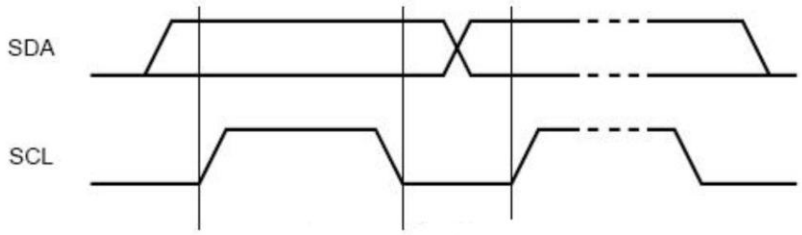
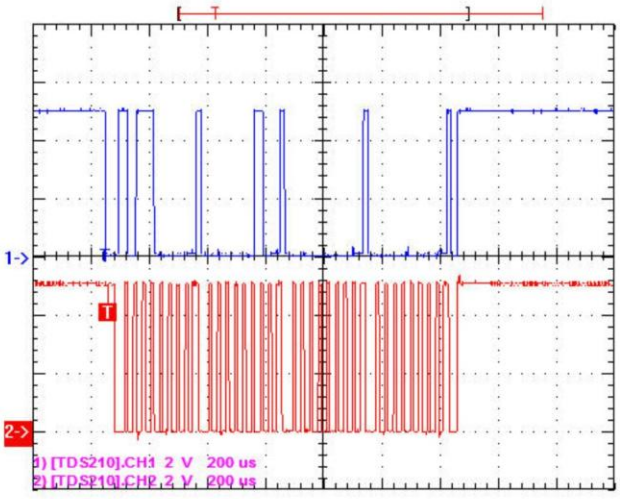


PROGRAMMING THE DALLAS/MAXIM DS1077 133MHZ I2C OSCILLATOR

Jeremy Clark
www.clarktelecommunications.com



Programming the Dallas/Maxim DS1077

133MHz I2C Oscillator

Jeremy Clark

Copyright Information

ISBN 978-0-9880490-1-7

Clark Telecommunications
Jeremy Clark
500 Duplex Suite 506
Toronto M4R-1V6, Ontario, Canada
416-488-5382
info@clarktelecommunications.com
www.clarktelecommunications.com

Contents

Introduction

The Dallas/Maxim DS1077 is an extremely useful programmable oscillator that works from 8KHz to 133MHz. It is available in several models; the one considered in this article DS1077Z-133 covers 16.3KHz to 133.333MHz and comes in an 8pin 150mil SOIC package. It is also reasonably priced at under $3.50 (Ref.1).

I will show how to program the DS1077Z-133 over an I2C bus using common micro-controllers such as the Parallax Stamp BS2 and Atmel ATtiny2313.

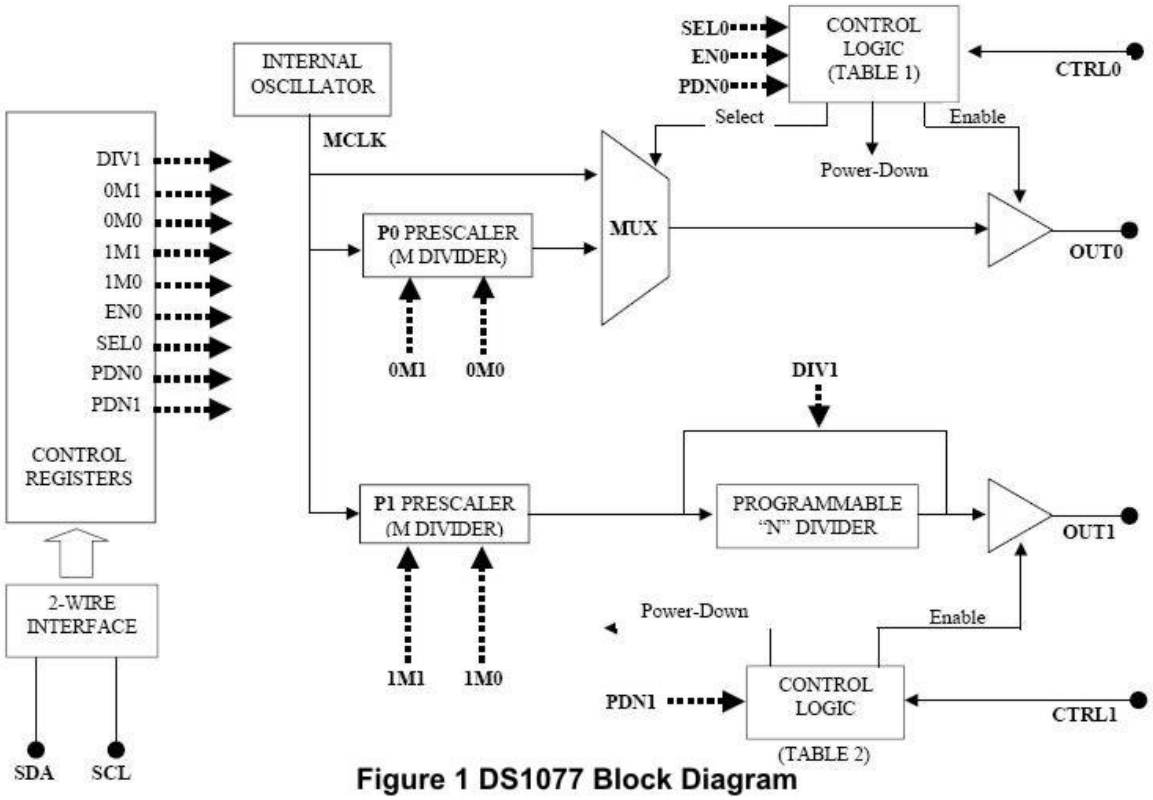


Figure 1 DS1077 Block Diagram

Figure 1 shows the DS1077 block diagram (Ref.2 page2/21). It consists of an internal oscillator working at the unit maximum frequency, in this case 133.333MHz. This oscillator then feeds two output chains – OUT0 & OUT1. OUT0 has a prescaler P0 allowing division by 1,2,4,8. OUT1 has more flexibility and has a prescaler P1 of 1,2,4,8 followed by a programmable divider N of 2,3, …..1025.

Other features are programmable as well. Various power down options are available as well as unit addresses and EEPROM writing options. The default I2C address is 1011A1A2A3 where A1=A2=A3=0. For simplicity, in this article I will use OUT1 only:

- Default address 1011000R/W (A1=A2=A3=0) = $B0 for Write, $B1 for Read
- EN0=SEL0=PDN0=CTRL0=0M1=0M0=0, OUT0 inactive HiZ state
- PDN1=CTRL1=0, OUT1 active
- DIV1=0, enable N for OUT1

In order to determine the settings for the prescaler P1 and divider N, I wrote a small ScicosLab program ds1077.sce. ScicosLab is a free open source scientific program that runs C like programs (Ref. 3). Dallas/Maxim also supplies an Excel program to do the same thing DS10775VFREQ.xls. Note that all code for this project is supplied as a download zip file (Ref.4) and is also listed in Appendix D.

Appendix A shows the two programs and gives a table for the settings for various output frequencies from 16.26KHz to 133333KHz. As an example, for an output of 8MHz, enter OUT1=8000 on line 10 and execute the program (Note that you can't get exactly 8MHz but the closest is 7.843MHz). The results show that the required prescaler is P1=1 and the closest divider N is 17. Note that the divider is reduced by 2 to 15, since divide by 2 is assigned the code 00 0000 0000. The divider N is assigned 10 bits, thus N = 0000001111. Since P1=1, then 1M1=1M0=0. Register bit assignments are shown for MUX and for DIV.

With our initial assumptions regarding default values and simplifications, we are now in a position to define the two setup registers required to use the DS1077. These are the MUX register and the DIV register.

MUX WORD

	MSB							LSB	MSB						LSB
Name	*	**PDN1**	**PDN0**	**SEL0**	**EN0**	**0M1**	**0M0**	**1M1**	**1M0**	**DIV1**	-	-	-	-	-	-
Default setting	0	0	0	0	0	0	0	0	0	0	x	x	x	x	x	x

DIV WORD

MSB							LSB	MSB							LSB
N9	N8	N7	N6	N5	N4	N3	N2	N1	N0	X	X	X	X	X	X

MUX = 0-PDN1-PDN0-SEL0-EN0-0M1-0M0-1M1-1M0-DIV1 x x x x x x
DIV = N9-N8-N7-N6-N5-N4-N3-N2-N1-N0 x x x x x x

So for our example of OUT1 = 8MHz (closest frequency actually 7.843MHz)

MUX = %0000 0000 0000 0000 = $0000 = 2 bytes
DIV = %0000 0011 1100 0000 = $03C0 = 2 bytes

DS1077 Programmed by Stamp BS2 Micro-Controller

Hardware Configuration

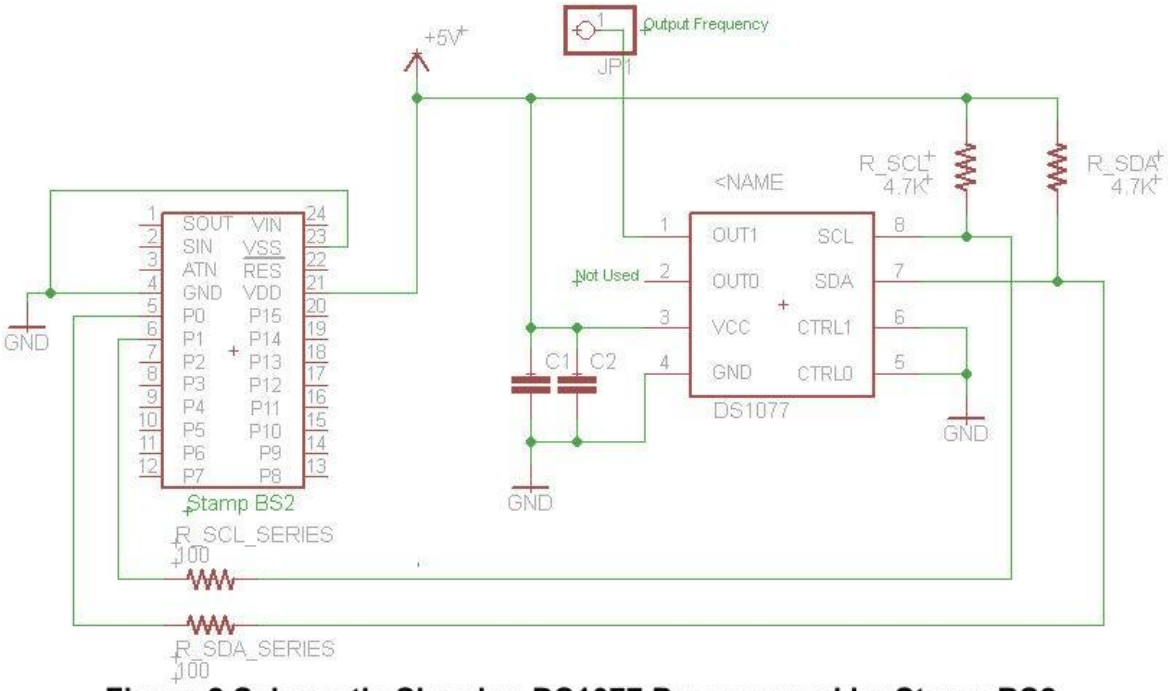


Figure 2 Schematic Showing DS1077 Programmed by Stamp BS2

In order to program the DS1077, I first mounted it on a Bellin Systems SO8 surface mount prototyping board shown in Figure 3 (Ref. 5). I placed pin headers on either side of the board to allow insertion onto the parallax USB protoboard shown in Figure 4 (Ref.6). The interconnection schematic is shown in Figure 2.

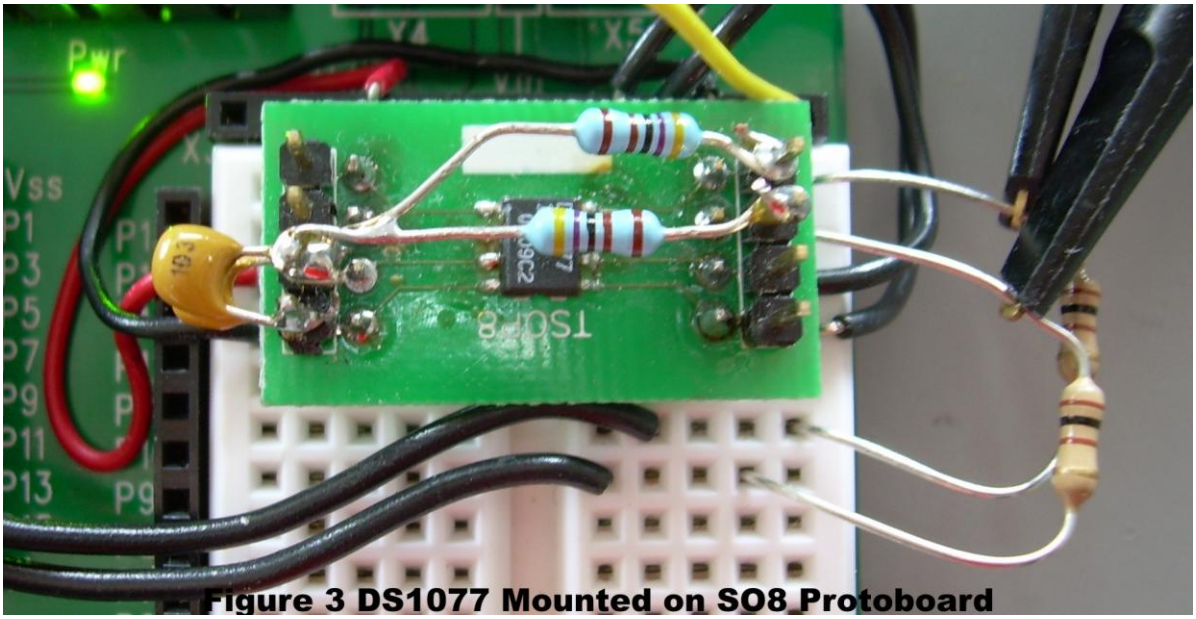


Figure 3 DS1077 Mounted on SO8 Protoboard

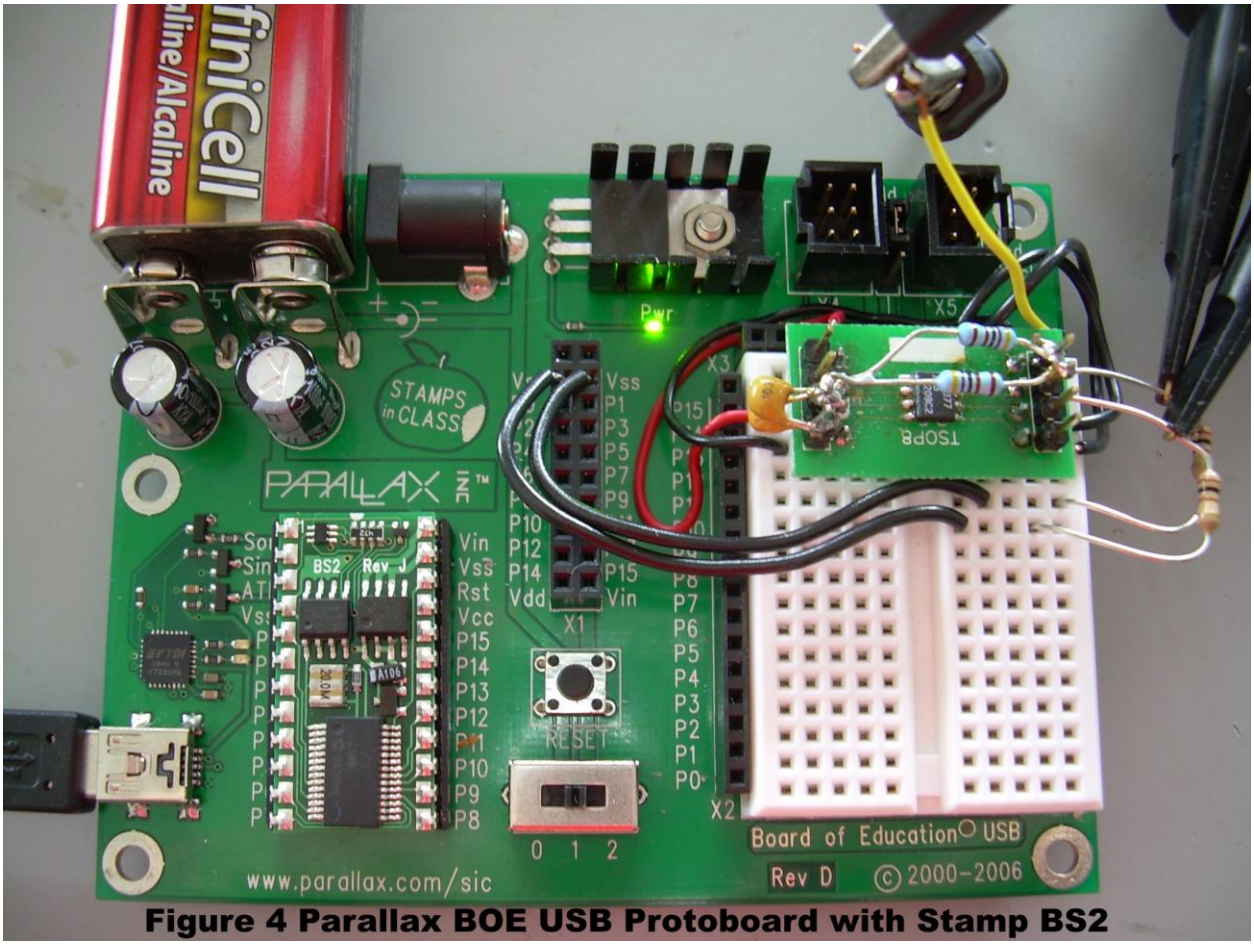


Figure 4 Parallax BOE USB Protoboard with Stamp BS2

Since we are working at an upper frequency of 133MHz, keep all leads as short as possible, less than several inches (<< 1 wavelength = 300/133 = 2.3m = 89in). DS1077 connections are as follows:

- Pin 1 = OUT1 square wave output set by P1 and N
- Pin 2 = OUT0 = HI-Z unused in our case
- Pin 3 = VCC = 5VDC = VDD on Parallax BOE USB board
- Pin 4 = GND = VSS on Parallax BOE USB board
- Pin 5 = CTRL0 = GND = VSS on Parallax BOE USB board
- Pin 6 = CTRL1 = GND = VSS on Parallax BOE USB board
- Pin 7 = SDA Data connected to P0 on Stamp BS2 module and Scope Channel1
- Pin 8 = SCL Clock connected to P1 on Stamp BS2 module and Scope channel2

DS1077 data pin7 & clock pin8 have 4.7K pull-up resistors connected to VCC. I connected these resistors right over the chip to keep the leads as direct as possible. Note the bypass capacitors on VCC pin3 of 0.1uF in parallel with 0.01uF to ground pin4.

The SDA data & SCL clock lines have series protection resistors of 100ohms connected between the Stamp BS2 microcontroller and the DS1077. The series resistors are a precaution against conflicting commands on the bus.

Note when first connecting the Parallax board, keep the board power switch at position 0. When the USB driver has properly loaded and the Stamp Editor detects the unit, then apply power to the board by switching the power switch to position 1.

I2C Waveforms UM10204 and DS1077 Data Sheet

The PBASIC2.5 program ds1077_write.bs2 writes to the DS1077 configuration registers MUX & DIV and the program ds1077_read.bs2 reads them back for confirmation. Both these programs are contained in the zip file ds1077_code.zip (Ref.4) and listed in Appendix D. I used the Basic Stamp Editor 2.5.3 to develop and load these files (Ref. 7). I used many ideas from Jon Williams at Parallax to develop this code (Ref.9).

Figures 5 – 8 show the I2C waveforms that are the basis for the code (Refs. 2 & 8). I just implemented these diagrams in simple bs2 code and later in C code. I am not a programmer, so I am sure there are more elegant ways of doing it. My objective was to understand the I2C bus and use the DS1077. I did not program all the I2C features, just those I needed to fire up the DS1077.

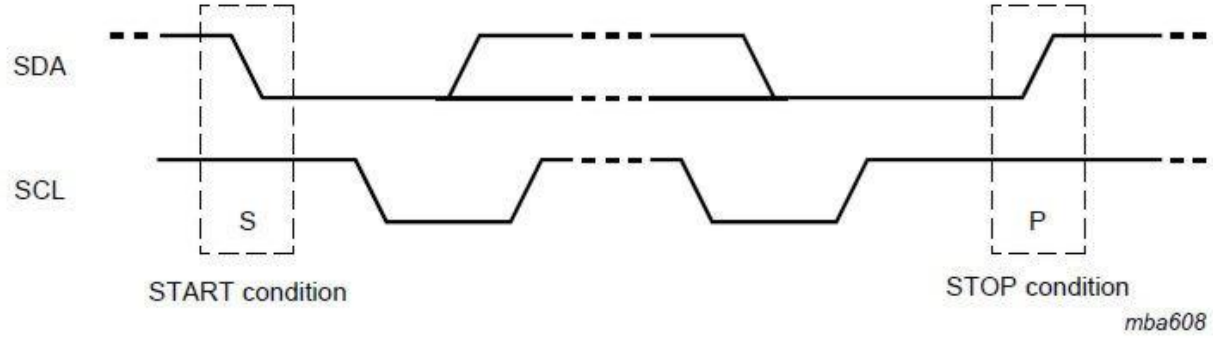


Figure 5 I2C Start and Stop UM10204 Spec

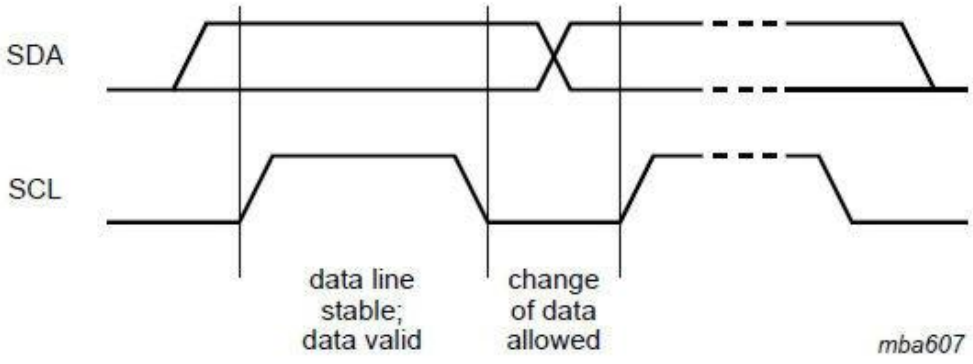


Figure 6 I2C Data Transfer SDA and Clock SCL UM10204 Spec

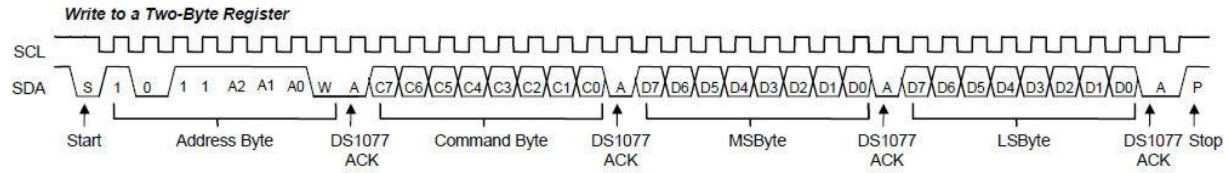


Figure 7 I2C Write Two Bytes to DS1077 Registers MUX and DIV

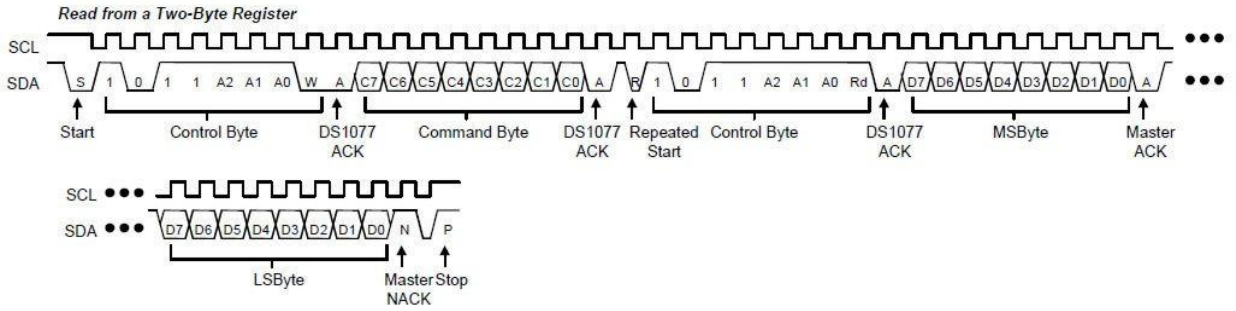


Figure 8 I2C Read Two Bytes from DS1077 Registers MUX and DIV

The BS2 microcontroller is the I2C master, which is responsible for all clocking and control of the bus. The DS1077 is the slave. Figure 5 shows the structure of a Start and Stop. A Start is formed by the SDA going from high to low while the SCL is high. A Stop is formed by the SDA going from low to high while the SCL is high. Figure 6 shows data transfer. SDA data is sampled in the middle of the SCL clock pulse when high.

Stamp Basic Program ds1077_write.bs2

Figure 7 “I2C Write Two Bytes to DS1077 Registers MUX & DIV” shows how we program the DS1077. The complete program structure is given in Figure 18. The BS2 generates a Start pulse and then clocks out (8 clock pulses) the default address of the DS1077, which is $B0 (for write). If this is received correctly, then the DS1077 pulls the SDA lead low generating an ACK=0.

Next the BS2 generates a command $02 to access the MUX register. If this is received correctly, then again the DS1077 pulls the SDA lead low. Then the BS2 generates the MSB of the MUX register, which is ACK’d, then the LSB of the MUX register, which is ACK’d, and finally the Stop pulse. This whole procedure is repeated for the DIV register. The Command Set for addressing registers is given in Ref.2 page13/21. MUX[02] and DIV[01].

According to the I2C standard, the SDA and SCL leads should be inputs on the Master and pulled high by the 4.7Kohm resistors when there is no action on the bus. This allows the slave to then pull the SDA low for an ACK. On the BS2, the SDA/SCL pins alternate between being outputs when sending data/clock and inputs when receiving ACKs and generating Start/Stop pulses.