Application Note: ADC on Teensy and Biasing LNA

Lhawang Thaye

Introduction

This application note will address how to implement an ADC using the Teensy 3.1, the procedure for Biasing the LNA for our system. The ADC is used to convert all analog signal that outputs from the radar system's receiver circuit into digital signals that is then read into an SD-card in our system and then the data would be processed. LNA is used to amplify the signal when it outputs from our receiver antenna.

Biasing The LNA

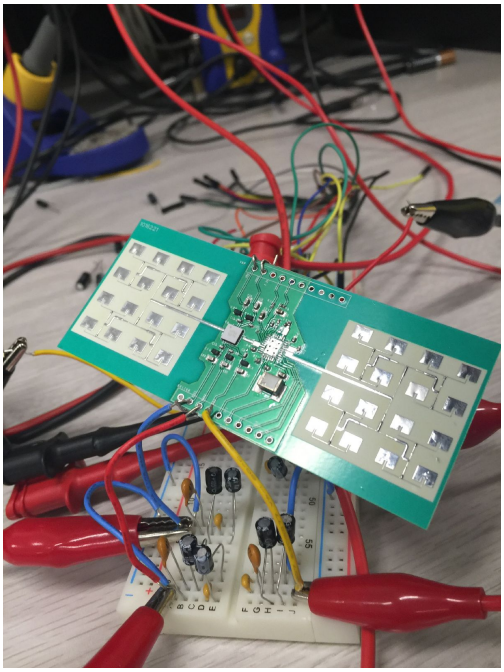

Figure 1: LNA being biased

The HMC752LC4 is a GaAs HEMT MMIC LNA that works in the 24 - 28 GHz range. The data sheet for the device is here:

http://www.analog.com/media/en/technical-documentation/data-sheets/hmc752.pdf

Another important document for this LNA biasing is the procedure manual made by the same company:

http://www.analog.com/media/en/technical-documentation/application-notes/mmic_amplifier_biasing_procedure.pdf?doc=hmc7586.pdf

The procedure for the "***Cascode Distributed Amplifier Bias Sequence*** " is in the link above and this is what I used to bias the LNA. Note that the circuit in the procedure is not identical to the LNA that we used, but the same general procedure can be taken to bias the LNA in our circuit.

ADC on Teensy

The code for the ADC and SD write is a modification of an example from the library by "pedvide" and it can be found here at https://github.com/pedvide/ADC. We used this code as it allowed for the Teensy to be able to read two analog inputs. The ADC and SD card code were combined and uploaded onto the teensy. The code is posted below. Most of the comments are from the original ADC example from pedvide or from the SD card tutorial, but I have added some that will help you understand where the writing and reading of ADC values take place in the code.

```
/* You can change the number of averages, bits of resolution and also the comparison value or range.
 * 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){
  ; //waiting for serial port to connect. This is needed for native USB port only.
  }

    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){
      //Serial.print("Writing to test.txt...");

      myFile.println(value*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");
}

// 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();
}
```