

Name/NetID:

Teammate/NetID:

Module 3D: Arduino as Signal Generator

Notes:

Laboratory Outline

In our continuing quest to access the development and debugging capabilities of the equipment on your bench at home – Arduino/RedBoard as signal generator. If you have followed the progression of the – Arduino as Bench Equipment – modules you should now be able to supply 3.3V, 5V, and the battery voltage to external circuitry and to use the analog input pins to read voltages at different points in a circuit.

Using the Digital Input/Output (I/O) pins you can craft a time-varying signal. Since the outputs are digital the voltage signal is either 5V or 0V so the resulting signal is a train of pulses of – a square wave. A square wave with a fixed period and a variable duty cycle can be generated with a single statement. Pulse Width Modulation is a technique that uses the variability of the duty cycle to control the behavior of circuit components like the brightness of an LED or the speed of rotation of a DC motor.

The Digital Output Pins on the Arduino/RedBoard

There are 14 pins available that are *digital* pins and can be configured as either Inputs or Outputs. The distinction between analog and digital refers to the nature of the voltages. All voltages are analog in the sense that they are continuous in time and can vary in value. What distinguishes a digital signal is that the voltages are assumed to take one of two values. For the hardware on the Arduino/RedBoard one value is 0V designated LOW when setting the output levels in the code, and 5V designated HIGH in the code. The digital output pins are very good at providing a reliable digital signal with only these 2 possible voltages. On input, since the board has no control over the external signals connected to a digital input pin, the distinction is more complicated so that a range of voltages are considered to be LOW and a separate range is considered to be HIGH.

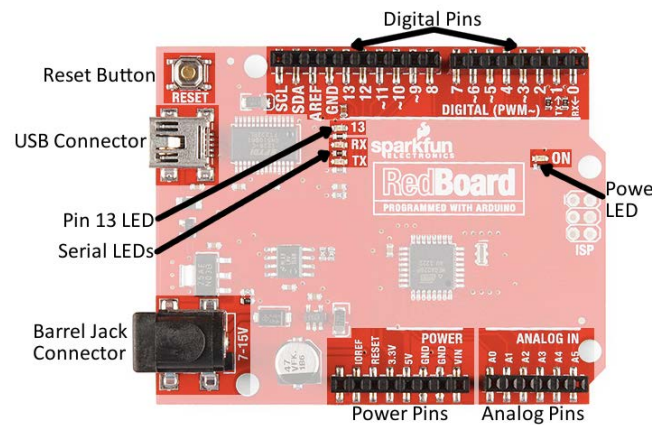


Figure 1: Physical layout of the RedBoard.

Generating a Square Wave

Using a very simple program you can generate a variable duty cycle square wave on any of the digital pins marked with a ~ next to the pin number. Because a common use for these signals is to control the behavior of devices like the motors where the speed is directly proportional to duty cycle the pins with the ~ are labeled PWM or Pulse Width Modulation. You will use this method in a later lab.


Question 1: Which of the digital pins can be used to generate a square wave?

- ✓ Below is the bare minimum code needed to output a square wave using the digital I/O pins. Open a new 'sketch' (the Arduino calls new code files sketches) and type in these few statements.

```
void setup() {  
  pinMode(9, OUTPUT);  
}  
  
void loop() {  
  analogWrite(9, 64);  
}
```

Question 2: This program has no comments so navigate to the Arduino site – arduino.cc. Using the help section indicate purpose of the statements `pinMode` and `analogWrite` by adding comments into the code. Include a copy

of your results – you may write them if you wish or print a copy but they must use the proper comment format. Explain why the **PinMode** command is not really necessary.

- ✓ After checking under the **Tools** menu that the software knows which board you are using (the RedBoard is a clone of the Arduino Uno) and which COM port you are using to upload the program to the board by clicking the  icon at the top of the window. NOTE: When you plug the USB cable into the lab computer the associated COM port is usually the highest numbered port. For Mac users the USB communication ports are the device file names.

Question 3: Once the program has loaded, using channel 1 of the oscilloscope, probe the voltage between the digital I/O pin 9 and any of the ground pins (GND) on the board. Plot the waveform.

Question 4: Modify the code by changing the second parameter of the **analogWrite** function to a different number in the range 0 – 255. Plot the waveform on top of the one you obtained in Questions 3.

Question 5: How is the second parameter related to the duty cycle? Try several different values. Draw a graph or write a simple equation relating the two.

Question 6: There is no parameter when using the `analogWrite` statement to specify the frequency of the square wave because the special hardware that generates the signal always outputs the same frequency. You can only change the duty cycle. What is the period of the square wave?

Question 7: Now add lines to the code so that pin 11 also outputs a square wave with a duty cycle of 50% and pin 9 outputs a duty cycle of 25%. Using channel 2 of the oscilloscope probe the voltage between pin 11 and another GND pin or establish a ground on the breadboard. Plot both signals on the same plot.

Question 8: Trigger the scope on channel 1, then channel 2 – you can do that a few times. Do both signals remain steady or does one drift when you change the trigger channel? You can trigger on channel 3 to see what untriggered signals look like.

Somehow the hardware onboard the Arduino/RedBoard synchronizes all of the PWM signals so that the rising edges occur at the same time even though the pulse widths may be different. This implies that any of the signals can be used to trigger the oscilloscope and all of the waveforms should remain stationary. For our uses it is probably not critical that the waveforms be synchronized but many circuits do rely on these signals to maintain strict timing constraints.

Manually Generating a Square-ish Wave

Using different code you can generate a string of pulses that can vary in width. While the sequence is periodic because the `loop{}` section cycles repeatedly, the signal can be more complicated than a square wave.

- ✓ Add the following code segment anywhere in the *loop {}* section.

```
delay(1);  
digitalWrite(13,HIGH);  
delay(1);  
digitalWrite(13,LOW);
```

- ✓ Upload the code to the board

Question 9: What do the statements *digitalWrite()* and *delay()* do?

Question 10: Using channel 3 of the oscilloscope probe the voltage between pin 13 and another GND pin or establish a ground on the breadboard. Be sure to trigger on Channel (or Source) 3 to get a steady signal. Plot the waveform on a new graph.

Question 11: When the oscilloscope is triggered using channel 3 what happens to the triggering of channels 1 and 2?

Question 12: What is the period of the new waveform?

- ✓ Now insert this statement `Serial.begin(9600);` into the `setup {}` section of the code. This statement establishes communication between the computer and the board as in the Arduino as Voltmeter module.
- ✓ Insert the following code at the end of the loop section. This is nonsense code designed to add a the time needed to complete one round in the `loop()` section. And to show you the looping statement `for()`.

```
for(int x = 0; x < 10; x++){  
  float jabberwocky=sqrt(3.141592654);  
  Serial.println(jabberwocky);  
}
```

Question 13: This has some new statements. What does the `for()` statement do? Explain the purpose of the 3 parameters. Those of you who know C or C++ you will recognize this statement.

Question 14: Plot the waveform on channel 3 on the graph you just made in Question 9. How has the new signal changed? Did the waveforms on channels 1 and 2 change significantly?

That is the power of using the pins that can generate a square wave using the pins marked with a ~. The processor generates signals on these lines as a response to the *analogWrite()* command that is locked to an internal clock. Other statements can be added in the *loop()* section and the timing remains fixed. You **can** do this with the delays but it requires the ability to program the board at the assembly language level.

- ✓ Remove the *for()* loop and program up a crazy looking signal using the **delay** statements.

Question 15: Plot it. Using this signal you can get your car to maneuver by turning the motors on and off for specific amounts of time.