A Simple SoC Platform for the Integrated Computer Engineering Lab Framework

Dr. Pong P. Chu, Cleveland State University

Dr. Chu is Associate Professor in the Department of Electrical Engineering and Computer Science. He has taught undergraduate and graduate digital systems and computer architecture courses for more than two decades, and he has received multiple instructional grants from the National Science Foundation and authored six textbooks in this area.

Dr. Chansu Yu, Cleveland State University

Chansu Yu received the B.S. and M.S. degrees in electrical engineering from Seoul National University, Korea, in 1982 and 1984, respectively, and the Ph.D. degree in computer engineering from the Pennsylvania State University in 1994. He is currently Professor and Chair of the Department of Electrical Engineering and Computer Science at the Cleveland State University in Cleveland, Ohio. Before joining the CSU, he was on the research staff at LG Electronics, Inc. He has authored/coauthored more than 120 technical papers and numerous book chapters in the areas of mobile computing, performance evaluation, and parallel systems. His research has been supported by both industry and government including National Science Foundation. Dr. Yu is a member of the IEEE, IEEE Computer Society, ACM, and ASEE.

Dr. Karla R. Hamlen, Cleveland State University

Dr. Karla Hamlen is an Associate Professor of Educational Research in the Department of Curriculum and Foundations. She specializes in educational research relating to both formal and informal entertainment technology use among students.

A Simple SoC (System on a Chip) Platform for the Integrated Computer Engineering Lab Framework

1. Introduction

A "spiral" lab framework is developed for the computer engineering curriculum. It is motivated by a study from the Carnegie Foundation [6], which recommends a "spiral model" to enhance the integration skills and to provide more effective learning experiences:

"*… the ideal learning trajectory is a spiral, with all components revisited at increasing levels of sophistication and interconnection. Learning in one area supports learning in another*."

Instead of treating the labs as the adjuncts that follow the learning of the theories and presenting them in a limited "component context," we use them as a cohesive framework to connect and integrate the individual courses in the computer engineering curriculum, including freshman engineering, introductory digital systems, advanced digital systems, computer organization, embedded systems, hardware-software co-design, and senior capstone design [9]. This goal of the lab framework is to make students aware of the big picture, help them to connect the individual subjects, and apply and integrate the previous learning in a new context. The framework consists of a series of sound- and video-theme based lab experiments and projects [7,8], whose complexities and abstraction levels gradually grow with the progress of curriculum.

The lab framework covers both hardware and software aspects of computer systems and the experiments are done in the SoC (system on a chip) context [14], in which a system contains a general-purpose processor for "housekeeping" tasks and hardware accelerators for computation-intensive tasks. The commercial SoC platforms are too complex and use the proprietary and encrypted bus interconnect and IP (intellectual property) cores. A simple, open, and vendor-neutral SoC platform is developed to support the lab framework [10,11]. The SoC platform can support the audio- and video-theme experiments and projects. It demonstrates many key design concepts and can be used to construct custom and functional embedded systems. Earlier articles covered the development of the sound theme [7] and video theme [8]. This article describes the SoC platform and discusses the hardware and software organization.

The remaining article is organized as follows: Section 2 discusses the design principles of the platform; Sections 3 and 4 highlight its hardware organization and software organization, respectively; Section 5 describes the implementation; and Section 6 summarizes the work.

2. Design principle of a "learning" SoC platform

The hardware acceleration is a key application of FPGA devices and each FPGA vendor has its own SoC development platform [15], which includes the software tools and a large collection of predesigned IP cores. However, these commercial platforms are inadequate for the proposed lab framework for several reasons:

- Commercial SoC platforms are intended for large high-end systems and their development flows are very involved. A significant amount of time will be spent on learning the software tool rather than studying and doing the design.
- Commercial IP cores are usually provided as black boxes with obfuscated source codes.
- The interface protocols are intended for high speed transfer and require special IP cores.

- The software drivers and libraries are difficult to comprehend.
- Because of the proprietary IP cores and software libraries, the design is frequently "locked" to an individual vendor. Thus, learning is tied to a particular platform and the developed IP cores are not portable.

To overcome the limitation, we define a simple SoC platform and call it FPro SoC (which is abbreviated from "FPGA Prototyping" or can be interpreted as "Fun and Professional"). It is composed of a video subsystem and a memory-mapped I/O subsystem and provides a collection of predesigned IP cores, including general-purpose peripherals, customized hardware accelerators, and a music synthesizer [10,11]. The goal of the platform is to facilitate the student learning in the areas of digital systems and hardware-software co-design and to serve as a pathway to comprehend and utilize commercial SoC platforms to develop full-fledged SoC systems in the future.

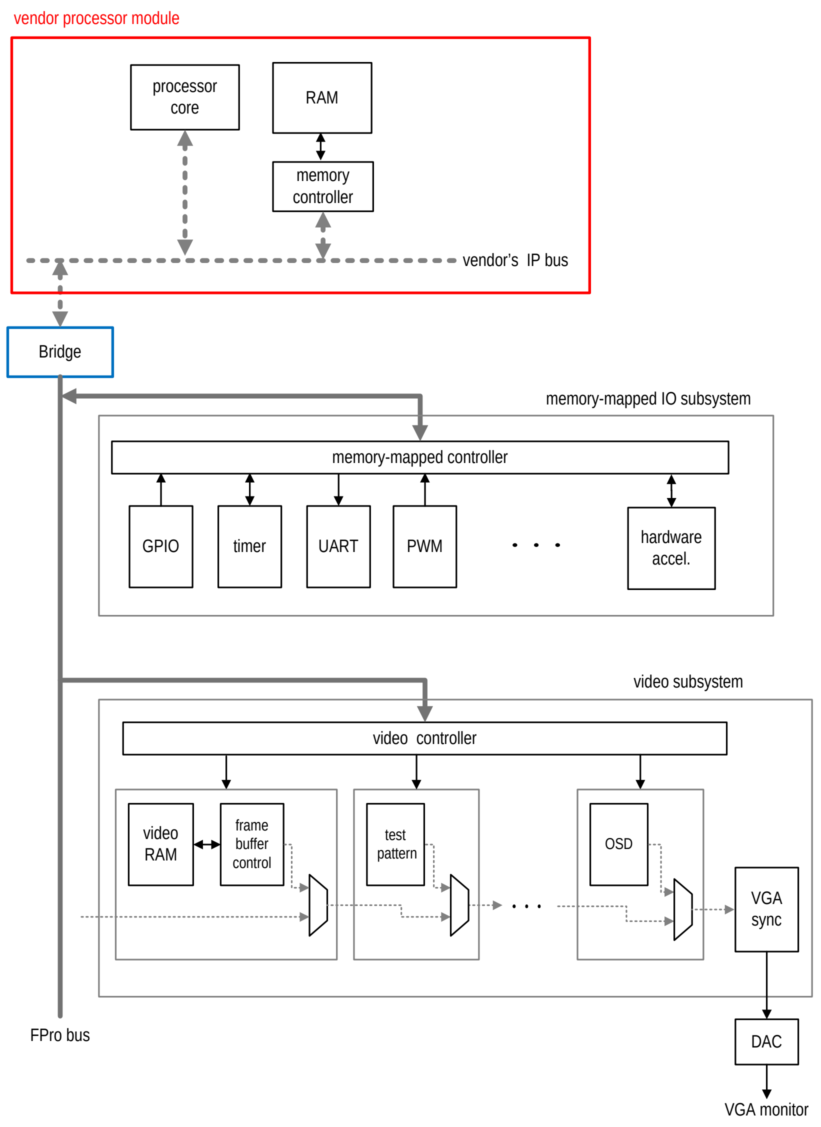
The guiding principles to develop the FPro platform are:

- Simple. The FPro platform defines a simple synchronous bus protocol and a straightforward software device driver structure. Once a custom hardware circuit is developed, it can be converted to an FPro IP core by adding an interface wrapping circuit and a device driver. The core then can be incorporated into the embedded system.
- Functional. The FPro platform provides commonly used I/O peripherals and serial interfaces (UART, SPI, and I^2C) and includes working device drivers. It resembles a bare-metal 32-bit microprocessor board.
- Portable. Except for the processor, FPro's IP cores are developed from scratch in HDL and do not use any vendor's proprietary components. The bus protocol and device drivers are not tied to any specific commercial platform or library, either. Thus, the IP cores and software are portable and independent of vendors. They can be reused for different FPGA devices and prototyping boards.
- "Upward compatible." While the FPro platform is simple, the development follows the rigorous design principles and the best practices. These knowledge and skills can be applied in the future for more complicated commercial platforms and larger projects. The FPro IP cores and drivers can be easily modified to be incorporated into existing commercial IP frameworks.
- Fun. Because the developed system resembles a real microprocessor board, it can incorporate existing I/O modules and quickly develop a functional prototyping project. In addition, this platform can provide hardware acceleration capability and custom peripherals and thus is more capable and more flexible than any microprocessor board. It allows students to develop interesting and challenging projects and makes studying hardware more "fun"' rather than "learning hardware for the sake of hardware."'

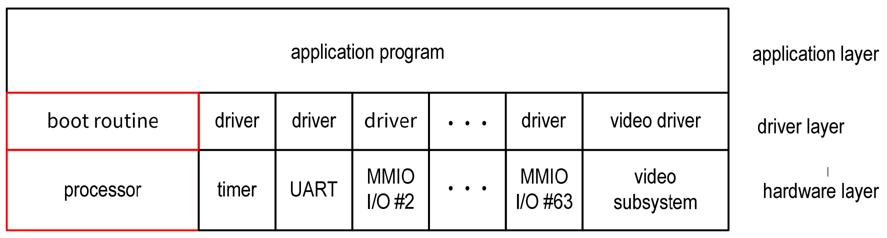
3. Platform hardware organization

The top-level conceptual diagram of an FPro system is shown in Figure 1(a). It is composed of four major parts:

- Processor module
- FPro bridge and FPro bus
- MMIO (memory mapped I/O) subsystem
- Video subsystem



(a). Top-level diagram of an FPro system



(b). Software hierarchy of an FPro SoC system

Figure 1. Hardware and software of an FPro system

3.1 Processor module

The processor module, shown as the red box in Figure 1(a), consists of a processor, a memory controller core, and RAM. It is the only part that is constructed from the vendor's IP cores. To be used in the FPro SoC platform, the processor core must exhibit the following characteristics:

- 32-bit-wide data path
- 32-bit memory address space
- Memory-mapped-I/O scheme for I/O access

The memory-mapped I/O scheme means that an I/O core is treated as a collection of registers and shares the same memory address space. The processor reads and writes the I/O core's register just like a normal memory word. Almost all FPGA-based processors support this feature [3,19]. There is no restriction on types of RAM. It can be FPGA's internal memory modules or external memory devices.

3.2 FPro bridge and FPro bus

The processor communicates with I/O cores using a bus or an *interconnect* structure. The modern interconnect, such as AMBA AXI [5], is designed to accommodate a wide variety of communication and data transfer needs and involves complex protocols. It is implemented by the proprietary IP cores. For the learning purposes, we define a simple synchronous bus protocol for the two subsystems and call it *FPro bus* [10,11]. The *FPro bridge*, shown as the blue box in Figure 1(a), converts vendor's native bus signals into the FPro bus signals. It needs to be redesigned for each processor.

3.3 MMIO subsystem

The MMIO (memory-mapped IO) subsystem provides a framework to accommodate MMIO IP cores, which can be general-purpose and special I/O peripherals as well as hardware accelerators. The MMIO subsystem consists of a controller to select a specific slot and can accommodate up to 64 instantiated MMIO IP cores.

For simplicity, we define a standard *slot interface* that conforms to the FPro bus protocol. An MMIO IP core can be created by "wrapping" custom digital logic with a *slot interface circuit*, which contains a collection of registers, decoders, and multiplexers. The conceptual diagram is shown in Figure 2.

A collection of MMIO IP cores is developed. All cores follow the slot interface specification and can be inserted into any slot of the MMIO subsystem [10,11]. These cores are

- General-purpose input port
- General-purpose output port
- Timer
- XADC controller
- UART (universal asynchronous receiver/transmitter) controller
- PWM (pulse-width modulation) controller
- Debouncing circuit
- Multi-digit seven-segment LED display controller
- I^2C controller
- SPI controller
- PS2 controller
- DDFS (direct digital frequency synthesis) synthesizer

- ADSR (attack-decay-sustain-release) envelope generator

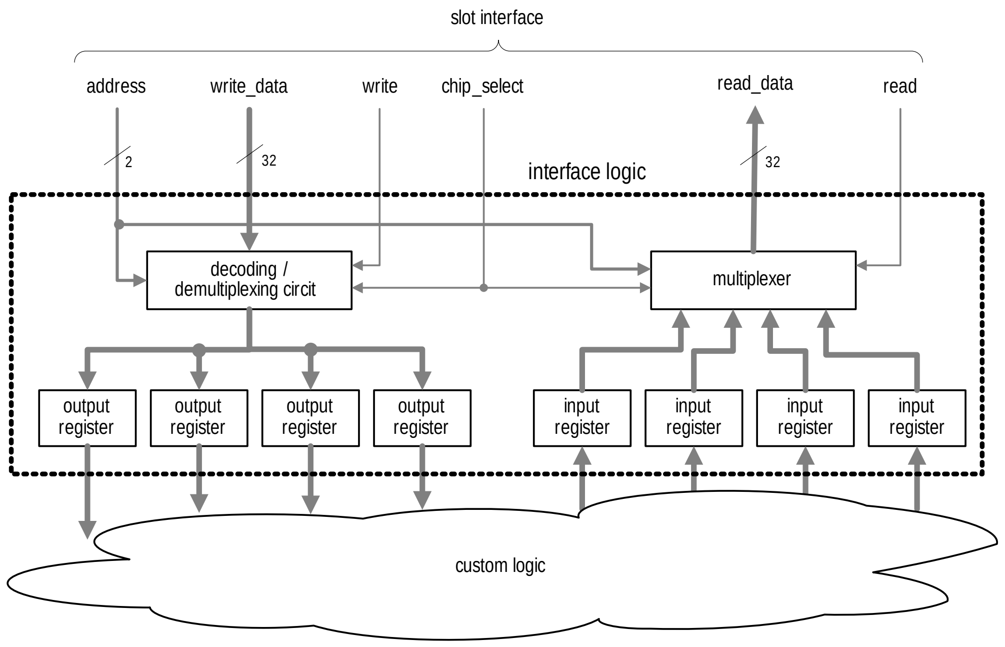


Figure 2. Block diagram of an MMIO IP core

3.4 Video subsystem

The video subsystem establishes a framework to coordinate the operation of video IP cores, as shown in the bottom of Figure 1(a). A video core generates or processes the video data stream. The cores are arranged as a cascading chain. The video data is "streamed" and "blended'" through each stage and eventually displayed on a VGA monitor. A collection of video IP cores is developed [10,11]:

- VGA synchronization core. It synchronizes the video stream and outputs the data to a VGA monitor for display.
- Test pattern generator. It produces a test screen.
- Color conversion circuit. It converts a color image to a greyscale image.
- Sprite control circuit. It generates and controls a *sprite*, which is a small animated object, on a VGA display. Two sprite circuits, one for the mouse pointer and one for a PacMan ghost character, are developed.
- OSD (on-screen display) controller. It contains a text buffer and generates characters on a VGA display.
- Frame buffer. It is a video memory that stores a frame (i.e., a screen) of pixel data.

The video subsystem demonstrates the principles of handling *stream data*, in which data are generated or arrived continuously, and then routed through a chain of components for processing.

4. Platform software organization

The FPro software hierarchy is shown in Figure 1(b). It contains a hardware layer, a device driver layer, and an application layer. The software runs as a bare-metal system, in which no operating system is installed and an application program interacts directly with the I/O peripherals via the device drivers [12]. In its simplest form, the processor boots directly into an infinite main loop, which contains functions to check input, perform computation, and write outputs.

The device drivers are portable and constructed from scratch. Only the boot routine, shown as the red box in Figure 1(b), is tied to the processor configuration. It first performs the basic initialization process, such as clearing the caches, configuring the stack and heap segments, and then transfers control to the main program. The boot routine is usually included in the processor's software toolchain and inserted into executable code automatically by the linker.

4.1 Direct I/O register access

The I/O register is treated as a normal memory location in the memory-mapped I/O scheme. During the hardware construction, each slot in the MMIO and video subsystems is assigned a base address. The address of a register in an IP core can be obtained by adding the offset to the base address. The application code can access its I/O registers by reading and writing the designated addresses (as pointers). A pair of macros can make the register read and write operations more expressive:

```
#define io_rd(addr) (*(volatile int *)(addr)
#define io_wr(addr, data)  (*(volatile int *)(addr) = (data))
```

Note that the macros are generic C codes and do not depend on the processor. For a processor with a data cache, the region associated with the MMIO and video subsystems should be marked as "non-cacheable." This is usually specified in software toolchain and included in the boot routine.

4.2 Device driver

A device driver is a collection of routines that interact with the I/O peripherals [33]. The driver presents the corresponding I/O functionalities in a more abstract way and thus shields the application program from the tedious details of the hardware. The FPro software framework constructs a driver for each IP core. A driver is implemented as a unique C++ class. The methods of the class are used to perform the desired functionalities and the private section of the instance is used to main the "state" and relevant information of the corresponding IP core. In this approach, a class becomes "self-contained"' and does not interact with other classes.

When a core is attached or removed from an FPro system, the corresponding driver files should be included or deleted from the software project. In the main application program, an instance will be created for each instantiated IP core. The methods in the class will be used to access and control the core and the state of the core is kept within the private section of the instantiated object. No external variable is involved.

5. Implementation

Xilinx and Intel (Altera) are two major FPGA manufacturers and they produce an array of education-oriented prototyping boards. The FPro SoC platform is successfully implemented in the entry-level boards from both manufacturers. The main task is to construct a processor module with the vendor's IP cores and to develop a bridge, as discussed in Sections 3.1 and 3.2. Since the boot routine is inserted by the linker in the respective toolchain, the software codes are identical for all prototyping board.

5.1 Intel prototyping boards

Intel provides a soft-core processor, Nios II, for its FPGA device [2] and uses the Avalon interface [1] to connect its IP cores. We construct an FPro processor module with a Nios II core and 128 KB on-chip memory. Using the FPGA's internal memory simplifies the design and makes the processor module more portable. The 128 KB RAM is large enough to accommodate most simple embedded applications. A custom FPro bridge translates the Avalon signals to the FPro bus signals. The FPro system is implemented and verified on two entry-level prototyping boards: DE0-CV, which contains a Cyclone 5 device, and DE10-Lite, which contains a MAX10 device [2,17]. The DE10-Lite board is shown in Figure 3(a). It contains an Arduino-like header, as indicated by the green outline. After instantiating proper MMIO IP cores, the board can be configured as an "embedded system" prototyping board and uses the I/O peripheral modules designed for the Arduino boards [14]. The cost of the DE10 Lite board is $55 ($85 for non-academic).

The software development is done in the Intel Nios II SBT (software built tool) platform. It is the Eclipse platform with a special Nios II plug-in, which invokes software toolchain to compile, link, and load software code to a Nios II system.

5.2 Xilinx prototyping boards

Xilinx provides a soft-core processor, MicroBlaze, for its FPGA device. MicroBlaze MCS (microcontroller system) is a complete system with a preconfigured MciroBlze core, on-chip memory, and an IO module [19]. We construct an FPro processor module with a MicroBlaze MCS instance of 128 KB on-chip memory. The MicroBlaze MCS communicates with an "I/O bus port." A custom FPro bridge translates the MCS I/O bus signals to the FPro bus signals. The FPro system is implemented and verified on three entry-level prototyping boards: Nexys 4 DDR, Basys 3, and Arty A7 [13]. All boards contain an Artix7 device [18]. The Arty A7 is shown in Figure 3(b). It also contains an Arduino-like header, as indicated by the green outline. As for the DE10-lite board, it can be configured as an "embedded system" prototyping board and uses Arduino I/O peripheral modules. Since the Arty A7 board does not have a built-in VGA port, an additional VGA module is needed if the video subsystem is instantiated. The cost of the Arty A7 board is $99.

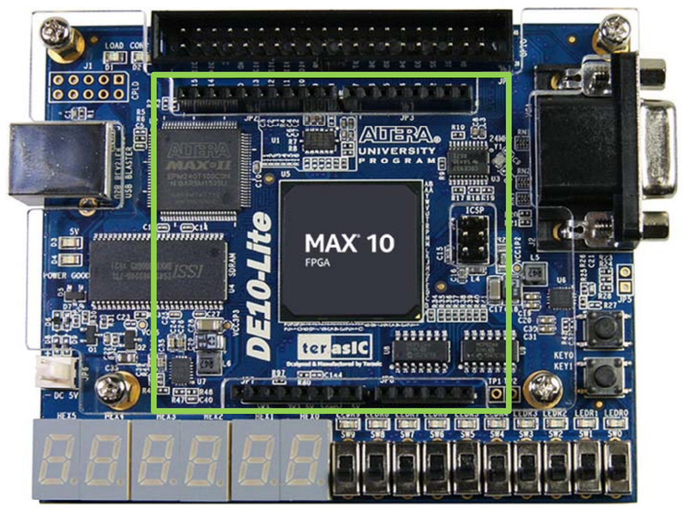
The software development is done in the Xilinx SDK (software development kit tool) platform. Similar to Nios II SBT, it is the Eclipse platform with a special MicroBlaze plug-in, which invokes software toolchain to compile, link, and generate the execution image.

6. Summary

A spiral lab framework is established for the computer engineering curriculum. The framework covers both hardware and software and the experiments are done in the SoC context. To support this effort, the FPro platform, a simple open and vendor-neutral SoC platform, is developed. The FPro platform defines a simple bus protocol and IP core interface and provides a collection of predesigned IP cores. It can be used to construct custom and functional embedded system and to demonstrates many key design concepts. The physical system is successfully implemented and tested with entry-level Intel and Xilinx prototyping boards.

7. Acknowledgments

Part of this material is based upon work supported by the IUSE program of the Division of Undergraduate Education of the National Science Foundation under Grant No. 504030. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation. The photos in Figure 3(a) and Figure 3(b) are courtesy of Intel and Xilinx, respectively.


(a). DE10 Lite board (source: Intel Inc.)

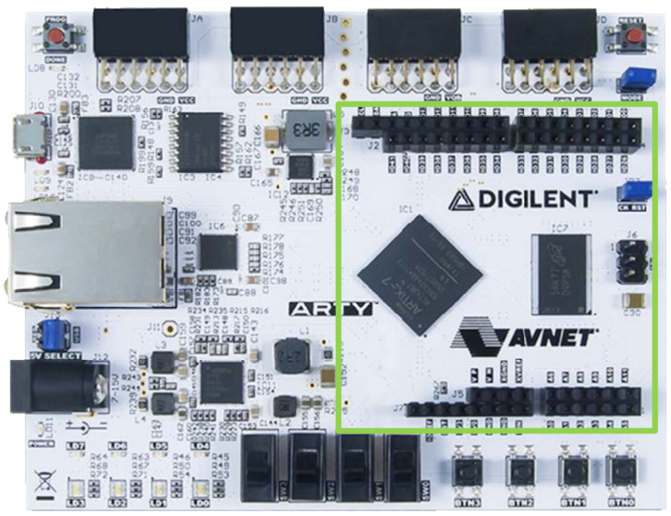

(b). Arty A7 board (source: Digilent Inc.)

Figure 3. FPGA prototyping boards

References

[1]. Altera, *Avalon Interface Specifications*, Intel Corp., 2017.
[2]. Altera, *MAX10 FPGA User Guides*, Intel Corp., 2017.
[3]. Altera, *Nios II Gen 2 Processor Reference Guide,* Intel Corp., 2016.
[4]. Arduino, "Arduino UNO R3 Board," https://store.arduino.cc/usa/arduino-uno-rev3
[5]. ARM, *AMBA AXI and ACE Protocol Specification*, ARM Holdings, 2011.
[6]. C. J. Atman, et al., *Enabling Engineering Student Success: The Final Report for the Center for the Advancement of Engineering Education*, 2010.
[7]. P. Chu, "Integrating Computer Engineering Labs with a Sound Theme," *Proceedings of ASEE Annual Conference*, 2016.
[8]. P. Chu, "Integrating Computer Engineering Labs with a Video Theme," *Proceedings of ASEE Annual Conference, 2017*.
[9]. P. Chu, Chansu Yu, and Karla Mansour, "Integrating Computer Engineering Lab Using Spiral Model," *Proceedings of ASEE Annual Conference, 2017*.
[10]. P. Chu, *FPGA Design by VHDL Examples: MicroBlaze MCS SoC edition*, Wiley & Sons, 2017.
[11]. P. Chu, *FPGA Design by SystemVerilog Examples: MicroBlaze MCS SoC edition*, Wiley & Sons, to be published in September 2018.
[12]. J. Corbet, A. Rubini, and G. Kroah-Hartman, *Linux Device Drivers: Where the Kernel Meets the Hardware*, O'Reilly Media, 2005.
[13]. Digilent, *Arty FPGA Board Reference Manual*, Digilent, 2017.
[14]. S. Monk, *Programming Arduino: Getting Started with Sketches*, McGraw-Hill, 2011.
[15]. R. Sass and A. G. Schmidt, *Embedded Systems Design with Platform FPGAs: Principles and Practices*, Morgan Kaufmann, 2010.
[16]. S. Sheppard, et al., *Educating Engineers: Designing for the Future of the Field*. Jossey-Bass, 2009.
[17]. Terasic, "*DE10 Lite Board*," http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=234&No=1021
[18]. Xilinx, *DS180 7 Series FPGAs Data Sheet: Overview*, Xilinx, 2017.
[19]. Xilinx, *PG116 MicroBlaze Micro Controller System*, Xilinx, 2013.