Application Note
Operation of the Jameco and SD Card Data Storage on the Teensy
Ferris Chu

Operation of the Jameco

For our system, we used the Jameco XR-2206, a monolithic function generator, to produce the signal entering the VCO. The XR-2206 IC is capable of producing high quality sine, square, triangle, ramp, and pulse waveforms of high-stability and accuracy. The output waveforms can be both amplitude and frequency modulated by an external voltage. Frequency of operation can be selected externally over a range of 0.01Hz to more than 1MHz.[1]

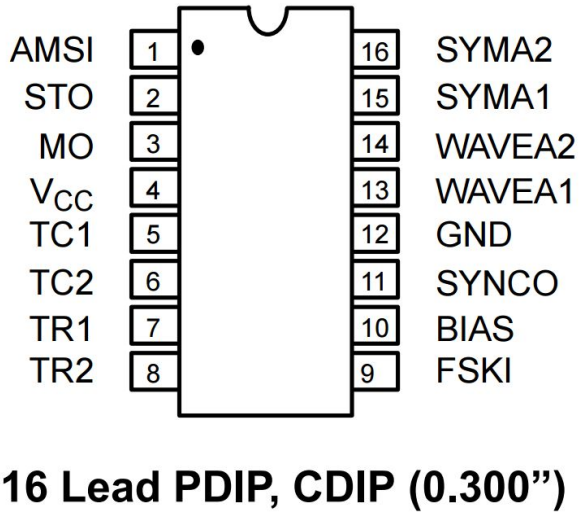

Figure 1: The pin assignments of the XR-2206 IC.

PIN DESCRIPTION

Pin #	Symbol	Type	Description
1	AMSI	I	**Amplitude Modulating Signal Input.**
2	STO	O	**Sine or Triangle Wave Output.**
3	MO	O	**Multiplier Output.**
4	V_{CC}		**Positive Power Supply.**
5	TC1	I	**Timing Capacitor Input.**
6	TC2	I	**Timing Capacitor Input.**
7	TR1	O	**Timing Resistor 1 Output.**
8	TR2	O	**Timing Resistor 2 Output.**
9	FSKI	I	**Frequency Shift Keying Input.**
10	BIAS	O	**Internal Voltage Reference.**
11	SYNCO	O	**Sync Output.** This output is a open collector and needs a pull up resistor to V_{CC}.
12	GND		**Ground pin.**
13	WAVEA1	I	**Wave Form Adjust Input 1.**
14	WAVEA2	I	**Wave Form Adjust Input 2.**
15	SYMA1	I	**Wave Symetry Adjust 1.**
16	SYMA2	I	**Wave Symetry Adjust 2.**

Figure 2: The pin description of the XR-2206.

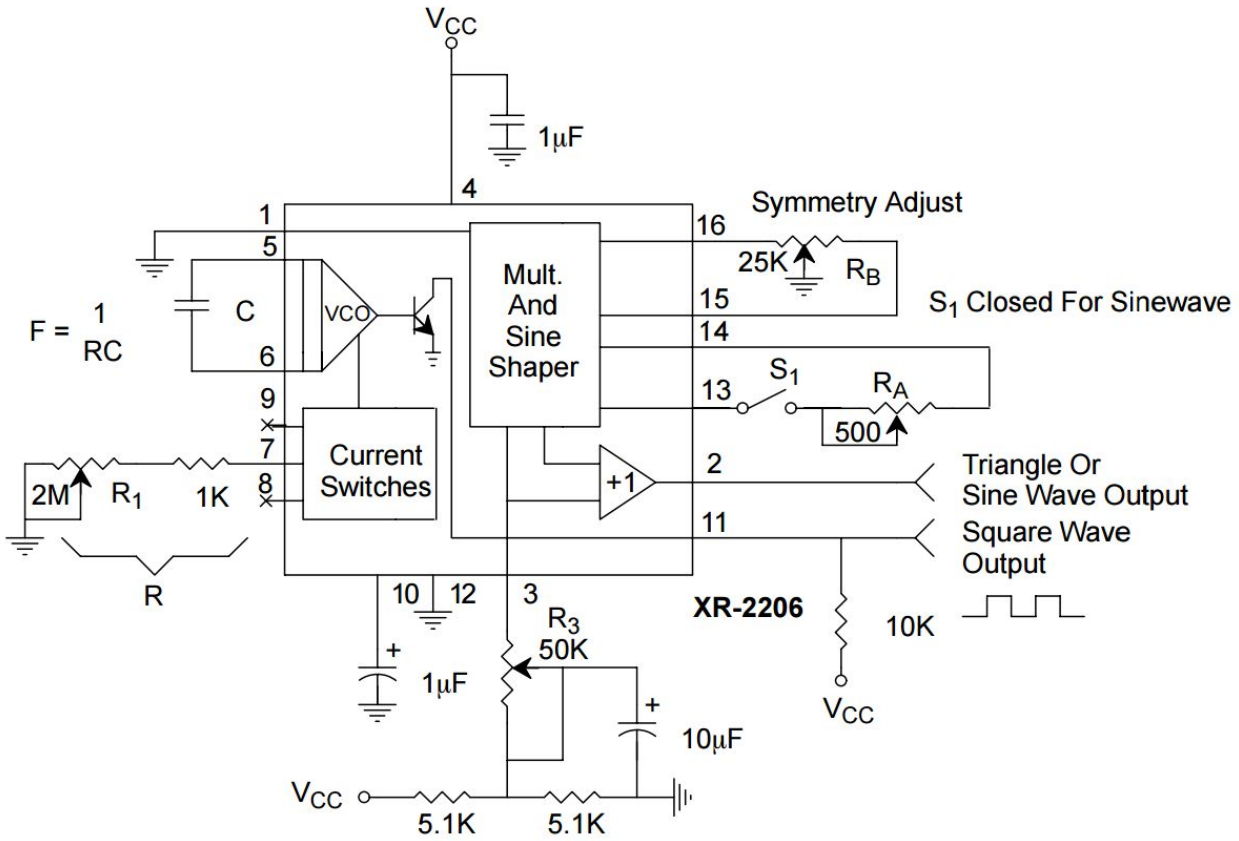

Figure 3: The circuit for sine wave generation with minimum harmonic distortion.

Our connections follow the connections in Figure 3; listed below are some specifications for connections:

Pin 1: GND
Pin 2: Triangle wave output (Going to the VCO (Infineon))
Pin 3: Connected to R3. The components are as follows:
R3 = 31.25 KOhms
Capacitor = 10 uF
Resistors (both) = 5.1 K
Pin 4: Connected to Vcc and 1 uF capacitor.
Pin 5 and 6: Connected by capacitor C (adjustable for frequency of oscillation)
Pin 7: Connected to resistor R (adjustable for frequency of oscillation)
Pin 8: Not used.
Pin 9: Not used.
Pin 10: 1 uF capacitor.
Pin 11: Square wave output (for the SYNC signal) and connected to 10KOhm resistor to GND.
Pin 12: GND.
Pin 13: Not used.
Pin 14: Not used.
Pin 15: Not used.
Pin 16: Not used.

The frequency of oscillation of the generated wave depends on the timing capacitor C (across Pin 5 and 6) and the timing resistor R (connected to Pin 7). The frequency is given as:

$$f_0 = \frac{1}{RC}\ Hz$$

It can be adjusted by varying either R or C. Recommended values of C are from 1000pF to 100uF. Recommended values of R are shown in Figure 4, below. They vary based on the desired operating frequency. Since we were using the VCO on the Infineon and it accepts 0.5 to 5V, we selected a peak output voltage of 5V for the triangle wave. We initially adjusted our potentiometer for R3 = 31.25 KOhms. This can also be adjusted later for fine tuning or for peak voltage of 4.5V.

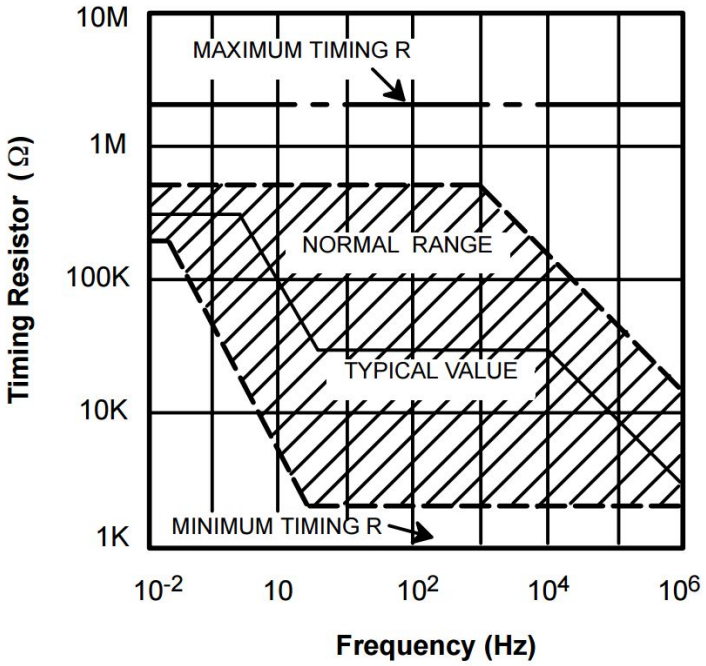

Figure 4: Recommended resistor R values based on desired operating frequency.

SD Card Data Storage on the Teensy

We used the Teensy 3.2 microcontroller to write to the SD Card module with these connections:

```
SD CARD MODULE                                  TEENSY 3.2
CS . . . . . . . . . . . . . . . . . . . . . . Pin 10
MOSI . . . . . . . . . . . . . . . . . . . . . Pin 11
MISO . . . . . . . . . . . . . . . . . . . . . Pin 12
CLK  . . . . . . . . . . . . . . . . . . . . . Pin 13
```

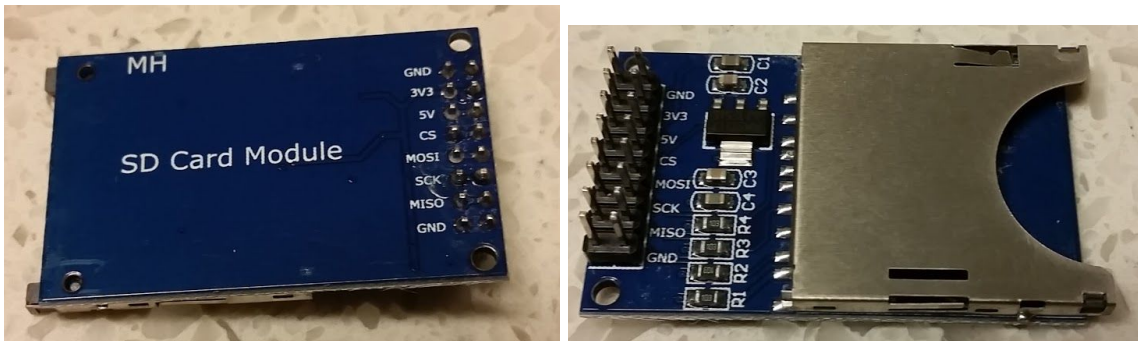

Figure 5: (a) Top view. (b) Bottom view.

We purchased the SD Card module from Amazon. Using the connections listed above, the Teensy could write to the SD Card that is inserted into the slot. Sections of the SD Card data storage code is shown in blue below. It is part of the ADCandSDcard.txt Teensy code.

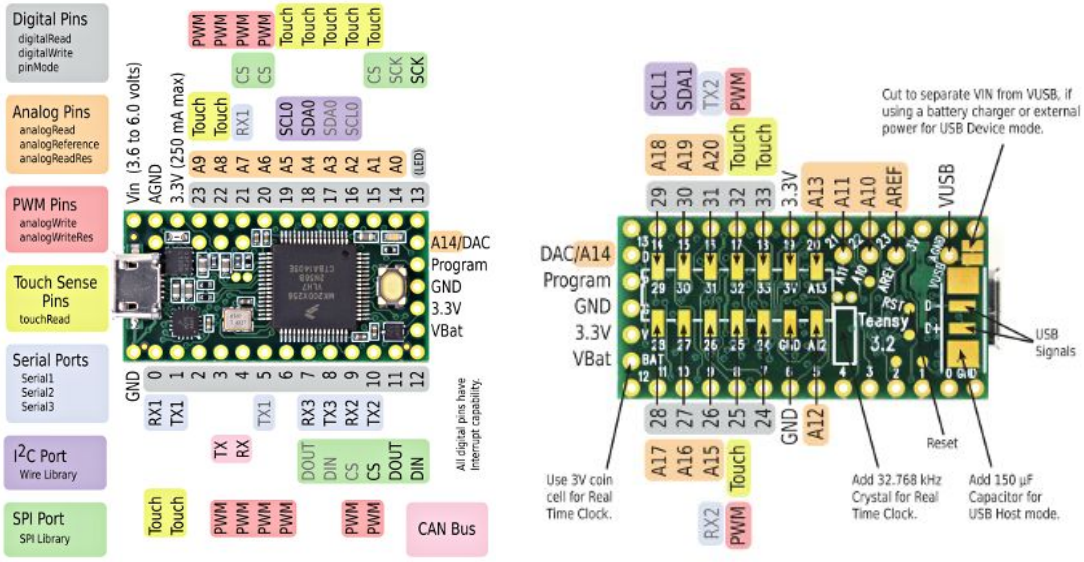

Figure 6: Teensy Pinout (a) Front [2] and (b) Back [3]

```
 * SD Card read/write
 * The SD card should be connected to the Arduino as follows:
 * -MOSI - pin 11
 * -MISO - Pin 12
 * -CLK - Pin 13
 * -CS - Pin 4
 teensy:
 * -MOSI - pin 11
 * -MISO - Pin 12
 * -CLK - Pin 13
 * -CS - Pin 10
 */

#include <ADC.h>
#include <SPI.h>
#include <SD.h>

File myFile;

const int readPin = A9; // ADC0
const int readPin2 = A2; // ADC1

ADC *adc = new ADC(); // adc object;

void setup() {

    pinMode(LED_BUILTIN, OUTPUT);
    pinMode(readPin, INPUT); //pin 23 single ended
    pinMode(readPin2, INPUT); //pin 23 single ended

    Serial.begin(10000);
    while (!Serial){
    ;
    }

      Serial.print("Initializing the SD card....");

      if (!SD.begin(10)){
        Serial.println("Initialization failed.");
      return;
      }
      Serial.println("Initialization done.");

    ///// ADC0 ////
    // reference can be ADC_REF_3V3, ADC_REF_1V2 (not for Teensy LC) or
ADC_REF_EXT.
    //adc->setReference(ADC_REF_1V2, ADC_0); // change all 3.3 to 1.2 if you
change the reference to 1V2

    adc->setAveraging(4); // set number of averages
    adc->setResolution(16); // set bits of resolution

    // it can be ADC_VERY_LOW_SPEED, ADC_LOW_SPEED, ADC_MED_SPEED,
ADC_HIGH_SPEED_16BITS, ADC_HIGH_SPEED or ADC_VERY_HIGH_SPEED
```

```
    // see the documentation for more information
    adc->setConversionSpeed(ADC_HIGH_SPEED); // change the conversion speed
    // it can be ADC_VERY_LOW_SPEED, ADC_LOW_SPEED, ADC_MED_SPEED,
ADC_HIGH_SPEED or ADC_VERY_HIGH_SPEED
    adc->setSamplingSpeed(ADC_HIGH_SPEED); // change the sampling speed

    //adc->enableInterrupts(ADC_0);

    // always call the compare functions after changing the resolution!
    //adc->enableCompare(1.0/3.3*adc->getMaxValue(ADC_0), 0, ADC_0); //
measurement will be ready if value < 1.0V
    //adc->enableCompareRange(1.0*adc->getMaxValue(ADC_0)/3.3,
2.0*adc->getMaxValue(ADC_0)/3.3, 0, 1, ADC_0); // ready if value lies out of
[1.0,2.0] V

    ////// ADC1 /////
    #if defined(ADC_TEENSY_3_1)
    adc->setAveraging(32, ADC_1); // set number of averages
    adc->setResolution(16, ADC_1); // set bits of resolution
    adc->setConversionSpeed(ADC_VERY_LOW_SPEED, ADC_1); // change the
conversion speed
    adc->setSamplingSpeed(ADC_VERY_LOW_SPEED, ADC_1); // change the sampling
speed

    // always call the compare functions after changing the resolution!
    //adc->enableCompare(1.0/3.3*adc->getMaxValue(ADC_1), 0, ADC_1); //
measurement will be ready if value < 1.0V
    //adc->enableCompareRange(1.0*adc->getMaxValue(ADC_1)/3.3,
2.0*adc->getMaxValue(ADC_1)/3.3, 0, 1, ADC_1); // ready if value lies out of
[1.0,2.0] V
    #endif

    Serial.println("End setup");

}

int value;
int value2;

void loop() {

    value = adc->analogRead(readPin); // read a new value, will return
ADC_ERROR_VALUE if the comparison is false.

    //Serial.print("Pin: ");
    //Serial.print(readPin);
    //Serial.print(", value ADC0: ");
    //Serial.println(value*3.3/adc->getMaxValue(ADC_0), DEC);

    myFile = SD.open("ADC_0.txt", FILE_WRITE); // CREATES FILE IF IT CANNOT
FIND ADC_0.txt

    //If the file opened fine, then write:
    if (myFile){
```

```
  myFile.println(value*3.3/adc->getMaxValue(ADC_0));
  myFile.print(", ");

  myFile.close();

} else{

}

// READING ADC VALUES AND PRINTING THEM IN SERIAL PORT

#if defined(ADC_TEENSY_3_1)
value2 = adc->analogRead(readPin2, ADC_1);

Serial.print("Pin: ");
Serial.print(readPin2);
Serial.print(", value ADC1: ");
Serial.println(value2*3.3/adc->getMaxValue(ADC_1), DEC);

myFile = SD.open("ADC_1.txt", FILE_WRITE);

//If the file opened fine, WRITES ADC VALUE ONTO SD CARD:
if (myFile){
  //Serial.print("Writing to test.txt...");

  myFile.println(value2*3.3/adc->getMaxValue(ADC_0));
  myFile.print(", ");
  // close the file
  myFile.close();
  //Serial.println("Done.");
} else{
  // if the file didn't open, print an error message:
  //Serial.println("Error opening test.txt");
}


#endif

/* fail_flag contains all possible errors,
    They are defined in  ADC_Module.h as

    ADC_ERROR_OTHER
    ADC_ERROR_CALIB
    ADC_ERROR_WRONG_PIN
    ADC_ERROR_ANALOG_READ
    ADC_ERROR_COMPARISON
    ADC_ERROR_ANALOG_DIFF_READ
    ADC_ERROR_CONT
    ADC_ERROR_CONT_DIFF
    ADC_ERROR_WRONG_ADC
    ADC_ERROR_SYNCH
```

```
        You can compare the value of the flag with those masks to know what's
the error.
    */

    if(adc->adc0->fail_flag) {
        Serial.print("ADC0 error flags: 0x");
        Serial.println(adc->adc0->fail_flag, HEX);
        if(adc->adc0->fail_flag == ADC_ERROR_COMPARISON) {
            adc->adc0->fail_flag &= ~ADC_ERROR_COMPARISON; // clear that error
            Serial.println("Comparison error in ADC0");
        }
    }
    #if defined(ADC_TEENSY_3_1)
    if(adc->adc1->fail_flag) {
        Serial.print("ADC1 error flags: 0x");
        Serial.println(adc->adc1->fail_flag, HEX);
        if(adc->adc1->fail_flag == ADC_ERROR_COMPARISON) {
            adc->adc1->fail_flag &= ~ADC_ERROR_COMPARISON; // clear that error
            Serial.println("Comparison error in ADC1");
        }
    }
    #endif

    digitalWriteFast(LED_BUILTIN, !digitalReadFast(LED_BUILTIN));

    //delay(10);
}

// If you enable interrupts make sure to call readSingle() to clear the
interrupt.
void adc0_isr() {
        adc->adc0->readSingle();
}
```

Sources:

[1] Jameco XR-2206 Monolithic Function Generator, Part Number 34972, http://www.jameco.com/Jameco/Products/ProdDS/34972.pdf

[2] Teensy Pinout Front, https://www.pjrc.com/teensy/teensy32_front_pinout.png

[3] Teensy Pinout Back, https://www.pjrc.com/teensy/teensy32_back_pinout.png