IoT Based Real-Time Synchronized Clock

A Thesis
Submitted to the Department of Electrical and Electronics Engineering Of
BRAC University
By

Mohammad Moinul Haq – 11121086
Aakash Rabbi- 12221008
Mustafa Jamil – 13221020
Araf Zahin- 13221019

Supervised by
Dr. A. K. M. Abdul Malek Azad
Professor
Department of Electrical and Electronic Engineering,
BRAC University, Dhaka.

In partial fulfillment of the requirements for the degree of

Bachelor of Science in Electrical and Electronic Engineering

Spring 2018, BRAC University, Dhaka

DECLARATION

We hereby declare that research work titled "*IoT Based Real-Time Synchronized Clock"* is our own work. The work has not been presented elsewhere for assessment. The places where material has been used from other sources, it has been properly referred.

Signature of	Signature of
Supervisor	Authors
……..	……
Dr. A. K. M. Abdul Malek Azad	Mohammad Moinul Haq
	……
	Aakash Rabbi
	……
	Mustafa Jamil
	……
	Araf Zahin

ACKNOWLEDGEMENTS

Firstly, we would like to thank Dr. A.K.M Abdul Malek Azad, Professor, Dept. of Electrical and Electronic Engineering (EEE), BRAC University; for his supportive guidance and feedbacks for completion of the thesis. Secondly, our gratitude is towards BRAC University for giving us incredible facilities. Last but not least, we would also like to thank every other individual whose contribution led to the theses' being a success.

ABSTRACT

Maintaining the perfect timing is a necessity in every human life. We all are accustomed to clocks, both the analog and its digital counterpart. But, different clocks, even within the same geographical position, show different times. They are not synchronized. This problem of de-synchronization and all the trouble it causes has motivated us to look for a feasible solution. Keeping in mind the course of current technological progress, we have proposed an IoT based solution to the problem of de-synchronization. We have come up with a digital clock which is smart enough to update its time with the help of internet by fetching the time from NTP servers at a regular interval and displaying it. We describe, in this paper, our system, its methodology, hardware and software implementation, integration and synthesis of different components, its technical and financial aspects, and finally, the future works.

CONTENTS

Chapter 1

Introduction

1.1: Background & Objective

Our lives, as human beings, are governed by time. It is the intrinsic property of nature that we can't avoid, no matter how hard we try. The productivity and efficiency of our work depends majorly on how well we manage time. As a result we are very familiar with the device that keeps the track of time for us: a clock, which might be digital or analog. Nowadays, as technology has progressed tremendously in last 50 to 100 years, we are much more accustomed towards using a digital clock instead of its analog counterpart. It gives us a more precise version of the quantity of our interest, and also it is much easier to observe. In Bangladesh, specifically, digital clocks are replacing its preceding analog version very fast. Workplaces and offices, schools and universities, transportation and vehicle terminals etc. are using more digital clocks and lesser number of analog ones. But, the problem that lingers, even after this much of advent of technology, is the exactness of time. For example, even in the same office, different clocks show different times. The clock in the managing director's cabin might show a time different from that in the workers' cubicle. To put it more simply, none of the clocks are synchronized. Yes, they might be showing almost the same time with minute disparity, but in many cases, this small difference can pose serious problem such as loss or order and harmony, and lack of discipline.

Our thesis is based on the idea of circumventing this issue. Here, we have modeled a practical solution to the problem both theoretically and physically. The solution involves some of the novel technological approach to the age old problem but considering the availability and flexibility we have in terms of technology nowadays, it is both a pragmatic approach and a feasible solution.

1.2: Contemporary Systems

Different ways to tackle the issue of clock de-synchronization has been implemented in the past. Each way has its share of advantages and disadvantages. Here we will try to realize three primary methods very briefly. Firstly, let us look into synchronization of clocks manually. It is a very cumbersome process. For this process, somebody has to have a reference clock which would show the actual time and the person would then go from place to place to synchronize other clocks with reference clocks by hand using the clocks' set and reset buttons. This process requires a lot of time and energy, and is also very inefficient and imprecise. There is no way to be sure of whether the reference clock itself is showing the correct time or not. Let's look into the second process. This process involves a master clock connected to the designated slave clocks by wires. This process is definitely much more efficient than manual synchronization but there are some drawbacks to it too. As it is a wired system requiring external electrical supply, it would be financially much more demanding because of the extra equipment required and external electrical consumption. There is also the loss of signal due to use of wires. The third and the final process involves the NTP servers. The clocks in our computers and smart devices keep

themselves synchronized with the NPT servers through its protocols. This system is a much more feasible compared to the other two given above. It requires the usage of internet and routers but in the modern society, those are very much available. Our work here involves the third method and we are going to tackle the de-synchronization problem with it.

1.3: Centralized and Decentralized Digital Clock System

In this section we are going to categorize the already mentioned methods of synchronization into two constituent parts: centralized and decentralized systems. We are going to briefly explain how each system gets the name it has, and how it is fundamentally different from its counterpart.

Let us discuss the decentralized system first and get it out of the way as our work hardly involves its working principle. The first method discussed in the previous section i.e. manually synchronizing the clocks, is the decentralized system of clock. Although the method involves a reference clock which we can also allude to as master clock, the process of manual synchronization gives rise to plenty of room for error. As the method requires active participation of an individual to monitor and synchronize the slave clocks with the referential one, man-made errors automatically comes into play. And also, as none of the clocks are identical to each other, time discrepancy between the clocks eventually takes place. There is no way of keeping track of all the inconsistency except for going from clock to clock and observing them, which requires time and energy as already stated, and is a very hectic and tedious process.

The second and third methods of synchronization detailed in the last section, gives us the centralized clock system. What makes it very different from its fellow system is the lack of human involvement in terms of synchronization. Wired system has all the slave clocks connected with the master through wires and the purpose of connection in wireless system is served through usage of wireless network. As previously discussed, wireless system is much superior in comparison to the wired one from many facets. Our work here exploits the benefits of wireless network and it will be seen further on in this thesis how we have utilized the advantages of wireless network to achieve the proposed solution.

1.4: Motivation

Motivation for this thesis work has come from the issue of desynchronized clocks and problems it poses in our daily lives as we have already discussed in the previous sections. We have tried to come up with our own answer to the problem which borrows a lot from available technologies, then merging it with our own ideas, and finally coming up with a concoction as a useful if not the ultimate solution. We hope that our proposal through this work will avail the institutions whose works are highly dependent on the accuracy of time keeping. With these in our mind, we present to you our work.

1.5: Overview of the Thesis

In this chapter, we have presented the reason for tackling the issue at hand and the purpose we are trying to serve. In Chapter 2, we go into details of concepts of wired and wireless systems

that we have put in a nutshell in this chapter. In Chapter 3, we discuss the network features and communication protocols utilized in building our system. In Chapter 4, we give an account, within the capacity of this thesis, of the hardware used. In Chapter 5, we describe the software and the coding involved. In Chapter 6, we build our proposed model, state the budget and synthesis of the hardware. In Chapter 7, we give the result we have achieved and an analysis of our system. In Chapter 8, we end our work with the challenges, both resolved and unresolved, possible solution for the unresolved challenges and further work that can be done in this topic.

Chapter 2

Comparison between Wired and Wireless System

2.1: Wired System

Our first attempt towards the solution of de-synchronization problem was via the usage of wired system. In our wired system we had one master clock, which served as the reference and several other slave clocks connected to the master by external wires. Each of the clock had an internal microcontroller. The microcontroller within the master clock generated pulse with a given frequency which the microcontroller then employed to produce the required time. This time was then distributed to the designated slave clocks through the connecting wires. If any alteration to the time were required, we would make the adjustment in the master clock which would then be updated in the slave clocks.

The slave clocks, on the other hand, kept itself attuned with the master clock by constantly adjusting itself with the transmitted time signal. The microcontrollers within each slave clock, receives the time from the master clock, updates itself with the collected time data, and finally displays the time on the LED display.

There are some challenges we faced in employing this system. Firstly, the microcontroller of the master clock couldn't communicate properly with more than a specified number of slave clocks and greater the distance was between the two types of clocks, the greater was the signal attenuation. We overcame this drawback by using a module called RS485 which allowed us to incorporate greater number (upto a maximum of 255) of slave clocks and negate the signal attenuation [1]. This module also gave us the freedom of increasing the distance between the clocks by more than 1 km [2]. The second and the biggest hurdle in using this system is its expense, cumbersomeness and requirement of frequent maintenance. These problems don't have any direct solution. Due to the severity of these problems, we had to apply the wireless system.

2.2: Wireless System

The problems with wired system has motivated us to look for alternative solutions. The only way of avoiding the hassle that came with wired system was to go for a wireless one. Implementation of wireless solution is very easy considering how widespread the use of wireless routers is nowadays. So, our final solution to the problem comprises of using the internet to collect the time. We will concisely describe our procedure here.

The atomic clock at Royal Museums Greenwich is one of the most precise time keeping devices [3]. The time recorded by this clock is updated on the NTP servers around the world. In our case, we are using UNIX time from the NTP servers. This UNIX time is fetched into our "smart-watch" through a process known as UDP protocol. When our system uses the UDP

protocol, a string of numbers is fetched to our clock from the designated NTP server which the number of seconds that has passed from zeroth hour of 1st January 1970 till now [4]. The microprocessor within the clock then converts this string into hours, minutes and seconds. This time is then shown on the LED display of the clock. That is how our system remains synchronized and updated. We will go into the details of the procedures in a separate chapter.

2.3: Superiority of the Wireless System and IoT

Wireless system is far more superior to its wired equivalent. One advantage of the wireless system is that it is a much more convenient and much easier to handle. We don't have to go through the hassle of using external wires and power supplies, separate modules and the issue of signal attenuation. Wireless system is much more compact and requires fewer components. It is also very much financially viable. The next and biggest advantage of the system is that we get our time directly from the most accurate source of timekeeping. Therefore, there is hardly any inconsistency in the time fetched to us.

This way of using internet to maintain our everyday appliances and devices has given rise to a very new field of informatics called internet of things. Our project is actually based of the core ideas of IoT(internet of things). Hopefully this will help future generations of students and researchers alike in building systems based on IoT.

Chapter 3

Network and Communication Protocol

3.1: NTP and UNIX Time

NTP stands as the short form of Network Time Protocol. This protocol is utilized to adjust the computers clocks to some particular time reference. This protocol was developed by Dr. David L. Mills of University of Delaware [5]. NTP has several useful features, some of which I have tried to present below as succinctly as possible.

Features of NTP

- NTP works by referring the time from a *reference clock* which by default shows the *true time.* All the clocks that are using NTP get synchronized towards the *true time*. In most cases the *reference time* is UTC time.
- NTP is a virtually flaw proof system. When a clock uses NTP, the server chooses the best time possible from several available options. The in-built algorithm in the protocol allows it to combine multiple different time sources to keep the error at a minimum.
- NTP is an extremely precise protocol. Its resolution can be as accurate as 1 nanosecond which is many times more accurate than popular protocol such as **rdate**.
- In case of network failure, NTP can do measurements of past time to find out the current time and associated error.
- The protocol keeps track of accuracy of the time.

NTP can be put into effect through UNIX operating system. It is favored over many other protocols because it boasts a whopping 175,000 hosts running NTP concurrently over the internet, according to a survey.

Now we come to our discussion about UNIX time. UNIX time, which is also known as POSIX time/UNIX Epoch time, is a mode for finding a particular instance of time, defined by numbers of seconds that have passed since 00:00:00 UTC(Coordinated Universal Time), Thursday, 1 January 1970, less the number of *leap seconds* that have occurred since then[6]. For our system to work, UNIX time is fetched to it, which is 32-bits of string, from the appointed NTP server, using the UDP protocol which is described below.

3.2: UDP Protocol

UDP, which stands of User Datagram Protocol, is a substitute correspondence protocol to TCP (Transmission Control Protocol) [7]. It is used to initiate links between applications on the internet which thus have low-latency and higher loss tolerance. The protocol sends short packets called datagrams but generally, it is unreliable and connectionless [8].

UDP has its share of advantages and disadvantages. We will discuss the advantages first. It has two features that IP layer doesn't have. It gives *port numbers* which differentiates between the user requests. It also provides a checksum which assures that data wasn't lost during transmission. On the other hand, due the simplicity of its transmission model, it doesn't use

handshaking dialogs, leading to lack of reliability, disorganization and compromise of data integrity. It doesn't process data at the network interface level.

For our project we have used the protocol to fetch the UNIX time from the NTP server. It has done the job quite satisfactorily in spite of the protocol's lacking.

3.3: WiFi Technology

In our present time, wifi is one of the most ubiquitous form of technology. This wireless networking technology uses radio waves to provide wireless high speed internet and network connections. The influence of this technology is so great in our lives that it is becoming one of the needs of human civilization at the very basic level. A huge portion of data and information that we come across each day is either transmitted to us or received by us using wifi. Lack or loss of wifi connection in workplaces can cause severe disorientation, loss of productivity, and can even lead to complete shutdown. This trademarked phrase means IEEE 802.11x [9].

The network that uses wifi doesn't have any physical wired connection. Instead, wifi uses electromagnetic waves of radio frequency in order to communicate between the transmitter and the receiver. The frequency usually lies within the range of 2.4 to 5.8 gigahertz[10]. When a current of radio frequency is provided to the antenna of the wifi transmitter, signals in the form of radio waves is produced which then travels through the medium of interest (in this case air) and is received by another wifi receiver. Wifi is broadcasted through an access point of the broadcasting device, such as a router. Devices of interest then connect to the network with their built-in wireless network adapter.

We have used this very handy technology in connecting our clock wirelessly with the internet, which in turn has made our system truly wireless. We integrated an ESP8266 module, a microchip with full TCP/IP stack and microcontroller capability, which connected the clock to the clock NTP servers. Then, by the User Datagram Protocol fetched the UNIX time to our clock microcontroller. After that the microcontroller calculated the time from the 32-bits string received and displayed it on the panel. We will go into the details of each hardware used in the next chapter.

Chapter 4

Hardware Description

In this chapter, we try to give an account of major electrical modules and equipment used in the project. We attempt to make our description as concise as possible and at the same time give an effort to not overlook any important feature of the component.

4.1: ESP8266

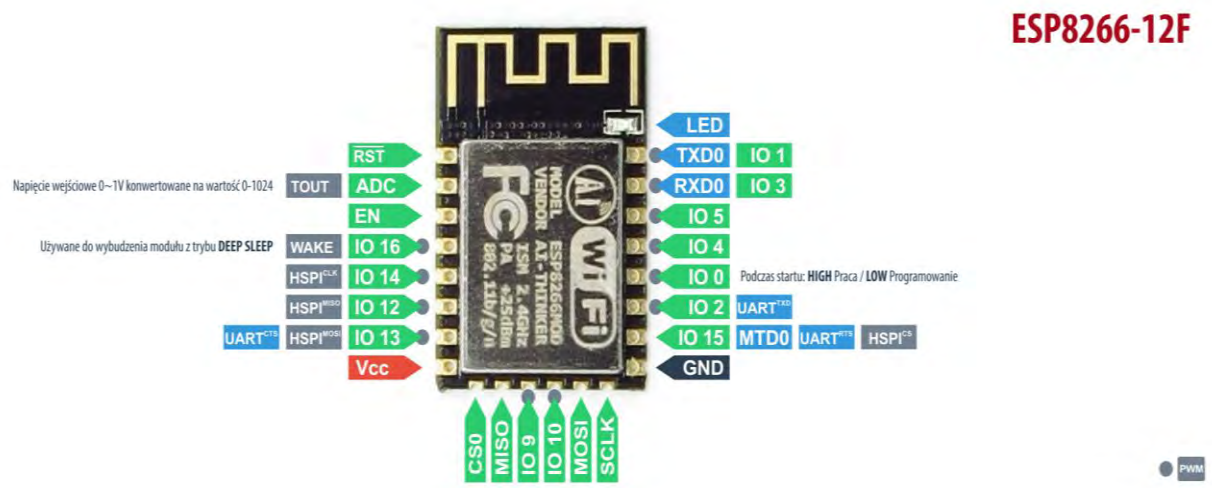


Figure 1:ESP8266 [21]

ESP8266 is a wifi module that incorporates both TCP/IP stack and microcontroller facilities in a single chip [11]. It is used to give any microcontroller access to the wifi network. The module can host applications and deposit all wifi networking procedure from other application processor. It has a built-in AT command set firmware, giving us the flexibility of setting it up directly with an Arduino microcontroller for its use with wifi network. The processing power and storage

capacity of ESP8266 module is quite high. It can be connected with sensors and other application specific devices using its general purpose input/output pin requiring minimal upfront development. It also has a very small runtime delay during loading. The components within the module is very densely packed, reducing the area and minimizing the need for external circuitry. The operating temperature range of the module extends from -40 degrees celsius to +125 degrees Celsius [12]. This feature along with its compact design makes the module very reliable, robust and durable in industrial working condition. It has automatic power saving delivery for voice over internet protocol and can simultaneously support bluetooth in its interface. Its self-calibrated RF circuitry allows it to work under various conditions, requiring no external RF component. It has come to become one of the integral components for IoT based projects and devices. It also boasts a very small power consumption making it ideal for low power system integration. Additionally, the three modes of operation called active mode, sleep mode and deep sleep mode, make it very suitable for battery powered designs. It also has a maximum clock speed of 160 kHz. This microcontroller is manufactured by a Shanghai-based Chinese company, named **Espressif Systems**. The processor follows the design of L106 32-bit RISC microprocessor core which is based on the TensilicaXtensa Diamond Standard 106 Micro running at 80 MHz. The different memory allocation size of this chip is as follows:

- 32 KiB instruction RAM
- 32 KiB instruction cache RAM
- 80 KiB user data RAM
- 16 KiB ETS system data RAM

We have used this amazing piece of equipment in our IoT based clock's microcontroller to keep it in-sync with the wifi and thus the NTP servers.

4.2:RTC

Figure 2:RTC [22]

Although the Arduino microcontroller we are using within our clock has built-in timekeeper called **millis()** and separate timers integrated into the chip that keeps track of longer period of time, we still need an RTC [13]. In this section, we discuss what a RTC is and why it is required for our given purpose.

Real Time Clock, or RTC for short, is an integrated circuit which is very similar to a watch running on battery. It uses DS1307 integrated circuit that is cheap to make, easy to solder and can run for a very long time on a small coin cell. ICs within an RTC maintain time using a crystal oscillator and do not rely on hardware clocks. Typical frequency of the oscillator is 32.768 kHz which is the same as that for quartz clocks and watches [14]. The ICs are extremely precise, most having an error count less than 5 parts per million. RTC also has a small memory which stores system description and setup values, including current time values stored by the real-time clock. In addition to keeping track of time, RTC ICs makes sure that all processes

occurring within the system are well synchronized with each other. Lithium coin cells are used in most RTCs. The cells can last three to five years. Once the operational battery dies, error message is shown which prompts the user to replace it.

Our reason for using RTC has to do with backing up our IoT based clock in case of power failure. Once there is power failure, clock in the Arduino microcontroller in set to zero. The clock within the microcontroller is reset every time it loses power. Therefore, we have to rely on RTC for consistent timekeeping as it doesn't reset during power failure or when the Arduino microcontroller is reprogrammed.

4.3: 7-Segment LED Display

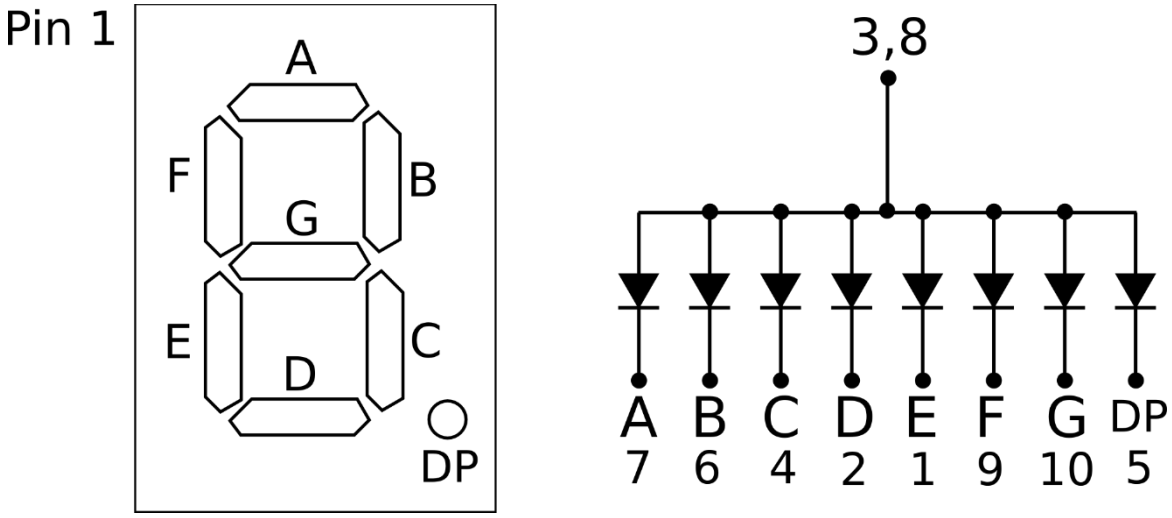


Figure 3:7-Segment [23]

We have used a 7-Segment LED display as the output of our IoT based Synchronized clock. We dedicate this section as an explanation for the mechanism by which the display works and how it serves its given purpose in this project.

We start by giving an account of what LED is and how it works. LED, short for light emitting diode, is a solid state device. Just like a regular diode, it has a pn-junction[15]. When an external voltage supply is applied as a forward bias to a LED, current flows in the junction, which causes the holes and electrons in the junction region to combine through a process called EHR (electron-hole recombination). The EHR process releases photons as energy; a pair of electron-hole recombine to produce a single photon. This emission of photons from electron hole pair recombination is called electroluminescence. Light produced by LED ranges from visible region, such as blue to red to orange, to invisible region (electromagnetic wave that cannot be detected by naked human eye), such as infrared. The spectral wavelength of the emitted light is decided by the type of impurity mixture used on the material during the manufacturing process of the LED. LEDs are much more efficient, cheaper and smaller than traditional filament light bulbs. They also have much higher longevity and variation of colors compared to conventional lamps. They are very easy to produce and integrate with electronic circuits, a trait that makes them ideal for their usage in 7-Segment display.

7-segment display, or SSD for short, is an electronic display device for displaying decimal numbers. They are widely used in digital clocks, calculators, different types of electronic meters, timers to show numerical data. As the name suggests, SSD consists of seven segments of LED, arranged in a rectangular fashion. The reason each of the seven LED is called a segment is because when the LEDs from different segments light up, their combination can form arabic numerals (0 to 9) to be shown. The display can also form hexadecimal numbers (0 to F) if required. Each of the LEDs has one pin coming out of them from the plastic package. These pins are designated from a tog to denote their positions on the display. The other pins are connected

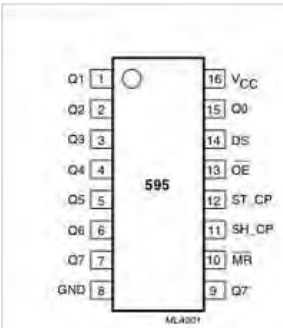
together to form a common pin. When we want to display a particular decimal or hexadecimal number on the display, we forward bias the appropriate pins. In this way, some of the segments will light up and others will stay dark, giving us the exact pattern for the number we are trying to show. The common pin is usually used to differentiate between two types of displays: Common Cathode (CC) and Common Anode (CA). CC display has all the cathodes of LEDs connected together and CA display has all the anodes connected together.

The LEDs in the SSD are driven by forward biased current. Intensity of light produced by each individual LED is dependent upon this current and their relationship is approximately linear. So, we must control the amount of current flowing through the LED to protect it from being damaged due to current overdrive. We used external resistors to stop excess current from flowing through the LEDs. The forward bias voltage to correctly illuminate an LED varies from 2 to 3.6 volts, depending on its color. The value of the voltage source should be greater than the required bias voltage of the LED. To prevent surplus current from flowing through, we add resistor in series with the LED. In our project, we have used SSD with blue LED segments. Each segment requires roughly 20mA of driving current to light up correctly. With 15V voltage source, we had to use resistors within the range of 200 to 220Ω to correctly illuminate the segments.

If we had connected the SSD in a straightforward fashion, we would need at least 44 digital output pins from the microcontroller. But, our particular microcontroller has only 9 pins for I/O. To save such huge amount of I/O pin requirement, we introduced both shift registers and multiplexing method. In this way we need only three digital pins of the microcontroller and two shift registers.

In the multiplexing method the typical 20mA current is not sufficient enough to fully glow the SSD LEDs due to a reduced duty cycle output. The average power requirement is a lot less than the usual because of this multiplexing method. According to the manufacturer's specification, with a duty cycle of 10%, we can apply a maximum current of 100 mA. With some calculations, we set the current to 80 mA limit with a 33 ohm current limiting resistor.

4.4: Shift Register



PINS 1-7, 15	Q0 " Q7	Output Pins
PIN 8	GND	Ground, Vss
PIN 9	Q7"	Serial Out
PIN 10	MR	Master Reclear, active low
PIN 11	SH_CP	Shift register clock pin
PIN 12	ST_CP	Storage register clock pin (latch pin)
PIN 13	OE	Output enable, active low
PIN 14	DS	Serial data input
PIN 16	Vcc	Positive supply voltage

Figure 4: Shift Register [24]

Our project makes use of shift register for data processing. Shift register is a group of flip-flop circuit cascaded in a particular fashion. The output of each flip-flop is connected to the 'data' input of the next flip-flop in the chain. Therefore, at the transition of the clock input, the data stored in the 'bit array' shifts by one position. Here, the data present at the input 'shifts in' and the last bit in the array 'shifts out' [16]. Shift registers, depending on data entry and exit, are categorized in four distinct subdivisions:

1. Serial Input Serial Output (SISO)
2. Serial Input Parallel Output (SIPO)
3. Parallel input Serial Output (PISO)
4. Parallel Input Parallel Output (PIPO)

We have used 74HC595 shift register in our clock which is a Serial-in-Parallel-out shift register. We have connected the DS pin of the IC to the I/O pin 2, the ST-CP to the I/O pin 5 and the SH-CP to the I/O pin 6 of the microcontroller. The ST-CP and SH-CP of all the shift registers are shorted together and the Q7' pin of the first IC is connected with the DS pin of the next IC. After making all physical connections, we send a string of integer numbers from the uC to the shift registers which in turn set some of the specific output pins of the shift registers high and the rest to a low value.

4.5: Power Section

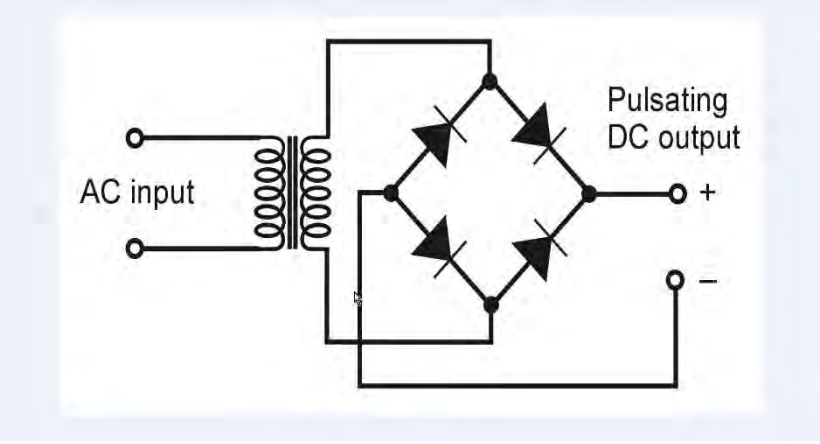


Figure 5: Full Wave bridge rectifier [25]

Our IoT based digital clock works with dc power supply. As the mains power supply is AC in nature, we had to first use a step-down transformer and then a full wave bridge rectifier. The

transformer converts 220V to 19V AC which then gets converted to DC through the rectifier. The clock also has a backup battery connected to it. In case of a power failure, the relay within the power section would switch from the mains supply to the backup battery. When the power comes back again, relay switches to mains supply and the backup battery starts charging.

Chapter 5
Software and Programming

5.1: IDE and Drivers

In this chapter, we discuss the software related materials used in our system. We have used the C programming language to program the microcontroller and Arduino IDE, which is a free standard license software for general usage, to write and burn our program into the IC. However, we could not directly use the software to burn the program into the microcontroller. In order to make it compatible with our IC, we needed to incorporate specific board and its corresponding libraries into Arduino IDE. We selected wemos d1 and it is available libraries for the job. For the installation process, we chose the board from the Board Manager of the Arduino IDE. We downloaded and installed it from there, which was a time consuming procedure.

The drivers for the particular IC used in the microcontroller module board are not available in the Operating System most of the time. In such case, we also need to update the driver for CH340 IC and only then it is possible for the OS to recognize the particular board we are using. For installations and updates, we needed a cable consisting of both mini-usb and regular usb facilities to establish a communication channel between computer and the microcontroller. However, after plugging in the usb cable, we must select the particular PORT we are using for communication and the specific baud rate for serial communication.

5.2: Programming Libraries

To make all the different features of the microcontroller available and for all the other peripheral devices to use many different sophisticated techniques, we had to incorporate several different libraries into our programming environment. To use the wifi technology available into our system, we used ESP8266WiFi.h library via which we could transmit and receive data over the wifi network from our microcontroller.

To make various time acquisition facilities available from system clock and other diverse means, we also integrated a distinct library which is TimeLib.h library. Using this particular library we programmed the dedicated Real Time Clock module which was capable of giving us a backup in case of wifi unavailability. The library is available at github.
(https://github.com/PaulStoffregen/Time)

To enable the ESP8266 chip with UDP protocol for time fetching from various different NTP servers, we used the WiFiUdp.h library. This library eased our programming difficulties for continuous time fetching from different time servers over the internet, while maintaining all the rules and regulations for UDP.

The microcontroller use I2C protocol to communicate with the Real Time Clock module. Although this particular communication protocol is very efficient, it is a rigorously comprehensive and painstakingly cumbersome procedure. However, by using the Wire.h library, we have significantly reduced the complicacy. Therefore, the programming code for I2C communications became more organized and focused on the main program-flow.

As we have used the Real Time Clock module for a long duration time keeping in case of power and/or wifi failure, we also included aRTClib.h library the memory of the RTC module can be edited and reprogrammed easily. This particular library allows the microcontroller to continuously save the time fetched from the NTP servers into the RTC memory which is then updated in the RTC every second with great precisions. This library is also available at github and free to anybody for general use (https://github.com/Makuna/Rtc).

5.3: Programming Code

Initially, we set up all the communication and wire protocols in Serial, I2C, and setup the wifi for the microcontroller. Firstly, we initialized the serial communication with the baud rate of 115200. For this we used in the programming command given below

Serial.begin(115200);

We used multiple shift registers to send the desired output to the LED display. Microcontroller D2, D5 and D6 digital output pins has been used to serve for the data, latch and clock pins of the registers. We used the following commands for this purpose:

pinMode(latchPin, OUTPUT);

pinMode(dataPin, OUTPUT);

pinMode(clockPin, OUTPUT);

The module gets connected to a particular wifi network when provided with the SSID and its passwords,. We used the following command in the coding,

```
WiFi.begin("SSID","password");
while ((!(WiFi.status() == WL_CONNECTED))){
delay(300);
attemptCnt++;
if(attemptCnt>=35){
attemptCnt =0;
break;
    }   }
```

At this point, if the wifi network is available, the microcontroller sets up a UDP protocol and attempts to fetch time from the NTP servers. If the time servers are accessible, the microcontroller fetches a string of number which consists of the accumulated seconds from 1970 to that particular moment in time, and process this number into year, month, day, hour, minute and second for the conventional time representation. The following commands are used to accomplish this particular task.

```
Udp.begin(localPort);
setSyncProvider(getNtpTime);
setSyncInterval(300);
```

After getting the NTP time, the microcontroller initiates the I2C communication and Real Time Clock module by saving the NTP time into the RTC memory. This in turn enables the microcontroller to show the time on LED display consistently. The following command has been used into the code to perform this particular task.

```
Wire.begin();
rtc.begin();
```

When the RTC is fully set up, the microcontroller is ready to run the clock display. To run the 6 seven segments digits and two portions for AM and PM display, the microcontroller sends a calculated value to the shift registers that makes some particular output pins of the shift registers high to generate some specific digits. The following command in the code has been used to implement the mentioned actions.

```
intdigitCode [] = {63, 6, 91, 79, 102, 109, 125, 7, 127, 111};
#define del 2
voiddo_peekAboo(void){
Serial.println(" ");
Serial.println("****OPERATION TO FOCUS ON***");
Serial.println("****************************");

 //fnHour
if(fnHour==1){
```

```
Serial.println(fnHour);
digitalWrite(latchPin, LOW);
shiftOut(dataPin, clockPin, MSBFIRST, 128 + digitCode[fnHour] >>8 );
shiftOut(dataPin, clockPin, MSBFIRST, 128 + digitCode[fnHour]);
digitalWrite(latchPin, HIGH);
delay(del);
 }

 //snHour
Serial.println(snHour);
digitalWrite(latchPin, LOW);
shiftOut(dataPin, clockPin, MSBFIRST, 256 + digitCode[snHour] >>8 );
shiftOut(dataPin, clockPin, MSBFIRST, 256 + digitCode[snHour]);
digitalWrite(latchPin, HIGH);
delay(del);

 //fnMinute
Serial.println(fnMinute);
digitalWrite(latchPin, LOW);
shiftOut(dataPin, clockPin, MSBFIRST, 512 + digitCode[fnMinute] >>8 );
shiftOut(dataPin, clockPin, MSBFIRST, 512 + digitCode[fnMinute]);
digitalWrite(latchPin, HIGH);
delay(del);
```

```
  //snMinute
Serial.println(snMinute);
digitalWrite(latchPin, LOW);
shiftOut(dataPin, clockPin, MSBFIRST, 1024 + digitCode[snMinute] >>8 );
shiftOut(dataPin, clockPin, MSBFIRST, 1024 + digitCode[snMinute]);
digitalWrite(latchPin, HIGH);
delay(del);

  //fnSecond
Serial.println(fnSecond);
digitalWrite(latchPin, LOW);
shiftOut(dataPin, clockPin, MSBFIRST, 2048 + digitCode[fnSecond] >>8 );
shiftOut(dataPin, clockPin, MSBFIRST, 2048 + digitCode[fnSecond]);
digitalWrite(latchPin, HIGH);
delay(del);

  //snSecond
Serial.println(snSecond);
digitalWrite(latchPin, LOW);
shiftOut(dataPin, clockPin, MSBFIRST, 4096 + digitCode[snSecond] >>8 );
shiftOut(dataPin, clockPin, MSBFIRST, 4096 + digitCode[snSecond]);
digitalWrite(latchPin, HIGH);
```

```
delay(del);
  //AM/PM
Serial.println(aMpM);
digitalWrite(latchPin, LOW);
shiftOut(dataPin, clockPin, MSBFIRST, aMpM + 0 >>8 );
shiftOut(dataPin, clockPin, MSBFIRST, aMpM + 0 );
digitalWrite(latchPin, HIGH);
delay(del);
}
```

To update the Real Time Clock memory every certain period of time, we devised an algorithm which continuously checks whether it is the time to update the RTC memory and when the conditions match, the microcontroller fetches the time from NTP servers and loads the new time into the RTC. The following command has been used to execute the mentioned tasks,

```
intupdateTimeAt = 30;
if ( access_key == true &&digitalClockDisplay('m') == updateTimeAt&&fkey == false ) {
Serial.println("NOW AT THE STATE TO UPDATE loadRTC_time");
loadRTC_time = digitalClockDisplay('m');
Serial.print("loadRTC_time = ");
Serial.println(loadRTC_time);
fkey = true;
access_key = false;
   }
```

Chapter 6

Proposed Model

6.1: Budget and Cost analysis

The budget of the total system can be segregated in various parts. Here in this project we can distinguish cost in to two major areas, Cost of the system and Peripheral cost. The cost of the system includes the different modules, display devices and other circuitries whilst the decoration and other costs falls under peripheral costs. The budget for Research and Development and the budget for Per Unit will both be provided here

ITEM	QUANTITY	PER UNIT COST	TOTAL
Digital Clock Display	1	2000	2000
Clock Casing	1	2000	2000
ESP8266 Wifi Module	1	680	680
Battery Backup Circuit	1	1500	1500
RTC Module	1	229	229
74HC595 Counter Shift Registers	6	25	150
7815 Voltage	3	35	105

Regulator			
7805 Voltage Regulator	1	35	35
7809 Voltage Regulator	2	35	70
2N2222	20	3	60
2N2907	25	3	75
PVC SHEET	1	200	200
Male-Male Jumper Wires	1	55	55
PCB [6”x3”]	1	440	440
Resistor	30	2	60
Capacitor	4	10	40
DC Power Jack (Male)	2	45	90
DC Power Jack (Male)	2	15	30
		TOTAL	7784

Table: Per Unit Cost

ITEM	QUANTITY	PER UNIT COST	TOTAL
Digital Clock Display	1	2000	2000

ESP8266 Wifi Module	2	680	1360
Digital Clock Casing	1	2000	2000
Battery Backup Circuit	1	1500	1500
RTC Module	2	229	458
Wifi Router LB-Link	1	1150	1150
74LS595 Counter Shift Registers	12	25	300
2N2222	20	3	60
PVC SHEET	2	200	400
3mm LED	50	1	50
Male-Male Jumper Wires	1	55	55
7805 Voltage Regulator	2	35	70
7809 Voltage Regulator	4	35	140
2N2222	30	3	90
2N2907	40	3	120
PCB [6”x3”]	2	440	880
Resistor	50	2	100
Capacitor	8	10	80

DC Power Jack (Male)	2	45	90
DC Power Jack (Female)	3	15	45
Vero Board	1	25	25
16 Base Pin	8	10	80
		TOTAL	10993

Table 2: R&D Cost

Cost Analysis:

Our system costs around 6000 Bdt without any decorative addons, whereas clocks of same size in market, is generally available from 2000-3000. However it must be considered that most of the clock available in the market doesn't have a power backup. In addition our IoT based synchronization ensures the accuracy of maximum 0.071 seconds delay/day comparing to 1.5secs delay/day found usually given the usual conditions remain same. So although this system is expensive compared to its existing competitors, it's added features, along with its accuracy gives it the edge over the rest. Moreover if mass produced in production line, the net cost of the total product will be further reduced.

Figure 6: IoT Based Real Time Synchronized Clock

6.2: Hardware synthesis

The function of the system can be devised in three distinct parts, the power unit the processing unit and the display unit. The power unit consist a backup power circuit which acts as a failsafe for power failure. The processor unit is the most integral part of the system consisting of the ESP8266 and RTC modules. This unit receives the data sent over Wifi , processes it and passes the processed data through a multiplexer circuit to the display

6.2.1: Algorithm

The ESP8266 module fetches the data from the NTP server, the data is then processed and sent to RTC which synchronizes and updates it's time then from RTC it's again sent to ESP8266 which then passes the data through a multiplexing circuit and sent to the display. Here in this system the loop is implemented to prevent the system from failing due Wi-Fi failure as the RTC will serve as a backup. The system is designed in such a way that the Wi-Fi module will search and synchronize every 45 minutes to update, however this time can be varied upon users preference.

1. ESP8266 searches for the NTP through Wi-Fi

2. Fetches and Processes Data from NTP

3. Data is sent to RTC

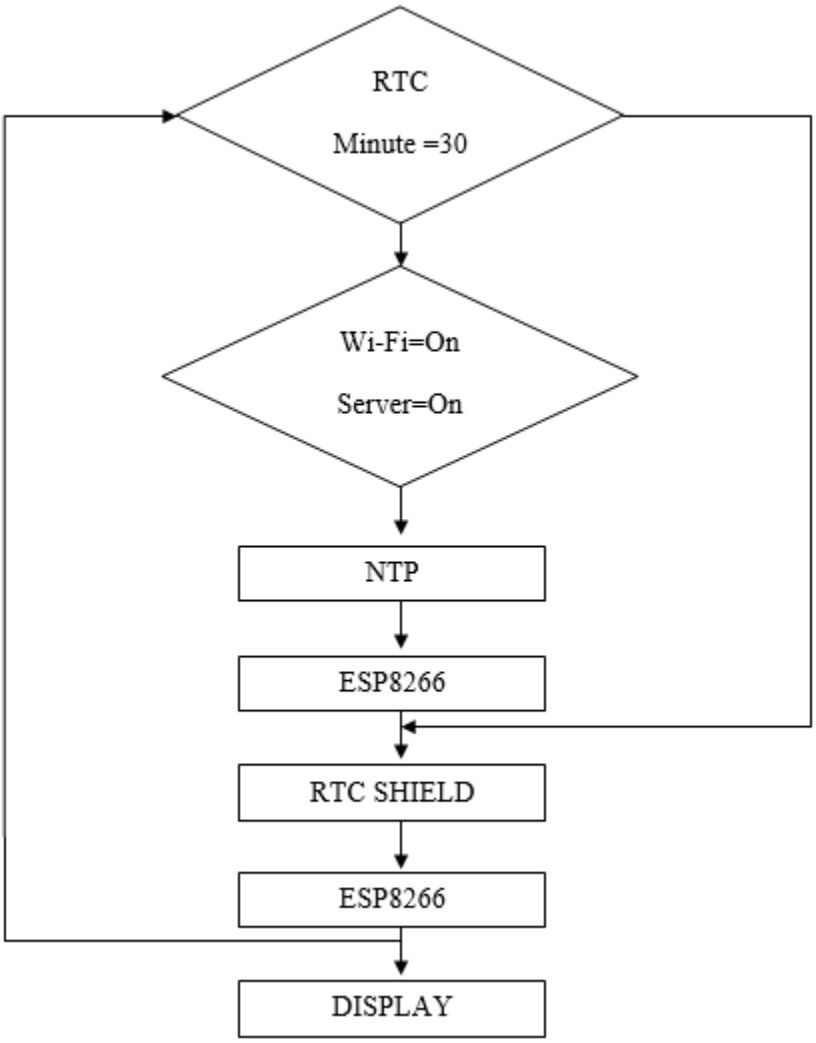
4. RTC updates and sends the data back to ESP8266

5. ESP8266 passes the data through a multiplexer circuit to the display

6. Process 4 and 5 repeats for 1 hour

7. Whole processes is repeated every 1 hour

6.2.2: Flow Chart



6.2.3: Circuit Diagram

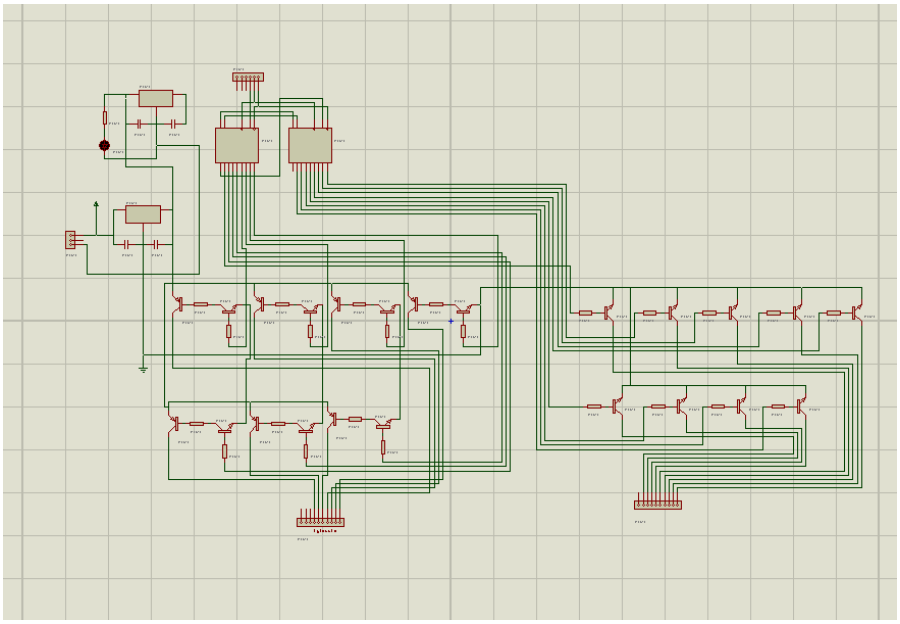


Figure 7: Circuit Diagram

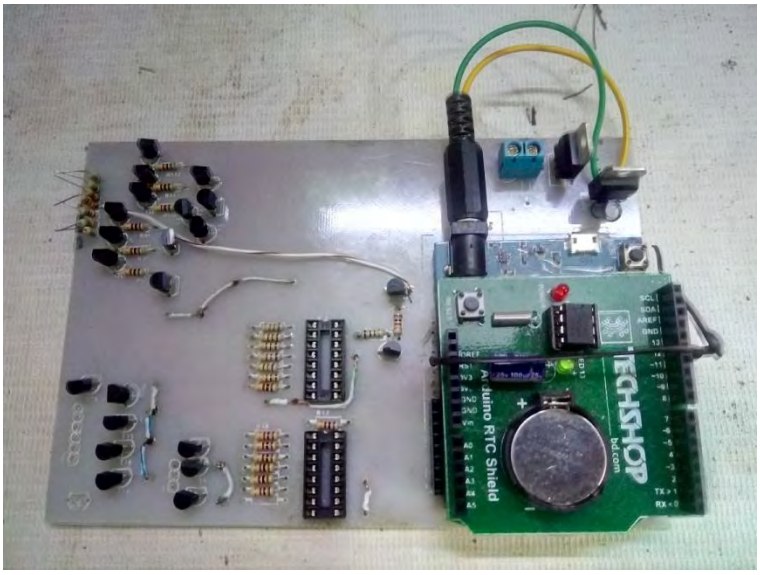


Figure 8: Implemented Circuit

Chapter 7

Results and Analysis

7.1: Result Analysis

The system was tested multiple times with different synchronization cycles. Variation of frequency of synchronization cycle resulted in a change of accuracy as well. The accuracy of the system is mainly dependent on the real-time clock (RTC) IC. This, itself, is dependent on the accuracy of the crystal used and its temperature characteristics. A significant change of temperature results in loss of accuracy. The IC used in this case, DS1307 has an accuracy of ±20 ppm at room temperature, or ±2 seconds per day. The same crystal slows down by an addition of 14 to 16 seconds per day at the temperature extremes of -40°C or +85°C.

The RTC in question uses a 32.768Khz frequency crystal. Although, it is fairly accurate at standard temperatures, delay being ±1.7 sec/day, due to its parabolic nature the accuracy falls significantly at temperature extremes which is approximately ±13 sec/day.[16]

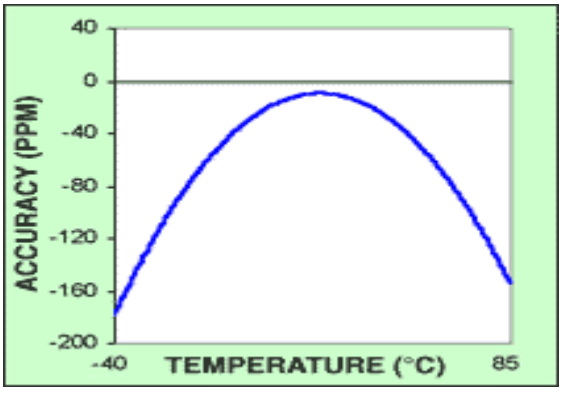


Figure 9: Accuracy vs Temparture [17]

The frequency deviation (Δf) of a typical crystal at a specific frequency (f) and temperature (T) is:

$\Delta f/f = k(T - T_o)2 + f_o$

Where,

f = Nominal crystal frequency

k = Curvature constant

T = Temperature

T_o = turnover temperature

f_o = frequency deviation at room temperature.

From this equation it can be easily observed that each crystal's frequency response over temperature is controlled by three variables, those being curvature constant, turnover temperature, and room-temperature frequency deviation. Although the curvature constant has the most significant effect on the parabolic nature of the frequency deviation over temperature, this constant has a very small deviation. Different turnover temperatures shift the deviation curve left or right, and different frequency deviations at room temperature shift the curve up or down.

In this project we have minimized the time deviation to an extent. As the RTC is basically being refreshed every 1 hour, the delay accumulated is also being refreshed. Therefore, by calculation if the delay is ±1.7 seconds/day under normal conditions, the accumulated delay will be around ±0.071 seconds every 1 hour which is consistent with the practical results. If the synchronization

time is decreased the accuracy increases, but as the during update cycle the data isn't provided to the display for around 0.5 seconds, so it might not be convenient.

As the algorithm used is similar to that of Cristian's [19], it is possible to measure the delay caused during synchronization cycle and accuracy. If T_{min} is the minimum time to transmit a message one-way, the earliest point at which NTP Server, S could have placed the time T, was T_{min} after Processor Module P sent its request. Therefore, the time at S, when the message by P is received, is in the range ($T + T_{min}$) to ($T + RTT - T_{min}$). The width of this range is ($RTT - 2* T_{min}$). This gives an accuracy of ($RTT/2 - T_{min}$).Here RTT is **round-trip time[20],** This is the length of time it takes for a signal to be sent plus the length of time it takes for an acknowledgement of that signal to be received.

7.2: Probable solutions

Although there have been limited options available to improve upon the crystal inaccuracies but certain applications can improve timekeeping accuracy through crystal screening, integrated crystals, calibration registers, or temperature-compensated crystal oscillators.

Crystal Screening

By using a screening process, a crystal manufacturer could analyze each crystals frequency deviation at room temperature and provide a subset of crystals that improve the room temperature accuracy from ±20 ppm to ±10 ppm or ±5 ppm. These "improved" crystals would still suffer from large inaccuracies at high and low temperatures. This method has no effect on the parabolic nature of the crystal's accuracy curve and comes at an added cost[17].

Integrated Crystals

Taking the crystal-screening a step farther and providing an integrated crystal reduces the designer's workload by eliminating crystal procurement issues. This alleviates concerns about the crystal parameters matching up with the timekeeping device requirements, and reduces printed circuit board (PCB) layout issues[17].

Temperature Compensation

Temperature compensation requires an extensive development and calibration investment. In order to develop a temperature compensation algorithm a temperature sensor with a timekeeping device that provides some form of analog or digital clock calibration is utilized. This includes the periodic measurement of temperature, and subsequent adjustment of either the crystal loading or the clock source according to the measured temperature[17].

Calibration Register

Some RTCs, like the DS1340, provide a digital calibration register that can be used to periodically adjust the time of day in discrete amounts. This method does not attempt to alter the crystal behavior at all, but instead, periodically adjust time according to the expected frequency deviation at a specified temperature. An RTC with a calibration register can be combined with a temperature sensor to achieve accuracy levels from -2.034ppm to +4.068ppm at one specific temperature. The total adjustment range is from -126ppm to +63 ppm so, at extreme high and low temperatures[17].

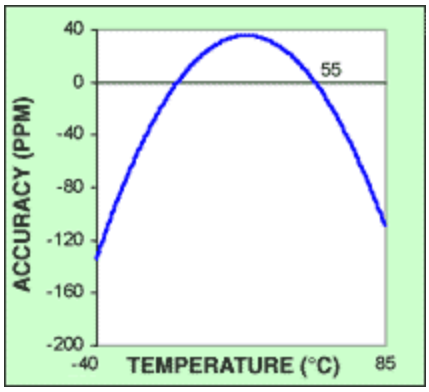


Figure 10: Typical crystal curve is shifted upward until the accuracy approaches 0.0ppm. [17]

Temperature-Compensated Crystal Oscillator

A TCXO includes an integrated sensor to periodically measure device temperature. The measurement is used to access a lookup table, whose output is used to calculate and apply a load-capacitance value for the integrated 32.768kHz crystal to achieve 0.0ppm accuracy. The lookup table exists on the device and requires no external inputs

These devices are factory calibrated, and can provide accuracy as good as ±7.5ppm over the industrial temperature range (-40°C to +85°C). The effect of a TCXO is to flatten the parabolic nature of the crystal curve over temperature (**Figure 3**).

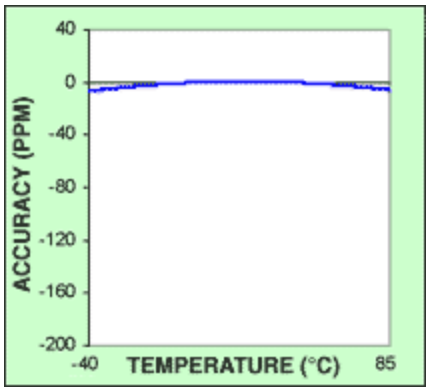


Figure 11: Crystal curve flattened by a TCXO [17].

RTC/TCXO/Crystal Integration

The ideal accurate timekeeping device would be one that integrates an RTC, a TCXO, and a quartz crystal into a single package thus providing unparalleled accuracy of ±2.0ppm from 0°C to +40°C, which is the equivalent of just over ±1.0 minute each year. Accuracy from -40°C to 0°C and +40°C to +85°C is ±3.5ppm, which is the equivalent of ±1.8 minute each year. The worst-case accuracy of this device is displayed in **Figure 4**. An on-chip aging register provides an adjustment for load capacitance and temperature compensation. This allows an application to also compensate for accuracy lost due to crystal aging[17].

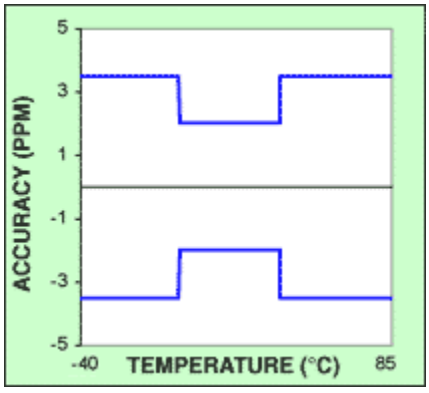


Figure 12: Worst-case accuracy of the DS3231S.

Chapter 8

Conclusion & Future Works

8.1: Conclusion

In our project, we have tested multiple methods of time synchronization for a reliable, accurate and feasible system. The objective of our thesis study was to give the investors as well as the consumers a clear solution. We considered various cases including both wired and wireless communication. We have discussed elaborately the pros and cons of every case and also showed the economic analysis. However, Experiment result of our model depicts that the results were quite consistent with the expected output therefore making this a viable solution

8.2: Challenges

There have been some difficulties and challenges that we faced while doing the thesis. Firstly, there did not have enough comparative analysis based papers in the internet so we had lack of ideas how we could present the cases more attractively. Secondly In a distributed system the problem of time synchronization takes on more complexity because a global time is not easily

known. The most used clock synchronization solution on the Internet is the Network Time Protocol (NTP) which is a layered client-server architecture based on UDP message passing.. In a wireless network, used in our case, the problem becomes even more challenging due to the possibility of collision of the synchronization packets on the wireless medium and the higher drift rate of clocks on the low-cost wireless devices. Thirdly, since the microcontroller simultaneously exchanges information over the wifi network as well as running the clock display, the one millisecond multiplexing time interval gets continuously interrupted and because of this reason there's a display flickering noticeable. However, this issue is surmountable by adding another uC and splitting the tasks of WiFi and display running between two uC. Lastly components availability and cost was also an issue which directly related to performance.

8.3: Future Works

The thesis has a vision of improvement its analysis with more resources. The possible future works of the thesis has been discussed below:

- This thesis work concentrates on time synchronization. In our future work we hope to add further features eg. Instant message display to make this a versatile solution
- Instead of communicating over Wi-Fi we hope to integrate our system with Low Power Wide Area Networks such as LoRaWAN. Unlike Wi-Fi or cellular networks LPWAN is designed to communicate within an widespread area whilst maintaining minimal power consumption and cost of deployment
- We also hope to incorporate device to device communication instead of being segregated

- Lastly, we wish to include higher end components in order to ensure the accuracy and efficiency of the system

REFERENCES:

[1]http://modbus.control.com/thread/1026158763#1026158763

[2]http://www.bb-elec.com/Learning-Center/All-White-Papers/Serial/Basics-of-the-RS-485-Standard.aspx

[3]http://collections.rmg.co.uk/collections/objects/79654.html

[4]https://www.unixtimestamp.com/

[5]http://www.ntp.org/ntpfaq/NTP-s-def.htm

[6]https://www.epochconverter.com/

[7]https://searchnetworking.techtarget.com/definition/UDP-User-Datagram-Protocol

[8]https://www.techopedia.com/definition/13460/user-datagram-protocol-udp

[9]https://www.techopedia.com/definition/508/ieee-80211x

[10]https://en.wikipedia.org/wiki/Wi-Fi

[11]https://www.sparkfun.com/products/13678

[12]https://www.espressif.com/en/products/hardware/esp8266ex/overview

[13]https://learn.adafruit.com/ds1307-real-time-clock-breakout-board-kit/what-is-an-rtc

[14]https://en.wikipedia.org/wiki/Real-time_clock

[15]https://www.electronics-tutorials.ws/blog/7-segment-display-tutorial.html

[16]https://www.electronics-tutorials.ws/sequential/seq_5.html

[17] https://www.maximintegrated.com/en/app-notes/index.mvp/id/3566

[18]https://www.maximintegrated.com/en/products/digital/real-time-clocks/highly-accurate-real-time-clocks.html

[19]https://en.wikipedia.org/wiki/Cristian%27s_algorithm

[20]https://en.wikipedia.org/wiki/Round-trip_delay_time

[21] https://nettigo.eu/

[22] https://techshopbd.com/

[23] http://marianlonga.com/7-segment-led-display/

[24] http://www.instructables.com/id/74HC595-Shift-register/

[25] https://goo.gl/images/4XqYZ1

Appendix:

Source Code:

```
#include <ESP8266WiFi.h>
#include <TimeLib.h>
#include <WiFiUdp.h>

#include <Wire.h>
#include "RTClib.h"

int aM = 8192;
int pM = 16384;
int aMpM;

RTC_DS1307 rtc;

char ntpServerName[] = "us.pool.ntp.org";
char ntpServerName[] = "172.16.2.104";

int timeZone = 6;
WiFiUDP Udp;
unsigned int localPort = 8888;

time_t getNtpTime();
int digitalClockDisplay(char d

void printDigits(int digits);
void sendNTPpacket(IPAddress &address);

void attempt_wifiConnect();
void showOutput();
void loadRTC();

void runDisplay();
void do_peekAboo();
```

```
int digitalClockDisplay(char m){
   Serial.println("NOW INSIDE THE digitalClockDisplay() FUNCTION");
   int interm;

   if (m=='h'){
     interm=hour();
     return interm;
   }

   if (m=='m'){
     interm=minute();
     return interm;
   }

   if (m=='s'){
     interm=second();
     return interm;
   }

   if (m=='j'){
     interm=day();
     return interm;
   }

   if (m=='n'){
     interm=month();
     return interm;
   }
   if (m=='y'){
     interm=year();
     return interm;
   }
}
const int NTP_PACKET_SIZE = 48;
byte packetBuffer[NTP_PACKET_SIZE];

time_t getNtpTime() {
   Serial.println("NOW INSIDE THE getNtpTime() FUNCTION");

   IPAddress ntpServerIP;
```

```
  while (Udp.parsePacket() > 0) ;

  WiFi.hostByName(ntpServerName, ntpServerIP);

  Serial.print(":");
  Serial.println(ntpServerIP);
  sendNTPpacket(ntpServerIP);
  uint32_t beginWait = millis();

  while (millis() - beginWait < 1500) {
    int size = Udp.parsePacket();

    if (size >= NTP_PACKET_SIZE) {

      ntpPhase = true;
      Udp.read(packetBuffer, NTP_PACKET_SIZE);  // read packet into the buffer
      unsigned long secsSince1900;

      secsSince1900 =  (unsigned long)packetBuffer[40] << 24;
      secsSince1900 |= (unsigned long)packetBuffer[41] << 16;
      secsSince1900 |= (unsigned long)packetBuffer[42] << 8;

      secsSince1900 |= (unsigned long)packetBuffer[43];
      return secsSince1900 - 2208988800UL + timeZone * SECS_PER_HOUR;
    }
  }

  Serial.println("No NTP Response :-(");
  ntpPhase = false;
  return 0;
}

void sendNTPpacket(IPAddress &address)  //4
{


  memset(packetBuffer, 0, NTP_PACKET_SIZE);
  packetBuffer[0] = 0b11100011;
  packetBuffer[1] = 0;
```

```
  packetBuffer[2] = 6;
  packetBuffer[3] = 0xEC;


  packetBuffer[12] = 49;
  packetBuffer[13] = 0x4E;
  packetBuffer[14] = 49;
  packetBuffer[15] = 52;

 Udp.beginPacket(address, 123);
  Udp.write(packetBuffer, NTP_PACKET_SIZE);
  Udp.endPacket();
}

boolean wifiPhase;
int attemptCnt = 0;


  if( WiFi.status() == WL_CONNECTED ) {
    wifiPhase = true;

    Serial.println("Your IP is");
    Serial.println((WiFi.localIP()));

    Udp.begin(localPort);

    setSyncProvider(getNtpTime);
    setSyncInterval(300);
  }

  Serial.print("THE wifiPhase: ");
  Serial.println(wifiPhase);
  Serial.print("THE ntpPhase: ");
  Serial.println(ntpPhase);

  //RTC
  Wire.begin();
  rtc.begin();
  Serial.println("wire and RTC have begun");
```

```
    loadRTC();

    Serial.println("");
}

void setup() {

    pinMode(latchPin, OUTPUT);
    pinMode(dataPin, OUTPUT);
    pinMode(clockPin, OUTPUT);

    Serial.begin(115200);

    WiFi.disconnect();
    delay(3000);
    Serial.println("START");


    ////////////////////////////////////////////////
    ////////////////////////////////////////////////
    ////////////////////////////////////////////////
    WiFi.begin("UB5 3rd Floor Wifi","poweradmin");
    ////////////////////////////////////////////////
    ////////////////////////////////////////////////
    ////////////////////////////////////////////////


    while ((!(WiFi.status() == WL_CONNECTED))){
        delay(300);
        Serial.print(".");
        attemptCnt++;

        if(attemptCnt >=35){
          attemptCnt =0;
          wifiPhase = false;
          ntpPhase = false;
          break;
        }
    }
```

```
  Udp.begin(localPort);
  setSyncProvider(getNtpTime);
  setSyncInterval(300);

  //RTC
  Wire.begin();
  rtc.begin();


  while (! rtc.isrunning() ) {
    attemptRTC++;

    Serial.println("attempt RTC..");
    if(attemptRTC >10){
      attemptRTC =0;
      Serial.println("RTC is NOT running!");
      rKey = false;
      break;
    }
else
{
       if(ntpPhase == true){
       //set the date and time
       rtc.adjust($year , $month , $day ,$hour , $minute , $second ));
       }
    }
  }
  attemptRTC =0;
  if (rtc.isrunning() )
{
    rKey = true;
    if(ntpPhase == true){

       //set the date and time
rtc.adjust($year , $month , $day ,$hour , $minute , $second ) );
       Serial.println("SETUP: RTC has been updated with NTP");
    }
  }
```

```
   attempt_wifiConnect();

   //  RTC
   //  Wire.begin();
   //  rtc.begin();

   Serial.println("");
   loadRTC_time = digitalClockDisplay('m');
   Serial.print("loadRTC_time = ");
   Serial.println(loadRTC_time);
   Serial.println("SETUP FUNCTION EXITING");
   Serial.println("");
}

int updateTimeAt = 30;

void loop()
  {

   /*
if ( (aMpM == aM ) && ( hour() < 6 ) ) {
     key = true;
   } else {
     key = false;
   }
   */


  ////////////////////////////////////////////////////////////////////////////////////////
  ////////////////////////////////////////////////////////////////////////////////////////
  /////////              AAAAALLLLLGGGGGOOOOORRRRRYYYYYTTTTTHHHHHMMMMM
//////////
  ////////////////////////////////////////////////////////////////////////////////////////
  ////////////////////////////////////////////////////////////////////////////////////////

   if ( ( $minute == updateTimeAt + 1 )  ) {
     Serial.println(" NOW acs_k");
     access_key = true;
   }
```

```
  if ( access_key == true &&$minute == updateTimeAt && fkey == false ) {

    Serial.println("NOW AT THE STATE TO UPDATE loadRTC_time");
    loadRTC_time = $minute;

    Serial.println(loadRTC_time);
    fkey = true;
    access_key = false;
   }

  if ( ( fkey == true ) )
{

     Serial.println(loadRTC_time);
     attempt_wifiConnect();
     fkey = false;
   }


  showOutput();
  runDisplay();

  // delay(15000);
 }



void runDisplay(void) {

 Serial.println("NOW IN runDisplay() FUNCTION");

  if( rKey == false){
     //run by ntp
  }
  else{
     //get the date and time from RTC

     DateTime now = rtc.now();
```

```
      minute2 = now.minute();                    // Multiplex Minute from RTC
      second2 = now.second();

 fnHour = hour2 / 10;                        // First Number Hour
      snHour = hour2 % 10;                        // Second Number Hour

 fnMinute = minute2 / 10;                    // First Number Minute
      snMinute = minute2 % 10;                    // Second Number Minute

 fnSecond = now.second() / 10;
      snSecond = now.second() % 10;

      Serial.println("***   OUTPUT OF DIGIT OPERATIONS   ***");
      Serial.println("***   fnHour, snHour   ***");

   Serial.println(fnHour);
      Serial.println(snHour);

    Serial.println("***   fnMinute, snMinute   ***");
      Serial.println(fnMinute);

     Serial.println(snMinute);
      Serial.println("***   fnSecond, snSecond   ***");

  Serial.println(fnSecond);
      Serial.println(snSecond);
      do_peekAboo();
  }
}

int digitCode [] = {63, 6, 91, 79, 102, 109, 125, 7, 127, 111};
#define del 2

void do_peekAboo(void){
  Serial.println(" ");
  Serial.println("****OPERATION TO FOCUS ON***");
  Serial.println("****************************");
```

```
//fnHour
if(fnHour==1){
  Serial.println(fnHour);
  digitalWrite(latchPin, LOW);
  shiftOut(dataPin, clockPin, MSBFIRST, 128 + digitCode[fnHour] >>8 );
  shiftOut(dataPin, clockPin, MSBFIRST, 128 + digitCode[fnHour]);
  digitalWrite(latchPin, HIGH);
  delay(del);
}

//snHour
Serial.println(snHour);
digitalWrite(latchPin, LOW);
shiftOut(dataPin, clockPin, MSBFIRST, 256 + digitCode[snHour] >>8 );
shiftOut(dataPin, clockPin, MSBFIRST, 256 + digitCode[snHour]);
digitalWrite(latchPin, HIGH);
delay(del);

//fnMinute
Serial.println(fnMinute);
digitalWrite(latchPin, LOW);
shiftOut(dataPin, clockPin, MSBFIRST, 512 + digitCode[fnMinute] >>8 );
shiftOut(dataPin, clockPin, MSBFIRST, 512 + digitCode[fnMinute]);
digitalWrite(latchPin, HIGH);
delay(del);

//snMinute
Serial.println(snMinute);
digitalWrite(latchPin, LOW);
shiftOut(dataPin, clockPin, MSBFIRST, 1024 + digitCode[snMinute] >>8 );
shiftOut(dataPin, clockPin, MSBFIRST, 1024 + digitCode[snMinute]);
digitalWrite(latchPin, HIGH);
delay(del);

//fnSecond
Serial.println(fnSecond);
digitalWrite(latchPin, LOW);
shiftOut(dataPin, clockPin, MSBFIRST, 2048 + digitCode[fnSecond] >>8 );
shiftOut(dataPin, clockPin, MSBFIRST, 2048 + digitCode[fnSecond]);
digitalWrite(latchPin, HIGH);
```

```
  delay(del);

  //snSecond
  Serial.println(snSecond);
  digitalWrite(latchPin, LOW);
  shiftOut(dataPin, clockPin, MSBFIRST, 4096 + digitCode[snSecond] >>8 );
  shiftOut(dataPin, clockPin, MSBFIRST, 4096 + digitCode[snSecond]);
  digitalWrite(latchPin, HIGH);
  delay(del);

  //AM/PM
  Serial.println(aMpM);
  digitalWrite(latchPin, LOW);
  shiftOut(dataPin, clockPin, MSBFIRST, aMpM + 0 >>8 );
  shiftOut(dataPin, clockPin, MSBFIRST, aMpM + 0 );
  digitalWrite(latchPin, HIGH);
  delay(del);

}


void showOutput(void) {
    Serial.println("NOW IN showOutput() FUNCTION");

    Serial.println("");
    Serial.println("NTP Time");
    Serial.print((digitalClockDisplay('h')));
    Serial.print(":");
    Serial.print((digitalClockDisplay('m')));
    Serial.print(":");
    Serial.print((digitalClockDisplay('s')));
    Serial.println("");

    Serial.println("");
    Serial.println("RTC TIME");
    DateTime now = rtc.now();

    //print out the date and time to the serial line
    Serial.print(now.year(), DEC);
    Serial.print('/');
```

```
    Serial.print(now.month(), DEC);
    Serial.print('/');
    Serial.println(now.day(), DEC);
    //Serial.println('   ');
    Serial.print(now.hour(), DEC);
    Serial.print(':');
    Serial.print(now.minute(), DEC);
    Serial.print(':');
    Serial.print(now.second(), DEC);
    Serial.println();
}
```