Engine Control Unit

ST: Reconfigurable Computing Final Project

Chris Taylor, Robert McInerney
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
cjtaylor@oakland.edu, ramciner@oakland.edu

Abstract - This project attempts to implement a co-design approach for a electronic control unit (ECU) for an internal combustion engine. The high speed VHDL circuitry works to compute the high speed and accurate DETAILS of a running internal combustion engine. These DETAILS can be reported to the user and used to operate additional electronics to increase performance and reliability of the internal combustion engine.

I. Introduction

All modern day automobiles contain electronic control units (ECU)s. The ECU functions as the 'brain' of the engine and automobile. By adding electronics to the internal combustion engine, the performance and reliability of the system can be improved. This project was adapted from an engine control unit being developed for a unique *rotary valve* internal combustion engine. In this design, the traditional poppet valve train was replaced with a rotating cylinder which allows fuel and air to flow in and out of the cylinder. Below in figure 1 is a comparison of the traditional valvetrain and a rotary valve. This valve design has possible advantages for the engine which were explored by Oakland University's Senior Design projects.

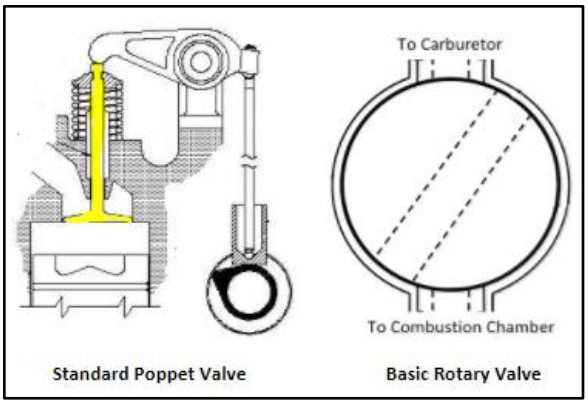


Figure 1: Valve Train Comparison

In order to rotate this unique valve, it was chosen to use electronics involving a servo-motor which creates the need for the ECU. In order for the rotary valve to be actuated correctly, a robust electronic control system which calculated rotational speed and crankshaft angular position. These two pieces of information are critical for driving not only the rotary valve, but any electronics which may be added to this platform in the future.

Utilizing the co-design nature of the Zynq SoC, the high speed digital circuitry created with VHDL is analyzed and configured by the Zynq PS, which has instructions written in C. The combination of these two enables this system to be more accessible and configurable for the future, if parameters change and additional electronic peripherals are added to this project. The entire co-design block diagram can be seen in Appendix A figure 1.

II. Methodology

A. Crank Angle Computer

As previously mentioned, the need for the ECU to compute crankshaft angular position and rotational speed is necessary for successful operation of the engine. A block diagram of the whole component and its derivatives can be seen in Appendix [A] figure 2.

The first important component is the 'pulse counter'. This component reads the incoming pulse train from the sensor and counts the number 'sampling clock ticks' that the pulse is high or low for. The sampling clock frequency was chosen based on the input frequency range, so that an 8 bit value would hold the number anywhere in the range of 1000-5000 rpm.

The gap synchronizer component contains a FSM which receives the tooth count from the tooth counter, and looks for the 'missing tooth' in the pulse train. The pulse counter simply counts the falling edges to generate a 'tooth count'. Because the teeth are a fixed length, and so is the 'missing tooth' it is easy to know which 'low pulse' seen is the missing tooth. This is the fundamental operation of the gap synchronizer. Once it finds the 'gap' it should know which tooth is present, and outputs the 'sync' signal, which essentially means the whole system is working and the angular output can be assumed to be valid. The FSM diagram for this component is shown in Appendix B figure 2 .

The angle counter component also contains a FSM. This component adds the degrees for each tooth, gap, and the missing tooth, whenever it receives the appropriate signals, and 'sync' is asserted. The output angle is in the FX format u[16 6] for accuracy.

The RPM calculator features a text based LUT to calculate the rotational speed (in rpm) based off the width of the previous tooth in sampling clock ticks. Because the tooth width is only 8 bits, and the RPM input range is 1000-5000 rpm, the resolution is rather poor. The output RPM is in the FX format u[16 3] for accuracy.

B. Peripheral Interface

The peripheral interface controller is an important part of our overall system. In the future there might arise a need in order to add other components to the system. These components can include things such as fuel injectors. By utilizing the calculated angular position, these peripherals can be commanded to operate at very precise times during the four-stroke cycle. Most peripherals may require a pwm interface with a duty cycle that can vary depending on the angle that is needed.

Shown in Appendix C Figure 1, the system works. Four independent channels were created for PWM control. It is shown that the duty cycle changes based on parameters that are given to the system via AXI full interface.It can also be observed how the component operates at specific stop and start angles. This is part of the AXI full interface as well. The components inside include an FSM that controls each of the pwm channels. The pwm channels are generic in nature in order to allow easy modification in the future.

C. Servo Controller

The servo controller interface has a similar architecture as the pwm interface. The servo requires a 50 percent duty cycle. This also must change based on the current rpm of the system. So the rpm is fed into the servo controller module to do the calculation.

The current rpm determines the period of the PWM waveform delivered to the servo motor. By adjusting the period between 1 kHz and 100 kHz, greater resolution in speed variation was achieved. To calculate the required period, a pipelined divider was used, as a non-constant division was required for this calculation.

D. RPM Readout Circuitry

It was decided that reporting the instantaneous and maximum RPM achieved to the user may be useful information. In order to do so, a register with a comparator was used to latch the maximum rpm achieved. This value was concatenated with the 16bit current rpm and loaded into the read FIFO. It worked out nicely that these numbers were 16 bits, that way only one read is required to get the maximum and instantaneous rpm.

III. Experimental Setup

A. Simulated Pulse Train

The pulse train received from the crank position sensor is the main input for this ECU to function properly. To ensure that development was continued accurately, the pulse train was captured using an oscilloscope, and a simulated pulse train was created for testbenching purposes. Great care was taken to ensure this simulated signal was as close as possible to the real sensor output. After so many teeth there is a larger gap and then the very next tooth also is a different size. In simulation this signal is created near perfectly. For demonstration purposes, this signal was created with an arduino. This pulse did not turn out as perfect.

B. Testbench Simulations

Great care was placed in ensuring the digital components being created were functioning correctly. Each component was simulated in a testbench. Appendix C Shows each component working.

C. Functional Implementation

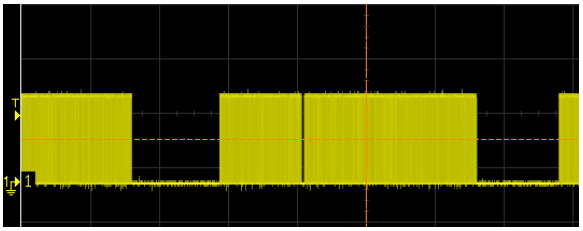
The experimental setup for this device included using multiple hardware tools. First an arduino microcontroller is used to simulate the input of the pulse train(combustion engine). In order to use the arduino with they zybo, a simple voltage divider is used to bring the logic level down from5 to 3.3V. Now the software was loaded onto the zybo. After configuring the zybo with the fpga fabric, the elf file generated from the c instructions was loaded on the board. This allows writing to the input fifo to the AXI Full interface. After two different writes are made to configure the peripheral channels, the appropriate commanded signals can be seen on the respective channel. As illustrated by Appendix C Figure 3, the pwm channel starts after sync is asserted in the second cycle.

IV. Results

A. Challenges

Multiple challenges were faced throughout the development of this project. Some of these shortcomings include that the implementation did not meet timing. The culprit of the timing issue was the use of numeric_std's '/' operator when calculating the period of the pwm signal

generated for the servo motor, as this value is based off the current RPM. The alternative of this was to add a pipelined divider, which resolved the timing issue. After utilizing an adaptation of Professor Llammoca's divider timing was achieved. The next problem to face was the successful operation of the servo motor in hardware. It was quickly observed that there was odd behavior from the pwm channel. Shown below in figure X is the scoped PWM output to the servo motor.



The amplitude and frequency were correct, but the signal was stopping and starting sporadically.

The first guess was that it was not staying in sync. For safety purposes, the system was designed to shut off all outputs when the system was not in sync. This safety feature was disabled, so that the servo would spin no matter if the system was synced or not. This attempt worked, but does cripple some of the control abilities of this system. Without proper 'syncing' the angular position calculation cannot be performed, and unfortunately any positional features will not work. For demonstration purposes the angular position features were removed. It is believed that the imperfect pulse train signal from the hardware is the culprit of the sync loss. With much more time and effort, this system could be improved to tolerate these imperfections and function properly in physical hardware implementation.

B. Functional Operation

After disabling the angular position features of the system, the entire ECU system which involved the AXI full interface appears to work marginally well. The improvements listed above will be good for continued development of this project, but the functional foundation of the project has been laid, and works well to be improved upon.

At first when we implemented the system we did not see exactly what we thought we would. This is due to the fact that the pwm channels did nothing. This did not make sense because we had done simulations without and with the AXI full testbench. We needed to dive into the issue further to really figure out why we saw nothing. We put an oscilloscope to the channel that was associated with running the servo. At this moment we were able to see that there was something happening very wrong in the system. The system was coming out of sync. If the system is out of sync then we will get nothing on the pwm channel because there will be no angle. In order to resolve this issue we tried to do post timing analysis simulations. The simulation of the axi full test bench was not allowing us to do a post timing analysis. So we essentially got rid of the angle component in the system and allowed for modification of the pwm channel and once turned on they stay on.

V. Conclusions

A. Goals Achieved

This project established how the very precise nature of VHDL circuitry can be applied to a real world problem, and can be improved with the use of a co-design combination of a microcontroller. For this particular application it was useful to incorporate user-control of the ECU by utilizing the Zynq PS. The highspeed circuitry was configured by sending explicit commands via the AXI full interface. This

feature was also incorporated for reporting the maximum and instantaneous RPM to the user via the AXI full interface and vivado SDK terminal.

B. Potential Improvements

This project was a very involved and ambitious project for a small team in only one semester. Because of this fact, not every component was working as robust as desired.
Robustness, higher resolution

C. Knowledge

Much insight into not only VHDL development, but co-design principles, the AXI full protocol, and the operation of internal combustion engines was gained through this project. The inclusion of the AXI full interface was very useful in the aspect of co-design. Another topic incorporated into this project is pipelining for improved computation speed, as well as utilizing text I/O's in vhdl, which proved to be a very handy tool to improve computation speed, and reduce design complexity.

References

[1] ECE 4900 lecture notes

Appendix A: Block Diagrams

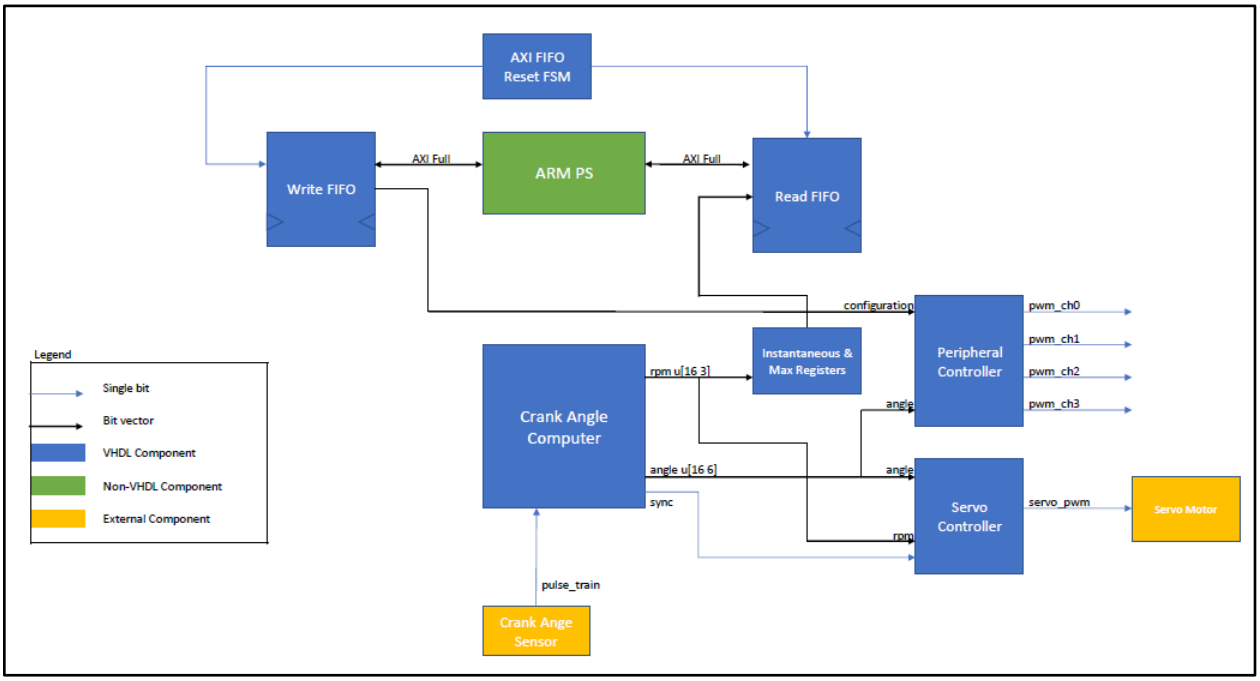


Figure 1: Entire Design

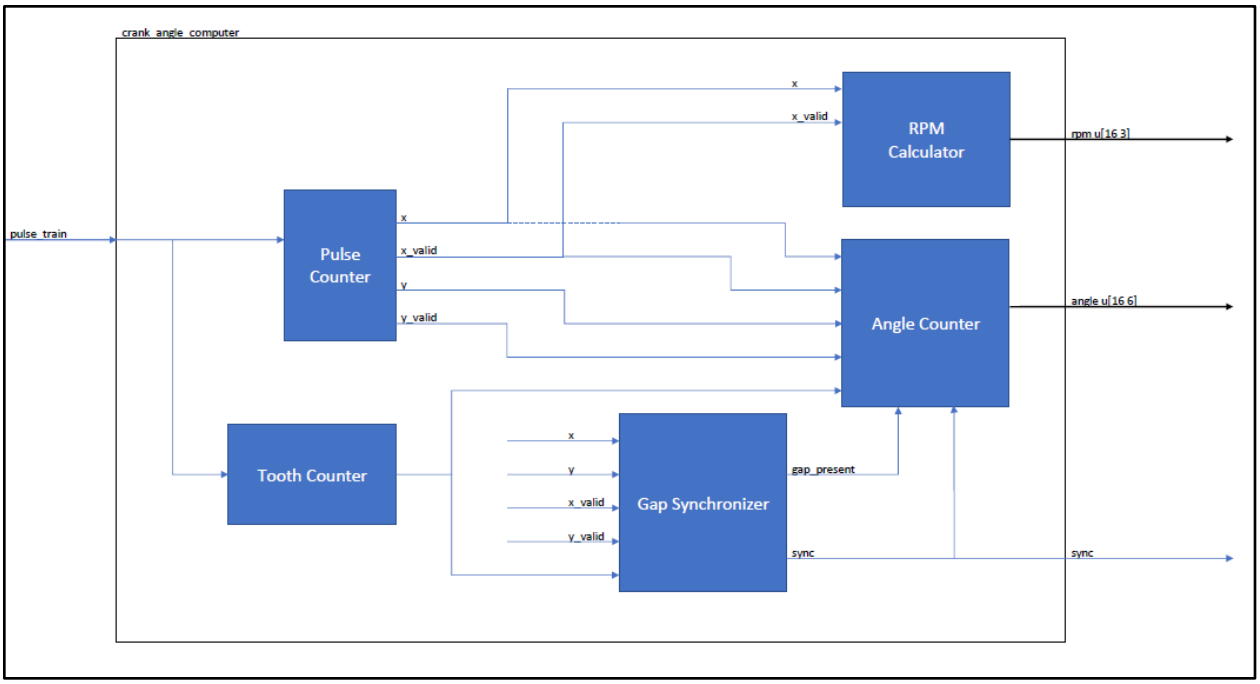


Figure 2: Crank Angle Computer

Appendix B: FSM Diagrams

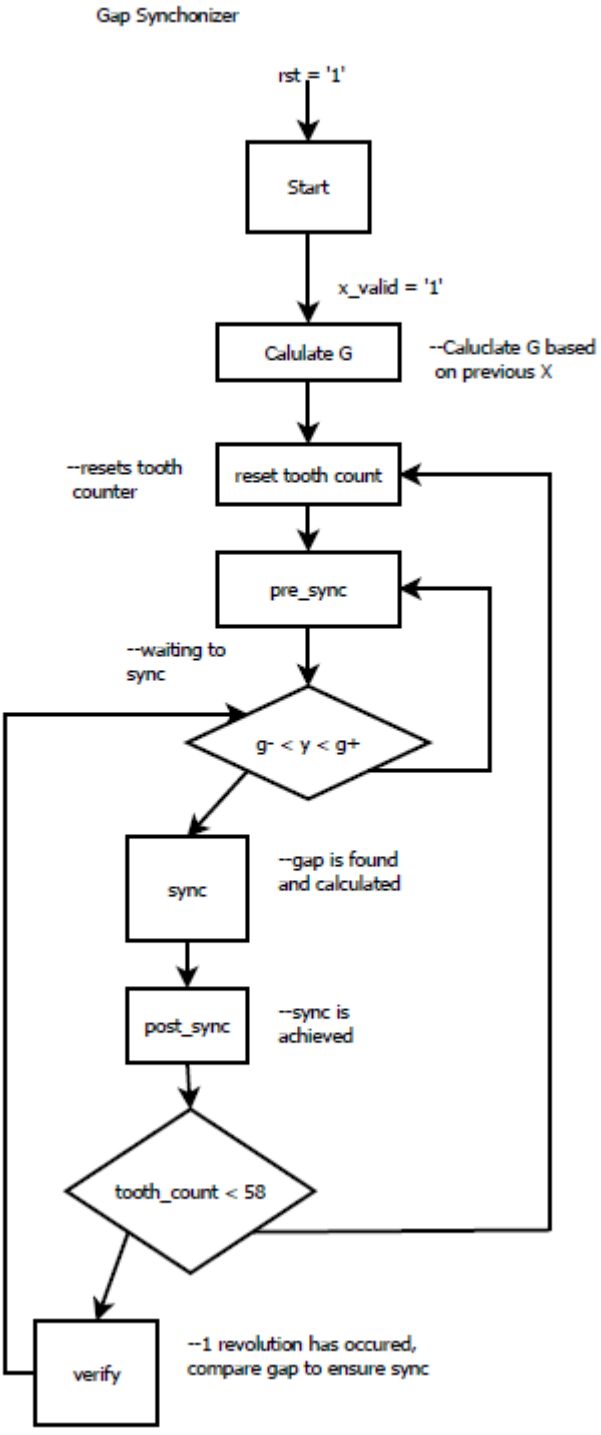


Figure 1: Gap Synchronizer FSM

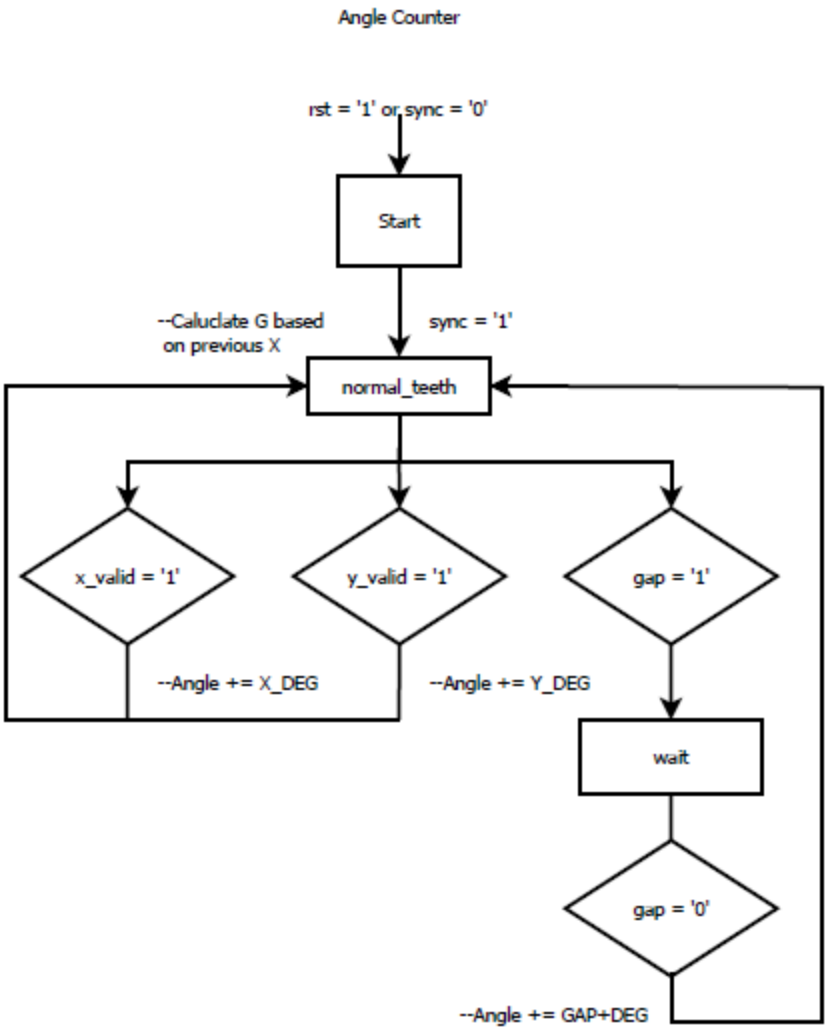


Figure 2: Angle Counter FSM

Appendix C: Timing Simulation Results

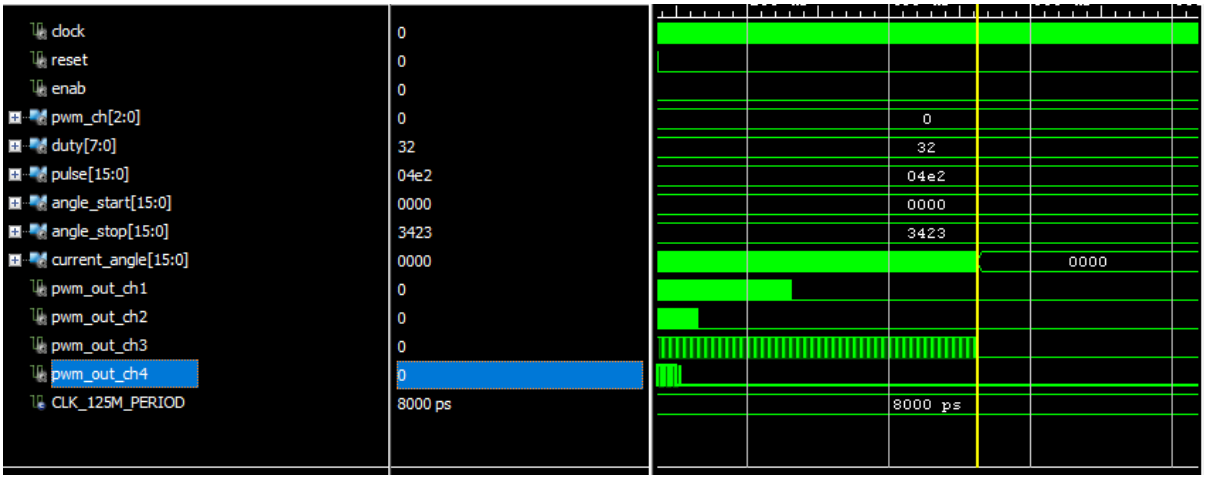


Figure 1: Peripheral Interface Simulation

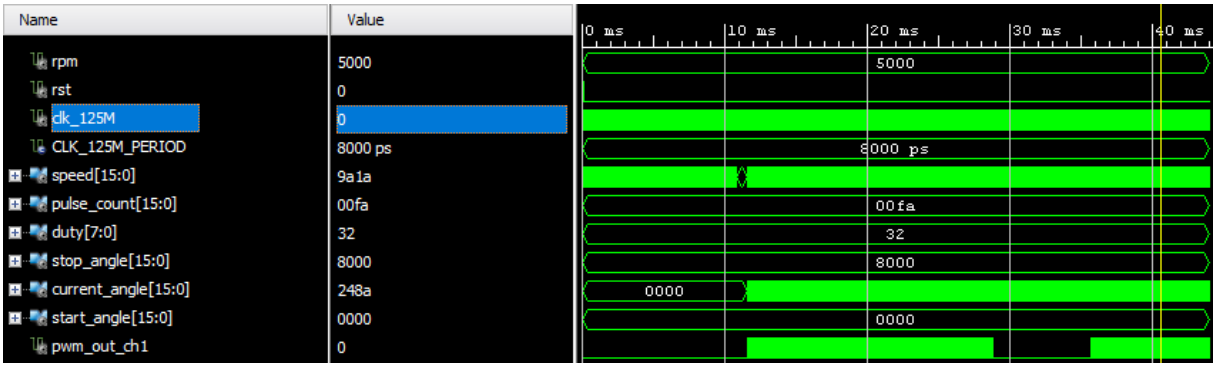


Figure 2. Peripheral Interface Simulation 2

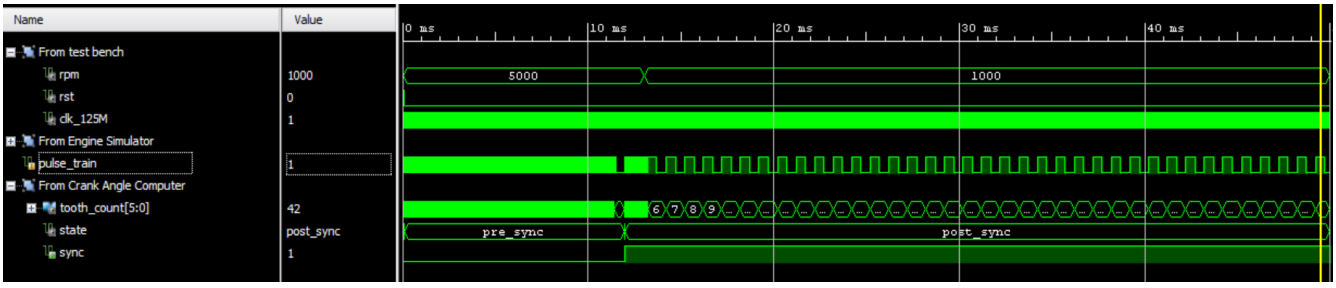


Figure 3. Synchronization after 1 Revolution and Change in input speed

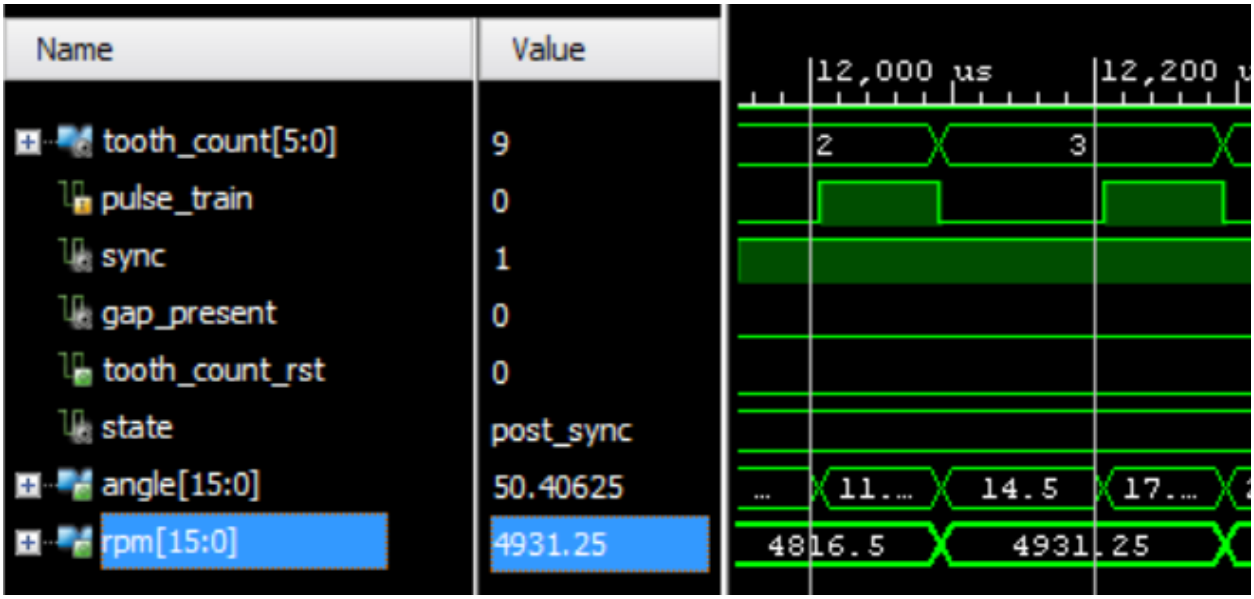


Figure 4: Working Angle and RPM Calculations