JevonAlarmClock development report

Project ID: 1282

Jeroen Vennegoor op Nijhuis

June 23, 2015

This report provides indepth information on the JevonAlarmClock. JevonAlarmClock is an electronics project by Jeroen Vennegoor op Nijhuis and was developed for the 2015 Keil/Arm "ARM Microcontroller Design Contest". Online project archive: `https://bitbucket.org/jeroen82/jevonalarmclock`

Contents

1. Introduction

In April 2015 I got an Elektor newsletter in my inbox. It mentioned the "ARM Microcontroller Design Contest" organized by Keil/ARM. This sounded like the perfect opportunity to test and improve my technical skills in the field of electronics and embedded software. So I applied and my proposal was accepted.
This document describes the development of the JevonAlarmClock: an Internet connected smart alarmclock that can control a Philips Hue light in order to wake you up in the morning. The entire project with sourcecode is available at:
`https://bitbucket.org/jeroen82/jevonalarmclock`

2. Architecture

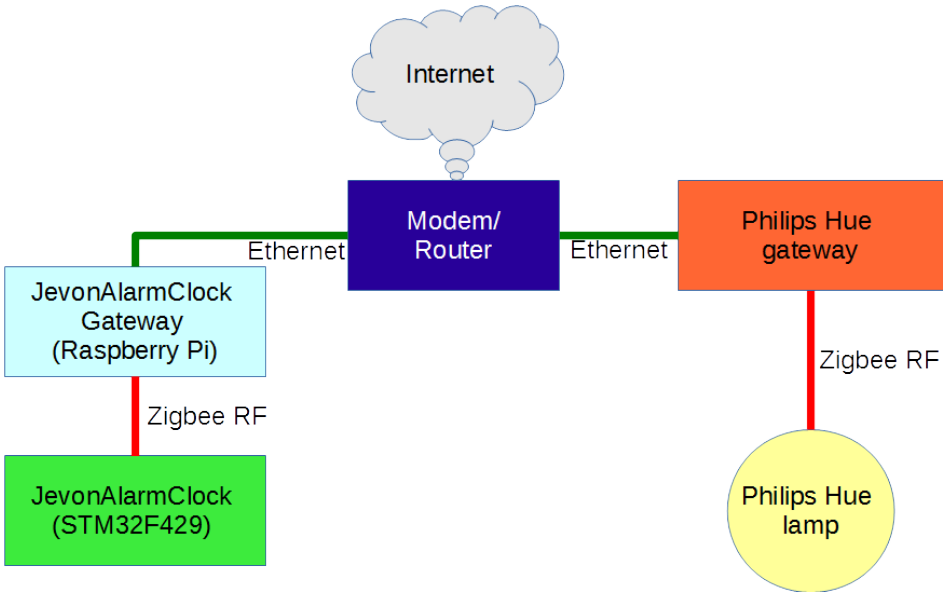

Figure 2.1.: System architecture

2.1. System architecture

JevonAlarmClock consists of 2 parts: the alarmclock itself and a gateway that connects the alarmclock to the Internet. The alarmclock wirelessly communicates with the gateway via Zigbee. The gateway itself is via Ethernet connected to the Internet and to the local Philips Hue gateway. The alarmclock can actually control 1 Philips Hue lamp. This is achieved by send a command to the gateway where it is translated to a HTTP REST call to the Philips Hue gateway. The basic functions of the gateway are:

- provide current time to the alarmclock
- translate light control commands from the alarmclock to HTTP REST calls to a local Philips Hue gateway.

It might seem somewhat odd that the JevonAlarmClock doesn't communicate directly with the Philips Hue lamp as both use Zigbee. Unfortunately the protocol between the Philips Hue gateway and the Philips Hue lamps is not "open" and Philips only allows access to the lamps via a REST interface on the Hue gateway.

3. Electronics

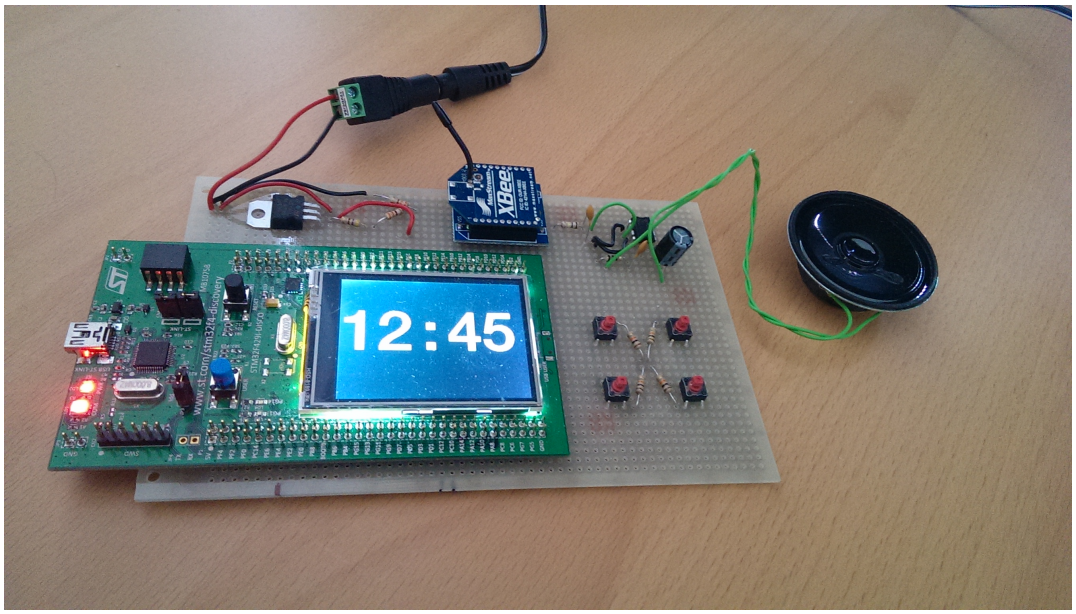

Figure 3.1.: JevonAlarmClock prototype: front

3.1. Introduction

The electronics for JevonAlarmClock has the STM32F429I-DISCO development board at it's core. Several components needed to be added to provide the following functionalities

- Power supply
- RF communication
- Audio output
- User input(buttons)

3.2. Power supply

The JevonAlarmClock uses a standard wallsocket power supply to provide a 5V DC input. To reduce this to the 3V required for the XBee, an LM317T is used to convert 5V DC to 3V DC. The LM317T is an adjustable voltage regulator: the output voltage can be configured by choosing the correct resistor values. The combination of a 470 Ω resistor and a 660 Ω gives an output of 3V.

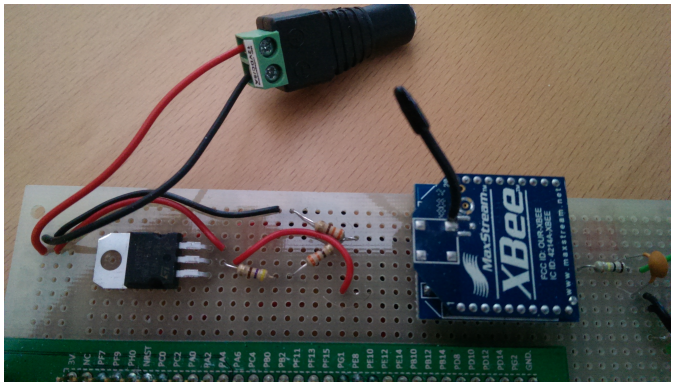

Figure 3.2.: Power suppply and Zigbee module

3.3. RF communication

For wireless communication between the JevonAlarmClock and the gateway, an RF module had to be added. In this case an XBee Zigbee module was chosen. The communication between the ARM microcontroller and the XBee is via UART (3V level) using only RX and TX at 9600 Baud. The XBee is by default in a "wireless serial interface" mode, so all the serial commands that are send to the XBee are automatically transferred to any other XBee in range. This is great for prototyping, but for an actual product the RF interfacing would need an overhaul.

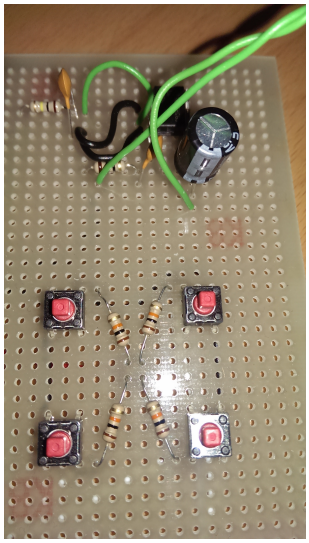

Figure 3.3.: Audio output and user input

3.4. Audio output

The STM32F429 doesn't provide any audio output functionality. In order to still create sound when the alarm goes off ,1 GPIO pin was configured as PWM output.An audio amplifier IC was added to drive a a small 8 Ω speaker.

3.5. User input

The JevonAlarmClock requires user input to switch the light on and off and to configure the alarmclock functionality. To enable this 4 pushbuttons have been added and they are connected to GPIO pins on the ARM microcontroller. These buttons don't have hardware debouncing, this will be taken care off in the software.

3.6. Schematics and breadboard

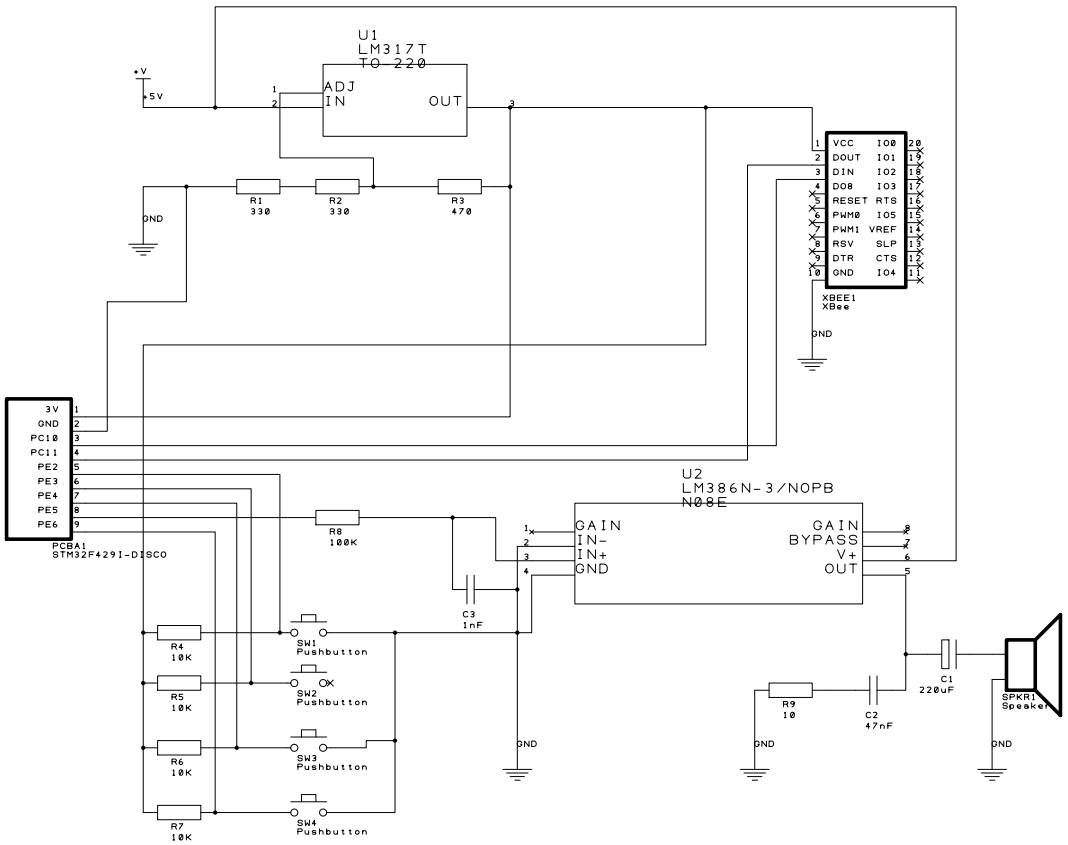

Figure 3.4.: Schematics

The schematics were produced using DesignSpark. The actual prototype was build using a perfboard. The STM32 development board was first attached to the perfboard and then all the peripherals were added. The wiring of the components was done using 0.2mm^2 multicore wire. The schematics of the JevonAlarmClock prototype can be found in figure 3.4. The full bill of materials can be found in appendix A.

4. JevonAlarmClock software

4.1. Tools

The embedded software for the JevonAlarmClock was developed using the Keil μVision 5 software. Additionally Eclipse Luna CDT was used for refactoring and formatting the sourcecode.

4.2. Architecture and design

The embedded software for the JevonAlarmClock uses the following frameworks and libraries:

- CMSIS-CORE
- CMSIS-Driver for UART and SPI
- CMSIS-RTOS
- Segger emWin for the LCD GUI
- STM32F4xx HAL drivers

The use of CMSIS RTOS(Keil RTX) enables the use of threads. The applications has the following user defined threads:

- Main thread
- UART thread
- GUI thread

The entire application is written in C and split into files per functionality.

4.3. Buttons

The 4 buttons are read using polling in the main thread. Debouncing is achieved by counting the amount of 'ticks' a button is pressed. The code also distinguishes between a short and a long press. The long press is used to increase the UP/DOWN stepsize when setting the alarm time.

4.4. Audio

The alarm sound is generated using the PWM functionality in the STM32 HAL library. The beep sound is just a simple PWM output at a fixed frequency that is being switched on and off. Switching the PWM output on and off is achieved using 2 timers: 1 to switch it on and 1 to switch if off.

4.5. Serial communication with XBee module

One thread is dedicated to UART communication with the Zigbee XBee module. The UART functionality is build upon the CMSIS UART driver. UART communication is using pins PC10 and PC11 for TX and RX respectively.

4.6. GUI

The GUI is updated using a dedicated GUI thread. The GUI functionality is build using the emWin library. The entire application uses only 1 dialog, in which a number of widgets are displayed. The exact contents of the GUI depend on the state of the program. There are 3 states defined:

- NORMAL: Default screen
- SETALARM: The screen to set the alarm time
- ALARM: The screen displayed when the alarm goes off.

5. Gateway software

5.1. Design and implementation

The Gateway software is written in C++ and uses a number of open source libraries to perform its job:

- Boost ASIO to enable access to the serial port for communication with the XBee module
  URL: `http://www.boost.org`
- Poco C++ libraries: used for the basic application structure , logging and various other functionalities.
  URL: `http://www.pocoproject.org`
- Curl: used as HTTP client to talk to the Philips Hue gateway
  URL: `http://curl.haxx.se/`

It was developed on PC running Ubuntu Linux and Netbeans as the main IDE. The XBee is connected to the Gateway hardware using the "XBee Explorer USB". This is a basic serial-to- USB converter. In Linux the device then appears as a "/dev/ttyUSB" device. The Gateway uses CMake as the build system. This allows for simple compilation of the software for the final target hardware: a Raspberry Pi. Because the Raspberry Pi is an ARM11 based platform, the software needs to be compiled again on the Raspberry Pi itself or crosscompiled using a crosscompiler toolchain on a Linux PC. See the next chapter for more info.

The Gateway software waits for a message from the alarmclock and responds with the required information. The following message types have been implemented:

- Time request: The Gateway receives a time request from the JevonAlarmClock and then replies with it's own system time(which is up to date thanks to NTP)
- GetLightState: The Gateway receives a GetLightState request from the JevonAlarmClock. It then sends out a HTTP GET request to the Hue gateway and relays this information back to the JevonAlarmClock.
- SetLightState: The Gateway receives a SetLightState request from the JevonAlarmClock including parameters that specify light settings. This information is then converted to JSON and using a HTTP PUT request send to the Hue gateway.

6. How to build your own JevonAlarmClock

6.1. JevonAlarmClock setup

- Build the electronics using the schematics and bill of materials as provided in this document.
- Install Keil μVision 5
- Get the application source code from
  `https://bitbucket.org/jeroen82/jevonalarmclock`
- Folder "2.AlarmClockSoftware" contains the Keil μVision project and all the required source code. Open the project, compile and flash the STM32F429 Discovery board.
- Connect the electronics to a external 5V power supply and you're good to go.

6.2. Gateway setup

To allow the Raspberry Pi to function as the gateway for the JevonAlarmClock, the software needs to be compiled for the Raspberry Pi. This can be cross compiled if you have access to a Linux PC, but in this case we will compile it on the Raspberry Pi directly. WARNING: this does a long time, so I hope you are not in a hurry.

- Go to http://www.raspberrypi.org and follow the instructions on how to install Raspbian on to the SD card.
- Connect your Raspberry Pi to the Internet using an Ethernet cable and boot it with the new Raspbian installation on the SD card.
- Several packages need to be installed in order to build the JevonAlarmClock gateway software on the Raspberry Pi. "Build-essential" and "git" package should be installed by default, but we need a few other debian packages as well. Log in to your Raspberry Pi and run the following command.
  `sudo apt-get install libboost-all-dev cmake libcurl4-gnutls-dev`
  Do the following to set GCC 4.7 as the default compiler
  `sudo apt-get install g++-4.7`
  `sudo update-alternatives -install /usr/bin/gcc gcc /usr/bin/gcc-4.6 60 -slave /usr/bin/g++ g++ /usr/bin/g++-4.6`

```
sudo update-alternatives -install /usr/bin/gcc gcc /usr/bin/gcc-4.7
40 -slave /usr/bin/g++ g++ /usr/bin/g++-4.7
sudo update-alternatives -config gcc
```
Choose GCC version 4.7 as the default.

- Create a new folder in /opt and change the owner: sudo mkdir /opt/jevonalarmclock sudo chown pi /opt/jevonalarmclock/
- Clone the JevonAlarmClock git archive in the home directory:
  `git clone https://bitbucket.org/jeroen82/jevonalarmclock.git`
- Go to the subfolder 3.GatewaySoftware
- Run "downloadexternal.sh" to download the 3rd party library Poco.
- Run "build.sh" to start compiling everything. This will take quite a while.
- If the local time on your Raspberry Pi is not correct , use the following command to set the timezone on your Raspberry Pi:
  `sudo dpkg-reconfigure tzdata` This is important as the JevonAlarmClock will just "copy" the local time from the Raspberry Pi.
- Go to '/opt/jevonalarmclock/bin/' and type:
  `./Application`
  This will start the application. If you now power up JevonAlarmClock it should directly get the correct time from the Gateway application. If you would like additional debug logging, use the following to start the application:
  `./Application debug`

- In order to start the application when starting your Raspberry Pi, add the following code the the /etc/rc.local file before `exit 0`:

```
cd /opt/jevonalarmclock/bin/
sudo -u pi ./Application
```

7. Conclusion and recommendations

The development of the JevonAlarmClock took more time then I originally anticipated, but it was an interesting journey. The development of the embedded software for the ARM Cortex was fairly new for me, as I was more accustomed to C++ for desktop applications.
I used the STM HAL library for a number of functionalities (e.g. RTC and PWM). There is basic API documentation for this library, but the amount of sample code is very limited. It took me quite a while to figure out to exactly use it.

In order to actually use the JevonAlarmClock, I would need to "replace" the wallmounted light switch in my bedroom with a RF based switch that actually sends a ON/OFF command to the Philips Hue gateway.
Ofcourse it would also need a cool housing.
There are numerous ways in which the JevonAlarmClock can be improved:

- The RTC(RealTime Clock) is not very accurate at the moment, because I don't use an external 32kHz crystal. I was planning to add an external 32kHz crystal, because then it won't be necessary to sync every few minutes with the "Internet time" on the gateway.
- Add a photo resistor to measure the light intensity in the bedroom and then adjust the LCD screen brightness accordingly.
- Redesign the audio output peripherals to create a more "sophisticated" alarm sound then a simple beep.

Appendices

A. Bill of materials

Table A.1.: Bill of materials JevonAlarmClock

Quantity	Unit	Description	Type/specification	Brand
1	pc	STM32F429 Discovery : development board	STM32F429I-DISCO	STM
1	pc	Voltage regulator	LM317T	STM
1	pc	Audio power amplifier	LM386	TI
1	pc	Zigbee module	XB24	MaxStream
1	pc	XBee Adapter Board	32403	Parallax
2	pc	Resistor	330 Ohm	
1	pc	Resistor	470 Ohm	
1	pc	Resistor	100K Ohm	
1	pc	Resistor	10 Ohm	
4	pc	Resistor	10K Ohm	
1	pc	Ceramic capacitor	1 nF	
1	pc	Ceramic capacitor	47nF	
1	pc	Electrolytic capacitor	220 uF	
1	pc	Speaker	K 50 (8 Ohm)	Visaton
1	pc	5V power supply		Voltcraft
1	pc	Mini USB cable		

Table A.2.: Bill of materials Gateway

Quantity	Unit	Description	Type/specification	Brand
1	pc	XBee Explorer USB	WRL-11812	SparkFun
1	pc	Zigbee module	XB24	MaxStream
1	pc	Raspberry Pi	model B	
1	pc	5V power supply USB		
1	pc	Micro USB cable		

B. About the author

Jeroen Vennegoor op Nijhuis is a Dutch engineer. He lives in Eindhoven in the south of the Netherlands. He studied at the University of Twente and has a MSc degree in Mechanical Engineering. His professional interests and skills now range from mechanical design to electronics and software. His professional experience has been mostly in R&D and the development of new products.
For more information:
LinkedIn: `https://nl.linkedin.com/in/jeroenvennegoor`