FPGA engineer - Assignment

Instructions

- State any assumptions you may have made clearly.
- Make sure to utilize the limited time in the best way possible when completing the assignment.
- The submitted assignment need ***not*** be buildable/runnable outside your development machine. However we expect you to build/run/simulate the submitted content in your development machine during the subsequent discussion.

Problem - Implement a PID controller to control motor speed

Objective of this assignment is to implement a PID controller to control the speed of a motor. This is to be implemented in two modules.

Module 1: A verilog/System Verilog based IP core which interfaces with an ARM processor through AXI interface. This IP core will expose the following registers.

Reg 1 (Read only): Speed register which gives the speed of the motor in RPM (Revolutions per minute). This can be a 32 bit read only register.

Reg 2 (Write only): PWM period register. This register accepts a 32 bit number which specifies the PWM period in microseconds.

Reg 3 (Write only): PWM dirty cycle register. This register accepts a 32 bit number which specifies duty cycle in microseconds. When the duty cycle is the same as the period, the generated PWM will always be high (100% duty cycle), when it is half the period the generated PWM will be high half the time of the period (50% duty cycle) and so on. The value written to this register needs to be smaller than the value in the period register.

In addition, this module accepts an encoder signal from a sensor attached to the motor which generates 10 pulses per revolution. The module can use this signal to calculate the RPM. This module is supposed to update the Speed register at a frequency of 10Hz. (That is, its speed register needs to be updated after each 100ms period). Make sure to keep the clock frequency and pulses per revolution as parameters in the module and expose them as IP core configurations which can be edited in Vivado. Please see Vivado IP packager documentation for more information on how to do this. (Hint: All you need to do is to expose these params from the top module of the IP core and they will be picked up by the IP packager).

This module should also generate the PWM signal based on the Period/Duty cycle specified through its registers above.

Module 2: A firmware written in C/C++ which runs on ARM.

The purpose of the firmware is to keep the speed of the motor at a given RPM by using a PID controller. The controller loop needs to read the RPM from module 1's speed register, run the PID controller algorithms and decide the duty cycle of the PWM signal that must be generated. The frequency of the PWM signal is fixed at 10KHz.
Note that there are many open source implementations of PID controllers that can run on embedded systems. You can use one of those in this implementation ported to run on your system.

Output

Following waveforms has to be shown in the simulation:

1. PWM output
2. Current motor speed
3. Expected motor speed

At the start of the first 10 second period, the motor speed has to be set to 3000 RPM and the expected motor speed is to be set to 5000 RPM. This should result in a PWM signal with higher duty cycle which is required to increase the speed of the motor.

Gradually simulate increasing motor speed through increased pulse rates from 1-10 second period. This should result in gradually reducing duty cycle as the error is gradually reduced. (It's up to you to decide on how to simulate higher motor speed by increasing pulse rates).

From 10-20 second period, the motor speed to be set to 5000 and the required speed to be set to 2000. Since speed is already higher, the duty cycle should be nearly zero in this period.

Notes

1. IP cores and other Verilog codes should be synthesizable.
2. You may have to write a signal pulse simulator code to simulate the sensor that generates pulses in order to showcase waveforms shown in the Output section above.
3. In case full scope of the assignment cannot be completed by the given time, we encourage you to submit what is completed.