FPGAFOC

Field Programmable Gate Array Field Oriented Control

Jackson Gray Aaron Yeiser

Introduction

We are planning on designing an FPGA implementation of Field Oriented Control (FOC) for brushless motors. An FPGA is perfect for controlling a brushless motors because it allows complex control loops with substantial amounts of digital processing to execute at high frequency and low latency. Our motor controller will implement field-oriented control---an advanced digital control strategy for driving various types of multiphase motors. Designing the FPGA implementation of field oriented control will involve optimizing and pipelining trig functions and matrix math. A variety of modules will need to be developed for a variety of mathematical operations, transforms, signal conditioning, as well as designs for interfacing with the hardware, including serial interfaces, pwm generators, and large telemetry file handling.

Brushless motors are three phase synchronous permanent magnet motors that require electronic commutation. The lack of mechanical contacts is advantageous for weight and longevity, but they require complex control electronics. There are many different methods for controlling brushless motors, and one of the most useful methods is field oriented control, or FOC. FOC is a control strategy that controls the phase and magnitude of the currents through the three phases of the motor. Specifically, the three observed motor phase currents are mathematically transformed into currents directly in phase with the motor's electrical angle (D current) and in quadrature with the motor's electrical angle (Q current). In an ideal electric motor with no rotor reluctance, we only want Q current to be nonzero. By changing D and Q voltages, we can do an inverse transform to get the target voltages for each motor phase. These voltages are used to generate PWM signals on each motor phase, with these PWM signals optimized to avoid excessive switching.

FOC is an extremely powerful motor control strategy. One advantage of FOC is that the control loop for FOC can be run substantially slower than other similar control strategies because the Q and D currents are relatively slow to change, even with a rapidly spinning motor. FOC can be easily adapted to induction motors, as well as motors with nonzero reluctance torque. With knowledge of motor parameters, sensorless control of the motor can be integrated into FOC. FOC also gives accurate torque control of the motor, enabling a brushless motor to be used as a servomotor.

Goals

The goal of our project is to build a functioning brushless motor controller using FOC with a real electric motor. We recognize that the task of designing interfaces for the power electronics will be challenging and error-prone, so our commitment goal will be to have a working model of the motor controller in simulation. Once we have a working FOC controller, we have many options for stretch goals, including sensorless control, servo position and velocity control, and adding a USB serial interface.

FOC - FPGA Implementation Details

Each block of FOC will be implemented as its own Verilog module, and all of the modules described here implicitly have clock and reset inputs. Each module will also have an individual testbench.

The SVPWM module serves two functions: synchronizing the control loop and generating PWM signals. This module takes in target voltages for each phase of the motor and outputs PWM signals optimized for driving a three phase motor. It also outputs a clock signal at the PWM frequency used for synchronizing the other blocks of the control loop. This frequency will be between 50 and 200 kHz, depending on the switching capabilities of the Prius inverter block. The SVPWM testbench will verify the clock frequency and the signal characteristics of the outputted PWM signals.

The ADC decoder module will be used to interface with the physical output pins dedicated to receiving serial data from an external ADC. It will operate synchronously with the PWM clock, and it will receive the SVPWM timing signal and gate drive signals as inputs. This is because the ADC decoder module should have the ability to reject ADC samples that occur while the inverter is switching due to the switching noise, assuming switching noise is a problem. The output will be the average of valid current sensor data over one PWM clock cycle. This module may also include additional noise reduction signal processing and extrapolation based on SVPWM timing signals.

The SPI receiver module will be used to interface with physical output pins dedicated to reading SPI signals from the resolver decoder. It will operate asynchronously of the PWM clock and will store the received signals in a buffer. The inputs are the SPI clock and data pins, and the outputs are the data buffer and a data ready wire. The testbench for this module will feed in valid SPI test signals and verify the buffer data and the data ready wire timing.

The resolver decoder outputs SPI packets for position and velocity, and we need to separate those two kinds of packets. The resolver demodulator module interfaces with the SPI decoder. It takes a data ready wire and a data buffer as inputs and has two output data buffers--one for position and one for velocity. It also outputs a pulse on the velocity ready and position ready wires when new data arrives. This module operates asynchronously from the FOC control loop and feeds into the position and velocity estimator module. Testing this module will involve putting position and velocity packets in the input buffer and making sure they are sorted correctly and that the data ready wires are triggered properly.

The position and velocity estimator module takes in asynchronous data from the resolver demodulator and outputs position and velocity synchronized with the FOC control loop, as well as a data ready signal. At the beginning of each FOC control cycle, as triggered by the SVPWM module, this module takes the last valid position estimate and multiplies the last valid velocity estimate by the time since the last valid position estimate was received. Then, it will put this new angular position estimate and the last valid velocity estimate into output registers and pulse the data ready wire. The testbench for this module will asynchronously input position and velocity and verify the correctness of the estimations.

The Clark/Park module takes in three signed inputs corresponding to currents through phases A, B, and C of the motor, as well as the angle and data ready line from the position/velocity estimator.
Once triggered by the data ready line, the Clark/Park module performs a sine/cosine lookup using the CORDIC module for the received angle and performs a 3x2 matrix multiplication followed by a 2x2 matrix multiplication to calculate the Q and D currents. These are output registers, as well as a data ready line.

The inverse Clark/Park module takes in Q and D target voltages and angle from the position/velocity estimator. It does sine/cosine lookup using CORDIC and then performs two matrix operations to produce target voltages for each phase. These target voltages are output registers, as well as a data ready line.

The PI controller module takes in a target current and data ready line and outputs a target voltage and data ready line. The Kp and Ki coefficients are module parameters.

The motor simulation testbench will be used to test the FOC control loop in simulation. Inputs will be the motor input voltages and the motor rotation speed, and the motor phase currents will be the outputs. This motor model will include back emf, as well as winding resistance and self-inductance. The SVPWM module outputs will feed into the motor emulator, and the angle and current outputs from the emulator will bypass the position/velocity estimator and ADC interface outputs, respectively.

FOC - Hardware Implementation Details

We will have a handful of hardware components required for our project. This includes the Inverter, a motor with integrated position sensor, a position sensor decoder, and an adapter pcb for interfacing the fpga with the other hardware components.

The Inverter we will be using is an integrated solution designed for the toyota prius (gen2) which has isolated gate driving, isolated current sense amplifiers, and hardware over-current and over-temperature protection. It can be controlled through a 26 pin control connector. This connector has three digital inputs for desired state of the gate drivers for the three phases, as well as two analog output lines representing the current across two of the three phases.

The motor we will be using is a Hyundai Sonata HSG motor, a 30kw 3-phase interior permanent magnet (IPM) motor. It has a variable reluctance resolver on end of the motor shaft, for detecting the rotor position. Additionally, the motor has connections for a temperature sensor and housing grounding.

The resolver connection from the motor will be received by the resolver decoder board, which has AD2S1205 resolver-to-digital IC and it's necessary peripheral circuitry. This IC excites the drive windings of the resolver, and observes the signal's coupling to the A and B sense coils. By observing which coupling is greater, the resolver decoder can deduce the angular position and velocity of the motor rotor. The IC then communicates the position and velocity information via an SPI bus every ~1Mhz.

All of these devices will interface with the FPGA through an adapter board. This board handles both power distribution to the components (except for main inverter power), as well as routing and logic level conversion for the various signals. This board will also have a four channel high frequency 12 bit ADC for converting the analog current sense signals to digital, and a couple smaller low frequency ADC's for reading vbus voltage, and different throttle inputs.

In lieu of the Nexys board, we intend to primarily use the CMOD-A7-35T, an ARTIX-7 FPGA dev board in a DIP format. This device offers us an ARTIX-7 FPGA in a much more compact and portable form factor, so that we can better integrate it into our system. It will interact connect with the motherboard via a high density mezzanine connector, to save space on the main board and so that we can develop a board for connecting through the Nexys pmod connectors, in the occasional event where we want the nexys' resources or peripherals.

Appendix - System Module Details

FPGA Software Module
Test Software
Hardware Task

*A * next to the name means we need to write a testbench for it*
Clean, reset, and enable lines are not explicitly written here

SVPWM Block * (Jackson)

- Inputs
  - Phase A Voltage
  - Phase B Voltage
  - Phase C Voltage
  - Switching frequency
- Output
  - PWM A
  - PWM B
  - PWM C
  - Loop update trigger

- Description
    - Will require one counter register for triangle wave generation
    - Possibly error compensation accumulators

Clark/Park block * (Aaron)
- Inputs
    - Phase A, B, C state vector
    - Angle
- Outputs
    - Q/D transformed signal
- Description
    - Use CORDIC module for sin/cos
    - Pipeline

Inverse Clark block * (Aaron)
- Inputs
    - sin/cos
- Outputs
    - Phase A, B, C state vector
- Description
    - 2x3 matrix multiply

PI controller * (Aaron)
- Inputs
    - Kp, Ki
    - Signal to PI control
- Outputs
    - ~~whatever the fuck you call~~ the output
- Description
    - It is a PI controller

SPI Module * (Jackson)
- Inputs
    - Data word to transmit
    - Write
- Outputs
    - Received Data Word (n bits)
    - Receive Ready
    - Send Ready
- Description
    - Generates or receives clocking dependant on master/slave
- Submodules
    - SPI Rx
    - SPI Tx

Resolver Demultiplexer * (Jackson)
- Inputs
    - Data words received from the spi module (12 bit)
- Outputs
    - Position (two's comp 12 bit)

    - Velocity (two’s comp 12 bit)
- Description
    - Outputs updated whenever a new data word is available from the spi module

Position Generator * (Aaron)
- Inputs
    - Position (two’s comp 12 bit)
    - Velocity (two’s comp 12 bit)
- Outputs
    - Motor Theta (Unsigned n=16 Bit)
- Description
    - Integrates velocity to position to estimate theta at arbitrary times between samples of position
    - Possibly implement fixed gain Kalman filter?

ADC interface (Aaron)
- Inputs
    - Serial interface
    - SVPWM timing signals
- Outputs
    - Voltage (two’s comp 12 bit)
    - ADC trigger
- Description
    - Reads serial signals from ADCs
    - If we determine that electrical is a problem, rejects data received during inverter switching

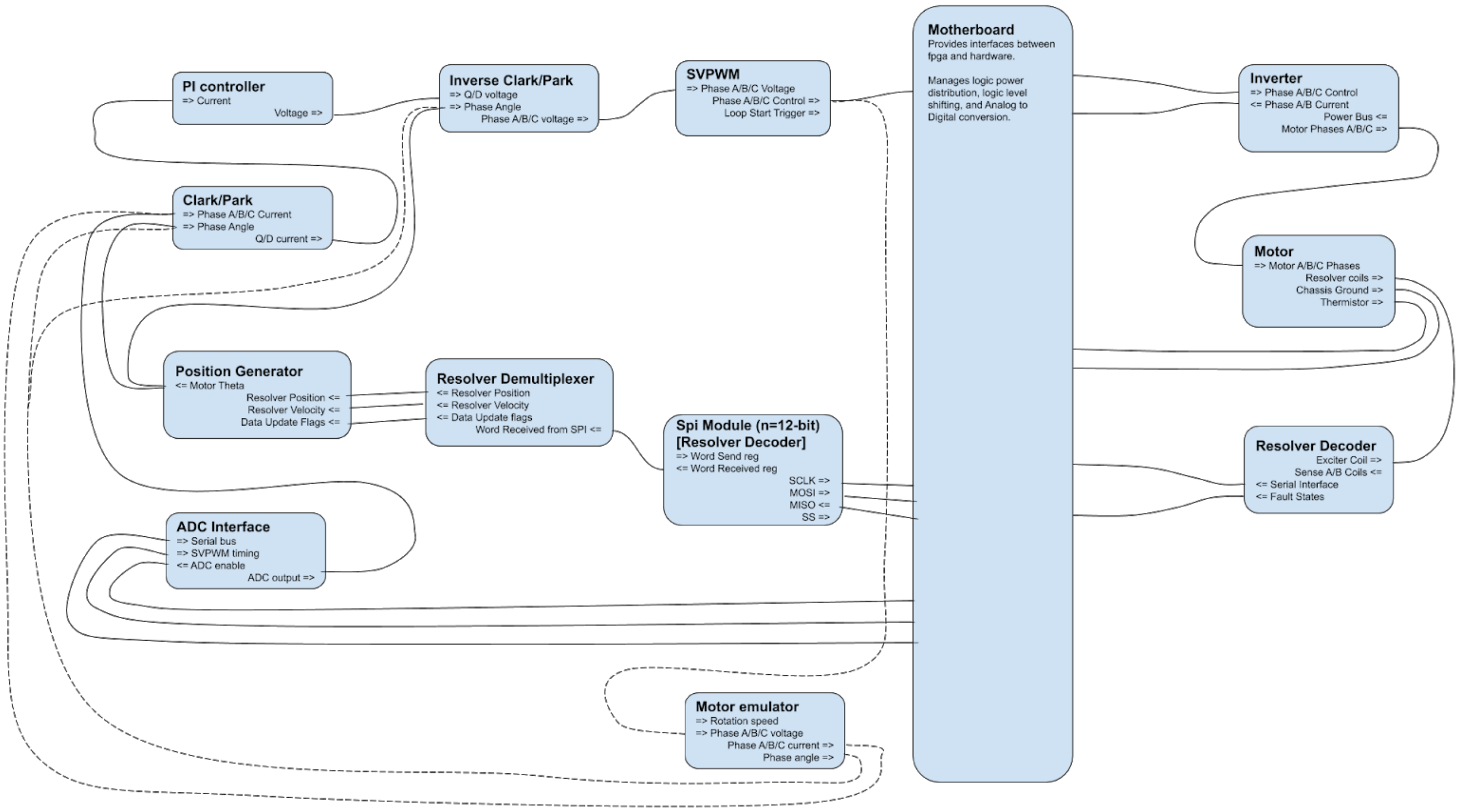
Motor emulator (Aaron)
- Inputs
    - Motor speed
    - Phase A, B, C voltages
    - Inductance, resistance, Kv
- Outputs
    - Phase A, B, C currents
    - Phase angle
- Description
    - A testbench of a sinusoidal linear idealized motor for testing the FOC control loop

Motherboard (Jackson)
- Description
    - Interfaces the FPGA with a variety of connectors and peripherals
    - Delivers power to fpga, resolver, and inverter logic.
    - Carries the high speed ADCs for the inverter current sensors.
    - Performs logic level conversion for the signals going to the inverter.
    - Has connectors for peripheral stuff, like throttle inputs and estops and others.

Inverter
- Inputs
    - Phase A, B, C gate drive
    - Power Bus +/-

    - Signal pwr/gnd
- Outputs
    - Current A, B
    - Phase A, B, C power
- Description
    - Prius power inverter block

Resolver
- Inputs
    - 6.144 - 10.24 MHz crystal oscillator between xtal and clkin
    - sin, cos hi/lo differential signals
- Outputs
    - SPI bus / SCLK
    - Parallel bus
    - EXC hi/lo exciter
    - Fault lines
- Description
    - AD2S1210



PI controller
=> Current
Voltage =>
Inverse Clark/Park
=> Q/D voltage
=> Phase Angle
Phase A/B/C voltage =>
SVPWM
=> Phase A/B/C Voltage
Phase A/B/C Control =>
Loop Start Trigger =>
Motherboard
Provides interfaces between fpga and hardware.
Manages logic power distribution, logic level shifting, and Analog to Digital conversion.
Inverter
=> Phase A/B/C Control
<= Phase A/B Current
Power Bus <=
Motor Phases A/B/C =>
Clark/Park
=> Phase A/B/C Current
=> Phase Angle
Q/D current =>
Motor
=> Motor A/B/C Phases
Resolver coils =>
Chassis Ground =>
Thermistor =>
Position Generator
<= Motor Theta
Resolver Position <=
Resolver Velocity <=
Data Update Flags <=
Resolver Demultiplexer
<= Resolver Position
<= Resolver Velocity
<= Data Update flags
Word Received from SPI <=
Spi Module (n=12-bit)
[Resolver Decoder]
=> Word Send reg
<= Word Received reg
SCLK =>
MOSI =>
MISO <=
SS =>
Resolver Decoder
Exciter Coil =>
Sense A/B Coils <=
<= Serial Interface
<= Fault States
ADC Interface
=> Serial bus
=> SVPWM timing
<= ADC enable
ADC output =>
Motor emulator
=> Rotation speed
=> Phase A/B/C voltage
Phase A/B/C current =>
Phase angle =>