LILLIAD: Connected learning center

Mageshwaran SEKAR
Department of Electronics and Computer Science
Polytech Lille

December 2015

Contents

Acknowledgement

This final year project (FYP) will not be a success without the willingness and commitment from a number of individuals. Thereby, I would like to thank everyone who assisted me during this project and a special thanks to the following person.

First of all, I would like to thank my tutors, Mr. BOE Alexander and Mr. VANTROYS Thomas who supervised me throughout the project. Throughout the project, they came out to help when we were stuck with problems. Secondly, I would like to thank the professor from my university, Mr. HOOGSTOEL Frédéric for helping me out during the conception of a database for the FYP.

I would like to extend a special thanks to my colleague ROCHE François with whom I worked in parallel to carry out important tasks especially the electronics parts.

1 Introduction

As a 5^{th} year student at Lille Graduate School of Engineering (Polytech'Lille), it is compulsory to carry out a final year project (FYP). Thus, I chose one of the Internet of Things (IoT) related project which is entitled ***Connected Learning Center*** since I find myself interested in this field.

This report is a description of the work done from September to December 2015. In the following chapter, details of the scope statement are given. Later on, I am going to explain the important tasks carried out during the FYP and give specific technical details. Finally, a conclusion is drawn from the experience.

2 Scope statement

2.1 Preamble

The library of University of Lille 1 is in the transformation phase to becoming LILLIAD Learning Center Innovation by 2016. To enrich the user experience, the new library will be equipped with multiple sensors (for temperature, air pollution, etc.) in large scale. However, it requires a lot of effort to implement and configure these sensors.

2.2 Problem statement

Since the sensors are implemented in large quantity, it is difficult and tiresome to reconfigure them one by one. And each sensor is configured in a different manner. Hence, we had to come with a solution that could automate the reconfiguration process.

2.3 Objective

The main goal of this project is to develop a user interface which allows the personnel of LILLIAD to use and maintain the deployed network of sensors. This interface should allow them to configure the sensors by uploading the required files to the main server. On the other hand, we will also be developing a reliable communication between the sensor boards and the server through radio communication (ex:ZigBee) and traditional Ethernet communication in case of wireless network failure.

2.4 Materials

These following materials are required for this project:

- Apache Web and MySQL database server
- Raspberry Pi 2
- Several ATxmega256A3BU cards
- Sensors for temperature, air pollution, luminosity and noise

2.5 Project phases

This FYP is divided into few important phases.

2.5.1 Reservation of time slot to roll-out configuration

During this phase of the project, we will be developing a Web site, which serves as a user interface where the admin of LILLIAD would be able to upload files that will be used later on for configuring the equipment (main board, daugtherboard, sensors, etc.) automatically through scripts.

2.5.2 Deploying configuration via network

In this phase, we are going to implement few scripts in Raspberry Pi 2 so that the configuration of mainboards (MB), daughter boards (DB) and daughter^{2}board (D^{2}B) could be carried out automatically. The result of the configuration (success or failure) will be passed to the main server so that the admin could verify it and take action in case of failure.

2.5.3 Programming the sensors

The sensors will be implemented in the D^{2}B cards and thus need to be programmed through *Program and Debug Interface* (PDI). On the other hand, the *Serial Peripheral Interface* (SPI) will be programmed to receive data from the sensors.

2.5.4 Collecting log and sensor data

The data from sensors will be sent in redundant: through SPI to the mainboard and through radio communication (ZigBee) to the master Raspberry Pi which in turn will redirect those data to the server.

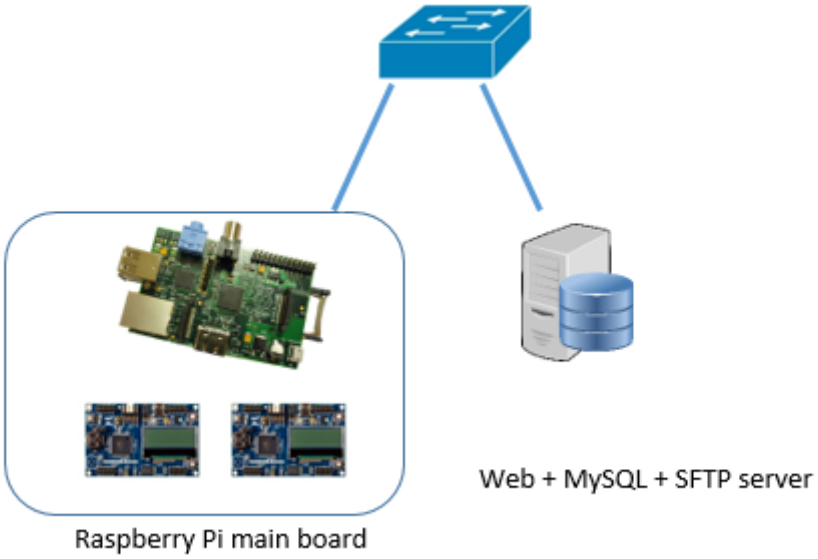

Figure 1: Partial architecture of the project

3 Implementation

During the period of September to December 2015, I worked mostly on the automatic configuration of the boards. At the same time, I worked in parallel with François ROCHE to configure few important parts of the board and carried out some basic tests.

3.1 Database

3.1.1 Analyse

First of all, I had to create a database which will be used to store all the important information regarding the files and the boards that will be used for the autoconfiguration. After having a few discussions with one of my professors, Mr. HOOGSTOEL Frédéric, we came up with a relational model for the database.

3.1.2 Relational model

The relational model of the database is based on the following logic:

- A user should have logged in before modifying the database
- A user can play more than one role (but only the admin can modify the database)
- A user can plan an autoconfiguration
- The autoconfiguration can be implemented in one or more boards but only with a single file
- The autoconfiguration is carried out during the reserved timeslot
- A board can be configured multiples times but during a different timeslot

This database model is represented in the figure below :

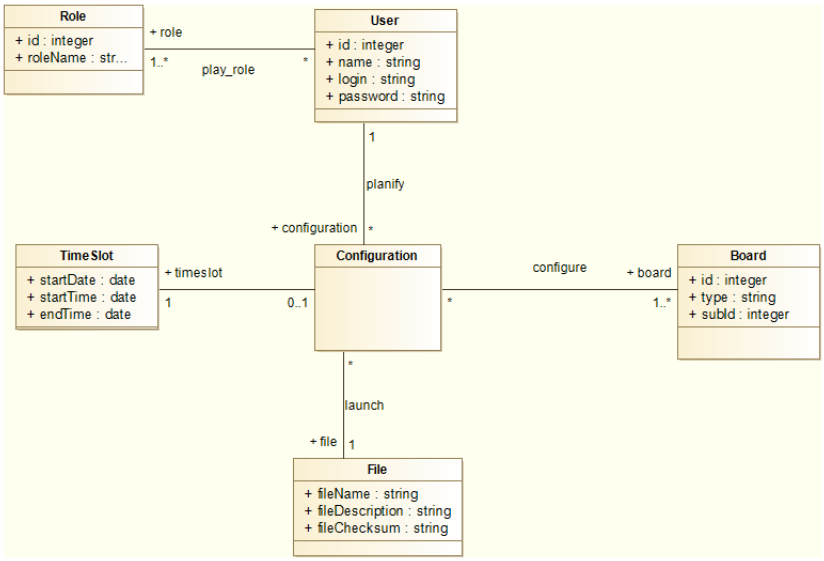

Figure 2: Database for autoconfiguration of boards

3.2 Website

The website will be used to upload files for autoconfiguration and determine the timeslot for the implementation. The website which is under construction is located here.

3.3 Automatic reconfiguration of boards

3.3.1 Configuration through PDI interface

The Program and Debug Interface (PDI) is an Atmel proprietary interface for external programming and on-chip debugging of XMEGA devices. It uses pin interface using the reset pin for clock input PDI_CLK and a dedicated data pin (PDI_DATA) for input and output. To program the ATXMEGA256A3BU cards, we connected the GPIO and ground pins of Raspberry to the PDI_CLK, PDI_DATA and ground pins of ATXMEGA respectively. Before uploading the file to the ATXMEGA card, we compiled a simple C program using avr-gcc. Then, we used PDI uploading program found at GitHub to configure the ATXMEGA256A3BU card.

3.3.2 Automatic file retrieval from the SFTP server

To implement the configuration, we have to be able to retrieve the files automatically from the server. To do this, I had set up a SFTP server from where the data files will be downloaded. To automatise this, I created a bash script (refer to annexe) where it will make a query to the MySQL database to retrieve the file name and the it will connect to the SFTP server and download the given configuration file.

3.3.3 Automatic configuration of mainboard

For the moment, we were able to reconfigure the mainboard. For this, we execute the script to retrieve file from SFTP server and execute it through another script (refer to annexe). This autocompile script will compile the C file, generate an hexadecimal file (.hex) and upload it to ATXMEGA through PDI. We were able to make a LED to blink automatically thanks to these scripts.

4 Future work

For now, we have finished implementing the main board. So, in the coming weeks, we have to work on the DB and D^2B boards to assure a two-way communication (on one hand, a PDI interface to be able to reprogram D^2B boards through the DB board and on the other hand, a SPI interface for sending the data from the sensors, connected to D^2B, up to the main board).

At the same time, we will be configuring Xbee modules (which will be integrated in the DB or D^2B board), to send data to the main server through radio communication. If this communication encounters problems, then the data should be sent in a downgraded mode (through serial communication).

Besides, we will be also working on the data collection from the sensors and sending them to the server through the network so that these values could be treated and displayed in a website. We have to distinguish differrent data from different sensors (of temperature, luminosity, etc.) so these values could be displayed appropriately.

Conclusion

One of the important part for the automatic configuration of the boards through the network, is the database. Although, we had a database course during our 3rd year in Polytech Lille, it was not sufficient to conceptualise the idea of this project. Thus, with help from Mr Frédéric HOOGSTOEL, I was able to come up with a relational model for the database.

On the other hand, I had to collaborate with François ROCHE who worked on the electronic aspect of the project. During this collaboration, we worked mainly on a proprietary programming interface, PDI.

Despite facing a lot of problems configuring PDI (due to the lack of documentation), we came out with a solution that enabled us to continue using PDI. Without the determination, we would not be able to solve the problem.

Annexes

Makefile

```
export CC = avr-gcc
export MCU = atxmega256a3b #ATXMEGA architecture
export TARGET_ARCH = -mmcu=$(MCU)

export CFLAGS =  -Wall -I. -DF_CPU=16000000 -Os
export LDFLAGS = -g $(TARGET_ARCH) -lm -Wl,--gc-sections
export PDI_BIN = /home/anon/PDI/bin/pdi #PDI binary
TARGET = prog

C_SRC = $(wildcard *.c)
OBJS = $(C_SRC:.c=.o)

CLK_GPIO = 7
DATA_GPIO = 8

all: $(TARGET).hex

clean:
        rm -f *.o *.hex *.elf *~

%.o:%.c
        $(CC) -c $(TARGET_ARCH) $(CFLAGS) $< -o $@

$(TARGET).elf: $(OBJS)
        $(CC) $(LDFLAGS) -o $@ $(OBJS)

$(TARGET).hex: $(TARGET).elf
        avr-objcopy -j .text -j .data -O ihex $(TARGET).elf $(TARGET).hex


upload: $(TARGET).hex
        $(PDI_BIN) -c $(CLK_GPIO) -d $(DATA_GPIO) -a 0x00800000 -F $(TARGET).hex
```

SFTP script

```
#!/bin/bash

DB_USER=root
DB_PASSWD=mypwd
DB_NAME=lilliad_rpi_conf
HOST=193.48.57.161
QUERY="SELECT fileName FROM file"
SFTP_USER=sftp
SFTP_PASSWD=sftppwd
cmd_list=""
read -ra results <<< $(mysql -h${HOST} -u${DB_USER} \
-p${DB_PASSWD} ${DB_NAME} -e "${QUERY}")

unset results[0] #remove array 0

for rows in "${results[@]}";do
  cmd_list+="get ${rows[0]}
"
done

lftp<<END
open sftp://${HOST}:619
user ${SFTP_USER} ${SFTP_PASSWD}
$cmd_list
bye
END
```

Auto compilation script

```
#!/bin/bash

strA='Compilation:success.'
strB='Upload:failed.'
strC='Upload:success.'
strD='Compilation:failed.Upload:aborted'
out=''

make clean
make

if [ "$?" -ne 0 ]; then
        echo $strD
        exit 1
else
        out=$strA
fi

make upload

if [ "$?" -ne 0 ]; then
        echo $out$strB
        exit 1
else
        echo $out$strC
fi
```

Sample C file

```
#include <avr/io.h>
#include <util/delay.h>

int main (void){
 int i;
 PORTB.DIR= 1;
 while(1){
  PORTB.OUT=0;
  _delay_ms(1000);
  PORTB.OUT= 1;
  _delay_ms(1000);
 }
}
```

References

[1] Unknown author. Bash scripting tutorial. http://linuxconfig.org/bash-scripting-tutorial.

[2] Unknown author. Basic Makefile for AVR-GCC. http://arduino.stackexchange.com/questions/12114/basic-makefile-for-avr-gcc.

[3] Unknown author. GPIO: Models A+, B+ and Raspberry Pi 2. https://www.raspberrypi.org/documentation/usage/gpio-plus-and-raspi2/.

[4] Unknown author. PDI programming. http://www.atmel.com/webdoc/avrdragon/avrdragon.pdi_description.html.

[5] Sandra HENRY-STOCKER. Unix: Flexibly moving files with lftp. http://www.itworld.com/article/2833203/operating-systems/unix--flexibly-moving-files-with-lftp.html.