Cellular Gateway Development using Embedded Linux Platform

Major Project Report

Submitted in fulfillment of the requirements
for the degree of

Master of Technology
in
Electronics & Communication Engineering
(Embedded Systems)

By

KHOSLA ROHAN

(17MECE17)

Electronics & Communication Engineering Department
Institute of Technology
Nirma University
Ahmedabad-382 481

May 2019

Cellular Gateway Development using Embedded Linux Platform

Major Project Report

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology

in

Electronics & Communication Engineering

By

KHOSLA ROHAN
(17MECE17)

Under the guidance of

External Project Guide:

Mr. Milap Patel

Product Manager(R & D)

Masibus Automation and instrumentation Pvt. Ltd.,

Gandhinagar.

Internal Project Guide:

Dr. Sachin Gajjar

Associate Prof.,M.Tech-ECE

Institute of Technology,

Nirma University, Ahmedabad.

Electronics & Communication Engineering Department

Institute of Technology-Nirma University

Ahmedabad-382 481

May 2019

Declaration

This is to certify that

a. The thesis comprises my original work towards the degree of Master of Technology in Embedded Systems at Nirma University and has not been submitted elsewhere for a degree.

b. Due acknowledgment has been made in the text to all other material used.

- KHOSLA ROHAN

17MECE17

Disclaimer

"The content of this paper does not represent the technology, opinions, beliefs or positions of Masibus Automation and Instrumentation Pvt. Ltd., its employees, vendors, customers, or associates."

Certificate

This is to certify that the Major Project entitled **"Cellular Gateway Development using Linux Embedded Platform"** submitted by **KHOSLA ROHAN (17MECE17)**, towards the fulfillment of the requirements for the degree of Master of Technology in Embedded Systems, Nirma University, Ahmedabad is the record of work carried out by her under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination.The results embodied in this major project, to the best of our knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Date: Place: Ahmedabad

Dr. Sachin Gajjar
Internal Guide
EC Department,

Dr. Nagendra Gajjar
PG Coordinator,
M.Tech - ECE

Dr. Dilip Kothari
Professor and Head,EC

Dr. Alka Mahajan
Director,

Statement of Originality

I, **KHOSLA ROHAN**, Roll. No. **17MECE17**, give undertaking that the Major Project entitled "**Cellular Gateway Development using Embedded Linux Platform**" submitted by me, towards the fulfillment of the requirements for the degree of Master of Technology in **Electronics and communication (Embedded System)** of Institute of Technology, Nirma University, Ahmedabad, contains no material that has been awarded for any degree or diploma in any university or school in any territory to the best of my knowledge. It is the original work carried out by me and I give assurance that no attempt of plagiarism has been made.It contains no material that is previously published or written, except where reference has been made. I understand that in the event of any similarity found subsequently with any published work or any dissertation work elsewhere; it will result in severe disciplinary action.

Date:

Place:

Endorsed by

Dr. Sachin Gajjar

Acknowledgements

I would like to express my gratitude and sincere thanks to **Dr. N.P. Gajjar**, PG Coordinator of M.Tech Embedded Systems and **Dr. Sachin Gajjar** for guidelines during the review process.

I take this opportunity to express my profound gratitude and deep regards to **Dr. Sachin Gajjar**, guide of my internship project for her exemplary guidance, monitoring and constant encouragement.

I would also like to thank **Mr. Milap Patel**, external guide of my internship project from **Masibus Automation and instrumentation Pvt. Ltd.**, for guidance, monitoring and encouragement regarding the project.

- KHOSLA ROHAN

17MECE17

Contents

List of Figures

Abstract

A cellular gateway is a portable device that will provide reliable and flexible internet access for monitoring and managing remote infrastructure with secure connectivity to one or many devices. It will give real time awareness of the operations. Cellular gateway will convert data from one format to another. This device will work as IP packet router or Network Address Translation device that will use mobile data for its Internet connection. Available data logging and monitoring devices were having Ethernet connectivity with them to transfer data to and fro which restricted there deployment in remote areas. Cellular gateway will provide internet access for data logging in remote power plants. The software application acts as a gateway between the cellular and the wired network; it is responsible for supporting the services provided by the wireless network and make them accessible and usable. The gateway device can be integrated easily on any complex hardware.

Abbreviation Notation and Nomenclature

TCP ..... Transmission control protocol
IP ..... Internet protocol
RTU ..... Remote terminal unit
S2E ..... Serial to Ethernet
SPI ..... Serial Peripheral Interface
DHCP ..... Dynamic Host Configuration Protocol
IDE ..... Integrated development environment
UART ..... Universal Asynchronous Receiver/Transmitter
LAN ..... Local Area Network
HVAC ..... Heating,ventilation and air conditioning
OEM ..... Original Equipment Manufacturer

Chapter 1

Introduction

1.1 Motivation

A cellular gateway is a portable device that will provide reliable and flexible internet access for monitoring and managing remote infrastructure with secure connectivity to one or many devices. It will give real time awareness of the operations. Cellular gateway will convert data from one format to another. This device will work as IP packet router or Network Address Translation device that will use mobile data for its Internet connection.

1.2 Limitations

Available data logging and monitoring devices were having Ethernet connectivity with them to transfer data to and fro which restricted there deployment in remote areas. Cellular gateway will provide internet access for data logging in remote power plants.

1.3 Company Overview

The Brand Masibus started its journey in 1975 as a tiny low company with some of individuals, and has currently fully grown into an outsized organization with over two hundred individuals in its men. Having been around for over four decades, the corporate continues to be powerfully stock-still within the moral principles ordered down by the institution fathers. when the initial baby steps, and with the Indian economy gap up in 1991, Masibus steady distended operations and stirred to our current headquarters in Gandhinagar, Gujarat. Since then, we have a tendency to still function answer suppliers in industrial automation and instrumentation phase to customers at intervals the country and across the world.[1]

I am operating with Data Acquisition team. Applications Of data acquisition products were multiple channel scanners, data monitoring for solar power plants, data loggers and string box monitors. Recognized united of the premier industrial automation answer suppliers, Masibus serves nearly 10,000 customers in regarding fifty vertical industrial segments, giving product, solutions and services through eight regional offices and a large network of Dealers and System integrator.

Masibus has been fast to adopt and incorporate new technologies like local area network and Wireless into the economic automation solutions that are offered.The export-worthy quality of our product and solutions stands out as someone for Masibus changing into a globally accepted company, facilitating U.S. to seek out our footing within the Near East, Africa, Europe and much East countries.Masibus operates a global workplace in Sharjah,UAE.[1]

1.4 Scope of Work

To develop cellular gateway on readily available embedded development board. As data acquisition devices were restricted for connectivity in remote areas, this device will provide remote area connectivity to data acquisition devices in remote and

hazardous locations. Here the phytec company's Segin imx6ul development board was given bring-up with Yocto OS and bitbake task executor to build particular packages from the recipes for the hardware. The image files generated from the bitbake process were dumped in the SD memory card and the card was inserted in the development board which will work as the memory for the board. After the booting process of the hardware, Ethernet interfaces were activated to provide remote connection of the board to the host computer. Then communication protocols were implemented on the board using embedded C and linux commands to make them communicate with other devices and analog sensors. To implement the communication protocols, particular pins of the required ports were made available through i/o pin-muxing in the device tree file of the development board. Here communication protocols UART, Modbus protocol over RS232 port, Serial peripheral interface(SPI) with ADC chip were made available on the board. GSM module with in-built TCP/IP stack available was interfaced with the development board to provide internet access to the device in remote areas. [2]

1.5 Outline of Thesis

In this thesis six chapters are there. First chapter is all about motivation of project, Limitation of Project, overview of company and Scope of work. Second Chapter is all about literature survey were all the manuals related to hardware schematic of development board and its booting process were referred. Third chapter is all about the communication protocols that were implemented on the board to connect the device with other peripherals ADC chip. Fourth chapter is all about interfacing of GSM module with the development board and its AT commands. Last and fifth chapter is about Software flow chart. It describes about programming flow of the device. The tools and technologies required in my project. Sixth chapter is all about the conclusion of my project which is basically to monitor the real time data from remote areas via cellular gateway.

Chapter 2

Board boot-up with Yocto OS and Bitbake

2.1 Yocto Project

The Yocto Project is an open source collaboration project that helps developers create custom Linux-based systems that are designed for embedded products regardless of the product's hardware architecture. Yocto Project provides a flexible tool set and a development environment that allows embedded device developers across the world to collaborate through shared technologies, software stacks, configurations, and best practices used to create these tailored Linux images.

Thousands of developers worldwide have discovered that Yocto Project provides advantages in both systems and applications development, archival and management benefits, and customization's used for speed, footprint, and memory utilization. The project is a standard when it comes to delivering embedded software stacks. The project allows software customizations and build interchange for multiple hardware platforms as well as software stacks that can be maintained and scaled.

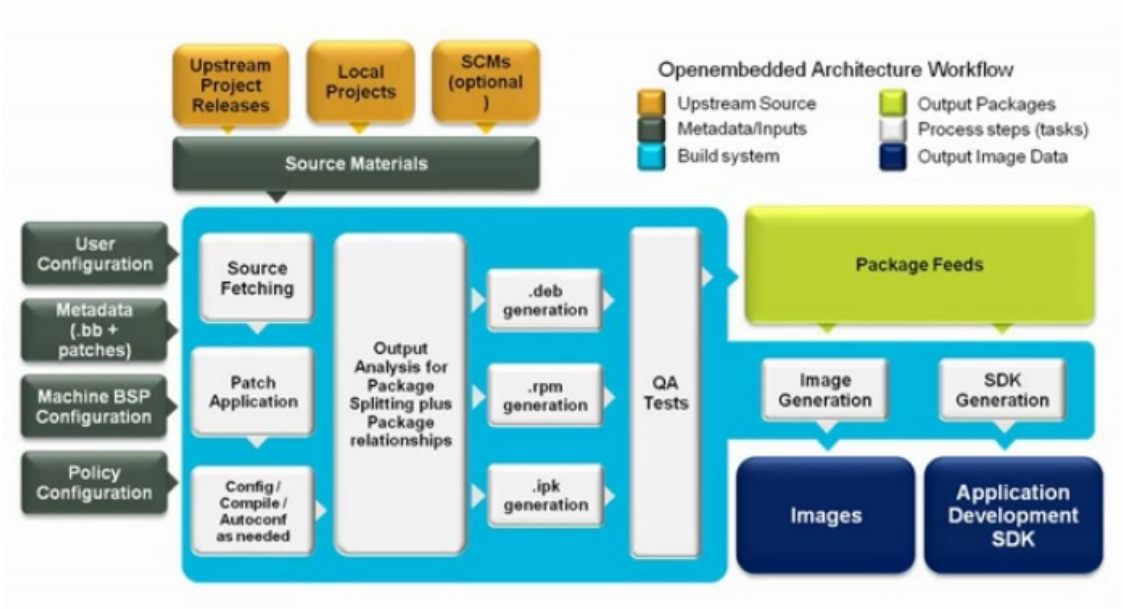

Figure 2.1: Yocto Procedure

2.1.1 Cross Development Tool chain Generation

The Yocto Project does most of the work for you when it comes to creating cross-development tool chains. This section provides some technical background on how cross-development toolchains are created and used. For more information on toolchains, you can also see the Yocto Project Application Development and the Extensible Software Development Kit (eSDK) manual.

In the Yocto Project development environment, cross-development toolchains are used to build images and applications that run on the target hardware. With just a few commands, the OpenEmbedded build system creates these necessary toolchains for you.

The following figure shows a high-level build environment regarding toolchain construction and use.

Poky is the Yocto Project reference system and is composed of collection of tools and metadata. Poky is platform-independent and performs cross-compiling, using Bitbake Tool, OpenEmbedded Core, and a default set of metadata. The main

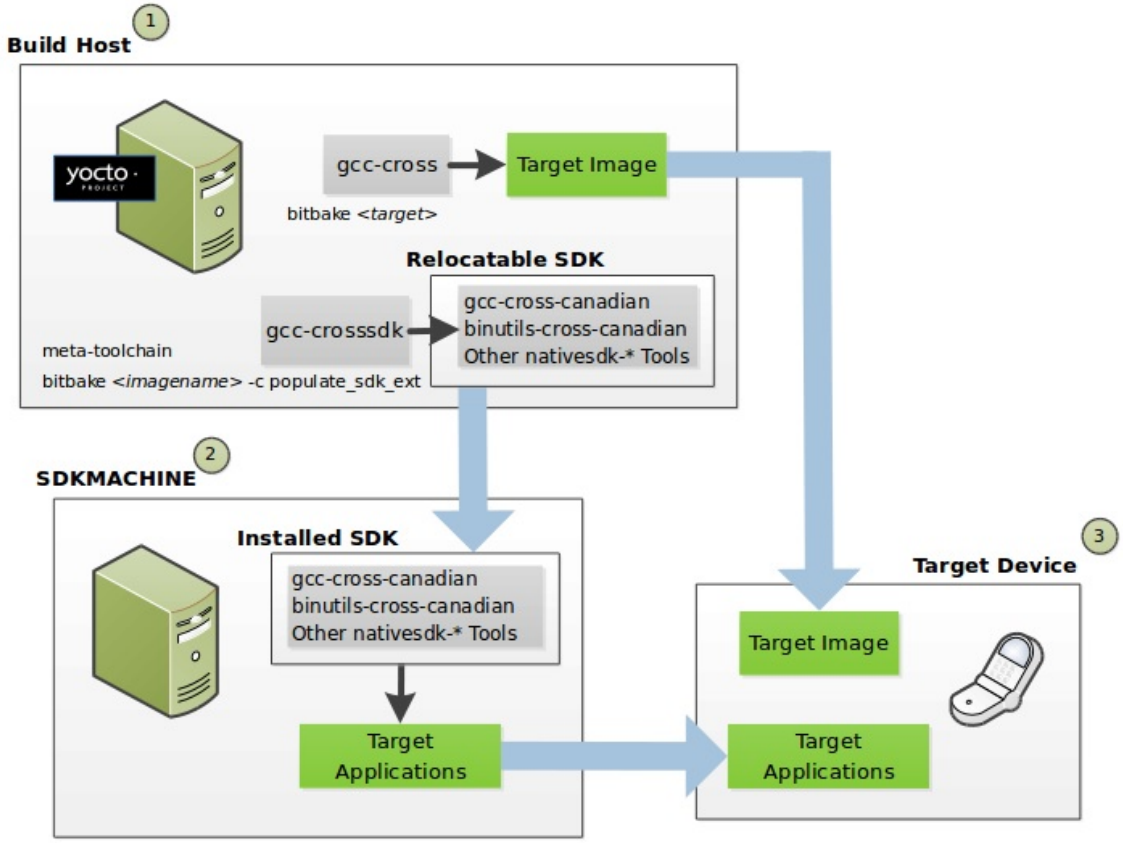

Figure 2.2: Cross Toolchain

objective of Poky is to provide all the features an embedded developer needs.

Bitbake is a task scheduler that parses Python and Shell script mixed code, which we called Recipes. The code parsed generates and runs tasks. They are a set of steps orders according to the code's dependencies.

Metadata is where all the Recipes are located. Metadata is composed of a mix of Python and Shell Script text files. Poky uses this to extend OpenEmbeddded Core, meta-yocto, and meta-yocto-bsp.

Recipes (.bb files) are fundamental components in the Yocto Project environment. Each software component built by the OpenEmbedded build system requires a recipe to define the component.

2.2 Bitbake Tool

BitBake is a generic task execution engine that allows shell and Python tasks to be run efficiently and in parallel while working within complex inter-task dependency constraints. One of BitBake's main users, OpenEmbedded, takes this core and builds embedded Linux software stacks using a task-oriented approach. Conceptually, BitBake is similar to GNU Make in some regards but has significant differences:

BitBake executes tasks according to provided metadata that builds up the tasks. Metadata is stored in recipe (.bb), configuration (.conf), and class (.bbclass) files and provides BitBake with instructions on what tasks to run and the dependencies between those tasks. BitBake includes a fetcher library for obtaining source code from various places such as source control systems or websites. The instructions for each unit to be built (e.g. a piece of software) are known as recipe files and contain all the information about the unit (dependencies, source file locations, checksums, description and so on). BitBake includes a client/server abstraction and can be used from a command line.

2.3 Phytec Segin Imx6ul Board

Steps to boot up Phytec Segin Board

1. Installing additional packages for host PC.
2. Git installation and configuration for host PC
3. Setup Site.conf The download directory is a place where Yocto stores all sources fetched from the internet.

It can contain tar.gz, Git mirror or anything else. It is very useful to set this to a common shared location on the machine. Create this directory with 777 access rights. To be able to share this directory with different users all files need to have group write access. This will most probably be in conflict with default umask

settings. One possible solution would be to use ACLs for this directory. The cache directory stores all stages of the build process. Create the two directories on a drive where you have approximately 50 GB of space and assign the two variables.[3]

4. Initialization of Board Support Package. Create a fresh project folder. Download and run the phyLinux script

5. Select the SOC in the script

6. Select the BSP version in the script

7. Select the Phytec Board in the script

8. Setup shell environment for the image builds

9. Bitbake build minimal core image for the board boot-up

10. Select the BSP images All images generated by Bitbake are deployed to machine folder in yocto build directory.

The following list shows for example all files generated for the i.MX 6 SoC, phyboard-segin-imx6ul-2 machine:

Barebox: barebox.bin Barebox configuration: barebox-defconfig Kernel: zImage Kernel device tree file: zImage-imx6ul-phytec-phyboard-segin-ff-rdk.dtb Kernel configuration: zImage.config Root filesystem: phytec-qt5demo-image-phyboard-segin-imx6ul-2.tar.gz, phytec-qt5demo-image-phyboard-segin-imx6ul-2.ubifs, phytec-qt5demo-image-phyboard-segin-imx6ul-2.ext4 SD card image: phytec-qt5demo-image-phyboard-segin-imx6ul-2.sdcard

11. Preparing bootable SD Card There are two ways to create a bootable SD card. You can either use:

- a single prebuild SD card image, or

- the four individual images (barebox-, kernel- and device tree image, and root filesystem).

Using four individual Images (barebox-, kernel- and device tree image, and root filesystem) Instead of using the single prebuild SD card image, you can also use the barebox-, kernel- and device tree image together with the root filesystem to create a bootable SD card manually. For this method a new card must be setup with 2

partitions and 8 MB of free space at the beginning of the card. Use the following procedure with fdisk under Linux: Create a new FAT partition with partition id C. When creating the new partition you must leave 8 MB of free space at the beginning of the card. When you go through the process of creating a new partition, fdisk lets you specify where the first sector starts. During this process fdisk will tell you where the first sector on the disk begins. If, for example, the first sector begins at 1000, and each sector is 512 bytes, then 8 MB / 512 bytes = 16384 sectors, thus your first sector should begin at 17384 to leave 8 MB of free space. The size of the FAT partition needs only be big enough to hold the zImage which is only a few megabytes. To be safe we recommend a size of 64 MB. Create a new Linux partition with partition id 83. Make sure you start this partition after the last sector of partition 1! By default fdisk will try to use the first partition available on the disk, which in this example is 1000. However, this is our reserved space! You must use the remaining portion of the card for this partition. Write the new partition to the SD card and exit fdisk. Create a file system on the partitions with (replace 'sde' with your device): Write the bootloader in front of the first partition (replace 'sde' with your device): In case you want to boot the whole Linux from SD card, also mount the ext4 partition. Then untar image to it.[3]

12. Install Putty or Minicom (Serial Console interface software) to connect host pc to embedded board.

13. Booting the Embedded Board Start the board and connect the USB to Serial connector to the host pc and board respectively and press m to stop autoboot. You will get a menu: Main menu 1: Boot default 2: Detect bootsources 3: Settings 4: Save environment 5: Shell 6: Reset Enter 2 and select the SD Card or other prefarable option.

14. Connect the Ethernet cable to the board and detect IP allocated

15. Download the tool-chain for the imx6ul processor from the Phytec FTP server.

16. Go to the folder containing binary files of the arm toolchain i.e /bin

17. Check the generated executable object file.

18. Open the serial terminal window and detect the IP address allocated to the board. Connect the board through SSH connection port 22 from Putty console window and tranfer the object file to the embedded board using scp commands.

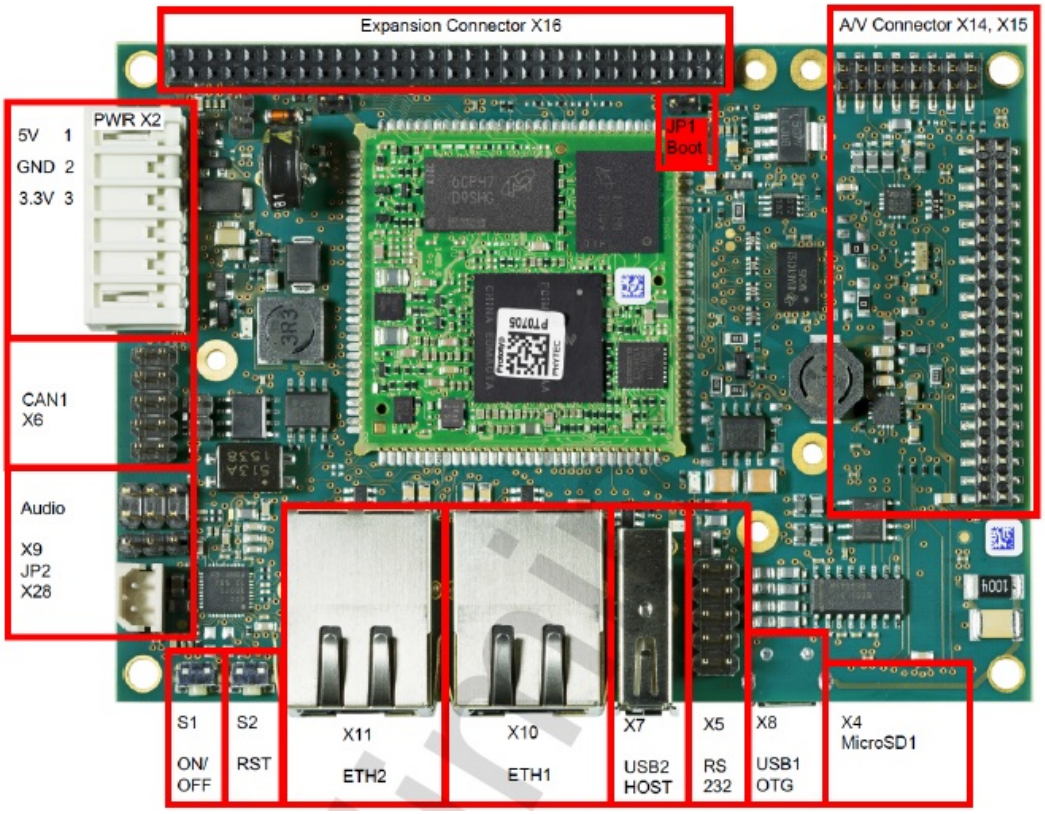

Figure 2.3: Phytec Segin Imx6ul

The board features with processor family of NXP imx6ul, processor architecture of ARM Cortex A7 and has a frequency of 696 MHz. It has Nand flash and DDR3 Ram upto 2 GB. The development board also includes 2 serial ports, 2 ethernet ports providing speed upto 100Mbps. [4]

Chapter 3

Implementation of Communication Protocols

3.1 About Modbus

Modbus devices communicate employing a master-slave(client-server) technique during which just one device (the master/client) will initiate transactions that's known as queries.The other devices like slaves or servers respond by supply the requested knowledge to the master,or by taking the exploit requested within the question. A slave is any peripheral device that processes information and sends its output to the master victimization Modbus. Masters will address individual slaves, or will initiate a show message to any or all slaves. Slaves come a response to any or all queries addressed to them singly, however don't reply to show queries. Slaves don't initiate messages on their own,they solely respond to queries from the master[5]

A masters question can include a slave address a operate code shaping the requested action, any required knowledge, and miscalculation checking field. A slaves response consists of fields approving the action taken, any knowledge to be came back, and miscalculation checking field.

3.2 Modbus Functions and Registers

The TCP/IP protocol suite provides all the resources for 2 devices to speak with one another over an local area network computer network or world WAN.Modbus is associate degree application protocol or electronic messaging structure that defines rules for organizing and decoding information freelance of the information transmission medium. ancient serial Modbus may be a register-based protocol that defines message transactions that occur between masters and slaves. Slave devices listen for communication from the master and easily respond as tutored. The master always that controls the communication and should communicate on to one slave, or all connected slaves, however the slaves cannot communicate directly with one another..[5]

The Modbus information model incorporates a straight forward structure that solely differentiates between four basic information types: 1) Discrete Inputs 2) Coils(Outputs) 3) Input Registers(Input Data) 4) Holding Registers(Output Data)

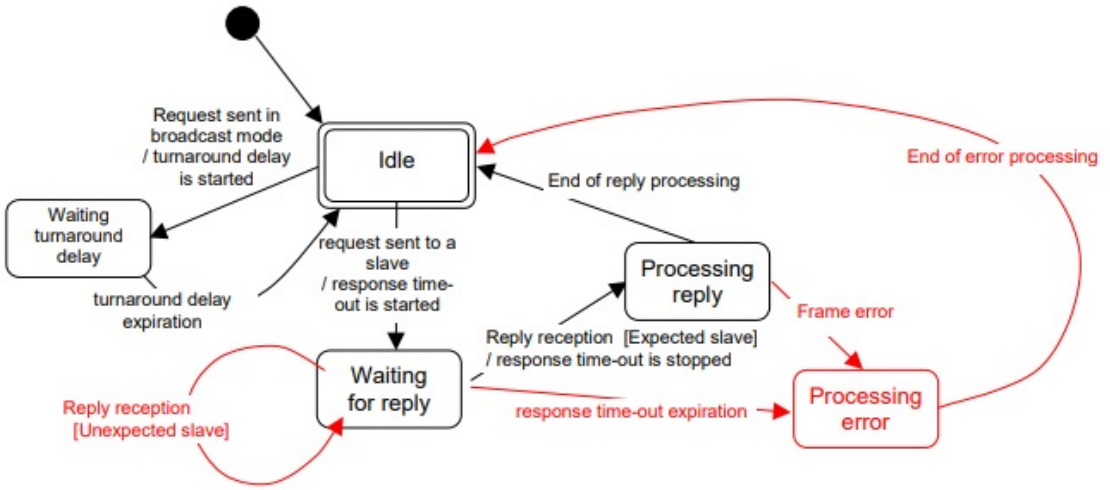

Figure 3.1: Modbus RTU Master state

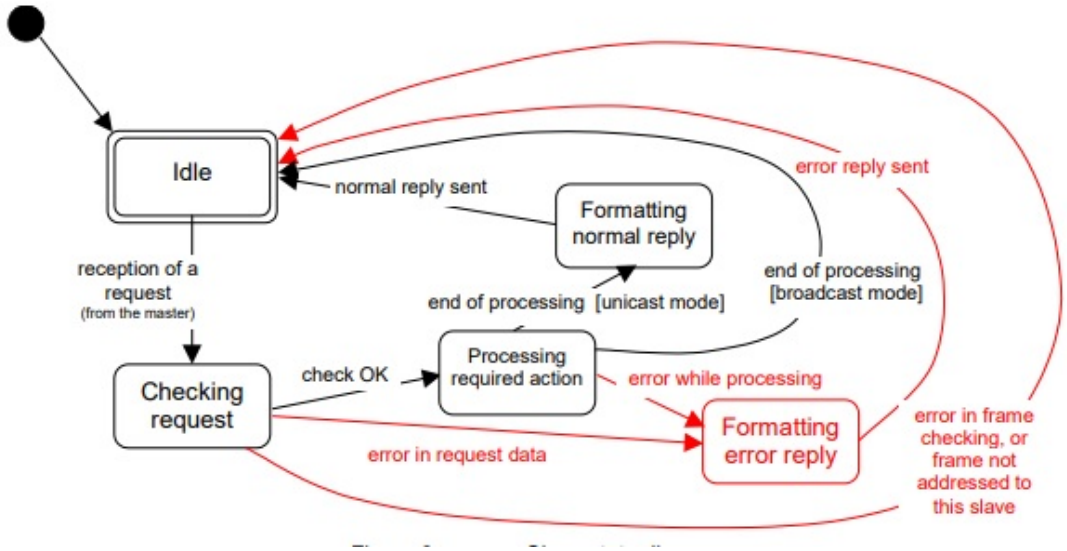

Figure 3.2: Modbus RTU Slave state

3.3 Using SPIDEV on Linux

The linux/spi/spi.h header file includes kernel doc, as does the main source code. SPI requests always go into I/O queues. Requests for a given SPI device are always executed in FIFO order, and complete asynchronously through completion callbacks. There are also some simple synchronous wrappers for those calls, including ones for

common transaction types like writing a command and then reading its response.[3]

There are two types of SPI driver, here called:

Controller drivers ... controllers may be built into System-On-Chip processors, and often support both Master and Slave roles. These drivers touch hardware registers and may use DMA. Or they can be PIO bitbangers, needing just GPIO pins.

Protocol drivers ... these pass messages through the controller driver to communicate with a Slave or Master device on the other side of an SPI link.

There is a minimal core of SPI programming interfaces, focusing on using the driver model to connect controller and protocol drivers using device tables provided by board specific initialization code.

3.4 SPI Protocol Overview

Electrically-coded information is called a serial data, which is transmitted bit by bit from one device to another through a set of protocols. In the embedded system, control sensors and actuators data is received or transmitted to the controller devices such as microcontrollers so that the data is further analyzed and processed. As the microcontrollers work with the digital data, the information from the analog sensors, actuators and other peripherals is converted into one byte (8-bit) binary word prior to being transmitted to the microcontroller.[5]

3.5 SPI Protocol

The client request information field provides the slave(server) with any extra data needed by the slave to complete the action nominal by the perform code within the client request.The information field usually includes register addresses,count values, and written information. for a few messages,this field might not exist(has zero length), as not all messages would require information. The SPI communication

stands for serial peripheral interface communication protocol, which was developed by the Motorola in 1972. SPI interface is available on popular communication controllers such as PIC, AVR, and ARM controller, etc. It has synchronous serial communication data link that operates in full duplex, which means the data signals carry on both the directions simultaneously.

SPI protocol consists of four wires such as MISO, MOSI, CLK, SS used for master/slave communication. The master is a microcontroller, and the slaves are other peripherals like sensors, GSM modem and GPS modem, etc. The multiple slaves are interfaced to the master through a SPI serial bus. The SPI protocol does not support the Multi-master communication and it is used for a short distance within a circuit board.[6]

When the slave device responds to the master, it uses the perform code field to point either a traditional (error-free) response, or that some quite error has occurred (an exception response). a traditional response merely echoes the initial perform code of the question, whereas associate degree exception response returns a code that's equivalent to the initial perform code with its most vital bit(msb) set to logic one..

The Read Holding Registers command has the perform code 0000 0011 (03H). If the slave device takes the requested action while not error, it returns the identical code in its response. However, if associate degree exception happens, it returns a thousand 0011 (83H) within the perform code field and appends a singular code within the information field of the response message that tells the master device what quite error occurred, or the explanation for the exception..[5]

- MISO (Master in Slave out): The MISO line is configured as an input in a master device and as an output in a slave device.
- MOSI (Master out Slave in): The MOSI is a line configured as an output

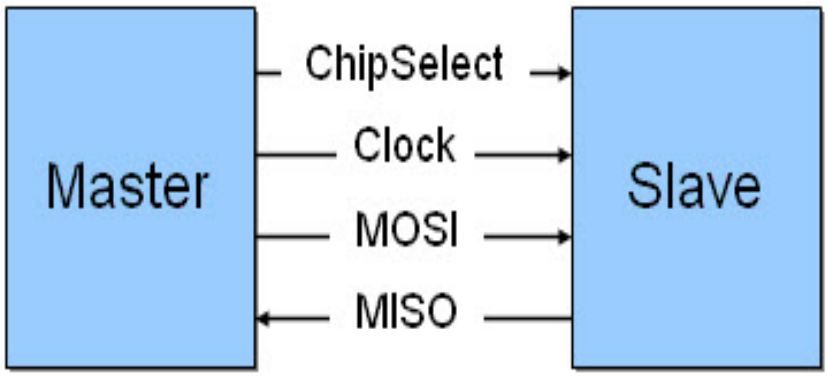

Figure 3.3: SPI Protocol

in a master device and as an input in a slave device wherein it is used to synchronize the data movement.

- SCK (serial clock): This signal is always driven by the master for synchronous data transfer between the master and the slave. It is used to synchronize the data movement both in and out through the MOSI and MISO lines.

- SS (Slave Select) and CS (Chip Select): This signal is driven by the master to select individual slaves/Peripheral devices. It is an input line used to select the slave devices.

3.5.1 SPI Communication Protocol

Many microcontrollers have inbuilt SPI protocols that handle all of the sending and receiving data. Any of the data mode operations (R/W) is controlled by a control and status registers of the SPI Protocol. Here, you can observe the EEPROM interface to the PIC16f877a microcontroller through the SPI protocol.

Here, 25LC104 EEROM is a 131072 bytes memory wherein the microcontroller transfers two bytes of data to the EEROM memory through a SPI serial bus.

3.5.2 SPI with ADC

Complete SPI transaction for the MCP3551. Phytec board asserts the chip select signal connected to the ADC by setting it to 0V. This is typically taken care of internally by the spidev driver whenever the proper ioctl() function is called. Phytec board sends a byte containing a value of '1' to the ADC. This is a start bit. At the same time the ADC sends back a 'don't care' byte to the Phytec board.[3] Phytec board then sends a second byte whose most significant nibble (SGL/DIFF,D2,D1 D0 bits) indicate the channel that we want to convert and whether we want single-ended or differential conversion. For example if this nibble is "1000", the conversion will be single-ended and take place on channel 0 (CH0 pin). The least significant nibble is sent as 'don't care'. At the same time, the ADC sends back the two most significant bits of the digital value (result) of the conversion (bits 8 and 9). Phytec board sends another 'don't care' byte to the ADC. At the same time the ADC send back a byte containing bits 7 through 0 0f the digital value (result) of the conversion. The Phytec board then merges bits 8 9 from the second received byte with bits 7 through 0 from the third received byte to create the 10-bit digital value resulting from the conversion.[3]

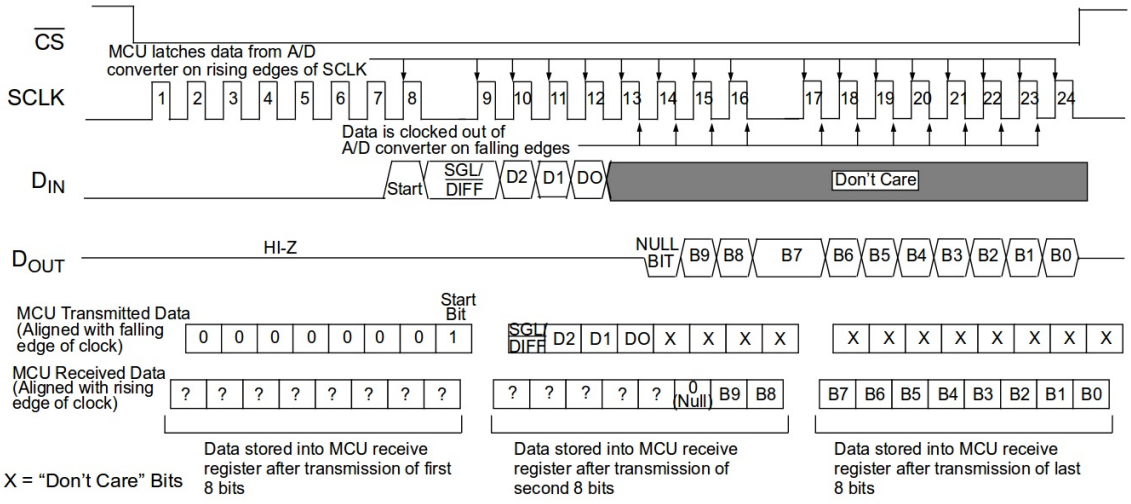

Figure 3.4: SPIDEV communication with ADC

Chapter 4

GSM Modem Interfacing

4.1 GSM Modem

SIM800C GSM/GPRS RS232 MODEM (DB9), the latest addition to rhydoLABZ GSM/GPRS modem, helps to add wireless connectivity to your project using RS232 UART interface. It is built with Quad Band GSM/GPRS engine SIM800C, that works on frequencies 850/ 900/ 1800/ 1900 MHz.You can connect the modem directly to PC as well as microcontroller with RS232 Chip(MAX232). The baud rate is configurable from 9600-115200 through AT command. The GSM/GPRS Modem is having internal TCP/IP stack to enable you to connect with internet via GPRS. The Modem is manufactured with Automatic Pick and place machine with high quality standard.The onboard Low dropout 3A Power supply allows you to connect wide range unregulated power supply . Using this modem,you can make audio calls, SMS, Read SMS, attend the incoming calls and internet etc through simple AT commands. [7]

Features of the Modem are that it Make and receive voice calls, Send and receive SMS messages, Send and receive GPRS data (TCP/IP, HTTP, etc.). Configurable Baud rate (9600-115200, factory default value: 9600) AT command interface Input Voltage : 5V-12 V, Sim card socket Network, Status and Power indication LEDs,

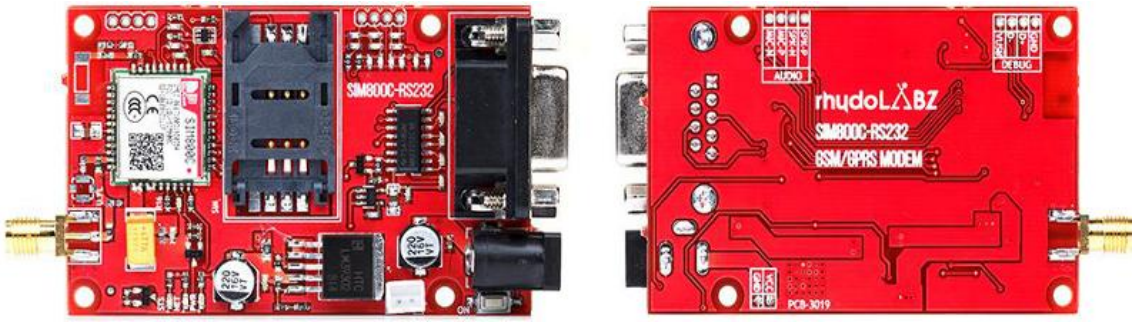

Figure 4.1: GSM Modem

Network, modem status,CTS/RI and RTS/RF SYNC can be taken via DB9 connector On board SMA connector for GSM antenna. Provision for UFL connector for external antenna. Option for audio interface Normal operation temperature: -40 C to +85 C. Provision for firmware updation. Low power consumption. M39302 linear voltage regulator is used for provides a low-dropout, high-current output Onboard MAX232 IC for level conversion. Power supply voltage is 3.3V 4.4V.

Both 1.8 volts and 3.0 volts SIM Cards are supported by SIM800C .the SIM card voltage type is automatically detected.[7]

Getting Started with Sim800C Assemble GSM Antenna to the modem Connect serial cable to the modem. Give power supply in between 5V to 12V through the power jack provided. Default factory Baud rate is 9600. When the modem is successfully powered-up, the PWR LED (RED) on the modem will be ON, the STS LED (GREEN) will light after 1-2 seconds and the NET LED(BLUE) will blink every second. After the Modem registers in the network (takes between 10-60 seconds), this LED will blink in step of 3 seconds.

Testing Sim800C You can directly connect SIM800C GSM/GPRS RS232 modem (DB9) to your PC through serial port, no need of any interfacing modules. If serial port is not there in your PC (some laptops doesnt have serial port inter-

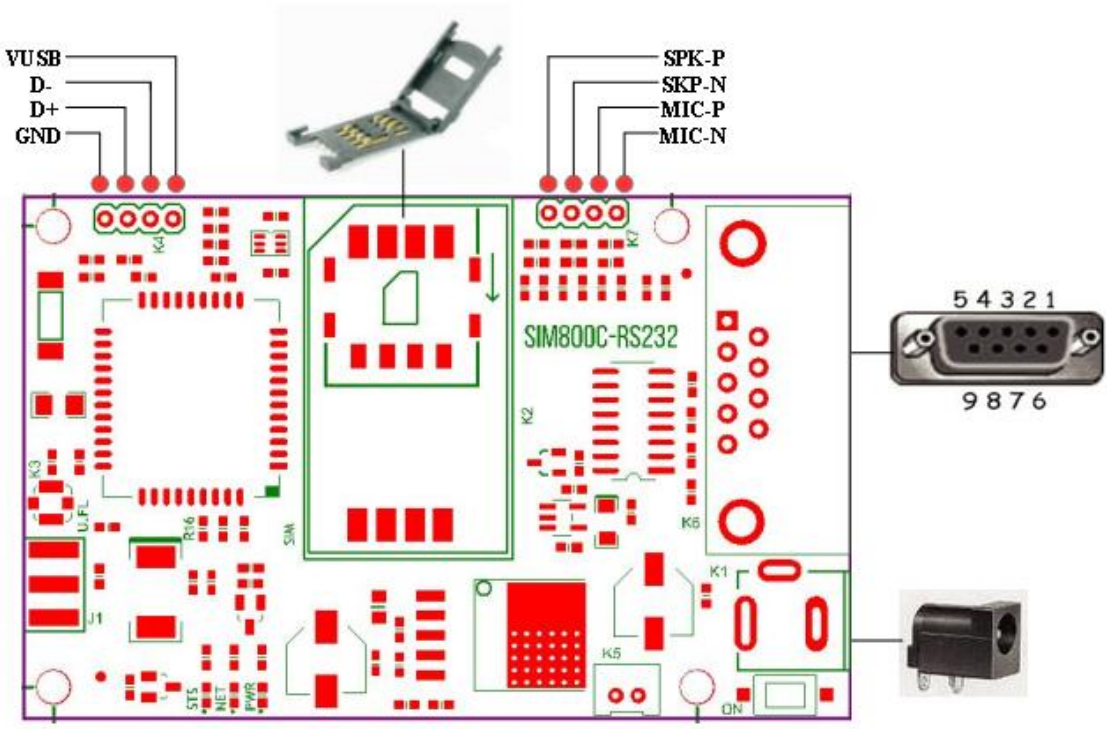

Figure 4.2: Pin diagram of the modem

face) then you can use USB to serial Adapter(DB9) for this purpose. Once you are done with all these procedures, check appropriate COM port that got assigned to the Communication Port in your system by looking into the device manager like as shown below.

Sending Message using AT Commands Send a message using SIM800C GSM/GPRS RS232 modem (DB9).For sending message we need to first send related AT commands to initialize the modem to send a message. AT command after OK response, this signifies that our Modem is working properly. ATE0 command is being sent to stop the echo.

Recieving Message in Modem Receive a message usingSIM800C GSM/GPRS RS232 modem (DB9). For receiving message we need to first send related AT commands to initialize the modem. AT+CNMI=2,1,0,0,0 command (followed by enter) set the modem to indicate received messages with +CMTI response indicates

that new message has been received and shows location number of last received message in above figure it is shown by number 20. Value in the range of location numbers supported by the associated memory and gives +CMGR response which includes status i.e REC UNREAD Received unread messages,Senders number, date, time and received message.

4.2 Socket programming for TCP/IP layer

The call to the function socket() creates an UN-named socket inside the kernel and returns an integer known as socket descriptor. This function takes domain/family as its first argument. For Internet family of IPv4 addresses we use AFINET. The second argument SOCKSTREAM specifies that the transport layer protocol that we want should be reliable ie it should have acknowledgement techniques. For example TCP. The third argument is generally left zero to let the kernel decide the default protocol to use for this connection. For connection oriented reliable connections, the default protocol used is TCP.

After the call to listen(), this socket becomes a fully functional listening socket. In the call to accept(), the server is put to sleep and when for an incoming client request, the three way TCP handshake* is complete, the function accept () wakes up and returns the socket descriptor representing the client socket. The call to accept() is run in an infinite loop so that the server is always running and the delay or sleep of 1 sec ensures that this server does not eat up all of your CPU processing. As soon as server gets a request from client, it prepares the date and time and writes on the client socket through the descriptor returned by accept(). Three way handshake is the procedure that is followed to establish a TCP connection between two remote hosts. We might soon be posting an article on the theoretical aspect of the TCP protocol.[7]

Here we create a client which will connect to the server and receive date and time from it. In the above piece of code :

We see that here also, a socket is created through call to socket() function. Information like IP address of the remote host and its port is bundled up in a structure and a call to function connect() is made which tries to connect this socket with the socket (IP address and port) of the remote host. Note that here we have not bind our client socket on a particular port as client generally use port assigned by kernel as client can have its socket associated with any port but In case of server it has to be a well known socket, so known servers bind to a specific port like HTTP server runs on port 80 etc while there is no such restrictions on clients.[8] Once the sockets are connected, the server sends the data (date+time) on clients socket through clients socket descriptor and client can read it through normal read call on the its socket descriptor.

Chapter 5

Work done in Masibus

5.1 Manual Study

I have studied 3 manuals and referred some videos for Yocto OS. To develop cellular gateway on readily available embedded development board. As data acquisition devices were restricted for connectivity in remote areas, this device will provide remote area connectivity to data acquisition devices in remote and hazardous locations. Here the phytec company's Segin imx6ul development board was given bring-up with Yocto OS and bitbake task executor to build particular packages from the recipes for the hardware. The image files generated from the bitbake process were dumped in the SD memory card and the card was inserted in the development board which will work as the memory for the board. After the booting process of the hardware, Ethernet interfaces were activated to provide remote connection of the board to the host computer. Then communication protocols were implemented on the board using embedded C and linux commands to make them communicate with other devices and analog sensors. To implement the communication protocols, particular pins of the required ports were made available through i/o pin-muxing in the device tree file of the development board. Here communication protocols UART, Modbus protocol over RS232 port, Serial peripheral interface(SPI) with ADC chip

were made available on the board. GSM module with in-built TCP/IP stack available was interfaced with the development board to provide internet access to the device in remote areas.

5.1.1 GDB Debugging

Stepping Through Code Stepping lets you trace the path of your program, and zero in on the code that is crashing or returning invalid input.

Run program until next line, then pause. If the current line is a function, execute the entire function, then pause. Next is good for walking through your code quickly.

Run the next instruction, not line. If the current instructions is setting a variable, it is the same as next. If its a function, it will jump into the function, execute the first statement, then pause. Step is good for diving into the details of your code.
finish

Finish executing the current function, then pause (also called step out). Useful if you accidentally stepped into a function.

Breakpoints And Watchpoints Breakpoints are one of the keys to debugging. They pause (break) a program when it reaches a certain location. You can examine and change variables, then resume execution. This is helpful when seeing why certain inputs fail, or testing inputs.

Set a breakpoint at line 45, or at myfunction. The program will pause when it reaches the breakpoint.
watch x == 3 Set a watchpoint, which pauses the program when a condition changes (when x == 3 changes). Watchpoints are great for certain inputs (myPtr != NULL) without having to break on every function call. continue Resume execution after being paused by a breakpoint/watchpoint. The program will continue until it hits the next breakpoint/watchpoint. delete N Delete breakpoint N (breakpoints are

numbered when created). Setting Variables And Calling Functions Viewing and changing variables at run-time is a huge part of debugging. Try giving functions invalid inputs or running other test cases to find the root of problems. Typically, you will view/set variables when the program is paused.

print x Print current value of variable x. Being able to use the original variable names is why the (-g) flag is needed; programs compiled regularly have this information removed. set x = 3 set x = y Set x to a set value (3) or to another variable (y) call myfunction() call myotherfunction(x) call strlen(mystring) Call user-defined or system functions. This is extremely useful, but beware calling buggy functions.

Constantly display value of variable x, which is shown after every step or pause. Useful if you are constantly checking for a certain value. Use undisplay to remove the constant display.

Backtrace And Changing Frames The stack is a list of the current function calls it shows you where you are in the program. A frame stores the details of a single function call, such as the arguments.

Backtrace, aka print the current function stack to show where you are in the current program. If main calls function a(), which calls b(), which calls c(). Move to the next frame up or down in the function stack. If you are in c, you can move to b or a to examine local variables.Return from current function. Crashes And Core Dumps

A core dump is a snapshot of memory at the instant the program crashed, typically saved in a file called core. GDB can read the core dump and give you the line number of the crash, the arguments that were passed, and more. This is very helpful, but remember to compile with (-g) or the core dump will be difficult to debug.

Debug myprogram with core as the core dump file. Print the backtrace (function stack) at the point of the crash. Examine variables using the techniques above.

Handling Signals Signals are messages thrown after certain events, such as a timer or error. GDB may pause when it encounters a signal; you may wish to ignore them instead.

a. handle [signalname] [action]

b. handle SIGUSR1 nostop

c. handle SIGUSR1 noprint

d. handle SIGUSR1 ignore

Tell GDB to ignore a certain signal (SIGUSR1) when it occurs. There are varying levels of ignoring. Integration With Emacs The Emacs text editor integrates well with GDB. Debugging directly inside the editor is great because you can see an entire screen of code at a time. Use M-x gdb to start a new window with GDB and learn more here.

Chapter 6

Conclusion

6.1 Conclusion

After extensive study, research and feasibility of options for data monitoring in remote areas, an attempt has been made to design an Cellular Gateway. By developing this device, data monitoring can be done in remote areas and its access any where through internet

In this project, the phytec company's Segin imx6ul development board was given bring-up with Yocto OS and bitbake task executor to build particular packages from the recipes for the hardware. The image files generated from the bitbake process were dumped in the SD memory card and the card was inserted in the development board which will work as the memory for the board. After the booting process of the hardware, Ethernet interfaces were activated to provide remote connection of the board to the host computer. Then communication protocols were implemented on the board using embedded C and linux commands to make them communicate with other devices and analog sensors. To implement the communication protocols, particular pins of the required ports were made available through i/o pin-muxing in the device tree file of the development board. Here communication protocols UART, Modbus protocol over RS232 port, Serial peripheral interface(SPI) with

ADC chip were made available on the board. GSM module with in-built TCP/IP stack available was interfaced with the development board to provide internet access to the device in remote areas.

Bibliography

[1] M. Automation and instrumentation Pvt. Ltd., "Details about Masibus." https://www.masibus.com/, 2018. [Online; accessed 6- DEC-2018].

[2] P. Upadhyay, K. Vyas, and A. Vibhakar, "Energy management by high speed remote monitoring of energy meters," *ETCEE–2015*, p. 94, 2015.

[3] P. Board, "Phytec Board ." https://www.phytec.in/, 2018. [Online; accessed 6- DEC-2018].

[4] B. Recipe, "Bit Brake Recipe in Yocto Project." https://www.yoctoproject.org/docs/1.6/bitbake-user-manual/bitbake-user-manual.html///, 2018. [Online; accessed 16-APR-2018].

[5] Wikipedia, "Details about Modbus." https://en.wikipedia.org/wiki/Modbus/, 2018. [Online; accessed 6- DEC-2018].

[6] Wikipedia, "Details about Modbus." https://www.rtaautomation.com/technologies/modbus-tcpip//, 2018. [Online; accessed 6- DEC-2018].

[7] G. Modem, "GSM -Architecture, Feature Working." https://www.elprocus.com/gsm-architecture-features-working/, 2018. [Online; accessed 6- DEC-2018].

[8] H. Das and L. C. Saikia, "Ethernet based smart energy meter for power quality monitoring and enhancement," in *2017 Recent Developments in Control, Automation Power Engineering (RDCAPE)*, pp. 187–191, Oct 2017.