An introduction to soft-core processors and a biomedical application

Dominic Romeo, Joseph LaMagna, Ian Hogan, and James Squire

Despite the ubiquity of microcontrollers, widespread use of soft-core microprocessors is much less common. Most undergraduate curricula have a digital course involving field programmable gate array (FPGA) programming, in languages such as VHDL or Verilog, and separately have a microcontroller course that uses the C language. But few synthesize these two topics to involve programming an FPGA to simulate a microcontroller.

This article describes a senior-level elective course in which students design a chemotherapeutic cancer research device using a soft-core processor at the heart of their product. The students describe how they chose to use a soft-core processor and how the implementation involved many nights of struggle, cumulating with debugging at 70 mi/h in a car to make the delivery deadline.

What are soft-core processors?

Microcontrollers are familiar to most electrical engineers. The word *microcontroller* is commonly associated with the PIC and ATMega chips and development boards such as the Arduino and Raspberry Pi, a few of the popular and widely-supported microcontrollers and development boards in use today by

Digital Object Identifier 10.1109/MPOT.2017.2733341
Date of publication: 8 March 2018

xxxxxxxxxx

0278-6648/18©2018IEEE

makers, students, and hobbyists. Microcontrollers pack a lot of functionality on a single chip, but for bigger or faster tasks, one may have to turn to FPGAs. When FPGAs are mentioned, engineers often begin to sweat a little. For many, FPGAs bring back memories of digital design and painful lines of hardware description language (HDL). Modern design tools have changed this and allow anyone needing the flexibility and performance of FPGAs while avoiding the difficulties of HDL programming.

So what is a soft-core processor? A soft-core processor is a program prewritten in HDL that provides a processing core that runs on an FPGA. It allows the programmer to define peripherals such as general purpose input/output lines (GPIOs), serial parallel interfaces (SPIs), and universal asynchronous receiver/transmitters (UARTs) at compile-time so he or she can design a powerful embedded system uniquely customized to his or her requirements. Need a system with 40 pulse-width modulated (PWM) outputs, a 16-b-wide parallel output port, and 12 serial ports? No problem. With the design tools available today, one can design an embedded system with a soft-core processor without writing a single line of HDL.

FPGA design tools such as Xilinx Vivado define the soft-core processor using graphical drag-and-drop operations. Then the C++ program that runs on the soft-core processor is compiled using a different design tool such as the Xilinx software development kit (SDK). On power-up, the hardware design is loaded onto the FPGA to create the soft-core processor and any additionally defined peripherals. Then the C++ program is loaded onto that soft-core processor, and finally the program executes.

FPGAs have development boards just as do many microprocessors. These development boards are great places to start designing embedded systems that use a soft-core processor. A relatively new development board for the Artix 7 FPGA is the ARTY by Digilent. What makes the ARTY particularly interesting is that it was designed with the Microblaze soft-core processor in mind. The ARTY reference page provides several getting-started examples using the Microblaze. The ARTY sports 256 MB of external RAM, external Flash for storing both the FPGA binary definition file and the C program that it runs, and a USB UART port to program the device. In terms of input/output (I/O), the ARTY boasts the familiar Arduino Uno footprint and an assortment of peripheral modular (PMOD) connectors; switches; push buttons; and red, green, and blue light-emitting diodes.

Advantages

In the Vivado design environment, digital logic is defined using a block diagram rather than writing traditional HDL code. That's right, no HDL is needed. Those who have used LabVIEW will see some similarities with using the Vivado block design tool. Adding a soft-core processor, such as the Microblaze, becomes as easy as dragging and dropping from a toolbox, and its peripherals are defined using a setup wizard. Wiring blocks are done by simply clicking and dragging. Vivado even automatically makes the basic connections to implement bare-bones microcontroller functionality.

If a project calls for both a microcontroller and FPGA, a soft-core processor can decrease the overall printed circuit board (PCB) footprint, speed development time, and permit more flexible redesigns by implementing both on a single chip. A soft-core processor allows the user to customize the functionality of the microcontroller. Setup wizards let the programmer choose any combination of I/O, microprocessor and SPI clock speeds, amounts of RAM, and, in some cases, interfacing logic levels. If a project calls for more computing power, more cores can be added, all operating simultaneously, while the FPGA supplies the glue logic needed to connect them to outboard peripherals.

From a data acquisitions applications standpoint, the Microblaze and other soft-core processors offer the advantage of deterministic timing, unlike application processors such as the Rasperry Pi that typically run Linux as an application-scheduling operating system. This lets one precisely control when data is being read from a pin without any jitter, since the FPGA nature of the device lets one rewrite interrupt routines to become truly independent parallel routines that are handled simultaneously. Develop-

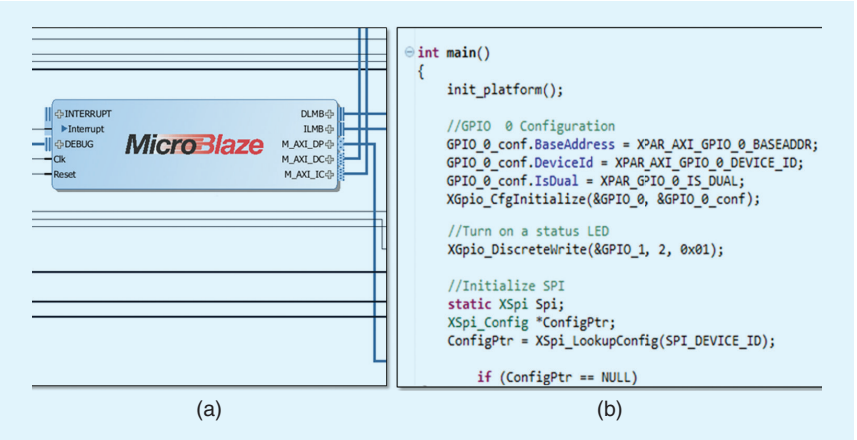


FIG1 (a) The part of a block design implementing a Microblaze soft-core processor. This shows the ease in implementing a soft-core processor using powerful design tools such as Viviado, which turns soft-core microprocessor design into a drag-and-drop operation. (b) Part of a program designed to run on this Microblaze that configures its GPIO and SPI interfaces at run-time. Programs are written in C++, a language familiar to most engineers.

TABLE 1. XXXX

	MICROBLAZE ON ARTY	ARDUINO UNO REV 3	RASPBERRY PI 3
Processor Speed	100 MHz	16 MHz	1.2 GHz
RAM	256 MB DDR3L	2K SRAM	1 GB LPDDR2
Operating System	No	No	Yes

A comparison of a few of the capabilities of the most popular hardware microcontroller development boards with that of the ARTY, a development board for the Artix 7 FPGA. The Raspberry Pi has a higher clock speed and more RAM than either the Arduino or ARTY, but it requires an operating system to run more than one thread, which may not permit timing-critical applications, such as data acquisition, that are intolerant of jitter.

ment boards such as the ARTY also provide prodigious amounts of RAM (256 MB of DDL3!), significantly more than most microcontroller development boards and provide simple block-design methods to interface them to the soft-core microcontroller.

The cons

Development environments range from lightweight and sleek to bulky and complicated. The popular development environment used with the ARTY for developing a Microblaze soft-core-processor-based system is Vivado, and it is definitely of the 800-lb-gorilla variety. Open Vivado and prepare to be blinded by windows, sidebars and menus. And Vivado is not the only design environment needed; the C code that runs on the Microblaze is developed on the Xilinx SDK. For those who enjoy the simplicity of the Arduino integrated development environment, Vivado is a totally different beast.

Moving from the prototype phase to a product with an FPGA is also much more complicated than when using microcontrollers. Prototypes using hardware microcontrollers can often be made using through-hole versions, or relatively simple small outline integrated circuit dual inline package adapters that can be hand soldered. Not so with FPGAs, which have much higher pin counts and pin densities precluding hand-assembly, and which often come in packages such as ball grid arrays that require specialized equipment to wave-solder. FPGAs also require external support circuitry, such as electrically erasable programmable read-only memory, to store the logic design, code, and random access memory (RAM) in separate chips. To produce a PCB with an FPGA and its circuitry typically requires at least four layers.

To get started or seek help implementing functions such as the UART or SPI on the soft-core microcontroller, one must turn to online communities and documentation. The wealth of carefully-documented functions and examples that accompany many hardware microcontrollers are simply not present for FPGA development objects such as the ARTY, Microblaze, and Vivado. The supplied application programming interface is helpful but often lacks reference examples. The getting started page for the ARTY is, similarly, a great resource but is littered with "work-in-progress" and "fix-me" tags suggesting what is written in those sections may not be true. As soft-core microcontrollers rise in popularity, this situation will continue to improve, but compared with the civilized world of microcontrollers, FPGA design definitely has a Wild West feeling.

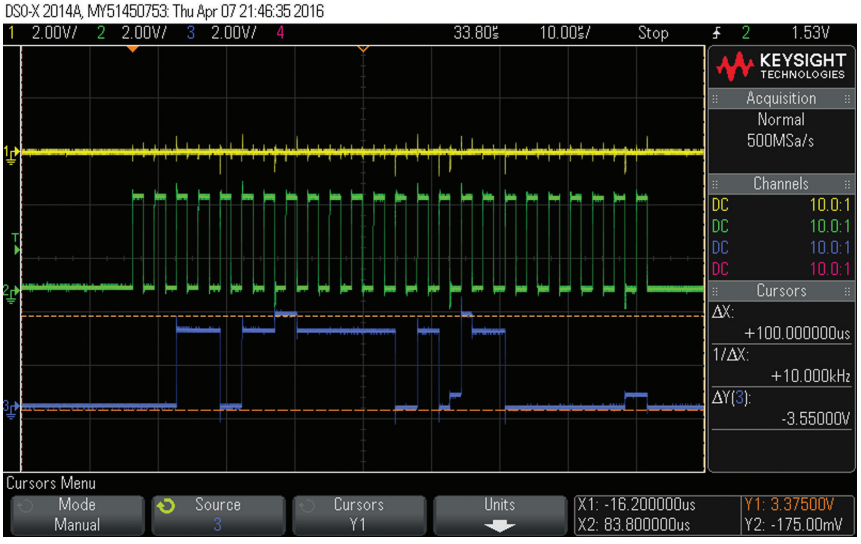


FIG2 The Wild West of FPGA design: A scope screenshot taken while debugging SPI communications errors. The blue channel should be a square wave showing data being sent from an external ADC to the MISO pin on the ARTY. The spikes on the channel suggest that the reference documentation for the software-designed SPI port have reversed the MOSI and MISO pin names, so the FPGA is attempting to (feebly) push data out on its input line. We assume reference documentation to be correct, but this scope capture shows that is not always the case.

Real-world application

At the Virginia Military Institute, Dr. James Squire offers a course in electro-mechanical design. In this course, senior electrical and computer engineering students are given a real-world engineering consulting project and must interact with an off-campus client to build a device that solves a problem. This past spring term, the author and students were asked to design a system to measure the impedance of cells as they grow in a culture well to be used in chemotherapeutic cancer research. It involved creating a front-end graphical user interface

(GUI) running on a PC written in C# that communicated to a real-time phase-locked differential amplifier run by a microcontroller. The analog subsystem design and PC/microcontroller interface turned out to be relatively simple; the logic subsystem that permitted analog to digital (A/D) acquisition of 18 b of data at a rate of 400,000 samples/s for up to 10 s at a time turned out to be significantly more complicated.

We completed the contract in three phases. The first phase simplified the engineering requirements to allow data of 8-b precision to be taken at 100 samples/s for 1 s. This speed enabled the use of a common microcontroller development board, the Teensy by PRJC. The Teensy hosts a 16-b microcontroller with more than enough RAM to hold the data collected at this slow speed, an integrated A/D unit, and onboard USB to allow the upload of collected data. When this was successfully built and tested, the project entered the second phase in which an external A/D converter was interfaced to increase the signal acquisition precision to 18 b. The A/D chosen was capable of up to 400 ksps, although at this phase, we remained at 100 samples/s. The Teensy remained to control the A/D, store the A/D data, and communicate with the PC to upload the data.

The final phase was by far the most challenging: to increase sampling speed by over three orders of magnitude to 400 ksps. The A/D was emitting data using a serial interface; 18 b at 400 ksps requires very tight timing, especially since over half of the time the A/D converter was performing the conversion and therefore unable to communicate. This speed required a microcontroller with a very fast clock.

Further, we wanted to embed a sampling timer on the microcontroller to trigger the start of each A/D conversion, but the fastest microcontrollers we could find ran Linux or another scheduling operating system, making it impossible to obtain clock-pulse-accurate jitterless A/D triggering. Another challenge was the raw amount of data: 18 b of data at 400 ksps over a 10-s experiment generates over 10 MB of data, requiring external RAM.

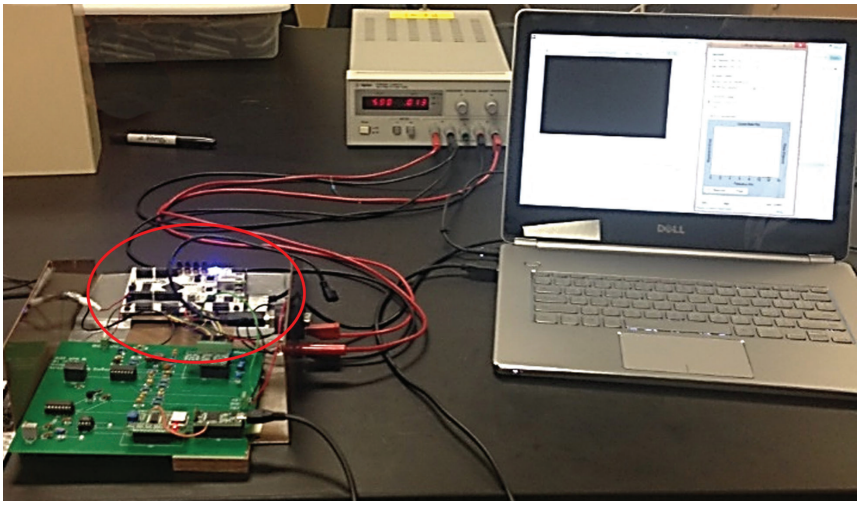


FIG3 The final product delivered to the client's biomedical lab. The product housing is removed to reveal the analog circuitry populated on a custom PCB that is controlled by the ARTY (circled in red). The laptop is running a GUI written in C# to control various data acquisition parameters and analyze and display the collected data.

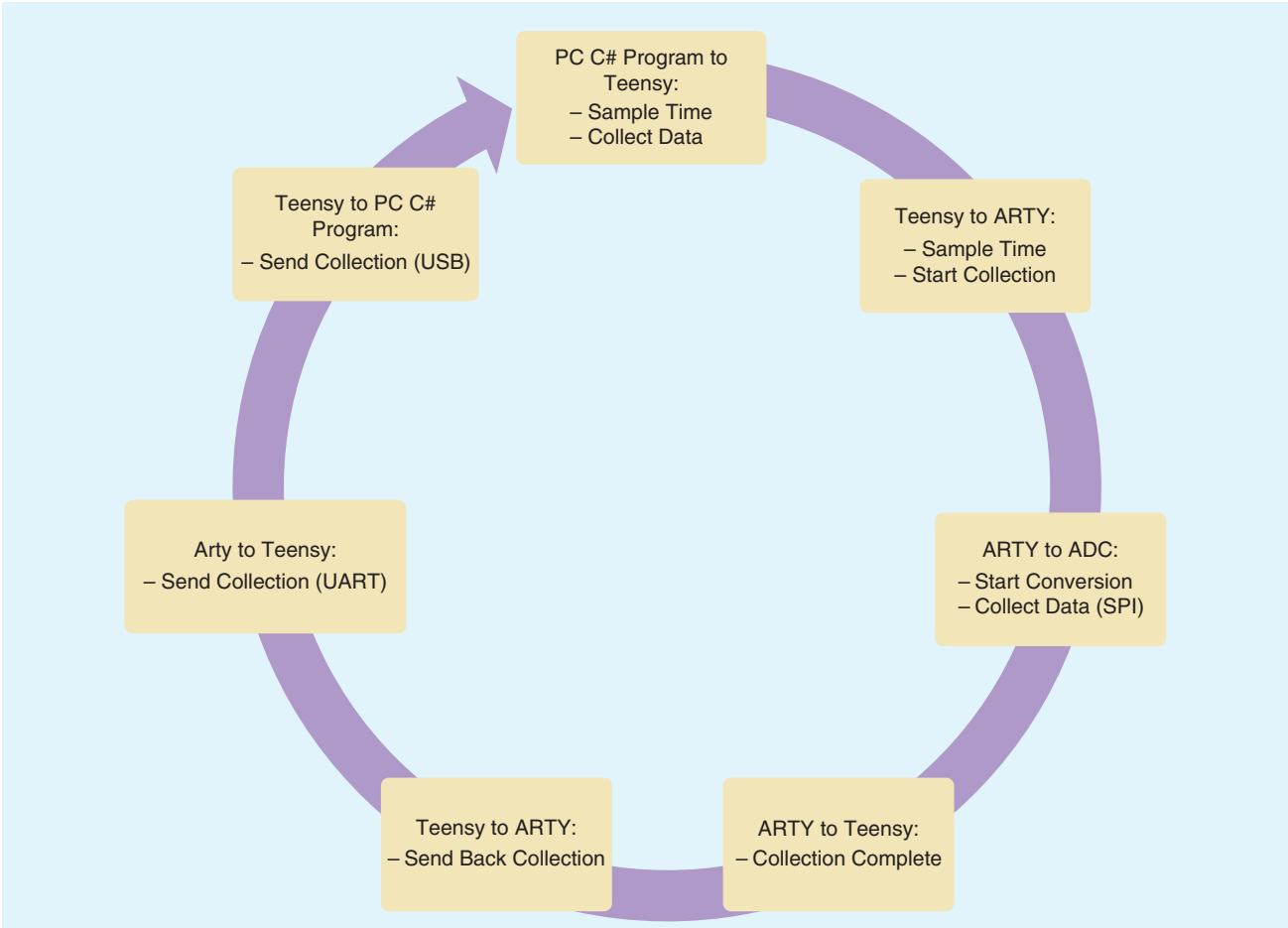
The solution came with the ARTY development board. The ARTY uses an Artix 7 FPGA and 256 MB of RAM, all on an easy-to-prototype development board. We used the Teensy microcontroller for all non-time-critical management functions including using its USB port to upload the sampled data to the PC. The ARTY was used in the time-critical loop to trigger the A/D conversions, serially clock the data out of the A/D, and store it in its DDL3 RAM. Then it provided the data to the Teensy in chunks using a simple handshaking protocol for the Teensy to send back to the PC for analysis and display.

The team's initial plan was not to use a soft-core microcontroller but rather to design a state machine in the FPGA to trigger the A/D converter, clock out the conversion serially, and store the conversion in successive RAM locations, since these are simple, repetitive operations. This was more complex than we anticipated. Unlike the detailed documentation, tutorials, and example code available for many popular microcontrollers, it quickly became apparent that FPGA programmers form a smaller community, and, therefore, FPGA documentation is less-extensive. Vivado is not intuitive to use.

The first problem we attempted to solve in building the state machine was accessing the onboard DDR4 RAM, and after countless hours searching and experimenting, we were still dead in the water. Looking through the ARTY tutorial page, we came across a section that mentioned the soft-core microcontroller named Microblaze in a "hello world" tutorial. This simple program gave the basic reference design for the softcore microcontroller, which ultimately laid the foundation to solve all the problems we encountered. It allowed us to change the approach of the team from being HDL coders to being traditional C++ coders, a language we had considerably more experience using. The tutorial also showed how to set up the RAM IP Block with the Microblaze, which solved the problem of memory interfacing. Further optimization could allow the Microblaze to communicate through the ARTY's USB port to the PC, eliminating the Teensy hardware microcontroller entirely.

Murphy's law strikes

Blazing down the highway at 70 mi/h, we headed to Massachusetts



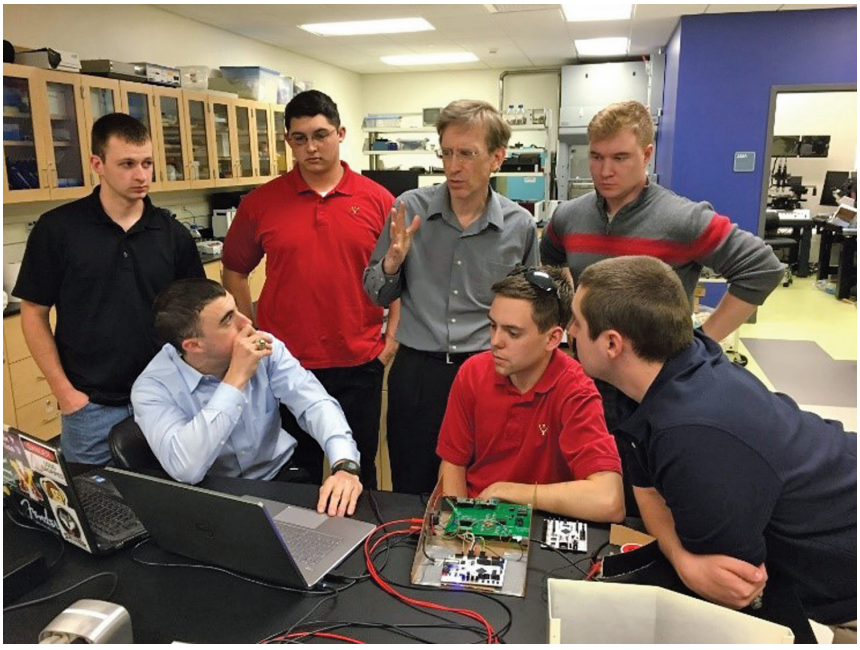


FIG5 After a hair-raising debugging ride to his laboratory, we present our client with the final product and demonstrate that it meets his requirements.

to deliver the product to Dr. Anthony English's lab at Western New England University. I sat in the back seat with a laptop, using the 10-h ride to clarify comments in the code and add reference documentation, both of which seemed trivial enough to leave for the car ride. Halfway there, I double-checked to make sure the newly-commented code would synthesize (FPGA-speak for *compile*) correctly, and I was alarmed to find it generating a ton of errors. After some digging, I realized the Vivado package had determined that the car ride over would be a good time to update its core libraries to a new, not fully compatible, version. With a couple of hours left I located the troublesome libraries and began patching the code. As we pulled into the parking lot, I frantically typed my last lines of code and made sure everything verified. It was a little closer than I would have liked to end the semester-long project, but it worked well and Dr. English is currently using it to obtain data.

If you need flexibility and performance in your next embedded

project, consider a soft-core processor. The ARTY provides an excellent introduction—purchase one, read the reference page on the Digilent website, complete a few of their tutorials, and get started on your own design.

About the authors

Dominic Romeo (dromeo116@aol.com) earned his B.S degree in electrical and computer engineering from the Virginia Military Institute in 2016. Upon graduation, he began working with Lockheed Martin—Rotary and Mission Systems in Manassas, Virgina. He currently works as software engineer and is part of the Engineer Leadership Development Program.

Joseph LaMagna (lamagnaj16@mail.vmi.edu) graduated from the Electrical and Computer Engineering Department at the Virginia Military Institute in 2016. Upon graduation, he was commissioned as an officer in the U.S. Army. He is currently stationed at Fort Rucker, Alabama, where he is training to become a helicopter pilot for the Army.

Ian Hogan (ianhogan93@yahoo.com) earned his B.S. degree from the Department of Electrical and Computer Engineering at the Virginia Military Institute. He currently works for the Department of the Air Force, supporting the TACP-M Program Office at Hanscom Air Force Base.

James Squire (squirejc@vmi.edu) is a professor of electrical engineering at the Virginia Military Institute. He earned his B.S. degree from the U.S. Military Academy and his Ph.D. degree from the Massachusetts Institute of Technology. He was awarded a Bronze Star in the Army in Desert Storm and was selected as Virginia's Rising Star professor in 2004. He is a licensed Professional Engineer and maintains an active consulting practice.