CHAPTER 15

Introducing System-on-a-Programmable-Chip

A small SOPC-based aircraft autopilot system that contains an FPGA with a Nios processor core, a DSP processor, and memory is seen above. The bottom sensor board contains a GPS receiver, an A/D converter, MEMS gyros and accelerometers for all three axes, an airspeed sensor, and an altitude sensor. Photograph ©2004 courtesy of Henrik Christophersen, Georgia Institute of Technology Unmanned Aerial Research Facility.

15 Introducing System-on-a-Programmable-Chip[5]

A new technology has emerged that enables designers to utilize a large FPGA that contains both memory and logic elements along with an intellectual property (IP) processor core to implement a computer and custom hardware for system-on-a-chip (SOC) applications. This new approach has been termed system-on-a-programmable-chip (SOPC).

15.1 Processor Cores

Processor cores can be classified as either "hard" or "soft." This designation refers to the flexibility/configurability of the core. Hard cores are less configurable; however, they tend to have higher performance characteristics than soft cores.

Hard processor cores use an embedded processor core (in dedicated silicon) in addition to the FPGA's normal logic elements. Hard processor cores added to an FPGA are a hybrid approach, offering performance trade-offs that fall somewhere between a traditional ASIC and an FPGA; they are available from several manufacturers with a number of different processor flavors. For example, Altera offers an ARM processor core embedded in its APEX 20KE family of FPGAs that is marketed as an Excalibur™ device. Xilinx's Virtex-II Pro family of FPGAs include up to four PowerPC processor cores on-chip. Cypress Semiconductor also offers a variation of the SOPC system. Cypress's Programmable-System-on-a-Chip (PSoC™) is formed on an M8C processor core with configurable logic blocks designed to implement the peripheral interfaces, which include analog-to-digital converters, digital-to-analog converters, timers, counters, and UARTs.[6]

Soft cores, such as Altera's Nios II and Xilinx's MicroBlaze processors, use existing programmable logic elements from the FPGA to implement the processor logic. As seen in Table 15.1, soft-core processors can be very feature-rich and flexible, often allowing the designer to specify the memory width, the ALU functionality, number and types of peripherals, and memory address space parameters at compile time. However, such flexibility comes at a cost. Soft cores have slower clock rates and use more power than an equivalent hard processor core.

With current pricing on large FPGAs, the addition of a soft processor core costs as little as thirty-five cents based on the logic elements it requires. The remainder of the FPGA's logic elements can be used to build application-specific system hardware. Traditional system-on-a-chip devices (ASICs and custom VLSI ICs) still offer higher performance, but they also have large

[5] Portions reprinted, with permission, from T. S. Hall and J. O. Hamblen, "System-on-a-Programmable-Chip Development Platforms in the Classroom," *IEEE Transactions on Education*, vol. 47, no. 4, pp. 502-507, Nov. 2004. © 2004 IEEE.

[6] D. Seguine, "Just add sensor - integrating analog and digital signal conditioning in a programmable system on chip," *Proceedings of IEEE Sensors*, vol. 1, pp. 665–668, 2002.
M. Mar, B. Sullam, and E. Blom, "An architecture for a configurable mixed-signal device," *IEEE J. Solid-State Circuits*, vol. 38, pp. 565–568, Mar. 2003.

development costs and longer turnaround times.[7] For projects requiring a hardware implementation, the FPGA-based SOPC approach is easier, faster, and more economical in low to medium quantity production.

Table 15.1 Features of Commercial Soft Processor Cores for FPGAs

Feature	Nios II 5.0	MicroBlaze 4.0
Datapath	32 bits	32 bits
Pipeline Stages	1-6	3
Frequency	Up to 200 MHz[8]	Up to 200 MHz[4]
Gate Count	26,000 – 72,000	30,000 – 60,000
Register File	32 general purpose & 6 special purpose	32 general purpose & 32 special purpose
Instruction Word	32 bits	32 bits
Instruction Cache	Optional	Optional
Hardware Multiply & Divide	Optional	Optional
Hardware Floating Point	Third Party	Optional

Typically, additional software tools are provided along with each processor core to support SOPC development. A special CAD tool specific to each soft processor core is used to configure processor options, which can include register file size, hardware multiply and divide, floating point hardware, interrupts, and I/O hardware. This tool outputs an HDL synthesis model of the processor core in VHDL or Verilog. In addition to the processor, other system logic is added and the resulting design is synthesized using a standard FPGA synthesis CAD tool. The embedded application program (software) for the processor is typically written in C or C++ and compiled using a customized compiler provided with the processor core tools.

15.2 SOPC Design Flow

The traditional flow of commercial CAD tools typically follows a path from hardware description language (HDL) or schematic design entry through synthesis and place and route tools to the programming of the FPGA. FPGA manufacturers provide CAD tools such as Altera's Quartus II and Xilinx's ISE software, which step the designer through this process. As shown in Fig. 15.1, the addition of a processor core and the tools associated with it are a superset of the traditional tools. The standard synthesis, place and route, and programming

[7] H. Chang et al., *Surviving the SOC Revolution a Guide to Platform-Based Design*. Norwell, MA: Kluwer, 1999.

[8] This speed is not achievable on all devices for either processor core. Some FPGAs may limit the maximum frequency to as low as 50 MHz.

functionality is still needed, and in the case of both Altera and Xilinx, the same CAD tools (Quartus II or ISE) are used to implement these blocks.

Processor Core Configuration Tools

Today, a number of pre-defined processor cores are available from various sources. GPL-licensed public processor cores can be found on the web (i.e., www.opencores.org and www.leox.org), while companies such as Altera (Nios II processor), Xilinx (MicroBlaze processor), and Tensilica (Xtensa processor) provide their processors and/or development tools for a fee.

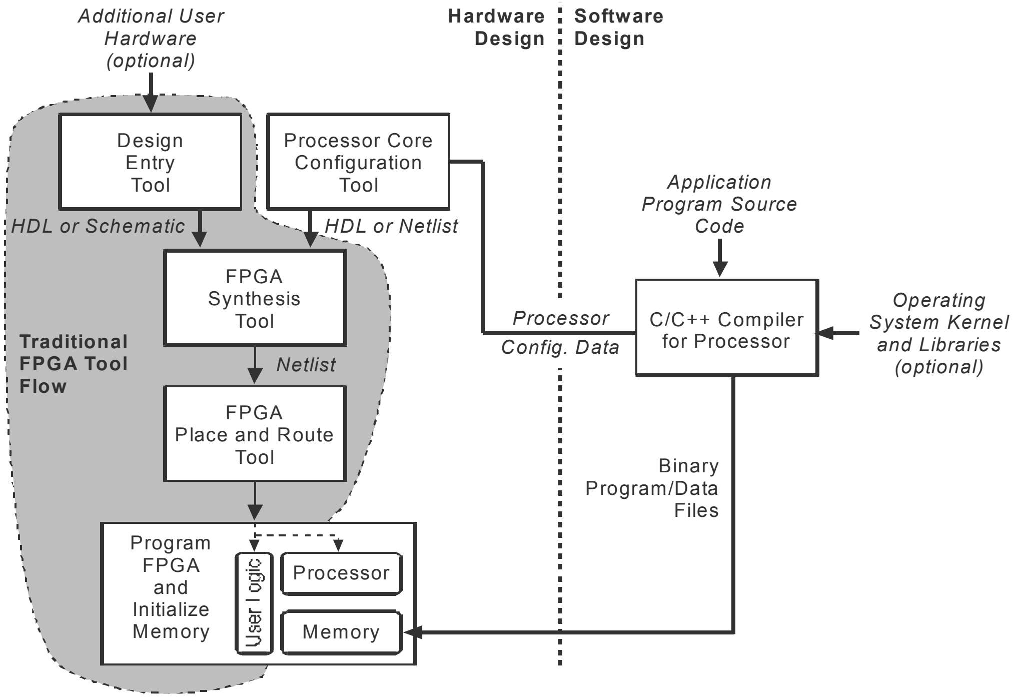
Additional User Hardware (optional)
Hardware Design
Software Design
Design Entry Tool
Processor Core Configuration Tool
Application Program Source Code
HDL or Schematic
HDL or Netlist
FPGA Synthesis Tool
Processor Config. Data
C/C++ Compiler for Processor
Operating System Kernel and Libraries (optional)
Traditional FPGA Tool Flow
Netlist
FPGA Place and Route Tool
Binary Program/Data Files
Program FPGA and Initialize Memory
User Logic
Processor
Memory

Figure 15.1 The CAD tool flow for SOPC design is comprised of the traditional design process for FPGA-based systems with the addition of the Processor Core Configuration Tool and software design tools. In this figure, the program and data memory is assumed to be on-chip for simplicity.

Processor cores provided by FPGA manufacturers are typically manually optimized for the specific FPGA family being used, and as such, are more efficiently implemented on the FPGA than a student-designed core (especially given the time and resource constraints of most class projects). The simple computer and MIPS processor cores developed earlier in this book were designed to be easy for students to understand and were not optimized for any particular FPGA. Additionally, FPGA companies provide extensive support tools to ease the customization and use of their cores, including high-level compilers targeted at the custom cores.

In the case of Altera and Xilinx, the Processor Core Configuration Tool block shown in Fig. 15.1 is realized in a user-friendly GUI interface that allows the designer to customize the processor for a particular project. The configurable parameters can include the datapath width, memory, address space, and peripherals (including arbitrarily defined general-purpose I/O, UARTs, Ethernet controllers, memory controllers, etc.). Once the processor parameters are specified in the GUI interface, the processor core is generated in the form of an HDL file (in Altera) or a netlist file (in Xilinx). This file can then be included within a traditional HDL or schematic design using the standard CAD tools. Specific pin assignments and additional user logic can be included at this point like any other FPGA design. Next, the full hardware design (processor core and any additional user logic) is compiled (synthesis, place and route, etc.), and the FPGA can be programmed with the resulting file using the standard tools. The hardware design is complete, and the FPGA logic has been determined.

High-level Compiler for Processor Core

As shown on the right side of Fig. 15.1, the next step is to write and compile the software that will be executed on the soft processor core. When the Processor Core Configuration Tool generates the HDL or netlist files, it also creates a number of library files and their associated C header files that are customized for the specific processor core generated. A C/C++ compiler targeted at this processor is also provided. The designer can then program stand alone programs to run on the processor. Optionally, the designer can compile code for an operating system targeted for the processor core. Several operating systems for the Nios II are available from third-party vendors along with the community supported open source eCos (www.niosforum.com).

15.3 Initializing Memory

Once a program/data binary file has been generated, it must be loaded into the processor's program and data memories. This loading can be done several ways depending on the memory configuration of the processor at hand.

On-chip Memory

If the application program is small and can fit into the memory blocks available on the FPGA, then the program can be initialized in the memory when the hardware configuration is programmed. This initialization is done through the standard FPGA tools, such as Altera's Quartus II software or Xilinx's ISE software. However, on-chip memory is typically very limited, and this solution is not usually an option.

Bootloader

In a prototyping environment, the application program will most likely be modified a number of times before the final program is complete. In this case, the ability to download the application code from a PC to the memory on an

FPGA board must be provided. This functionality, typically called a "bootloader" or "boot monitor," can be implemented in either software or hardware.

A software bootloader is comprised of code that is loaded into an on-chip memory and starts running on power up. This program is small enough (1-2 KB) to fit in most on-chip memories, and its primary function is to receive a program binary file over the serial port (or other interface), load it into external memory, and then start the new code executing. In this way, a new program can be stored into external memory (SRAM, SDRAM, Flash memory, etc.) by downloading it over the serial or JTAG port (or other interface) on the fly without having to reload the FPGA's hardware configuration. Xilinx provides a boot monitor for their MicroBlaze soft-core processor that includes the ability to download a program binary over the serial port (or other interface), store it in memory, and start the code executing. They also provide a more enhanced version called XMDstub that adds debugging capabilities. Altera's legacy Nios processors included a bootloader called GERMS. The Nios II processor still includes limited support for the GERMS monitor; however, a hardware bootloader is the preferred solution in Nios II.

A hardware bootloader provides functionality very similar to a software bootloader; however, it is implemented in dedicated logic within the processor core. Typically, the processor will be paused or stalled upon power up and the hardware bootloader will have direct access to memory or the memory registers in the processor's datapath. The bootloader hardware can start and stop the processor and can control the downloading of a program over the JTAG or serial interface to the desired memory locations. Altera's hardware bootloader is a part of the JTAG debug module, which resides within the Nios II processor. This logic uses the JTAG interface with the PC to receive the execution code, and it then writes the code to the appropriate memory. Finally, the bootloader hardware overwrites the processor's program counter with the start address of the code just downloaded and releases the pause bit to allow the processor to begin executing the downloaded code.

External Non-volatile Storage

The application program code can be stored on an external EEPROM, Flash memory, or other form of non-volatile memory. The application program can either be pre-programmed in the external memory module (for a production run) or a bootloader program can be used to store the application program in non-volatile storage. For low-speed applications, the code can be executed directly from the external memory. However, if high-speed functionality is required, then a designer could use three memories as shown in Fig. 15.2. In this scheme, the on-chip memory is initialized with a bootloader, which handles the movement of the application program between the memories. (On-chip memory is replaced with a hardware bootloader on some systems including the Nios II processor.)

The fast, volatile memory (i.e., SDRAM) is used to store the application program during execution. Finally, the slower, non-volatile memory (i.e., Flash or EEPROM) is used for the permanent storage of the application program. The bootloader program can be modified to initialize the system, retrieve a program from non-volatile memory, store it in the faster, volatile memory, and then start it executing from the faster memory. This scheme provides the advantages of permanent storage, fast execution, and the ability to modify the application program when needed. Of course, it comes at the expense of having additional memory.

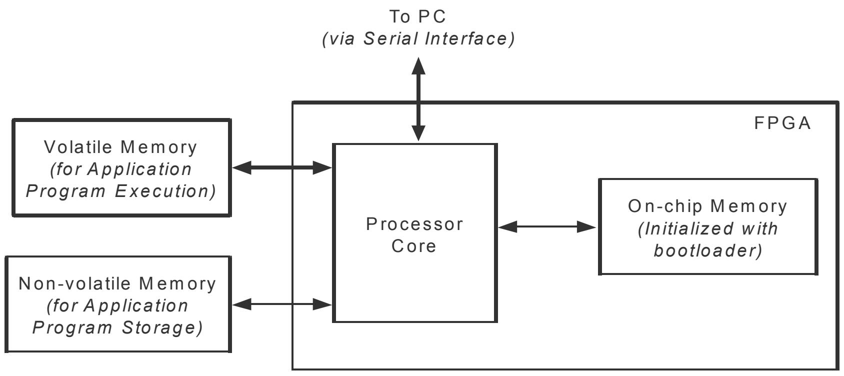
To PC
(via Serial Interface)
FPGA
Volatile Memory
(for Application Program Execution)
Processor Core
On-chip Memory
(Initialized with bootloader)
Non-volatile Memory
(for Application Program Storage)

Figure 15.2 This arrangement of on-chip and external memories provides flexibility and performance to an SOPC system.

15.4 SOPC Design versus Traditional Design Modalities

The traditional design modalities are ASIC and fixed-processor design. SOPC design has advantages and disadvantages to both of these alternatives as highlighted in Table 15.2. The strengths of SOPC design are a reconfigurable, flexible nature and the short development cycle. However, the trade offs include lower maximum performance, higher unit costs in production, and relatively high power consumption.

The benefit of having a flexible hardware infrastructure can not be overestimated. In many new designs, features and specifications are modified throughout the design cycle. For example, marketing may detect a shift in demand requiring additional features (e.g., demand drops for cell phones without cameras), a protocol or specification is updated (e.g., USB 2.0 is introduced), or the customer requests an additional feature. In traditional design modalities (including ASIC and fixed-processor designs), these changes can dramatically effect the ASIC design, processor selection, and/or printed circuit board design. Since the hardware architecture is often settled upon early in the design cycle, making changes to the hardware design later in the cycle will typically result in delaying a product's release and increasing its cost.

Flexible infrastructure can also be beneficial in extending the life (and thus reducing the cost) of a product's hardware. With flexible, reconfigurable logic, often a single printed circuit board can be designed that can be used in multiple product lines and in multiple generations/versions of a single product. Using reconfigurable logic as the heart of a design, allows it to be reprogrammed to implement a wide range of systems and designs. Extending the life of a board design even one generation can result in significant savings and can largely offset the increased per-unit expense of reconfigurable devices.

Table 15.2 Comparing SOPC, ASIC, and Fixed-Processor Design Modalities

Feature	SOPC	ASIC	Fixed-Processor
S/W Flexibility	●	●	●
H/W Flexibility	●	○	○
Reconfigurability	●	○	○
Development Time/Cost	●	○	●
Peripheral Equipment Costs	●	●	○
Performance	◉	●	●
Production Cost	◉	●[9]	●
Power Efficiency	○	●	●

Legend: ● – Good; ◉ – Moderate; ○ – Poor

The SOPC approach is ideal for student projects. SOPC boards can be used and reused to support an extremely wide range of student projects at a very low cost. ASIC development times are too long and mask setup fees are too high to be considered for general student projects. A fixed-processor option will often require additional hardware and perhaps even a new printed circuit board (PCB) design for each application. Given the complexity of today's multilayer surface mount PCB designs, it is highly unlikely that students would have sufficient time and funds to develop a new printed circuit board for a design project.

15.5 An Example SOPC Design

The SOPC-based autopilot system seen in the photograph on the first page of this chapter and the sensor board that mounts below it (described earlier in Section 13.5) makes an interesting case study in SOPC design[10]. The autopilot system continuously reads in sensor data that indicates attitude, altitude, speed, and location. It then uses this data to solve the control system motion equations for the aircraft and then outputs updated signals to control the aircraft.

[9] In very large quantities.

[10] Henrik B. Christophersen; R. W. Pickell; James C. Neidhoefer; Adrian A. Koller; Suresh K. Kannan; Eric N. Johnson, "A Compact Guidance, Navigation, and Control System for Unmanned Aerial Vehicles", *Journal of Aerospace Computing, Information, and Communication,* pp.,1542-9423, vol.3 no.5.

The flexibility of SOPC design allows the use of FPGA's logic elements to interface to a wide range of sensors without the need for additional I/O support chips that would be needed if a more traditional fixed-processor option was used. This results in an extremely small and low weight PCB design. An ASIC could be used instead of the FPGA, but the small production quantities needed for this system do not justify the greatly increased development time and cost needed for an ASIC.

Different types of aircraft also require markedly different I/O standards for the control outputs. Some aircraft controls use serial interfaces, while others use PWM or even parallel I/O. Here again, the flexibility of using the FPGA's logic elements to implement the I/O interface is of great benefit. By varying the logic in the interface peripherals, the same programmable processor core and PCB board can be used to support a wide range of aircraft without any hardware changes to the PCB.

The autopilot system requires intensive floating-point calculations to solve the complex control system equations for the aircraft. While it would be possible to perform floating-point calculations using a larger FPGA, the decision was made to use a fixed-processor DSP chip for the intensive floating-point calculations. By offloading the algorithmic computations to a fixed processor, the Nios II processor is primarily acting as an intelligent I/O processor for the system. This partitioning of the system between a fixed-processor DSP and soft-core processor results in higher computational performance than using just an FPGA (with floating-point hardware logic) and higher interface flexibility than using just a fixed processor in the system. However, new generations of FPGAs with DSP features such as hardware multipliers and floating-point IP cores are currently changing this set of design tradeoffs.

15.6 Hardware/Software Design Alternatives

The SOPC-based approach offers new design space alternatives. It is possible to explore design options that use software, dedicated hardware, or a mixture of both. Hardware solutions offer faster computations, but offer less flexibility and may require a larger FPGA. Implementation of solutions using software is easier to design for more complicated algorithms.

It is also possible to consider a combination of both approaches. Some processor cores allow the user to add custom instructions. If an application program requires the same calculation repeatedly in loops, adding a custom instruction using extra hardware to accelerate the inner loop code can greatly speed up the application.

15.7 For additional information

This chapter has provided a brief overview of SOPC systems and designs. More information about SOPC systems can be found from manufacturers such as Altera, Xilinx, Cypress Semiconductor, Stretch Incorporated, and Tensilica. SOPC systems are an active area of research. Publications of interest include the following:

- T. S. Hall and J. O. Hamblen, "System-on-a-Programmable-Chip Development Platforms in the Classroom," *IEEE Transactions on Education*, vol. 47, no. 4, pp. 502-507, Nov. 2004.
- C. Snyder, "FPGA processor cores get serious," in *Cahners Microprocessor Report*, http://www.MPRonline.com/, Sept. 2000.
- D. Seguine, "Just add sensor - integrating analog and digital signal conditioning in a programmable system on chip," *Proceedings of IEEE Sensors*, vol. 1, pp. 665–668, 2002.
- M. Mar, B. Sullam, and E. Blom, "An architecture for a configurable mixed-signal device," *IEEE J. Solid-State Circuits*, vol. 38, pp. 565–568, Mar. 2003.
- H. Chang and et. al., *Surviving the SOC Revolution A Guide to Platform-Based Design*. Kluwer Academic Publishers, 1999.
- J. Fisher, P. Faraboschi, and C. Young, *Embedded Computing : A VLIW Approach to Architecture, Compilers and Tools,* Morgan Kaufmann, 2004.
- A. Jerraya, H. Tenhunen, and W. Wolf, "Multiprocessor Systems-on-Chips," *IEEE Computer*, vol. 38, no. 7, pp. 36-41, July 2005.
- S. Liebson and J. Kim, "Configurable Processors: A New Era in Chip Design," *IEEE Computer*, vol. 38, no. 7, pp.51-59, July 2005.

15.8 Laboratory Exercises

1. Compare the instruction formats and the instruction set of the Nios II processor to the MIPS processor from Chapter 14. Information on the Nios II instruction set architecture is available at Altera's website (www.altera.com) in the Nios II Processor Reference Handbook.

2. A system needs a processor to run a control program, but the application also needs to compute FFTs at a somewhat high data rate. FFTs require a large number of multiply and add operations on an array in nested loops. What SOPC hardware/software design tradeoffs would you need to consider? Justify your answer.

3. List several types of products that could likely take advantage of the SOPC design approach. Explain your reasoning.

4. Compare the memory read access time of the FPGA's Flash and SRAM memory chips. Information can be found in each chip's datasheet. If the processor did not have an instruction cache, how much faster could a program read instructions from SRAM?

5. You are asked to specify the memory types and sizes for an SOPC design that will execute a program with a 60 KB length or footprint. During execution, the program requires 16 KB of data memory for the stack and heap. If the SOPC hardware mandates a single memory (for program and data memory), select the type and size of memory. Perform an online search to find a manufacturer and model number for the memory you

selected. You may have to modify your initial selection based on availability and cost of various memories. Justify your selection considering cost, specification, performance, and availability. Don't forget that you need non-volatile memory to boot the system.

6. Given the SOPC system outlined in Problem 5, select the type and size of memory needed for this system when program and data memory are separate. Justify your selection considering cost, specification, performance, and availability. Compare the single memory option from Problem 5 with the dual-memory option from this problem. Which memory configuration is preferable? Justify your answer.

7. There are a number of different non-volatile memory technologies available to SOPC designers. For a system with a 256 KB code footprint, compare the cost, reprogrammability, configuration time, access time (reading only), and longevity for PROM, EEPROM, and Flash memories.