FPGA IMPLEMENTATION OF CONTROL SYSTEMS

ПРИМЕНЕНИЕ FPGA В СИСТЕМАХ УПРАВЛЕНИЯ

АБДУЛМОХСИН Х., БАТЫРКАНОВ Ж.И.
izvestiya@ktu.aknet.kg

Abstract—Small robots can be beneficial to important applications such as civilian search and rescue and military surveillance, but their limited resources constrain their functionality and performance. To address this, a reconfigurable technique based on field-programmable gate arrays (FPGAs) may be applied, which has the potential for greater functionality and higher performance, but with smaller volume and lower power dissipation. This project investigates an FPGA-based PID motion control system for small, self-adaptive systems. For one channel control, parallel and serial architectures for the PID control algorithm are designed and implemented. Based on these one-channel designs, four architectures for multiple-channel control are proposed and two channel-level serial (CLS) architectures are designed and implemented. Functional correctness of all the designs was verified in motor control experiments, and area, speed, and power consumption were analyzed. The tradeoffs between the different designs are discussed in terms of area, power consumption, and execution time with respect to number of channels, sampling rate, and control clock frequency. The data gathered in this paper will be leveraged in future work to dynamically adapt the robot at run time.

I. INTRODUCTION

Because small robots have greater access to confined areas and are cheaper to deploy in large numbers, they are beneficial for many tasks such as urban search and rescue, military surveillance, and planetary exploration. But their small size constrains resources such as volume, payload capacity, and power. Consequently, computational capacity, mechanical abilities such as locomotion and manipulation, and sensor modalities are also constrained. For a small robot to be used in a variety of applications, versatile functionality is required. Not all functions are required for all applications and not all functions are required at all times. Thus, if limited resources can be reconfigured to tailor the robot to a specific application, it may need fewer resources for equal or more functionality. Research has been conducted in the area of robot reconfigurability in mechanism, in software, but to a much lesser extent in hardware, specifically using the relatively new technology of field-programmable gate arrays (FPGAs). In this study, we researched the design of a digital control system implemented in reconfigurable hardware. Specifically, we looked at closed-loop proportional-integral-derivative (PID) control for a robot with high degrees-of-freedom, which when implemented in software, requires a lot of CPU time and a real-time operating system. By moving control to hardware, the robot can dedicate the CPU to other tasks. However, we want both options available, as a software implementation is more appealing when the robot is not engaged in highly articulated movement, thus the need to implement in reconfigurable hardware.

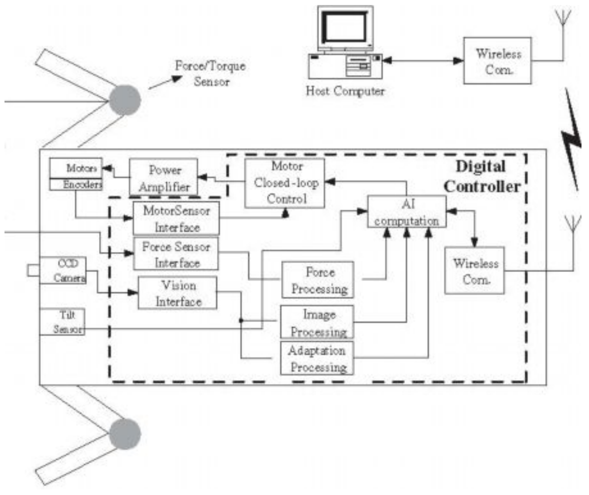


Fig. 1. CRAWLER functional architecture. The FPGA-based system will be an on-board, power efficient implementation of all functionality within the dashed line. Currently, it is implemented on 2 tethered PCs.

This research is a comparative study of FPGA-based PID control designs, relative to speed, area, and power consumption. It is important to understand the trade-offs of the various implementations to best utilize resources during reconfigurability. We expect that the study methodologies used in this project can be extended to study other functions.
Are implemented on two desktop computers tethered to the robot. The FPGA-based digital control system will be a power-efficient implementation of all components (within the dashed line of Figure 1) and embedded in the robot body.

II. RELATED WORK The benefit of software-based computation over hardware-based computation is the ability to reconfigure on-the-fly. This Run Time Reconfiguration (RTR) of computation is the basis for the flexibility and rapid growth of software-based solutions. But software requires a hardware target on which to run. FPGA chips can be reconfigured, too, but most only permit Compile Time Reconfiguration (CTR). However, a new breed of FPGA chips, such as XC6000 and Virtex-II Pro, allow Run Time Reconfiguration (RTR) of logic [2]. Furthermore, some of these new FPGA chips, such as the Virtex-II Pro, have embedded microprocessor units (MPU), making it possible to build power-efficient and highly reconfigurable system-onchip designs. These systems combine reconfigurable software, a power-efficient hardware target on which the software can run, and reconfigurable hardware all on a single chip. “HW/SW co-design” usually refers to methodologies that permit the hardware and software to be developed at the same time -splitting some functions to be implemented in hardware for additional speed, while others are implemented in software to free up logic resources. This is normally done offline. The combination of HW/SW co-design techniques with online RTR capability at both the hardware and software levels can optimally assign functions between the FPGA and software dynamically [3]. In order to achieve this level of RTR, the system specification must be partitioned into temporal exclusive segments, a process known as temporal partitioning. A challenge for RTR is to find an execution order of a set of sub-tasks that meet system design goals, a process known as context scheduling. Several approaches can be found in the literature describing these problems (e.g. [11]). All these approaches depend on performance and resource requirements of the requisite sub-tasks to make an optimal tradeoff [3]. This paper investigates the power, area, and speed characteristics of particular PID control implementations so that they may be used in future work on hardware/software run time reconfiguration of a SoC robotic controller. FPGA-based SoC designs have been widely applied in digital system applications and RTR research has been addressed by many researchers. Elbirt et al. explored FPGA implementation and performance evaluation for the AES algorithm [4]. Weiss et al. analyzed different RTR methods on the XC6000 architecture [2]. Shirazi described a framework and tools for RTR [5]. Noguera and Badia proposed a HW/SW co-design algorithm for dynamic reconfiguration [3]. FPGA power modeling and power-efficient design have also been studied by various researchers [6]–[10]. Closed-loop control algorithms, in particular, have been studied and implemented. Li et al. implemented a parallel PID algorithm with fuzzy gain conditioner on an FPGA and

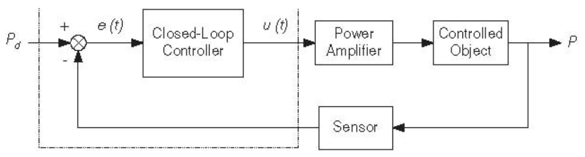


Fig. 2. Closed-loop control system. The output of the controller, u(t), is an attempt to reconcile the desired value Pd and the measured value P.

conducted a simulation-based study of it [12]. Chen et al. implemented a complete wheelchair controller on an FPGA with parallel PID design [13]. Samet et al. designed three PID architectures for FPGA implementation -parallel, serial and mixed [14]. Speed and area are the tradeoffs on the three designs. Simulation results show the correct functions and fast response time. All these prior works are for one channel control and the important issue of FPGAs, namely power consumption, for closed-loop control was not addressed.

In this work, different designs for the closed-loop PID control algorithm are implemented on an FPGA for both one and multiple channels. These designs are evaluated in terms of area, power, and speed. In this paper, first the PID algorithm is introduced, then, parallel and serial one-channel designs are described in detail in Section IV. Multiple-channel designs based on the one-channel designs are presented in Section V. Section VI introduces the test platform and methodologies, then presents the results. A discussion of those results follows in Section VII.

III. CLOSED-LOOP CONTROL SYSTEM

A closed-loop control system is shown in Figure 2, which is used to control a device such as a servo motor. P and Pd correspond to the controlled variable (e.g. rotational position) and its desired value, which is provided at a higher control level. The goal is to eliminate the error between P and Pd. The value of P is measured by the sensor, which is compared with Pd to generate the error e(t). The output to the controlled device, u(t), from the closed-loop controller is a function of e(t). Typically, this is a weak signal that requires amplification.

A. PID Control Algorithm

In this project, the PID algorithm is applied for closed-loop control. This is the most commonly used control law and has been demonstrated to be effective for DC servo motor control.

The PID controller is described in a differential equation as:

$$u(t)= K_p\, e(t)+ e(t)dt+ \frac{1\ de(t)}{T_d\,(1)}\ {}^{T}{}_{i\,0} \qquad dt$$

where Kp is the proportional gain, Ti is the integral time constant and Td is the derivative time constant. For a small sample interval T, this equation can be turned into a difference equation by discretization [16]. A difference equation can be implemented by a digital system, either in hardware or software. The derivative term is simply replaced by a first-order difference expression and the integral by a sum, thus the difference equation is given as:

$$u(n)= K_p \quad e(n)+ e(j)+\frac{TT_d}{T_i\,T}{}^{n}_{j=0}(e(n)- e(n-1))$$

(2) Equation (2)

can be rewritten as:

$$u(n)= K_p e(n)+K_i\, e(j)+K_d(e(n)- e(n-1))\ (3)\ {}^{n}_{j=0}$$

where Ki = KpT/Ti is the integral coefficient, and Kd = KpTd/T is the derivative coefficient. To compute the sum, all past errors, e(0)..e(n), have to be stored. This algorithm is called the “position algorithm” [16].

An alternative recursive algorithm is characterized by the calculation of the control output, u(n), based on u(n− 1) and the correction term Δu(n). To derive the recursive algorithm, first calculate u(n − 1) based on Eq. (3):

$$u(n-1) = K_p e(n-1) + K_i \,{}_{j=0}^{n-1} e(j) + K_d(e(n-1) - e(n-2)) \qquad (4)$$

Then calculate the correction term as:

$$\begin{aligned}\Delta u(n) &= u(n) - u(n-1)\\ &= K_0 e(n) + K_1 e(n-1) + K_2 e(n-2)\end{aligned} \qquad (5)$$

where

$$\begin{aligned}K_0 &= K_p + K_i + K_d\\ K_1 &= -K_p - 2K_d\\ K_2 &= K_d\end{aligned}$$

Equation (5) is called the “incremental algorithm”. The current control output is calculated as:

$$u(n) = u(n-1) + \Delta u(n) = u(n-1) + K_0 e(n) + K_1 e(n-1) + K_2 e(n-2) \qquad (6)$$

In the software implementation, the incremental algorithm (Eq. 6) can avoid accumulation of all past errors $e(n)$and can realize smooth switching from manual to automatic operation, compared with the position algorithm [16]. More advantages will be shown for the hardware implementation in Section IV.
In PID control, increasing the proportional gain K_p can increase system response speed, and it can decrease steady-state error but not eliminate it completely. Additionally, the performance of the closed-loop system becomes more oscillatory and takes longer to settle down after being disturbed as the gain is increased. To avoid these difficulties, integral control K_i and derivative control K_d can eliminate steady-state error and improve system stability ([17], respectively).

IV. ONE-CHANNEL DESIGNS

First, we constructed a one-channel design based on the PID control algorithm. The PID incremental algorithm (Eq. 6) is

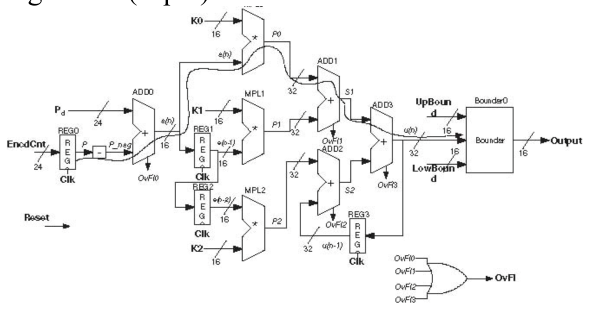


Fig. 3. Parallel design of incremental PID algorithm.

decomposed into its basic arithmetic operations:

$$\begin{aligned}e(n) &= P_d + (-P)\\ p0 &= K_0 * e(n)\\ p1 &= K_1 * e(n-1)\\ p2 &= K_2 * e(n-2) \quad (7)\\ s1 &= p0 + p1\\ s2 &= p2 + u(n-1)\\ u(n) &= s1 + s2\end{aligned}$$

For a parallel design, each basic operation has its own arithmetic unit – either an adder or multiplier. It is mainly combinational logic. For serial design, which is composed of sequential logic, all operations share only one adder and one multiplier.

A. Parallel Design

Figure 3 shows our parallel design of the PID incremental algorithm. The design requires 4 adders and 3 multipliers, corresponding to the basic operations shown in 7. All bold signals are I/O ports, while others are internal signals.
The clock signal clk is used to control sampling frequency. EncdCnt, the encoder counter value, represents the current position P. The negation of P, P neg, is generated by bit-wise complementing and adding 1. At the rising edge of control, signal $e(n)$of the last cycle is latched at register REG1, thus becomes $e(n-1)$ of this cycle. In the same manner, $e(n-2)$ and $u(n-1)$ are recorded at REG3 and REG4 by latching $e(n-1)$ and $u(n)$ respectively. The registers can be set to initial values of 0 by asserting the reset signal, Reset. As long as the desired position Pd is also initialized to 0 when the

system is reset, the control output is 0, which can keep the controlled device (i.e. the motor, in this system) static.
The computed control output u(n) may exceed the range that the controlled device can bear. Bounder, as shown in Fig. 3, is a value limitation logic that keeps the output in the user defined range of UpBound and LowBound.
Control can become unsteady and fail in the event of an overflow in any of the adders. OvFlx is asserted in the case of an overflow in adder x. All overflow signals are ORed together to generate the OvFl signal. When asserted, this signal can be used to shut down the controlled device.

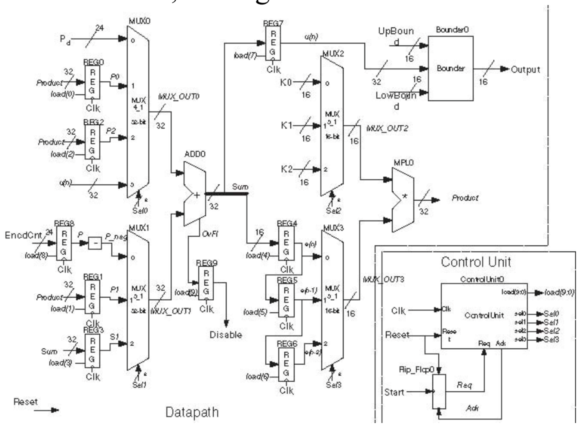


Fig. 4. Serial design of incremental PID algorithm.

The module is shown in Figure 3. The critical path of the incremental algorithm is also marked in Fig. 3. Its delay includes 1 delay from the 16-bit adder, 2 from the 32-bit adders and 1 from the 16-bit multiplier, expressed as

$$Dpos = Dadd16 + 2 * Dadd32 + Dmpl16 \quad (8)$$

B. Serial Design
To minimize area, in the serial design only one arithmetic operator is used for each kind of arithmetic operation. As shown in Figure 4, there is one adder and one multiplier. Other parts, including arithmetic operators, registers, multiplexers and other logic, are called the datapath. Registers are used to store intermediate results. At the rising edge of the clock, the input signal of the register can be latched only if the load input signal is asserted. In each clock cycle, the control unit, which is a finite state machine, sets selection signals of multiplexers according to the current state and input, effectively defining the input to each operator.
The adder is 32-bit, consequently, the multiplexers MUX0 and MUX1 that are used to select input for the adder are also 32-bit. Desired position, Pd, and negation of position feedback, P neg, are 24-bit thus need to be extended. The error value e(n) is computed by the adder but only the low 16 bits are latched to REG4, which will be input to the 16bit multiplier. The module shown in Figure 4 is implemented using VHDL.
V. MULTIPLE-CHANNEL DESIGNS In multiple-channel design, either a PID unit is dedicated to each channel, referred to as channel-level parallel (CLP) design, or one PID unit is shared by all channels, referred to as channel-level serial (CLS) design. The tradeoffs are in area, speed, and complexity. A parallel design occupies quite a large area as the number of channels increase, whereas a serial design requires a more complex control unit and obviously takes longer to compute all channels. Because CLP designs are so straight-forward, only CLS designs are presented in

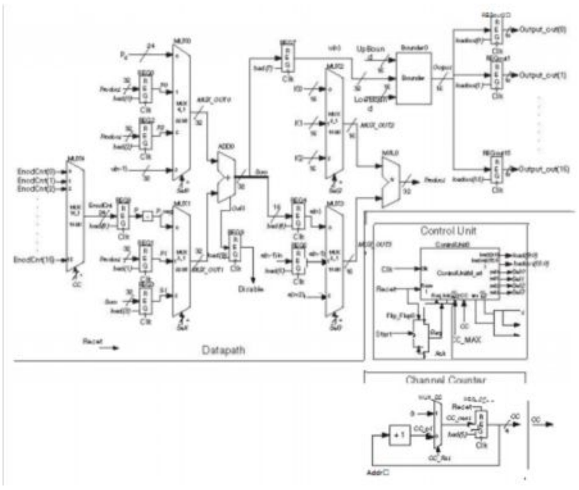


Fig. 5. Serial PID based multiple-channel design.

this section. The PID units of each design can be either serial (referred to as serial PID based design) or parallel (referred to as parallel PID based design).
In CLS design, context switching must occur prior to the computation of each channel output. For example, in switching from channel 0 to channel 1, the computed results u0(n)and e0(n) must be stored, and the parameters Pd, K0, K1, K2, UpBound, and LowBound for channel 1 and the previously calculated results u1(n − 1), e1(n − 1), and e1(n − 2) need to be loaded. Registers exist in slices in FPGAs, therefore it would consume a large number of resources to use registers as off-datapath storage. Fortunately, FPGAs also have block and distributed RAM, which we made use of for context switching storage.

A. Serial PID Based Multiple-Channel Design

The serial PID based CLS design is shown in Figure 5. It requires two cycles for context switching in addition to the four cycles required for a serial PID control unit. One read cycle is required before the start of the PID algorithm to load both the previous computation results and the channel-specific parameters from RAM. Also, a write-back cycle is required after completion of the PID algorithm to store those data.
There exist other distinctions from the one-channel design. Position feedback, EncdCnt, in the multiple-channel design is selected through the multiplexer MUX4 from the current channel in the read cycle, while the computation output needs to be latched at register REGoutN corresponding to the current channel N in the write-back cycle. The current channel is determined by the channel counter signal, CC, which can be set to 0 by an asynchronous reset signal Reset or by a synchronous reset signal CC Rst during operations. CC MAX is an input signal that determines the maximum channel number to control.

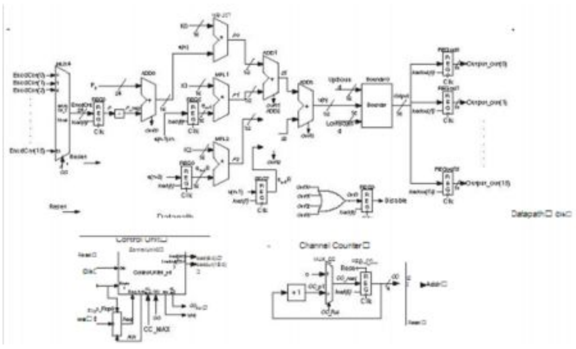


Fig. 6. Parallel PID based multiple-channel design (*PIDm par*).

B. Parallel PID based multiple-channel design

The parallel PID based CLS design is shown in Figure 6. In this design, the PID algorithm is executed in only one cycle, but context switching is still required. The read cycle for both parallel and serial based designs are the same. The writing of the results to RAM could be implemented either as a separate write-back cycle or as part of the PID computation cycle. A separate cycle increases the cycle count for each channel, thus increases the delay, which is significant because the critical path delay of the parallel design is quite long relative to the serial design. Therefore, the write-back of the results is included in the PID algorithm.

In the read cycle, the input signal $e(n-2)$ and $u(n-1)$ are latched at registers REG0 and REG7, instead of being connected to arithmetic operators directly, like in serial based design. This is because RAM write signals are asserted during the computation cycle. Thus the new value for $e(n-2)$ and $u(n-1)$, that is $e(n-1)$ and $u(n)$of this cycle, will be written into RAM.

VI. TEST METHODOLOGY AND RESULTS

A. Experiment Platform and Motor Control Interface Design

As seen in Figure 7, the required components of a complete system for motor control include a trajectory generator, a PID module, a PWM module, an amplifier and motor, a shaft encoder, and an encoder interface. The trajectory generator is implemented in software on the microprocessor of a Game Boy Advanced, the PWM module and encoder interface is implemented on the FPGA, and the amplifier, motor, and shaft encoder are external to the system. The PID module, which is the focus of this research, is implemented both in hardware on the FPGA, and for comparison, in software on the microprocessor.

All experiments were performed on the system configuration as shown in Figure 7, which includes an experimental Spartan II FPGA system with Xport 2.0 [18], [19]. This FPGA does not have an onboard CPU, thus we used a Game Boy Advanced System with an ARM7TDMI processor, which

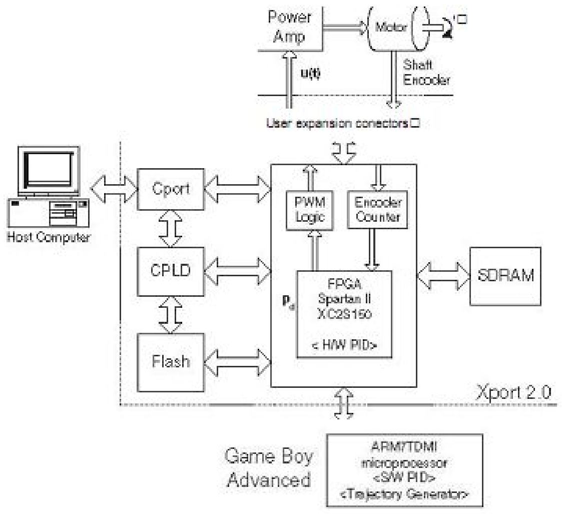


Fig. 7. FPGA experimental system.

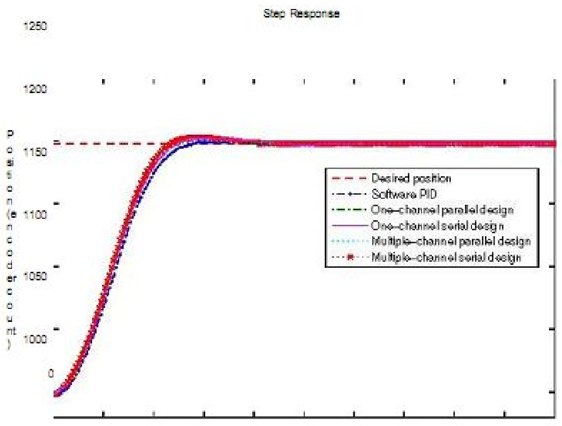


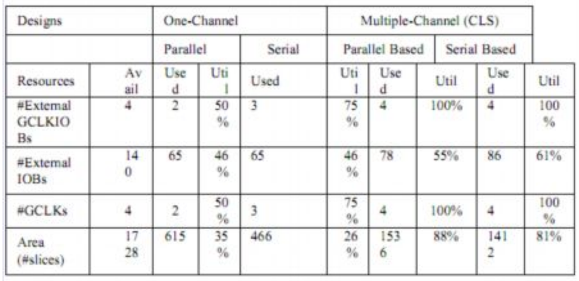
Fig. 8. Step response control experiment results.

is a 32-bit RISC CPU. External systems were connected to the FPGA through user expansion connectors. The FPGA configuration and CPU code was downloaded to the system from a host computer through the Cport. Ultimately, the controller will be implemented on an FPGA with an onboard CPU, such as the Xilinx Virtex-II Pro FPGA.

B. Function Test

A performance evaluation is meaningful only after the design is verified as functionally correct. Each PID hardware design, as described above, was implemented and used to perform step response control of a

motor. Additionally, a software implementation was developed and tested. In software, a set of preliminary PID parameters and control periods were determined by performing the Ziegler-Nichols [16] experimental method, and then the parameters were tuned to an ideal step response. The same parameters and sampling period were applied to the hardware PID implementations in the FPGA to perform the step response control tests. Experiment results of step response control for all designs are shown in Figure 8.
The parameter tuning experiment yielded the following results: proportional gain K_p = 463, integral coefficient K_i



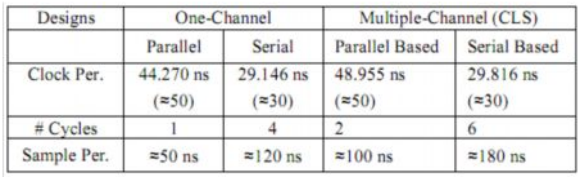
Designs		One-Channel				Multiple-Channel (CLS)			
		Parallel		Serial		Parallel Based		Serial Based	
Resources	Avail	Used	Util	Used	Util	Used	Util	Used	Util
#External GCLKIOBs	4	2	50 %	3	75 %	4	100%	4	100 %
#External IOBs	140	65	46 %	65	46 %	78	55%	86	61%
#GCLKs	4	2	50 %	3	75 %	4	100%	4	100 %
Area (#slices)	1728	615	35 %	466	26 %	1536	88%	1412	81%

= 0, derivative coefficient Kd = 4640, and the control period T = 0.833 ms, which were used to perform control testing. To test motor control for each design, the motor was set to an initial position of 1000, then a desired position command of 1200 was issued. From Figure 8, the horizontal dashed line is the desired position, while the other curves are the real responses sampled from the encoder counter. The results show that all the designs performed correctly and similarly. Response speed is fast, overshoot is small, and static accuracy is high. The average rise time is 30.32ms and the standard deviation of the rise time is 0.7451. The steady state error is 0.

C. Performance tests

Once the correctness of the designs was verified, performance was analyzed. All FPGA designs were implemented using the VHDL language in Xilinx ISE software. Xilinx provides a variety of performance analyses, including resource utilization, speed, and power consumption, based on simulations of the hardware design. Performance was based on these reports.
1) Resource Utilization : Resource utilization for each design is listed in Table I. The second column indicates the number of available resources, which includes the number of I/O blocks (IOBs), global clocks (GCLK), and Configurable Logic Block (CLB) slices. (CLB slice count, as opposed to logic gate count, is a more reliable area measurement [4].)
2) Speed : The Xilinx Timing Analyzer provided detailed and accurate timing information, including minimum clock period and delays along the data path. In each design, there were two timing concerns. The first was the control clock frequency, which controls the timing of the cycles of the PID algorithm. The control clock frequency depends on the delays experienced along the data paths of the PID. The second timing concern was the sampling rate, which corresponds to the rate that the PID algorithm generates torque commands. This frequency depends on the control clock frequency.
The one-channel parallel design is mainly a combinational logic requiring only a single cycle to complete. The sampling and control clock frequency are identical. The timing report showed that the longest delay was 44.270 ns, therefore we chose 50 ns as the minimum sampling cycle.
The longest delay in the one-channel serial design was 29.146 ns, therefore we chose 30 ns as the minimum clock

DEVICE RESOURCES UTILIZATION OF DESIGNS. CLOCKS AND EXECUTION TIMES OF DESIGNS.
TABLE I TABLE II



Designs	One-Channel		Multiple-Channel (CLS)	
	Parallel	Serial	Parallel Based	Serial Based
Clock Per.	44.270 ns (≈50)	29.146 ns (≈30)	48.955 ns (≈50)	29.816 ns (≈30)
# Cycles	1	4	2	6
Sample Per.	≈50 ns	≈120 ns	≈100 ns	≈180 ns

TABLE III POWER DISSIPATION OF ONE-CHANNEL PARALLEL DESIGN.

Sampling		Simulation	Power dissipation (mW)	
Freq. (Mhz)	Period (ns)	Time (µs)	Stable state	Dynamic state
0.0012	833000	8330	11.39793	11.6651
0.012	83300	833	11.81836	13.14453
0.12	8330	83.3	16.02	17.97
0.25	4000	40	21.08	25.13
0.5	2000	20	30.81	38.9
1.25	800	8	59.99	80.23
2.5	400	4	108.63	149.21
6.25	160	1.6	254.56	355.77
8.3325	120	1.2	335.63	470.58

cycle. Each PID computation requires four cycles, so the minimum sampling period for the one-channel serial design was 120 ns.

Similar analyses were done for CLS parallel PID and serial PID based multiple-channel designs. We chose 50 ns for the minimum clock period for the CLS parallel design. Each PID computation requires two cycles, so the minimum sample period for each channel is 100 ns. We chose 30 ns for the minimum clock cycle of the CLS serial design. Each PID computation requires six cycles, so the minimum sample period for each channel is 180 ns. All clock and sample periods are listed in Table II.

3) Power Dissipation : Power consumption is dependent upon the sample and control clock frequency. Thus, to compare power performance, we both generated motor commands and ran the PID module at various frequencies. For any single comparison, such as one-channel serial versus one-channel parallel, frequencies were the same for both. This required significant delays in the parallel-based implementations, but it presented a more valid comparison.

The test data obtained in the step response experiments were used as input to the hardware simulation of each PID design. For a static state analysis, the desired position was identical to the current position. For dynamic state analysis, the initial position was set to 1000 and the desired to 1200. Once this desired position was reached, it increased by 1 every 278 µs. This dynamic process was based on the sampled data in the function tests. Simulation resolution is 100 ps.

Power test results with different sampling frequencies for one-channel parallel design and serial design are shown in Table III and IV, respectively.

The simulation time for getting a power test value is extremely long, so multiple-channel designs are only tested for sampling frequency 0.12MHz. Power was tested for different numbers of channels with the control clock frequency at

TABLE IV POWER DISSIPATION OF ONE-CHANNEL SERIAL DESIGN.

Sampling Freq. (MHz)	Control clock Freq. (MHz)	Simulation Time(µs)	Power (mW) (Dynamic state)
0.0012	0.0048	8330	11.40
0.012	0.048	833	11.69
0.12	0.48	83.3	13.83
0.25	1	40	16.43
0.5	2	20	21.51
1.25	5	8	36.74
2.5	10	4	62.16
6.25	25	1.6	123.64
8.3325	33.33	1.2	158.15

TABLE V POWER DISSIPATION OF MULTIPLE-CHANNEL PARALLEL BASED DESIGN.

Sampling Freq. (MHz)	Control clock Freq. (MHz)	Simulation Time (µs)	Number of channels	Power (mW) (Dynamic state)
	10	83.3	9	459.34
			1	147.33
			4	276.77
0.12	8.33	83.3	8	415.98
			12	564.48
			16	704.81
	5	83.3	9	454.66
	4.17	83.3	9	453.88

8.33MHz in parallel based design and at 25MHz in serial based design, resulting in the same execution time for serial and parallel based designs. For other control clock frequencies, power was only tested for 9 channels. Power test results for CLS multiple-channel parallel PID based design are shown in Table V.

Power test results for serial PID based design are shown in Table VI.

VII. DISCUSSION

A. Area

In one-channel and multi-channel serial designs, all arithmetic operations share one multiplier and one adder, while in parallel designs there are 3 multipliers and 4 adders. Because of this, serial designs have an obvious space advantage. However, some of this space savings is used up with additional control logic. As shown in Table I, the one-channel serial design is only 24.2% smaller than the parallel design, and a mere 8% smaller with multiple-channel serial based design.

TABLE VI POWER DISSIPATION OF MULTIPLE-CHANNEL SERIAL BASED DESIGN.

Sampling Freq. (MHz)	Control clock Freq. (MHz)	Simulation Time (μs)	Number of channels	Power (mW) (Dynamic state)
	33.33	83.3	9	231.74
			1	168.03
			4	190.40
0.12	25	83.3	8	214.78
			12	239.87
			16	267.79
	12.5	83.3	9	206.87

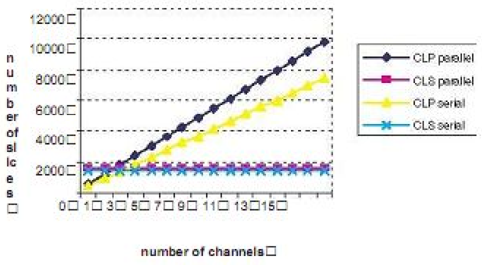


Fig. 9. Area comparison of multiple-channel designs.

Although channel-level serial designs also have only one PID unit, they need more support resources, such as registers, decoders, RAM, and control logic. So multiple-channel designs, even channel-level serial, require much more area than corresponding one-channel designs, as shown in Table I. The multiple-channel parallel based design requires 2.5 times the area of the one-channel parallel design. The multiple-channel serial based design requires 3.0 times the area of the one-channel serial design.

Figure 9 compares area requirements for CLS and CLP designs. CLS area requirements remain constant while CLP requirements grow with the number of channels. For a large number of channels, CLS has a significant advantage, however, for a small number of channels, CLP is advantageous.

B. Speed

Area and speed are conversely related. The advantages in area requirements shown for serial design are countered by their disadvantage in speed. While the datapath in the serial design is shorter, thus the delay is shorter, more clock cycles are required. As expected, execution times for serial design are longer, as shown in Table II.

CLS multiple-channel designs require more cycles for context switching than one-channel designs, so execution times are longer for each channel. The execution time of the CLS design for N channels is N times the execution time for one channel, while execution time of CLP design is always equivalent to that of the one-channel design.

C. Power Analysis

Table III shows that the dynamic process consumes more power than the stable state in the one-channel parallel design. It also shows that power dissipation increases as sampling frequency is increased, as expected. Likewise, this trend appears in the serial design data in Table IV. Surprisingly, there is minimal difference between the two at reasonable sampling frequencies as shown in Figure 10. (An early hypothesis was that the parallel design would be more power efficient due to slower operation.)

In multiple-channel designs, for the same sampling frequency and control clock frequency, power dissipation increases as the number of channels increases. This feature is shown in Figure 11, where the sampling frequency is 0.12MHz. Power is approximately linear with the number of channels. As mentioned above, it was expected that for the same sampling frequency and execution time, the multiple-channel parallel based design would consume less power, because the clock frequency of the parallel based design is lower. For one channel, the parallel based design does consume less power, however, for a large number of channels, the parallel-based design consumes *more* power than the serial-based (an example of this is depicted in Figure 11).

Figure 11 also shows that for the same sampling frequency, the channel-level parallel design with a serial PID unit consumes the least power, but the area requirements of the CLP design (Figure 9) rapidly exceed the capacity of the FPGA as the number of channels increases.

VIII. CONCLUSION

In this project, preliminary work was conducted to explore control system design for a resource-constrained robot based on an FPGA technique. Parallel and serial architectures of the PID control algorithm were designed and implemented for one-channel closed-loop control. Two architectures of channel-level serial (CLS) multiple-channel PID were designed based on parallel and serial one-channel designs respectively. Step response control experiments verified functional correctness of these four designs.

Performance tests show that for a small number of channels, CLP serial based design has the smallest area and consumes the least power. For more channels (more than three for Spartan II FPGA), the CLS serial based design has the smallest area requirements and the CLP serial design still consumes the least power, but the area requirements of the CLP serial based design increases very quickly. Thus, for a large number

of channels, CLS serial based design is suggested. CLS serial based design has the longest execution times, but the execution time is not a major concern for these designs as long as the number of channels is not so large that the total execution time exceeds the sampling period.

In addition, the results show that the dynamic process consumes more power than the stable state, that higher sampling frequency and control clock frequency consumes more power than lower frequency, and that power dissipation increases as the number of channels increases.

REFERENCES

[1] R.M. Voyles and A.C. Larson, ”TerminatorBot: A Novel Robot with Dual-Use Mechanism for Locomotion and Manipulation,” in *IEEE/ASME Transactions on Mechatronics*, v. 10, n. 1, 2005, pp. 17-25.

[2] K. Weiss, R. Kistner, A. Kunzmann and W. Rosenstiel, “Analysis of the XC6000 Architecture for Embedded System Design”, in *Proc. of IEEE Symp. on FPGAs for Custom Computing Machines*, Apr, 1998, pp. 245-252

[3] J. Noguera and R.M. Badia, “HW/SW Codesign Techniques for Dynamically Reconfigurable Architectures”, in *IEEE Transactions on Very Large Scale Integration Systems*, Vol. 10, No.4, August 2002, pp. 399-415.

[4] A.J. Elbirt, W. Yip, B. Chetwynd, and C. Paar, “An FPGA-Based Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists”, in *IEEE Transactions on Very Large Scale Integration Systems*, Vol. 9, No.4, August 2001, pp. 545-557.

[5] N. Shirazi, W. Luk and P.Y.K. Cheung, “Framework and tools for run-time reconfigurable designs”, in *IEE Proceedings of Computers and Digital Techniques*, Vol. 147, No. 3, May 2000, pp. 147-151.

[6] S. Choi, R. Scrofano, V.K. Prasanna, and J-W. Jang, “Energy-Efficient Signal Processing Using FPGAs”, in *Proceedings of ACM/SIGDA 11th ACM International Symposium on FPGA*, Monterey CA, Feb. 23-25, 2003, pp. 225-233.

[7] L. Shang and N.K. Jha, “High-level Power Modeling of CPLDs and FPGAs”, in *Proceedings of 2001 IEEE International Conference on Computer Design*, Sep. 23-26, 2001, pp.46-51

[8] S. Gupta and F.N. Najm, “Power Modeling for High-Level Power Estimation”, in *IEEE Transactions on Very Large Scale Integration Systems*, Vol. 8, No.1, Feb. 2000, pp. 18-29.

[9] L. Shang and N.K. Jha, “Hardware-Software Co-Synthesis of Low Power Real-Time Distributed Embedded Systems with Dynamically Reconfigurable FPGAs”, in *Proceedings of the 15th IEEE International Conference on VLSI Design (VLSID’02)*, Jan. 2002, pp. 345-352.

[10] A. Garcia, W. Burleson, and J.L. Danger, “Low Power Digital Design in FPGAs: A Study of Pipeline Architectures implemented in a FPGA using a Low Supply Voltage to Reduce Power Consumption”, in *Proceedings of 2000 IEEE International Symposium on Circuits and Systems*, Geneva, vol.5, May 28-31, 2000, pp. 561-564.

[11] R. Maestre, F.J. Kurdahi, M. Fernandez, R. Hermida, “A Framework for Scheduling and Context Allocation in Reconfigurable Computing”, *Proc. of the International Symposium on System Synthesis*, 1999, pp. 134-140.

[12] J. Li and B.-S. Hu, “The Architecture of Fuzzy PID Gain Conditioner and Its FPGA Prototype Implementation”, in *Proceedings of 2nd IEEE International Conference on ASIC*, Oct. 21-24, 1996, pp. 61-65