Project 4: Digital to Analog Conversion using Two-Wire

Jacob Hollenbeck, Boise State University 03/19/2016

Contents

1 Overview

This document demonstrates how digital signals are converted to analog voltages using an embedded processor and a DAC breakout board. An individual voltage level is sent to the DAC using I^2C communication protocol. That signal is then processed using the DAC and output as a voltage. The project requirements state that the atmega324p must output a sine wave at 1 Hz and 2 Hz. This output will be switched at the push of a button. This is done using the following files:

- main.c
- i2c.* - two-wire communication functionality
- systick.* - system counter
- counter.* - counter functionality
- DAC_MCP4725.* - DAC functionality and data packet
- push.* - push button functionality
- sin.h - sine table definition

This project is small relative to the previous ones. It does not use USART, ADC, or PWM. It does require timer and counter functionality though. The frequency of the DAC is controlled by the frequency of the updated index. This is done by using the counter. When the counter period

has elapsed, the index is incremented. Because this is the slowest function on the board, this will control the frequency. By changing this counter period, the frequency can be changed. An abstract view of the project can be viewed in Figure 1.

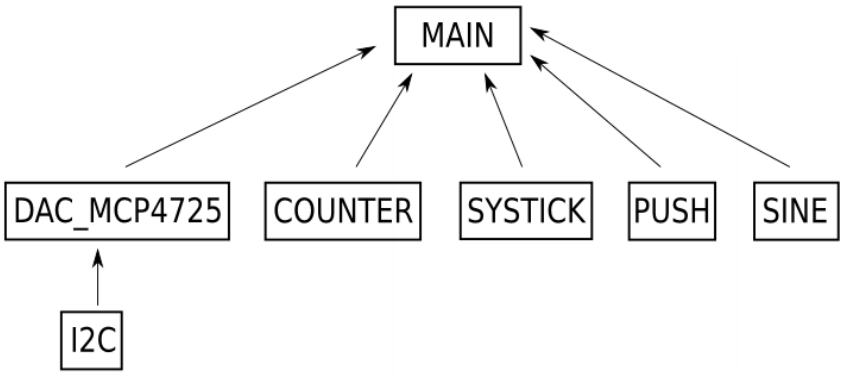

Figure 1: Abstract Overview

2 Supplies

- Atmega324p Microprocessor PDIP
- Four-prong button
- Atmel-ICE programmer with ISP cable
- Breadboard
- JTAG Header
- MCP4275 Breakout board

3 I2C

I2C is a two wire communication protocol. I2C can be used to communicate with many slave devices using the SDA and SCL wires. The SDA wire communicates the device information and data and the SCL wire communicates the clock. To use I2C, particular data packets must be sent everytime. The device address is sent first. Next the configuration command is sent. In this instance, the write command is sent. The write command for this breakout board is 0x40. After the configuration command is sent, the data is sent. For this device, the sparkfun website featured a sine wave table[2]. This table is sent as the data.

To initialize the I2C, I2C_start is configured. This function sends the start condition assigns the device address. After the start condition has been sent, the device waits for the completion of the transmission. Then the device address is sent. Again, the transmission is waited on.

Finally, the TWI status is checked with the STATUS code. Note that the codes used in this were provided from the avr documents located in twi.h. This can be seen in the following code:

```
  // send START code
  TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

  // wait for tx
  while(!(TWCR & (1<<TWINT)))
    ;

  // check status
  status = TW_STATUS & 0xF8;
  if ( (status != TW_START) && (status != TW_REP_START))
    return 1;

  // send device address
  TWDR = address;
  TWCR = (1<<TWINT) | (1<<TWEN);

  // wail for tx
  while(!(TWCR & (1<<TWINT)))
    ;

  // check value of status with MR and MT codes
  status = TW_STATUS & 0xF8;
  if ( (status != TW_MT_SLA_ACK) && (status != TW_MR_SLA_ACK) )
    return 1;

  return 0;
```

After the first part of the data packet has been sent, the actually data can be sent. To send this, the data is written to the TWDR register and transmitted. The data status is then checked.

```
unsigned char i2c_write(unsigned char data)
{
    uint8_t   status;

  // send data
  TWDR = data;
  TWCR = (1<<TWINT) | (1<<TWEN);

  // wait for tx
  while(!(TWCR & (1<<TWINT)))
    ;

  // check value of status with MT DATA
  status = TW_STATUS & 0xF8;
  if( status != TW_MT_DATA_ACK)
    return 1;
  return 0;
}
```

4 Digital to Analog

This project was designed with scalability for future projects. This means that this project was designed to be used with different DACs. This was done by creating a generic device access two-wire protocol. These are the I2C.c and I2C.h files. These files are generic, parameterized communication protocol codes for the atmega324p chip. The next level of code is for the DAC breakout board. Specifically the MCP4725 breakout board. These are titled DAC_MCP4725.c and DAC_MCP4725.h. These files were designed to use the I2C functions but with specific headers and definitions. This allows for the breakout board to be changed without major code rewrite. This will allow for different bit resolutions to be designated as well.

A state structure is first defined. This state structure contains the entire data packet as well as other potentially wanted information. This is defined as follows:

```
typedef struct {
  uint16_t DAC_code;
  uint16_t total_packets_sent;
  uint16_t error_count;
  uint16_t time_to_send;
  uint16_t data;
} dac_data_packet;
```

This data packet is initialized with everything set to 0. A request to send data function is used next. This function waits for the errors to be cleared before sending data. This is done as follows:

```
unsigned char request_to_send_data(dac_data_packet * pc)
{
  unsigned char error = 0;
  error = i2c_start(DEV_ADDR);       // DEV_ADDR = 0xC0
  if(error)
  {
    pc->error_count++;               // Increment error count
    i2c_stop();                      // Strop sending data
    return 0;
  }
  return 1;
}
```

If everything has gone well up to this point, the send dac data function is used. The send dac data function writes DAC instruction following by the 12 bits of data. This is done as follows:

```
void send_dac_data(dac_data_packet * pc,  uint16_t code, uint16_t data)
{
  unsigned char err0, err1, err2;

  pc->total_packets_sent++;
  pc->DAC_code = code;               // Code should be 0x40 to write
  pc->data = data;                   // Data from sine table

  // Set all errors to 0
  err0 = 0;
  err1 = 0;
```

```
  err2 = 0;

  // Write information to DAC
  err0 = i2c_write(pc->DAC_code);
  err1 = i2c_write(pc->data >> OFFSET);              // Shift 4 LSB bits off
  err2 = i2c_write((pc->data & 0xF) << OFFSET);     // Grab only 4 LSB bits

  // Check for errors
  if(err0 || err1 || err2)
  {
    pc->error_count++;
  }
}
```

This is called in the main file as follows:

```
  if(request_to_send_data(&my_dac))
  {
    // send data if no error
    send_dac_data(&my_dac, WRITE_ADDR, sin_wave[sin_index]);
  }
```

The DAC board used can be found at sparkfun.com. The full reference can be found at [1]. The hardware configuration for the DAC is simple as well. The respective SDA and SCL lines on the breakout board should be wired to the lines on the MCU named the same. The out pin displays the output of the DAC. The Vcc and GND pins must be connected as well.

5 Push Button

The configuration of the push button has been discussed in project 2. The actual functionality will be elaborated on here. The push button ISR toggles a bit. This is done with getters and setters located inside of the push button code. The value is set with the toggle function used inside of the ISR. This value is continually checked. The output is used to determine if the frequency of the updated index will be 1 Hz or 2 Hz. The flag switch is shown in the following code:

```
      // Set frequency depending on push input
      if(switch_flag){
        if(counter_elapsed(&_2Hz_C)){
          sin_index = (sin_index + 1) & 0x1FF;    // rollover index
        }
      } else {
        if(counter_elapsed(&_1Hz_C)){
          sin_index = (sin_index + 1) & 0x1FF;    // rollover index
        }
      }
```

6 Compilation

Because multiple files are used in this program, a Makefile is used to compile the .elf file. The linker process was covered in the previous report. To use the Makefile included, change to the project directory and use the following commands:

```
$ make
$ make write
```

The make write command will write the compiled .elf file to the MCU.

References

[1] *DAC Breakout Board*. URL: https://www.sparkfun.com/products/12918.

[2] *DAC Breakout Board Hookup guide*. URL: https://learn.sparkfun.com/tutorials/mcp4725-digital-to-analog-converter-hookup-guide?_ga=1.117540809.1725391151.1448739723.