Introduction to Digital Design

Week 11: Register-Transfer Level Design

Yao Zheng
Assistant Professor
University of Hawai'i at Mānoa
Department of Electrical Engineering

Overview

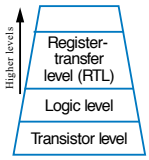
- RTL Introduction
  - Modern digital design involves creating processor-level components
  - High-level state machines
- RTL design process
  - Capture behavior: Use HLSM
  - Convert to circuit: A. Create datapath B. Connect DP to controller C. Derive controller FSM
- More RTL design
  - More components, arrays, timers, control vs. data dominated
- Determining fastest clock frequency
  - By finding critical path
- Behavioral-level design – C to gates
  - By using method to convert C (subset) to high-level state machine

2

5.1

Introduction

- Week 1 – Week 5
  - Capture Comb. behavior: Equations, truth tables
  - Convert to circuit: AND + OR + NOT → Comb. logic
- Week 7 – Week 8
  - Capture sequential behavior: FSMs
  - Convert to circuit: Register + Comb. logic → Controller
- Week 9
  - Datapath components, simple datapaths
- Week 11 – Week 12
  - Capture behavior: High-level state machine
  - Convert to circuit: Controller + Datapath → Processor
  - Known as "RTL" (register-transfer level) design



Levels of digital design abstraction

Processors:
- Programmable (microprocessor)
- Custom

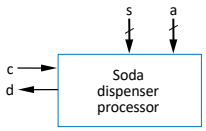
3

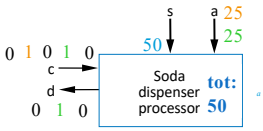
Note: Slides with animation are denoted with a small red "a" near the animated items

5.2

High-Level State Machines (HLSMs)

- Some behaviors too complex for equations, truth tables, or FSMs
- Ex: Soda dispenser
  - *c*: bit input, 1 when coin deposited
  - *a*: 8-bit input having value of deposited coin
  - *s*: 8-bit input having cost of a soda
  - *d*: bit output, processor sets to 1 when total value of deposited coins equals or exceeds cost of a soda
- FSM can't represent…
  - 8-bit input/output
  - Storage of current total
  - Addition (e.g., 25 + 10)

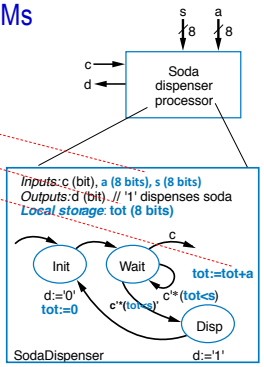




4

HLSMs

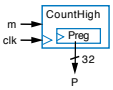
- High-level state machine (HLSM) extends FSM with:
  - Multi-bit input/output
  - Local storage
  - Arithmetic operations
- Conventions
  - Numbers:
    - Single-bit: '0' (single quotes)
    - Integer: 0 (no quotes)
    - Multi-bit: "0000" (double quotes)
  - == for equal, := for assignment
  - Multi-bit outputs *must* be registered via local storage
  - // precedes a comment

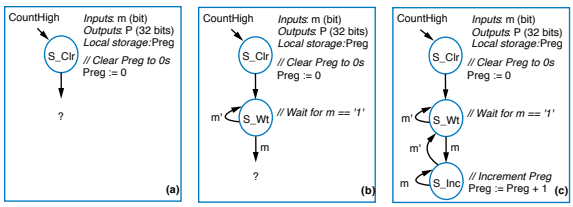


5

Ex: Cycles-High Counter

- P = total number (in binary) of cycles that m is 1
- Capture behavior as HLSM
  - Preg required (multibit outputs must be registered)
    - Use to hold count

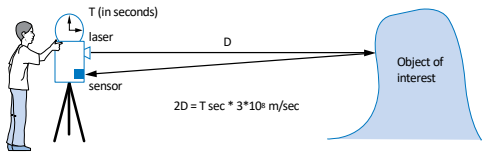




Note: Could have designed directly using an up-counter. But, that methodology is ad hoc, and won't work for more complex examples, like the next one.

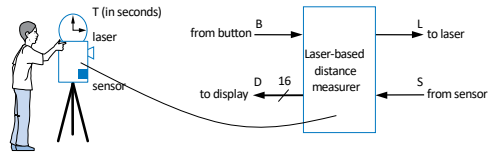
6

Example: Laser-Based Distance Measurer



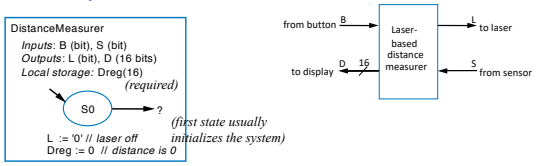
- Laser-based distance measurement – pulse laser, measure time T to sense reflection
  - Laser light travels at speed of light, $3*10^8$ m/sec
  - Distance is thus D = (T sec * $3*10^8$ m/sec) / 2

Example: Laser-Based Distance Measurer



- Inputs/outputs
  - *B*: bit input, from button, to begin measurement
  - *L*: bit output, activates laser
  - *S*: bit input, senses laser reflection
  - *D*: 16-bit output, to display computed distance

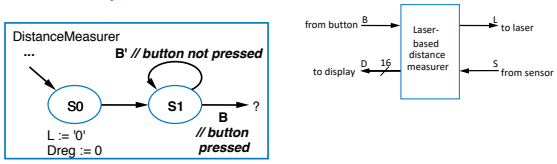
Example: Laser-Based Distance Measurer



- Declare inputs, outputs, and local storage
  - Dreg required for multi-bit output
- Create initial state, name it **S0**
  - Initialize laser to off (L:='0')
  - Initialize displayed distance to 0 (Dreg:=0)

Recall: '0' means single bit, 0 means integer

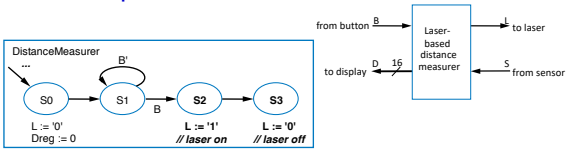
Example: Laser-Based Distance Measurer



- Add another state, **S1**, that waits for a button press
  - B' – stay in **S1**, keep waiting
  - B – go to a new state **S2**

Q: What should S2 do? A: Turn on the laser

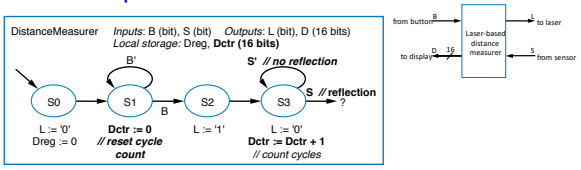
Example: Laser-Based Distance Measurer



- Add a state **S2** that turns on the laser (L:='1')
- Then turn off laser (L:='0') in a state **S3**

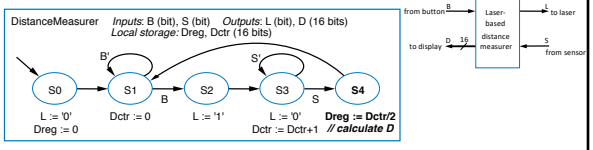
Q: What do next? A: Start timer, wait to sense reflection

Example: Laser-Based Distance Measurer



- Stay in **S3** until sense reflection (S)
- To measure time, count cycles while in **S3**
  - To count, declare local storage *Dctr*
  - Initialize *Dctr* to 0 in **S1**. In **S2** would have been O.K. too.
    - Don't forget to initialize local storage—common mistake
  - Increment *Dctr* each cycle in **S3**

Example: Laser-Based Distance Measurer

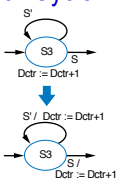


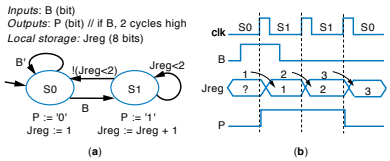
- Once reflection detected (S), go to new state **S4**
  - Calculate distance
  - Assuming clock frequency is $3x10^8$, *Dctr* holds number of meters, so Dreg:=Dctr/2
- After **S4**, go back to **S1** to wait for button again

13

HLSM Actions: Updates Occur Next Clock Cycle

- Local storage updated on clock edges only
  - Enter state on clock edge
  - Storage writes in that state occur on *next* clock edge
  - Can think of as occurring on outgoing transitions
- ***Thus***, transition conditions use the OLD value, not the newly-written value
  - Example:



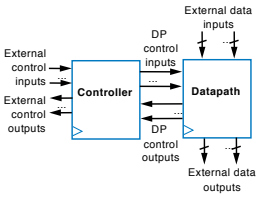


14

5.3

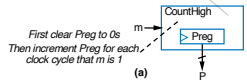
RTL Design Process

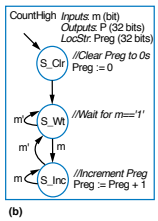
- Capture behavior
- Convert to circuit
  - Need target architecture
  - Datapath capable of HLSM's data operations
  - Controller to control datapath



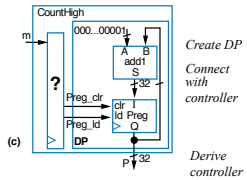
15

Ctrl/DP Example for Earlier Cycles-High Counter





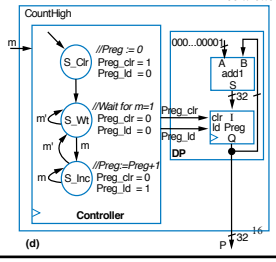
We created this HLSM earlier



Create DP

Connect with controller

Derive controller



16

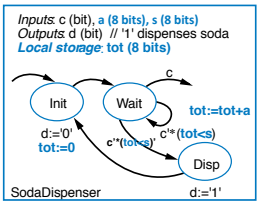
RTL Design Process

		Step	Description
Step 1: Capture behavior		*Capture a high-level state machine*	Describe the system's desired behavior as a high-level state machine. The state machine consists of states and transitions. The state machine is "high-level" because the transition conditions and the state actions are more than just Boolean operations on single-bit inputs and outputs.
Step 2: Convert to circuit	**2A**	*Create a datapath*	Create a datapath to carry out the data operations of the high-level state machine.
	2B	*Connect the datapath to a controller*	Connect the datapath to a controller block. Connect external control inputs and outputs to the controller block.
	2C	*Derive the controller's FSM*	Convert the high-level state machine to a finite-state machine (FSM) for the controller, by replacing data operations with setting and reading of control signals to and from the datapath.

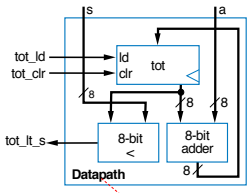
17

Example: Soda Dispenser from Earlier

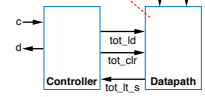
- Quick overview example. More details of each step to come.



Step 1



Step 2A

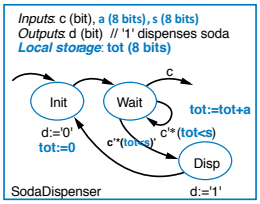


Step 2B

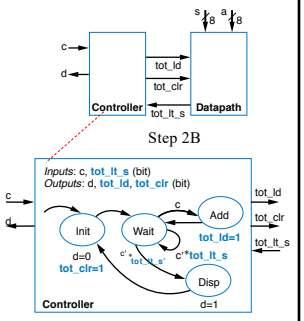
18

Example: Soda Dispenser

- Quick overview example.
  More details of each step to come.



Step 1



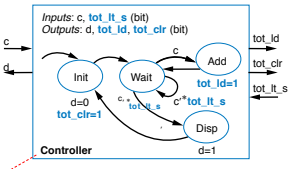
Step 2B

Step 2C

Example: Soda Dispenser

- Quick overview example.
  More details of each step to come.

	s1	s0	c	d	tot_lt_s	n1	n0	d	tot_ld	tot_clr
Init	0	0	0	0		0	1	0	0	1
	0	0	0	1		0	1	0	0	1
	0	0	1	0		0	1	0	0	1
	0	0	1	1		0	1	0	0	1
Wait	0	1	0	0		1	1	0	0	0
	0	1	0	1		0	1	0	0	0
	0	1	1	0		1	0	0	0	0
	0	1	1	1		1	0	0	0	0
Add	1	0	0	0		0	1	0	1	0
			• • •					• • •		
Disp	1	1	0	0		0	0	1	0	0
			• • •					• • •		

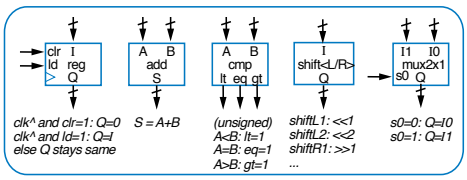


Step 2C

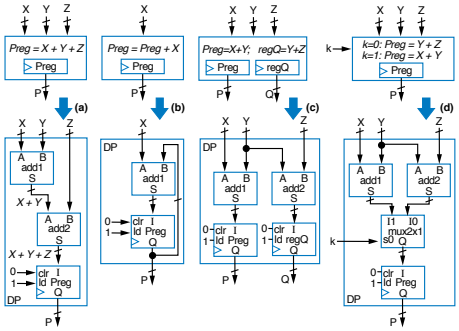
Use controller design process (Ch3) to complete the design

RTL Design Process—Step 2A: Create a datapath

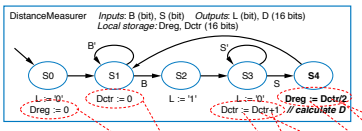
- Sub-steps
  - HLSM data inputs/outputs → Datapath inputs/outputs.
  - HLSM local storage item → Instantiated register
    - "Instantiate": Add new component ("instance") to design
  - Each HLSM state action and transition condition data computation → Datapath components and connections
    - Also instantiate multiplexors as needed
- Need component library from which to choose



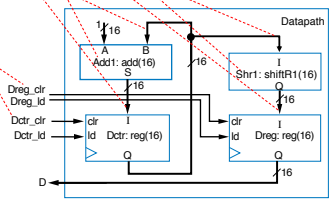
Step 2A: Create a Datapath—Simple Examples



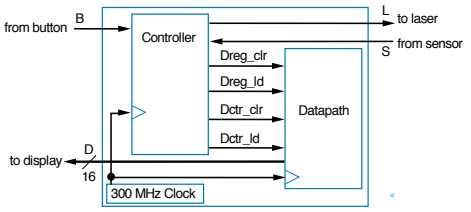
Laser-Based Distance Measurer—Step 2A: Create a Datapath



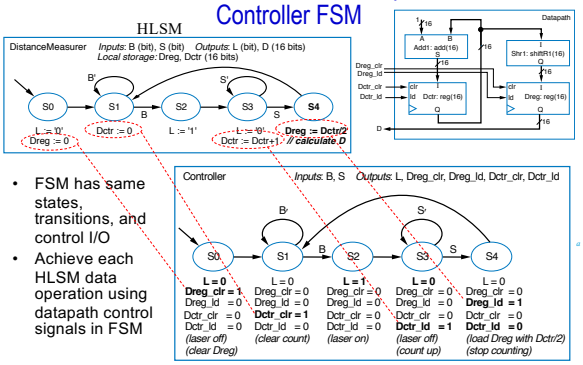
- HLSM data I/O → DP I/O
- HLSM local storage → reg
- HLSM state action and transition condition data computation → Datapath components and connections



Laser-Based Distance Measurer—Step 2B: Connecting the Datapath to a Controller



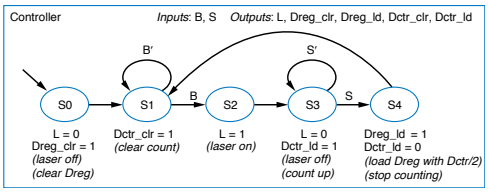
Laser-Based Distance Measurer—Step 2C: Derive the Controller FSM



- FSM has same states, transitions, and control I/O
- Achieve each HLSM data operation using datapath control signals in FSM

25

Laser-Based Distance Measurer—Step 2C: Derive the Controller FSM



- Same FSM, using convention of unassigned outputs implicitly assigned 0

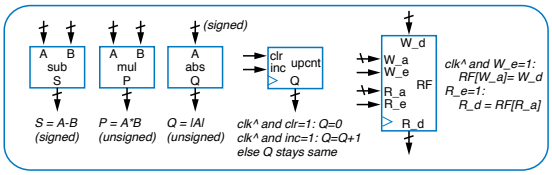
Some assignments to 0 still shown, due to their importance in understanding desired controller behavior

26

5.4

More RTL Design

- Additional datapath components



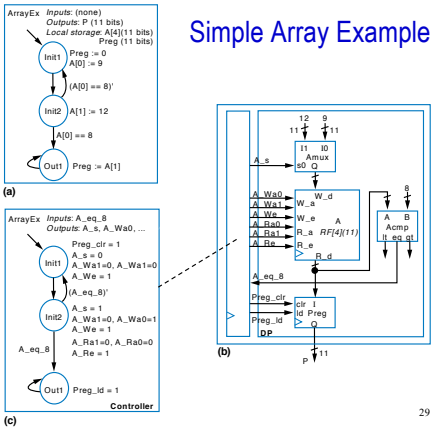
27

RTL Design Involving Register File or Memory

- HLSM *array*: Ordered list of items
  - Ex: Local storage: A[4](8-bit) – 4 8-bit items
  - Accessed using notation "A[i]", i is *index*
  - A[0] := 9; A[1] := 8; A[2] := 7; A[3] := 22
    - Array contents now: <9, 8, 7, 22>
    - X := A[1] will set X to 8
    - Note: First element's index is 0
- Array can be mapped to instantiated register file or memory

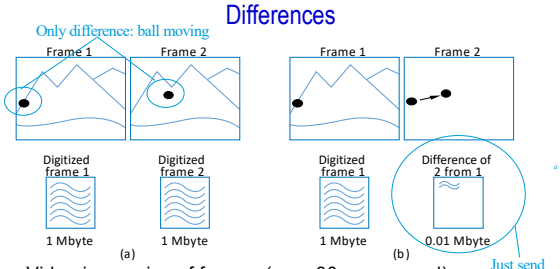
28

Simple Array Example



29

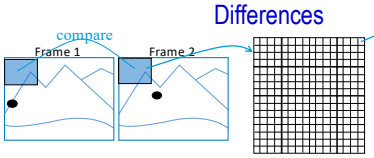
RTL Example: Video Compression – Sum of Absolute Differences



- Video is a series of frames (e.g., 30 per second)
- Most frames similar to previous frame
  - Compression idea: just send difference from previous frame

30

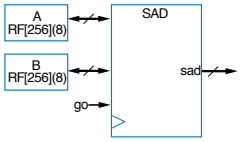
RTL Example: Video Compression – Sum of Absolute Differences



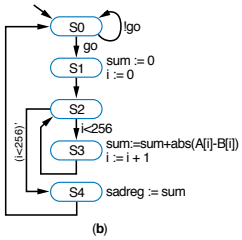
Each is a pixel, assume represented as 1 byte (actually, a color picture might have 3 bytes per pixel, for intensity of red, green, and blue components of pixel)

- Need to quickly determine whether two frames are similar enough to just send difference for second frame
  - Compare corresponding 16x16 "blocks"
    - Treat 16x16 block as 256-byte array
  - Compute the absolute value of the difference of each array item
  - Sum those differences – if above a threshold, send complete frame for second frame; if below, can use difference method (using another technique, not described)

Array Example: Video Compression—Sum-of-Absolute Differences

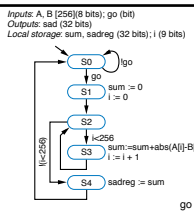


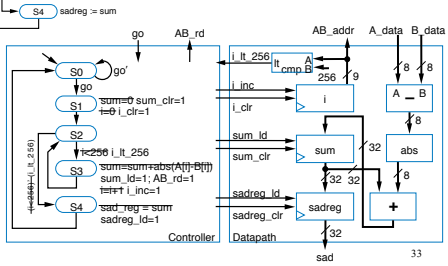
Inputs: A, B [256](8 bits); go (bit)
Outputs: sad (32 bits)
Local storage: sum, sadreg (32 bits); i (9 bits)



- **S0**: wait for *go*
- **S1**: initialize *sum* and *index*
- **S2**: check if done (*(i<256)'*)
- **S3**: add difference to *sum*, increment index
- **S4**: done, write to output *sad_reg*

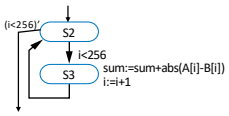
Array Example: Video Compression—Sum-of-Absolute Differences





Circuit vs. Microprocessor

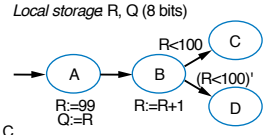
- Circuit: Two states (**S2** & **S3**) for each *i*, 256 *i*'s→ 512 clock cycles
- Microprocessor: Loop (*for i = 1 to 256*), but for each *i*, must move memory to local registers, subtract, compute absolute value, add to sum, increment *i* – say 6 cycles per array item → 256*6 = 1536 cycles
- Circuit is about *3 times* (300%) faster (assuming equal cycle lengths)
- Later, we'll see how to build SAD circuit that is *much* faster

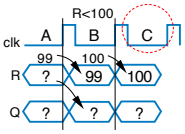


Common RTL Design Pitfall Involving Storage Updates

- Questions
  - Value of Q after state A?
  - Final state is C or D?
- Answers
  - Q is NOT 99 after state A
  - Q is 99 in state B, so final state is C
  - Storage update actions in state occur *simultaneously* on *next* clock edge
    - Thus, order actions are written is irrelevant
    - A's actions same if:
      - Q:=R R:=99 or
      - R:=99 Q:=R

Local storage R, Q (8 bits)

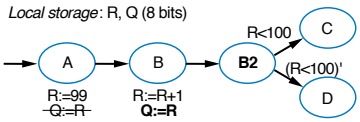


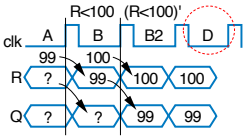


Common RTL Design Pitfall Involving Storage Updates

- New HLSM using extra state so read of R occurs after write of R

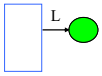
Local storage: R, Q (8 bits)

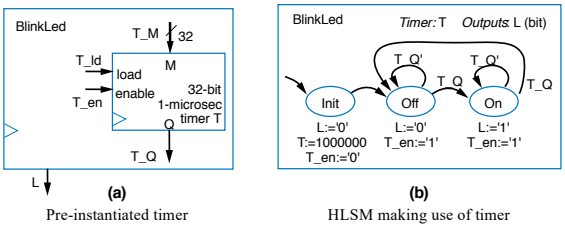




RTL Design Involving a Timer

- Commonly need explicit time intervals
  - Ex: Repeatedly blink LED on 1 second, off 1 second
- Pre-instantiate timer that HLSM can then use



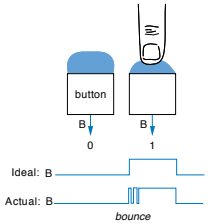


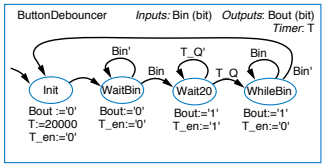
Pre-instantiated timer

HLSM making use of timer

Button Debouncing

- Press button
  - Ideally, output changes to 1
  - Actually, output bounces
    - Due to mechanical reasons
    - Like ball bouncing when dropped to floor
- Digital circuit can convert actual signal closer to ideal signal





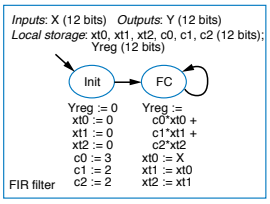
Data Dominated RTL Design Example

- Data dominated design: Extensive DP, simple controller
- Control dominated design: Complex controller, simple DP
- Example: Filter
  - Converts digital input stream to new digital output stream
  - Ex: Remove noise
    - 180, 180, 181, 180, 240, 180, 181
    - 240 is probably noise, filter might replace by 181
  - Simple filter: Output average of last *N* values
    - Small *N*: less filtering
    - Large *N*: more filtering, but less sharp output

Data Dominated RTL Design Example: FIR Filter

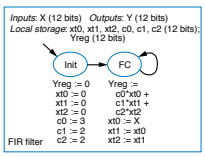
- FIR filter
  - "Finite Impulse Response"
  - Simply a configurable weighted sum of past input values
  - y(t) = c0*x(t) + c1*x(t-1) + c2*x(t-2)
    - Above known as "3 tap"
    - Tens of taps more common
    - Very general filter – User sets the constants (c0, c1, c2) to define specific filter
- RTL design
  - Step 1: Create HLSM
    - Very simple states/transitions

y(t) = c0*x(t) + c1*x(t-1) + c2*x(t-2)

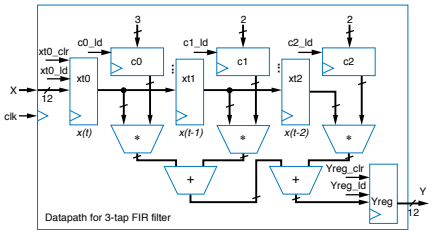


Assume constants set to 3, 2, and 2

FIR Filter



- Step 2A: Create datapath
- Step 2B: Connect Ctrlr/DP (as earlier examples)
- Step 2C: Derive FSM
  - Set clr and ld lines appropriately



Circuit vs. Microprocessor

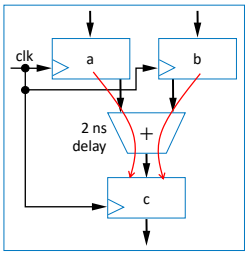
y(t) = c0*x(t) + c1*x(t-1) + c2*x(t-2)

- Comparing the FIR circuit to microprocessor instructions
  - Microprocessor
    - 100-tap filter: 100 multiplications, 100 additions. Say 2 instructions per multiplication, 2 per addition. Say 10 ns per instruction.
    - (100*2 + 100*2)*10 = 4000 ns
  - Circuit
    - Assume adder has 2 ns delay, multiplier has 20 ns delay
    - Longest path goes through one multiplier and two adders
      - 20 + 2 + 2 = 24 ns delay
    - 100-tap filter, following design on previous slide, would have about a 34 ns delay: 1 multiplier and 7 adders on longest path
  - Circuit is more than 100 times faster (4000/34). Wow.

5.5

Determining Clock Frequency

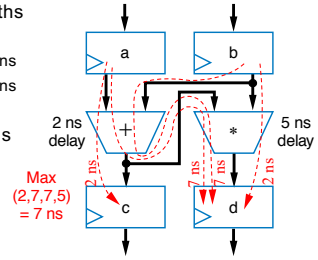
- Designers of digital circuits often want fastest performance
  - Means want high clock frequency
- Frequency limited by ***longest register-to-register delay***
  - Known as ***critical path***
  - If clock is any faster, incorrect data may be stored into register
  - Longest path on right is 2 ns
    - Ignoring wire delays, and register setup and hold times, for simplicity



43

Critical Path

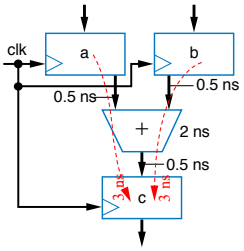
- Example shows four paths
  - a to c through +: 2 ns
  - a to d through + and *: 7 ns
  - b to d through + and *: 7 ns
  - b to d through *: 5 ns
- Longest path is thus 7 ns
- Fastest frequency
  - 1 / 7 ns = 142 MHz



44

Critical Path Considering Wire Delays

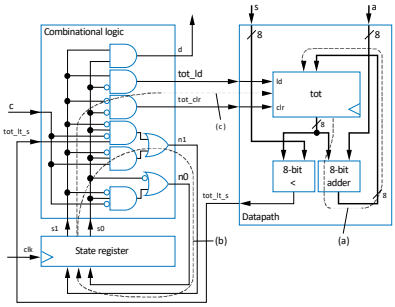
- Real wires have delay too
  - Must include in critical path
- Example shows two paths
  - Each is 0.5 + 2 + 0.5 = 3 ns
- Trend
  - 1980s/1990s: Wire delays were tiny compared to logic delays
  - But wire delays not shrinking as fast as logic delays
    - Wire delays may even be greater than logic delays!
- Must also consider register setup and hold times, also add to path
- Then add some time to the computed path, just to be safe
  - e.g., if path is 3 ns, say 4 ns instead



45

A Circuit May Have Numerous Paths

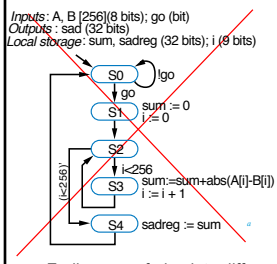
- Paths can exist
  - In the datapath
  - In the controller
  - Between the controller and datapath
  - May be hundreds or thousands of paths
- Timing analysis tools that evaluate all possible paths automatically very helpful



46

5.6

Behavioral Level Design: C to Gates



C code

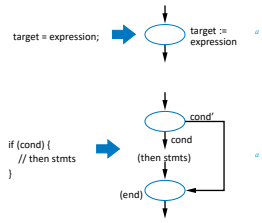
```
int SAD (byte A[256], byte B[256]) // not quite C syntax
{
    uint sum; short uint I;
    sum = 0;
    i = 0;
    while (i < 256) {
        sum = sum + abs(A[i] – B[i]);
        i = i + 1;
    }
    return sum;
}
```

- Earlier sum-of-absolute-differences example
  - Started with high-level state machine
  - C code is an even better starting point -- easier to understand

47

Converting from C to High-Level State Machine

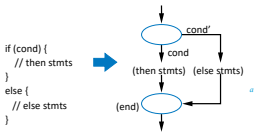
- Convert each C construct to equivalent states and transitions
- *Assignment* statement
  - Becomes one state with assignment
- *If-then* statement
  - Becomes state with condition check, transitioning to "then" statements if condition true, otherwise to ending state
    - "then" statements would also be converted to states



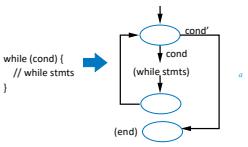
48

Converting from C to High-Level State Machine

- *If-then-else*
  - Becomes state with condition check, transitioning to "then" statements if condition true, or to "else" statements if condition false

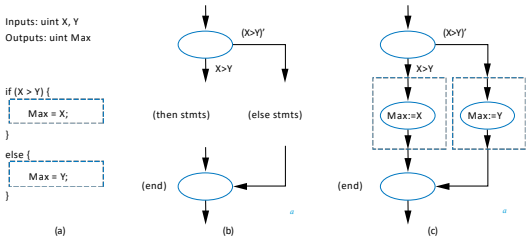


- *While loop* statement
  - Becomes state with condition check, transitioning to while loop's statements if true, then transitioning back to condition check



49

Simple Example of Converting from C to High-Level State Machine

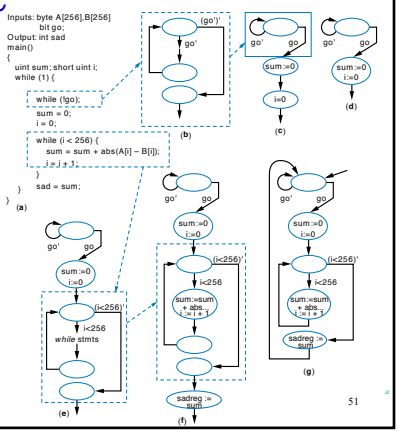


- Simple example: Computing the maximum of two numbers
  - Convert if-then-else statement to states (b)
  - Then convert assignment statements to states (c)

50

Example: SAD C code to HLSM

- Convert each construct to states
  - Simplify, e.g., merge states
- RTL design process to convert to circuit
- Can thus convert C to circuit using straightforward process
  - Actually, subset of C (not all C constructs easily convertible)
  - Can use language other than C



51

Summary

- RTL Introduction
  - Modern digital design involves creating processor-level components
  - High-level state machines
- RTL design process
  - Capture behavior: Use HLSM
  - Convert to circuit: A. Create datapath B. Connect DP to controller C. Derive controller FSM
- More RTL design
  - More components, arrays, timers, control vs. data dominated
- Determining fastest clock frequency
  - By finding critical path
- Behavioral-level design – C to gates
  - By using method to convert C (subset) to high-level state machine

52