B38DB: Digital Design and Programming
Datapath Components – Registers

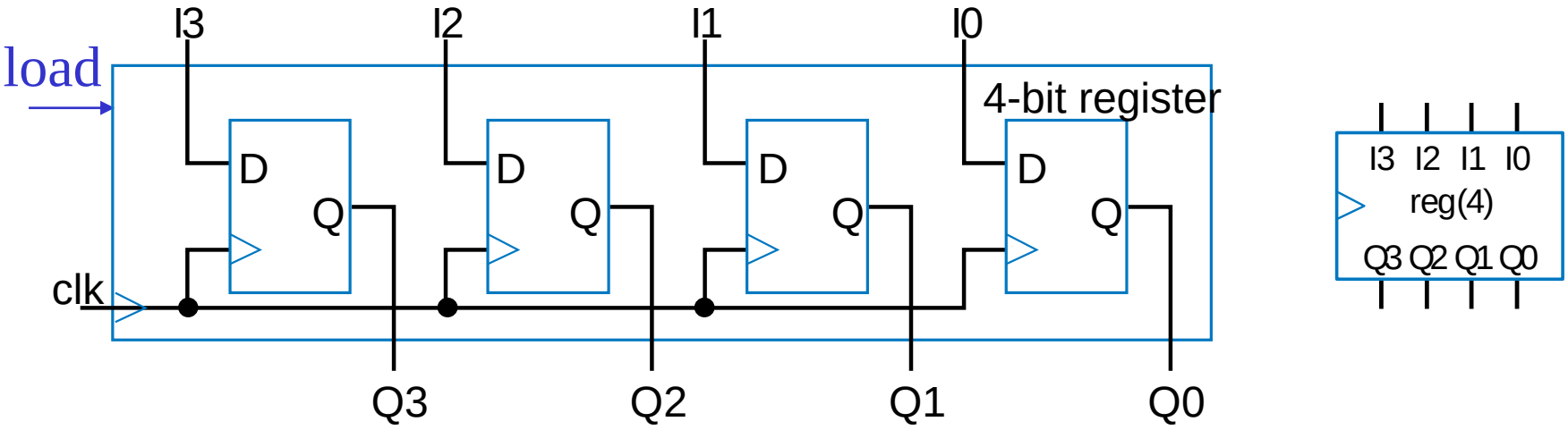
Mustafa Suphi Erden

Heriot-Watt University
School of Engineering & Physical Sciences
Electrical, Electronic and Computer Engineering
Room: EM 2.01
Phone: 0131-4514159
E-mail: m.s.erden@hw.ac.uk

Introduction

- Up to now we learned about increasingly complex digital building blocks
  - Gates, multiplexors, decoders, basic registers, and controllers
- Controllers are good for systems with control inputs/outputs
  - ***Control*** input: Single bit (or just a few), representing environment event or state
    - e.g., 1 bit representing button pressed
  - ***Data*** input: Multiple bits collectively representing single entity
    - e.g., 7 bits representing temperature in binary
- We need building blocks for data processing
  - ***Datapath components***, in other words ***register-transfer-level (RTL) components***, to store/transform data
    - Put datapath components together to form a ***datapath***

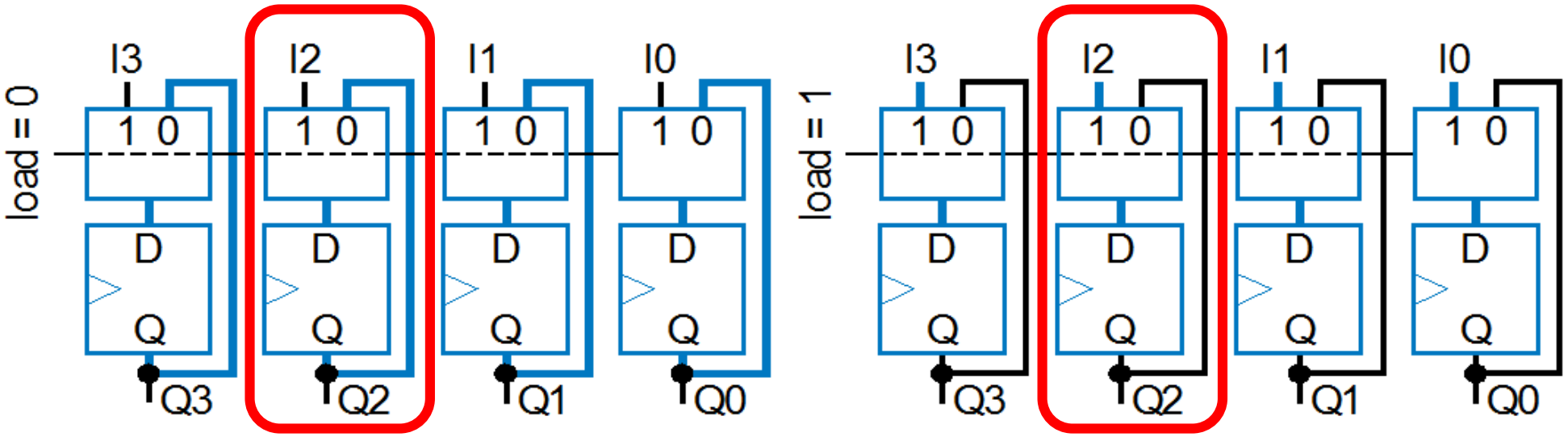
Registers



Basic register loads on every clock cycle.

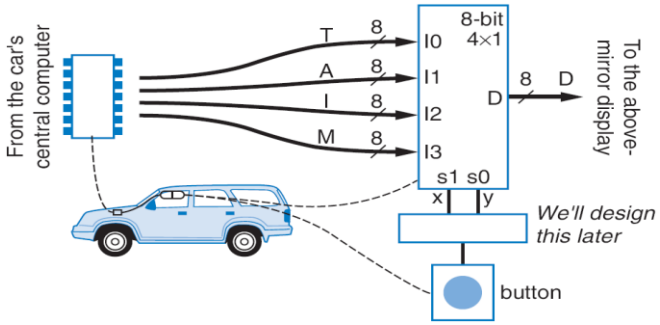
How to control the loading to happen only when it is desired?

Register with Parallel Load

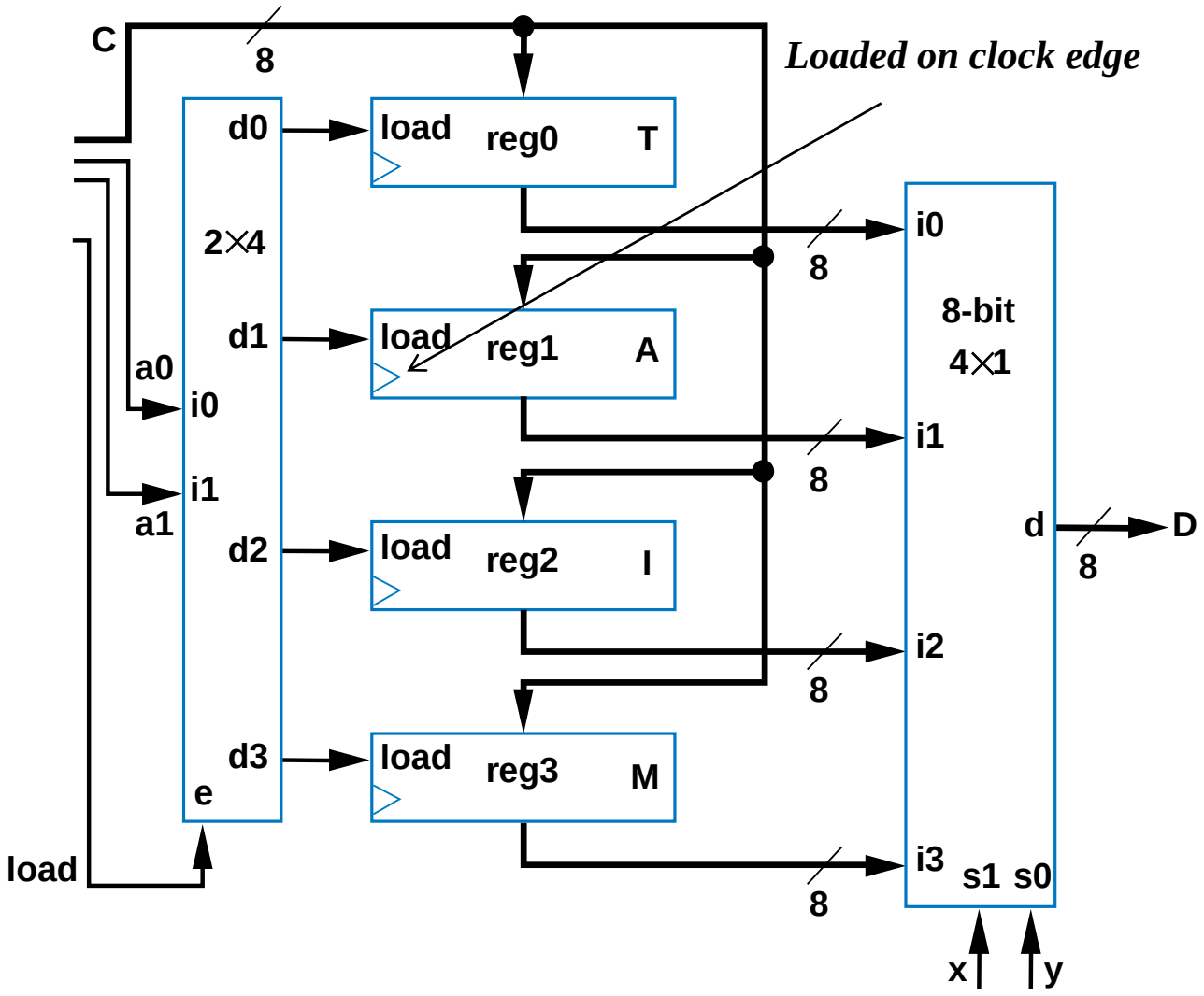


- Add 2x1 Mux to the front of each flip-flop
- Register's load input selects Mux input to pass
  - Either the existing flip-flop value, or a new value is loaded

Register Example: Above-Mirror Display

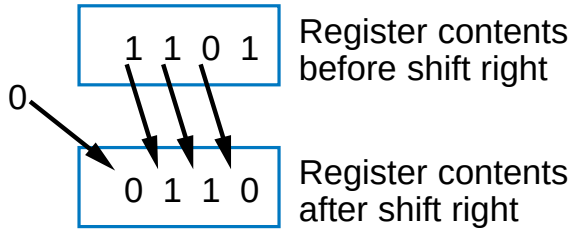


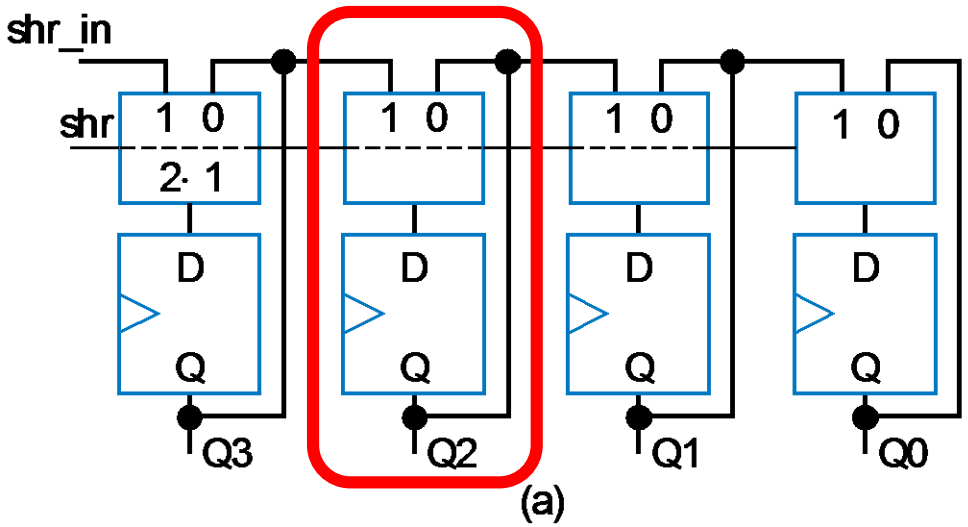
- Initial design: Four simultaneous values from car's computer
- To reduce wires: Computer writes only 1 value at a time, loads into one of the four registers
  - Was: 8+8+8+8 = 32 wires
  - Now: **8 +2+1 = 11** wires

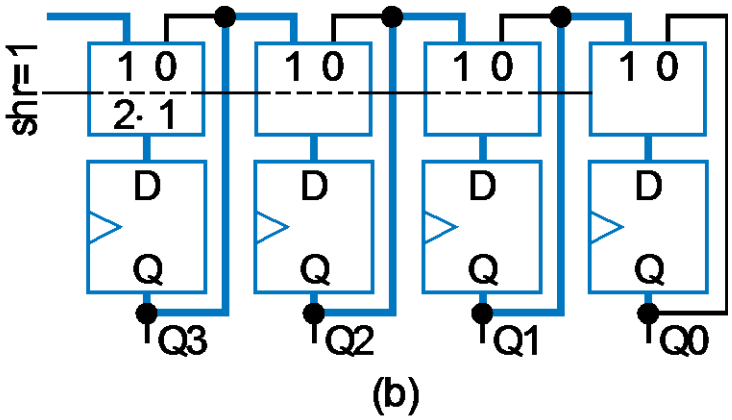


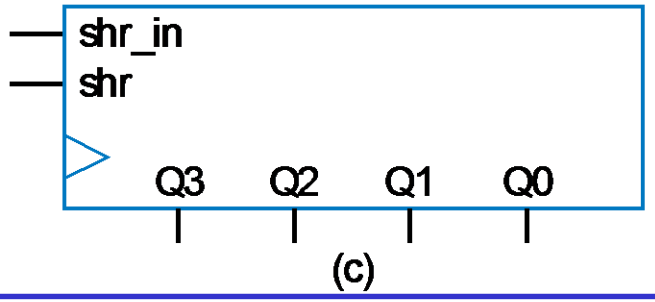
Shift Register

- Shift right
  - Move each bit one position right
  - Shift in 0 to leftmost bit



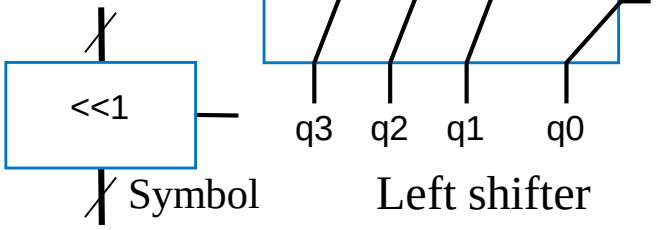


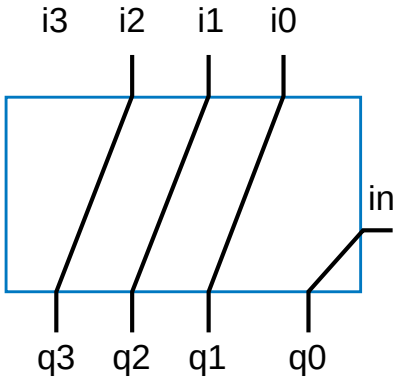




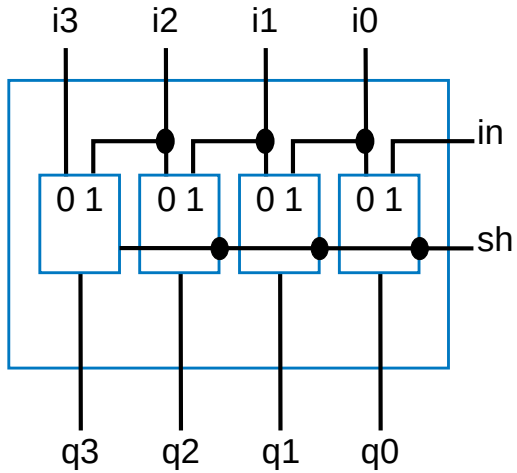
Shifters

- Shifting (e.g., left shifting 0011 yields 0110) useful for:
  - Manipulating bits
  - Converting serial data to parallel (remember earlier above-mirror display example with shift registers)
  - Shift left once is same as multiplying by 2 (0011 (3) becomes 0110 (6))
  - Shifting right once is the same as dividing by 2

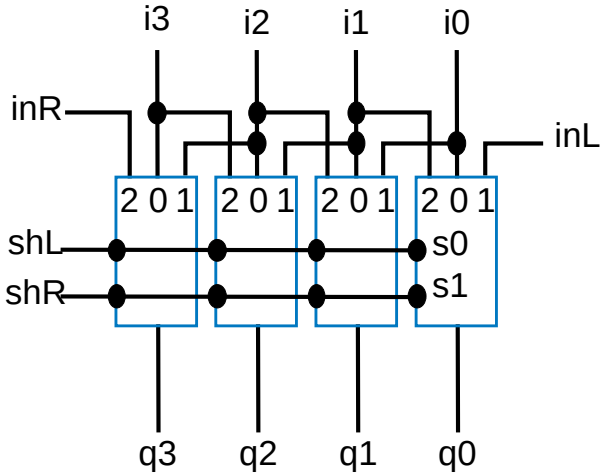




Left shifter



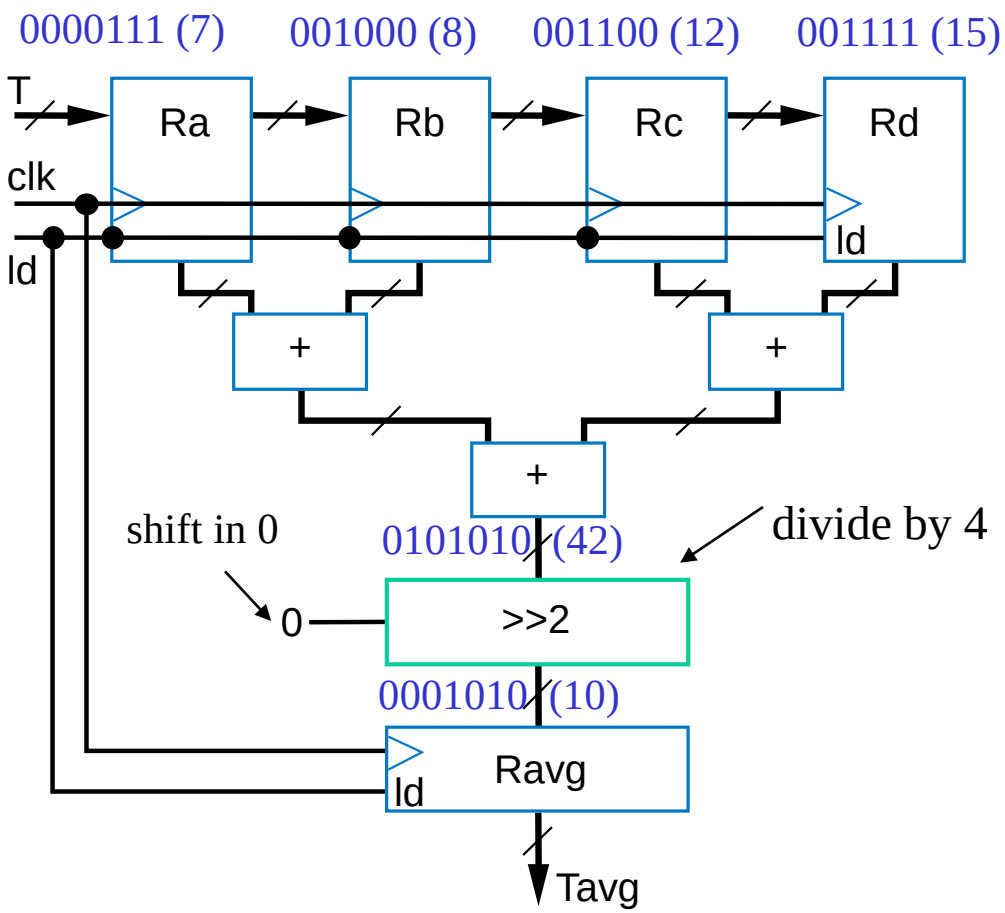
Shifter with left shift or no shift



Shifter with left shift, right shift, and no shift

Shifter Example: Temperature Averager

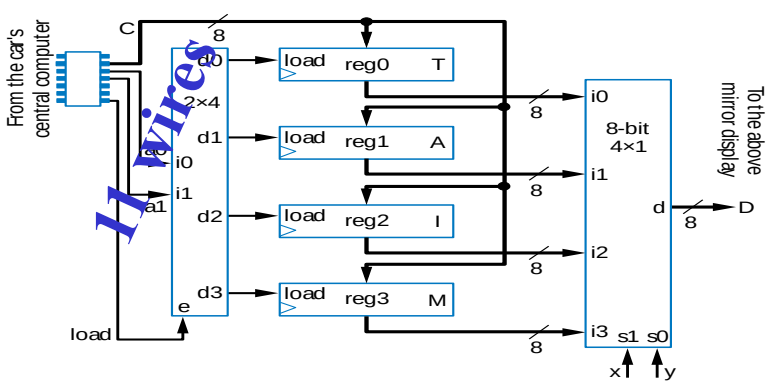
- Four registers storing a history of temperatures
- Want to output the average of those temperatures
- Add, then divide by four
  - Same as shift right by 2
  - Use three adders, and right shift by two

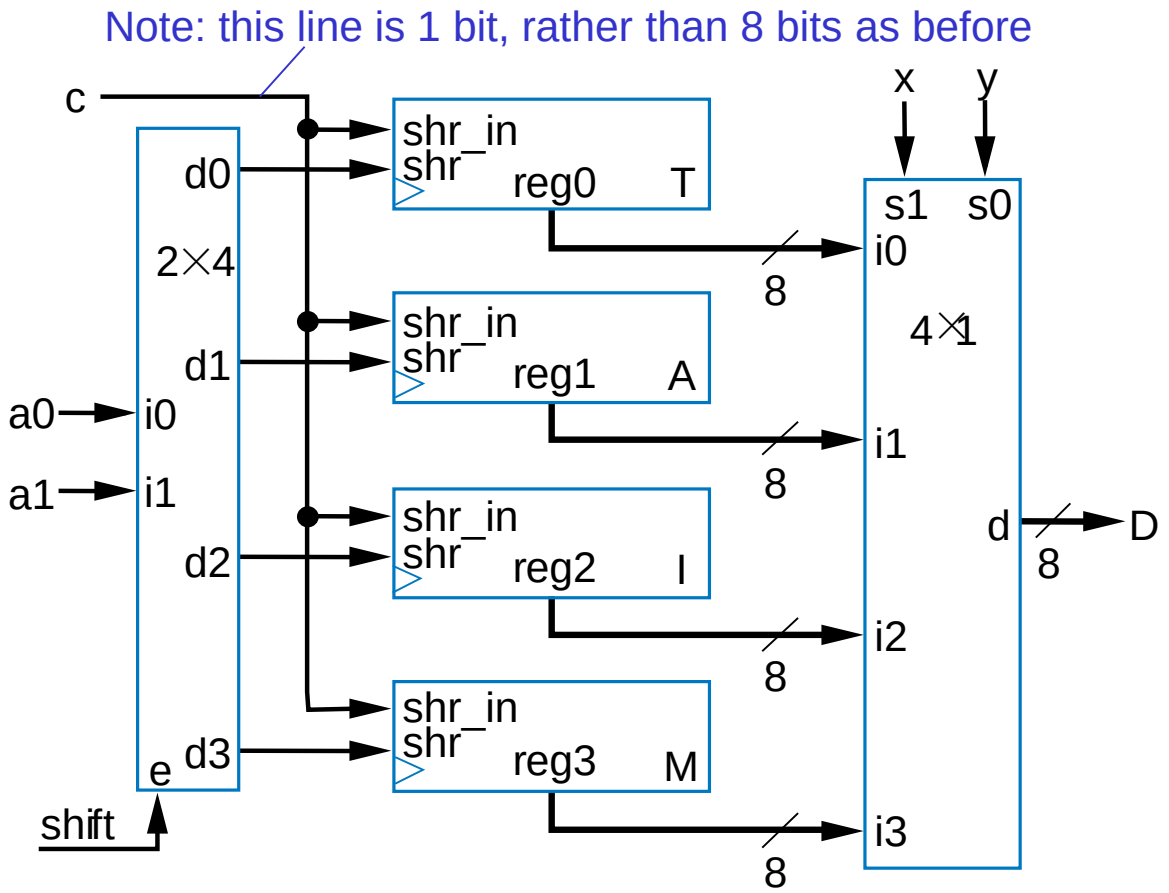


Shift Register Example: Above-Mirror Display

- Earlier example:
  8 +2+1 = 11 wires
  from car's computer to above-mirror display

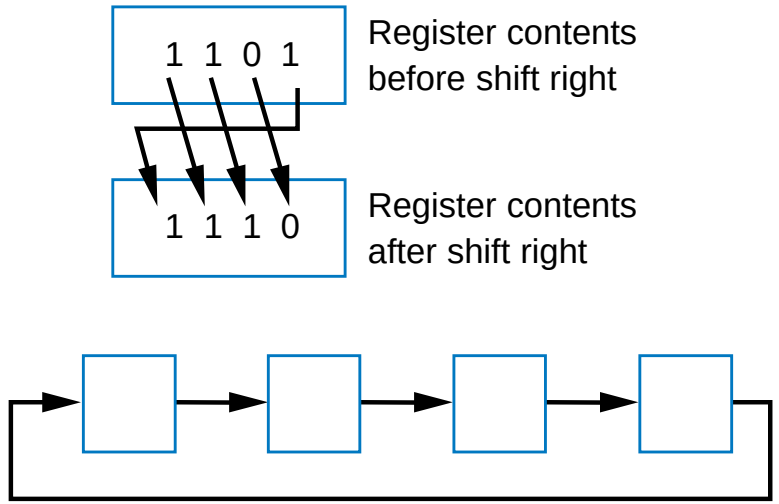
- Use shift registers
  - **Wires: 1+2+1=4**
  - Computer sends one value at a time, one bit per clock cycle





Rotate Register

- Rotate right: leftmost bit comes from rightmost bit

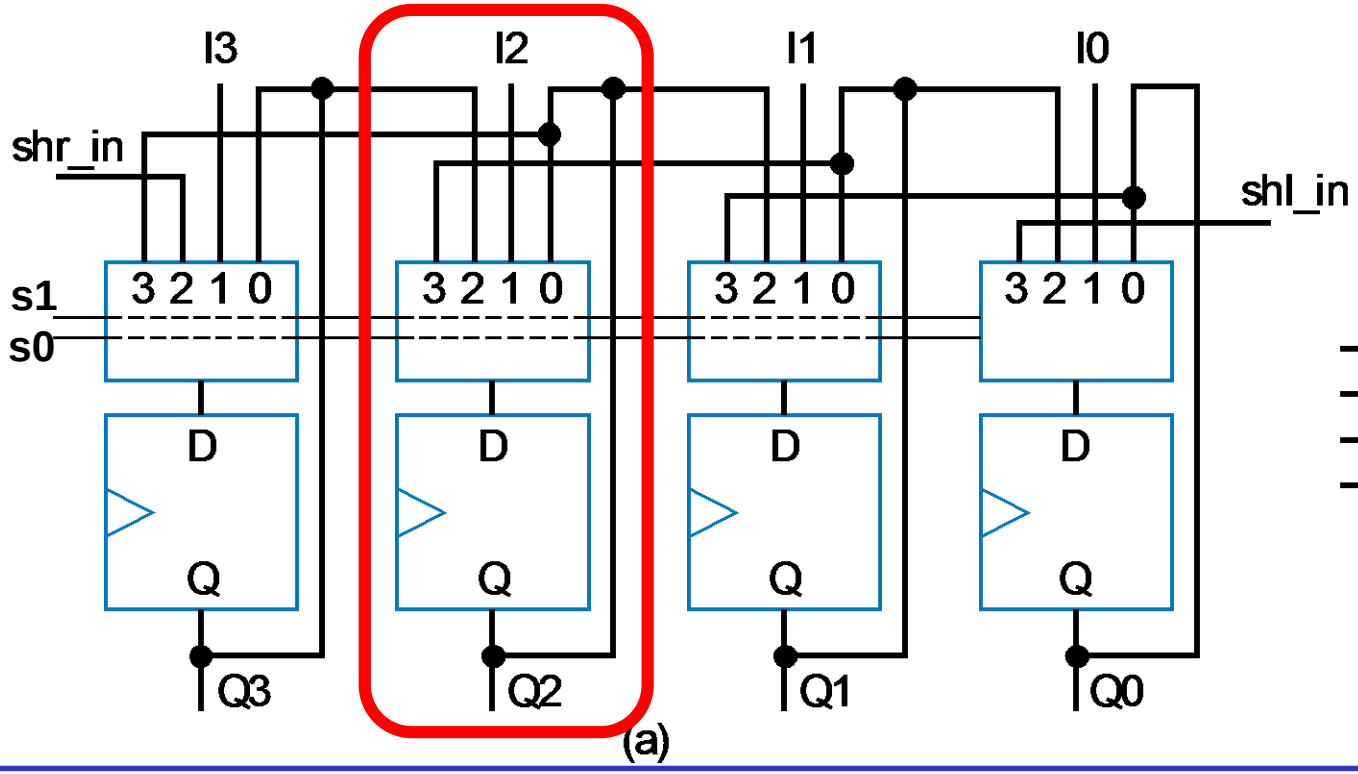


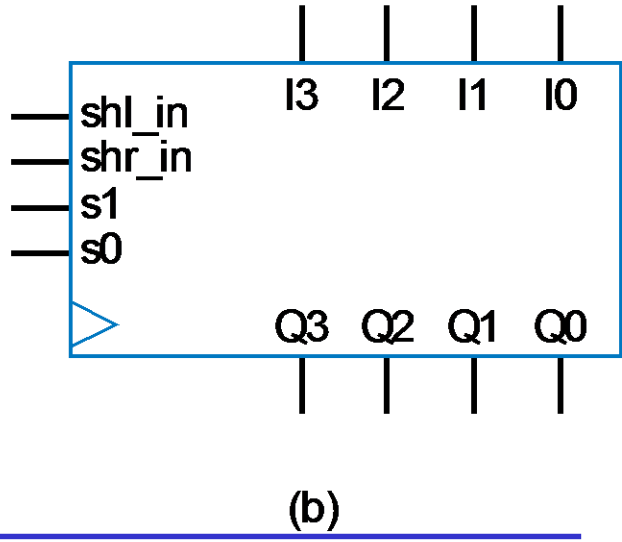
Work out the actual connections with flip-flops and multiplexers!

Multifunction Registers

Operation Table:

s1	s0	Operation
0	0	Maintain present value
0	1	Parallel load
1	0	Shift right
1	1	Shift left



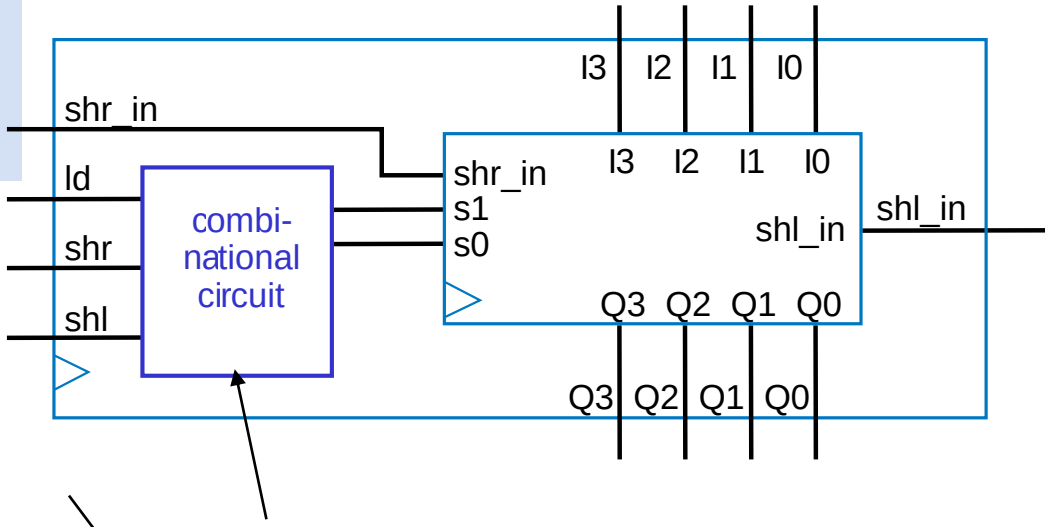


Multifunction Registers with Separate Control Inputs

ld	shr	shl	Operation
0	0	0	Maintain present value
0	0	1	Shift left
0	1	0	Shift right
0	1	1	Shift right – shr has priority over shl
1	0	0	Parallel load
1	0	1	Parallel load – ld has priority
1	1	0	Parallel load – ld has priority
1	1	1	Parallel load – ld has priority

Truth table for combinational circuit

Inputs			Outputs		Operation
ld	shr	shl	s1	s0	
0	0	0	0	0	Maintain value
0	0	1	1	1	Shift left
0	1	0	1	0	Shift right
0	1	1	1	0	Shift right
1	0	0	0	1	Parallel load
1	0	1	0	1	Parallel load
1	1	0	0	1	Parallel load
1	1	1	0	1	Parallel load

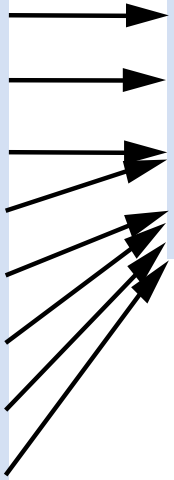


s1 = ld'*shr'*shl + ld'*shr*shl' + ld'*shr*shl

s0 = ld'*shr'*shl + ld

Register Operation Table

Inputs			Outputs		Note
ld	shr	shl	s1	s0	Operation
0	0	0	0	0	Maintain value
0	0	1	1	1	Shift left
0	1	0	1	0	Shift right
0	1	1	1	0	Shift right
1	0	0	0	1	Parallel load
1	0	1	0	1	Parallel load
1	1	0	0	1	Parallel load
1	1	1	0	1	Parallel load



ld	shr	shl	Operation
0	0	0	Maintain value
0	0	1	Shift left
0	1	X	Shift right
1	X	X	Parallel load

- Register operations are typically shown using compact version of table
  - X means same operation whether the value is 0 or 1
    - One X expands to two rows
    - Two Xs expand to four rows
  - Put the highest priority control input on the left to make the reduced table simpler

Register Design Process

Four-step process for designing a multifunction register.

	Step	Description
1.	*Determine mux size*	Count the number of operations (don't forget the maintain present value operation!) and add in front of each flip-flop a mux with at least that number of inputs.
2.	*Create mux operation table*	Create an operation table defining the desired operation for each possible value of the mux select lines.
3.	*Connect mux inputs*	For each operation, connect the corresponding mux data input to the appropriate external input or flip-flop output (possibly passing through some logic) to achieve the desired operation.
4.	*Map control lines*	Create a truth table that maps external control lines to the internal mux select lines, with appropriate priorities, and then design the logic to achieve that mapping

Register Design Example (1/2)

- Desired register operations
  - Load, shift left, synchronous clear, synchronous set

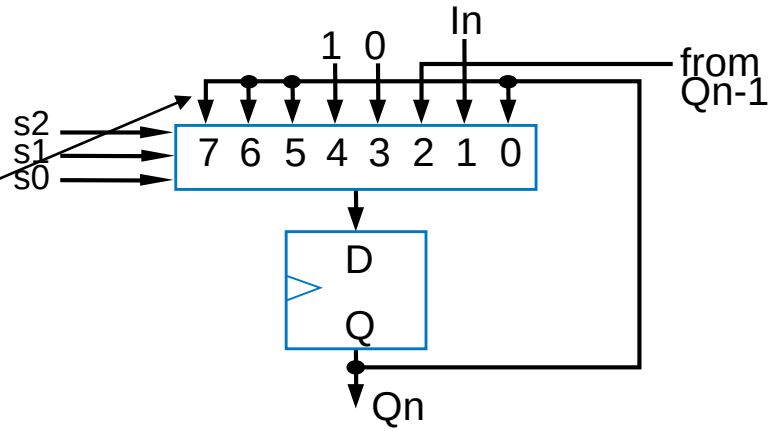
s2	s1	s0	Operation
0	0	0	Maintain present value
0	0	1	Parallel load
0	1	0	Shift left
0	1	1	Synchronous clear
1	0	0	Synchronous set
1	0	1	Maintain present value
1	1	0	Maintain present value
1	1	1	Maintain present value

Step 1: Determine mux size

5 operations (with maintain present value, don't forget this one!) --> **Use 8x1 mux**

Step 2: Create mux operation table

Step 3: Connect mux inputs

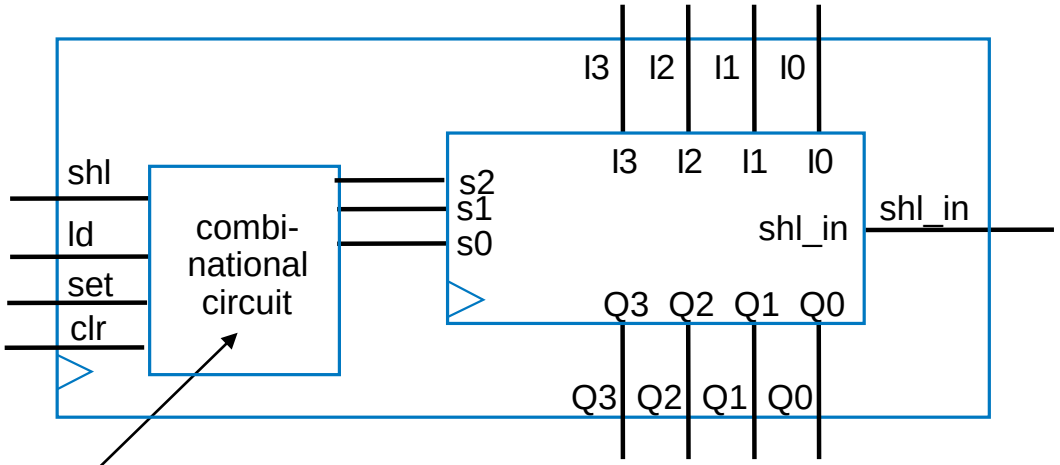


Step 4: Map control lines

s2 = clr'*set
s1 = clr'*set'*ld'*shl + clr
s0 = clr'*set'*ld + clr

Inputs				Outputs			
clr	set	ld	shl	s2	s1	s0	Operation
0	0	0	0	0	0	0	Maintain present value
0	0	0	1	0	1	0	Shift left
0	0	1	X	0	0	1	Parallel load
0	1	X	X	1	0	0	Set to all 1s
1	X	X	X	0	1	1	Clear to all 0s

Register Design Example (2/2)



Step 4: Map control lines

s2 = clr'*set
s1 = clr'*set'*ld'*shl + clr
s0 = clr'*set'*ld + clr

Inputs				Outputs			
clr	set	ld	shl	s2	s1	s0	Operation
0	0	0	0	0	0	0	Maintain present value
0	0	0	1	0	1	0	Shift left
0	0	1	X	0	0	1	Parallel load
0	1	X	X	1	0	0	Set to all 1s
1	X	X	X	0	1	1	Clear to all 0s