Electronic design with FPGAs

Custom Logic Options

In most digital designs, the circuitry can be classified by the following categories:

- *Standard products*. These products provide a functionality which is not associated with a specific application area but common to a broad range of devices. Typical parts in this category are processors and memories.
- *Application Specific Standard Products* or ASSPs. These products provide functionality which is not associated with a specific implementation, but common to an application area. Typical parts in this category are MPEG decoders.
- *Custom Logic*. This logic is associated with a specific application and is the essence of what distincts one product from another. Often this is *glue logic*, connecting standard products or ASSPs with each other.

There are several options on how to implement custom logic, FPGAs being one amongst them. These options are discussed in the following paragraphs.

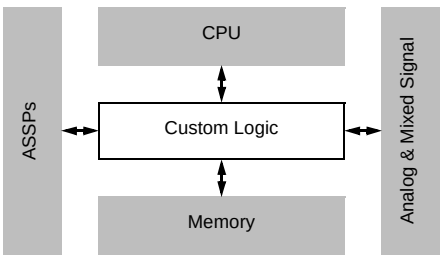

Figure 1: Typical Digital System

ASICs

ASIC is the abbreviation of *Application Specific Integrated Circuit*. This denotes an integrated circuit, that is fully customized to the requirements of a specific application. Today's ASICs are usually designed with a *Standard Cell* approach. This means, that the circuit is not designed on the level of transistors, but on the level of gates, flips-flops, and memory blocks. The basis for this is a library of primitives, which is provided by the silicon foundry. ASICs can be characterized by the following items:

- Lowest production cost
  - a single wafer for $1,000 to $3,000 yields thousands of chips
- Highest possible design density
- Highest NRE and Re-Spin cost:
  - starting at about $2,500 for 2um process on multi-project wafer
  - about $250,000 for 0.13um process
- Highest development effort
  - timing
  - fault-coverage vectors
  - stringent verification
  - physical design by separate set of engineers

As a standard cell layout results in a nearly optimal implementation of a given circuit, the production cost is at the lowest possible level.

This advantage is compromised by high *Non-Recurring Engineering* costs (NRE) which are mainly caused by expensive chip masks. In case re-spins are required due to silicon faults or design changes, new masks are required, resulting further significant costs.

ASICs require an enormous development effort. This is especially caused by the difficulty to simulate timing and by the need to provide test vectors to the production foundry.

Due to these facts, ASIC development is associated with high volume productions. To achieve these high volumes, special care has to be taken, that the chips fit universally into a broad set of applications. This often requires a significant amount of configuration options.

Gate Arrays

Gate arrays are circuits providing a fixed *Sea-of-Gates* fabric which is defined by the manufacturer, and an interconnection infrastructure which is customized according to the application's requirements. Gate Arrays can be characterized by the following items:

- Low production cost
  - comparable to ASICs
- Low design density
  - fixed architecture leads to low gate utilization
- High NRE and Re-Spin cost
  - about $100,000 for 0.35um process
- High development effort
  - timing
  - fault-coverage vectors

As the silicon remains unaltered throughout a gate array family and the top metal layers are the only parameter which is customized to the application, production cost is comparable to ASICs.

For the same reason, the NRE and re-spin costs are drastically reduced, compared to ASICs but still on a high level.

The given and proven silicon results in a reduced development effort, especially as there are less parameters to be considered for timing closure of the design.

Combining these facts, Gate Arrays seem to be a much more attractive choice than ASICs. Unfortunately process development stopped at about 0.35um, which leads to a much lower transistor density, compared to recent ASICs. The low gate utilization resulting from the fixed architecture even augments this, resulting in a a comparable low design density.

FPGAs

FPGA is the abbreviation of *Field Programmable Gate Array*. This denotes an integrated circuit which is programmed in the field, i.e. by the system manufacturer. FPGAs can be characterized by the following items:

- High production cost
- Low design density
  - programmable fabric adds significant overhead
- No NRE and Re-Spin cost
- Low development effort
  - Low dead-time
  - simplified timing
  - no test vectors
  - relaxed verification
  - physical design is "hands-off"

The production cost of FPGAs is usually significantly higher than the production cost of ASICs. This is mainly caused by the fact, that FPGAs are built from a programmable fabric of logic cells which emulate the user-defined functionality. The resulting implementation overhead is discussed in the *Gate count metrics* paragraph.

Due to their programmable nature, FPGAs are rather *Programmable ASSPs* than custom logic. As the customer buys them off-the-shelf, the NRE and re-spin costs are reduced to zero.

Furthermore, the FPGAs nature of a Programmable ASSP simplifies development by a high degree: The devices are fully tested and characterized, timing is simple and predictable.

The missing NRE allows a change in tools. During design and debug, tools may favour compilation time over quality-of-results, leading to lower dead-time. For the final runs before production, the tools are configured to produce the best possible performance.

The low dead-time in turn leads to a change in mind. Logic design more and more resembles software development with an iterative approach, designers taking a higher risk and choosing more innovative approaches.

FPGAs are only available in certain sizes. This leads to a marginal cost of zero for an extra gate or routing resource. For this reason, area minimization my not always be a factor, again leading to new design approaches.

Industry Dynamics & Time-to-Market

Contemporary consumer electronics drive a change in industry dynamics. New products are taking less time to go into volume. At the same time, new products also stay in volume for shorter periods, as *Figure 2* shows.

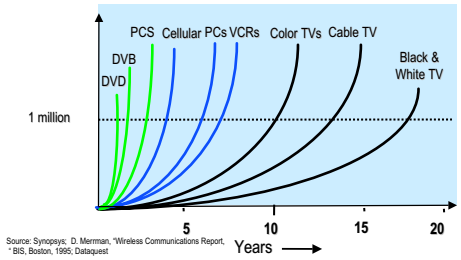

Figure 2: Changing Industry Dynamics

This gives the time-to-market discussion a new dimension, as time to market is more critical than ever. Missing a market window, or being late to market with a product because of a long development/debugging cycle can have a profoundly negative effect on the profitability of a product over its life. According to the Market Consulting Firm McKinsey and Co., late market entry has a larger effect on profits than development cost overruns or a product price that is too high. This is especially true in highly competitive markets, and those that have short market window. As *Figure 3* and *Figure 4* illustrate, a six month delay costs one third of the profits over the lifetime of the product.

A fast ramp to full production is a primary advantage of FPGAs. FPGA deliveries come off the shelf from the factory or from the inventory of

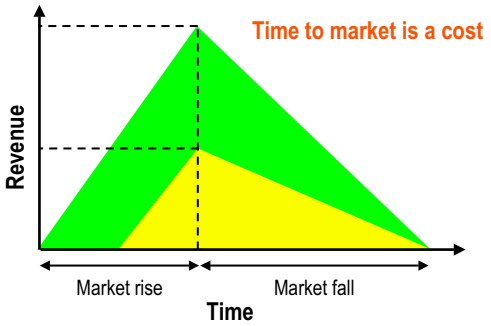

Figure 3: Time-to-Market is Critical

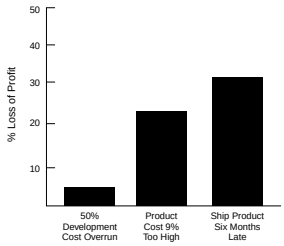

Figure 4: Time-to-Market Cost

the local distribution partners, while typical ASIC lead times run from 8-16 weeks. Immediate FPGA production enables fast stocking of sales channels and the rapid penetration of the customer base. FPGAs help to avoid the delays resulting from the long ASIC lead times that may substantially decrease revenues and profits throughout the life of the product.

Taking NRE cost and time-to-market into account, the comparison between ASICs and FPGAs changes in favour for FPGAs.

Moore's Law and the Deep Submicron Era

Process-technology innovations

Silicon process migration was relatively stable from the early 1980s until about 1993, working its way down from 5 to 1 micron, yielding state-of-the-art devices comprising about 50,000 gates. Breakthroughs in silicon design and fabrication equipment, coupled with customer demand for higher integration, spurred the rapid migration from 1-micron technology down to the 0.13-micron processes now in development. The capability to create multi-million gate designs is a reality, but with it comes a host of issues for ASIC technology, many of which FPGAs inherently resolve.

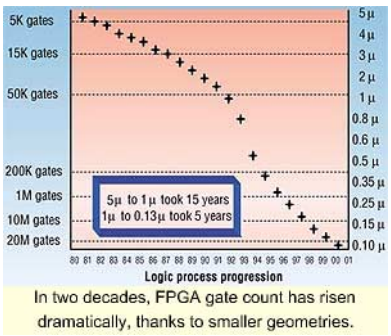

Figure 5: Two Decades of FPGA Development

Mask gate arrays are actually penalized when migrating to the advanced deep sub-micron technologies at 0.35-micron and beyond. The penalty occurs because the transistors in deep sub-micron devices have shrunk much faster than metal lines. The result is that interconnect delay now dominates gate delay. Minimizing interconnect delay requires adding metal mask layers to create more routing resources. Each photo mask for the 0.35-micron process costs the ASIC supplier from $12,000-$15,000, as well as extending the prototype fab time. Since most gate arrays today are fabricated with four or five custom metal layers, a $60,000-$75,000 cost for photo masks for each different customer design easily results in more than $100,000 in Non-Recurring Engineering (NRE) charges to the customer! A deep sub-micron gate array loses much of its value when NREs are increased to more than $100,000 and prototype time is extended. This is a primary reason that gate array vendors such as LSI Logic and Motorola have exited the gate array business to focus on the complex Standard Cell market.

The most pressing issue for the ASIC industry brought on by deep submicron technology is the difficulty achieving closure from the logical to the physical design realm. Wire delays account for 80% or more of the total delay equation in deep submicron processes, which reduces simulation margins and requires highly accurate modeling. Even with exhaustive timing simulations and delay modeling, the post-route timing can have little relation to the timing achieved through simulation. Furthermore logic designers are concerned with preventing transistor placement that could induce what's known as second- or third-order effects. These effects are artifacts of deep submicron technology; crosstalk, physical transistor-level degradation, and reflections or noise from simultaneously switching outputs are just a few of the most common. The net impact on standard-cell designers is an iterative design cycle that can take months to close. Standard-cell NRE now averages from $250,000 to over $500,000 per prototype cycle. For many companies, this is not only a major expense, but a risky one as well, since designs so often require more than one "spin" prior to completion.

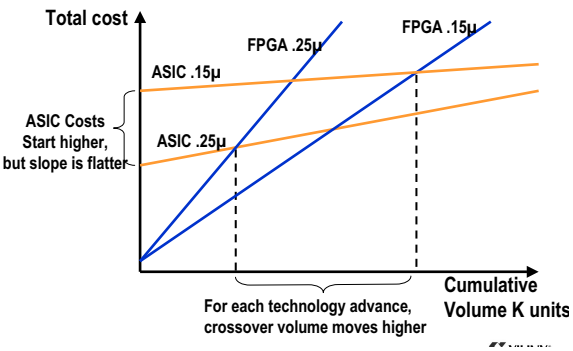

Figure 6: Technology Advances vs. Crossover Volume

By comparison, the FPGA methodology has unique advantages in the area of handling deep submicron issues. FPGA tools are integrated throughout the entire design flow, so there's no separation between logical and physical design. The design methodology flows from synthesis and simulation directly through to place and route. Designers have real silicon to test designs, so testing is done in-system, versus a "simulation-only" environment. Because FPGAs are standard products, the actual transistor-level

placement and routing is done by the FPGA vendor during product development. As the FPGA vendor already solved the general case through careful development and qualification of a robust technology, FPGA designers concentrate on maximizing the content and performance of the design, rather than wrestling with device physics.

Pad Limitation

Because FPGAs are high-volume standard products, FPGA vendors are able to utilize the most advanced fabrication processes. By doing so, die sizes and manufacturing costs can be substantially reduced. With recent FPGA families, die size has been decreased to limits imposed by the I/O pads and, as a result, has reached die size parity with many gate arrays. These FPGAs and a mask ASIC with the same number of pins are equivalent in both size and cost.

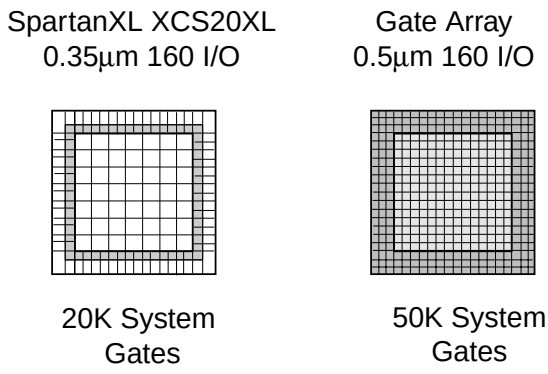

Figure 7: Effect of Pad Limitation

The graph below plots FPGA devices according to system gates and number of I/O. As indicated, FPGAs are most cost-competitive in the low-density/ high-I/O (lower right) segment. The graph may be used as a reliable indicator to determine where FPGAs can be most competitive with the ASIC in mass production. In short, when FPGAs have the same number of I/Os as the ASIC and meet the density needs of the design, the FPGA will clearly be a better choice. By choosing FPGA devices you receive the time-to-production and reprogrammability advantages of FPGAs at ASIC prices.

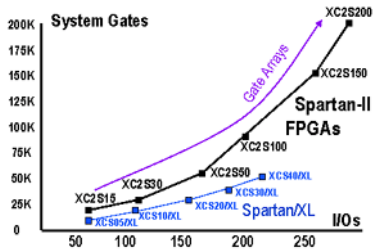

Figure 8: Pad-Limited Gate Count

Being pad limited devices, FPGAs are a perfect illustration of Moore's law, which states that for each improvement in process technology, density doubles while cost is cut in half. In high volume, a 100,000-gate FPGA sells for under $10, making it a viable solution in consumer products such as cell phones and small household appliances. On the high-density end, programmable logic users now pay the same price for 2 million gates that they paid for 20,000 gates just a few years ago. This trend is expected to continue, with 500,000 system gates costing less than $10 by 2003.

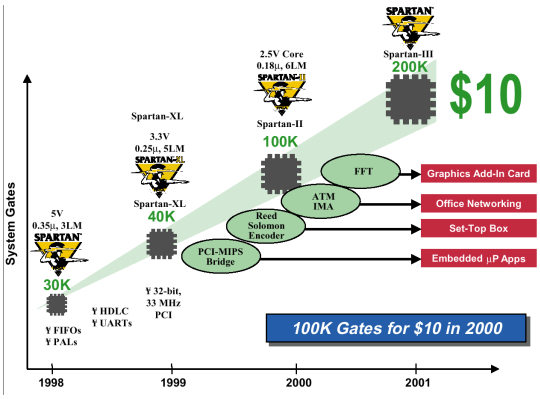

Figure 9: The $10 Corridor

FPGA Usage

In the past, FPGAs have been a niche market. Typical applications were prototyping and emulation systems. Complexity was too low to implement real products and pricing was too expensive for moderate volumes.

In the past four years, FPGA technology made large success, changing the way FPGAs are used dramatically:

- FPGAs are being used in mainstream products
  - Networking
  - Telecom
  - DSP
  - Consumer electronics
- More FPGA design starts than ASIC design starts
- 2 FPGA companies in the top 10 chip suppliers

The following paragraphs detail the use of FPGAs in different application scenarios.

Use in Emulation systems

Emulation systems are used to functionally debug complex systems. These emulation systems may be standalone units, or integrated into a computer environment to serve as simulation accelerators. There are several vendors providing off-the-shelf emulation systems; still, for specific applications, customized systems may be required.

Time-to-market	Fairly high; Fast compile times
Performance	Not stringent
Volume	Very low per application

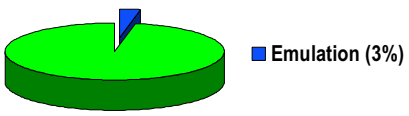

Figure 10: Use in Emulation Systems

Use in Prototyping Systems

Prototypes are units, which closely resemble the final product in functionality, however appearance, performance and technology may differ vastly from the final product. Prototypes are built in low volumes, to further investigate a system's behavior, feasibility or performance. In most cases, prototypes are used in the laboratory, however prototypes may be deployed in the field in small quantities as well.

Time-to-market	Fairly high; Fast compile times
Performance	Not stringent
Volume	Low per application

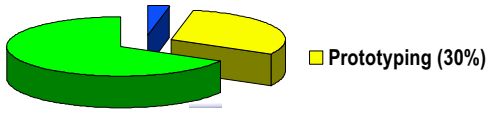

Figure 11: Use in Prototyping Systems

Use in Pre-Production Systems

During production ramp-up, manufacturers often chose to ship units with FPGAs. These FPGAs are central to the system and are planned to migrate to ASICs later on. Purpose of this is, to reduce time-to-market and the potential risk of chip fault. Interesting enough, most designs keep their FPGAs and do not migrate to ASICs, because of the appealing charme of reprogrammability.

Time-to-market	Fairly high; Fast compile times
Performance	Very critical
Volume	Moderately high per application

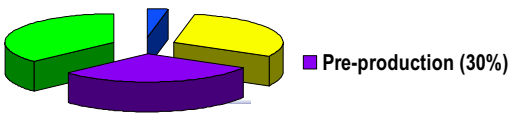

Figure 12: Use in Pre-Production Systems

Use in Production systems

FPGAs are more and more used in production systems, where they are central to the system. Replacement by ASICs is not planned, reasons are volumes or reprogrammability.

Time-to-market	Fairly high; Fast compile times
Performance	Very critical
Volume	High per application

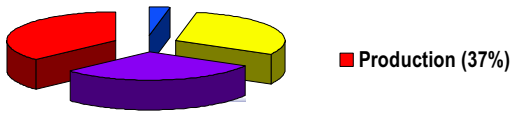

Figure 13: Use in Production systems

Case Studies

Just four years ago, FPGAs were an important but peripheral niche in the semiconductor industry. Known for their flexibility, FPGAs were widely used for ASIC emulation, glue-logic consolidation, or as a solution for applications with volatility and changing standards. FPGA process technology usually lagged the industry by at least one process generation, and the incredible power of re-programmability was paced by high production prices, slow internal performance, and tools that required 10 to 20 hours to compile 50,000 gates.

Fast forward to the year 2000, and FPGAs are often the heart of the system, being designed into mainstream as well as state-of-the-art high-volume products. Market dynamics, technology innovation, cost models, and tools have all been affected by the deep submicron era. All these factors have made FPGAs a viable alternative to ASIC design. FPGAs facilitate flexible, quick time to market and are being used as production solutions in everything from networking and industrial to consumer applications. Architecturally and by design, FPGAs minimize issues common to deep submicron technology, while maintaining the ASIC abstraction. In keeping with the entire semiconductor segment, the industry-leading FPGA companies are gaining momentum, while the smaller players consolidate or seek out specific niches.

What's fueling the popularity of FPGA technology to the point that it's now replacing both gate-array and standard-cell ASICs in many applications? A unique combination of accelerating product cycles, needs for higher integration than is possible in a gate array, fast migration to deep submicron process technology, and the explosion of networked applications are the keys.

Set-top-box

Re-programmability is the driver, allowing designers to take greater risks and innovate more during the development phases without losing time to market. It also forgives changing standards or last-minute feature changes, and enables true in-system reconfigurability. This lets telecom developers use one chip for many applications, rather than a separate ASIC for each specific custom requirement.

One example comes from a set-top-box maker who discovered that one FPGA could replace three ASICs, each one geared toward the custom specifications of Asia, Europe, and North America.

The intellectual property (IP) portfolio of FPGA suppliers is growing to include key system-level cores, including PCI, DSP, microprocessor peripherals, and key telecommunications cores. Designers cut their time to market by “dropping in” cores, eliminating the need to re-create the core design.

Designers are increasingly using FPGAs in production. Their reasons are as varied as their applications, but all are directly related to the changes that the logic industry is undergoing.

Fibre Channel switch

In another example, a designer chose an FPGA for a Fibre Channel switching project, because his ASIC plans couldn't be realized in a reasonable time frame. This Fibre Channel system is intended to increase bandwidth from 200 Mbits/s to 1 Gbit/s for IBM protocols. The system had to allow multiplexing and demultiplexing to create a smoother transition to the higher bandwidth.

Originally, the plan was to prototype the entire system using FPGAs, then migrate to ASICs once initial production turned to volume. The ASIC designs came back from the fabrication process with errors, and at the same time, the FPGA prices were continuing to drop. With their limited resources, the designers chose to pursue new product development over continuing to reduce cost. They also didn't want to lose the benefits of programmability.

ASIC End-of-Life

In yet another example, the designer chose FPGAs as a production solution when his ASIC vendor began to obsolete products in rapid succession. Some of the end products can have life spans ranging for more than ten years, so the company must support and produce its existing products as long as customers need them. When an ASIC becomes an obsolete part, engineers can either respin another ASIC or use FPGAs/CPLDs.

When inventory on gate array devices ran out, even after a “last time buy,” the company evalu-

ated programmable logic and found that they could replace obsolete ASICs with FPGAs at a competitive price point. In addition, they could move their code to the FPGA at a high level.

Remote Hardware Updating

Of all the advantages programmable logic has over ASICs, the most promising is the move toward using networks to remotely upgrade the digital hardware, not just software, in electronic equipment already installed at a customer's site. Updating software remotely with new enhancements and bug fixes is fairly common. Remotely updating hardware in this same manner might seem slightly more challenging because the system hardware is typically a fixed entity that can only be updated by manual replacement, such as board swapping.

The enabling technology to make this happen is the FPGA. Of course, electronic-equipment makers have been using FPGAs for years to create their own unique ICs. A software bitstream programs SRAM-based logic elements inside the FPGA to perform basic binary operations. Customers literally “re-wire” the FPGAs any number of times during the design process until they perfect the circuit, then use the final bit stream to program as many FPGAs as necessary for a given production run.

The types of systems that could benefit from being field updatable are wide-ranging. Almost any system that has some type of connectivity to the “outside world” could potentially benefit from being designed to support field updates. Typical products include portable phones, network appliances, set-top boxes, security systems, network equipment, cellular base stations, satellite communications systems. Other likely applications are HDTV, video and image processing, encryption, military communications, surveillance, radar, and sonar. Fixed logic solutions based on gate array or standard cell technology can't offer this capability.

Feature Replacement

The board diagram in *Figure 14* illustrates how many different functions can be integrated into an FPGA to achieve significant cost savings. This example design includes a PCI master/ target controller, some HSTL translators, a cache controller, SSTL-3 translators for SDRAM, a backplane interface, some glue logic, and the clock management device. All of these functions can be integrated into the 100k gate FPGA device which costs just $10.00, almost two-thirds less than the discrete solution, with room to spare for more logic. The FPGA solution also uses less board real estate, requires less power, and provides higher reliability.

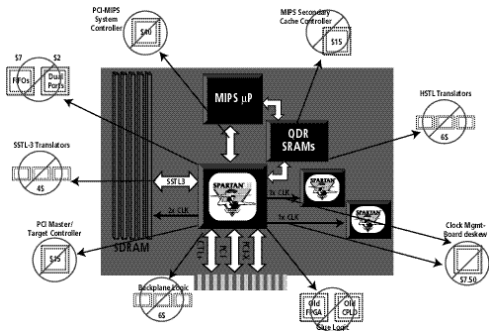

Figure 14: An Example of FPGA Value

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.

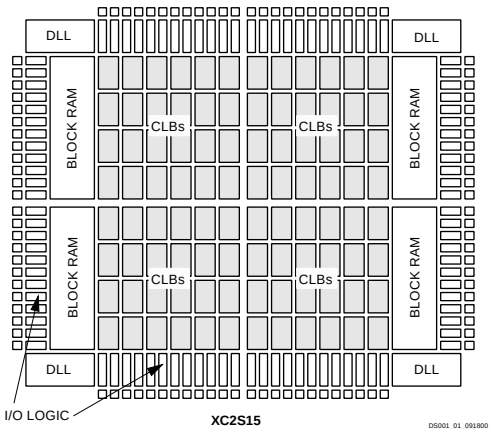

Figure 15: Basic Spartan-II Block Diagram

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.

Configurable Logic Block

The basic building block of the CLBs is the logic cell (LC). An LC includes a 4-input function gen-

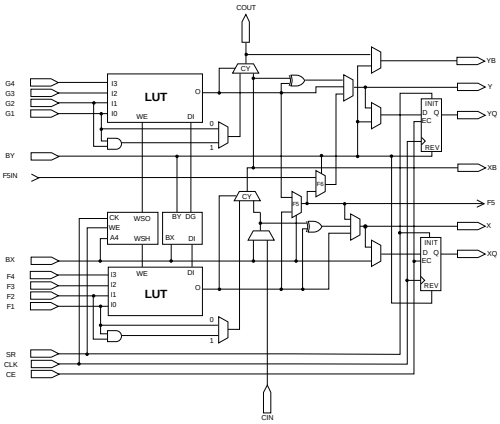

Figure 16: FPGA Slice

erator, 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.

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

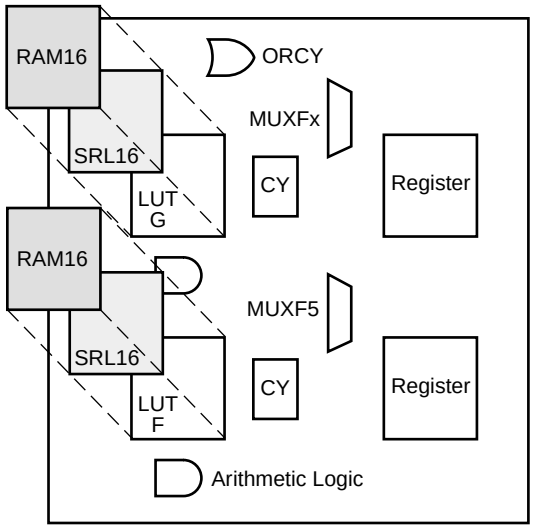

Figure 17: Slice Configuration

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*.

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 func-

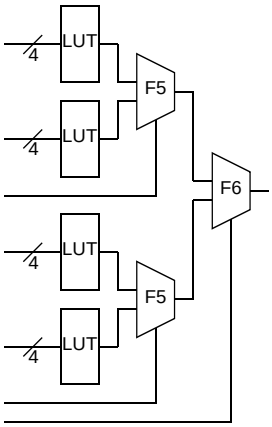

Figure 18: F5 and F6 Multiplexer

tions. 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.

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)

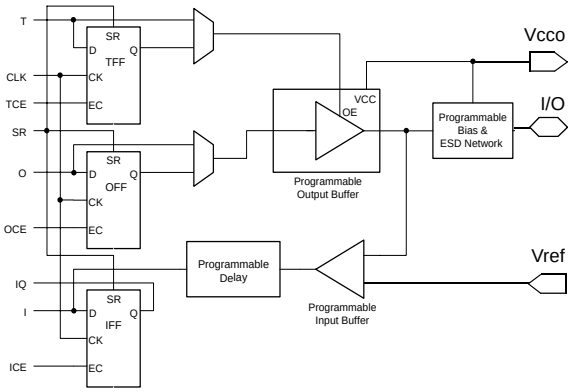

Figure 19: Input/Output Block (IOB)

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 Ref. Voltage (Vref)	Output Source Voltage (Vcco)	Board Term. 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.

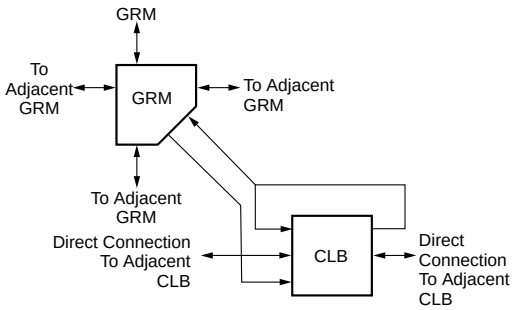

Figure 20: Local Routing

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.

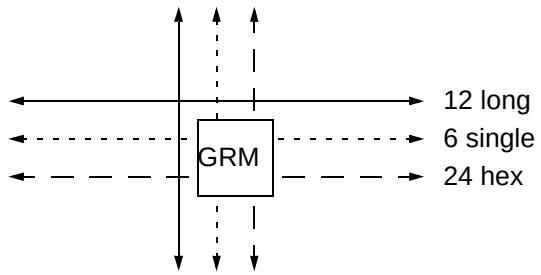

Figure 21: General Purpose Routing

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.

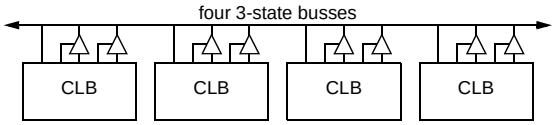

Figure 22: BUFT Connections to Dedicated Horizontal Bus Lines

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.

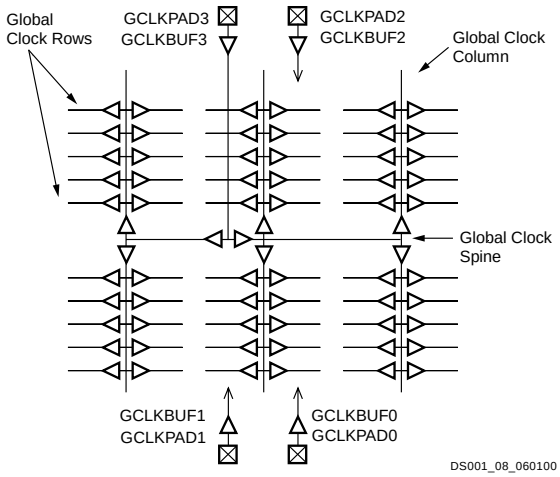

Figure 23: Global Clock Distribution Network

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.

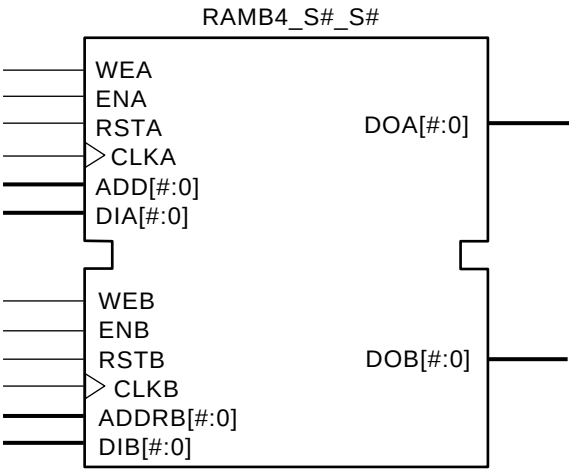

Figure 24: Dual-port Block RAM

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 array’s library.)

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.

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

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 Resources

Thus, assuming that all three LUTs and both flip-flops are utilized, a single CLB 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/CLB 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 be-

tween 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 “count gates” in their products. A better methodology for comparing the relative logic capacity of competing manufacturers’ 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
configuration file size	1,335,840
block RAM bits	57,344
bits used for logic	1,278,496
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	bits
configuration storage 544 bits/slice x 4 gates/bit	2,176
behavior 544 bits/slice x 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 “gates per gates” 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.

Description	Pin-to-Pin (w/ I/O delays) [MHz]
16-bit Address Decoder	109
32-bit Address Decoder	87
64-bit Address Decoder	74
4:1 MUX	127
8:1 MUX	113
16:1 MUX	99
32:1 MUX	88
Combinatorial (pad to LUT to pad)	139

Table 9: XC2S200-5 Pin-to-Pin Performance

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.

Description	Register-to-Register [MHz]
16-bit Address Decoder	181
32-bit Address Decoder	144
64-bit Address Decoder	124
4:1 MUX	237
8:1 MUX	230
16:1 MUX	180
32:1 MUX	155
Register to LUT to Register	285
8-bit Adder	183
16-bit Adder	175
64-bit Adder	77
64-bit Counter	88
64-bit Accumulator	72

Table 10: XC2S200-5 Register-to-Register Performance

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.

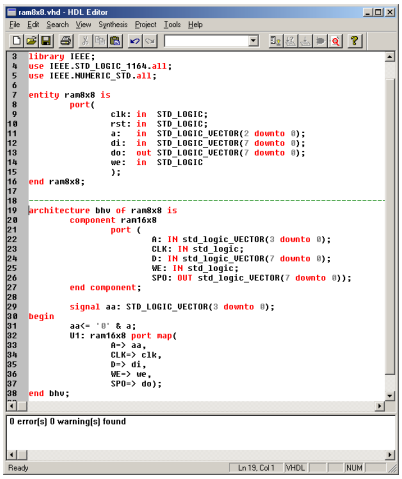

Figure 25: HDL Editor

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.

For creating structural designs, block diagram editors are available. Like FSM editors, these tools create VHDL or EDIF from the graphical

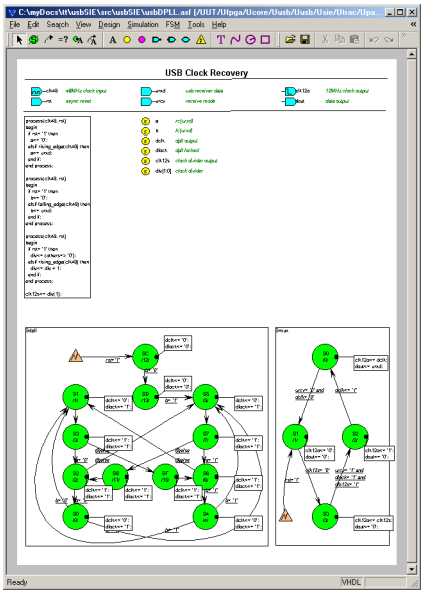

Figure 26: FSM Editor

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.

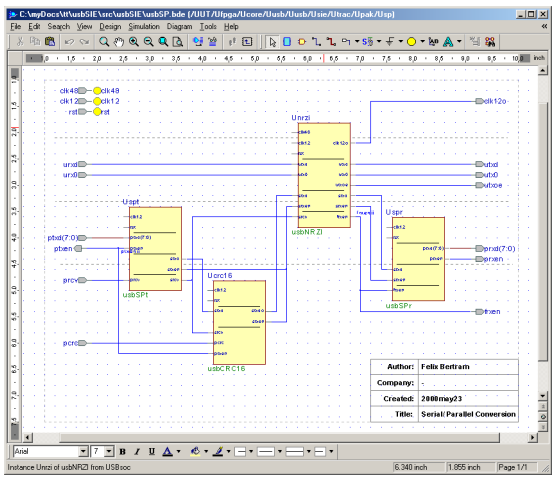

Figure 27: Block Diagram Editor

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 devel-

opment; 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.

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.

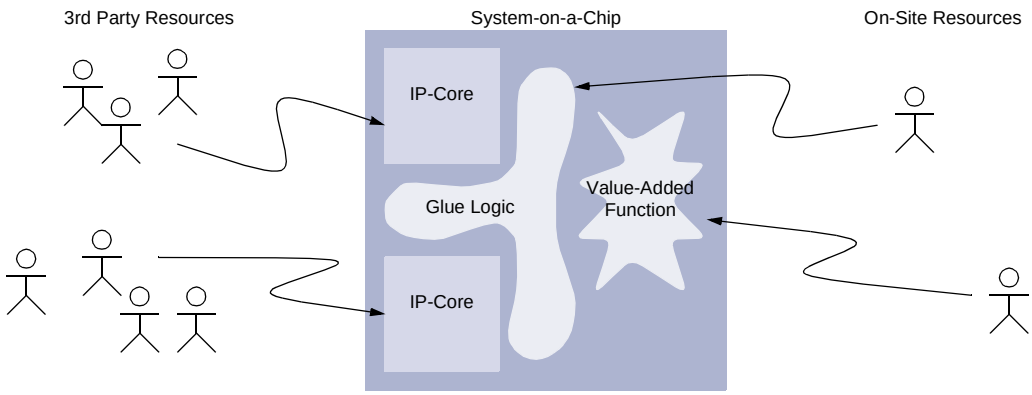

Figure 28: Building a system with IP-Cores

References

- *Physical Design for FPGAs*
  Rajeev Jayaraman
  Xilinx, Inc.
  April 16, 2001
- *FPGAs become a mainstream ASIC alternative*
  Shelly Davis, Xilinx
  Portable Design Magazine
  July 19, 2001
- *The Spartan-II Family - The Complete Package*
  WP106
  Xilinx, Inc
  January 10, 2000
- *Total Cost of Ownership: Xilinx FPGAs vs. Traditional ASIC Solutions*
  WP112
  Xilinx, Inc
  February 23, 2000
- *How Spartan Series FPGAs Compete for Gate Array Production*
  XAPP120
  Xilinx, Inc
  December 2, 1998
- *The new Spartan-II Kiss your ASIC Good-Bye*
  Xilinx, Inc
  November 13, 2000
- *Gate Count Capacity Metrics for FPGAs*
  XAPP059
  Xilinx, Inc
  February 1, 1997
- *Spartan-II 2.5V FPGA Family Preliminary Product Specification*
  Xilinx, Inc
  October 31, 2000
- *Virtex 2.5V Field Programmable Gate Arrays Preliminary Product Specification*
  Xilinx, Inc
  March 9, 2000
- *XC4000E and XC4000X Series Field Programmable Gate Arrays Product Specification*
  Xilinx, Inc
  May 14, 1999
- *FPGA Praktikum WS2000/2001*
  Kolja Sulimma
  Uni Frankfurt
  http://www.em.informatik.uni-frankfurt.de/~prak/ss01/Woche1/sld001.htm
- *What are IP Cores*
  Trenz Electronic
  August 15, 2000
- WebACE
  ASIC Cost Estimator
  Xilinx, Inc
  http://www.xilinx.com/products/webace/WebAceAppletHolder.html

Copyright

Revision History

Version	Date	Who	Description
1.0	2001nov05	FB	Created

Table 11: Revisions History