



# CSCE 491 Capstone Computer Design Project

2005.8.31

## Week 2 Digital Systems Design Methods-1

© 2003 Dr. James P. Davis  
Some figures from Tanenbaum, 4<sup>th</sup> ed., ©1999 Prentice-Hall.

---

---

---

---

---

---

---

---

---

---

## Week 2 - Outline

- Digital Systems Design Methods.
  - ✓ Structural – decomposition, refinement
  - ✓ Functional – input/output response specification.
  - ✓ Behavioral/ASM – sequencing and scheduling of operations in the data path.
  - ✓ Behavioral/Timing – input/output response over some time frame.
- Clocking specification.
  - ✓ Use waveform specification as a means for defining characteristic system behavior, based on arrival of input signals and the resultant response of the system, and its internal, intermediate actions.
- ASM Diagrams.
  - ✓ We can also start with a truth table to define the equations for the outputs based on value combinations of the inputs.
  - ✓ If we include all inputs—both data and control—then the truth tables can become quite large.
  - ✓ However, what we are looking to identify from the equations is the Sum of Products (SoP) form, that can be used to identify which operations are to be scheduled in which states of the state machine (should one be required).
  - ✓ Note: we use truth tables for combinational logic function specification, but sometimes these combinational logic functions are under the sequencing control of a state machine.



© 2003 Dr. James P. Davis Page 2

---

---

---

---

---

---

---

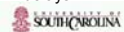
---

---

---

## Week 2 - Objective

- Objective: To gain comprehensive enough understanding of Algorithmic State Machine notation in order to use it to create digital circuit and systems architecture.
- Means:
  - ✓ Modeling the *control path* as a Finite State Machine (FSM).
  - ✓ Modeling the *data path* operations using the Register Transfer Notation (RTN).
  - ✓ Applying the underlying assumptions about timing behavior based on whether we use *registered* or *non-registered* elements in the data path. Registered elements have a one-cycle delay from when they are scheduled in the FSM and when the result appears on the output of data path registers.
  - ✓ Assuming "unit delay" in the scheduling of data path operations from the control path of the state machine.
- Result: Construct architectures easily from abstract system models described in UML, where these models are "cycle accurate", but not necessarily accurate in terms of a set of specific VLSI circuit delays.



© 2003 Dr. James P. Davis Page 3

---

---

---

---

---

---

---

---

---

---

## I. Overview:

### Review of Logic Concepts and Basic Device Functions



---

---

---

---

---

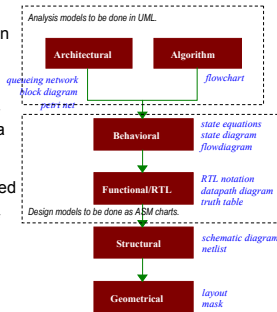
---

---

---

### Levels of Abstraction in VLSI System Design

- A design transforms from "concept" to "implementation" in a series of ordered levels.
- From the highest level to lower levels of design "abstraction", a design is iteratively refined.
- The design description is verified and validated at each level, often cycling between levels of abstraction.
- Design descriptions are described using one or more domain representations (Behavior, Structure, Physical).



---

---

---

---

---

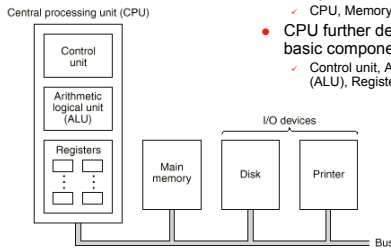
---

---

---

### Systems Design – Structure of Computers

- Basic systems components of Computer:
  - ✓ CPU, Memory, I/O Devices, Bus
- CPU further decomposes into basic components:
  - ✓ Control unit, Arithmetic Logic Unit (ALU), Registers



Source: A. Tanenbaum, 4<sup>th</sup> ed., 1999.



---

---

---

---

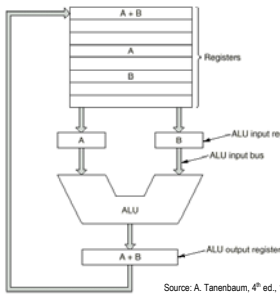
---

---

---

---

## Systems Design – Structure of Computers



Source: A. Tanenbaum, 4<sup>th</sup> ed., 1999.

- Von Neumann computer architecture:
  - ✓ Developed in the late 1940s by John Von Neumann (Princeton U.)
  - ✓ Combined the elements of *stored program machine*: one machine could run many programs (CSCE 212).
  - ✓ Program instructions stored in memory and fetched, in sequence, to be executed by CPU.
  - ✓ Results of execution stored in Registers, later written back to memory.
  - ✓ The Motorola 68000 follows this style of architecture (CSCE 313).



© 2003 Dr. James P. Davis Page 7

## Review of Boolean Logic Functions

$2^q$  functions of  $q$  Boolean variables  $\Rightarrow (q = 2 \Rightarrow |F| = 16; F = \{f_0, f_1, \dots, f_{15}\})$

$x_2$	$x_1$	$f_0(x)$	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$	$f_5(x)$	$f_6(x)$	$f_7(x)$	$f_8(x)$	$f_9(x)$	$f_{10}(x)$	$f_{11}(x)$	$f_{12}(x)$	$f_{13}(x)$	$f_{14}(x)$	$f_{15}(x)$
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	0	0	0	1	1	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	0	0	1	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

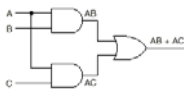
Labels for functions:  $f_0$  (AND),  $f_1$  (PASS),  $f_2$  (PASS),  $f_3$  (NOR),  $f_4$  (NOT),  $f_5$  (NOT),  $f_6$  (NAND),  $f_7$  (ANDNOT),  $f_8$  (ANDNOT),  $f_9$  (XOR),  $f_{10}$  (OR),  $f_{11}$  (XNOR),  $f_{12}$  (ORNOT),  $f_{13}$  (ORNOT),  $f_{14}$  (ORNOT),  $f_{15}$  (Tautology).



© 2003 Dr. James P. Davis Page 8

## Review of Boolean Representation

Source: Tanenbaum, 4<sup>th</sup> ed. © 1999, Prentice-Hall



A	B	C	AB	AC	AB + AC
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	1

$$f(A,B,C) = AB + AC$$

- **Boolean algebra formulation:**
  - These laws are used to transform Boolean equations into appropriate forms.
  - They were formulated in the 1930's by mathematician George Boole.
  - There is an equivalence between a Boolean expression, a Truth table and a Gate-level diagram representation. They are all used to convey different forms of the same information.
  - We start with a mathematical formulation of the desired function, and we want an efficient circuit to realize this function.
  - We apply a set of "well formed" logical operators that are mathematically "sound" and "complete".



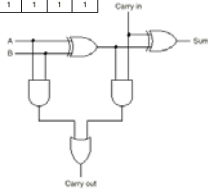
© 2003 Dr. James P. Davis Page 9



## Review of Gate-level Devices

Source: Tanenbaum, 4<sup>th</sup> ed. © 1999, Prentice-Hall

A	B	Carry in	Sum	Carry out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



© 2003 Dr. James P. Davis Page 13

- **Full Adder circuit:**
  - ✓ This circuit takes two single-bit inputs and adds them together to produce a Sum as output.
  - ✓ The Full Adder also has a Carry (called a Carry Out) like the Half Adder.
  - ✓ The Full Adder also has a Carry In signal, allowing Carry Out from earlier adder stage to be connected.
  - ✓ This allows a multi-bit, multi-stage adder circuit to be constructed.

---

---

---

---

---

---

---

---

---

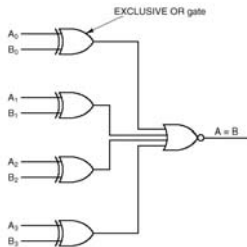
---

---

---

## Review of Gate-level Devices

Source: Tanenbaum, 4<sup>th</sup> ed. © 1999, Prentice-Hall



© 2003 Dr. James P. Davis Page 14

- **Comparator (EQ):**
  - ✓ COMP allows comparison of two different inputs to check if they are equal. Alternate circuits evaluate the relative magnitude of two inputs.
  - ✓ If they are equal, the output is HIGH, but if they are not equal, the output is LOW.
  - ✓ The comparison must be done with two signal inputs of equal width.
  - ✓ The output of the Comparator operation is a single-bit signal.

---

---

---

---

---

---

---

---

---

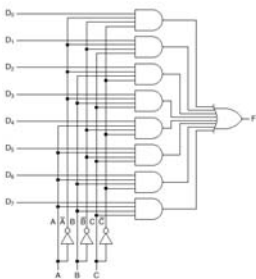
---

---

---

## Review of Gate-level Devices

Source: Tanenbaum, 4<sup>th</sup> ed. © 1999, Prentice-Hall



© 2003 Dr. James P. Davis Page 15

- **8-input Multiplexer (MUX):**
  - ✓ MUX allows the signal value of one of its data inputs (D0 – D7) to pass to the output F.
  - ✓ The selection of the signal to be passed is controlled by SELECT lines (A, B, C).
  - ✓ The number of select lines, n, is based on a power of 2 for the number of inputs, m. So, if we have m inputs, we'll need n select lines so that  $2^n = m$ .
  - ✓ The MUX inputs and output must be the same width, and the SELECT lines are 1-bit each.

---

---

---

---

---

---

---

---

---

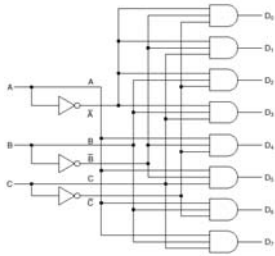
---

---

---

## Review of Gate-level Devices

Source: Tanenbaum, 4<sup>th</sup> ed © 1999, Prentice-Hall



### • 3:8 Decoder ( $n$ to $2^n$ DECO):

- ✓ DECO takes a binary encoded input of  $n$  data bits, and decodes it into individual data output lines, where one output ( $D_0 - D_7$ ) is enabled, depending on whether the encoded value corresponds to the data line number.
- ✓ A Decoder input with  $n$  lines means we can encode  $2^n$  possible binary encoded values.
- ✓ With  $2^n$  possible encoded values on the input, we'll need exactly  $n$  output lines, one for each possible encoded input value.
- ✓ The output line corresponding to the decoded value will be enabled.



© 2003 Dr. James P. Davis Page 16

---

---

---

---

---

---

---

---

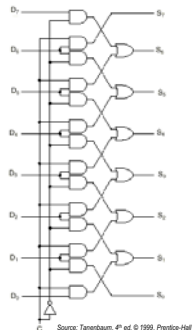
---

---

---

---

## Review of Gate-level Devices



### • 1-bit Left-Right Shifter (LSHFT, RSHFT):

- ✓ The  $n$ -bit input signal ( $D_0 - D_7$ ) is shifted either Left or Right by one bit.
- ✓ The shifted value is propagated to the output ( $S_0 - S_7$ ).
- ✓ The bit-width of the input must equal the bit-width of the output.
- ✓ If we shift Left (downward in the figure), the value on input signal  $D_0$  is lost.
- ✓ If we shift Right (upward in the figure), the value on input  $D_7$  is lost.



© 2003 Dr. James P. Davis Page 17

---

---

---

---

---

---

---

---

---

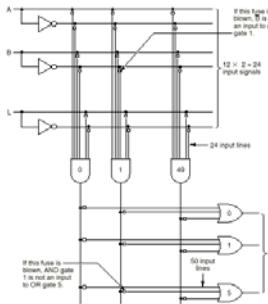
---

---

---

## Programmable Logic Devices - PLA

Source: Tanenbaum, 4<sup>th</sup> ed © 1999, Prentice-Hall



### • Programmable Logic Array (PLA):

- ✓ PLA is a structure with an AND and OR array of gates.
- ✓ The fuses allow specific connections (signal paths) to be burned so that a signal can pass through the AND or the OR part of the array.
- ✓ The PLA is used to implement custom logic functions using a standard circuit package.
- ✓ The fuses are arranged in two matrix structures: top one for AND, bottom one for OR.



© 2003 Dr. James P. Davis Page 18

---

---

---

---

---

---

---

---

---

---

---

---

## II. Overview:

# Digital Systems Modeling & Design




---

---

---

---

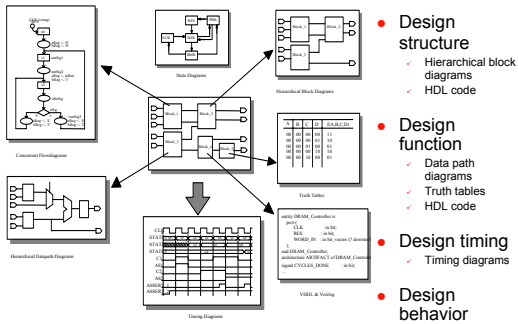
---

---

---

---

## Summary - Specifying a Digital System



- Design structure
  - ✓ Hierarchical block diagrams
  - ✓ HDL code
- Design function
  - ✓ Data path diagrams
  - ✓ Truth tables
  - ✓ HDL code
- Design timing
  - ✓ Timing diagrams
- Design behavior
  - ✓ State diagrams




---

---

---

---

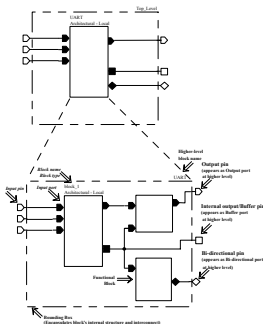
---

---

---

---

## Systems Design - Structure



- **Block diagram:** Used to partition and decompose a design into its functional units.
  - ✓ Step #1: Identify "top level" and all functional operations that transform the application's data.
  - ✓ Step #2: Group the functions by how the data is manipulated, or by what resources are needed.
  - ✓ Step #3: Separate different functions from each other by "interconnect" that shows the flow of data.
  - ✓ Step #4: Decompose more "abstract" functions into more "primitive" sets of functions that operate on data.
  - ✓ Step #5: Repeat Step #4 until all the functions are fully decomposed into a hierarchy of "primitive" units.




---

---

---

---

---

---

---

---



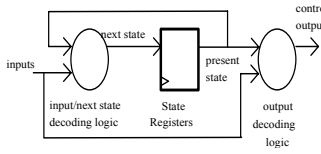






## Finite State Machine Model

### • Mealy machines:



- ✓ The *control outputs* of the state machine are dependent on the *inputs* and *present state information*.
- ✓ The *control outputs* can be asynchronous, in that outputs can change value as the *inputs* change value, provided the appropriate *present state information* is maintained.
- ✓ The *control outputs* are gated by the *present state*.
- ✓ Mealy machines are used to create control blocks that respond quickly to external signal changes.
- ✓ Care must be taken to isolate the design from transients and race conditions.



© 2003 Dr. James P. Davis Page 34

---

---

---

---

---

---

---

---

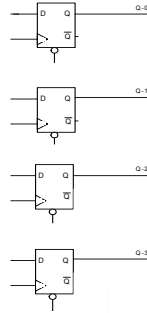
---

---

## State Machine - FSM Encoding

### • Encoding Techniques – we are interested in encoding the output lines from state registers:

- **Binary** – ascending count of binary numbers representing each state in the sequence. Control output signals require decoding (combinational logic) to generate a specific control signal.
- **Gray** – sequencing of the states so that only one bit between adjacent values changes at a time. This minimizes “glitches” since ascending Binary encoding involves transitioning through intermediate values when more than a single bit changes value.
- **One-hot** – use one D-FF for each state output line, where only a single line is active for each state. No encoding of the lines, so output control signals can be derived directly off the state lines.



© 2003 Dr. James P. Davis Page 35

---

---

---

---

---

---

---

---

---

---

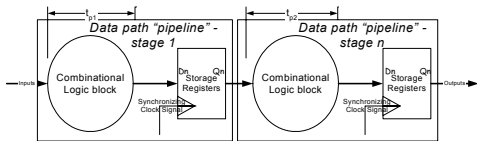
## Pipelined Datapath Model

### • Use memory elements in the data path to store signal values.

- ✓ Purpose: synchronize behavior of complex circuits.
- ✓ Benefits of circuit synchronization:
  - Eliminate unpredictability of output behavior due to timing skew.
  - Create signal stability, as they must have stable values for certain period of time (setup, hold).
  - Better isolate signals from noise transients.

### • Use of memory to create complex control structures.

- ✓ Controller sequences operations in the data path.



© 2003 Dr. James P. Davis Page 36

---

---

---

---

---

---

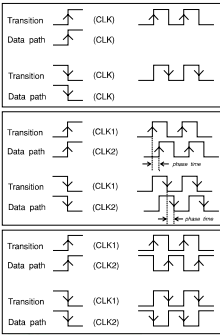
---

---

---

---

## Register Level Design – Clocking Schemes



- **Scheme 1**
  - ✓ Single-phase, rising or falling edge clocks.
- **Scheme 2**
  - ✓ Two-phase overlapping, rising or falling edge clocks.
- **Scheme 3**
  - ✓ Two-phase non-overlapping, rising and/or falling edge clocks.




---

---

---

---

---

---

---

---

---

---

## III. Lecture:

### Model-Based Design Notation: The Algorithmic State Machine




---

---

---

---

---

---

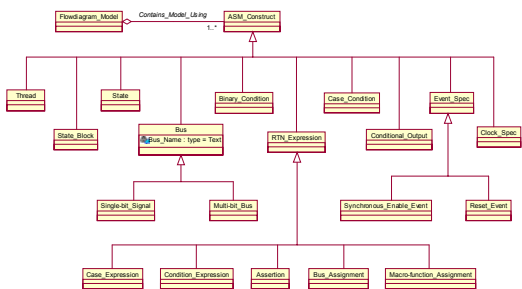
---

---

---

---

## Taxonomy of ASM Diagram Concepts




---

---

---

---

---

---

---

---

---

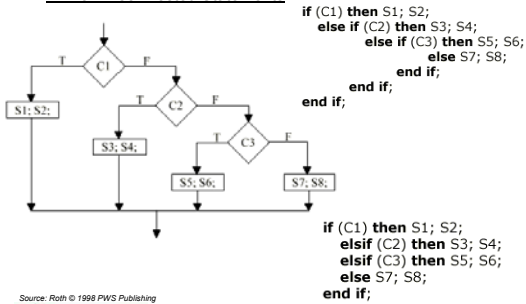
---





## ASM Diagram – Nested Control Structures

### If-Then-Else: Nested Statements



Source: Roth © 1998 PWS Publishing



© 2003 Dr. James P. Davis Page 46

---

---

---

---

---

---

---

---

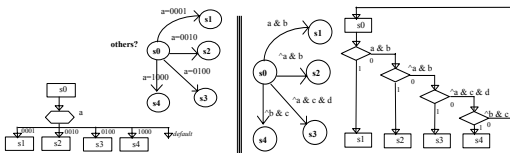
---

---

---

---

## ASM vs. State Diagrams – Branching Control Structures



### Selection: Multi-way Branching, Single Variable

ASM diagrams use case construct for multi-way branch conditions of a single, multi-bit variable/bus. Branch conditions are binary or enumerated values.

In ASM diagrams, all undefined transitions are tied to a default transition. This eliminates possible transitions to unspecified states, a common cause of design failure in the field.

State diagrams have no such mechanism, and thus are ambiguous.

### Selection: Multi-way Branching, Multi-variable

In State diagrams: (1) the ordering of transitions is ambiguous; (2) all transitions aren't specified (device problems likely in the field); (3) behavior is unpredictable under all conditions.

In ASM diagrams: (1) ordering of transitions is explicit; (2) transitions specified for all possible input combinations (including MVL values); (3) behavior is predictable.



© 2003 Dr. James P. Davis Page 47

---

---

---

---

---

---

---

---

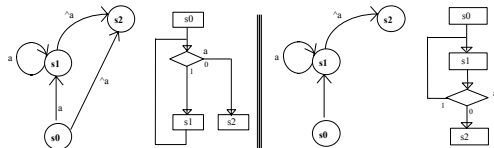
---

---

---

---

## ASM vs. State Diagrams – Loop Control Structures



### Repetition: "While-Do" Control Loop

While-Do control structure is more apparent in the ASM diagram, and is consistent with hardware description language (HDL) constructs.

Single decision point in ASM diagram handles complexity more easily when multiple terms and variables are used in looping condition expression.

This type of control construct is used often for counting the number of loop iterations, where you test the condition prior to each execution of the loop, including the first pass.



### Repetition: "Repeat-Until" Control Loop

Placement of the decision "diamond" defines the type of looping construct, and the type of control behavior.

Repetition control structures are used in various styles of polling loops for implementing handshaking protocols.

This type of control construct is used often for counting the number of loop iterations, where you test the condition after completing each execution of the loop, requiring execution of at least the first pass.

© 2003 Dr. James P. Davis Page 48

---

---

---

---

---

---

---

---

---

---

---

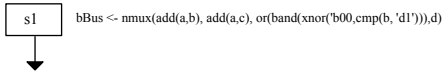
---





## ASM - Modeling Complex Macro-functions

- Create complex datapath descriptions that are efficient, in terms of gate count and delay.
  - Pure functional representation that is mathematically sound.
  - Lends itself to compact annotation without requiring operator precedence rules.
- Things to remember:
  - Macro arguments should match, in terms of bus width, signal type (binary, MVL9, etc.) and signal mode (input, output, etc.).
  - Macros should not have an output argument fed back as an input without proper buffering (with register or latch definition for signal/bus specified on LHS of expression).
  - Don't drive a signal/bus as both *registered* and *unregistered* (as this causes a *multi-drive error*, unless using MVL data types with resolution tables).




---

---

---

---

---

---

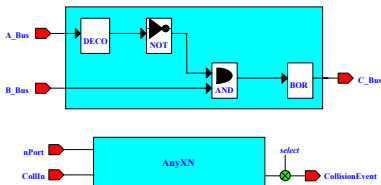
---

---

---

---

## ASM - User Defined Datapath Operations



- RTL macro-functions: approx. 30 primitive data path elements in a library.
- Macros are "scalable" - with any number of buses and any bus widths.
- Macros can be used to construct more complex user-defined data path functions.
  - ✓ **Macro-function definition:** AnyXN(A\_bus,B\_bus) ::= BOR(AND(B\_bus,NOT(DECO(A\_bus))))
  - ✓ **Macro-function binding:** CollisionEvent <- AnyXN(nPort,CollIn)




---

---

---

---

---

---

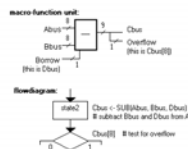
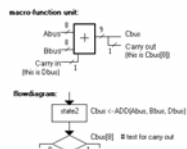
---

---

---

---

## ASM - Arithmetic Macro-functions



Carry In is optional input argument to the ADD macro. Carry Out can be accessed as the most-significant-bit (MSB) of the output bus.

ADD uses Carry In and provides Carry Out. INCR provides Carry Out. Carry Out bit is MSB of output bus Cbus if width of Cbus is equal to Abus width + 1.

Borrow is optional input argument to the SUB macro. Overflow flag can be accessed as the MSB of the output bus to which the result of the operation is assigned.

SUB uses Borrow and generates Overflow. DECR also generates Overflow. Overflow bit is MSB of output bus Cbus if width of Cbus is equal to Abus width + 1.

The ADD, INCR, SUB, DECR macros scale well up to 32 bits; beyond that, it is better to create custom arithmetic designs that scale more efficiently [Parhami, 2000].




---

---

---

---

---

---

---

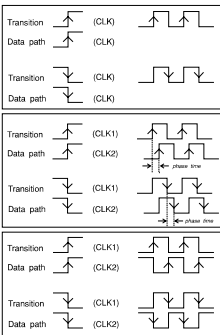
---

---

---



## Register Level Design – Clocking Schemes



- **Scheme 1**  
✓ Single-phase, rising or falling edge clocks.
- **Scheme 2**  
✓ Two-phase overlapping, rising or falling edge clocks.
- **Scheme 3**  
✓ Two-phase non-overlapping, rising and/or falling edge clocks.



© 2003 Dr. James P. Davis Page 61

---

---

---

---

---

---

---

---

---

---

## IV. Examples:

### Mapping ASM Algorithm-level models to Register-level and Gate-level representations



© 2003 Dr. James P. Davis Page 62

---

---

---

---

---

---

---

---

---

---

## Examples – Manual Analysis of ASM

- **Objective:** To understand various methods for mapping the logic of the *controller* and *datapath* blocks of a digital component.
- **Approach:** We take a couple of ASM thread examples, and discuss the following mappings:
  - ASM to register-level datapath diagram, and from data flow/datapath diagram to ASM.
  - Starting from a control flow-centric or dataflow-centric point of view with regards to your design units.
  - Obtaining decoding circuits for *next state* and *output* equations derived by inspection from the ASM threads for all FSMs.
  - ASM model mapping to concurrent state machines, including how we generate both the combinational and register logic using LUT and register structures found in Xilinx® FPGA CLB resources.



© 2003 Dr. James P. Davis Page 63

---

---

---

---

---

---

---

---

---

---

## Example 1 – ASM Controller Model

- **Structure of an ASM Thread:**

- ✓ We specify clocking and reset on initial state.

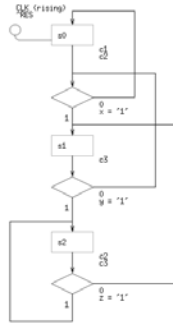
- ✓ The single-bit signals 'c1', 'c2', and 'c3' are asserted rather than assigned, which implies that we will use only a state machine, and not define any registers in the data path for RTL assignments.

- ✓ Note the state transition path from state 's1' to either 's0' or back into 's1' through the two conditions (**next state** equ.):

$$s1: (s1 \cdot (\sim(y=1) \cdot (x=1))) + (s0 \cdot (x=1)) + (s2 \cdot \sim(z=1))$$

- ✓ Note that the ^RES signal is not part of next state decoding logic. It represents the "reset" pin for the state registers, and the CLK is tied to clock pin.

Example: Lee © 2000 Prentice-Hall Publishers, Inc.



© 2003 Dr. James P. Davis Page 64

---

---

---

---

---

---

---

---

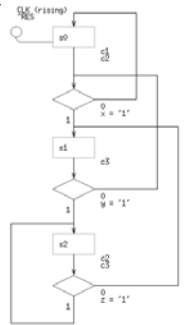
---

---

## Example 1 – ASM to FSM Model

- Derive the **next state** equations for the FSM.

- Derive the **output** equations for the FSM:



Example: Lee © 2000 Prentice-Hall Publishers, Inc.

© 2003 Dr. James P. Davis Page 65

---

---

---

---

---

---

---

---

---

---

## Example 1 – ASM to FSM Model

- Derive the **next state** equations for the FSM:

$$s1: (s1 \cdot (\sim(y=1) \cdot (x=1))) + (s0 \cdot (x=1)) + (s2 \cdot \sim(z=1))$$

$$s2: (s1 \cdot (y=1)) + (s2 \cdot (z=1))$$

$$s0: (s0 \cdot \sim(x=1)) + s1 \cdot (\sim(y=1) \cdot \sim(x=1))$$

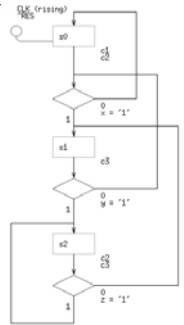
- Derive the **output** equations for the FSM:

$$c1: s0$$

$$c2: s0 + s2$$

$$c3: s1 + s2$$

(These are Moore-style outputs, dependent only on present state.)



Example: Lee © 2000 Prentice-Hall Publishers, Inc.

© 2003 Dr. James P. Davis Page 66

---

---

---

---

---

---

---

---

---

---





## Example 2 – ASM to FSM Model

- Derive the *next state* equations for the FSM:

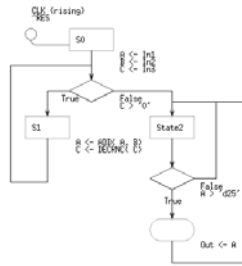
```

s0: // we only enter this state on reset.
s1: s0 ← (C=0) + s1 ← (C=0)
s2: s0 ← (C=0) + s1 ← (C=0) + s2
    // while in state s2, we circulate here.
    
```

- Derive the *output* equations for the FSM:

```

A ← In1: s0
A ← ADD: s1
B ← In2: s0
C ← In3: s0
C ← DECRNG: s1
Out ← A: s2 ← (A > 25)
    // Each output assignment represents separate
    // output equations. The first 6 are Moore
    // assignments, the last is a Mealy assignment.
    
```



Example: Lee © 2000 Prentice-Hall Publishers, Inc.  
© 2003 Dr. James P. Davis Page 73

---

---

---

---

---

---

---

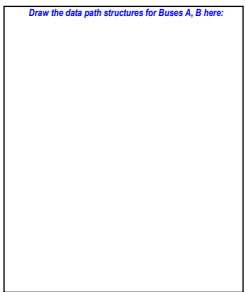
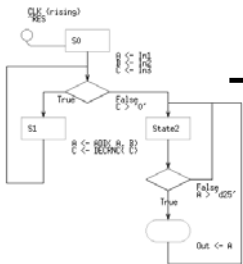
---

---

---

## Example 2 – ASM to Data Path diagram

- Given the ASM model below, what is the basic structure of the associated register-level data path circuit?
- Can you draw a RTL diagram of the data path elements represented by the expressions on the model?



Example: Lee © 2000 Prentice-Hall Publishers, Inc.

© 2003 Dr. James P. Davis Page 74

---

---

---

---

---

---

---

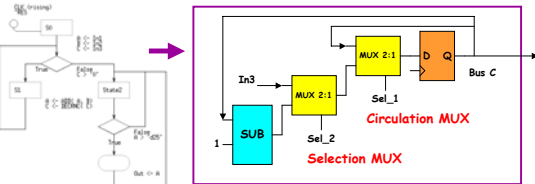
---

---

---

## Example 2 – ASM to Data Path diagram

- Using the implied Selection MUX:**
  - If you are controlling the setting of a value on a data path bus, by setting the value in more than one state, the implied logic is to control the data path operation using a MUX, whose 'select' line is derived from the FSM output decoding logic.
- Using the implied Circulation MUX:**
  - If you are controlling the setting of a bus in more than one state, but do not explicitly set this bus in all referenced states in the thread, then the implied logic is to control the value of this register by "circulating" the value back into itself while the FSM is in any state where the value is not explicitly assigned.



Example: Lee © 2000 Prentice-Hall Publishers, Inc.

© 2003 Dr. James P. Davis Page 75

---

---

---

---

---

---

---

---

---

---