Smart Home Equipment Control System with Raspberry Pi and Yocto

Arghya Biswas[1], Dwaipayan Biswas[2], Sudakar Singh Chauhan[3], Antara Borwankar[4]
[1] School of VLSI and Embedded System Design, National Institute of Technology, Kurukshetra
[2] Government College of Engineering and Textile Technology, Berhampore
[3] ECE Department, National Institute of Technology, Kurukshetra
[4] Intel Technology India Private Limited
[1] arghyabiswas05@gmail.com, [2] dwaipayanbiswas.official@gmail.com, [3] sudakarnith@gmail.com,
[4] antara.borwankar@intel.com

***Abstract*: Yocto is a customizable lightweight Linux distribution for embedded devices with any kind of hardware architecture. With the Yocto, it is possible to compile an operating system with the same feature for the different types of hardware in a very short period of time. This paper proposed to use the Yocto with the Raspberry Pi to form a high-speed Internet of Things (IoT) hardware cum software solution with the only specific features that we need. The very common Linux distribution for the Raspberry Pi has a number of extra features that are not necessary for some dedicated applications like IoT application, which needs to run only some specific set of tasks. Here in this project, we designed a prototype of a smart home equipment control system with the two different models of Raspberry Pi and this embedded Linux operating system called Yocto. This system is able to control all the home equipment from the control interface display or from the specific website world wise.**

Keywords: Home Automation, Home equipment control, Raspberry Pi, Yocto, Linux, Internet of Things, Operating System, Attiny88 Microcontroller.

I. INTRODUCTION

Home equipment control is a very popular domain for the Internet of Things (IoT) [1] [2] applications. This paper is also proposing a prototype of a lightweight software solution for the home equipment control system based on the customizable embedded operating system called Yocto [3]. On top of the basic layer of the operating system we add a new layer for our application which will do the all necessary action needed for the equipment control system and it also has an interactive Graphical User Interface (GUI) that can help the user to control the equipment.

II. LITERATURE SURVEY

H. Khandelwal, P. Mankodi proposed an enhanced system for automated test-bed using Yocto [3]. They created an automated python file that can turn-on and turn-off the GPIO pin of their test board. They add that python script with a service file in a custom layer in the Yocto and build their image. After flashing the image the service file automatically calls the python file and test the GPIO pins. With this implementation they were able to reduce the time that need to do manually testing for different 300 boards.

Author D. Muthuswamy and A. P. Navik proposed Dual Band WLAN Gateway Solutions in Yocto Linux for IoT Platforms [4]. They took the Yocto as a base operating system and work on the dual-band Wi-Fi. Wi-Fi 2.4 GHz & 5 GHz ISM bands are a very good option for IoT platforms. They enable the feature of dual-band in the access point.

Author R. Şerban and I. Culic [5] proposed a very useful way to convert the Cisco IR829GW router to an IoT device. They install Yocto in that router and also install the Node.js, Avahi, Node-RED, Redis and Wyliodrin Server on top of the Yocto. As a result they were able to convert the router to an IoT device with dual-band transceiver facility. They also proposed to add a microphone and distance sensor with the system and then it can take the human voice as input and may perform some easy tasks.

Author A. S. [6] Haron configure and install the embedded operating system Yocto on a hard CPU in an FPGA. They choose Altera DE1-SoC with onboard hard ARM CPU for their experiment. They configure the hard processor system (HPS), Parallel I/O ports, Peripheral pins, SDRAM to make the Yocto compatible with the specific hardware. After that they also implement the HDL code to program the FPGA. With this all changes they were able to run the Yocto on that FPAG board.

Author Z. Liu, F. Jing [7] proposed an FPGA and ARM-SoC based canny edge detection system. They use the embedded operating system Yocto for ARM

SoC and FPGA to process the image to show on a VGA display. They also conclude that the ARM-SoC with embedded OS and FPGA based canny edge detection system has a high portability aspect. And the high-speed and low power consumption is also a plus point in their system.

Author V. Patchava, H. B. Kandala [8] proposed a technique for smart home automation with Raspberry Pi and camera and motion sensor. The motion sensor and computer vision was used to control the home equipment and monitored from a web interface. For security and the surveillance they used the camera. The live feed from the camera directly can be observed form the web interface. Whenever any unspecified motion has been detected, the Raspberry Pi will be triggered and the camera starts the recording, the alert alarm will be triggered and an SMS will be sent to the user mobile.

E. Rohadi et al. [9] proposed a Raspberry Pi based home security system. They added a webcam with the USB interface of the Raspberry Pi and set up an apache2 server on the Raspberry Pi. As the Raspberry Pi was connected to the local network, the web-server could be accessed from any other device on that home network. The made some changes to that web-server and able to transmit the webcam feed to the local network. They set up a special URL with the local network IP address. If any device browses that URL form that home network then it can observe the feed of the webcam. They were able to transmit different resolution video, but it added some extra delay to get the feed.

III. Motivation

Most of the IoT project based on Raspberry Pi uses the Raspbian [10] operating system. This operating system has a lot of features for general purpose use. But if we use a custom operating system that has only those features that are needed for the IoT application then we may reduce the power consumption as well as the processing load, as a result, a high-speed lightweight device. These things motivated us to work with custom embedded operating system Yocto and C++ language with the Qt framework for the GUI.

IV. Research Gaps

In most of the home equipment control system projects, the software is mainly classified into major two types. The first one some kind of firmware and another one is some custom application that is running on top of the general-purpose operating system. So for the firmware [11] [12] the processing power and speed is very low. On the other hand the general-purpose operating system has a number of features that are not used ever that is one kind of a waste of resources. So we are proposing a prototype with a custom operating system that can have only the features that are needed for the entire system. So the custom OS will have fewer loads with high throughput.

V. System Overview

This entire project has three main components as the software perspective and from the hardware prospect there are two main components. The software components are web interface for the application, a custom operating system based on the Yocto and the GUI application itself. For the hardware we use the very popular Raspberry Pi and attiny88 micro-controller for IO extension.

A. Web Interface

To control the home equipment we designed deployed a web interface that has the all button with the equipment name and status. A signal will be sent to the Raspberry Pi when any switch is being pressed. And the Raspberry Pi will take the responsibility of further processing.

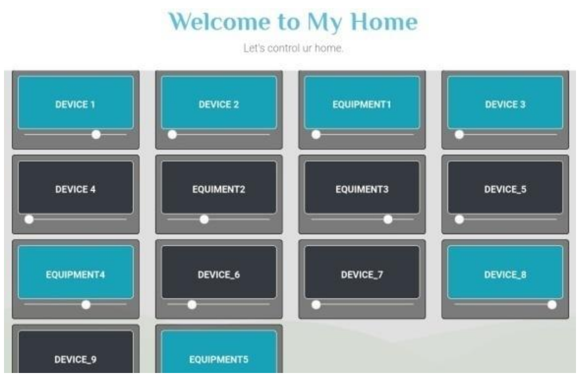


Fig 1: Web interface for the application

Here figure 1 is the screen short if the web interface. We added some switches and the equipment name. The color of the switch is intentionally used to recognize the state of the equipment.

B. Yocto

Yocto is an open-source project that helps us to create the custom Linux architecture based operating system with the custom features for different kinds of computers. For this project, we need only some basic features. We added the Qt framework base package, i2c library package, Wi-Fi and DHCP-server.

Figure 2 is the architectural diagram of Yocto. Make-like build tool bitbake [3] is used to build the Yocto image. The source code has been downloaded from the upstream repository or copied from the local directory as mentioned in the recipe files. The bitbake tool follows multiple numbers of steps to complete the image build process. Figure 3 is showing all the steps that are needed to execute by the bitbake tool.

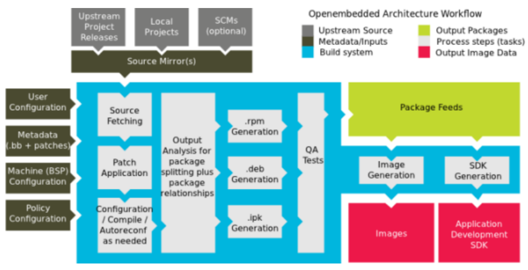


Fig 2: architectural diagram of Yocto

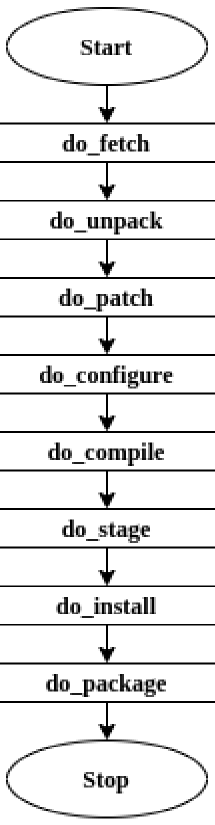


Fig 3: Workflow of Yocto

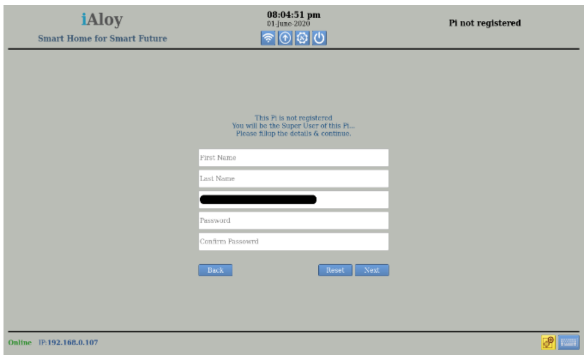
We build the Yocto for the Raspberry Pi. So we import the official Raspberry Pi layer for the Yocto on top of the basic layers. The Qt layout is also imported after the Raspberry Pi layer. We made a custom layer for our application. And then add the layer on the top.

We enable features like i2c, DHCP-server, Wi-Fi and the required dependency packages from the local.conf configuration file. We added some services in our custom application layer that can help to lunch our application just after the boot process.

C. The Main Application

The main application that will going to be executed on the Yocto image is a GUI based application written in the C++ language using the Qt framework [13] [14]. This application has three main windows. The first one is to register a user.

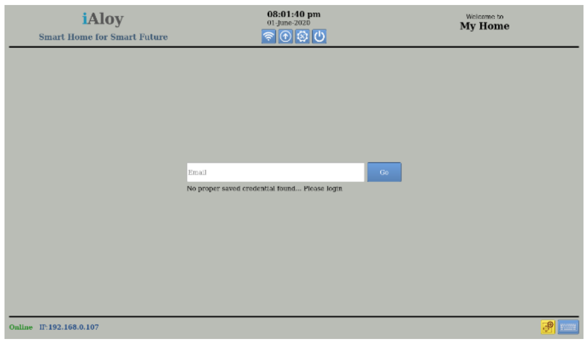
Figure 4 shows the registration window. It needs the very basic information about the user. We added the user account verification via sending a one-time password (OTP) on the user email id. The second window in figure 5 is used to login to the user account.



This Pi is not registered
You will be the Super User of this Pi...
Please fillup the details & continue.

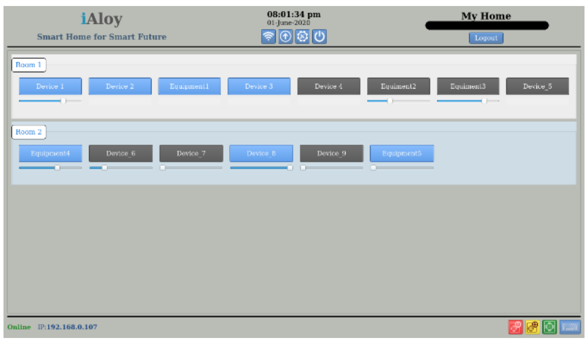
[redacted]

Fig 4: Registration window of the main application



No proper saved credential found... Please login

Fig 5: Login window of the main application



[redacted]

Fig 6: Dashboard window of the main application

The dashboard window in figure 6 has the all switch buttons that are needed to control the equipment. To ensure the better user experience we grouped the switches room wise.

This application will be synchronized with the web application. In Figure 1 and 6 is also showing the synchronized status for the equipments. When a button in the application is pressed then the application will generate two signals. One signal comes to the i2c bus of the Raspberry Pi to update the equipment state and the other one come to the web interface and update the web database as well as the web GUI. When a button is pressed in the web GUI the same thing happens and it updates the equipment status as well as the GUI in both web and local application.

Some extra GUI based features are added in this application because the operating system doesn't have any other GUI for the OS related features. So this application provides the GUI solution for features like selection of the Wi-Fi, reboot, power-off, virtual keyboard, font style change, font size change, screen resolution, brightness control, system update etc. Figures 4, 5 and 6 showing five buttons on the top portion of the application for those features.

D. Raspberry Pi

Raspberry Pi is a very compact single-board computer having a decent processing speed and the memory available which is perfect for our project. The Yocto OS and the main application are running on this Raspberry Pi. The application uses the dedicated i2c pins of the Raspberry Pi and sends the data to the attiny88 micro-controller.

We are mainly developing the project for the Raspberry Pi version 4. But it has the backward compatibility with the previous versions of Raspberry Pi. As the processor is different in those boards so we need to compile the OS separately for all those boards.

E. Attiny88 microcontroller

Attiny88 is an 8-bit AVR micro-controller. We use the Dual Input Package (DIP) version of the micro-controller. It has 20 General Purpose Input Output (GPIO) pins.

Figure 7 is a complete module with an attiny88 micro-controller and 8 LEDs and 8 push switches with it. We made multiple PCBs of this module. Here we use an Arduino pro mini board with SPI to flash the code to the micro-controller. The attiny88 micro-controller module will be connected with the Raspberry Pi via i2c connection. As we are using the i2c protocol for the communication so there is scope of connecting 112 modules like this. But for the project prototype purpose we added only two modules with the Raspberry Pi.

Here the push switches are representing the physical switches and the LEDs are representing the home equipment.

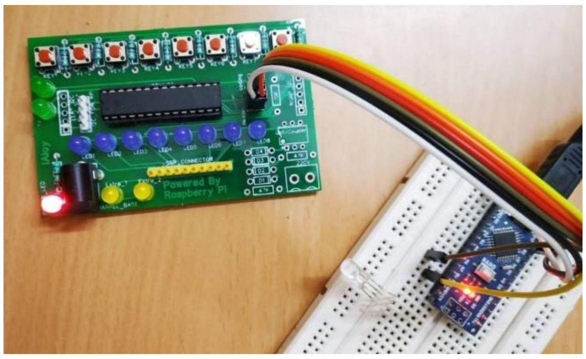


Fig 7: Attiny88 microcontroller with LEDs and switches, connected with an Arduino to flash the program

VI. WORKFLOW

The workflow of the entire project is divided into mainly three parts. All the home equipment can be controlled by the physical switches, from both the application that is installed on the Raspberry Pi and from the web interface. Those three modes of control are heavily synchronized. If the equipment is updated from any of those interfaces, then other interfaces will automatically update it.

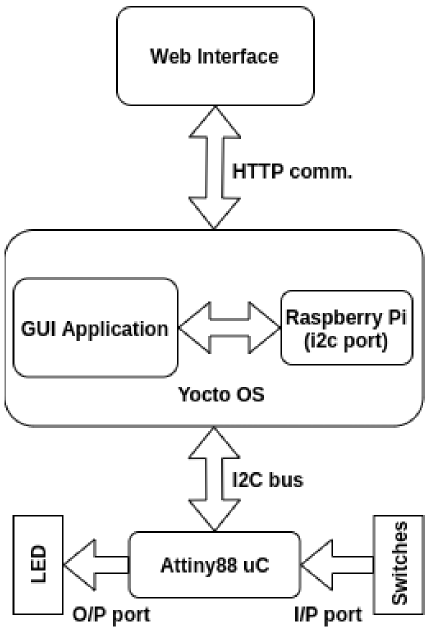


Fig 8: block diagram of the entire project

The Yocto OS helps the application to communicate with the attiny88 micro-controller through the i2c communication bus of the Raspberry Pi. Here the Raspberry Pi is working as the master in the i2c communication and the attiny88 micro-controllers are working as the slaves. Raspberry Pi can send some data to write on the output port of the micro-controller and also able to send the request to read the input port of the micro-controller.

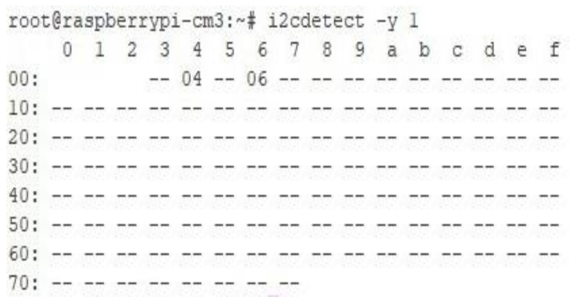
In figure 8 it is clearly visible that the web interface and the Yocto are talking via HTTP connection. The main GUI application reads the web database and then updates itself and finally sends the command to the micro-controller module to update the equipment status. The Yocto helps the application to send the data to the i2c port of the Raspberry Pi and read the data from the i2c port. On the other hand multiple attiny88 micro-controllers are connected in that i2c bus. The application can access those micro-controllers via their unique address. We are using the 7-bit addressing mode so total possible micro-controllers that can be attached to this bus is 112. Each micro-controller can drive 8 LEDs and 8 switches. So, it is very clear that a huge number of equipment can be controlled by a single Raspberry Pi.

VII. IMPLEMENTATION DETAILS

We created a dedicated operating system for this project. The application is built in a pre-installed fashion with the operating system. We also added a service file to run the application just after the booting completion. As the GUI application is built with the Yocto OS so it is a bit difficult to update some part of the application. So we introduce an update manager concept in the application. We can simply add the updated binary and dependency files in a zip folder and just upload it to a web repository from any other computer. And from the old GUI application user can download and install the new binary just pressing update and restart button.

As we are using the Yocto as the base of the project so the other required features (like Wi-Fi driver, Ethernet driver, Display driver etc) doesn’t need so much attention. Just add a single line for in the configuration file in the Yocto and Yocto will take care of all those things.

We are mainly using the Raspberry Pi 4 as base hardware. So we first build the image for the Raspberry Pi 4. And later we also test the entire project in the Raspberry Pi 3 model. We flashed each of the attiny88 micro-controller with a unique address which is needed to communicate over i2c bus.



```
root@raspberrypi-cm3:~# i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- 04 -- 06 -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Fig 9: Available i2c device on the bus

Here in figure 9 clearly visible that there are two devices are available on the bus with unique address 0x04 and 0x06.

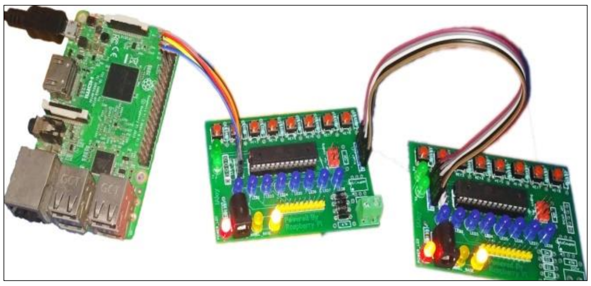


Fig 10: Full hardware setup

The original hardware setup is shown in figure 10. Two attiny88 micro-controller modules are connected via an i2c bus with the Raspberry Pi. The PCB of the attiny88 micro-controller module is designed in such a way that every module has two same i2c ports. So one port can be used to connect with the previous module and another port can be used to connect with the next module. So, it makes the entire setup clean.

The GUI application continuously fetches the web database and the push switch status from the attiny88 micro-controller. So, if any state change is noticed by the application then it updates the other one. As it is working as a bridge between the web and the physical system, but it also has the controlling interface as shown in figure 6. So, all the equipment can be controlled by the physical switches, local UI interface or web interface.

Here in this prototype we are using LEDs to representing the equipment. But in real life it needs some solid-state relay to drive AC mainline equipment. The relays can be attached to the micro-controller and the micro-controller drive the relay to control the equipment.

VIII. CONCLUSION & FUTURE WORK

This prototype can be implemented at home, school, college, or in some industry to observe and control the electrical equipments. Beside the GUI interface, this prototype re-uses the switches that are already installed with the old equipment. So it will make the user experience better than ever. It is also possible to add a touch screen with Raspberry Pi, and then it will be easier to use the interface.

This project is the only prototype. So there are a few things that need some more research work. Some up-gradation can be done in the software part some in the hardware part. We are using the Raspberry Pi as the heart of the project. This has a huge number of features like the Camera Serial Interface (CSI), Display Serial Interface (DSI), Bluetooth those we are not using. So it is a waste of resources. But if we can customize the hardware platform and keep only those part that is needed for our project, then it will be more efficient in respect of power consumption.

Between the GUI interface and the web database we are using HTTP connection as a communication medium. Due to this HTTP connection there is a certain delay between the button presses in the web interface and reflects the response in the hardware. So, if we replace the Http connection by the TCP connection then this delay can be avoided.

REFERENCE

[1] Agarwal K, Agarwal A, Misra G (2019) Review and Performance Analysis on Wireless Smart Home and Home Automation using IoT. 2019 Third International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC). doi: 10.1109/i-smac47947.2019.9032629.

[2] Vishwakarma S, Upadhyaya P, Kumari B, Mishra A (2019) Smart Energy Efficient Home Automation System Using IoT. 2019 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU). doi: 10.1109/iot-siu.2019.8777607.

[3] Khandelwal H, Mankodi P, Prajapati R (2017) Enhancement of automation testing system using Yocto project. 2017 International conference of Electronics, Communication and Aerospace Technology (ICECA). doi: 10.1109/iceca.2017.8203630.

[4] Navik A, Muthuswamy D (2017) Dual band WLAN gateway solutions in Yocto Linux for IoT platforms. 2017 International Conference on Internet of Things for the Global Community (IoTGC). doi: 10.1109/iotgc.2017.8008968.

[5] Serban R, Culic I (2016) Configuring a cisco IR829GW as an internet of things device. 2016 15th RoEduNet Conference: Networking in Education and Research. doi: 10.1109/roedunet.2016.7753219.

[6] Haron A, Talip M, Khairuddin A, Izam T (2017) Internet of Things Platform on ARM/FPGA Using Embedded Linux. 2017 International Conference on Advanced Computing and Applications (ACOMP). doi: 10.1109/acomp.2017.26.

[7] Liu Z, Jing F, Fan J, Wang Z (2019) Implementation of a FPGA-ARM-based Canny Edge Detection System. 2019 Chinese Control Conference (CCC). doi: 10.23919/chicc.2019.8865695.

[8] Patchava V, Kandala H, Babu P (2015) A Smart Home Automation technique with Raspberry Pi using IoT. 2015 International Conference on Smart Sensors and Systems (IC-SSS). doi: 10.1109/smartsens.2015.7873584.

[9] Rohadi E, Suwignjo S, Pradana M et al. (2018) Internet of Things: CCTV Monitoring by Using Raspberry Pi. 2018 International Conference on Applied Science and Technology (iCAST). doi: 10.1109/icast1.2018.8751612.

[10] Balon B, Simic M (2019) Using Raspberry Pi Computers in Education. 2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). doi: 10.23919/mipro.2019.8756967.

[11] Yohan A, Lo N, Santoso L (2019) Secure and Lightweight Firmware Update Framework for IoT Environment. 2019 IEEE 8th Global Conference on Consumer Electronics (GCCE). doi: 10.1109/gcce46687.2019.9015316.

[12] Son M, Kim H (2019) Blockchain-based secure firmware management system in IoT environment. 2019 21st International Conference on Advanced Communication Technology (ICACT). doi: 10.23919/icact.2019.8701959.

[13] Jinhui Q, Hui L, Junchao Y (2012) The Application of Qt/Embedded on Embedded Linux. 2012 International Conference on Industrial Control and Electronics Engineering. doi: 10.1109/icicee.2012.346.

[14] Xibo W, Jianan Y (2009) Add Touch Screen Support for QT/Embedded. 2009 International Forum on Computer Science-Technology and Applications. doi: 10.1109/ifcsta.2009.314.