CHAPTER 8

DIGITAL INTEGRATED CIRCUITS: A PRACTICAL APPLICATION

Todd Rhoad
Austin, Texas

Keith Folken
Peoria, Illinois

1 REDUCTION OF PROCESSOR LOAD 239

2 FIELD-PROGRAMMABLE GATE ARRAYS 240

3 LOGIC BLOCKS 241

4 LOOKUP TABLES 242

5 FLIP-FLOPS 243
5.1 D Flip-Flops 244
5.2 R–S Flip-Flops 244
5.3 T Flip-Flops 245
5.4 J–K Flip-Flops 246

6 BOOLEAN LOGIC NOTATION 246

7 LOGIC GATES 246

8 COUNTERS 247

9 ADDERS, REGISTERS, AND MULTIPLEXERS 249

10 MEMORIES 249

11 PROGRAMMABLE I/O BLOCKS 251

12 PROGRAMMABLE INTERCONNECTS 251

REFERENCES 253

1 REDUCTION OF PROCESSOR LOAD

Today's processors can make many calculations in a very short amount of time. Embedded controllers make many repeated calculations several times a second. In many embedded controllers, such as an engine control or plant process control, the processor takes several inputs from outside sources, does some computation with those values, and then takes some action. This may be in the form of an output to the system, an alarm, or just a log of the data. The processor's mathematical calculations can be very time intensive. If accuracy is required, these calculations must done using several digits, sometimes with floating-point numbers. To take some of this load off of the processor, a field programmable gate array (FPGA) may be used.

An FPGA is a device of logic gates. These logic gates can be used to take the place of the computationally intensive mathematical operations that the processor would normally perform. For example, a process controller is asked to measure 10 analog inputs every 200 ms. The controller must compute a running average of the last three samples of each input. Using this average, several multiplication and division computations are made. To offload the processor, an FPGA could be used to make these repetitive calculations and feed the results back to the processor.

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

2 FIELD-PROGRAMMABLE GATE ARRAYS

Field programmable gate array technology permits the design of many different complex digital circuits using a single off-the-shelf device. The general structure for FPGAs is shown in Fig. 1. *Field programmable* refers to the ability of the FPGA function to be designed by the user in the field, as opposed to the manufacturer. Depending on the type of device, the function either is ''burned'' in permanently or semipermanently as part of the board assembly process or is loaded in from external memory every time the FPGA is powered on. FPGAs comprise an array of uncommitted circuit elements, referred to as logic blocks (LBs), programmable input/output (I/O) blocks, and programmable interconnects. Configuration of an FPGA is performed by the end user through programming. The I/O blocks function as the interface between the external device pins, the central processing unit (CPU) in this example, and the internal logic. The array of logic blocks provides the functional elements from which the end user's logic is constructed. Each logic block can independently take on any one of a limited set of personalities. These personalities are typically implemented through both Boolean logic and latching data. The individual logic and I/O blocks are interconnected by a matrix of wires and programmable switches.

FPGA architectures can vary in the size, structure, and number of logic and I/O blocks as well as the amount of connectivity of the interconnects. The secret to density and performance in these devices lies in the logic contained in their logic blocks and the performance and efficiency of their routing architecture. There are two primary classes of FPGA architectures: fine grained and coarse grained. Coarse-grained architectures consist of fairly large logic blocks, often containing two or more lookup tables and two or more flip-flops (FFs). In a majority of these architectures, a four-input lookup table implements the actual logic. The larger logic block usually corresponds to improved performance. The other architecture type is called fine grained. In these devices, there are a large number of relatively simple logic blocks. The logic block usually contains either a two-input logic function or a 4-to-1 multiplexer and a FF. These devices are good at systolic functions and have some benefits for designs created by logic synthesis. Another difference in architectures is the underlying process technology used to manufacture the device. Currently, the highest density FPGAs

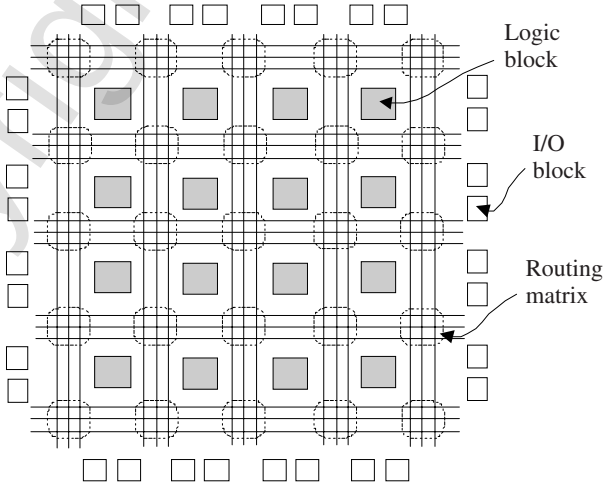


Figure 1 The FPGA contains configurable logic blocks, routing matrices, and input/output (I/O) blocks that can configure each I/O pin for a different function.

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

are built using static random-access memory (SRAM) technology, similar to microprocessors. The other common process technology is called antifuse, which has benefits for more plentiful programmable interconnect.

Reprogrammable FPGA technologies tend to be SRAM based. The routing matrices are usually implemented using pass gates driven by the value of the SRAM memory cell assigned to it. Lookup table (LUT) function generators are used to program the function of the logic block. These LUTs are small SRAM cells that have four or five inputs, which are the address lines of the SRAM cell. The output of the SRAM cell is the output of the function generator. The programmable FF after the logic block is programmed using pass gates. Then, to program the function of the logic block, the contents of the SRAM cell are loaded and configure the block to be either registered or combinatorial. Without any sort of external memory, SRAM-based FPGAs will lose their configuration if the device is turned off. Therefore, erasable programmable read-only memory (EPROM), or flash memory, connected serial or parallel, is used to provide the configuration to the FPGA during startup. That is, this configuration memory holds the program that defines how each of the logic blocks functions, which I/O blocks are inputs and outputs, and how the blocks are interconnected together.

There are a few advantages and disadvantages in using FPGA-assisted designs. Pipelining is a hardware technique of segmenting a complex operation into stages so that multiple computations can be performed simultaneously. Pipelining is a natural choice due to the discrete and regular quantities of FPGA logic blocks. Memory located at the end of the logic blocks completes the design and uses the storage as pipeline stage registers. The biggest feature of FPGA pipelining is the ability to combine numerous complex computations into a single pipeline. FPGAs have the ability to chain together any sequence of operations as required by the algorithm. Such deep pipelines are difficult to abstract for general-purpose CPUs; therefore, FPGAs offer computational potential found elsewhere only in ASICs (application-specific integrated circuits). Another major advantage is the ability to implement a large degree of computational parallelism. The parallelism most concerned here is data and algorithmic parallelism. Data parallelism is the ability to regularly process a large data set concurrently. Algorithmic parallelism refers to the ability to operate multiple independent algorithms at the same time. A combination of these two types can result in high-throughput devices with low-speed parts, which is simply streaming high-speed data through an array of low-speed computational units. FPGAs can also minimize computational logic and delay by utilizing partial evaluation techniques to reduce multivariable functions to less complicated functions based on information provided at compilation.

FPGAs do suffer physical constraints in that they are slower and are degraded in density, when compared to ASICs, due to the use of configurable SRAM-based devices to implement gates and interconnections. Gate density is extremely important since the size of the gate array determines how much logic can be implemented. Aside from physical constraints, FPGAs require relatively long delays for reprogramming the device. Current chips require configurations to be downloaded one at a time, resulting in significant dead time on the order of a few milliseconds. For applications where the process time is of the same order, significant issues can arise. Other weaknesses for FPGA-based designs include poor floating-point performance, poor data proximity, and overall design complexity.

3 LOGIC BLOCKS

The logic block of an FPGA is the core building block and is used to implement combinational and sequential logic. This block implements the actual logic functions for configurable computing. A general representation of a logic block is shown in Fig. 2. This model

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

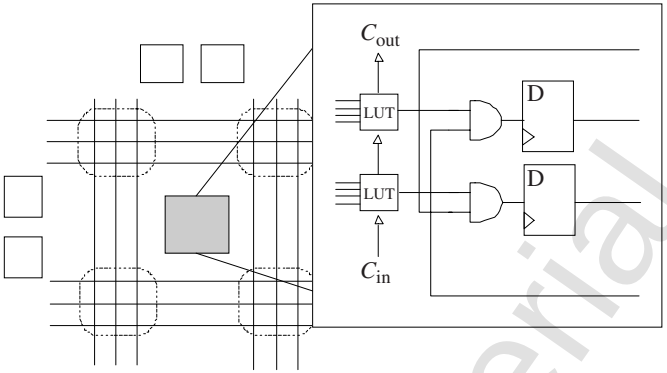


Figure 2 General logic block with two-input logic and D-type FFs.

uses three stages. The first stage is function generation, which utilizes an array of programmable LUTs. These LUTs typically have between three and five inputs, with four inputs being cited as the best compromise between LUT access time and the average desired function delay.[1] The second stage, the programmable routing stage, follows the function generators and allows function results to be supplied to the logic block's final stage with minimal delays. Finally, in the memory stage, the logic block uses D FFs to implement bit storage within the logic block. The FFs are beneficial for distributed storage in configured designs, including their use as registers between pipeline stages.

Logic blocks have enough flexibility to operate in three distinct modes: combinational logic, arithmetic or ripple modes, and dedicated memory storage. In combinational logic mode, the LUTs are loaded at configuration time with truth tables to implement a logic function that is independent of LUT inputs. The arithmetic or ripple mode uses carry gates to rapidly propagate signals from a logic block to its neighbors. Lastly, the memory stage allows the LUT to be configured as RAM or ROM units. This mode increases the amount of internal memory within the device and is useful for embedding memory elements throughout a design.

4 LOOKUP TABLES

A LUT of a regular, single-context FPGA consists of a bank of SRAM cells and a multiplexer. The depth of the SRAM cell is one, and it should be as wide as the data input of the multiplexer. The output bits of the SRAM are directly connected to the data input pins of the multiplexer. The inputs to the LUT control the select signals of the multiplexer. Based on the LUT inputs, an SRAM output is selected by the multiplexer and sent to the LUT output.

A K-input lookup table (K-LUT) is a digital memory with K address lines and a one-bit output. This memory contains $2K$ bits and is capable of implementing any Boolean function of K variables. A LUT of K inputs can implement $22K$ different Boolean functions. A simple example of a two-input LUT implementing a function $f = a + \overline{b}$ is shown in Fig. 3, where the LUT is described in terms of a multiplexer. In the LUT, there are two address lines (a and b) and four memory cells. The inputs to the LUT are the select lines of the multiplexer, and the memory cells serve as the inputs to the multiplexer. If $a = 0$ and $b = 1$, for example, then the memory cell corresponding to select line 01 will be connected to

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

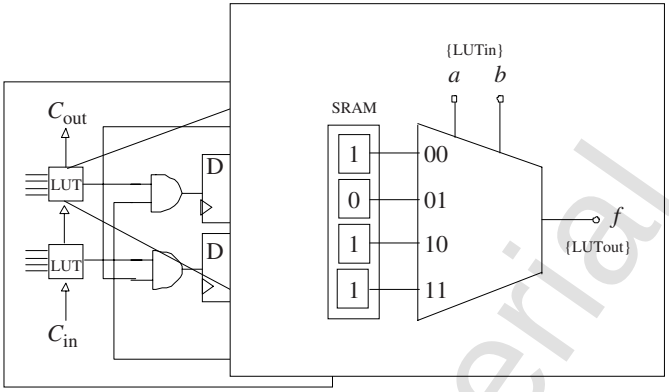


Figure 3 A two-input LUT implementing the function $f = a + \overline{b}$.

the output f. The contents of the memory cells are calculated from the evaluation of $f = a + \overline{b}$ for all the combinations of logic values of a and b.

5 FLIP-FLOPS

Flip-flop devices are used in the digital field for a variety of purposes such as storing data temporarily, multiplication or division, counting, or receiving and transferring information. Flip-flops are bistable multivibrators. Multivibrators are classified into three types: bistable, monostable, and astable. Their behavior is illustrated by the simple mechanical analogies in Fig. 4. The bistable monovibrator, or FF, is analogous to the seesaw or teeter-totter of a children's playground. If an input in the form of a sufficient downward force is applied to the up end, then it will assume the down position and the other end will move into the up

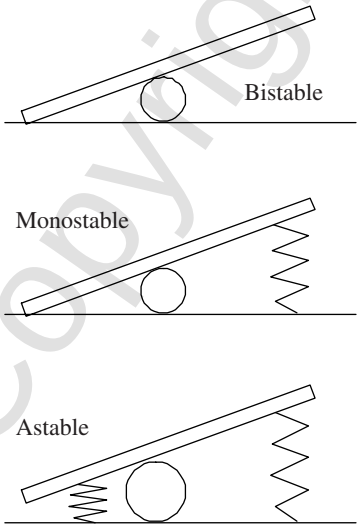


Figure 4 Mechanical analogies for bistable, monostable, and astable multivibrators.

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

position and remain there after the input is removed. Further inputs corresponding to downward forces on this end will have no effect. Also, the state of the system can be described by the position of either end, and if logical variables are introduced to designate these states, the variables are complementary. The system exhibits the characteristics of memory in that it remains in the state it has been set to with no further input.

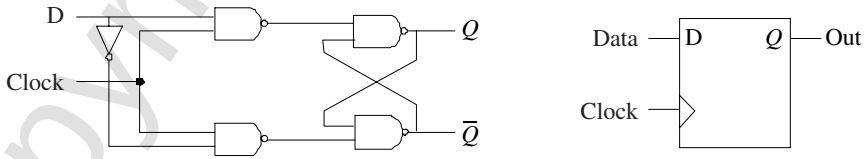
The types of FFs used in digital equipment are identified by the inputs. They may have from two up to five inputs depending on the type. They are all common in one respect in that they have only two distinct output states. The outputs are normally labeled Q and $\overline{Q}$ and should always be complementary. If $Q = 1$, then $\overline{Q} = 0$, and vice versa. In this section, the four types of FFs that are most common to digital equipment will be discussed and are the D, R–S, T, and J–K FFs.

5.1 D Flip-Flops

The D FF gets its name from its ability to hold data in its internal storage. It is sometimes referred to as a gated D latch and is shown in Fig. 5. This is because the presence of the clock pulse at the enable pin will transfer the input D to the output of the FF. The D FF has two inputs: D and an Enable or Clock. The state of its output, Q, follows that of input D at the time of the clock signal's transition from low to high and is delayed by the device propagation time. In other words, the D-type latch changes its output state when the clock input is in the high state and retains that value when the clock input is in the low state. Thus regardless of the initial state condition the final state assumes the value of the data input. This value will be held until the next clock pulse, at which time it may be updated. The influence of the clock exists for a finite albeit short time around the edge. For proper operation it is necessary that the logical level D be established for an appropriate time preceding the clock edge. This time is referred to as the setup time. The level must also be maintained for a time following the edge. This is referred to as the hold time. Clearly the time interval between clock pulses, which controls the ultimate speed with which a digital system performs, must always exceed the sum of the setup and hold times.

5.2 R–S Flip-Flops

The R–S FF is used to temporarily hold or store information until it is needed. A single R–S FF will store one binary digit, either a 1 or a 0. Storing a four-digit binary number would



CLK	D	Q	
0	1	0	Start
↑	1	1	Store 1
0	0	Q	No charge
↑	0	0	Store 0

Figure 5 Clocked D FF configuration (upper left), symbol (upper right), and truth table (bottom).

Copyrighted Material

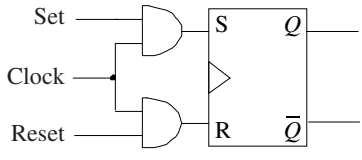
Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

require four R–S FFs. The name for the R–S FF is derived from the inputs, R for reset and S for set, and is often referred to as an R–S latch. The R–S FF has two output conditions. When the Q output is high and $\overline{Q}$ is low, the FF is set. When Q is low and $\overline{Q}$ is high, the FF is reset. When the R and S inputs are low, the Q and $\overline{Q}$ outputs will both be HIGH. When this condition exists, the FF is considered to be jammed and the outputs cannot be used. The jammed condition is corrected when either S or R goes high. To set the FF requires a high on the S input and a low on the R input. To reset, the opposite is required; S input low and R input high. When both R and S are high, the FF will hold, or "latch," the condition that existed before both inputs went high.

In a complex digital system the signal levels are dynamic, and it is important to define precisely the time at which transitions take place, thereby defining the time at which the correct state is assumed. This is accomplished by issuing a clock pulse generated by an astable multivibrator to all FFs in the system. Such a configuration is shown in Fig. 6 where the AND gates ensure that the set and reset logic levels are only applied at clock time. Here the AND gates play the role of clock-controlled switches. In the absence of the clock signal, shown as the triangle on the FF, the system is in the hold mode with S and R at a value of 0. The clock pulse may be of significant duration. However, these devices are typically fairly sensitive to changes on the input lines during the period when the clock pulse is present. Therefore, the clock pulse is normally shortened to a narrow spike before being applied to the two AND gates. This can be accomplished by differentiating the clock pulse. The assembly of differentiator and clocked FF constitutes an edge-triggered FF since transitions are restricted to a narrow time at the edge of the clock pulse. These circuits are triggered by either the rising edge or the falling edge of the clock pulse. This presents two device states of concern: the state prior to the edge of the clock pulse (Q) and the new state after the edge of the clock pulse (Q'). Figure 6 shows the possible states for this configuration.

5.3 T Flip-Flops

The T, or toggle, FF is a bistable device that changes state on command from a common input terminal. The T-type FF is often used in counting applications. As with the other types of FFs, the T FF is a derivative of the R–S FF. Comparatively speaking, the S and R inputs are internally connected, R to Q and S to its complement, so that the only external input is the clock. If an inverter precedes the T input, it indicates that the FF will toggle on a high-to-low transition of the input pulse. The absence of an inverter indicates the FF will toggle on a low-to-high transition of the pulse.



S	R	Q	Q'
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	x
1	1	1	x

Figure 6 R–S FF configuration (left) and truth table (right).

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

5.4 J–K Flip-Flops

The J–K FF is very similar to the R–S FF except that there are no indeterminate states. From Fig. 7, it can be observed that for the first three combinations of the truth table, the J–K FF behavior is identical to the R–S FF with J = S and K = R. The observable difference is in the combination J = 1, K = 1. For in this state, the device becomes a toggle FF. A convenient aspect of this arrangement is that the indeterminate state of the RS FF cannot occur and all four combinations of the external inputs are allowed.

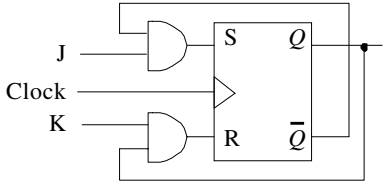
6 BOOLEAN LOGIC NOTATION

George Boole created Boolean logic, sometimes referred to as Boolean algebra, in the nineteenth century. He based his concepts on the assumption that most quantities have two possible conditions: true and false. A Boolean expression is a description of the input conditions required to get the desired output, which are based on Boole's laws and theorems. Boolean algebra is primarily used by design engineers to arrange logic gates to accomplish specific tasks while enabling the designers to achieve the desired output by using the fewest number of logic gates. Naturally, space, weight, and cost are important factors in the design of equipment; one would usually want to use as few parts as possible.

A good starting point is the existence of only two binary values—true (T or 1) and false (F or 0)—and three operators—AND (Boolean multiply), OR (Boolean addition), and NOT (Boolean inversion)—with behavior described by their truth tables. Boolean algebraic formulas follow certain conventions. Assume our function is $f = A + B$. This basically states that "f is TRUE when A or B is TRUE." Boolean logic also follows commutation, association, and distribution laws. AND and OR operators commute, such as $A + B \equiv B + A$. Distributive property is such that $A + (B + C) \equiv (A + B) + C$. Lastly, the distributive property is evident from $A \cdot (B + C) \equiv (A \cdot B) + (A \cdot C)$. The conventional hierarchy of operator action in Boolean expressions is NOT, then AND, last OR. Another important law useful in Boolean algebra to convert AND to OR and OR to AND is De Morgan's law, which states that $\overline{A \cdot B} \equiv \overline{A} + \overline{B}$ and $\overline{A + B} \equiv \overline{A} \cdot \overline{B}$.

7 LOGIC GATES

Given the laws of commutation and association, many logic gates can be implemented with more than two inputs, and for reasons of space in circuits, usually multiple-input, complex gates are made. The gates discussed in this section are the NOT, AND, OR, NAND, and NOR. The logical NOT is denoted by the overscore and is defined by variables listed in the truth tables, which show all possible combinations of inputs and outputs. For example, if we have input A and its current value is 0, then NOT A is equal to 1. Figure 8 represents the truth table for a single input variable and its NOT function values.



J	K	
0	0	Q
0	1	0
1	0	1
1	1	$\overline{Q}$

Figure 7 J–K FF configuration (left) and truth table (right).

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

A	$\overline{A}$
0	1
1	0

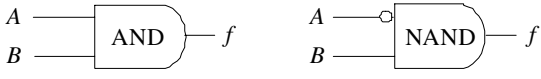
Figure 8 Truth table for input A and NOT A.

The AND gate performs logical multiplication, commonly known as the AND function. The AND gate has two or more inputs and single output. The output of the AND gate, denoted as f in Fig. 9, is high only when all its inputs are high. The Boolean expression for this operation is $f = A \cdot B$. It may also be written without the dot, or simply $f = AB$. The expression is spoken, "$f = A$ AND B." The dot, or lack of, indicates the AND function. A simple way to remember the operation of the AND gate: In order to produce a 1 output, all the inputs must be 1. If any or all of the inputs are 0, then the output will be 0. By placing an inverter on input A of the AND gate, we get the NAND gate. The function that would describe this operation is $f = \overline{A} \cdot B$ and may be spoken as "f = Not A AND B." By inverting input A, we are specifying that the only condition that will cause a high output (or 1) is for input A to be a 0 and input B to be a 1. Every other combination will result in a low (or 0) output.

The logical OR gate differs from the AND gate in that only one input signal needs to be high to get a high (or 1) output. As with the AND gate, there can be numerous input signals. However, for simplicity, we will only consider two inputs. For an example, we will use the gates shown in Fig. 10 to produce a high output: variable A, variable B, or both must be high. The Boolean expression for this operation is $f = A + B$ and is spoken "$f = A$ OR B." The plus sign indicates the OR function and should not be confused with addition. By inverting inputs A and B (shown as small circles on the inputs of the gate), the output changes from any high generating a high output as in the OR to any low (or 0) input generating a high output.

8 COUNTERS

A counter is simply a device that counts. Counters may be used to count operations, quantities, or periods of time. They may also be used for dividing frequencies, for addressing information in storage, or for temporary storage. Counters are a series of FFs wired together to perform the type of counting desired. They will count up or down by 1s, 2s, or more. A counter's modulus refers to the total number of counts or stable states a counter can indicate. For example, the modulus of a four-stage counter would be 16 (base 10), since it is capable



$f = A \cdot B$ $\quad$ $f = \overline{A} \cdot B$

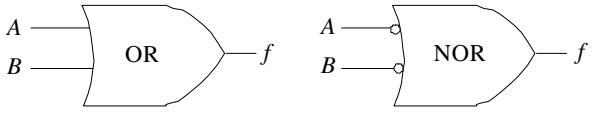
A	B	f
0	0	0
0	1	0
1	0	0
1	1	1

A	B	f
0	0	0
0	1	1
1	0	0
1	1	0

Figure 9 Gate logic symbol for AND and NAND gates with their respective truth table.

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com



$f = A + B$

A	B	f
0	0	0
0	1	1
1	0	1
1	1	1

$f = \overline{A} + \overline{B}$

A	B	f
0	0	1
0	1	1
1	0	1
1	1	0

Figure 10 Gate logic symbol for OR and NOR gates with their respective truth table.

of indicating 0000 to 1111 (base 2). The term *modulo* is used to describe the count capability of counters; that is, modulo-16 for a four-stage binary counter, modulo-11 for a decade counter, modulo-8 for a three-stage binary counter, and so on. Ripple counters, as shown in Fig. 11, are so named because the count is like a chain reaction that ripples through the counter because of the time involved. The ripple counter is also called an asynchronous counter. Asynchronous means that the events (setting and resetting of FFs) occur one after the other rather than all at once. Because the ripple count is asynchronous, it can produce erroneous indications when the clock speed is high. A high-speed clock can cause the lower

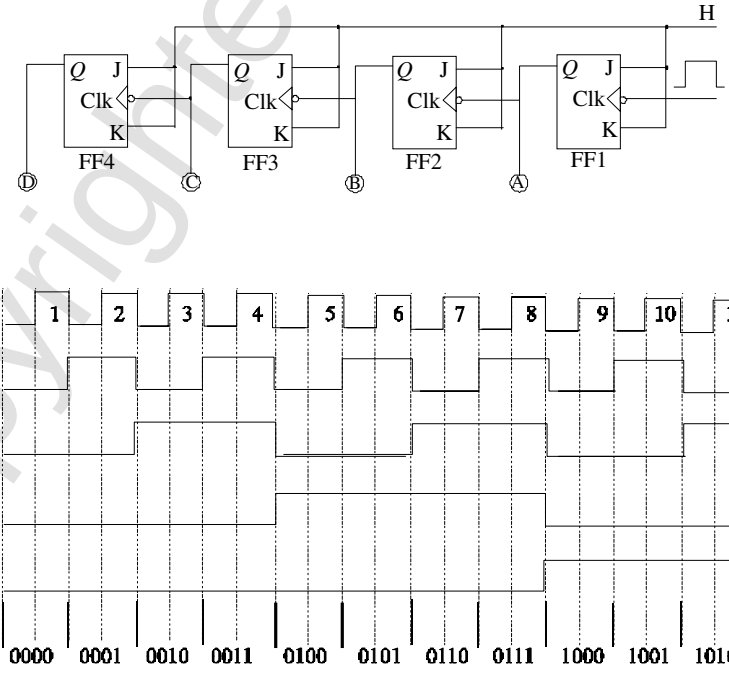


Figure 11 Logic and timing diagram of a four-stage ripple counter.

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

stage FFs to change state before the upper stages have reacted to the previous clock pulse. The errors are produced by the FFs' inability to keep up with the clock. High-frequency operations require that all the FFs of a counter be triggered at the same time to prevent errors. The synchronous counter is similar to a ripple counter with two exceptions: The clock pulses are applied to each FF, and additional gates are added to ensure that the FFs toggle in the proper sequence.

9 ADDERS, REGISTERS, AND MULTIPLEXERS

Development of most digital systems is a fairly complicated task that usually requires designers to divide the work into manageable building blocks and standardize subunits. For digital designers, these building blocks are typically adders, registers, and multiplexers. These circuits are either combinational or sequential. Combinational circuits have outputs that depend only on the present value or state of the input. Sequential circuits take into consideration the state of the past outputs. A multiplexer is a device for selecting one of several possible input signals and making it available to a specified output. A simple mechanical analogy is that of a manual selector switch such as the dial on a radio that can select AM, FM, cassette, CD, and so on. A register is an ordered set of FFs and is often used for temporary storage of a set of bits for a particular operation. One of the most elegant of registers is that of the D register because of the ability to connect the system clock directly to the device's clock input. Other registers such as the shift register perform lateral movements of data from one bit position to an adjacent position. Shift registers are commonly configured as serial in, parallel out; parallel in, serial out; parallel in, parallel out; and serial in, serial out.

In the previously mentioned FPGA example, we discussed using the FPGA to offload mathematical calculations to reduce the workload on the main processor. To do this, the logic blocks must contain circuits to do these calculations. For example, multiplication is easily accomplished with repeated applications of an adder to find the product of two numbers. Consider, for example, the case where we wanted to use decimal arithmetic to sum two numbers and determine the hundreds digit. This requires not only the addition of the hundreds digit of each number but also the carryover of the tens digits, assuming one exists. This requirement is also true in binary arithmetic. It can be computed by first adding the two bits corresponding to the 2^n digit and then adding the resultant to the carry from the 2^{n-1} digit. With only two inputs, a half adder can be constructed to perform this operation. If three inputs are provided, a full adder can be used.

10 MEMORIES

Read-only memory is programmed with transistors to supply the desired values. These values are usually set by programming them when the memory device is offline or inactive. Programming of ROM is typically done before inserting it into an embedded system. ROM has many purposes, such as the storing of a software program for use by a general-purpose processor, converting binary code to another type of code, and implementing a combinational circuit. ROM can be used to implement any combinational function of k variables by using a $2^k \times 1$ ROM and can implement n functions of the same k variables by using a $2^k \times n$ ROM. For this application, the ROM is simply programmed to implement a truth table for these functions.

Most other systems use user-programmable ROM devices, or PROM, which can be programmed by the chip's user well after the chip has been manufactured. These devices

Copyrighted material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

are best suited to prototyping and to low-volume applications. Programming a PROM is accomplished by the user who provides a file indicating the desired ROM contents. A tool called a ROM programmer then configures each programmable connection according to the user-specified file. A basic PROM uses a fuse for each programmable connection. The ROM programmer blows the selected fuses in the array by passing a large current at high voltage through them. However, once a fuse is blown, the connection can never be reestablished. This type of programming cannot be undone. For this reason, basic PROM is often referred to as one-time-programmable device, or OTP. As a result of this ability, OTP devices are frequently used in military and space applications for their robustness in the presence of radiation.[3]

Programmable ROMs do come in erasable versions. One type of erasable PROM, called EPROM, utilizes a floating-gate complementary metal–oxide–semiconductor (CMOS) device as the programmable component, as shown in Fig. 12.[4] Typically, a higher voltage (12–25 V) is applied to the control gate and the drain. This causes avalanche injection whereby electrons tunnel into the floating gate. Once the voltage is removed, the charges are trapped in the floating gate and programming is complete. To erase the programming, the electrons must be excited enough to escape from the gate. One source of energy used is ultraviolet light, or UV. The exposure time to UV is typically 5–30 minutes through a small quartz window on top of the chip. In an effort to eliminate the time spent erasing the programming, the development of an electronically erasable device, or EEPROM, was developed. These devices are more expensive but are also more convenient to use. One type of EEPROM is FLASH memory, which has the advantage of being able to reprogram certain regions of the memory rather than the whole memory at once.

Random-access memory, or RAM, has the advantage of being able to read from and write to. In contrast to ROM, RAM is not programmed. It is empty when placed in the embedded system and is written to or read from during its execution. There are two basic types of RAM: static and dynamic. Static RAM, or SRAM, is the faster version and uses a memory cell consisting of a FF to store a bit. Six transistors are required to hold this bit. The term static refers to the fact that the RAM will hold its data only as long as power is applied. SRAM is typically used for high-end applications such as cache memory.

Dynamic RAM (DRAM) is more compact than SRAM because it has one MOS transistor and a capacitor to store each bit. The disadvantage is that the capacitor cannot store the information indefinitely because it will leak charge and eventually lose the data. This forces the requirement to recharge or refresh this memory. Refresh rates are typically around 15.625 μs. Refreshing can be done by reading the contents of the cell, which forces the data to be stored in a buffer and then rewritten to the memory cell. Refreshing the memory cells tends to make DRAM slower than SRAM.

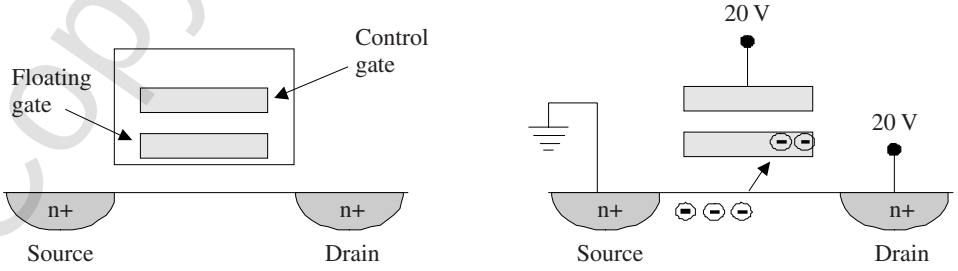


Figure 12 Floating gate transistor programming for erasable PROMs. EPROMs, EEPROMS, and Flash use different methods for controlling the charge of the floating gate.

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

11 PROGRAMMABLE I/O BLOCKS

An I/O interface block is peripheral logic built to interface the chip to external devices through its package pins. The term *programmable* indicates that the pins can be programmed for a wide variety of I/O standards such as an input, output, or bidirectional. In state-of-the-art FPGA devices, these I/O resources are typically grouped into banks. The compatibility of these standards is based on the standard's supply and reference voltage requirements. In complicated designs that use more than one standard, complications arise that lead to constrained I/O pad placement issues, such as not being able to place certain I/O pads in the same bank. This is a real problem if there is only one bank in the design. Current FPGAs support several I/O standards. For example, some standards require use of a differential amplifier input. With this standard, the external reference voltage must be applied to the amplifier. This is an advantage in that it reduces the I/O voltages swings, which allows for faster switching. Another standard may require a power supply for the I/O blocks. Figure 13 illustrates some standard requirements and an I/O bank layout.

Multiple functionality of I/O resources is important because the number of pins available is limited by the size of the device package. Input/output block (IOB) paths will be placed close to the device I/O pins to overcome issues at input and output. At output, long delays from internal logic block outputs to the IOBs make it difficult to connect at very high clock rates to external synchronous systems. External system setup and hold time issues occur if there are long delays from the I/O pins to the logic block inputs if the external inputs are clocked directly into the logic block FFs without being captured initially by a FF at the IOB pin. A simplified I/O block configuration is shown in Fig. 14.

12 PROGRAMMABLE INTERCONNECTS

The logic blocks in an FPGA are arranged in rows and columns. The number of rows and columns must be equal. Communication among the logic blocks and the I/O cells is typically done through a programmable interconnect network consisting of many wires organized into channels in both the horizontal and vertical directions. Connections between the intersections of the wires (horizontal and vertical) are made at programmable interconnect points (PIPs), sometimes referred to as switch blocks. These points are controlled by writing the configuration RAM, same as with the logic blocks. Wire segments in the programmable interconnect network are bounded by these PIPs and are considered to be either global or local routing

I/O Standard	Direction	V_{ref} Requirement	V_{cco} Requirement
Peripheral component interface (PCI)	Input	Not req.	3.3 V
	Output	Not req.	3.3 V
	Bidirectional	Not req.	3.3 V
Gunning transceiver logic (GTL)	Input	0.8 V	Not req.
	Output	Not req.	Not req.
	Bidirectional	0.8 V	Not req.
High-speed transceiver logic class I (HSTL I)	Input	0.75 V	Not req.
	Output	Not req.	1.5 V
	Bidirectional	0.75 V	1.5 V

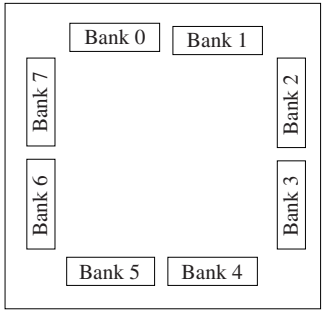


Figure 13 Some I/O standard voltage requirements (left) and organization of Virtex-E FPGA (right).[2]

Copyrighted material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

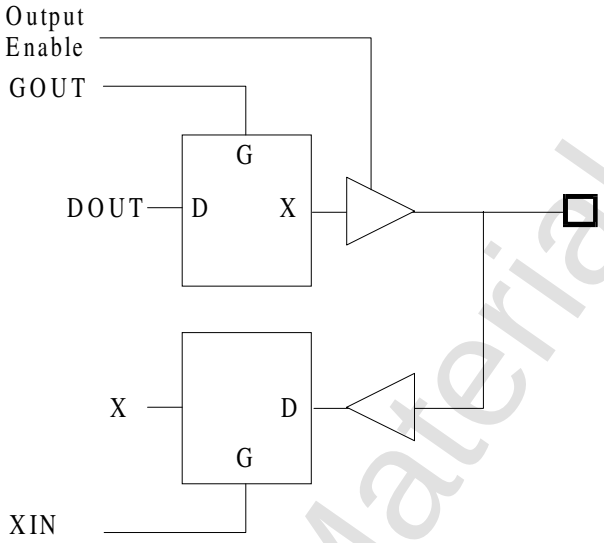


Figure 14 Simplified I/O block.

resources. Global routing resources connect nonadjacent logic blocks and are used to define the combinations of channels. Local routing resources connect a logic block to global routing resources or to adjacent logic blocks. In short, local routing refers to which wire is in each channel. Programming technologies used include Antifuse, SRAM, and EEPROM.

The Antifuse switch, shown in Fig. 15, is a two-terminal device. It is designed so that it offers high resistance when not programmed and once programmed the resistance drops close to 0 Ω. This method offers only one-time programming and any error made during programming cannot be corrected. The Antifuse array is formed with horizontal and vertical wires and with an Antifuse switch at every intersection. Based on the required configuration of the FPGA, high voltage is applied to particular switches. The main disadvantage of this technology is that it requires a separate on-chip circuitry to deliver the high voltage for programming.[5]

In SRAM-based technology, pass gates are used as switches controlled by RAM cells. These RAM cells are usually implemented by LUTs and sometimes multiplexers. Some disadvantages are the huge space requirements needed and the large *RC* delay to the signal, which makes SRAM-based technology slower than Antifuse.[6] Another major disadvantage is that it is volatile, or it loses its memory every time it is powered up. The major advantage is the reprogrammability, which allows for full testing after fabrication.

Lastly, the EEPROM is a viable option. As mentioned previously, it has a control gate and a floating gate. During normal operation, these transistors look like open circuits, and

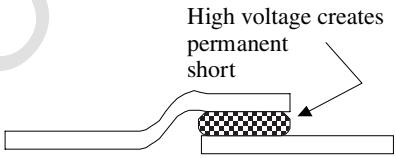


Figure 15 Antifuse technology.

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

during power-down, the data are held due to the floating gate. This technology has the nonvolatility of the Antifuse with the programmability of the SRAM. Unfortunately, the on-state resistance is higher than that of the Antifuse (before high voltage is applied) and the programming time required is larger than the SRAM technology.

REFERENCES

1. V. Betz and J. Rose, "How Much Logic Should Go in an FPGA Block?" *IEEE Design and Test of Computers,* January–March 1998, pp. 10–15.
2. J. Anderson, J. Saunders, S. Nag, C. Madabhushi, and R. Jayaraman, "A Placement Algorithm for FPGA Designs with Multiple I/O Standards," paper presented at the International Conference on Field-Programmable Logic and Applications (FPL), Villach, Austria, 2000, in *Lecture Notes in Computer Science,* Vol. 1896, Springer-Verlag, pp. 211–220.
3. I. Cyliax, "Learning the Ropes: The FPGA Tour." *Circuit Cellular Online,* Nov. 1999, www.circuitcellular.com/online.
4. R. Patel, M. Wong, J. Costello, D. Reese, V. Bocchino, M. Chu, and J. Turner, "A 90.7 MHz-2.5 Million Transistors CMOS PLD with JTAG Boundary Scan and In-System Programmability," in *Proc. IEEE Custom Integrated Circuits Conference,* May 1–4, 1995, Santa Clara, California, pp. 507–510.
5. M. Ahrens, A. El Gamal, D. Galbraith, J. Greene, and S. Kaptanoglu, K. R. Dharmarajan, L. Hutchings, S. Ku, P. McGibney, J. McGowan, A. Sanie, K. Shaw, N. Stiawalt, T. Whitney, T. Wong, W. Wong, and B. Wu, "An FPGA Family Optimized for High Densities and Reduced Routing Delay," in *Proc. IEEE Custom Integrated Circuits Conference,* May 13–16, 1990, San Jose, California, pp. 31.5.1–31.5.4.
6. H. Hsieh, W. Carter, J. Y. Ja, E. Cheung, S. Schreifels, C. Erickson, P. Freidin, and L. Tinkey, "Third-Generation Architecture Boosts Speed and Density of Field-Programmable Gate Arrays," in *Proc. Custom Integrated Circuits Conf.,* 1990, pp. 31.2.1–31.2.7.

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com