[Ricardo Stefano Reyna]
[16/June/2016]
[21010521]

Function Generator Serial D2A Micro-Controller Applications

Introduction

This module introduces the SPI interface; this allows communication between a master device which sends a clock signal towards the slave. In order to achieve this purpose we used the DAC to create for different type of waves, change the amplitude and frequency.

Design

The setup of the microcontroller with the DAC is similar to the one reference in the module document except instead of using ports 2.0, 2.1, and 2.2 I used 3.6 for the enable, 3.4 to send the data, and 3.0 for the clock. The initializations for the SPI and part of the writing I used the TI example codes they have. The way I initialized the SPI is using the 8-bit mode. For writing the SPI I shifted twice to the right to get rid of the don't cares, later I grabbed the top 4 bits for the DAC command, and then sent the last 8 bits worth of data to display the wave on the o-scope.. Another different connection I made was to use 5V for the Vref instead of 3.3V this will allow me to change the amplitude using a potentiometer to either let the complete voltage through or lower it significantly. After wiring everything together I proceeded to create the look up tables of each wave. The triangle and sine waves I just used a website to create an array of size 50 with a range of 0 to 1023 (because of the 10 bits). The sawtooth is just the twice the triangle length and bringing it to 0 at the half point of each peak, and the square wave is the first half 1023, and second half 0. I included some functions to allow me toggle between different waves. Next thing I added was using the adc to change the frequency, most of the code was using the previous module except some tweaks of trial and error to get the bounds of 10Hz to 100Hz. After I get the value of the adc I plug it into a delay function as the limit from 0 to said value. Last thing I added were the switches which I put them in pins 2.1 and 2.2, I used the push buttons (which are low true) after looking up the documentation and doing a test setup using the multimeter and a power supply to further understand how it worked.

Conclusion

I really didn't have trouble with this lab other than setting up the switches at first which I solved by connecting 1k resistors from VCC to the switch. The module allowed me to practice a new concept that we never had experience in. Adding a delays to the SPI and ADC allowed for a better synchronization.

Appendix Code:

```
#include <msp430.h>
#include <stdint.h>
#define SINE 0
#define SQUARE 1
#define TRAINGLE 2
#define SAWTOOTH 3
//LOOK UP TABLE
int sine[50] =
{512,576,639,700,758,812,862,906,943,974,998,1014,1022,1022,1014,998,974,943,906,862,81
2,758,700,639,576,512,447,384,323,265,211,161,117,80,49,25,9,1,1,9,25,49,80,117,161,211,26
5,323,384,447};
int square[50] =
{1023,1023,1023,1023,1023,1023,1023,1023,1023,1023,1023,1023,1023,1023,1023,1023,1023
,1023,1023,1023,1023,1023,1023,1023,1023,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
int sawtooth[50] =
{0,20,41,62,83,104,125,146,167,187,208,229,250,271,292,313,334,354,375,396,417,438,459,48
0,501,521,542,563,584,605,626,647,668,688,709,730,751,772,793,814,835,855,876,897,918,93
9,960,981,1002,1023};
int traingle[50] =
{41,82,123,164,205,246,286,327,368,409,450,491,532,573,614,655,696,737,777,818,859,900,9
41,982,1023,982,941,900,859,818,777,737,696,655,614,573,532,491,450,409,368,327,286,246,
205,164,123,82,41,0};
/*
* main.c
*/
void adc_setup(void);
int adc_value(void);
void SPI_setup(void);
void SPI_write(unsigned int data);
int choose_wave(void);
void output_waveform(int freq_value, int arr[]);
void delay1(void);
void apply_freq(int val);
void toggle_waves(int freq);
```

```
int main(void) {
WDTCTL = WDTPW | WDTHOLD; // Stop watchdog timer
P2DIR = 0x00;
adc_setup();
SPI_setup();
int count = 0;                              r
volatile int freq = adc_value();            e
while(1) {                                  t
if (count == 24) {                          u
freq = adc_value();                         r
count = 0;                                  n
}                                           0
count++;                                    ;
toggle_waves(freq);                         }
}
// ADC10 interrupt service routine
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void) {
__bic_SR_register_on_exit(CPUOFF); // Clear CPUOFF bit from 0(SR)
}
void adc_setup(void) {
// ADC
ADC10CTL0 = ADC10SHT_2 + ADC10ON + ADC10IE; //0x1018, 16xADC10CLks, ADC10 on, and
ADC10 interupt enable
ADC10AE0 |= 0x01; //Enable reg 0, Pin2.0
}
int adc_value(void) {
volatile unsigned int value;
ADC10CTL0 |= ENC + ADC10SC; // conversion start
__bis_SR_register(CPUOFF + GIE); // enter the LPM
value = ADC10MEM; // save the value
if (value < 75) {
return 1;
}
else if (value > 1010) {
return 190;
}
else {
return (value / 4) -17; //ADC to delay
}
}
```

```
void SPI_setup(void) {
P3OUT = 0x40; // Set slave reset
P3DIR |= 0x40; //
P3SEL |= 0x31; // P3.0,4,5 USCI_A0 option select
UCA0CTL0 |= UCCKPL + UCMSB + UCMST + UCSYNC; // 3-pin, 8-bit SPI master
UCA0CTL1 |= UCSSEL_3; // SMCLK
UCA0BR0 |= 0x02; // /2
UCA0BR1 = 0; //
UCA0MCTL = 0; // No modulation
UCA0CTL1 &= ~UCSWRST; // **Initialize USCI state machine**
P3OUT &= ~0x40; // Now with SPI signals initialized,
P3OUT |= 0x40; // reset slave
}
int choose_wave(void) {
int type;
int P2_input = P2IN & 0x06; //bitmask pin 1 and 2
if (P2_input == 0)
type = SAWTOOTH;
else if (P2_input == 2)
type = SQUARE;
else if (P2_input == 4)
type = TRAINGLE;
else
type = SINE;
return type;
}
```

```
void toggle_waves(int freq) {
switch(choose_wave()) {
case SINE:
output_waveform(freq,sine);
break;
case SQUARE:
output_waveform(freq,square);
break;
case TRAINGLE:
output_waveform(freq,traingle);
break;
case SAWTOOTH:
output_waveform(freq,sawtooth);
break;
}
}
void SPI_write( unsigned int data) {
uint8_t byte2Transmit = data & 0x003F;
byte2Transmit <<= 2;
data >>= 6 ;
uint8_t byte1Transmit = data & 0x000F;
byte1Transmit |= 0xF0;
P3OUT &= 0b10111111;
delay();
UCA0TXBUF = byte1Transmit; //Byte to SPI TXBUF
while(!(IFG2 & UCA0TXIFG)); //USCI_A0 TX buffer ready?
UCA0TXBUF = byte2Transmit;
while((UCB0STAT & BIT0));
delay();
P3OUT += 0x40;
}
void output_waveform(int freq_value, int arr[]) {
int i;
for (i = 0; i < 50; i++) {
SPI_write(arr[i]);
apply_freq(freq_value);
}
}
void delay(void) {
volatile unsigned int i;
for(i = 0; i < 2; i++);
}
void apply_freq(int val) {
volatile unsigned int i;
for (i = 0; i < val; i++);
}
s
```