Home Automation System

An Internet enabled embedded system that remotely controls home appliances and monitors the surrounding temperature.

Tarek Kaddoura
Jigar Nahar
Group 16

Abstract

With the advent of technology, everyday tasks have become more and more automated. Home automation is the use of computers and the Internet to control home appliances such as lights, thermometers, and even coffee makers. With the use of home automation systems, the control of these appliances can be centralized to one location – the cloud. In this home automation system, the Altera DE2 board is used to allow a user to control X10-controlled appliances in their home through a common web interface. The board communicates with an X10 USB Transceiver, which then communicates with the rest of the X10 Home Automation System.

Overall, the project worked successfully. The X10 device was interfaced successfully with the DE2 board. The drivers worked successfully as well.

Table of Contents

Functional Requirements of project

The Home Automation System is an embedded system that is capable of remotely controlling home appliances. The system will allow a user to control home appliances from any device with a basic web browser. The following are the functional requirements of this project:

- A fully functional web server running on the board.
- A nicely designed website with an authorization system for controlling the appliances with the ability to dynamically add or remove appliances through the web browser.
- Ability to control an X10 USB Transceiver.
- Modular and scalable design so that one can easily add or remove appliances.
- USB Host capabilities

The above mentioned requirements are the minimal requirements necessary and the goals set for this project.

Design and Description of Operation

There are two main components to the Home Automation System: the controller, and the X10 Automation system.

The controller is the core of the Home Automation System. It is connected to the internet via an Ethernet connection. The controller implements a web server and hosts the web site. This allows the controller to be remotely controlled by a user through the web site. The controller is also connected to the X10 USB Transceiver module. This module is a wireless transceiver that connects via USB and can talk to the X10 Transceiver Base.

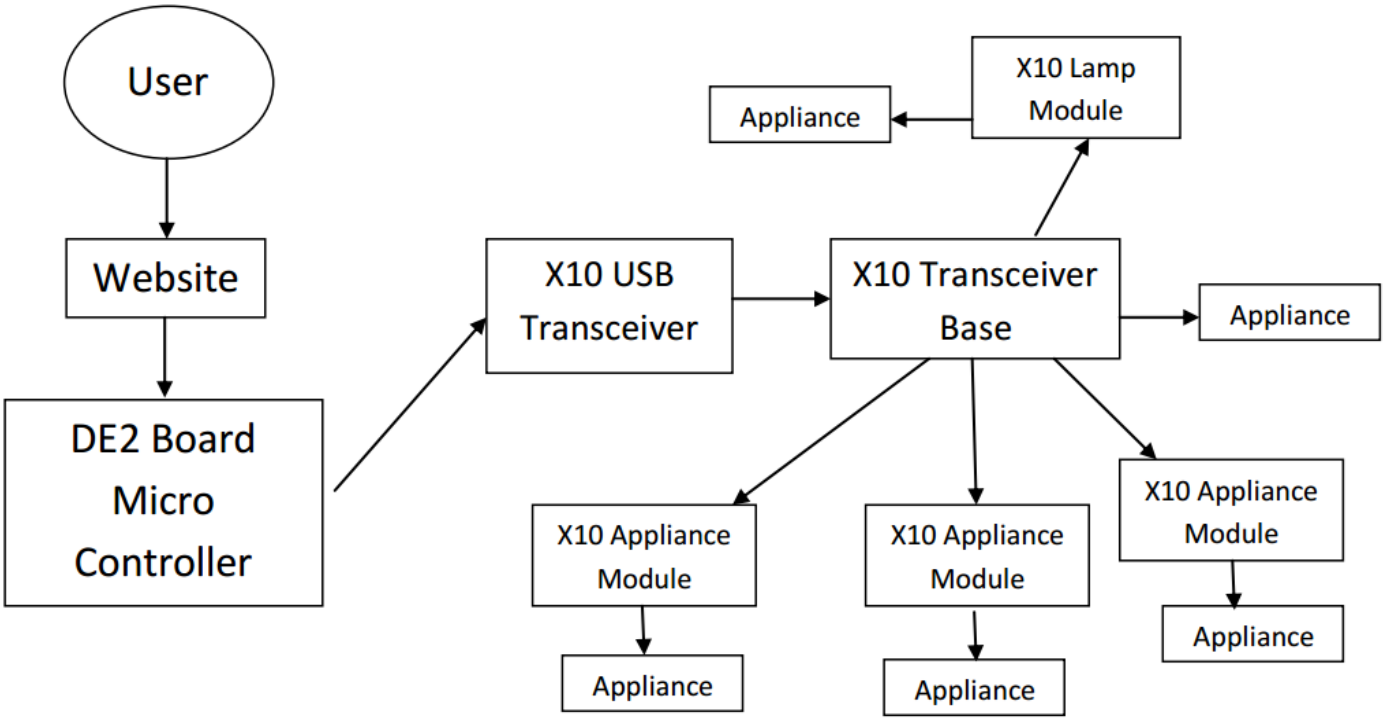
The other part of the design is the X10 system. The X10 system consists of a transceiver base and appliance modules communicating together through the electrical circuitry of the house. The base accepts commands from the USB Transceiver module via RF signals.

Description of Operation

The user begins by choosing what to do on the web site hosted on the controller (DE2 Board). Once the web server sees the request from the user, it evaluates the request and sends the required commands to the X10 USB Transceiver module. The USB module then communicates with the X10 Transceiver Base by RF signals. These signals are then interpreted in the X10 base and communicated over to the X10 appliance modules via the electric circuitry of the house.

The figure below shows the interactions between the user, the controller, and the external appliances.

Fig 1: Operation of Home Automation System



Bill of Materials

TM751 Transceiver Module w/ Built-in Appliance Module - X10 - $19.99 (Quantity = 1 unit)

The transceiver module for the X10 system that converts RF signals from remote controllers and relays them to the electrical circuitry of the house

Website: http://www.thehomeautomationstore.com/tm751.html

Datasheet: ftp://ftp.x10.com/pub/manuals/tm751-is.pdf

CM19A USB PC Transceiver – X10 - $15.99 (Quantity = 1 unit)

USB module that acts as a remote controller for the transceiver module.

Website: http://www.thehomeautomationstore.com/cm19a.html

Datasheet: ftp://ftp.x10.com/pub/manuals/cm19a-is.pdf

AM466 Appliance Module 3-Pin Grounded - X10 - $15.99 (Quantity = 2 unit)

Appliance module that switches appliances on and off.

Website: http://www.thehomeautomationstore.com/am466.html

Datasheet: ftp://ftp.x10.com/pub/manuals/am486-466-is.pdf

LM465 Lamp Module – X10 - $15.99 (Quantity = 1 unit)

Lamp module that switches lamps on and off.

Website: http://www.thehomeautomationstore.com/lm465.html

Datasheet: ftp://ftp.x10.com/pub/manuals/lm465-is.pdf

Altera/Terasic DE2 development board box - $517.72 (quantity = 1 unit)

The Box includes:

- The 8 x 6 inch DE2 board with a Cyclone II EP2C35 (672-pin package) FPGA
- 9V AC/DC adaptor
- USB cable
- Plexiglas cover for the DE2 board
- Installation guide: ftp://ftp.altera.com/up/pub/Webdocs/DE2_UserManual.pdf

Website:http://www.altera.com/education/univ/materials/boards/de2/unv-de2-board.html

5 V 2A power supply required for the board to run.

Sources of Reusable Design Units

A driver needs to be written to interface the ISP1362 USB controller correctly on the board. Based on research, the following VHDL interface is considered:

ISP1362 VHDL interface for DE2

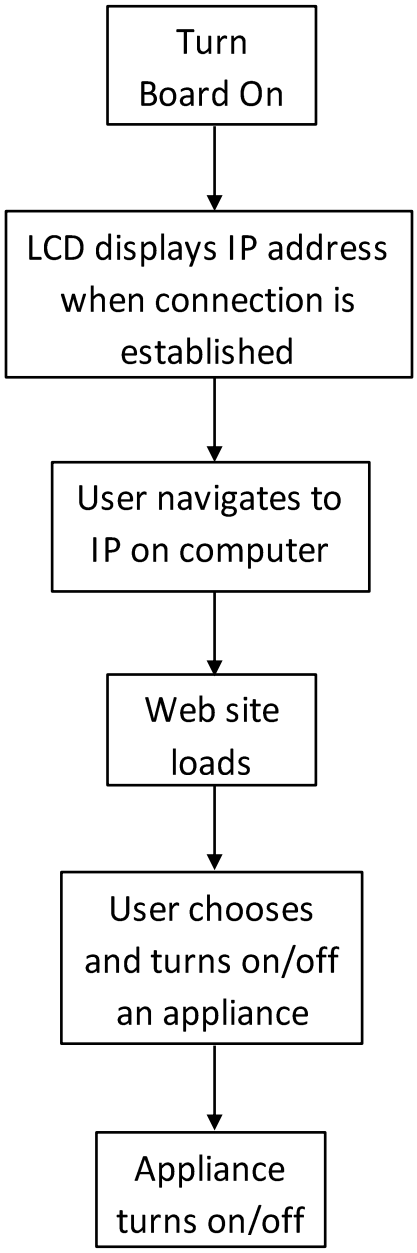
This project provides a standalone interface to the ISP1362 USB controller. It allows communication with the controller without the need for the Nios II.

Limitations of this project include a transfer rate of 180 KB/s and 2 Bytes per transaction. Also, the project doesn't implement hardware flow control for overflow. Hence, the software must make sure to avoid hardware overflow.

Source: http://mzakharo.github.com/usb-de2-fpga/

Datasheet

User-Perspective Block Diagram



Voltage and Power

Throughout the program's run, the current fluctuates between 0.47 and 0.49. The voltage was measured at 9 V. Taking the average of the current, the average power consumption is P = 4.32 W

Background Reading

The research of enhancing the transfer rate in an embedded USB-Host system [4]

The article talks about various methods of enhancing the transfer rate in an embedded USB-Host system. It highlights several measures of merit for USB performance, as well as methods on enhancing those statistics.

The methods mentioned in this paper to enhance the transfer rate can be used to speed up the rate at which commands are sent to the USB module. Even though this project does not transfer data onto a USB flash disk as in this article, the paper's research and results can still be expanded on for this project.

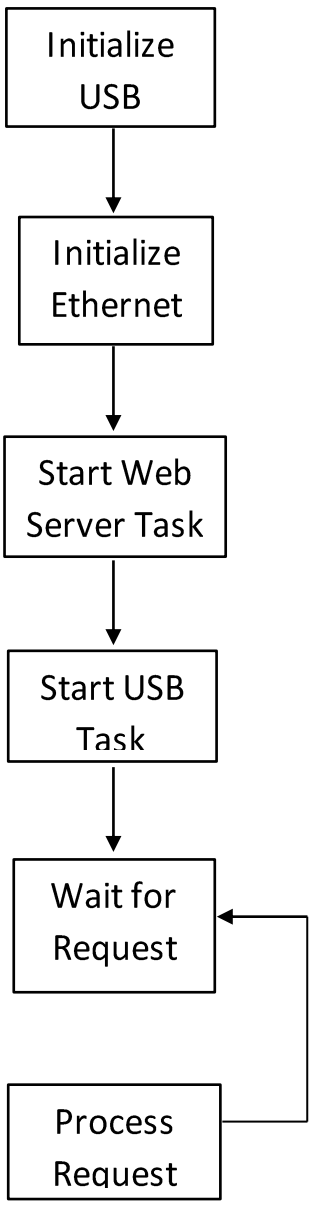
Building automation through web interface [5]

This paper talks about the design of a system for controlling devices through a web interface. The paper discusses methods for testing such a system, and optimizing the system and its results. The authors also expanded on the authentication used in the web interface as well as the issues that arise with it.

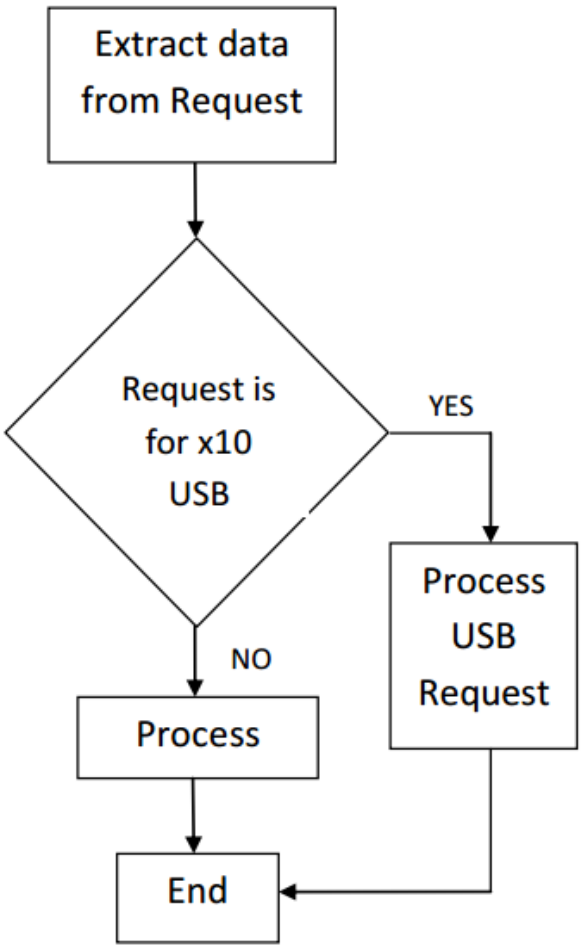
A lot of the research and testing procedures in this paper can be directly applied to our project. The optimizations listed in the paper can also help optimize many factors of our system. In addition, the authentication discussed in this paper also helps in deciding the authentication scheme to use in this project.

Software Design

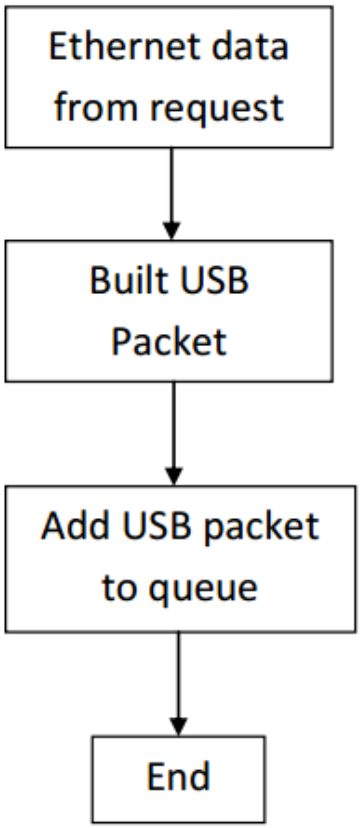
Starting Up



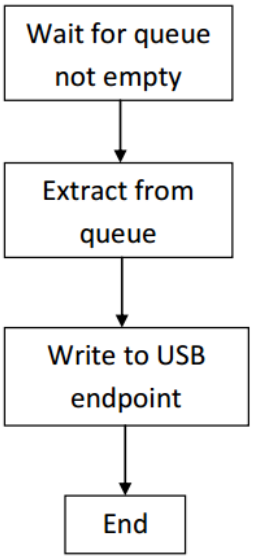
Web server task: Processing a Request



Web Server Task: Process USB Request



USB Task



Program Flow and Task Interactions

The software is implemented with the uC/OS RTOS.

There are three tasks defined in this software context:

1. Startup Task
2. Web Server Task
3. USB Task

To begin, the USB and Ethernet devices are first initialized. Then, the startup task is started. This task initializes the the Interniche stack. Once the Interniche stack is initialized, the web server task is started by the Interniche stack. At this point, the startup task then proceeds to start the USB task.

Once the web server is started, it waits for requests indefinitely. Once a request is made, it is processed and then the server continues waiting for more requests. No new tasks are started to take care of requests. Basically, the web server task polls the available connections in the sockets in a while loop. At any point in time, the server is either waiting for a request, or processing one.

When a request is made, the user could either request to perform a change on the list of devices, or request that a certain device switch on or off. The request is first examined to determine what the request is asking to do. At this point, if the request is asking to modify the devices list, then it is processed and the devices are changed according to the request data. In the other case, if the request is attempting to talk to the devices, then the request will be handed off to a USB request handler in the web server task. After the request has been successfully processed, the server goes back to its waiting state.

If the request was handed off to the USB request handler, then the data is first extracted from the request and then built into a USB packet according to X10 specifications. This USB packet is then added to the mailbox of USB packets to send.

The USB mailbox is the synchronization structure between the USB request handler in the web server task, and the USB communication task. The USB task waits until the mailbox is not empty, and sends each mailbox data. All packets in this queue are allowed to get delivered, but are waiting on their turn. Once a packet reaches the front of this queue, it gets written to the USB endpoint directly. This sends the packet on its way to the X10 interface. At this point, the operating system switches back to the web server task.

Drivers and Libraries

The software talks to the ISP1362 USB controller and the DM9000A Ethernet controller.

In order to talk to the USB controller, the software uses the drivers for the USB provided in the Altera DE2 demonstration projects. Also, the software uses the drivers provided by the manufacturer for the DM9000A controller.

In order to ease the USB communication process, a library was created that performs the complicated tasks of USB initialization and communication. This library is Framework.c in the drivers.

For the web server, the Interniche stack is used.

X10 Communication and USB Basics

Talking to the USB device consists of several steps. The USB hardware needs to first be initialized, then the USB device needs to be initialized.

The ISP1362 supports three different types of transfers: (a) Isochronous transfers, (b) interrupt transfers, and (c) control/bulk transfers. Consequently, each transfer has its own buffer on the controller.

In order to initialize the USB device, control packets are sent through the control transfers by using the ATL buffer in the controller.

The USB hardware is initialized by the following process:

1. Write 0x00F6 into the HcReset register
2. Call reset_usb() function
3. Initialize the ATL parameters for control packets
    a. w16(HcControl,0x6c0);
    b. w16(HcUpInt,0x1a9);
    c. w16(HcBufStatus,0x00);
4. Call set_operational();
    a. This sets either of the ports on the DE2 to operational
5. Call enable_port();
    a. This enables either port if a device is present.
6. Call assign_address(1, 2, 0);
    a. This function assigns the address 1 and 2 to the first and second port on the DE2 respectively if a device is present as well.

Once the USB hardware is initialized, the USB device should be initialized as well. This involves choosing which configuration the device should use. For the X10 device, it is a very simple USB device. It only has one configuration: 0. Then, to initialize the X10 device, the software simply calls *set_config(2, 0);* which sets the configuration to 0 for the device enumerated as address 2. This function returns 0 on success.

Now that both the hardware and the device are initialized, the software can talk to the device with interrupt writes. To use interrupt writes, a write endpoint address is needed. This can be read from the endpoint descriptors of the USB. For the X10, the endpoint address for writing is ***0x08. In order to do this, the following steps are necessary:

1. Setup the interrupt parameters. There are three parameters that need to be set up:

a. HcIntSkip: Defines a map for the buffer to choose which PTDs to send and which to not send.
b. HcIntLast: Defines the last PTD in the buffer
c. HcIntBlkSize: Defines the size of the interrupt bulk
d. For the X10, only one PTD will be sent from the buffer, so the parameters are:
    i. unsigned long int_skip=0xFFFFFFFE;
    ii. unsigned long int_last=0x00000001;
    iii. unsigned int int_blk_size=64;
e. Then, the interrupt parameters can be setup:
    i. w32(HcIntSkip,int_skip);
    ii. w32(HcIntLast,int_last);
    iii. w16(HcIntBlkSize,int_blk_size);

2. Create an interrupt PTD. The PTD contains the header of the USB packet.
    a. make_int_ptd(cbuf, OUT, EP, pbytes, 0, address, port, freq);
        i. The function creates an OUT packet (for writing), with an endpoint of EP, a payload of size pbytes in bytes, and a frequency of freq. It then stores the result in cbuf. The frequency can be 0 for this example.
3. Add the payload to the PTD. Simply append the payload to the address "cbuf+4" (assuming cbuf is an integer array).
4. Send the interrupt using the send_int() function.
    a. send_int(cbuf, rbuf);
        i. This sends the PTD from cbuf that was just created. If there is any reply from the device, it will be stored in rbuf. But, there doesn't have to be a reply in order for the data to have been sent.

In order to simplify this process, wrapper functions (from Framework.c) are used to perform the several steps of each action:

- InitializeUSB(): Initializes the USB controller
- InitializeX10Device(): Initialized the X10 USB Device
- X10SendCommand(): Sends a payload to the device

Test Plan

Testing is performed on the software components of the design as well as the hardware components.

For the X10 USB Transceiver, it is first tested on linux using the python driver obtained from [6]. This confirms that the python driver's usage actually performs the necessary functions successfully. The driver was then ported to the board. The USB transceiver is tested on the board by sending the same commands as in the linux python driver.

From the board, several commands were sent and the results were observed:

- The USB RF transceiver flashes red when a command is sent to it. This confirms that the command was sent to the transceiver.
- The respective appliance module is turned on or off depending on the command

For the web server, the tests consist of confirming that:

- Board successfully obtains an IP via static IP allocation
- Board successfully obtains an IP via DHCP through a router
- Can successfully view the web site on the board from another computer connected to the router

After the USB device and the web server are successfully tested, the integration testing is performed by turning a device on through the web interface.

All the above tests passed in this project.

Results of Experiments and Characterization

The board initially ran at a frequency of 50 MHz. This frequency was found to be too slow to accommodate a complicated web site. The requests from the web server were being interrupted without cause. To accommodate more requests, the frequency was overclocked to 100 MHz. This allowed the hardware to be more responsive as a whole when the web server was processing requests.

The web site created for this project references a lot of external script files. The web server used on this project was modified originally from the Altera web server in the NIOS II IDE. This web server processes requests in such a way that make it impossible for the board to be responsive. It splits the files to send to the client into chunks, and sends multiple files concurrently. This causes the board to become unresponsive for more complicated web sites even after overclocking to 100 MHz. To solve this problem, the board will simply host a basic HTML file which will reference external scripts and images from an external internet server instead. This allowed for the board to load a complex site in milliseconds, as opposed 15 seconds when all the scripts were hosted on the board.

By increasing the clock to 100 MHz, the speed increases but the power increases as well. This is a tradeoff but not significant enough. The power consumption is still acceptable at 100 MHz.

References

[1] Altera. (2013, Mar.). Nios II Processor: The World's Most Versatile Embedded Processor. Available: http://www.altera.com/devices/processor/nios2/ni2-index.html

[2] Terasic. (2013, Mar.) Altera DE2 Board. Available: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=53&No=30&PartNo=4

[3] Terasic. (2013, Mar.) Altera DE2-70 Board. Available: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=53&No=226&PartNo=4

[4] Chen Wei; Fan Zhentao, "The research of enhancing the transfer rate in an embedded USB-Host system," Computer Science and Education (ICCSE), 2010 5th International Conference on , vol., no., pp.957,960, 24-27 Aug. 2010

[5] Cheah Jun Hong; Lew Kim Luong; See Yuen Chark, "Building automation through web interface," Sustainable Utilization and Development in Engineering and Technology (STUDENT), 2012 IEEE Conference on , vol., no., pp.299,304, 6-9 Oct. 2012

[6] Michael LeMay. (2013, Mar.). X10 CM19A Linux Driver. Available: http://m.lemays.org/projects/x10-cm19a-linux-driver

Appendix

Appendix 1: Quick start manual

There are few things you need to do before starting up the DE2 board.

1. Connect an Ethernet cable from the wall into the wired router.
2. Connect another Ethernet cable from the router to the board.
3. Connect the RF USB transceiver to the USB on the board.
4. Plug the transceiver base module in the plug on the wall.
5. Connect the appliances and devices to the sockets on the appliance modules.
6. Plug those appliance modules into the plugins on the wall in your house.
7. Make sure that the appliance modules and the transceiver module are plugged under the same wiring system.
8. Start up the DE2 board.
9. Go to the website and start using your new Home automation system.

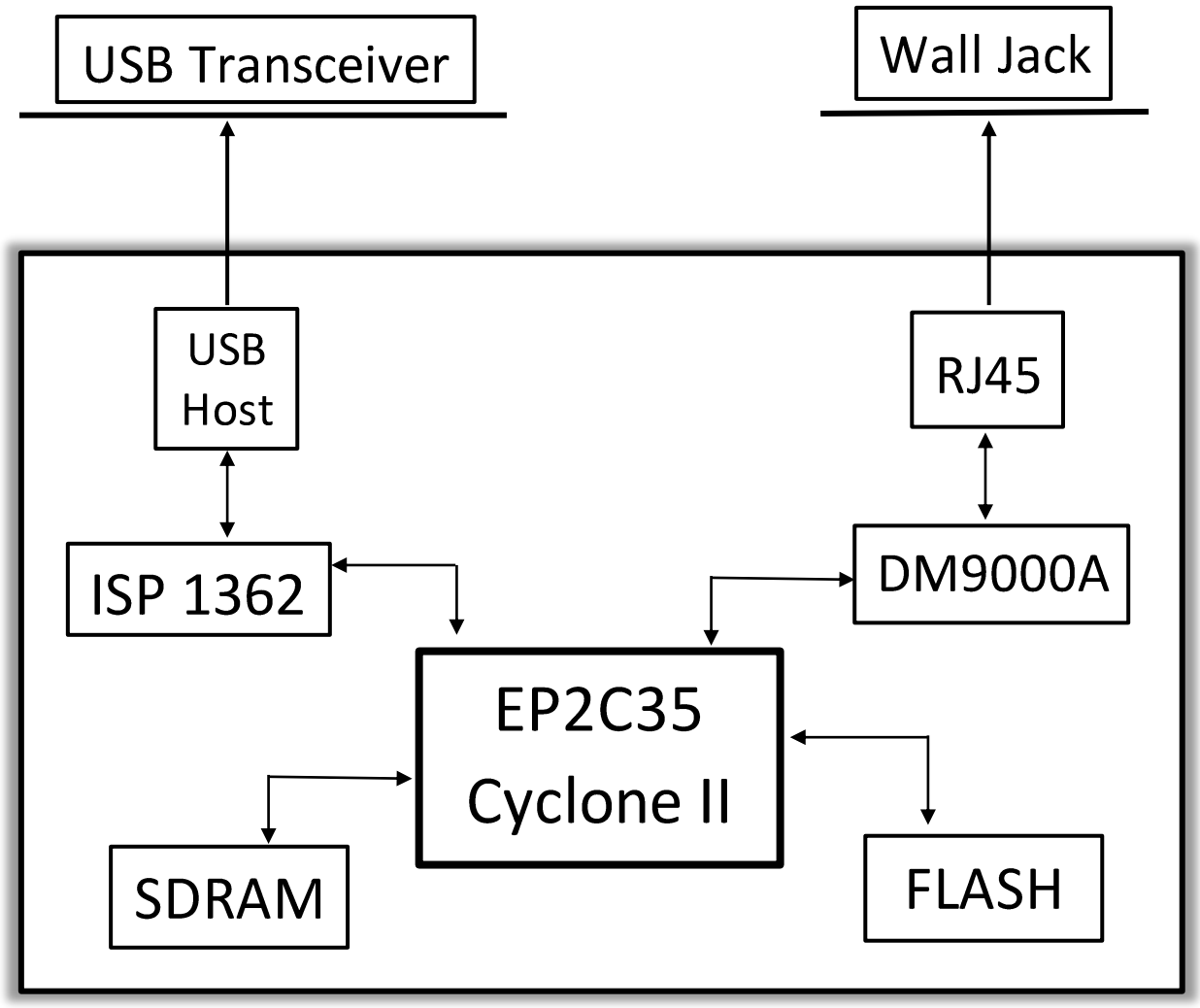
Pressing key 0 will reset the DE2 board and restart in case it does not work.

Appendix 2: Future work

These features that can be explored and added in the future:

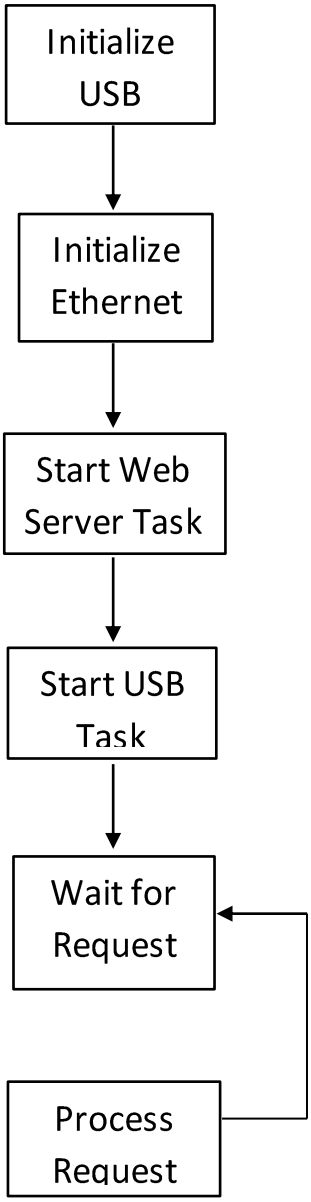
- Visual alarm clock that turns selected external lamps on/off or dim/bright when a configurable time is reached.
- Adding a thermostat to regulate the temperature in the house.
- Having security cameras that work with this system and live video surveillance through the website.
- Building a mobile app instead of controlling it from the website.

Appendix 3: Hardware Documentation

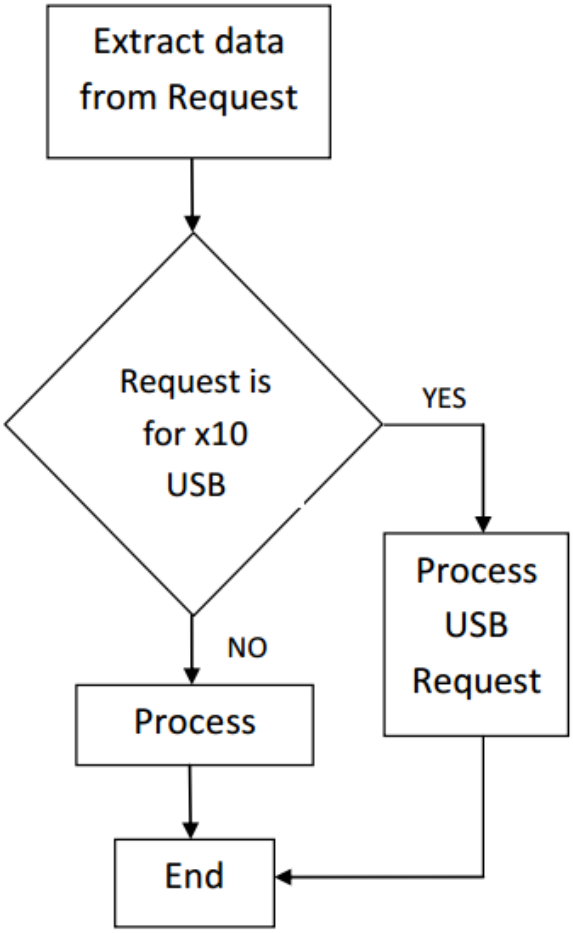


Appendix 4: Source Code Section

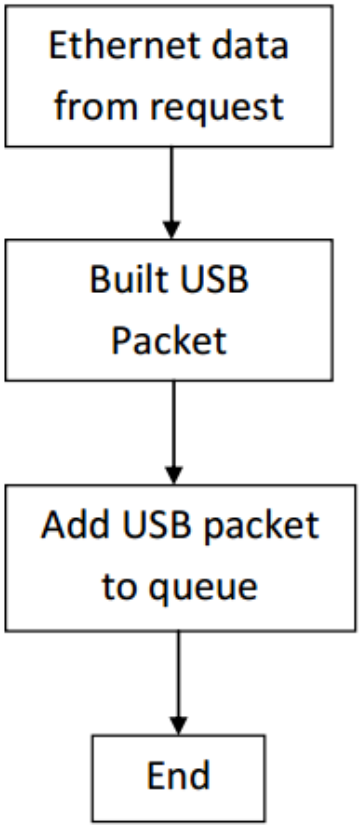
Starting up



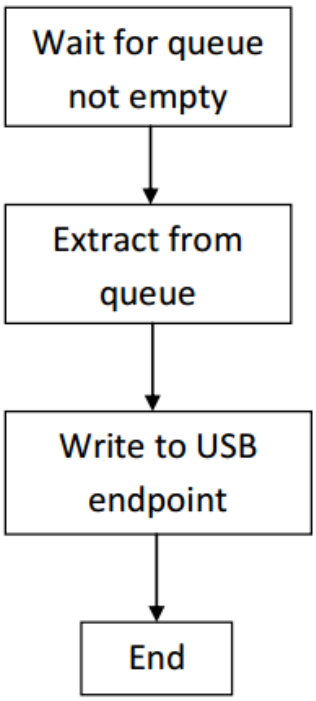
Processing a request



Processing a USB request



USB Queue Process



The source code files are submitted as a tar file along with this project titled HomeAutomation.tar.