Air Drum Controlada Pelo Microcontrolador MSP430G2553

Filipe de Souza Freitas
FGA
Universidade de Brasília
Gama, Brasil
filipe.desouzafreitas@gmail.com

***Resumo*—Através da utilização de um sensor acelerômetro, microcontrolador msp430 implementar um sistema de posicionamento que reconheça a posição para um dado referencial e através de determinada posição possa executar um som referente a um instrumento de percussão pertencente a uma bateria(instrumento musical).**

Palavras-chave—Microcontrolador,MSP430G2553, Acelerômetro, Percussão,Bateria,I2C,UART.

I. Introdução

O uso de baterias e demais instrumentos de percussão é bastante popular , entretanto, o uso de tais instrumentos podem causar desconforto a terceiros e possuem preço inacessível a muitas pessoas, o que pode desencorajar como primeiro instrumento a ser aprendido.

II. Desenvolvimento

Através do uso de um sensor acelerômetro inercial pretende-se captar aceleração instantânea em um sistema de referência cartesiano em 3 dimensões representadas por coordenadas x,y e z e integrá-lo duas vezes para obter a posição instantânea.

O instrumento em foco neste trabalho é a bateria, através do posicionamento das baquetas pretende-se definir se houve contato entre uma das peças da bateria e a baqueta definindo a posição da bateria em função da posição inicial da baqueta, para isso a baqueta será modelada como uma partícula, onde obtém-se de um sensor acelerômetro a aceleração vetorial instantânea da partícula.

A aceleração é a taxa de variação em função do tempo da velocidade , que por sua vez é a taxa de variação da posição em função do tempo, se pegarmos o tempo t e fizermos tender a zero obtemos para os dois casos, aceleração e velocidade instantâneas.[2][4]

“A integral de uma função nada mais é do que a área formada abaixo do seu gráfico e que esta área pode ser calculada dividindo-a em n retângulos cuja largura tende a zero e em seguida somando-se a área de cada um desses retângulos, conforme ilustra a figura abaixo.”[1].

Assim pode-se mostrar o sinal em intervalos fixos e calcular a área formada através das amostras coletadas[1][2].

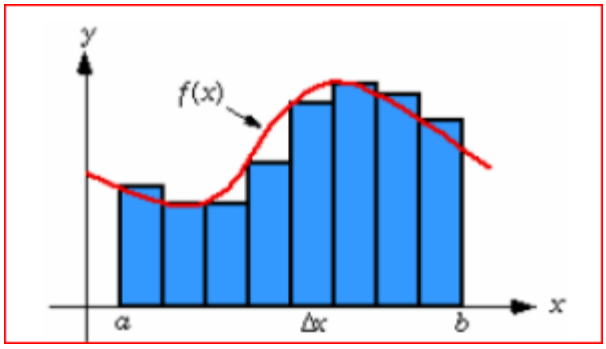

Figura 1 - Área abaixo da curva.

A área total de cada divisão pode ser obtida pela soma da Área 1 com a Área 2:

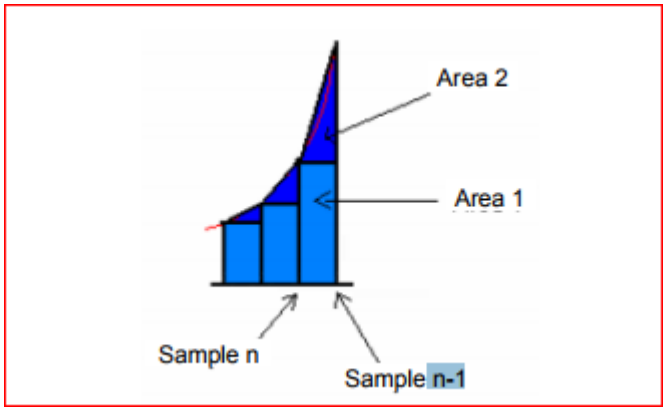

Figura 2 - Área abaixo da curva.

A área 1 pode ser aproximada por um retangulo e a área dois por um triângulo logo:

$$Área_n = Amostra_n + [(Amostra_n - Amostra_{n-1}) * T]/2 \quad (1)$$

Onde :

- T é o tempo proporcional a taxa de amostragem;
- $Área_n$ é a integral obtida;
- $Amostra_n$ é a amostra obtida do sensor;
- $Amostra_{n-1}$ é a amostra anterior obtida do sensor;

Assim , obtém-se uma aproximação aplicando-se duas vezes o procedimento acima, uma vez para velocidade e outra para posição.

De acordo com o data sheet da InvenSense, fabricante da placa com o dispositivo acelerômetro usado no projeto, para aplicações com algoritmos de processamento de movimento a taxa de amostragem deve ser superior a 200 Hz para prover um resultado com pouca latência.[6]

A taxa de amostragem da posição ao dispositivo externo será de 9600 Hz, ou seja, uma informação se alguma peça da bateria foi tocada pela baqueta será transmitida a cada 104,17 us.

Descrição do hardware

A tabela 1 apresenta os materiais necessários:

Numero	Descrição	Quantidade
1	MSP-EXP430G2 LaunchPad Development Kit	1
2	MPU-6050	1
3	01 - Barramento de Pinos 90 Graus(8 pinos)	1
4	Resistores de 10K Ohms	2
5	Jumpers macho-femea	4
6	Protoboard	1

Abaixo encontra-se um esquema de ligação entre as partes a protoboard não é necessária podendo ser substituída por uma PCB desde que sejam feitas as mesmas conexões.

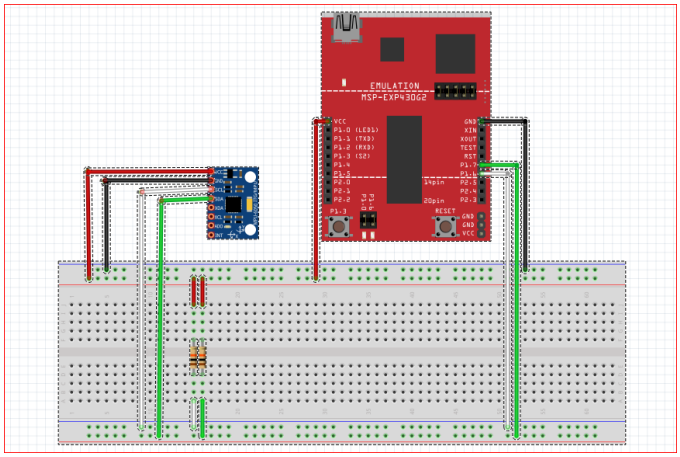

Figura 3- Diagrama para montagem do circuito.

Na figura 3 os resistores são resistores de pull-up, servem para não deixar a conexão sem uma tensão positiva.

Descrição do software

Através de dados obtidos no datasheet para um sistema single master receiver/transmitter os protocolos de comunicação são descritos a seguir:

Operação de escrita de dados (Escrever dados nos registradores do MPU-6050) para mandar um único byte[6]:

1. Condição de START.
2. AD+W -> Master envia : Slave address + write bit.
3. Slave envia ACK.
4. RA -> Master envia : Register Address to be write.
5. Slave envia ACK.
6. Dados para RA.
7. Slave envia ACK.
8. Condição de STOP.

Este protocolo foi implementado na launchpad e encontra-se no anexo 3 nas funções:

- void i2cInit(void);
- void i2cWrite(unsigned char);
- void i2cRead(unsigned char);
- E na interrupção: __interrupt void USCIAB0TX_ISR(void).

Um fluxograma mais abrangente é apresentado na figura a seguir:

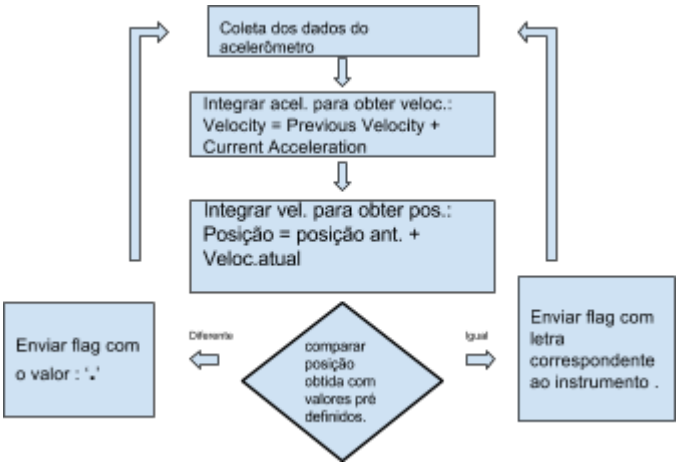

Figura 4 - Fluxograma do código.

O fluxograma da figura 3 exemplifica o código que "roda" na launchpad , a flag enviada é do tipo char, sendo o receptor um programa feito na plataforma processing que ao reconhecer um caracter específico executa o arquivo de áudio relativo ao instrumento que a flag(caracter) representa.

O algoritmo foi feito da seguinte maneira:

1. Calculou-se a área 2(Figura 2) e somou-se a área 1(Figura 2).
2. A aceleração obtida está em termos de metros por $segundo^2$ portanto basta somar ao resultado anterior para obter a velocidade.
3. De forma análoga com as áreas 1 e 2 da velocidade calculadas soma-se o novo valor da velocidade à posição.

Após a execução dos passos acima o valor é comparado , então uma flag é enviada via comunicação UART para o processing em um computador fazer o processo de decisão se algum arquivo de áudio que representa uma parte da bateria deve ser executado, como mostrado no fluxograma da figura 5.

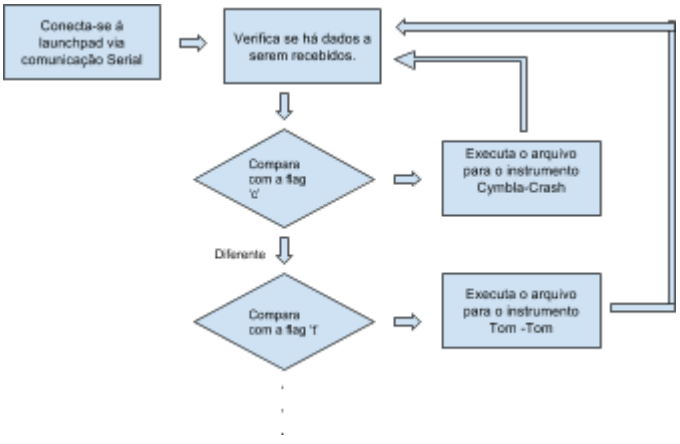

Figura 5 - Fluxograma do codigo escrito no processing.

Resultados

A aceleração, velocidade e posicionamento foram obtidos com sucesso, porém o acelerômetro não retorna ao valor zero quando parado fora do local e posição em que se encontra quando o programa é iniciado, pelo fato do sensor ser do tipo capacitivo quando deslocado da posição de origem, posição com offset ajustado e calibrado, a distância entre as placas é diferente , provocando uma capacitância diferente e consequentemente uma aceleração com offset diferente.

O processing funcionou perfeitamente, para teste, fora enviado flags específicas para verificar se o código executa o arquivo de áudio desejado, o código se encontra no anexo 3.

Conclusão

Através do presente trabalho foi possível aprender sobre alguns dos principais tipos de comunicação serial: I2C e UART. O projeto foi implementado com sucesso, entretanto o uso de um plataforma mais específica e voltada a captura da posição forneceria um resultado melhor embora a MPU-6050 contenha uma plataforma DMP(digital motion process) , esta é melhor aproveitada para determinar rotação, guinada e inclinação compondo o movimento completo(junto dos dados do acelerômetro) de um um corpo rígido como a baqueta, por exemplo, podendo fundir os movimentos de translação e rotação para uma resposta mais satisfatória.

References

[1]como-medir-velocidade-e-deslocamento-a-partir-de-um-acelerometro.Disponível em: https://camilasoares.wordpress.com/2009/05/10/como-medir-velocidade-e-deslocamento-a-partir-de-um-acelerometro.

[2]aplication note.Disponível em: http://cache.freescale.com/files/sensors/doc/app_note/AN3397.pdfhttp://cache.freescale.com/files/sensors/doc/app_note/AN3397.pdf

[3]User-guide MSP-430.

Disponívle em:http://www.ti.com/product/MSP430G2553

[4]George B Thomas,Maurice D.Weir, Joel Hass , Cálculo,volume 2;tradução Carlos Scalici;12 ed.-São Paulo:

Pearson Education do Brasil,2012.

[5]Datasheet msp430g2553. Disponivel em :http://www.ti.com/product/MSP430G2553

[6]Datasheet mpu6050.Disponível em:https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf

[7]Arduino for Production! How to Communicate with I2C or TWI (1 of 2) - Tutorial on I2C TWI Part 3. Disponível em: https://www.youtube.com/watch?v=7aDjYqOMu9w

[8]Processing.Disponível em: https://processing.org/

Anexo I

```
/* //first integration
  *          velocityx[1] = velocityx[0] + accelerationx[0] + ((accelerationx[1] - accelerationx[0])>>1)
 * //second integration to obtain position
  *           positionX[1] = positionX[0] + velocityx[0] + ((velocityx[1] - velocityx[0])>>1);
 *
 * I2C_Accel_MPU6050
 *
  * Version 1:  Read raw accelerometer X, Y and Z data continuously
 *
 *  P1.6                UCB0SCL
 *  P1.7                UCB0SDA
 *
 * MPU-6050 Accelerometer & Gyro
 * NOTE:  Default state of the MPU-6050 is SLEEP mode.
  *          Wake up by writing 0x00 to the PWR_MGMT_1 register
 * NOTE:  No-name version from Amazon has a 5V to 3.3V regulator, and Vcc MUST be 5V !!!
 *       10-kOhm pull-up resistors are included on the board
 *        330-Ohm resistors are included in series on SCL and SDA
 *             (safe to connect P1.6 & P1.7 directly to SCL and SDA)
 *
 * Slave address:  0x68 (AD0=0) or 0x69 (AD0=1)
 *
 * Z-data buffer addresses:
  *                                0x3B  ACCEL_XOUT_H  R ACCEL_XOUT[15:8]
 *                0x3C ACCEL_XOUT_L R ACCEL_XOUT[ 7:0]
  *                                0x3D  ACCEL_YOUT_H  R ACCEL_YOUT[15:8]
 *             0x3E ACCEL_YOUT_L R ACCEL_YOUT[ 7:0]
  *                                0x3F  ACCEL_ZOUT_H  R ACCEL_ZOUT[15:8]
 *             0x40 ACCEL_ZOUT_L R ACCEL_ZOUT[ 7:0]
 *
  * pins not used:    INT (interrupt for data ready in the 1024-byte FIFO bufer)
 *                XCL, XDA (external clock and data lines for MPU-6050 I2C bus)
 *
 * Reading the raw values: disable sleep mode
 *             0x6B PWR_MGMT_1 --> set to 0
 *
 *             AFS_SEL => full scale range.
 *
 */
```

```
#include <msp430g2553.h>
#define AFS_SEL 0x00

#define BAUD_9600   0
#define BAUD_19200  1
#define BAUD_38400  2
#define BAUD_56000  3
#define BAUD_115200 4
#define BAUD_128000 5
#define BAUD_256000 6
#define NUM_BAUDS   7

#define RX BIT1
#define TX BIT2

unsigned char RX_Data[6];
unsigned char TX_Data[2];
unsigned char RX_ByteCtr;
unsigned char TX_ByteCtr;

unsigned char slaveAddress = 0x68;  // Set slave address for MPU-6050  0x68 for ADD pin=0, 0x69 for ADD pin=1

const unsigned char ACCEL_XOUT_H = 0x3B;        // MPU-6050 register address
const unsigned char ACCEL_XOUT_L = 0x3C;        // MPU-6050 register address
const unsigned char ACCEL_YOUT_H = 0x3D;        // MPU-6050 register address
const unsigned char ACCEL_YOUT_L = 0x3E;        // MPU-6050 register address
const unsigned char ACCEL_ZOUT_H = 0x3F;        // MPU-6050 register address
const unsigned char ACCEL_ZOUT_L = 0x40;        // MPU-6050 register address

int pos_post[3]  = {0,0,0};
int vel_post[3]  = {0,0,0};
int acel_post[3] = {0,0,0};

int acel_ant[3]  = {0,0,0};
int vel_ant[3]   = {0,0,0};
int pos_ant[3]   = {0,0,0};

void i2cInit(void);
void i2cWrite(unsigned char);
void i2cRead(unsigned char);

void get_Accel_value(void);
void posicao(void);

void setClock(void);

void Send_Data(unsigned char c);
char* instrumento(int position[3]);
```

```
void Init_UART(unsigned int baud_rate_choice);

long map(long,long,long,long,long);

int Accel[3];

int main(void)
{   volatile static unsigned char count = 0;
     WDTCTL = WDTPW + WDTHOLD;                  // Stop
WDT Set clock speed (default = 1 MHz)
   setClock();
   Init_UART(BAUD_9600);
   TA0CCR0 = 62500 - 1; //10000-1;
   TA0CCR1 = 62500 - 1; //10000-1;
   TA0CTL = TASSEL_2 + ID_3 + MC_1 + TAIE;
   _BIS_SR(GIE);
}

void Init_UART(unsigned int baud_rate_choice)
{
  unsigned char BRs[NUM_BAUDS] = {104, 52, 26, 17, 8, 7,
3};
    unsigned char MCTLs[NUM_BAUDS] = {UCBRF_0 +
UCBRS_1,
                          UCBRF_0 + UCBRS_0,
                          UCBRF_0 + UCBRS_0,
                          UCBRF_0 + UCBRS_7,
                          UCBRF_0 + UCBRS_6,
                          UCBRF_0 + UCBRS_7,
                          UCBRF_0 + UCBRS_7
                         };
  if (baud_rate_choice < NUM_BAUDS)
  {
   // Habilita os pinos para transmissao serial UART
   P1SEL2 = P1SEL = RX + TX;
      // Configura a transmissao serial UART com 8 bits de
dados,
   // sem paridade, comecando pelo bit menos significativo,
   // e com um bit de STOP
   UCA0CTL0 = 0;
   // Escolhe o SMCLK como clock para a UART
   UCA0CTL1 = UCSSEL_2;
   // Define a baud rate
   UCA0BR0 = BRs[baud_rate_choice];
   UCA0BR1 = 0;
   UCA0MCTL = MCTLs[baud_rate_choice];
  }
}


char instrumento(int position[3]){
     /*Return the name of the instrument with the data of the
mpu6050
     *turned into position matching with the programmed for an
piece
     *of the instrument */
    /*
     * Tomtoms
     * Cymbalcrash
     * */
          if((position[1]<=10 &&  position[1]>=-10) &&
(position[2]<= 10&&  position[2]>=-10) &&(position[3]<=-5
&&  position[3]>=-20))
   {return 't';};

    /*if((position[1]<=- &&  position[1]>=) && (position[2]<=
&&  position[2]>=) &&(position[3]<= &&  position[3]>=))
   {return 'c';};*/


   return  '.';
}

void Send_Data(unsigned char c)
{
   while((IFG2&UCA0TXIFG)==0);
   UCA0TXBUF = c;
}

void setClock(void){
     DCOCTL  = 0;             // Select lowest DCOx and MODx
settings
    BCSCTL1 = CALBC1_1MHZ;   // Set range
       DCOCTL  = CALDCO_1MHZ;    // Set DCO step +
modulation
}



void i2cInit(void)
{
   // set up I2C module
   // set up I2C pins
    P1SEL |= BIT6 + BIT7;                   // Assign I2C pins to
USCI_B0
    P1SEL2|= BIT6 + BIT7;                   // Assign I2C pins to
USCI_B0
   UCB0CTL1 |= UCSWRST;              // Enable SW reset
      UCB0CTL0 = UCMST + UCMODE_3 + UCSYNC;
// I2C Master, synchronous mode
    UCB0CTL1 = UCSSEL_2 + UCSWRST;             // Use
SMCLK, keep SW reset
     UCB0BR0 = 10;                      // fSCL = SMCLK/12 =
~100kHz
   UCB0BR1 = 0;
    UCB0CTL1 &= ~UCSWRST;                // Clear SW reset,
resume operation
}
```

```
void i2cWrite(unsigned char address)
{
  __disable_interrupt();
  UCB0I2CSA = address;              // Load slave address
  IE2 |= UCB0TXIE;              // Enable TX interrupt
    while(UCB0CTL1 & UCTXSTP);              // Ensure stop condition sent
   UCB0CTL1 |= UCTR + UCTXSTT;          // TX mode and START condition
    __bis_SR_register(CPUOFF + GIE);           // sleep until UCB0TXIFG is set ...
}

void i2cRead(unsigned char address)
{
  __disable_interrupt();
  UCB0I2CSA = address;              // Load slave address
  IE2 |= UCB0RXIE;              // Enable RX interrupt
    while(UCB0CTL1 & UCTXSTP);              // Ensure stop condition sent
  UCB0CTL1 &= ~UCTR;              // RX mode
  UCB0CTL1 |= UCTXSTT;              // Start Condition
    __bis_SR_register(CPUOFF + GIE);           // sleep until UCB0RXIFG is set ...
}

long map(long Accel, long in_min, long in_max, long out_min, long out_max)
{
  return (Accel - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}
void get_Accel_value(void)
{
  // Initialize the I2C state machine
  i2cInit();

  // Wake up the MPU-6050
  slaveAddress = 0x68;              // MPU-6050 address
    TX_Data[1] = 0x6B;                          // address of PWR_MGMT_1 register
   TX_Data[0] = 0x00;                       // set register to zero (wakes up the MPU-6050)
  TX_ByteCtr = 2;
  i2cWrite(slaveAddress);

   set full scale
  slaveAddress = 0x68;              // MPU-6050 address
  TX_Data[1] = 0x1C;               // address of acel config
    TX_Data[0] = AFS_SEL;                       // set register AFS_SEL
  TX_ByteCtr = 2;
  i2cWrite(slaveAddress);

  // Point to the ACCEL_ZOUT_H register in the MPU-6050
  slaveAddress = 0x68;              // MPU-6050 address
  TX_Data[0] = 0x3B;               // register address
  TX_ByteCtr = 1;
  i2cWrite(slaveAddress);

  // Read the two bytes of data and store them in zAccel
  slaveAddress = 0x68;              // MPU-6050 address
  RX_ByteCtr = 6;
  i2cRead(slaveAddress);

  acel_post[0]  = RX_Data[5] << 8;          // MSB
  acel_post[0] |= RX_Data[4];             // LSB
  acel_post[1]  = RX_Data[3] << 8;          // MSB
  acel_post[1] |= RX_Data[2];             // LSB
  acel_post[2]  = RX_Data[1] << 8;          // MSB
  acel_post[2] |= RX_Data[0];             // LSB

  acel_post[0] = map(acel_post[0],0,17000,0,20);
  acel_post[1] = map(acel_post[1],0,-17000,0,20);
  acel_post[2] = map(acel_post[2],0,17000,0,20);
}

void posicao(void){


   if ((acel_post[0] <= 3)&&(acel_post[0] >= -3)) //Discrimination window applied  to the X axis acceleration variable
  {
    acel_post[0] = 0;
  }

 if ((acel_post[1] <= 3)&&(acel_post[1] >= -3))
  {
    acel_post[1] = 0;
  }

 if ((acel_post[2] <= 3)&&(acel_post[2] >= -3))
  {
    acel_post[2] = 0;
  }

//first inetegration

    vel_post[0] = vel_ant[0] + acel_ant[0] + ((acel_post[0] - acel_ant[0])>>1);
    vel_post[1] = vel_ant[1] + acel_ant[1] + ((acel_post[1] - acel_ant[1])>>1);
    vel_post[2] = vel_ant[2] + acel_ant[2] + ((acel_post[2] - acel_ant[2])>>1);

//second integration
     pos_post[0] = pos_ant[0] + vel_ant[0] + ((vel_post[0] - vel_ant[0])>>1);
```

```
      pos_post[1] = pos_ant[1] + vel_ant[1] + ((vel_post[1] -
vel_ant[1])>>1);
      pos_post[2] = pos_ant[2] + vel_ant[2] + ((vel_post[2] -
vel_ant[2])>>1);

//previous value atualization

   acel_ant[0] = acel_post[0];
   acel_ant[1] = acel_post[1];
   acel_ant[2] = acel_post[2];

   vel_ant[0] = vel_post[0] ;
   vel_ant[1] = vel_post[1] ;
   vel_ant[2] = vel_post[2] ;

   pos_ant[0] = pos_post[0] ;
   pos_ant[1] = pos_post[1] ;
   pos_ant[2] = pos_post[2] ;

}

#pragma vector = USCIAB0TX_VECTOR
__interrupt void USCIAB0TX_ISR(void)
{
    if(UCB0CTL1 & UCTR)               // TX mode (UCTR ==
1)
   {
       if (TX_ByteCtr)                // TRUE if more bytes
remain
      {
         TX_ByteCtr--;              // Decrement TX byte counter
         UCB0TXBUF = TX_Data[TX_ByteCtr];    // Load TX
buffer
      }
      else                     // no more bytes to send
      {
         UCB0CTL1 |= UCTXSTP;          // I2C stop condition
           IFG2 &= ~UCB0TXIFG;        // Clear USCI_B0 TX
int flag
         __bic_SR_register_on_exit(CPUOFF);  // Exit LPM0
      }
   }
   else // (UCTR == 0)               // RX mode
   {
       RX_ByteCtr--;                      // Decrement RX byte
counter
      if (RX_ByteCtr)                 // RxByteCtr != 0
      {
            RX_Data[RX_ByteCtr] = UCB0RXBUF;     // Get
received byte
         if (RX_ByteCtr == 1)          // Only one byte left?
         UCB0CTL1 |= UCTXSTP;          // Generate I2C stop
condition
      }
      else                     // RxByteCtr == 0
      {
         RX_Data[RX_ByteCtr] = UCB0RXBUF;    // Get final
received byte
         __bic_SR_register_on_exit(CPUOFF);  // Exit LPM0
      }
   }
}



#pragma vector = TIMER0_A1_VECTOR
__interrupt void USCIAB0TX_ISR(void)
{
 Send_Data(instrumento(pos_post));
 TA0CTL &= ~TAIFG;
}
```

Anexo 2

```
import processing.sound.*;
SoundFile cymbalcrash1;
SoundFile tomtomdrum1;

import processing.serial.*;

Serial msp;  // The serial port:

void setup() {
  // List all the available serial ports:
  printArray(Serial.list());
  // Open the port you are using at the rate you want:
  msp = new Serial(this, Serial.list()[0], 9600);
  cymbalcrash1 = new SoundFile(this, "cymbalcrash1.wav");
  tomtomdrum1 = new SoundFile(this,"tomtomdrum1.wav");

}

void draw() {
  while (msp.available() > 0) {
    char inByte = msp.readChar();
    if( inByte == 'c'){ cymbalcrash1.play();}

    if( inByte == 't'){ tomtomdrum1.play();}
    println(inByte);

  }
}
```

Anexo 3

```
#include <msp430g2553.h>

#define BTN BIT3
#define RX BIT1
#define TX BIT2

#define BAUD_9600   0
#define BAUD_19200  1
#define BAUD_38400  2
#define BAUD_56000  3
#define BAUD_115200 4
#define BAUD_128000 5
#define BAUD_256000 6
#define NUM_BAUDS   7

void Send_Data(unsigned char c);
void Init_UART(unsigned int baud_rate_choice);

int main(void)
{
  volatile int i = 0;
  WDTCTL = WDTPW + WDTHOLD;

  BCSCTL1 = CALBC1_1MHZ;
  DCOCTL = CALDCO_1MHZ;

  Init_UART(BAUD_9600);
  TA0CCR0 = 62500-1; //10000-1;
  TA0CTL = TASSEL_2 + ID_3 + MC_1;
  while(1)
  {
    while((TA0CTL & TAIFG)==0);
    TA0CTL &= ~TAIFG;
    Send_Data('c');

//    while((TA0CTL & TAIFG)==0);
//    TA0CTL &= ~TAIFG;
//    Send_Data('t');
  }
  return 0;
}

void Send_Data(unsigned char c)
{
  while((IFG2&UCA0TXIFG)==0);
  UCA0TXBUF = c;
}

void Init_UART(unsigned int baud_rate_choice)
{
  unsigned char BRs[NUM_BAUDS] = {104, 52, 26, 17, 8, 7,
3};
        unsigned char MCTLs[NUM_BAUDS] =
{UCBRF_0+UCBRS_1,
              UCBRF_0+UCBRS_0,
              UCBRF_0+UCBRS_0,
              UCBRF_0+UCBRS_7,
              UCBRF_0+UCBRS_6,
              UCBRF_0+UCBRS_7,
              UCBRF_0+UCBRS_7};
  if(baud_rate_choice<NUM_BAUDS)
  {
    // Habilita os pinos para transmissao serial UART
    P1SEL2 = P1SEL = RX+TX;
      // Configura a transmissao serial UART com 8 bits de
dados,
    // sem paridade, comecando pelo bit menos significativo,
    // e com um bit de STOP
    UCA0CTL0 = 0;
    // Escolhe o SMCLK como clock para a UART
    UCA0CTL1 = UCSSEL_2;
    // Define a baud rate
    UCA0BR0 = BRs[baud_rate_choice];
    UCA0BR1 = 0;
    UCA0MCTL = MCTLs[baud_rate_choice];
  }
}
```