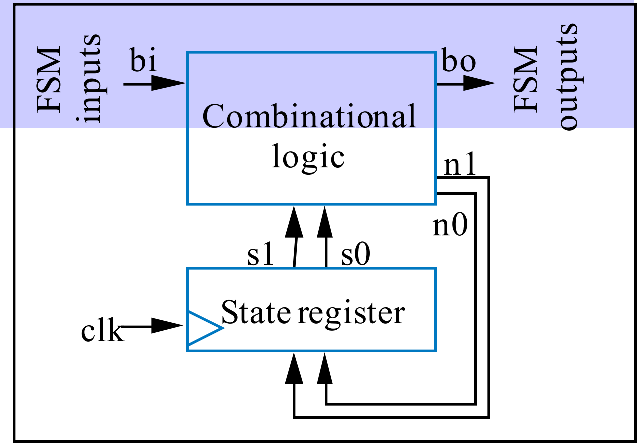
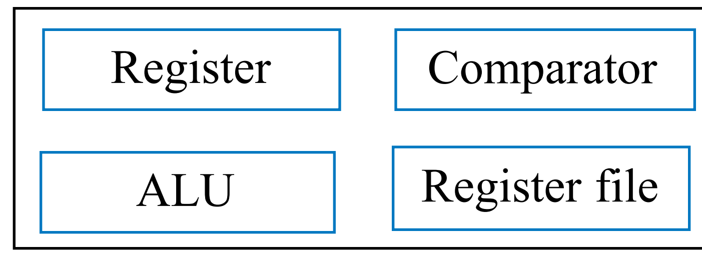
Topic 13

RTL Design

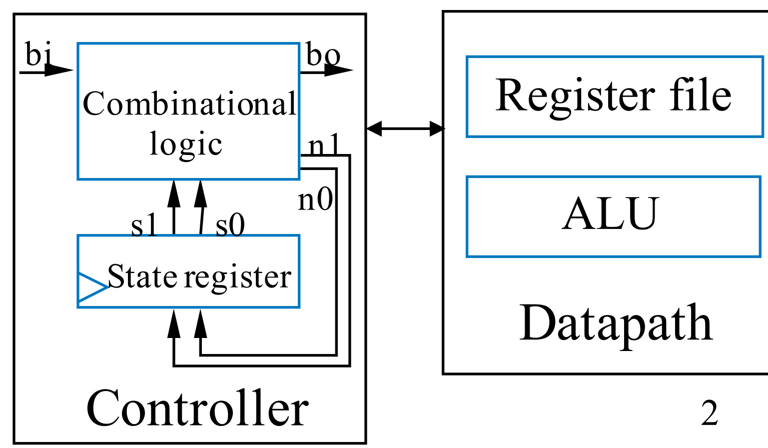
Introduction



- Controllers (FSM)
  - Describes behavior of circuits
  - Takes inputs, generates outputs
  - Implemented with state register and combinational logic



- Datapath components
  - Operations on data
  - Path that data flows through
  - Places data is stored
- Digital Device
  - Controller and datapath components working together
  - To implement an algorithm
  - Design on Register Transfer Level

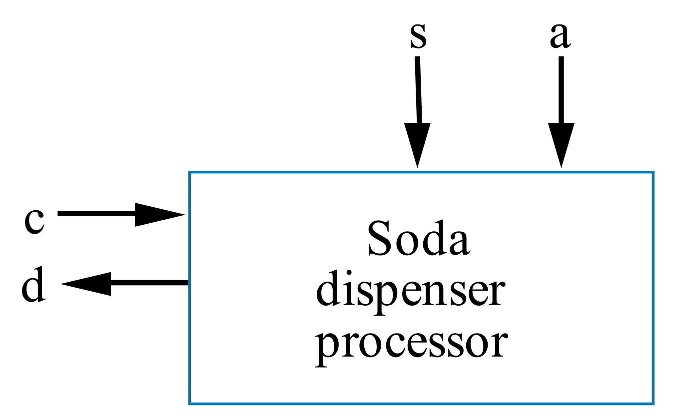


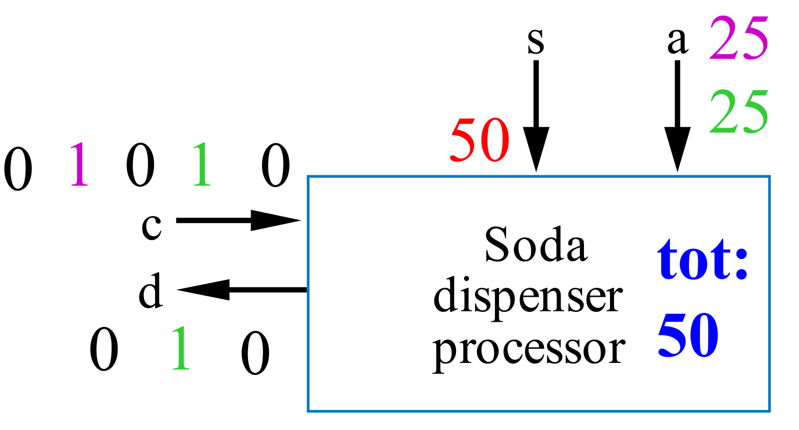
RTL Design Method

	Step	Description
Step 1	*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 bit inputs and outputs.
Step 2	*Create a datapath*	Create a datapath to carry out the data operations of the high-level state machine.
Step 3	*Connect the datapath to a controller*	Connect the datapath to a controller block. Connect external Boolean inputs and outputs to the controller block.
Step 4	*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: Vending Machine (Selling Soda)

- Soda dispenser
  - *c*: 1-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*: 1-bit output, processor sets to 1 when total value of deposited coins equals or exceeds cost of a soda

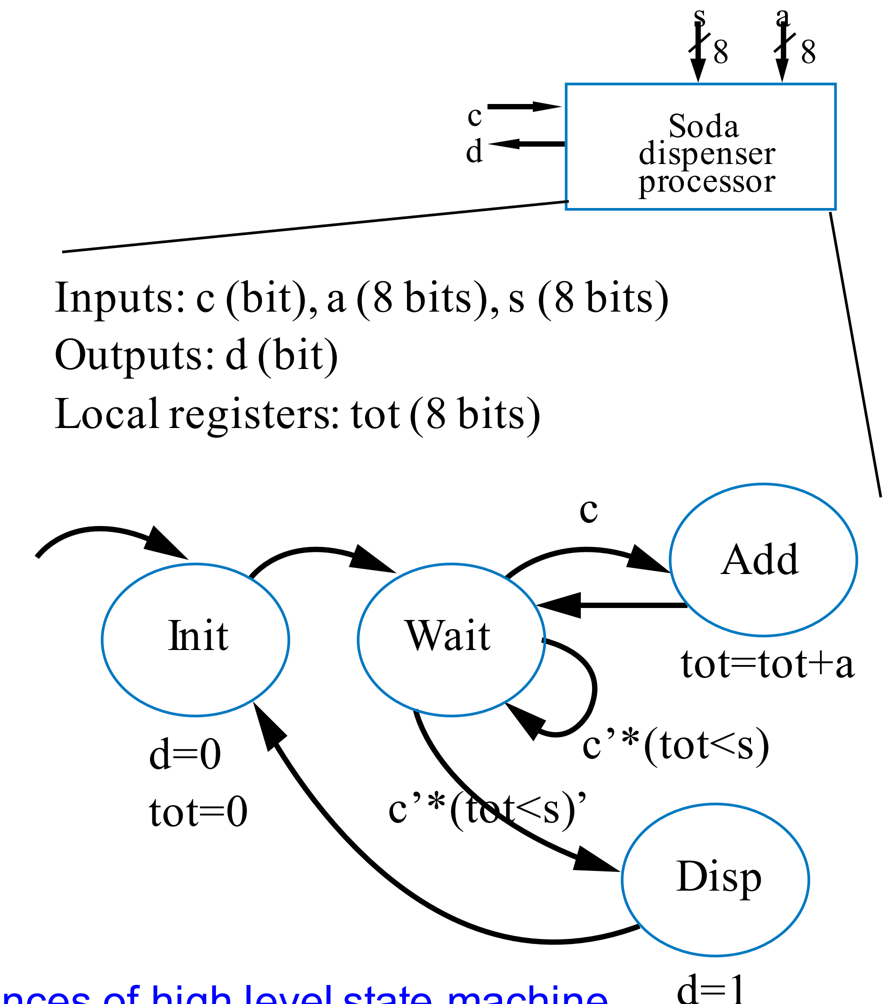




How can we precisely describe this processor's behavior?

Example: Step 1 – Capture High-Level State Machine

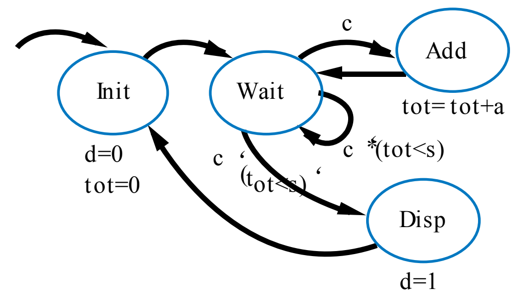
- Declare local register *tot*
- **Init** state: Set d=0, tot=0
- **Wait** state: wait for coin
  - If see coin, go to **Add** state
- **Add** state: Update total value: tot = tot + a
  - Remember, *a* is present coin's value
  - Go back to **Wait** state
- In **Wait** state, if tot >= s, go to **Disp** state
- **Disp** state: Set d=1 (dispense soda)
  - Return to **Init** state



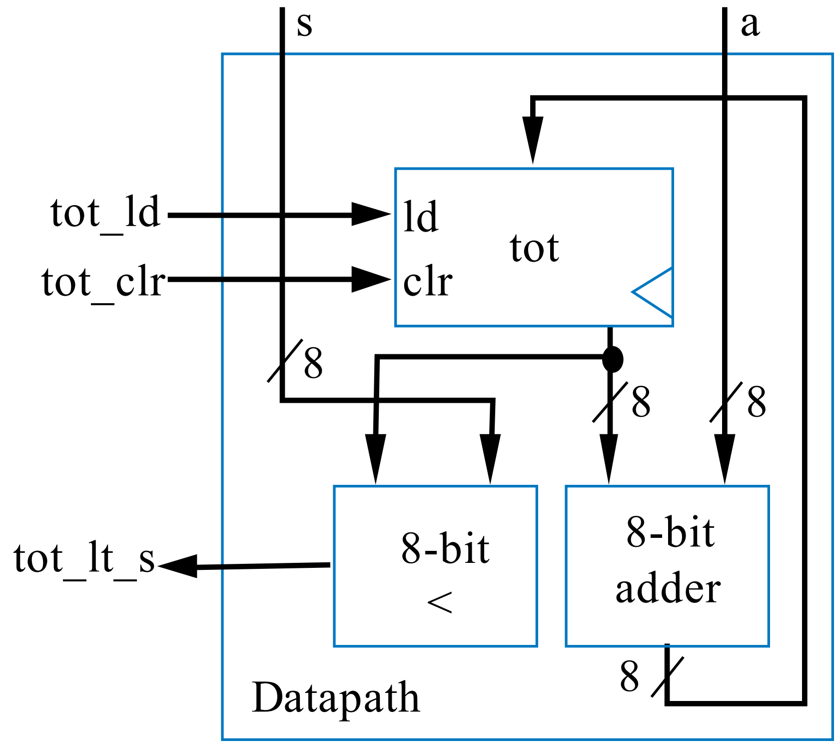
Differences of high level state machine
•Data types beyond just bits
•Arithmetic operations in states

Example: Step 2 – Create Datapath

Inputs : c (bit), a(8 bits), s (8 bits)
Outputs : d (bit)
Local registers : tot (8 bits)

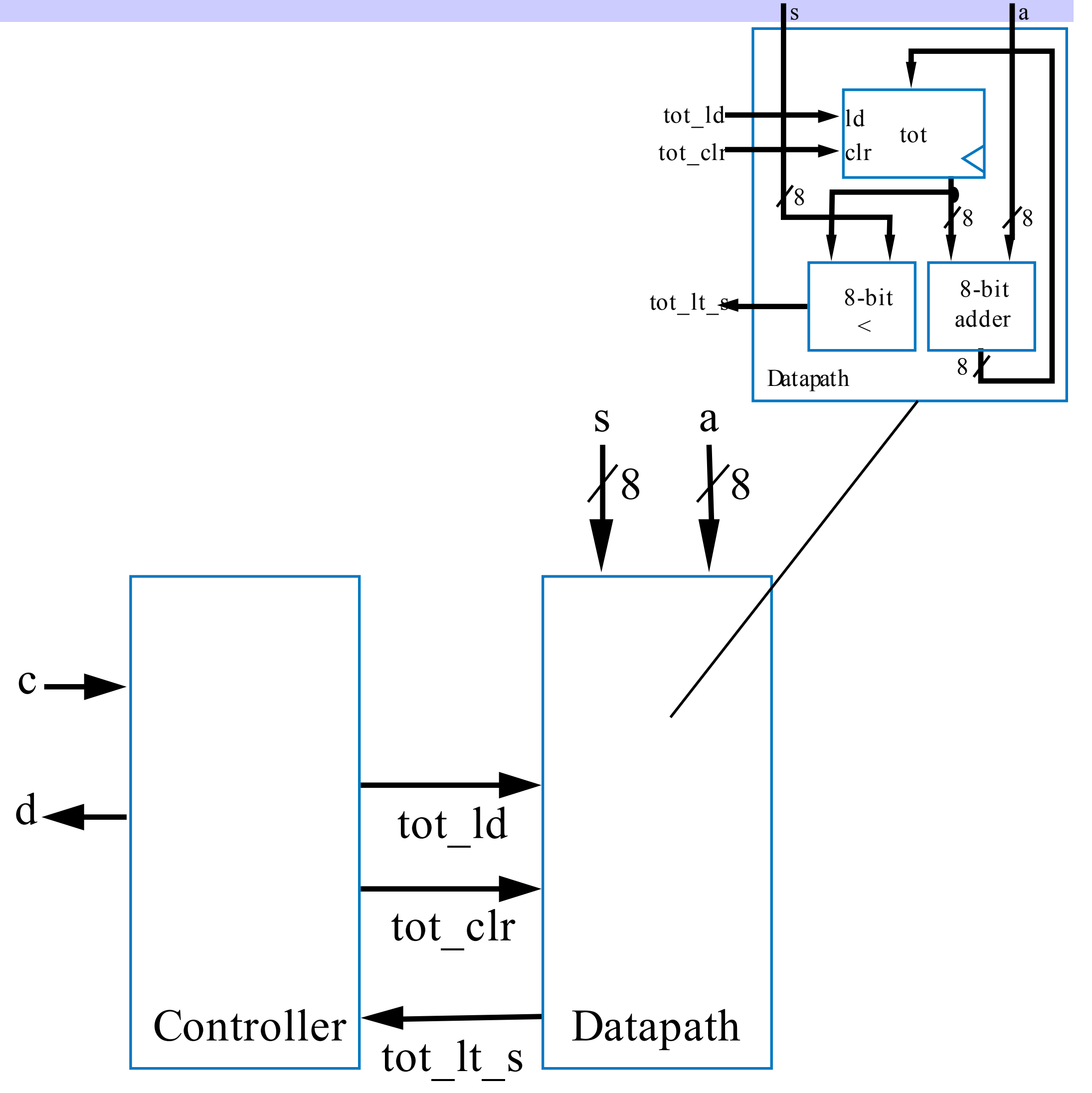


- Need *tot* register
  - To hold value between states
- Need 8-bit comparator
  - To compare *s* and *tot*
- Need 8-bit adder
  - To perform tot = tot + a
- Create control input/outputs for datapath components
  - Give them names



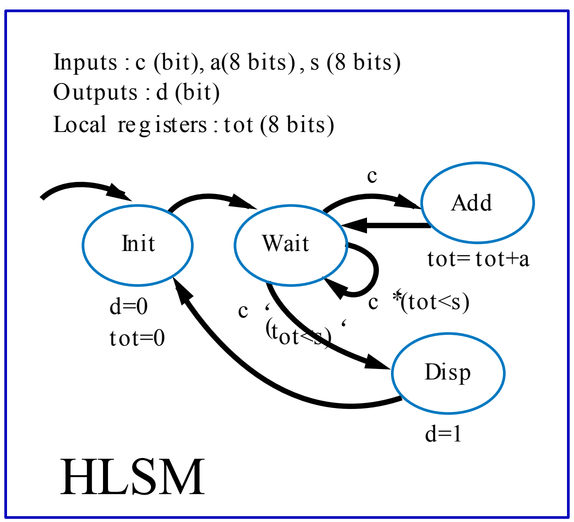
Example: Step 3 – Connect Datapath to a Controller

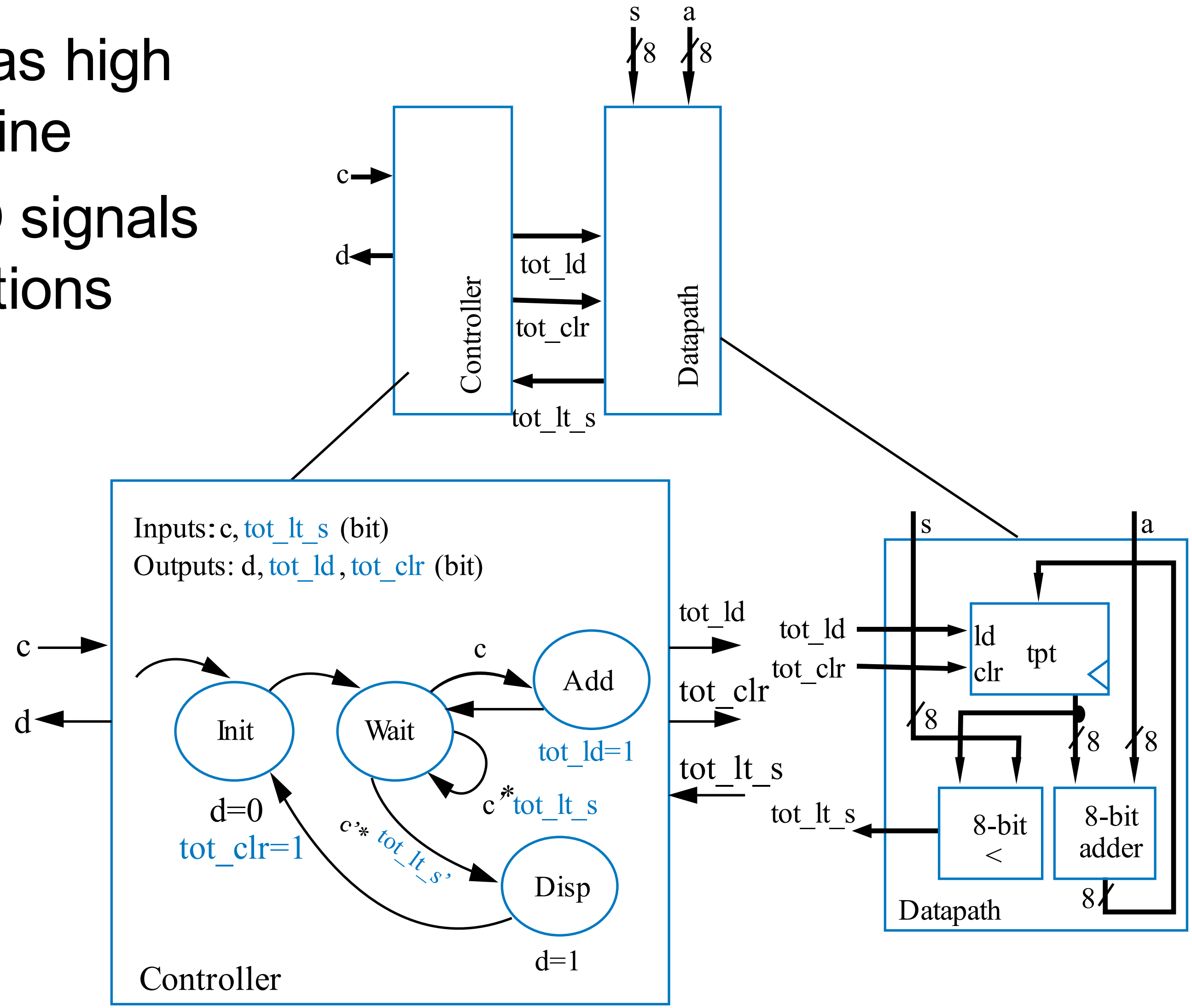
- Identify controller's inputs
  - *c* (coin detected)
  - comparator's output, which we named *tot_lt_s*
- Identify controller's outputs
  - *d* (dispense soda)
  - Signals to control datapath: *tot_ld* and *tot_clr*



Example: Step 4 – Derive the Controller's FSM

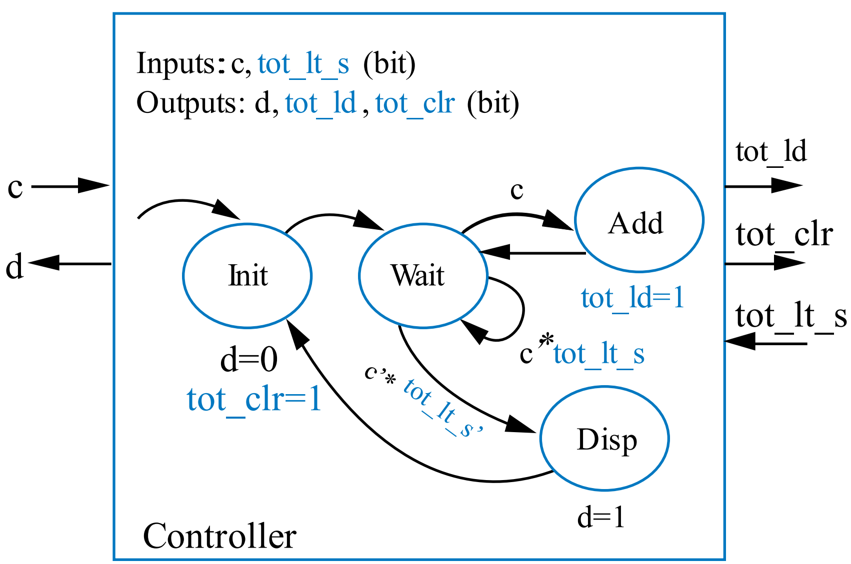
- Same structure as high level state machine
- But deal with I/O signals instead of operations





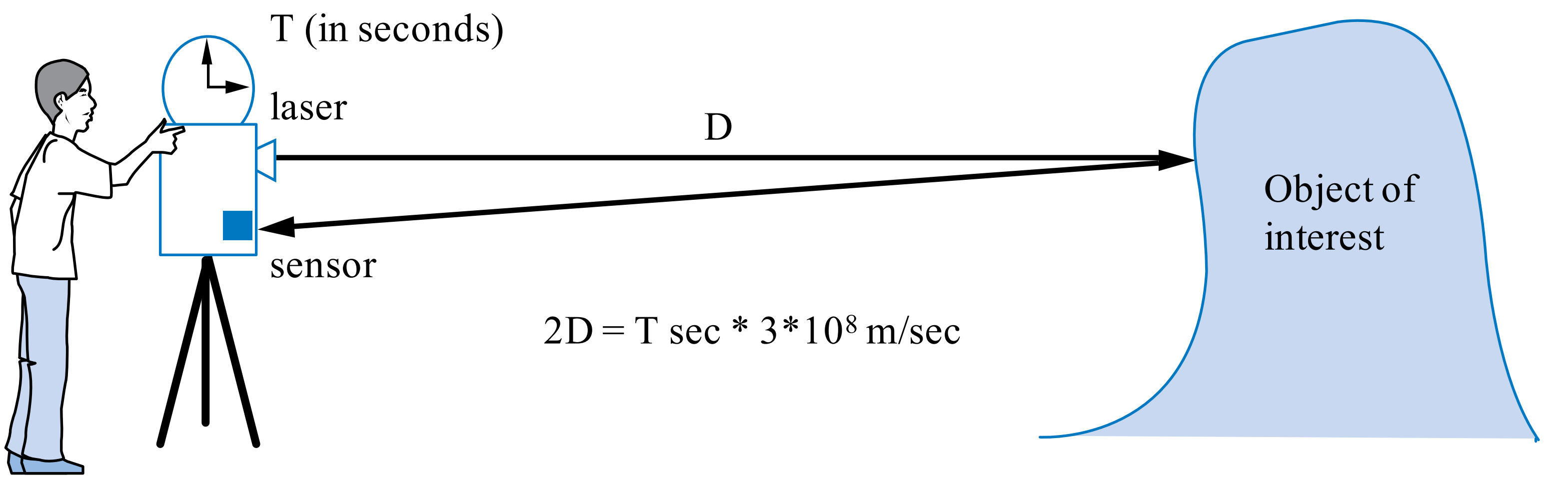
Example: Completing the Design

- Implement the FSM as a state register and logic



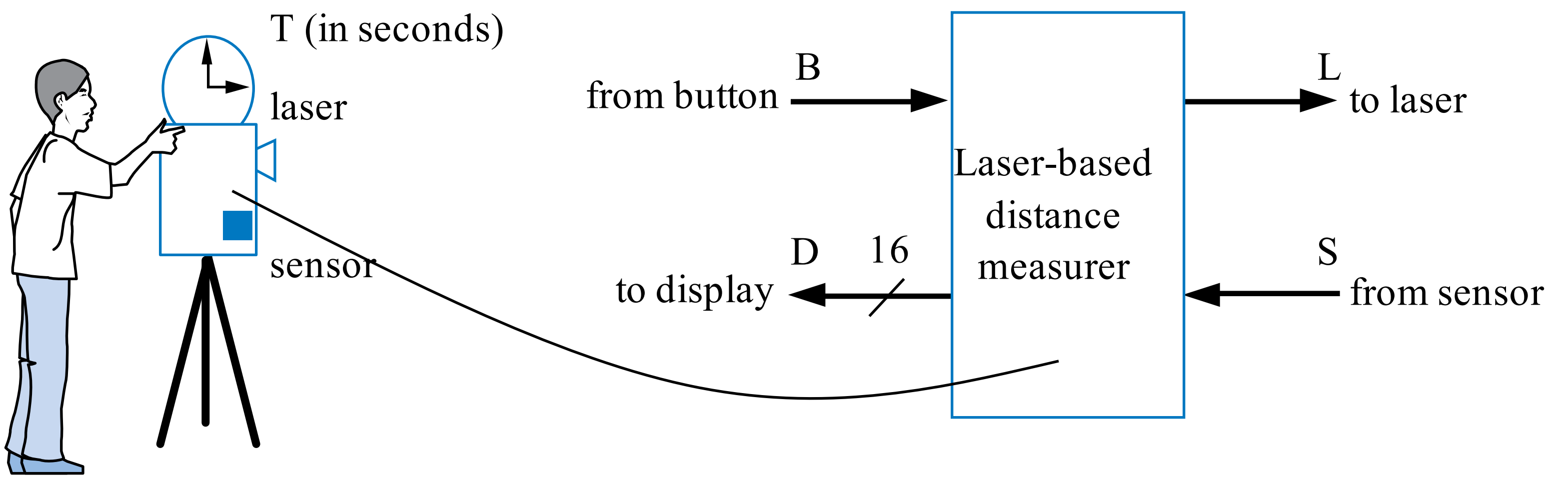
	s1	s0	c	tot_lt_s	n1	n0	d	tot_ld	tot_clr
Init	0	0	0	0	0	1	0	0	1
	0	0	0	1	0	1	0	0	1
	0	0	1	0	0	1	0	0	1
	0	0	1	1	0	1	0	0	1
Wait	0	1	0	0	1	1	0	0	0
	0	1	0	1	0	1	0	0	0
	0	1	1	0	1	0	0	0	0
	0	1	1	1	1	0	0	0	0
Add	1	0	0	0	0	1	0	1	0
		•••				•••			
Disp	1	1	0	0	0	0	1	0	0
		•••				•••			

Example: Laser-Based Distance Measurer



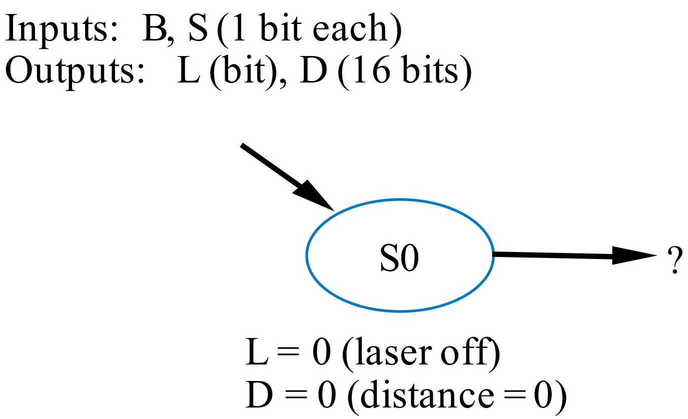
- Laser-based distance measurement – pulse laser, measure time T to sense reflection
  - Laser light travels at speed of light, $3*10^8$ m/sec
  - Distance is thus D = T sec * $3*10^8$ m/sec / 2

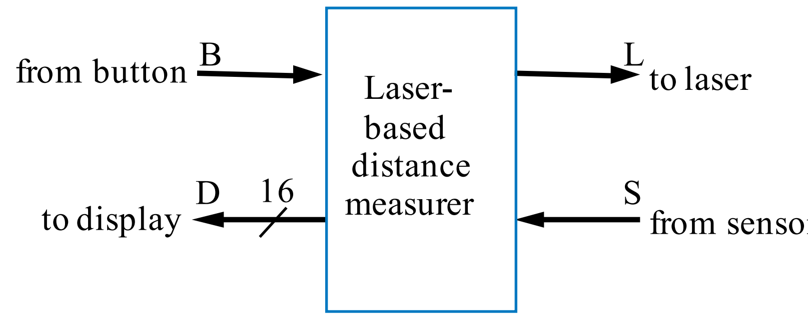
Example: Laser-Based Distance Measurer



- Inputs/outputs
  - *B*: 1-bit input, from button to begin measurement
  - *L*: 1-bit output, activates laser
  - *S*: 1-bit input, senses laser reflection
  - *D*: 16-bit output, displays computed distance

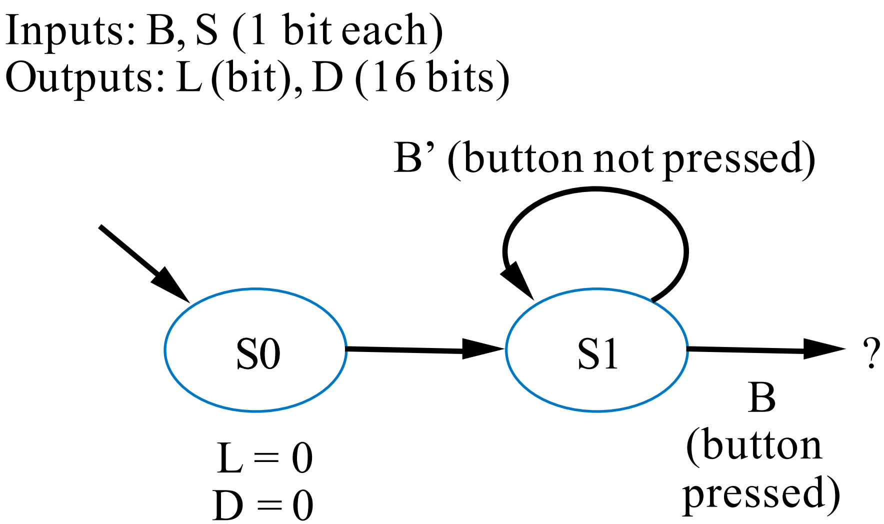
Step 1: Capture High-Level State Machine

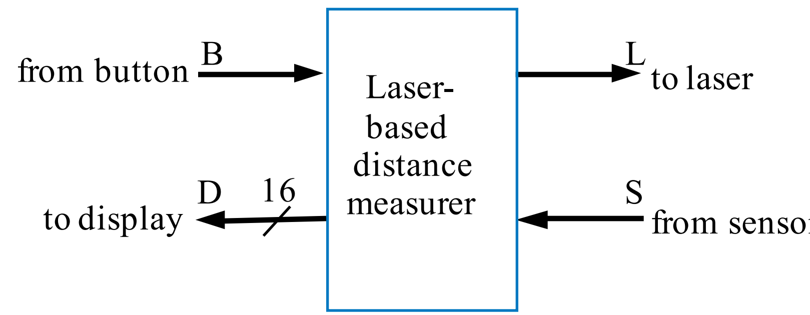




- Step 1: Create high-level state machine
  - Begin by declaring inputs and outputs
  - Create initial state, name it **S0**
    - Initialize laser to off (L=0)
    - Initialize displayed distance to 0 (D=0)

Step 1: Capture High-Level State Machine



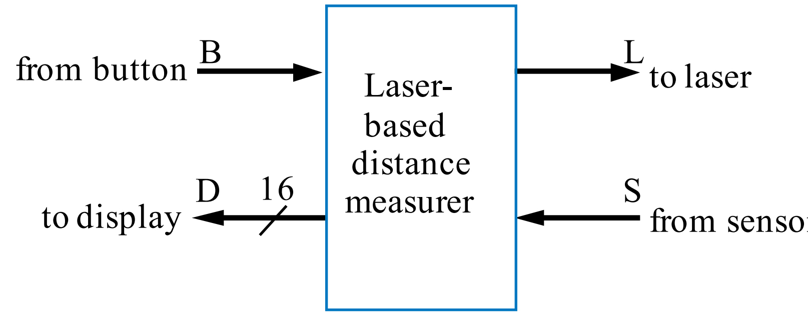


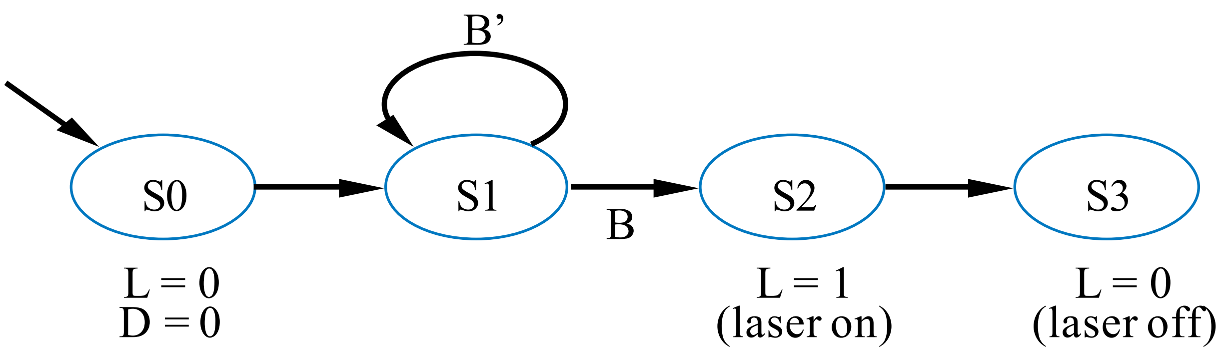
- Add another state, call **S1**, that waits for a button press
  - B’ – stay in **S1**, keep waiting
  - B – go to a new state **S2**

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

Step 1: Capture High-Level State Machine

Inputs: B, S (1 bit each)
Outputs: L (bit), D (16 bits)





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

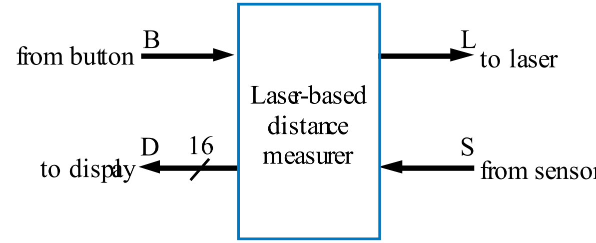
Q: What do next? A: Start timer, wait to sense reflection

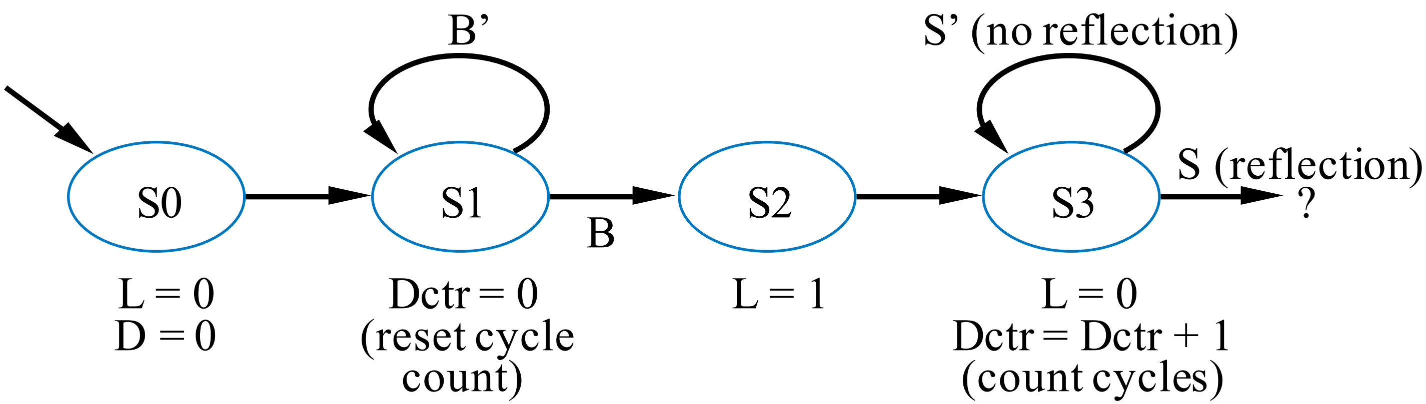
a

Step 1: Capture High-Level State Machine

Inputs: B, S (1 bit each)   Outputs: L (bit), D (16 bits)

Local Registers: Dctr (16 bits)

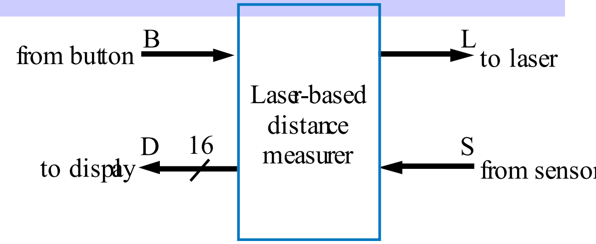




- Stay in **S3** until sense reflection (S)
- To measure time, count cycles for which we are in **S3**
  - To count, declare local register *Dctr*
  - Increment *Dctr* each cycle in **S3**
  - Initialize *Dctr* to 0 in **S1**. **S2** would have been O.K. too

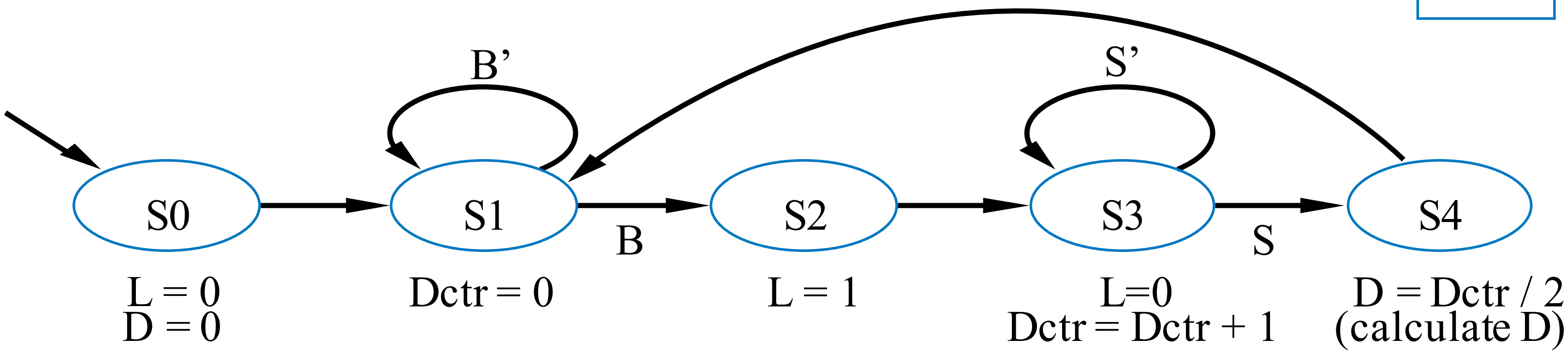
Q: What do next? A: Stop timer, calculate distance

Step 1: Capture High-Level State Machine



Inputs: B, S (1 bit each) Outputs: L (bit), D (16 bits)

Local Registers: Dctr (16 bits)



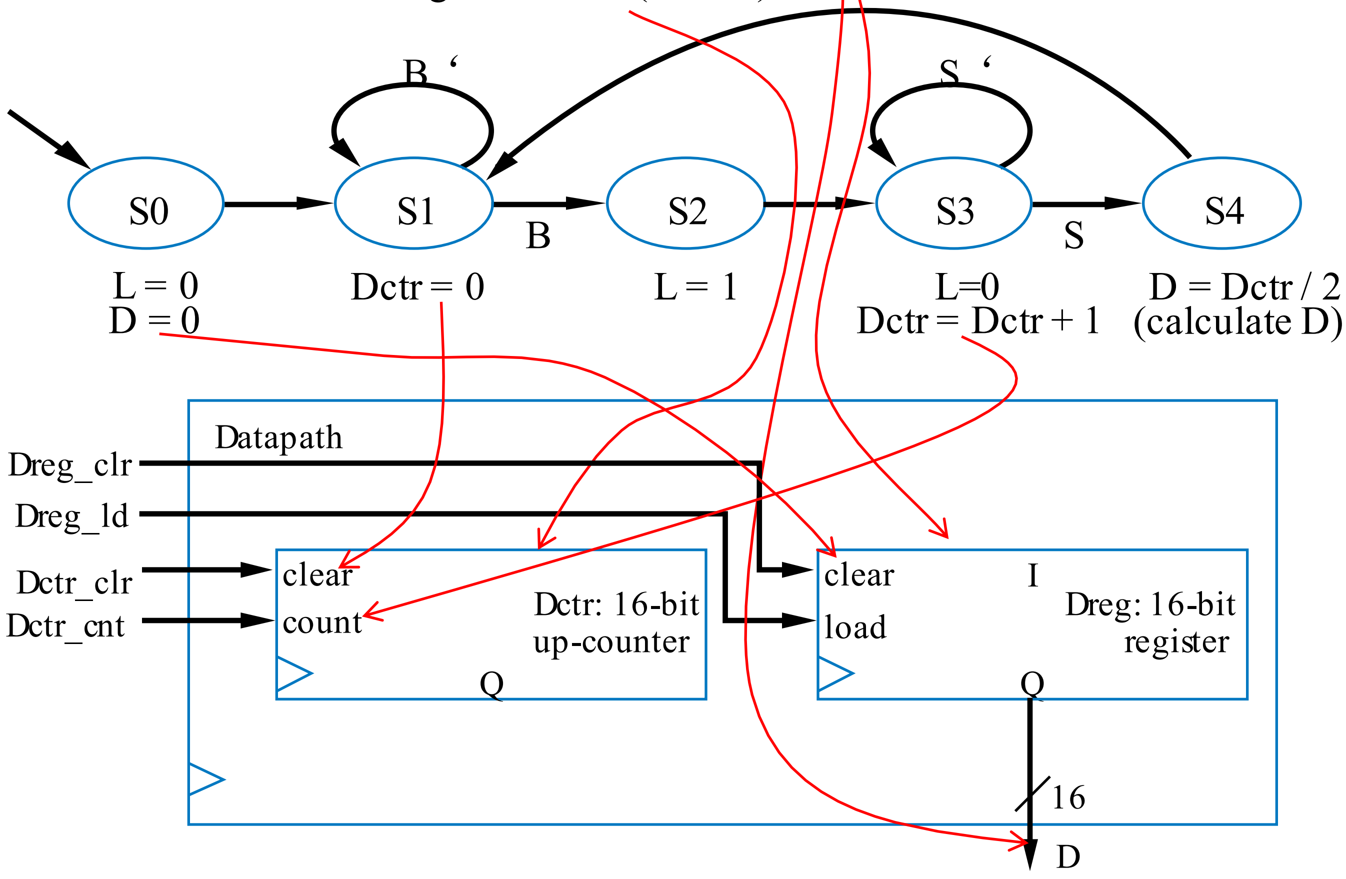
- Once reflection detected (S), go to new state **S4**
  - Calculate distance
  - Assuming clock frequency is $3x10^8$, *Dctr* holds number of meters, so D=Dctr/2
- After **S4**, go back to **S1** to wait for button again

Step 2: Create a Datapath

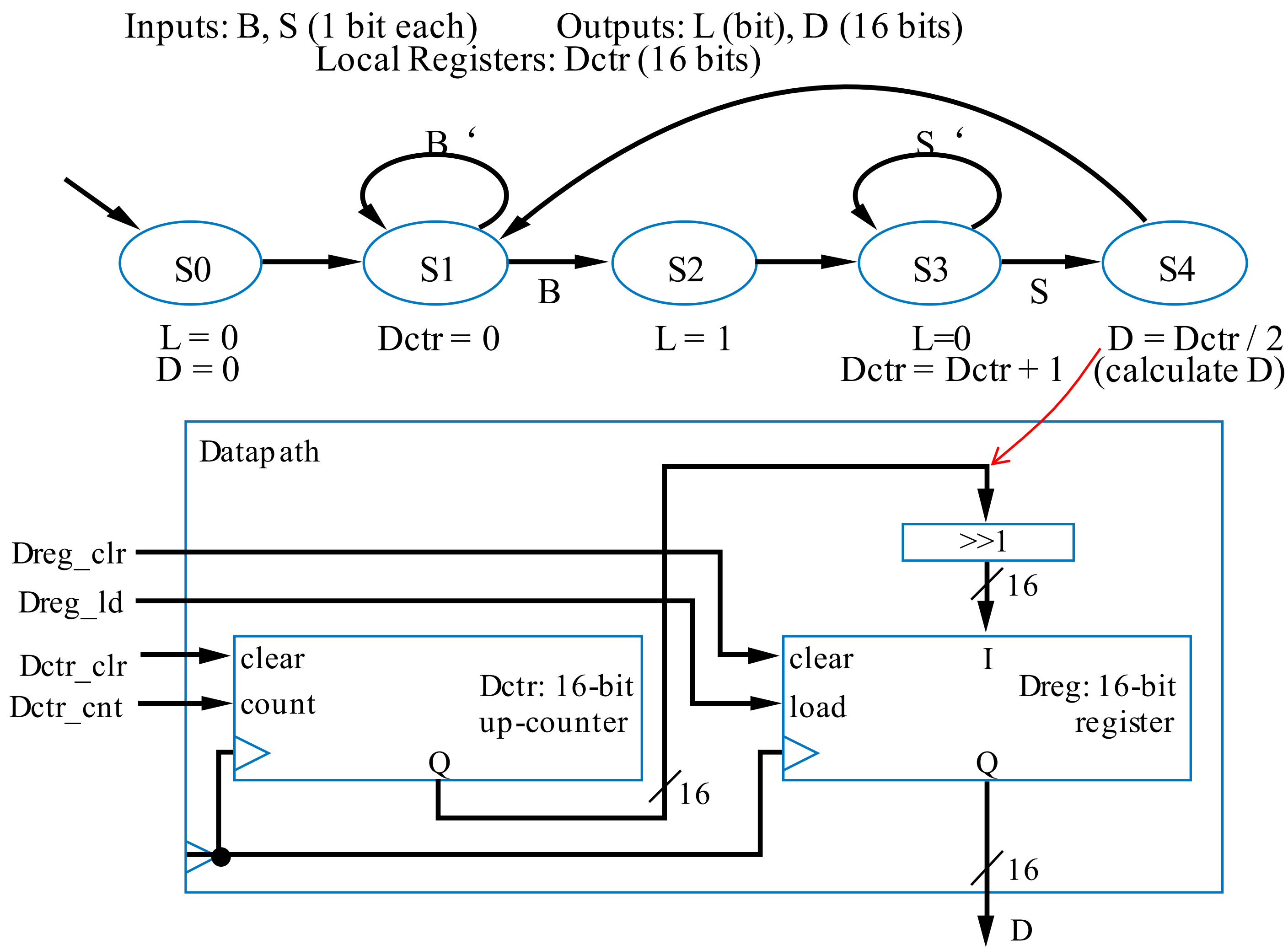
- Datapath must
  - Implement data storage
  - Implement data computations
- Look at high-level state machine, instantiate required components

Step 2: Create a Datapath

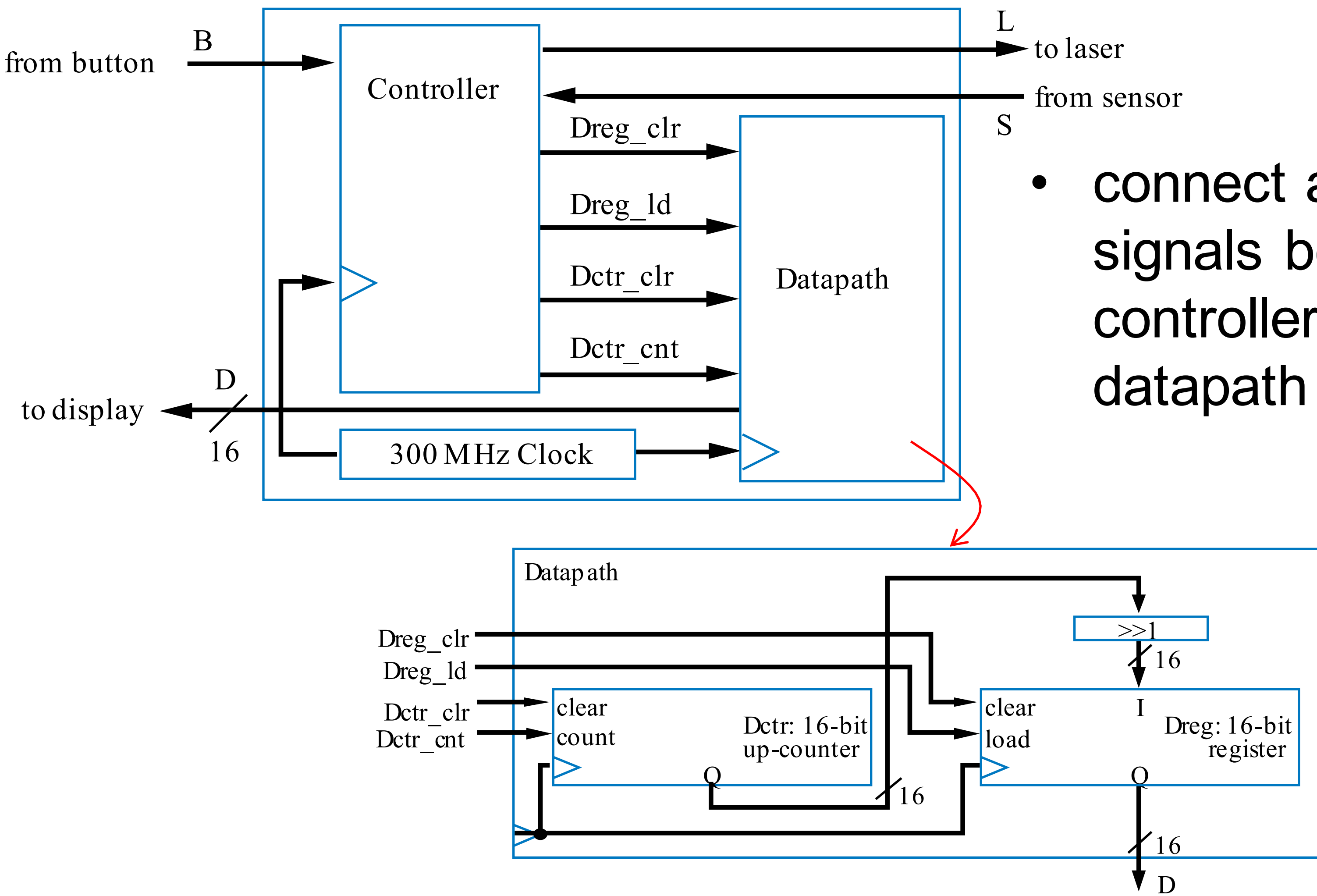
Inputs: B, S (1 bit each) Outputs: L (bit), D (16 bits)
Local Registers: Dctr (16 bits)



Step 2: Create a Datapath

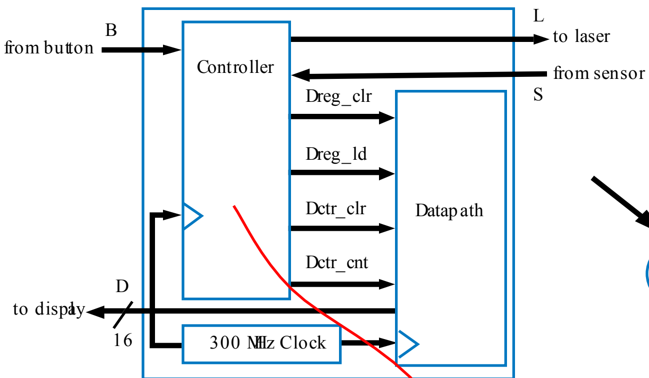


Step 3: Connecting the Datapath to a Controller

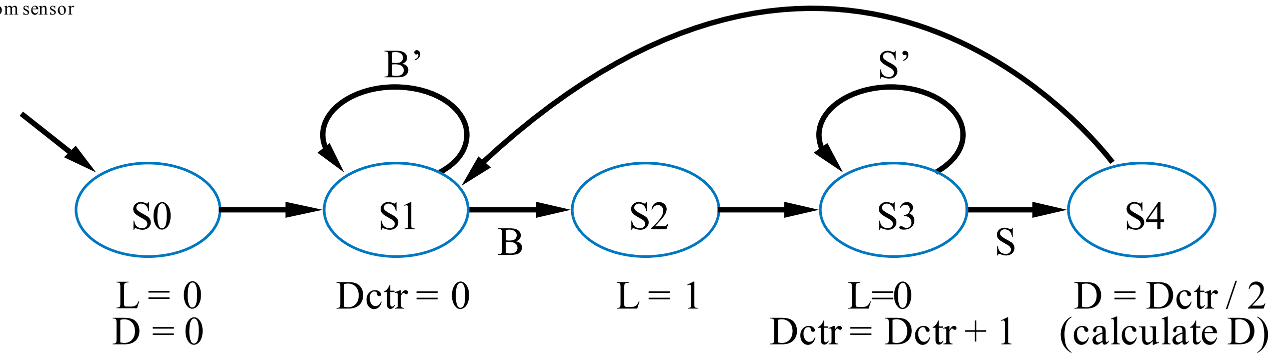


- connect all control signals between controller and datapath

Step 4: Deriving the Controller’s FSM

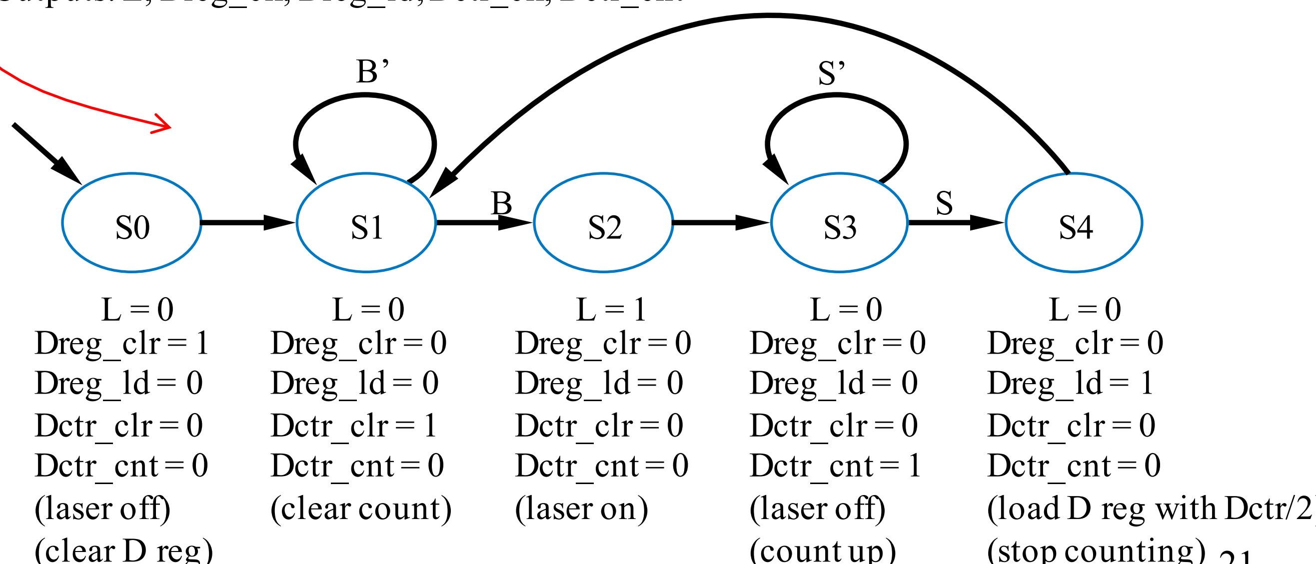


Inputs: B, S (1 bit each) Outputs: L (bit), D (16 bits)
Local Registers: Dctr (16 bits)

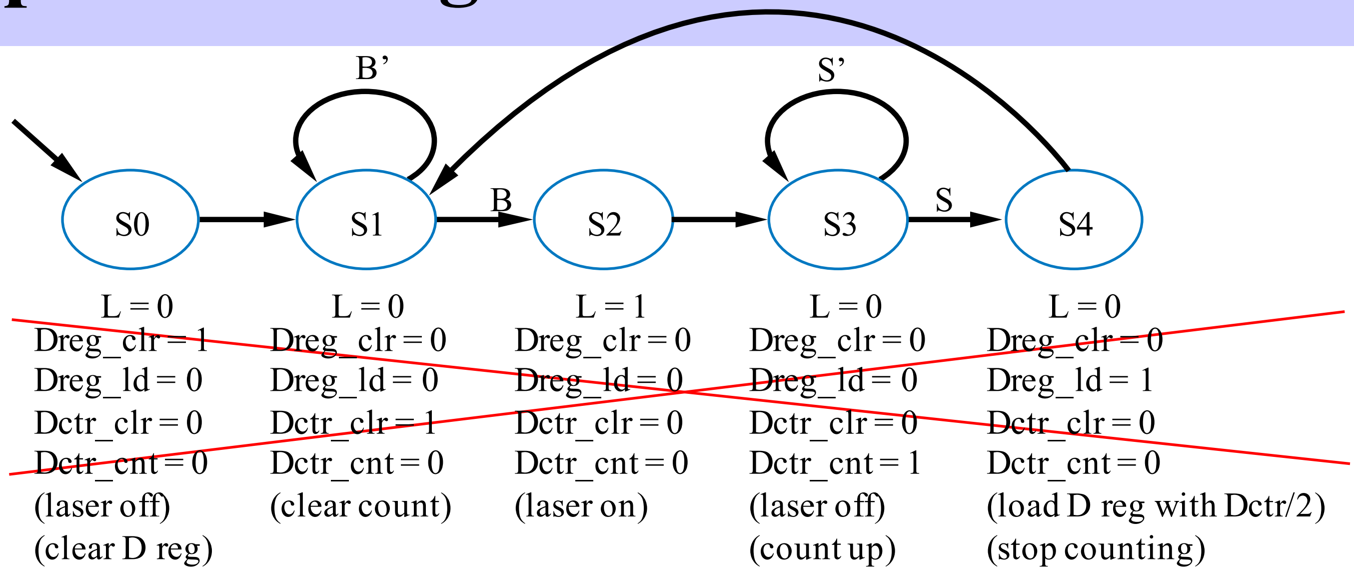


- Replace data operations by bit operations using datapath

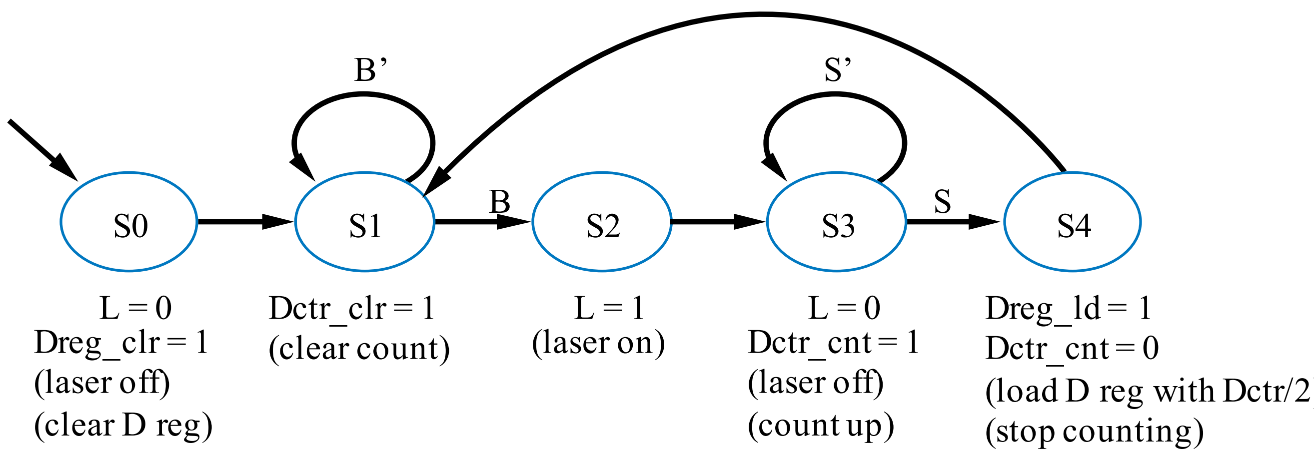
Inputs: B, S
Outputs: L, Dreg_clr, Dreg_ld, Dctr_clr, Dctr_cnt



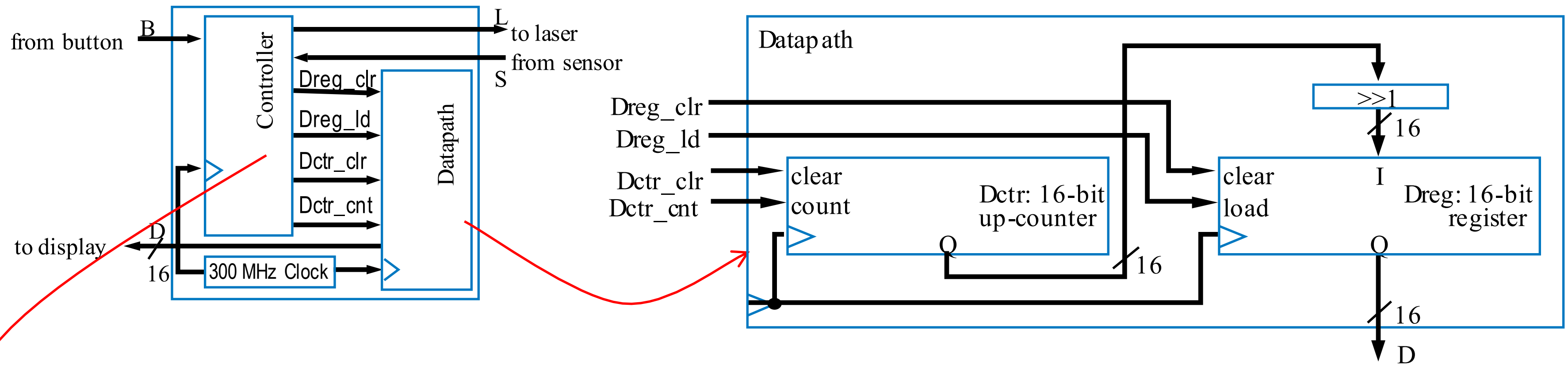
Step 4: Deriving the Controller's FSM



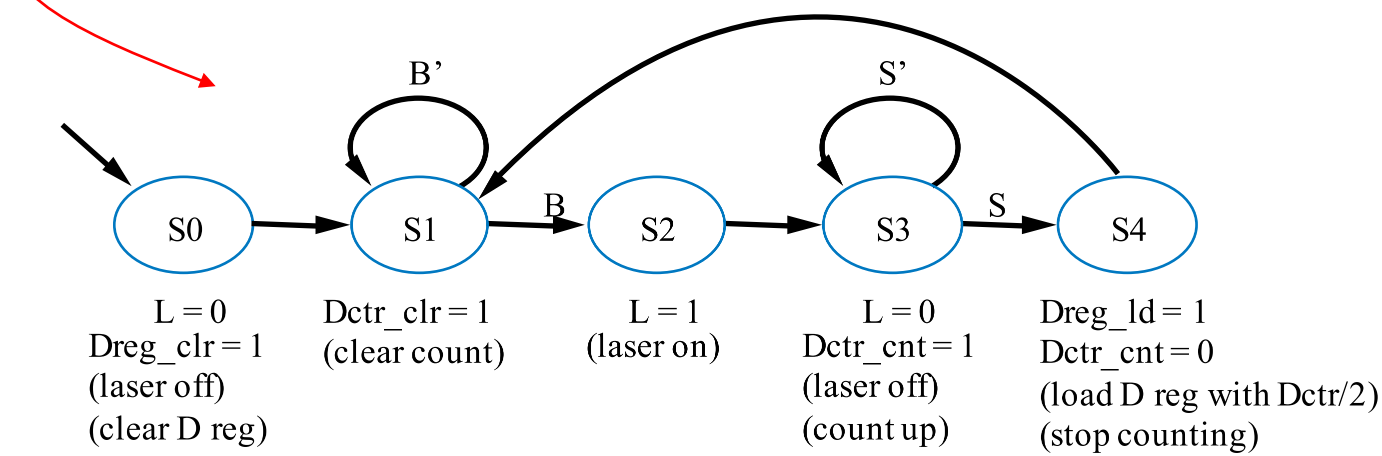
Inputs: B, S   Outputs: L, Dreg_clr, Dreg_ld, Dctr_clr, Dctr_cnt



Step 4: Deriving the Controller's FSM

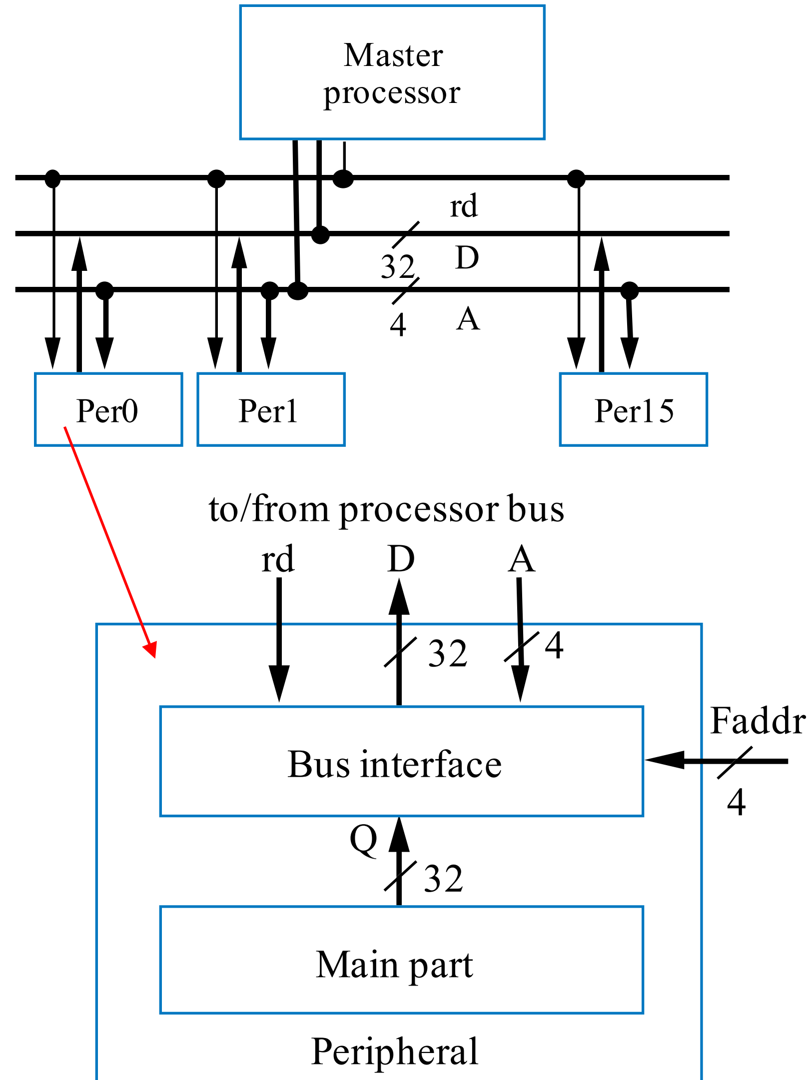


Inputs: B, S   Outputs: L, Dreg_clr, Dreg_ld, Dctr_clr, Dctr_cnt

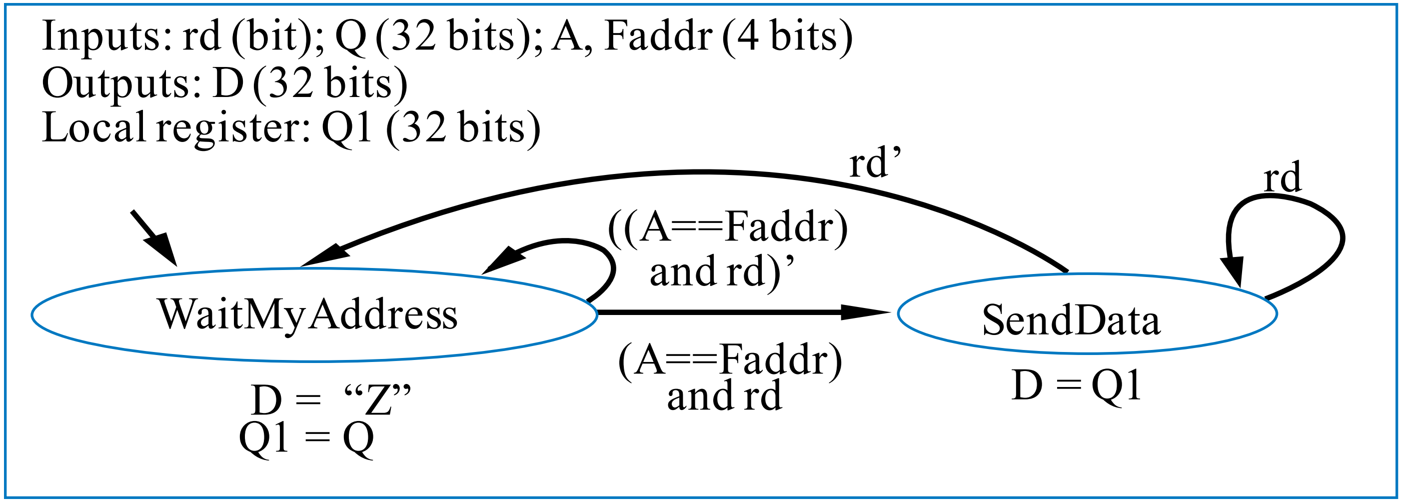


RTL Example: Bus Interface

- Example: **Bus interface**
  - Master processor can read register from any peripheral
    - Each register has unique 4-bit address
    - Assume 1 register/peripheral
  - Sets *rd=1*, *A=address*
  - Appropriate peripheral places register data on 32-bit *D* lines
    - Peripheral's address provided on *Faddr* inputs (maybe from DIP switches, or another register)

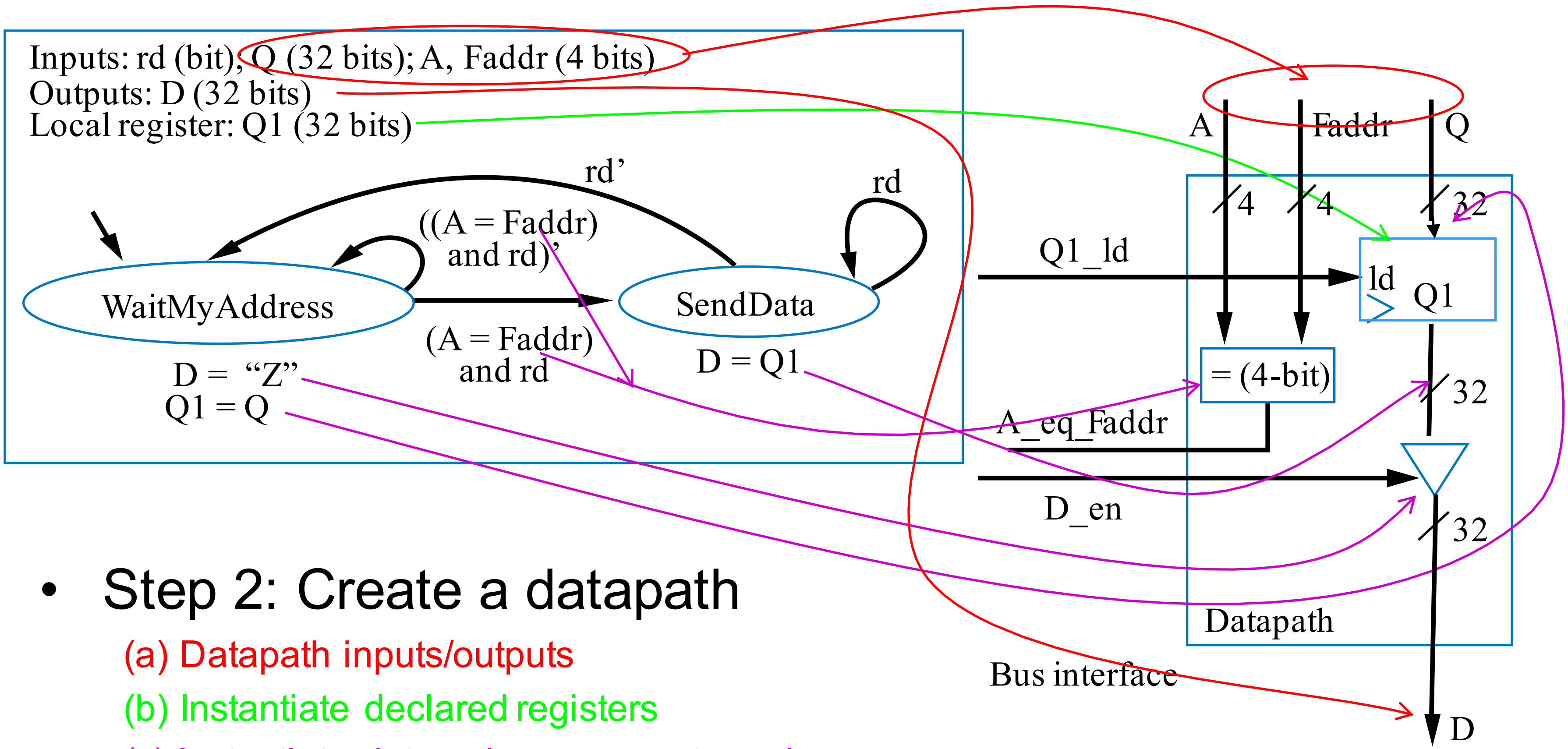


RTL Example: Bus Interface



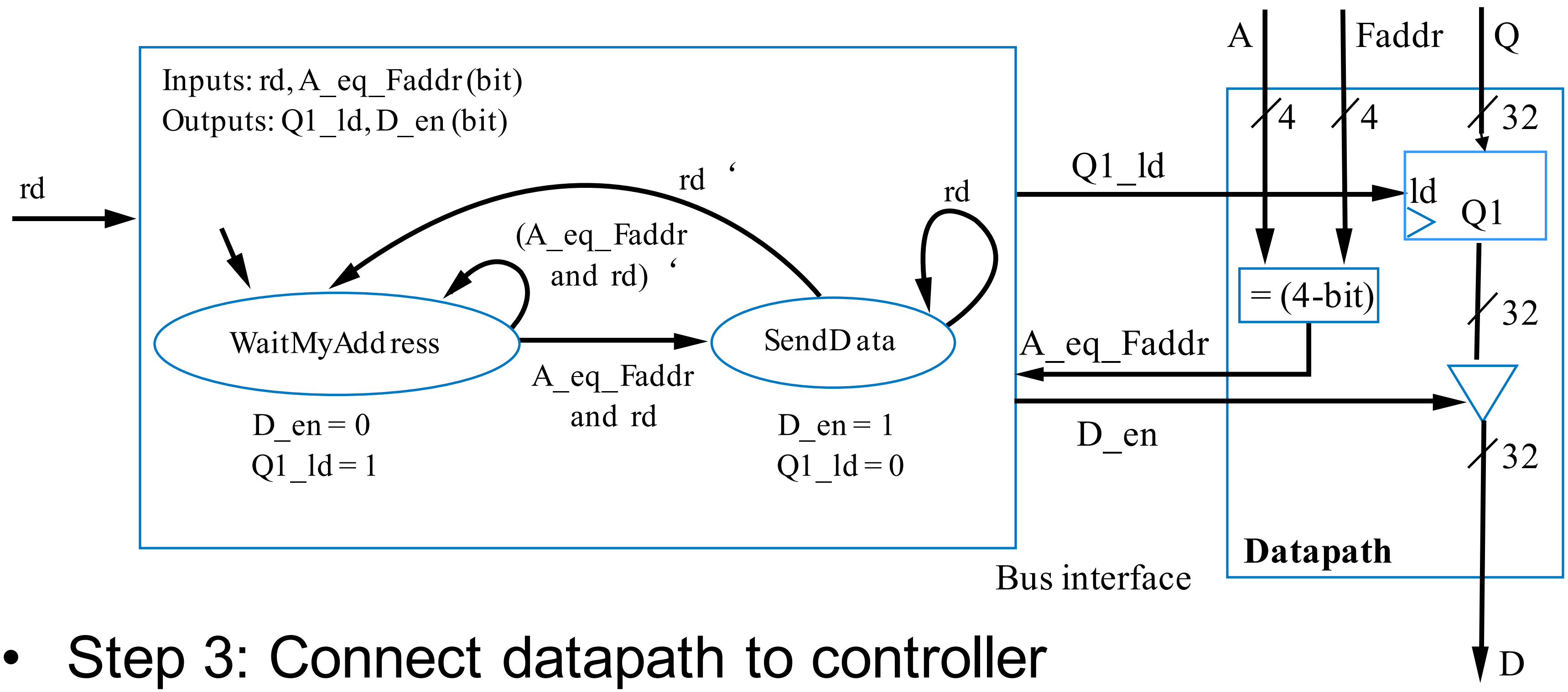
- Step 1: Create high-level state machine
  - State **WaitMyAddress**
    - Output “nothing” (“Z”) on *D*, store peripheral’s register value *Q* into local register *Q1*
    - Wait until this peripheral’s address is seen (*A==Faddr*) and *rd=1*
  - State **SendData**
    - Output *Q1* onto *D*, wait for *rd=0* (meaning main processor is done reading the *D* lines)

RTL Example: Bus Interface



- Step 2: Create a datapath
  - (a) Datapath inputs/outputs
  - (b) Instantiate declared registers
  - (c) Instantiate datapath components and connections

RTL Example: Bus Interface

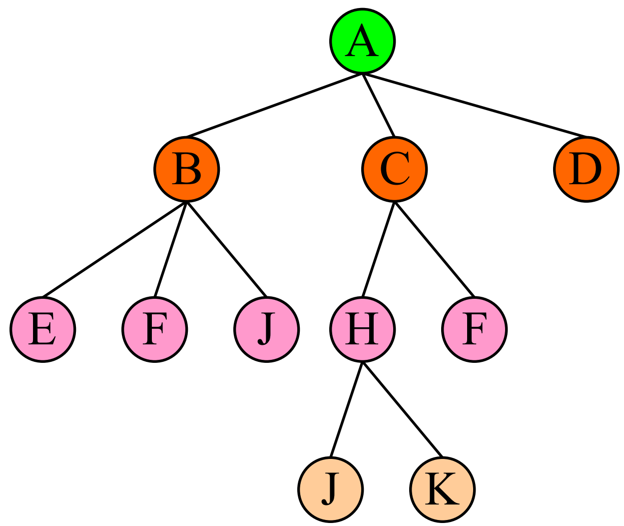


- Step 3: Connect datapath to controller
- Step 4: Derive controller's FSM

Handle the Complexity with Hierarchical Design

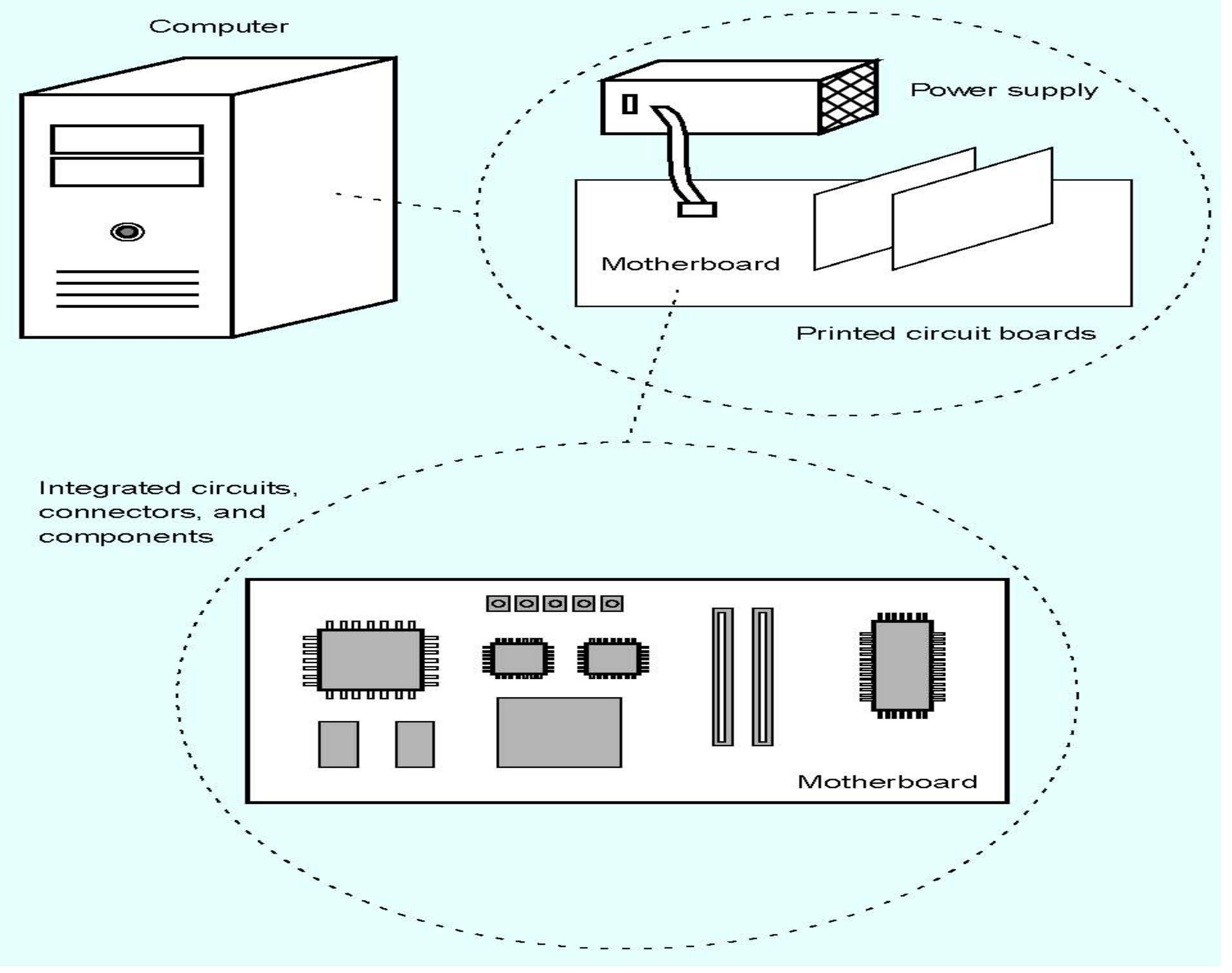
- Hierarchical design structure at different levels of abstraction
- Levels of abstraction: hiding the details in lower levels

•Hide details
•Reuse subsystems

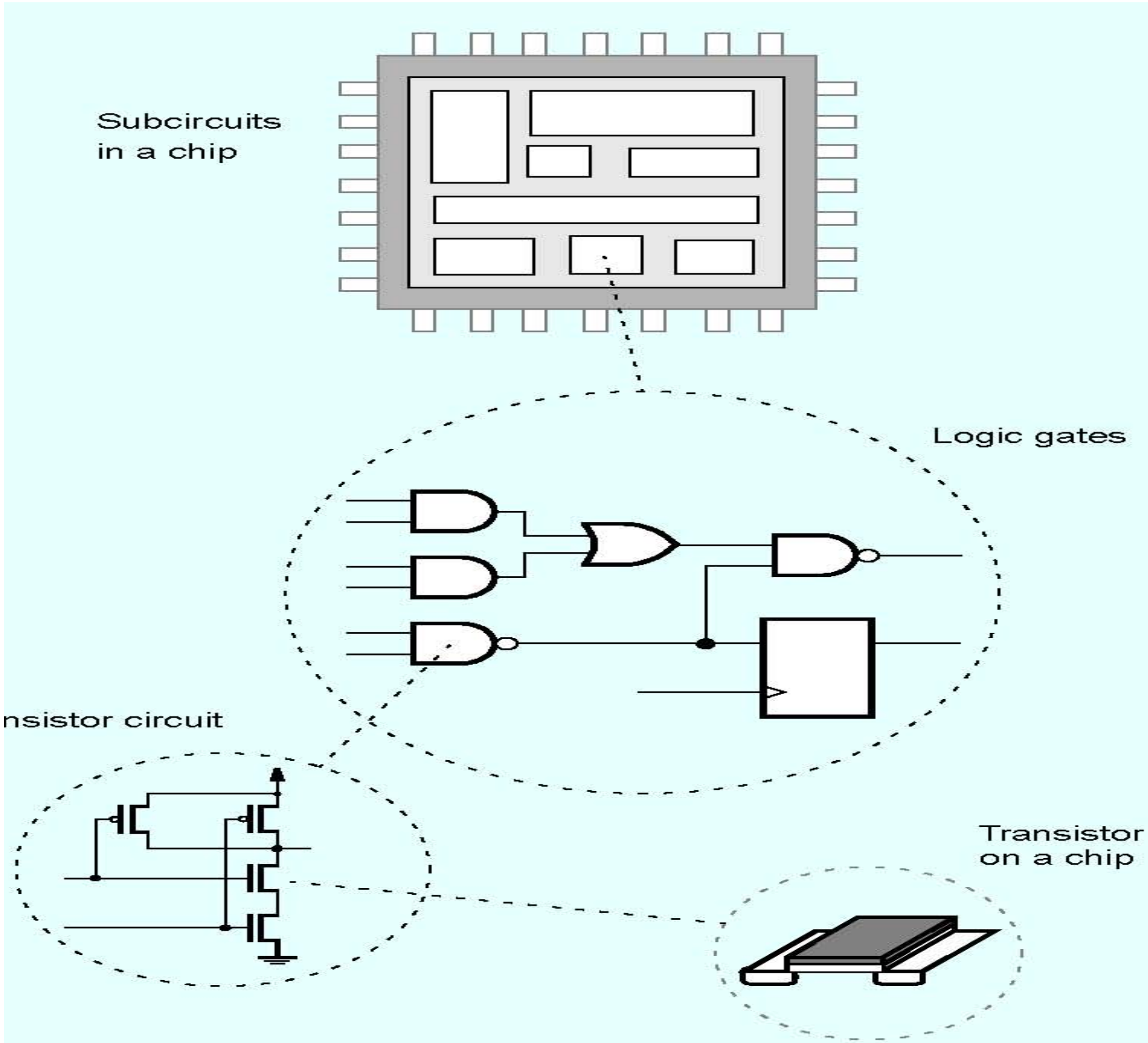


Hierarchical Structure of a Design

Hierarchical Digital System



Hierarchical Digital System



Pacemaker

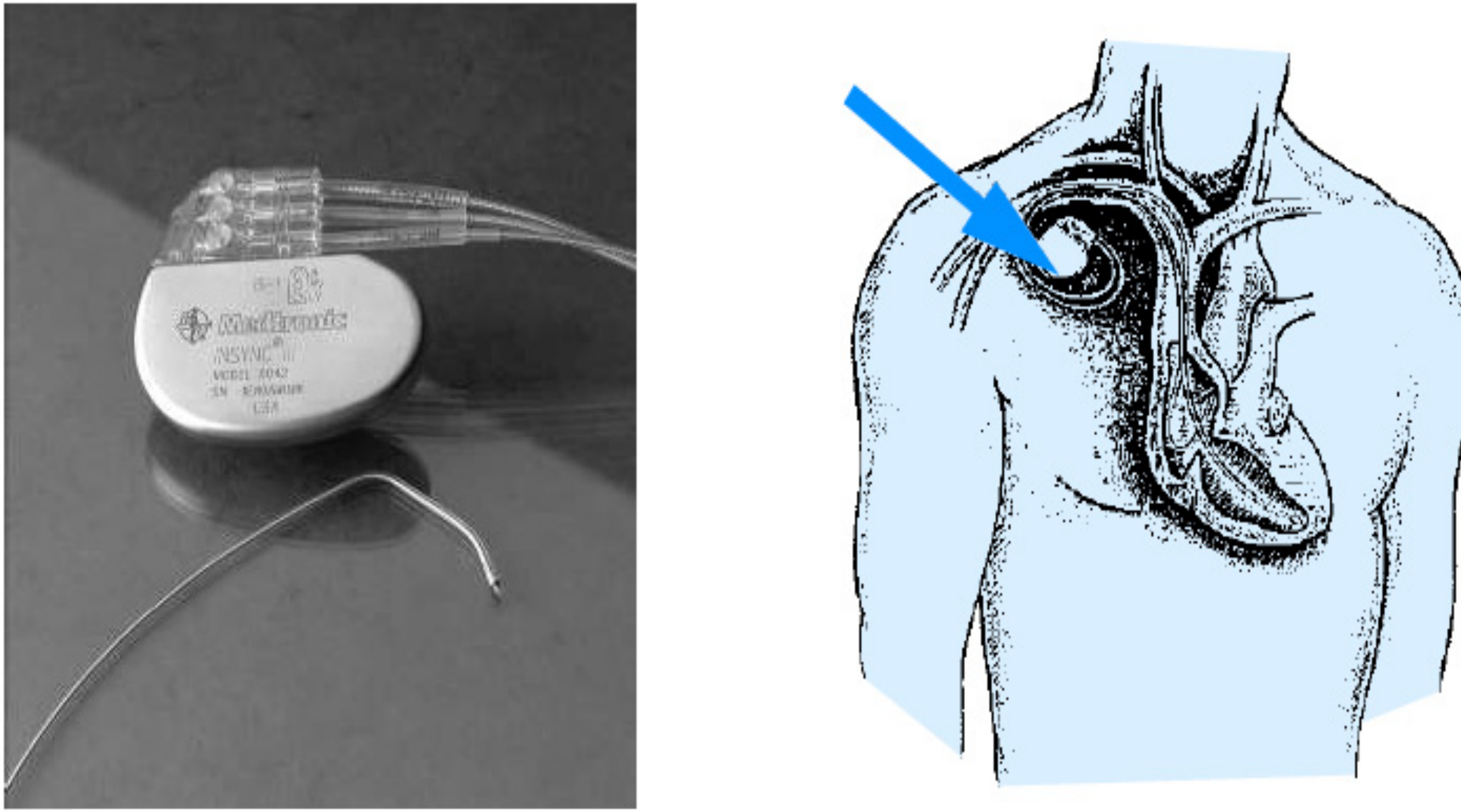
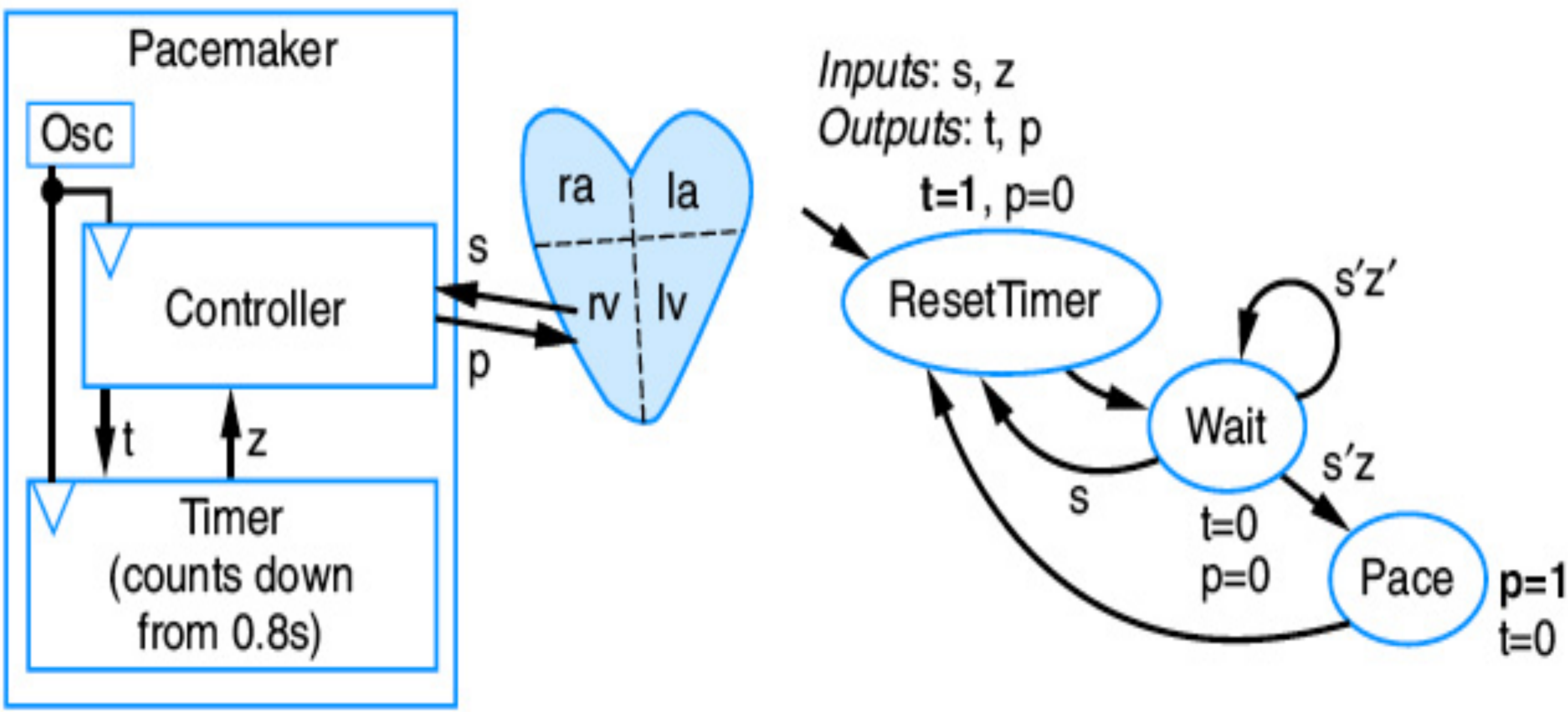
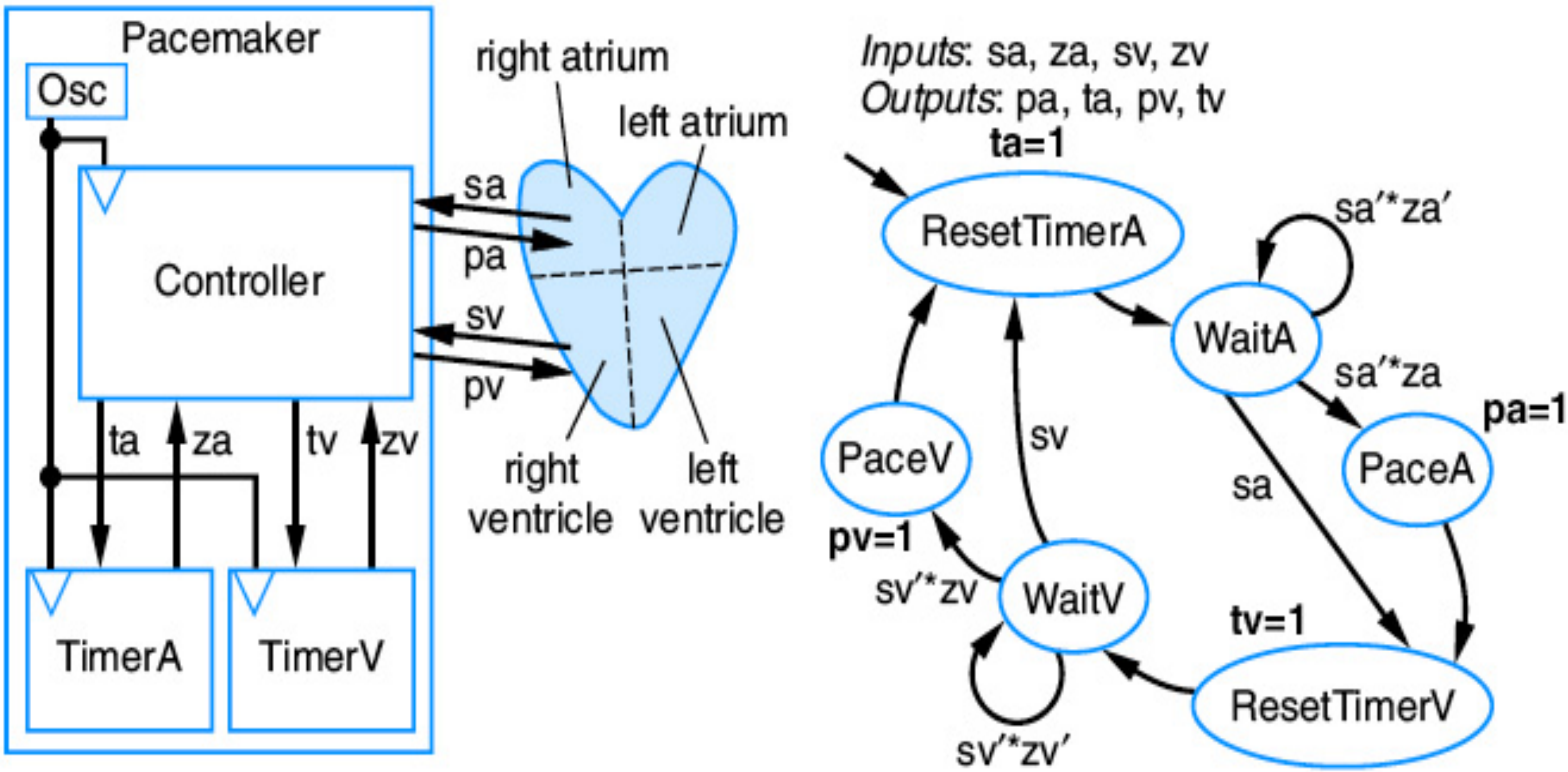


Figure 3.68 Pacemaker with leads (left), and pacemaker's location under the skin (right). Courtesy of Medtronic, Inc.

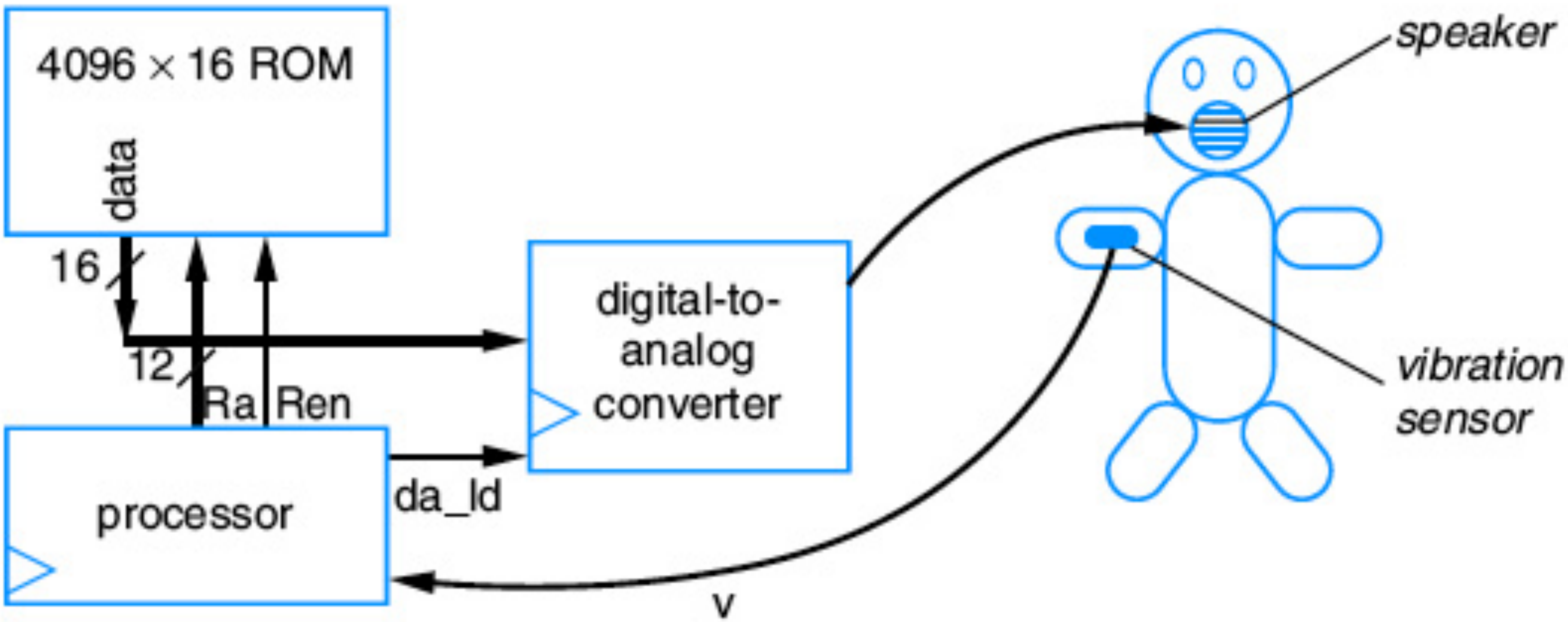


Pacemaker
Osc
Controller
Timer
(counts down
from 0.8s)
t
z
s
p
ra
la
rv
lv
Inputs: s, z
Outputs: t, p
t=1, p=0
ResetTimer
s′z′
Wait
s′z
s
t=0
p=0
Pace
p=1
t=0



Pacemaker
Osc
Controller
TimerA
TimerV
ta
za
tv
zv
sa
pa
sv
pv
right atrium
left atrium
right ventricle
left ventricle
Inputs: sa, za, sv, zv
Outputs: pa, ta, pv, tv
ta=1
ResetTimerA
sa′*za′
WaitA
sa′*za
pa=1
PaceA
sa
sv
PaceV
pv=1
sv′*zv
WaitV
tv=1
ResetTimerV
sv′*zv′

Talking Doll



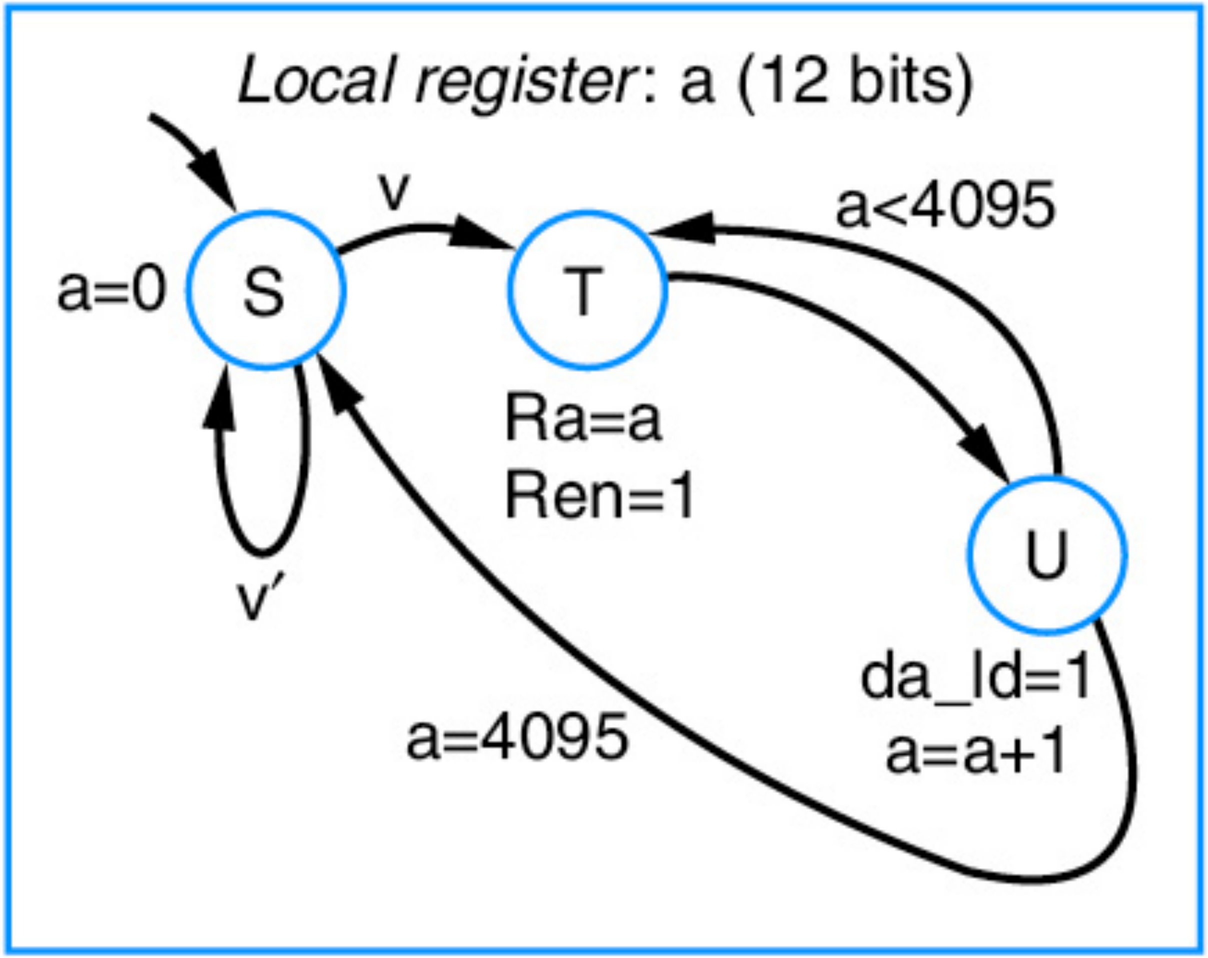
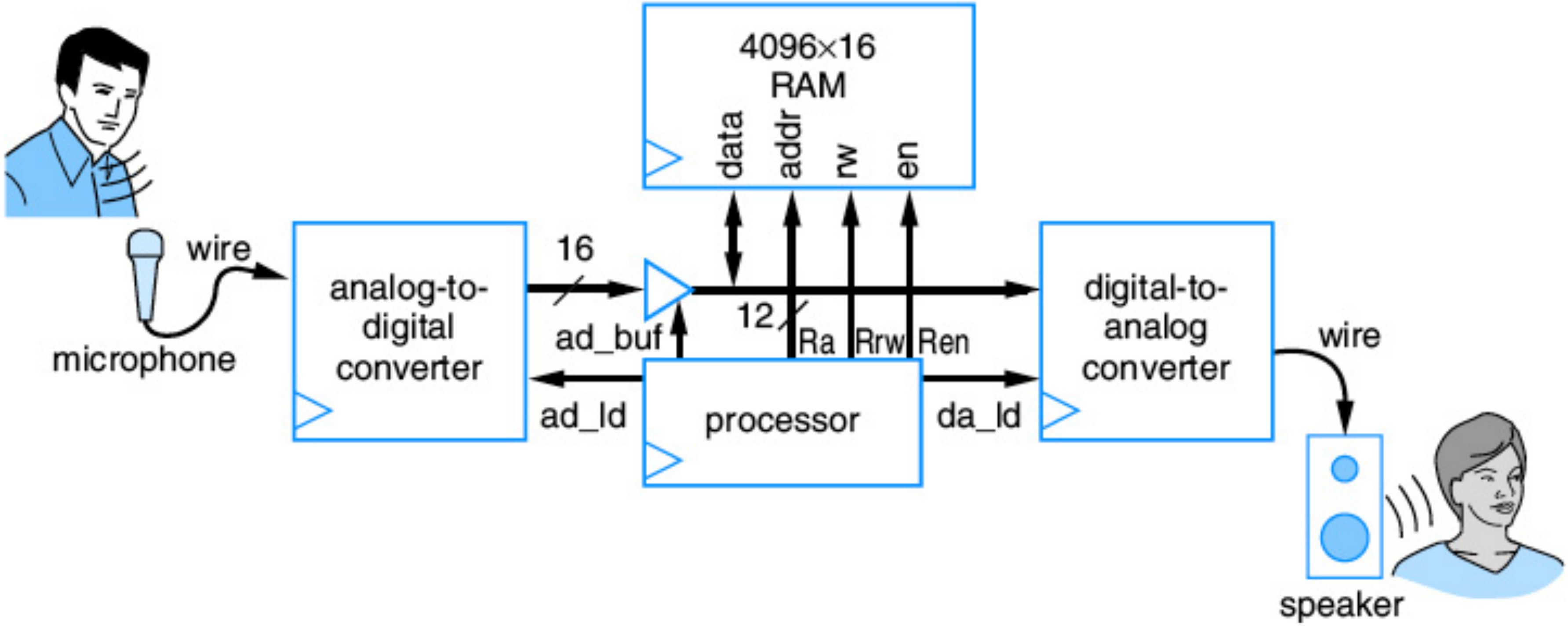


Figure 5.72 State machine for reading the ROM.

Answering Machine



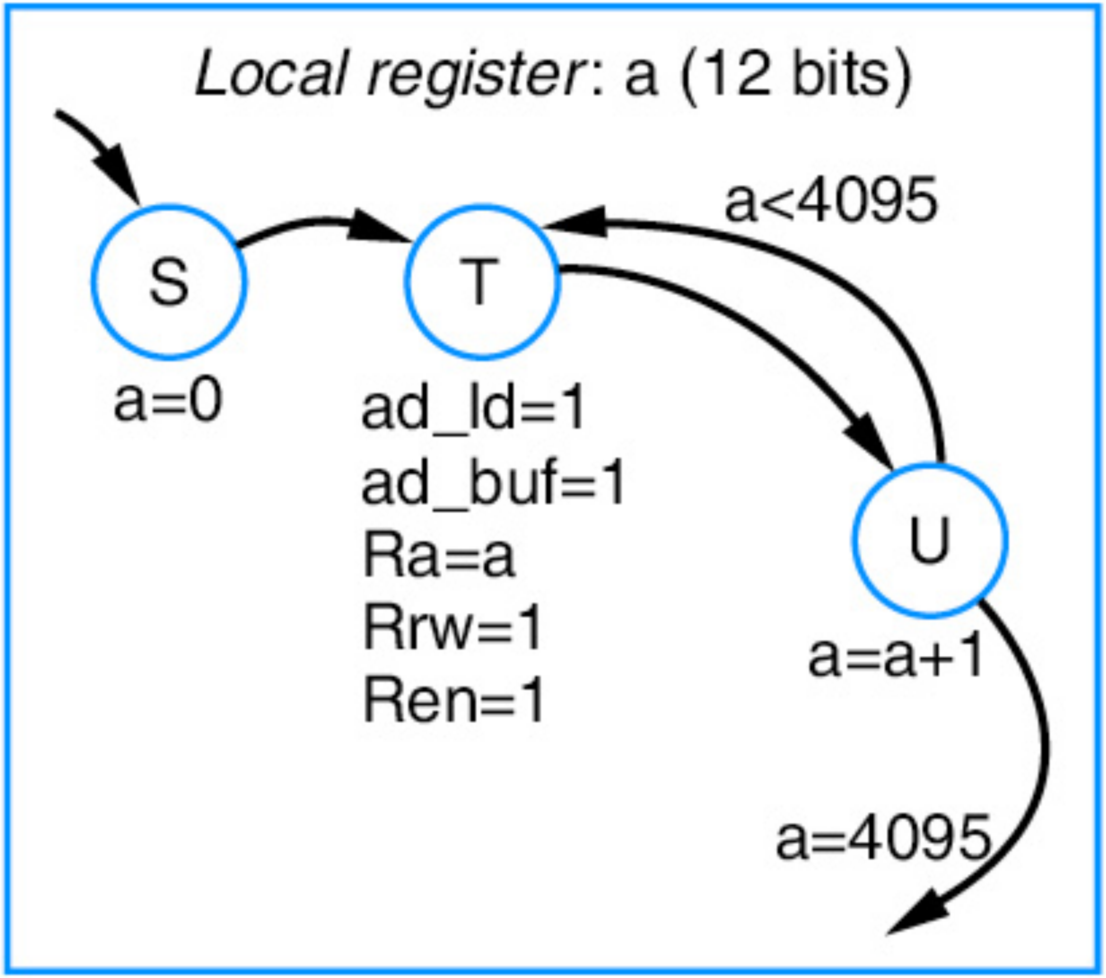


Figure 5.62 State machine for storing digitized sound in RAM.

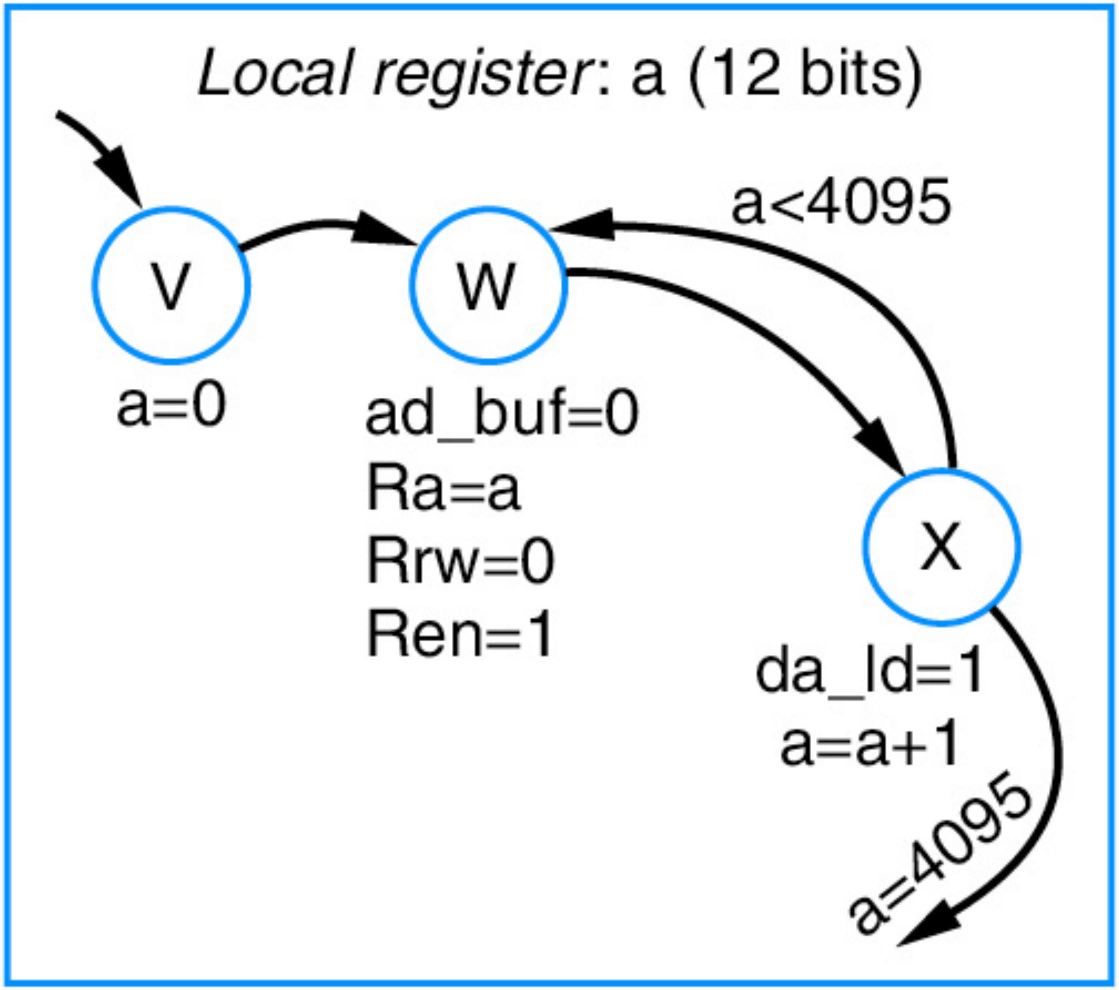


Figure 5.63 State machine for playing sound from the RAM.