Controlul PC-ului prin gesture

Nume : Dimoiu Mihai – Lorin

Grupa : 333 AA

In acest proiect vom controla mouse-ul PC-ului printr-un senzor accelerometru conectat la o placa de dezvoltare Arduino UNO R3.

Am inceput prin a ma documenta despre senzorii accelerometru si din varietatea lor, am ales sa folosesc senzorul accelerometru ADXL345.

Senzorul ADXL345 este un senzor ultra-low power consumand doar 40 microA in modul de masurare iar 0.1 microA in modul standby. Alimentarea acestui senzor este intre 2.0 V respective 3.6 V.

Acest senzor este capabil sa dea un raspuns pentru fiecare axa (x,y,z) pe 10 biti. De asemenea are integrat un vector de intreruperi folosit pentru a detecta miscarile la nivel hardware. In acest vector sunt stocati biti care marcheaza spre exemplu daca s-a produs o detectie de cadere libera, sau de dubla atingere intr-un inverval setat prin program.

Conectarea acestui senzor se face prin 2 moduri configurand un pin CS (Chip Select). Daca acest pin se pune la V_+, senzorul va rula in modul I2C. Daca se pune la GND senzorul va rula in modul SPI.

In acest proiect am folosit modul I2C conectand CS la V_+ .

In urmatorul tabel se evidentiaza configurarea pinilor :

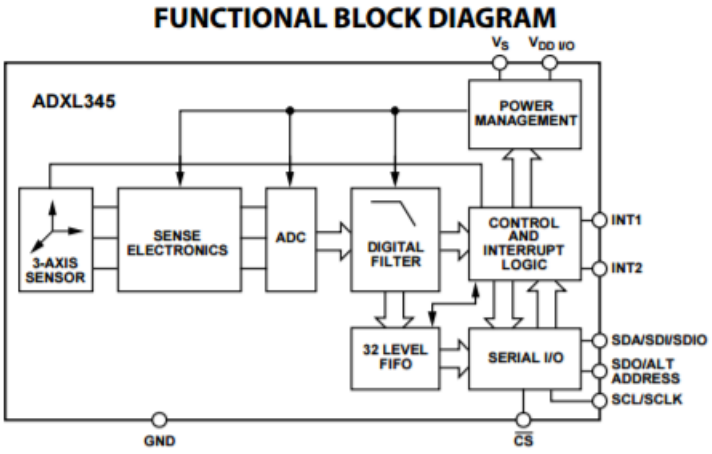
Arduino Pin	ADXL345 Pin
GND	GND
3V3	VCC
3V3	CS
GND	SDO
A4	SDA
A5	SCL

S-au folosit 2 rezistente de PULL-UP de 4K7 pentru SDA respective SCL.

Dimensiunea senzorului este de 3mm x 5mm x 1mm LGA.

Rezolutia sa este de 4mb/LSB facand posibila masurarea inclinatiei sub 1 grad.

Diagrama senzorului ADXL345 dupa care functioneaza este :



Materialele folosite :

- Arduino UNO R3
- Senzor Accelerometru ADXL345
- Fire
- BreadBoard
- 2x Rezistente 4K7

Despre Arduino UNO

Arduino UNO este o platforma de procesare open-source, bazata pe software si hardware flexibil si simplu de folosit. Consta intr-o platforma de mici dimensiuni (6.8 cm / 5.3 cm – in cea mai des intalnita varianta) construita in jurul unui procesor de semnal si este capabila de a prelua date din mediul inconjurator printr-o serie de senzori si de a efectua actiuni asupra mediului prin intermediul luminilor, motoarelor, servomotoare, si alte tipuri de dispozitive mecanice. Procesorul este capabil sa ruleze cod scris intr-un limbaj de programare care este foarte similar cu limbajul C++ - ANSI C.

Placa Arduino UNO se conecteaza la portul USB al calculatorului folosind un cablu de tip USB A-B. Poate fi alimentata extern folosind un alimentator extern. Alimentarea externa este necesara in situatia in care consumatorii conectati la placa necesita un curent mai mare de cateva sute de miliamperi. In caz contrar, placa se poate alimenta direct din PC, prin cablul USB.

Specificatii :

- Microcontroler: ATmega328
- Tensiune de lucru: 5V
- Tensiune de intrare (recomandat): 7-12V
- Tensiune de intrare (limita): 6-20V
- Pini digitali: 14 (6 PWM output)
- Pini analogici: 6
- Curent per pin I/O: 40 mA
- Curent 3.3V: 50 mA
- Memorie Flash: 32 KB (ATmega328) 0.5 KB pentru bootloader
- SRAM: 2 KB (ATmega328)
- EEPROM: 1 KB (ATmega328)
- Clock Speed: 16 MHz

Am continuat despre documentarea citirii de pe portul serial in limbajul C++. Am configurat parametrii necesari pentru a face sincronizarea cu placa de dezvoltare de pe care se trimiteau date colectate de la senzorul accelerometru.

Am colectat date de la placa de dezvoltare pentru a intelege si dezvolta un algoritm in vedere miscarii mouse-ului.

Am folosit cele 2 axe x, respective y pentru a misca mouse-ul sus – jos, respective stanga – dreapta. Iar axa z pentru a simula click-ul stanga al mouse-ului.

Daca senzorul este intors pe axa y la 90 de grade catre stanga, soft-ul dezvoltat in C++ va simula click-ul dreapta al mouse-ului.

Daca senzorul este rotit pe axa x, in fata, softu-ul va simula scroll-ul mose-ului.

In continuarea voi prezenta codul scris pentru placa de dezvoltare Arduino :

```
#include <SparkFun_ADXL345.h>
ADXL345 adxl = ADXL345();             // I2C Comunicatie
int x = 0;
int y = 0;
int z = 0;
void setup(){

  Serial.begin(9600);
  adxl.powerOn();
  adxl.setRangeSetting(8);           //Range setting
  adxl.setTapDetectionOnXYZ(0, 0, 1);
  adxl.setDoubleTapLatency(80);       // 1.25 ms per increment
  adxl.setDoubleTapWindow(200);       // 1.25 ms per increment
  adxl.setFreeFallThreshold(7);       // (5 - 9) recomandat - 62.5mg per increment
  adxl.setFreeFallDuration(30);       // (20 - 70) reomandat - 5ms per increment
  adxl.FreeFallINT(1);
  adxl.doubleTapINT(1);
}
void loop(){
  adxl.readAccel(&x, &y, &z);
  Serial.write(x);
  Serial.write(y);
  Serial.write(z);
  delay(70);
}
```

S-a folosit libraria SparkFun_ADXL345 pentru a facilita dezvoltarea mai rapida a codului.

In continuare este prezentat programul scris in limbajul C++ pentru a citi datele colectate de catre placuta Arduino :

```
#include <stdio.h>

#include <tchar.h>

#include "SerialClass.h"

#include <string>

#include "Serial.cpp"

#include <cstdint>
```

```
#include <windows.h>
#include <stdlib.h>
void LeftClickDown()    // Click stanga
{
   mouse_event(MOUSEEVENTF_LEFTDOWN, 0, 0, 0, 0);
}
void leftClickUp()
{
   mouse_event(MOUSEEVENTF_LEFTUP, 0, 0, 0, 0);
}
void RightClickDown()    // Click stanga
{
   mouse_event(MOUSEEVENTF_RIGHTDOWN, 0, 0, 0, 0);
}
void RightClickUp()
{
   mouse_event(MOUSEEVENTF_RIGHTUP, 0, 0, 0, 0);
}
void MoveWheel()    // Rotita in jos cu un ecran
{
   mouse_event(MOUSEEVENTF_WHEEL,0,0,-850,0);
}
int _tmain(int argc, _TCHAR* argv[])
{
   printf("Inceperea citirii . . . \n\n");
   Serial* SP = new Serial("COM3");    //Portul
   if (SP->IsConnected())
```

```
   printf("Conectat");
char incomingData[256];                    //Prealocarea memoriei
for(int i = 0 ; i<256; i++)
   incomingData[i] = 0;
int dataLength = 6;
int readResult = 0;
int x=0,y=0,z=0;
POINT p;    //variabila pentru recuperarea pozitiei curente a mouse-ului
          //respectiv pentru actualizarea pozitiei
while(SP->IsConnected())
{
   system ("cls");
   readResult = SP->ReadData(incomingData,dataLength);
   GetCursorPos(&p);
   // printf("Bytes cititi: (0 = nu sunt date disponibile) %i\n",readResult);
   incomingData[readResult] = 0;
   x = (int)incomingData[0];
   y = (int)incomingData[1];
   z = (int)incomingData[2];
   //Citirea valorilor :
   printf("x = %d\n",x);
   printf("y = %d\n",y);
   printf("z = %d\n\n",z);
   if( (int)incomingData[2] > 'M') //Codul ascii pentru comparare M = 77
   {
      LeftClickDown();
      leftClickUp();
```

```
        }
        if( (int)incomingData[1] > '(') //Codul ascii pentru comparare ( = 40
        {
            RightClickDown();
            RightClickUp();
        }
        if( (int)incomingData[0] > '(') //Codul ascii pentru comparare ( = 40
        {
            MoveWheel();
        }
        SetCursorPos( p.x-y, p.y-x );
    }
    return 0;
}
```

In cele ce urmeaza se va prezenta clasa impreuna cu header-ul ei pentru configurarea portului serial :

Serial.cpp :

```
#include "SerialClass.h"

Serial::Serial(const char *portName)
{
    //We're not yet connected
    this->connected = false;

    //Try to connect to the given port throuh CreateFile
```

```
this->hSerial = CreateFile(portName,
        GENERIC_READ | GENERIC_WRITE,
        0,
        NULL,
        OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL,
        NULL);

//Check if the connection was successfull
if(this->hSerial==INVALID_HANDLE_VALUE)
{
    //If not success full display an Error
    if(GetLastError()==ERROR_FILE_NOT_FOUND){

        //Print Error if neccessary
        printf("ERROR: Handle was not attached. Reason: %s not available.\n", portName);

    }
    else
    {
        printf("ERROR!");
    }
}
else
{
    //If connected we try to set the comm parameters
    DCB dcbSerialParams = {0};
```

```
//Try to get the current
if (!GetCommState(this->hSerial, &dcbSerialParams))
{
   //If impossible, show an error
   printf("failed to get current serial parameters!");
}
else
{
   //Define serial connection parameters for the arduino board
   dcbSerialParams.BaudRate=CBR_9600;
   dcbSerialParams.ByteSize=8;
   dcbSerialParams.StopBits=ONESTOPBIT;
   dcbSerialParams.Parity=NOPARITY;
   //Setting the DTR to Control_Enable ensures that the Arduino is properly
   //reset upon establishing a connection
   dcbSerialParams.fDtrControl = DTR_CONTROL_ENABLE;

    //Set the parameters and check for their proper application
    if(!SetCommState(hSerial, &dcbSerialParams))
    {
       printf("ALERT: Could not set Serial Port parameters");
    }
    else
    {
        //If everything went fine we're connected
        this->connected = true;
```

```
            //Flush any remaining characters in the buffers
            PurgeComm(this->hSerial, PURGE_RXCLEAR | PURGE_TXCLEAR);
            //We wait 2s as the arduino board will be reseting
            Sleep(ARDUINO_WAIT_TIME);
        }
    }
  }

}

Serial::~Serial()
{
  //Check if we are connected before trying to disconnect
  if(this->connected)
  {
    //We're no longer connected
    this->connected = false;
    //Close the serial handler
    CloseHandle(this->hSerial);
  }
}

int Serial::ReadData(void *buffer, unsigned int nbChar)
{
  //Number of bytes we'll have read
  DWORD bytesRead;
  //Number of bytes we'll really ask to read
```

```
unsigned int toRead;

//Use the ClearCommError function to get status info on the Serial port
ClearCommError(this->hSerial, &this->errors, &this->status);

//Check if there is something to read
if(this->status.cbInQue>0)
{
   //If there is we check if there is enough data to read the required number
   //of characters, if not we'll read only the available characters to prevent
   //locking of the application.
   if(this->status.cbInQue>nbChar)
   {
      toRead = nbChar;
   }
   else
   {
      toRead = this->status.cbInQue;
   }

   //Try to read the require number of chars, and return the number of read bytes on success
   if(ReadFile(this->hSerial, buffer, toRead, &bytesRead, NULL) )
   {
      return bytesRead;
   }

}
```

```
    //If nothing has been read, or that an error was detected return 0
    return 0;

}



bool Serial::WriteData(const char *buffer, unsigned int nbChar)
{
    DWORD bytesSend;

    //Try to write the buffer on the Serial port
    if(!WriteFile(this->hSerial, (void *)buffer, nbChar, &bytesSend, 0))
    {
        //In case it don't work get comm error and return false
        ClearCommError(this->hSerial, &this->errors, &this->status);

        return false;
    }
    else
        return true;
}
bool Serial::IsConnected()
{
    //Simply return the connection status
    return this->connected;}
```

Serial.h :

```
#ifndef SERIALCLASS_H_INCLUDED
#define SERIALCLASS_H_INCLUDED
#define ARDUINO_WAIT_TIME 2000
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
class Serial
{
   private:
      //Serial comm handler
      HANDLE hSerial;
      //Connection status
      bool connected;
      //Get various information about the connection
      COMSTAT status;
      //Keep track of last error
      DWORD errors;
   public:
      Serial(const char *portName);
      ~Serial();
      int ReadData(void *buffer, unsigned int nbChar);
      bool WriteData(const char *buffer, unsigned int nbChar);
      bool IsConnected();};

#endif // SERIALCLASS_H_INCLUDED
```