SoC Basics

Avnet Silica & Enclustra Seminar “Getting started with Xilinx Zynq SoC”
Fribourg, April 26, 2017

Oliver Bründler
Project Leader
Enclustra GmbH
SoC Basics
Avnet Silica & Enclustra Seminar “Getting started with Xilinx Zynq SoC”
Fribourg, April 26, 2017

SoC Basics
Agenda

- Introduction
- What is a SoC?
- FPGA Building Blocks
- SoC Development Process
- Code Generation
- SoC Partitioning
- Conclusions
SoC Basics
Avnet Silica & Enclustra Seminar “Getting started with Xilinx Zynq SoC”
Fribourg, April 26, 2017

SoC Basics
Agenda

- Introduction
- What is a SoC?
- FPGA Building Blocks
- SoC Development Process
- Code Generation
- SoC Partitioning
- Conclusions
Introduction - Cool new Stuff

Everybody knows, that new technologies are usually driven by application requirements. A nice example for this is, that we developed portable super-computers with a nice set position/orientation sensors as well as integrated cameras and displays (also known as Smartphone) to satisfy the urgent requirement for augmented reality applications such as Pokemon Go. Right?

Well, not exactly... This is a nice example of an application that emerged from multiple technologies that where merged together for completely different reasons. Nevertheless, someone figured out how to make use of this great feature-set of smartphones and got rich.

The merging of FPGA and CPU technology along with some standard peripherals into one SoC device was of course driven by key markets such as communications and automotive. Independently of whether you are working in these markets or not, SoCs now exist. The goal of this presentation is to give you an overview of their feature-set, so you can find out, what great applications you can realize by using them (... and possibly get rich).
Introduction – History of SoCs

After FPGAs overcame their time as pure glue-logic components, the heart of many embedded systems was a microcontroller/FPGA pair. Each of these technologies has its distinctive advantages and disadvantages and they complement each other very well. Microcontrollers allow using a standard software development processes while FPGAs allow for fastest response times and parallel processing power at the cost of more development time.

The pairing of CPU and FPGA became so popular that soft-CPU, such as the Microblaze, were developed in order to provide single-chip solutions. Even first SoCs with PowerPC hard processors were produced but they never gained much traction on the market because the technology was not yet at the level to be economically interesting.

In 2013 Xilinx introduced the Zynq-7000. It gained traction very quickly since technology was now at a level to provide a economically interesting product. The Zynq devices also solved the problem of having to decide between a slow softcore processor or the bottleneck between a fast standalone processor and the FPGA.

Encouraged by the success of the Zynq-7000 Xilinx developed the Zynq-MPSoC which is entering the market currently. Compared to the last generation the processing power is massively increased.
SoC Basics
Avnet Silica & Enclustra Seminar “Getting started with Xilinx Zynq SoC”
Fribourg, April 26, 2017

SoC Basics
Agenda

- Introduction
- **What is a SoC?**
- FPGA Building Blocks
- SoC Development Process
- Code Generation
- SoC Partitioning
- Conclusions
What is a SoC? – SoC Overview

Why does it make such a big difference if a processor and some FPGA fabric are integrated on the same chip (SoC) or if they sit side by side on a PCB?

There are several answers to that question. One is that the bandwidth between the processor system and the FPGA fabric is way than with discrete components. Another important point is that there is not just one data-bus connecting the two parts but different types of connections:

- Multiple GP ports for general purpose I/O
- Multiple HP ports for high performance data access
- ACP port for cache coherent access

Of course the SoC also has to be compared to a traditional FPGA containing a softcore processor. Here the big difference is the performance of the CPU system. No softcore CPU can compete with a full-blown multi-core ARM-A9/A63 processing system in terms of performance, cache or features such as FPU engines. In contrast to soft CPUs, the ARM architecture is a de-facto industry standard and has a huge ecosystem, so more libraries and tools are available. Additionally many standard peripherals such as SPI, CAN, UART, USB, DMA engines or memory controllers are already available. The memory controller is even easily accessible from FPGA fabric.
What is a SoC? – Key Selling Points

The key selling points of an SoC as shown on this slide allow achieving higher performance with shorter development times and less risk compared to other systems containing CPUs and programmable logic. At least one of these points is crucial for nearly every project developed.

Of course SoCs also have their drawbacks: They will never compete with pure microcontrollers in terms of price or development time. So if there is no need for FPGA logic in a given application, a SoC is not really an option.
SoC Basics

Avnet Silica & Enclustra Seminar “Getting started with Xilinx Zynq SoC”
Fribourg, April 26, 2017

SoC Basics
Agenda

- Introduction
- What is a SoC?
- FPGA Building Blocks
- SoC Development Process
- Code Generation
- SoC Partitioning
- Conclusions
**FPGA stands for**

**Field Programmable Gate Array,**

*describing an integrated circuit designed to be configured to a specific function after manufacturing.*

---

**FPGA Building Blocks - What is an FPGA?**

An FPGA consists of a regular array of different elements listed below.

- Logic blocks
- DSP blocks
- Embedded RAM blocks
- Configurable I/O blocks
- Dedicated clock management blocks
- Connected through configurable routing resources
  - Global routing (long, fast, few lines)
  - Local routing (short, abundant)
- Switch boxes
- Configuration data stored in distributed SRAM cells
FPGA Building Blocks – Logic Blocks

Look-Up Tables (LUTs)

LUTs are used to implement logic functions. A single LUT can handle an arbitrary logic function with 1 to 4 (6) inputs. More complex logic functions are implemented with LUT trees or carry chain architectures. Increased logic levels result in increased propagation delay for a logic function, while inter-LUT routing delay is dominant over LUT propagation delay.

Flip-Flops (FFs)

FFs are used for storing data and for pipelining. Pipelining (i.e. inserting a FF to break a logic path) becomes necessary if the propagation delay of a logic function is larger than the associated clock period. Since every LUT has an associated FF, pipelining in FPGAs is cheap in comparison to pipelining in an ASIC, where FFs are generally expensive.

Availability and Performance

• Up to 500 MHz clock frequency for real-world designs
• Up to more than a million configurable logic blocks per FPGA
DSP blocks are used to efficiently implement fixed-point arithmetic operations commonly used in DSP algorithms (up to 3’000 GMAC/s in today’s FPGAs).

FPGA Building Blocks - DSP Blocks

The exact implementation of DSP blocks varies between different families of FPGAs but the most important features are the same regardless of the FPGA family:

- Multiplier
- Adder/accumulator
- Pre-adder for symmetric FIR filters
- Highly pipelined
- Support for carry and adder chains

In general performance for similar FPGA families from different vendors is comparable. However, depending on project specific requirements, one or the other FPGA family can better match (e.g. more 18 x 18 multipliers vs. less 25 x 18 multipliers).

Thanks to these DSP slices, FPGAs have breathtaking DSP performance:

- ~50 ... 5’000 DSP blocks per FPGA
- Up to ~300 ... 600 MHz clock frequency
- More than 3’000 GMAC/s per FPGA

Of course these numbers are theoretical since not all resources will ever be fully utilized in a real-world design but even if a realistic 50% utilization is assumed, the numbers are astonishing.
**FPGA Building Blocks - Memory Blocks**

Exactly as for DSP Blocks, the details of the RAM block implementation depends on the FPGA family but in general they are always some kind of dual-port RAM and can be used to realize the following things:

- Simple data storage
- Shared memory
- Synchronous/asynchronous FIFOs
- Configurable large delays
- ROM (content is part of FPGA bit stream)
- Data tap / coefficient storage for FIR filters
- Program/data memory for embedded soft processors
- They are widely configurable in
  - Width/depth aspect ratio
  - Size (cascadeable)
  - Read behavior (registered/combinatorial)

Since all these memory blocks reside within the chip, exorbitant access rates can be achieved. Its not unusual that SDR designs end up using many Giga Memory Accesses per second. The number below give some indication of the availability of RAM blocks in todays FPGAs:

- ~50 ... 10'000 RAM Blocks per FPGA
- Up to ~200 ... 600 MHz clock frequency
- Granularities of 10 kb ... 36 kb
SoC Basics
Avnet Silica & Enclustra Seminar “Getting started with Xilinx Zynq SoC”
Fribourg, April 26, 2017

- 14 -

SoC Basics
Agenda

- Introduction
- What is a SoC?
- FPGA Building Blocks
- SoC Development Process
- Code Generation
- SoC Partitioning
- Conclusions

SoC Development Process
Since the development processes available for embedded software are widely known, this section focuses on the process for the development of the FPGA fabric part of a SoC design.
### SoC Development Process – HDL development

The first part of the FPGA development process is well comparable to a good software development process with strong unit-testing. Requirements are defined, then the system architecture is developed and implemented piece by piece while tests are developed in parallel. The behavioral simulation of the FPGA development process corresponds to running unit-tests in the software world.

At this points the process becomes very different from software development. During synthesis the HDL code is converted to a circuit structure. In the next step, implementation, the circuit is placed on the device used and all elements are connected (routed). Static timing analysis then checks if the logic implemented achieves all timing requirements (e.g. if the required clock speed is achieved).

If everything went well, the configuration bitstream can be generated and downloaded to the FPGA for test on hardware.

In strong contrast to software development, the actual circuit that results after synthesis and implementation must always be kept in mind during development. In general it can be said, that the abstraction level of FPGA development is lower than the abstraction level in software development by order of magnitudes.
FPGA Development Flow – System Integration

Today’s tools allow connecting all parts of the system graphically. This allows attaching bus systems, peripherals and custom logic quickly and easily to the processing system. This point is of great importance in terms of quick and easy system development since it means, that the system integration engineer does not necessarily have to be a black-belt FPGA crack. Of course some knowledge is required but tools made great advantages in this area during the past few years.
Code Generation

SoC Vendors work hard on raising the abstraction level for FPGA design. Code generation is the keyword here. We will therefore touch this topic very briefly.
**Code Generation – Simulink (System Generator)**

There are different tools to generate HDL code from Simulink.

All tools basically contain two types of Simulink blocks:

- Basic elements such as additions, multiplications and delays
- Advanced elements such as FIR-filters, encoders/decoders, FFTs

Advanced elements usually can be parametrized and lead to good simulation performance. They also strongly reduce the time required for design entry and lead to efficient FPGA implementations. If the functionality required is not available as library element, it must be built from scratch by using basic elements. At this point the effort for design entry increases, simulations run slower and more effort is required to achieve an efficient FPGA implementation. As a result, it is important to select a tool that supports large parts of the functionality required as library elements.

Another point worth analyzing before selecting a tool is the flexibility of the library components. If you are at the very limit of the supported ranges, slight changes can force you to remove a pre-designed library element and design the functionality from scratch using basic elements. If your tool for example only support FFTs of up to 16k and specification changes require a 32k FFT, the FFT must be designed from scratch which is a huge effort and of course also bears risk.
**Code Generation - C/C++ (HLS)**

First of all: Despite marketing claims it, code generation from C/C++ is not yet at the level to produce efficient FPGA implementations from code written by a software developer with no FPGA knowledge. Maybe it will never be. What it can do already today is improving the productivity of FPGA firmware developers.

In contrast to code entry using Simulink, code generation from C/C++ does not rely so much on pre-designed library elements. Some are available but most structures required can be well described with standard software constructs such as loops and branches. So this approach is more flexible in general.

On the other hand, C/C++ describes algorithms in sequential fashion while FPGA logic works in parallel. So the conversions done by the tool are more complex than in Simulink which natively fits the parallel architecture of FPGAs. To help the tools, the code must be strongly constrained for an efficient FPGA implementation and it is generally more difficult to get the tool to implement exactly what you want.
SoC Basics

Avnet Silica & Enclustra Seminar “Getting started with Xilinx Zynq SoC”
Fribourg, April 26, 2017

- 20 -

SoC Basics

Agenda

- Introduction
- What is a SoC?
- FPGA Building Blocks
- SoC Development Process
- Code Generation
- SoC Partitioning
- Conclusions
SoC Basics

SoC Basics
SoC Partitioning

A distinctive feature of the river Pegnitz...

SoC Partitioning – Shorter vs. Faster

In traditional software development, faster always translates to less clock cycles and therefore less operations. With SoCs this is not always true. It may be more efficient to choose an algorithm that requires more operations but is well suited for parallelization and implementation in FPGA fabric. For example recursive algorithms often lead to very elegant solutions in software but in a SoC an iterative algorithm requiring more operations can lead to faster runtimes thanks to easy parallel implementation in FPGA fabric.

A nice analogy to this the river Pegnitz in Bavaria. The river is split and one part flows straight through a tunnel below the mountain «Wasserberg». The other part of the river flows around the mountain in a huge loop. Even though the distance through the tunnel is only a fraction of the loop-size, experiments have shown, that it takes the water about four time as long to flow through the tunnel. So nature teaches us, that the shortest path (least operations) can be slower (more execution time) than a longer path with a wider river-bed (parallel execution in FPGA fabric).
**SoC Basics**

**Avnet Silica & Enclustra Seminar “Getting started with Xilinx Zynq SoC”**

Fribourg, April 26, 2017

---

### SoC Basics

#### SoC Partitioning

**Software**
- Instructions are processed sequentially
- Most algorithms are natively described sequentially
- Floating-point is nearly as quick as fixed-point
- Readable to almost every engineer
- Achieving target speed
- Less instructions per execution!

---

**HDL (Logic)**
- Everything happens in parallel
- Algorithms must be converted to a circuit structure
- Fixed-point is way more efficient
- Hard to read and understand
- Achieving target speed
  - Increasing clock speed?
  - More operations in one clock cycle?
  - More parallelism?
  - Latency vs. Throughput

---

**SoC Partitioning - HDL is no Software**

People are often heard complaining about HDL (hardware description language) development for FPGAs/SoCs being so much less intuitive than writing software for a conventional processor. Similar complaints are heard for code generation. On one hand this is really true, on the other hand it is comparing apples to ... no, not pears ... more something like elephants. It is comparing software to circuit-hardware. These two things have completely different levels of complexity and abstraction but also of capability and efficiency. The fact that HDL code for FPGAs can be viewed and edited in a text-editor does not mean it is software.

In general, if it can be done in software, all the effort and complexity of the FPGA should be avoided. But what if it cannot be done in software? At this point it is not worth claiming that the software flow is more easy since it obviously is not an option.

All these points make FPGA/SoC design looking a bit daunting but it suddenly looks more attractive when you start thinking about getting 30 GMAC/s filter performance and beyond from an off-the-shelf device.

In comparison to traditional FPGA devices, SoCs easily allow realizing only the most performance-relevant algorithms in HDL and describe everything else easily in traditional software.
SoC Basics
Avnet Silica & Enclustra Seminar “Getting started with Xilinx Zynq SoC”
Fribourg, April 26, 2017

SoC Basics
SoC Partitioning

Communication between CPU and FPGA fabric
- Types of communication
  - Data streams
  - Memory mapped access
  - Event signaling
- Performance characteristics
  - Bandwidth
  - Latency
  - CPU Load
  - Flow-control

SoC Partitioning - Communication between CPU and FPGA Fabric

When partitioning a SoC system and deciding which tasks should be done in software and which ones are implemented in FPGA fabric, the communication is a key point. In general there are three communication mechanisms required in most systems:

- Data streams: Fast transfer of huge amounts of data. Often these data-streams are continuous. Good examples are video or audio streams.

- Memory mapped access: Access to configuration or status registers of the logic implemented in FPGA fabric

- Event signaling: The FPGA fabric must be able to signal events to the CPU system. In general this means IRQs are required.

Depending on the partitioning and the requirements of the system, one or the other communication mechanism can be more important. The same applies to different performance characteristics. Depending on the system architecture for example the optimal trade-off between bandwidth and latency may be at a completely different point.

For high performance systems an exact analysis of the requirements and different architectures must be executed in order to achieve optimal performance.
SoC Basics
Avnet Silica & Enclustra Seminar “Getting started with Xilinx Zynq SoC”
Fribourg, April 26, 2017

SoC Basics
Agenda

- Introduction
- What is a SoC?
- FPGA Building Blocks
- SoC Development Process
- Code Generation
- SoC Partitioning
- Conclusions
SoCs have great power ...  
... use it wisely!

- SoCs provide huge amounts of processing power
- System partitioning is important
- Use FPGA fabric only where required/beneficial
- The development effort for FPGA fabric is higher than for software
- Know-how from both domains (SW/FPGA) is required

Conclusions – Use SoC Power wisely

Of course every engineer is enthusiastic about all the processing power one can get out of a SoC. On the other hand, implementing fast algorithms in FPGA logic to achieve this requires quite some effort. As a result, FPGA fabric should only be used for the most performance critical parts of a system and not just for everything. It is not easy to take the right decisions about what should be implemented in which part of the SoC. The main learning from the many projects at Enclustra is, that the engineers responsible for system design must have in-depth know-how from in domains.
**Conclusions**  
**HDL Code Generation vs. HDL Design Entry**

Choose the appropriate tool for every task in your FPGA design project.

- HDL code generation is generally a good choice for processing blocks.
- HDL design entry generally is (still) the way to go for communication interfaces and logic-heavy blocks.
- Choosing the best-fitting tool is an important decision – do feasibility studies.
- Tools do not replace technology specific know-how.

---

**Conclusions – HDL Code Generation vs. HDL Design Entry**

HDL code generation is very well suited for implementing algorithms, while the most efficient way to describe interfaces and control logic still is direct HDL entry.

Choosing the appropriate tool for each part of a system is very important since it can save or waste big amounts of development effort. Often it is a good decision to first do some feasibility studies to find out what tool fits the project requirements best and where its limitations are. This not only helps selecting the right tool, it also allows doing reasonable estimations of the development effort and schedule.

Despite code generation tools can greatly reduce the time a FPGA firmware engineer requires to implement an algorithm, they cannot replace the technology specific know-how of the engineer. Ignoring this limitation of the tools can easily lead to budget-overruns as well as inefficient or even non-functional circuits.
Questions?

Upcoming Events:

- FPGA Kongress
  July 11–13, 2017
  Munich
  Germany

Oliver Bründler
Project Leader
oliver.brundlier@enclustra.com
+41 43 343 39 48

Quarterly newsletter:
subscribe@enclustra.com