

# ECE 274 Digital Logic

## Datapath Components – Multifunction Registers

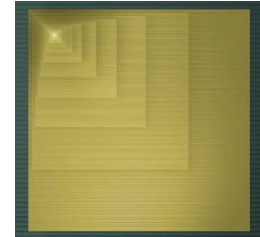
Digital Design 4.1 – 4.2



# Digital Design

## Chapter 4: Datapath Components

Slides to accompany the textbook *Digital Design*, First Edition,  
by Frank Vahid, John Wiley and Sons Publishers, 2007.  
<http://www.ddvahid.com>



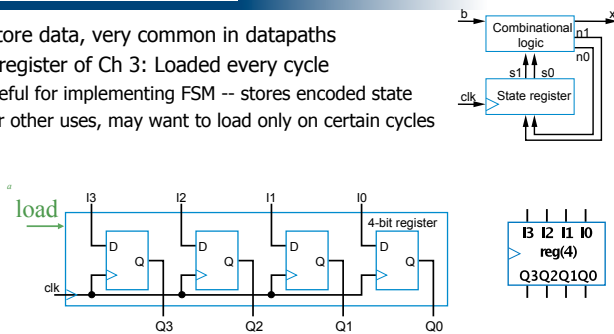
Copyright © 2007 Frank Vahid

Instructors of courses requiring Vahid's *Digital Design* textbook (published by John Wiley and Sons) have permission to modify and use these slides for customary course-related activities, subject to keeping this copyright notice in place and unmodified. These slides may be posted as unannotated pdf versions on publicly-accessible course websites. PowerPoint source (or pdf with animations) may not be posted to publicly-accessible websites, but may be posted for students on internal protected sites or distributed directly to students by other electronic means. Instructors may make printouts of the slides available to students for a reasonable photocopying charge, without incurring royalties. Any other use requires explicit permission. Instructors may obtain PowerPoint source or obtain special use permissions from Wiley – see <http://www.ddvahid.com> for information.

## Datapath Components Registers

4.2

- Can store data, very common in datapaths
- Basic register of Ch 3: Loaded every cycle
  - Useful for implementing FSM -- stores encoded state
  - For other uses, may want to load only on certain cycles

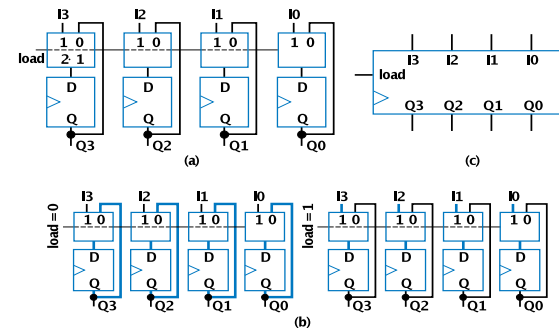


Basic register loads on every clock cycle  
How extend to only load on certain cycles?

3

## Datapath Components Register with Parallel Load

- Add 2x1 mux to front of each flip-flop
- Register's load input selects mux input to pass
  - Either existing flip-flop value, or new value to load

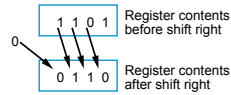


4

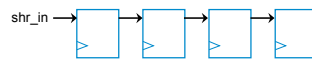
## Datapath Components

### Shift Registers

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



- Shift Register
  - Connect register's flip-flop's outputs to next flip-flop's input
  - This design would always shift on every clock cycle
  - *How can we control it?*



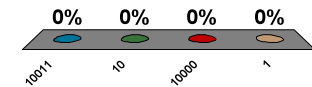
5

## Datapath Components

### Shift Registers

- What is the result after shifting 10011 four times to the right?

1. 10011
2. 00010
3. 10000
4. 00001

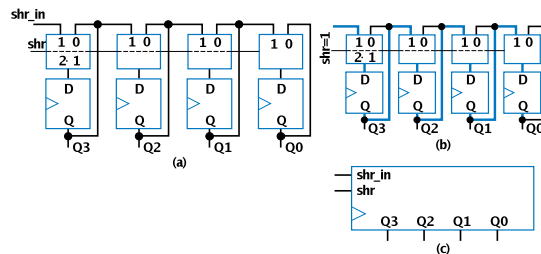


6

## Datapath Components

### Shift Register

- To allow register to either shift or retain, use 2x1 muxes
  - *shr*: 0 means retain, 1 shift
  - *shr\_in*: value to shift in
    - May be 0, or 1
- Note: Can easily design shift register that shifts left instead

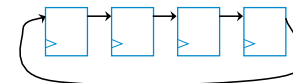
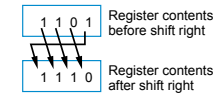


7

## Datapath Components

### Rotate Register

- Rotate right
  - Like shift right, but leftmost bit comes from rightmost bit



8

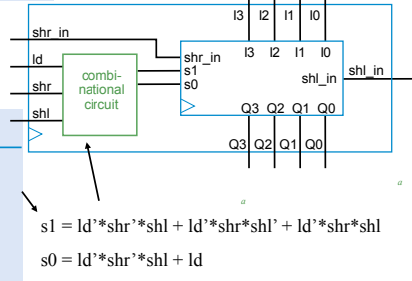


## Datapath Components Multifunction Registers with Separate Control Inputs

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



13

## Datapath Components Register Operation Table

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

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

14

## Datapath Components Register Design Process

- Can design register with desired operations using simple four-step process

TABLE 4.1 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

15

## Datapath Components Register Design Example

- Desired register operations
  - Synchronous clear, synchronous set, load, shift left (with this priority)

### Step 1: Determine mux size

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

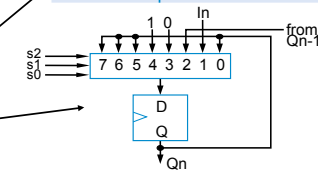
### Step 2: Create mux operation table

### Step 3: Connect mux inputs

### Step 4: Map control lines

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

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

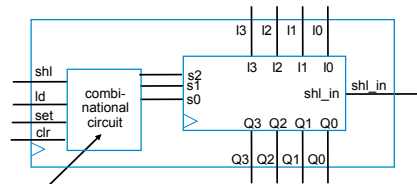


Inputs	clr	set	ld	shl	Outputs	s2	s1	s0	Operation
0	0	0	0	0	0	0	0	0	Maintain present value
0	0	0	1	0	0	1	0	0	Shift left
0	0	1	X	0	0	0	1	0	Parallel load
0	1	X	X	1	0	0	0	0	Set to all 1s
1	X	X	X	0	0	1	1	0	Clear to all 0s

16

## Datapath Components

### Register Design Example



#### Step 4: Map control lines

$$s2 = \text{clr}' * \text{set}$$

$$s1 = \text{clr}' * \text{set}' * \text{ld}' * \text{shl} + \text{clr}$$

$$s0 = \text{clr}' * \text{set}' * \text{ld} + \text{clr}$$

Inputs	Inputs			Outputs			Operation
	clr	set	ld	shl	s2	s1	
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