# Developing FPGA-based Embedded Controllers using Matlab/Simulink

T. Barlas and M. Moallem
*Mechatronics System Engineering*
*Simon Fraser University*
*Surrey, BC, Canada*
*Email (corresponding author): mmoallem@sfu.ca*

**Abstract**
Field Programmable Gate Arrays (FPGAs) are emerging as suitable platforms for implementing embedded control systems. FPGAs offer advantages such as high performance and concurrent computing which makes them attractive in many embedded applications. As reconfigurable devices, they can be used to build the hardware and software components of an embedded system on a single chip. Traditional FPGA design flows and tools, requiring the use of Hardware Description Languages (HDLs), are in a different domain than standard control system design tools such as MATLAB/Simulink. This paper illustrates development of FPGA-based controllers by utilizing popular tools such as MATLAB/Simulink available for the design and development of control systems. The capability of DSP Builder is extended by developing a custom library of control system building blocks that facilitates rapid development of FPGA-based controllers in the familiar Matlab/Simulink environment. As a case study, this paper presents how the tools can be utilized to develop a FPGA-based controller for a laboratory scale air levitation system.
**Keywords:** Embedded controllers, control firmware/software design, computer aided design tools

## 1. Introduction

Embedded control systems are found in a wide range of applications such as consumer electronics, medical equipment, robotics, automotive products, and industrial processes (Chan, Moallem, & Wang, 2007). Embedded control systems typically use microprocessors, microcontrollers or Digital Signal Processors (DSPs) for their implementation. For such systems, control algorithms are implemented as software programs that execute on a fixed architecture hardware processor. The processor itself is connected to various peripherals such as memories, Analog to Digital converters, and other I/O devices. Alternatively, FPGAs are increasingly becoming popular as implementation platforms on which the control algorithms can be implemented by programming reconfigurable hardware logic resources of the device. FPGAs have characteristics that make them suitable for realizing

hardware implementations of algorithms and systems. They offer excellent features such as computational parallelism, reconfigurable customization, and rapid-prototyping (Chan, Moallem, & Wang, 2007; Tessier and W. Burleson, 2001). Recently, there has been a growing interest in developing FPGA-based control systems. Casalino, Giorgi, Turetta, & Caffaz (2003), and Oh, Kim, & Lim (2003) used an FPGA as part of an embedded solution for controlling the motion of a four-fingered robotic hand. Koutroulis, Dollas, & Kalaitzakis (2006) implemented a PWM generator on a FPGA that was capable of generating signals of frequencies up to 3.985 MHz with a duty cycle resolution of 1.56%. Tjondronugroho, Al-Anbuky, Round, & Duke (2004) and Jung, Chang, Jyang, Yeh, & Tzou (2002) compared DSP-based and FPGA-based implementations of a multi-loop control strategy to control single-phase inverters. The simulation capability of the system-level tool System Generator from Xilinx was exploited by Ricci & Le-Huy (2002) to build the computational engine for variable-speed drives using FPGAs. Ramos, Biel, Fossas, and Guinjoan implemented a fixed-frequency quasi-sliding control algorithm on an FPGA for control of a buck inverter. In this application, the high switching frequency (ranging from 20 to 40 kHz) required fast computation of the control law, effectively ruling out software-based implementations on microprocessors or DSPs. In addition to control algorithms, FPGAs can also be used to implement various other components of the control system. For example, Zhao, Kim, Larson, & Voyles (2005) used a FPGA to implement a control system-on-a-chip for a small-scale robot.
While microcontrollers and DSPs have had the advantage of lower device cost over FPGAs, the gap in cost is narrowing with advancing technology, making FPGAs more and more attractive devices. Moreover, FPGA-based implementations may reduce overall cost of a system since multiple components of the design can be implemented as a system-on-a-programmable-chip. In some cases, FPGA-based implementations may give higher levels of performance for a design as compared to other implementations. For such cases, FPGAs may be the only choice for implementation because microcontrollers and DSPs would not meet the performance requirements, and Application Specific Integrated Circuits (ASICs) may have too high development costs.

## 2. Control System Design Tools and FPGA Design Flows

With FPGAs emerging as viable platforms for controller implementation, there exists a gap between what typical control engineers are used to in a control system design tool and what is required from existing FPGA tools and flows. Traditional FPGA design flows often require the use of Hardware Description Languages such as Verilog or VHDL for development. However, hardware description languages are in a low-level domain of bits, registers and logic functions as opposed to the high-level domain of signals, variables and mathematical functions. Nonetheless, high-level development tools for FPGAs are emerging to reduce this gap. Some of these tools are based on existing design environments such as Matlab/Simulink. In this work, a tool designed to work in Simulink, called DSP Builder, is studied as a development tool for FPGA-based controllers. DSP Builder allows the familiar and easy-to-use design environment of Simulink to be used for development for FPGAs. In this way, the gap between the tools used by control engineers and FPGA development environments is narrowed. Hence, FPGA platforms can utilized to build controllers alongside microcontrollers and DSPs.

## 3. Development of Custom Control Library for FPGAs

The mathematical frameworks for designing DSP systems and digital control systems are similar. This section looks at the DSP Builder tool in more detail and discusses its applicability as a high-level development tool for FPGA-based controllers. The capability of DSP Builder for developing controllers is further extended by developing a Custom Control Library in Simulink. The custom library provides DSP Builder-based blocks that can be used to rapidly develop FPGA-based controllers. Development using DSP Builder requires both Simulink and Quartus II software packages. The Simulink design flow is a block diagram based design flow where predefined blocks, grouped in different libraries such as Math Operations, Logic and Bit Operations, and Continuous/Discrete libraries, can be dragged and dropped into a canvas model file. The blocks are then appropriately interconnected with virtual wires that propagate signals amongst the blocks. Using this method, any kind of static or dynamic system in various domains such as signal processing, control system, imaging, and communications can be specified. The entire system can then be simulated by using various numerical solvers to determine the time domain response of the system or the behavior of internal signals.

The DSP Builder provides special libraries of blocks for use in Simulink that are directly synthesizable into hardware logic for Altera FPGA devices. FPGA-implementable algorithms and systems can be developed by simply dragging and dropping DSP Builder Library blocks into Simulink model file and making desired connections between them. Each DSP Builder block has a direct HDL representation (in either Verilog HDL or VHDL) of the function it performs, i.e., the blocks encapsulate HDL modules. A special block, called SignalCompiler, when invoked, reads the Simulink model file and translates each of the DSP Builder blocks and their interconnections to corresponding HDL representations. Each of the DSP Builder blocks can have their parameters specified via dialog boxes in Simulink, and these parameter choices are propagated into their HDL representations. Finally, SignalCompiler combines the whole design into one HDL top level entity that can be processed through the FPGA design flow stages using Quartus II. The HDL entity will be functionally equivalent to the system in the Simulink model file when it executes on the FPGA (DSP Builder User Guide, 2005).

DSP Builder library blocks also come with bit- and cycle-accurate simulation models that can be invoked by the Simulink discrete-time numerical solvers to perform design simulation within the Simulink environment. Furthermore, for simulation purposes only, existing Simulink blocks (such as input sources) can be interconnected with DSP Builder blocks to perform richer simulations involving real-world subsystems that interact with the FPGA. This has an obvious advantage for embedded control system design in the sense that the expected behaviour of the DSP Builder-based controller can be simulated with a Simulink based plant model. Thus the response of the controller can be conveniently simulated and analyzed. This simulation uses Simulink's numerical solvers and is different from HDL-level simulation using RTL simulators. Nonetheless, because of the bit- and cycle-accuracy, the behaviour of the DSP Builder-based design in Simulink will match the behaviour of the generated HDL system when it executes on the FPGA. Thus the design can be simulated, analyzed, and then modified at a higher level of abstraction than at HDL-level. This process eliminates the need to go through FPGA design flow at each and every design iteration, resulting in reduced development time.

There are five main categories of blocks provided by DSP Builder: Arithmetic blocks, Logical operation and flow control blocks (in the Gate & Control library), Input, Output ports and bus manipulation blocks (in the IO & Bus library), Clock domain and sampling time blocks, and Blocks for storing signals and data (in the Storage library). Given a mathematical description of a controller (particularly in canonical form), the controller can be implemented by using three operators: multipliers, adders and delay elements. Hence any controller can be easily implemented in DSP Builder and realized as hardware on an FPGA. While the Multiplier, Adder and Delay blocks may be sufficient, development using these blocks only may be tedious and time-consuming. A major contribution of this work is to provide more sophisticated control system building blocks by developing what is referred to as the Custom Control Library in this paper.

## 3.1 Custom Control Library

DSP Builder library blocks offer basic functionalities that can be used to develop custom blocks for performing more complex functions. They can be utilized to develop a custom library of control system building blocks for quick implementation of FPGA-based embedded controllers.

An example of implementing a custom control library component is shown in Figure 1 where the "Parallel Adder" block adds the input *e(k)* to the previous input *e(k-1)*, which is produced from the "input_Delay" block, and then multiplied by *Ts/2* (=*T/2*). The output of the "Multiplier" block is added to the previous value of the output, which is stored and made available from the "output_Delay" block. Because the current output is fed back to the "output_Delay" block, the bit width of the signal into the block is indeterminate, and so needs to be explicitly defined. This can be achieved by the "AltBus" block in Figure 1. The bit width for "AltBus" is automatically chosen to be the same as the bit width of the result of the multiplier. The bit width for the output signal "output" is also chosen to be the same as that of the multiplier result.

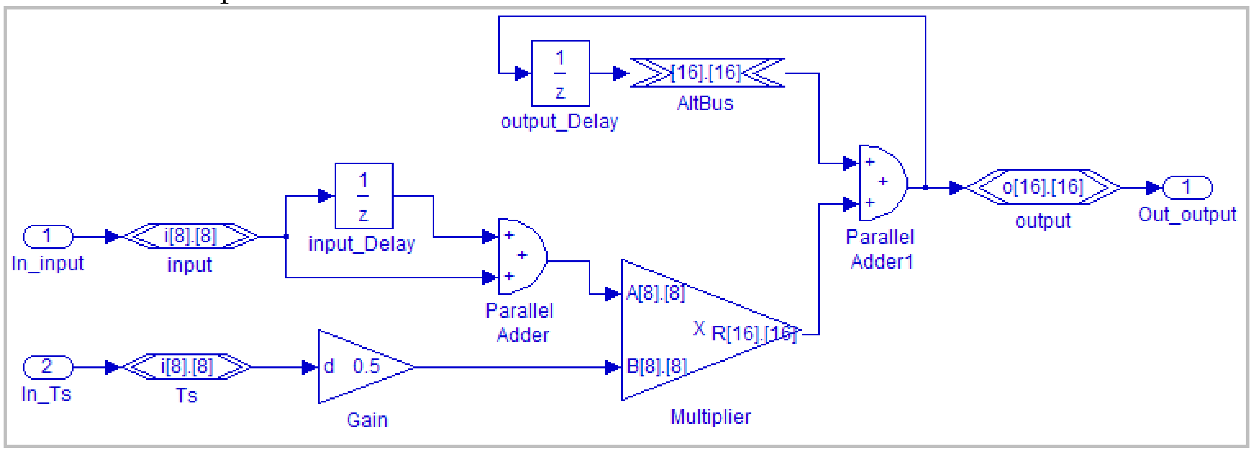

Fig. 1. Implementation of Custom Control Library Integrator (Trapezoidal) block using DSP Builder blocks.

Widely used transfer function blocks such as lead, lag, or PID, can be implemented in the Custom Control Library. For example, Figure 2 shows a discretized PID block as it appears in Simuink. The implementation uses the Integrator block (Trapezoidal implementation) and the Discrete Derivative block from the Custom Control Library. The block has inputs for specifying the gains (input ports labeled "Kp", "Kd" and "Ki"), the sample rate (port "Ts"), the sampling frequency (port "fs") and the input to the block (labeled "input"). The "output" port is the control signal computed by the PID algorithm. The bit widths for all the ports can be specified via the block's dialog box. Hardware implementation details such as the number of bits can also be specified from the dialog box.

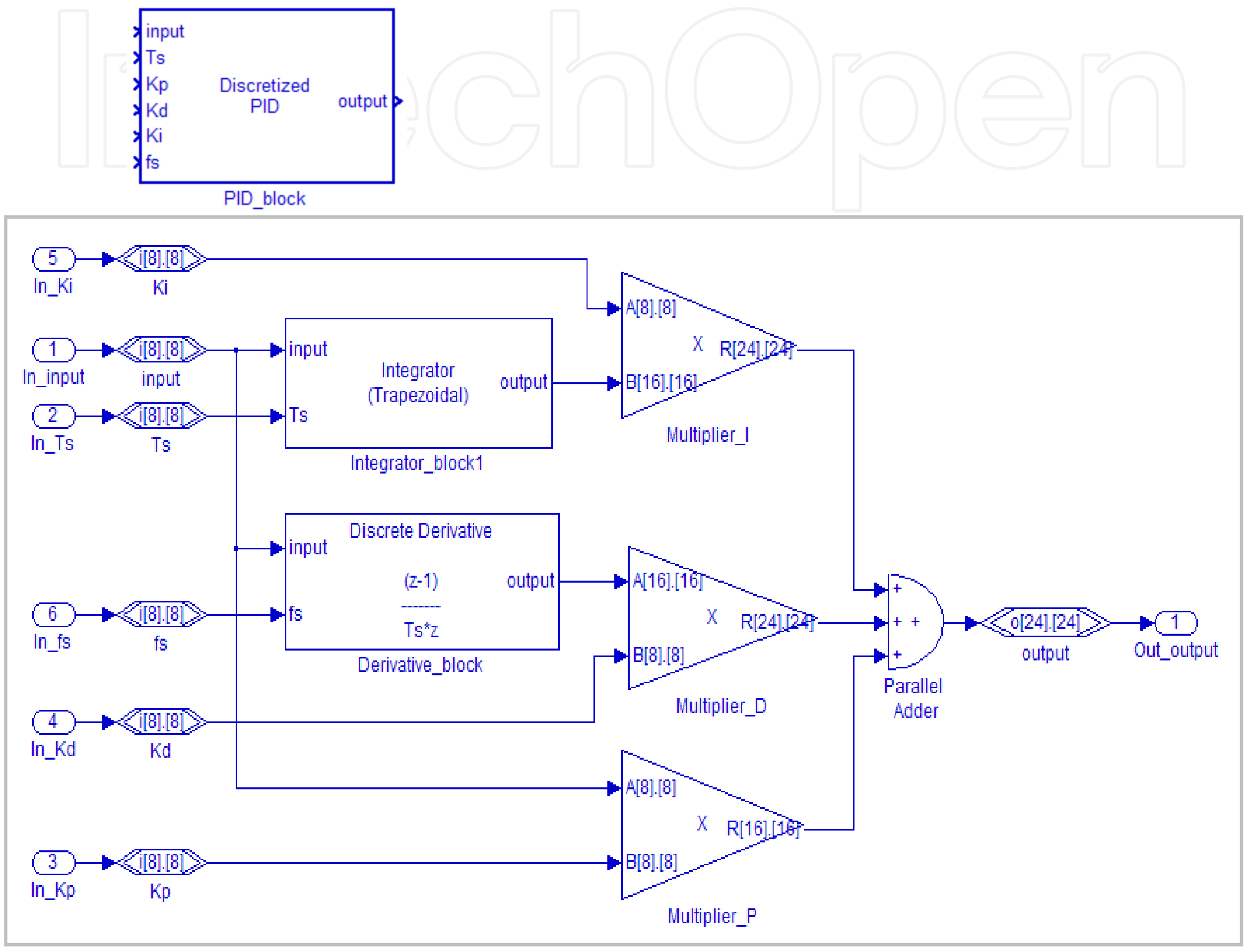

Fig. 2. Discretized PID controller.

### 3.1.1 Pulse Width Modulation Generator Block

Pulse Width Modulation (PWM) is frequently used in digital control systems, especially in DC motor drives. A digital PWM Generator is implemented for the Custom Control Library. Figure 3 shows the block as it appears in Simulink. In Figure 3, the counter counts up to a value of Clock Frequency divided by the PWM frequency. This value is set as the modulo of the counter so that the counter resets once it has counted up that value. While the count value is less than a value representing the current duty cycle in terms of the number of clock cycles, the output is logical high. Otherwise, the output is logical low. The number of clock cycles representing the current duty cycle is calculated by multiplying the duty cycle input in percentage by a value which, at 100% duty cycle would give the required clock cycles to output a logical high for the entire duration of the pulse period.

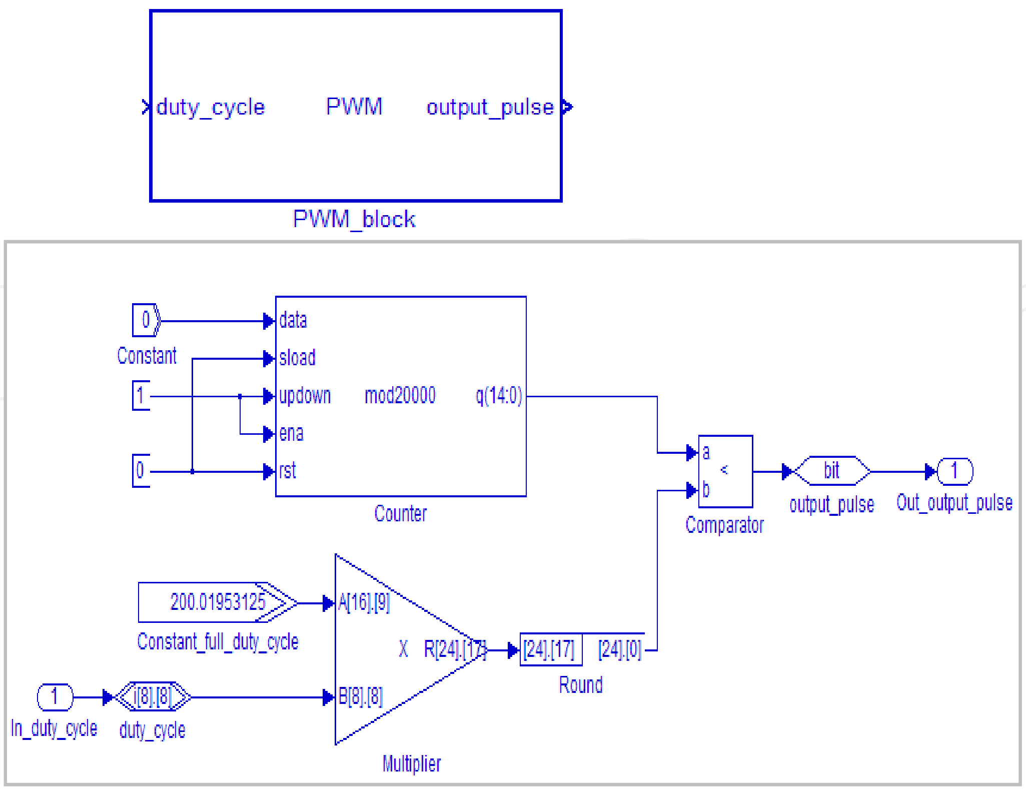

Fig. 3. Implementation of Custom Control Library PWM Generator block using DSP Builder blocks.

### 3.1.2 Analog to Digital Converter Controller Block

Analog to Digital (A/D or A2D) converters are necessary in digital control systems. There are various types of A/D converters based on different implementation technologies. Nonetheless, the converted data is transmitted to digital devices either as a parallel word over a digital bus or as a serial bit-stream over a single digital line.

Serial A/D converters are cheaper and simpler (in terms of circuit board placement) and therefore more commonly used. Hence, a block was implemented for the Custom Control Library that can be used to interface a FPGA with a serial A/D converter. When connected to a serial A/D converter, the master device (microcontroller, or, in this case FPGA) pulls the CS signal low to create a falling edge, which indicates to the A/D converter to initiate the conversion process. The A/D converter starts the conversion process, and after a brief waiting period, outputs the data word on the DOUT line in the form of a bit-stream starting with the most significant bit and ending with the least significant bit. The data transfer is usually synchronized to the falling edges of the CLK input from the master device. Then there is a wait period in which the DOUT line goes to a high impedance state. After the wait period, the CS signal has to go high again so that the next conversion can be initiated. The rate of data conversion is determined by the CLK clock signal. The goal of the custom ADC interface was to design a parameterizable block that can be used to interface FPGAs with a variety of SPI serial A/D converters.

Figure 4 shows the block as it appears in Simulink. The block has three inputs for configuring the A/D converter's registers ("In_initialize", "In_range_spec" and "In_control_spec"); an input for starting the conversion ("In_acquire"); a reset input; and an input for the serial converted data that comes from the A/D converter. The block outputs two signals that are inputs to the A/D converter: the “Out_Cs_n” signal is CS; “Out_DIN” is DIN. The third output of the block, “Out_data_out” is the converted data outputted as a parallel word. The input at “In_range_spec” is the binary data that needs to be loaded onto the A/D converters range register (for those converters that have programmable analog input ranges). Similarly, the input at “In_control_spec” is the binary data for the A/D converters control register (which is used to configure the A/D converters if it can operate in multiple modes).

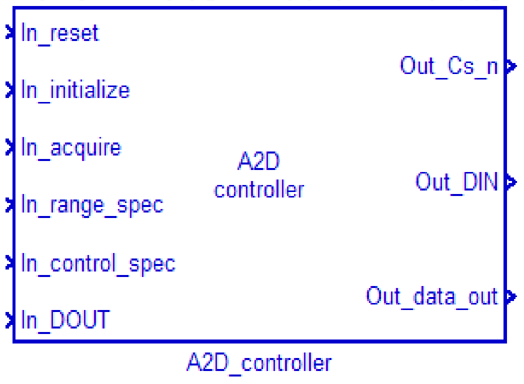

Fig. 4. A/D Controller block of the Custom Control Library.

In order to implement the interfacing process, the state machine shown in Figure 5 was used. The outputs at each state are also given. The converted word in parallel form (at the “Out_data_out” port) is available once the system moves out of the Acquire state. The internal implementation is achieved by using two State Machine blocks, two Parallel to Serial blocks to convert the range and control data into bit-streams for outputting on the “Out_DIN” port, and a Shift Taps block is used to convert the incoming serial data on “In_DOUT” into a parallel form.

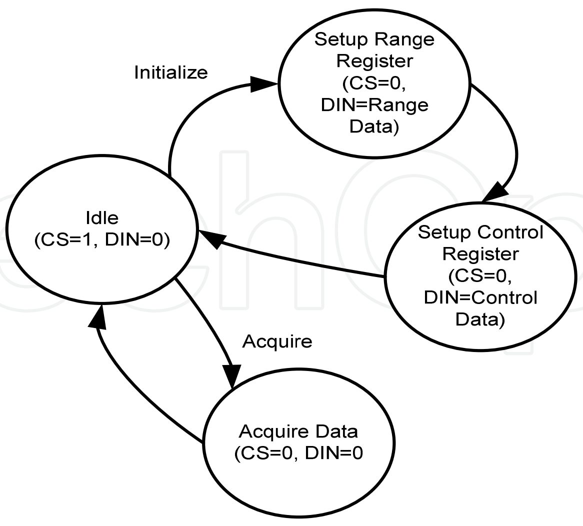
Initialize
Setup Range Register (CS=0, DIN=Range Data)
Idle (CS=1, DIN=0)
Setup Control Register (CS=0, DIN=Control Data)
Acquire
Acquire Data (CS=0, DIN=0

Fig. 5. State machine that implements SPI interface for A/D Controller block of the Custom Control Library.

### 3.2 Resource Usage of Custom Control Library Blocks

Logic Elements (LEs) and embedded multipliers are the two main resources used frequently in different designs. All FPGAs contain LEs but some devices may not contain embedded multipliers. For such devices, the multiplication operation has to be implemented by a group of LEs. However, multiplication is a resource intensive operation in terms of LE usage; hence FPGAs with embedded multipliers are preferred for control systems because the LEs can be freed up for more efficient use by other tasks. The results reported here are for individual implementation of the blocks on the Stratix EP1S80 FPGA chip. Each block was configured to use its default parameters. The Stratix family of chips contain DSP blocks that can be used to perform multiply, multiply-add, and multiply-accumulate operations found commonly in DSP systems. Each DSP block can implement 9×9 fixed-point multiplication with each input 9 bits wide and the output 18 bits wide, for any fixed-point bus format (unsigned integer, signed integer or signed fractional). For inputs greater than 9 bits, two or more DSP blocks configured as 18×18 multipliers or as 36×36 multipliers are used. Each 18×18 multiplier uses two 9×9 multipliers and each 36×36 multiplier uses eight 9×9 multipliers.

The results indicate that a very small number of LEs are used by the custom blocks when embedded multipliers are used. However, when embedded multipliers are not used, the LEs usage increases dramatically. This illustrates the resource intensiveness of the multiplication operation when implemented by LEs. Given a FPGA chip, a designer may decide to implement some blocks with embedded multipliers and some blocks without embedded multipliers, in order to fit the design into the FPGA chip. For example, the designer may implement the discretized PID block using embedded multipliers, because otherwise it would consume too many LEs (3036 LEs as compared to 205 LEs). However, the designer may implement the PWM Generator block without using embedded multipliers, thereby saving 8 embedded multipliers but only increasing the LE requirement from 40 LEs to 239 LEs.

## 4. Implementation Case Study

In order to demonstrate usage of the Custom Control Library to develop FPGA-based embedded controllers in the Simulink design environment, a control system was designed and developed for controlling the position of a levitating ball. This chapter describes the air levitation apparatus and the implementation of the control system. It is shown that all the necessary components of the control system are developed using system-level tools and then implemented on a single FPGA chip.

The air levitation apparatus consists of a transparent hollow tube, 43.5 cm in length, held vertically by a base which consists of a fan that blows air upwards into the tube. A table tennis ball is placed inside the hollow tube and has one degree of freedom to move either up or down. The thrust from the air flow causes the ball to be pushed upwards against the downward gravitational pull. The position of the ball is measured by an Infrared (IR) sensor placed at the top of the tube (the sensor does not block the air flow). There is an onboard serial A/D converter that can be used to convert the sensor voltage to a digital word; there are four connectors on the apparatus that enable an external digital device to interface with the converter. Figure 6 shows a block diagram of the apparatus with a picture of the system given in Figure 7.

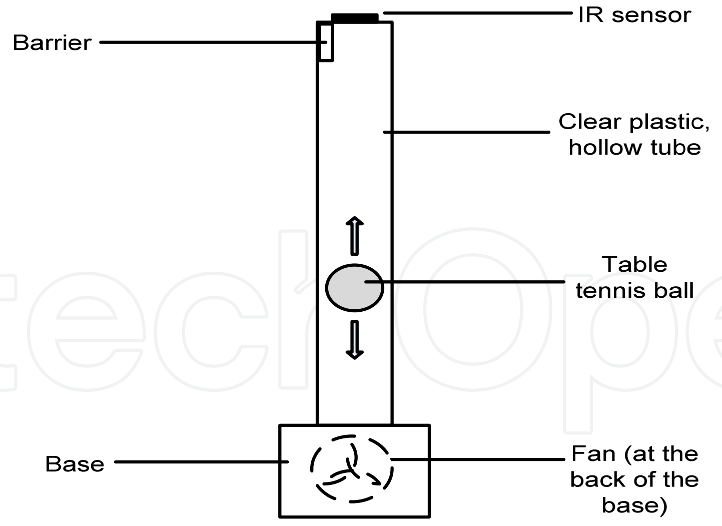

Fig. 6. Block diagram of the Air Levitation apparatus.

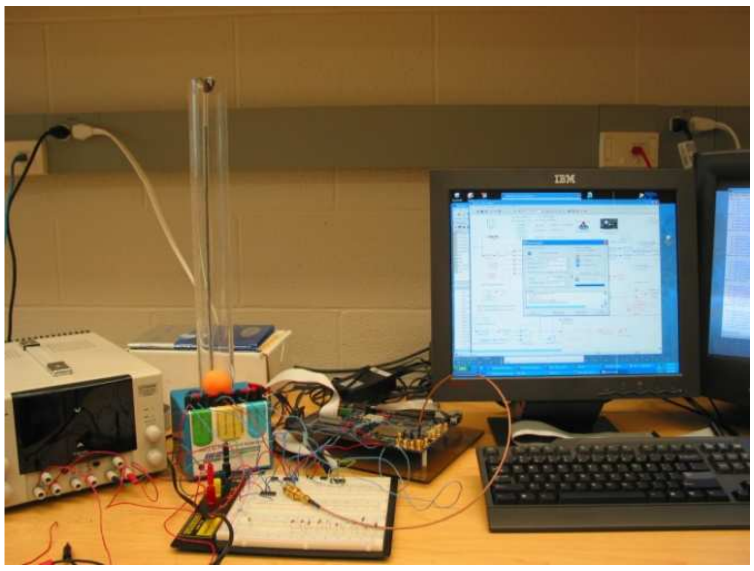

Fig. 7. Picture of the FPGA-based system.

The IR sensor is a Sharp GP2D120 Optoelectronic distance measuring sensor that outputs an analog voltage. When the ball is at the highest position (5.5 cm from the sensor), the sensor output voltage is 2.5 Volts. When the ball is at the lowest position (43.5 cm from the sensor or 38 cm from the barrier), the sensor voltage is 0.2 Volts.

The onboard A/D converter is a 12-bit, 250 kilo-samples-per-second, 2-channel serial converter with a 4-wire SPI compatible interface. It is capable of converting voltage from 0 to 5 Volts; thus it has a resolution of 1.2207 mV ($5V/2^{12bits}$). The four interface pins are: an external clock input for synchronizing the serial data transfer; a data input pin for shifting-in the A/D configuration 2-bit data word into the chip; a data-out pin for shifting-out the A/D converted result; and a convert input used to signal the device to start the conversion. The rate of conversion depends on the clock frequency: it takes 12 clock cycles plus the conversion time (less than 3.2 microseconds) for the data to be available.

The onboard fan uses a power supply of 8 Volts but its speed is controlled by a Pulse Width Modulated (PWM) signal that ranges from 0 to 5 Volts. The fan is constrained to rotate in one direction; therefore it cannot accept negative voltage at its PWM input.
The objective of the control system is to stabilize the position of the ball within the tube at any level. Figure 8 shows the block diagram of a closed-loop control system architecture. The input to the controller is the error signal between the desired voltage (representative of the desired position of the ball) and the measured voltage (representative of the actual position of the ball). The output of the controller is the control signal; which needs to be converted to a PWM signal to control the speed of the fan such that the upward thrust on the ball tries to negate the downward pull due to gravity and keep the ball level at a desired position.

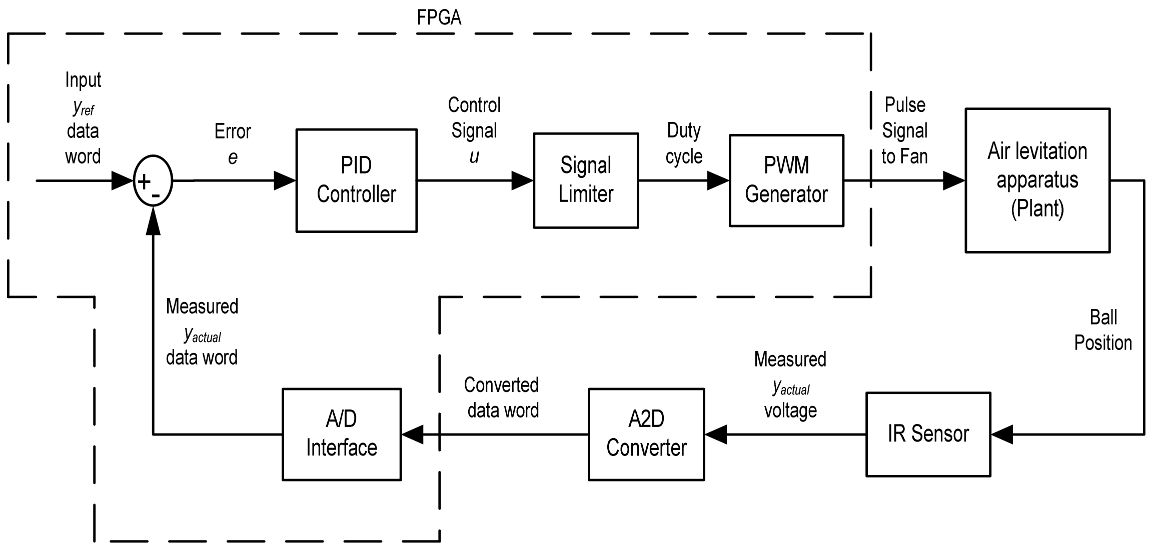

Fig. 8. Control system architecture used for air levitation controller.

The control system components were implemented on a single FPGA chip. The FPGA interfaces with the A/D converter on the apparatus to control its data conversion and acquisition. The 12-bit sampled data word acquired from the A/D converter is multiplied by the resolution of the A/D converter and converted into a fixed-point signed fractional number representing the real value of the sensor's voltage. The reference input $y_{ref}$, error $e$, control signal $u$, and all internal signals are represented in fixed-point signed fractional number format. $y_{ref}$ is the desired value of the sensor voltage. The computed error between $y_{ref}$ and $y_{actual}$ is fed into a discretized PID controller.

### 4.1 Controller Development on FPGA Board

The air levitation control system was implemented on an Altera FPGA development board. The board was part of the Altera DSP Development Kit Professional Edition, which had the Altera Stratix EP1S80 FPGA chip on it. The Stratix chip had 79,040 Logic Elements, 7,427,520 total RAM bits, 176 9×9 embedded multipliers, 12 PLLs, and 679 I/O pins. The development board had an onboard 80 MHz oscillator that was used as the FPGA's main clock source. The Stratix EP1S80 FGPA is a high-end and expensive chip that is meant for development of

large and complex DSP algorithms and systems. However, other FPGAs such as Cyclone III that are low-end, inexpensive and with fewer number of device resources.
The discretized PID, PWM and A/D controller blocks of the Custom Control Library are all used for the corresponding components of the control system in Figure 8. The system is developed in Simulink by dragging and dropping the custom blocks and DSP Builder library blocks onto a model file, configuring their parameters and making connections between the blocks. Figure 9 shows the Simulink model file of the developed system. The model file was translated into VHDL using the SignalCompiler block of the DSP Builder library and a Quartus II project file was generated. The Quartus II software was subsequently used for the FPGA design flow steps of Analysis, Synthesis, Placement & Routing, and Programming the design onto the FPGA chip.

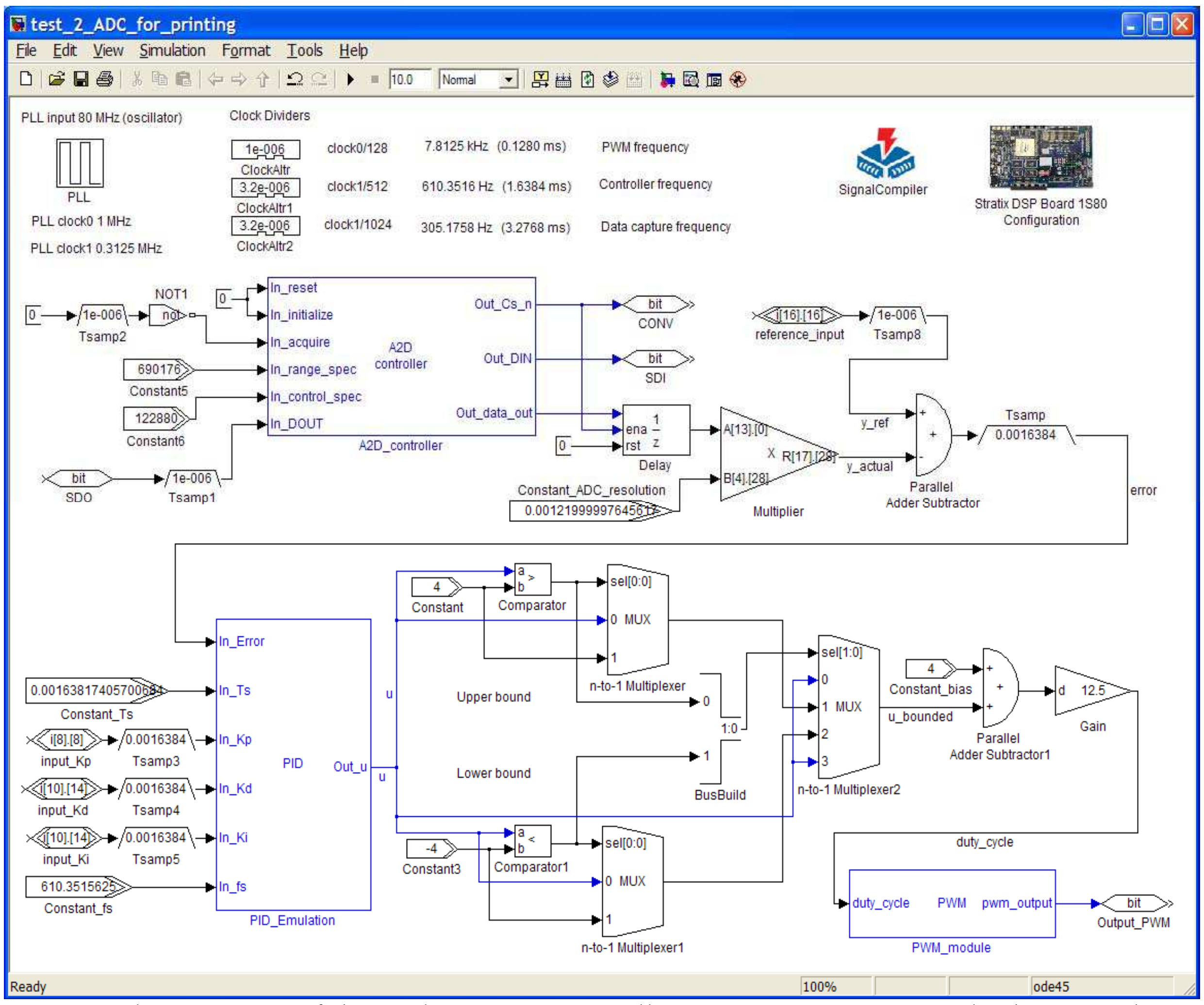

Fig. 9. Implementation of the air levitation controller using Custom Control Library and DSP Builder blocks.

The A/D Controller block is configured to operate at a clock frequency of 1 MHz (which is generated by a PLL block that divides the 80 MHz clock signal from the oscillator on the development board). It takes 21 clock cycles for the entire conversion process (the conversion time plus the serial data read time); hence the rate of sampled data acquisition is

47.619 kHz. The 12-bit sampled data is in unsigned integer format. It is converted to 13 bits signed fractional format but with zero bits for the fractional part (the most significant bit is the sign bit).
The discretized PID block is configured to operate at a frequency of 610 Hz resulting in a controller period of 1.6 milliseconds. The frequency 610 Hz is generated by a clock divider. The error signal into the PID block is represented using 16 bits for the integer part and 16 bits as for the fractional part (to allow for sufficient range and precision). The control signal output of the PID block is represented using 24 bits for the fractional part and 24 bits for the integer part allowing for sufficient range and precision. The control signal is then bounded and shifted into a positive value and then converted into a duty cycle. The duty cycle value, represented using 8 bits for the fractional part and 8 bits for the integer part, is inputted to the PWM block, which generates a single logical pulse whose width corresponds to the duty cycle. The PWM module uses a clock frequency of 1 MHz and the pulse frequency is 7.8125 kHz. This allows the duty cycle signal to have a resolution of 0.78%. The frequency 7.8125 kHz is also generated by a clock divider.

### 4.2 System Implementation Results

The control system was synthesized, placed & routed, and programmed into the Stratix EP1S80 FPGA. The FPGA-based controller was connected to the air levitation apparatus. Three sets of tests were carried out to evaluate the performance of the control system. The first set investigated the step response, the second set investigated the steady state response, and the third set of tests investigated the response to an external disturbance. A disturbance was created by manually holding a hand at the top of the tube to obstruct the air flow. All three tests used the following gains for the discretized PID controller: $K_P = 2$, $K_I = 0.038$ and $K_D = 3$. Figure 10 shows the response of the system once the ball has stabilized at a desired reference value of 0.6V, which represents the mid-point of the effective length of the tube (19 cm from the bottom).

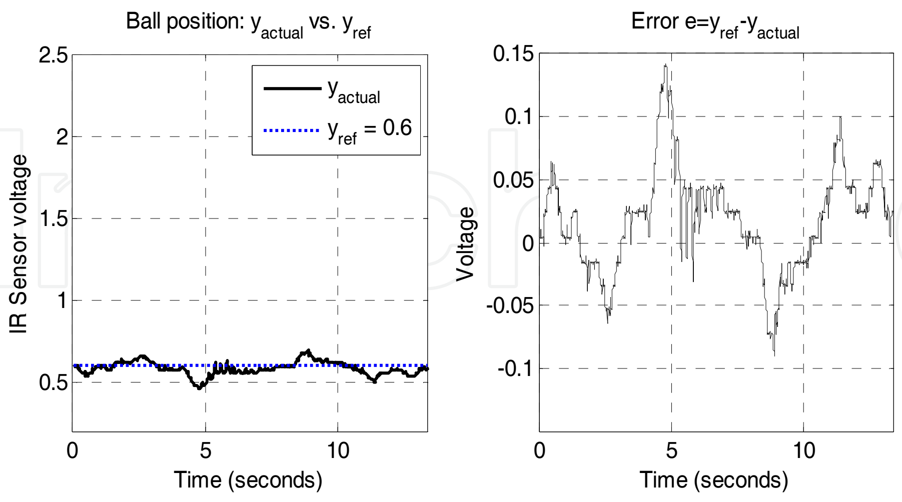

Fig. 10. Steady state response of air levitation controller to a reference input of 0.6V.

### 4.3 Reduced-size Controller

Large bit widths were used earlier in the development in order to capture signals at higher precisions for purposes of data plotting and analysis. However, such large bit widths were not required for the custom controller to operate correctly. Hence, in order to reduce the embedded multiplier usage, the custom controller of Figure 9 was modified to use smaller bit widths for its signals. The smaller bit widths take into account the possible range of the signals as the controller operates. For instance, the reference input bit width was changed from [16].[16] to [3].[13] because 3 bits for the integer part were sufficient to represent references from 0.2V to 2.5V, and 13 bits for the fractional part were sufficient to represent the reference in increments of $2^{-13}\mu V$. In addition to changes to the bit widths, certain multiplier elements in the custom controller were purposely configured to be implemented as Logic Elements instead of dedicated embedded multipliers. All changes were made in Simulink using the dialog boxes of the blocks, and were propagated to their HDL representations automatically.

The reduced custom controller required 1875 LEs and 14 9×9 dedicated embedded multipliers, as compared to 955 LEs and 70 9×9 embedded multipliers required by the original custom controller. This represents a 96% increase in LE usage, but a 400% decrease in 9×9 embedded multiplier usage. Consequently, the reduced custom controller can fit into a smaller and less expensive FPGA, such as the Cyclone II EP2C5, which contains 4,608 LEs and 26 9×9 embedded multipliers. Thus, the reduction in bit widths and tradeoffs between LE usage and embedded multiplier usage was worthwhile in this case. In conclusion, the reduced controller had more favorable resource usage but equivalent performance as the original controller.

## 5. Conclusions

The use of the system-level tool DSP Builder for high-level development of FPGA-based controllers was studied. The capabilities of the DSP Builder tool were further extended by developing the Custom Control Library. The custom library is comprised of widely used components such as discretized integrators, PID controller, PWM generator, and A/D controller. DSP Builder and the Custom Control Library together can be used to rapidly develop controllers in the familiar and standard Simulink design environment for FPGA implementation. An implementation case study demonstrated usage of DSP Builder and the Custom Control Library to develop a FPGA-based controller for an air levitation system in the Matlab/Simulink environment.

## 6. Acknowledgement

This research was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) and by the Canadian Microlelectronics Corporation (CMC), Kingston, ON, Canada.

## 7. References

Altera Corporation (2005). DSP Builder User Guide, Version 5.1.0.

Casalino, G., Giorgi, F., Turetta, A., & Caffaz, A. (2003). Embedded FPGA-based control of a multifingered robotic hand, *IEEE Int. Conf. on Robotics & Auto.m,* pp. 2786-2791.

Chan, Y.F., Moallem, M. & Wang, W. (2007). Design and implementation of modular FPGA-based PID controllers, *IEEE Transactions on Industrial Electronics,* 54(4), pp. 1898-1906.

Jung, S-L., Chang, M-Y., Jyang, J-Y., Yeh, L-C., & Tzou, Y-Y. (1999). Design and implementation of an FPGA-based control IC for AC-voltage regulation, *IEEE Transactions on Power Electronics,* 14(3), pp. 522-532.

Koutroulis, E., Dollas, A., & Kalaitzakis, K. (2006). High-frequency pulse width modulation implementation using FPGA and CPLD ICs," *Journal of Systems Architecture,* 52(6), pp. 332-344.

Oh, S-N., Kim, K-I., & Lim, S. (2003). Motion control of biped robots using a single-chip drive", *IEEE Int.l Conf. on Robotics & Autom.*, pp. 2461-2465.

Tessier, R., & Burleson, W. (2001). Reconfigurable computing for digital signal processing: A Survey, *The Journal of VLSI Signal Processing*, 28(1), pp. 7-27.

Tjondronugroho, A., Al-Anbuky, A., Round, S., & Duke, R. (2004). Evaluation of DSP and FPGA based digital controllers for a single-phase PWM inverter," in *Australasian Universities Power Engineering Conference (AUPEC 2004)*, Sept. 2004. [Online]. Available: http://www.itee.uq.edu.au/~aupec/aupec04/papers/PaperID56.pdf. [Accessed: May 26, 2007].

Ramos, R.R., Biel, D., Fossas, E., and Guinjoan, F. (2003). A fixed-frequency quasi-sliding control algorithm: application to power inverters design by means of FPGA implementation, *IEEE Transactions on Power Electronics*, 18(1), pp. 344-355.

Ricci, F., & Le-Huy, H. (2002). An FPGA-based rapid prototyping platform for variable-speed drives, *28th Annual Conference of the IEEE Industrial Electronics Society*, pp. 1156-1161.

Zhao, W., Kim, B.H., Larson, A.C., & Voyles, R.M. (2005). FPGA implementation of closed-loop control system for a small-scale robot," *12th International Conference on Advanced Robotics*, pp. 70-77.