Digital Teknik II / Digital Design II

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Digital Teknik / Digital Design

Modul 3

PLD, VHDL, Latches, Flip-Flops & Counters

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Programmable Logic Devices

- PLA
- PAL
- PLD
- CPLD
- FPGA
- ASIC

Programmable Logic Devices

PLA contains a 2 level structure of AND and OR gates with user programmable connections (sum of the product, basically min terms).

PLA structure was enhanced to PAL programmable logic arrays and today such devices are called PLD programmable Logic Devices.

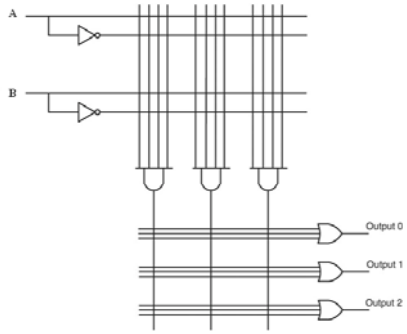
However, PAL lack scalability as the 2 level cannot be scaled to larger sizes, so instead of PLD, CPLD Complex PLD are introduced which basically contains large number of PLDs and interconnection structure that is also programmable so providing a rich design capability.

The later version of CPLD which contains large number of individual blocks is called FPGA.

Possible min terms: $\bar{A}\,\bar{B}$, $\bar{A}\,B$, $A\,\bar{B}$, $A\,B$

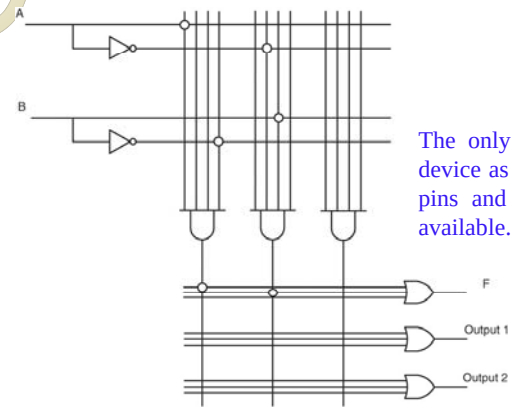
Example PLD

Sum of the product: AND gates provide the product and OR gates as sum.



Embedded Systems: A Contemporary Design Tool by James K. Peckol
DE3 Course
Dr. D. M. Akbar Hussain
Department of Electronic Systems
5

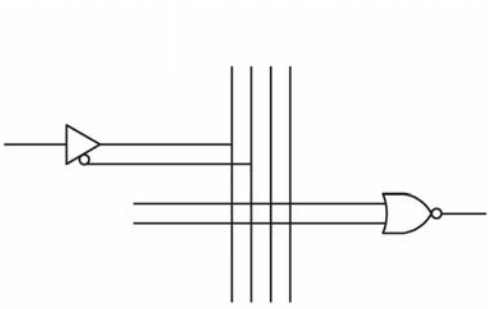
Programming $F = A\,\bar{B} + \bar{A}\,B$



The only limitation here is size of the device as the number of input and output pins and the number of product terms available.

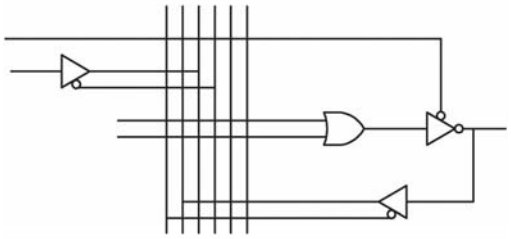
Embedded Systems: A Contemporary Design Tool by James K. Peckol
DE3 Course
Dr. D. M. Akbar Hussain
Department of Electronic Systems
6

True & Negated Output

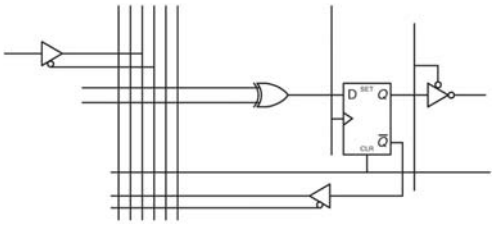


To make it more flexible and enhancing the capability of these circuits an input driver with true and negated option.

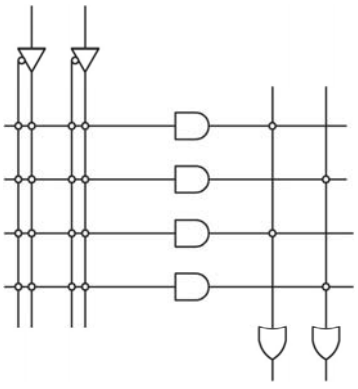
Tri-state Option



Latched Tri-state output

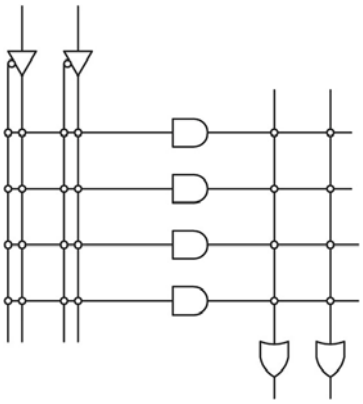


PAL



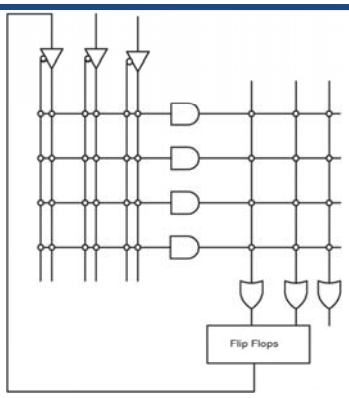
In PAL, only AND gates are programmable where as OR gates are fixed, so there is a limit on producing product terms in sum.

PLA



In this case both are programmable, so product terms are re-usable.

PLS Programmable Logic Sequencer



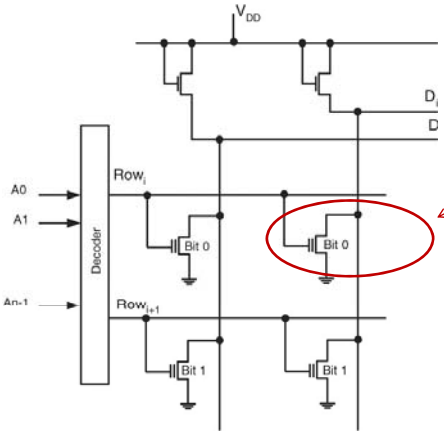
This is just the extension of PLA, it contains a flip-flop as a storage element and typically the output of the flip-flop is fed back to the logic to support finite state machine design.

Technology for Programing

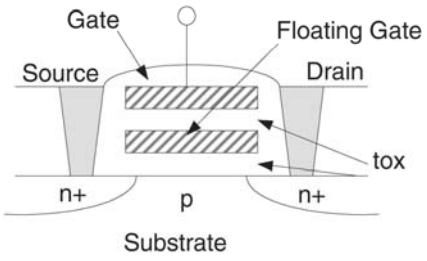
One time: Antifuse (Metal crystalline alloy is created for conduction.

Reprogrammable: Floating Gate.

Programmable Gate

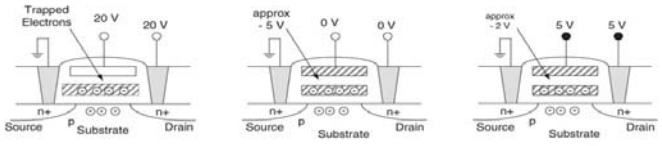


Floating Gate



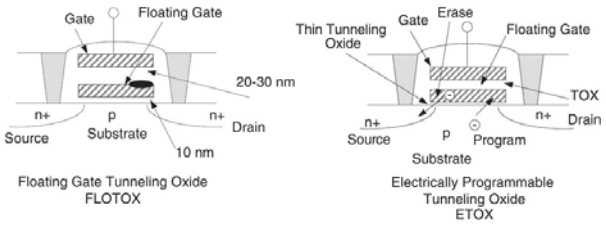
Initially, floating gate has no charge so it has no effect on circuit operation and all transistors are effectively connected, to program the logic high voltage is applied to the link where no connection is required.

Programming Gate



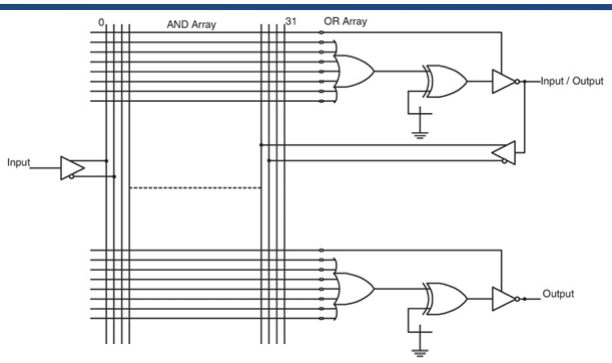
Through avalanche injection a negative charge is placed to the floating gate so once you apply a normal 1 logic transistor does not turned on and typically it is believed that the charge remains there for 10 years. Ultraviolet light was used to erase the device, because it becomes conductive under ultraviolet light.

Electronically Erasable



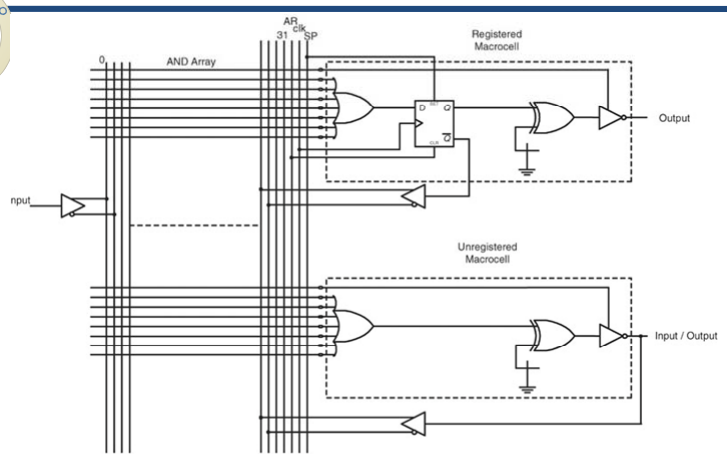
Modern devices are manufactured in a different way using a tunneling effect, the advantage is that one can electronically erase the device.

Architectures



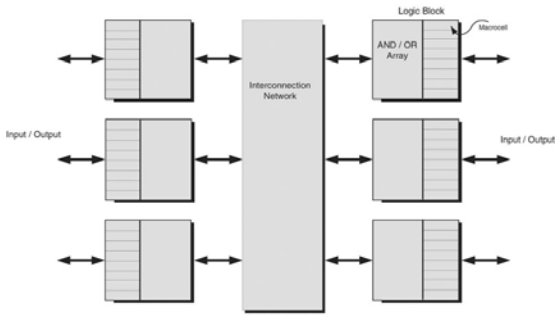
PLD and CPLD has coarse grain architecture meaning fewer interconnections, where as FPGA has fine grain architecture having a rich inter connection topology. Generic (GAL) Gate Array logic is also a PAL as shown in figure.

Registered/Un-registered Output



Output which contains a flip-flop is called registered output where as the other is obviously called un-registered output.

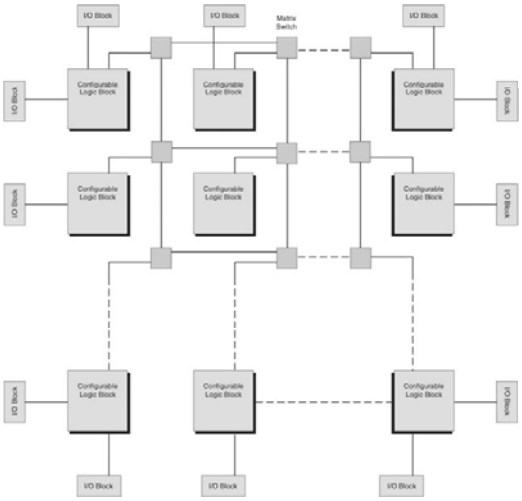
CPLD



Contains 6 logic block with each containing 6 macro-cell, interconnection is supported at different levels some may be connected some may be not.

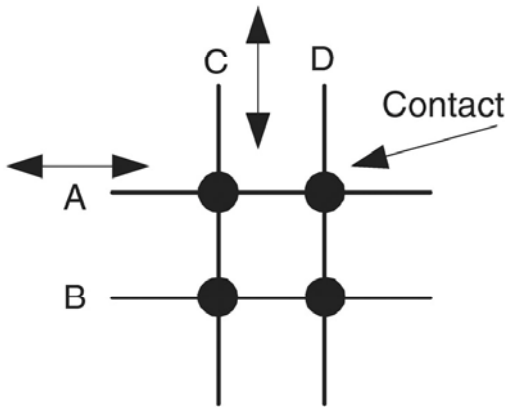
FPGA

The basic logic blocks are much simpler than the CPLD, but it is fine grain interconnection topology.



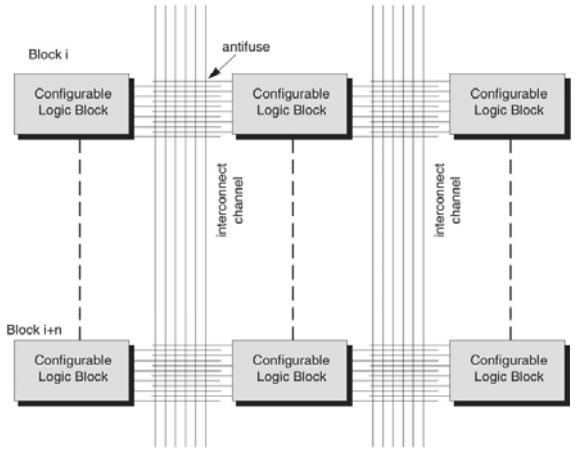
Embedded Systems: A Contemporary Design Tool by James K. Peckol
DE3 Course
Dr. D. M. Akbar Hussain
Department of Electronic Systems
21

Switch Matrix



Embedded Systems: A Contemporary Design Tool by James K. Peckol
DE3 Course
Dr. D. M. Akbar Hussain
Department of Electronic Systems
22

Antifuse Based FPGA



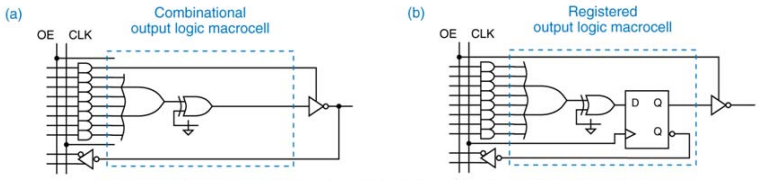
Embedded Systems: A Contemporary Design Tool by James K. Peckol

DE3 Course

Dr. D. M. Akbar Hussain
Department of Electronic Systems

23

Example



Output logic macrocells for the 16V8R: (a) combinational; (b) registered.

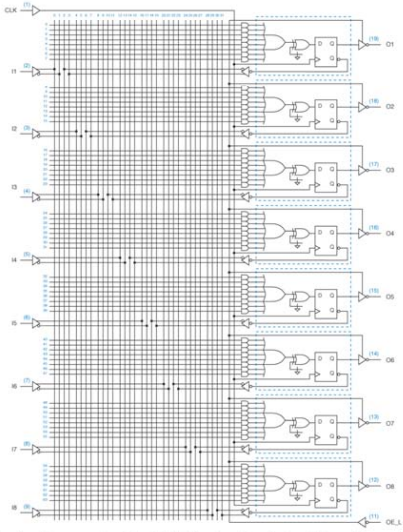
Output which contains a flip-flop is called registered output .

DE3 Course

Dr. D. M. Akbar Hussain
Department of Electronic Systems

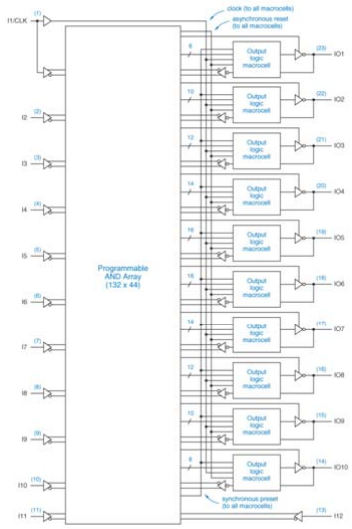
24

Example 16V8 GAL



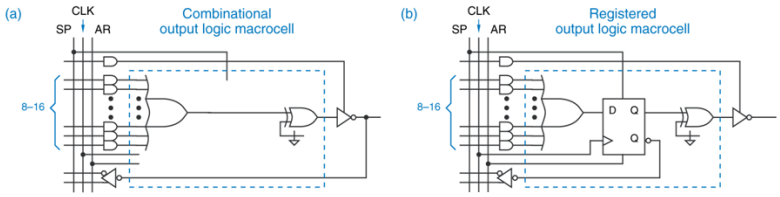
Logic diagram for the 16V8 in the "registered" configuration.

Example 22V10 GAL



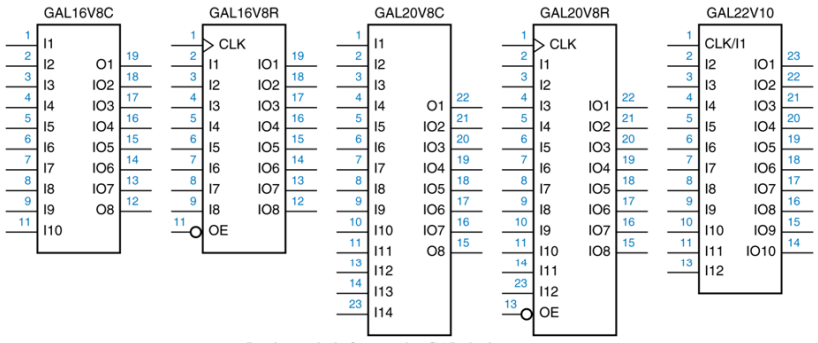
Logic diagram for the 22V10.

Macro-cell for 22V10



Output logic macrocells for the 22V10: (a) combinational; (b) registered.

Example GAL Devices



Logic symbols for popular GAL devices.

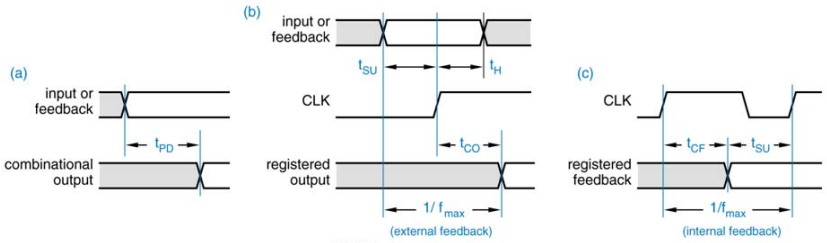
Timing Specifications

Part numbers	Suffix	t_{PD}	t_{CO}	t_{CF}	t_{SU}	t_H
GAL16V8D, GAL20V8B	-7	7.5	5	3	7	0
GAL16V8D, GAL20V8B	-10	10	7	6	10	0
GAL16V8D, GAL20V8B	-15	15	10	8	12	0
GAL16V8D, GAL20V8B	-25	25	12	10	15	0
GAL22V10D	-7	7.5	4.5	3	4.5	0
GAL22V10D	-10	10	7	2.5	7	0
GAL22V10D	-15	15	8	2.5	10	0
GAL22V10D	-25	25	15	13	15	0

Timing specifications, in nanoseconds, of popular CMOS GAL devices in DIPs.

t_{PD} : Propogation delay
t_{CO} : Propagation delay from the rising edge of the clock
t_{CF} : PD from the rising edge of the clock to the macrocell's output
t_{SU} : Setup time during which signal must be stable
t_H : Signal at D input must be hold for that period of time

Timing Parameters



PLD timing parameters.

It should be remembered that timing parameters in PLDs have longer setup times because of AND & OR gates, so for critical timing requirements it should be kept in mind for implementation.

Which gate is faster ?

HDL: Hardware Description Language
Verilog HDL
ABEL: Advanced Boolean Equation Language

HDL and Verilog are hardware description languages.

HDL has its root in PASCAL & ADA

Where as Verilog has its root in C.

VHDL

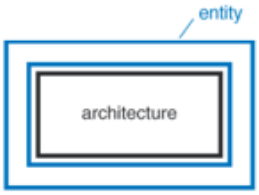
Documentation and Modelling Language

Developed and designed with principles of structured programming in mind (ideas from PASCAL and Ada).

- Design is decomposed.
- Each design element has both well defined interface and precise functional specifications.
- Concurrency, timing and clocking can be modelled.
- Logical operations and timing behaviour can be simulated.

VHDL is basically, VHSIC Hardware Description Language, Very High Speed Integrated Circuits HDL.

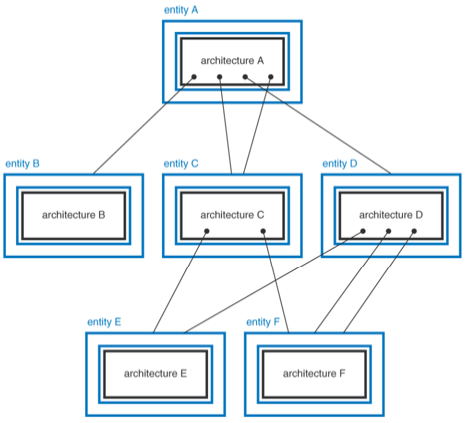
VHDL Entity & Architecture



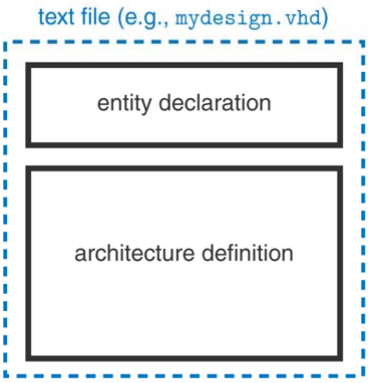
Entity is declaration of modules input and output. It is like a wrapper hiding the details of what is inside.

Where as architecture is more detailed description of the modules internal behavior or structure. This allows a hierarchical design.

Hierarchical Structure



VHDL Program



Inside the text file entity and architecture is defined separately.

VHDL Program Example

```
entity Inhibit is     -- also known as 'BUT-NOT'
  port (X,Y: in BIT;    --  as in 'X but not Y'
        Z:   out BIT);  --  (see [Klir, 1972])
end Inhibit;

architecture Inhibit_arch of Inhibit is
begin
  Z <= '1' when X='1' and Y='0' else '0';
end Inhibit_arch;
```

VHDL Syntax

```
architecture architecture-name of entity-name is
   type declarations
   signal declarations
   constant declarations
   function definitions
   procedure definitions
   component declarations
begin
   concurrent-statement
   ...
   concurrent-statement
end architecture-name;
```


VHDL Predefined Types

`bit`	`character`	`severity_level`
`bit_vector`	`integer`	`string`
`boolean`	`real`	`time`

integer Operators		*boolean* Operators	
`+`	addition	`and`	AND
`-`	subtraction	`or`	OR
`*`	multiplication	`nand`	NAND
`/`	division	`nor`	NOR
`mod`	modulo division	`xor`	Exclusive OR
`rem`	modulo remainder	`xnor`	Exclusive NOR
`abs`	absolute value	`not`	complementation
`**`	exponentiation		

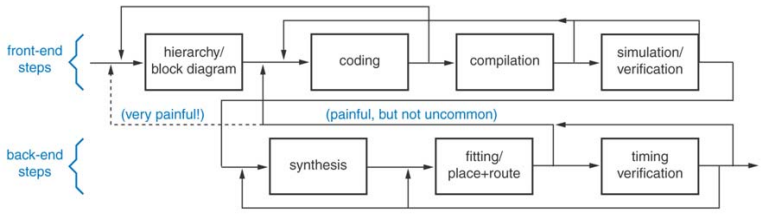
VHDL Function Decleration

```
function function-name (
      signal-names : signal-type;
      signal-names : signal-type;
      ...
      signal-names : signal-type
) return return-type is
  type declarations
  constant declarations
  variable declarations
  function definitions
  procedure definitions
begin
  sequential-statement
  ...
  sequential-statement
end function-name;
```


VHDL Program with Function

```
architecture Inhibit_archf of Inhibit is

function ButNot (A, B: bit) return bit is
begin
  if B = '0' then return A;
  else return '0';
  end if;
end ButNot;

begin
  Z <= ButNot(X,Y);
end Inhibit_archf;
```

VHDL Design Flow



VHDL Design Flow

Before we start going into any language we must understand the overall language in this case VHDL design flow.

It is divided into 2 parts, front end and back end.

Typically, large logic designs has the capacity to be divided into blocks of different functionality so one can have a hierarchical structure.

Verification or testing has 2 dimensions, functional verification which is concerned with logical behaviour of the circuit where as timing is concerned with more on gate delays etc, so in this case one test the actual delay verses the estimated delay. However, in the front end it is still limited verification.

In the back end the design is converted into set of primitives which can be assembled into the target technology (PLD/CPLD etc). Fitting is mapping to the technology. Final step is the timing verification due to many things including length of wires etc.

ABEL Program Structure

```
module module name
title string
deviceID device deviceType;
pin declarations
other declarations
equations
equations
test_vectors
test vectors
end module name
```

For Reference

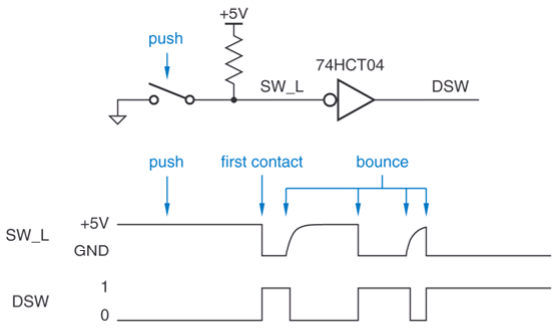
ABEL Program Example

```
module Alarm_Circuit
title 'Alarm Circuit Example
J. Wakerly, Micro Systems Engineering'
ALARMCKT device 'P16V8C';

" Input pins
PANIC, ENABLEA, EXITING      pin 1, 2, 3;
WINDOW, DOOR, GARAGE         pin 4, 5, 6;
" Output pins
ALARM                        pin 11 istype 'com';

" Constant definition
X = .X.;

" Intermediate equation
SECURE = WINDOW & DOOR & GARAGE;

equations
ALARM = PANIC # ENABLEA & !EXITING & !SECURE;

test_vectors
([PANIC,ENABLEA,EXITING,WINDOW,DOOR,GARAGE] -> [ALARM])
[     1,     .X.,     .X.,    .X., .X.,    .X.] -> [    1];
[     0,       0,     .X.,    .X., .X.,    .X.] -> [    0];
[     0,       1,       1,    .X., .X.,    .X.] -> [    0];
[     0,       1,       0,      0, .X.,    .X.] -> [    1];
[     0,       1,       0,    .X.,   0,    .X.] -> [    1];
[     0,       1,       0,    .X., .X.,     0] -> [    1];
[     0,       1,       0,      1,   1,     1] -> [    0];

end Alarm_Circuit
```

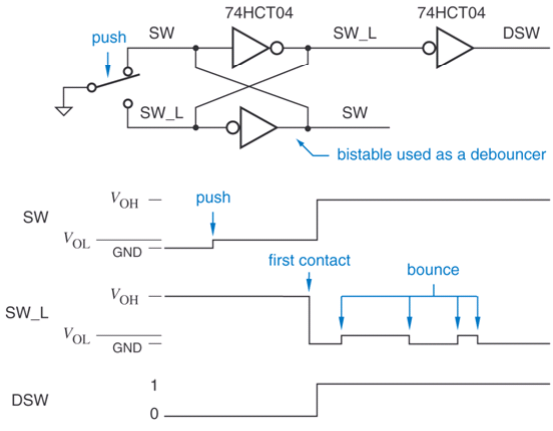
```
test_vectors  (input-list -> output-list)
              input-value -> output-value;
              ...
              input-value -> output-value;
```

For Reference

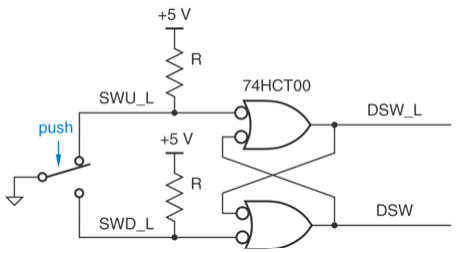
Input Without a Debouncer



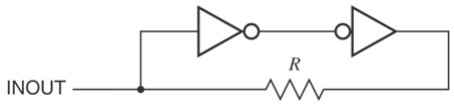
Debouncer with a Bi-stable



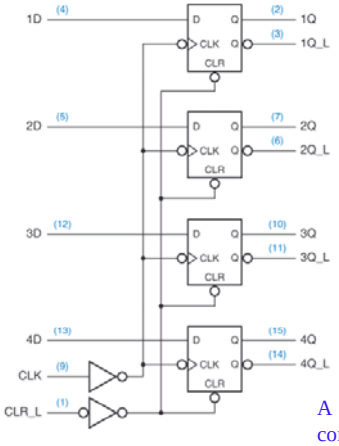
Debouncer with S-R Latch

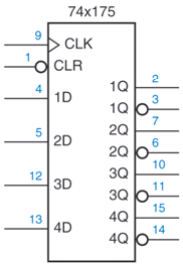


Bus Holder



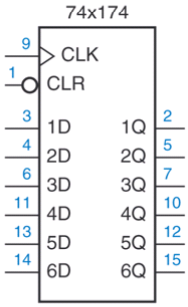
Multi-bit Registers & Latches



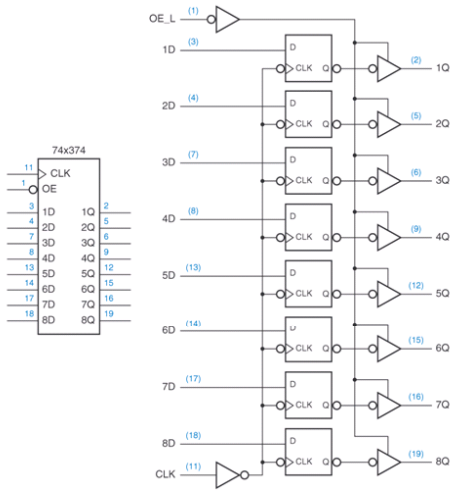


A collection of 2 or more D type flip flops with a common clock is called a register, registers are used to store collection of related bits. Constraint is that all the bits gets updated with the same clock.

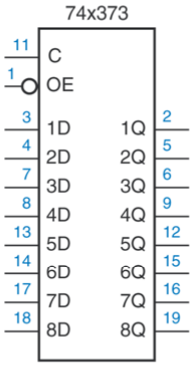
6 Bit Register



8 Bit Register

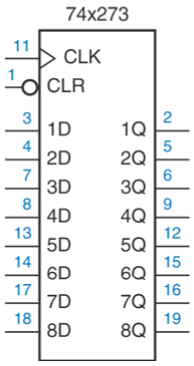


8 Bit Register without edge trigger



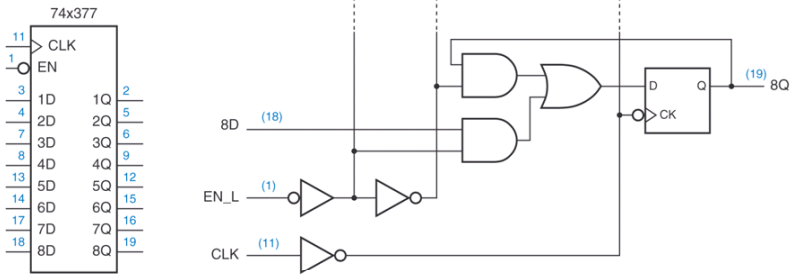
Logic symbol for the 74×373 8-bit latch.

8 Bit Register with clear & without 3 state output



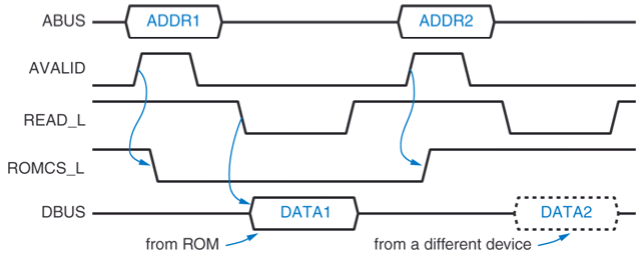
Logic symbol for the 74×273 8-bit register.

8 Bit Register with gated clock



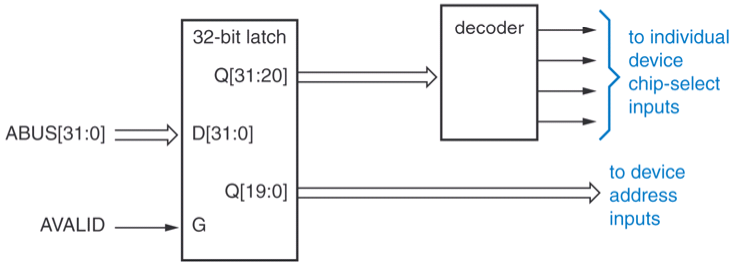
If EN_L is asserted then at the rising edge of clock the flip flops are loaded with the data inputs, otherwise they retain their present values.

Read Operation



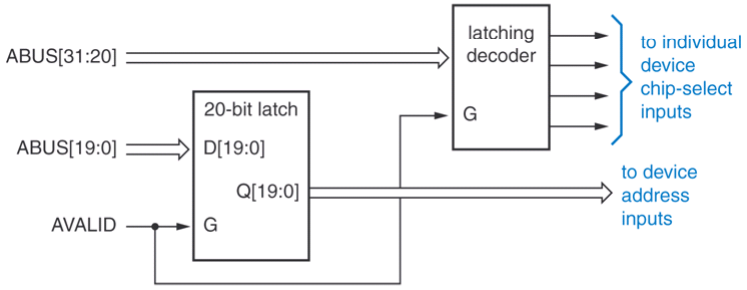
The reason of having latching the address is that typically address bus and data bus are multiplexed in 3 state logic to save pins. ROMCS_L is ROM chip select effective low.

Address Latching & Decoding



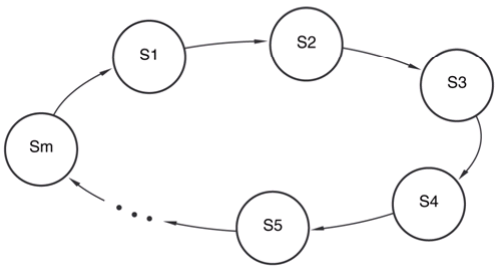
In this case the higher order bits (12 bits) are used for chip select, where as the 20 lower order bits are used for address location of the selected device.

Combined Address Latching & Decoding Circuit



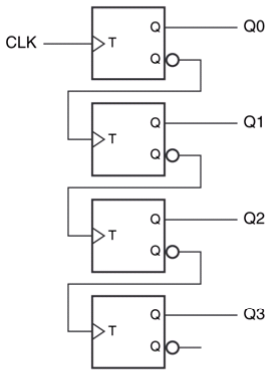
It may produce quick chip select for devices, higher order address bits are directly given to latching decoder and the address valid is also applied directly to latching decoder.

Counters



Clocked sequential circuit whose state diagram contains a single cycle, the modulus of the counter is the number of states in the cycle.

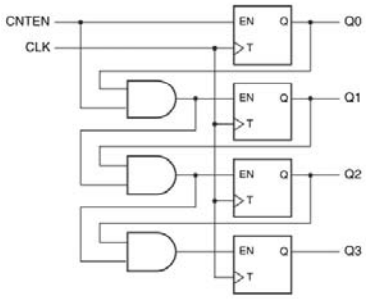
Binary Counter



Most commonly used counter is n bit binary counter. This is a ripple binary counter as carry is propagated to the next counter, one bit at a time.

T is the toggle flip flop which changes (toggles) its state at each clock pulse.

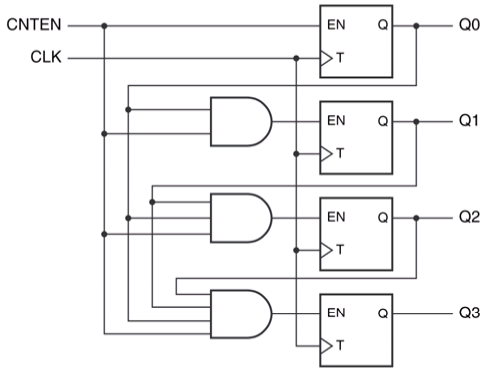
Synchronous Serial Counter



Ripple counters are slow and one can imagine if it is a large ripple counter the most significant output may not be valid because of the propagation delay.

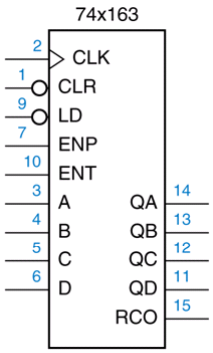
The counter shown is with a Serial Enable, it is called serial counter because the combinational enable signal propagate serial from LSB to MSB. It can have problem if the clock period is two short, it will not have enough time that LSB change is propagated to MSB

Synchronous Parallel Counter



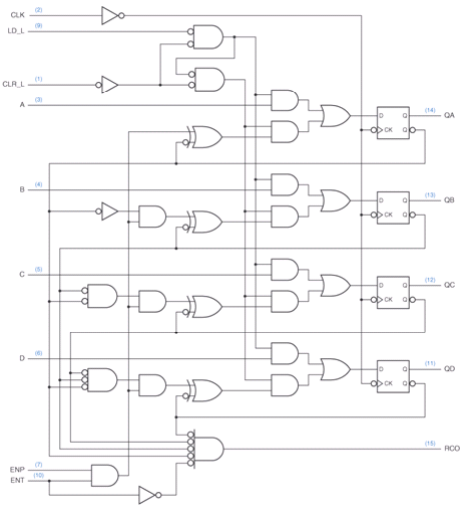
The counter with Parallel Enable, which eliminates the problem and this is the fastest binary counter structure

MSI Counters & Applications

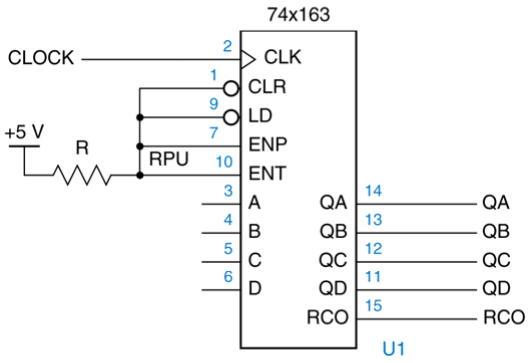


Inputs				Current State				Next State			
CLR_L	LD_L	ENT	ENP	QD	QC	QB	QA	QD+	QC+	QB+	QA+
0	x	x	x	x	x	x	x	0	0	0	0
1	0	x	x	x	x	x	x	D	C	B	A
1	1	0	x	x	x	x	x	QD	QC	QB	QA
1	1	x	0	x	x	x	x	QD	QC	QB	QA
1	1	1	1	0	0	0	0	0	0	0	1
1	1	1	1	0	0	0	1	0	0	1	0
1	1	1	1	0	0	1	0	0	0	1	1
1	1	1	1	0	0	1	1	0	1	0	0
1	1	1	1	0	1	0	0	0	1	0	1
1	1	1	1	0	1	0	1	0	1	1	0
1	1	1	1	0	1	1	0	0	1	1	1
1	1	1	1	0	1	1	1	1	0	0	0
1	1	1	1	1	0	0	0	1	0	0	1
1	1	1	1	1	0	0	1	1	0	1	0
1	1	1	1	1	0	1	0	1	0	1	1
1	1	1	1	1	0	1	1	1	1	0	0
1	1	1	1	1	1	0	0	1	1	0	1
1	1	1	1	1	1	0	1	1	1	1	0
1	1	1	1	1	1	1	0	1	1	1	1
1	1	1	1	1	1	1	1	0	0	0	0

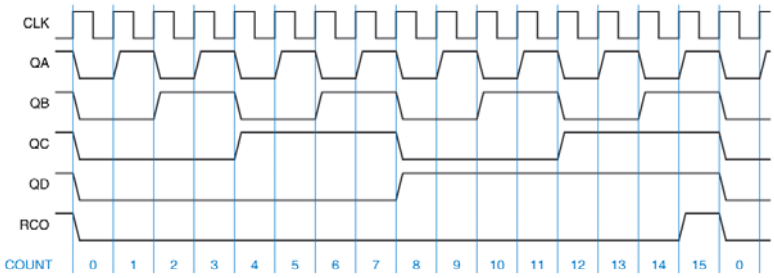
Synchronous 4 bit Binary Counter



Free Running Mode

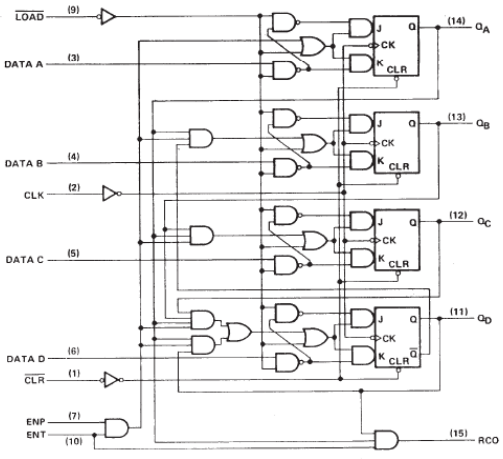


Free Running Divide by 16

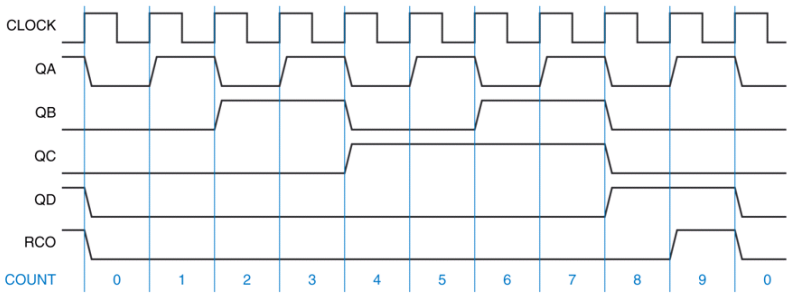


If you look at the 4 output signals it can be realised easily that each output is divided by 2 at each step, so the counter can be used as divide by 2, 4, 8 and 16 counter.

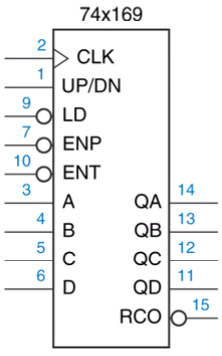
Decade Counter / Modulo 10 Counter



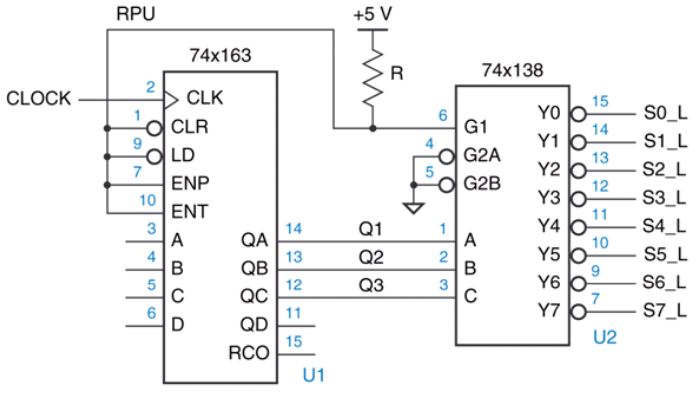
Free Running Divide by 10



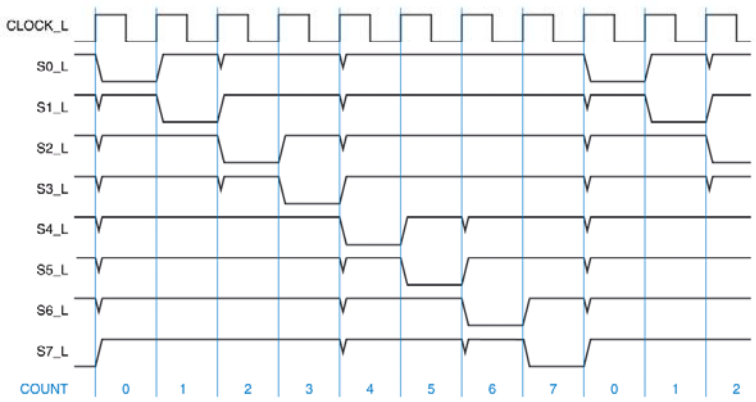
Up/Down Counter



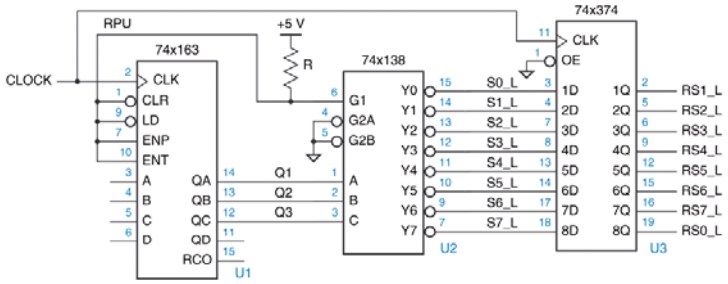
A modulo 8 Binary Counter & Decoder



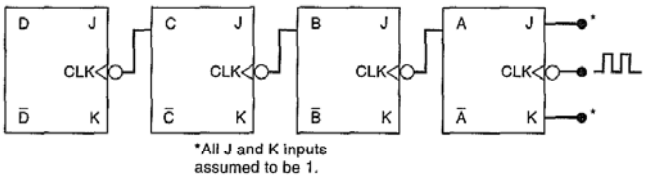
Decoding Binary-Counter States



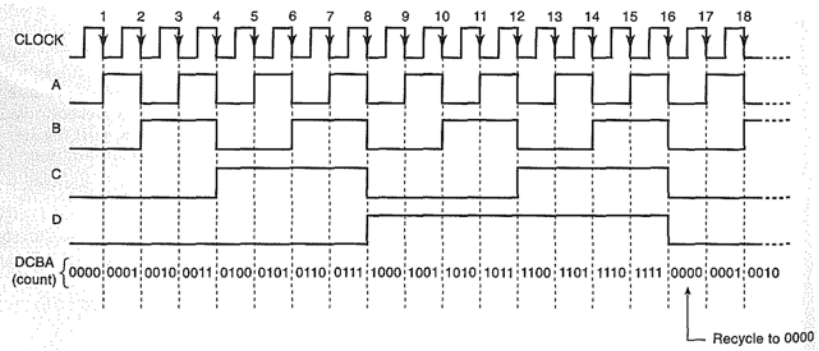
Glitch Free Outputs



4 Bit Ripple Counter (Asynchronous)

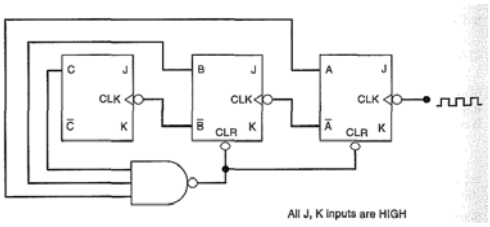


Clock & Output Sequence



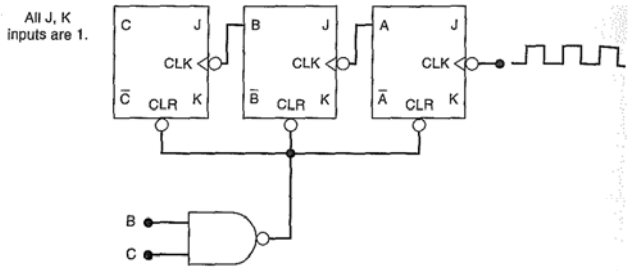
Drill Problems

Determine the state sequence for the following circuit



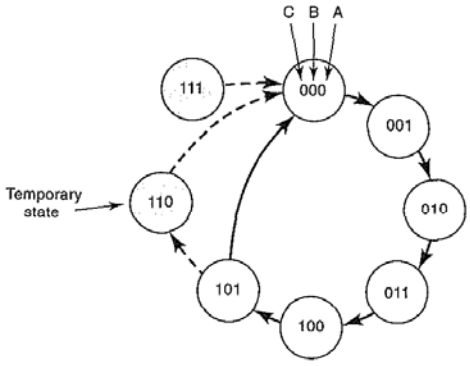
Drill Problems

What is the MOD of this counter



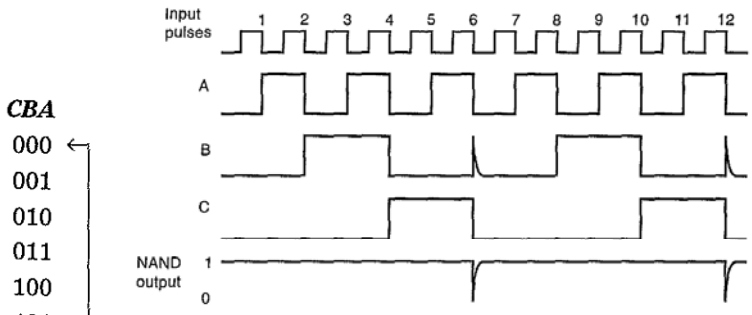
Drill Problems

State Transition Diagram



Drill Problems

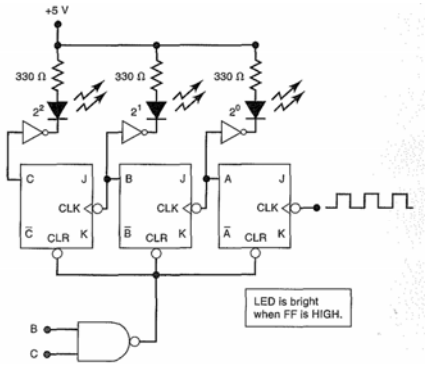
Output Sequence



CBA	
000	←
001	
010	
011	
100	
101	
110	→ (temporary state needed to clear counter)

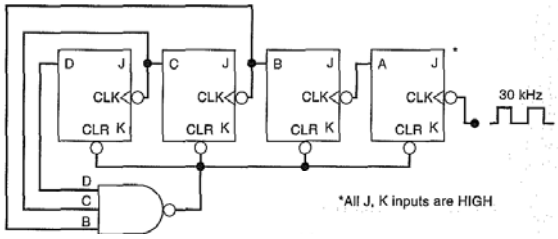
Drill Problems

Display for Code



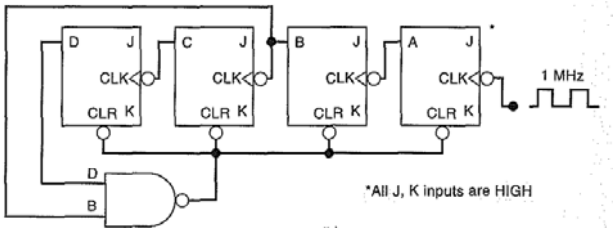
Drill Problems

Determine the MOD and determine the frequency at D output

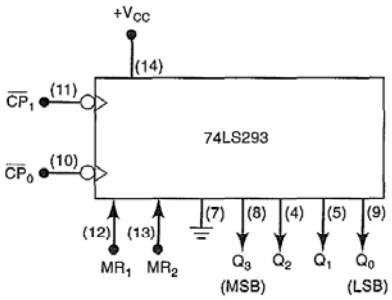


Drill Problems

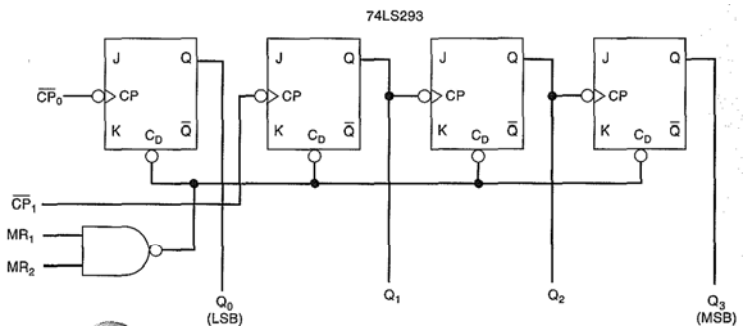
Determine the MOD and determine the frequency at D output



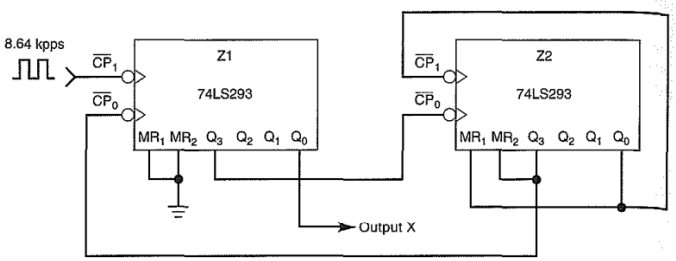
74LS93 Asynchronous Binary Counter



Logic Diagram for 74LS93 Asynchronous Binary Counter

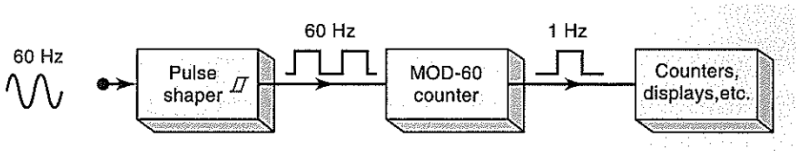


Determine the frequency at output X



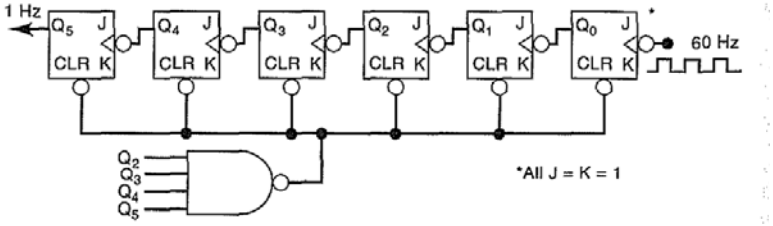
Drill Problems

Building Digital Clock



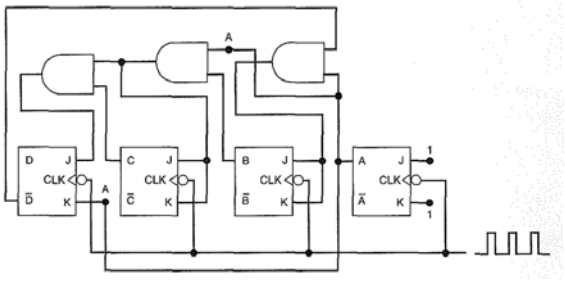
Drill Problems

MOD 60 Counter



Drill Problems

Determine the states the following circuit goes through




AALBORG UNIVERSITET
ESBJERG
Spørgsmål
Opgaver
DE3 Course
Dr. D. M. Akbar Hussain
Department of Electronic Systems
87