FPGA technology in detail

FPGAs are chips, which are programmed by the customer to perform the desired functionality. The chips may be programmed either

- *once:* Antifuse technology, e.g. devices manufactured by Quicklogic
- *several times:* Flash based, e.g. devices manufactures by Actel
- *dynamically:* SRAM based, e.g. devices manufactured by Actel, Altera, Atmel, Cypress, Lucent, Xilinx

Each technology has its own advantages, which shall be discussed only very briefly:

- Antifuse FPGAs:
  - devices are configured by burning a set of fuses. Once the chip is configured, it cannot be altered any more.
  - bug fixes and updates possible for new PCBs, but hardly for already manufactured boards.
  - ASIC replacement for small volumes.
- Flash FPGAs
  - devices may be re-programmed several thousand times and are non-volatile, i.e. keep their configuration after power-off
  - with only marginal additional effort, the chips may be updated in the field
  - expensive
  - re-configuration takes several seconds
- SRAM FPGAs
  - currently the dominating technology
  - unlimited re-programming
  - additional circuitry is required to load the configuration into the FPGA after power-on
  - re-configuration is very fast, some devices allow even partial re-configuration during operation
  - allows new approaches and applications- buzzword *reconfigurable computing*, e.g. a circuit, that searches for a specific DNA pattern, or a mobile phone that downloads the latest protocol update

General Overview

There are several families of FPGAs available from different semiconductor companies. These device families slightly differ in their architecture and feature set, however most of them follow a common approach: A regular, flexible, programmable architecture of *Configurable Logic Blocks* (CLBs), surrounded by a perimeter of programmable *Input/Output Blocks* (IOBs). These functional elements are interconnected by a powerful hierarchy of versatile *routing channels*.

The following paragraphs describe the architecture implemented by *Xilinx Spartan-II* FPGAs, a device family launched in mid 2000, which is typically used in high-volume applications where the versatility of a fast programmable solution adds benefits.

The user-programmable gate array, shown in Figure 15, is composed of five major configurable elements:

- *IOBs* provide the interface between the package pins and the internal logic
- *CLBs* provide the functional elements for constructing most logic
- Dedicated *BlockRAM* memories of 4096 bits each
- Clock *DLLs* for clock-distribution delay compensation and clock domain control
- Versatile multi-level interconnect structure

As can be seen in Figure 15, the CLBs form the central logic structure with easy access to all support and routing structures. The IOBs are located around all the logic and memory elements for easy and quick routing of signals on and off the chip.

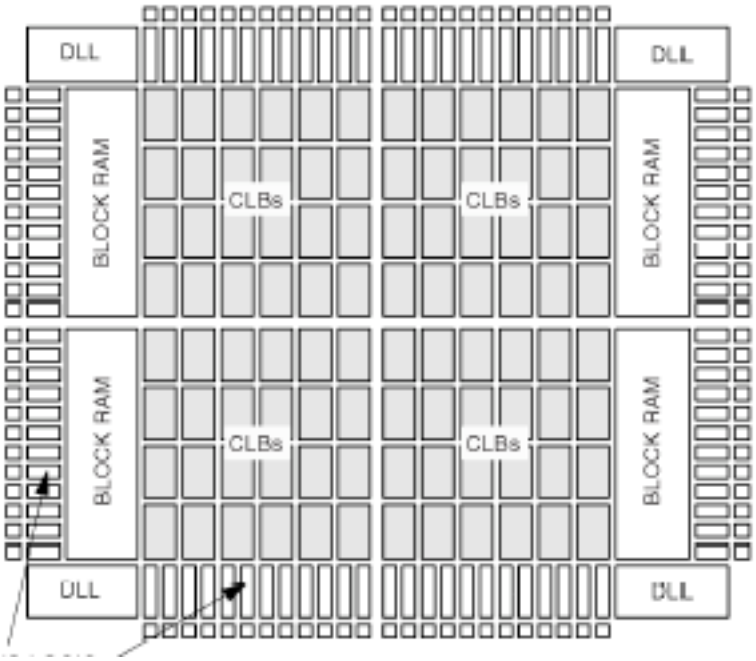

Figure 15: Basic Spartan-II Block Diagram

Values stored in static memory cells control all the configurable logic elements and interconnect resources. These values load into the memory cells on power-up, and can reload if necessary to change the function of the device.

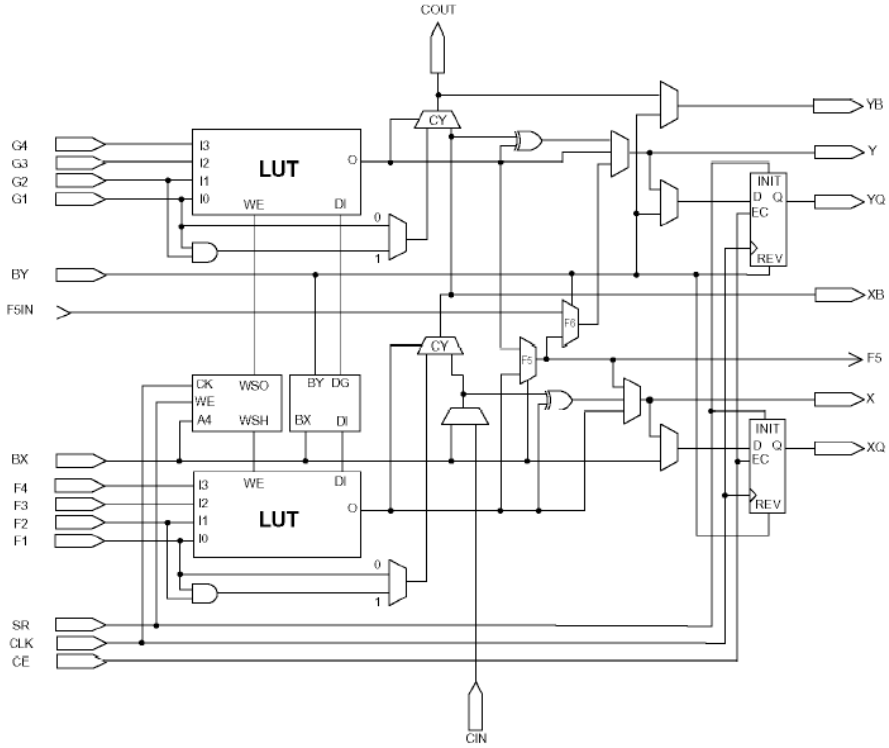

Figure 16: FPGA Slice

Configurable Logic Block

The basic building block of the CLBs is the logic cell (LC). An LC includes a 4-input function generator, carry logic, and a storage element. The output from the function generator in each LC drives both the CLB output and the D input of the flip-flop. Each CLB contains four LCs, organized in two similar slices; a single slice is shown in Figure 16.

In addition to the four basic LCs, the CLBs contains logic that combines function generators to provide functions of five or six inputs. Consequently, when estimating the number of system gates provided by a given device, each CLB counts as 4.5 LCs.

Look-Up Tables

The function generators are implemented as 4-input look-up tables (LUTs). In addition to operating as a function generator, each LUT can provide a 16x1-bit synchronous RAM. Furthermore, the two LUTs within a slice can be combined to create a 16x2-bit or 32x1-bit synchronous RAM, or a 16x1-bit dual-port synchronous RAM.

The LUT can also provide a 16-bit shift register that is ideal for capturing high-speed or burst-mode data. This mode can also be used to store data in applications such as Digital Signal Processing. See Figure 17 for details on slice configuration.

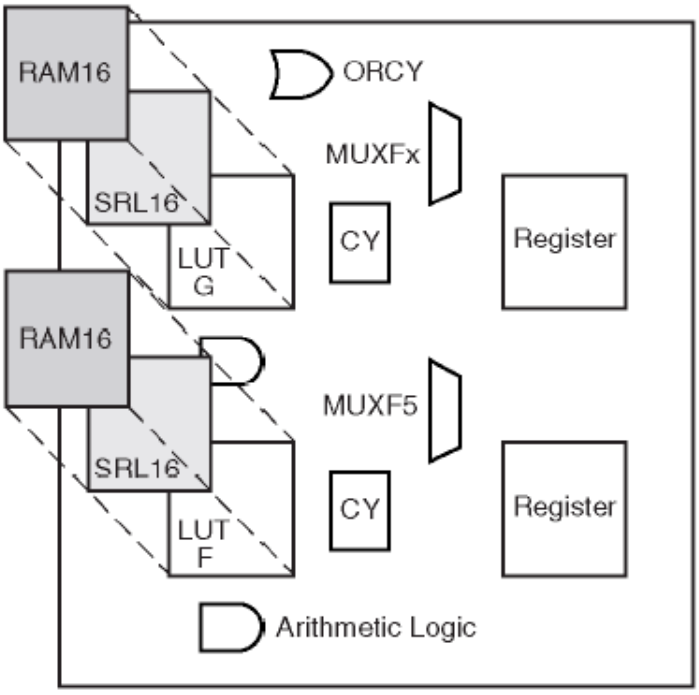

Figure 17: Slice Configuration

Storage Elements

The storage elements in the Spartan-II slice can be configured either as edge-triggered D-type flip-flops or as level-sensitive latches. The D inputs can be driven either by the function generators within the slice or directly from slice inputs, bypassing the function generators

In addition to Clock and Clock Enable signals, each slice has synchronous set and reset signals (SR and BY). SR forces a storage element into the initialization state specified for it in the configuration. BY forces it into the opposite state. Alternatively, these signals may be configured to operate asynchronously.

All of the control signals are independently invertible, and are shared by the two flip-flops within the slice.

Additional Logic

The F5 multiplexer in each slice combines the function generator outputs. This combination provides either a function generator that can implement any 5-input function, a 4:1 multiplexer, or selected functions of up to nine inputs.

Similarly, the F6 multiplexer combines the outputs of all four function generators in the CLB by selecting one of the F5-multiplexer outputs. This permits the implementation of any 6-input function, an 8:1 multiplexer, or selected functions of up to 19 inputs. Usage of the F5 and F6 multiplexer is shown in Figure 18.

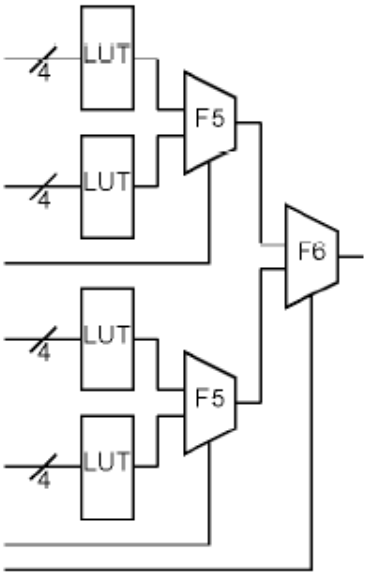

Figure 18: F5 and F6 Multiplexer

Each CLB has four direct feedthrough paths, one per LC. These paths provide extra data input lines or additional local routing that does not consume logic resources.

Arithmetic Logic

Dedicated carry logic provides fast arithmetic carry capability for high-speed arithmetic functions. The CLBs supports two separate carry chains, one per slice. The height of the carry chains is two bits per CLB.

The arithmetic logic includes an XOR gate that allows a 1-bit full adder to be implemented within an LC. In addition, a dedicated AND gate improves the efficiency of multiplier implementation. The dedicated carry path can also be used to cascade function generators for implementing wide logic functions.

BUFTs

Each CLB contains two 3-state drivers (BUFTs) that can drive on-chip busses (see Dedicated Routing for further details). Each Spartan-II BUFT has an independent 3-state control pin and an independent input pin.The 3-state drivers in conjunction with the on-chip busses may be used to implement wide multiplexers efficiently.

Input/Output Block

The IOB, as seen in Figure 19, features inputs and outputs that support a wide variety of I/O signaling standards. These high-speed inputs and outputs are capable of supporting various state of the art memory and bus interfaces. Table 1 lists several of the standards which are supported along with the required reference, output and termination voltages needed to meet the standard.

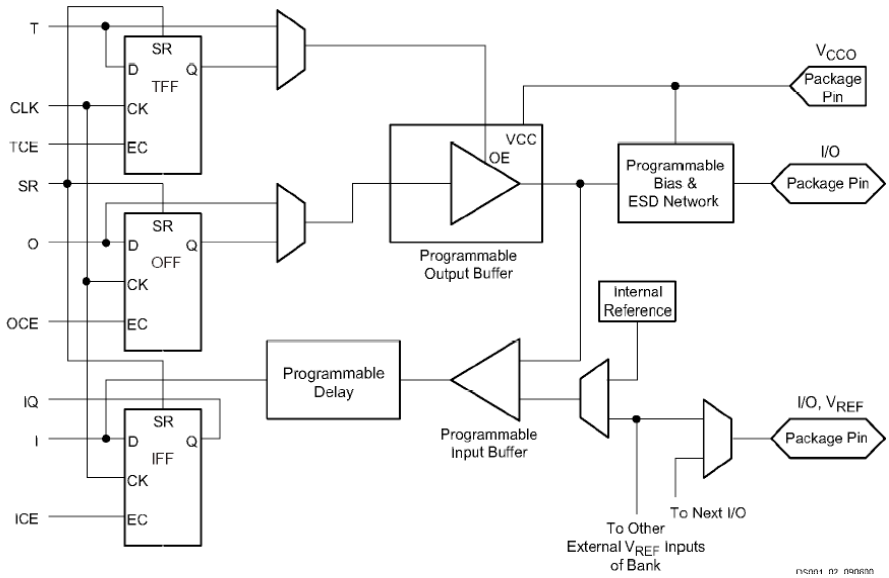

Figure 19: Input/Output Block (IOB)

The three IOB registers function either as edge-triggered D-type flip-flops or as level- sensitive latches. Each IOB has a clock signal (CLK) shared by the three registers and independent Clock Enable (CE) signals for each register.

In addition to the CLK and CE control signals, the three registers share a Set/Reset (SR). For each register, this signal can be independently configured as a synchronous Set, a synchronous Reset, an asynchronous Preset, or an asynchronous Clear.

I/O Standard	Input Reference Voltage (Vref)	Output Source Voltage (Vcco)	Board Termination Voltage (Vtt)
LVTTL (2-24 mA)	N/A	3.3	N/A
LVCMOS2	N/A	2.5	N/A
PCI (3V/5V, 33 MHz/66 MHz)	N/A	3.3	N/A
GTL	0.8	N/A	1.2
GTL+	1.0	N/A	1.5
HSTL Class I	0.75	1.5	0.75
HSTL Class III	0.9	1.5	1.5
HSTL Class IV	0.9	1.5	1.5
SSTL3 Class I and II	1.5	3.3	1.5
SSTL2 Class I and II	1.25	2.5	1.25
CTT	1.5	3.3	1.5
AGP-2X	1.32	3.3	N/A

Table 1: Standards Supported by IOB

Optional pull-up and pull-down resistors and an optional weak-keeper circuit are attached to each pad. Prior to configuration all outputs not involved in configuration are forced into their high-impedance state. The pull-down resistors and the weak-keeper circuits are inactive, but inputs may optionally be pulled up.

Input Path

A buffer In the IOB input path routes the input signal either directly to internal logic or through an optional input flip-flop.

An optional delay element at the D-input of this flip-flop eliminates pad-to-pad hold time. The delay is matched to the internal clock-distribution delay of the FPGA, and when used, assures that the pad-to-pad hold time is zero.

Each input buffer can be configured to conform to any of the low-voltage signaling standards supported. In some of these standards the input buffer utilizes a user-supplied threshold voltage, Vref. The need to supply Vref imposes constraints on which standards can be used in close proximity to each other.

Output Path

The output path includes a 3-state output buffer that drives the output signal onto the pad. The output signal can be routed to the buffer directly from the internal logic or through an optional IOB output flip-flop.

The 3-state control of the output can also be routed directly from the internal logic or through a flip-flip that provides synchronous enable and disable.

Each output driver can be individually programmed for a wide range of low-voltage signaling standards. Each output buffer can source up to 24 mA and sink up to 48 mA. Drive strength and slew rate controls minimize bus transients.

In most signaling standards, the output high voltage depends on an externally supplied Vcco voltage. The need to supply Vcco imposes constraints on which standards can be used in close proximity to each other.

An optional weak-keeper circuit is connected to each output. When selected, the circuit monitors the voltage on the pad and weakly drives the pin High or Low to match the input signal. If the pin is connected to a multiple-source signal, the weak keeper holds the signal in its last state if all drivers are disabled. Maintaining a valid logic level in this way helps eliminate bus chatter.

Programmable Routing Matrix

It is the longest delay path that limits the speed of any worst-case design. Consequently, the routing architecture and the place-and-route software have to be defined in a single optimization process. This joint optimization minimizes long-path delays, and consequently, yields the best system performance.

The joint optimization also reduces design compilation times because the architecture is software-friendly. Design cycles are correspondingly reduced due to shorter design iteration times.

Local Routing

The local routing resources, as shown in Figure 20, provide the following three types of connections:

- ◇ Interconnections among the LUTs, flip-flops, and General Routing Matrix (GRM)
- ◇ Internal CLB feedback paths that provide high-speed connections to LUTs within the same CLB, chaining them together with minimal routing delay
- ◇ Direct paths that provide high-speed connections between horizontally adjacent CLBs, eliminating the delay of the GRM.

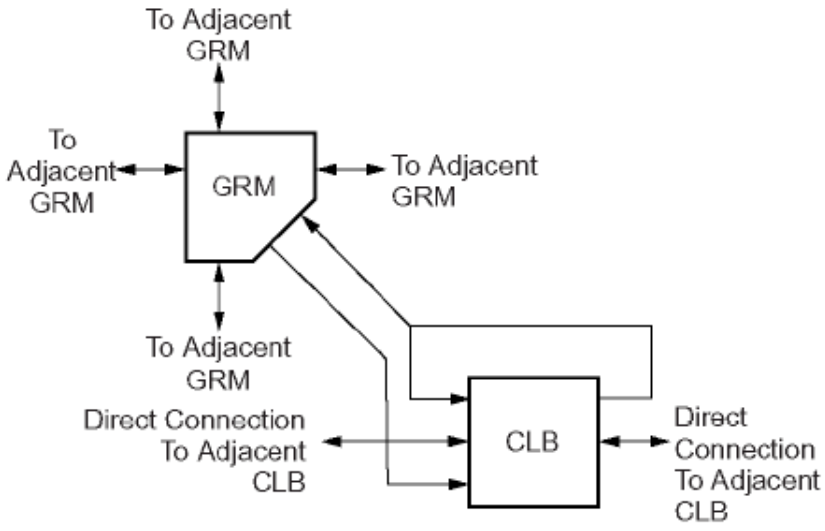

Figure 20: Local Routing

General Purpose Routing

Most signals are routed on the general purpose routing, and consequently, the majority of interconnect resources are associated with this level of the routing hierarchy. The general routing resources are located in horizontal and vertical routing channels associated with the rows and columns CLBs. The general-purpose routing resources are listed below.

- ◇ Adjacent to each CLB is a General Routing Matrix (GRM). The GRM is the switch matrix through which horizontal and vertical routing resources connect, and is also the means by which the CLB gains access to the general purpose routing.
- ◇ 24 single-length lines route GRM signals to adjacent GRMs in each of the four directions.
- ◇ 96 buffered Hex lines route GRM signals to other GRMs six blocks away in each one of the four directions. Organized in a staggered pattern, Hex lines may be driven only at their endpoints. Hex-line signals can be accessed either at the endpoints or at the midpoint (three blocks from the source). One third of the Hex lines are bidirectional, while the remaining ones are unidirectional.
- ◇ 12 Longlines are buffered, bidirectional wires that distribute signals across the device quickly and efficiently. Vertical Longlines span the full height of the device, and horizontal ones span the full width of the device.

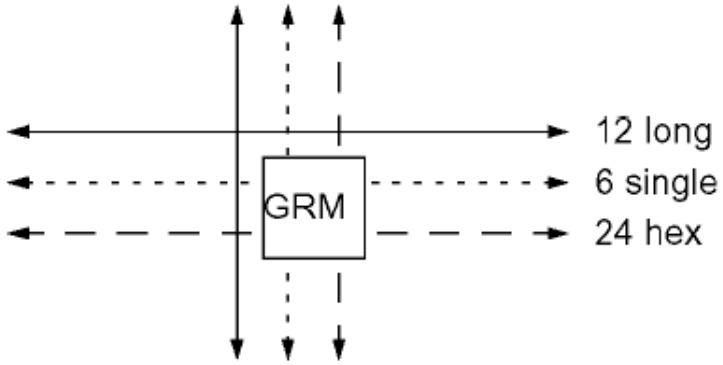

Figure 21: General Purpose Routing

I/O Routing

Devices may have additional routing resources around their periphery that form an interface between the CLB array and the IOBs. This additional routing, called the VersaRing, facilitates pin-swapping and pin-locking, such that logic redesigns can adapt to existing PCB layouts. Time-to-market is reduced, since PCBs and other system components can be manufactured while the logic design is still in progress.

Dedicated Routing

Some classes of signals require dedicated routing resources to maximize performance. In recent architectures, dedicated routing resources are provided for two classes of signals:

- ◇ Horizontal routing resources are provided for on-chip 3-state busses. Four partitionable bus lines are provided per CLB row, permitting multiple busses within a row, as shown in Figure 22.
- ◇ Two dedicated nets per CLB propagate carry signals vertically to the adjacent CLB.

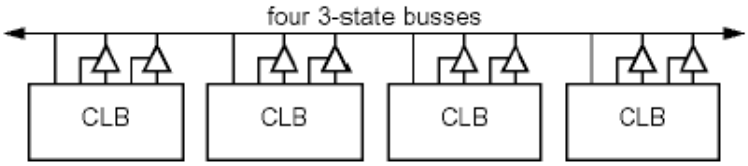

Figure 22: BUFT Connections to Dedicated Horizontal Bus Lines

Global Routing

Global Routing resources distribute clocks and other signals with very high fanout throughout the device. Recent devices include two tiers of global routing resources referred to as primary and secondary global routing resources.

- ◇ The primary global routing resources are four dedicated global nets with dedicated input pins that are designed to distribute high-fanout clock signals with minimal skew. Each global clock net can drive all CLB, IOB, and block RAM clock pins. The primary global nets may only be driven by global buffers. There are four global buffers, one for each global net.
- ◇ The secondary global routing resources consist of 24 backbone lines, 12 across the top of the chip and 12 across bottom. From these lines, up to 12 unique signals per column can be distributed via the 12 longlines in the column. These secondary resources are more flexible than the primary resources since they are not restricted to routing only to clock pins.

Clock Distribution

Typical FPGA families provide high-speed, low-skew clock distribution through the primary global routing resources described above. A typical clock distribution net is shown in Figure 23.

Four global buffers are provided, two at the top center of the device and two at the bottom center. These drive the four primary global nets that in turn drive any clock pin.

Four dedicated clock pads are provided, one adjacent to each of the global buffers. The input to the global buffer is selected either from these pads or from signals in the general purpose routing.

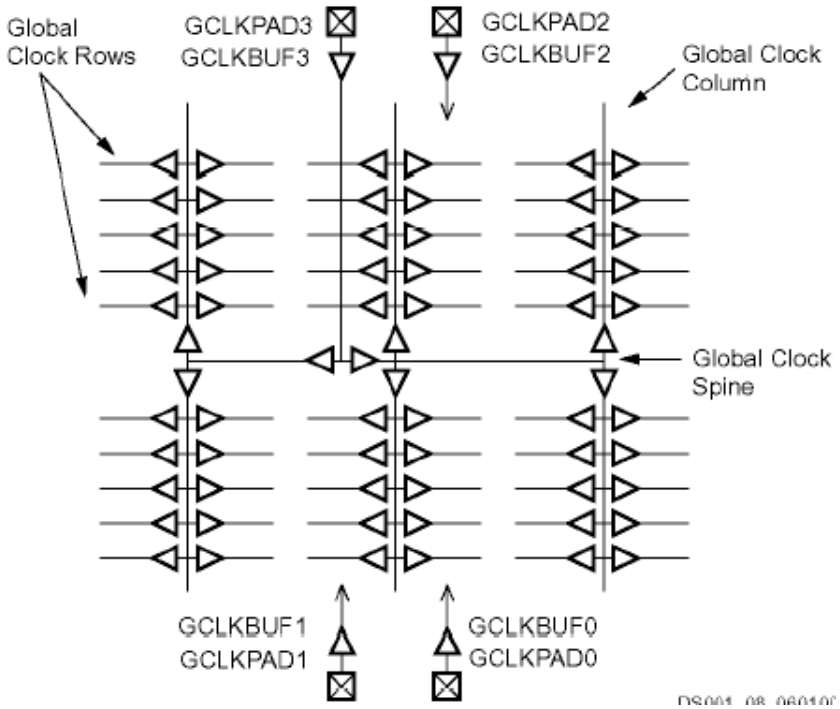

Figure 23: Global Clock Distribution Network

Delay-Locked Loop (DLL)

Associated with each global clock input buffer is a fully digital Delay-Locked Loop (DLL) that can eliminate skew between the clock input pad and internal clock-input pins throughout the device. Each DLL can drive two global clock networks. The DLL monitors the input clock and the distributed clock, and automatically adjusts a clock delay element. Additional delay is introduced such that clock edges reach internal flip-flops exactly one clock period after they arrive at the input. This closed-loop system effectively eliminates clock-distribution delay by ensuring that clock edges arrive at internal flip-flops in synchronism with clock edges arriving at the input.

In addition to eliminating clock-distribution delay, the DLL provides advanced control of multiple clock domains. The DLL provides four quadrature phases of the source clock, can double the clock, or divide the clock by 1.5, 2, 2.5, 3, 4, 5, 8, or 16. It has six outputs. The DLL also operates as a clock mirror. By driving the output from a DLL off-chip and then back on again, the DLL can be used to deskew a board level clock among multiple Spartan-II devices.

In order to guarantee that the system clock is operating correctly prior to the FPGA starting up after configuration, the DLL can delay the completion of the configuration process until after it has achieved lock.

Block RAM

Recent FPGA families incorporate several large block RAM memories. These complement the distributed RAM Look-Up Tables (LUTs) that provide shallow memory structures implemented in CLBs. The number of memory blocks depends on the size of the FPGA device, e.g. a Xilinx XC2S200 device contains 14 blocks totalling to 56k bits of memory

Each block RAM cell, as illustrated in Figure 24, is a fully synchronous dual-ported 4096-bit RAM with independent control signals for each port. The data widths of the two ports can be configured independently, providing built-in bus-width conversion.

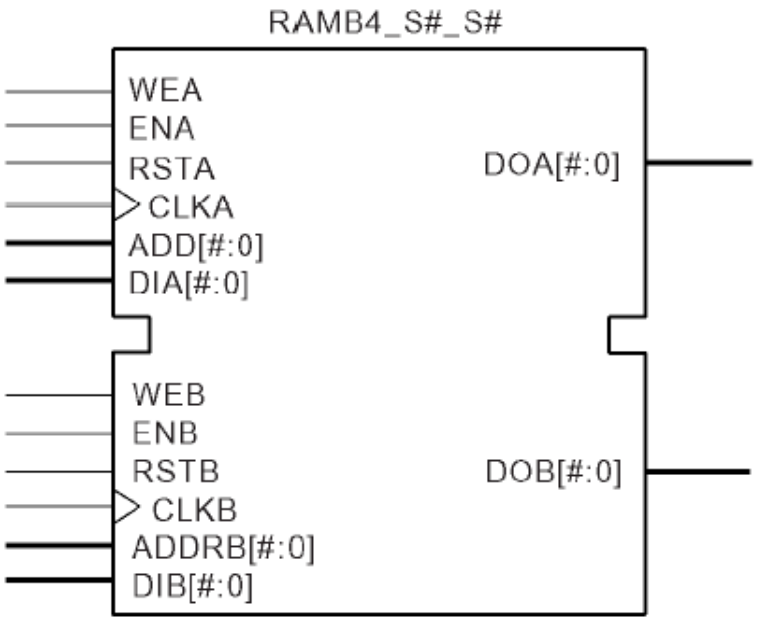

Figure 24: Dual-port Block RAM

Gate count metrics

Introduction

Every user of programmable logic at some point faces the question: „How large a device will I require to fit my design?“ In an effort to provide guidance to their users, FPGA manufacturers describe the capacity of FPGA devices in terms of „gate counts“. Gate counting involves measuring logic capacity in terms of the number of 2-input NAND gates that would be required to implement the same number and type of logic functions. The resulting capacity estimates allow users to compare the relative capacity of different FPGA devices.

Xilinx uses three metrics to measure the capacity of FPGAs in terms of both gate counts and bits of memory: „Maximum Logic Gates“, „Maximum Memory Bits“, and „Typical Gate Range“.

Maximum Logic Gates

„Maximum Logic Gates“ is the metric used to estimate the maximum number of gates that can be realized in the FPGA device for a design consisting of only logic functions. (On-chip memory capabilities are not factored into this metric.) This metric is based on an estimate of the typical number of usable gates per configurable logic block or logic cell multiplied by the total number of such blocks or cells. This estimate, in turn, is based on an analysis of the architecture of the logic block and empirical data obtained by comparing the implementation of entire system-level designs in the FPGA devices and traditional gate arrays.

The slices of the Spartan-II Series devices each contain three function generators and two registers (Figure 16). Additional resources in the block include dedicated arithmetic carry logic. Using Table 2 as a guide, the potential gate count for a single slice can be derived. The table lists the gate counts for a sampling of logic functions; these gate counts are taken directly from a typical mask-programmed gate arrayís library.)

Function	Gates
Combinational functions	
2-input NAND	1
2-to-1 Multiplexer	4
3-input XOR	6
4-input XOR	9
2-bit carry-save full adder	9
Register functions	
D flip-flop	6
D flip-flop with set or reset	8
D flip-flop with reset and enable	12

Table 2: Gate Counts for Common Functions

The function generators are implemented as memory look-up tables (LUTs); the F and G function generators are 4-input LUTs, and the H function generator is a 3-input LUT. Each LUT is capable of generating any logic function of its inputs; thus, in a given application, a 4-input LUT might be used for any operation ranging from a simple inverter or 2-input NAND (1 gate) to a complex function of 4 inputs, such as a 4-input exclusive-OR (9 gates) or, along with the built-in carry logic, a 2-bit full adder (9 gates). Similarly, the registers in the CLB account for anywhere from 6 to 12 equivalent gates each, dependent on whether built-in functions such as the asynchronous preset/clear and clock enable are utilized.

CLB Resource	Gate Range
Gate range per 4-input LUT (2 per slice)	1 to 9
Gate range per 3-input LUT	1 to 6
Gate range per flip-flop (2 per slice)	6 to 12
Total gate range per slice	15 to 48
Estimated typical number of gates per slice	28.5

Table 3: Capacity ranges for CLB slice Resources

Thus, assuming that all three LUTs and both flip-flops are utilized, a single CLB slice may hold anywhere from 15 to 48 gates of logic (Table 3). Using empirical data based on the compilation of system-level designs, the actual obtainable usage is estimated as about 28.5 gates per Spartan-II Series slice.

Of course, in a given application, all the resources in every CLB will not be utilized. Therefore, this metric is a maximum in that it assumes that every CLB is being used. This simple analysis does not take into account the many other logic resources available in the FPGA architecture, including on-chip three-state buffers, global clock buffers, global reset, the registers and multiplexers in the I/O blocks, readback circuitry, and JTAG boundary scan test circuitry.

Maximum Memory Bits

Some FPGA devices, such as the are capable of integrating RAM or ROM memory functions as well as logic functions on chip. This metric, quite simply, is the maximum number of memory bits that can be implemented on the device.

The F and G function generators optionally can be configured as a 32x1 or 16x2 block of asynchronous or synchronous RAM or ROM memory. Thus the maximum distributed RAM bits are the number of slices multiplied by 32. In addition to the distributed RAM, recent FPGA families offers a number of block RAMs, each providing 4k bits of memory.

Typical Gate Range

FPGA users should realize that there can be considerable variation in the logic capacity of a given FPGA device dependent on factors such as how well the application‘s logic functions match the architecture of the FPGA device, the efficiency of the tools used to synthesize the logic and map, place, and route the device in the FPGA, and the skill and experience of the designer. For example, a given design is unlikely to use every available CLB or logic cell. For this reason, the Maximum Logic Gates metric is complemented with a Typical Gate Range estimate. Based on empirical data, this metric is intended to set realistic expectations by providing both a low end and high end estimate of FPGA capacity.

Most large system-level designs will include some memory as well as logic functions, and it is reasonable to assume that some memory functions would be implemented on an FPGA in the typical system. FPGA architectures allow the on-chip integration of memory as well as logic functions, and the Typical Gate Range capacity metric takes this capability into account. In a sea-of-gates gate array, memory functions require about 4 logic gates per bit of memory. Thus, each Spartan-II Series slice is capable of implementing 32x4=128 gates of memory functions.

The low end of the Typical Gate Range assumes that all the CLBs are used for logic, with a utilization of about 18 gates/slice. In addition, about 10% of the block RAM resources are used. Table 4 summarizes the calculation for a Xilinx XC2S200 device.

Resource	Gates
100% Logic: 1.0 x 1176 CLBs x 2 slices/ CLB @ 18 gates/ slice	42,336
10% of Block Memory: 0.1 x 14 Blocks x 4096 bits/ Block @ 4 gates/ bit	22,938
Total	**65,274**

Table 4: XC2S200 Low End of Gate Range

The high end of the Typical Gate Range assumes 20% of the CLBs are used as memory, and the remaining CLBs are used as logic, with 128 gates/CLB for memory functions and 26 gates/slice for logic functions. Furthermore, 40% of the block RAMs are utilized and add to the capacity. Table 5 summarizes the calculation for a Xilinx XC2S200 device.

Resource	Gates
80% Logic: 0.8 x 1176 CLBs x 2 slices/ CLB @ 26 gates/ slice	48,922
20% Distributed Memory: 0.2 x 1176 CLBs x 2 slices/ CLB @ 128 gates/ slice	60,211
40% of Block Memory: 0.4 x 14 Blocks x 4096 bits/ Block @ 4 gates/ bit	91,750
Total	**200,883**

Table 5: XC2S200 High End of Gate Range

Using Gate Counts as Capacity Metrics

As long as the metrics used to establish gate counts are fairly consistent across product families, these metrics are useful when migrating between FPGA families, or when applying the experience gained using one family to help select the appropriately-sized device in another family. The gate count metrics also are a good indicator of relative device capacities within each FPGA family, although comparing the number of CLBs or logic cells among the members of a given family provides a more direct measure of relative capacity within that family.

Unfortunately, claimed gate capacities are not a very good metric for comparing the density of FPGAs from different vendors. There is considerable variation in the methodologies used by different FPGA manufacturers to ìcount gatesî in their products. A better methodology for comparing the relative logic capacity of competing manufacturersí devices is to examine the type and number of logic resources provided in the device.

Footprint Compatibility Lessens Risk

Designers do not always guess right when initially selecting the FPGA family member most suitable for their design. Thus, footprint compatibility is an important feature for maximizing the flexibility of FPGA designs. Footprint compatibility refers to the availability of FPGAs of various gate densities with the same package and with an identical pinout. When a range of footprint-compatible devices is available, users have the ability to migrate a given design to a higher or lower density device without changing the printed circuit board (PCB), thereby lowering the risk associated with initial device selection. If the selected device turns out to be too small, the design is migrated to a larger device. If the selected device is too big, the design can be moved to a smaller device. In either case, with footprint-compatible devices, potentially expensive and time-consuming changes to the PCB are avoided.

FPGA Implementation Overhead

Having a gate-count metric for a device, helps to estimate the overhead created by the programmable logic fabric, compared to a standard-cell ASIC.

First, we calculate the number of configuration bits required for a single logic slice. The block RAM bits are excluded from this calculation, as the block RAM implementation is close to the ideal implementation and therefore can be directly compared between ASICs and FPGAs.

Resource	bits
config file size	1,335,840
Block RAM bits	57,344
bits used for logic	1,278,496
# of slices	2,452
bits per slice	**544**

Table 6: Configuration Bits per Slice

Each configuration bit has to be stored in a single flip-flop. This flip-flop in turn, controls a specific attribute of the logic cell‘s behavior. Assuming every bit drives a single additional gate is very conservative.

Resource	gates
config storage 544 bits/ slice @ 4 gates/ bit	2,176
behavior 544 bits/ slice @ 1 gate/ bit	544
gates per slice	**2,720**

Table 7: Gates per Slice

These gates numbers may be divided by the typical gates per slice to gain an impression of the ìgates per gatesî implementation overhead. As the feature of distributed RAM is part of the overhead, we have to take it into account here.

Resource	gates
gates per slice	2,720
average gates per slice 0.8 x 26 + 0.2 x 128	46.4
implementation overhead	**59**

Table 8: Implementation Overhead

Taking the square root of this result, gives an impression of the overhead in geometric units. The overhead of 59 implies, that a 0.15-micron FPGA easily reaches die size parity with a 1.2-micron standard-cell ASIC.

Performance Characteristics

According to the data sheets, Spartan-II devices provide system clock rates up to 200 MHz and internal performance as high as 333 MHz. This section provides the performance characteristics of some common functions. Unlike the data sheet figures, these examples have been described in VHDL and ran through the standard synthesis and implementation tools to achieve an understanding of the real world performance.

Function	Pin-to-Pin (w/ IO delays) [MHz]	
	XC2S200-5	XC2S200E-6
16-bit Address Decoder	109	118
32-bit Address Decoder	87	90
64-bit Address Decoder	74	82
4:1 Mux	127	138
8:1 Mux	113	115
16:1 Mux	99	99
32:1 Mux	88	93
pad-LUT-pad	139	156

Table 9: Pin-to-Pin Performance Spartan-II vs. Spartan-IIE

Table 9 provides pin-to-pin values including IOB delays; that is, delay through the device from input pin to output pin. In the case of multiple inputs and outputs, the worst delay is reported; all values are reported in MHz.

Table 10 shows internal (register-to-register) performance. Again, values are reported in MHz.

For all performance data it should be remembered, that about 50% of the delays are caused by routing delays. The routing delays are highly dependent on device utilization and the quality of the place & route process.

Function	Register-to-Register [MHz]	
	XC2S200-5	XC2S200E-6
16-bit Address Decoder	181	232
32-bit Address Decoder	144	199
64-bit Address Decoder	124	157
4:1 Mux	237	328
8:1 Mux	230	295
16:1 Mux	180	216
32:1 Mux	155	171
Register-LUT-Register	285	407
8-bit Adder	183	227
16-bit Adder	175	211
64-bit Adder	77	112
64-bit Counter	88	121
64-bit Accumulator	72	111

Table 10: Register-to-Register Performance Spartan-II vs. Spartan-IIE

FPGA design flow

Design Entry

Design Entry is the process of creating the design and entering it into the development system. The following methods are widely used for design entry:

- ◇ HDL Editor
- ◇ State Machine Editor
- ◇ Block Diagram Editor

Typing a design into an HDL Editor is the most obvious way of entering high-level languages like VHDL into the development system. Recent editors offer functionality like syntax highlighting, auto completion or language templates to speed-up design entry. The main advantage of using an HDL Editor for design entry is, that text files are simple to share across tools, platforms and sites. On the other side, text may not be the most convenient way of editing a design; however this is highly dependent on the design.

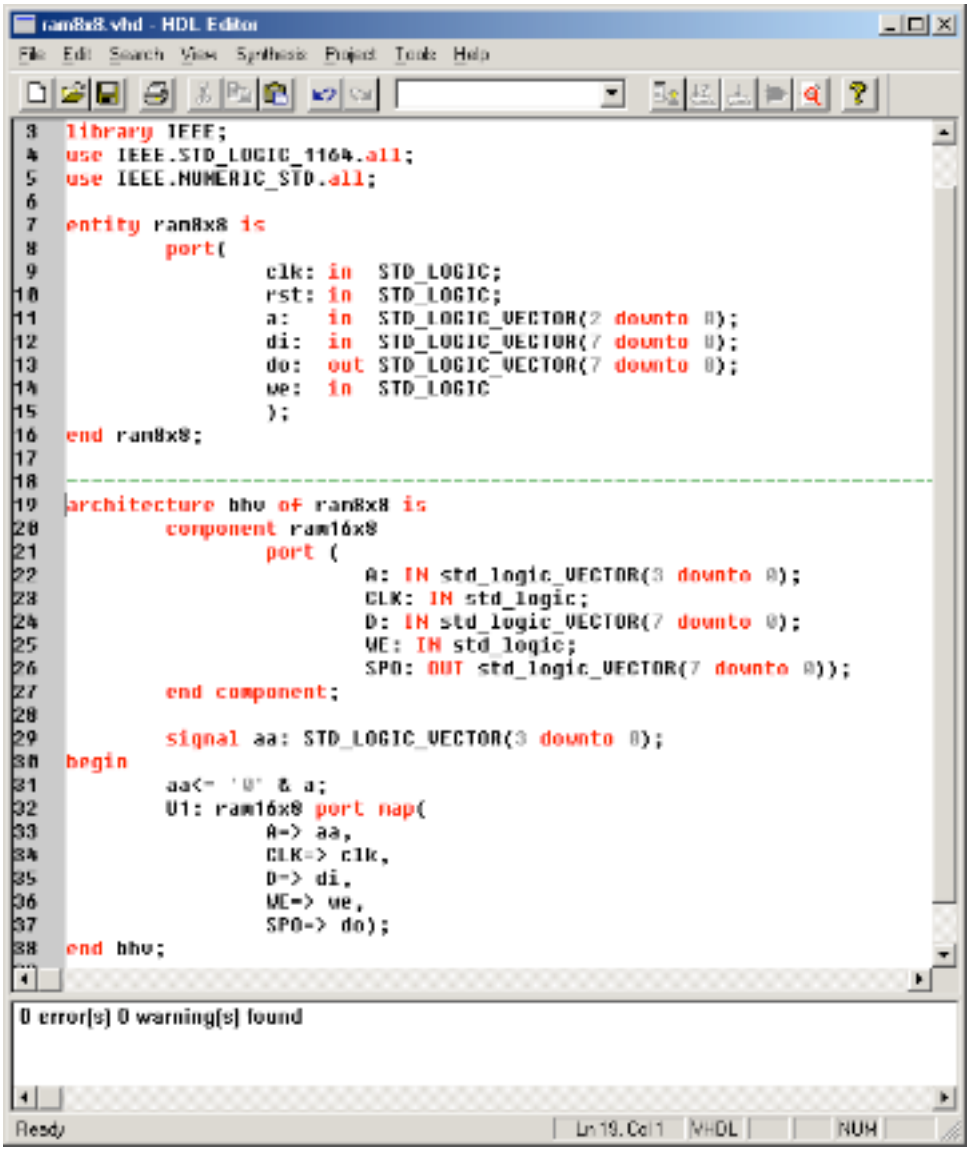

Figure 25: HDL Editor

For creating finite state machines, special editors are available. Using these editors is a convenient way of creating FSMs by graphical entry of bubble diagrams. Most tools create VHDL from the graphics representation, but hide this process completely from the user. The main advantage is, that the graphical representation is much easier to understand and maintain. On the other side, sharing a design across tool or platform boundaries may be difficult.

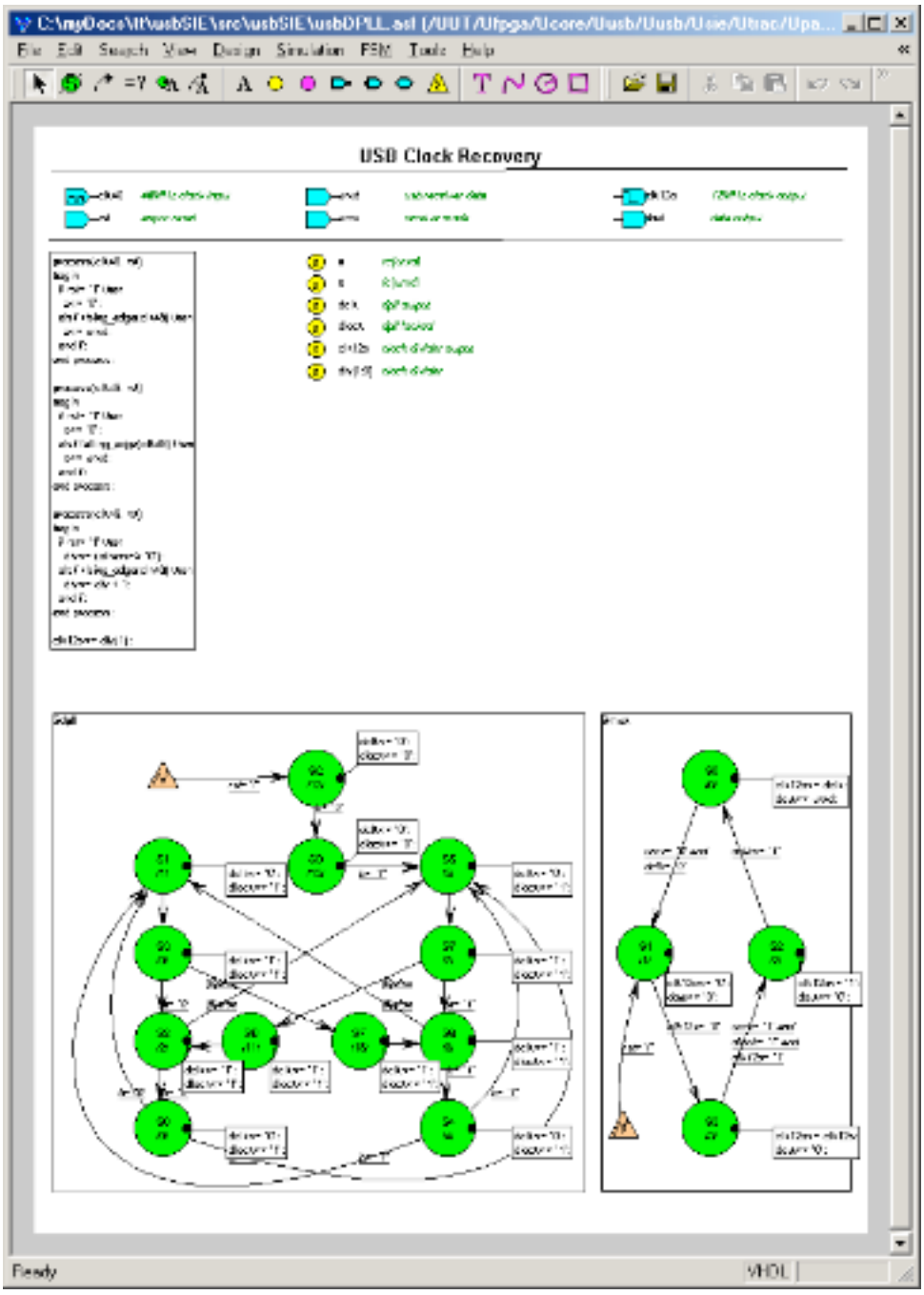

Figure 26: FSM Editor

For creating structural designs, block diagram editors are available. Like FSM editors, these tools create VHDL or EDIF from the graphical representation and hide this process from the user. Again, the main advantage is, that the graphical representation is easier to understand and maintain, with the drawback of a reduced compatibility across tool or platform boundaries.

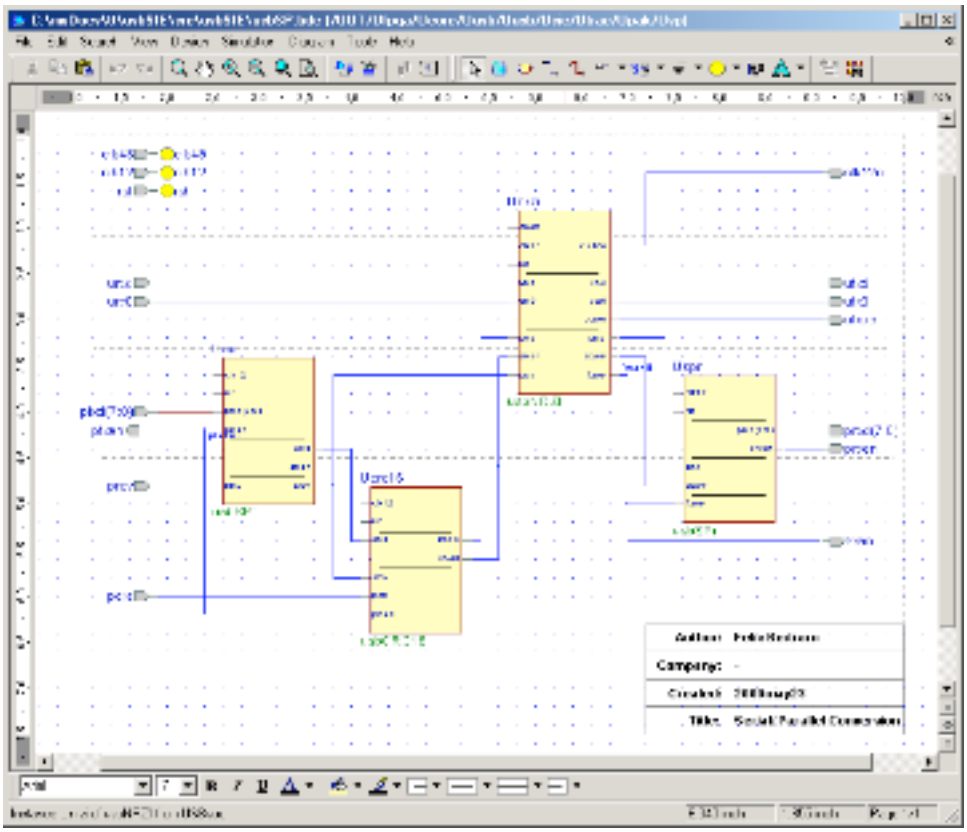

Figure 27: Block Diagram Editor

Behavioral Simulation

After design entry, the design is verified by performing behavioral simulation. To do so, a high-level or behavioral simulator is used, which executes the design by interpreting the VHDL code like any other programming language, i.e. regardless of the target architecture. At this stage, FPGA development is much like software development; signals and variables may be watched, procedures and functions may be traced, and breakpoints may be set. The entire process is very fast, as the design is not synthesized, thus giving the developer a quick and complete understanding of the design. The downside of behavioral simulation is, that specific properties of the target architecture, namely timing and resource usage are not covered.

Synthesis

Synthesis is the process of translating VHDL to a netlist, which is built from a structure of macros, e.g. adders, multiplexers, and registers. Chip synthesizers perform optimizations, especially hierarchy flattening and optimization of combinational paths. Specific cores, like RAMs or ROMs are treated as black boxes. Recent tools can duplicate registers, perform re-timing, or optimize their results according to given constraints.

Post-Synthesis Simulation

After performing chip synthesis, post-synthesis simulation is performed. Timing information is either not available, or preliminary based on statistical assumptions which may not reflect the actual design. As the design hierarchy is flattened and optimized, tracing signals is difficult. Due to the mapping of the design into very basic macros, simulation time is lengthy. When post-synthesis results differ from behavioral simulation, most likely initialization values have been omitted, or don't-cares have been resolved in unexpected ways.

Implementation

Implementation is the process of translating the synthesis output into a bitstream suited for a specific target device. This process consists of the following steps:

- ◇ translation
- ◇ mapping
- ◇ place & route

During *translation*, all instances of target-specific or external cores, especially RAMs and ROMs are resolved. This step is much like the linking step in software development. The result is a single netlist containing all instances of the design.

During *mapping*, all macro instances are mapped onto the target architecture consisting of LUTs, IOBs, and registers. With this step completed, the design is completely described in primitives of the target architecture.

During *place & route*, all instances are assigned to physical locations on the silicon. This is usually an iterative process, guided by timing constraints provided by the designer. The process continues, until the timing constraints are either met, or the tool fails to further improve the timing.

Timing Simulation

After implementation, all timing parameters are known, therefore a real timing simulation may be performed. Timing simulation is a lengthy task, as the structure of the silicon including timing is simulated. Furthermore, it is difficult to create testbenches, which exercise the critical timing paths. For this reason, most designers do not perform timing simulation, but a combination of behavioral simulation and static timing analysis.

Static Timing Analysis

Static timing analysis computes the timing of combinational pathes between registers and compares it against the timing constraints provided by the designer. The confidence level of this method depends on the coverage and correctness of the timing constraints. However, for synchronous designs with a single clock domain, static timing analysis may render timing simulation obsolete.

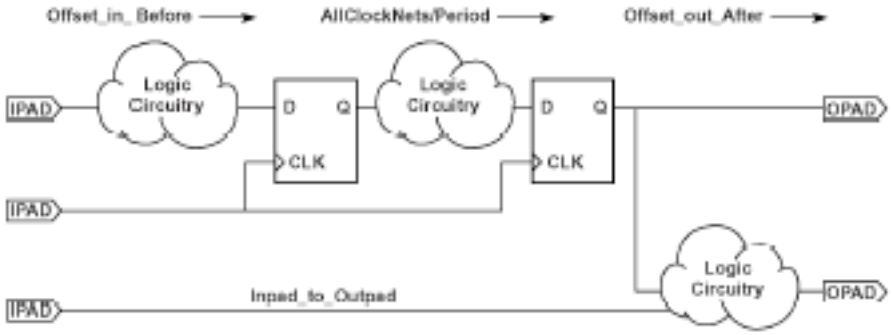

Figure 28: Timing Constraints

Intellectual Property

What are IP-Cores?

In the terminology of a chip designer, IP-Cores are building blocks of intellectual property. These blocks encapsulate specific standard functionality of a chip in a way much like standard circuits do on a PCB. The intended use of IP-Cores is in both FPGA and ASIC type devices as part of a system-on-a-chip design solution.

Why use IP-Cores?

While semiconductor technology is rapidly evolving, hardware designers are faced with a new dimension of problems:

- *Increased design effort*. Forced by the embedded revolution with virtually every digital hardware product using sophisticated, brand-new and microprocessor controlled technology, design effort moves quickly along an upwards spiral.
- *Shortened product life cycles.* At the same time, product life cycles are getting shorter, especially in competitive business areas like digital consumer products.
- *Increased design risk.* The obvious answer to the above facts is to start product development earlier than ever before, resulting in leading-edge product design being started before standards are fully established. This in turn dramatically increases design risk.

One solution to address these issues is the use of IP-Cores. Using IP-Cores adds the following beneficial properties to development:

- *Built-in expert know-how.* Building up the expertise required to successfully master challenging technology is a time-consuming task. IP-Cores are designed by experts, incorporating IP-Cores into your design gives access to their expert skills.
- *Built-in confidence.* Testing hardware under worst-case conditions is another time-consuming task. IP-Cores are standard products which ran through a variety of design-flows, were evaluated by lots of skilled engineers, and have been applicated in several state-of-the-art products. This creates the confidence of proven functionality.
- *Built-in future.* IP-Cores are fully documented modules, which ship with source code and test bench available. No risk of discontinued circuits, no problem with rotating employees. Maintenance of a design is possible- even years later with a new design team.
- *Built-in profession.* With standard circuits being implemented in IP-Cores designers come back to their real profession as engineers: Concentrating on designing a unique and distinctive product.

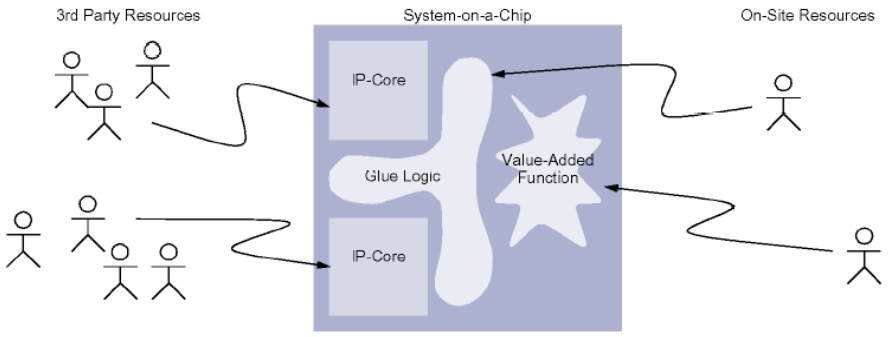

Figure 29: Building a system with IP-Cores

The future of FPGA technology

- Microprocessor Systems
  - ◇ PPC405 core
- Digital Signal Processing
  - ◇ embedded Multipliers
- I/O Processing
  - ◇ Rocket I/O
  - ◇ DCI

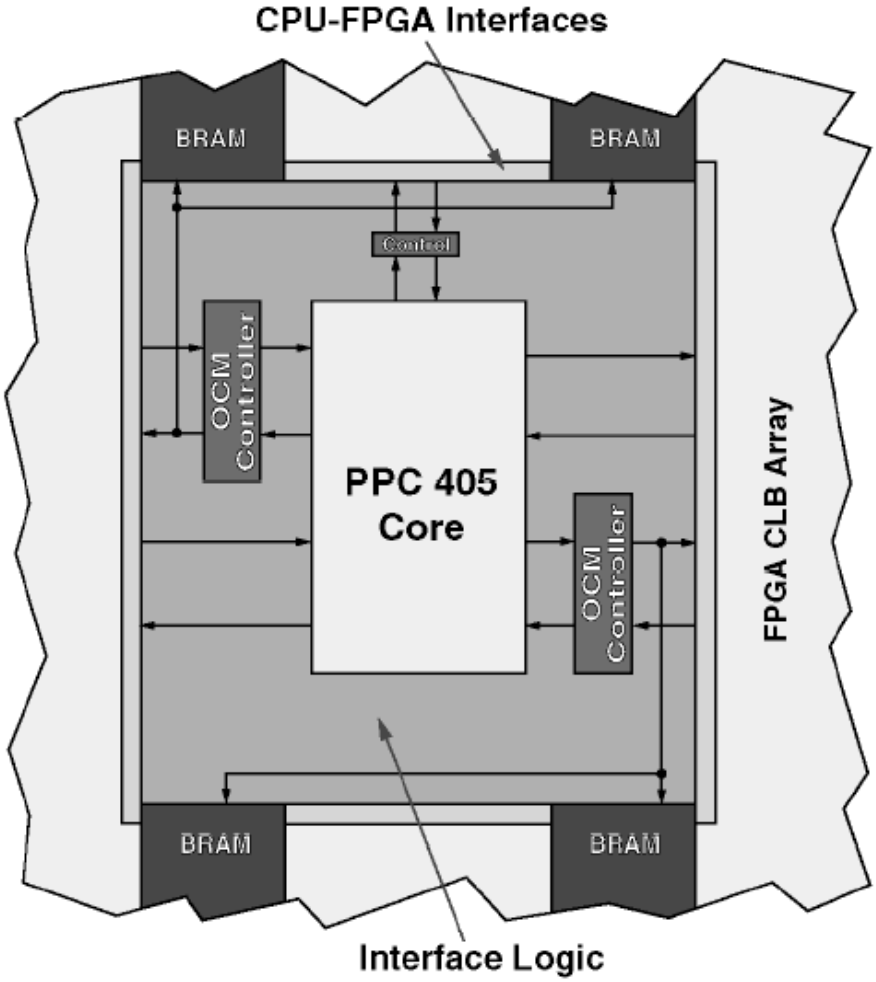

Figure 30: Embedded PowerPC

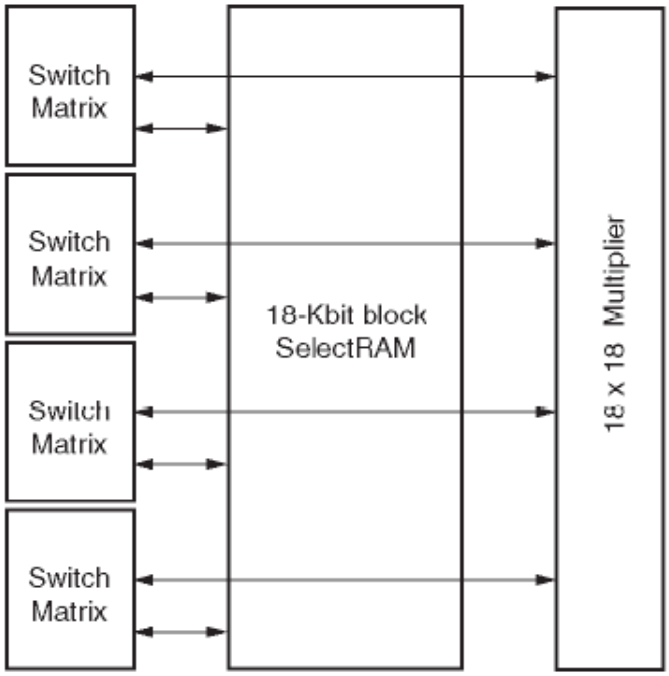

Figure 31: Embedded Multipliers

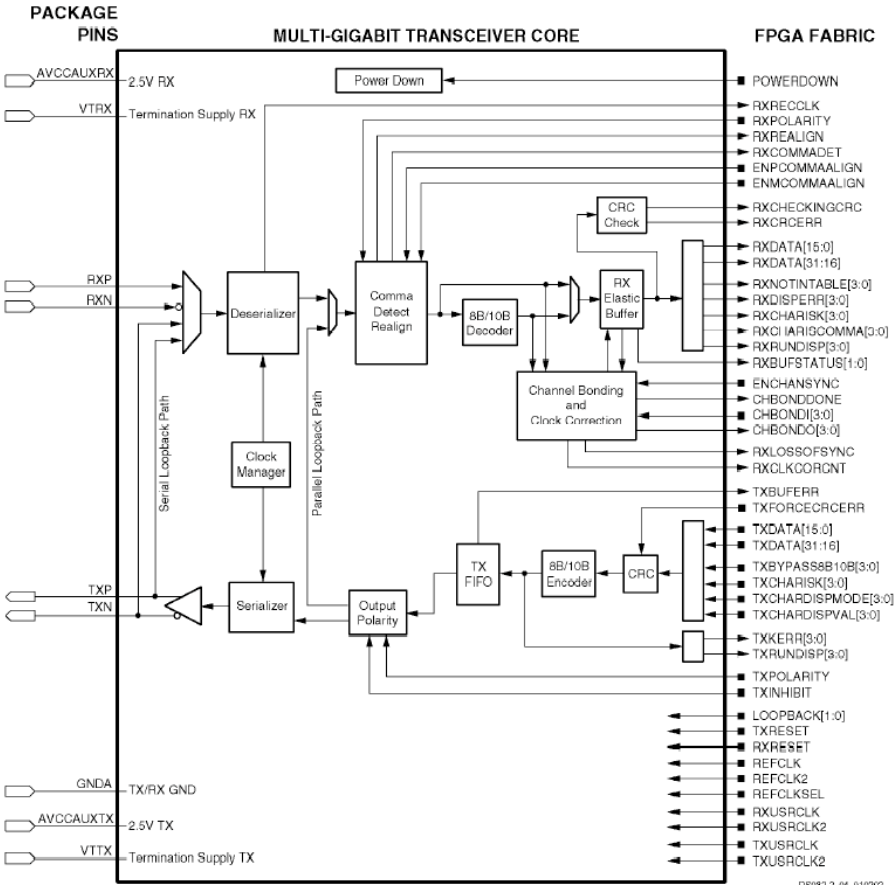

Figure 32: Rocket I/Os

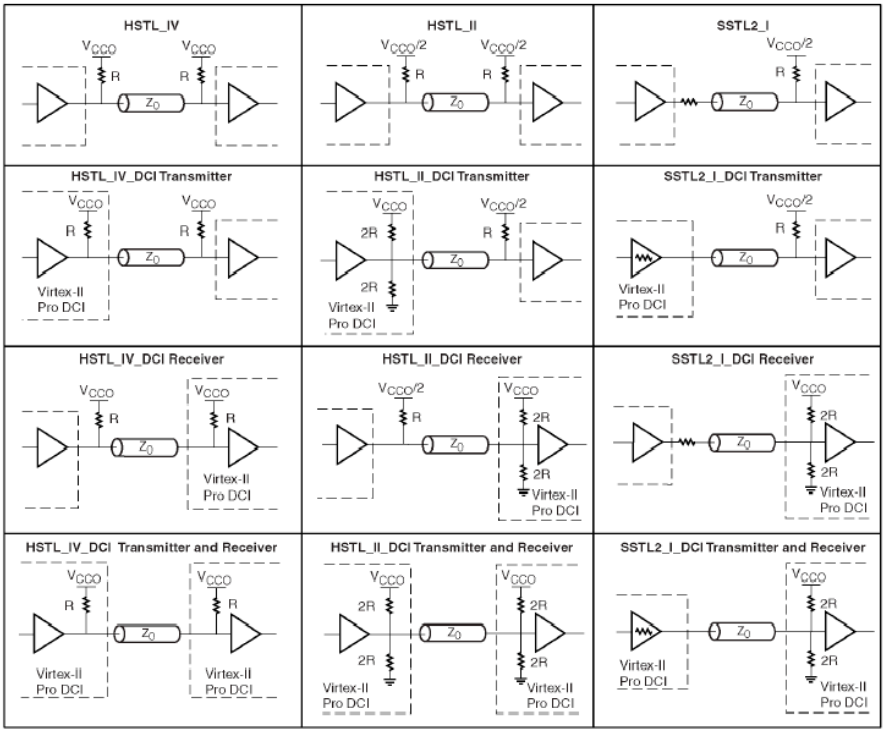

Figure 33: Digitally Controlled Impedance