Design of Digital Systems II
Sequential Logic Design Practices (1)

Moslem Amiri, Václav Přenosil

Embedded Systems Laboratory
Faculty of Informatics, Masaryk University
Brno, Czech Republic

amiri@mail.muni.cz
prenosil@fi.muni.cz

Fall, 2014

Timing Diagrams and Specifications

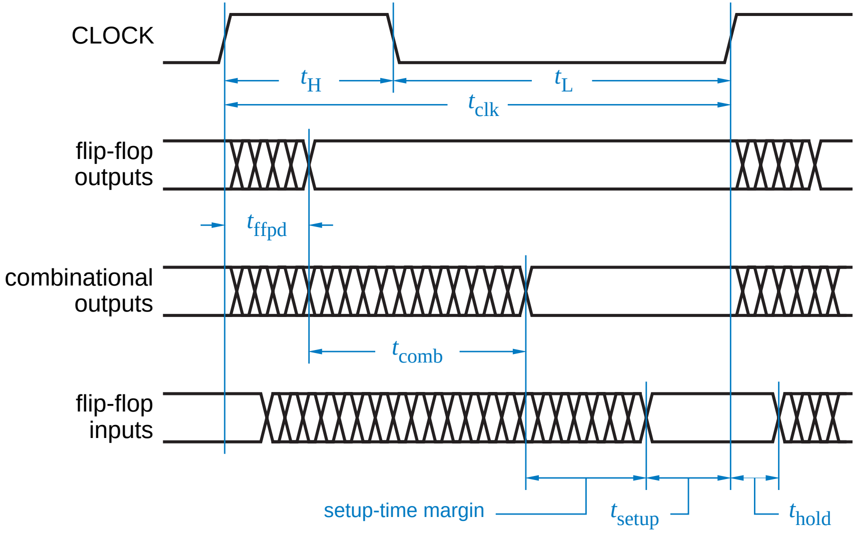


Figure 1: A detailed timing diagram showing propagation delays and setup and hold times with respect to the clock.

Timing Diagrams and Specifications

- In Fig. 1
  - First line shows system clock and its nominal timing parameters
  - Second line shows that flip-flops change their outputs at some time between rising edge of CLOCK and time t_{ffpd} afterward
    - External circuits that sample these signals should not do so while they are changing
  - Third line shows t_{comb} required for flip-flop output changes to propagate through combinational logic elements, such as flip-flop excitation logic
  - Excitation inputs of flip-flops and other clocked devices require a setup time of t_{setup} as shown in fourth line
  - For proper circuit operation: $t_{clk} - t_{ffpd} - t_{comb} > t_{setup}$
  - Timing margins indicate how much "worse than worst-case" the individual components of a circuit can be without causing circuit to fail
    - Well-designed systems have positive, nonzero timing margins
  - Setup-time margin: $t_{clk} - t_{ffpd(max)} - t_{comb(max)} - t_{setup}$
  - For proper circuit operation: $t_{ffpd(min)} + t_{comb(min)} > t_{hold}$
  - Hold-time margin: $t_{ffpd(min)} + t_{comb(min)} - t_{hold}$

Timing Diagrams and Specifications

- In most circuits, there are timing differences between different flip-flop inputs or combinational-logic signals
  - E.g., one flip-flop's Q output may be connected directly to another flip-flop's D input
    - t_{comb} for that path is zero, while another's may go through a long combinational path before reaching a flip-flop input
  - When proper synchronous design methodology is used, these relative timings are not critical, since none of these signals affect state of circuit until a clock edge occurs
  - Merely finding longest delay path in one clock period to determine whether circuit will work is enough
    - Requires analyzing several different paths in order to find worst-case one

Timing Diagrams and Specifications

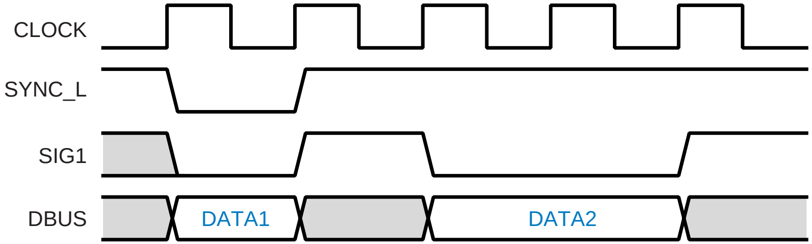


Figure 2: Functional timing of a synchronous circuit.

- Functional timing diagram shows only functional behavior and is not concerned with actual delay amounts
  - Lining up everything on clock edge allows timing diagram to display more clearly which functions are performed during each clock period
  - Shading or cross-hatching is used to indicate "don't-care" signal values

SSI Latches and Flip-Flops

- SSI latches and flip-flops have been eliminated to a large extent in modern designs as their functions are embedded in PLDs and FPGAs
  - Nevertheless, some of them still appear in many digital systems

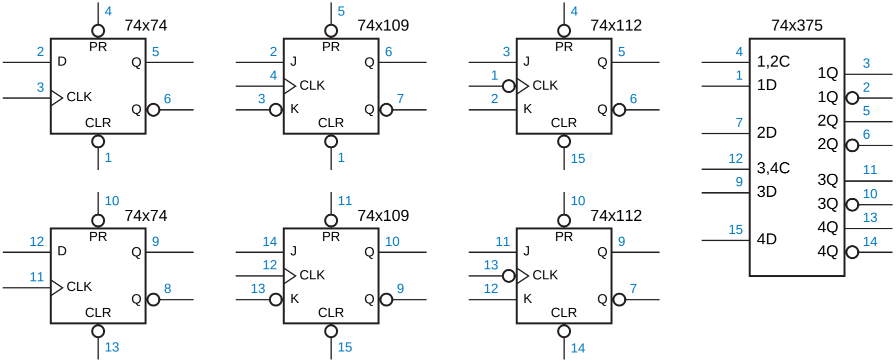


Figure 3: Pinouts for SSI latches and flip-flops.

SSI Latches and Flip-Flops

- In Fig. 3
  - The only latch is 74x375, which contains four D latches
    - Because of pin limitations, latches are arranged in pairs with a common C control line for each pair
  - The most important device is 74x74
    - It contains two independent positive-edge-triggered D flip-flops with preset and clear inputs
  - 74x109 is a positive-edge-triggered J-$\overline{\text{K}}$ flip-flop with an active-low K input
  - Another J-K flip-flop is 74x112, which has an active-low clock input

Switch Debouncing

- A common application of bistables and latches is switch debouncing
- Switches connected to sources of constant logic 0 and 1 are often used in digital systems to supply user inputs
- A simple make or break operation done by slow-moving humans, has several phases in high-speed digital logic

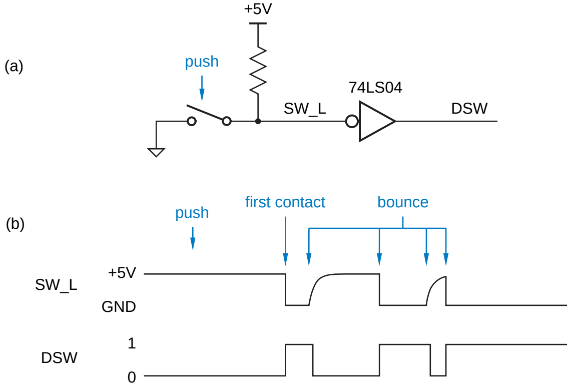


Figure 4: Switch input without debouncing.

Switch Debouncing

- Fig. 4 shows how a single-pole, single-throw (SPST) switch is used to generate a single logic input
  - After wiper hits bottom contact, it bounces a few times before finally settling
    - Results in several transitions on SW_L and DSW
    - This behavior is called **contact bounce**
  - Typical switches bounce for 10-20 ms, a very long time compared to switching speeds of logic gates
- Contact bounce is a problem if a switch is used to count or signal some event
  - We must provide a circuit to debounce switch

Switch Debouncing

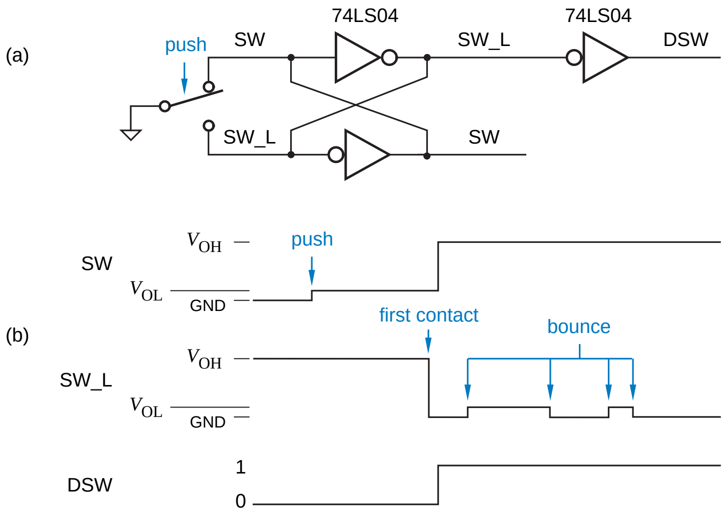


Figure 5: Switch input using a bistable for debouncing.

Switch Debouncing

- Fig. 5 shows a switch debouncing application for bistable element
  - This circuit uses a single-pole, double-throw (SPDT) switch
  - Before button is pushed
    - Top contact holds SW at 0 V $\longrightarrow$ a valid logic 0
  - When button is pushed and contact is broken
    - Feedback in bistable holds SW at V_{OL} $\longrightarrow$ still a valid logic 0
    - V_{OL} = output low voltage ($\leq$ 0.5 V for TTL)
  - When wiper hits bottom contact
    - Suddenly, SW_L is shorted to ground
    - A short time later, forced logic 0 on SW_L propagates through two inverters of bistable
    - At this point, top inverter output is no longer shorted to ground
    - Feedback in bistable maintains logic 0 on SW_L even if wiper bounces off bottom contact
  - Advantages of this circuit
    - It has a low chip count
    - No pull-up resistors are required
    - Both polarities of input signal (active-high and active-low) are produced
- In situations where momentarily shorting gate outputs must be avoided, a $\overline{\text{S}} - \overline{\text{R}}$ latch and pull-up resistors are used

Switch Debouncing

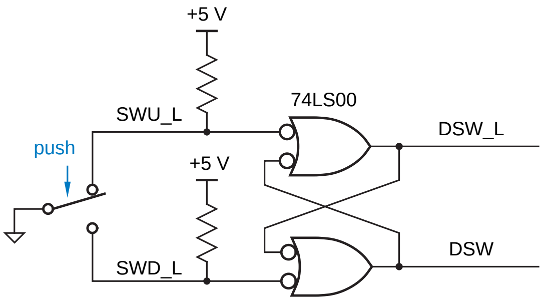


Figure 6: Switch input using an $\overline{S} - \overline{R}$ latch for debouncing.

Multibit Registers and Latches

- A collection of two or more D flip-flops with a common clock input is called a **register**

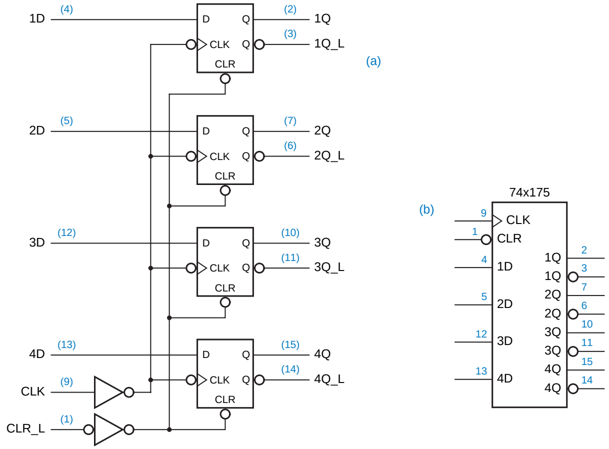


Figure 7: The 74x175 4-bit register: (a) logic diagram, including pin numbers for a standard 16-pin dual in-line package; (b) traditional logic symbol.

Multibit Registers and Latches

- In 74x175, both CLK and CLR_L are buffered before fanning out to four flip-flops
  - A device driving one of these inputs sees only one unit load instead of four
- 74x174 is similar to 74x175, except that it eliminates active-low outputs and provides two more flip-flops instead

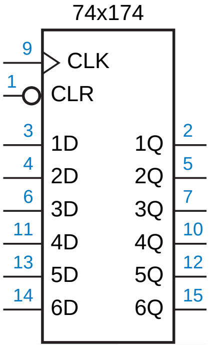


Figure 8: Logic symbol for the 74x174 6-bit register.

Multibit Registers and Latches

- Many digital systems process information 8, 16, or 32 bits at a time
  - ICs that handle eight bits are very popular

Multibit Registers and Latches

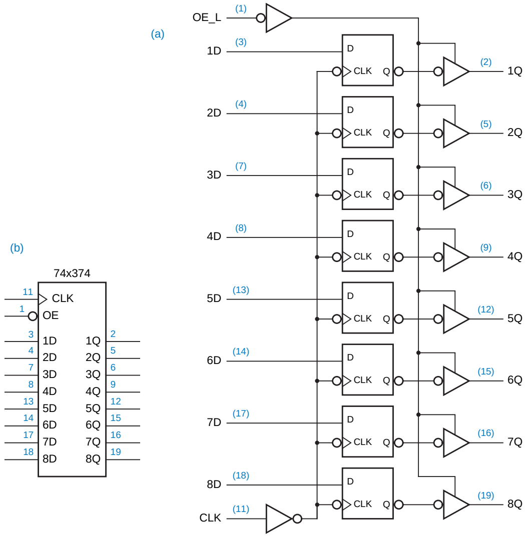


Figure 9: The 74x374 8-bit register: (a) logic diagram, including pin numbers for a standard 20-pin dual in-line package; (b) traditional logic symbol.

Multibit Registers and Latches

- 74x374
  - It contains eight edge-triggered D flip-flops that all sample their inputs and change their outputs on rising edge of a common CLK input
  - Each flip-flop output drives a three-state buffer that in turn drives an active-high output
  - All of three-state buffers are enabled by a common active-low OE_L (output enable) input
  - Control inputs (CLK and OE_L) are buffered so that they present only one unit load to a device that drives them

Multibit Registers and Latches

- 74x373 is a variation of 74x374 which uses D latches instead of edge-triggered flip-flops
  - Its outputs follow corresponding inputs whenever C is asserted and latch the last input values when C is negated

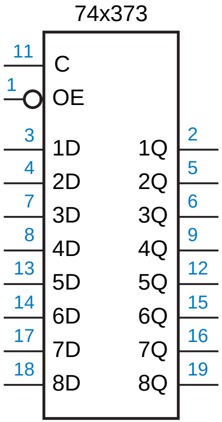


Figure 10: Logic symbol for the 74x373 8-bit latch.

Multibit Registers and Latches

- 74x273 is another variation of 74x374 which has non-three-state outputs and no OE_L input
  - It uses pin 1 for an asynchronous clear input CLR_L

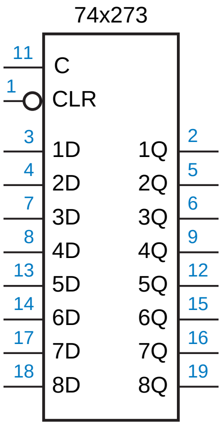


Figure 11: Logic symbol for the 74x273 8-bit register.

Multibit Registers and Latches

- 74x377 is an edge-triggered register like ’374, but it does not have three-state outputs
  - Instead, pin 1 is used as an active-low clock enable input EN_L
  - If EN_L is asserted (LOW) at rising edge of clock, flip-flops are loaded from data inputs; otherwise, they retain their present values

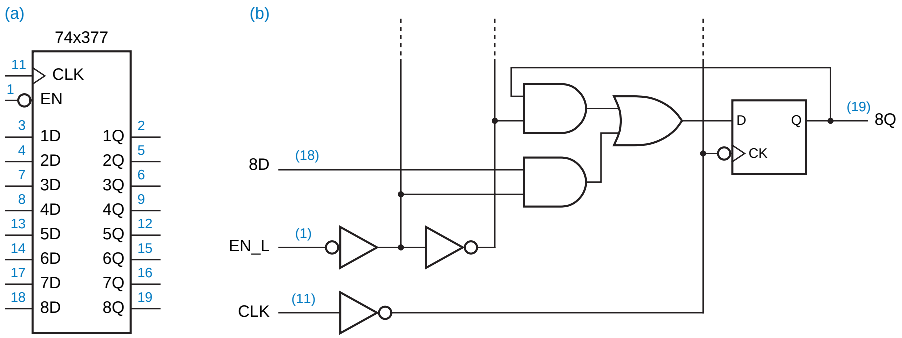


Figure 12: The 74x377 8-bit register with gated clock: (a) logic symbol; (b) logical behavior of one bit.

Registers and Latches in Verilog

Table 1: Verilog behavioral module for a D latch.

```
module VrDlatch( C, D, Q, QN );
  input C, D;
  output Q, QN;
  reg Q, QN;

  always @ (C or D or Q) begin
    if (C==1) Q <= D; else Q <= Q;
    QN <= !Q;
  end
endmodule
```

- Tab. 1
  - We could omit "`else Q <= Q`" clause and get the same results
  - Such code would not say what to do when `C` is 0, so compiler would infer a latch
  - It is better coding style to use an explicit `else` clause for "latch closed" case

Registers and Latches in Verilog

Table 2: Behavioral Verilog for a positive-edge-triggered D flip-flop.

```
module VrDff(CLK, D, Q);
  input CLK, D;
  output Q;
  reg Q;

  always @ (posedge CLK)
    Q <= D;
endmodule
```

- To describe edge-triggered behavior in a flip-flop, we need to use Verilog's `posedge` or `negedge` keyword in sensitivity list of an `always` statement

Registers and Latches in Verilog

Table 3: Verilog module for a 16-bit register with many features.

```
module Vrreg16( CLK, CLKEN, OE_L, CLR_L, D, Q );
  input CLK, CLKEN, OE_L, CLR_L;
  input [1:16] D;
  output [1:16] Q;
  reg [1:16] IQ;

  always @ (posedge CLK or negedge CLR_L)
    if (CLR_L==0) IQ <= 16'b0;
    else if (CLKEN==1) IQ <= D;
    else IQ <= IQ;

  assign Q = (OE_L==0) ? IQ : 16'bz;

endmodule
```

- Registers can be modeled by defining data inputs and outputs to be vectors, and additional functions can be included
- Tab. 3
  - Models a 16-bit register with three-state outputs and clock-enable, output-enable, and clear inputs

Sequential PLDs: Bipolar Sequential PLDs

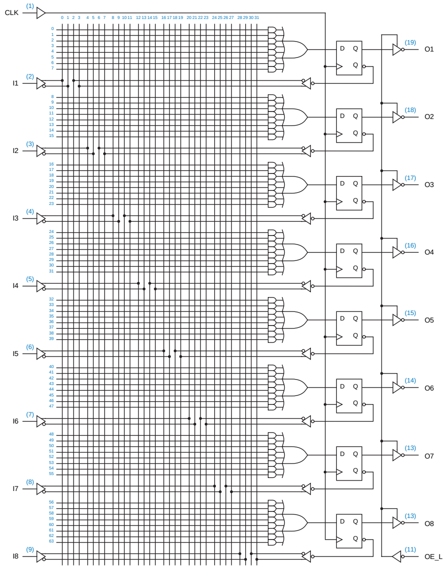


Figure 13: PAL16R8 logic diagram.

Sequential PLDs: Bipolar Sequential PLDs

- PAL16R8
    - It is representative of first generation of sequential PLDs, which used bipolar (TTL) technology
    - It has eight primary inputs, eight outputs, and common clock and output-enable inputs, and fits in a 20-pin package
    - It has edge-triggered D flip-flops between AND-OR array and its eight outputs, O1-O8
    - Each flip-flop drives an output pin through a 3-state buffer
    - Registered output pins contain complement of signal produced by AND-OR array
    - Possible inputs to AND-OR array are eight primary inputs (I1-I8) and eight D flip-flop outputs
    - Connection from D flip-flop outputs into AND-OR array makes it easy to design shift-registers, counters, and general state machines
    - D flip-flop outputs are available to AND-OR array whether or not O1-O8 three-state drivers are enabled
        - Internal flip-flops can go to a next state that is a function of current state even when O1-O8 outputs are disabled

Sequential PLDs: Bipolar Sequential PLDs

- Many applications require combinational as well as sequential PLD outputs
  - There are a few variants of PAL16R8 without D flip-flops on some output pins
- PAL16R6
  - It has only six registered outputs
  - Two pins, IO1 and IO8, are bidirectional
    - They serve both as inputs and as combinational outputs with separate 3-state enables
  - Possible inputs to AND-OR array are eight primary inputs (I1-I8), six D flip-flop outputs, and two bidirectional pins (IO1, IO8)

Sequential PLDs: Bipolar Sequential PLDs

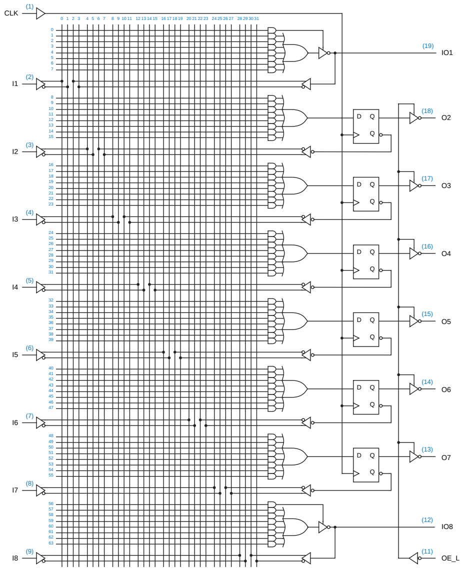


Figure 14: PAL16R6 logic diagram.

Sequential PLDs: Sequential GAL Devices

- GAL16V8 electrically erasable PLD
  - Two "architecture-control" fuses are used to select among three basic configurations of this device
    1. 16V8C ("complex") configuration, which was introduced in combinational section before
    2. 16V8S ("simple") configuration, which provides a slightly different combinational logic capability
    3. 16V8R ("registered") configuration, which allows a flip-flop to be provided on any or all of outputs

Sequential PLDs: Sequential GAL Devices

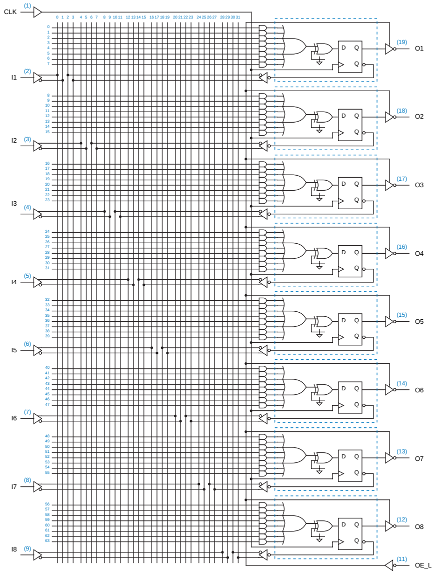


Figure 15: 16V8R logic diagram.

Sequential PLDs: Sequential GAL Devices

- In Fig. 15
  - Circuitry inside each dotted box is called an **output logic macrocell**
  - Each macrocell may be individually configured to bypass flip-flop to produce a combinational output
  - It is possible to program the device to have any set of registered and combinational outputs, up to eight total

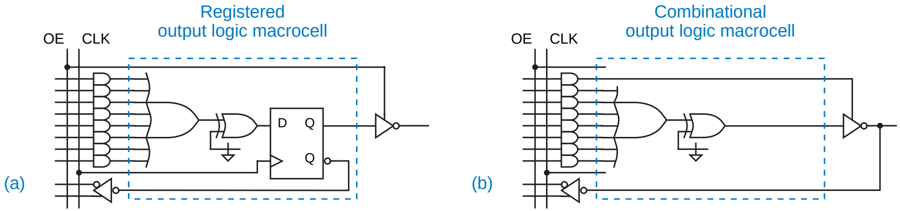


Figure 16: Output logic macrocells for the 16V8R: (a) registered; (b) combinational.

Sequential PLDs: Sequential GAL Devices

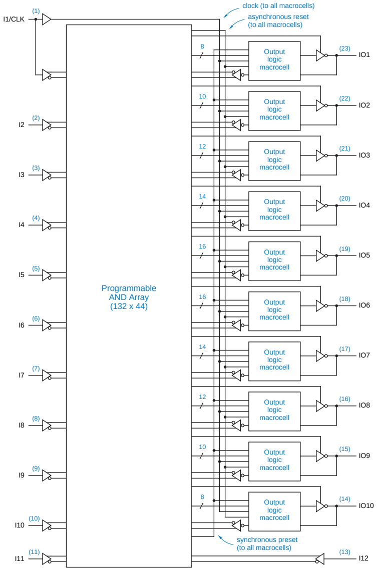
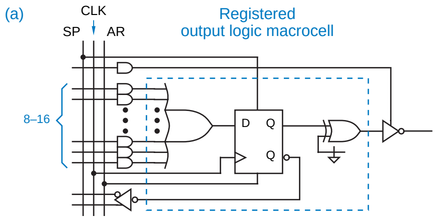


Figure 17: Logic diagram for the 22V10.

Sequential PLDs: Sequential GAL Devices



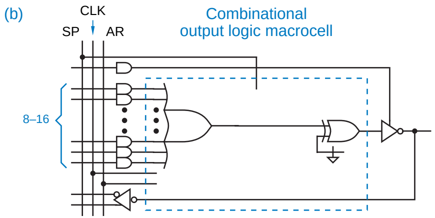


Figure 18: Output logic macrocells for the 22V10: (a) registered; (b) combinational.

- 22V10
  - It does not have "architecture control" bits like 16V8's, but it can realize any function that is realizable with a 16V8, and more
  - Each output logic macrocell is configurable to have a register or not
  - A single product term controls output buffer
  - Every output has at least eight product terms available
    - More product terms are available on inner pins, with 16 available on each of two innermost pins

Sequential PLDs: Sequential GAL Devices

- 22V10
  - Clock signal on pin 1 is also available as a combinational input to any product term
  - A single product term is available to generate a global, asynchronous reset signal that resets all internal flip-flops to 0
  - A single product term is available to generate a global, synchronous preset signal that sets all internal flip-flops to 1 on rising edge of clock
  - It has programmable output polarity
    - However, in registered configuration, polarity change is made at output of D flip-flop. This affects details of programming when polarity is changed but does not affect overall capability of 22V10

Counters

- A **counter** is a clocked sequential circuit whose state diagram contains a single cycle
- **Modulus** of a counter is the number of states in the cycle
- A counter with *m* states is **modulo-m counter** or a **divide-by-m counter**
- The most commonly used counter type is an **n-bit binary counter**
    - It has *n* flip-flops and 2^n states

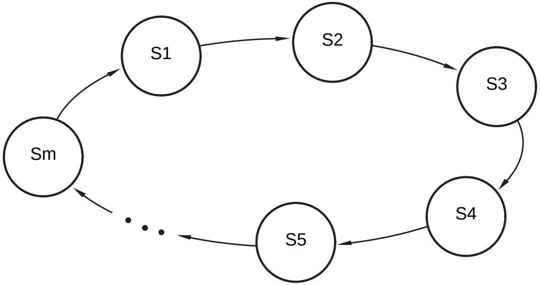


Figure 19: General structure of a counter's state diagram—a single cycle.

Counters: Ripple Counters

- An *n*-bit binary counter can be constructed with just *n* flip-flops
- In Fig. 20, each bit of counter toggles if and only if the immediately preceding bit changes from 1 to 0
  - This corresponds to a normal binary counting sequence
  - When a particular bit changes from 1 to 0, it generates a carry to next most significant bit

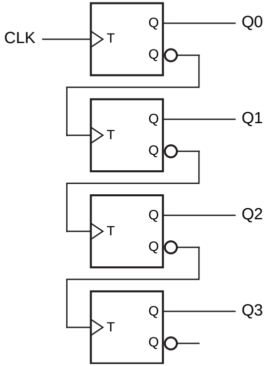


Figure 20: A 4-bit binary ripple counter.

Counters: Synchronous Counters

- A ripple counter requires fewer components than any other type of binary counter
  - But it is slower than any other type of binary counter
- A **synchronous counter** uses T flip-flops with enable inputs

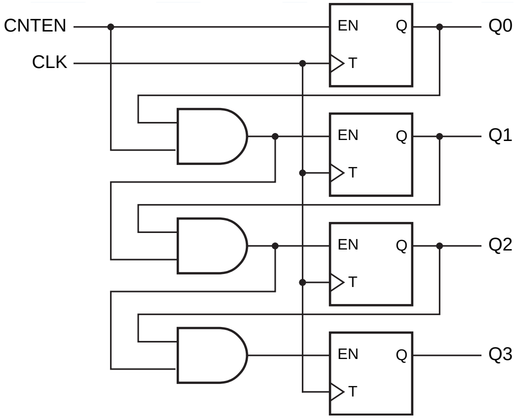


Figure 21: A synchronous 4-bit binary counter with serial enable logic.

Counters: Synchronous Counters

- In Fig. 21
  - All of flip-flop clock inputs are connected to same common CLK signal
    - All of flip-flop outputs change at same time
  - CNTEN is a master count-enable signal
  - Each T flip-flop toggles if and only if CNTEN is asserted and all of lower-order counter bits are 1
  - It is called a **synchronous serial counter** because combinational enable signals propagate serially from least significant to most significant bits
  - If clock period is too short, there may not be enough time for a change in counter's LSB to propagate to MSB
  - This problem is eliminated in **synchronous parallel counters**
- A synchronous parallel counter is the fastest binary counter structure

Counters: Synchronous Counters

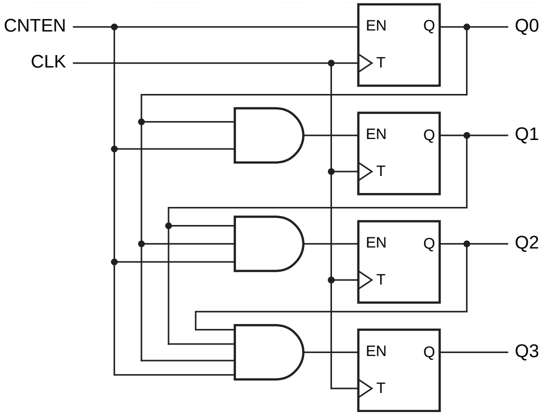


Figure 22: A synchronous 4-bit binary counter with parallel enable logic.

Counters: MSI Counters and Applications

- The most popular MSI counter is 74x163, a synchronous 4-bit binary counter with active-low load and clear inputs

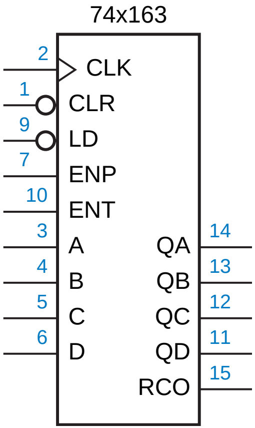


Figure 23: Traditional logic symbol for the 74x163.

Counters: MSI Counters and Applications

Table 4: State table for a 74x163 4-bit binary counter.

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

Counters: MSI Counters and Applications

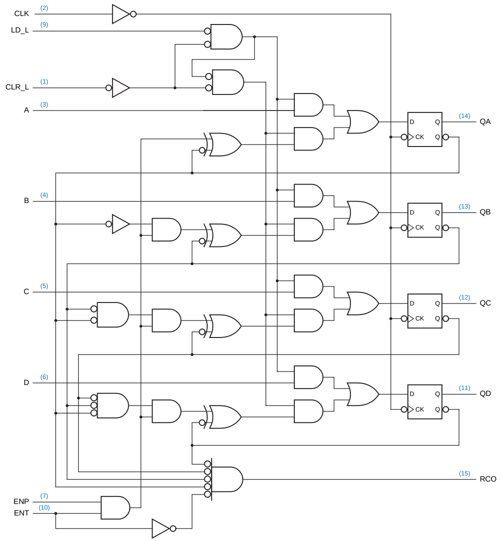


Figure 24: Logic diagram for the 74x163 synchronous 4-bit binary counter, including pin numbers for a standard 16-pin dual in-line package.

Counters: MSI Counters and Applications

- 74x163
  - It uses D rather than T flip-flops to facilitate load and clear functions
  - Each D input is driven by a 2-input multiplexer consisting of an OR gate and two AND gates
  - Multiplexer output is 0 if CLR_L input is asserted, otherwise, top AND gate passes data input (A, B, C, or D) to output if LD_L is asserted
  - If neither CLR_L nor LD_L is asserted, bottom AND gate passes output of an XNOR gate to multiplexer output
  - XNOR gates perform counting function
    - One input of each XNOR is the corresponding count bit (QA, QB, QC, or QD)
    - Other input is 1, which complements count bit, if and only if both enables ENP and ENT are asserted and all of lower-order count bits are 1
  - RCO (ripple carry out) signal indicates a carry from most significant bit position
    - It is 1 when all of count bits are 1 and ENT is asserted

Counters: MSI Counters and Applications

- Even though most MSI counters have enable inputs, they are often used in a free-running mode in which they are enabled countinuously

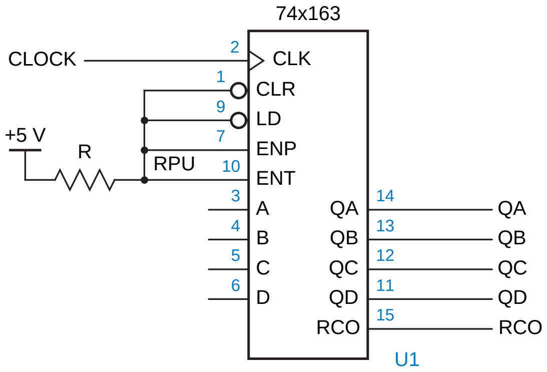


Figure 25: Connections for the 74x163 to operate in a free-running mode.

Counters: MSI Counters and Applications

- Fig. 26 shows resulting output waveforms for a free-running '163
  - Starting with QA, each signal has half frequency of preceding one
  - A free-running '163 can be used as a divide-by-2, -4, -8, or -16 counter, by ignoring any unnecessary high-order output bits

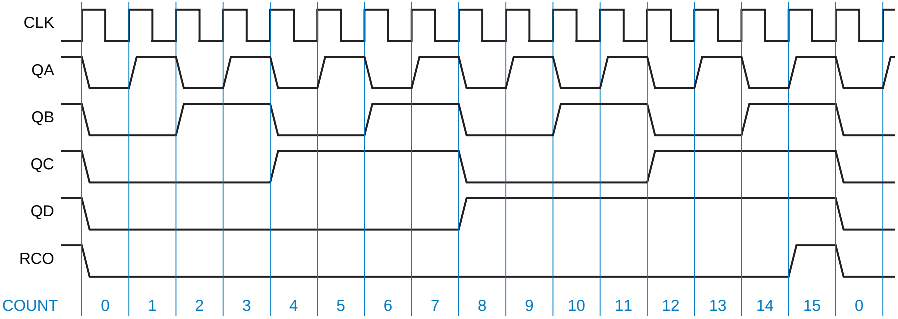


Figure 26: Clock and output waveforms for a free-running divide-by-16 counter.

Counters: MSI Counters and Applications

- ’163 is fully synchronous
  - Its outputs change only on rising edge of CLK
  - 74x161 has same pinout but provides an asynchronous clear function; its CLR_L input is connected to asynchronous clear inputs of its flip-flops
- 74x160 and 74x162 have same pinouts and functions as ’161 and ’163
  - Except that counting sequence is modified to go to state 0 after state 9
  - These are modulo-10 counters, called **decade counters**

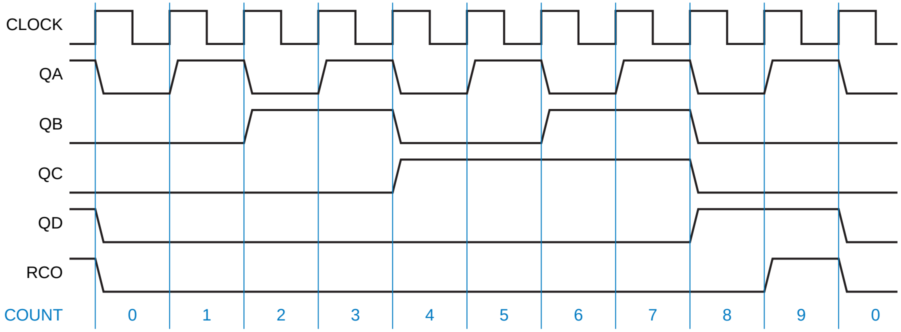


Figure 27: Clock and output waveforms for a free-running divide-by-10 counter.

Counters: MSI Counters and Applications

- In Fig. 27, although QD and QC outputs have one-tenth of CLK frequency, they do not have a 50% duty cycle
- '163 is a modulo-16 counter, but it can be made to count in a modulus less than 16
  - Use CLR_L or LD_L input to shorten normal counting sequence

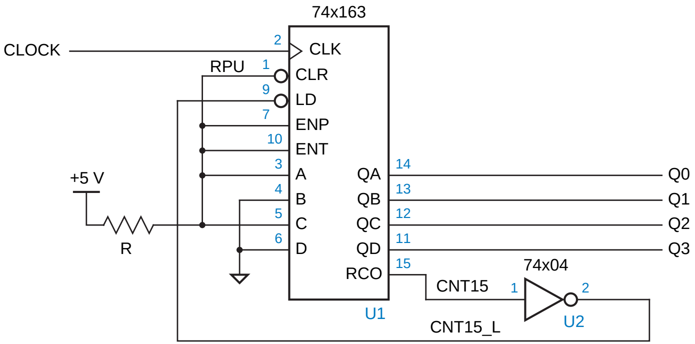


Figure 28: Using the 74x163 as a modulo-11 counter with the counting sequence $5, 6, \ldots, 15, 5, 6, \ldots$.

Counters: MSI Counters and Applications

- Fig. 28 shows one way of using '163 as a modulo-11 counter
    - RCO output, which detects state 15, is used to force next state to 5
    - Circuit counts from 5 to 15, for a total of 11 states per counting cycle
- Fig. 29 shows a different approach for modulo-11 counting with '163

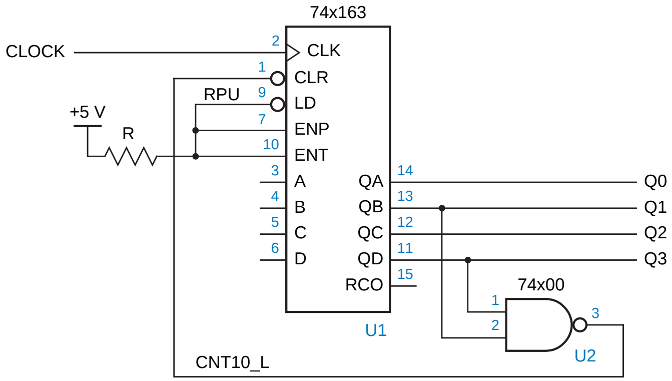


Figure 29: Using the 74x163 as a modulo-11 counter with the counting sequence $0, 1, 2, \ldots, 10, 0, 1, \ldots$.

Counters: MSI Counters and Applications

- In general, to detect state N in a binary counter that counts from 0 to N, we need to AND only state bits that are 1 in binary encoding of N
- **Excess-3 code** word for each decimal digit is the corresponding BCD code word plus 0011_2
  - Because excess-3 code words follow a standard binary counting sequence, standard binary counters can easily be made to count in excess-3 code

Table 5: Decimal codes.

Decimal digit	BCD (8421)	Excess-3
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

Counters: MSI Counters and Applications

- In Fig. 30, a NAND gate detects state 1100 and forces 0011 to be loaded as next state

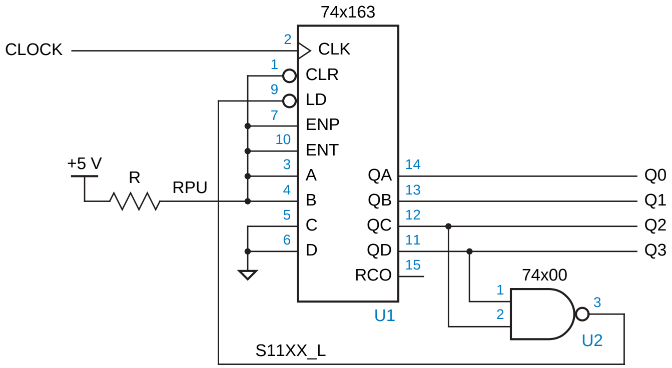


Figure 30: A 74x163 used as an excess-3 decimal counter.

Counters: MSI Counters and Applications

- In Fig. 31, Q3 output has a 50% duty cycle, which may be desirable for some applications

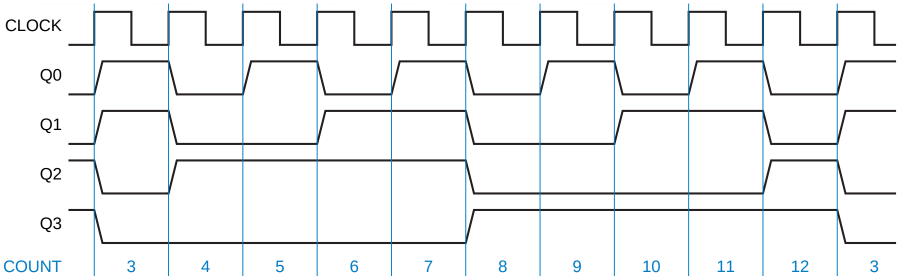


Figure 31: Timing waveforms for the '163 used as an excess-3 decimal counter.

Counters: MSI Counters and Applications

- A binary counter with a modulus greater than 16 can be built by cascading 74x163s as in Fig. 32
  - In Fig. 32, RCO4 output is asserted if and only if low-order '163 is in state 15 *and* CNTEN, master count-enable, is asserted
  - Scheme of Fig. 32 can be extended to build a counter with any desired number of bits

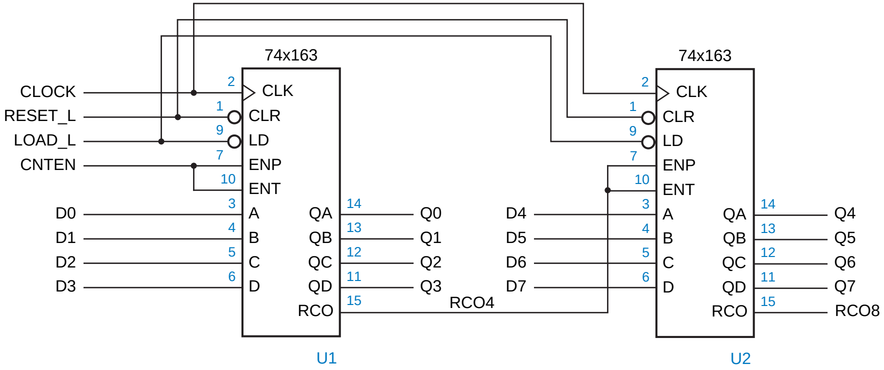


Figure 32: General cascading connections for 74x163-based counters.

Counters: MSI Counters and Applications

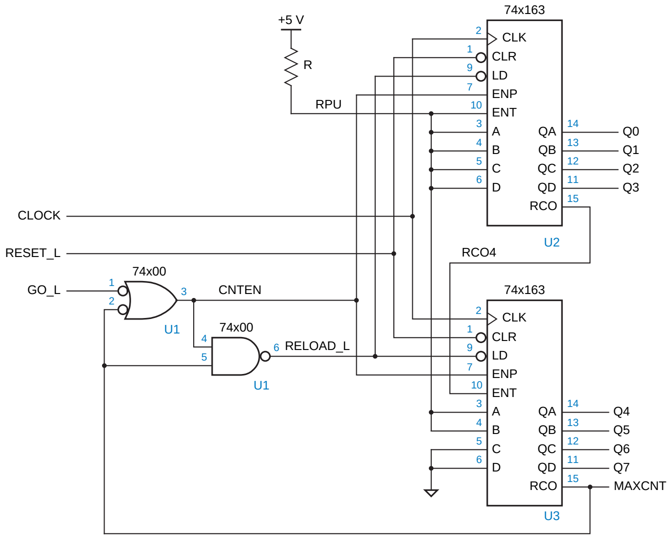


Figure 33: Using 74x163s as a modulo-193 counter with the counting sequence $63, 64, \ldots, 255, 63, 64, \ldots$.

Counters: MSI Counters and Applications

- Fig. 33
  - It is a modulo-193 counter that counts from 63 to 255
  - MAXCNT output detects state 255 and stops counter until GO_L is asserted
  - When GO_L is asserted, counter is reloaded with 63 and counts up to 255 again
    - Value of GO_L is relevant only when counter is in state 255
  - To keep counter stopped, MAXCNT must be asserted in state 255 even while counter is stopped
    - In Fig. 24, both ENP and ENT enable inputs must be asserted for counter to count. However, ENT goes to ripple carry output as well
    - Therefore, in Fig. 33, low-order counter's ENT input is always asserted, its RCO output is connected to high-order ENT input, and MAXCNT detects state 255 even if CNTEN is not asserted
  - To enable counting, CNTEN is connected to ENP inputs in parallel
  - A NAND gate asserts RELOAD_L to go back to state 63 only if GO_L is asserted and counter is in state 255

Counters: MSI Counters and Applications

- Another counter with functions similar to 74x163's is 74x169
  - '169 is an **up/down counter**
  - It counts in ascending or descending binary order depending on value of an input signal, UP/DN
  - '169 counts up when UP/DN is 1 and down when UP/DN is 0

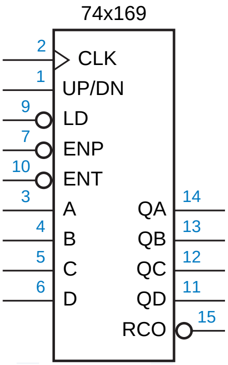


Figure 34: Logic symbol for the 74x169 up/down counter.

Counters: Decoding Binary-Counter States

- A binary counter may be combined with a decoder to obtain a set of 1-out-of-*m*-coded signals, where one signal is asserted in each counter state
  - This is useful when counters are used to control a set of devices where a different device is enabled in each counter state

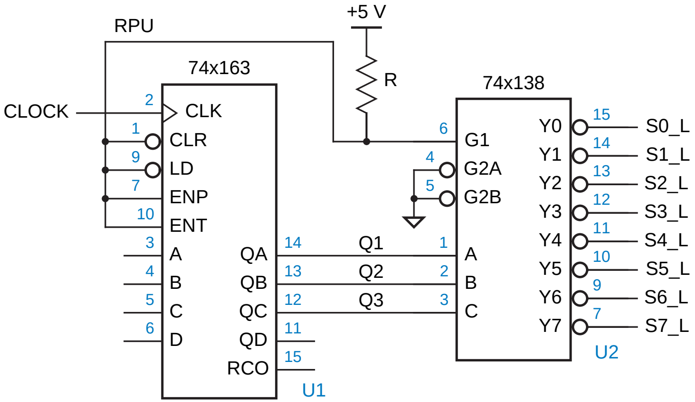


Figure 35: A modulo-8 binary counter and decoder.

Counters: Decoding Binary-Counter States

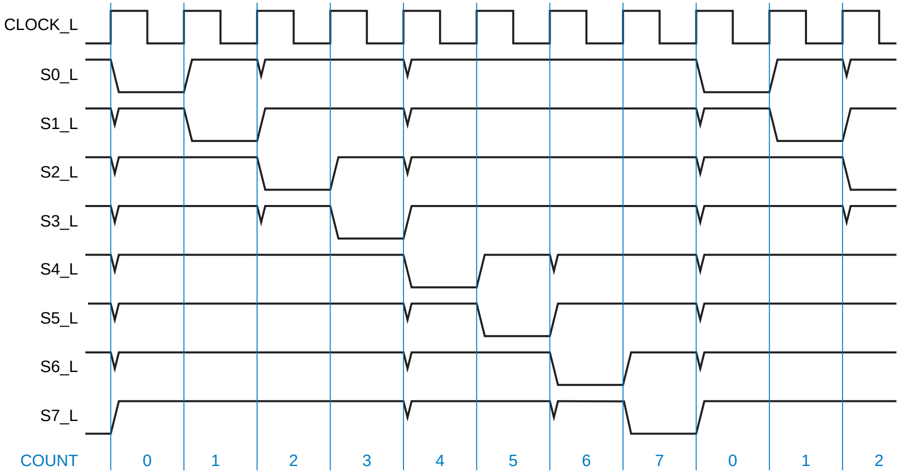


Figure 36: Timing diagram for a modulo-8 binary counter and decoder, showing decoding glitches.

Counters: Decoding Binary-Counter States

- Fig. 36
  - Decoder outputs may contain glitches on state transitions where two or more counter bits change, even though '163 outputs are glitch free and '138 does not have any static hazards
  - In a synchronous counter like '163, outputs don't change at exactly the same time
    - Also, multiple signal paths in a decoder like '138 have different delays
    - E.g., path from B to Y1_L is faster than path from A to Y1_L
    - Thus, even if input changes simultaneously from 011 to 100, decoder may behave as if input were temporarily 001, and Y1_L output may have a glitch
  - In most applications, decoder output signals are used as control inputs to registers, counters, and other edge-triggered devices
    - In such a case, decoding glitches are not a problem
    - They occur *after* clock tick
  - Glitches would be a problem if they were applied to something like inputs of an $\overline{\mathsf{S}} - \overline{\mathsf{R}}$ latch

Counters: Decoding Binary-Counter States

- One way to clean up glitches in Fig. 36 is to connect '138 outputs to another register that samples stable decoded outputs on next clock tick
  - A less costly solution is to use an 8-bit "ring counter" which provides glitch-free decoded outputs directly

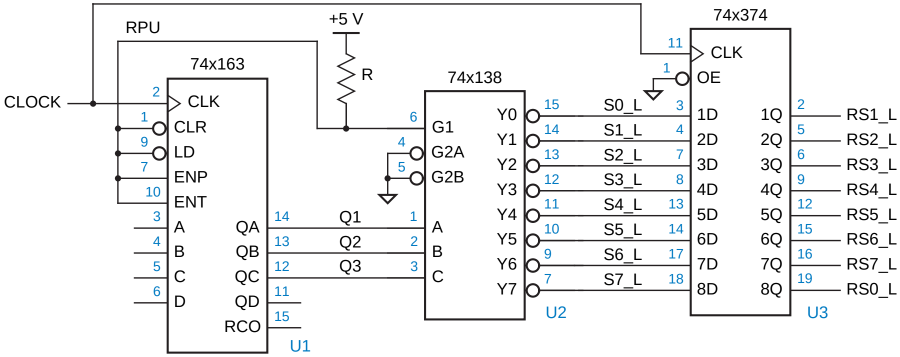


Figure 37: A modulo-8 binary counter and decoder with glitch-free outputs.

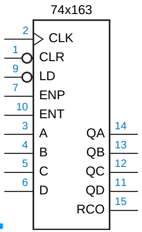
Counters in Verilog

Table 6: Verilog module for a 74x163-like 4-bit binary counter.

```
module Vr74x163( CLK, CLR_L, LD_L, ENP, ENT, D, Q, RCO );
  input CLK, CLR_L, LD_L, ENP, ENT;
  input [3:0] D;
  output [3:0] Q;
  output RCO;
  reg [3:0] Q;
  reg RCO;

  always @ (posedge CLK)  // Create the counter f-f behavior
    if (CLR_L == 0)                         Q <= 4'b0;
    else if (LD_L == 0)                     Q <= D;
    else if ((ENT == 1) && (ENP == 1)) Q <= Q + 1;
    else                                    Q <= Q;

  always @ (Q or ENT)     // Create RCO combinational output
    if ((ENT == 1) && (Q == 4'd15))     RCO = 1;
    else                                RCO = 0;
endmodule
```



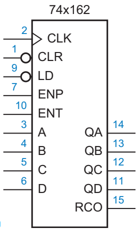
- In Tab. 6, usual state-machine coding style is not used
  - Since next-state logic is simple, it is put in the same `always` block with the edge-triggered flip-flop behavior

Counters in Verilog

Table 7: Verilog code for a 74x162-like 4-bit decimal counter.

```
always @ (posedge CLK)   // Create the counter f-f behavior
  if (!CLR_L)                          Q <= 4'b0;
  else if (!LD_L)                      Q <= D;
  else if (ENT && ENP && (Q == 4'd9))  Q <= 4'b0;
  else if (ENT && ENP)                 Q <= Q + 1;
  else                                 Q <= Q;

always @ (Q or ENT)      // Create RCO combinational output
  if (ENT && (Q == 4'd9))              RCO = 1;
  else                                 RCO = 0;
```

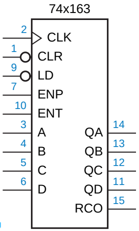


Counters in Verilog

Table 8: Verilog code for the excess-3 decimal counting sequence.

```
always @ (posedge CLK)   // Create the counter f-f behavior
  if (!CLR_L)                             Q <= 4'd3;
  else if (!LD_L)                         Q <= D;
  else if (ENT && ENP && (Q == 4'd12))    Q <= 4'd3;
  else if (ENT && ENP)                    Q <= Q + 1;
  else                                    Q <= Q;

always @ (Q or ENT)      // Create RCO combinational output
  if (ENT && (Q == 4'd12))                RCO = 1;
  else                                    RCO = 0;
```

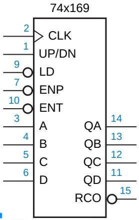


Counters in Verilog

Table 9: Verilog code for a 74x169-like 4-bit up/down counter.

```
always @ (posedge CLK)        // Create the counter f-f behavior
  if (!CLR_L )                                 Q <= 4'b0;
  else if (!LD_L)                              Q <= D;
  else if (!ENT_L && !ENP_L &&  UPDN)          Q <= Q + 1;
  else if (!ENT_L && !ENP_L && !UPDN)          Q <= Q - 1;
  else                                         Q <= Q;

always @ (Q or ENT_L or UPDN) // Create RCO_L combinational output
  if       (!ENT_L &&  UPDN && (Q == 4'd15))  RCO_L = 0;
  else if (!ENT_L && !UPDN && (Q == 4'd0 ))   RCO_L = 0;
  else                                         RCO_L = 1;
```



References

John F. Wakerly, *Digital Design: Principles and Practices (4th Edition)*, Prentice Hall, 2005.