浙江大學
ZHEJIANG UNIVERSITY

Digital Logic Design

Chapter 7 – Register & Register Transfers(I)

Registers, Micro-operations & Implementations

浙江大学计算机学院　王总辉
zhwang@zju.edu.cn
http://course.zju.edu.cn
2021年10月

Overview

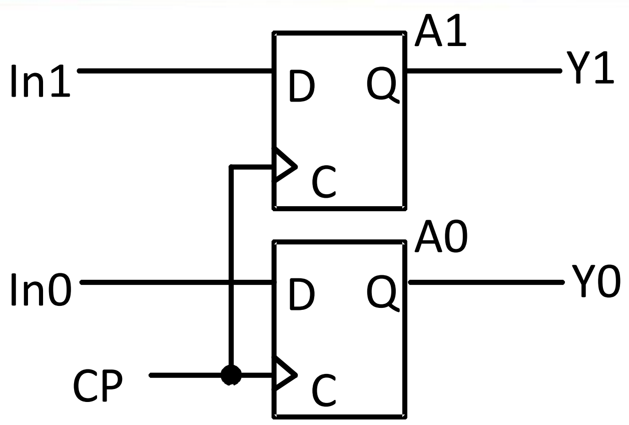
- **Part 1 - Registers, Microoperations and Implementations**
    - Registers and load enable
    - Register transfer operations
    - Microoperations - arithmetic, logic, and shift
    - Microoperations on a single register
        - Multiplexer-based transfers
        - Shift registers
- **Part 2 - Counters, Register Cells, Buses, & Serial Operations**
- **Part 3 – Control of Register Transfers**

Registers

- Register – a collection of binary storage elements
- In theory, a register is sequential logic which can be defined by a state table
- More often, think of a register as storing a vector of binary values
- Frequently used to perform simple data storage and data movement and processing operations

Example: 2-bit Register

- How many states are there?
- How many input combinations? Output combinations?
- What is the *output function*?
- What is the *next state function*?
- Moore or Mealy?



State Table:

Current State A1 A0	Next State A1(t+ 1) A0(t+ 1) For In1 In0 = 00 01 10 11	Output (=A1 A0) Y1 Y0
0 0	00 01 10 11	0 0
0 1	00 01 10 11	0 1
1 0	00 01 10 11	1 0
1 1	00 01 10 11	1 1

- What are the *quantities* above for an *n*-bit register?

Register Design Models

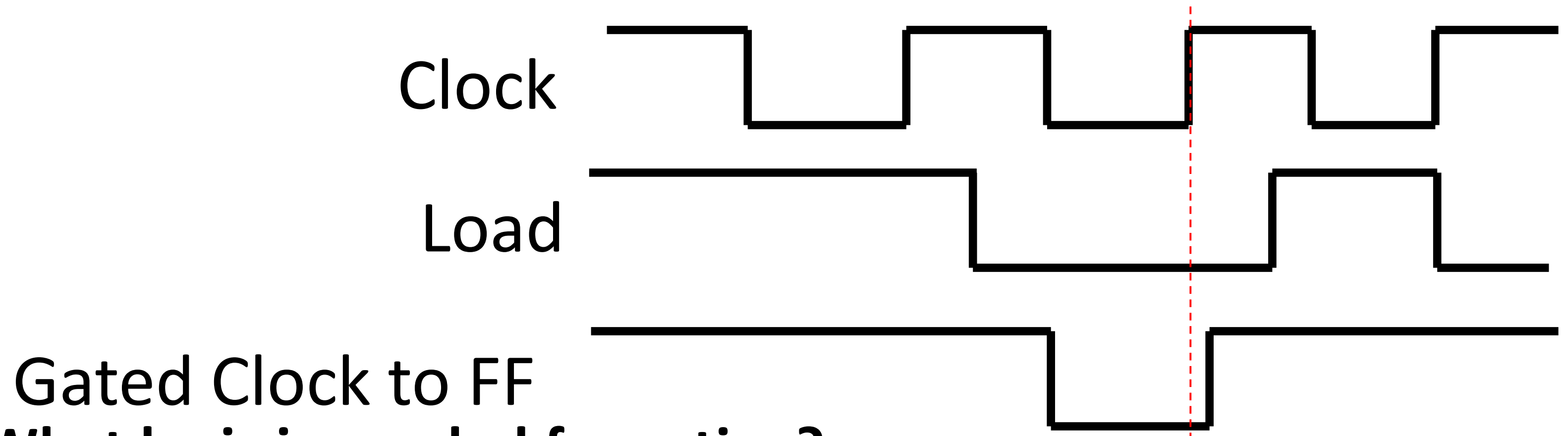
- **Due to the large numbers of states and input combinations as *n* becomes large, the state diagram/state table model is not feasible!**
- **What are methods we can use to design registers?**
  - Add predefined combinational circuits to registers
    - Example: To count up, connect the register flip-flops to an incrementer
  - Design *individual cells* using the state diagram/state table model and combine them into a register
    - A 1-bit cell has just two states
    - Output is usually the state variable

Register Storage

- **Expectations:**
  - A register can store information for multiple clock cycles
  - To “store” or “load” information should be controlled by a signal
- **Reality:**
  - A D flip-flop register loads information on every clock cycle
- **Realizing expectations:**
  - Use a signal to *block* the clock to the register,
  - Use a signal to control *feedback* of the output of the register back to its inputs, or
  - Use other SR or JK flip-flops, that for (0,0) applied, store their state
- **Load is a frequent name for the signal that controls register storage and loading**
  - Load = 0: Load the values on the data inputs
  - Load = 1: Store the values in the register

Registers with Clock Gating

- **The $\overline{Load}$ signal enables the clock signal to pass through if 0 and prevents the clock signal from passing through if 1.**
- **Example: For Positive Edge-Triggered or Negative Pulse Master-Slave Flip-flop:**



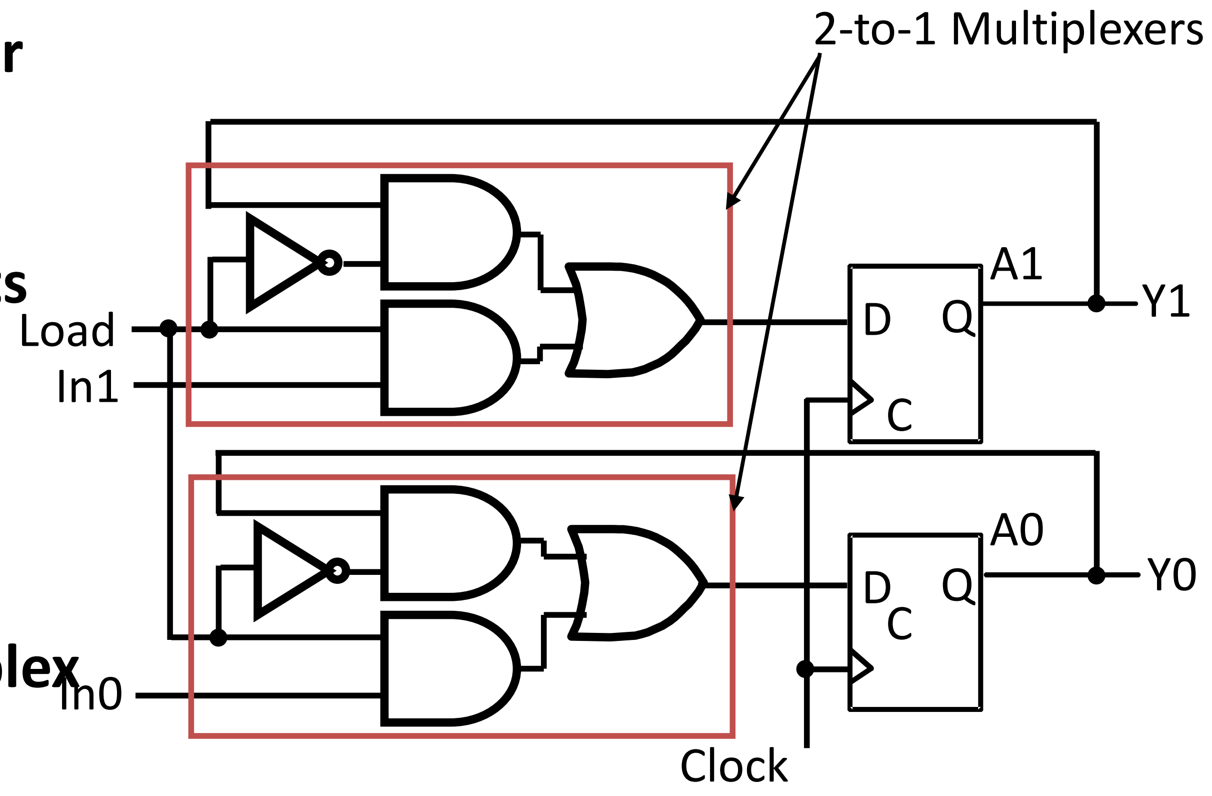
- **What logic is needed for gating?**
- **What is the problem?**

Gated Clock = Clock + $\overline{Load}$

Clock Skew of gated clocks with respect to clock or each other

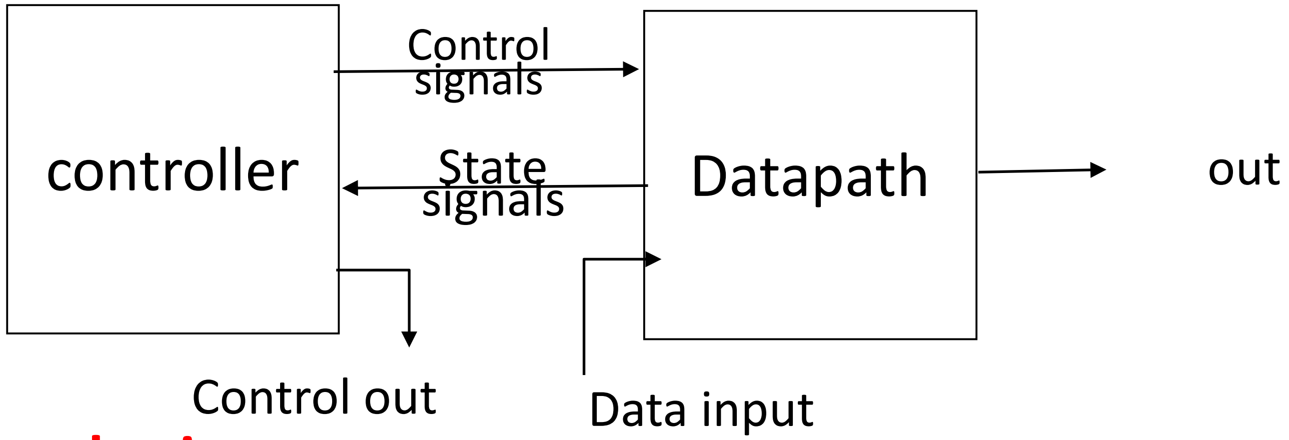
Registers with Load-Controlled Feedback

- **A more reliable way to selectively load a register:**
  - Run the clock continuously, and
  - Selectively use a load control to change the register contents.
- **Example: 2-bit register with Load Control:**
- **For Load = 0, loads register contents (hold current values)**
- **For Load = 1, loads input values (load new values)**
- **Hardware more complex than clock gating, but free of timing problems**



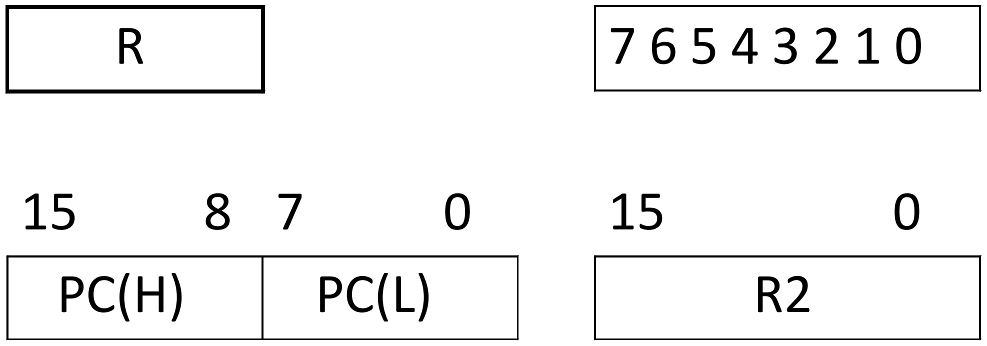
Register Transfer Operations

- **Register Transfer Operations – The movement and processing of data stored in registers**



- **Three basic components:**
  - Set of registers
  - Operations
  - control of operations
- **Elementary Operations -- load, count, shift, add, bitwise "OR", etc.**
  - Elementary operations called *microoperations*

Register Notation



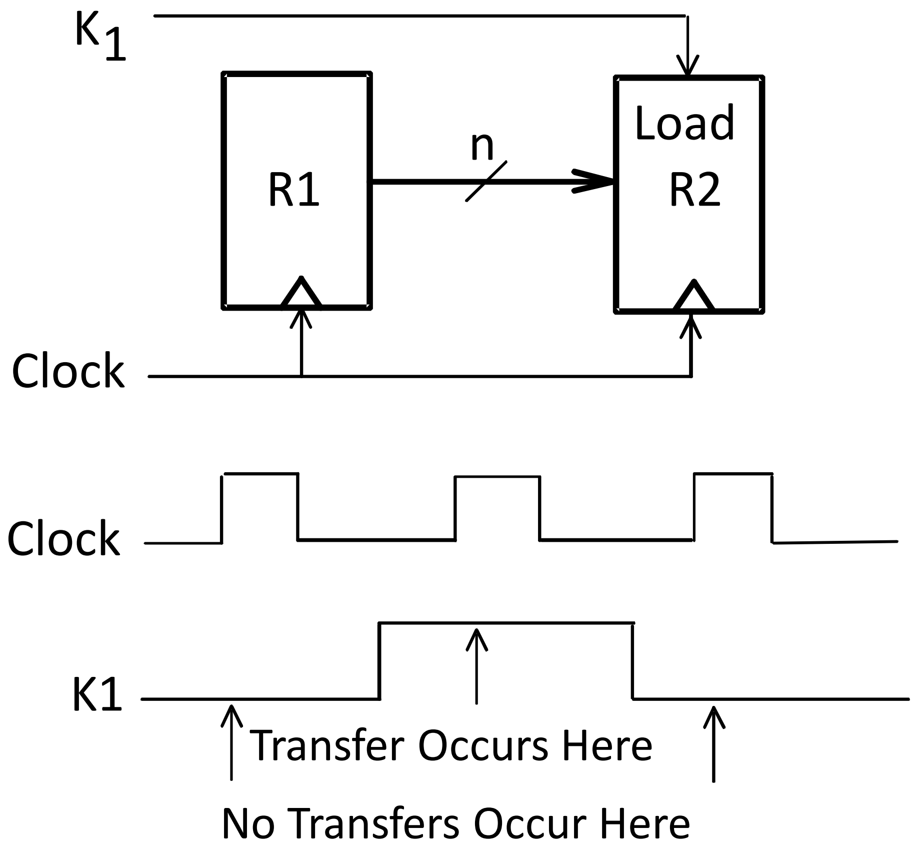
- **Letters and numbers – denotes a register (ex. R2, PC, IR)**
- **Parentheses ( ) – denotes a range of register bits (ex. R1(1), PC(7:0), PC(L))**
- **Arrow (←) – denotes data transfer (ex. R1 ← R2, PC(L) ← R0)**
- **Comma – separates parallel operations**
- **Brackets [ ] – Specifies a memory address (ex. R0 ← M[AR], R3 ← M[PC])**

Conditional Transfer

- **If (K1 =1) then (R2 ← R1) is shortened to**

  K1: (R2 ← R1)

  where K1 is a control variable specifying a conditional execution of the microoperation.



Microoperations

- **Logical Groupings:**
  - Transfer - move data from one register to another
  - Arithmetic - perform arithmetic on data in registers
  - Logic - manipulate data or use bitwise logical operations
  - Shift - shift data in registers

Arithmetic operations

+ Addition

– Subtraction

*** Multiplication**

/ Division

Logical operations

$\vee$ Logical OR

$\wedge$ Logical AND

$\oplus$ Logical Exclusive OR

$\overline{}$ Not

Example Microoperations

- **Add the content of R1 to the content of R2 and place the result in R1.**

  $R1 \leftarrow R1 + R2$

- **Multiply the content of R1 by the content of R6 and place the result in PC.**

  $PC \leftarrow R1 * R6$

- **Exclusive OR the content of R1 with the content of R2 and place the result in R1.**

  $R1 \leftarrow R1 \oplus R2$

Example Microoperations (Continued)

- Take the 1's Complement of the contents of R2 and place it in the PC.

$$PC \leftarrow \overline{R2}$$

- On condition K1 OR K2, the content of R1 is Logic bitwise OR with the content of R3 and the result placed in R1.

$$(K1 + K2):\ R1 \leftarrow R1 \vee R3$$

- NOTE: "+" (as in $K_1 + K_2$) and means "OR" In $R1 \leftarrow R1 + R3$, + means "plus."

Control Expressions

- **The control expression for an operation appears to the left of the operation and is separated from it by a colon**
- **Control expressions specify the logical condition for the operation to occur**
- **Control expression values of:**
  - Logic "1" -- the operation occurs.
  - Logic "0" -- the operation is does not occur.

- Example:

  $\overline{X}\,K1: \; R1 \longleftarrow R1 + R2$

  $X\,K1: \; R1 \longleftarrow R1 + \overline{R2} + 1$
- Variable K1 enables the add or subtract operation.
- If X =0, then $\overline{X}$ =1 so $\overline{X}$ K1 = 1, activating the addition of R1 and R2.
- If X = 1, then X K1 = 1, activating the addition of R1 and the two's complement of R2 (subtract).

Arithmetic Microoperations

- **From Table 7-3:**

Symbolic Designation	Description
$R0 \leftarrow R1 + R2$	Addition
$R0 \leftarrow \overline{R1}$	One's Complement
$R0 \leftarrow \overline{R1} + 1$	Two's Complement
$R0 \leftarrow R2 + \overline{R1} + 1$	R2 minus R1 (2's Comp)
$R1 \leftarrow R1 + 1$	Increment (count up)
$R1 \leftarrow R1 - 1$	Decrement (count down)

- **Note that any register may be specified for source 1, source 2, or destination.**
- **These simple microoperations operate on the whole word**

Logical Microoperations

- **From Table 7-4:**

Symbolic Designation	Description
$R0 \leftarrow \overline{R1}$	Bitwise NOT
$R0 \leftarrow R1 \vee R2$	Bitwise OR (sets bits)
$R0 \leftarrow R1 \wedge R2$	Bitwise AND (clears bits)
$R0 \leftarrow R1 \oplus R2$	Bitwise EXOR (complements bits)

Logical Microoperations (continued)

- **Let R1 = 10101010, and R2 = 11110000**
- **Then after the operation, R0 becomes:**

R0	Operation
01010101	$R0 \leftarrow \overline{R1}$
11111010	$R0 \leftarrow R1 \vee R2$
10100000	$R0 \leftarrow R1 \wedge R2$
01011010	$R0 \leftarrow R1 \oplus R2$

Shift Microoperations

- **From Table 7-5:**
- **Let R2 = 11001001**
- **Then after the operation, R1 becomes:**

Symbolic Designation	Description
$R1 \leftarrow$ sl R2	Shift Left
$R1 \leftarrow$ sr R2	Shift Right

R1	Operation
10010010	$R1 \leftarrow$ sl R2
01100100	$R1 \leftarrow$ sr R2

- Note: These shifts "zero fill". Sometimes a separate flip-flop is used to provide the data shifted in, or to “catch” the data shifted out.
- Other shifts are possible (rotates, arithmetic) (see Chapter 10).

□谢 谢！