Ignition Timing Computer

Team Flux Capacitors

Calvin Nelson

Ryan Kachline

**Executive Summary**

Our project was to create an Ignition timing delay box. This device will serve to delay ignition signals on older cars that still use variable reluctor ignition triggers. These vehicles lack the ability for the user to easily control or adjust the timing of ignition events. We desire to be able to control ignition timing to ensure proper engine operation after modifications are added to the engine. Examples of modifications that would require ignition timing control are turbo chargers, nitrous spray, or methanol injection. Being able to readily adjust the ignition timing is useful for reducing engine misfires, and engine knock.

To implement our project we use an ATmega328 microcontroller. The microcontroller is placed in series with the existing ignition circuitry. When an ignition signal is sent to a cylinder, the signal is sensed by our ignition timing delay box. The ignition timing delay box then decides whether or not an ignition delay is necessary. Our current design implements a manifold absolute pressure sensor. The ignition timing delay box maps the manifold absolute pressure to a delay which is dependent upon the pressure at the time of an incoming ignition signal.

**Problem Statement**

We created an Automobile Ignition Timing delay box for automobiles with Variable Reluctor Ignition Triggers, and HEI 7 pin signal interpreters. The device functions by taking in a reference signal, which is set to have a negative edge at a specific point in relation to every spark event. It takes this reference signal, and creates an output signal, that has a programmable pulse width, that has a negative edge at the desired shifted spark event. Figure 1 below gives a visual representation of the problem.

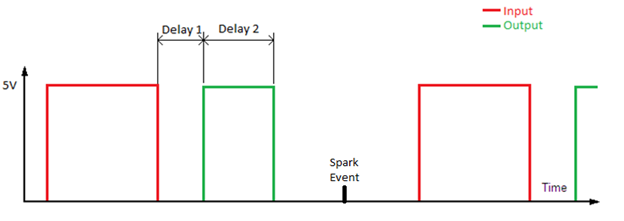


Figure 1: A visual representation of the how our device functions

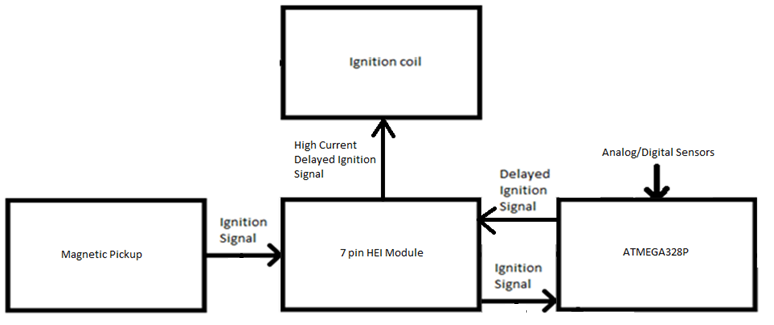


Figure 2: Block Diagram

Justification

This device will be used on older cars that do not have the ability to electronically adjust their ignition timing. On an older car, all of the ignition timing alterations are done mechanically. They mainly operate in response to intake plenum pressure changes, and/or engine speed changes. This device allows for complete replacement of the mechanical components, for increased reliability, and sensor range. Also it allows the ignition timing to react to an additional analog sensor.

This is useful, in cases when you install power adders, such as a supercharger(s), turbocharger(s), and/or nitrous to a car. In these cases, the thermal expansion rate of the combustion charge is increased. In simpler terms, the more air, fuel, and, consequently, heat that is forced into the engine, the larger and faster the combustion is. At low RPM when the engine does not feel the effects of a power adder, you want the ignition timing to act like a stock car. Alternatively, at higher RPM and/or when the power adder becomes active, you want the ignition timing to take into account this increased rate of thermal expansion (faster combustion).

This can be accomplished a number of ways, one example being factoring in the incoming air temperature. As the incoming air temperature increases, so does the rate of thermal expansion. Also, as a safeguard to the engine, devices called knock sensors can be added to the equation as well. Knock is when the ignition timing happens early enough that the maximum force applied by the combustion charge happens before the piston reaches the top of its compression stroke. This can be extremely harmful to an engine, and prevention of it is paramount. Clearly this device is useful in the cases explored above.

Design Requirements

The device will:

* Apply an ignition delay with a minimum step value of 0.5 degrees for a maximum operating speed of 6000 revolutions per minute.
* Use an ATMEGA328P microcontroller
* Involve the design and implementation of a printed circuit board.
* Be able to function in an automobile using automobile power supply.

Calculations

Assuming that the engine will have a maximum operating speed of 6000 revolution per minute, and the engine will be a 4 stroke, 8 cylinder engine, our worst case time between ignition events will be:



This means that the microcontroller in the Arduino will have the following number of cycles to calculate the following ignition delay:



Assuming that the minimum step amount of angular ignition delay is 0.5 degrees, the minimum step length at 6000 revolution per minute, will be:



The Arduino is accurate for any delay above 3 microseconds. Therefore it will suit our needs.

**Proposed Solution**

We accomplished this by using an ATMEGA328P microcontroller, using the Arduino coding environment. We determined that the amount of time it took the processor to calculate delays was not an issue for creating an accurate delay, because the speed of the processor is so much faster than the operating period of an automobile engine. Therefore we did not require an FPGA, or a device that provided more accurate timing.

The device follows the following algorithm:

1. Check manifold pressure
2. Check engine speed
3. Lookup requested ignition timing
4. Interpolate values from table
5. Wait for ignition event.
6. Delay for requested number of degrees.
7. Send ignition event.
8. Repeat.

A strategy employed is using the down time between ignition events to calculate the following ignition delay. By doing this, the speed of the processor is not a huge priority (See calculations section for more details).

Our group is using the Arduino Uno for the following reasons:

1. It is fast enough for our application, and can provide an accurate delay on the order of microseconds.
2. It is relatively cheap.
3. We already own one.
4. Has required number of digital I/O ports for up to an 8 cylinder 4 stroke engine.

We used the Arduino Software IDE for all development purposes. We chose this because we know that it is compatible with the microcontroller. The language that the code is written in is C, thus it uses a variant of a C compiler to execute and load the code.

The design is to include a printed circuit board. Initially we thought that we would need some noise filtering circuitry and this was to be implemented with a circuit board. After testing, however, we found that noise was not a big problem. We then decided to use the printed circuit board to house the microcontroller, as opposed to keeping the microcontroller mounted on the Arduino Uno development board.

**Implementation Details**

Code

As discussed above, the device follows the following algorithm:

1. Check manifold pressure
2. Check engine speed
3. Lookup requested ignition timing
4. Interpolate values from table
5. Wait for ignition event.
6. Delay for requested number of degrees.
7. Send ignition event.
8. Repeat.

For clarification, we will go into more detail on some of the action items.

1. Check manifold pressure:

The Manifold Absolute Pressure sensor functions as a voltage divider. The sensor readings are linearly proportional to the manifold pressure, with a maximum sensor value of VREF. We set VREF to 5V, and used Analog input 1 on the ATMEGA328P microcontroller as our input for this sensor.

2. Check engine speed

The engine speed is obtained by taking the inverse of the period between ignition events, and converting units as needed.

3. Lookup requested ignition timing

The microprocessor has a table that contains all of the requested ignition timing values at all of our load points, as seen in Figure 3.

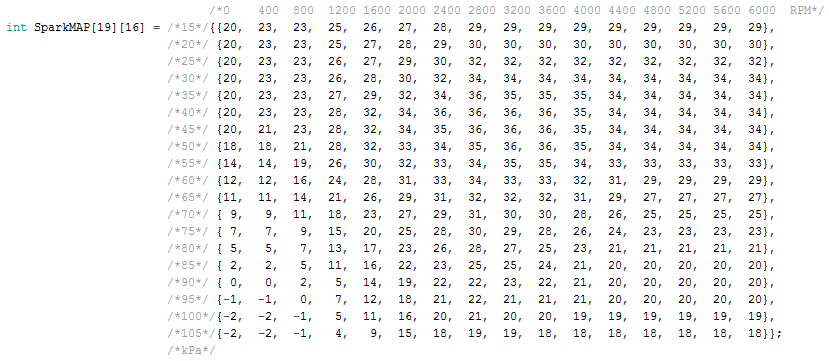
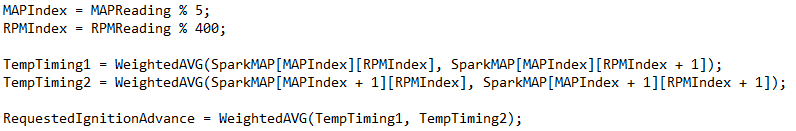


Figure 3: Spark Map

When the program looks up the requested ignition, it grabs the surrounding numbers as well for interpolation.

4. Interpolate values from table

The program interpolates the values that were pulled from the spark map, in order to generate a requested ignition advance value that takes into account how close the spark map index is to one element or another. In order to accomplish this, it follows the following method:



(Note: this is pseudocode, that is intended to demonstrate the process. This is NOT a code snippet taken directly from the code.)

The weighted average functions by linearly interpolating the return value between the two values based on how close the array indices are to one or the other.

5. Wait for ignition event

Specifically, the program is waiting for a negative edge on the reference ignition signal.

6. Delay requested number of degrees

Before we go too far, we must discuss what the generated signal must look like. The generated signal must bring the output high when the coil is to start charging. It will remain high for a requested delay, and then be brought low to create a spark. The code must take this delay into account, as the reference ignition signal does not.

This portion of the code has two conditional paths that it goes to depending on whether the generated ignition signal straddles the incoming ignition event or not. This is required, because the device does not have the ability to predict the future. In the case that the generated ignition is straddling the reference ignition signal, it will look like the scenario in figure 4.

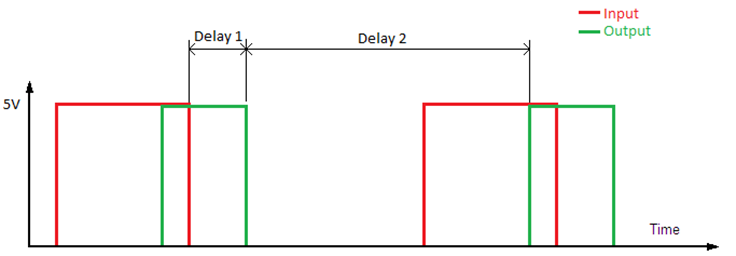


Figure 4: The case that generated ignition is straddling the reference ignition signal.

In order to charge the coil and discharge at the proper time, the code must predict when the coil needs to begin charging for the next cycle before it has any information on when the next cycle is beginning. Engine speed changes slowly in relation to RPM, so this yields very accurate results.

The Algorithm for this process is:

1. Calculate each of the delays
2. Execute the first delay
3. Bring output low
4. Execute second delay
5. Bring output high

In the case that the generated ignition is not straddling the reference ignition signal, it will look like the scenario in figure 5.

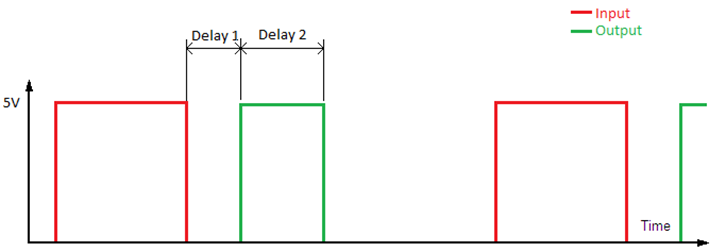


Figure 5: The case that generated ignition is not straddling the reference ignition signal.

In this branch, the code has all of the information, therefore

* Delay 1 = Ignition Delay – Coil Charge Time
* Delay 2 = Coil Charge Time

The Algorithm for this process is:

1. Calculate each of the delays
2. Execute the first delay
3. Bring output high
4. Execute second delay
5. Bring output low

PCB

The main purpose of the circuit board is to house and power the ATmega328 microcontroller. The microcontroller is attached to the board with a 28 pin DIP socket. The microcontroller requires a 5V power supply, so this was also implemented on the board. A power supply that takes in 12V from a car battery and outputs 5V was designed. The power supply circuitry uses an LM317 voltage regulator to output the desired 5V. An led is also included to show when the power is on. The ATmega328 also requires a clock to be added for it to function so we have a 16 MHz crystal at the clock input pins of the microcontroller.

Remaining parts on the board included 2 LF35N op-amp chips. The op-amps were added to allow for future added functionality of the device. We implemented circuits with the op-amps to step a 12V voltage to 5V and vice versa. The switch that can be seen on the circuit board is used to set the gain on the op-amp circuits between the 12V to 5V (or 5V to 12V) gain or a gain of 1.

The last part of interest on the board is the wiring harness seen at the top of the board. The harness has a terminal for the 12V input, along with all the input and outputs necessary for the device to function: MAP sensor input, ignition signal input, ignition signal output..

See figure 6 for a picture of the completed circuit board with all components attached.

**Conclusion/Results**

We successfully made a device that delayed ignition timing on a vehicle equipped with a variable reluctor ignition trigger. Everything, worked with regards to the core function of the device, but we had a few issues with some of the extremities.

Some issues that we had were:

* The op amp portions of the PCB are incorrectly wired.
* There was a trace that had no destination on the PCB.

Some things that we would have done differently are:

* Specified larger package size for resistors
* Organized silk screen to be more assembler friendly on PCB

Some things that we would have liked to have done, but ran out of time to do, are:

* Added rewrite capabilities for the microcontroller on the PCB
* Added reset switch to PCB

All things considered, we are very happy with the outcome. Our device functions flawlessly.

**Appendices**

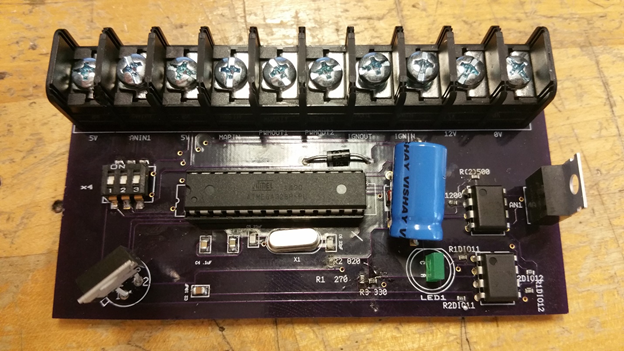


Figure 6: Populated Printed Circuit Board