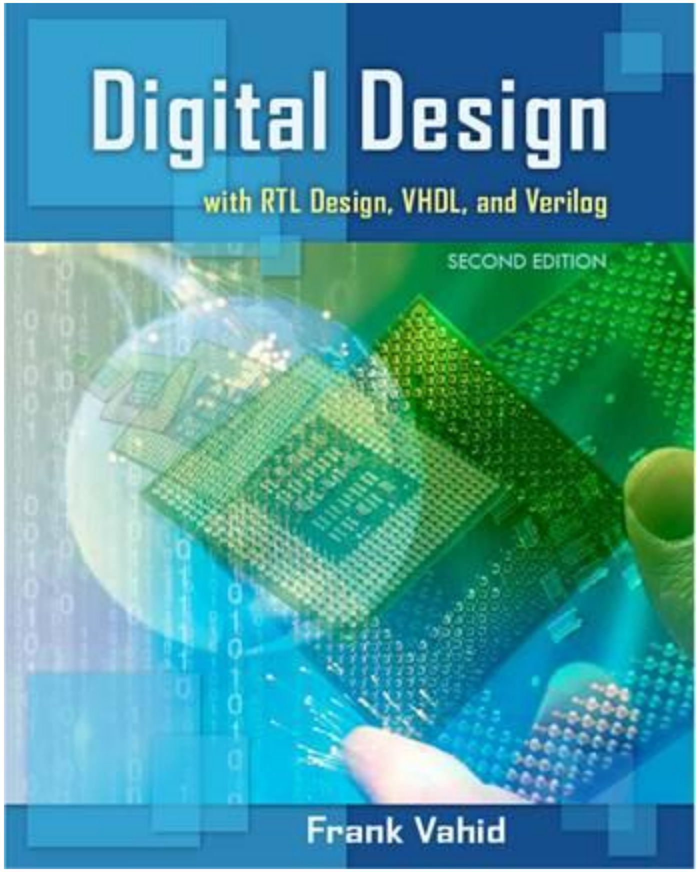


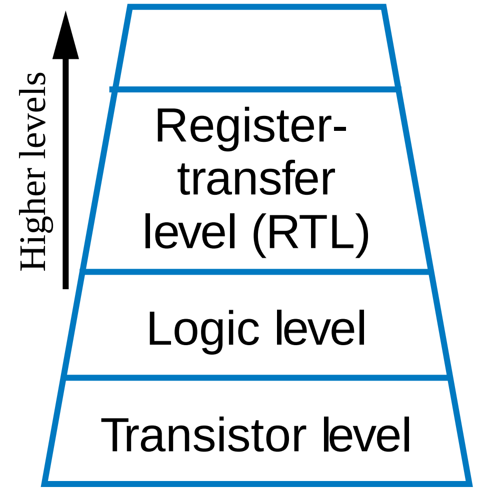

# Digital Design

## Chapter 5:
## Register-Transfer Level (RTL) Design

Slides to accompany the textbook *Digital Design, with RTL Design, VHDL, and Verilog*, 2nd Edition,
by Frank Vahid, John Wiley and Sons Publishers, 2010.
http://www.ddvahid.com

# Introduction

- Chpt 2
  - Capture Comb. behavior: Equations, truth tables
  - Convert to circuit: AND + OR + NOT → Comb. logic
- Chpt 3
  - Capture sequential behavior: FSMs
  - Convert to circuit: Register + Comb. logic → Controller
- Chpt 4
  - Datapath components, simple datapaths
- Chpt 5
  - 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

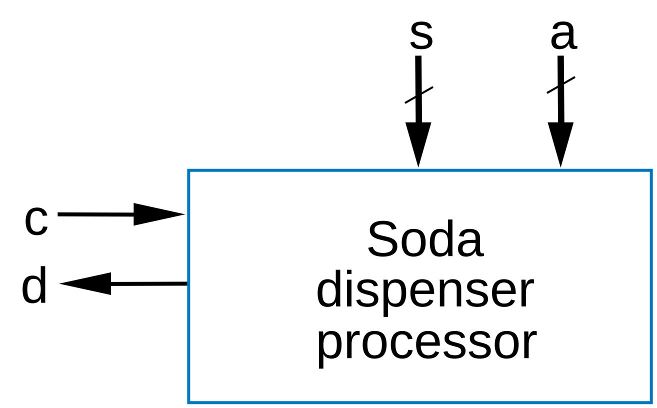


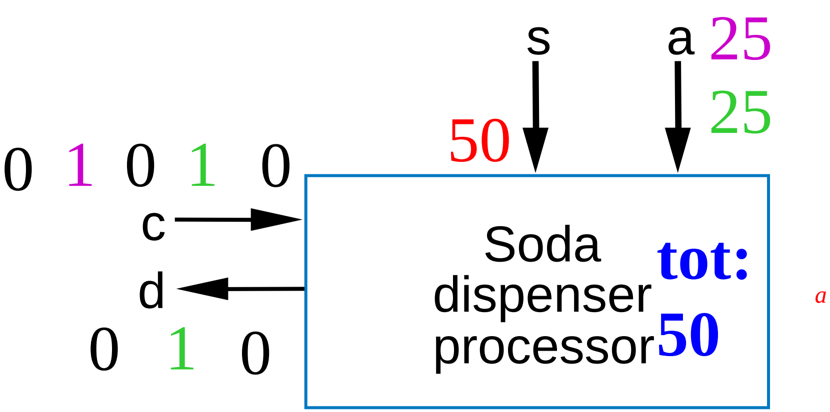


Note: Slides with animation are denoted with a small red "a" near the animated items

# High-Level State Machines (HLSMs)

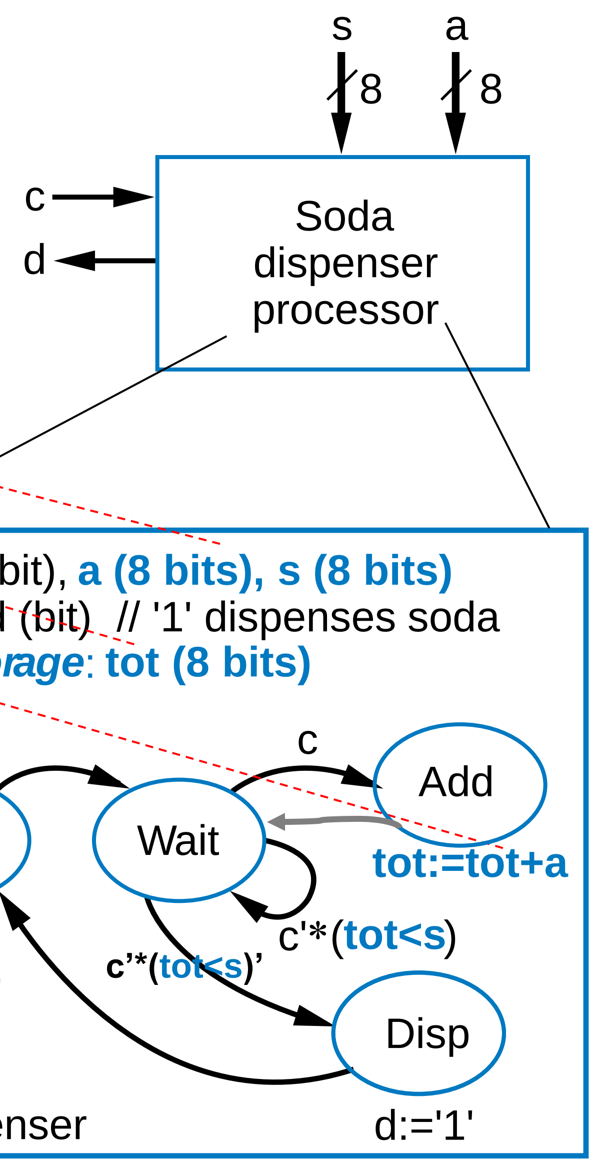
- Some behaviors too complex for equations, truth tables, or FSMs
- Ex: Soda dispenser
  - *c*: bit input, 1 when coin deposited
  - *a*: 8-bit input having value of deposited coin
  - *s*: 8-bit input having cost of a soda
  - *d*: bit output, processor sets to 1 when total value of deposited coins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



# HLSMs

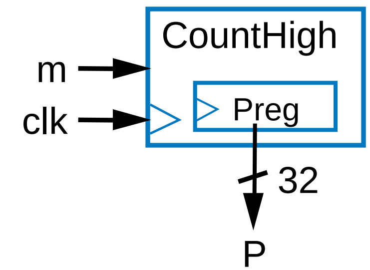
- High-level state machine (HLSM) extends FSM with:
  - Multi-bit input/output
  - Local storage
  - Arithmetic operations

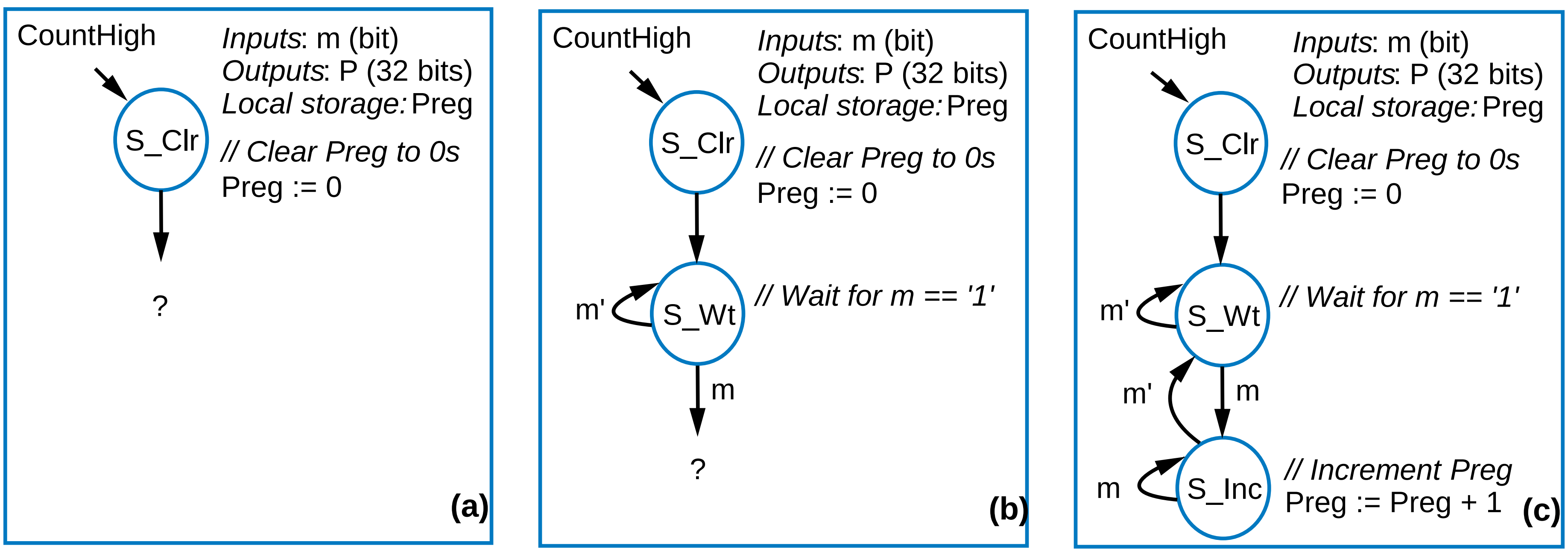



- Conventions
  - Numbers:
    - Single-bit: '0' (single quotes)
    - Integer: 0 (no quotes)
    - Multi-bit: “0000” (double quotes)
  - == for equal, := for assignment
  - Multi-bit outputs *must* be registered via local storage
  - // precedes a comment

# Ex: Cycles-High Counter

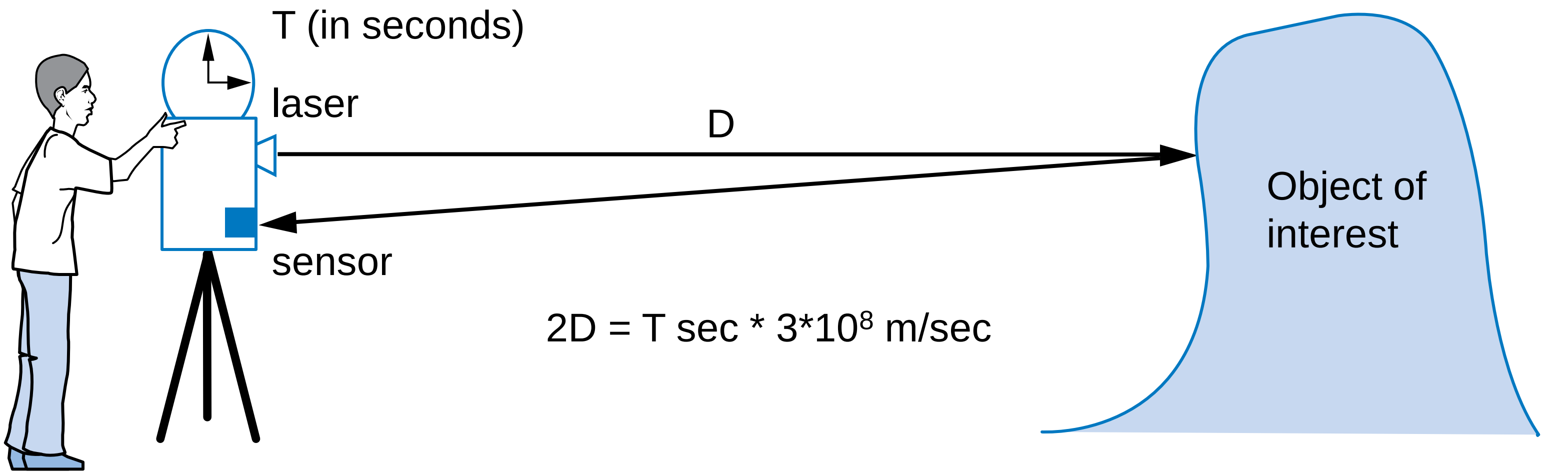
- P = total number (in binary) of cycles that m is 1
- Capture behavior as HLSM
  - Preg required (multibit outputs must be registered)
    - Use to hold count




*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.*

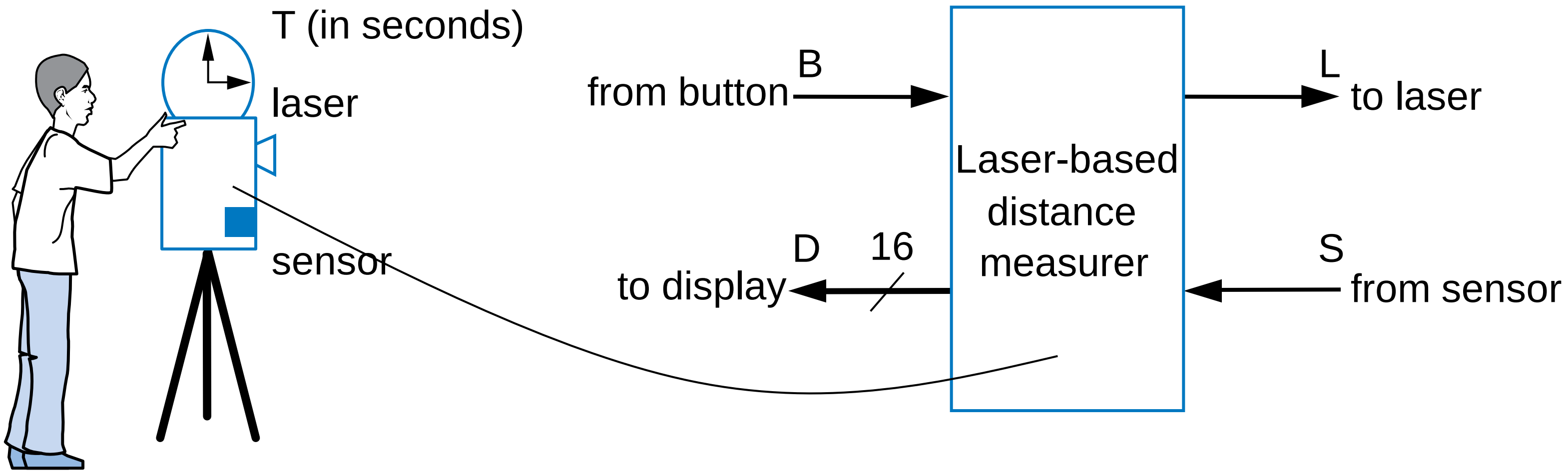
# Example: Laser-Based Distance Measurer





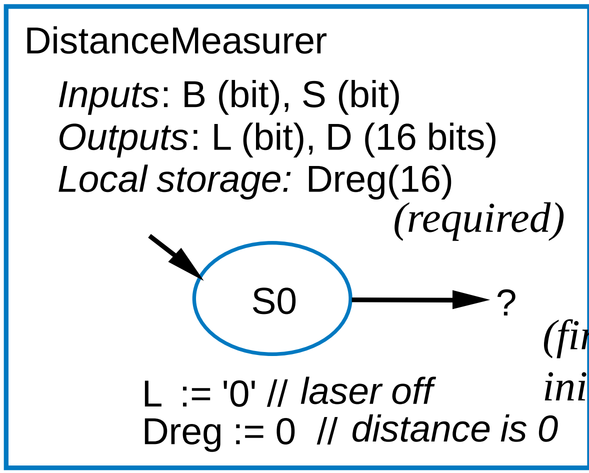

- Laser-based distance measurement – pulse laser, measure time T to sense reflection
  - Laser light travels at speed of light, $3*10^8$ m/sec
  - Distance is thus $D = (T \text{ sec} * 3*10^8 \text{ m/sec}) / 2$

# Example: Laser-Based Distance Measurer

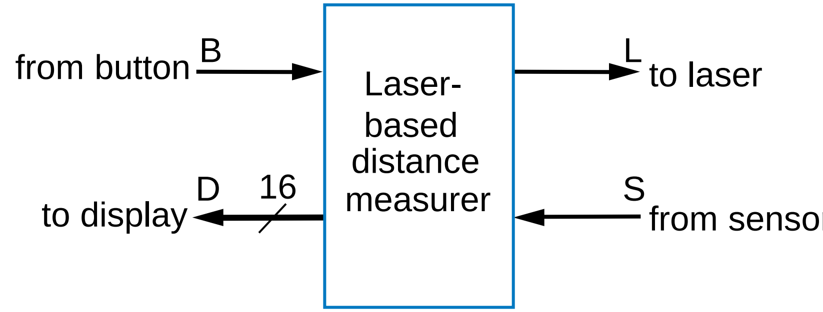



- Inputs/outputs
  - *B*: bit input, from button, to begin measurement
  - *L*: bit output, activates laser
  - *S*: bit input, senses laser reflection
  - *D*: 16-bit output, to display computed distance

# Example: Laser-Based Distance Measurer



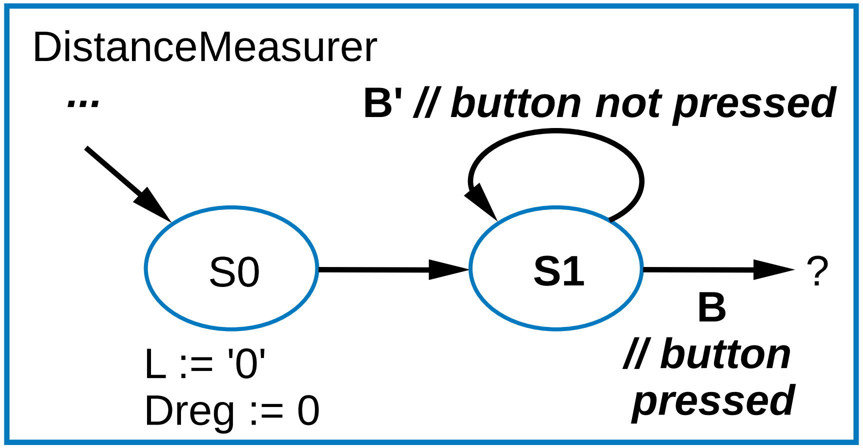

*(first state usually initializes the system)*

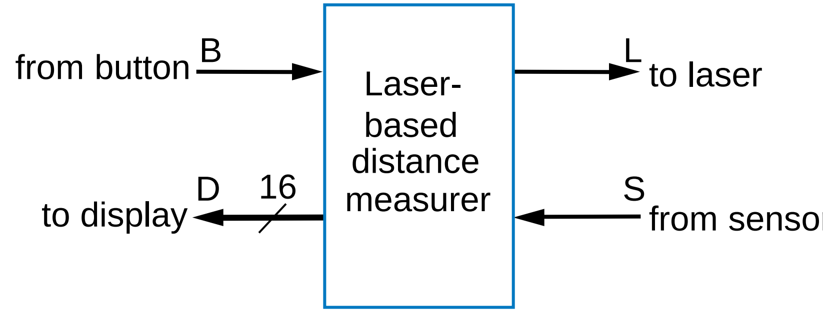



- Declare inputs, outputs, and local storage
  - Dreg required for multi-bit output
- Create initial state, name it **S0**
  - Initialize laser to off (L:='0')
  - Initialize displayed distance to 0 (Dreg:=0)

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

# Example: Laser-Based Distance Measurer



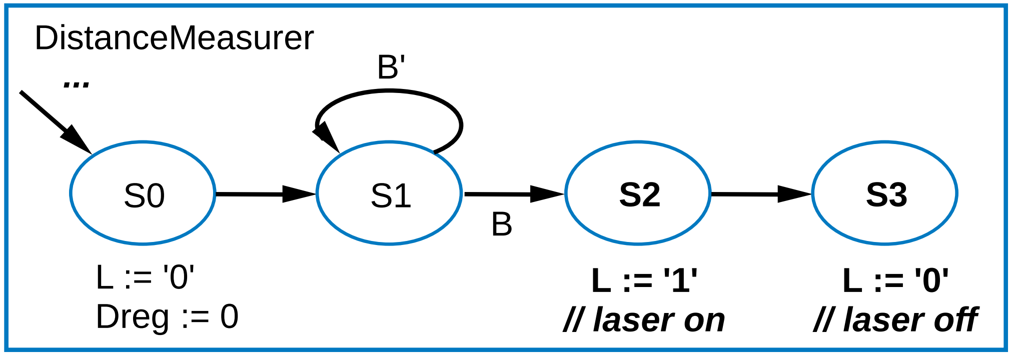



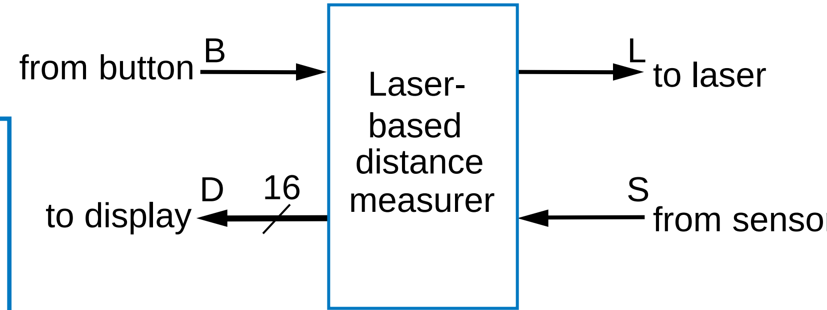

- Add another state, **S1**, that waits for a button press
  - B' – stay in **S1**, keep waiting
  - B – go to a new state **S2**

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

*a*

# Example: Laser-Based Distance Measurer



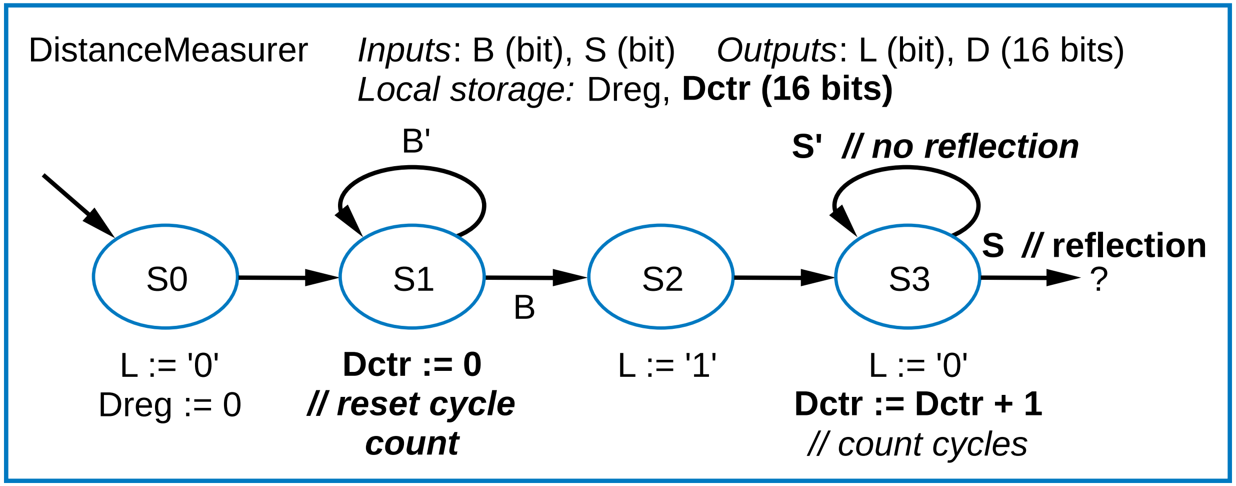



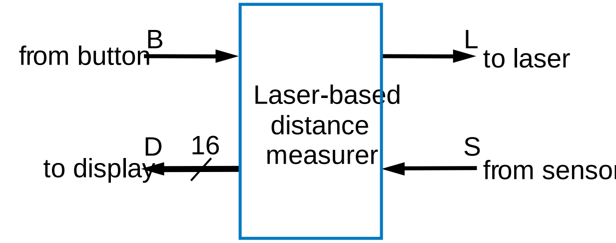

- Add a state **S2** that turns on the laser (L:='1')
- Then turn off laser (L:='0') in a state **S3**

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

*a*

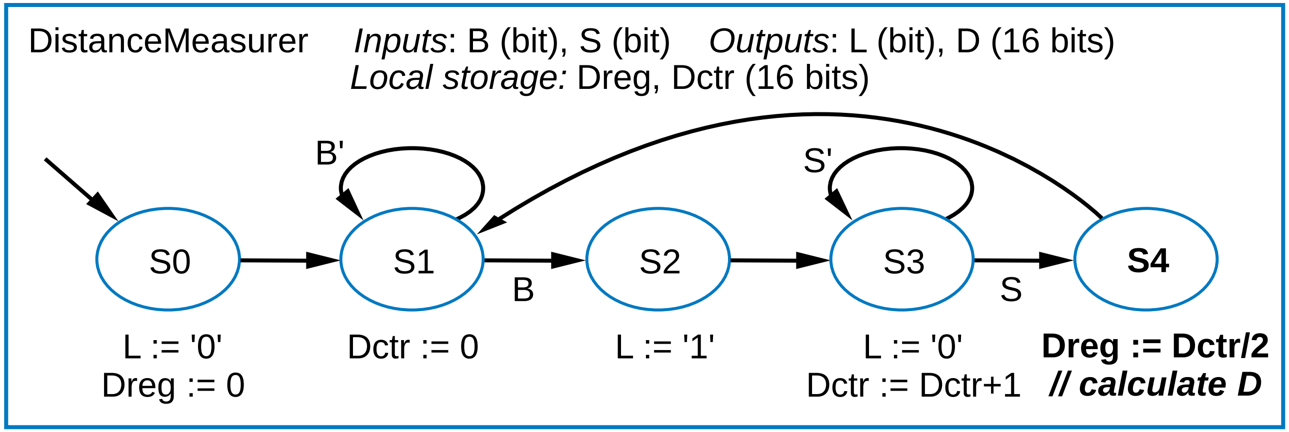
# Example: Laser-Based Distance Measurer

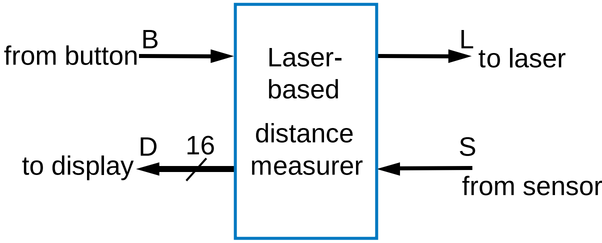



- Stay in **S3** until sense reflection (S)
- To measure time, count cycles while in **S3**
  - To count, declare local storage *Dctr*
  - Initialize *Dctr* to 0 in **S1**. In **S2** would have been O.K. too.
    - Don't forget to initialize local storage—common mistake
  - Increment *Dctr* each cycle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

# Example: Laser-Based Distance Measurer



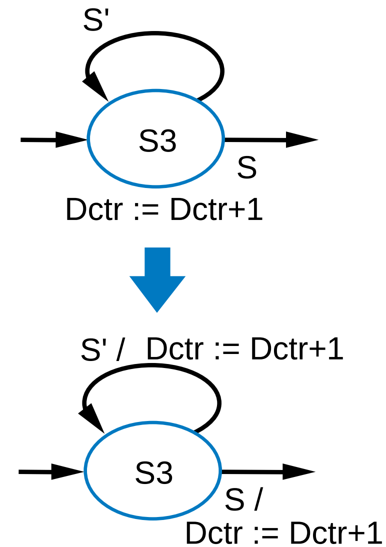




*a*

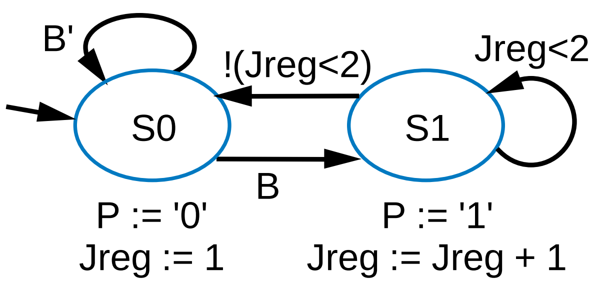
- Once reflection detected (S), go to new state **S4**
  - Calculate distance
  - Assuming clock frequency is $3\text{x}10^8$, *Dctr* holds number of meters, so Dreg:=Dctr/2
- After **S4**, go back to **S1** to wait for button again

# HLSM Actions: Updates Occur Next Clock Cycle

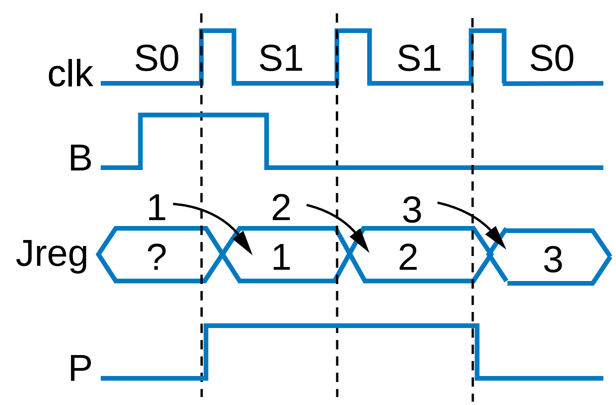
- Local storage updated on clock edges only
  - Enter state on clock edge
  - Storage writes in that state occur on *next* clock edge
  - Can think of as occurring on outgoing transitions
- ***Thus***, transition conditions use the OLD value, not the newly-written value
  - Example:




*Inputs*: B (bit)
*Outputs*: P (bit) // if B, 2 cycles high
*Local storage:* Jreg (8 bits)



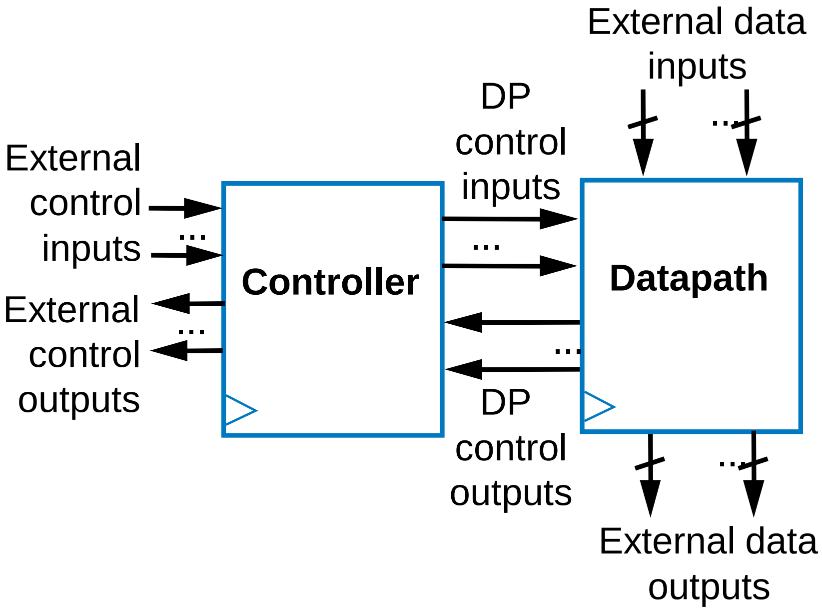

**(a)**



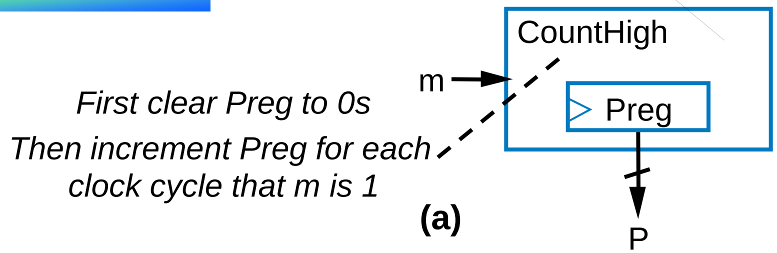

**(b)**

# RTL Design Process

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

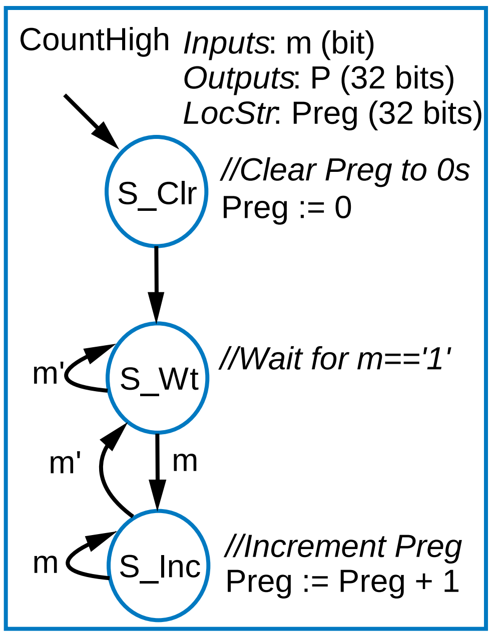


# Ctrl/DP Example for Earlier Cycles-High Counter




(a)

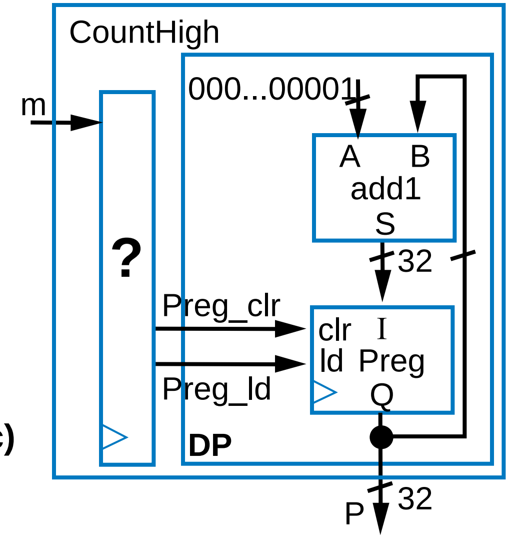
*We created this HLSM earlier*




(b)

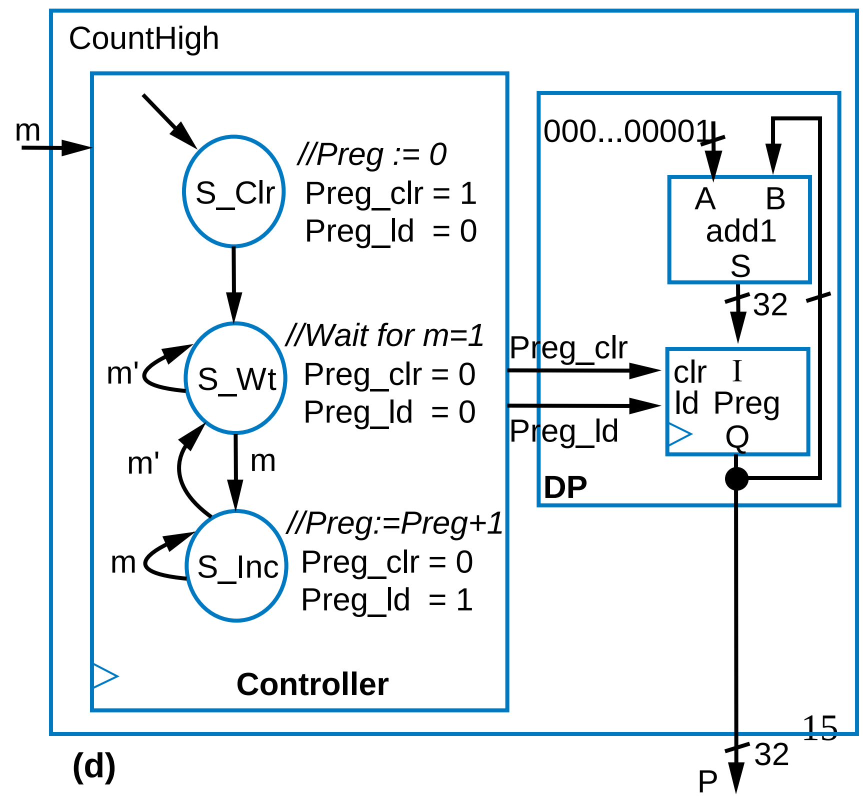
*Create DP*

*Connect with controller*




(c)

*Derive controller*



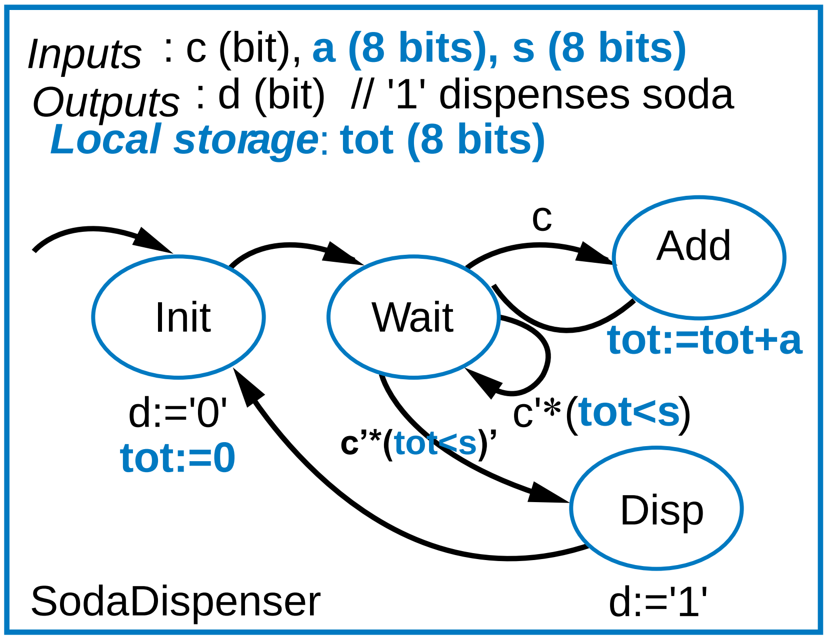

(d)

# RTL Design Process

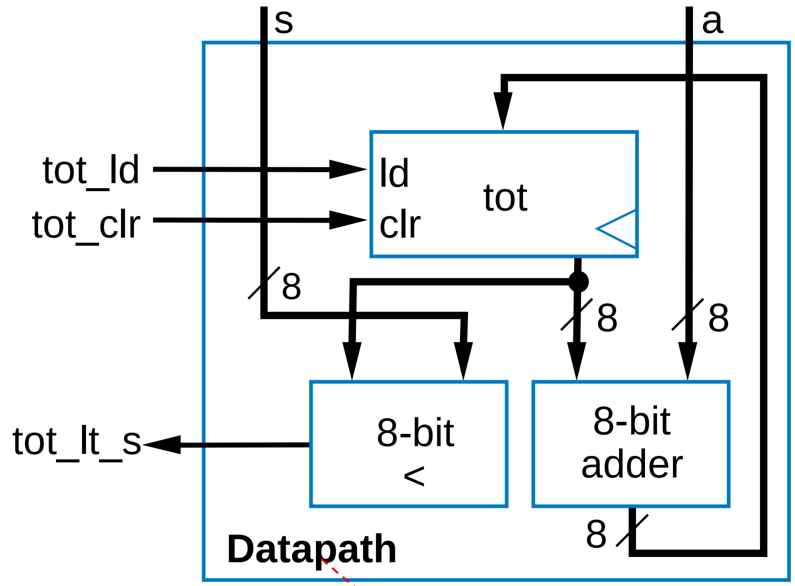
| | | Step | Description |
|---|---|---|---|
| **Step 1:** Capture behavior | | *Capture a high-level state machine* | Describe the system's desired behavior as a high-level state machine. The state machine consists of states and transitions. The state machine is "high-level" because the transition conditions 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. |

# Example: Soda Dispenser from Earlier

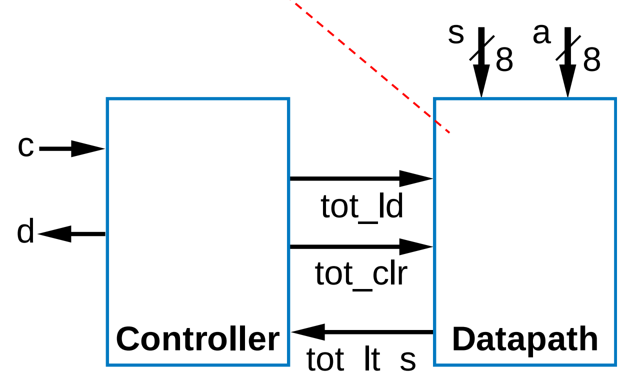
- Quick overview example.
  More details of each step to come.




Step 1



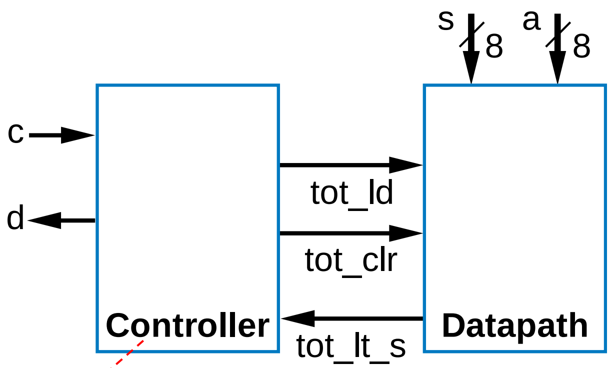

Step 2A



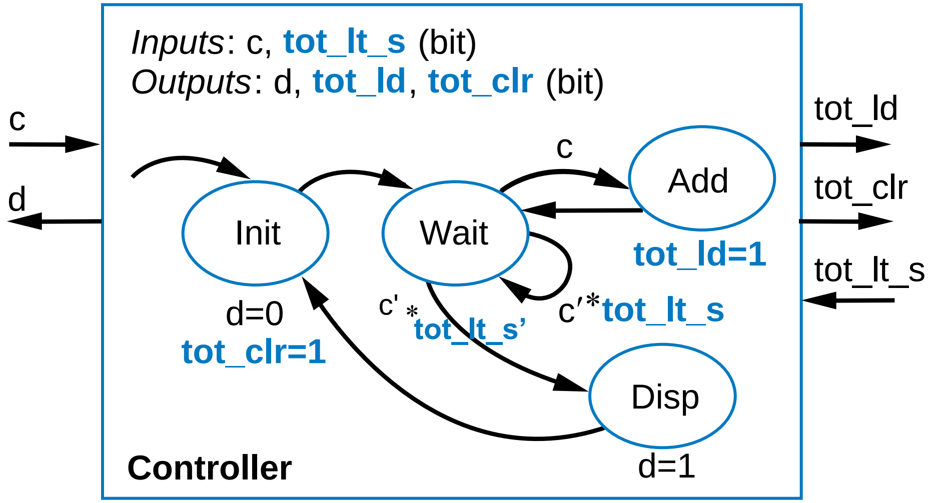

Step 2B

# Example: Soda Dispenser

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




Step 2B



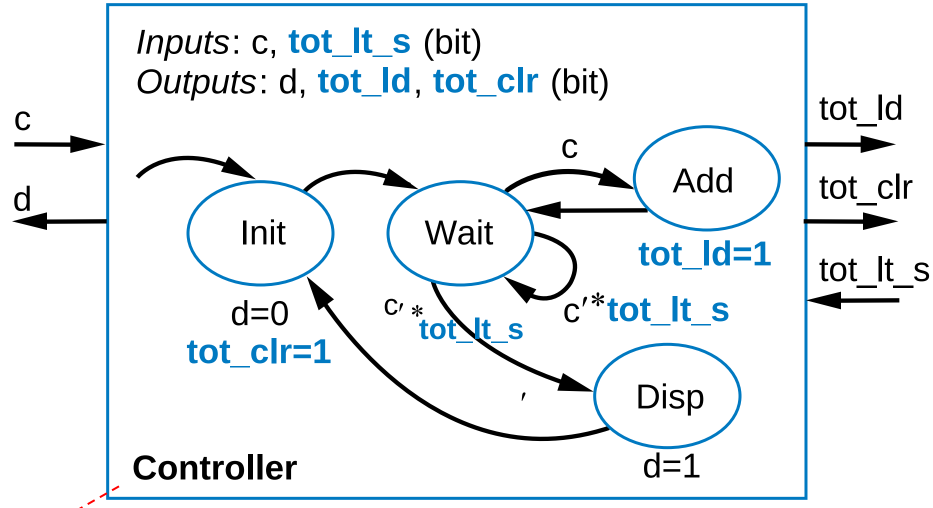

Step 1

Step 2C

# Example: Soda Dispenser

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

| | s1 | s0 | c | 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 |
| | | ••• | | | | ••• | | | |



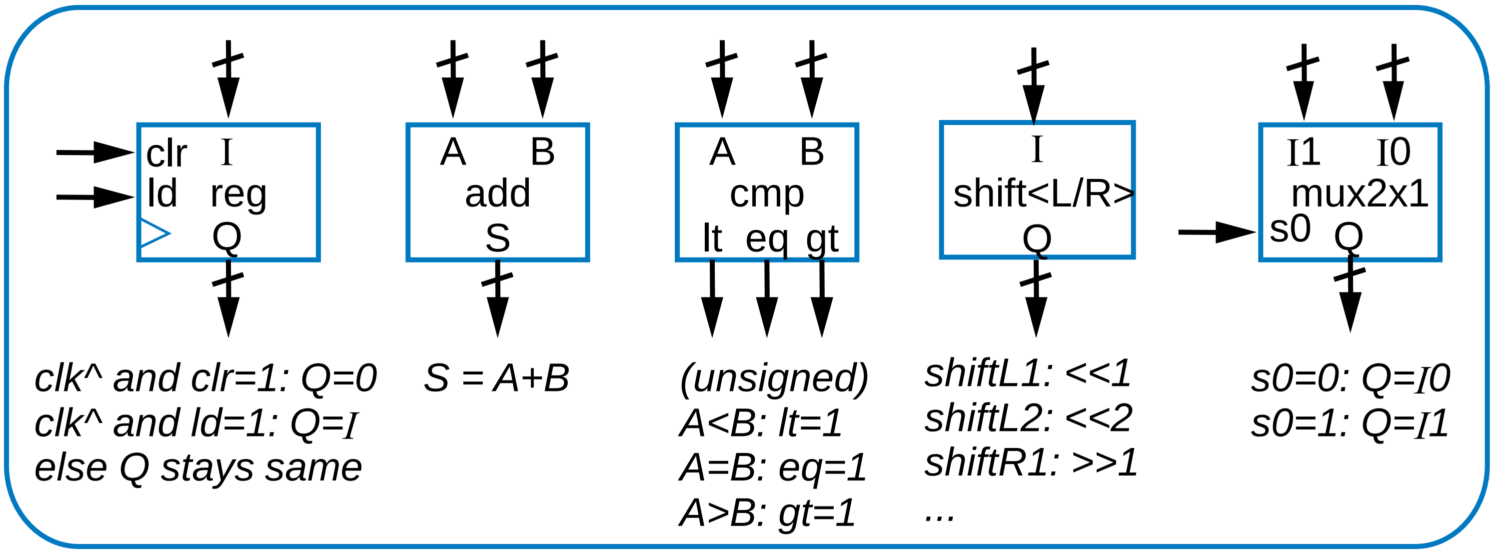

Step 2C

Use controller design process (Ch3) to complete the design

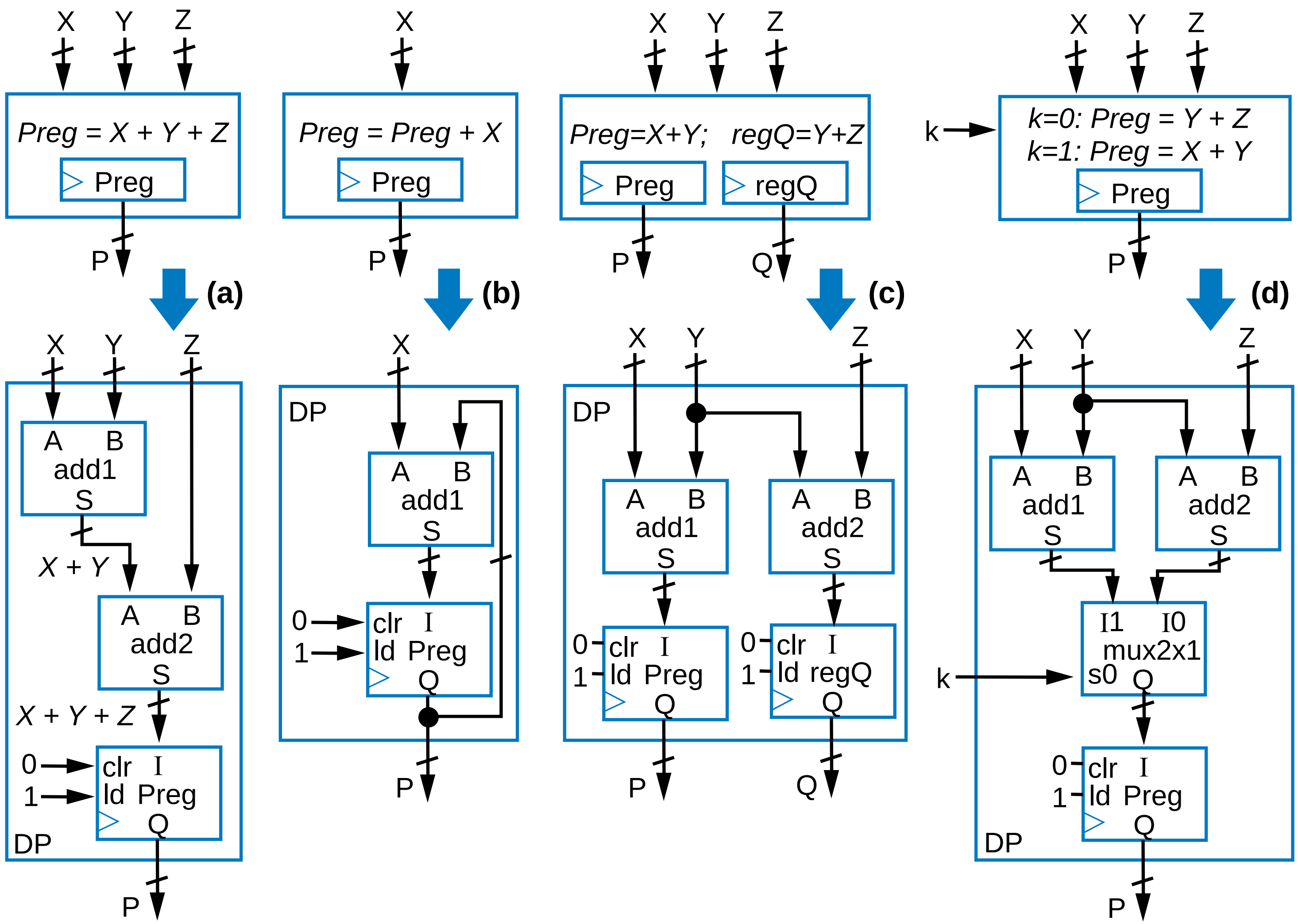
*a*

# 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

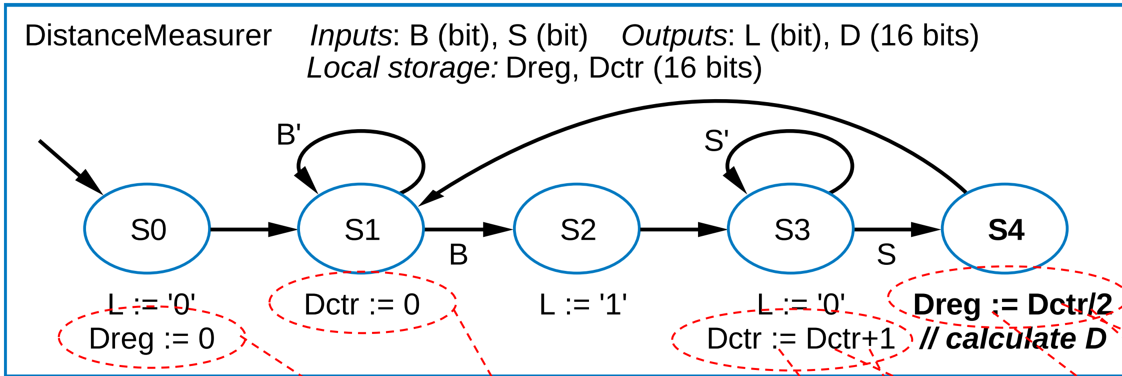


# Step 2A: Create a Datapath—Simple Examples



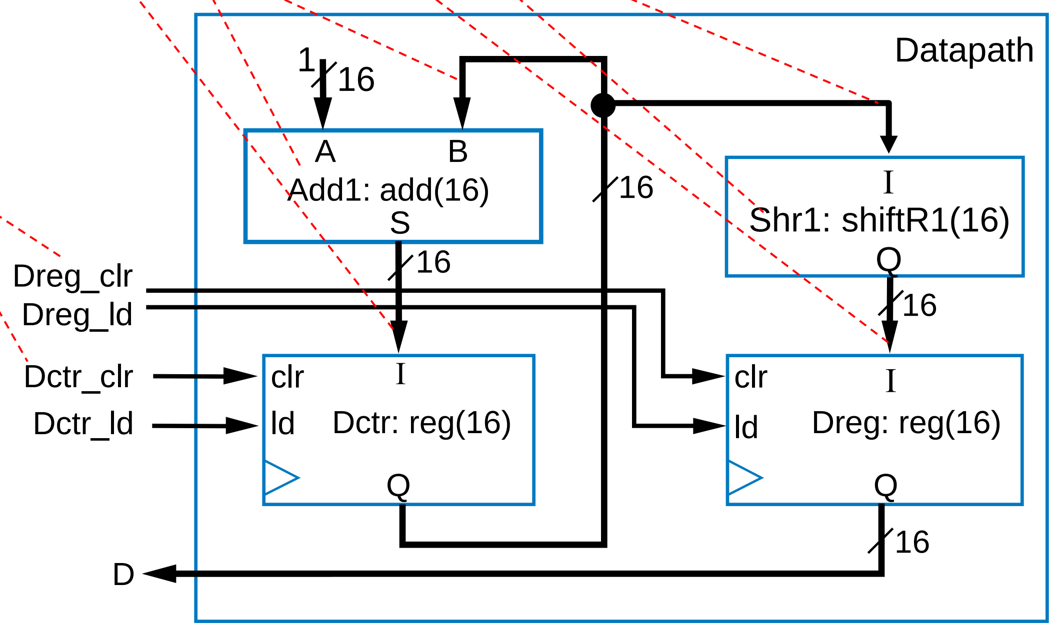

a

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

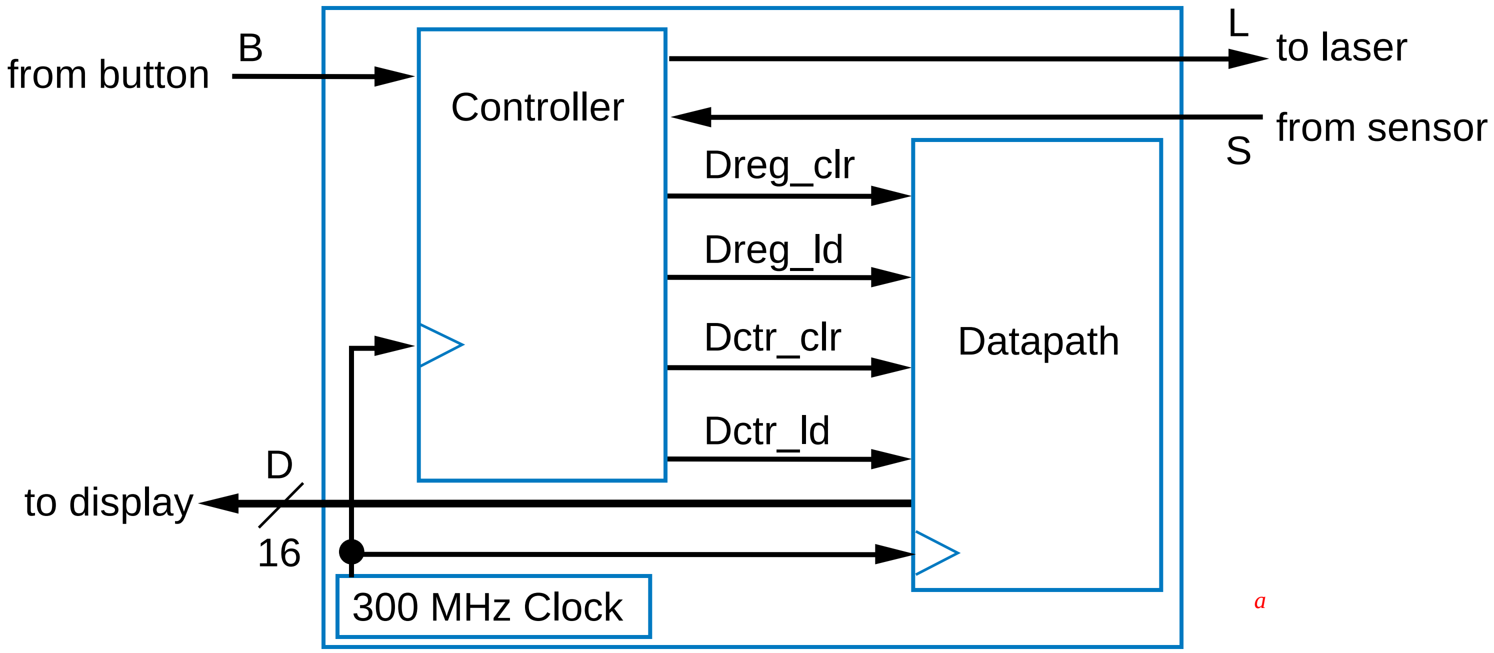



a

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

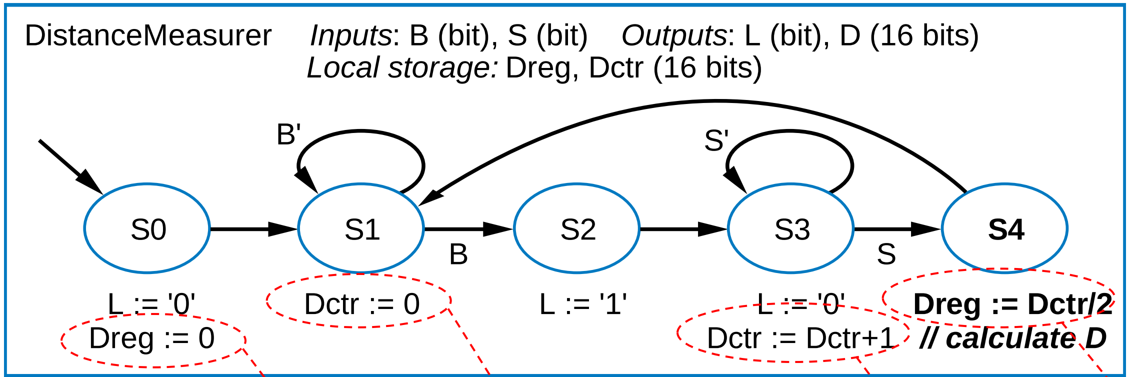


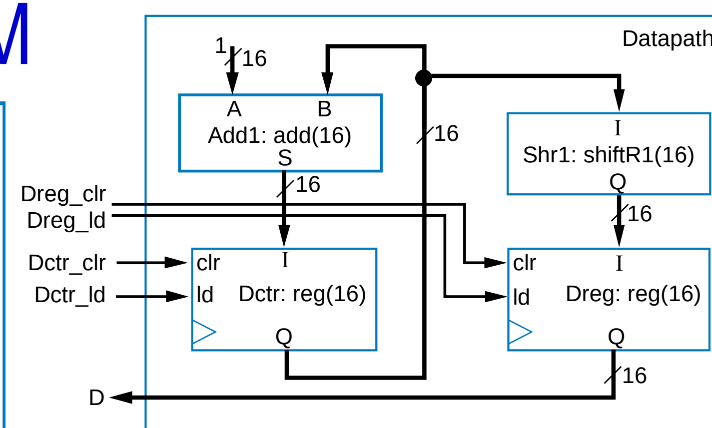
# Laser-Based Distance Measurer—Step 2B: Connecting the Datapath to a Controller



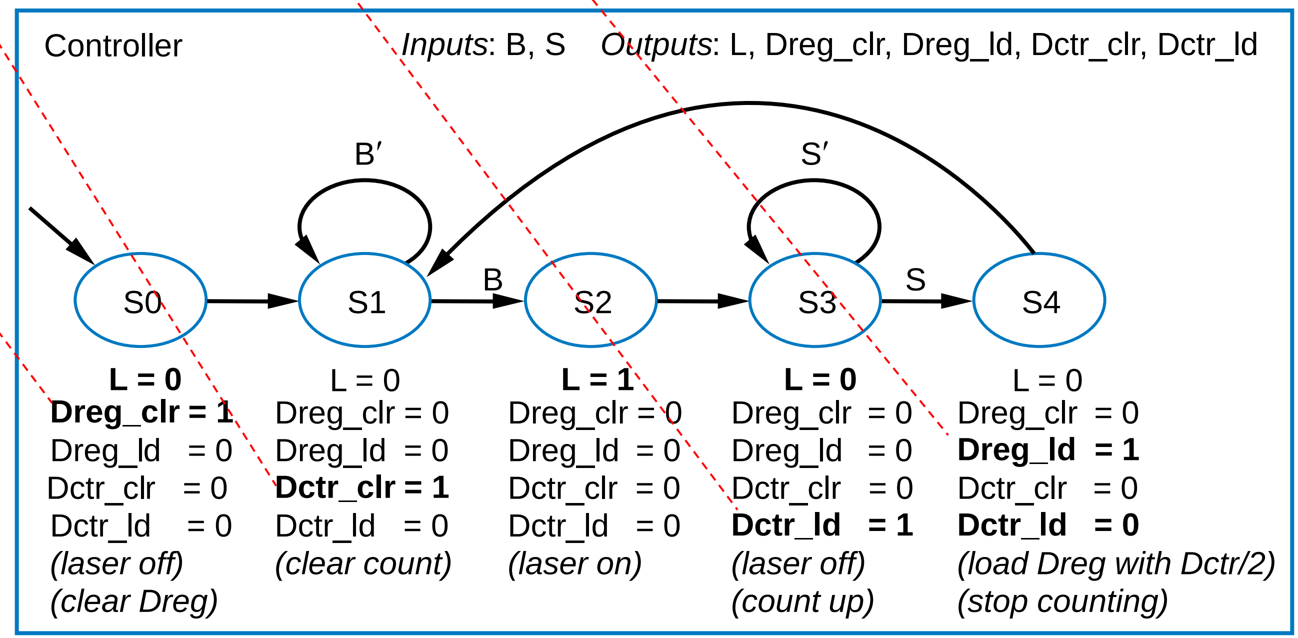
# Laser-Based Distance Measurer—Step 2C: Derive the Controller FSM

HLSM

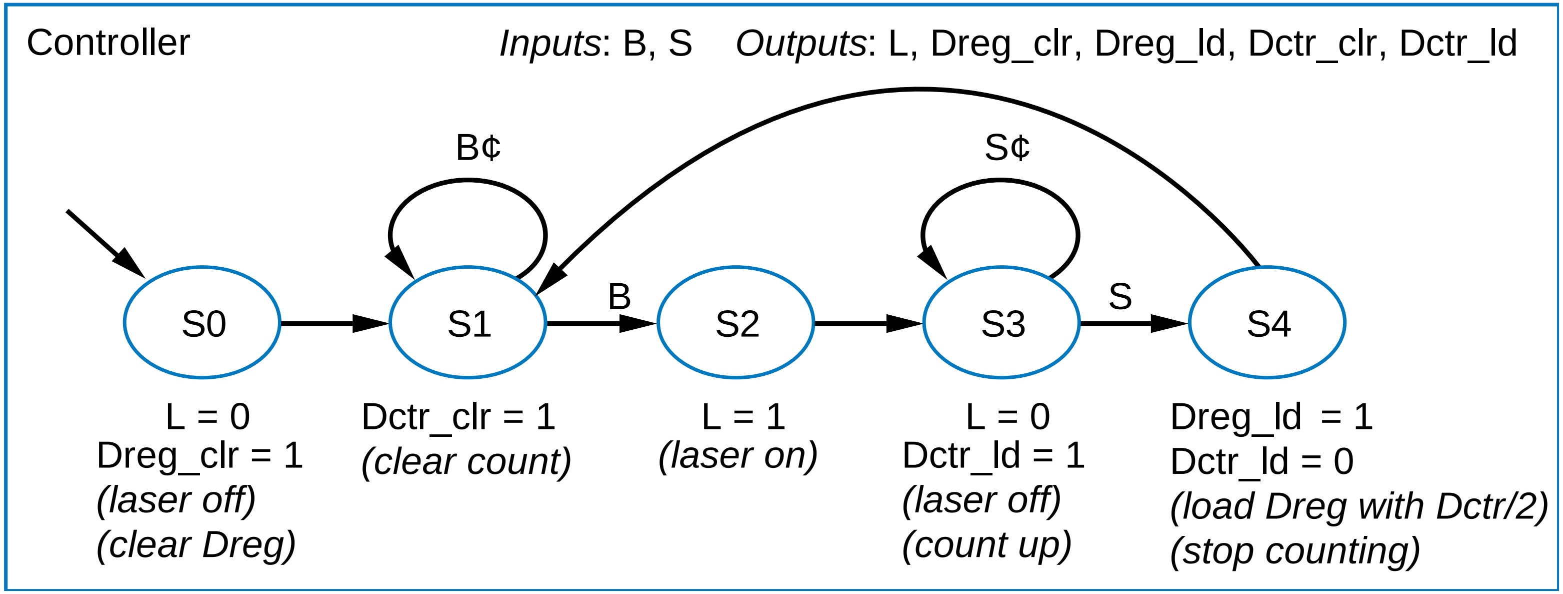



- FSM has same states, transitions, and control I/O
- Achieve each HLSM data operation using datapath control signals in FSM



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

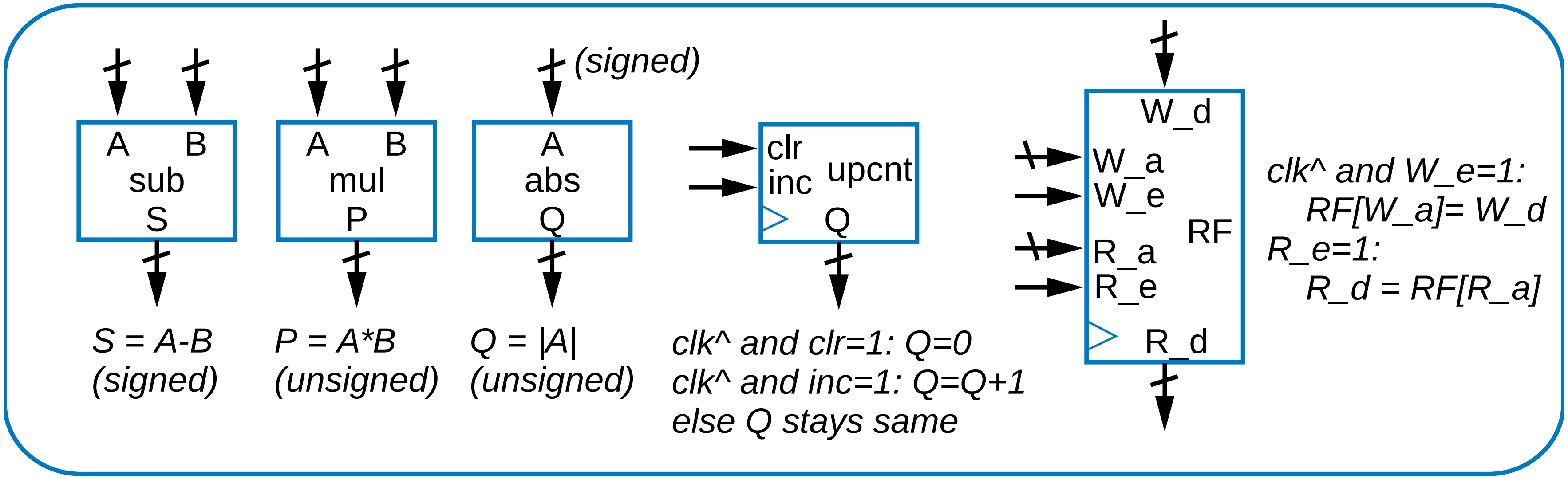



- Same FSM, using convention of unassigned outputs implicitly assigned 0

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

# More RTL Design

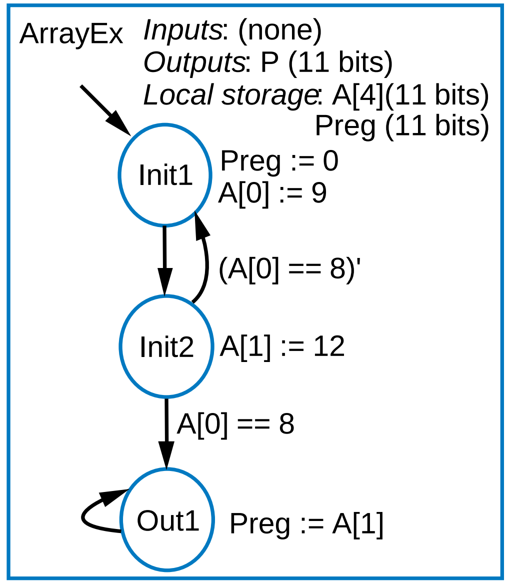
- Additional datapath components



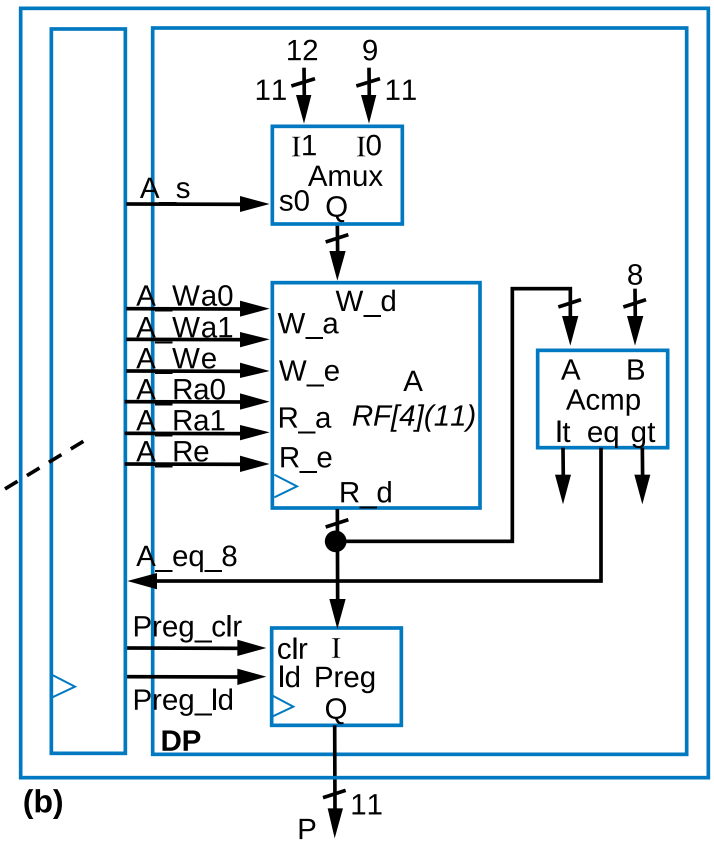
# RTL Design Involving Register File or Memory

- 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

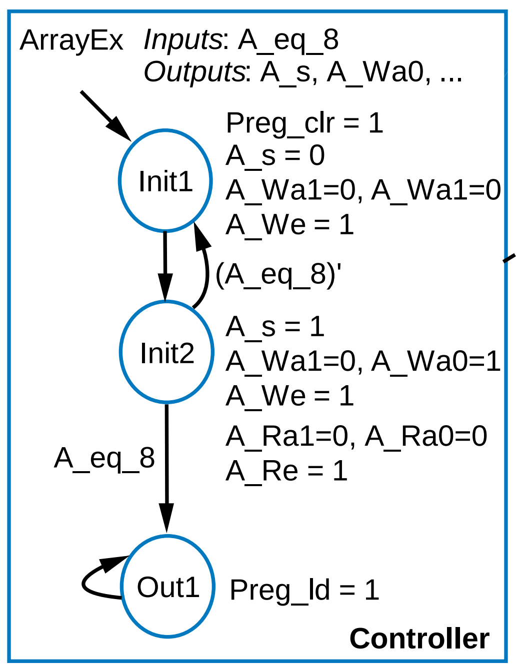
# Simple Array Example




(a)

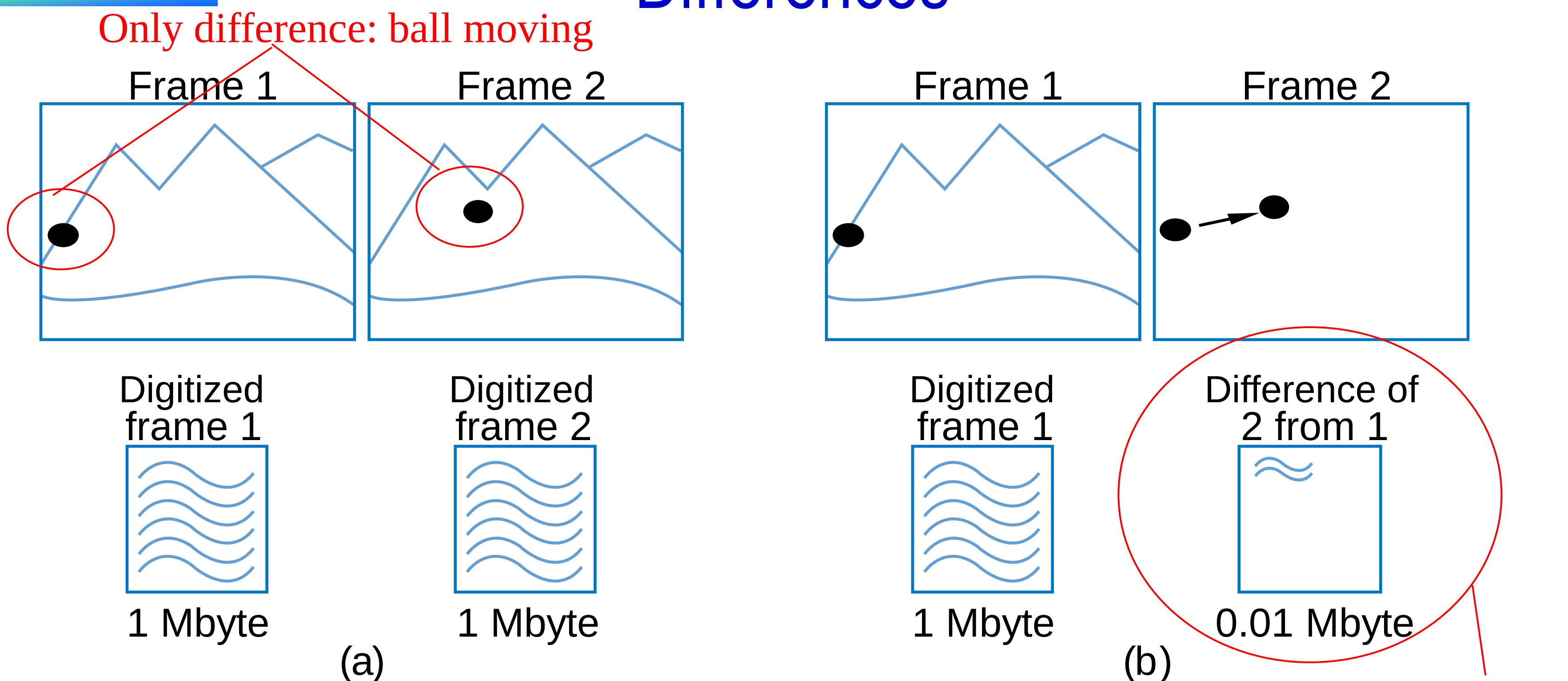



(b)



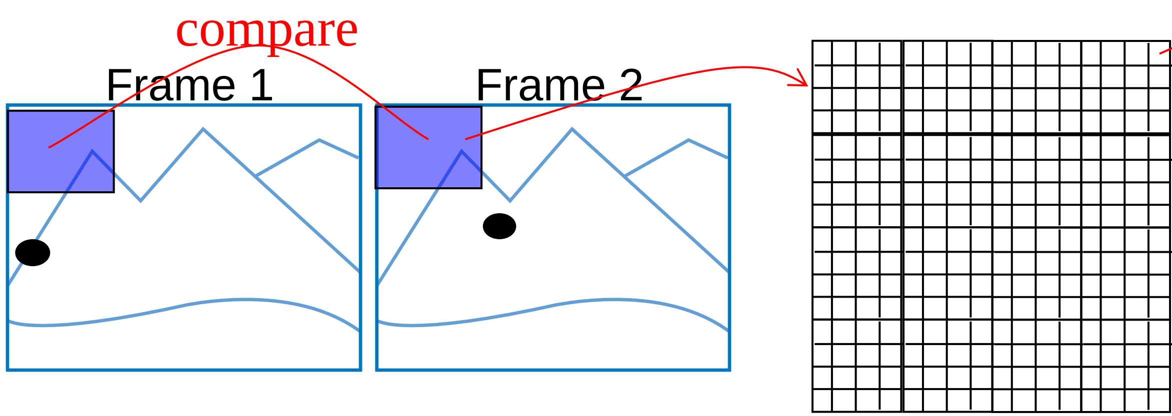

(c)

# 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

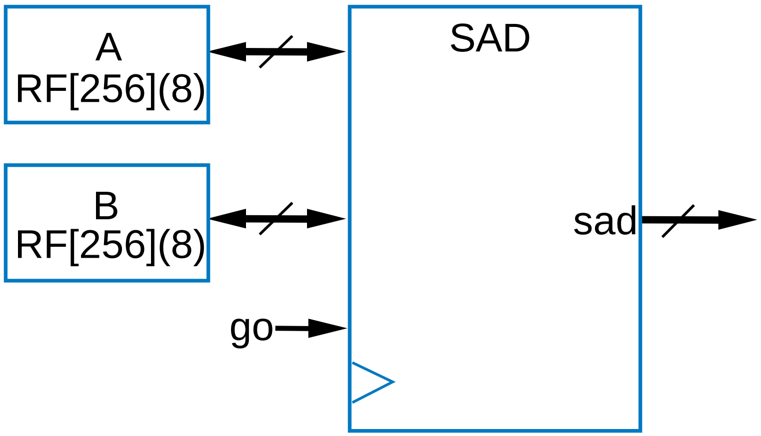
# RTL Example: Video Compression – Sum of Absolute Differences



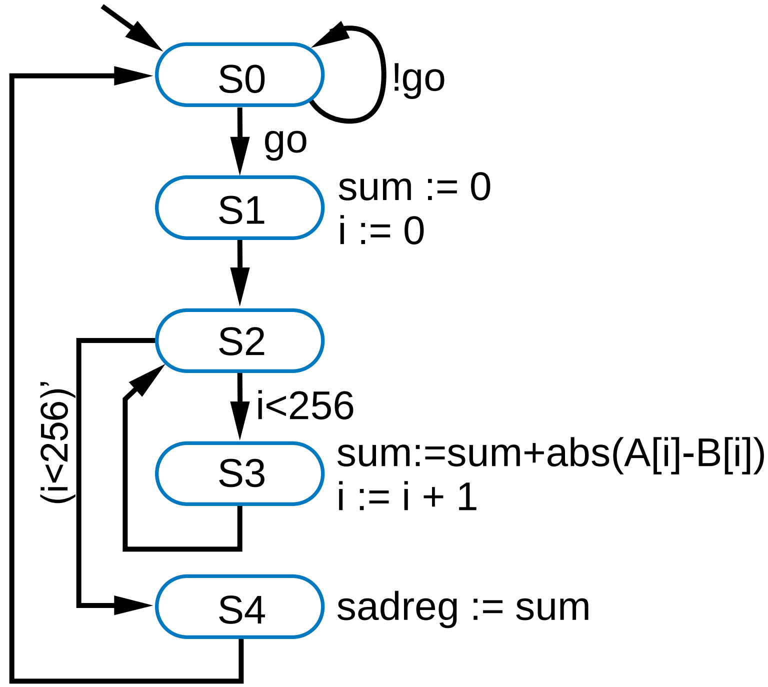

Each is a pixel, assume represented as 1 byte (actually, a color picture might have 3 bytes per pixel, for intensity of red, green, and blue components of pixel)

- 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)

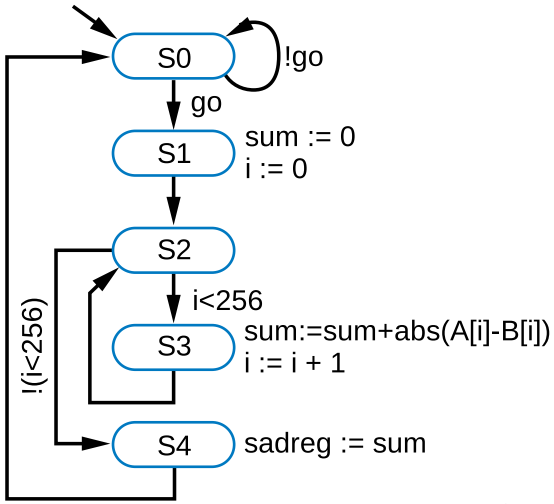


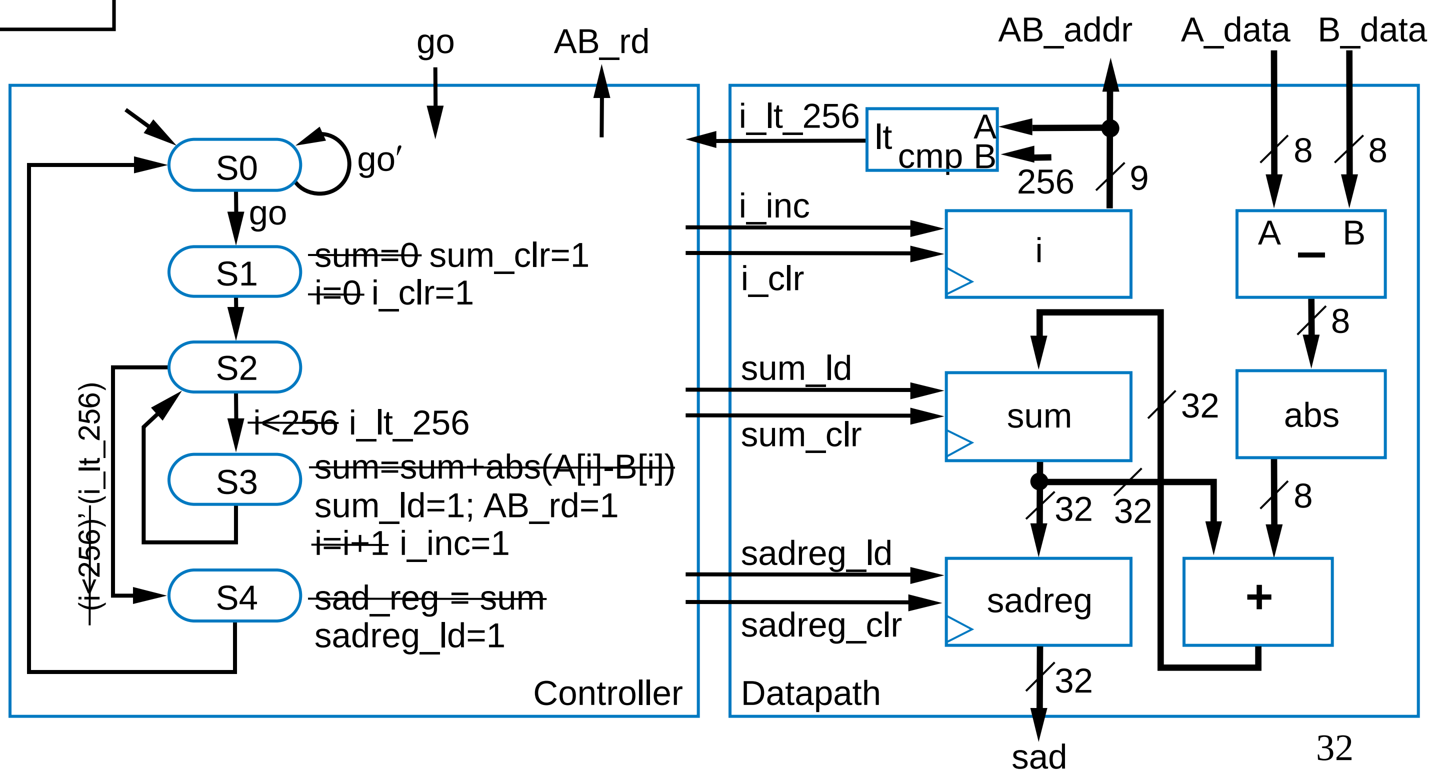

(**b**)

- **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

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

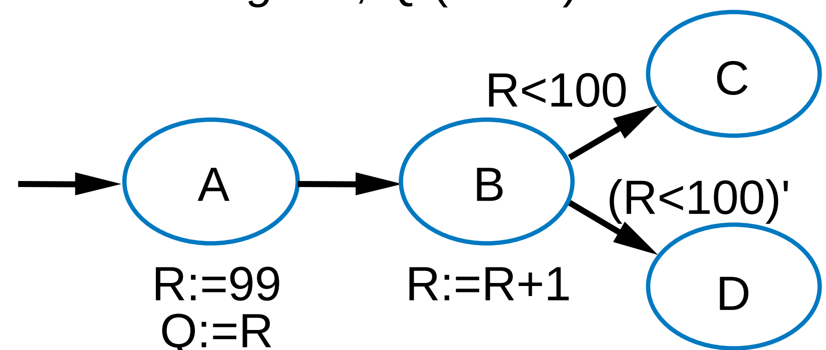


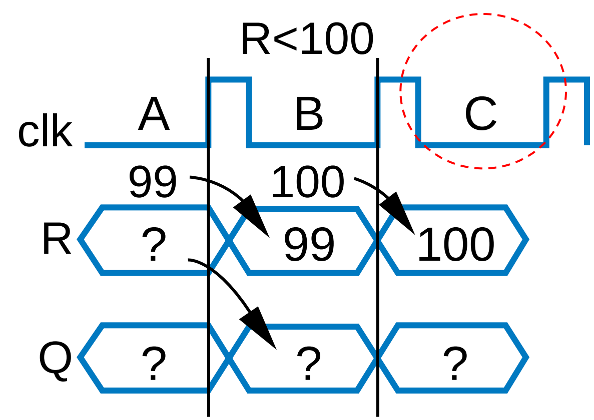



# Common RTL Design Pitfall Involving Storage Updates

- 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)

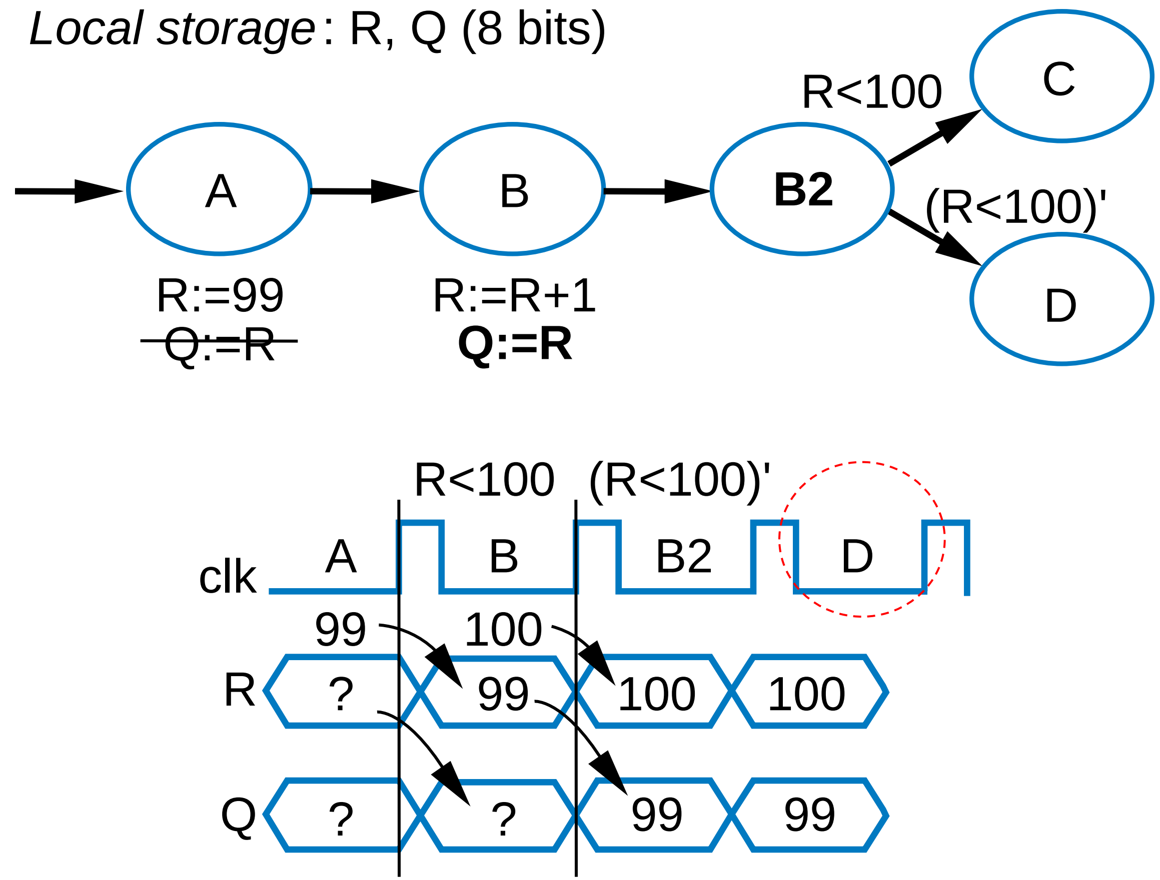


# Common RTL Design Pitfall Involving Storage Updates

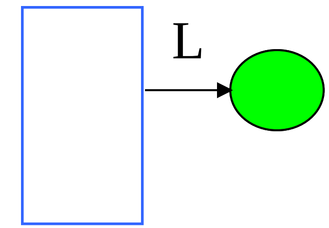
- New HLSM using extra state so read of R occurs after write of R

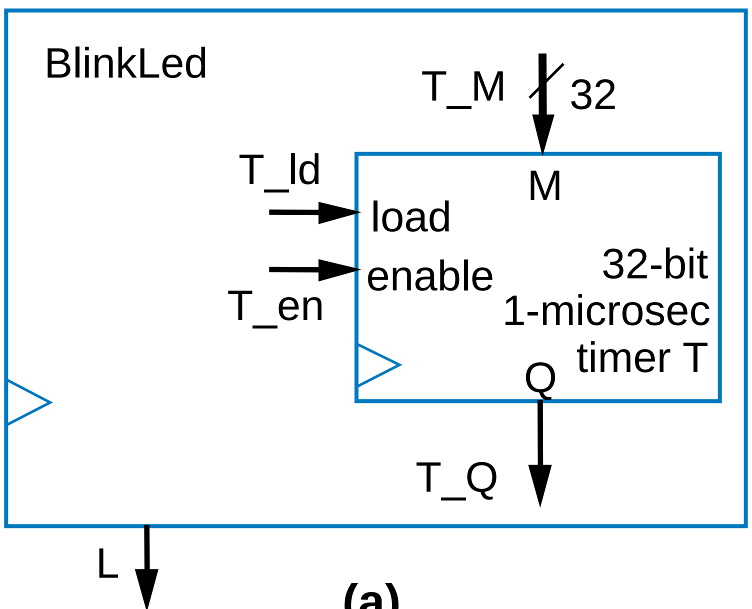
*Local storage*: R, Q (8 bits)



# RTL Design Involving a Timer

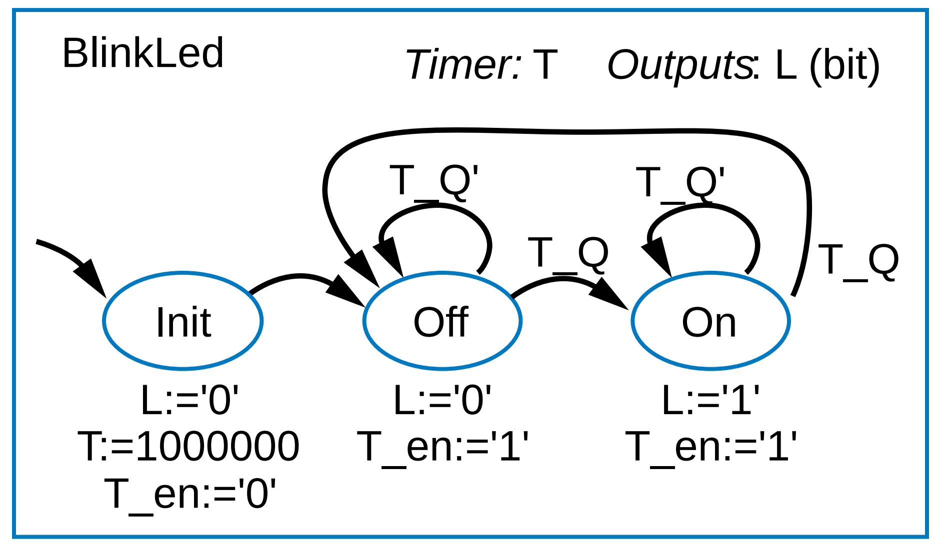
- Commonly need explicit time intervals
  - Ex: Repeatedly blink LED on 1 second, off 1 second
- Pre-instantiate timer that HLSM can then use




**(a)**

Pre-instantiated timer

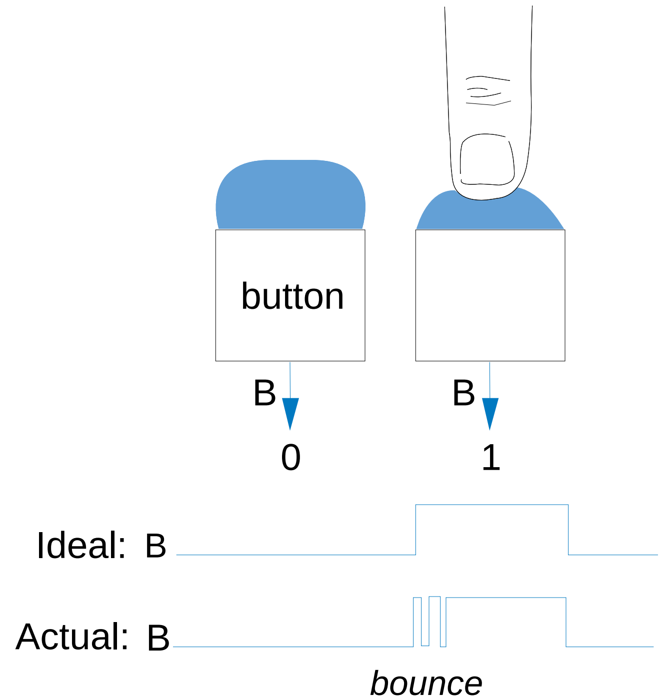


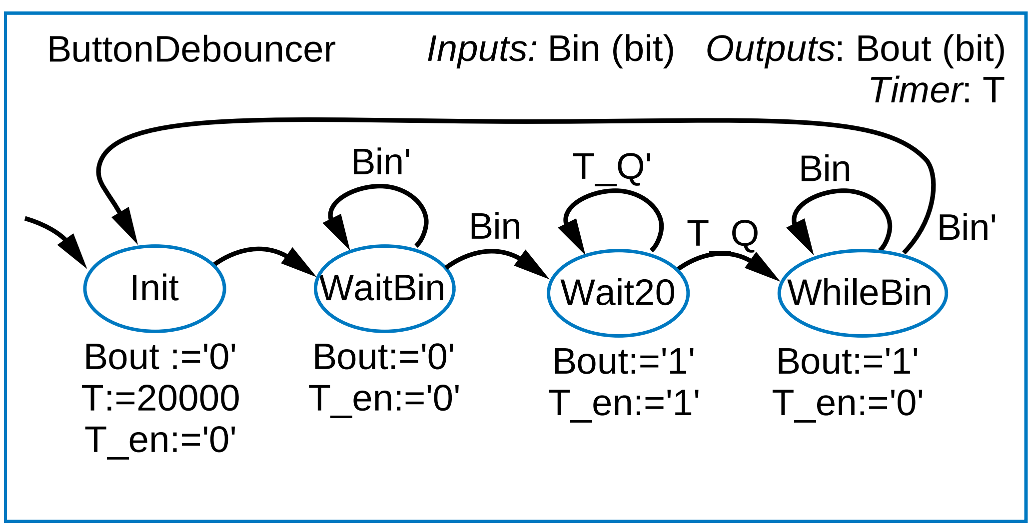

**(b)**

HLSM making use of timer

# Button Debouncing

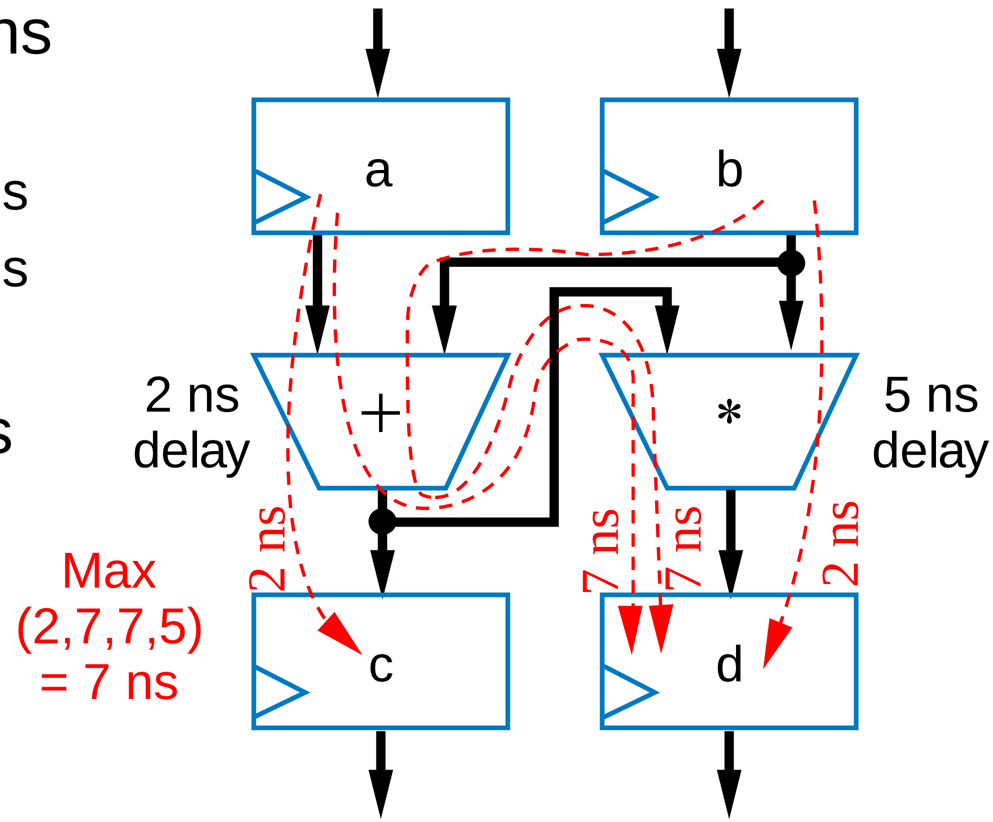
- Press button
  - Ideally, output changes to 1
  - Actually, output bounces
    - Due to mechanical reasons
    - Like ball bouncing when dropped to floor
- Digital circuit can convert actual signal closer to ideal signal



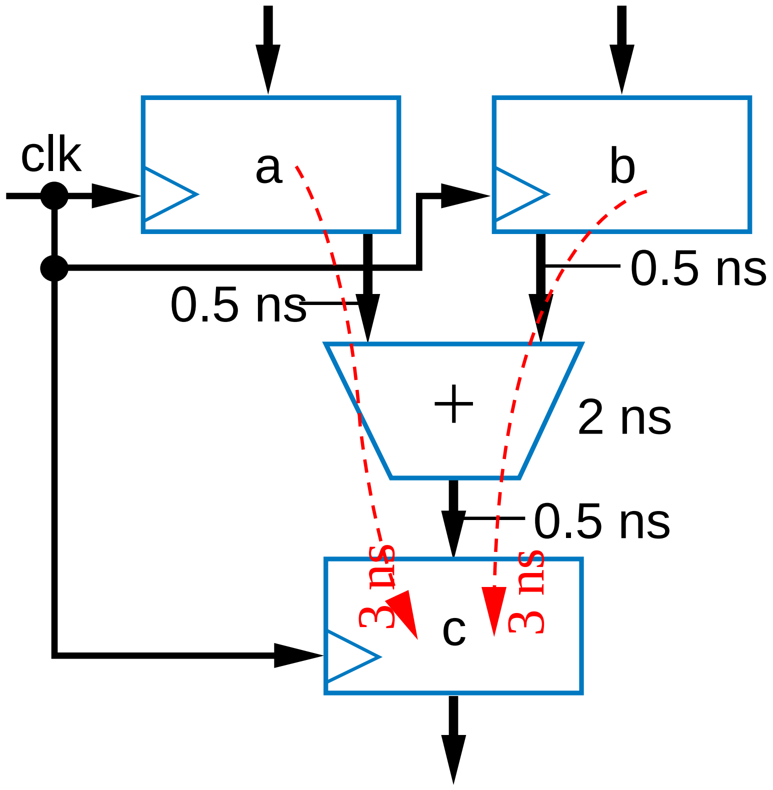
# Critical Path

- 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



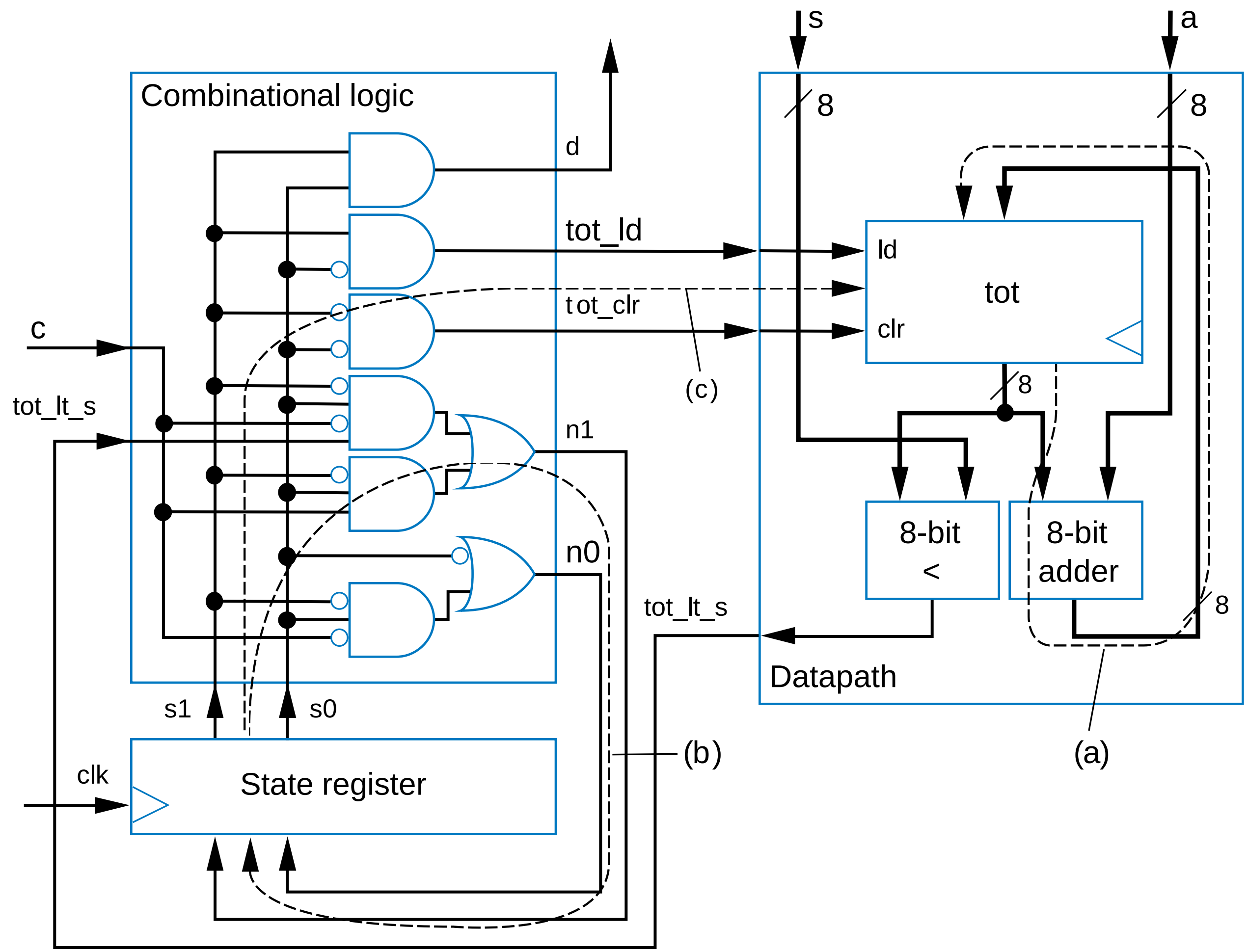
# Critical Path Considering Wire Delays

- 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

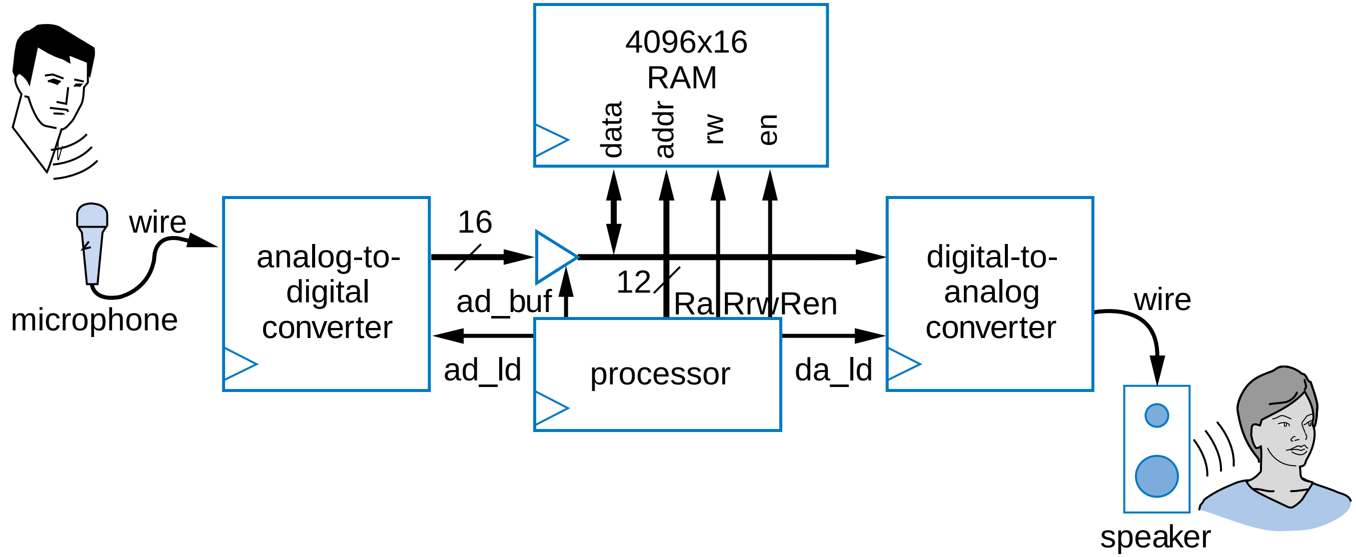


# A Circuit May Have Numerous Paths

- 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



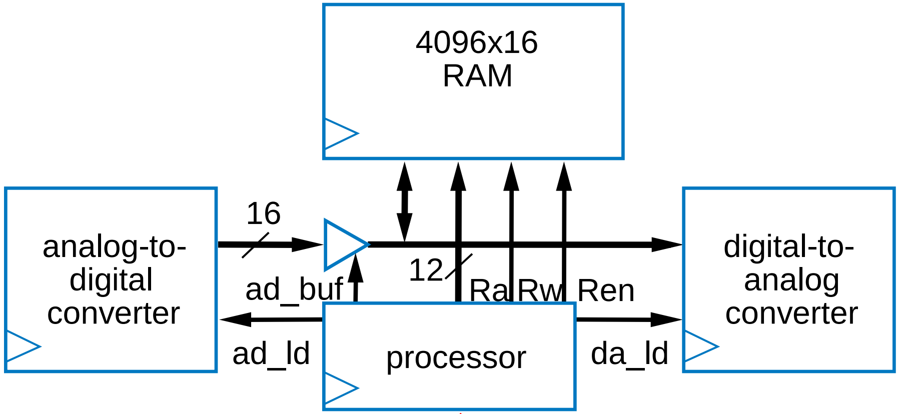
# RAM Example: Digital Sound Recorder



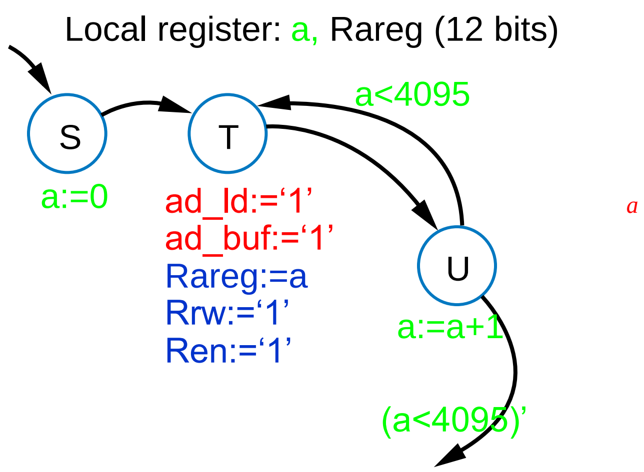

- Behavior
  - Record: Digitize sound, store as series of 4096 12-bit digital values in RAM
    - We'll use a 4096x16 RAM (12-bit wide RAM not common)
  - Play back later
  - Common behavior in telephone answering machine, toys, voice recorders
- To record, processor should read a-to-d, store read values into successive RAM words
  - To play, processor should read successive RAM words and enable d-to-a

# RAM Example: Digital Sound Recorder

- RTL design of processor
  - Create HLSM
  - Begin with the *record* behavior
  - Create local storage *a*
    - Stores current address, ranges from 0 to 4095 (thus need 12 bits)
  - Create state machine that counts from 0 to 4095 using *a*
    - For each *a*
      - Read analog-to-digital conv.
        - » ad_ld:='1', ad_buf:='1'
      - Write to RAM at address *a*
        - » Rareg:=a, Rrw:='1', Ren:='1'

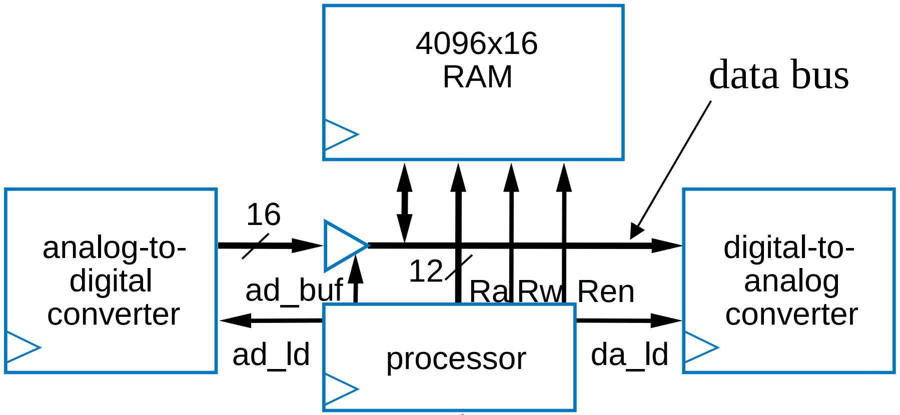



*Record* behavior

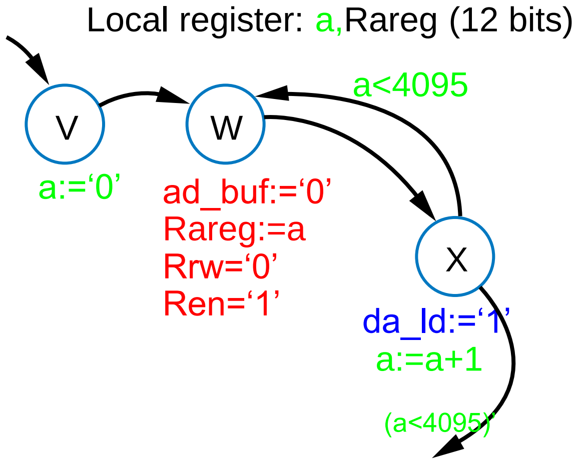


# RAM Example: Digital Sound Recorder

- Now create *play* behavior
- Use local register *a* again, create state machine that counts from 0 to 4095 again
  - For each *a*
    - Read RAM
    - Write to digital-to-analog conv.
  - Note: Must write d-to-a one cycle *after* reading RAM, when the read data is available on the data bus
- The record and play state machines would be parts of a larger state machine controlled by signals that determine when to record or play




*Play* behavior



# ROM Example: Digital Telephone Answering Machine Using a Flash Memory

- HLSM
  - Once *rec=1*, begin erasing flash by setting *er=1*
  - Wait for flash to finish erasing by waiting for *bu=0*
  - Execute loop that sets local register *a* from 0 to 4095, reading analog-to-digital converter and writing to flash for each *a*