Registered Logic Design

INTRODUCTION

In the previous section we discussed combinatorial designs, circuits whose outputs are totally independent of any system clock. In this section we will discuss sequential circuits, where outputs store their previous values until a new clock is applied. The storage elements which retain the previous output values are called flip-flops. A bank of these flip-flops forms a register, although individual flip-flops are often called registers.

Before we discuss purely registered designs, let us take a look at designs which combine both registered and combinatorial portions. Registered and combinatorial outputs are often mixed on a single device. There can be two distinct designs, one registered and one combinatorial (often glue logic) combined on a single device for higher integration. There may also be a design requirement where registered outputs need to be decoded using combinatorial logic.

There are a number of devices which provide both registered and combinatorial outputs. Most devices provide programmable register bypass, which allows outputs to be programmed as registered or combinatorial.

In most design software packages, the output registers are signified by the ":=" assignment symbol, as opposed to the "=" sign for a combinatorial output. This helps to easily identify registers in each equation. In devices which provide outputs configurable as either registered or combinatorial, this sign is also used by the software to configure the outputs.

General Device Selection Considerations

The same set of general device selection considerations discussed in the PLD design methodology section apply to registered designs. The list of items which must be considered is repeated in Figure 1 for convenience. A device can be conveniently selected based upon the specific input and output requirements.

- Number of input pins
- Number of output pins
- Number of I/O pins
- Device speed
- Device power requirements
- Number of registers
- Number of product terms
- Output polarity control

Figure 1. General Device Selection Considerations

Maximum Frequency

For registered designs, speed is a parameter which needs careful consideration. Most combinatorial designs use the propagation delay (t_{PD}) for ensuring that enough time is allowed for the data from the inputs to appear at the outputs. In registered designs the effects of the clock must be taken into account. This is reflected in the maximum frequency (f_{MAX}) parameter. The flexibility inherent in PLD design provides a choice of configurations from which different f_{MAX} parameters can be calculated.

In the first type of design, the PLD is used for a standalone registered design. In order to decide the next logic level of the registers, the present logic level needs to be available at the inputs of the registers before they are clocked (Figure 2.) Under these conditions the clock period is limited by the internal delay from the flip-flop outputs through the internal feedback and logic to the flip-flops inputs. This f_{MAX} is designated "f_{MAX} *internal.*" A simple internal counter is a good example of this type of design, therefore, this parameter is sometimes called "f_{CNT}."

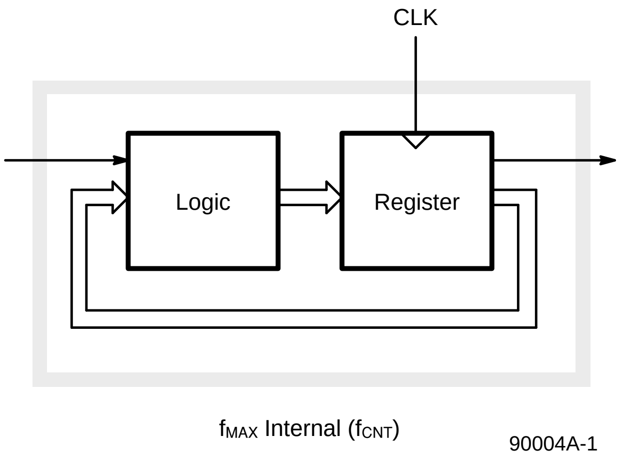
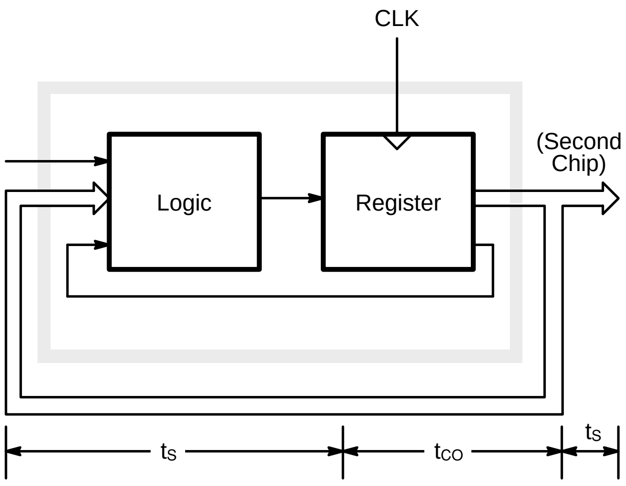


Figure 2. Internal f_{MAX}

Publication# **90004** Rev. **A** Amendment **/0**
Issue Date: **February 1996**

The second type of system configuration is when a number of logic devices with registers, including PLDs, are clocked with a common clock. This is probably the most prevalent configuration. In this case, the registered outputs are sent off-chip back to the device inputs or to the inputs of a second device. The slowest path defining the period (Figure 3) is the sum of the clock-to-output time and the input setup time for the external signals (t_S+t_{CO}). The reciprocal, f_{MAX}, is the maximum frequency with external feedback or in conjunction with an equivalent speed device. This f_{MAX} is designated "f_{MAX} *external.*"

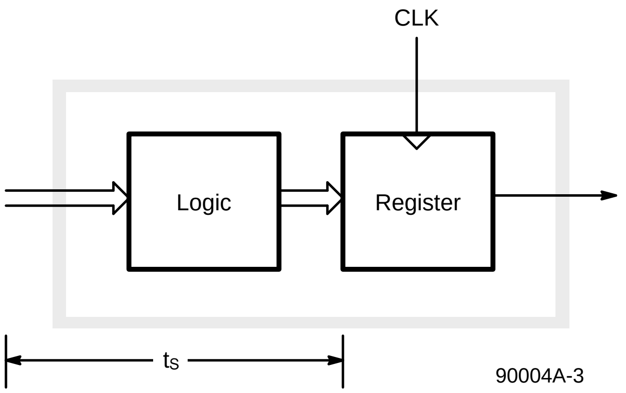


f_{MAX} External: 1/(t_S + t_{CO})

90004A-2

Figure 3. External f_{MAX}

The third type of design is a simple data path application. In this case, input data is presented, to the flip-flop and clocked; no feedback is employed (Figure 4). In this case, the period is limited by the sum of data setup time and data hold time (t_S+t_H). However, the minimum clock period (t_{WH} + t_{WL}) is usually a stricter limit. Thus, the third f_{MAX} designated "f_{MAX} *no feedback"* will be the lesser of 1/ (t_S + t_H) or 1/ (t_{WH} + t_{WL}).



90004A-3

f_{MAX} No Feedback: 1/(t_S + t_H) or 1/(t_{WH} + t_{WL})

Figure 4. f_{MAX} with No Feedback

f_{MAX} external and f_{MAX} no feedback are calculated parameters. f_{MAX} internal is measured.

Flip-Flop Types

There are four basic types of flip-flops; S-R, J-K, T and the popular D-type. These flip-flops are described in the "PLD Design Basics" section of this data book.

Almost all registered PLDs provide the basic D-type flip-flops. D-type flip-flops are the simplest to design with and will be used throughout this section. Some PLDs provide the capability of configuring output registers as either D, T, J-K or S-R. Configurable flip-flops in some cases can reduce the number of product terms required for certain designs. The effect of the configurable flip-flops will be discussed wherever relevant.

Synchronous vs. Asynchronous

Registered designs can be easily classified into two categories; synchronous and asynchronous. In synchronous designs the clock inputs of all the registers are tied together to a common clock. With asynchronous designs, the flip-flops' clock inputs may not be tied together, and the clocks may be gated or even driven by other flip-flops. We will first discuss synchronous registered designs and then asynchronous registered designs.

Synchronous Registered Designs

Synchronous registered designs are used for two major functions: data handling and control. Registered synchronous designs for data handling include counters and shift registers. There are various types of counters. Some are; binary counters, modulo counters, Johnson counters, and Gray-code counters. These counters are differentiated by the sequence of values through which the counter travels. A binary counter is the simplest form of a counter, and is used most often for data functions. Any system requiring a regular count uses a binary counter. Modulo, Gray-code, and Johnson counters are also used for control.

All counters are actually subsets of a larger class of digital designs called state machines. State machines are discussed in detail in the next chapter of this handbook.

Counters

Counters are the most commonly used sequential circuits. A set of registers, that cycles through a predetermined, unvarying sequence, is called a counter. A general model of a synchronous counter is illustrated in Figure 5. This shows a common clock to all the flip-flops, whose outputs are fed back to a combinatorial logic array called the next-state (count) decoder. The next count is generated by this logic based upon the present count and control inputs. Most PLDs use the standard sum-of-products form of array for this logic.

The relationship between a four-bit counter and its signal timing diagram is illustrated in Figure 6. The counters can also be represented by state diagrams (Figure 7). The state diagrams are bubble-and-arrow diagrams. Each bubble represents a count value and each arrow a transition from one count to the next. More detail on state diagrams is given in the next chapter on state machine design. For counters, the state diagrams are a convenient representation tool and will be used in the discussion when necessary.

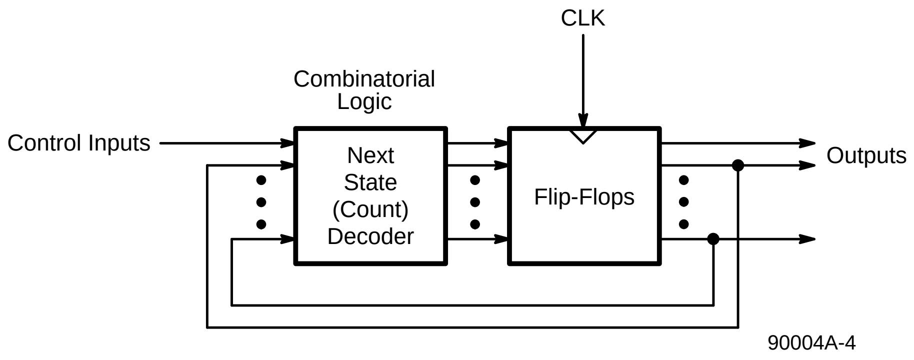


Figure 5. General Model of a Counter

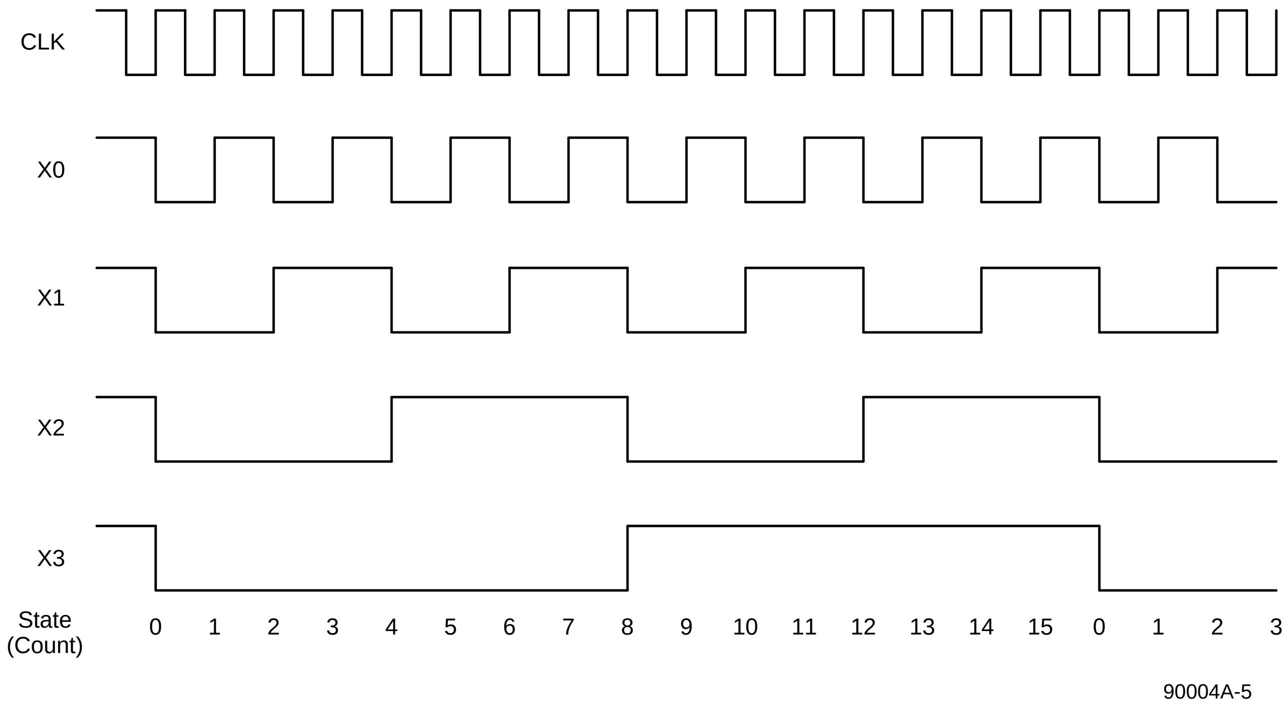


Figure 6. Timing Diagram of a Four-Bit Binary Counter

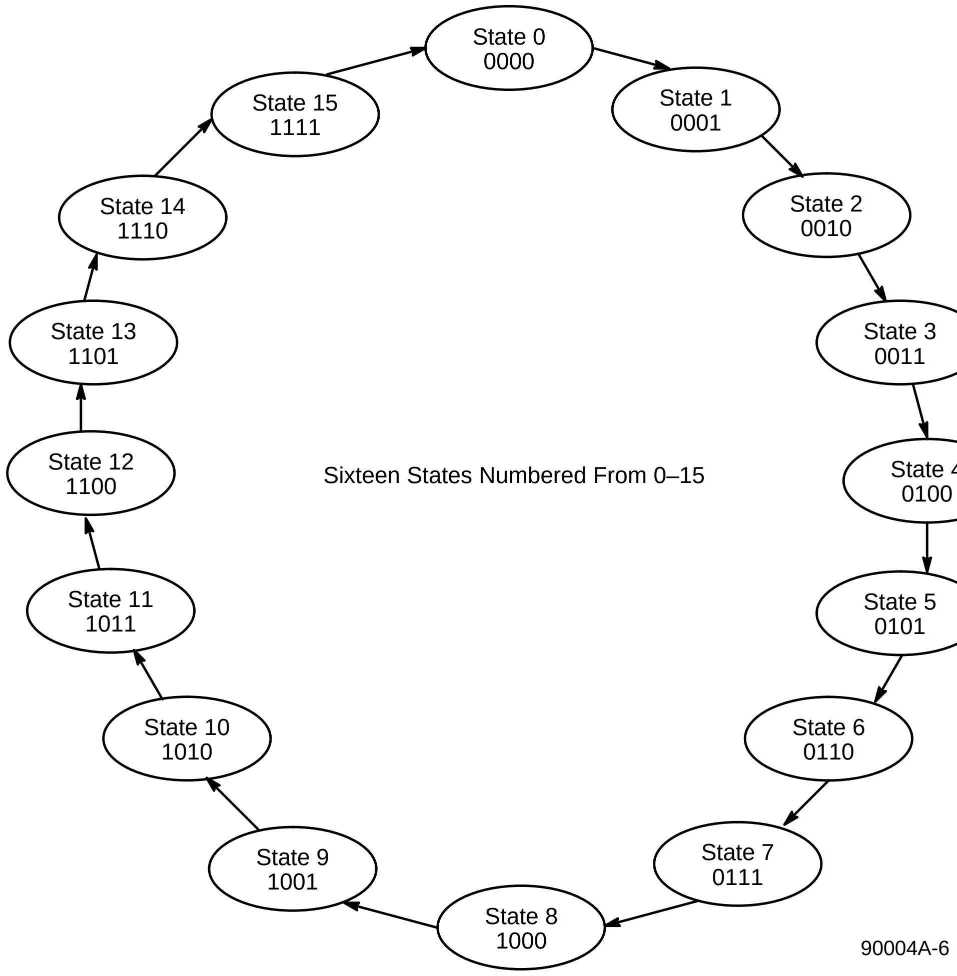


Figure 7. State Diagram of a Four-Bit Binary Counter

Binary Counters

Let us examine a four-bit binary counter. The truth table (also called the transition table) for such a counter is given in Table 1. The table lists the next state values of all the output registers based upon their present values.

Table 1. The Truth Table for a Four-Bit Binary Counter

Present State				Next State			
X3	**X2**	**X1**	**X0**	**X3**	**X2**	**X1**	**X0**
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	1	0	1	0
1	0	1	0	1	0	1	1
1	0	1	1	1	1	0	0
1	1	0	0	1	1	0	1
1	1	0	1	1	1	1	0
1	1	1	0	1	1	1	1
1	1	1	1	0	0	0	0

We derive Boolean equations for each bit directly from the above truth table by collecting all the product terms where outputs are asserted HIGH (ones). This yields:

```
X3  :=     /X3  *   X2  *   X1   *   X0
       +    X3  *  /X2  *  /X1   *  /X0
       +    X3  *  /X2  *  /X1   *   X0
       +    X3  *  /X2  *   X1   *  /X0
       +    X3  *  /X2  *   X1   *   X0
       +    X3  *   X2  *  /X1   *  /X0
       +    X3  *   X2  *  /X1   *   X0
       +    X3  *   X2  *   X1   *  /X0
X2  :=     /X3  *  /X2  *   X1   *   X0
       +   /X3  *   X2  *  /X1   *  /X0
       +   /X3  *   X2  *  /X1   *   X0
       +   /X3  *   X2  *   X1   *  /X0
       +    X3  *  /X2  *   X1   *   X0
       +    X3  *   X2  *  /X1   *  /X0
       +    X3  *   X2  *  /X1   *   X0
       +    X3  *   X2  *   X1   *  /X0
X1  :=     /X3  *  /X2  *  /X1   *   X0
       +   /X3  *  /X2  *   X1   *  /X0
       +   /X3  *   X2  *  /X1   *   X0
       +   /X3  *   X2  *   X1   *  /X0
       +    X3  *  /X2  *  /X1   *   X0
       +    X3  *  /X2  *   X1   *  /X0
       +    X3  *   X2  *  /X1   *   X0
       +    X3  *   X2  *   X1   *  /X0
X0  :=     /X3  *  /X2  *  /X1   *  /X0
       +   /X3  *  /X2  *   X1   *  /X0
       +   /X3  *  /X2  *  /X1   *  /X0
       +   /X3  *   X2  *   X1   *  /X0
       +    X3  *  /X2  *  /X1   *  /X0
       +    X3  *  /X2  *   X1   *  /X0
       +    X3  *   X2  *  /X1   *  /X0
       +    X3  *   X2  *   X1   *  /X0
```

These Boolean equations are for devices with active-HIGH outputs. These equations can be inverted for devices with active LOW outputs. The Boolean equations for active-LOW devices can also be directly derived from the truth table by collecting all the product terms where the active-LOW outputs (zeros) are asserted.

Manipulating the equations with Boolean algebra, we obtain the Boolean logic equations:

```
X0  := /X0
X1  := X1  :+: X0
X2  := X2  :+: (X1   * X0)
X3  := X3  :+: (X2   * X1   * X0)
```

Similarly, for active-LOW output devices (since /(A :+: B) = /A :+: B):

```
/X0 := X0
/X1 := /X1 :+: X0
/X2 := /X2 :+: (X1   * X0)
/X3 := /X3 :+: (X2   * X1   * X0)
```

These equations could also be obtained from the Boolean equations developed for an adder in the combinatorial design section.

Rewriting the equations for an adder:

```
X0  =  A0  :+: B0 :+: Cin
X1  =  A1  :+: B1 :+: C0
```

where

```
C0  =  A0  *  B0   +  (A0 + B0)  *  Cin
```

```
X2  =  A2  :+: B2 :+: C1
```

where

```
C1  =  A1  *  B1   +  (A1 + B1) * (A0 * B0)
    +  (A1 + B1)  *  (A0 + B0)  *  Cin
```

```
X3  =  A3  :+: B3 :+: C2
```

where

```
C2  =  A2  *  B2   + (A2 + B2) *(A1 * B1)
    +  (A2 + B2)  *  (A1 + B1)  *(A0 * B0)
    +  (A2 + B2)  *  (A1 + B1)  *(A0 * B0)
       *  Cin
```

Assuming one of the operands in the adder is the number itself and the second operand is one (X3–X0 = A3–A0, B3–B0 = 0001 and Cin = 0) we get the following equations for a counter:

```
X0  := /X0
X1  := X1  :+: X0
X2  := X2  :+: (X1   * X0)
X3  := X3  :+: (X2   * X1   * X0)
```

These are, of course, the same equations as the ones derived directly from the truth table. The equations for a binary counter are very regular. The general equation for an n-bit binary counter can be directly expressed:

```
Xn := Xn :+: (Xn–1* Xn–2 ... X0)
```

For devices with active-LOW outputs, the general Boolean equations can be derived by inverting both sides of the equation:

```
/Xn := /Xn :+: (Xn–1* Xn–2 ... X0)
```

These equations represent a binary UP counter. Counting backwards for a DOWN counter, the Boolean equations can be similarly generated, either from the truth table or from the adder Boolean equations. The general equation for a DOWN counter is:

```
Xn := Xn :+: (/Xn–1* /Xn–2 ... /X0)
```

This equation is for active-HIGH outputs. For active-LOW output devices the Boolean equation for a DOWN counter is:

```
/Xn := /Xn :+: (/Xn–1* /Xn–2 ... /X0)
```

Further control functions can be added to these counter equations directly either at the truth-table stage or in the equations. For example, a load data function is required in most counters. This allows registers to be loaded with a count under the control of another input signal (LOAD). When the LOAD signal is HIGH the counter is loaded with the input data, and when the LOAD signal is LOW the counting is resumed.

Binary Counter Device Selection Considerations

One major device selection consideration is the logic requirement.

The binary counter Boolean equations make use of exclusive-OR functions in the output. In most of the registered PLDs, the XOR functions are implemented in their sum-of-products logic form. This usually requires a large number of product terms. Most standard PAL devices provide eight product terms per output. However, for larger counters, a greater number of product terms is required.

Some PLDs provide a dedicated XOR gate on the outputs. This allows an AND-OR-XOR implementation of the Boolean logic, and consequently requires fewer product terms.

Cascading Binary Counters

Situations are occasionally encountered in digital system designs where very long counters are required. Binary counters can be easily cascaded into two or more devices to construct such large counters. The design of long counters is very simple. These are designed as simple binary counters with a count enable control. The less significant counters generate an extra output signal at the penultimate count. These signals are ANDed together to form the count enable signal for the higher-order counter. For a down counter the reverse scheme is implemented.

Cascading counters is a lot easier than cascading adders because the carry-look-ahead circuitry is not required. The only thing to remember is that the more significant counter toggles only when the penultimate count of all of the less significant counters is reached.

Flip-Flop Selection

Until now, all the designs have been implemented in devices with D-type flip-flops. What happens if the counter design is implemented in a device that allows both J-K and T-type registers? The Boolean logic equations for such a design can be derived from the truth table. This requires advanced knowledge of the functionality of the J-K and T-type registers. For the J-K register the output is asserted when the J input goes HIGH and the output is unasserted when the K input goes HIGH. Toggle type registers require the T input to be asserted for every change in the output level.

Table 2. Truth Table for D, J-K and T-Type Flip-Flops

Present State				Next State															
				X3				X2				X1				X0			
X3	X2	X1	X0	D	J	K	T	D	J	K	T	D	J	K	T	D	J	K	T
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	1	0	0	1	1
0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	1
0	0	1	1	0	0	0	0	1	1	0	1	0	0	1	1	0	0	1	1
0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	1
0	1	0	1	0	0	0	0	1	0	0	0	1	1	0	1	0	0	1	1
0	1	1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	1	0	1
0	1	1	1	1	1	0	1	0	0	1	1	0	0	1	1	0	0	1	1
1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1
1	0	0	1	1	0	0	0	0	0	0	0	1	1	0	1	0	0	1	1
1	0	1	0	1	0	0	0	0	0	0	0	1	0	0	0	1	1	0	1
1	0	1	1	1	0	0	0	1	1	0	1	0	0	1	1	0	0	1	1
1	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	1	0	1
1	1	0	1	1	0	0	0	1	0	0	0	1	1	0	1	0	0	1	1
1	1	1	0	1	0	0	0	1	0	0	0	1	0	0	0	1	1	0	1
1	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Table 2 shows the truth table for both a J-K and a T-type register implementation for a binary counter. Deriving and optimizing the equations from the table, we get the following results:

```
X3–J   := /X3  *   X2  *   X1  *   X0
X3–K   :=   X3  *   X2  *   X1  *   X0
X2–J   := /X2  *   X1 *X0
X2–K   :=   X2  *   X1 X0
X1–J   := /X1  *   X0
X1–K   :=   X1  *   X0
X0–J   := /X0
X0–K   :=   X0
X3–T   :=   X2  *   X1  *   X0
X2–T   :=   X1  *   X0
X1–T   :=   X0
X0–T   :=     1
```

As we can see from these equations, the number of product terms used for J-K and T-type implementations are smaller than the number of product terms required for a D-type implementation.

Which flip-flop is most efficient depends on the relative number of transitions or holds required. As a counter traverses from one count (state) to another, every output either makes a “transition” (changes logic level) or “holds” (stays at the same logic level). Small counters in general require more transitions and fewer holds. As the designs get larger, the higher-order bits require fewer transitions and more holds.

D-type flip-flops use up product terms only for active transitions from logic LOW level to HIGH level, and for logic HIGH level holds only. J-K and T-type flip-flops use up product terms for both LOW-to-HIGH and HIGH-to-LOW transitions, but eliminate hold terms. Generally, the requirements of transition and hold terms depends upon the count sequence selection. D-type flip-flops are more efficient for small designs. Conversely J-K and T-type flip-flops can be more efficient for large designs, which require more hold terms.

A comparison of product term requirements of 2-, 3-, 4- and 5-bit binary counters can be representative for other types of counters and state machines. Table 3 shows the transition terms and the hold terms required for these counters. For a J-K type flip-flop implementation, after optimizing, total product terms required are 4, 6, 8, and 10 respectively. The D-type implementation requires 3, 6, 10, and 15 respectively, and is relatively less efficient for large counters.

Table 3. Product Term Requirements for Configurable Flip-Flops

Binary Counter	Transitions	Holds	D Product Terms	J-K Product Terms	T Product Terms
2-Bit	6	2	3	4	1
3-Bit	14	10	6	6	1
4-Bit	30	34	10	8	1
5-Bit	62	98	15	10	1

Modulo Counters

The number of unique states a counter traverses is generally referred to as the modulus. A typical n-bit binary counter has a maximum modulus of 2n. It is often necessary to introduce signal delays into the logic design to meet timing requirements. This makes it possible to allow for bus-skew, access time, or differential propagation delays between devices along two different signal paths. A typical example of this is the introduction of wait states to allow for access times of different memory elements. Counters and delay lines are commonly used to introduce the delay. Counters in PLDs have the added advantage of programmability to select the required delay. Such applications where precise timing duration control is required usually use modulo counters with a non-power-of-two modulus. Other applications of modulo counters include waveform generators and arbiters.

Table 4. Truth Table for a BCD Counter

Present State					Next State			
Q3	Q2	Q1	Q0		Q3	Q2	Q1	Q0
0	0	0	0	0 -> 1	0	0	0	1
0	0	0	1	1 -> 2	0	0	1	0
0	0	1	0	2 -> 3	0	0	1	1
0	0	1	1	3 -> 4	0	1	0	0
0	1	0	0	4 -> 5	0	1	0	1
0	1	0	1	5 -> 6	0	1	1	0
0	1	1	0	6 -> 7	0	1	1	1
0	1	1	1	7 -> 8	1	0	0	0
1	0	0	0	8 -> 9	1	0	0	1
1	0	0	1	9 -> 0	0	0	0	0

A good example of a modulo counter is a BCD counter. Such a counter is useful in applications where the computer's outputs are generated using a decimal system. While a four-bit binary counter can count to sixteen, the BCD counter terminates the count at the modulus of 10.

Modulo counters can be designed in a variety of ways. One direct way is to use the truth table to implement a count to a modulus and directly derive the equations from it. The truth table for a BCD count (from zero to nine) is shown in Table 4.

Now let us consider what happens if the device accidentally powers up in one of the count values from ten to fifteen. These are illegal counts (states) and, for a good design, a mechanism must be built into the equations to allow it to recover back into a legal state. What we actually need is to consider the truth table in Table 5 in conjunction with the one in Table 4 for deriving the Boolean equations.

Table 5. Truth Table for Illegal State Recovery to Count Zero

Present State					Next State			
Q3	Q2	Q1	Q0		Q3	Q2	Q1	Q0
1	0	1	0	10 -> 0	0	0	0	0
1	0	1	1	11 -> 0	0	0	0	0
1	1	0	0	12 -> 0	0	0	0	0
1	1	0	1	13 -> 0	0	0	0	0
1	1	1	0	14 -> 0	0	0	0	0
1	1	1	1	15 -> 0	0	0	0	0

A state diagram for the BCD counter is shown in Figure 8. For active-LOW outputs, the Boolean equations can be derived directly from the truth table and optimized using Karnaugh maps or the software minimizer.

The Boolean equation for Q3 is:

```
/Q3 :=     /Q3  * /Q2  * /Q1  * /Q0
       +   /Q3  * /Q2  * /Q1  *  Q0
       +   /Q3  * /Q2  *  Q1  * /Q0
       +   /Q3  * /Q2  *  Q1  *  Q0
       +   /Q3  *  Q2  * /Q1  * /Q0
       +   /Q3  *  Q2  * /Q1  *  Q0
       +   /Q3  *  Q2  *  Q1  * /Q0
       +    Q3  * /Q2  * /Q1  *  Q0
       +    Q3  * /Q2  *  Q1  * /Q0
       +    Q3  * /Q2  *  Q1  *  Q0
       +    Q3  *  Q2  * /Q1  * /Q0
       +    Q3  *  Q2  * /Q1  *  Q0
       +    Q3  *  Q2  *  Q1  * /Q0
       +    Q3  *  Q2  *  Q1  *  Q0
```

The equation can be reduced to the following:

```
/Q3 :=     /Q3  * /Q2
       +   /Q3  * /Q1
       +   /Q2  *  Q0
       +    Q3  *  Q1
       +    Q3  *  Q2
```

Similar Boolean equations can be generated for Q2, Q12 and Q0.

Figure 9 shows the circuit diagram of a loadable dual BCD counter.

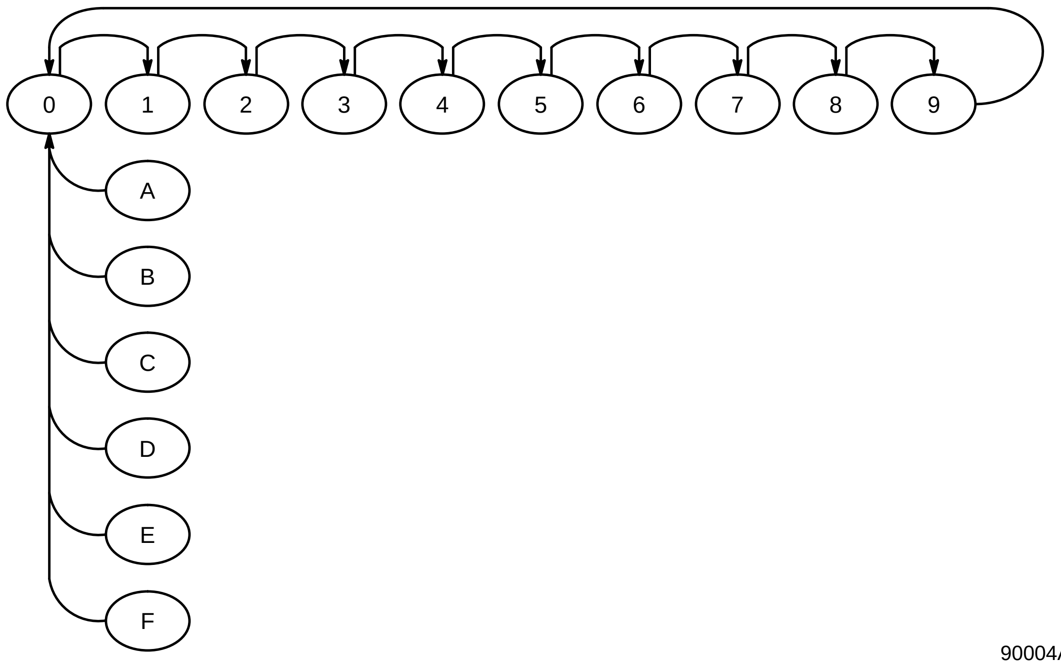


Figure 8. State Sequence of a BCD Counter Showing Illegal State Recovery

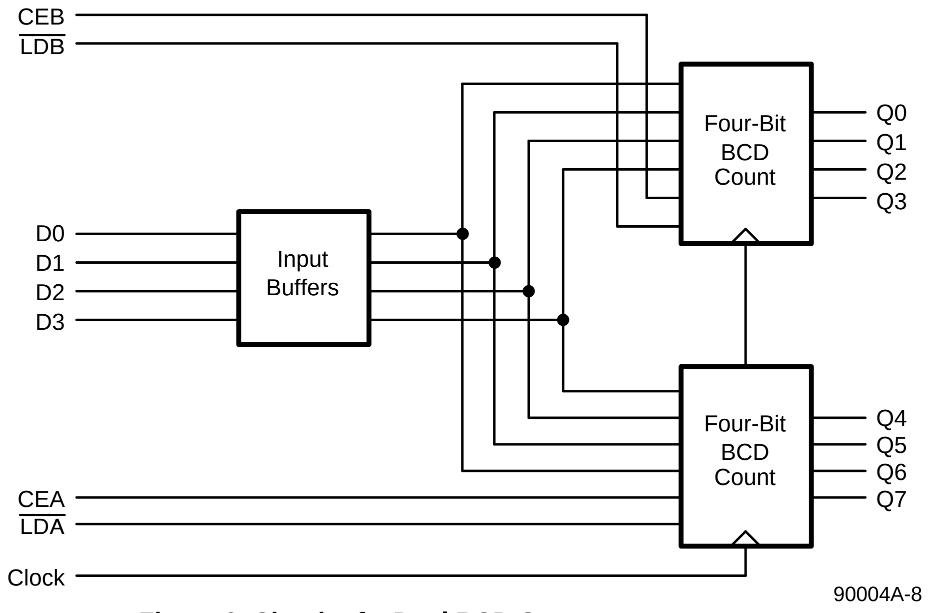


Figure 9. Circuit of a Dual BCD Counter

Modulo Counter Device Selection Considerations

We have illustrated a counter that counts from zero to a fixed modulus. The same technique can be applied for a counter which counts down from a maximum power-of-two number to a fixed modulus, or even a counter which counts from one modulus to another. The important considerations will be the number of product terms used.

The registered PLDs used for modulo counters are similar to the ones selected for other counters. Since the counts used are binary, devices with J-K, T-type flip-flops, or XOR gates will help optimize the number of product terms used. The product term usage also depends upon the modulus selected. Generally, a power-of two or a multiple-of-two modulus will require fewer product terms.

Another factor for flip-flop selection is the illegal states. D-type flip-flops are generally better suited for illegal state recovery than the J-K or T-type flip-flops. This is because when no product term is asserted, the D-type flip-flops reset to zero. Designers using J-K or T-type flip-flops must design-in illegal state recovery.

Certain devices allow the use of a synchronous RESET product term for modulo counters. The idea is to use the minimal number of product terms to build a binary counter that counts up to a power-of-two number. However, this counter is RESET to zero using the synchronous RESET product term when the desired modulus is reached. It then begins counting afresh from zero, and the procedure is repeated. Similar operation can also be achieved with a synchronous PRESET product term for a down counter.

Using synchronous RESET and PRESET product terms allows the counter to recover from illegal states. Notice that the logic product terms in the counter are designed for a complete binary count. If the counter powers up in any illegal state (as shown in Figure 10), it will continue the count until the terminal count and then, return to zero, where the correct modulo count will begin. This illegal state recovery will take an unpredictable number of clock cycles, and you may wish to design a more systematic recovery system.

Cascading Modulo Counters

For large modulo counters, the technique of generating Boolean equations from the truth tables is very tedious and time consuming. Another approach for designing modulo counters is to divide it into two smaller modulo counters. In addition to simplifying the design, this approach usually helps optimize the number of product terms.

As an example, a modulo-360 counter can be directly implemented with nine register bits. However, instead of implementing this as a straight 9-bit counter, we can implement this as two counters: one four-bit counter (counting from zero to 14) and another five-bit counter (counting from zero to 23). Together, the two counters count up to 360. The terminal count output, MOUT, is asserted when the count reaches 360, as shown in Figure 11.

The design requires nine inputs, nine outputs, one clock pin, one LOAD pin, one RESET and one MOUT (module output signal) pin. Note that no extra flip-flops or pins were needed. Obviously, the count values of this counter are not the same as a straight modulo-360 counter. Actually, this is what contributes to the optimization of the number of product terms used.

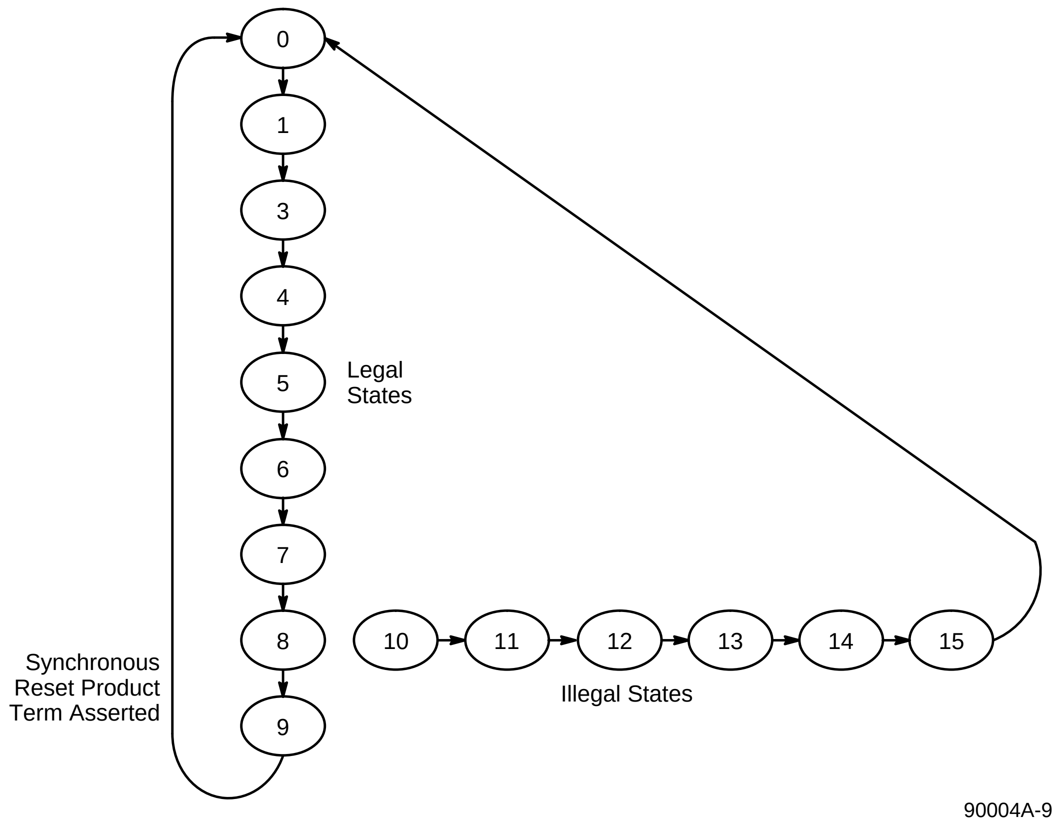


Figure 10. A BCD Counter Using Synchronous RESET Product Term

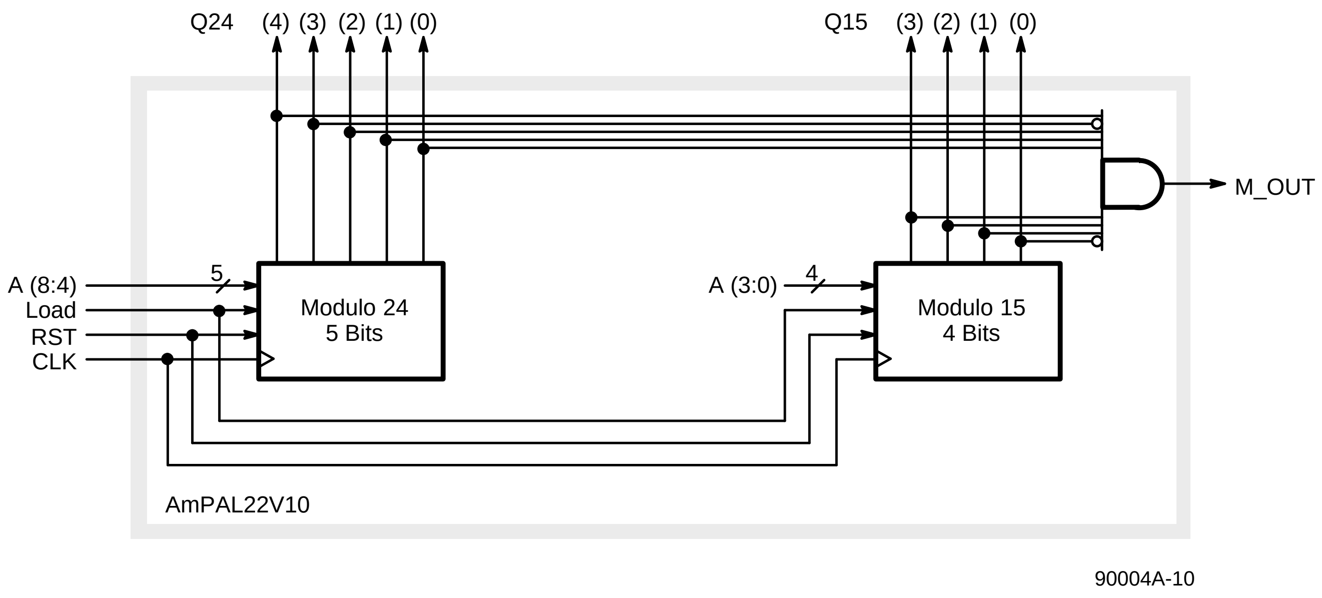


Figure 11. A Modulo-360 Counter

Counters with Encoding

Until now, we have discussed counters that generate binary output sequences. Most peripherals require a predetermined sequence of control signals. Custom control sequences can be generated by decoding the binary sequence with combinatorial logic. Figure 12 shows a general model of a counter with combinatorial output decoding circuitry. This combinatorial circuit modifies the counter bits and generates output signals in the manner required for peripheral timing and control. Since these circuits require extra combinatorial logic, they are not very efficient. They are also more susceptible to hazards and output glitches.

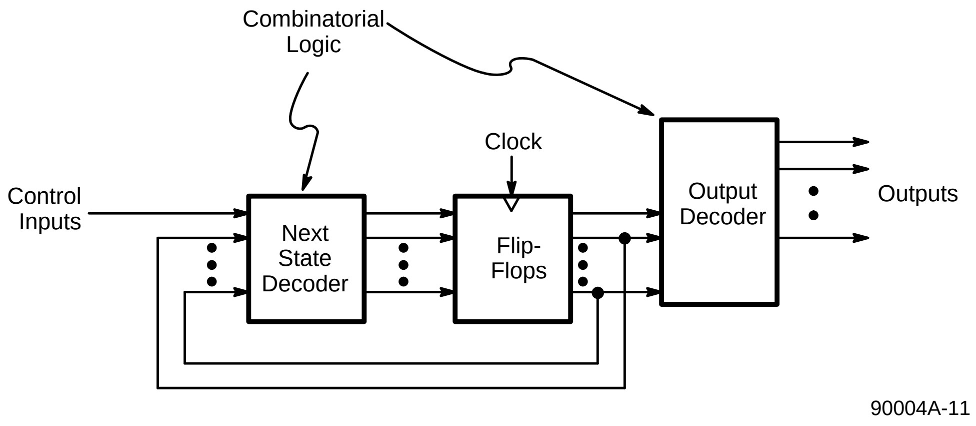


Figure 12. Counter with an Output Decoder

It is possible to have a different output coding for a four-bit counter, as shown in Table 6. This code, called Gray code, allows only one output bit to toggle for each new count value. This code can be easily derived from a four-bit binary counter code (also shown in Table 6) using an output decoder.

Table 6. Generating Gray Code from a Binary Code

Binary Code				Gray Code			
X3	X2	X1	X0	G3	G2	G1	G0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

We can derive the Boolean equations for the combinatorial output decoder from the truth table. The equations are:

```
G3 = X3
G2 = X3 :+: X2
G1 = X2 :+: X1
G0 = X1 :+: X0
```

A more efficient and easier technique for generating control signals is to implement the decode circuitry before the registers. This alternative is shown in Figure 13. This essentially generates a non-standard counter with state values that are not a binary progression. It can be considered as a counter where the product terms for a binary count and encoding the outputs have been combined.

Many different codes can be generated using such techniques. We will limit ourselves to the ones that are most commonly used: Gray-code counters and Johnson counters.

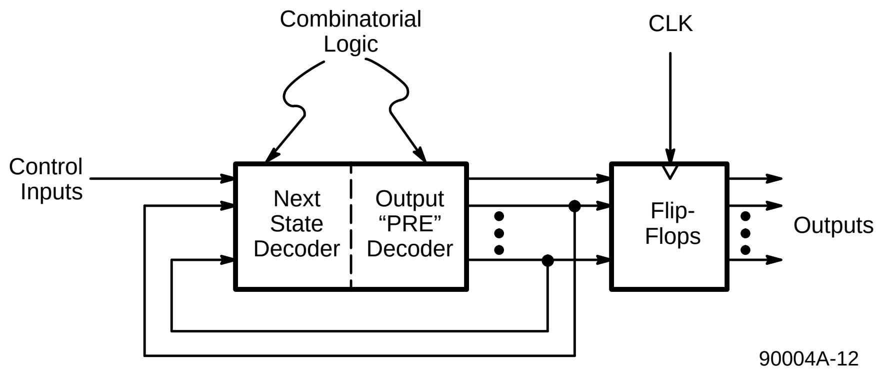


Figure 13. Counter with Combined Next State Generation and Output Encoding Circuit

Gray-Code Counters

Gray-code counters are often used in digital designs for control timing functions. The primary advantage of Gray-code counters stems from the characteristic that only one output bit changes value for every clock cycle. These output signals can be easily decoded using a combinatorial decoder without any risk of hazards. Gray-code counters are used extensively as system clocks, since the different output bits provide different clock pulses, without the risks of hazards. Gray-code is also used in high-speed data communication applications, where data is transmitted from one part of the system to another, and where the error susceptibility increases with the number of bit changes between adjacent numbers in a sequence. These are also used for such specialized applications as shaft encoders and real-time process control.

The implementation of a Gray-code counter is very simple. A truth table can be derived from the transition table as is done for a binary counter. The Boolean equations can then be directly derived from the truth table. The truth table for the Gray-code counter is shown in Table 7.

Table 7. Truth Table for a Four-Bit Gray-Code Counter

Present State				Next State			
X3	X2	X1	X0	X3	X2	X1	X0
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	1
0	0	1	1	0	0	1	0
0	0	1	0	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	0	1	0	1
0	1	0	1	0	1	0	0
0	1	0	0	1	1	0	0
1	1	0	0	1	1	0	1
1	1	0	1	1	1	1	1
1	1	1	1	1	1	1	0
1	1	1	0	1	0	1	0
1	0	1	0	1	0	1	1
1	0	1	1	1	0	0	1
1	0	0	1	1	0	0	0
1	0	0	0	0	0	0	0

The Boolean logic equations for a Gray-code counter are:

```
X3  :=     /X3  *  X2  * /X1  * /X0
       +    X3  *  X2  * /X1  * /X0
       +    X3  *  X2  * /X1  *  X0
       +    X3  *  X2  *  X1  *  X0
       +    X3  *  X2  *  X1  * /X0
       +    X3  * /X2  *  X1  * /X0
       +    X3  * /X2  *  X1  *  X0
       +    X3  * /X2  * /X1  *  X0
X2  :=     /X3  * /X2  *  X1  * /X0
       +   /X3  *  X2  *  X1  * /X0
       +   /X3  *  X2  *  X1  *  X0
       +   /X3  *  X2  * /X1  *  X0
       +   /X3  *  X2  * /X1  * /X0
       +    X3  *  X2  * /X1  * /X0
       +    X3  *  X2  * /X1  *  X0
       +    X3  *  X2  *  X1  *  X0
X1  :=     /X3  * /X2  * /X1  *  X0
       +   /X3  * /X2  *  X1  *  X0
       +   /X3  * /X2  *  X1  * /X0
       +   /X3  *  X2  *  X1  * /X0
       +    X3  *  X2  * /X1  *  X0
       +    X3  *  X2  *  X1  *  X0
       +    X3  *  X2  *  X1  * /X0
       +    X3  * /X2  *  X1  * /X0
X0  :=     /X3  * /X2  * /X1  * /X0
       +   /X3  * /X2  * /X1  *  X0
       +   /X3  *  X2  *  X1  * /X0
       +   /X3  *  X2  *  X1  *  X0
       +    X3  *  X2  * /X1  * /X0
       +    X3  *  X2  * /X1  *  X0
       +    X3  * /X2  *  X1  * /X0
       +    X3  * /X2  *  X1  *  X0
```

Johnson Counters

A Johnson counter is part of a family of counters known as “ring counters.” These counters are used for special applications where code symmetry is desired. Ring counters are also often used for timing purposes, since all the outputs are essentially a series of pulses. This code symmetry also allows use of the fewest possible product terms with a D-type register. Devices that provide a small amount of logic per cell, can implement Johnson counters very easily.

Johnson counters are also known as circular-shift counters. The sequence for a five-stage Johnson counter is shown in Table 8. As can be seen in the truth table, the counter first fills up with 1’s from left to right and then it fills up with zeros again. Note from the output sequence that only one of the Johnson counter bits changes for every clock period, like the Gray-code counter. One major advantage of the Johnson counter is that it can be readily decoded with small two-input NAND gates and hence is suitable for high-speed applications.

Note that the five-stage sequence has a table of 10 legal states and 22 illegal states (Table 9). In general, an n-bit Johnson counter will produce a modulus of 2n. Figure 14 shows the state diagram of the five-bit counter.

Table 8. Five-Bit Johnson Counter Truth Table

Legal States

Present State					Next State				
Q4	Q3	Q2	Q1	Q0	Q4	Q3	Q2	Q1	Q0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	1	1	0	0	0	0	1
0	0	1	1	1	0	0	0	1	1
0	1	1	1	1	0	0	1	1	1
1	1	1	1	1	0	1	1	1	1
1	1	1	1	0	1	1	1	1	1
1	1	1	0	0	1	1	1	1	0
1	1	0	0	0	1	1	1	0	0
1	0	0	0	0	1	1	0	0	0

The implementation of a Johnson counter is relatively straight-forward, and is the same regardless of the number of stages. When D-type flip-flops are used, the Q output of each flip-flop is connected to the D input of the following stage. The single exception is the Q output of the last stage, which is complemented and connected to the D input of the first stage.

Table 9. Illegal States for a Five-Bit Johnson Counter

Illegal States

Present State					Next State				
Q4	Q3	Q2	Q1	Q0	Q4	Q3	Q2	Q1	Q0
0	0	0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	1	0	0	1	0	0	0	0	0
0	1	0	1	0	0	0	0	0	0
0	1	0	1	1	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0
0	1	1	0	1	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0
1	0	0	1	1	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0
1	0	1	0	1	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0
1	0	1	1	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0
1	1	1	0	1	0	0	0	0	0

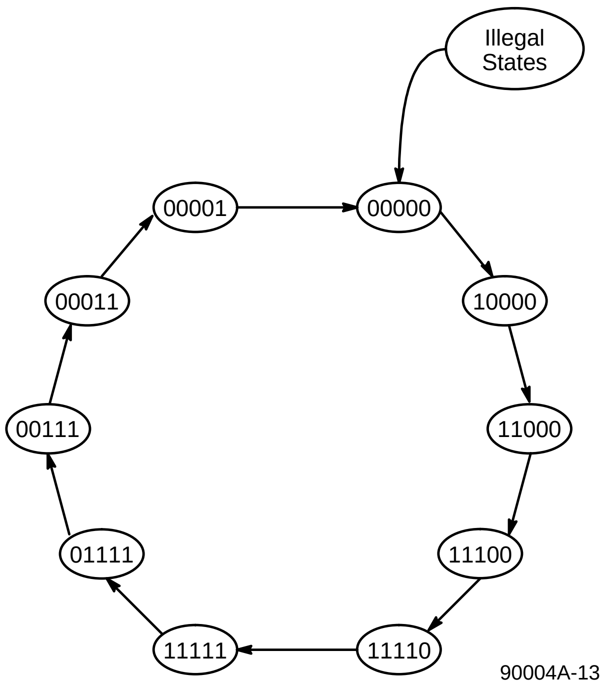


Figure 14. State Diagram of a Five-Bit Johnson Counter

One disadvantage of the counter is the number of invalid (or illegal) states. The invalid states increase exponentially with the length of the counter. The bigger the counter becomes, the greater are its chances of entering an illegal state. Johnson counters are very susceptible to illegal states, and can “hang up” very easily. Noise or improper use can cause this counter to end up in an illegal state. Therefore, a design with illegal state recovery circuitry is always recommended.

Figure 15 shows a nine-bit Johnson counter that can be derived by directly extending the design of a five-bit Johnson counter.

Shift Registers

A Shift Register is a special digital circuit often used as a primary building block in digital computer systems. It is closely related to a ring counter. Its fundamental usage is for temporary data storage and bit-wise data manipulation for advanced arithmetic and multiplication operations. Shift registers are also frequently used in communications, for converting parallel byte-wide data from the microprocessor to a serial data bit-stream for transmission. Shift registers are also used in graphics systems for serializing parallel data for use by the display monitor. A number of examples of video shift registers are included in the graphics section.

The fundamental purpose of a shift register (Figure 16) is to shift data from one flip-flop to another. There are several types of shift registers. They are classified by the way in which incoming data is received (parallel or serial), and how outgoing data is transmitted (parallel or serial).

In the following example, we will discuss a simple universal shifter that provides both serial and parallel input and output functions. Depending upon the control signals I0 and I1, the data is shifted from one flip-flop to another in the left or the right direction. These inputs also control when the new parallel data is loaded onto the registers. When shifting left or right, serial data can be received and transmitted on serial pins LIRO and RILO. Since the flip-flop outputs appear on the output pins at all times, the parallel output data is always available. The truth table is shown in Table 10.

The Boolean logic equations can be directly derived from the truth table, and are shown Figure 17.

Shift registers can be modified to suit various system design requirements. This universal shift register can be used for serial in/serial out, parallel in/parallel out, serial in/parallel out and parallel in/serial out functions.

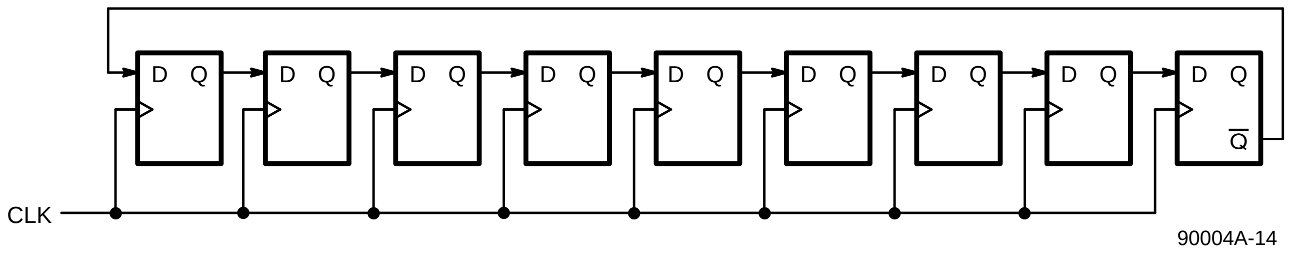


Figure 15. Block Diagram of a Nine-Bit Johnson Counter

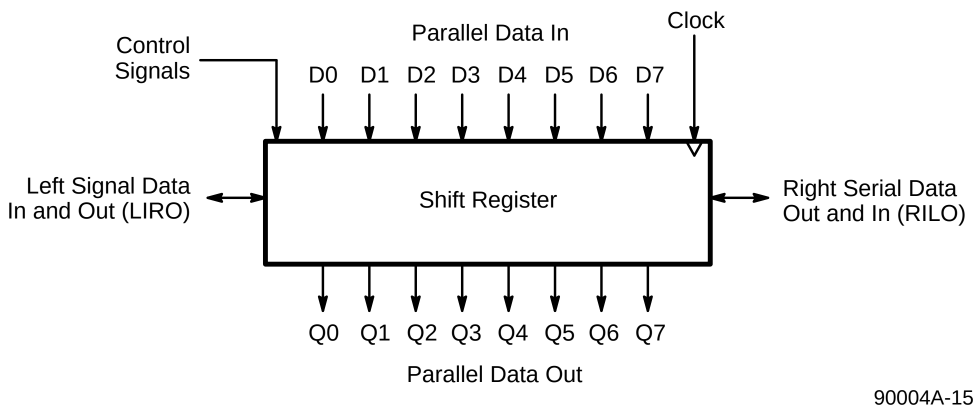


Figure 16. A Shift Register Block Diagram

Table 10. The Truth Table for a Universal Shift Register

Q7	**Q6**	**Q5**	**Q4**	**Q3**	**Q2**	**Q1**	**Q0**	**I1**	**I0**	
Q7	Q6	Q5	Q4	Q3	Q2	Q1	Q0	0	0	;Retain Data
RILO	Q7	Q6	Q5	Q4	Q3	Q2	Q1	0	1	;Shift Right
Q6	Q5	Q4	Q3	Q2	Q1	Q0	LIRO	1	0	;Shift Left
D7	D6	D5	D4	D3	D2	D1	D0	1	1	;Load Data

```
Equations
/Q0  :=     /I1*/I0*/Q0             ;HOLD Q0
      +     /I1*I0*Q1               ;SHIFT RIGHT
     :+:     I1*/I0*/LIRO           ;SHIFT LEFT
      +      I1*I0*/D0              ;LOAD D0
/Q1  :=     /I1*/I0*/Q1             ;HOLD Q1
      +     /I1*I0*/Q2              ;SHIFT RIGHT
     :+:     I1*/I0*/Q0             ;SHIFT LEFT
      +      I1*I0*/D1              ;LOAD D1
/Q2  :=     /I1*/I0*/Q2             ;HOLD Q2
      +     /I1*I0*/Q3              ;SHIFT RIGHT
     :+:     I1*/I0*/Q1             ;SHIFT LEFT
      +      I1*I0*/D2              ;LOAD D2
/Q3  :=     /I1*/I0*/Q3             ;HOLD Q3
      +     /I1*I0*/Q4              ;SHIFT RIGHT
     :+:     I1*/I0*/Q2             ;SHIFT LEFT
      +      I1*I0*/D3              ;LOAD D3
/Q4  :=     /I1*/I0*/Q4             ;HOLD Q4
      +     /I1*IO*/Q5              ;SHIFT RIGHT
     :+:     I1*/I0*/Q3             ;SHIFT LEFT
      +      I1*I0*/D4              ;LOAD D4
/Q5  :=     /I1*/I0*/Q5             ;HOLD Q5
      +     /I1* I0*/Q6             ;SHIFT RIGHT
     :+:     I1*/I0*/Q4             ;SHIFT LEFT
      +      I1* I0*/D5             ;LOAD D5
/Q6  :=     /I1*/I0*/Q6             ;HOLD Q6
      +     /I1*I0*/Q7              ;SHIFT RIGHT
     :+:    I1*/I0*/Q5              ;SHIFT LEFT
      +     I1*I0*/D6               ;LOAD D6
/Q7  :=     /I1*/I0*/Q7             ;HOLD Q7
      +     /I1*I0*/RILO            ;SHIFT RIGHT
     :+:    I1*/I0*/Q6              ;SHIFT LEFT
      +     I1*I0*/D7               ;LOAD D7
            /LIRO = /Q0             ;LEFT IN RIGHT OUT
            LIRO.TRST = /I1*I0
            /RILO = /Q7             ;RIGHT IN LEFT OUT
            RILO.TRST = I1*/I0
```

Figure 17. Boolean Logic Equations for an Octal Shift Register

Barrel Shifters

In most data processing systems, some form of data shifting or rotation is necessary. In typical computer systems, the shifter is located at the output of the ALU, and usually requires a single-cycle shift and add function (Figure 18). For such applications as floating-point arithmetic or string manipulation, ordinary shift registers are inefficient, since they require n clock cycles for an n-bit shift.

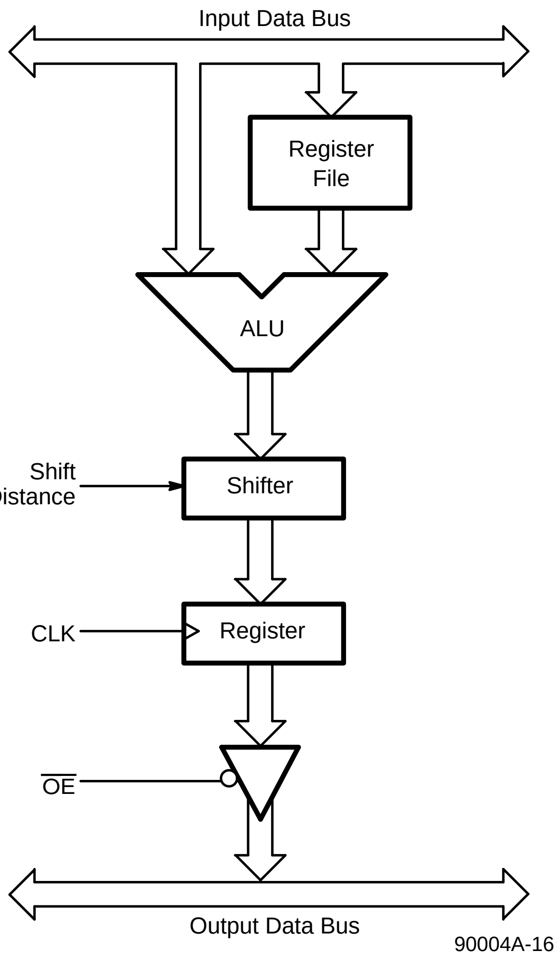


Figure 18. Typical ALU Architecture

A specialized shift register, called a "barrel shifter," is used to shift (or rotate) data by any number of bits in a single clock cycle. The name "barrel shifter" is used because of the circular nature of the shift operation. The storage registers on the output of the shifter are used in this architecture to pipeline the data operation, increasing throughput. The three-state buffer on the output registers is also useful for providing an interface to the data bus.

The design of a barrel shifter proceeds in the same manner as a regular shift register. The truth table is drawn, and the Boolean equations are then written based upon the truth tables. An eight-bit barrel shifter requires at least eight data inputs, eight registered data outputs, three control lines to specify the shift distance, a clock input and an output enable that controls the three-state buffer on the register output.

Figure 19 shows the block diagram for an eight-bit registered barrel shifter, while Table 11 shows the truth table. The registered barrel shifter requires a total of 14 inputs and 8 outputs.

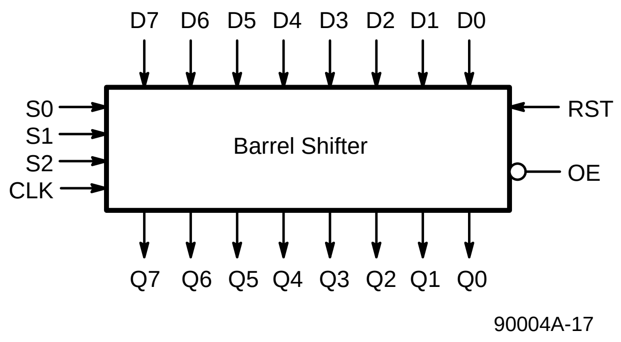


Figure 19. Block Diagram of an Eight-Bit Barrel Shifter

Table 11. Truth Table for an Eight-Bit Barrel Shifter

S2	S1	S0	Q7	Q6	Q5	Q4	Q3	Q2	Q1	Q0
0	0	0	D7	D6	D5	D4	D3	D2	D1	D0
0	0	1	D6	D5	D4	D3	D2	D1	D0	D7
0	1	0	D5	D4	D3	D2	D1	D0	D7	D6
0	1	1	D4	D3	D2	D1	D0	D7	D6	D5
1	0	0	D3	D2	D1	D0	D7	D6	D5	D4
1	0	1	D2	D1	D0	D7	D6	D5	D4	D3
1	1	0	D1	D0	D7	D6	D5	D4	D3	D2
1	1	1	D0	D7	D6	D5	D4	D3	D2	D1

Gray-Code, Johnson Counter and Shift Register Device Selection Considerations

Gray-code counters, Johnson counters and shift registers are not very logic-intensive; the number of product terms required is minimal. The D-type flip-flops provide the most efficient implementations, allowing these designs to be easily implemented in most PAL devices.

Since Gray-code counters are often used as system clocks, very high speed PAL devices provide the highest resolution clocks.

Barrel shifters are very logic-intensive and require many product terms, since data from all the inputs needs to be accessible at any output. Registered PLDs with a large number of product terms are ideal for barrel shifters. Large barrel shifters can also be partitioned into a number of PLDs.

Asynchronous Registered Designs

Until now, we have discussed strictly synchronous registered designs, where a common system clock is used. In asynchronous registered designs, a common clock is not used. The register clock may be generated by the output of another register, or by a logical combination of various other signals. Such designs are usually slow for such applications as timing generation, because when the output of one register is used to clock another, multiple delays are encountered before all the register outputs stabilize. On the other hand, designs can be very fast for asynchronous applications such as bus arbitration and control, where a fast response to a bus signal can be provided without waiting for a common system clock.

Although asynchronous designs are easier to visualize, they present larger problems in implementation. Combinatorial hazard conditions can cause false clocking of registers, destroying the logic intended by the designer. The designer also needs to worry about race conditions when clocking a number of register simultaneously. Careful design analysis is strongly recommended before implementing any asynchronous design.

Ripple counters are probably the easiest examples of such asynchronous designs. Figure 31 shows the logic diagram of a five-bit binary ripple counter. These counters clearly have the advantage of design simplicity. The output from one stage is fed as the clock to the next stage. However, this results in a slower counting rate, since the clock signals need to propagate through all five registers before the next count is reached.

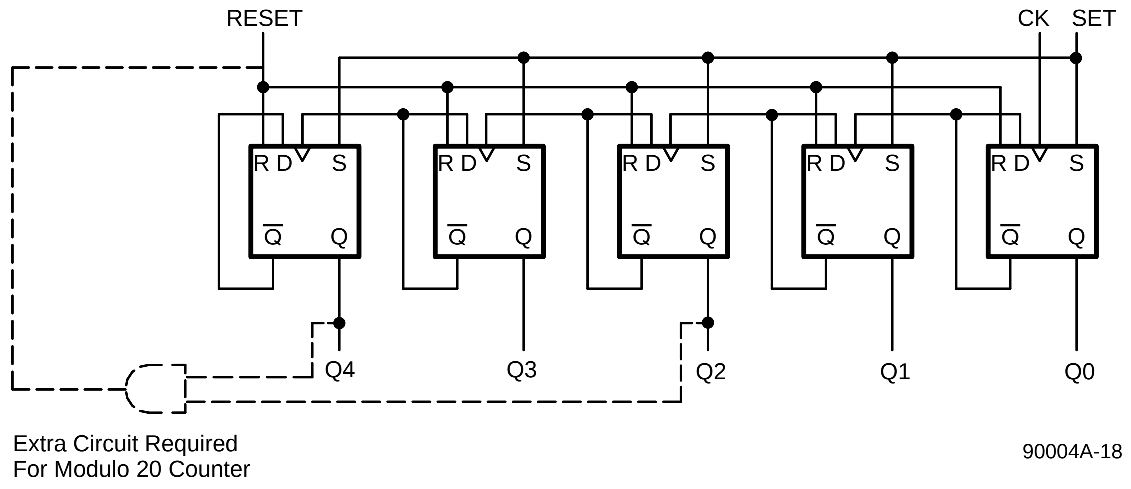


Figure 20. A Five-Bit Ripple Counter

Figure 20 shows the implementation of a modulo-20 counter that is RESET when output bits Q4 and Q2 are both HIGH. Since the RESET is implemented with a product term, the extra AND gate shown can be implemented directly within the PAL device.

Asynchronous Designs Device Selection Considerations

The device selection for asynchronous designs is easy. As the clock signals require logic, only PLDs that allow implementations of Boolean logic on the clock signals are useful.

OTHER APPLICATIONS OF REGISTERED PLDs

Registered PLDs are used for a number of miscellaneous applications that are not covered by the synchronous and asynchronous design applications discussed up to now. One such application is as a frequency divider.

- Frequency dividers
- Addressable Registers

Frequency Dividers

Standard synchronous counters provide the basic capability of dividing an input frequency. A single register of a PAL device will let us divide by two.

If we stack these registers, a binary counter provides symmetrical division by 2, 4, 8, 16, etc. This divider has been a standard for years, and the PAL device has always been on excellent choice for such applications.

One unique application of PAL devices is for dividing input frequencies by odd numbers. This has been done historically by designing a counter that cycles an odd number modulo, and decoding the specific states of the counter. The disadvantage of this approach is that the output is not symmetrical and the duty cycle is not 50%.

Let us examine a simple divide-by-five counter. This counter can be implemented using three flip-flops that start at zero and reset at four, resulting in a five-state counter. Table 12 shows the outputs of the three individual flip-flops.

Table 12. Truth Table for a Five-Bit Counter

Present State			Next State			
Q2	Q1	Q0	Q2	Q1	Q0	
0	0	0	0	0	1	State zero to one.
0	0	1	0	1	0	State one to two.
0	1	0	0	1	1	State two to three.
0	1	1	1	0	0	State three to four.
1	0	0	0	0	0	State four to zero.

The Boolean equations are:

```
Q2  := /Q2   * Q1   * Q0    ;MSB bit
Q1  := /Q1   * Q0   + Q1   * /Q0
Q0  := /Q2   */Q0          ;LSB bit
```

The waveforms for this divider are shown in Figure 21. Notice that the Q2 output goes HIGH for one state and that this output is one fifth of the input frequency, but it is a 20% duty cycle. Q1 is active for two states; it provides the same frequency, but with a 40% duty cycle. If we want a 50% duty cycle, we are going to have to divide a state in half.

To provide the 50% duty cycle, the two edges should be evenly spaced in the count sequence, one edge in the middle of state two and one at the beginning of state zero. The first edge can be formed by logically "ANDing" state_2 with the falling edge of the clock. The second edge can be formed by decoding state zero.

```
edge_1 =/clock * /Q2 * Q1*/Q0
                       ;edge between
                       ;states two and
                       ;three
edge_2= /Q2  */Q1 */Q1 ;edge at state
                       ;zero
```

The logical "OR" of these two equations will provide the needed rising edges. To provide a clean output, this signal should clock another output register.

The next step in the design is to pick the appropriate PAL device to fit this design. Our biggest concern is that we need the capability of clocking the counter at one speed and the output flip-flop at another. To do this, we cannot use a PAL device that has a dedicated clock pin; we need an architecture that allows programmable clocks.

The clock signal requires two product terms (one for each edge). Another technique is to use the independent asynchronous SET and asynchronous RESET product terms of the output register. A HIGH on the SET product term asserts the register output, and a HIGH on the RESET product term unasserts the register output. Due to the asynchronous nature of the product terms some adjustment in timing is required. The SET product term is asserted when in state 0 (Q2=0, Q1=0 and Q0=0), and the RESET product term is asserted when between states two and three.

```
OUTPUT.SET = /clock * /Q2 * Q1 * /Q0
                          ;set between
                          ;states 2 & 3
OUTPUT.RESET = /Q2 */Q1 * /Q0
                          ;reset at
                          ;state zero
```

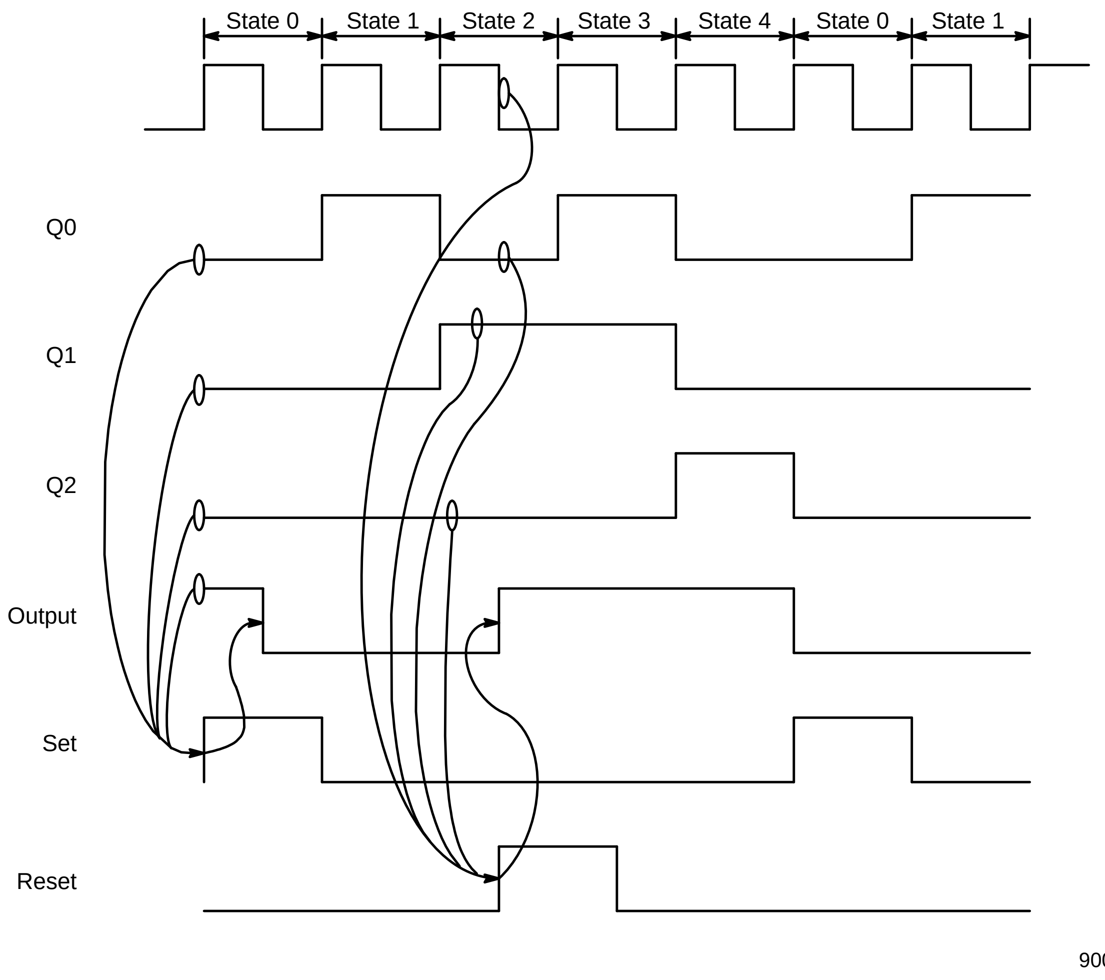


Figure 21. Waveform for a Frequency Divider

Addressable Registers

Addressable registers are commonly-used MSI functions, often implemented in PAL devices. Addressable registers are used as building blocks for digital computers. Depending upon the address input one of the many flip-flops in the register retain their previous values.