Graphical Design Environment for Reconfigurable Processor

Tu Le, Gregory Donohoe, David Buehler and Pen-Shu Yeh*
Institute of Advanced Microelectronic ECE/CAMBR University of Idaho
NASA GSFC code 567 *

Abstract - *The Field Programmable Processor Array (FPPA) is a new reconfigurable architecture developed by NASA/GSFC and the University of Idaho under ESTO funding. When fabricated onto a processor chip, the FPPA architecture promises high-throughput, radiation-tolerant, low-power data processing, for spacecraft instruments [1].*

The FPPA implements a synchronous fixed-point data flow computational model, which is not easily captured in procedural languages like C, but is easy to represent graphically. This motivates our Simulink-based design environment for the FPPA. In a process familiar to all Simulink users, the algorithm designer selects functional blocks from the menu, places them on a work screen, and connects them by drawing interconnect lines to create a FPPA data flow pipeline. A click of a button executes the data flow pipeline simulation, or translates the data flow pipeline to compiler codes, which can be used to configure the FPPA hardware. This tool will simplify programming the FPPA, suppressing architectural details.

I. Introduction

The Field Programmable Processor Array[1] (FPPA) was developed by NASA/GSFC and the University of Idaho to provide high-throughput, radiation-tolerant, low-power data processing, for spacecraft instruments [1]. The FPPA architecture employs sixteen reconfigurable processing elements (PE) with programmable interconnection, which allow the designer to generate a data flow pipeline to tackle complex applications such as the Fast Fourier Transform (FFT) and Focal Plane Array Sensor Readout Correction [2], [4]. Mapping application to the FPPA entails a daunting amount of architectural detail. This motives a graphical programming approach base on Simulink for the FPPA as a means to allow a design to be carried out entirely by the application programmer, without detailed knowledge of the FPPA architecture hardware or interface.

The rest of the paper is organized as follows. Section II, give an overview of the FPPA architecture followed by a discussion of the FPPA application development tools in section III. Section IV introduces the idea of graphical programming for a reconfigurable processor. The FPPA Simulink model, and Configuring the FPPA using the Simulink graphical design environment are explained in sections V and VI, respectively. In section VII, we demonstrate two application examples. Sections VIII and IX present future work and conclusions.

[1] Earlier versions of the FPPA were called the Reconfigurable Data Path Processor, or RDPP.

II. FPPA architecture

The FPPA reconfigurable processor implements a synchronous integer data flow pipeline. The FPPA employs sixteen reconfigurable processing elements (PEs), programmable interconnect, four 16-bit-wide bidirectional input/output (IO) ports and one 16-bit-wide dedicated output port, distributed on-board program memory, internal host interface and a micro-sequencer unit as well as the interface control signal unit [1]. Figure 1 shown a high level view of the FPPA architecture.

Figure 2 shows a simple example of the FPPA synchronous data flow pipeline. The five 16-bit IO ports provide a data interface to external devices and the 256x21 bit distributed on-board program memory synchronize the firing of the sixteen processing elements and five IO modules. In addition, sixteen interface control signals enable the FPPA to communicate with external peripherals. Up to five run-time program loops can be programmed into the micro-sequencer unit for looping through the program memory. Each of the processing elements contains a 17-bit signed multiplier, a 32-bit arithmetic logic unit (ALU) with carry out, delay elements, data formatter, switching logic and the signal conditioning logic [1].

The FPPA serves as an accelerator to a host computer via a byte-oriented communication protocol. The FPPA architecture works in two phases: Configuration and Execution. In the configuration phase, the processing elements are configured to a specific behavior, programmable interconnects configured to form a processing pipeline, and distributed on-board program memory for the PE and IO modules are loaded as well. The program memory specifies sequences of PE and IO module "firings" individually during the execution phase. (In dataflow terminology, the module "fires" when it latches new data and beings processing it.) In the execution phase, the FPPA reads and processes the input stream of data and writes the result to the programmed output ports [1].

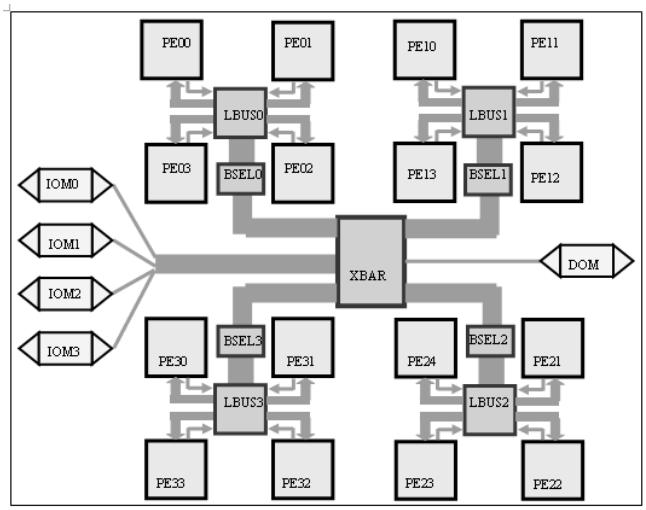

Figure 1: Glaze look at the FPPA architecture

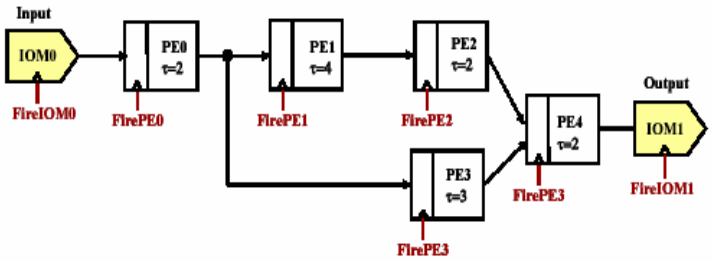

Figure 2: FPPA synchronous data flow pipeline

III. FPPA applications development tools

The reconfigurable FPPA chip will be fabricated in a radiation tolerant .25 u CMOS process [3]. Software support for applications development includes configuration and run-time compilers as well as a text-based stand-alone functional simulator [1]. This paper presents a graphical user interface (GUI) design entry tool for the FPPA based on Simulink.

Using the text-based stand-alone functional simulator and the compilers, we set out to validate the FPPA architecture concept by the development of two challenge applications: Fourier Transform Hyper spectral Imager data conversion, and Focal Plane Array Sensor Readout Correction [2], [4], [5].

In developing these applications we found the FPPA to be quite flexible, but the text-based application development tools are complex to use. Even for the simple four-tap finite impulse response (FIR) filter, an application programmer might take sometime to finish the job [6]. The FPPA application programmer must understand how to configure a PE to perform a task, how to connect multiple PEs together to form data flow pipeline, and how to program run-time behavior. The FPPA programming process requires a lot of book-keeping and can introduce human error. Also it is difficult to visualize the data flow pipeline through the coded texts. The graphical programming environment simplifies the job enormously.

IV. Graphical programming

We developed the FPPA Simulink model based on a floating point computational data flow. The physical FPPA architecture is built on a fixed-point computation data flow. Using floating point in Simulink simplifies the FPPA model complexity tremendously, and gives the FPPA algorithm designer a means to compare his/her ideal FPPA implementation to a Matlab or C reference. Future versions of this environment will also permit fixed-point simulation that faithfully models the behavior of the FPPA data path.

Figure 3 shows the FPPA graphical programming design flow, which demonstrates a process that takes the run-time or program memory information from the Simulink application model and feed it directly to the FPPA simulator. In addition, the data path (synchronous data flow pipeline consisting configured PEs and input and output ports) and the floating point input data format from Simulink are feed into the SIFopt tool, which then convert floating point data flow pipeline to an integer format in an optimized fashion [7, 8]. The output of the SIFOpt tool feed into the Configasm (Configuration Assembler) tool, which converts it to compiler code and feeds it to the FPPA standalone simulator. We use the PERL programming language to glue SIFOPT, ConfigASM, FPPA simulator and FPPA Simulink model together.

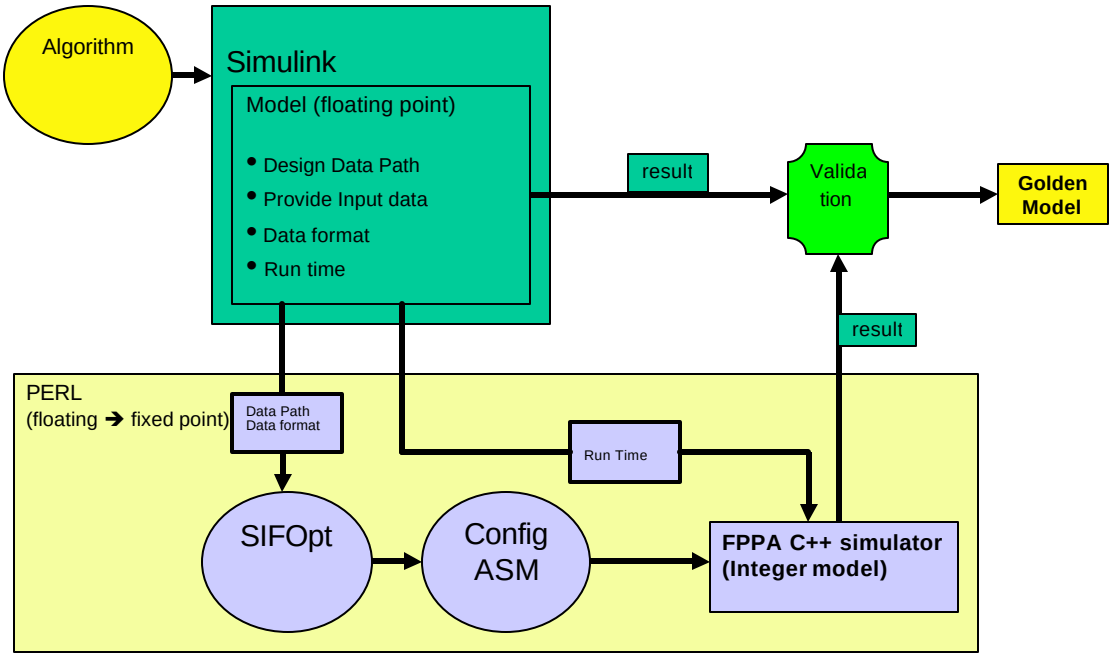

Figure 3: FPPA graphical programming design flow

V. FPPA Simulink Model

We model the FPPA architecture from the PE level, which allows us to incorporate the unconditional, conditional and runtime functionalities to the PEs model. Each of the PEs can be viewed as shown in Figure 4, where X, Y and W are input data streams to the PE and the "1/Z" represents an input latch, which synchronizes data transfer between PEs. Constants C0 and C1, data path function DP, and runtime firing RT are set in the configuration phase. Data path DP determines what function the PE will compute. For example, the PE might be configured to multiply Y by constant C1, and add X to the result. Runtime variable RT contains a firing pattern for the PE during the execution phase. The PE is composed of combinational logic; therefore we can think of the PE as a function that maps variables X, Y, W, C0, C1, DP and RT into a value that is passed to the output port. The PE "fires" when it latches new data at input ports X, Y and W; the result appears at the output after a period determined by combinational logic delays. If the PE doesn't fire, it holds the output value constant.

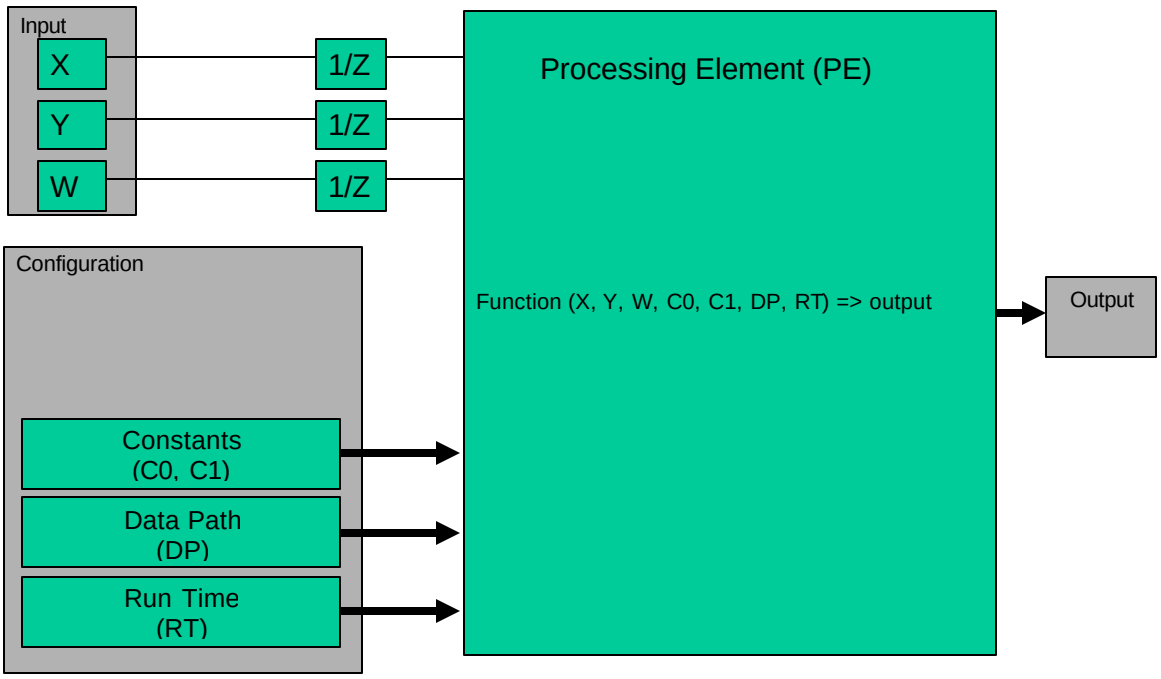

Figure 4: Input / Output signals diagram of the Processing Element

In addition to computation, the PE implements conditional or unconditional data path selection. Referring to Figure 4, under the unconditional category, the PE can be configured or programmed to perform such expressions as shown in Table 1, where ">>" and "<<" are right and left shifts; however, Table 1 does not represent an exhaustive lists of possible PE operations.

X+Y	X - Y	(X - Y) * W	(X + Y) >> 2
X*C1	Y + C0	(X + Y) * W	(X * Y + W) << 5
X and Y	X or Y	X * W	(X and Y) >> 7
X xor Y	neg X	X * Y + W	W << 13
X nor Y	neg Y	X * Y - W	Delay

Table 1: the sample list of possible behaviors of the PE

Under the conditional computation category; the PE performs an "if else" conditional statement base on two possible computational expression. The pseudo code of the "if else" statement, which the PE employs shown in Figure 5.

```
If (condition) then
        Perform expression A;
Else
        Perform expression B;
End if
```

Figure 5: The PE pseudo code for the conditional computational

VI. Configuring FPPA using the Simulink graphical design environment

We took advantages of Simulink's rich graphical user interface (GUI) programming capabilities and programmed a user friendly GUI that allow the FPPA algorithm designer to configure PEs and connect the configured PEs using the familiar click and drag operations to construct a synchronous data flow pipeline. We implemented different PE operations as Simulink functional blocks. Figure 6; shown the some of the PE blocks currently available in the FPPA PE Simulink library models.

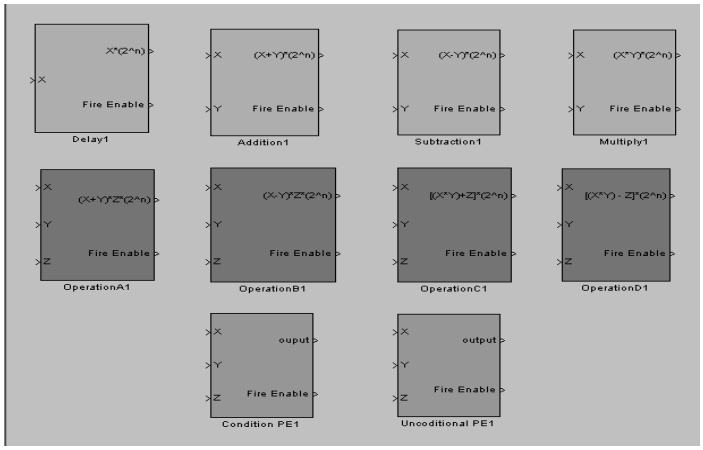

Figure 6: FPPA simulink model of the PE

Figure 7 shown how we would configure the expression Z * (C0 + Y) where C0 = 10 and also note that PE is set to fire on every clock cycles because the Fire Pattern is set to [1]. However, if the Fire Pattern is set to [1 0], the PE will fire on every other clock cycles; on the alternate cycles, it will be hold the last input data in the PE internal register. Likewise if Fire Pattern is set to [1 0 0] the PE is set to enable one every three clock cycles.

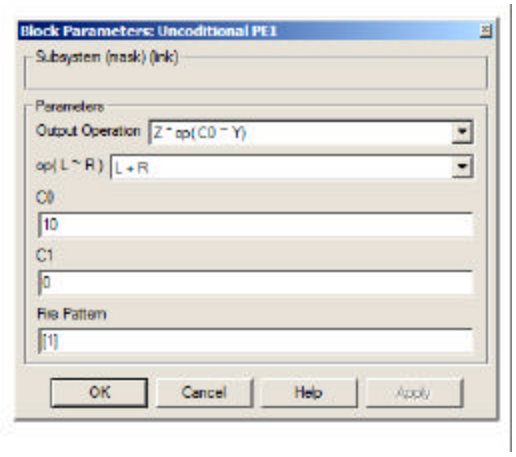

Figure 7: Unconditional PE configuration GUI, which configures PE to perform Z * (C0 + Y) where C0 = 10 and the PE is set to enable on every clock cycle.

VII. FPPA example applications using Simulink graphical environment

For this section, using the FPPA's PE Simulink library along with the Simulink graphical environment we will demonstrate graphical programming for the reconfigurable processor via a four tap FIR filter and a down sampling application. Note that for each of these applications we will implement the floating point Simulink model, and not convert it to a fixed point model.

Application One: 4-tap finite impulse response (FIR) filter

The four tap FIR filter equation that suit to implement in the FPPA architecture is shown below:

Let

w1, w2, w3 and w4 be the FIR filter coefficients
X be the input value
Y be the output value

Then the 4-tap FIR filter can be expressed as [6]:

$$Y = w0 \cdot x(k) + w1 \cdot x(k-1) + w2 \cdot x(k-2) + w3 \cdot x(k-3)$$

Using the expression above of the four tap filter with the coefficients as .15, .35, .15 and .35, respectively; we constructed Figure 7, which produces the following pipeline:

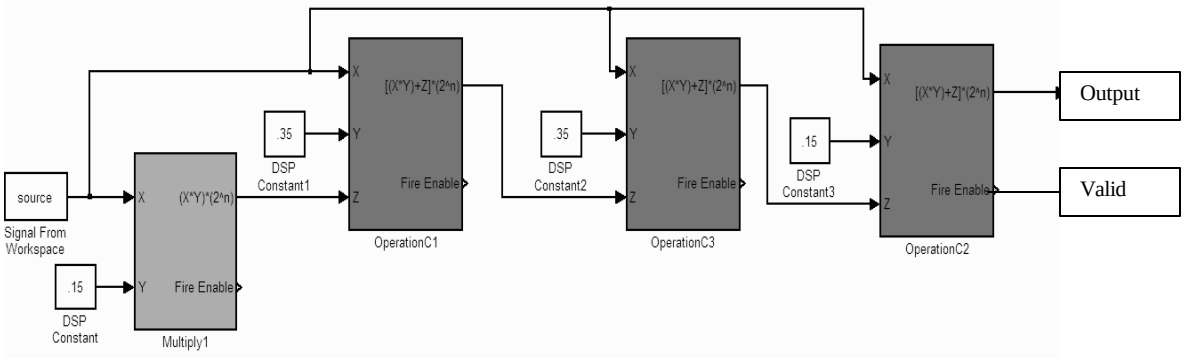

Figure 8: FIR filer data flow pipeline using the FPPA architecture

The configuration, or function, of each of the PE is shown on the top right corner of each Simulink model adjacent to the output port, as shown if Figure 8. Furthermore, each PE is configured to fire on every cycle, and the "valid" output signal from the FIR filter is *valid* = 1 if the "Output" signal is stable to read; otherwise the *valid*=0 indicating the "Output" signal is not ready to read. Figure 9 and 10 show the input and output signal from the FIR filter, respectively.

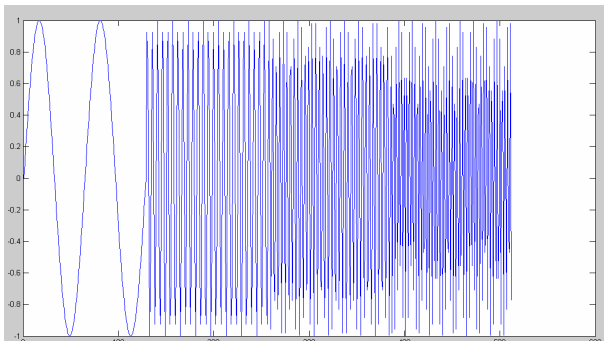

Figure 9: Input source signal with 512 samples

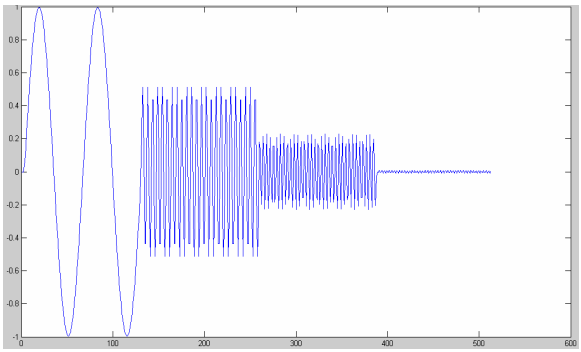

Figure 10: the FIR filtered output signal

Application Two: 4 taps FIR filter with down sample by 2

For this section we are implementing a same filter shown in the previous section; however, we also incorporate down sampling by a factor of 2 at the output of the FIR filter. This combination of an FIR filter followed by down sampling is typical of a wavelet decomposition operation. The block diagram of application two are shown in Figure 11.

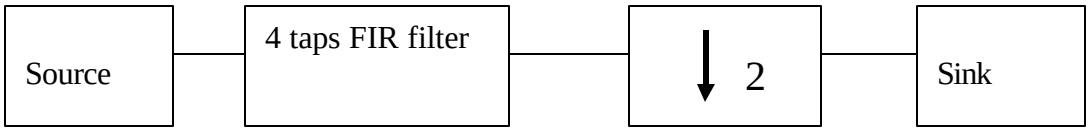

Figure 11: High level signal flow diagram of the FIR filter with down sample by 2

We use the firing pattern concept to implement down sampling; that is to set the fire pattern of the last PE of the data flow pipeline to [1 0], which mean that the last PE will fire the PE to process data on every other clock cycle. Figure 12 shows the FPPA Simulink model for the FIR filter followed by down sample by a factor of 2. Using the input signal shown in Figure 9, and running the simulation with the data flow pipeline shown in Figure 12 produces the result in Figure 13. Note that the output wave form is now 256 samples.

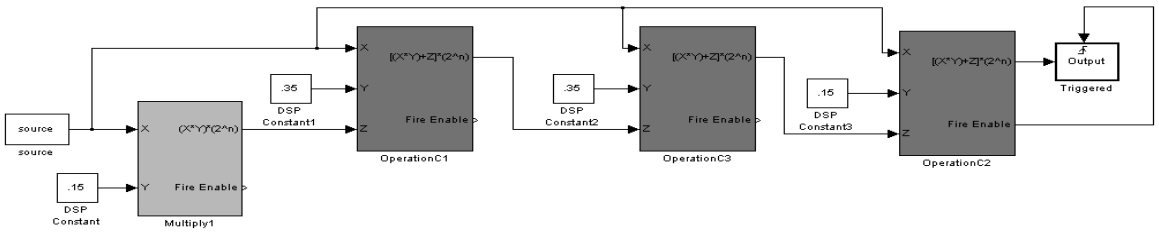

Figure 12: FIR filter follow by a down sample by 2

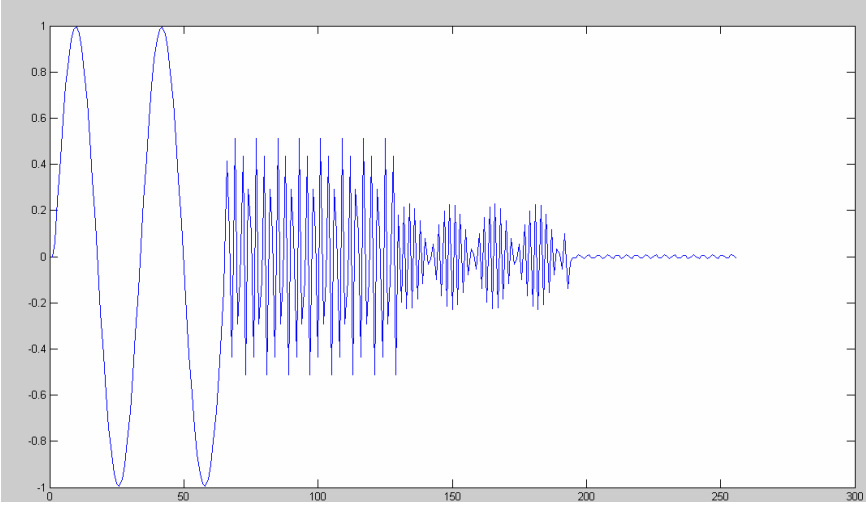

Figure 13: Output waveform of the FIR filter follow by a down sample by a factor of 2

VIII. Future work

Up to now, the majority project effort aimed on the functionality and validation between floating point and integer base design as well as demonstrated how graphical programming is intuitive for reconfigurable processing; especially the FPPA. To continue making the reconfigurable processor FPPA graphical design environment easy to use, flexible and practical for algorithms designer we are particularly interest in integrate three new features for the FPPA graphical interface.

- Extend the FPPA Simulink model to simulate a fixed-point data flow pipeline, which faithfully model the functional behavior of the FPPA chip.
- Develop a cook book for optimized common used signal processing components such as; trigonometry functions, matrix computation, log, accumulator, up and down sampling, iteration methods, etc.
- Compile the Simulink model into configuration and run-time files for the FPPA.
- Implement automatic optimization features, to optimize for speed and/or physical resources.

IX. Conclusions

This paper presented a graphical design environment for programming the Field Programmable Processor Array; enable a programmer to generate applications for the FPPA in an intuitive and visual fashion, while suppressing unnecessary architectural detail. This graphical programming environment was illustrated on two three sample problems. This programming method promises to greatly simplify programming the FPPA, reducing error and increasing programmer productivity.

References

[1] Gregory Donohoe, Pen-Shu Yeh, "Reconfigurable Data Path Processor", Proc. NASA Earth Science Technology Conference, University of Maryland, August 29, 2001.

[2] Gregory Donohoe, John Purviance, Pen-Shu Yeh, "The Fast Fourier Transform on a Reconfigurable Processor", *Proc. NASA Earth Sciences Technology Conference, Pasadena*, CA, June 11-13, 2002.

[3] Gregory Donohoe, Pen-Shu Yeh, "Low-Power Reconfigurable Processor", *Proc. IEEE Aerospace Conference*, Big Sky, MT, March 9-16, 2002.

[4] Jagdish Sabde, David Buehler, Gregory Donohoe, "Focal Plane Array Sensor Readout Correction on a Reconfigurable Processor", Proc. *NASA Symposium on VLSI Design, Coeur d'Alene, Idaho, May 28-29, 2003.*

[5] Gregory W. Donohoe, Pen-Shu Yeh, "Sensor Data Processing on a Reconfigurable Processor", *Proc. NASA Earth Sciences Technology Conference*, Laurel, MD, June 24-26, 2003.

[6] Gregory Donohoe, David Buehler, "Reconfigurable Data path Programmable Tutorial Processor", A white paper, August 13, 2003.

[7] Gregory Donohoe, David Buehler, Stephen Bruder, "A Design Strategy for Fixed-Word-Length Data Paths", 9th NASA symposium on VLSI Design, Albuquerque, NM, Nov 8-9, 2000.

[8] Buehler, David M., *A Methodology for Designing and Analyzing Fixed-Point Implementations of Computational Data Paths*, Ph.D. Dissertation, University of Idaho, 2004.