

Research Article

A Low-Cost Wireless Interface Linking a Microcontroller to a Microcomputer Server

Ivano Folgosa¹ and Peter Excell^{2,*}

¹Siemens Mobility GmbH, Hong Kong

ivano.folgosa@gmx.de

²Wrexham Glyndŵr University, Wrexham, UK

p.excell@glyndwr.ac.uk

*Correspondence: p.excell@glyndwr.ac.uk

Received: 25th January 2020; Accepted: 7th March 2020; Published: 1st April 2020

Abstract: The development of a low-cost solution for the connection of wireless sensor nodes to the Internet is presented. The solution has the additional benefit of extremely low power consumption and hence very long battery life and low maintenance cost. The core of the solution is the Atmel ATtiny series of wireless transceivers and the designs of minimised printed circuit boards for three versions of the node are presented. To provide the fixed node at minimum cost, the Raspberry Pi microcomputer was used, attached to the Nordic Semiconductor nRF24L01+ transceiver module. Software to implement the system in each of the nodes was developed from zero-cost sources, the software being tailored for the particular application and devices selected. A World Wide Web interface for the Raspberry Pi was created, having an intuitive display and offering functionalities relevant for general automation tasks, data display and manual control, in conjunction with wirelessly connected clients. The Web interface can serve as the basis for a home automation system, a topic which is becoming more and more important and popular nowadays, but industrial applications are equally feasible.

Keywords: *ISM wireless band; Low-power Transceiver; Raspberry Pi microcomputer; ShockBurst wireless functionality; Web interface; Wireless sensor network*

1. Introduction

The Internet of Things (IoT) will require ubiquitous networking of a range of devices, linked to communication hubs: in general, the great majority of such links will be wireless. These links must be realised at minimum cost due to the commoditisation of IoT devices. An investigation was thus undertaken to develop a low-cost wireless interface between a standard microcontroller and a current low-cost microcomputer (Raspberry Pi) acting as the communication server. Additionally, various software programs for the implementation of a Web interface were investigated and the results were implemented in the design phase of this project. Application scenarios which could be

covered include: home automation and security; accumulation of environmental data; automatic test equipment; basic SCADA applications; general controllability of actuators and other high power devices.

The background design research was therefore divided into two overall topics, hardware and software. Hardware research was subdivided into the important components required: the microcontroller; the wireless component and the Raspberry Pi. Software research focused on the software required to operate the chosen hardware devices and was therefore mainly conducted after completion of the hardware design. However compatibility and integration aspects between the hardware and software had to be taken into account. To maximise adoption, all software packages used had to be issued license-free, for example with the GNU General Public License (GNU GPL) [1] or a Copyleft license [2].

2. Available solutions

Several solutions for wireless communication can be found online, with variations in respect to the utilized transceivers, microcontrollers and their intended purpose. A particular solution which had a substantial coverage of the hardware requirements was identified [3]: this covered general wireless connectivity using an inexpensive transceiver module directly connected to a Raspberry Pi microcomputer. This scheme was adapted for the present purpose as described in more detail below. However, no existing generic Web-interface software to control microcontrollers could be found and hence this became the main software development focus.

2.1. Available Technologies

Several wireless connection methods are available, with a range of prices and functionalities. These can be categorized by operating frequency and their network performance and capabilities. Devices in the 2.4 GHz unlicensed ISM (Industrial, Scientific and Medical) Band include those using WLAN, Bluetooth, XBee and ZigBee. Devices in other frequency bands include X10 devices, generic 433MHz receivers and transmitters but also ZigBee and XBee devices operating at 868MHz (permitted in Europe).

Two well-known microcontroller manufacturers are Atmel and Microchip. Due to the wide range of functionalities and the established support network, Atmel microcontrollers were the preferred option for this investigation. These microcontrollers have, in general, several integrated hardware interfaces such as I²C and SPI to communicate with peripheral devices, they are widely available and, depending on the complexity of the microcontroller, also offer a good cost/performance ratio.

As the communications server, the Raspberry Pi microcomputer is an outstanding choice at the present time. Connecting a wireless module to this can either be done via the I²C, SPI or UART interface, depending on the interface of the wireless module: the Raspberry Pi offers such interfaces via its GPIO (general purpose input output) pins.

2.2. Web-based User Interface

The main development focus was to create a flexible modular Web site which could be used to interact with remote boards via relevant control elements on the site (e.g. switch a device on/off via a button) and display data received from these boards. The Web interface was furthermore required to

be application-oriented, for example, if one board is configured as a temperature sensor the interface should provide temperature logging and plot functionalities. Another key desired aspect of the Web-interface was the compatibility with various end devices.

Several different ways to develop an appropriate Website exist, such as "Content Management Systems" (CMS) and so-called "Web Frameworks" or the simplest solution of a plain html site. To implement a degree of dynamic interoperability between the Web interface and the program controlling the wireless module the programming language that offered maximum functionality was Python.

3. Design

The design phase consisted of two major elements. The first was the hardware design of an inexpensive microcontroller board which connects wirelessly to a Raspberry Pi. The second element was software design which included the programming of the microcontroller board and the Raspberry Pi to communicate with each other, plus the Web-interface on the Raspberry Pi to display data and control the microcontroller.

3.1. Hardware Requirements

The following hardware requirements were defined for the microcontroller and RF device: low price; low energy consumption; small form factor; communication via SPI, UART or I2C; operating voltage between 3.3V and 5V; several different inputs and outputs (analogue and digital).

3.1.1. Selection of wireless module

The main selection criterion for the RF module was the price. Solutions such as WLAN, ZigBee/XBee and Bluetooth offer a broad set of functionalities such as direct addressing and the setup of different network types (star, mesh); but they are also the most expensive at the present time. Cheaper solutions are generic 2.4GHz RF transceiver modules and 433 MHz receivers and transmitters. Although compact 433MHz transceivers exist, they are expensive compared to the price of a single receiver and transmitter which, on the other hand, are fairly large due to the required antenna.

The wireless module with lowest cost that was found was one equipped with the nRF24L01+ transceiver IC. It offers connectivity to a MCU via the SPI interface (with 5V tolerant inputs), low power consumption, power saving functionalities, operating voltage between 1.9V and 3.6V and a small form factor with header pins for connection to SPI and power supply. It also operates in the 2.4GHz ISM band and therefore does not require any additional licensing.

A further notable feature of this module, as listed in the datasheet [4], is a proprietary functionality called "Enhanced ShockBurst" which provides the following features: 1 to 32 bytes dynamic payload length; automatic packet handling; automatic packet transaction handling; 6 data pipe MultiCeiver™ for 1:6 star networks [4].

3.1.2. Selection of microcontrollers

A microcontroller selection tool is provided on the Atmel homepage [5]. The criteria set in the tool were: flash memory size between 8Kb and 16Kb; pin count between 6 and 28; CPU 8-bit AVR; operating voltage between 1.8V and 5.5V. The result showed seven microcontrollers, all from the "ATtiny" family. Five of these were available in DIP package format and two, the ATtiny84 and the ATtiny85, were significantly cheaper. The smaller 8-Pin ATtiny85 offers six configurable I/O Pins, an

integrated temperature sensor, 8Kb of flash memory and an SPI compatible serial interface (Universal Serial Interface).

The larger 14-Pin ATtiny84 has the same characteristics as the ATtiny85 except that it offers 12 configurable I/O Pins. The ATtiny85 is intended for simple battery-operated tasks such as remote sensors. The larger ATtiny84 is intended for tasks requiring more I/O Pins, such as actuator and sensor control, hence development proceeded embracing both of these devices.

3.1.3. PCB Board Design

Prototype PCBs were designed for cost evaluation using the minimum amount of parts required for operation. As two MCUs were selected for different types of applications, different schematics and board layouts were developed: two smaller boards for the ATtiny85, one for battery (coin cell) operation (Figure 1) the other powered via USB (Fig. 2), and one large board for the ATtiny84 (Fig. 3) were designed.

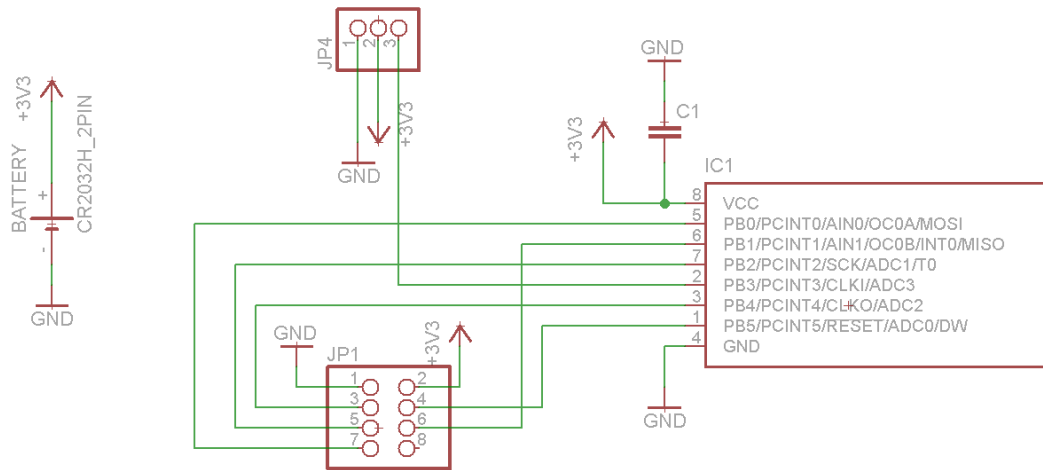


Figure 1. ATtiny85 PCB Schematic (Battery powered)

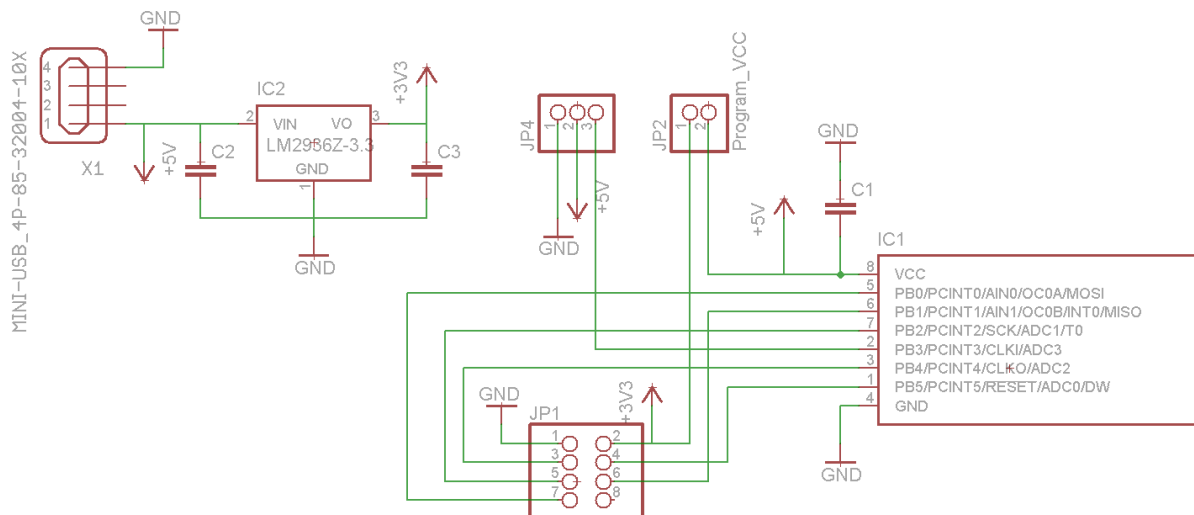


Figure 2. ATtiny85 PCB Schematic (USB powered)

As illustrated in Figure 3, each free I/O pin of the ATtiny84 board was routed together with +5V and GND to separate 3-way header pins. This offered increased connectivity with sensors and other elements requiring +5V power supply and ground pins. As power supplies for USB-type devices are now widely available, the design included a USB Type B connector to accommodate this.

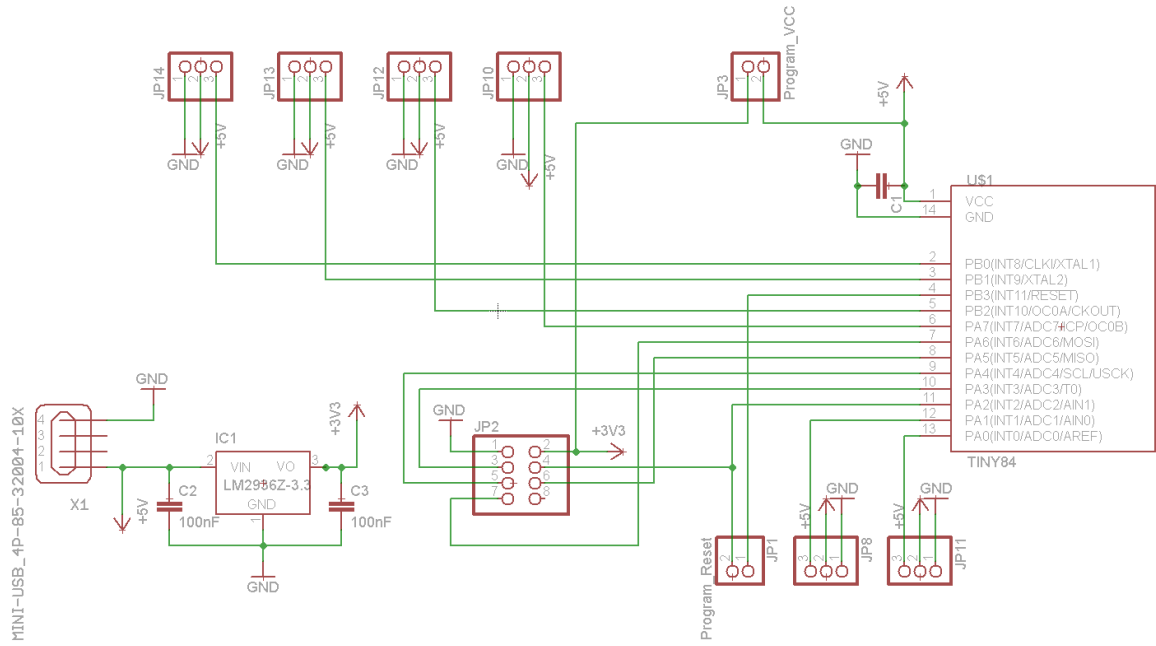


Figure 3. ATtiny84 PCB Schematic (USB powered)

It was important that the nRF24L01+ module was supplied with an appropriate input voltage. For the USB powered boards additional voltage regulators were therefore required to supply the RF module with +3.3V. The battery supplied board did not require any voltage regulation as both the MCU and the RF module can be operated with a 3V coin cell.

For all boards, a female 2x4 header was used to connect the nRF24L01+ module. As the SPI interface uses the same I/O pins as the ISP programming interface the same header can be used with a small adapter to program the microcontrollers via any ISP programmer. For the ATtiny84 board additional jumper headers were inserted to connect the reset pin of the MCU with the respective pin on the RF header. This was not implemented for the ATtiny85 board due to the I/O pin limitation and it was therefore necessary to activate the external reset disable fuse (RSTDISBL) while programming the device. Reprogramming can then only be performed with a High Voltage Serial Programming device. The USB-type boards include an additional power supply jumper header which connects the +5V from the ISP programmer via the +3.3V of the RF header with VCC of the MCU.

Based on the designed PCB boards all defined parts and their prices were evaluated. The total costs showed that the design of the boards was cost efficient and that a simple wireless sensor node could be manufactured for less than \$4.

3.1.4. Connecting the nRF24L01+ module to the Raspberry Pi

The nRF24L01+ module does not require any additional circuitry to be operated with a Raspberry Pi; it can be wired directly to the respective GPIO pins. The Serial Peripheral Interface (SPI) pins (MOSI, MISO and SCK) on the Raspberry Pi are hardware defined and cannot be changed; only the CE (Chip enable) and CSN (Chip Select) pin required by the nRF24L01+ module to switch between receiver/transmitter and send/receive data or commands can be selected freely.

3.2. Software

In order for a MCU or Raspberry Pi to communicate with the nRF24L01+ via SPI, both MCU and Raspberry Pi have to be enabled for SPI communication. Within the active Arduino community, libraries have been created to operate the nRF24L01+ modules with an Arduino MCU. These libraries have also been modified to be used with a Raspberry Pi. For test purposes such a library [6] was used for initial testing of the communication between Raspberry and MCU (Arduino Nano). At a later stage code was written in C for the defined MCUs, based on information in Reference [7].

3.2.1. Raspberry Pi: Python Code for SPI communication

As a flexible way of achieving SPI communication with the Raspberry Pi, Python was used to interact with the nRF24L01+ module. The installation procedure as well as basic Python code to communicate with the module is available online [3]. The setup of the Python code was similar to that for the Arduino libraries. Firstly all registers and configuration values were defined and then functions were defined to operate the module, e.g. read/write to a register, receive/transmit payload.

The figure 4 shows an example of how the Python code works.

```

SET_ACK = 0x01
EN_AA = 0x01
WRITE_REG = 0x20

def doOperation(self,operation):
    time.sleep(SMALL_PAUSE)
    toReturn = self.nrf24.transaction(operation)
    return toReturn

bytes = [WRITE_REG|EN_AA]
bytes.append(SET_ACK)
self.doOperation(writing(bytes))

```

Figure 4. Python code example

The first 3 lines define the settings value (SET_ACK; Set Acknowledgment Byte), the register (EN_AA; Enable Auto Acknowledgement) and the write command (WRITE_REG; SPI command of the RF module). The subroutine “doOperation” which transmits the data to the nRF24L01+ module is called in the last line after the write command; the register and the value are appended to the variable “bytes”. This code extract shows the basic principle of communication with the RF module using Python.

3.2.2. The nRF24L01+ Module: SPI Commands, Registers and Addressing

The communication with the nRF24L01+ module is established using the commands given in its specification [4]. The most important commands are: “R_REGISTER” to read a register; “W_REGISTER” to write to a register; “R_RX_PAYLOAD” to read the received payload; “W_TX_PAYLOAD” to write (send) the payload; and “R_RX_PL_WID” to read the payload length (dynamic payload only).

To read from a register of the nRF24L01+ module the CSN pin is first set to LOW and the Read-Command byte is transmitted via the MOSI pin to the RF module. The RF module always responds after the CSN pin level changes from HIGH to LOW by sending back its Status Register (S7-S0), if commands were transmitted the Status Register data is followed by the respective registers data via the MISO pin. To write to a register the CSN pin is first set to LOW then the Write-Command is transmitted via the MOSI pin followed by the data to be written to the respective register. For both reading and writing the SCK pin provides the clock rate for bit transfer. The order

of transmission is LSByte to MSbyte and MSbit to LSbit. This byte order had to be taken into account for the deciphering of the received and transmitted messages.

To configure the nRF24L01+ module, for example to operate with 250kbps, the respective bits in the relevant register have to be set. The register to set the data rate is RF_SETUP which has the register address "06". In this register the bits 3 (RF_DR_HIGH) and 5 (RF_DR_LOW) are used to set the data rate. All RF modules were configured identically except for the RX and TX addresses.

To communicate with each other, the nRF24L01+ modules have up to six receiver and one transmitter address, depending on the activated data pipes. To successfully transmit data, the TX address has to be identical to the RX address of the module which is supposed to receive the data. The Raspberry Pi, acting as the server, listens to messages on five different addresses which, if required for improved overview, can be grouped by the user for specific applications. For example all environmental sensors send data to data pipe 1, application specific devices use data pipe 2 and so on.

The address functionality included a device ID byte in the available payload and additionally through a device database on the Web-interface. This requires each device to be programmed with a unique device ID which can be used by receiving devices as an extra verification that the correct device has received the message, but also for messages sent from the Raspberry Pi to devices using the same RX address. In such a case each device receives the payload but only the specified device responds accordingly. More importantly it is used by the Raspberry Pi to identify from which device a message has been received in order to process the data correctly.

3.2.3. Web Interface

The underlying program of the Web-interface is a Python-based Web framework, called "Bottle" [8]. It has an integrated Web server, which simplifies the installation process, and provides routing and general HTML functionalities such as HTML forms (input boxes and buttons) and file access, as well as the ability to integrate other Python modules (such as Pygal for data plotting [9]).

Access to Web-pages is based on routing functions. If, for example, the IP address of the Raspberry Pi is entered in a Web browser together with the corresponding port of the Web-framework, a specific function for this exact URL is called and returns the defined data or a HTML page. Further benefits of this framework are that data can easily be sent to HTML files which can also include Python functions for data handling, formatting and display.

3.3. MCU Software

Additional code implemented for the MCUs was limited to demonstration purposes and included functionalities to cover the most relevant basic operation modes. This included: Analogue reading; Pulse-width Modulation (PWM); Digital input and output.

The main software functions required to operate the device in conjunction with the RF modules were the communication function, utilizing the Universal Serial Interface (USI) of the microcontrollers for SPI communication with the RF module; functions to read and write to the RF modules' registers; the initialisation function of the RF module to set the configuration of the RF device, and finally functions to read and transmit payload data. The transmission of the temperature value measured by the integrated temperature sensor was also integrated as a standard function.

As soon as the device is powered-up, the initialisation of the MCU is started, followed by the RF module. The MCU then enters a "receive and transmit" loop: the device constantly awaits data input

on its respective data pipe (address) by checking the RF module's status register. If the relevant bit is set in the status register the MCU retrieves the data and performs whatever application is encoded with the payload.

Figure 5 illustrates the general program flow of the microcontroller's software. This example applies for constantly-powered devices awaiting data and transmitting data if required. Battery powered devices are modified to preserve battery life by entering "power down" modes and mostly being restricted to transmission of data only.

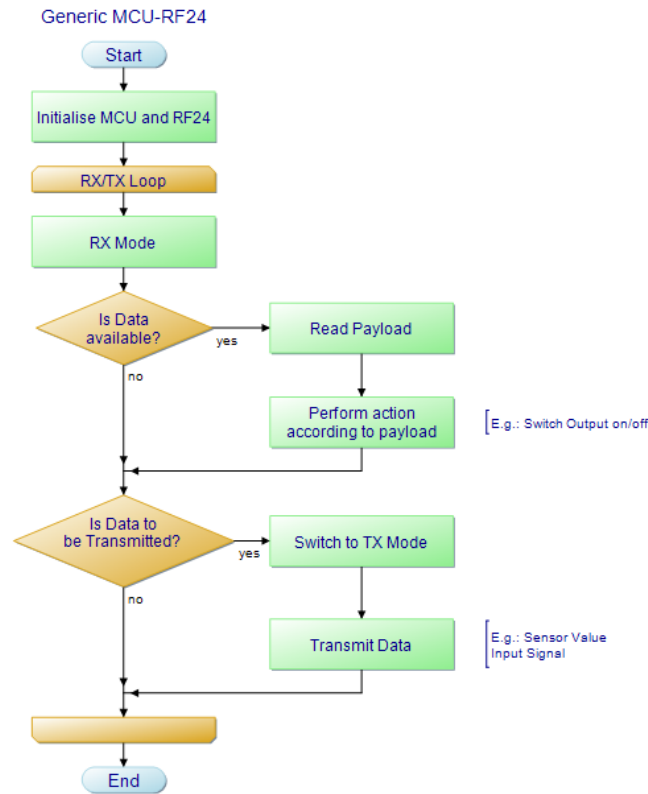


Figure 5. Generic MCU - RF24 Program Flow

3.3.1. Raspberry Pi RF Software

The Python based script, obtained from [3], to interact with the RF module offered general transmission and receiving functions, however it had to be extensively modified, especially in conjunction with the Web-framework.

Major modifications included: Message queue for interaction with Web interface; Automatic RX/TX mode switching; Automatic TX address switching with access to device database; TX prioritisation (through queue); Additional Retransmission if transmission failed; RX/TX Logging with timestamp; Message formatting.

The script is executed as a thread of the Web-interface script as soon as that is started. This ensures that both scripts can operate simultaneously and parallel to each other. Utilizing the queuing function both scripts can send and receive data from one another.

The overall program flow is illustrated in Figure 6 and shows similarities with the previously described MCU program flow. The RF24 module is initialized after the script is executed. It then enters a loop, continuously checking for new data as long as no message is in the TX queue. If a message is available in the TX queue the script finishes receiving data and switches to TX mode, checks the database for the device ID-specific TX address and changes its address accordingly if

necessary. Then the data is retrieved from the TX queue and transmitted. If data transmission was successful the data is stored in the log file together with the time and data. Should the status register show a failed transmission the message is retransmitted. If the message was unsuccessfully retransmitted a fourth time the retransmission loop stops and the RF24 returns to RX mode unless further data is available in the TX queue. In RX mode the status register is continuously checked to see whether new data is available (Bit 6, RX Data Ready is HIGH). If the respective bit is set, the payload is read from the RX FIFO register. As the data is read from the most significant to the least significant byte, the message order is reversed, saved to the log file with time and date and the data is moved to the RX queue. The RX loop is repeated until a message is to be sent.

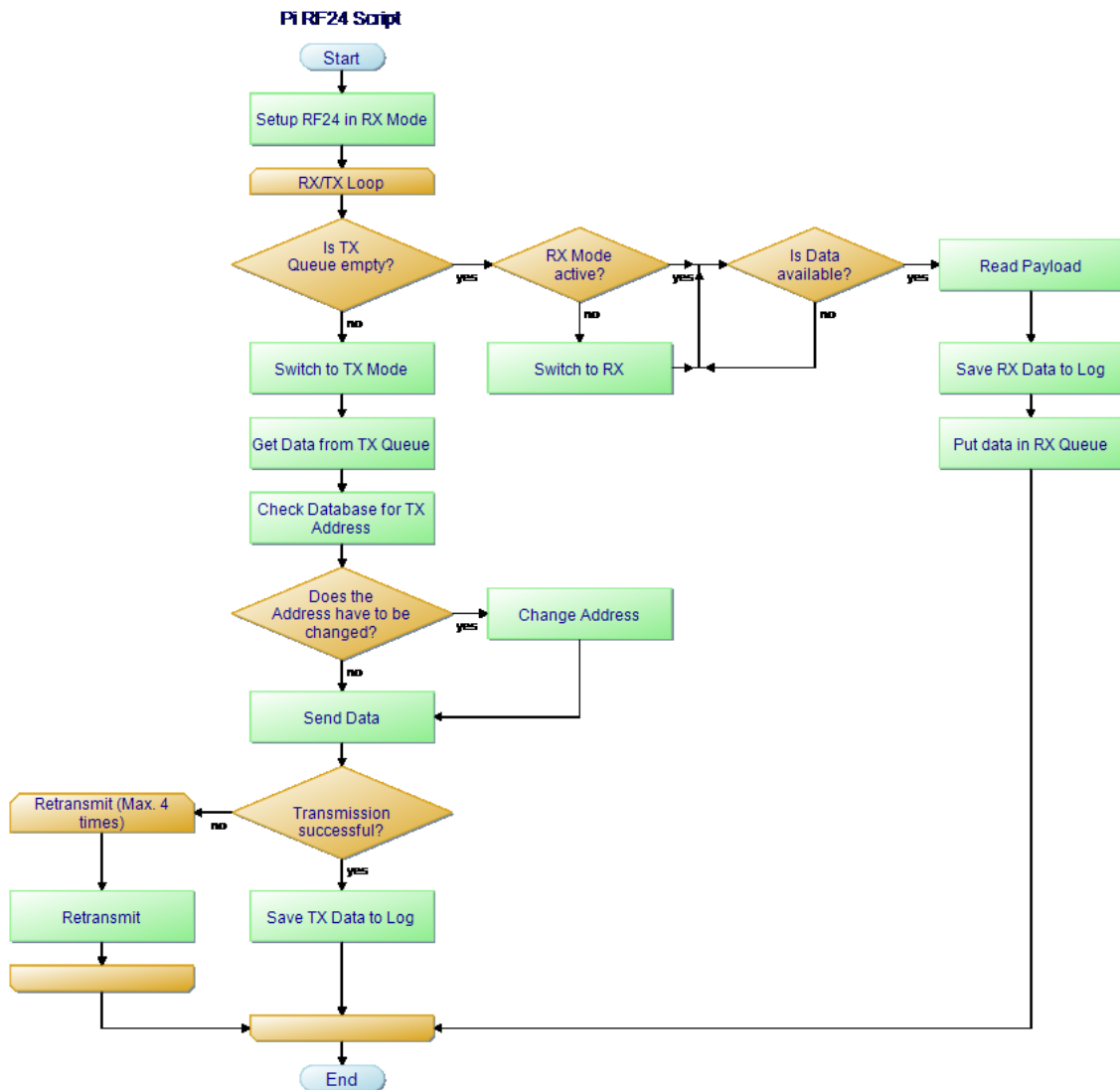


Figure 6. Raspberry Pi RF24 Overall Program Flow

4. Development of the Web-Interface

The development of the Web-interface consisted of the implementation of the RF Python script, defining a basic template for the Website and the included pages as well as developing applications controllable via the Web-interface.

Development proceeded with the Web-framework "Bottle" [8]. The overall program flow is depicted in Figure 7. Upon execution of the Web-framework the integrated Web server is started,

followed by the threaded scripts (RF script and an application subprogram). The Web-framework then waits for Web requests issued via a remote computer through a Web browser. If a request has been received the specific sub function (or sub routine) allocated to the requested address (URL) is executed. After the code in the sub function has been executed, the Web-framework returns a defined Webpage to the requesting computer.

If the request results in data to be transmitted the relevant data in the sub function is assembled in the payload frame and sent to the transmission queue. At this point the RF script is aware that data is available in the respective queue, the data is retrieved and the transmission is initiated.

The rate at which a Webpage is returned to the requesting device depends on the functions being executed in the subroutine. For example a plotting function requires more time, as the data has to be read and a picture file created with the plotted data.

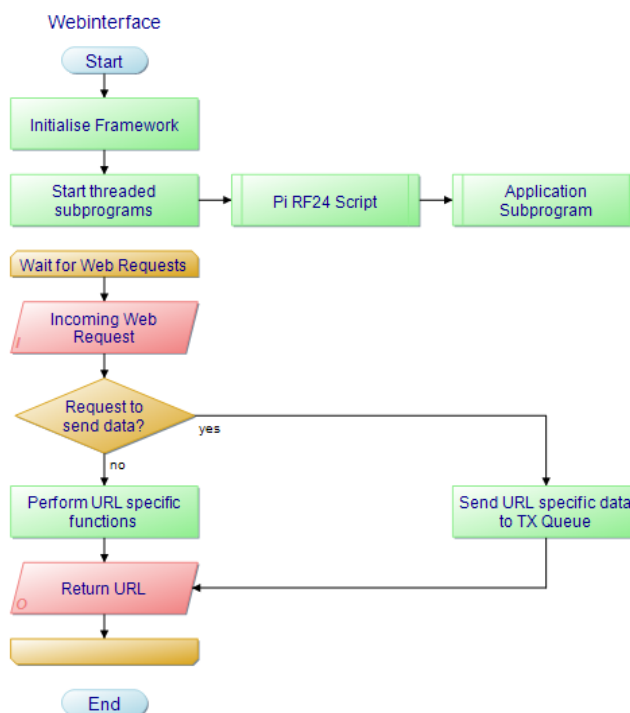


Figure 7. Overall Web-Interface Program Flow

4.1. Implementing the RF Script

To implement the Python script controlling the RF device a method had to be found to run both the RF script and the Web-framework in parallel and for it to process received data and data to be transmitted after a request via the Web-interface was issued. A suitable method, using the Python modules "threading" and "queue", was found, but the implementation was not possible as both scripts had to operate with the same Python version. As the RF script and the underlying Quick2Wire Python API [9] required Python version 3 (or higher) and the Web-framework Web.Py [10], initially used for development, used Python version 2.7 without upward compatibility, the Web-framework "Bottle" was selected instead [8]. The threading module enabled both scripts to operate at the same time and data could be transferred between the scripts via the queue module.

The main functions added to the RF script included database access to change the TX address of the RF device depending on the device ID included in the message, switching between RX and TX

mode and logging of received and transmitted data in a CSV format, which can be downloaded via the Web-interface. Typical logged data is shown below and consists of a TX/RX identifier, the time the message was received or transmitted, the date, the status register, the individual defined frame bytes [Device ID, Frame & PID, Data/Command ID] and the rest of the payload:

RX,13:54:55,04/03/14,42,10,40,14,0000000000000914

The frame bytes are used to include a unique device ID byte, two frame type bits defining whether it is a data or command package, and a Packet ID which can be used for identifying data divided into several messages and for assembling the messages back together.

4.2. Creating the Website

Using the built-in functions and HTML code an initial page layout was created and remote accessibility was ensured. The overall page layout was divided into three frames. The top frame displays the main navigation bar from which all main menus (Application pages) can be accessed. The left frame shows main-menu specific links. The larger frame is used to display the content of individual menu pages. The overall layout and the start page are depicted in 8.



Figure 8. Start page of the Web Interface

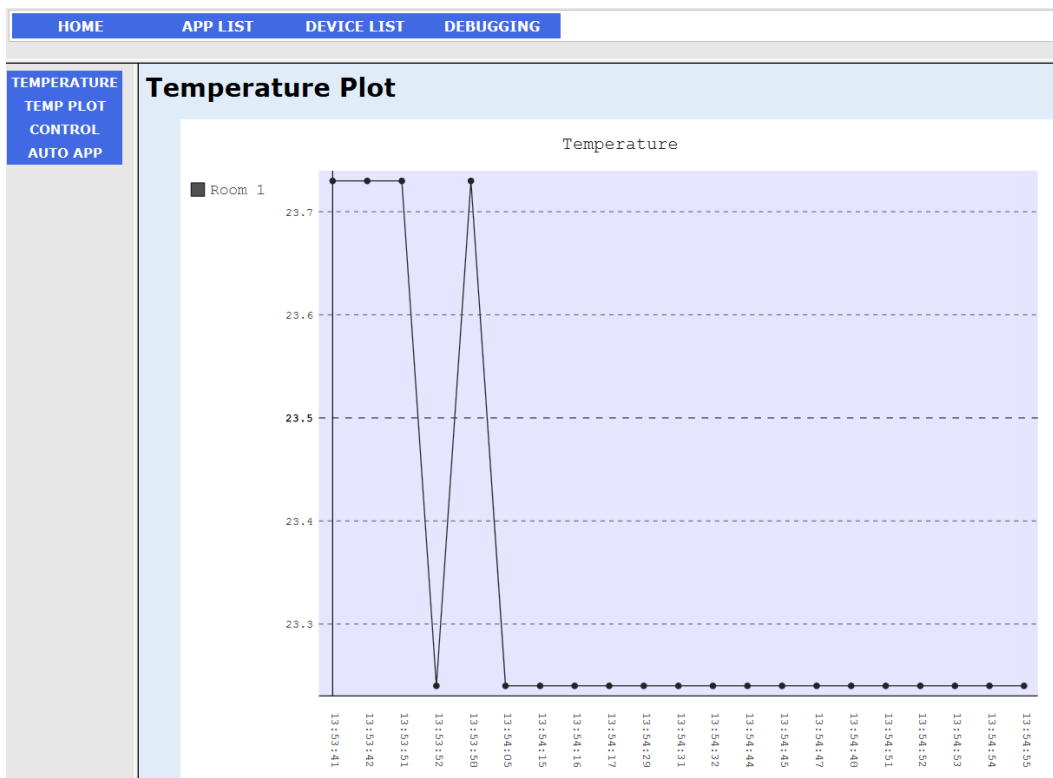


Figure 9. Example of a temperature plotting page

Additional features and modules were integrated into the overlying Web-interface throughout the development process. The following figures describe common applications scenarios which are applicable for automation processes in general, static information display and manual control of devices which were added to the Web-interface.

An example of a static information display, as shown in Figure 9, is created by displaying several values of a device in a plotted diagram. Upon issuing the request for this Webpage a certain number of values last received from a defined device are looked up in the log file and a picture file with the plotted diagram is returned. In order to create such data plots the Python module Pygal [9] was used.

A manual control application is shown in Figure 10. The Webpage contains several HTML form objects, buttons to switch devices on and off, and a range slider (to dim a LED in this example). Upon issuing the request for this Webpage the button pressed is identified and the specific subroutine is executed which contains the data to be moved into the TX queue for transmission.

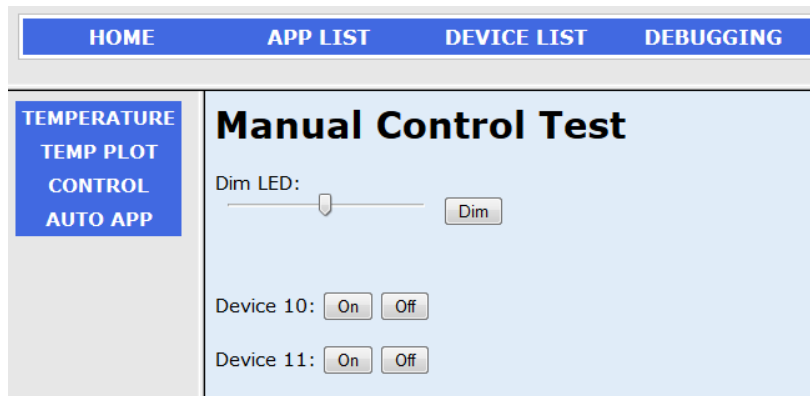


Figure 10. Manual Control Page

For automatic application scenarios the examples in Figure 11 display data handled by a threaded script which uses its own queue to communicate with the Web-interface. The threaded script continuously checks the sensor value received from a defined device in the log file. Should the value reach a defined threshold, a message to switch on a LED is sent to a different device. If the sensor value is below the threshold a message is sent to switch off the device. The status of this automated process is displayed upon accessing the Webpage. Additionally a picture indicating the status is loaded.

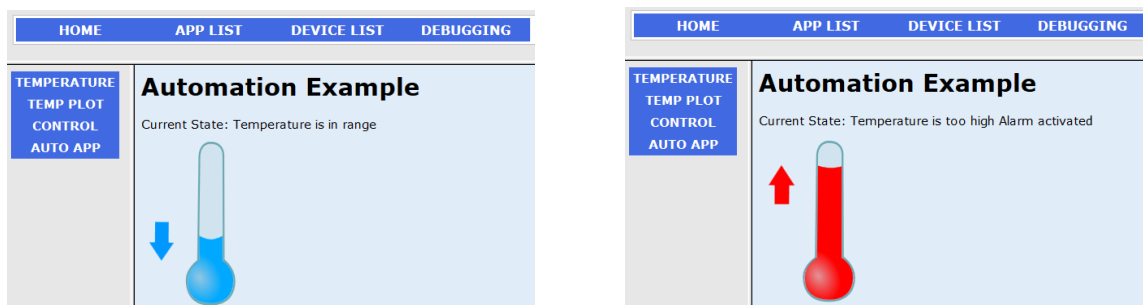
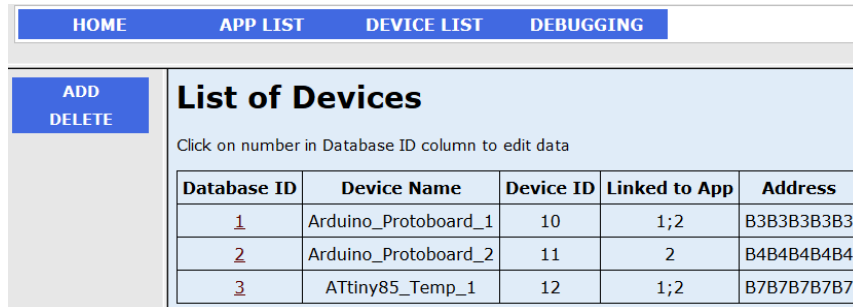


Figure 11. Automated Program Page (Temperature monitoring)

The implemented database includes a database file containing registered applications and a separate database file containing registered devices. Database entries can be added, edited and deleted via the Web interface.

The device list Webpage is an essential component as the unique device ID and its TX address can be defined for all devices (Fig. 12).

The developed applications cover the basic but most important features of a system intended for data display, device control and automation processes.



Database ID	Device Name	Device ID	Linked to App	Address
1	Arduino_Protoboard_1	10	1;2	B3B3B3B3B3
2	Arduino_Protoboard_2	11	2	B4B4B4B4B4
3	ATTiny85_Temp_1	12	1;2	B7B7B7B7B7

Figure 12. Device Database Page

5. Testing

The initial testing was performed to verify the operation of the RF module in conjunction with the Raspberry Pi and the Python code. The software tests during development were extensive, as several functions to be implemented had to be checked and very often alternative programming approaches had to be identified to find a more suitable solution. Initial testing was performed to verify the correct connection of the wireless module to the Raspberry Pi. The test environment consisted of a Raspberry Pi with the nRF24L01+ module connected to it and an Arduino Nano also equipped with the RF module.

The program on the Raspberry Pi was based on the Python script to control the nRF24L01+ module [4]: a few settings were modified, such as activating dynamic payloads and addressing all six data pipes. The program on the Arduino was based on the example sketch "ping-out" included in the RF24 library [6], with the same modified settings as in the Raspberry Pi program. Every second the Arduino sent its "up-time" in milliseconds via the nRF24L01+ module. The correct functionality of the module could be verified as the Raspberry Pi received the data and displayed it in a terminal window.

Further testing was performed continuously throughout the development phase. Whenever a new feature was implemented it was tested to see if the chosen method of implementation of the feature was correct and operated as required. If the program did not behave as anticipated, further software research was conducted utilizing the error state reported by the Web interface. This error message stated the line of code which caused the function or program to abort. If the function was identified as being inappropriate then alternatives had to be researched and/or the approach to the specific function had to be changed.

Besides testing the software installed and used on the Raspberry Pi, the software for the developed ATtiny boards was also tested for basic operability (Send and receive data; utilize internal temperature sensor). The overall system worked as intended and was sufficient to prove the concept. A few intermittent glitches occurred, mainly in the wireless transmission: these are difficult to investigate, but will be monitored in future: nonetheless, they do not detract from the proof of concept.

6. Conclusion

The development of a low-cost solution for the connection of wireless sensor nodes to the Internet has been presented. The solution has the additional benefit of extremely low power consumption and hence very long battery life. The core of the solution was the Atmel ATtiny series of wireless transceivers, which cost about one dollar at current prices: with the ancillaries that were necessary, the bill of materials comes to around four dollars per node at current prices, which is believed to be highly competitive. The designs of minimised printed circuit boards for three versions of the node have been presented.

The possibility of using an IEEE 802.x wireless interface standard was investigated, but it was concluded that the Atmel "ShockBurst" protocol offered significant advantages of minimised power consumption, as well as lower cost.

To provide the fixed node at minimum cost, a solution exploiting the Raspberry Pi microcomputer was implemented, attached to the innovative, but widely used, Nordic Semiconductor nRF24L01+ transceiver module. Software to implement the system in each of the nodes was developed from zero-cost sources (Copyleft and GNU GPL) in order to minimise the overall cost of the system. The software had to be tailored for the particular application and devices selected and the modifications are described.

A World Wide Web interface for the Raspberry Pi was created as the HCI portal, again minimising cost. This has an intuitive display and offers basic functionalities relevant for general automation tasks, data display and manual control, in conjunction with wirelessly connected clients. The implemented hardware control functions (digital in- and output, PWM, analogue reading) cover the most important features required for operating various types of peripheral devices. The Web interface can serve as the basis for a home automation system, a topic which is becoming more and more important and popular nowadays. The inexpensive wireless boards designed can either be interconnected with additional circuitry, for instance with a relay circuit to control high power devices, or to gather environmental and other sensor data to influence such high powered devices. With the Raspberry Pi acting as the server, several other functions, such as voice command or e-mail alert messaging, could be implemented in the future.

References

- [1] GNU Operating System, "GNU General Public License", Version 3, 29 June 2007, Free Software Foundation, Inc. Available: <https://www.gnu.org/copyleft/gpl.html> (Accessed: 19 March 2020).
- [2] GNU Operating System, "What is Copyleft?", 15 December 2018, Free Software Foundation, Inc. Available: <https://www.gnu.org/licenses/copyleft.en.html> (Accessed: 19 March 2020).
- [3] Hagström, K., "Raspberry pi: nRF24L01 and TCP", Gizmosnack, 27 May 2013. Available: <http://gizmosnack.blogspot.se/2013/05/raspberry-pi-nrf24l01-and-tcp.html> (Accessed: 19 March 2020).
- [4] Nordic Semiconductor ASA, "nRF24L01+ Single Chip 2.4GHz Transceiver", Product Specification v1.0, September 2008. Available: https://infocenter.nordicsemi.com/pdf/nRF24L01P_PS_v1.0.pdf?cp=10_4_0_0 (Accessed: 11 March 2020).
- [5] Microchip Technology Inc., "Atmel Microcontroller Selector". Available: <https://www.microchip.com/maps/Microcontroller.aspx> (Accessed: 11 March 2020).
- [6] Doxygen, "Driver for nRF24L01(+) 2.4GHz Wireless Transceiver", RF24 v1, 15 January 2012. Available: <http://maniacbug.github.io/RF24> (Accessed: 19 March 2020).

- [7] Hagström, K., "Tutorial nRF24L01 and AVR", Gizmosnack, 8 April 2013. Available: <http://gizmosnack.blogspot.se/2013/04/tutorial-nrf24l01-and-avr.html> (Accessed: 19 March 2020).
- [8] Hellkamp, M., "Bottle: Python Web Framework", Bottle, 19 March 2020. Available: <http://bottlepy.org/docs/dev/index.html> (Accessed: 19 March 2020).
- [9] Mounier, F., "Pygal: Simple Python Charting", Pygal, Revision 7a556a2e, 2016. Available: <http://www.pygal.org/en/stable> (Accessed: 19 March 2020).
- [10] Swartz, A., "Web.py". Available: <http://webpy.org> (Accessed: 19 March 2020).



© 2020 by the author(s). Published by Annals of Emerging Technologies in Computing (AETiC), under the terms and conditions of the Creative Commons Attribution (CC BY) license which can be accessed at <http://creativecommons.org/licenses/by/4.0>.