Step tracker based on MSP430 microcontroller

Jose Carlos Sanchez Morales | Wireless and mobile networks | 15/06/18

Technical document about specifications and configurations of the developed project

Index

Introduction

Nowadays people can handle with interactive maps thanks to smartphones' sensors (concretely GPS, gyroscopes and accelerometers) to be guided in a foreign environment, or search any unknown building, square, street, even any kind of shops.

But interactive maps are only functional when it is used at outdoors, so it is a problem if it desired to track a close enviroment.

The suggested solution consists in a low-power device, were using a loaded map, it could be used for tracking a person in a closed environment.

System

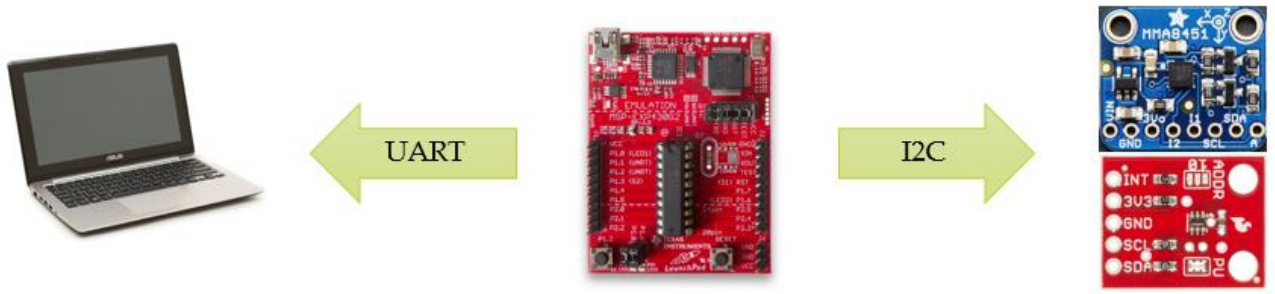


Figure 1: *Project functionality's representation.*

The system consists in collecting data provided by sensors and send it to microcontroller for process it. Then, the data is sent to laptop, where is graphically represented in one graphic interface, doing a simulation like a wearable.

Devices could be divided in three parts:

- **Microcontroller:** Takes the processing of sensors' data. The microcontroller selected is MSP430G2553 TI launchpad.

- **Sensors:**
  - **TSL2561:** A luminosity sensor. Takes the measure of ambient light and send the data to microcontroller.
  - **MMA8451:** An accelerometer. Takes the measure of amount of acceleration given and send the data to microcontroller.

- **Laptop:** Receive the data, shows it in a graphic interface, and program the microcontroller to do a correct processing.

The connection between devices are different. There are two kinds of protocols:

- **UART:** allow data transmission between microcontroller and laptop. The microcontroller sends the data and laptop receive it.
- **I2C:** allow communication between sensors and microcontroller. Microcontroller ask for the data and sensors replies with it.

The aim of the project consists in develop a low-power system able to guide people in closed areas.

Protocols

It is necessary explain the system's protocols for understand completely the functionally and why was selected for the project.

I2C

Invented by Phillips (now NXP semiconductors) in 1982, is a synchronous master-slave protocol where master can send data calling to address devices, configured in a bus-architecture. The architecture could handle with more than one master, but it is not recommended.

If is not embedded pull-up resistors on device, it is necessary configurate for SDA and SCL pins to avoid signal noise. SDA and SCL are exclusive from this protocol:

- **SDA (Serial Data Line):** allows data transmission between master and slave.
- **SCL (Serial Clock Line):** transmit the same clock signal between devices.

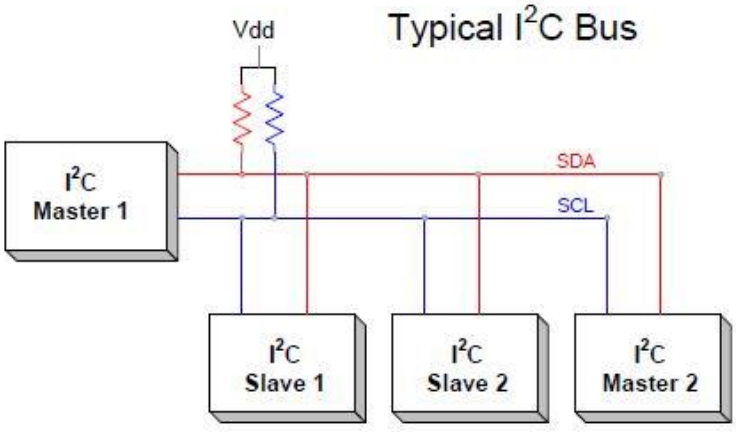


Figure 2: *I2C configuration's schematic*

UART

It is a very common protocol used for connecting devices.

There are two different pins for do a connection: Transmission pin and Reception pin.

- **Transmission pin (Tx):** is an output pin to transmit data.
- **Reception pin (Rx):** is an input pin to receive data.

The transmission is asynchronously, which means the payload included start and stop bits and if the reception UART detects start bit, begins to read in one specific band rate.

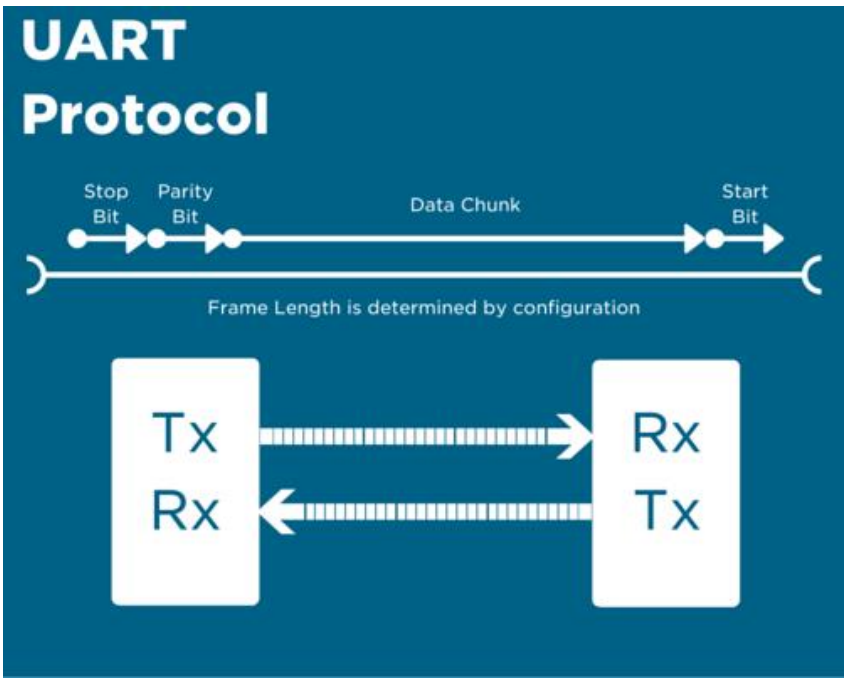


Figure 3: *UART configuration's schematic.*

Hardware

This section will try to explain different reasons why this hardware was selected specifically for the system.

MSP430G2553

The MSP430G2553 is a microcontroller manufactured by Texas Instrument. It could be comparable with Arduino Uno, but it has low power consumption and the programming is more efficient. On the other hand, MSP430G2553 is more difficult to program and does not have the same support than Arduino.

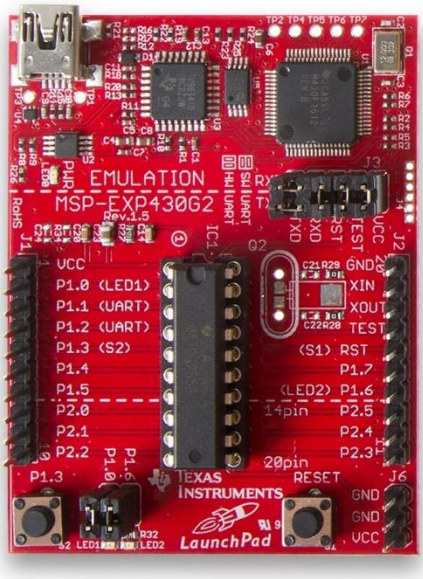


Figure 4: *MSP430G2 TI launchpad*

In the next table we can appreciate the specifications of MSP430G2553:

Flash Memory	16 kB
RAM Memory	512 B
GPIOs	16
Timers	2 * 16-bits
Watchdog	yes
Brownout Reset	Yes
USCI (use UART, I2C, SPI)	Yes
ADC	8 channels 10-bit
Comparator	8 channels

The launchpad can run with 3.3v and 5V, includes a micro crystal oscillator of 32.768kHz, and has a power supply of 3.6V with a current output of 4 mA each pin approximately.

TSL2561

Focusing on the system, the main characteristics about the luminosity sensor are two: can work with I2C protocol, and the power voltage is 3.3V. The lux level fits in the range from 0.1 to +40k. Include pull-up resistors, and one output interrupt programable pin.

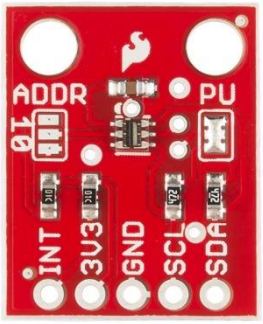


Figure 5: *TSL2561 Sparkfun*

MMA8451

the accelerometer is the master piece of system, so it must be a good choice. It was chosen because has a high precision (14-bit ADC) and the range to use it could be from ± 2g to ± 8g. The device has I2C connection, and a power voltage of 3.3V, like TSL2561. Include pull-up resistors, and one output interrupt programable pin.

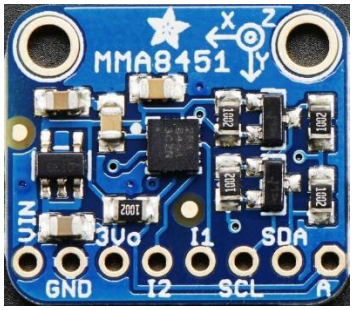


Figure 6: *MMA8451 Adafruit*

Software

This section will try to explain the used software and the program language used.

Languages

It was selected two languages for this project:

1) C is one of the best programming language for embedded systems, and it is used for the development of microcontroller's functionalities. Code Composer Studio (CCS) is the selected IDE for it.

2) C++ is good if it is used on object-oriented programs, which means that could fit good in graphic interface's development. Qt Creator is the selected IDE for it.

Code

The entire code is in this GitHub repository: https://github.com/hoskbreaker/Step-tracker.

The project is divided in two folders, one for processing sensors, and the other one for interface. The next parts are going to explain every kind of them.

Processing Sensors (C language, developed in CCS)

This folder contains building folders, header files and source files. The main files are headers and sources.

Headers

Contains the main functions of our code. This is the header list:

- **I2C.h:** Implement I2C functionalities.
- **uart.h:** Implement UART functionalities.
- **MMA8451.h:** Implement MMA8451 functionalities.
- **TSL2561.h:** Implement TSL2561 functionalities.

I2C.h

void init_i2c(void);	Initialize I2C functionalities on MSP430G2553
void i2c_write (uint8_t, uint8_t *, int, int);	Call to write data function on a device's memory. For this function it is need: o uint8_t deviceAddress o uint8_t * data o int count -> count of bytes used o int stop-> stop bit
void i2c_read (uint8_t, uint8_t *, int);	Call to read data function on a device's memory. It is need: o uint8_t deviceAddress o uint8_t * data o int count -> count of bytes used

int i2c_busy(void);	Checks if I2C bus is busy or not.
int i2c_nack_received(void);	Checks if device has not received data (no Acknowledge).

Uart.h

void init_uart(void);	Initialize UART functionalities on MSP430G2553
void putchar_uart(char);	Wait for transmission to finish
void print_uart(char *);	Print character
void printf_uart(char *, ...);	Printf function on uart

MMA8451.h

uint8_t MMA8451Init (void);	Initialize MMA8451. Returns success or failed
uint8_t MMA8451Reset (void);	Reset MMA8451. Returns success or failed
void MMA8451OrientDetect (void);	Initialize the orientation detection
uint8_t MMA8451GetXAxis (void);	Returns the value of X axis
uint8_t MMA8451GetYAxis (void);	Returns the value of Y axis
uint8_t MMA8451GetZAxis (void);	Returns the value of Z axis
uint8_t MMA8451Status (void);	Returns the orientation state
void MMA8451StandBy (void);	Enable standby mode
void MMA8451Active (void);	Enable active mode
void MMA8451_2g (void);	Configure the device for ± 2g
void MMA8451_4g (void);	Configure the device for ± 4g
void MMA8451_8g (void);	Configure the device for ± 8g
void MMA8451DataRate (void);	Configure device data rate
void MMA8451OverSampling (uint8_t mode);	Configure the oversampling mode. The mode could be: o normal o low power/low noise o High resolution o Low power

TSL2561.h

int tsl2561_power_up (uint8_t);	Initialize TSL2561
void tsl2561_power_down(uint8_t);	Shutdown TSL2561
uint8_t tsl2561_get_id(uint8_t);	Returns device Address
void tsl2561_set_gain_int_time(uint8_t, enum TSL2561_GAIN, enum TSL2561_INT_TIME);	Set the lux ADC gain, and the data rate. It is need: - uint8_t slaveAddress - enum TLS2561_GAIN -> it could be:

	• TSL2561_GAIN_1X = 0 • TSL2561_GAIN_16X = 1 - enum TSL2561_INT_TIME -> it could be: • TSL2561_INT_13_7_MS = 0, • TSL2561_INT_101_MS = 1, • TSL2561_INT_402_MS = 2, • TSL2561_INT_MANUAL = 3
uint32_t tsl2561_get_lux(uint8_t);	Returns the lux obtained

Sources

In source files, header functions are developed and can contain properly functions. In this part it is going to be shown the added functionalities to source code, because in header part it is already explained. This is the source list:

- **I2C.c:** Implement I2C functionalities.
- **uart.c:** Implement UART functionalities.
- **MMA8451.c:** Implement MMA8451 functionalities.
- **TSL2561.c:** Implement TSL2561 functionalities.
- **Main.c:** is the main source where the functionalities are called.

I2C.c

void rx_interrupt(void) __attribute__((interrupt(USCIAB0RX_VECTOR)));	This function is activated when I2C protocol receive the rx signal (USCIAB0RX_VECTOR)
void tx_interrupt(void) __attribute__((interrupt(USCIAB0TX_VECTOR)));	This function is activated when I2C protocol receive the tx signal (USCIAB0TX_VECTOR)

Uart.c

static void print_dec(unsigned int);	Print decimal number
static void print_hex(unsigned int);	Print hexadecimal number
static void print_bin(unsigned int);	Print binary number

MMA8451.c

static inline void write_byte(uint8_t, uint8_t, uint8_t);	Write one byte in I2C - Uint8_t slaveAddress - Uint8_t reg -> register address - Uint8_t byte
static inline void write_word(uint8_t, uint8_t, uint16_t);	Write two bytes in I2C - Uint8_t slaveAddress - Uint8_t reg -> register address - Uint8_t word
static inline uint8_t read_byte(uint8_t, uint8_t);	Read one byte in I2C

	- Uint8_t slaveAddress - Uint8_t reg -> register address
static inline uint16_t read_word(uint8_t, uint8_t);	Read two bytes in I2C - Uint8_t slaveAddress - Uint8_t reg -> register address
void MMA8451FilterCutOff (void)	Stablish a filter cut off
void MMA8451SetDeviceAddr (uint8_t addr)	Sets a device address

TSL2561.c

static inline void write_byte(uint8_t, uint8_t, uint8_t);	Write one byte in I2C - Uint8_t slaveAddress - Uint8_t reg -> register address - Uint8_t byte
static inline void write_word(uint8_t, uint8_t, uint16_t);	Write two bytes in I2C - Uint8_t slaveAddress - Uint8_t reg -> register address - Uint8_t word
static inline uint8_t read_byte(uint8_t, uint8_t);	Read one byte in I2C - Uint8_t slaveAddress - Uint8_t reg -> register address
static inline uint16_t read_word(uint8_t, uint8_t);	Read two bytes in I2C - Uint8_t slaveAddress - Uint8_t reg -> register address

Main.c

void timer_interrupt(void) __attribute__((interrupt(TIMER0_A0_VECTOR)));	This function is activated when the signal TIMER_A_VECTOR is activated

Interface (C++ language, developed in Qt Creator)

This folder contains building files, header files and source files. The main files are headers and sources.

headers

Contains the main functions of our code. This is the header list:

Interface.h

All the functions belong to Interface class. Functions are divided in public and private slots.

Public:

explicit Interface (QWidget *parent = 0);	Constructor function
~Interface ();	Destructor function
void ClearData();	Clear data function

Private slots:

void on_INICIO_clicked ();	Function activated when INICIO was clicked
void read ();	Reads the received data
void DataValues (QString &Data, int n, int &i, int j);	Split the amount data in different variables. - QString &Data -> String of amount data - Int n -> number of iteration to control variable value. - Int &i -> character selected of data array - Int j -> character selected of value array
QString dirx (QString x, qreal &mx);	Translates the value of X axis to know the direction. - QString x -> value of X axis - qreal &mx -> value of the movement This function is available if it is check "GAME" button
QString diry (QString y, qreal &my);	Translates the value of Y axis to know the direction. - QString y -> value of Y axis - qreal &my -> value of the movement This function is available if it is check "GAME" button
void diryNEW (QString y, qreal &my);	Translates the value of Y axis to know the direction. - QString y -> value of Y axis - qreal &my -> value of the movement
void dirxNEW (QString x, qreal &mx);	Translates the value of X axis to know the direction. - QString x -> value of X axis - qreal &mx -> value of the movement
void dirz (QString z, qreal &mz);	Translates the value of Z axis to know the direction. - QString z -> value of Z axis - qreal &mz -> value of the movement
void MoveBall (qreal w, qreal h);	Move the position of the ball - qreal w -> add weight value - qreal h -> add height value
void LimitBall (qreal &h, qreal &w);	Take care of not exceed the windows limit - qreal w -> add weight value - qreal h -> add height value

void on_PAUSE_clicked ();	Pause the collecting data when PAUSE is clicked
void on_GAME_clicked ();	Change between Step-tracker and GAME

Sources

Is where header functions are developed and can contain properly functions. In this part it is going to be shown the added functionalities to source code, because in header part it is already explained. This is the source list:

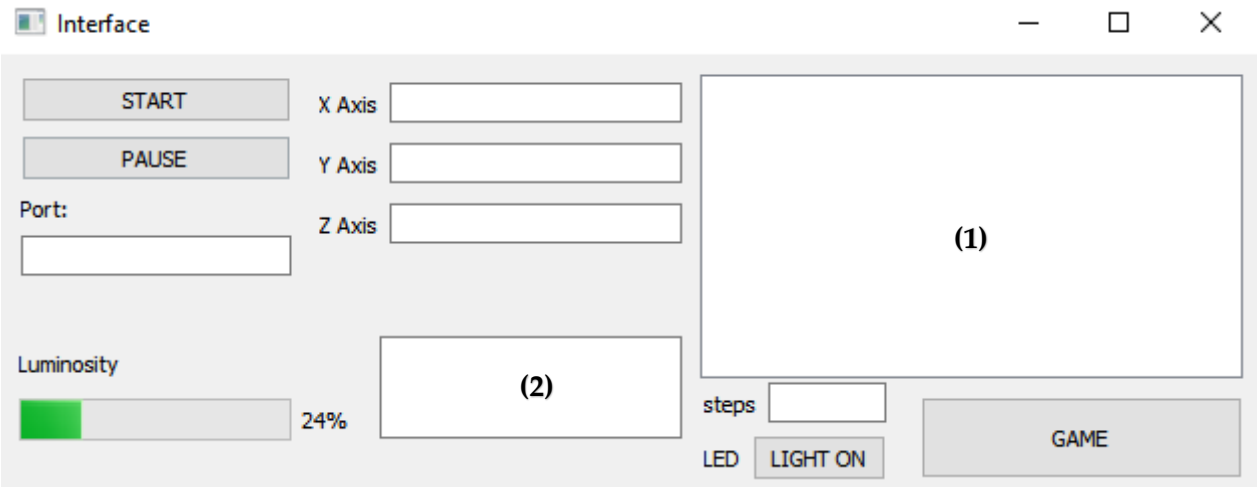
- **Interface.cpp**
- **Main.cpp**

Main.cpp

int main (int argc, char *argv[])	Initialize the main program

Images

This is the created interface:



- **START:** start interface.
- **PAUSE:** pause interface.
- **Port:** shows port number where microcontroller is connected.
- **Luminosity:** shows the measure of ambient light.
- **X/Y/Z Axis:** shows axis values.
- **Window (1):** is a simulation of LCD graphic. Shows a ball following the step-tracker movement. In step-tracker mode, it is enable when user do a simulation of watching the wearable, otherwise is not visible. If it changed to GAME mode, the ball receives movement based on stability of the sensor.
- **Window (2):** Advertising window. If you pause the interface shows "PAUSED". If you select "GAME" shows the direction of the ball.
- **Steps:** shows number of steps given.
- **LED (Is not a button):** Representation of a LED device. Is enable when ambient light is not enough and the simulation of watching wearable is done.

Future implementations

In the future I would like to implement a wireless connection instead of UART protocol. It could be interesting to implement and study the latency of communication.

Also, it is going to be implemented a Nokia LCD with SPI protocol, remembering the final goal of this project is create a complete wearable device which indicates you the position inside of an indoor space.

If a wireless connection is implemented, I would like to send maps via Bluetooth LE or radiofrequency.