Case study of PID control in an FPGA

Paul Schad and David Carney, Plexus - January 17, 2011

This paper provides an account of how to design and verify an FPGA PID controller. The presented design uses PID to control a constant power pulse. The FPGA measures voltage and current and controls those measurements to a power setpoint.

This article is from a class at DesignCon 2011. Click here for more information about the conference.

Its output is a DAC value that controls an electronic load connected to the source. A previous generation product used a single board computer for PID control, but the loop sample rate was not fast enough for the nextgen product.

A DSP was considered, but there was already an FPGA present in the previous generation so we investigated how to take advantage of it to solve this design problem. Most FPGAs have embedded multipliers to allow easy implementation of DSP operations, and they are potentially faster than a DSP because they can parallelize tasks including the calculations, external communications, and multiple control loops if applicable. We will also provide detail on the design problems inherent in doing PID control in an FPGA.

We will also cover how we selected a PID equation, the architecture of the FPGA, and the system level timing analysis. Addressed are some of the implementation details including how to implement the multipliers, considerations with fixed-point math, resource usage, and tuning methods. Finally, the paper will cover how the design was verified with a test bench.

How to select a PID equation
The traditional analog PID equation is given as follows [1].

$$u(t) = K\left(e(t) + \frac{1}{T_I}\int_0^t e(t)dt + T_D\frac{d}{dt}e(t)\right)$$

The adjustable PID parameters are K, TI and TD, while u(t) is the control output, and e(t) is the error signal (setpoint response level â€“ measured response). The pulse transfer function of the digital PID controller is given as follows [1].

$$U(z) = K_P E(z) + \frac{K_I}{1-z^{-1}}E(z) + K_D(1-z^{-1})E(z)$$

For the purposes of digital implementation, it is convenient to express this equation in incremental discrete sampled form using backward differences as follows [2].

$$u(k) = u(k-1) + K_P\left(e(k)-e(k-1)\right) + K_I T_s e(k) + \frac{K_D}{T_s}\left(e(k)-2e(k-1)+e(k-2)\right)$$

The terms of this equation are defined as follows:

k = sample number

Ts = sample period

e(k) = error term = SP(k) â€“ y(k)

SP(k) = setpoint value to control measured input to

y(k) = input value being controlled

u(k) = controller output value

KP = the gain of the proportional control

KI = the gain of the integral control

KD = the gain of the derivative control

This form of digital PID control is often referred to as Type A [3]. Two additional forms of the equation have been used by engineers to improve the response in real world systems. Type B is created by removing the setpoint from the calculation of the KD term and is shown as follows [3].

$$u(k) = u(k-1) + K_p\left(e(k) - e(k-1)\right) + K_i T_s e(k) + \frac{K_D}{T_s}\left(-y(k) + 2y(k-1) - y(k-2)\right)$$

Type C is created by further removing the setpoint from the calculation of the Kp term and is shown as follows [3].

$$u(k) = u(k-1) + K_p\left(-y(k) + y(k-1)\right) + K_i T_s e(k) + \frac{K_D}{T_s}\left(-y(k) + 2y(k-1) - y(k-2)\right)$$

These so called Type B and Type C equations are a special case of setpoint weighting in which the setpoint weighting factors for both proportional and derivative control are set to 0. Eliminating the setpoint from the proportional control gives a smoother response with less overshoot when large changes in the setpoint occur.

Eliminating the setpoint from the derivative control limits the transients that occur in the output response due to large changes in the setpoint. Reducing the derivative response also improves the control loop behavior in the presence of high frequency noise disturbances [2].

In general, it is easier to achieve a stable desired response with the setpoint value removed from the proportional and derivative terms of the PID controller. The Type C equation was chosen for this application for these reasons.

System-level simulation

Matlab Simulink was used to simulate the behavior of the entire control loop system with a Type C digital controller equation. Two aspects of the FPGA architecture were studied in this simulation. The first was the Type C equation, and the second was the effect of a 5.877 Âµs delay between taking a current and voltage sample and updating the DAC output (see System Timing Analysis section for details on this delay). Below in **Figure 1 below** is the top-level diagram of the Simulink simulation.

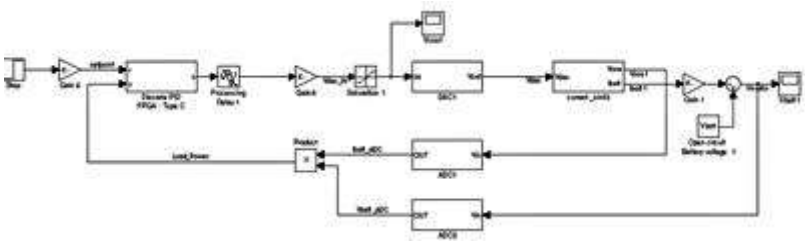


***Figure 1: Simulink Model** (To view larger image click here)*

This simulation takes into account an inner and an outer control loop. The inner loop consists of the analog hardware (current sensor, electronic load MOSFET, error amplifier, and compensator), and is everything to the right of the DAC and ADCs in the figure. Everything to the left is part of the outer control loop implemented in the FPGA.

In addition to this simulation, the inner loop was tested using an arbitrary waveform generator in the lab, meant to simulate the changing output of the DAC, and monitored with an oscilloscope on the load. The results from the testing and simulation showed that the Type C equation was acceptable so we did not implement and test any of the other equations.

Page 2

FPGA architecture

The block diagram below illustrates the FPGA used for the PID control implementation. The functionality of this FPGA, in addition to the PID Controller, includes the following:

- Calibration calculations
- Moving average filter
- Waveform setpoint tables
- Waveform control
- PID state machine

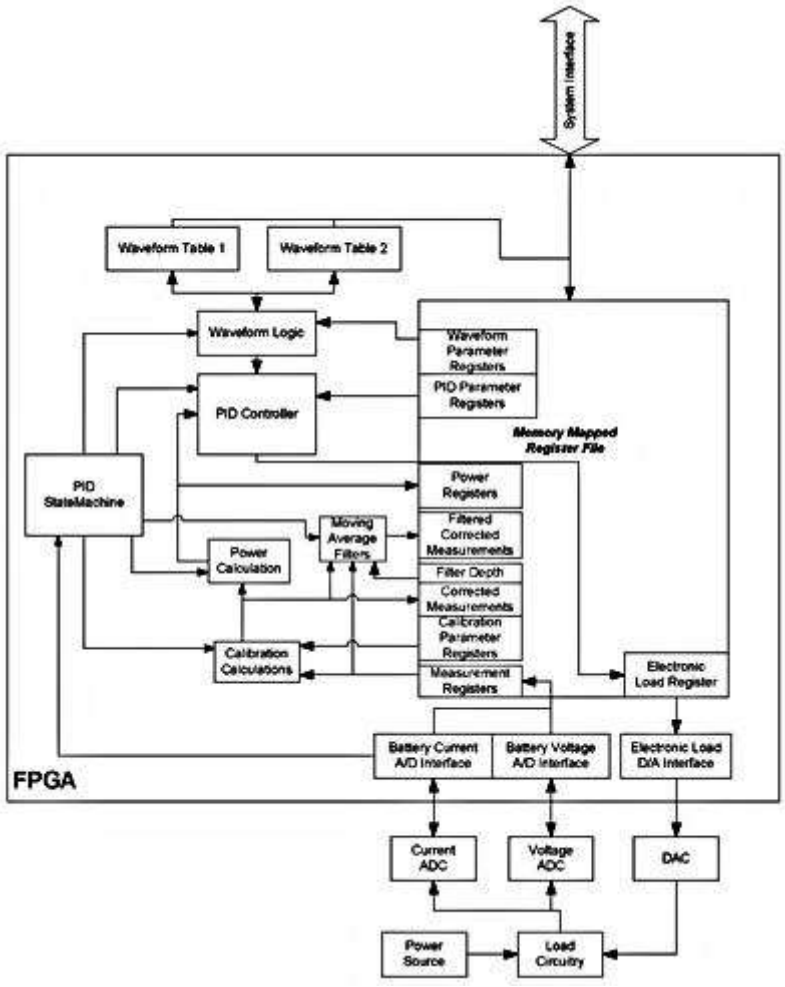


***Figure 2: FPGA PID Controller Block Diagram**(To view larger image click here.)*

The ADCs are 16-bit, 250 kSPS. The DAC is 14-bit, 250 kSPS.

Calibration calculations

One of the requirements of this product was high accuracy voltage and current measurements (± 5 mV and ± 5 mA). To achieve this accuracy, it was necessary to calibrate the electronic load circuitry, which had small non-linearities and an inherent offset. So the first calculation done on the data, after it was read from the ADCs, was to apply calibration parameters found previously using independent, external measurement equipment.

These calibration parameters are loaded into FPGA registers at power-up.

The calibration parameters were an offset and a gain. The voltage and current offset was simply the ADC value at 0 V or 0 A. The voltage and current gain represent the linearity of the load circuitry. These parameters are expressed by the following equations:

$$Gain = \frac{ADC_{ideal1} - ADC_{ideal0}(ideal\_counts)}{ADC_{meas1} - ADC_{meas0}(meas\_counts)} \times 2^{15}$$

$$Offset = ADC_{meas0}$$

The electronic load hardware is relatively linear so the gain parameter is going to be close to unity. It was assumed, based on circuit analysis and results from the previous generation product, that the gain would never be larger than two. To add the most amount of precision possible to the gain using that assumption, the software loads a gain value in [1.15] fixed-point format.

After the gain is applied to the measurement, the result is rounded to the nearest whole ADC value, eliminating the decimal digits just introduced. From this point, all of the calculations in the loop are done in whole numbers. The remaining math in the control loop is done with whole numbers.

ADC0 is the ADC value for a 0 V or 0 A input. ADC1 is the ADC value for a given voltage or current input. The ideal values are the values that the ADC would output if it were perfectly linear. The ideal ADC value for a given input can be calculated from the following equations,

$$Ideal\_ADC = (2^{16} - 1)\frac{input(V)}{full\_scale\_input(V)}$$

All of the above equations are used by the software to calculate the calibration parameters. The following equation is used by the FPGA to find the calibrated value, and it is illustrated in **Figure 3 below.**

Calibrated _ Value =(Raw _ Value - Offset) ×Gain

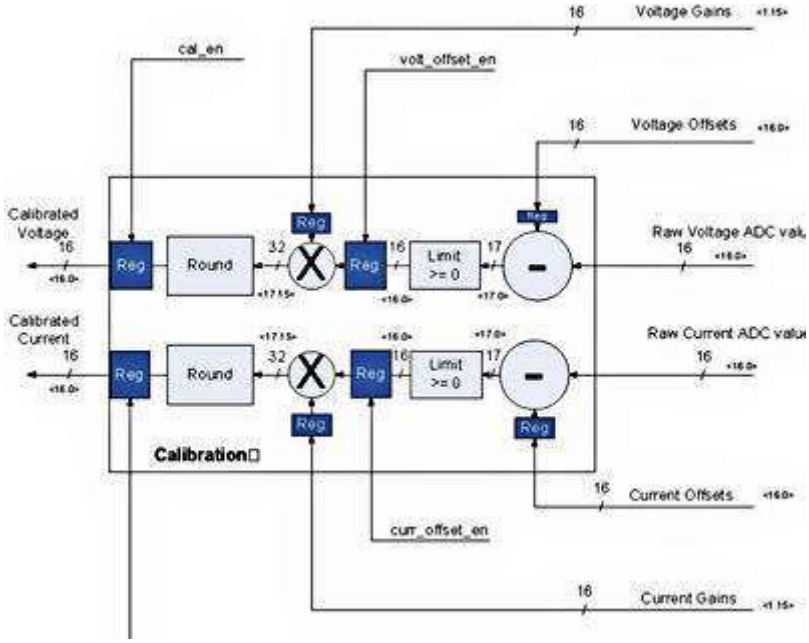


Figure 3: Calibration Block Diagram (*To view larger image* ***click here.)***

To further improve the accuracy of the measurements, different calibration parameters are selected depending on the value of the voltage and the current. For voltage calibration, there are 8 sets of calibration constants, each applying to a range of raw voltage readings as shown in **Table 1 below:**

Calibration Range	Low Raw ADC Count Range	High Raw ADC Count Range
1	0x0000	0x1FFF
2	0x2000	0x3FFF
3	0x4000	0x5FFF
4	0x6000	0x7FFF
5	0x8000	0x9FFF
6	0xA000	0xBFFF
7	0xC000	0xDFFF
8	0xE000	0xFFFF

Table 1: Calibration Parameter Ranges

For current calibration, there are 16 sets of calibration constants, with two groups of 8. Each of the two groups applies to the range of raw current readings similar to the voltage calibration

ranges. One group is used when the voltage is in the range of 0x0000 to 0x7FFF, and the other is used when the voltage is in the range of 0x8000 to 0xFFFF.

Moving average filter

We employed a moving average filter (**Figure 4, below**) in the FPGA in order to help the FPGA report highly accurate voltages and currents to the software. This filter is a 128 sample deep FIFO block that averages all of the samples. The filter operates on both voltage and current simultaneously, and it will operate on either calibrated or raw ADC values (but not at the same time). The filter is not used in the PID control loop. It is only used for reporting the values to the software for display to the user and computing energy.

Software can control the moving average filter through the register file interface. The features of the filter include the depth of the filter (max of 128 samples) and clearing the filter contents. A Filter Full bit is reported to the software to indicate that the output of the filter is valid. Software can also control whether the filter uses calibrated or uncalibrated data as its input. Using uncalibrated data is necessary during a board calibration.

When reading the average voltage, the average current at that instant is stored in a separate register. Software can then read this stored value to get a snapshot of what both the current and voltage were at the time of reading the voltage. The most recent current measurement is also made available through a separate register.

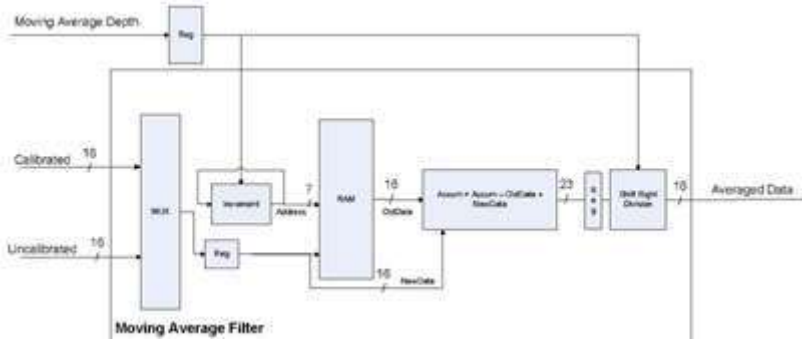


Figure 4: Moving Average Filter Block Diagram and State Machine Diagram *(To view larger image, click here.)*

Waveform tables

Waveform tables (**Figure 5, below**) are used to ramp the power setpoint over a period of time, creating a pulse edge, instead of instantaneously switching it on or off. There are two tables: one for the rising edge and one for the falling edge. The waveform table data scales the actual setpoint from 0 to 100 % (0x0000 to 0xFFFF). How this data is used is shown in the Waveform Control section below. The last data point in the waveform table is the steady state scale factor of the setpoint until the software triggers another pulse edge.

The benefit of the using the waveform tables is that the resultant waveform edge was much smoother when using the waveform table than without. The drawback is that the rise and fall times are naturally increased because the error on the early samples is much smaller than with a step input.

These tables proved to be useful once we got to the PID tuning stage of the project. Without the waveform table, non-idealities were present in the edges.

Having smooth, predictable edges was important in this system.

The content of the tables are written by software external to the FPGA design, through a system communication bus, at power-up. The software then only has to issue a single command to either call the rising edge or falling edge table.

The tables are just two blocks of FPGA internal RAM that are accessible by software. Each block of RAM has 16-bit wide data, 256 words deep. The PID loop frequency is 250 kHz, for a total edge length of approximately 1 ms.

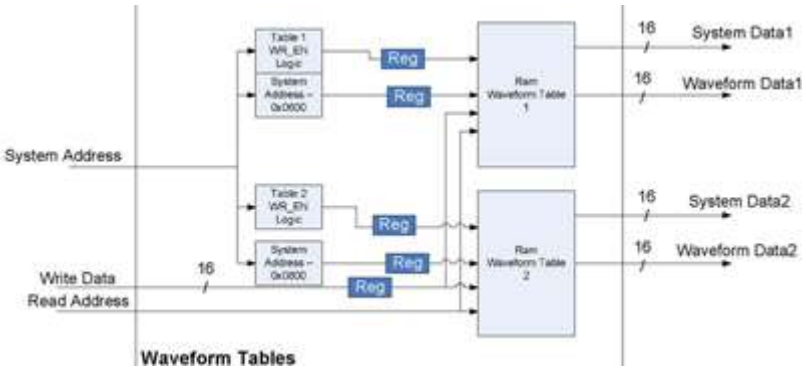


Figure 5: Waveform Table Block Diagram *(To view larger image, click here)*

Waveform control

The waveform control block (**Figure 6, below**) is the interface between the Waveform Tables and core PID control block. When this block receives a start signal from the software, it starts to read the waveform tables.

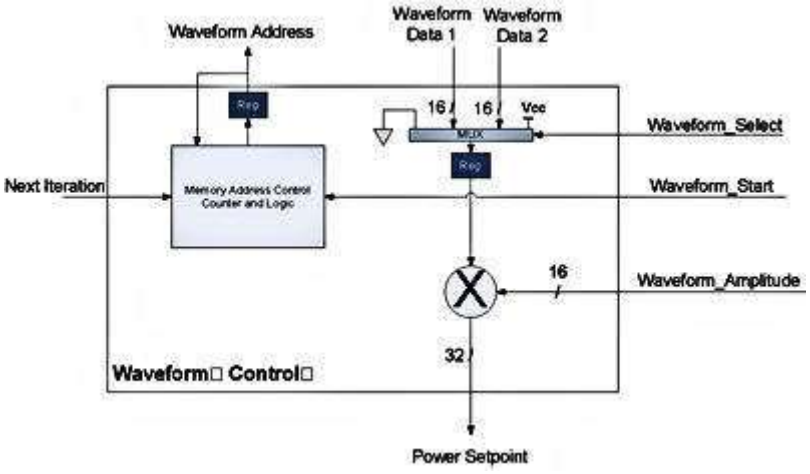


Figure 6: Waveform Control Block Diagram*(To view larger image, click here.)*

It selects which waveform table to read, and then multiplies that normalized waveform data by the Waveform Amplitude from the register file to get a power setpoint for the PID control block.

Page 3

PID state machine

The PID State Machine block of the FPGA (**Figure 7 below**) controls the flow of the entire digital PID loop. It controls each step in the pipeline from reading the raw voltage and current ADC registers through sending a value to the DAC. The current and voltage ADC interfaces send a ready signal to the state machine when they are ready to start another cycle.

The ADCs are enabled separately from the PID loop so that samples may be taken without trying to control the load. The ADCs will continue to take samples at 250 kSPS as long as they are enabled. The state machine will then clear the ready signals, and when both are activated the calibration starts. Once calibration is complete, the moving average filter will be triggered.

If the PID is enabled, the control loop will continue and compute a new DAC value. If the PID is disabled, the DAC will be set to a known value that is configurable by the software. When the PID is enabled again the math results registers will be cleared so that the PID loop will start from a clean state and not be influenced by any previous calculations.

The DAC communication will be started two clock cycles after the ADC conversion starts. This ensures that there will be no DAC activity during the ADC aperture time. The state machine diagram is below.

This state machine includes states for a pipelined implementation of the calculations. A simpler state machine with fewer pipeline states and wait states to wait for long combinational logic and embedded multiplier paths could be used to save register resources and potentially reduce total loop time. Using all of the intermediate pipeline registers does make it easier to debug the design in simulation.

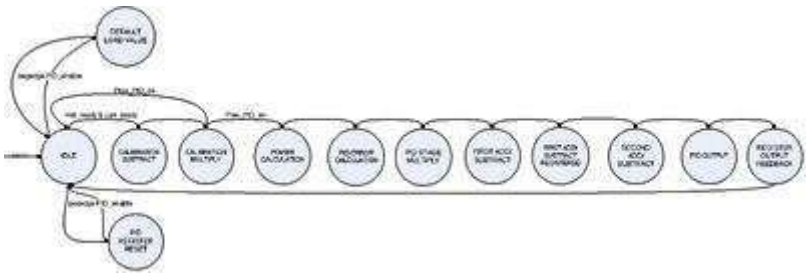


Figure 7: PID State Machine Diagram*(To view larger image, click here)*

PID Control

The PID Control block compares the input power level (calculated from the calibrated voltage and current) to the setpoint power level and calculates an error. Based on this error, a DAC value is calculated that should result in a smaller error on the next iteration of the loop, assuming your parameters are tuned properly.

There are a number of ways to implement PID control in an FPGA. Our equation was based on the Type C equation described in the How to Select a PID Equation section. This is the same equation with some of the terms reordered to optimize the FPGA implementation [4]. The PID equation implemented in the FPGA is the following equation.

$$u(k) = u(k-1) + a_0 e(k) + a_1 y(k) + a_2 y(k-1) + a_3 y(k-2)$$

Where,

e(k) = error term = power setpoint – input power
y(k) = input power
y(k-1) = input power delayed by one sample
y(k-2) = input power delayed by two samples
a_x = xth PID parameter

The PID parameter inputs for this equation are slightly different from the traditional KP, KI, and KD. This control algorithm uses parameters referred to as a0, a1, a2, and a3. They can be calculated from the traditional parameters and the PID control sample rate (Ts) of the loop. Those equations are the following:

$$a_0 = K_I T_s$$

$$a_1 = -K_P - \frac{K_D}{T_s}$$

$$a_2 = K_P + \frac{2K_D}{T_s}$$

$$a_3 = -\frac{K_D}{T_s}$$

The ax PID parameters are integer numbers between –(2-1) and 2-1. The power input and setpoint are integer numbers between 0x0000_0000 and 0xFFFF * 0xFFFF = 0xFFFE_0001. These numbers are in terms of ADC counts, not watts.

In order to deal with negative numbers, the sign of all numbers at each stage of the pipeline that could possibly be negative are tracked in a register separately from the unsigned math operations. The PID parameters could be negative so the signs of those numbers are saved into a register that is separate from the magnitude value.

Logic is then used on the magnitudes and signs of the operands to determine the appropriate math operation to perform and the sign of the result (i.e. a+b, a-b, or b-a for any two operands a and b). It was necessary to

implement the math operations in this way instead of using signed numbers, because the embedded multipliers in the FPGA only work with unsigned numbers.

The PID Controller's(**Figure 8, below**) first calculation in a loop iteration is to find the present error, e(k). It subtracts the input power from the power setpoint. The input power and setpoint are both 32 bits wide so the 33rd bit of the result is then checked to see if the error is negative. If it is then its sign bit is set and the two's complement is performed on the lower 32 bits, making the magnitude value positive again. e(k), y(k), y(k-1), and y(k-2) are then multiplied by the PID parameters.

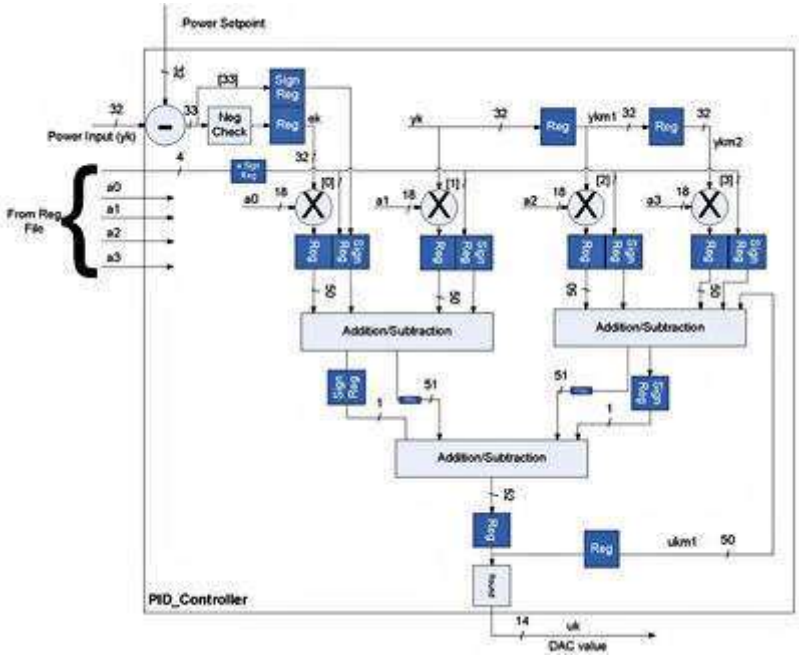


Figure 8: PID Control Block Diagram *(To view larger image, **click here**)*

The sign bits of the multiplicands are XOR'd to get the sign bit of the product. Some logic will then compare the magnitudes and signs of each product (as well as that of u(k-1)) to determine the next appropriate operations (addition or subtraction and their order). The 14 MSBs (DAC data width) are passed to the output, excluding the sign and overflow bits.

All of the inputs into the PID Control block are integers. There are no decimal points in the math in this block. Each of the two Addition/Subtraction levels adds a carry bit to the paths so that at the output there are two carry bits.

These bits are not passed to the output because they only indicate an overflow. If either of these bits is set after the final addition/subtraction stage (see block diagram below), the output will be limited to its max (0x3FFF).

The feedback of this output back into the loop also needs to be limited to prevent windup [2]. If this prevention is not taken, it is possible for the controller to be continually accumulating error when the DAC is maxing out the load, and all of this accumulated error would need to be de-accumulated when the setpoint returns in the opposite direction.

Page 4

System timing analysis

A system timing analysis was performed with a sample rate of 250 kSPS (4 µs sample period). Results from this analysis, and the Simulink simulations described in the System Level Simulation section were necessary to make sure the design would meet the system level goals.

Figure 9 below shows the high-level system timing analysis (not to scale). As can be seen from the figure, the sum total time of all of the individual tasks is greater than the 4 µs (250 kSPS) allocated for the sample rate. Because of this, it was necessary to run the ADC and DAC communication in parallel.

The PID calculations take very little time (10 clock periods in this case) and are completed within one sample period so the calculated values are ready for the DAC output in the next sample period. The consequence of this implementation is that there is a delay of longer than 1 sample period between taking a sample and updating the DAC based on the calculations from that sample. The Simulink simulation was used to make sure this delay would not have a significant impact on the system performance.

During the first sample period (X=0), a default seeded value is sent to the DAC. For every sample thereafter, the values calculated by the PID control during sample X will be sent during sample X+1. The timing analysis was also used to make sure there was no change to the DAC during the ADC aperture window to prevent noise or crosstalk onto the ADC inputs.

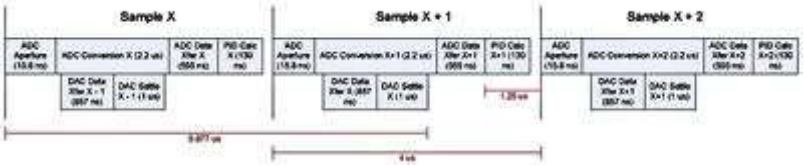


Figure 9: System Timing Diagram *(To view larger image,* ***click here****)*

PID implementation

This section contains some important details about the Verilog HDL implementation of the design for the target Altera FPGA. It also discusses how the control loop was tuned and tested once it was run on real hardware.

Multipliers. These can be implemented in several different ways in an FPGA. There are two main types of multipliers: soft and firm [5]. There are several different variations of soft multipliers, but in general, they use LUTs to store the results for each of the possible input combinations.

Firm multipliers use a combination of embedded multipliers and, if the operation is larger than the embedded multiplier, combinational logic blocks. Soft multipliers are going to always take multiple clock cycles because the result is stored in RAM so they are potentially slower than firm multipliers. There are wizards that will create the custom code for a soft multiplier. Firm multipliers, however, are implemented by default by just using the multiplication (*) operator.

We used firm multipliers to implement multiplication operations in this design, and our FPGA (Cyclone II EP2C50) had plenty of spare embedded multipliers. For more details on the resource utilization of the design, refer to the Resource Utilization and Maximum Clock Frequency section below.

Fixed point math considerations. To create the PID control loop with fixed-point math, one has to keep track of the bit widths of the data at every operation. Every addition or subtraction will add an extra bit. Every multiplication result will have a bit width equal to the sum of the number of bits in the inputs.

As the bit widths grow, so does the amount time required to complete an operation on that data. A 36 x 36 bit multiplication will take longer than an 18 x 18 bit multiplication. This is because the number of logic elements and/or paths increases, and thus, the maximum path delay will increase.

It may be desirable to truncate or round math results at intermediate steps to reduce delay, and to preserve resources. Truncation introduces up to 1 LSB of error, so rounding was used in this design where necessary. For example in the calibration block, the outputs were rounded to the same 16 bit format as the inputs.

Resource Utilization and Maximum Clock Frequency. Our design used an Altera Cyclone II EP2C50F484C6 FPGA. A design with the features shown above throughout this paper, including interfaces to the ADCs, and DAC (both SPI) will consume the approximate number of resources shown in **Table 2 below**. Keep in mind that

maximum frequency can vary depending on the FPGA used and the specific implementation (our system clock was 63 MHz).

Type	Resources
Logic Elements	5900
Total Registers	3200
Total Memory Bits	20480
9-bit Embedded multipliers	24
Maximum Clock Frequency	95 MHz

Table 2: FPGA Resources

With the 10 cycle pipelining present in this design, and running at a 95 MHz clock frequency, the maximum PID loop sample rate that the design supports is 9.5 MHz. This is how fast the PID loop could be run assuming that the A/D and D/A converters used in the system could keep up with this sample rate.

Tuning. We tuned the system using a method very similar to the Ziegler-Nichols method [6]. Since the Type C equation was used, there is no setpoint in the proportional term. This means it was not possible to perform the first step of the Ziegler-Nichols method which is to set the KI and KD terms to zero and increase the proportional gain, KP, until the system oscillates.

So our first step was to choose a very small value of KI, and then increase KP until some overshoot and ringing occurred to measure the frequency to be used for the Ziegler-Nichols formulas based on the ringing frequency. It wasn't possible to achieve a completely unstable oscillating response within the limits of KP in the design.

The next step was to calculate the Ziegler-Nichols values for a PI controller and then manually adjust KP and KI from there to achieve the desired response. The design implemented the full PID controller, but the final configuration only used P and KI. KD was set to 0, as KP and KI alone were enough to achieve the desired response.

The PID control equation used in the FPGA is organized in terms of 'a' parameters to make the implementation easier and to be more efficient with resources and processing time. However, the equation does not lend itself to intuitive tuning for those accustomed to the traditional PID parameters: KP, KI, and KD.

A GUI was developed that interfaced to the FPGA that allowed for entering KP, KI, and KD which were then converted to 'a' parameters and set in the FPGA. The GUI also provided the ability to generate repetitive setpoint step responses. These features were crucial to the tuning process. They allowed the controls engineer to quickly adjust the PID parameters and receive immediate feedback on those parameters' behavior which greatly increased efficiency.

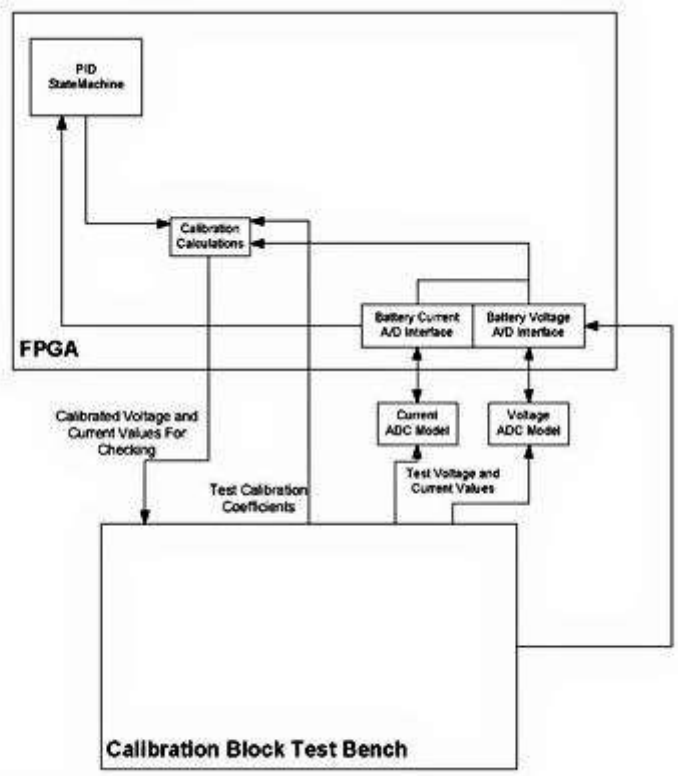
Verification. The FPGA design was verified using simulation with Mentor Questasim 6.4. The simulations were performed using Verilog test benches that provided stimulus to the design and checked the outputs against the ideal, expected outputs. Two separate test benches were created, one to test just the calibration calculations and the other to test the PID control loop as a whole. These are described in the following sections.

Page 5

Calibration block-level test bench

Since the calibration block is quite complicated with different calibration constants used depending on which range the measured raw values of voltage and current fall within, it was decided to verify the calibration block

stand alone so calibration parameters would not affect the test bench of the overall PID control loop. The block diagram in **Figure 10 below** describes the test bench of the calibration block.



Calibration Block Level Test Bench Wrapper

Figure 10: Calibration test-bench block diagram

The purpose of the test bench is to test the calibration calculations block (**Table 3 below**). However, to simplify the test bench, the PID state machine block, which sends pipeline enable signals to the calibration block, was included. The A/D converter interfaces were also included in the simulation to make sure the pipeline control worked properly with these to get the correct data through the calibration block. A Verilog model was created for the ADC devices which had a serial interface to the FPGA. The Calibration Block Test Bench performed the following steps:

•Supply input voltage and current values to the ADC models

•Supply calibration parameters to the Calibration Calculations block

•Send conversion signal to the A/D converter interfaces in the FPGA

•Check the calibration calculation results against expected

•Repeat The test bench was set up to perform the following tests.

Test	Description	Test Details
1	Test each of the voltage calibration ranges	One by one, test each voltage range by setting calibration coefficients for that range as [gain = 1, offset = 0], while coefficients for all other ranges are [gain = 0, offset = 0]. Perform calibration tests with voltage set at bottom, top, and middle of range with expected result being same as input voltage. Perform calibration tests with voltage set 1 count below the bottom and 1 count above the top of range with expected result being 0.
2	Test each of the current calibration ranges	Same as test 1 except for current instead of voltage, and testing is done with voltage set both to a low value and a high value for each current range check. When the voltage is not in the range corresponding to the current range being tested the expected result is 0, otherwise it is the same as test 1.
3	Test voltage calibration calculations across entire range of calibration coefficients	One by one, test each voltage range with calibration coefficients set as follows: • Gain = maximum, offset = 0 • Gain = minimum non zero, offset = 0 • Gain = 1, offset = maximum • Gain = 1, offset = typical For each choice of coefficients, test each range at the low end and high end.
4	Randomized testing of voltage calibration calculations	Set the voltage calibration coefficients in each range randomly. Perform voltage calibration with random voltages for 100 iterations.
5	Test current calibration calculations across entire range of calibration coefficients	Same as test 3 except for current ranges.
6	Randomized testing of current calibration calculations	Same as test 4, except for current calibration instead of voltage.

Table 3: Calibration Block Level Test Bench Tests

The test bench calculates the expected values of calibrated voltage and current in tests 3, 4, 5, and 6 using fixed point math and then rounding the result to the appropriate 16 bit value for comparison with the FPGA calculation.

These calculations are probably similar to what is done in the FPGA code, however, the test bench coding was done by a different engineer than the FPGA designer and the FPGA code was not referenced. Originally, the test bench was designed to compute expected voltage and current calibration values using floating point arithmetic in Verilog with a final scaled conversion to fixed point for comparison with the FPGA. The results matched to within +/-1 or 2 counts with this method because of differences in precision between the fixed and floating point calculations.

PID-loop test bench

A test bench was created to test the entire PID loop, with the calibration coefficients for gain set to 1 and offset set to 0. Some testing was done with typical settings for the calibration coefficients as well, but the final test bench had these values. The block diagram in **Figure 11 below** describes the test bench for the entire PID-loop design.

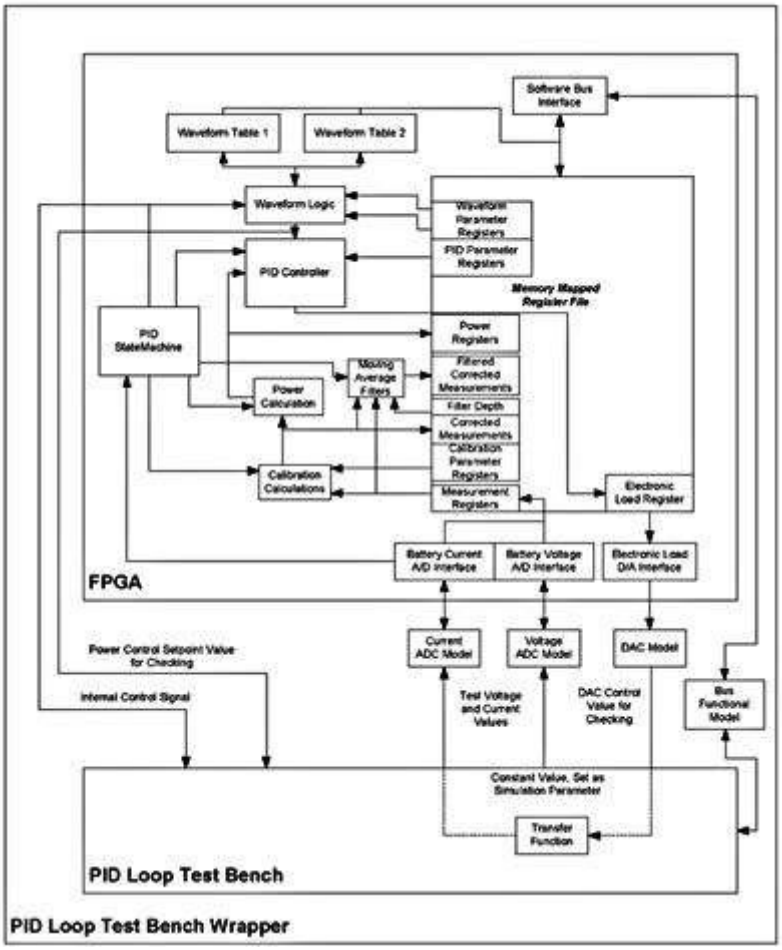


Figure 11: PID-loop test bench block diagram

This test bench uses the same Verilog models of the ADC created for the Calibration Block Level Test Bench in addition to one for the DAC. This test bench operates by setting registers inside the FPGA and writing to the internal waveform tables in the FPGA through the bus interface.

A bus functional model was created to be used by the test bench to interface with the bus in the FPGA. The input value for voltage is a fixed constant set in the simulation, while the current is calculated based on the DAC output control setting using an equation provided by the analog engineer.

The setpoint value had to be extracted from the internal FPGA logic for use in verifying the waveform table functionality. There was also one internal state machine control signal that was used to trigger a register write in the FPGA to make sure that the waveform tables were properly aligned with the simulation test bench (this was confirmed to not be a design issue, but only an issue of making sure the test bench results matched up with the FPGA).

The test bench was originally designed to perform all internal mathematical operations in floating point and then convert the expected DAC output value to fixed point for comparison to the FPGA. In this approach, similar to in the Calibration Block Level Test Bench, differences in precision between floating point and fixed point led to a difference in the results.

For this reason, a similar approach was taken for the math in this test bench where fixed point calculations were performed for comparison to the FPGA results by an independent engineer who did not refer to the FPGA code. The test bench performs the following tests (**Table 4 below**).

Test	Description	Test Details
1	Forced rising and falling step, normal PID parameters, mid level voltage.	The PID control is started with an instantaneous power setpoint step of 0x38A2C75D and voltage set to 0x8000. The pulse runs for 7 ms, then the setpoint switches to 0x00000000 for 5 ms.
2	Forced rising and falling step, normal PID parameters, high voltage.	The PID control is started with an instantaneous power setpoint step of 0x38A2C75D and voltage set to 0xFFFF. The pulse runs for 7 ms, then the setpoint switches to 0x00000000 for 5 ms.
3	Forced rising and falling step, normal PID parameters, low voltage, power setpoint not reachable.	The PID control is started with an instantaneous power setpoint step of 0x2D4ED2B1 and voltage set to 0x199A. The pulse runs for 7 ms, then the setpoint switches to 0x00000000 for 5 ms. Since the PID loop cannot get close to the setpoint (current cannot be made large enough to achieve this power level with this fixed voltage), the loop might exhibit windup and when the setpoint switches back to 0, the output does not come back down. If the FPGA behaved this way, it would be a bug as the FPGA should be designed to avoid windup.
4	Forced rising and falling step, normal PID parameters, mid voltage, power setpoint not reachable.	The PID control is started with an instantaneous power setpoint step of 0xFFFE0001, and voltage set to 0x9999. The pulse runs for 7 ms, then the setpoint switches to 0x00000000 for 5 ms. This test is testing the same windup behavior as test 3 with different setpoint and voltage inputs.
5	Waveform table rising and falling edge, normal PID parameters mid voltage, and mid power setpoint.	The PID control is started using the waveform table for the rising edge up to a setpoint of 0x090FF6F0 and voltage set to 0xCCCC. The pulse runs for 20 ms, then the falling waveform table is used to switch the setpoint to 0x00000000 for 5 ms. This test was meant to simulate a normal system use case.
6	Waveform table rising and falling edge, normal PID parameters, mid voltage, and mid power setpoint.	The PID control is started using the waveform table for the rising edge up to a setpoint of 0x43F6BC09, and voltage set to 0x8000. The pulse runs for 2 ms, then the falling waveform table is used to switch the setpoint to 0x00000000 for 2 ms. This test was meant to be a fast test of the waveform table for use in debugging waveform table behavior, but also provides some interesting coverage of the PID loop because the loop does not have time to settle out from the rising edge before the falling edge starts.
7	PID tuning test, with mid voltage, and mid power setpoint.	The PID control is started using the waveform table for the rising edge up to a setpoint of 0x3FFFC000, and voltage set to 0x8000. The pulse runs for 5 ms, then the falling waveform table is used to switch the setpoint to 0x00000000 for 2 ms. This test was meant to be used to quickly try different PID parameter settings for choosing the normal PID parameters for use in simulation. This simulation was not used for the actual tuning of the real system since the test bench does not include a real model of the control plant.

Table 4: PID Loop Test Bench Tests

Normal PID parameters as discussed in the above table were determined experimentally using test 7 to provide a reasonably shaped response. The purpose of this test bench is to test whether the PID calculations are performed correctly and that the digital design functions as specified and not really to assess the PID control loop performance in system. Therefore, the PID parameters used in this simulation differ from the final tuned values used in the system.

This test bench was designed in a modular way using Verilog functions and tasks to allow different types of tests as shown in the table to be easily constructed. The high level code to perform one of the tests is shown in the following Verilog code.

```
// ********************************
// Execute Test 6
//********************************
if (RUN_TEST_6)
begin
  current_test_number = 6;
  $display("");
  $display("*****************************************************************");
  $display(" ### Test 6 - Quick test of the rising and falling waveform");
  $display(" ###          tables");
  $display(" ###          Constant voltage = 2.5V, Setpoint power = 0.15 W");
  $display(" ###          Run waveform table 1, pause 2 ms, run waveform");
  $display(" ###          table 2, pause 2 ms, Normal PID parameters");
  $display(" ###          Simulation time is %t", $time);
  $display("*****************************************************************");

  voltage_data = 2.5;
  // PID control loop register settings in the FPGA
  waveform_setpoint = 0.15;
  power_setpoint(waveform_setpoint, waveform_setpoint_reg);
  // calculate_pid_parameters task call:
  //   calculate_pid_parameters(Ki, Kp, Kd, T, a0, a1, a2, a3, a_sign)
  calculate_PID_parameters(2.1e9, 1.6e5, 0, 4e-6, a0_reg, a1_reg, a2_reg, a3_reg,
                           a_sign_reg);
  PID_loop_calculation(1'b1, 0, 0);
  voltage_gain       = 1;
  voltage_offset     = 0;
  current_gain       = 1;
  current_offset     = 0;
  setup_pid_registers();
  turn_on_pid(2'b01);

  pulse_delay(2000000);

  // force the falling edge of the pulse
  waveform_select_reg_select      = 2'b10;
  waveform_falling_edge_register_settings(1'b1);

  pulse_delay(2000000);

  // Reset the PID parameters and turn off the adc sampling
  turn_off_pid();

  #500;
end
```

(To view a larger image, **click here**)

As can be seen in the code, the test bench was designed so that key values could be entered in real world units (such as voltage in V, current in A, and power in W) instead of the fixed point ADC numbers used in the design.

The test bench automatically converts these numbers to fixed point representations for use in the simulation, but this interface is much easier for the test bench user to change parameters. One of the key tasks in the test bench is the pulse_delay() task.

This task will allow the FPGA to run, performing the PID control loop, for the amount of time in ns passed in to it. The test bench triggers off of the A/D converter conversion signal to perform its own calculations and check the values of the FPGA design against those expected results for every sample.

The total time to run all 7 of the tests in Mentor QuestaSim on a PC with an Intel Core 2 Duo E8300 CPU running at 2.83 GHz and 4 GB of RAM was 37.5 minutes (as reported by the simstats command from the QuestaSim command line).

The hardware PID option

This design used 5,900 logic elements, 3,200 registers, and 24 multipliers in an Altera Cyclone II FPGA. Based on these numbers, it would be able to comfortably fit into any device in the family as small as the EP2C15. In the more recent Altera Cyclone IV family, it would be possible to get the design to fit into an EP4CE10 device and possibly even the smallest device in the family, EP4CE6.

A hardware implementation for PID controller is not necessarily the best choice for every circumstance, as very low-cost DSP and microcontroller solutions are available, but it can be done with a comparable amount of design effort and extremely high performance.

Paul E. Schad *is a design engineer at Plexus Corp. Current responsibilities include software development and verification planning, board level design, HDL design, and product verification. Received BS Electrical Engineering from University of Wisconsin -Platteville in 2008.*

David T. Carney P.E*. is senior design engineer at Plexus Corp. Current responsibilities include software development, board level design, signal integrity design, and HDL logic design. Recent experience includes systems engineering for telecom rack equipment, software design using C#, System Verilog verification of FPGAs, and software verification planning. Received BSCE from Milwaukee School of Engineering in 1997 and MSE from Milwaukee School of Engineering in 2005.*

Acknowledgements: *The authors would like to thank* ***Scott Zastoupil*** *of Plexus for his input and support with the analog aspects of the PID control loop. Without his efforts, this work would not have been possible.*

References

[1] Ogata, Katsuhiko. 1995. Discrete-Time Control Systems Second Edition. Prentice-Hall, Inc.: Upper Saddle River, NJ.

[2] ?ström, K. and T. Hägglund. 1995. PID Controllers: Theory, Design, and Tuning Second Edition. Instrument Society of America: Research Triangle Park, NC.

[3] Ang, Kiam Heong, Gregory Chong, and Yun Li. July 2005. “PID Control System Analysis, Design, and Technology.” IEEE Transactions on Control Systems Technology, Vol. 13, No. 4.

[4] Lima, João, Ricardo Menotti, João M. P. Cardoso, and Eduardo Marques. October 8-11, 2006. “A Methodology to Design FPGA-based PID Controllers.” 2006 IEEE International Conference on Systems, Man, and Cybernetics. Taipei, Taiwan.

[5] Altera. 2004. “Application Note 306: Implementing Multipliers in FPGA Devices” [Internet, WWW, PDF]. Available: Available in .PDF format; Address: http://www.altera.com/literature/an/an306.pdf. [Accessed: 9-November-2010].

[6] Ogata, Katsuhiko. 2002. Modern Control Engineering Fourth Edition. Prentice-Hall Inc.: Upper Saddle River, NJ.