Preprints of the 18th IFAC World Congress
Milano (Italy) August 28 - September 2, 2011

Methodologies for Implementing FPGA-Based Control Systems

Zaher M. Kassas *

* *Electrical & Computer Engineering Department, The University of Texas at Austin, USA, e-mail:* `zkassas@ieee.org`

Abstract: Recent years have witnessed an exponential growth in the complexity and performance of field-programmable gate arrays (FPGAs). This has lead to high adoption of FPGAs in embedded applications, such as consumer electronics, medical devices, and microelectromechanical system. Due to the FPGAs attractive features, choosing FPGAs as a control systems deployment platform is also spreading. This paper presents a number of development techniques for systematically implementing FPGA-based control systems. These techniques are applied towards developing a basic control systems building block, the transfer function. Additionally, a design cycle that is inspired by the "Design by Emulation" approach for digital control, is introduced for implementing general-purpose FPGA-based control systems. A case study is presented to demonstrate the application of the proposed methodologies towards an observer-based position motion control system.

Keywords: Digital implementation, FPGA, embedded systems, programmable controllers

1. INTRODUCTION

Field-programmable gate array (FPGA) adoption is growing rapidly in many industrial and embedded control applications, Monmasson and Cirstea [2007], Dase et al. [2006]. Since the FPGA approach to hardware design combines the best of many worlds, FPGAs are replacing many systems utilizing custom hardware, microprocessors, digital signal processors (DSPs), and application-specific integrated circuits (ASICs), Restle [2000]. In fact, recent benchmark reports suggest that some FPGAs have bypassed processors and DSPs with respect to cost and power consumption per signal processing unit, BDTI [2007].

FPGAs have the reliability and determinism of pure hardware solutions with loop rates in the order of tens to hundreds of MHz, while maintaining extremely low jitter. Moreover, FPGAs run in a completely parallel fashion; hence, designers can create any number of task-specific cores that all run like simultaneous parallel circuits inside one FPGA chip. Microprocessors and DSPs lack this functionality, since they process one instruction at a time using complex scheduling or a threaded approach to achieve pseudo-parallelism. Even with trends like multi-core, a processor-based solution will never get the parallelism offered by FPGAs. Furthermore, unlike ASICs, which suffer from fixed functionality, FPGAs are reprogrammable. Updating an FPGA with a tweaked or a completely new hardware personality is straightforwardly achievable.

Traditionally, FPGA designers would have been hardware designers with a significant amount of knowledge and experience in circuit design using hardware description languages (HDL) like VHDL or Verilog. These languages limited the mainstream adoption of FPGAs, as their concepts are unfamiliar to most software programmers. Recently, tools vendors started providing an increasing number of C-like FPGA programming languages (e.g. SystemC, Starbridge, and AccelDSP), which effectively accelerated the adoption of FPGAs. These pseudo-C languages provide a more familiar development flow that provides a significant level of abstraction away from the underlying hardware. Fig. 1 illustrates a typical hardware-based FPGA design flow and a software-based counterpart, Wain et al. [2006].

Additionally, other tools vendors started providing graphical, "system-level" design tools to empower control, simulation, and signal processing engineers to harness the full power of the FPGA technology, while providing a competitive performance and resource usage, as compared to traditional text-based HDL methods, Falcon and Trimborn [2006]. For example, Fig. 2 demonstrates a high-level graphical LabVIEW FPGA program for resonance mitigation in a torsional laser-etching machine. The FPGA program consists of a configurable low pass filter, a proportional-integral-derivative controller, and a notch filter. Note that programming this graphical, data flow-based code in VHDL would take thousands of lines of code.

A very useful feature of system-level design tools is their seamless integration with simulation environments that are indispensable in a control systems development process. Such integration allows the FPGA code to be simulated in a functionally-correct and a time-correct manner in a mixed continuous-time (CT) and discrete-time (DT) simulation environment with variable time-step solvers. These environments allow for "cycle-by-cycle" or "cycle-accurate" simulation, where the discrete execution rate of the FPGA block can be specified without skipping execution cycles, MacCleery and Kassas [2008].

[1] The author was with the Control Design & Simulation R&D, National Instruments, Austin, TX, USA.

Copyright by the International Federation of Automatic Control (IFAC)

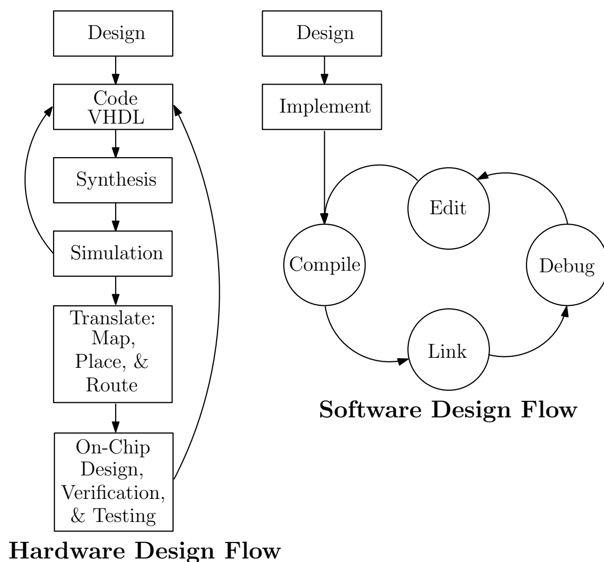

Fig. 1. Hardware vs. software FPGA design flows

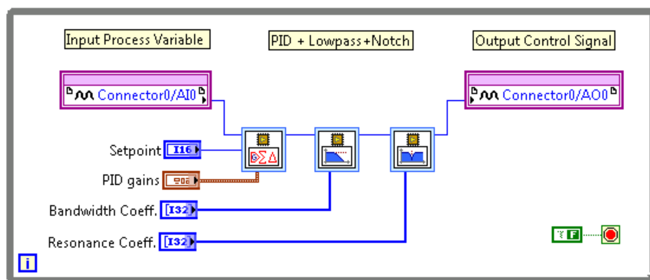

Fig. 2. High-level FPGA code for resonance mitigation

This paper presents a number of development techniques for implementing FPGA-based control systems. The presentation is kept intentionally simple, as the intended audience is control systems designers with a limited knowledge in FPGA programming. In this respect, the paper discusses how to balance speed and resource trade-offs into realizing a basic control systems building block, the single-input single-output (SISO) transfer function (TF). In addition, a design cycle that is inspired by the well-known "Design by Emulation" approach for digital control, is introduced for implementing general-purpose FPGA-based control systems. The discussed techniques are applied in a motion control case study, which requires high loop rates; hence, making FPGA targets an attractive implementation platform.

This paper is organized as follows. Section 2 discusses some crucial FPGA programming fundamentals. Section 3 illustrates realizing TFs on FPGAs. Section 4 presents a design cycle for FPGA-based control systems. Section 5 applies the discussed methodologies to a motion control case study. Concluding remarks are given in Section 6.

2. FPGA PROGRAMMING FUNDAMENTALS

2.1 FPGA Architecture

An FPGA is a programmable chip composed of three basic components, as illustrated in Fig. 3. The logic blocks are where the bits get processed. These logic blocks are connected together with programmable interconnects, which serve as a micro switch matrix to route signals from one logic block to the next. The interconnects can also route signals to the input/output (I/O) blocks, which are connected to the pins on the chip for two-way communication with surrounding circuitry. FPGAs can be viewed as a blank silicon canvas, which can be programmed to be any type of custom digital hardware.

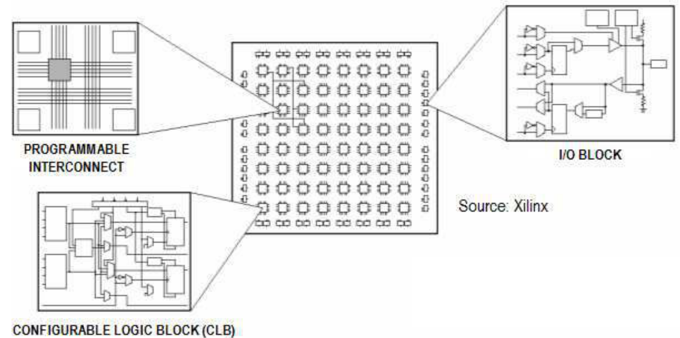

Fig. 3. Internal architecture of a typical FPGA chip

2.2 Floating-Point and Fixed-Point Considerations

Most FPGAs are limited to finite precision signal processing using fixed-point (FXP) arithmetic, because of the cost and complexity of floating-point (FP) hardware. Consequently, the FPGA designer must determine the word-length and radix setting of input, intermediate, and output signals in a design implementation to ensure the fidelity of the algorithm. A number of "automatic" conversion tools are becoming readily available to assist with such process, Cerna et al. [2008]. In this paper, the FXP notation `<e, wl, iwl>` will be adopted, where `e` specifies the binary encoding (signed or unsigned), `wl` specifies the word-length total number of bits, and `iwl` specifies the number of bits allocated for the integer part.

2.3 Speed and Resource Trade-Offs

Speed and resource trade-offs are a common attribute in FPGA-based implementations. Generally, FPGA designs aim at minimizing the resource consumption, by taking advantage of the high execution speeds of FPGA targets. FPGA resources include logic gates, look-up tables (LUTs), flip flops (FFs), multipliers, and block random access memories (BRAMs). To avoid resource consumptions, FPGA designers usually adopt the two common techniques described in the following subsections.

Use BRAMs to save arrays: To reduce resource consumption, saving fixed arrays onto the FPGA fabric must be avoided. For control systems designs, this means that "brute-force" linear algebra operations involving fully-sized vectors and matrices should be performed scarcely. For instance, consider the FPGA implementation of a full-state feedback control law $\mathbf{u} = -\mathbf{K}\mathbf{x}$, where $\mathbf{u}$ is the control signal, $\mathbf{x}$ is the system state, and $\mathbf{K}$ is the control gain matrix, which could have been obtained through a linear quadratic regulator (LQR) design. Assume that $\mathbf{x}$ and $\mathbf{u}$ are communicated to and from the FPGA code through I/O nodes, respectively. Good FPGA design practices advocate not implementing $\mathbf{K}$ as an input to the FPGA code, since upon code compilation, the matrix elements

will consume considerable resources on the FPGA fabric. Depending on the the matrix elements' FXP word-length setting, i.e. the number of bits allocated for the integer and fractional parts, medium to large-scale matrices could consume most or all of the FPGA slices. To illustrate this, just compiling a 10×10 matrix whose elements' are <s, 32, 16> onto a 1 million (1M) gate FPGA target *without* any additional code consumes 73% of the FPGA slices.

One approach to avoid consuming FPGA resources for matrices and vectors is to store the elements in BRAMs and access them whenever needed. In such approach, the following constraint must be respected. At a given cycle, one can either read from or write to a given BRAM memory address. Moreover, at a given cycle, one can read the contents of 2 different addresses of a particular BRAM or write into 1 particular address.

Interleave operations: A common approach to minimizing FPGA resource consumption is to pipeline or interleave certain operations so to re-use the same blocks recursively. Obviously, such approach will in turn increase the number of cycles necessary to produce a valid output.

For instance, consider the simple example of a matrix-vector multiplication operation, defined by $\mathbf{Ax}=\mathbf{b}$, where $\mathbf{A} \in \mathbb{R}^{n\times n}$ and $\mathbf{x} \in \mathbb{R}^n$. Assume that the matrix $\mathbf{A}$ is fixed, whereas the vector $\mathbf{x}$ is dynamic in the sense that it is being updated elsewhere in the FPGA code. Such operation can be implemented as follows:

Without pipelining: This is the brute force method. The resources utilization will be n^2 multipliers and $2n$ adders, and 1 cycle will be needed to produce a valid output.

With pipelining but without BRAM utilization: This method uses the same resources in a recursive fashion, but the matrix and vector elements are not saved in BRAMs. The resources utilization will be n multipliers and $n-1$ adders, and n cycles will be needed to provide a valid output.

With pipelining and with BRAM utilization: This method uses the same resources in a recursive fashion, and the matrix and vector elements are saved in BRAMs. The resources utilization will be 3 multipliers and 3 adders, and $\frac{n^2}{2}$ cycles will be needed to provide a valid output. Algorithm 1 illustrates this approach.

Algorithm 1 Computing $\mathbf{Ax} = \mathbf{b}$

Require: Matrix $\mathbf{A}$ already saved in BRAM1 as shown in Fig. 4.
Ensure: Vector $\mathbf{x}$ has been updated and is saved in BRAM2 as shown in Fig. 4.

sum $\leftarrow 0$
for $i = 1 : n$ **do**
  for $j = 1 : \frac{n}{2}$ **do**
    Read from BRAM1: $a_{i,2j-1}$ and $a_{i,2j}$
    Read from BRAM2: x_{2j-1} and x_{2j}
    sum $\leftarrow$ sum $+ (a_{i,2j-1} \times x_{2j-1}) + (a_{i,2j} \times x_{2j})$
  end for
  $b_i \leftarrow$ sum
  Write to BRAM3: b_i
  sum $\leftarrow 0$
end for

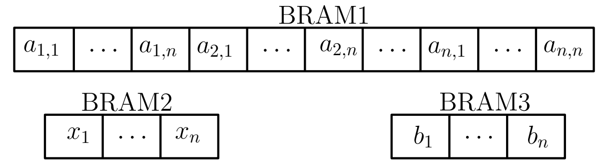

Fig. 4. Memory contents in BRAM1, BRAM2, & BRAM3

3. REALIZING DT TFS ON FPGAS

Consider the SISO monic proper DT TF

$$H(z) = \frac{Y(z)}{U(z)} = \frac{b_n z^n + b_{n-1}z^{n-1} + \cdots + b_1 z + b_0}{z^n + a_{n-1}z^{n-1} + \cdots + a_1 z + a_0},$$

where $U(z)$ and $Y(z)$ are the z-transforms of the system input and output, respectively. Such TF can be realized in state-space (SS) via the controllable canonical realization

$$\mathbf{x}(k+1) = \mathbf{A}\,\mathbf{x}(k) + \mathbf{b}\,u(k) \tag{1}$$

$$y(k) = \mathbf{c}\,\mathbf{x}(k) + d\,u(k), \tag{2}$$

where $\mathbf{x} = [\,x_0 \;\, x_1 \;\, \cdots \;\, x_{n-1}\,]^{\mathsf{T}}$ and

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ -a_0 & -a_1 & -a_2 & \cdots & -a_{n-1} \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$$

$$\mathbf{c} = [\,b_0 - b_n a_0 \;\; b_1 - b_n a_1 \;\; \cdots \;\; b_{n-1} - b_n a_{n-1}\,], \quad d = b_n.$$

By exploiting the sparsity of the $\mathbf{A}$ matrix and the super-diagonal state assignment, the linear algebra operations governing the SS system can be avoided and replaced by the algebraic operations, given by

$$x_i(k+1) = x_{i+1}(k), \quad i = 0, 1, \ldots, n-2$$

$$x_{n-1}(k+1) = -\left[\sum_{i=0}^{n-1} a_i x_i(k)\right] + u(k)$$

$$y(k) = \left[\sum_{i=0}^{n-1} (b_i - b_n a_i) x_i(k)\right] + b_n u(k).$$

This realization was programmed onto an FPGA using the LabVIEW FPGA Module, while employing the pipelining techniques discussed in Subsection 2.3. Fig. 5 illustrates the optimized FPGA code. Subsequently, the code was compiled for various TF orders. Fig. 6 presents the FPGA slices consumption of a 1M gate target as a function of the TF order. Note that while the curves are monotonically increasing, they are non-smooth due to the non-determinism in the routing algorithms employed by FPGA compilers.

4. DESIGN CYCLE FOR FPGA-BASED CONTROL SYSTEMS

The design cycle for FPGA-based control systems will be inspired by the "Design by Emulation" approach, in which a properly-designed CT controller is replaced by a DT-equivalent that emulates the CT design. Note that to adapt the proposed design cycle to the "Direct Digital Design" approach, the first phase of the design cycle is

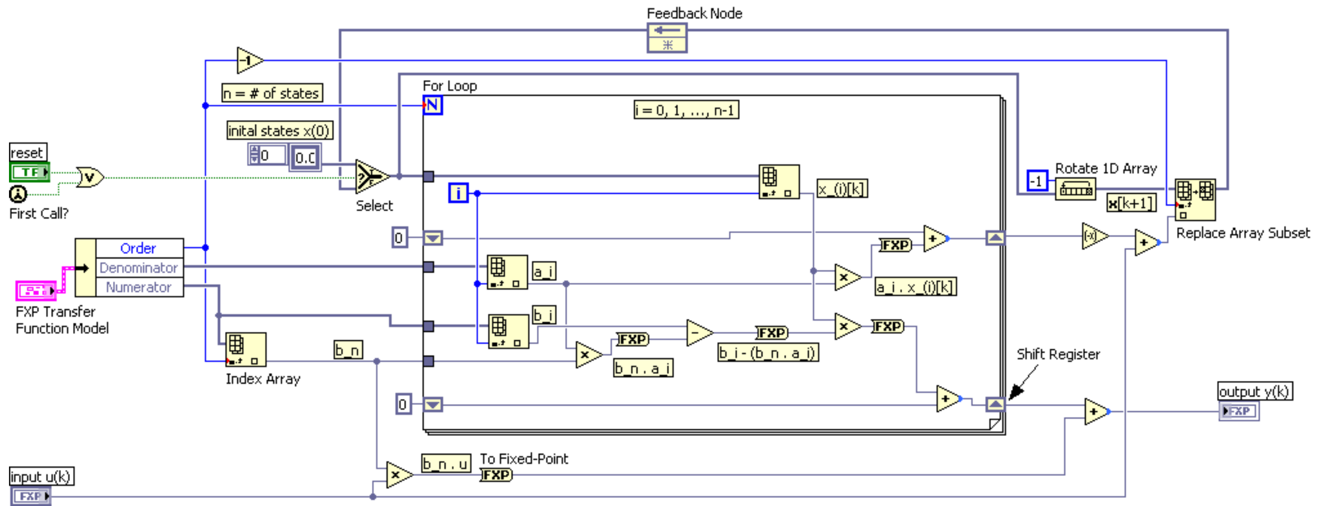

Fig. 5. Optimized FPGA code for realizing a monic proper DT TF

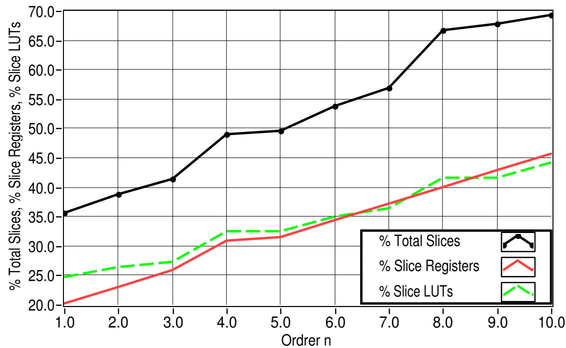

Fig. 6. 1M gate FPGA slice consumption vs. TF order

skipped. The proposed design cycle is depicted in Fig. 7 and consists of the following three phases:

CT controller design: The controller is designed to comply with a set of desired specifications, such as rise time, maximum peak overshoot, settling time, steady-state error, etc. Then, the designed controller is simulated in the closed-loop with the CT plant.

DT FP controller design: The properly-designed CT controller is discretized through a discretization method (e.g. Tustin, matched pole-zero, hold equivalents), based on the sampling period, T, at which the digital system will execute. Generally, T needs to be neither too large nor too small with respect to the system dynamics. Too large or too small of a T may cause system instabilities (numerical or otherwise) or failure to meet desired specifications. The following guidelines should be considered for choosing T:

- Choose a small enough T so to avoid aliasing effects. In particular, choose T in accordance with the Shannon-Nyquist sampling criterion according to $\frac{\pi}{T} > \omega_c$, where ω_c is the desired bandwidth of the closed-loop system. A general rule of thumb for selecting T is $5\omega_c \leq \frac{2\pi}{T} \leq 100\omega_c$.
- Choose a small enough T so to react fast enough to disturbances affecting the system.
- Choose a large enough T to avoid numerical issues. Numerical issues are actually "amplified" with the FXP representations, as will be discussed later.

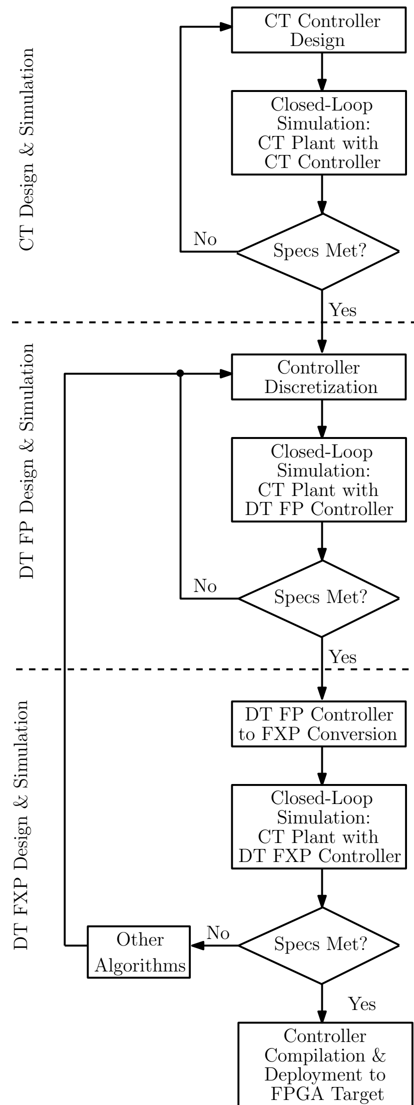

Fig. 7. FPGA-based control systems design cycle

Upon discretizing the controller, the closed-loop consisting of the CT plant and the DT controller is simulated. Such simulation will assess the stability of the closed-loop

system and determine whether the desired specifications are still met with the DT FP controller. If the closed-loop system is unstable or the desired specifications are not met, then the controller needs to be re-discretized by either changing the descritization method, and/or increasing/decreasing T. As $T \rightarrow 0$, it is known that discretization via the discrete delta operator, δ, offers a more meaningful connection to the underlying CT system and avoids some numerical issues encountered in discretization methods based on the shift operator model descriptions, Goodwin et al. [2010].

DT FXP controller design: The properly-discretized controller is converted to FXP. Note that the sampling period that was chosen in the previous phase will now be assumed to be the period at which the FPGA system will run at. Consequently, the resulting FXP controller is simulated in the closed-loop along with the CT plant. Such simulation will assess the stability of the closed-loop system and determine whether the desired specifications are still met with the DT FXP controller. If the closed-loop system is unstable, such instability might be arising either from too low of a T or word-length truncation effects due to the FXP representation. To illustrate this, recall that the relationship between a CT pole at s and its DT-equivalent is given by $z = e^{sT}$. Hence, $z \xrightarrow{T\rightarrow 0} 1$, no matter where the CT poles are. On one hand, choosing too small of a T, and given the truncation effects due to the FXP representation, may result in instability, as the DT system will possess multiple poles at $z = 1$. On the other hand, choosing too small of a T may result in a DT system that corresponds to a CT system different than the original one. To demonstrate this, consider a system with 2 CT poles located at $s_1 = -1$ and $s_2 = -5$. Assume the FXP representation to be `<s, 16, 8>`. The DT-equivalent poles with a sampling period of $T = 1$ms will result in DT FP poles located at $z_1 = 0.999$ and $z_2 = 0.995012$, respectively. The FXP representation of these DT poles is $z_1 = 1$ and $z_2 = 0.996094$, respectively. Converting the DT FP poles into the s-domain results in equivalent poles located at $s_1 = -1$ and $s_2 = -5$, respectively. However, converting the DT FXP poles to the s-domain results in poles located at $s_1 = 0$ and $s_2 = -3.91365$, respectively. Note that with such a FXP representation and a sampling period, the resulting DT FXP system corresponded to a different CT system than the one we originally started with. To circumvent this problem, either the FXP representations needs to be modified so to assign more bits to the fractional part or the sampling period needs to be increased. If neither of these solutions succeeds, then the TF needs to be implemented in an alternative fashion. Generally, the coefficients of higher-order terms in a TF are more sensitive to numerical errors. Consequently, a "quick" approach is to represent the TF through its partial fraction expansion and consequently implement the TF as a sum of the partial fractions. A more involved approach is to implement the high order TF as a combination of second-order sections (SOS) in series or parallel (e.g. direct forms I and II), Williamson [1992].

5. CASE STUDY

This case study illustrates the application of the proposed design cycle into designing an FPGA-based position controller for a particular DC motor of interest. The reference signal the motor should track is a 2 Hz square wave with an amplitude of ± 1. The controller $G_c(s)$ is a proportional-integral (PI) controller with gains $K_i = 30$ and $K_p = 1.5$. The amplifier $G_a(s)$ is a power converter, modeled as a two-pole low-pass filter (LPF) with a bandwidth of 50 Hz and a damping ratio ζ of 0.707. The plant $G_p(s)$ is modeled as an integrator with proper scaling. The sensor $G_s(s)$ is modeled as a single-pole LPF with a bandwidth of 20 Hz. The respective system TFs are given by

$$G_c(s) = \frac{1.5s + 45}{s},\ G_a(s) = \frac{98,696}{s^2 + 444.221s + 98,696}$$
$$G_p(s) = \frac{50}{s},\ G_s(s) = \frac{125}{s + 125}.$$

Ideally, the feedback control loop should use the "true" system output. However, such signal is usually inaccessible, and the available signal for feedback is the one measured by the sensor. Typical issues arising from sensors are phase lag, attenuation, and noise. To circumvent this problem, an observer will be used to produce feedback signals that are superior to the sensor output alone. Here, the approach described in Ellis [2002] will be adopted, where the true system output is compared with the estimated system output, which is governed by the estimated plant and sensor TFs, $\hat{G}_p(s)$ and $\hat{G}_s(s)$, respectively. The resulting error signal is fed into a compensator, $G_{co}(s)$, that will drive such error to zero. The observer-based control system block diagram is depicted in Fig. 8 and the observer block diagram is depicted in Fig. 9(a). The observer is a multiple-input single output (MISO) TF given by

$$\hat{Y}(s) = \left[\frac{G_p}{1 + G_p G_{co} G_s} \ \ \frac{G_p G_{co}}{1 + G_p G_{co} G_s} \right] \left[\begin{matrix} U(s) \\ Y(s) \end{matrix} \right]. \quad (3)$$

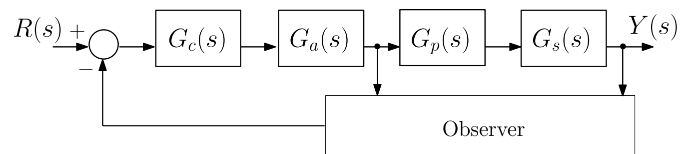

Fig. 8. Motion Control Block Diagram with Observer

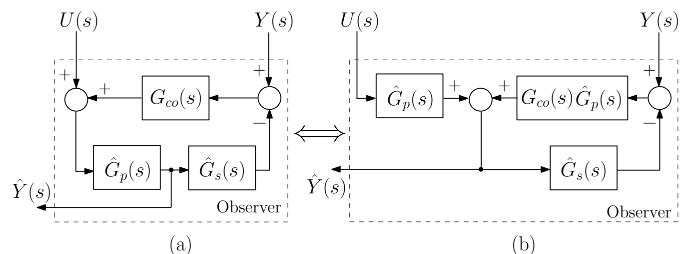

Fig. 9. Equivalent Observer Structures

The compensator $G_{co}(s)$ will be a proportional-integral-derivative (PID) controller with TF

$$G_{co}(s) = 60 \left(1 + \frac{10,000}{s} + 0.1s \right).$$

Since the TF $G_{co}(s)$ is not proper, simple block diagram manipulation of the observer block diagram in Fig. 9(a) yields the equivalent block diagram Fig. 9(b), which consists of proper SISO TFs.

The CT system in Fig. 8 was simulated using the LabVIEW Control Design and Simulation Module with and without an observer. The resulting responses are plotted in Fig. 10 as $y_1(t)$ and $y_2(t)$, respectively. The improvement in the response due to the inclusion of the observer in the control system is obvious. To implement the observer on the FPGA, the following steps were performed.

- The observer MISO TF (3) was discretized through zero-order hold. It was noted that a sampling period of at least $T_{\min} = 10\mu s$ was needed to obtain a response that emulates the one achievable by the CT observer. The response with the DT FP observer, discretized at $T_{\min}$, is plotted in Fig. 10 as $y_3(t)$.
- The MISO DT FP TF was converted to FXP with a representation of <s, 32, 12>. The I/O signals of the FXP TF block were set to <s, 32, 16>. The system response is plotted in Fig. 10 as $y_4(t)$.
- Instead of realizing the observer as a MISO TF, it was realized through the individual SISO TF blocks as depicted in Fig. 9(b). Each SISO TF was discretized at $T_{\min}$ and then converted into FXP with a representation of <s, 32, 12>. The corresponding system response is plotted in Fig. 10 as $y_5(t)$.

Note that while $y_5(t)$ emulates $y_3(t)$, the response $y_4(t)$ does not. This is due to FXP numerical errors. Essentially, at such small T, the coefficients of the DT FXP MISO TF are not well-captured by the chosen word-length and radix setting, due to truncation effects. Here, increasing T is not an option as this will cause the response of the system with the DT observer to fail to emulate its CT counterpart and can actually destabilize the closed-loop. This argues for decomposing higher-order TFs into lower-order TFs as discussed in Section 4. Fig. 10 plots the error between $y_3(t)$ and $y_5(t)$. Note that the error is of the order of 1×10^{-4}, which is in agreement with the FXP resolution guaranteed by the selected word-length and radix setting.

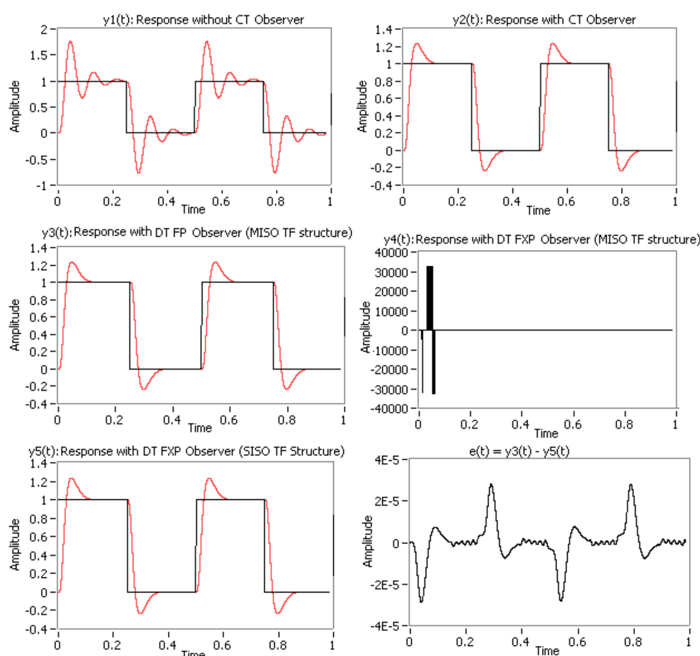

Fig. 10. System response with various observers

6. CONCLUSION AND FUTURE WORK

Traditionally, FPGA designs were limited to hardware designers. Control design and simulation tool chains are enabling control systems designers to deploy their designs to FPGAs by providing enough "system-level" abstraction. Nevertheless, control systems designers need to be familiar with a number of basic FPGA programming concepts. This paper introduced such concepts in a language comprehensible to control systems designers. It also proposed a design cycle for designing FPGA-based control systems. An example was presented to illustrate the application of the proposed methodologies towards an observer-based position motion control system, which requires high loop rates; hence, making FPGA targets an attractive implementation platform.

ACKNOWLEDGEMENTS

The authors would like to thank Javier Gutierrez, Ben Weidman, and Jim Lewis for their insightful feedback.

REFERENCES

BDTI. *Berkeley Design Technology Inc. Focus Report: FPGAs for DSP*. Berkeley, CA, 2nd edition, 2007.

M. Cerna, B. Marker, J. Nagle, and L. Wenzel. Fixed-point filter design and Riemannian geometry. In *42nd Asilomar Conference on Signals, Systems and Computers*, pages 566–569, 2008.

C. Dase, J.S. Falcon, and B. Maccleery. Motorcycle control prototyping using an FPGA-based embedded control system. *IEEE Control Systems Magazine*, 26:17–21, 2006.

G. Ellis. *Observers in Control Systems: A Practical Guide*. Academic Press, London, UK, 2002.

J.S. Falcon and M. Trimborn. Graphical programming for field programmable gate arrays: applications in control and mechatronics. In *Proceedings of the American Control Conference*, pages 1394–1400, Jun. 2006.

G.C. Goodwin, J.I. Yuz, Aguuero J.C., and M. Cea. Sampling and sampled-data models. pages 1–20, Jul. 2010.

B. MacCleery and Z.M. Kassas. New mechatronics development techniques for FPGA-based control of electromechanical systems. In *Proceedings of the 17th IFAC World Congress*, pages 4434–4439, Jul. 2008.

E. Monmasson and M.N. Cirstea. FPGA design methodology for industrial control systems - a review. *IEEE Transactions on Industrial Electronics*, 54(4):1824–1842, Aug. 2007.

R.C. Restle. Choosing between DSPs, FPGAs, μPs, and ASICs to implement digital signal processing. In *International Conference on Signal Processing Applications and Technology*, Oct. 2000.

R. Wain, I. Bush, M. Guest, M. Deegan, I. Kozin, and C. Kitchen. An overview of FPGAs and FPGA programming: Initial experiences at Daresbury. Technical report, Computational Science and Engineering Department, Nov. 2006.

D. Williamson. *Digital Control and Implementation: Finite Wordlength Considerations*. Prentice Hall, 1992.