AC 2008-2023: FOSTERING DESIGN ACROSS MULTIPLE DISCIPLINES WITH GRAPHICAL PROGRAMMING AND FPGAS.

Shekhar Sharad, National Instruments

Greg Crouch, National Instruments

Reid Lee, National Instruments

Brian Johnson, National Instruments

Fostering Design Across Multiple Disciplines With Graphical Programming and FPGAs.

Abstract

Design has become an essential component of today's engineering education curriculum. However traditional tools and techniques that are primarily text-based have hindered students and professors from taking advantage of the various hardware platforms such as FPGAs that are available in order to be able to teach design effectively. This is especially true for disciplines other than electrical engineering and computer science, where the students are not experts in programming with textual languages. With the evolution of graphical programming tools, it is now possible to leverage the intuitive and powerful technology of graphical programming to target hardware platforms such as FPGAs in order to teach design across multiple disciplines. In this paper we demonstrate a new plug-in that we have built in order to be able to target a commonly used Xilinx Spartan 3E evaluation board with NI LabVIEW, a commonly used graphical programming software. We will also show a demonstration of this plug-in and discuss the pros and cons of such an approach

Introduction

Achieving proficiency in designing systems with real-world signals has become a necessity in every engineering discipline today. Systems of different scale are being created and used by mechanical, aerospace, biomedical, automotive, chemical and electrical engineers alike[1,2,3]. This is a very good development – now domain experts from mechanical, biomedical and chemical engineering can create much more efficient embedded systems that help solve problems in their area of expertise.

However, it is worth noting that unlike electrical engineers, the chemical, biomedical and mechanical engineers are not core-embedded programmers. This is significant because the tools used to teach design of embedded systems are still based on traditional textual approaches. In addition, embedded devices are increasing in complexity – for example, the number of gates on an FPGA have increased keeping pace with Moore's law, resulting in more sophisticated embedded platforms. This makes it exceedingly difficult to teach the design process effectively using traditional techniques. The traditional tools also restrict introducing embedded systems to introductory engineering courses which serve as an avenue to create excitement for design in the engineering education.

In this paper, we will examine graphical programming as a possible avenue to leverage to teach design. We will also examine an FPGA(Field Programmable Gate Array) board found in many embedded design laboratories, the Xilinx SPARTAN-3E XUP board and finally talk about the a plug-in that we developed for a leading graphical programming language, LabVIEW to target the Xilinx SPARTAN-3E FPGA board.

Traditional Vs Graphical Programming for Design

Design has evolved to comprise of two distinct components, software and hardware. From the software side, literature[4,5,6] shows that the actor-oriented or graphical programming languages are better suited for embedded design because they are based on the dataflow paradigm. Figure 1 shows an example of an actor-oriented graphical programming language, NI LabVIEW[7].

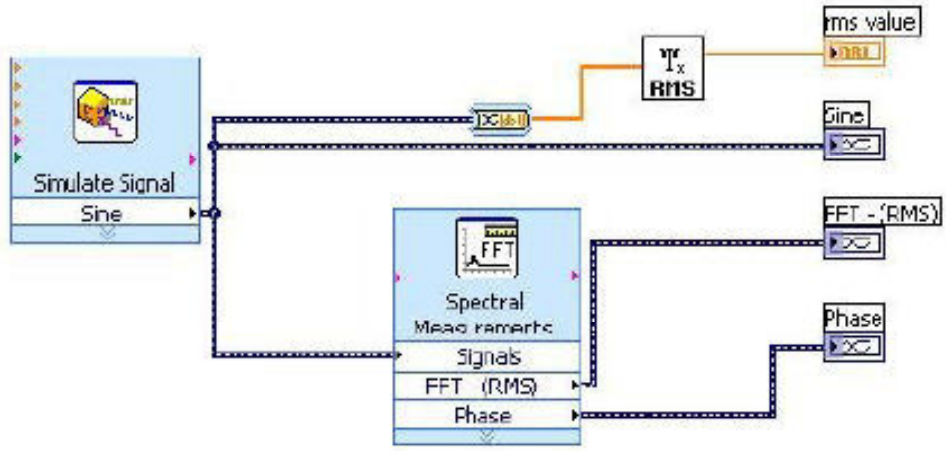
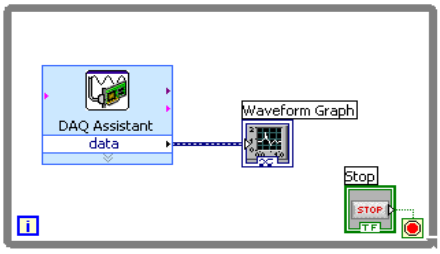


Figure 1. NI LabVIEW, Example of actor-oriented, graphical programming languages

In order to better understand how actor-oriented, graphical programming languages are better for teaching design, let us take a simple example of acquiring and presenting data from an external source, like a sensor – a situation that is present in most real-world, useful systems. Data could be analog or digital in nature, students could acquire from a thermocouple to regulate a temperature or a switch to initiate or abort a particular process. Figure 2a shows how one would go about achieving this with an actor-oriented, graphical programming language while figure 2b shows how the same can be achieved with a textual representation.



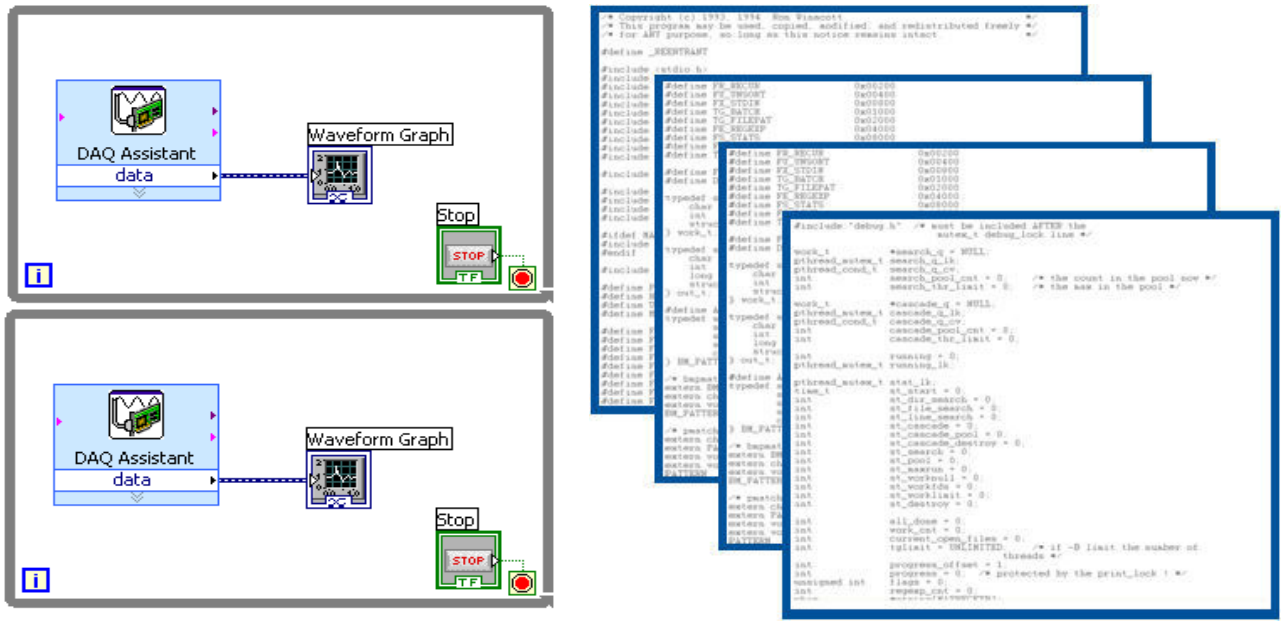

```
#include "nidaqex.h"

void main(void)
{
    i16 iStatus = 0;
    i16 iRetVal = 0;
    i16 iDevice = 1;
    i16 iChan = 1;
    i16 iGain = 1;
    f64 dSampRate = 1000.0;
    u32 ulCount = 100;
    f64 dGainAdjust = 1.0;
    f64 dOffset = 0.0;
    i16 iUnits = 0;
    i16 iSampTB = 0;
    u16 uSampInt = 0;
    static i16 piBuffer[100] = {0};
    i16 iDAQstopped = 0;
    u32 ulRetrieved = 0;
    static f64 pdVoltBuffer[100] = {0.0};
    i16 iIgnoreWarning = 0;
    i16 iFieldUm = 1;

    iStatus = DAQ_Rate(dSampRate, iUnits, &iSampTB, &uSampInt);

    iStatus = DAQ_Start(iDevice, iChan, iGain, piBuffer, ulCount,
     iSampTB, uSampInt);

    iRetVal = NIDAQErrorHandler(iStatus, "DAQ_Start", iIgnoreWarning);

    while ((iDAQstopped != 1) && (iStatus == 0)) {

        iStatus = DAQ_Check(iDevice, &iDAQstopped, &ulRetrieved);

        iRetVal = NIDAQYield(iFieldUm);
    }

    iRetVal = NIDAQErrorHandler(iStatus, "DAQ_Check", iIgnoreWarning);

    iStatus = DAQ_VScale(iDevice, iChan, iGain, dGainAdjust, dOffset,
     ulCount, piBuffer, pdVoltBuffer);

    iRetVal = NIDAQErrorHandler(iStatus, "DAQ_VScale",
     iIgnoreWarning);

    iStatus = DAQ_Clear(iDevice);

    iRetVal = NIDAQPlotWaveform(pdVoltBuffer, ulCount, WFM_DATA_F64);

    printf(" The data is available in 'pdVoltBuffer'.\n");

}
```

(a) (b)

Figure 2. Acquiring data from an external sensor in (a) graphical programming language and (b) textual programming language

It can be seen that the graphical program in figure 2a is intuitive enough for someone who does not have any programming knowledge to create. There is a loop to ensure that data is collected continuously and

there is a function block that collects data and a graph that displays it. Figure 2b, on the other hand, uses the standard set of calls to initiate the same, which while a little more complicated, may still be possible to teach. Now, let us go ahead and acquire from two sensors, in parallel. The resulting programs would be as shown in figure 3a and 3b.



(a) (b)

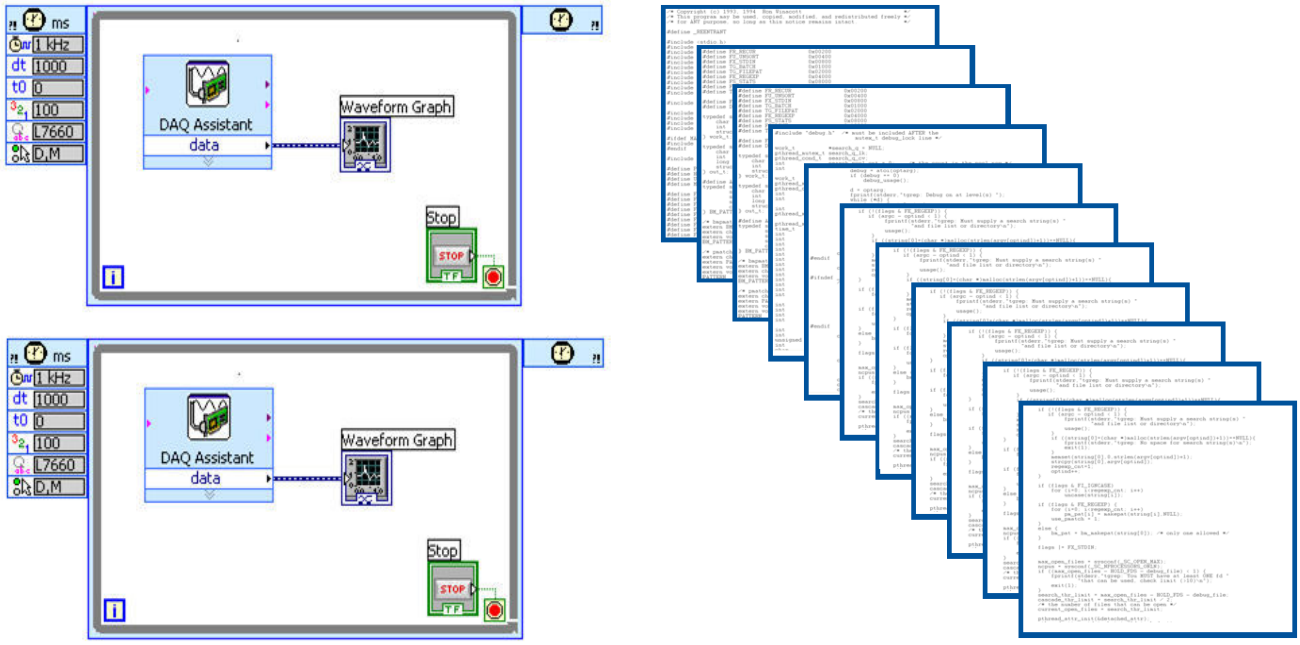
Figure 3. Acquiring from two sensors, using (a) graphical programming and (b) textual programming

In this case, the key characteristic to note is that since actor-oriented, graphical programming languages are based on dataflow paradigm, all that the student/educator needs to do is to replicate the block of code that was written for acquiring data from one sensor and reconfigure the block to give it the new information as shown in figure 3a. On the other hand, to achieve the same task, in parallel, with text based languages requires a significantly higher effort than just copying the code written previously. This is because the user has to configure the buffers, parallelize the code and make sure that there are no shared resources or that the right shared resources are allocated and so on.

The key takeaway from this particular example is since the dataflow paradigm is inherently parallel, graphical programming languages, based on dataflow paradigm are also inherently parallel, making them suitable to program both multi-core processor based systems and FPGAs. This translates to a significant advantage while designing systems with FPGAs as we will discuss later in this paper.

Further extending the same example, in order to do anything real with the real-world signals, timing is essential. We may need to acquire from different sensors at either different rates, or those sensors are controlled by different clocks and we need to be able to control and modify the program written in figure 3 to do this. Figure 4a shows how this can be done with graphical programming and figure 4b shows the same implementation with textual code. The change in the graphical programming code is subtle but intuitive, the loop can now be configured to include timing, without having to learn a convoluted series of syntax and terms – from a pedagogical perspective, this can be a big advantage when dealing with

hardware that has multiple timing sources. Figure 4b has some issues for pedagogy – first, debugging such systems is difficult and cumbersome. Second, learning the various intricacies of programming timed, heterogeneous, parallel embedded-systems and completing the project may not be possible in one or even two semesters for engineering students that are non-EE or CS majors. Hence, for the purpose of teaching design to non-EE and CS majors as well as freshmen engineering students, graphical programming languages in our view provides significant advantages over traditional, textual methods. In this paper, we have developed a plug-in driver for LabVIEW, a leading graphical programming language to target Xilinx SPARTAN-3E XUP board thus bringing the advantages of graphical programming and FPGAs to non-EE and CS disciplines.



(a) (b)

Figure 4. Acquiring from two sensors with timing in (a) graphical programming and (b) textual programming

FPGAs for Design

The other half of the design equation involves hardware. Some of the areas we evaluate for a hardware platform for academia include:

- Features – does it have the necessary peripherals that students would need?
- Relevancy to industry – is this technology being used by industry?
- Cost – is it cheap enough to buy for the whole lab?
- Support – can this platform be debugged if needed?

There are a lot of core silicon pieces that meet these demands, FPGAs, DSPs, Microprocessors and others. The authors have chosen FPGA for this paper as an example to illustrate how graphical programming can be used to teach design with real-world hardware (silicon). FPGAs also bring several other important topics that are relevant for real-world design – parallelism in design, integration with wide variety of

peripherals, quick prototyping platform that is widespread in industry as a platform to test out a design before going for full scale production.

Xilinx SPARTAN-3E XUP Student Board

The Xilinx SPARTAN-3E XUP Starter Board is based on the Spartan 3E FPGA from Xilinx. At its core, it has a 500,000 gate Spartan 3E FPGA with a 32 bit RISC processor and DDR interfaces. The board also features a Xilinx Platform Flash, USB and JTAG parallel programming interfaces with numerous FPGA configuration options via the onboard Intel StrataFlash and ST Microelectronics Serial Flash. The Spartan 3E Starter board is also compatible with the MicroBlaze Embedded Development Kit (EDK) and PicoBlaze from Xilinx. Further technical specifications for this board can be found at [8]. Figure 5 shows the Xilinx SPARTAN-3E XUP board.

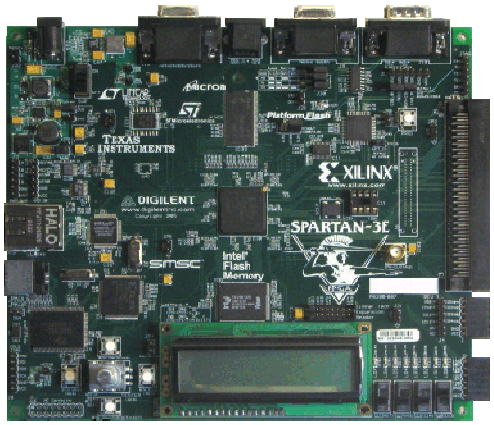


Figure 5. Xilinx SPARTAN-3E XUP Board

LabVIEW FPGA Module

NI LabVIEW is uniquely suited for FPGA programming because of its ability to clearly represent parallelism and dataflow as discussed in the previous section. The NI LabVIEW FPGA Module uses LabVIEW Embedded technology to extend LabVIEW graphical development to target FPGAs on NI reconfigurable I/O (RIO) hardware. With the LabVIEW FPGA Module, educators can create custom measurement and control hardware without low-level hardware description languages or board-level design and perform unique timing and triggering routines, ultrahigh-speed control, interfacing to digital protocols and digital signal processing (DSP). One of the features listed on the LabVIEW FPGA page [9] is its ability to target FPGAs without having to write any textual code. It can also integrate existing VHDL code if needed[10]. Figure 6 shows an example of code written using LabVIEW FPGA module.

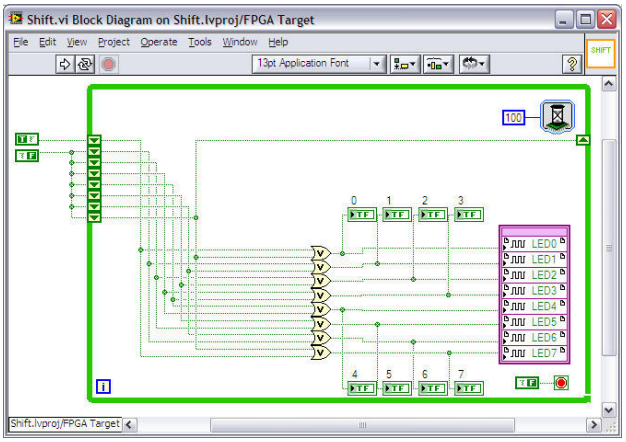


Figure 6. An Example of LabVIEW FPGA code

LabVIEW FPGA driver for Xilinx SPARTAN 3E XUP Board

In order to facilitate bringing real-world design to engineering especially to non-EE and CS disciplines, we have developed a plug-in for LabVIEW FPGA that lets educators and students target the Xilinx SPARTAN-3E board from LabVIEW. Previously, only National Instruments hardware could be targeted with LabVIEW FPGA making it hard for educators to take advantage of this technology. This plug-in is available as a download for educators worldwide[11]. Figure 7 shows how this plug-in shows up in LabVIEW FPGA after it is installed.

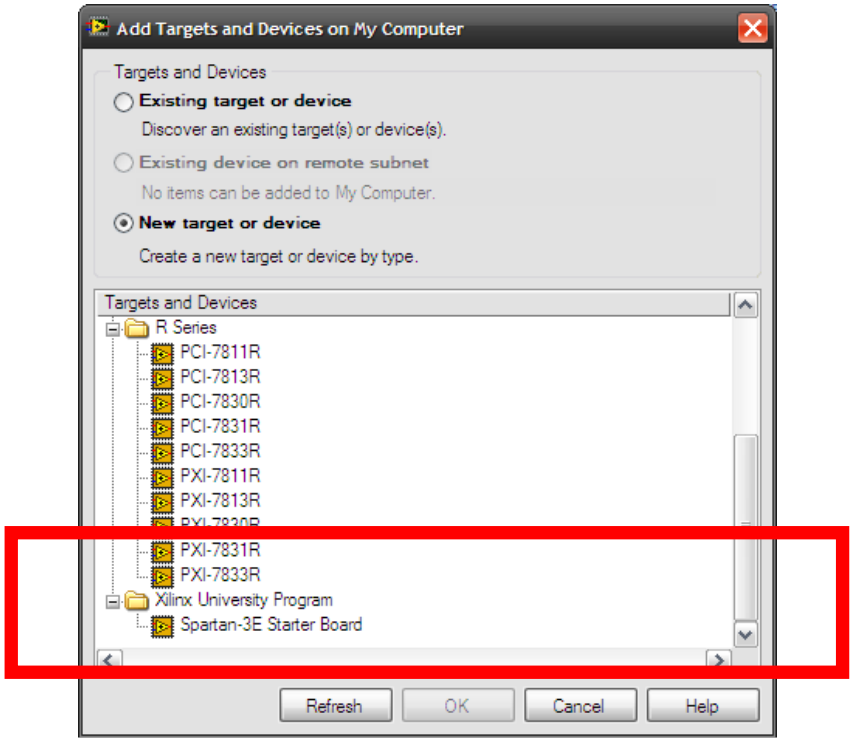


Figure 7. Xilinx SPARTAN-3E Starter Board in LabVIEW FPGA

Additionally, we have developed support for all the peripherals on the Xilinx SPARTAN-3E Starter Board in this plug-in. Figure 8 shows all the of peripherals that we support on the plugin for LabVIEW FPGA.

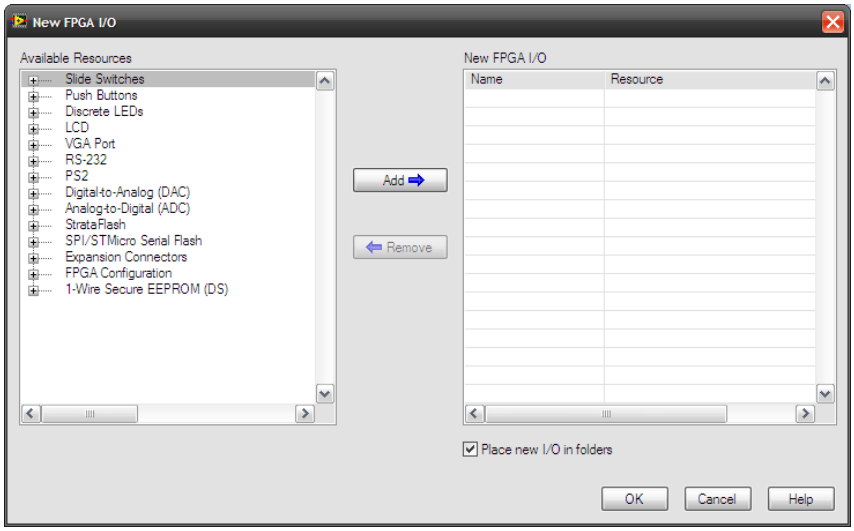


Figure 8. Peripherals of the SPARTAN-3E for LabVIEW FPGA

We have also taken care to build this plug-in so that it is compatible to work with all of the internal LabVIEW FPGA systems. This means that educators are only limited by the size of the FPGA to create or reuse any existing programs created with LabVIEW for FPGA. One such example is shown in figure 9 wherein we are reusing an existing program to create a 16-bit counter that increments every time the button is pressed.

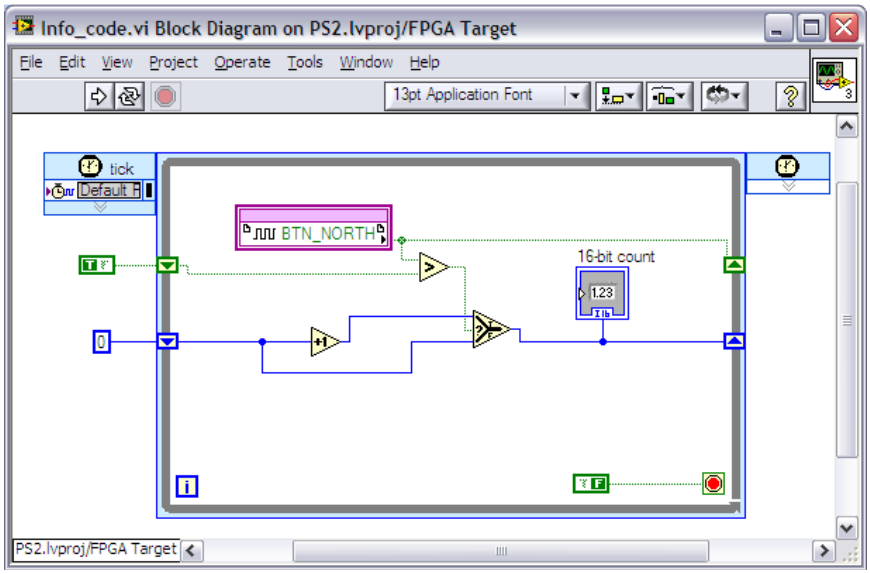


Figure 9. A counter program reused with LabVIEW for the Xilinx SPARTAN-3E

Conclusion

Design is becoming more and more crucial to creating well-rounded engineers. Real-world engineering projects revolve around creating designs and the engineers creating them are no longer just embedded experts from EE and CS. It is therefore critical to find innovative ways to teach design with intuitive tools and cost-effective hardware. In this paper, we present one such approach to teaching design, with graphical programming and FPGAs. We have created a plug-in for NI LabVIEW to target the Xilinx SPARTAN-3E XUP board and incorporated peripheral support and native LabVIEW FPGA integration thus enabling the educators and students to not only create their own FPGA designs but also reuse existing designs created in LabVIEW FPGA. We believe that such an approach to teaching design with graphical programming tools and reconfigurable hardware will foster design in the non-EE and CS disciplines where the steep learning curve of traditional tools have posed several barriers – both to teaching and learning.

References

[1] Rapid Prototyping of an FPGA based sensor system for Biomedical Monitoring, Newman, Kimberly; Laramie, Nathan; Medina, Casey, WSEAS Transactions on Biology and Biomedicine. Vol. 3, no. 2, pp. 97-103. Feb. 2006

[2] A new approach to detect-mitigate-correct radiation-induced faultsfor SRAM-based FPGAs in aerospace application, Yanmei Li, Dongmei Li, Zhihua Wang, National Aerospace and Electronics Conference, 2000. NAECON 2000. Proceedings of the IEEE 2000

[3] Position feedback control with a 'smart' controller based on an FPGA, Seals, R C, IEE COLLOQ DIG. no. 017, pp. 6/1-6/2. 1994

[4] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data flow programming language lustre. *Proceedings of the IEEE*, 79(9):1305–1319, 1991.

[5] U. Banerjee, R. Eigenmann, A. Nicolau, and D. Padua. Automatic program parallelization *Proceedings of the IEEE*, 81(2):211–243, 1993.

[6] G. Berry. The effectiveness of synchronous languages for the development of safety-critical systems. White paper, Esterel Technologies, 2003.
[7] Overview of LabVIEW, http://zone.ni.com/devzone/conceptd.nsf/webmain/F34045D2CC5357F486256D3400648C0F?OpenDocument&node=200067_us
[8] Overview of Xilinx SPARTAN-3E XUP Board, http://www.xilinx.com/products/devkits/HW-SPAR3E-SK-US-G.htm,
[9] What is LabVIEW FPGA Module, http://www.ni.com/fpga/what_is.htm
[10] Integrating existing VHDL code into LabVIEW FPGA, http://zone.ni.com/devzone/cda/tut/p/id/3483
[11] Download the Xilinx SPARTAN-3E Board plug-in for the LabVIEW FPGA Module, http://digital.ni.com/express.nsf/bycode/spartan3e?opendocument&lang=en&node=seminar_US