Bateria virtual controlada pela Launchpad MSP430G2553

Rodrigo Magalhães Caires
FGA
Universidade de Brasília
Gama, Brasil
rodrigo_macrt@hotmail.com

Palavras-chave —Microcontrolador, MSP430, Percussão, Bluetooth, I2C, UART.

I. Resumo

Sistema de percurssão virtual controlada pela Launchpad MSP430G2553 e estimulada a partir de sensores de aceleração e giro presos ao braço ou baqueta do usuário.

II. Introdução

O conceito deste projeto é simular a experiência de tocar bateria com apenas um ou dois sensores, um microcontrolador, um bluetooth e um computador ou smartphone, ou seja, sem utilizar os componentes físicos como tambores, caixas e pratos. As vantagens deste produto estão na redução de custo, tendo em vista que um bom kit de bateria custa milhares de reais enquanto esta alternativa reduz o gasto para aproximadamente 100 reais, na flexibilidade, já que o protótipo é portátil e, portanto, de fácil montagem e deslocamento, e na comodidade, sendo capaz de reter todo som e barulho a um fone de ouvido.

III. Desenvolvimento

O prótotipo da bateria virtual utiliza um microcontrolador MSP430G2553 cujas funções principais são executar a comunicação intermediaria entre os periféricos e fazer o processamento dos dados, um sensor acelerômetro e giroscópio MPU6050, utilizado para determinar o sentido do deslocamento e fazer uma estimativa da posição relativa da "baqueta", e um módulo bluetooth HC-05, responsável pela transmissão dos dados para um computador ou smartphone que, por fim, fará a emulação do som.

O funcionamento deste projeto se dá na captação dos movimentos dos braços ou baquetas do usuário. A captação dos dados de aceleração e giro serão executadas pelo sensor e enviadas para o microcontrolador que, por sua vez, fará a interpretação e processamento dos dados, retornando informações tangíveis como qual peça da bateria foi tocada. Essas informações serão enviadas via bluetooth para um aplicativo ou estação de aúdio com sampler de bateria, onde será reproduzido o som correspondente.

O projeto de software apresentou vários pontos problemáticos durante o desenvolvimento sendo os mais importantes a precisão dos dados durante a coleta, a propagação de erro durante as etapas de integração discreta para transformar aceleração em deslocamento e a consideração da aceleração gravitacional e a sua influência em cada eixo de acordo com o giro do sensor. As soluções utilizadas a princípio são rudimentares e para fins de teste, mas soluções mais sofisticadas foram encontradas e deverão ser aplicadas assim que as

adaptações estiverem de acordo com as necessidades do projeto.

IV. Descrição de Hardware

O protótipo é composto por uma Launchpad MSP430G2553, um módulo bluetooth HC-05, um sensor acelerômetro e giroscópio MPU6050, dois resistores de 4,7 KΩ e cabos de conexão variados.

Após as devidas configurações, os pinos P1.1 e P1.2 da Launchpad podem ser utilizados como RX (Receptor) e TX (Transmissor) para a comunicação com o módulo bluetooth enquanto os pinos P1.6 e P1.7 podem ser usados como SCL (Clock serial) e SDA (Dado serial) para a comunicação com a MPU6050.

A alimentação dos periféricos é compartilhada e realizada pelos pinos VCC e GND da Launchpad. Para que a comunicação entre a Launchpad e o bluetooth funcione, a conexão dos pinos de tranmissão e recepção devem ser cruzadas, ou seja, o RX (P1.1) e TX (P1.2) da MSP430 devem ser conectados ao TX (TXD) e RX (RXD) do HC-05 respectivamente. Já para a MPU6050, a conexão de pinos é direta, ligando o pino P1.6 (SCL) ao SCL do sensor e o pino P1.7 (SDA) ao DAS do sensor.

Porém, para que ambas as comunicações funcionem corretamente, devem ser feitas algumas alterações adicionais. Para a comunicação UART, realizada entre o bluetooth e a MSP430, deve-se alterar a configuração dos jumpers TXD e RXD da vertical para a horizontal de acordo com a Figura 1 (Anexo I). Já para a comunicação I2C, realizada entre o sensor e a MSP430, deve-se remover o jumper P1.6 conectado ao LED2, vide Figura 2 (Anexo I), pois o mesmo pode afetar o funcionamento do SCL (também conectado ao pino P1.6). Além disso, devem ser adicionados dois resistores externos de pullup, um entre os SCLs e um entre os SDAs. Isso deve ser feito para regularizar o sinal que é passado entre os dois MOSFETs presentes nos terminais dos pinos I2C. A montagem completa do hardware pode ser conferida no diagrama da Figura 3 (Anexo I).

V. Descrição de Software

O programa utiliza bibliotecas especiais para facilitar a comunicação nos protocolos UART (bluetooth), I2C e para a obtenção de dados do sensor MPU6050. Faz-se também o uso das variáveis de controle ACCEL_XOUT_ANT, ACCEL_YOUT_ANT, ACCEL_ZOUT_ANT e control, sendo as três primeiras do tipo signed int e utilizadas para armazenar valores anteriores de aceleração nos eixos X, Y e Z respectivamente enquanto control é uma variável auxiliar do tipo int utilizada para controlar o comportamento de um trecho do código durante um laço de verificação.

As primeiras definições da main são correspondentes ao desligamento do Watchdog Timer e a seleção de um clock de 16 MHz. Em seguida, é chamada a função de inicialização do protocolo I2C, passando como parâmetro o endereço do sensor MPU6050 (0x68). Dentro da função de inicialização, estabelece-se a funcionalidade dos pinos P1.6 e P1.7 como SCL (Serial Clock) e SDA (Serial Data), respectivamente, através dos registradores P1SEL. Logo após, define-se um clock de 100 KHz para sincronizar a comunicação entre o mestre e o escravo e armazena-se o endereço do sensor no registrador UCB0I2CSA.

Após a inicialização do I2C, realiza-se a configuração do sensor MPU6050 por meio de outra função, definindo os endereços de acesso aos registradores responsáveis pelo armazenamento dos dados de aceleração e giro nos 3 eixos. Depois de configurar o sensor, é feita a inicialização do protocolo UART para comunicação via bluetooth. Primeiramente, define-se as funcionalidades RXD (Receiver) e TXD (Transmitter) para os pinos P1.1 e P1.2 respectivamente, novamente por meio dos registradores P1SEL. Em seguida, configura-se o clock e a baud rate da comunicação. Para terminar a inicialização do programa, atribue-se os valores iniciais para variáveis de controle

O laço while é responsável por fazer a leitura, processamento e transmissão de dados e, a princípio, funciona indefinidamente. Primeiramente são feitas as leituras do acelerômetro pela função *Get_Accel_Values()*. Porém, durante as primeiras leituras, os dados do acelerômetro tendem a valores muito altos que tendem a diminuir e se estabilizar em leituras subsequentes. Levando isso em conta, o laço interno *while(control == 0)* faz uma filtragem inicial desses valores de pico e, enquanto os dados do acelerômetro não atingirem uma faixa de valores predefinida, a leitura continuará sendo feita. Quando as condições são cumpridas, atribue-se valores iniciais para as variáveis ACCEL_XOUT_ANT, ACCEL_YOUT_ANT e ACCEL_ZOUT_ANT, responsáveis pelo armazenamento das acelerações nos 3 eixos de referência e altera-se o valor de control de 0 para 1, impedindo que o programa volte a entrar neste laço.

Após a filtragem inicial dos valores de pico, utiliza-se outros três filtros rudimentares para isolar os ruídos e trepidações nos dados de aceleração dos três eixos. A função desse filtros é comparar a variação entre os valores de aceleração atuais e passados. Se a diferença for ínfima, despreza-se a variação, igualando o valor atual ao valor anterior. Após o segundo estágio de filtragem, os dados são enviados para o dispositivo destino via bluetooth, informando o eixo da coleta seguido do valor coletado. Por fim, os valores anteriores são igualados aos valores atuais para que possam ser usados no próximo ciclo.

VI. Resultados

Os resultados obtidos são os valores de aceleração para os 3 eixos do sensor e podem ser verificados a partir do software de emulação de terminal PuTTY ao se conectar o módulo HC-05 a um computador e acessar a porta COM correspondente ao receiver do computador. Por enquanto, os problemas de variação da aceleração gravitacional observada por cada eixo de acordo com o giro do sensor não foram resolvidos, portanto, os dados são entregues de forma praticamente crua e passam por um processamento grosseiro quando o eixo X do sensor é paralelo à aceleração gravitacional. Esse processamento permite observar os valores das acelerações axiais como valores próximos de 0 quando o sensor se encontra parado, visto que as condições mencionadas anteriormente sejam atendidas, e, devido a suas limitações, só será utilizado para testes. A saída do terminal pode ser observada na Figura 4 (Anexo I). O código pode ser encontrado no Anexo II.

VII. Conclusão

Apesar das dificuldades apresentadas, o projeto é definitivamente possível de ser feito. As partes de montagem de hardware e da comunicação da Launchpad com os periféricos estão prontas e o que resta ser resolvido são problemas puramente matemáticos como estimar a posição através da integração dupla dos dados de aceleração e extrair a influência da aceleração gravitacional exercida no sensor por meio de calculos com a matriz de rotação.

VIII. Referências

[1] Texas Instruments, ChronosDrums. Disponível em: http://processors.wiki.ti.com/index.php/ChronosDrums

[2] Xanthium. Serial Communication Using MSP430 UART(USCI-A). Disponível em: http://xanthium.in/Serial-Communication-MSP430-UART-USCI_A

[3] SLAU144 USER GUIDE.

[4] MPU6050 DATASHEET.

[5] HC-05 DATASHEET.

[6] CHRobotics, Using accelerometers to estimate position and velocity. Disponível em: http://www.chrobotics.com/library/accel-position-velocity.

ANEXO I - Imagens

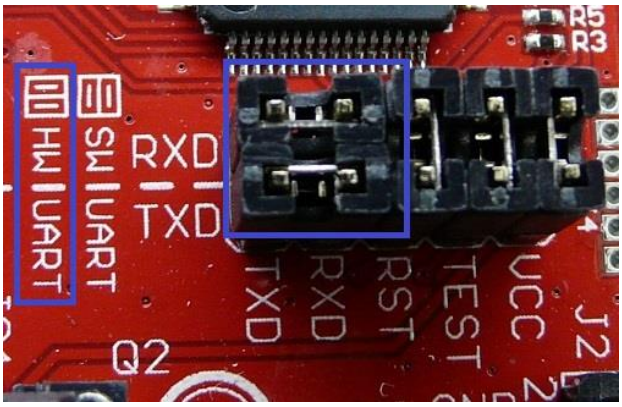

Figura 1. Jumpers UART.

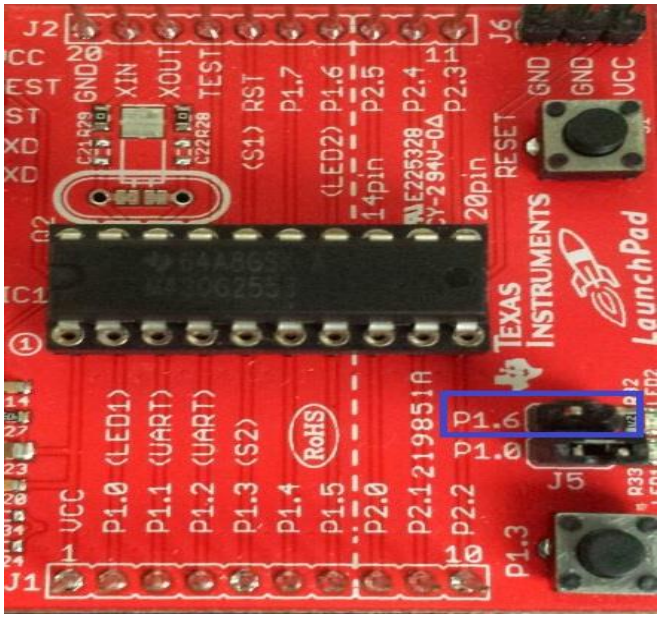

Figura 2. Jumper P1.6.

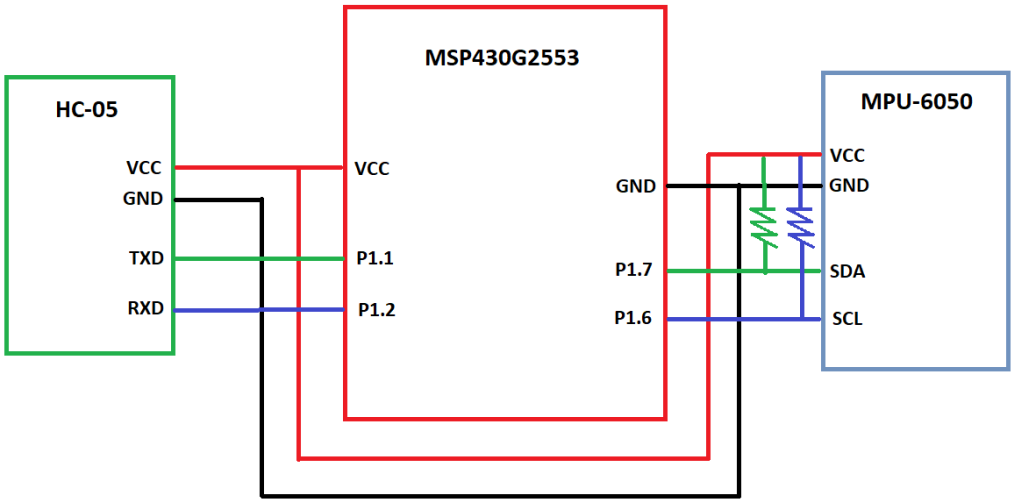

Figura 3. Montagem do hardware.

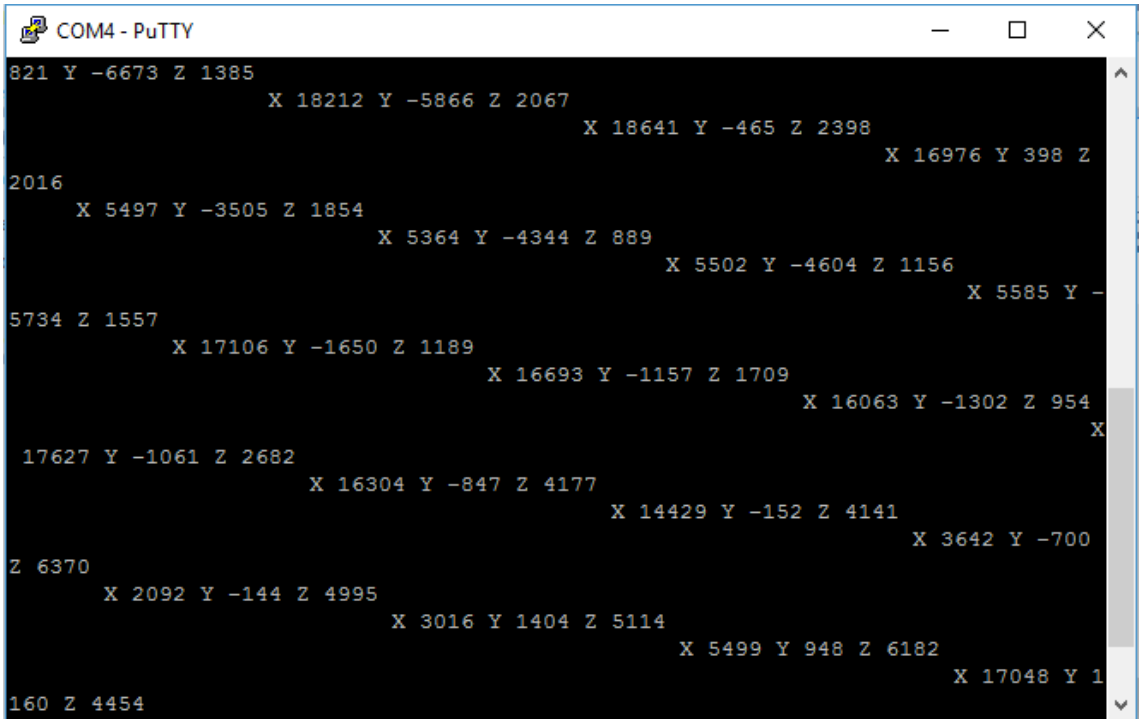

Figura 4. Dados das acelerações axiais.

ANEXO II – Main.c

```
#define USE_I2C_USCI

#define USE_MPU6050

#define USE_UART
```

```
#include <math.h>
#include <stdlib.h>
#include "msp430g2553.h"
#include "Global.h"
#include "UART.h"
#include "I2C_USCI.h"
#include "MPU6050.h"

signed int ACCEL_XOUT_ANT, ACCEL_YOUT_ANT, ACCEL_ZOUT_ANT;
int control;

void main()
{
        WDTCTL = WDTPW | WDTHOLD;
        BCSCTL1 = CALBC1_16MHZ;
        DCOCTL = CALDCO_16MHZ;
        __delay_cycles(100000);
        P1DIR = BIT0;

        Init_I2C_USCI(MPU6050_ADDRESS); //1.7 SDA, 1.6 SCL
        Setup_MPU6050();
        __delay_cycles(320000);

    UART_Init();

    ACCEL_XOUT_ANT = 16900;
    ACCEL_YOUT_ANT = -570;
    ACCEL_ZOUT_ANT = 700;
    control = 0;
```

```
        while(1)
    {
        Get_Accel_Values();
        __delay_cycles(1000000);

        while(control == 0){

          if(abs(ACCEL_XOUT-16900) < 300 && abs(ACCEL_YOUT+570) < 300 &&
abs(ACCEL_ZOUT-700) < 300){
            control = 1;
            ACCEL_XOUT = 16900;
            ACCEL_YOUT = -570;
            ACCEL_ZOUT = 700;
          }

          else{
            Get_Accel_Values();
          }

          __delay_cycles(1000000);

        }

        if(abs(abs(ACCEL_XOUT) - abs(ACCEL_XOUT_ANT)) < 300){
          ACCEL_XOUT = ACCEL_XOUT_ANT;
        }

        if(abs(abs(ACCEL_YOUT) - abs(ACCEL_YOUT_ANT)) < 300){
          ACCEL_YOUT = ACCEL_YOUT_ANT;
        }
```

```
        if(abs(abs(ACCEL_ZOUT) - abs(ACCEL_ZOUT_ANT)) < 300){
          ACCEL_ZOUT = ACCEL_ZOUT_ANT;
        }

        UART_Write_Char('X');
        UART_Write_Char(' ');
        UART_Write_Float((float)(ACCEL_XOUT-16900),0);
        UART_Write_Char(' ');

        UART_Write_Char('Y');
        UART_Write_Char(' ');
        UART_Write_Float((float)(ACCEL_YOUT+570),0);
        UART_Write_Char(' ');

        UART_Write_Char('Z');
        UART_Write_Char(' ');
        UART_Write_Float((float)(ACCEL_ZOUT-700),0);
        UART_Write_Char(' ');

                UART_Write_Char('\n');

         ACCEL_XOUT_ANT = ACCEL_XOUT;
         ACCEL_YOUT_ANT = ACCEL_YOUT;
         ACCEL_ZOUT_ANT = ACCEL_ZOUT;

        P1OUT ^= BIT0;

        __delay_cycles(1000000);
    }

}
```