Manual de Utilização de Bibliotecas para comunicação ADXL 345 com MSP430G2553 com interface SPI

Carlos Alberto Pereira
Ícaro Nascimento Queiroz
Rodrigo Travassos Sapucaia

Salvador, 2016

Introdução

A biblioteca criada estabelece uma comunicação serial do acelerômetro **ADXL 345** da Adafruit com o micro controlador **MSP430G2553**. As funções se dividem em três propósitos, os quais são estabelecer a comunicação por meio de protocolo SPI entre os dispositivos, configurar a leitura e escrita dos registradores por SPI e obtenção dos valores de aceleração nos três eixos: x, y e z, sendo composta por dois arquivos **`adxl345.c`** e **`adxl345.h`.**

1. *Defines*

Os defines da biblioteca são utilizados para definir os endereços dos registradores do acelerômetro ADXL345.

1. 1. `#define DATAX0 0x32`
Define o endereço do registrador referente ao primeiro byte (lsByte) da leitura do eixo x

1. 2. `#define DATAX1 0x33`
Define o endereço do registrador referente ao segundo byte (msByte) da leitura do eixo x

1. 3. `#define DATAY0 0x34`
Define o endereço do registrador referente ao primeiro byte (lsByte) da leitura do eixo y

1. 4. `#define DATAY1 0x35`
Define o endereço do registrador referente ao segundo byte (msByte) da leitura do eixo y

1. 5. `#define DATAZ0 0x36`
Define o endereço do registrador referente ao primeiro byte (lsByte) da leitura do eixo z

1. 6. `#define DATAZ1 0x37`
Define o endereço do registrador referente ao segundo byte (msByte) da leitura do eixo z

2. Configuração da comunicação SPI entre dispositivos

2. 1. ***start_adxl345***
Esta função viabiliza a comunicação entre o acelerômetro ADXL345 e o MSP430G2553 por meio do protocolo SPI

2. 1. 1. Estrutura

```
void start_adxl345(void)
   //Configure the MSP430 to use SPI interface
{
   P1DIR |= BIT5;
   P1OUT |= BIT5;
   P1SEL = BIT1 | BIT2 | BIT4;
   P1SEL2 = BIT1 | BIT2 | BIT4;
   UCA0CTL1 = UCSWRST;
   UCA0CTL0 |= UCCKPH + UCMSB + UCMST + UCSYNC;
   UCA0CTL1 |= UCSSEL_2;
   UCA0CTL1 &= ~UCSWRST;
}
```

2. 1. 2. Argumentos

A função não possui argumentos configuráveis, apenas viabiliza as conexões entre os dispositivos

2. 1. 3. Valor de Retorno

Esta função não retorna nenhum valor

3. Configuração das funções de leituras dos eixos pelo ADXL345

3. 1. read_x

A função read_x é responsável por realizar a leitura da informação referente à aceleração no eixo x.

3. 1. 1. Estrutura

```
int read_x(void)
{
    int msByte,msByte16,lsByte,measure;

    lsByte = read_spi(DATAX0);
    msByte = read_spi(DATAX1);
    msByte16 = (msByte << 8);
    measure = msByte16 | lsByte;
    return measure;
}
```

3. 1. 2. Argumentos

`DATAX0` – Parte da leitura da aceleração no eixo x no tamanho de 8 bits

`DATAX1` – Parte da leitura da aceleração no eixo x no tamanho de 8 bits

`msByte` – Byte mais significativo da informação

`msByte16` – Recebe o valor de 8 bytes deslocado de 8 bits em uma variável de tamanho de 16 bytes

`lsByte` – Byte menos significativo da informação

`measure` – Informação completa no tamanho de 16 bytes

3. 1. 3. Valor de Retorno

Retorna um número de 16 bits com o valor da aceleração no eixo x.

3. 2. read_y

A função read_y é responsável por realizar a leitura da informação referente à aceleração no eixo y.

3. 2. 1. Estrutura

```
int read_y(void)
{
    int msByte,msByte16,lsByte,measure;
    lsByte = read_spi(DATAY0);
    msByte = read_spi(DATAY1);
    msByte16 = (msByte << 8);
    measure = msByte16 | lsByte;
    return measure;
}
```

3. 2. 2. Argumentos

`DATAY0` – Parte da leitura da aceleração no eixo y no tamanho de 8 bits
`DATAY1` – Parte da leitura da aceleração no eixo y no tamanho de 8 bits
`msByte` – Byte mais significativo da informação
`msByte16` – Recebe o valor de 8 bytes deslocado de 8 bits em uma variável de tamanho de 16 bytes
`lsByte` – Byte menos significativo da informação
`measure` – Informação completa no tamanho de 16 bytes

3. 2. 3. Valor de Retorno

Retorna um número de 16 bits com o valor da aceleração no eixo y.

3. 3. read_z

A função read_z é responsável por realizar a leitura da informação referente à aceleração no eixo z.

3. 3. 1. Estrutura

```
int read_z(void)
{
    int msByte, msByte16, lsByte, measure;
    lsByte = read_spi(DATAZ0);
    msByte = read_spi(DATAZ1);
    msByte16 = (msByte << 8);
    measure = msByte16 | lsByte;
    return measure;
}
```

3. 3. 2. Argumentos

`DATAZ0` – Parte da leitura da aceleração no eixo z no tamanho de 8 bits
`DATAZ1` – Parte da leitura da aceleração no eixo z no tamanho de 8 bits
`msByte` – Byte mais significativo da informação
`msByte16` – Recebe o valor de 8 bytes deslocado de 8 bits em uma variável de tamanho de 16 bytes
`lsByte` – Byte menos significativo da informação
`measure` – Informação completa no tamanho de 16 bytes

3. 3. 3. Valor de Retorno

Retorna um número de 16 bits com o valor da aceleração no eixo z.

4. Configuração de escrita e leitura SPI

4. 1. read_spi

Esta função permite a recepção de dados via protocolo SPI. Ela é utilizada também para configurar diversos parâmetros do ADXL345.

4. 1. 1. Estrutura

```
int read_spi(int addr)
{
    int info;
    P1OUT &= (~BIT5);
    while (!(IFG2 & UCA0TXIFG));
    UCA0TXBUF = addr;
    while (!(IFG2 & UCA0RXIFG));
    info = UCA0RXBUF;
    P1OUT |= (BIT5);
    return info;
}
```

4. 1. 2. Argumentos

`addr` – Endereço do registrador a ser lido.

`info` – Informação passada pelo registrador lido do ADXL345

4. 1. 3. Valor de retorno

Informação contida no registrador lido.

4. 2. write_spi

Essa função é transmitida duas vezes, a primeira para informar qual o endereço do registrador onde será escrita a informação e a segunda vez para informar o dado que será escrito.

4. 2. 1. Estrutura

```
void write_spi(int addr, int info)
{
    P1OUT &= (~BIT5);
    while (!(IFG2 & UCA0TXIFG));
    UCA0TXBUF = addr;
    while (!(IFG2 & UCA0TXIFG));
    UCA0TXBUF = info;
    P1OUT |= (BIT5);
}
```

4. 2. 2. Argumento

`addr` – Endereço do registrador a ser lido.

`info` – Informação passada pelo registrador lido do ADXL345

4. 2. 3. Valor de retorno

Esta função não retorna nenhum parâmetro por ser uma função de escrita.