Chapter 7

Registers & Counters & Memory Unit
 Registers

• **A register** is a set of flip flops, often supplemented by additional circuits to control input and output.
• can have parallel I/O or serial I/O or combination
• Usually, registers are used to store a set of related bits.
  – bits that collectively represent an integer value
  – bits of an ASCII character code
  – status bits for a device in a computer system (disk controller)
• **Counters** are registers that store numeric values along with circuits to increment/decrement the stored value.
  – up-counters, down-counters, up-down counters
  – generalized counters
  – BCD counters, gray-code counters, ...
Registers

• A register is a group of binary cells suitable for holding information.

• An n-bit register has a group of n-flip flops & can store any binary information containing n-bits.

• In addition to the flip flops, a register may have combinational gates that control when & how new information is transferred into the register.
Clock pulse triggered the F.F. in the positive edge and the information in the inputs transferred to the F.F.

When clear = 0 all F.F. goes to 0

Fig. 6-1 4-Bit Register
Register with parallel load

• Load: is the transfer of new information into a register
• Parallel load: is to load all bits of input information to the register simultaneously with a common clock pulse.
• In order to keep the information of a register unchanged period of time, we can insert some gates on:
  - clock pulse signals
  - F.F. input signals
• But to fully synchronize the system, we must ensure that all clock pulses arrive at the same time anywhere in the system so that all F.F. trigger simultaneously, for this reason, it is advisable to control the F.F. inputs.
• A load signal with a combinational logic are used
If load=1 then the data in the four inputs are transferred into the register, otherwise the register unchanged (output feedback taken)

Fig. 6-2 4-Bit Register with Parallel Load
Register with parallel load using 2X1 MUX
Registers

• Shift Registers:

- **Right Shift**
- **Left Shift**
Registers

- Design a 4-bit right shift register:
  4 F.F. – D Type

Fig. 6-3 4-Bit Shift Register
Registers
 Registers

- Design a 4-bit left shift register:
  4 F.F. – D Type
Fig. 6-4  Serial Transfer from Register $A$ to register $B$
 Registers

Bidirectional Shift Register with Parallel Load:

Problem: Design a 2-bit register that can do the following:

1. No Change
2. Shift Right
3. Shift Left
4. Parallel Load

<table>
<thead>
<tr>
<th>$S_1$</th>
<th>$S_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Registers

Cp

4 x 1 MUX

DR1

4 x 1 MUX

DR2

R1

R2

SI

S0

S1

DR1

DR2

SI

S0

S1
Registers

- Shift Left
- Shift Right

Serial In – Serial Out Register

Parallel in – Parallel out register $\Rightarrow (P_i \rightarrow P_o)$ Registers
Registers

Problem: Design 2-bit $P_i - P_o$ register (Using D F.F. & SR F.F.)
Registers

Parallel in – Serial Out Register:

[Diagram of a parallel in - serial out register with labels: load, I1, SI, I2, CP, D, Q, SO]
Registers

Serial In – Parallel Out Register:

\[ \text{SI} \rightarrow \text{D} \rightarrow \text{Q} \]

\[ \text{Cp} \rightarrow \text{D} \rightarrow \text{Q} \]
Counters

- Synchronous (Parallel) Counters: Binary, BCD.
- Asynchronous (Ripple) Counters: Binary, BCD.

The synchronous counters are distinguished from Ripple counters in that the clock pulses are applied to the Cp inputs of all the flip flops.
Counters

Ripple Counters:

In a ripple counter, the F.F. output transition serves as a source for triggering other F.F.

Ex: Design a 4-bits binary Ripple counter.
Binary counter: a counter that follows the binary number sequence.
An n-bit binary counter consists of n flip-flops and can count in binary from 0 through $2^n-1$
Counters
Counters

Design a BCD ripple counter

[Diagram of a BCD ripple counter with J-K flip-flops connected in a ripple counter configuration.]

Count

Logic-1

Q1

Q2

Q4

Q8
Counters

Block Diagram of a Three-Decade Decimal BCD Counter
– Synchronous Counters:
– Binary:
– Design a 4-bit synchronous binary counter?
Counters
Counters

Design a 4-bit up/down binary counter.
Counters

4-Bit Binary Counter with Parallel Load

Count
Load
$I_0$
$I_1$
$I_2$
$I_3$
Clear
CLK
$A_0$
$A_1$
$A_2$
$A_3$
Carry-output

dr.abu-argoub
Counters

Design a BCD synchronous counter.

(a) Using the load input

(b) Using the clear input

Two ways to Achieve a BCD Counter Using a Counter with Parallel Load
Logically, a *random access memory* contains an array of numbered storage locations, called *words*.

- When read/write is high, data_out is equal to the value stored in word specified by address inputs.
- When read/write is low, the value on data_in replaces the value in word specified by address outputs.
- Separate *enable* signal also usually provided.
- Simplest RAMs are *asynchronous* - no clock input.
- Synchronous circuits using RAMs must ensure that RAM timing requirements are satisfied to ensure correct operation.
RAM

A memory unit stores binary information in groups of bits called words.

The block diagram of the memory unit (RAM):
<table>
<thead>
<tr>
<th>Memory address</th>
<th>Binary</th>
<th>decimal</th>
<th>Memory contest</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000000</td>
<td>0</td>
<td>101010101011101</td>
<td></td>
</tr>
<tr>
<td>00000000001</td>
<td>1</td>
<td>101010110001001</td>
<td></td>
</tr>
<tr>
<td>00000000010</td>
<td>2</td>
<td>000011010001110</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1111111101</td>
<td>1021</td>
<td>1001110100010100</td>
<td></td>
</tr>
<tr>
<td>1111111110</td>
<td>1022</td>
<td>0000110100011110</td>
<td></td>
</tr>
<tr>
<td>1111111111</td>
<td>1023</td>
<td>110111000100101</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 7-3  Content of a 1024 × 16 Memory
RAM

(a) Logic diagram

(b) Block diagram

Fig. 7-5 Memory Cell
Fig. 7-6 Diagram of a $4 \times 4$ RAM
Timing of RAM Operations

- **Read cycle**
  - *access time*: time from “last” address change until output data is valid
- **Write cycle**
  - \( t_1 \): min time from address stable and enable asserted until r/w’ is lowered
  - \( t_2 \): min time that input data must remain stable before r/w’ can be raised
  - \( t_3 \): min time that address stays valid after r/w’ is raised
  - *cycle time* is \( t_1 + t_2 + t_3 \)