b(b), .xored(xored));
    half_adder h1 (.a(a), .b(b), .sum(ha_sum),
            .c_out(ha_c_out));
    full_adder f1 (.a(a), .b(b), .c_in(c_in), .sum(fa_sum),
            .c_out(fa_c_out));
```

Main Code for ALU Cont….

```
always @ (a or b or c_in or s0 or s1) begin
        if (s1 == 1'b0 && s0 == 1'b0) begin
                z = ored;
                c_out = 1'b0;
        end
        if (s1 == 1'b0 && s0 == 1'b1) begin
                z = xored;
                c_out = 1'b0;
        end
        if (s1 == 1'b1 && s0 == 1'b0) begin
                z = ha_sum;
                c_out = ha_c_out;
        end
        if (s1 == 1'b1 && s0 = 1'b1) begin
                z = fa_sum;
                c_out = fa_c_out;
        end
    end
endmodule
```

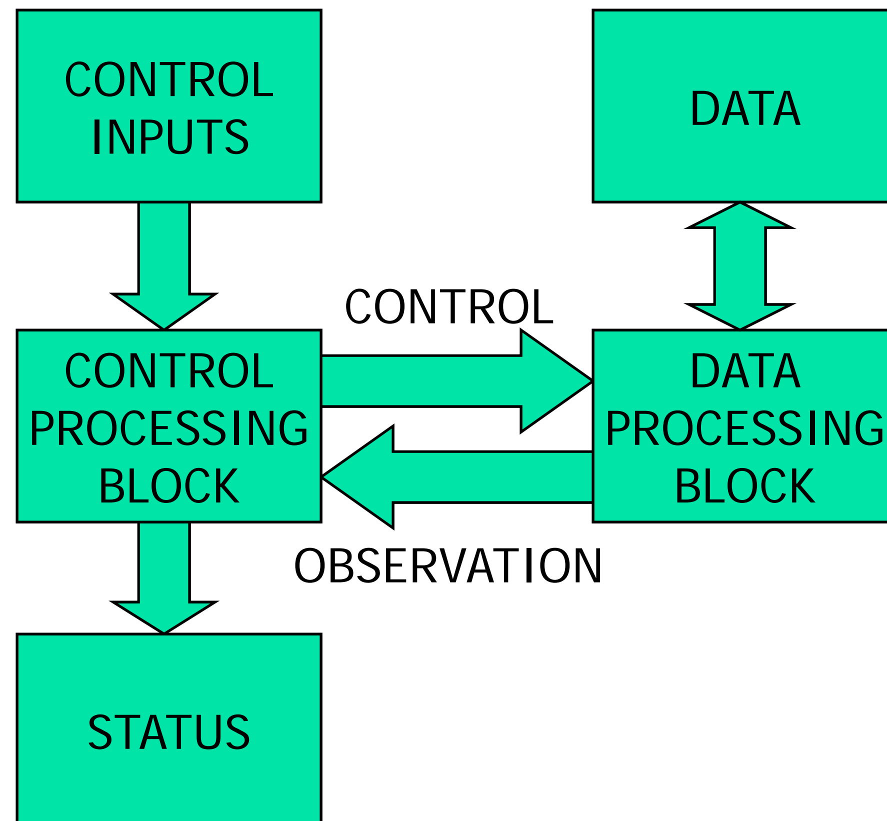
The College of New Jersey

CONTROL AND DATA PATH ORGANIZATION

Control and Data Path Organization

- Most complex digital circuits can be broken up into two parts:
  - Control
  - Data Path

Control and Data Path Organization

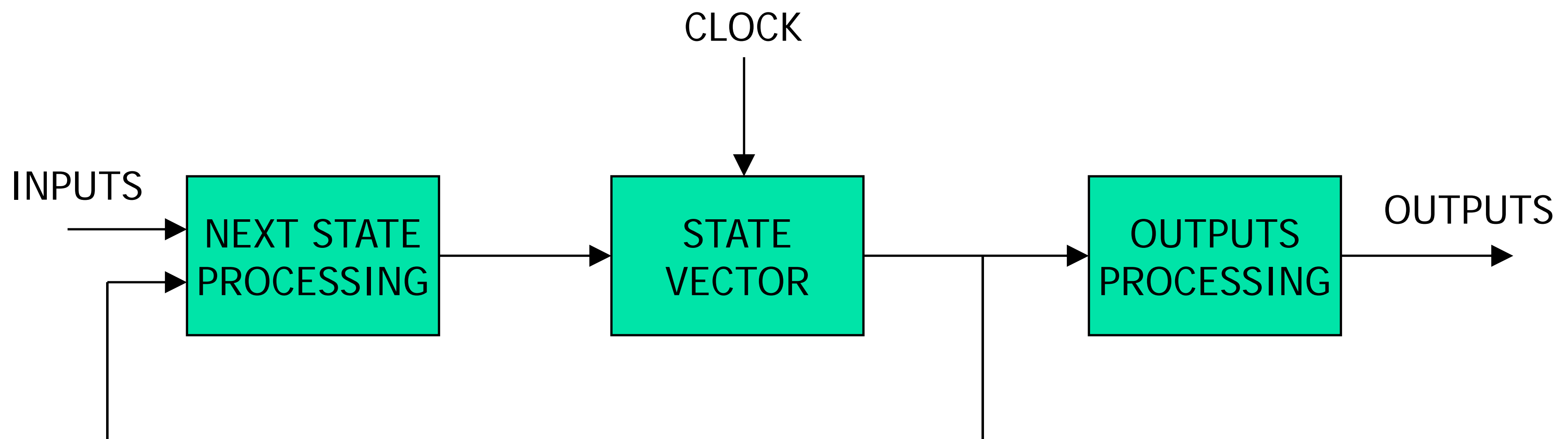


Finite State Machines

- Two Classes of Finite State Machines (FSMs):
  - Moore Machines
  - Mealy Machines

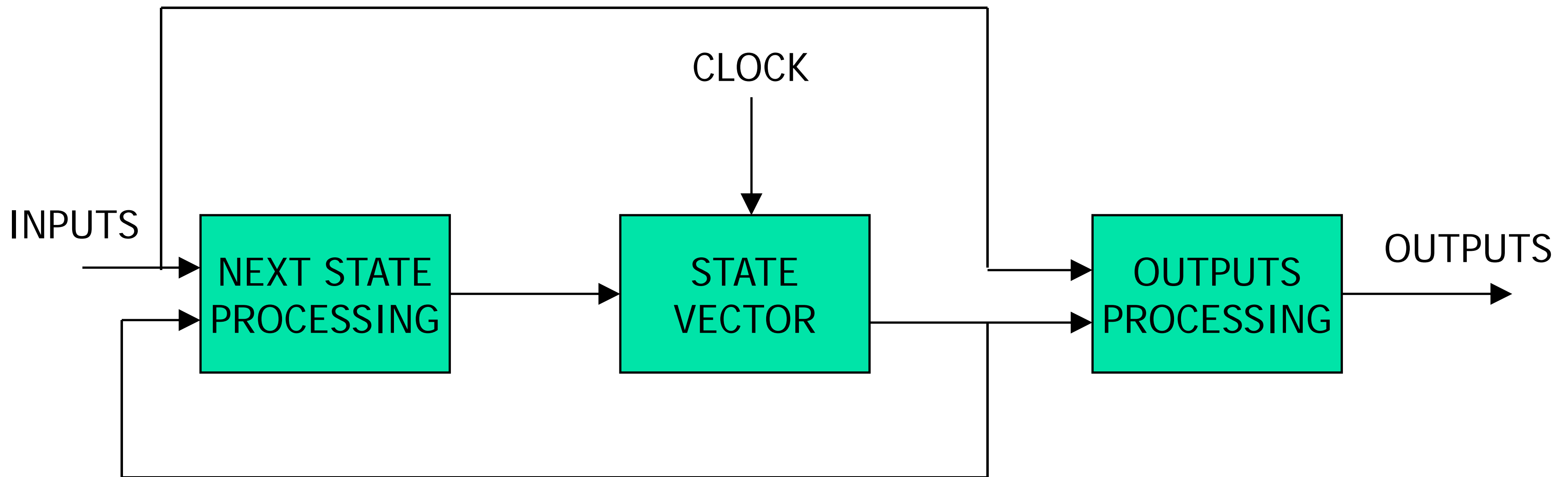
Moore Finite State Machines

- **Outputs depend only on the state**
- State and Outputs Processing are combinational elements
- State Vector is Sequential Elements



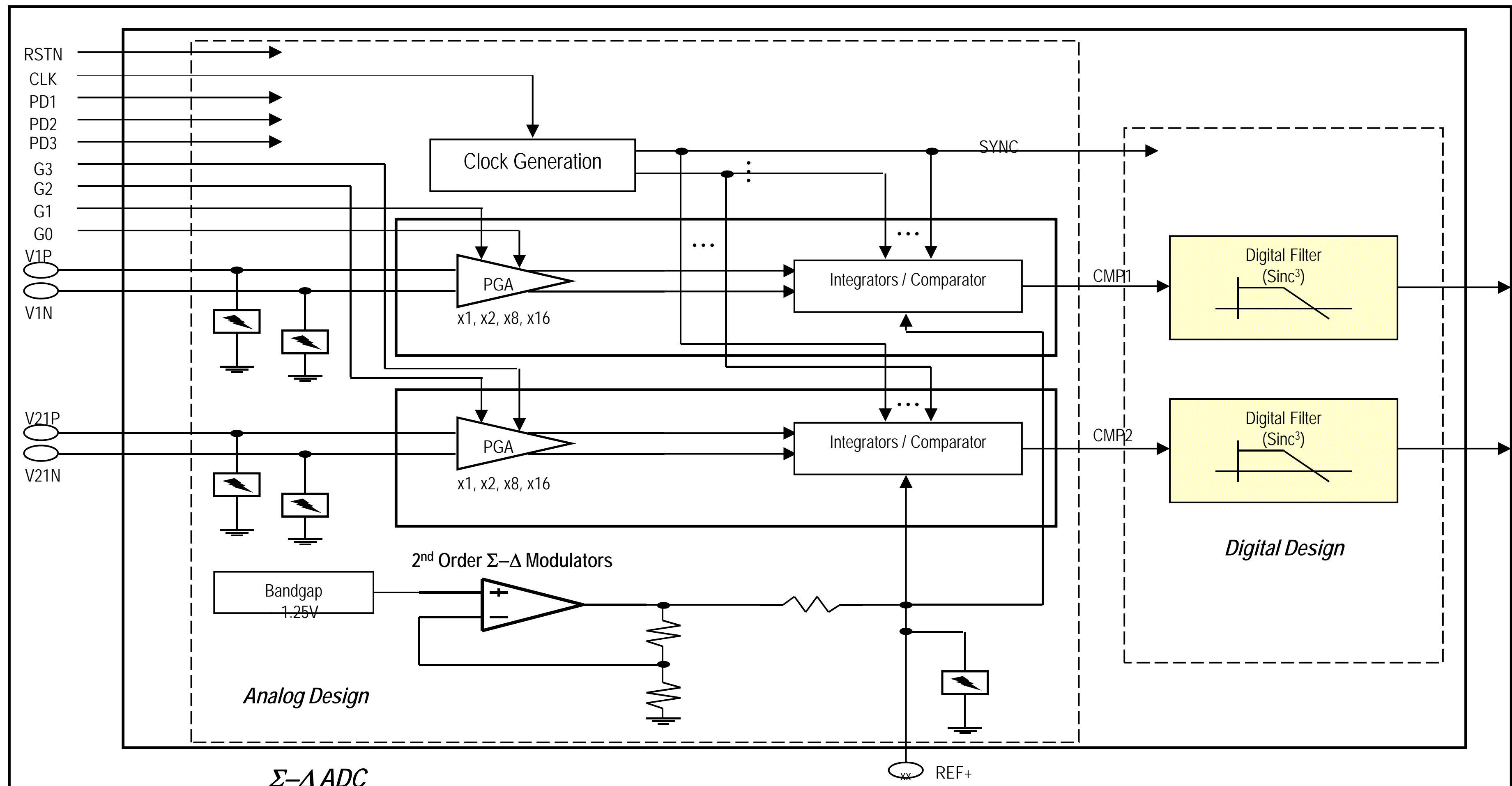
Mealy Finite State Machines

- Outputs depend on the state and the inputs



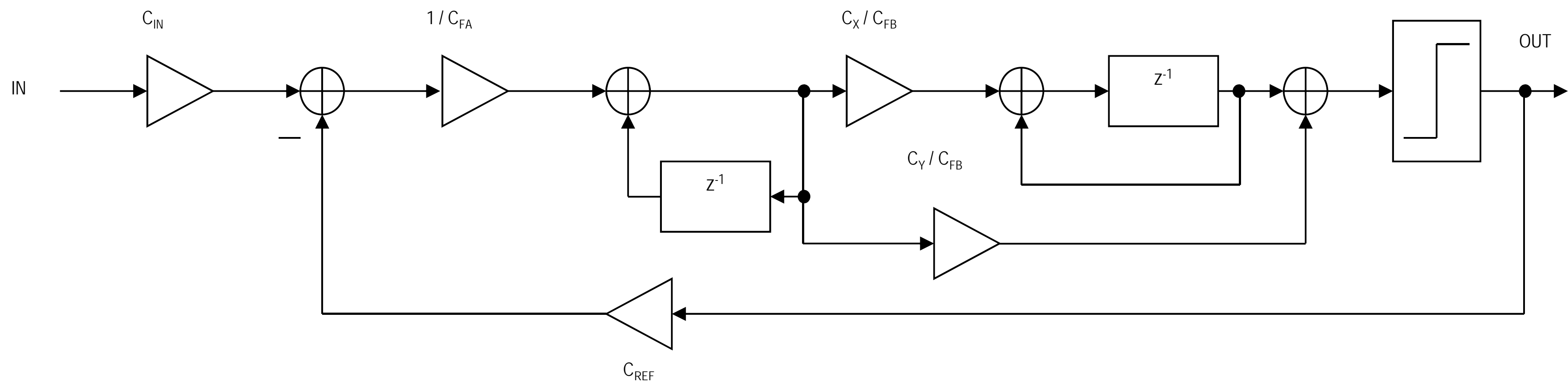
Verilog IMPLEMENTATION EXAMPLES – A Decimation Filter for a Sigma-Delta Analog to Digital Converter

2-Ch Σ-Δ Analog to Digital Converter



2nd Order Σ-Δ Modulator (block/algorithmic)

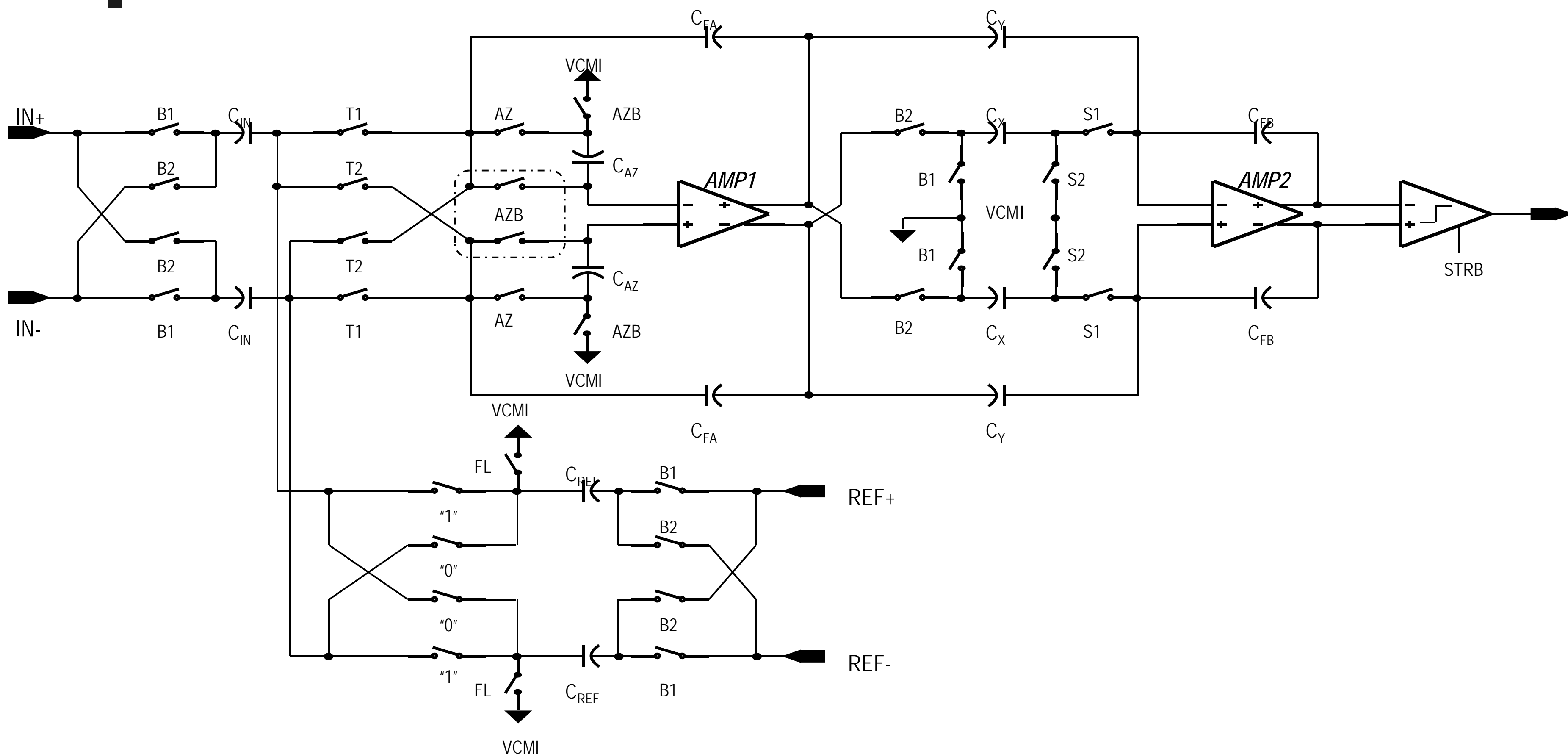
The College of New Jersey



- This can be modeled in Behavioral Verilog

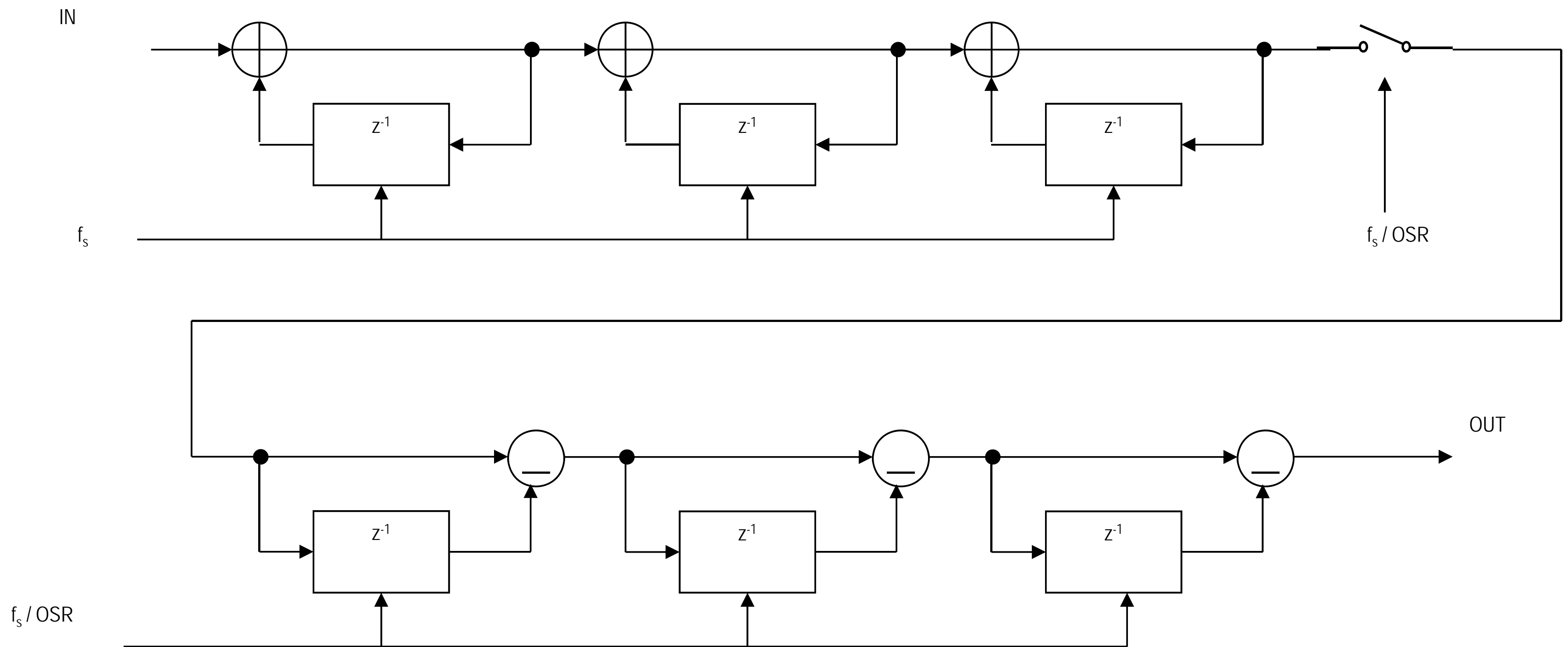
2nd Order Σ-Δ Modulator (circuit)

The College of New Jersey



- This can also be modeled in Behavioral Verilog

Decimation Digital Filter



Decimation Digital Filter

- Cubic sinc
- Bits of noise free accuracy for delta-sigma ADC's:
  - BITS = 3 * LOG(OSR) / LOG(2) + 2
  - Assume OSR=32, then BITS=17, and set BITS=16

Decimation Digital Filter

- First Filter Equations
  - $H_1(z) = Y_1(z)/X(z) = 1/(1 - 3z^{-1} + 3z^{-2} - z^{-3})$
  - $y_1(n) = x(n) + 3y_1(n-1) - 3y_1(n-2) + y_1(n-3)$
- Second Filter Equations
  - $H_2(z) = Y(z)/X_1(z) = 1 - 3z^{-1} + 3z^{-2} - z^{-3}$
  - $y(n) = x_1(n) - 3x_1(n-1) + 3x_1(n-2) - x_1(n-3)$
- Decimation (Retiming)
  - $x_1(n)=y_1(n/OSR)$
  - $x_1(n)=y_1(n/32)$

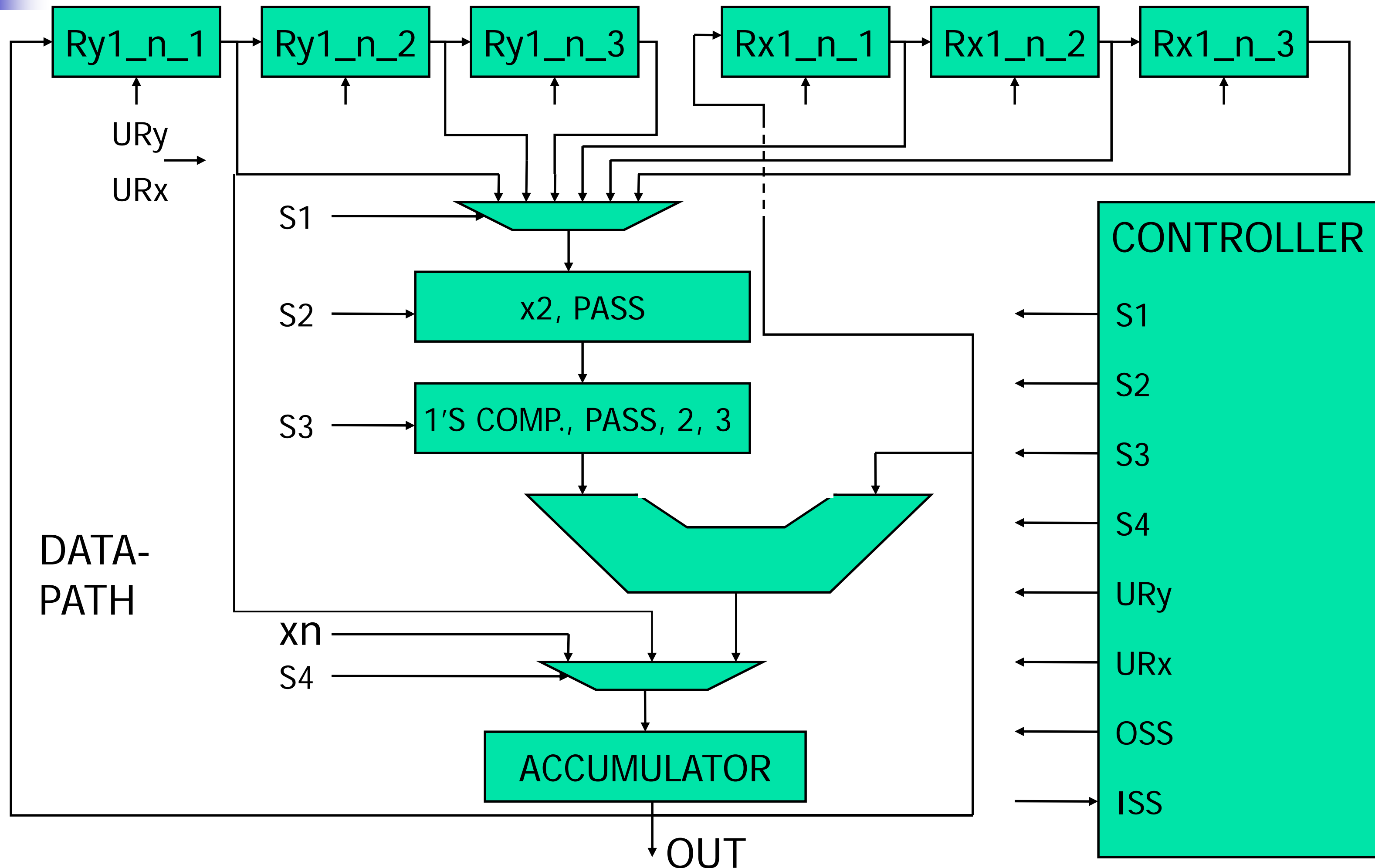
What do we need for our design?

- $y_1(n) = x(n) + 3\, y_1(n-1) - 3\, y_1(n-2) + y_1(n-3)$
- $y(n) = x_1(n) - 3\, x_1(n-1) + 3\, x_1(n-2) - x_1(n-3)$
- $x_1(n)=y_1(n/32)$

- Control
  - On every x(n)
    - S02: Store x(n) in accumulator, count x(n) mod 32
    - S03: Accumulate 2 $y_1(n-1)$
    - S04: Accumulate $y_1(n-1)$
    - S05: Accumulate 1's complement of 2 $y_1(n-2)$
    - S06: Accumulate 1's complement of $y_1(n-2)$
    - S07: Accumulate 2
    - S08: Accumulate $y_1(n-3)$
    - S09: Update y registers
  - On every $x_1(n)$ (every 32nd $y_1(n)$)
    - S10: Accumulate 1's complement of 2 $x_1(n-1)$
    - S11: Accumulate 1's complement of $x_1(n-1)$
    - S12: Accumulate 2 $x_1(n-2)$
    - S13: Accumulate $x_1(n-2)$
    - S14: Accumulate 1's complement of $x_1(n-3)$
    - S15: Accumulate 3, output result
    - S16: Store $y_1(n-1)$ in accumulator
    - S17: Update x registers
- Data Path
  - 16 bits
  - Adder-Accumulator
  - 1's complement
  - Shift left by one (x 2)
  - Store $y_1(n-1)$, $y_1(n-2)$, $y_1(n-3)$
  - Store $x_1(n-1)$, $x_1(n-2)$, $x_1(n-3)$
  - Constants: 2 & 3

The College of New Jersey

Decimation Digital Filter Architecture



Verilog code for Data_Path

```
// Data_Path
module Data_Path (CLK, reset, xn, URy, URx,
                  S2, S3, S4, S1,
                  OUTPUT);

   input CLK, reset, xn, URy, URx, S2;
   input [1:0] S3, S4;
   input [2:0] S1;
   output [15:0] OUTPUT;
```

The College of New Jersey

Verilog code for Data_Path

```
reg [15:0] Ry1_n_1, Ry1_n_2, Ry1_n_3;
reg [15:0] Rx1_n_1, Rx1_n_2, Rx1_n_3;
reg [15:0] ACCUMULATOR;

parameter my_zero = 16'b0000000000000000;

reg [15:0] T1, T2, T3, T4, T5;
reg my_msb;

assign OUTPUT = ACCUMULATOR;

always @ (posedge CLK or reset) begin
 if (reset == 1'b1) begin
   Ry1_n_1 = my_zero; Ry1_n_2 = my_zero; Ry1_n_3 = my_zero;
   Rx1_n_1 = my_zero; Rx1_n_2 = my_zero; Rx1_n_3 = my_zero;
   ACCUMULATOR = my_zero;
 else if (CLK == 1'b1) begin
   if (URy == 1'b1) begin
     Ry1_n_3 = Ry1_n_2; Ry1_n_2 = Ry1_n_1;
     Ry1_n_1 = ACCUMULATOR;
   end
   if (URx == 1'b1) begin
     Rx1_n_3 = Rx1_n_2; Rx1_n_2 = Rx1_n_1;
     Rx1_n_1 = ACCUMULATOR;
   end
   case (S1)
     3'b000 : T1 = Ry1_n_1;
     3'b001 : T1 = Ry1_n_2;
     3'b010 : T1 = Ry1_n_3;
     3'b011 : T1 = Rx1_n_1;
     3'b100 : T1 = Rx1_n_2;
     3'b101 : T1 = Rx1_n_3;
   endcase
   case (S2)
     1'b0 :
       my_msb = T1[15];
       T2 = T1 << 1;
       T3 = T2 & 16'b0111111111111111;
       T4 = T3 | {my_msb, 15'b000000000000000};
     1'b1 : T4 = T1;
   endcase
   case (S3)
     2'b00 : T5 = ~T4;
     2'b01 : T5 = T4;
     2'b10 : T5 = 16'b0000000000000010;
     2'b11 : T5 = 16'b0000000000000011;
   endcase
   case (S4)
     2'b00 : ACCUMULATOR = {15'b000000000000000, xn};
     2'b01 : ACCUMULATOR = Ry1_n_1;
     2'b10 : ACCUMULATOR = ACCUMULATOR + T5;
   endcase
  end
 end
endmodule
```

Verilog code for Controller

```
// Controller
module Controller (CLK, reset, ISS, URy, URx,
                   S2, OSS, S3, S4, S1);

  input CLK, reset, ISS;
  output URy, URx, S2, OSS;
  output [1:0] S3, S4;
  output [2:0] S1;
```

Verilog code for Controller

```
parameter S00 = 5'h00, S01 = 5'h01, S02 = 5'h02, S03 = 5'h03,
  S04 = 5'h04, S05 = 5'h05, S06 = 5'h06, S07 = 5'h07, S08 = 5'h08,
  S09 = 5'h09, S10 = 5'h0A, S11 = 5'h0B, S12 = 5'h0C, S13 = 5'h0D,
  S14 = 5'h0E, S15 = 5'h0F, S16 = 5'h10, S17 = 5'h11;

reg URy, URx, S2, OSS;
reg [1:0] S3, S4;
reg [2:0] S1;

reg [4:0] PRState, NXState;

reg [4:0] Counter;

always @ (PRState) begin
  if (PRState == S09) URy = 1'b1;
  else URy = 1'b0;
  if (PRState == S17) URx = 1'b1;
  else URx = 1'b0;
  if (PRState == S05 || PRState == S06) S1 = 3'b001;
  else if (PRState == S08) S1 = 3'b010;
  else if (PRState == S10 || PRState == S11) S1 = 3'b011;
  else if (PRState == S12 || PRState == S13) S1 = 3'b100;
  else if (PRState == S14) S1 = 3'b101;
  else S1 = 3'b000;
  if (PRState == S03 || PRState == S05 ||
     PRState == S10 || PRState == S12) S2 = 1'b0;
  else S2 = 1'b1;
  if (PRState == S05 || PRState == S06 ||
     PRState == S10 || PRState == S11 ||
     PRState == S14) S3 = 2'b00;
  else if (PRState == S07) S3 = 2'b10;
  else if (PRState == S15) S3 = 2'b11;
  else S3 = 2'b01;
  if (PRState == S02) S4 = 2'b00;
  else if (PRState == S16) S4 = 2'b01;
  else S4 = 2'b10;
  if (PRState == S15) OSS = 1'b1;
  else OSS = 1'b0;
end
```

The College of New Jersey

Verilog code for Controller

```
always @ (posedge CLK or reset) begin
 if (reset == 1'b1) begin
   PRState = S00;
   Counter = 5'b00000;
  end
 else begin
   PRState = NXState;
  if (Counter == 5'b11111) Counter = 5'b00000;
   else Counter = Counter + 5'b00001;
  end
end
always @ (PRState or ISS) begin
 case (PRState)
   S00 : if (ISS == 1'b1) NXState = S02;
   S01 : if (ISS == 1'b1) NXState = S02;
   S02 : NXState = S03;
   S03 : NXState = S04;
   S04 : NXState = S05;
   S05 : NXState = S06;
   S06 : NXState = S07;
   S07 : NXState = S08;
   S08 : NXState = S09;
   S09 : if (Counter == 5'b00000) NXState = S10;
         else NXState = S01;
   S10 : NXState = S11;
   S11 : NXState = S12;
   S12 : NXState = S13;
   S13 : NXState = S14;
   S14 : NXState = S15;
   S15 : NXState = S16;
   S16 : NXState = S17
   S17 : NXState = S01;
  endcase
 end
endmodule
```

The College of New Jersey

Main Code for FILTER

```
module FILTER (reset, CLK, ISS, xn,
                          OSS, OUTPUT);

  input reset, CLK, ISS, xn;
  output OSS;
  output [15:0] OUTPUT;
```

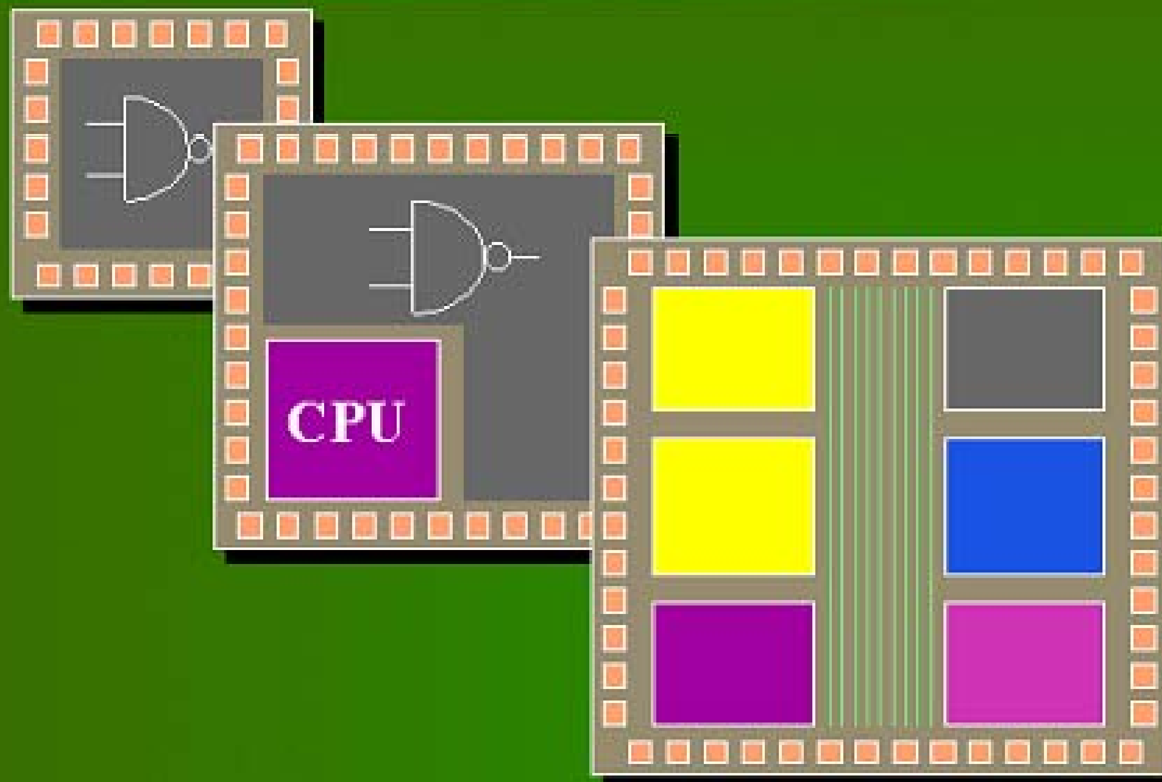
The College of New Jersey

Main Code for FILTER Cont...

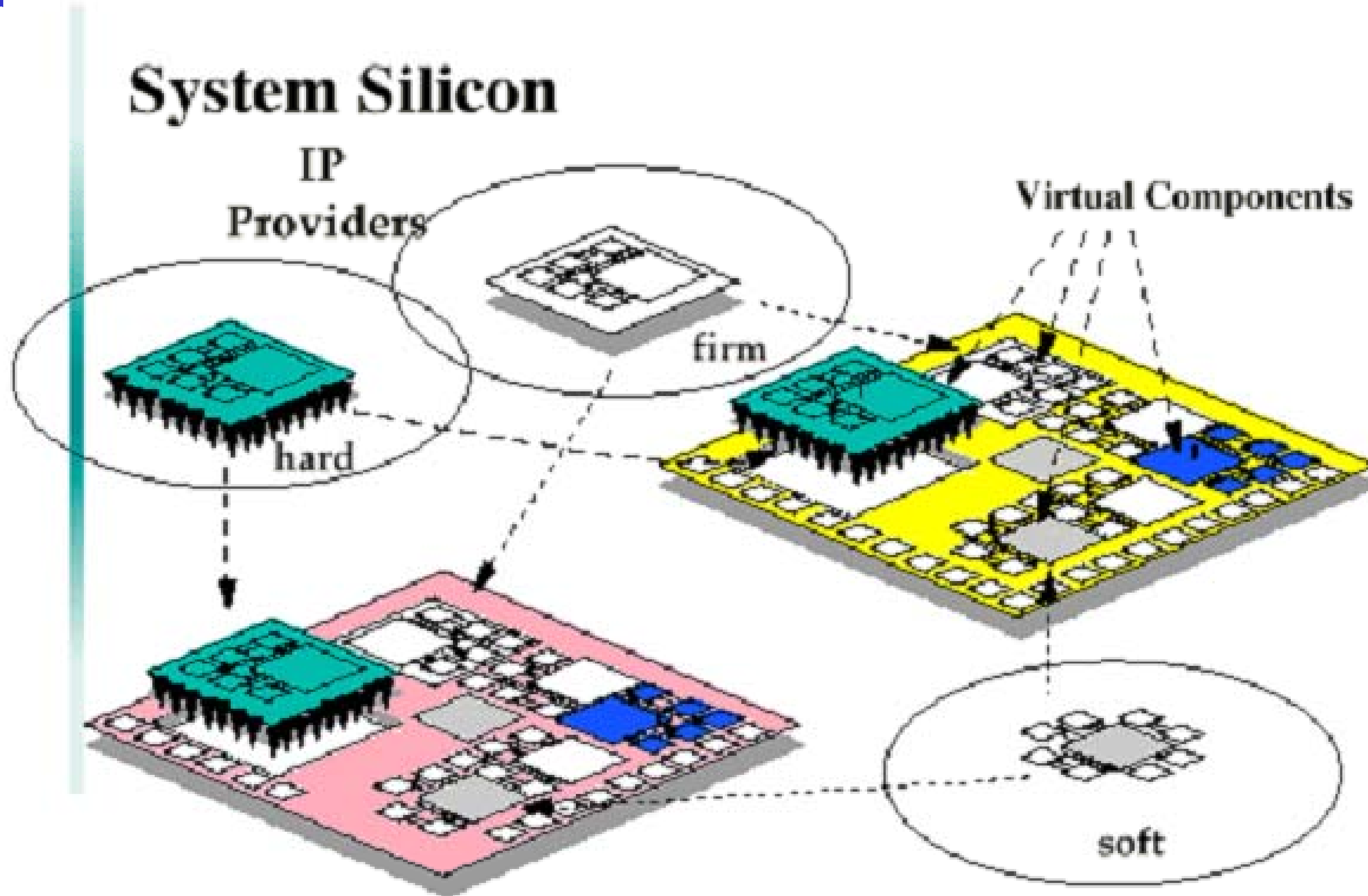
```
    wire URy, URx, S2;
    wire [1:0] S3, S4;
    wire [2:0] S1;

    Controller c (.CLK(CLK), .reset(reset),
            .ISS(ISS), .URy(URy), .URx(URx), .S2(S2),
            .OSS(OSS), .S3(S3), .S4(S4), .S1(S1));
    Data_Path dp (.CLK(CLK), .reset(reset),
            .xn(xn), .URy(URy), .URx(URx), .S2(S2),
            .S3(S3), .S4(S4), .S1(S1), .OUTPUT(OUTPUT));
endmodule
```

Conclusions

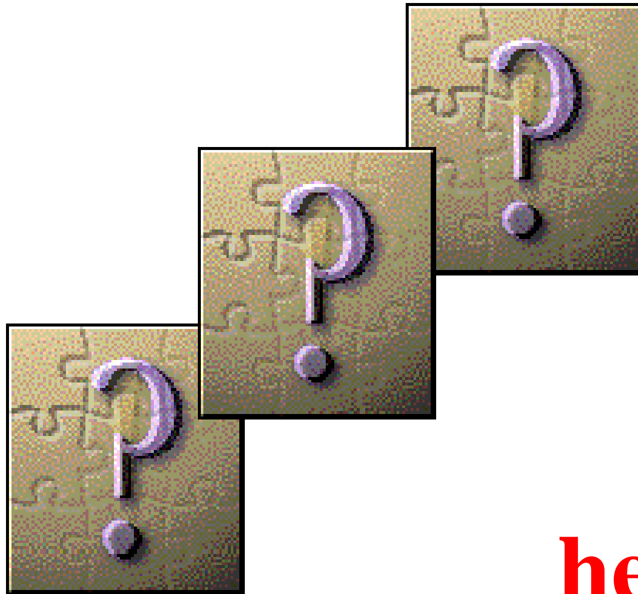


Conclusions



The College of New Jersey

Thanks……



hernande@tcnj.edu

http://www.tcnj.edu/~hernande/