Interfacing the LDC1000 with Arduino

Pelonomi Moiloa
University of Witwatersrand
School of Electrical Engineering

30 June 2014

Contents

1 Introduction

This document details the process of interfacing the LDC1000 with micro-controller Arduino (obtaining proximity and frequency data). Both Arduino Uno and Arduino Mega2560s are used.

2 LDC1000 brief description

The LDC1000 is an inductive sensor. The properties of the sensor change depending on how close the inductive sensor coil is situated to a conductor and also depending on what type of conductor it is. Two sets of data are read by the LDC1000 chip. The proximity data is the first set and gives an indication of the distance of the conductor from the sensor coil. The frequency data is the second set and gives an indication of the type of conductor interacting with the sensor coil.

3 SPI

The LDC1000 communicates with an Arduino using SPI communication. The Arduino acts as the master and the LDC1000 as the slave. At this point the master(Arduino)only communicates with a single slave(LDC1000)but it is possible for the master to communicate with many slaves. A slave is selected by the master by changing the Chip Select Bit ($\bar{SS}$ in Figure 1.)in some way specified by the slaves data sheet.

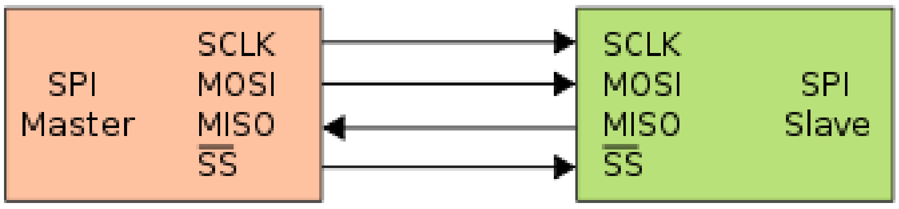

Figure 1: SPI communication

4 Hardware

The hardware components include: an LDC1000 chip with its provided inductor coil sensor connected to the INA and INB pins; An arduino (either an Uno or an Atmega) and an external clock (explained a little later).

The LDC1000 is connected to the Arduino according to Table 1 and Table 2 for SPI communication. These connections can be confirmed with the LDC1000 and Arduino data sheets. (NB: Arduino chip pins numbers do not correspond to numbers found on the Arduino board)

Note: Both ground pins of the LDC1000 must be connected to a common ground for communications to be successful.

4.1 Pin Connections

Either the Arduino Mega or the Arduino Uno may be used. Table 1 and Table 2 show which pins should be connected.

Table 1: LDC1000 Hardware set up top

Slave	Master	LDC1000	Arduino Uno pin	Arduino Mega2560 pin	Other
		5 V	5 V	5 V	
		LDCLK			External Clock
SCLK	SCLK	SCLK	13	52	
SD1	MOSI	SD1	11	51	
		Vio	3.3 V	3.3 V	

Table 2: LDC1000 Hardware set up underside

Slave	Master	LDC1000	Arduino Uno pin	Arduino Mega2560 pin	Other
		GND	Ground	Ground	
		INT			
CSB	CSB	CSB	10	53	
SD0	MISO	SD0	12	50	
		GND	Ground	Ground	

Table 3: Pin Descriptions

Pin	Description
LDCLK	Pin connection for the external clock
SCLK	internal SPI clock
SD1/MOSI	master out slave in (information sent from master to slave)
INT	interrupt pin
CSB	chip select bit
SD0/MISO	master in slave out (information sent from slave received by master)

5 Software

Code is implemented to read registers of the LDC1000 and code is also written to set up an external clock. Refer to arduino.ino files for code for sections 5.2 to 5.3.

5.1 LDC1000 software overview

There are a few notes to be considered before one can transfer information. These can all be found in the LDC1000 data sheet and are all important for successful data transfer:

1. The CSB line is an active LOW
   (a) The CSB line going to the slave is at a default HI, it is pulled LOW to initiate conversation between the master and the slave
   (b) The CSB line must be deasserted (returned to default) after the 16th clock (of SCLK) or otherwise at every $8 * (N + 1)$ clock cycle, for data is written into the LDC1000 registers on the rising edge of the sixteenth clock (SEE: Extended SPI Transactions in the LDC1000 data sheet)
2. The SPI mode is 0 (although mode 3 also works)
   (a) The base clock is idle at logical zero (CPOL = 0)
   (b) data is driven/propagated on the falling edge (CPHA = 0)

   Note: CPOL and CPHA mode combinations can be found on Wikipedia

3. Data is sent from the Arduino to the LDC1000 in byte form MSB first, 16 bits at a time over 16 clocks
   (a) The first byte holds a read(MSB = 1)/write(MSB = 0) instruction in its most significant bit and the register address in question in the remaining 7 bits.
   (b) The second byte is either 0x00 to receive information to be stored or it is the variable the user wishes to store in that register
4. The SCLKmax = 4MHz (Stated by LDC1000 data sheet).

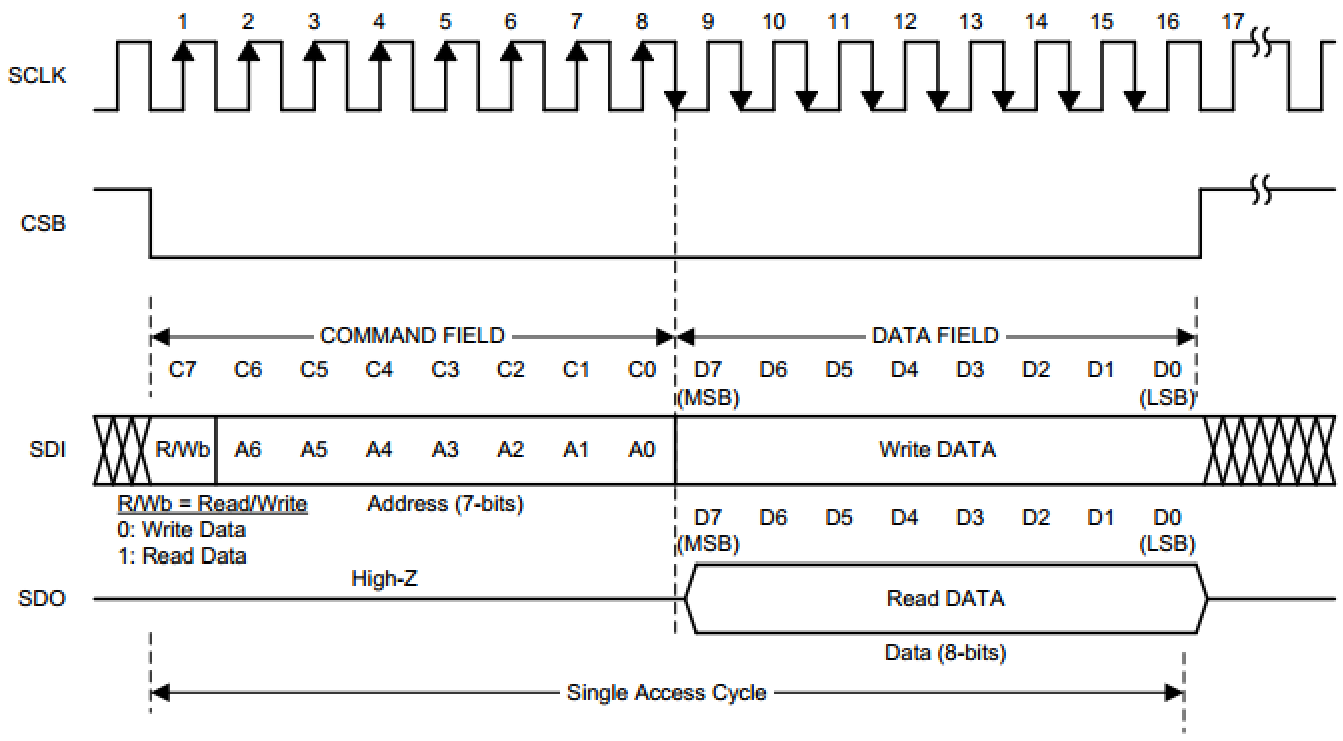

Figure 2: SPI Timing Diagram/ Timing Overview

5.2 Setting up an external clock

Note:

- A frequency generator, crystal generator connected to the Xin and Xout pins or any other 8MHz clock omits the use of this section.
- The external clock is only necessary to read the frequency data.
- The OC2A (PWM output pin) shares the same pins used in SPI communication for the Arduino Uno thus an additional Arduino would have to serve the function of the external clock.

Table 4: Timer2 Control Registers of both Arduino Mega2560 and Arduino Uno

Register	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
TCCR2A	COM2A1	COM2A0	COM2B1	COM2B0	-	-	WGM21	WGM20
TCCR2B	FOC2A	FOC2B	-	-	WGM22	CS22	CS21	CS20

Arduino source code:

```
const int pwm8 = 10;

void setup() {
  // initialize the digital pin as an output.
  pinMode(pwm8, OUTPUT);
  TCCR2A = ((1<<WGM21) | (1<<COM2A0));
  //set WGM to CTC mode and enable output A
  TCCR2B = (1 << CS20);// set prescaler to 1
  TIMSK2 = 0;// disable timer interrupt just in case
}

void loop() {
  Serial.begin(9600);
}
```

Figure 3: External clock Set up Source code

- The clock is set up using Timer2
- OC2A: PWM Timer2 output for Arduino Uno = pin 11/ Mega = pin 10
- **WGM := 010** to select CTC mode (clear timer on comparator match)
- **CS := 001** technically we want clk/2 but by trial and error clk/1 gives us 8MHz
- **COM2A/B := 10** non-inverted PWM input

5.3 SPI communication

- Check to see if communication is successful
- Read proximity data from proximity registers
- Connect external clock and read frequency registers

5.3.1 Communication Check: Reading register defaults

The best way to check that the Arduino is communicating with the LDC1000 correctly, is to read the default values of the registers of the LDC1000. These default values can be found in the LDC1000 data sheet.

Arduino source code seen in Figure 4. The resulting value of the example provided should be 0x1B.

5.3.2 Reading proximity registers

Reading the proximity registers follows the same procedure as reading the default registers but it is necessary to initialize the LDC1000 first. The LDC1000 was initialized to the values found on the LDC1000EVM Gui unless otherwise stated seen in Table 5.

Note:

- The PWR_MODE must be set to 0 to configure register values and brought back up to 1 after the value/s have been initialized to allow for reading of proximity data.

- Sensor frequency, LDC configuration and CLK configuration are configured only when it is required to read frequency data

Table 5: LDC1000 Register Configuration

Register Address	**Register Name**	**Gui Value**	**Initialised Value**
0x01	Rpmax	0x0E	0x0E
0x02	Rpmin	0x3B	0x3B
0x03	Sensor freq	0x94	-
0x04	LDC Config	0x17	-
0x05	CLK Config	0x02	-
0x06	Thres Hi LSB	0x50	0x50
0x07	Thres Hi MSB	0x14	0x14
0x08	Thres Lo LSB	0xC0	0xC0
0x09	Thres Lo MSB	0x12	0x12
0x0A	INTB Config	0x04	0x04
0x0B	PWR_MODE	0x01	0x01

5.3.3 Reading Both Proximity and Frequency data Registers

Reading the frequency data registers follows the same procedure as reading the proximity data registers. However, additional configurations must be made and the TBCLK pin of the LDC1000 connected to an 8MHz external clock. Refer to data sheet for confirmation of the following notes.
Note:

- Clock output connected to LDCLK/TBCLK (from pin 10 if Arduino Mega is used) and ground of clock must e connected to ground of LDC1000/Arduino Master
- Sensor frequency: minimum resonating frequency is set set to 148
- LDC configuration sets amplitude and frequency to 384 and 6144 respectively
- CLK configuration is set to external clock (in comparison to crystal) used and LDCLK/TBCLK (external time based clock) enabled.

Table 6: LDC1000 Additional Register Configurations for reading Frequency Data

Register Address	**Register Name**	**Gui Value**	**Initialized Value**
0x03	Sensor freq	0x94	0x94
0x04	LDC Config	0x17	0x17
0x05	CLK Config	0x02	**0x00**

```
#include "SPI.h" // include arduino SPI library

const int CSB = 10; // chip select bit for Arduino Uno


void setup()
{
  Serial.begin(9600);
  // start SPI library/ activate BUS
  SPI.begin();

  pinMode(CSB, OUTPUT);
}

void loop()
{
  unsigned int val = 0;
  byte READ = 0x80; // MSB = 1 which is a 'read' bit
  byte reg = 0x04; // register address


  SPI.setBitOrder(MSBFIRST);
  // CPOL = 0 and CPH = 0 mode 3 also works
  SPI.setDataMode(SPI_MODE0);
  // set SCLK @ 4MHz, LDC1000 max is 4MHz DIV2 also works
  SPI.setClockDivider(SPI_CLOCK_DIV4);

  // begin data transfer
  digitalWrite(CSB, LOW);
  // MSB = read bit, remaining 7 bits is register address
  byte data_2_send = READ + reg;
  SPI.transfer(data_2_send);
  // send 0x00 to signify a read/store of next 8 bits
  val = SPI.transfer(0x00);

  // prints 8 bit decimal register value
  Serial.println(val);

  // end data transfer
  digitalWrite(CSB, HIGH);

  delay(500);

}
```

Figure 4: Reading default registers