FPGA Basics

Introduction

Field programmable gate arrays (FPGAs) are digital integrated circuits (ICs) that contain configurable (programmable) blocks of logic, configurable interconnects between these blocks and *input/output blocks (IOBs)* [1]. **Figure 1** shows the top-down view of a simple general FPGA architecture.

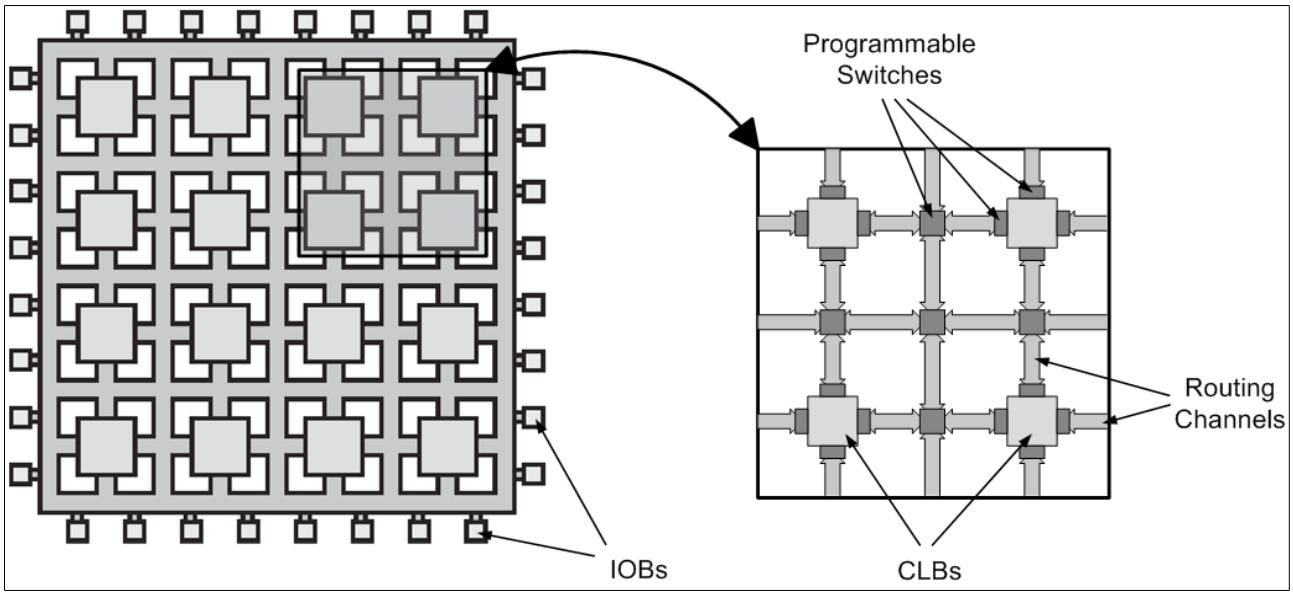

Figure 1 : Top-down view of a simple general FPGA architecture

The *configurable logic blocks (CLBs)* are arranged in a two-dimensional array. Configurable interconnects consist from interconnection wires and *programmable switches*. The interconnection wires are organized as horizontal and vertical *routing channels* between rows and columns of logic blocks [2]. Interconnection wires and programmable switches allow the logic blocks to be interconnected in many ways, resulting in multilevel logic function implementations. IOBs are used for the connection of CLBs with the FPGA package pins. IOBs connect to the CLBs the same way CLBs connect to each other. The actual number of CLBs, programmable switches and wires and IOBs in an FPGA varies in commercially available chips. FPGAs can be used to implement logic circuits of more than a million equivalent gates in size and 1000 I/O ports.

The “field programmable” portion of the FPGA’s name refers to the fact that its programming takes place “in the field” (as opposed to devices whose internal functionality is hard-wired by the manufacturer). This may mean that FPGAs are configured in the laboratory, or it may refer to modifying the function of a device resident in an electronic system that has already been deployed in the outside world [1].

FPGA structure and basic operation

Although many technologies are used for FPGA implementation (e.g. Antifuse, E^2PROM, FLASH,

SRAM) [1], only SRAM-based FPGAs (implementation using SRAM-based programmable cells) will be presented because SRAM-based FPGAs are used in the majority of typical applications. The two most-used technologies are SRAM and Antifuse [3]. Antifuse, in contrast to SRAM, is one time programmable (OTP) [1]. Antifuse-based FPGAs are relatively immune to the effects of radiation. This is why Antifuse-based FPGAs are used in military and aerospace applications [1].

SRAM technology

Figure 2 shows an SRAM cell, which is used in SRAM (Static Random Access Memory) memories [2] to store 1 bit of information. It operates as follows. To store data into the cell, the *Sel* input is set to 1, and the data value to be stored is placed on the *Data* input. The SRAM cell may include a separate input for the complement of the data, indicated by the transistor shown in magenta in the figure. For simplicity, is assumed that this transistor is not included in the cell. After waiting long enough for the data to propagate through the feedback path formed by the two NOT gates, *Sel* is changed to 0.

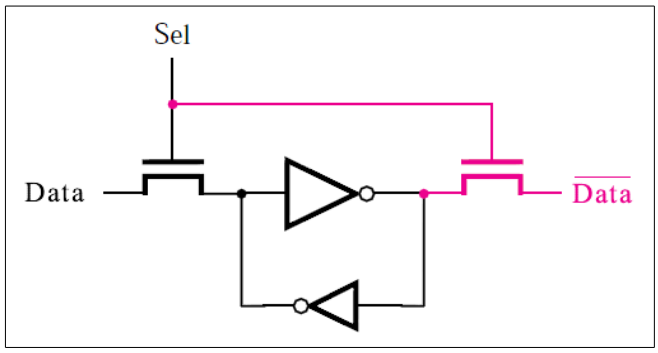

Figure 2 : An SRAM cell

A possible problem is that when *Sel* = 1, the value of *Data* may not be the same as the value being driven by the small NOT gate in the feedback path. Hence, the transistor controlled by *Sel* may attempt to drive the stored data to one logic value while the output of the small NOT gate has the opposite logic value. To resolve this problem, the NOT gate in the feedback path is built using small (weak) transistors, so that its output can be overridden with new data.

To read data stored in the cell, *Sel* is set to 1. In this case the *Data* node would not be driven to any value by external circuitry, so that the SRAM cell can place the stored data on this node. The *Data* signal is passed through a buffer, not shown in the figure, and provided as an output of the SRAM block.

The stored data remains in the feedback loop indefinitely unless it is specifically altered or until power is removed from the system (SRAM memory is volatile – SRAM cell's data is lost when power is removed from the system).

An SRAM pass-transistor switch (or SRAM switch) consists from one SRAM cell and one transistor (called pass-transistor) [1], [4]. The value stored in the SRAM cell drives the pass-transistor's gate. Depending on the contents of the SRAM cell (logic 0 or logic 1), the pass-transistor will either be OFF (disabled) or ON (enabled). **Figure 3** shows an SRAM switch.

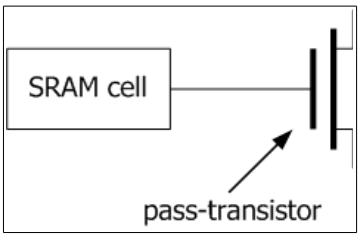

Figure 3 : An SRAM switch

SRAM cells, which provide the SRAM-based FPGAs the programmable capability, can also be used in the FPGA fabric as latches, flip-flops and distributed memory for design purposes [1]. Since SRAM is volatile, the FPGA must be programmed at the time of chip power-up. This requires external permanent memory to provide the programming bitstream. Besides volatility, a major disadvantage of SRAM technology is its large area. At least five transistors are needed to implement an SRAM cell, plus at least one transistor to implement an SRAM switch. Since on-chip programming is done with SRAM cells, the programming of the FPGA can be done an unlimited number of times. This allows prototyping to proceed iteratively, re-using the same chip for new design iterations. Reprogrammability has advantages in systems as well. In cases where parts of the logic in a system are not needed simultaneously, they can be implemented in the same reprogrammable FPGA and FPGA logic can be switched between applications [4].

Early FPGA devices

The first FPGAs were based on CMOS and used SRAM cells for configuration purposes. Although these early devices were comparatively simple and contained relatively few gates by today's standards, many aspects of their underlying architecture are still employed to this day [1].

The early devices were based on the concept of a programmable logic block, which basically comprised a *3-input lookup table (LUT)*, a register that could act as a *flip-flop* or a *latch*, and a *multiplexer (mux)*. **Figure 4** shows a simple programmable logic block.

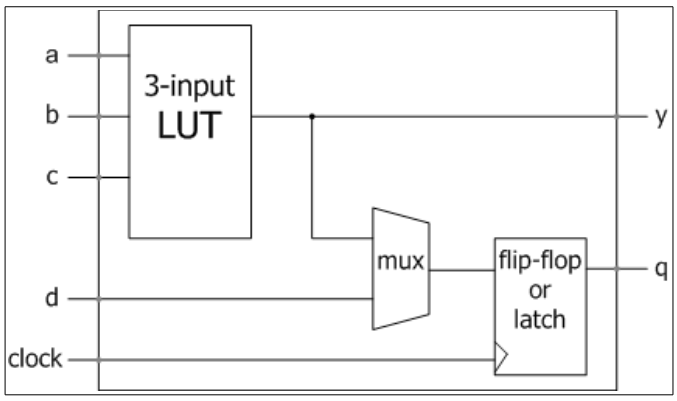

Figure 4 : A simple programmable logic block

Each FPGA contained a large number of these programmable logic blocks. By means of appropriate SRAM programming cells, every logic block in the device could be configured to perform a different

function. Each register could be configured to initialize containing a logic 0 or a logic 1 and to act as a flip-flop or a latch. If the flip-flop option were selected, the register could be configured to be triggered by a positive- or negative-going clock (the clock signal was common to all of the logic blocks). The multiplexer feeding the flip-flop could be configured to accept the output from the LUT or a separate input to the logic block, and the LUT could be configured to represent any 3-input logical function.

The underlying concept behind a LUT is relatively simple. A group of input signals is used as an index (pointer) to a lookup table. The contents of this table are arranged such that the cell pointed to by each input combination contains the desired value [1]. A LUT is also called a logic-function generator. **Figure 5** shows the implementation of the function *Y = (a AND b) OR (NOT c)* using a 3-input LUT.

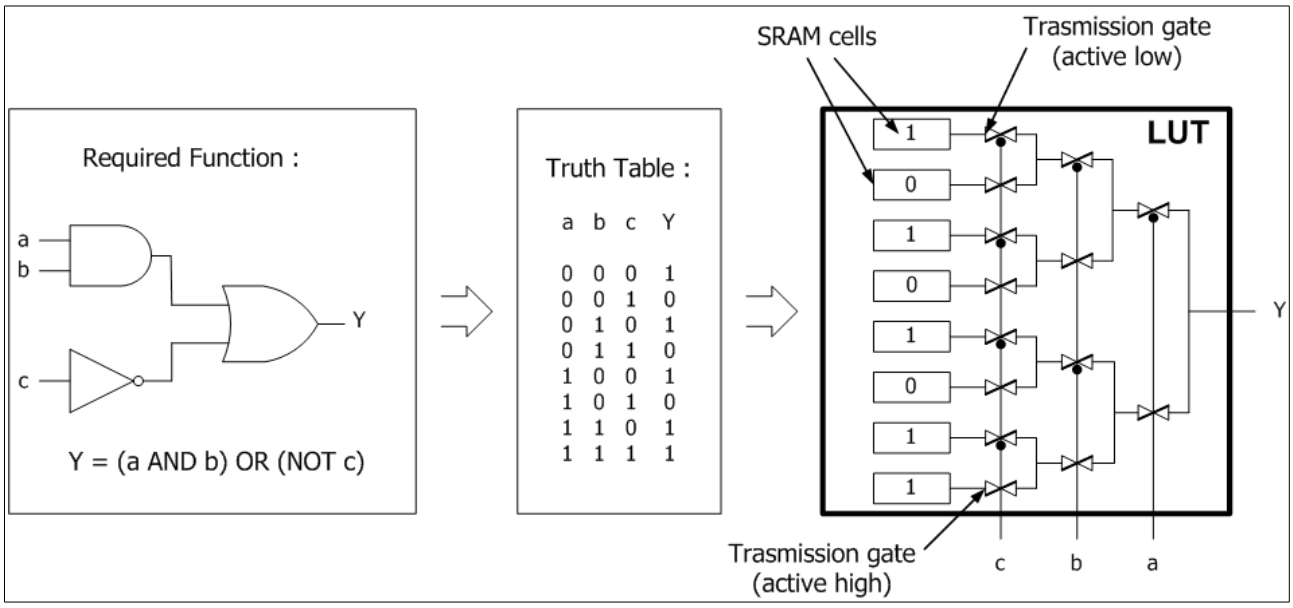

Figure 5 : The implementation of a logic function using a 3-input LUT

If a *transmission gate* is enabled (active), it passes the signal seen on its input through to its output. If the gate is disabled, its output is electrically disconnected from the wire it is driving. The transmission gate symbols shown with a small circle (called a “bobble” or a “bubble”) indicate that these gates will be activated by a logic 0 on their control input. By comparison, symbols without bobbles indicate that these gates will be activated by a logic 1. Based on this understanding, it’s easy to see how different input combinations can be used to select the contents of the various SRAM cells.

A small FPGA that has been programmed to implement a circuit is depicted in **Figure 6**. The FPGA has two-input LUTs, and there are four wires in each routing channel. The figure shows the programmed states of both the logic blocks and wiring switches (e.g. SRAM switches – **Figure 3**) in a section of the FPGA [2]. Programmable wiring switches are indicated by an X. Each switch shown in magenta is turned on and makes a connection between a horizontal and vertical wire. The switches shown in black are turned off. The truth tables programmed into the logic blocks in the top row of the FPGA correspond to the functions $f1 = x1x2$ and $f2 = \overline{x2}x3$. The logic block in the bottom right of the figure is programmed to produce $f = f1 + f2 = x1x2 + \overline{x2}x3$.

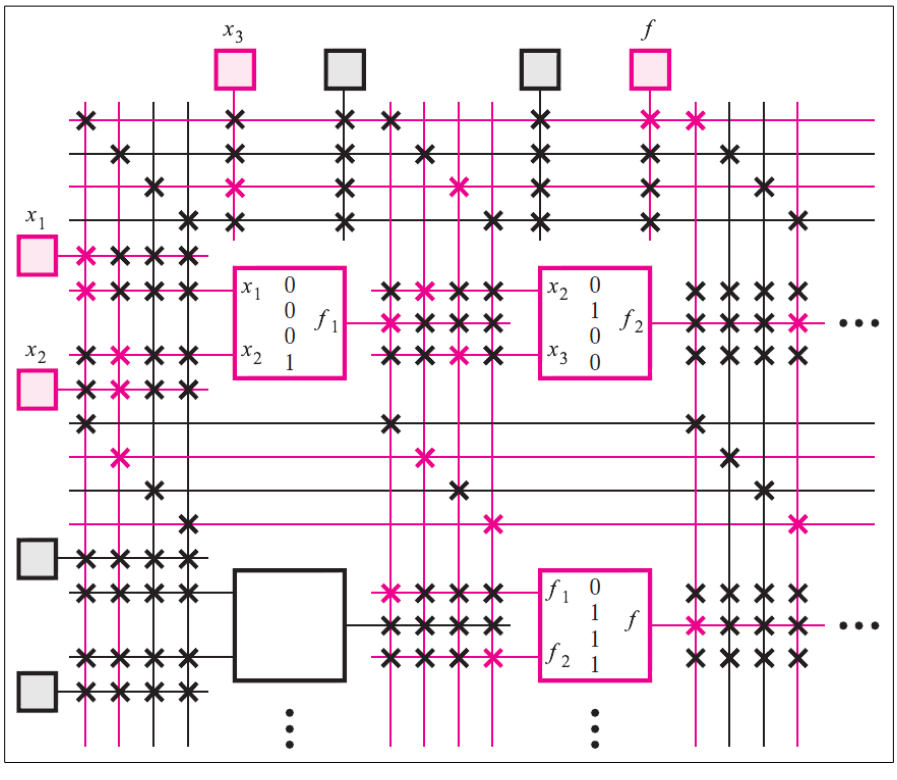

Figure 6 : A section of a programmed FPGA

Recent SRAM-based FPGA devices

CLBs, Slices and LUTs

The configurable logic blocks are the main logic resources for implementing sequential as well as combinatorial (or combinational) circuits. Depending on the FPGA family (e.g. Virtex-5, Virtex-6, Spartan-6) each CLB consists of a number of Slices and each Slice (again, depending on the FPGA family) contains a number of LUTs, storage elements, multiplexers and arithmetic logic. A Virtex-5 FPGA Slice contains four 6-input LUTs and four storage elements while a Virtex-6 FPGA Slice contains four 6-input LUTs and eight storage elements. A Spartan-6 CLB consists of two Slices. Each CLB element is connected to a switch matrix for access to the general-routing resources. **Figure 7** shows the arrangement of Slices within the Spartan-6 CLB [5].

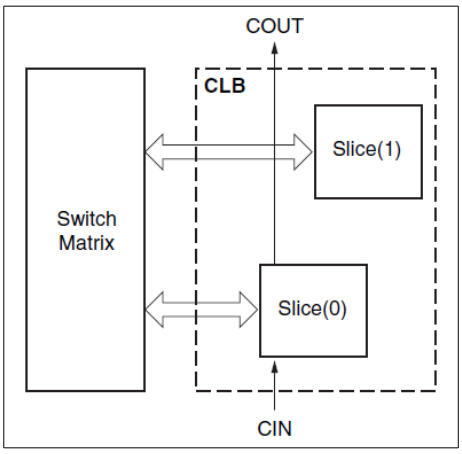

Figure 7 : The arrangement of Slices within the Spartan-6 CLB

Every Spartan-6 Slice contains four 6-input LUTs and eight storage elements. These elements are used by all Slices to provide logic and ROM functions. There are three types of CLB Slices in the Spartan-6 architecture : SLICEM, SLICEL, and SLICEX. SLICEX is the basic Slice. SLICELs also contain an arithmetic carry structure (shaded with purple in **Figure 8**), that can be concatenated vertically up through the Slice column, and wide-function multiplexers. This special carry logic and dedicated routing boosts the performance of logical functions such as counters and arithmetic functions such as adders. The SLICEMs contain the carry structure and multiplexers, and add the ability to use the LUTs as 64-bit distributed RAM and as variable-length shift registers (maximum 32-bit). **Table 1** shows the features of Spartan-6 Slices [5].

Feature	SLICEX	SLICEL	SLICEM
6-Input LUTs	√	√	√
8 Flip-flops	√	√	√
Wide Multiplexers		√	√
Carry Logic		√	√
Distributed RAM			√
Shift Registers			√

Table 1 : The features of Spartan-6 Slices

Each Spartan-6 column of CLBs contain two Slice columns (**Figure 7**). One column is a SLICEX column, the other column alternates between SLICEL and SLICEM. Thus, approximately 50% of the available Slices are of type SLICEX, while 25% each are SLICEL and SLICEM [5]. SLICEM represents a superset of elements and connections found in all Slices. **Figure 8** shows a Spartan-6 SLICEM.

The function generators in Spartan-6 FPGAs are implemented as six-input LUTs. There are six independent inputs (A inputs – *A1* to *A6*) and two independent outputs (*O5* and *O6*) for each of the four function generators in a Slice (A, B, C, and D – **Figure 8**). The function generators can implement any arbitrarily defined six-input boolean function. Each function generator can also implement two arbitrarily defined five-input boolean functions, as long as these two functions share common inputs. Only the *O6* output of the function generator is used when a six-input function is implemented. Both *O5* and *O6* are used for each of the five-input function generators

implemented. In this case, *A6* is driven High. The propagation delay through a LUT is independent of the function implemented, or whether one six-input or two five-input generators are implemented [5].

In addition to the basic LUTs, SLICEL and SLICEM contain three multiplexers (*F7AMUX*, *F7BMUX*, and *F8MUX*). These multiplexers are used to combine up to four function generators to provide any function of seven or eight inputs in a Slice. *F7AMUX* and *F7BMUX* are used to generate seven input functions from Slice A and B, or C and D, while *F8MUX* is used to combine all Slices to generate eight input functions. Functions with more than eight inputs can be implemented using multiple Slices. There are no direct connections between Slices to form function generators greater than eight inputs within a CLB or between Slices, but CLB outputs can be routed through the switch matrix and directly back into the CLB inputs (**Figure 7**) [5].

Virtex-5 and Virtex-6 FPGAs follow a similar to Spartan-6 Slice configuration, using SLICEL and SLICEM types of Slices, as these are defined in each FPGA family. **Tables 2** and **3** show the Virtex-6 and Spartan-6 FPGA Logic Resources respectively [5], [6].

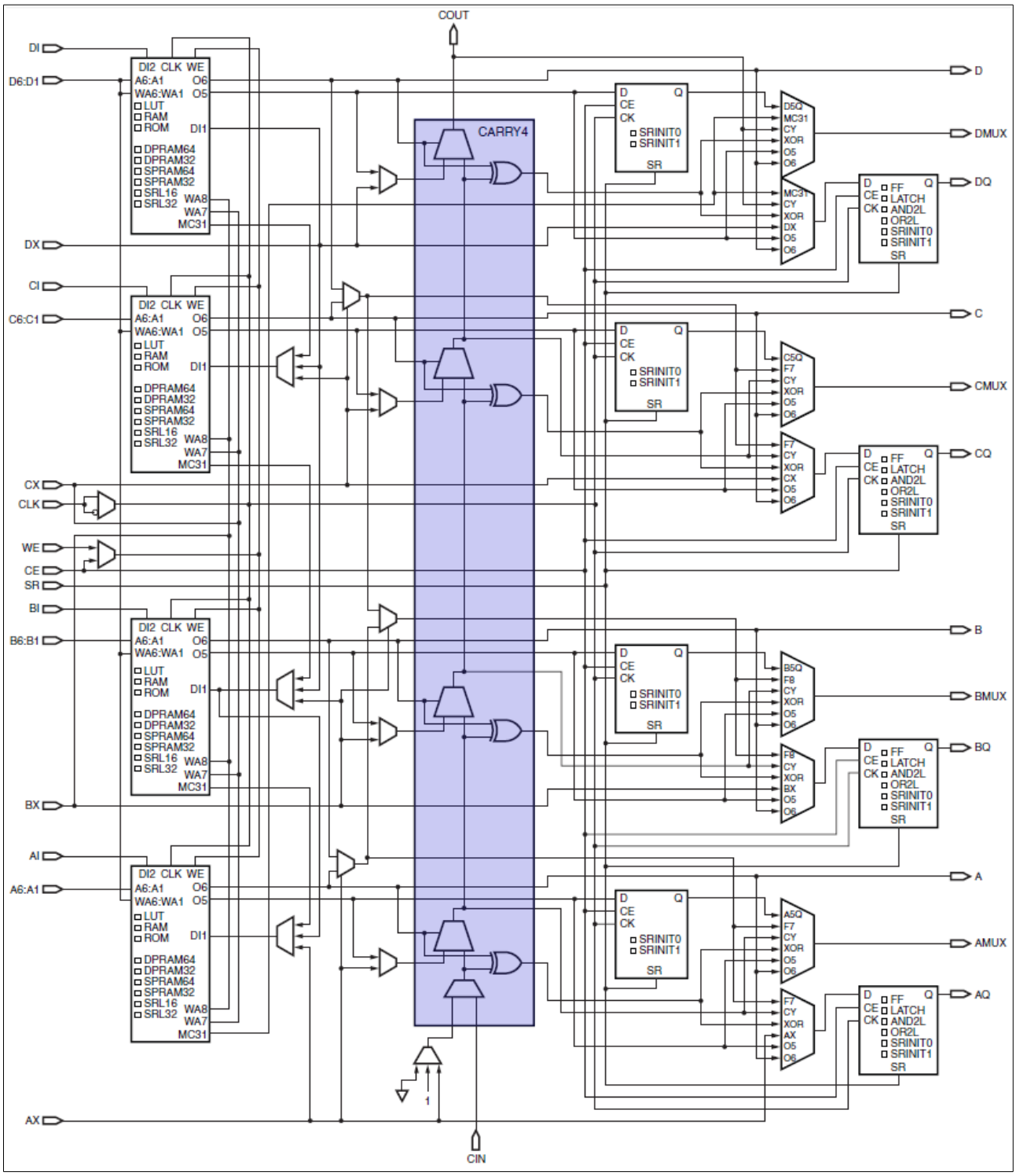

Figure 8 : A Spartan-6 SLICEM

Device	Total Slices	SLICELs	SLICEMs	Number of 6-Input LUTs	Maximum Distributed RAM (Kb)	Shift Register (Kb)	Number of Flip-Flops
XC6VLX75T	11,640	7,460	4,180	46,560	1,045	522.5	93,120
XC6VLX130T	20,000	13,040	6,960	80,000	1,740	870	160,000
XC6VLX195T	31,200	19,040	12,160	124,800	3,140	1,570	249,600
XC6VLX240T	37,680	23,080	14,600	150,720	3,770	1,885	301,440
XC6VLX365T	56,880	40,360	16,520	227,520	4,130	2,065	455,040
XC6VLX550T	85,920	61,120	24,800	343,680	6,200	3,100	687,360
XC6VLX760	118,560	85,440	33,120	474,240	8,280	4,140	948,480
XC6VSX315T	49,200	28,840	20,360	196,800	5,090	2,545	393,600
XC6VSX475T	74,400	48,840	30,560	297,600	7,640	3,820	595,200
XC6VHX250T	39,360	27,200	12,160	157,440	3,040	1,520	314,880
XC6VHX255T	39,600	27,400	12,200	158,400	3,050	1,525	316,800
XC6VHX380T	59,760	41,520	18,240	239,040	4,570	2,285	478,080
XC6VHX565T	88,560	63,080	25,480	354,240	6,370	3,185	708,480

Table 2 : The Virtex-6 FPGA Logic Resources

Device	Total Slices	SLICEMs	SLICELs	SLICEXs	Number of 6-Input LUTs	Maximum Distributed RAM (Kb)	Shift Registers (Kb)	Number of Flip-Flops
XC6SLX4	600	300	0	300	2,400	75	38	4,800
XC6SLX9	1,430	360	355	715	5,720	90	45	11,440
XC6SLX16	2,278	544	595	1,139	9,112	136	68	18,224
XC6SLX25	3,758	916	963	1,879	15,032	229	115	30,064
XC6SLX45	6,822	1,602	1,809	3,411	27,288	401	200	54,576
XC6SLX75	11,662	2,768	3,063	5,831	46,648	692	346	93,296
XC6SLX100	15,822	3,904	4,007	7,911	63,288	976	488	126,576
XC6SLX150	23,038	5,420	6,099	11,519	92,152	1,355	678	184,304
XC6SLX25T	3,758	916	963	1,879	15,032	229	115	30,064
XC6SLX45T	6,822	1,602	1,809	3,411	27,288	401	200	54,576
XC6SLX75T	11,662	2,768	3,063	5,831	46,648	692	346	93,296
XC6SLX100T	15,822	3,904	4,007	7,911	63,288	976	488	126,576
XC6SLX150T	23,038	5,420	6,099	11,519	92,152	1,355	678	184,304

Table 3 : The Spartan-6 FPGA Logic Resources

Clocking Resources

All of the synchronous elements inside an FPGA — for example, the registers configured to act as

flip-flops inside the CLBs — need to be driven by a clock signal. Such a clock signal typically originates in the outside world, comes into the FPGA via a special clock input pin, and is then routed through the device and connected to the appropriate registers [1].

Both global and regional clocking techniques are used to disperse clocking across the FPGA fabric. Global clocking includes the implementation of global steering logic and buffers for distributing the clock within the FPGA. Global clocking typically begins in the middle of the device and then branches into smaller regions forming a clock tree [1], [7]. This structure is used to ensure that all of the flip-flops see their versions of the clock signal as close together as possible. If the clock were distributed as a single long track driving all of the flip-flops one after another, then the flip-flop closest to the clock pin would see the clock signal much sooner than the one at the end of the chain. This is referred to as skew, and it can cause all sorts of problems (even when using a clock tree, there will be a certain amount of skew between the registers on a branch and also between branches). The clock tree is implemented using special tracks and is separate from the general-purpose programmable interconnect [1]. FPGA devices are typically divided into four or more clocking regions. Regional clocking can also be provided to individual FPGA regions. **Figure 9** shows a simplified clocking implementation [7].

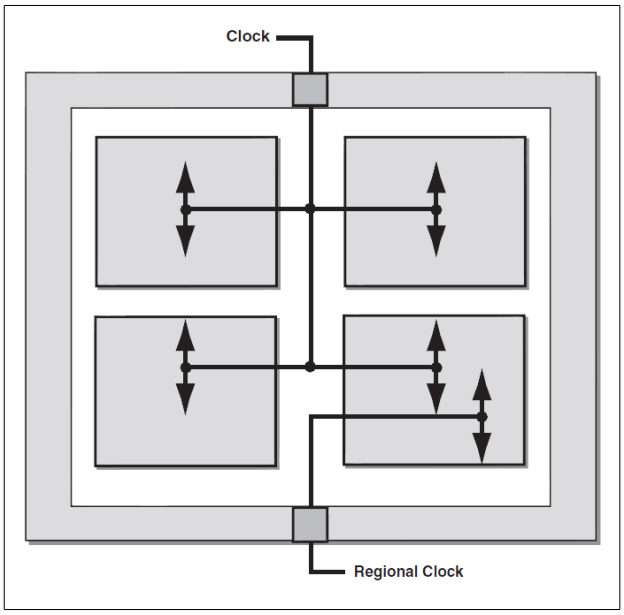

Figure 9 : A simplified clocking implementation

Differential clocking is typically implemented for global signal distribution and is essential for high-speed memory I/O interface (e.g. DDR2 interface). The desirable characteristics of differential clocking include faster edge rates, improved noise immunity, and inherently balanced duty cycles. Differential clocking also supports higher frequency operation and more reliable data transmission [7].

Instead of configuring a clock pin to connect directly into an internal clock tree, that pin can be used to drive a special hard-wired function (block) called a clock manager that generates a number of daughter clocks. These daughter clocks may be used to drive internal clock trees or external output pins that can be used to provide clocking services to other devices on the host circuit board. Each family of FPGAs has its own type of clock manager [1]. In Virtex-5 and Spartan-6 FPGAs

a Clock Management Tile (CMT) contains two Digital Clock Manager (DCM) blocks (self-calibrating, fully digital), and one phase-locked loop (PLL) block (self-calibrating, analog) for clock distribution delay compensation, clock multiplication/division, coarse-/fine-grained clock phase shifting, and input clock jitter filtering [8], [9]. A DCM is based on a delay lock loop (DLL). In a Virtex-6 FPGA a CMT contains two mixed-mode clock managers (MMCMs), which are PLL based [10]. Virtex-5 and Spartan-6 FPGA families support up to 6 CMTs in one device. Virtex-6 FPGA family supports up to 9 CMTs.

PLLs generate the desired clock phase or frequency output by making adjustments to a voltage-controlled oscillator (VCO). PLLs are inherently analog circuits and therefore they perform better when supplied with “clean” power and ground. DLLs access signals from a calibrated tapped delay line circuit internal to the FPGA to produce the desired clock phase or frequency. DLLs are digital circuits. **Figure 10** shows a graphical representation of a PLL and a DLL.

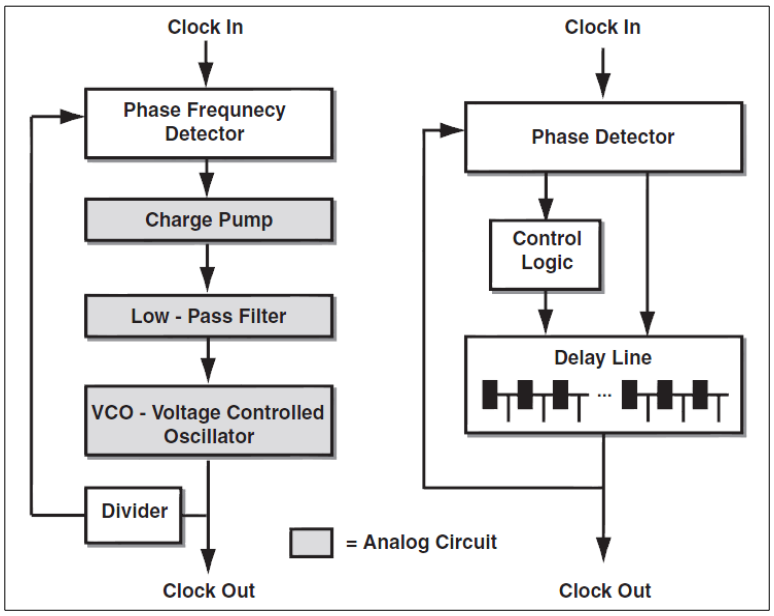

Figure 10 : A graphical representation of a PLL (left) and a DLL (right)

In today's FPGAs, PLLs support up to six independent clock outputs. Designs using several different clock outputs should use PLLs. When the application requires a fine phase shift or a dynamic variable phase shift, a DCM could be a better solution.

I/Os and I/O blocks

Today’s FPGA packages have 1000 or more pins, which are arranged as an array across the base of the package. Similarly, when it comes to the silicon chip inside the package, flip-chip packaging strategies allow the power, ground, clock, and I/O pins to be presented across the surface of the chip [1].

I/O blocks (IOBs) provide the interface between package pins and the internal configurable logic. Most popular and leading-edge I/O standards are supported by programmable I/O blocks. A “standard” refers to electrical aspects of the signals, such as their logic 0 and logic 1 voltage levels.

The IOBs can be connected to very flexible special logic for enhanced source-synchronous interfacing. Source-synchronous optimizations include per-bit deskew (on both input and output signals), data serializers/deserializers, clock dividers, and dedicated I/O and local clocking resources [7].

I/O signals are organized into a number of banks. Each bank can be configured individually to support a particular I/O standard (or a particular combination of I/O standards). Thus, in addition to allowing the FPGA to work with devices using multiple I/O standards, this allows the FPGA to actually be used to interface between different I/O standards (and also to translate between different protocols that may be based on particular electrical standards) [1]. **Figure 11** shows the I/O banks of Virtex-5 xc5vfx70tff1136-1 FPGA.

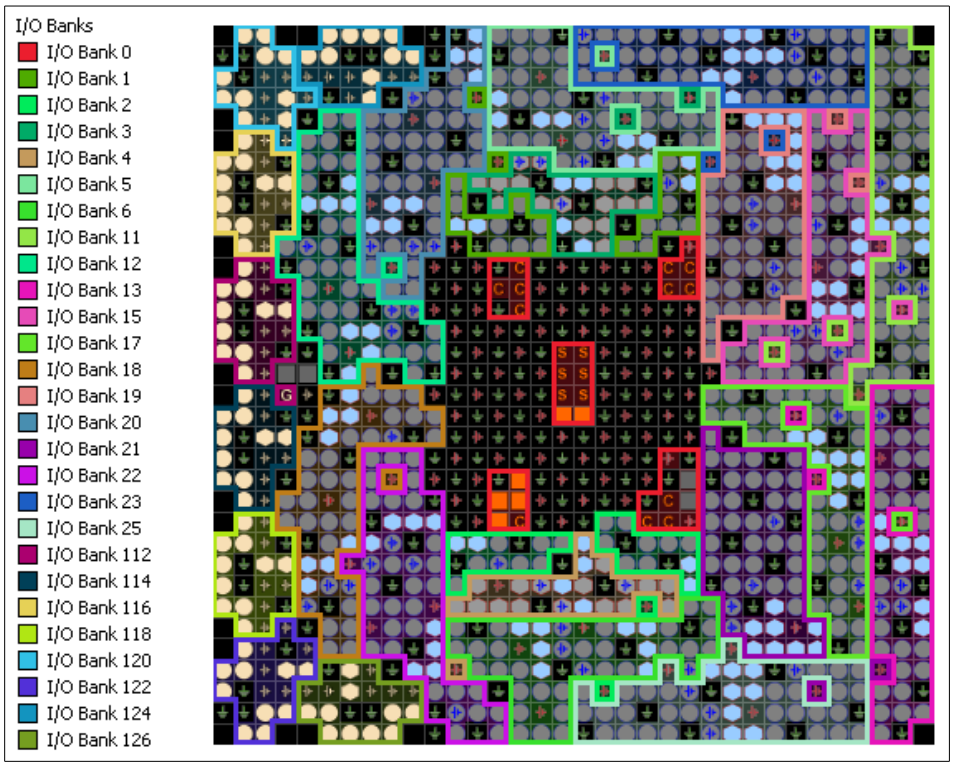

Figure 11 : The I/O banks of Virtex-5 xc5vfx70tff1136-1 FPGA

The signals used to connect devices on today's circuit board often have fast edge rates (this refers to the time it takes the signal to switch between one logic value and another). In order to prevent signals reflecting back (bouncing around), it is necessary to apply appropriate terminating resistors to the FPGA's input or output pins. In the past, these resistors were applied as discrete components that were attached to the circuit board outside the FPGA. However, this technique became increasingly problematic as the number of pins started to increase and their pitch (the distance between them) shrank. For this reason, today's FPGAs allow the use of internal terminating resistors whose values can be configured by the user to accommodate different circuit board environments and I/O standards [1] (e.g. the Digitally Controlled Impedance (DCI) I/O feature, found in some FPGAs, can be configured to provide on-chip termination for each single-ended I/O standard and some differential I/O standards).

Embedded blocks

Some functions are inherently slow and occupy a significant portion of the FPGA if they are implemented by connecting a large number of CLBs together. Since these functions are required by a lot of applications, many FPGAs incorporate special hard-wired blocks like :

- *Block RAM* (BRAM). Depending on the device, such a RAM might be able to hold anywhere from a few thousand to tens of thousands of bits [1]. Furthermore, a device might contain anywhere from tens to hundreds of these RAM blocks, thereby providing a total storage capacity of a few hundred thousand bits all the way up to several million bits. Each block of RAM can be used independently, or multiple blocks can be combined together to implement larger blocks. These blocks can be used for a variety of purposes, such as implementing standard single- or dual-port RAMs, first-in first-out (FIFO) functions, state machines, and so forth.

- *Digital Signal Processing blocks* (DSP blocks (or DSP Slices)). To allow more complex designs, which may consist of either digital signal processing or just some assortment of multiplication, addition, and subtraction, Digital Signal Processing blocks have been added to many FPGA devices [11]. As with the Block RAM, it is possible to implement these components within the configurable logic, yet it is more efficient in terms of performance, and power consumption to embed multiple of these components within the FPGA fabric. At a high level, the DSP blocks consist of a multiplier, accumulator, adder, and bitwise logical operations (such as AND, OR, NOT, and NAND). It is possible to combine DSP blocks to perform larger operations, such as single and double precision floating point addition, subtraction, multiplication, division, and square root. The number of DSP blocks is device dependent; however, they are typically located near the BRAMs, which is useful when implementing processing requiring input and/or output buffers.

- *Multi-Gigabit Transceivers* (MGTs). Over the last 20 years, digital I/O standards have varied between serial and parallel interfaces. Serial interfaces time-multiplex the bits of a data word over fewer conductors while parallel interfaces signal all the bits simultaneously. While the parallel approach has the apparent advantage of being faster (data are being transmitted in parallel), this is not always the case in the presence of noise [11]. For this reason, today's high-end FPGAs include special hard-wired gigabit transceiver blocks. These blocks use one pair of differential signals (which means a pair of signals that always carry opposite logical values) to transmit data and another pair to receive data [1]. These transceivers operate at incredibly high speeds, from 100 Mb/s to 11.0 Gb/s, which means that they can be configured to support a number of different standards, including Fiber Channel, 10G Fiber Channel, Gigabit Ethernet, Infiniband and PCI Express. As with the other aforementioned FPGA embedded blocks, the transceivers can be configured to work together. For example, two transceivers can be used to effectively double the bandwidth. This is called channel bonding (multilane sand trunking are common synonyms) [11].

- *Hard/Soft processor cores*. For many designs requiring a processor, often choosing an FPGA device with an embedded processor can simplify the design process greatly while reducing resource usage and power consumption [11]. A variety of interfaces exist to connect the embedded processors to the FPGA programmable logic to allow interaction with custom

hardware cores. Not all FPGAs come with a processor embedded into the FPGA fabric. For these devices the processor must be implemented within the FPGA fabric as a soft processor core. Soft cores are simpler (more primitive) and slower than their hard-core counterparts. However, they have the advantage that you only need to implement a core if you need it and also that you can instantiate as many cores as you require until you run out of resources in the form of CLBs [1].

Some FPGAs may also contain Integrated Endpoint blocks for PCI Express designs and 10/100/1000 Mb/s Ethernet media-access control blocks for Ethernet capability. **Figure 12** shows the arrangement of embedded blocks placed throughout the FPGA. **Table 4** shows the Virtex-6 FPGA feature summary.

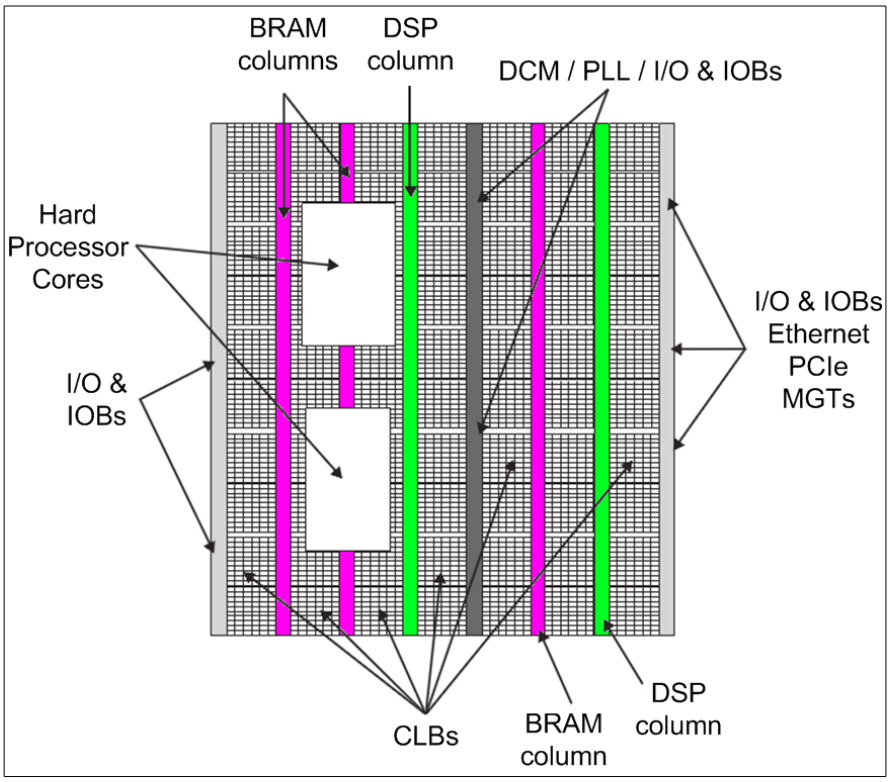

Figure 12 : The arrangement of the embedded blocks

Device	Logic Cells	Configurable Logic Blocks (CLBs)		DSP48E1 Slices[2]	Block RAM Blocks			MMCMs[4]	Interface Blocks for PCI Express	Ethernet MACs[5]	Maximum Transceivers		Total I/O Banks[6]	Max User I/O[7]
		Slices[1]	Max Distributed RAM (Kb)		18 Kb[3]	36 Kb	Max (Kb)				GTX	GTH		
XC6VLX75T	74,496	11,640	1,045	288	312	156	5,616	6	1	4	12	0	9	360
XC6VLX130T	128,000	20,000	1,740	480	528	264	9,504	10	2	4	20	0	15	600
XC6VLX195T	199,680	31,200	3,040	640	688	344	12,384	10	2	4	20	0	15	600
XC6VLX240T	241,152	37,680	3,650	768	832	416	14,976	12	2	4	24	0	18	720
XC6VLX365T	364,032	56,880	4,130	576	832	416	14,976	12	2	4	24	0	18	720
XC6VLX550T	549,888	85,920	6,200	864	1,264	632	22,752	18	2	4	36	0	30	1200
XC6VLX760	758,784	118,560	8,280	864	1,440	720	25,920	18	0	0	0	0	30	1200
XC6VSX315T	314,880	49,200	5,090	1,344	1,408	704	25,344	12	2	4	24	0	18	720
XC6VSX475T	476,160	74,400	7,640	2,016	2,128	1,064	38,304	18	2	4	36	0	21	840
XC6VHX250T	251,904	39,360	3,040	576	1,008	504	18,144	12	4	4	48	0	8	320
XC6VHX255T	253,440	39,600	3,050	576	1,032	516	18,576	12	2	2	24	24	12	480
XC6VHX380T	382,464	59,760	4,570	864	1,536	768	27,648	18	4	4	48	24	18	720
XC6VHX565T	566,784	88,560	6,370	864	1,824	912	32,832	18	4	4	48	24	18	720

Notes:

1. Each Virtex-6 FPGA slice contains four LUTs and eight flip-flops, only some slices can use their LUTs as distributed RAM or SRLs.
2. Each DSP48E1 slice contains a 25 x 18 multiplier, an adder, and an accumulator.
3. Block RAMs are fundamentally 36 Kbits in size. Each block can also be used as two independent 18 Kb blocks.
4. Each CMT contains two mixed-mode clock managers (MMCM).
5. This table lists individual Ethernet MACs per device.
6. Does not include configuration Bank 0.
7. This number does not include GTX or GTH transceivers.

Table 4 : The Virtex-6 FPGA feature summary

Interconnect Resources

Interconnect is the programmable network of signal pathways between the inputs and outputs of functional elements within the FPGA, such as IOBs, CLBs, DSP blocks, and block RAM. Interconnect, also called routing, is segmented for optimal connectivity [5]. CLBs are arranged in a regular array inside the FPGA. Each connects to a switch matrix for access to the general-routing resources, which run vertically and horizontally between the CLB rows and columns. A similar switch matrix connects other resources, such as the DSP blocks and block RAM resources. **Figure 13** shows the Spartan-6 CLB array and interconnect channels.

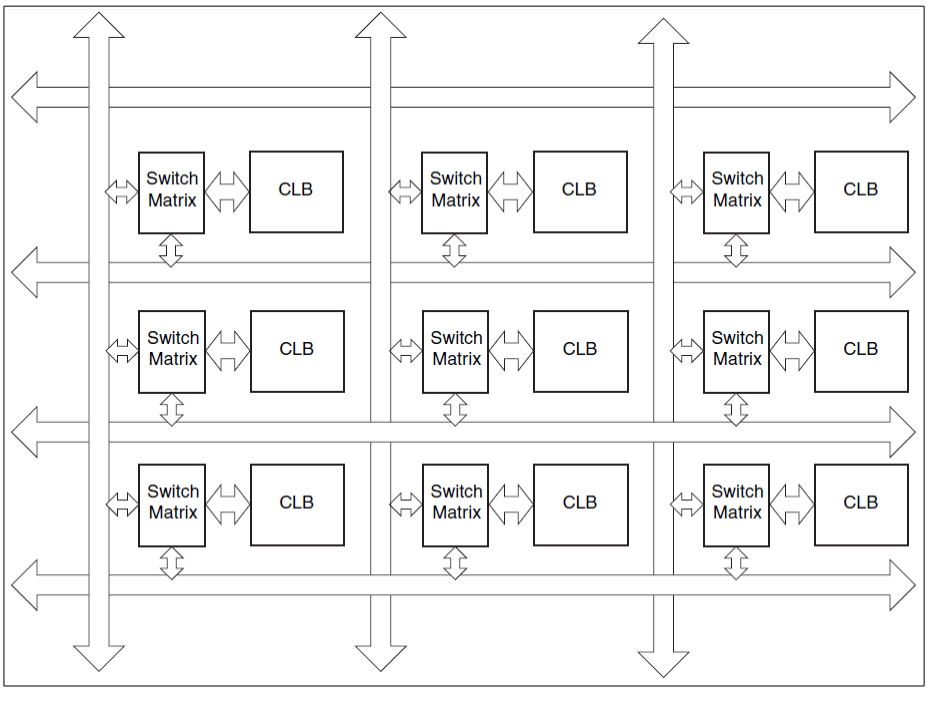

Figure 13 : The Spartan-6 CLB array and interconnect channels

The various types of routing in the FPGA architecture are primarily defined by their length. Longer routing elements are faster for longer distances [5]. **Figure 14** shows the interconnect types for the Spartan-6 architecture.

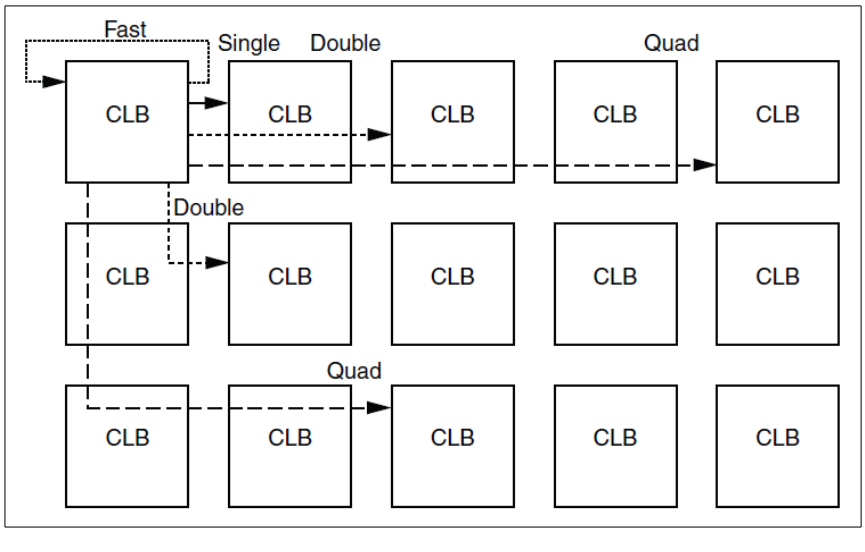

Figure 14 : The interconnect types for the Spartan-6 architecture

Fast connects route block outputs back to block inputs. Along with the larger size of the CLB, fast connects provide higher performance for simpler functions. *Singles* route signals to neighboring tiles, both vertically and horizontally. *Doubles* connect to every other tile, both horizontally and vertically, in all four directions, and to the diagonally adjacent tiles. *Quads* connect to one out of every four tiles, horizontally and vertically, and diagonally to tiles two rows and two columns distant. Quad lines provide more flexibility than the single-channel long lines of earlier generations.

SRAM-based FPGA configuration

The end result of all FPGA design techniques is a configuration file (sometimes called a bit file), which contains the information that will be uploaded into the FPGA in order to program it to perform a specific function [1].

In the case of SRAM-based FPGAs, the configuration file contains a mixture of configuration data (bits that are used to define the state of programmable logic elements directly) and configuration commands (instructions that tell the device what to do with the configuration data). When the configuration file is in the process of being loaded into the device, the information being transferred is referred to as the configuration bitstream [1].

Because FPGA configuration data is stored in SRAM configuration latches, the FPGA must be reconfigured after it is powered down. The bitstream is loaded each time into the device through special configuration pins. FPGAs can load themselves from an external nonvolatile memory device or they can be configured by an external smart source, such as a microprocessor, DSP processor, microcontroller, PC, or board tester. In any case, there are two general configuration datapaths. The first is the serial datapath that is used to minimize the device pin requirements. The second datapath is the 8-bit, 16-bit, or 32-bit wide datapath that is used for higher performance or access (or link) to industry-standard interfaces, ideal for external data sources like processors, or x8- or x16-parallel flash memory [12].

The FPGA and the external nonvolatile configuration PROMs are usually parts of a board. These devices are in-system programmable (ISP) – are capable of being programmed while remaining resident in a higher-level system. The FPGA designer can connect the board with the PC using a configuration cable and program directly the FPGA or program the configuration PROM and set the FPGA to be programmed from that PROM at board power-up. **Figure 15** shows the FPGA programming setup.

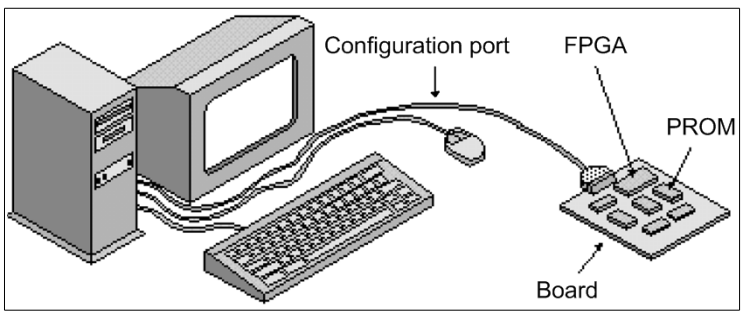

Figure 15 : The FPGA programming setup

References

[1] Clive “Max” Maxfield, The Design Warrior’s Guide to FPGAs, Newnes - Elsevier, 2004

[2] Stephen Brown and Zvonko Vranesic, Fundamentals of Digital Logic with VHDL Design, 3rd Edition, McGraw-Hill, 2009

[3] Bob Zeidman, Designing with FPGAs and CPLDs, CMP Books, 2002

[4] Zoran Salcic and Asim Smailagic, Digital Systems Design and Prototyping Using Field Programmable Logic and Hardware Description Languages, 2nd Edition, Kluwer Academic Publishers, 2002

[5] Xilinx Corporation, Spartan-6 FPGA Configurable Logic Block User Guide, 2010, http://www.xilinx.com

[6] Xilinx Corporation, Virtex-6 FPGA Configurable Logic Block User Guide, 2009, http://www.xilinx.com

[7] R.C. Cofer and Benjamin F. Harding, Rapid System Prototyping with FPGAs, Elsevier, 2006

[8] Xilinx Corporation, Virtex-5 Family Overview, 2009, http://www.xilinx.com

[9] Xilinx Corporation, Spartan-6 Family Overview, 2011, http://www.xilinx.com

[10] Xilinx Corporation, Virtex-6 Family Overview, 2009, http://www.xilinx.com

[11] Ron Sass and Andrew G. Schmidt, Embedded Systems Design with Platform FPGAs Principles and Practices, Elsevier, 2010

[12] Xilinx Corporation, Virtex-6 FPGA Configuration User Guide, 2011, http://www.xilinx.com