

DOCUMENTATION.

Name of the Project- Smart Management of Bins

Duration- Jan-May 2018

Client- Dr. Vikram Bajaj.

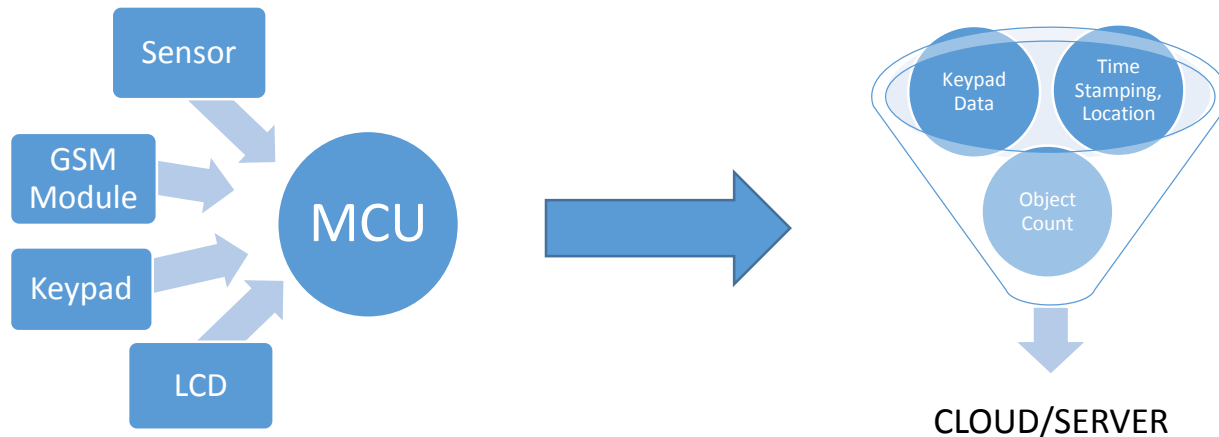
Intern- Vivek Onkarnath Rathi.

Guide/Mentor- Mr. Lalit Kumar.

ABSTRACT-

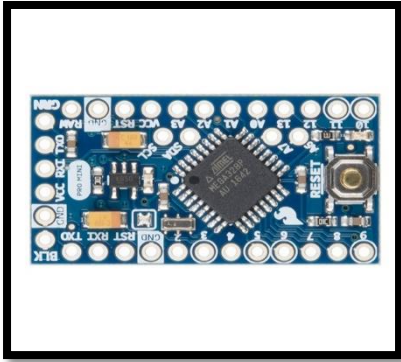
We are surrounded by electronic gadgets, why not make them smart and efficient with a flavor of IoT. Internet of Things is the next big thing in this era and one of the utility for India's 2020 mission. In this IoT based project, we are using primitive technologies to send the environmental data to the cloud platform. In a nutshell, it connects to the cloud and sends the relevant data packets to the cloud. Entire project is built using GSM-2G, UART, TCP, HTTP technologies. Furthermore it has incorporated a boot-loader and IAP aka In Application Programming. In this specific application, assume a dockyard, where a worker puts/loads an object in the inventory, object is being detected by the Ultrasonic Sensor. Then the worker enters his/her ID (a mobile number) using a keypad and presses enter key to send that relevant data to the cloud, with LCD displaying his ID and relevant message after he/she hits the enter key. Using the available database through the device at the cloud side various data handling algorithms can be applied as per the client requirement.

CONCEPT-



PROJECT RUDIMENTRY STAGE-

Why Arduino pro mini-?



1. Small and compact in size i.e. form factor.
 2. Works on both 3.3V and 5V logic levels.
 3. Quectel MC60 works on 3.3V logic level.
 4. Owing to point 3 we started the prototype with pro mini.
- (*however later it was verified through experimentation that Quectel EVB board converts both 3.3V or 5V logic levels compatible to the voltage level of Quectel MC60)

Ref- <https://learn.sparkfun.com/tutorials/using-the-arduino-pro-mini-33v/what-it-is-and-isnt>

TOOLS/HARDWARE-

1. Quectel MC60

MC60 is GSM/GPRS+ GNSS combo module.

GPRS- General Packet Radio Service.

GNSS- Global Navigation Satellite System.

Voltage- 3.3 to 4.6 V DC (4V Typically)

Current- 1.2 mA

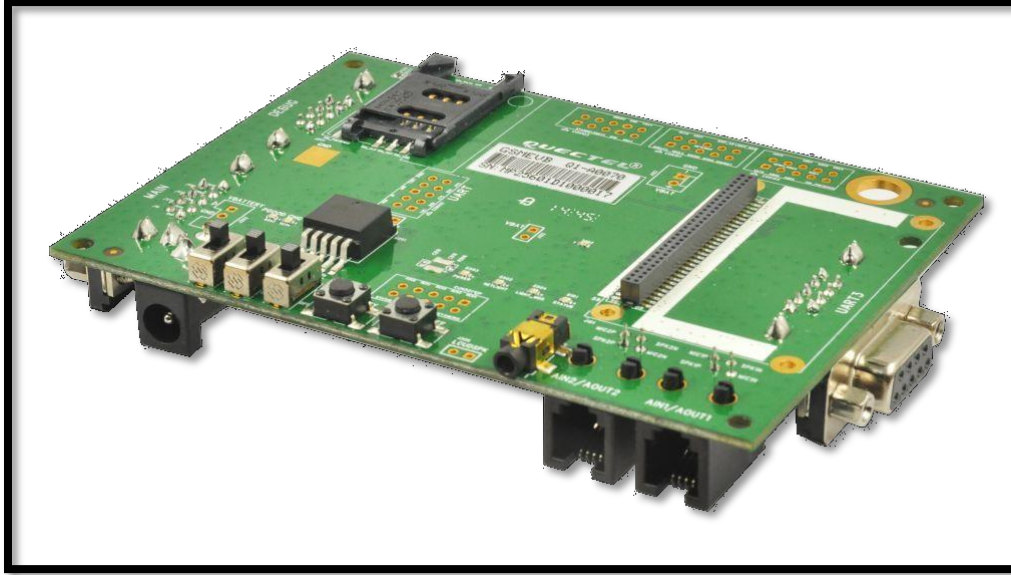
Antennas must be connected on the appropriate locations on the MC60 breakout board.



2. Quectel EVB board-

1. Here UART works at 3.3V logic levels.
2. Hence 5V logic signals are incompatible for UART TX.
3. The board has inbuilt circuitry that converts 5V signals or 3.3V signals to desired logic levels of Quectel MC60.

4. No need to make a separate voltage divider circuitry- (It doesn't work when such circuitry is used- Tried this, never worked, also debugged using voltages through multimeter.)
5. On board for the intended working, following must be the configuration of the DIP switches.
POWER- ON
VCHG- OFF
D/L- ON



(Refer MC60 documentation for profound knowledge)

3. Arduino Pro Mini-

Programming through FTDI

Wrote interfaces for I2C-LCD, 4*3 Keypad, GSM Communication

In GSM Communication-

1. UART
2. Used Software Serial library
3. Basic AT Commands
4. TCP server connection
5. Send data to the server

(Refer default libraries and documentation of arduino)

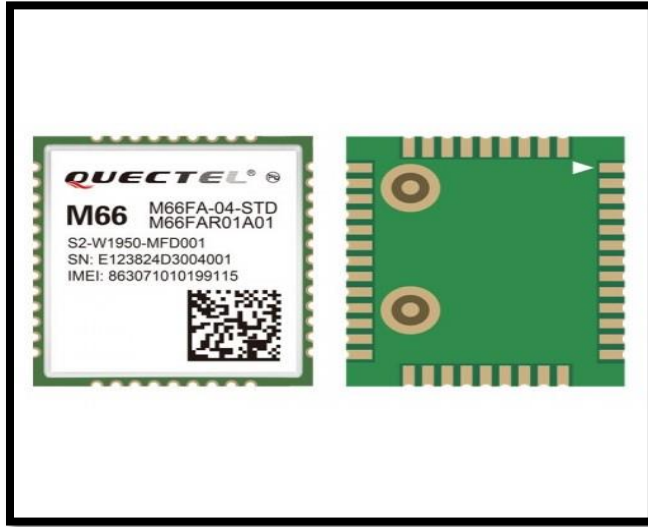
*When working with pro mini for UART, baud rate must be doubled in IDE, i.e. if baud rate was set to 9600 bps in the code, in IDE for Serial Monitor use baud rate of 19200 bps. Why? Answer- Pro Mini works on 8 MHz crystal and IDE has been configured for 16 MHz crystal.

4. Quectel M66

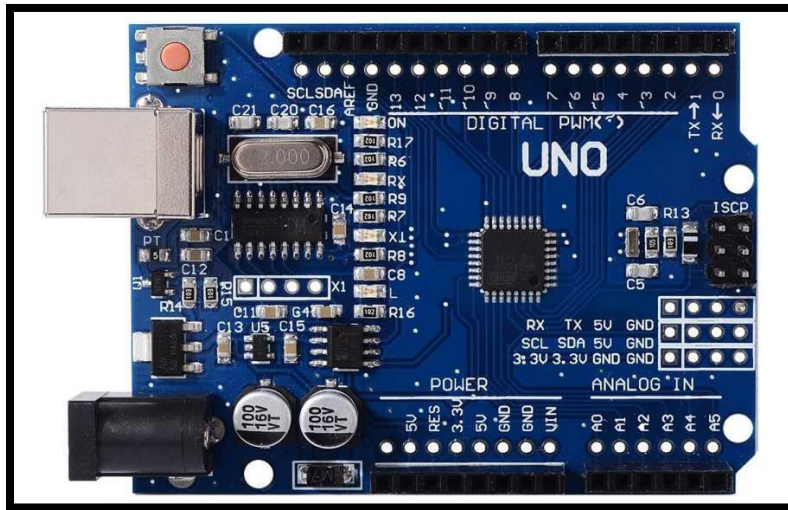
GSM/GPRS module

Voltage- 3.3 to 4.6 V DC (4V Typically)

Current- 1.3 mA



5. Arduino UNO-



It was easier to use UNO for testing the functionality due to 5V logic levels.

Wrote interfaces for I2C-LCD, 4*3 Keypad, GSM Communication.

In GSM Communication-

1. UART
2. Used Software Serial library
3. Basic AT Commands
4. TCP server connection
5. Send data to the server

(**For both arduinos in the IDE Serial monitor, use return type as both CR and NL)

Problems encountered with Arduinos

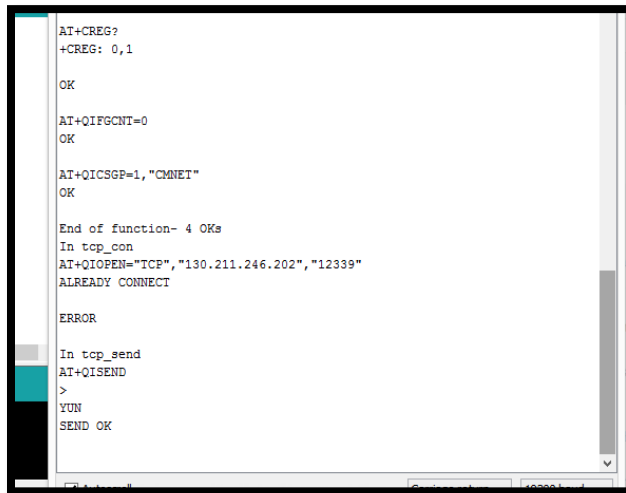
1. I2C LCD- pro mini works on 3.3 V logic levels, Hence LCD doesn't understand high and low signals send through pro mini.
How?=> The code and same hardware was tested for Arduino UNO, it worked fine which led to the conclusion that the LCD didn't work on 3.3V logic level, also the LCD we had, was 5V DC LCD rather than 3.3V DC LCD.
2. I2C LCD- minimum voltage between VDD and VEE must be 4.5 V DC.
3. Keypad- When a single key is pressed, it responds twice or thrice- i.e. de-bouncing issue.
How? => Using Serial Communication we used to verify the pressed key, also later LCD was used to verify the pressed key, here pressing a key once responded with the o/p twice/thrice, hence it was concluded that it was de-bouncing issue. It could be solved only by using hardware i.e. resistors or something like that.
4. GSM Communication-
It is basically UART-
 1. "Send Ok" issue, after data has been sent to TCP server. Why?=> The UART buffer of the pro mini doesn't get cleared or flushed.

As it can be seen here, after data has been sent to the server "SEND OK" wasn't received. Hence, we arrived at the conclusion that the buffer might not be cleared even after using Serial.read()

2. Serial.println()- issue, hence used Serial.readString()

Serial.readString()- reads characters from the serial buffer into a string. The function terminates if it times out.

Now, “SEND OK was received”

A screenshot of a serial monitor window with a black background and white text. The text shows the following sequence of AT commands and responses: AT+CREG? followed by +CREG: 0,1 and OK; AT+QIFGCNT=0 followed by OK; AT+QICSGP=1,"CMNET" followed by OK; a status message "End of function- 4 OKs"; "In tcp_con"; AT+QIOPEN="TCP","130.211.246.202","12339" followed by ALREADY CONNECT; ERROR; "In tcp_send"; AT+QISEND followed by > and YUN; and finally SEND OK. The window has a scrollbar on the right and a status bar at the bottom with buttons like "Data transfer", "Open serial port", and "Reset".

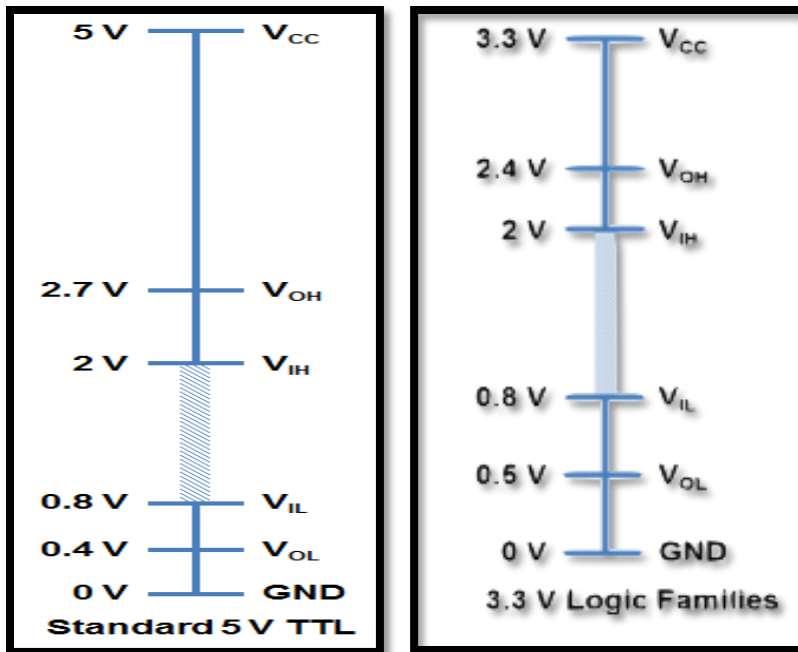
```
AT+CREG?  
+CREG: 0,1  
  
OK  
  
AT+QIFGCNT=0  
OK  
  
AT+QICSGP=1,"CMNET"  
OK  
  
End of function- 4 OKs  
In tcp_con  
AT+QIOPEN="TCP","130.211.246.202","12339"  
ALREADY CONNECT  
  
ERROR  
  
In tcp_send  
AT+QISEND  
>  
YUN  
SEND OK
```

The “received” statement wasn’t received always. Sometimes only. The above code worked fine for most of the time but it wasn’t a robust solution. It was grueling to analyze the problem and get the solution.

5. *Difficult to debug the program using Arduino IDE*
6. Alternative Solution- use Atmel Studio and debug using register values
7. *Better shift to advanced processor*

TAKEAWAYS FROM ARDUINO-

1. Working with 3.3V logic levels
Ref -<https://learn.sparkfun.com/tutorials/logic-levels>

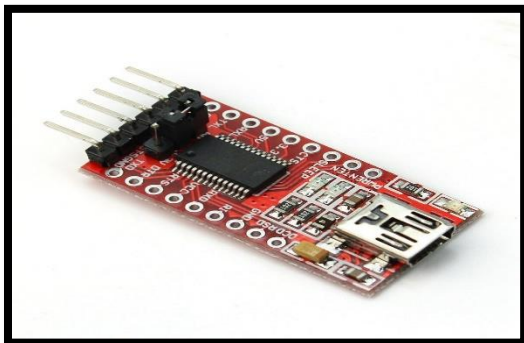


For 3.3V devices the highest voltage is 3.3V itself and GPIO pins provide I/O voltages with VCC i.e 3.3V/5V as high and 0V as low. Working with devices or sensors that need 5V becomes cumbersome when working with controller working on 3.3V logic levels. This leads to various voltage divider resistor ladder network which makes system kind of complicated or calls for a level shifter IC. We need 5V for LCD, Ultrasonic Sensor- better migrate to 5V logic level.

2. What is FTDI? (**Future Technology Devices International**)

USB to UART chip/converter (FT232R)/ USB to Serial Converter

Arduino pro mini lacks the circuitry for USB to Serial Converter unlike UNO, hence to program it one needs to use FTDI breakout board.



3. What is GSM? (Global System for Mobile Communication)

Ref- https://www.tutorialspoint.com/gsm/gsm_overview.htm

GSM is also known as 2G-

Why 2G?

1. Hardware Cost
2. Data Service Cost

4. What are AT commands?

AT commands are instruction used to control a modem. AT is abbreviation of Attention. Every command line starts with “AT” or “at”. Modem commands are called as AT commands.

Basic structure-

/r/n command /n /r

<CR><LF>Command<CR><LF>

<CR>- carriage return- i.e end of command (ASCII-13)

<LF>- line feed, i.e. new line (ASCII- 10)

(Refer Quectel Documents for AT commands)

5. What is TCP? (Transmission Control Protocol)

1. 2 hosts establish a connection and exchange streams of data
2. It guarantees delivery of data and also the order in which data was sent.

6. UART Working

7. I2C communication

8. I2C driver IC- PCF8574

It is basically 8 bit I/O expander for I2C Bus. It can be used to extend the GPIO connections.

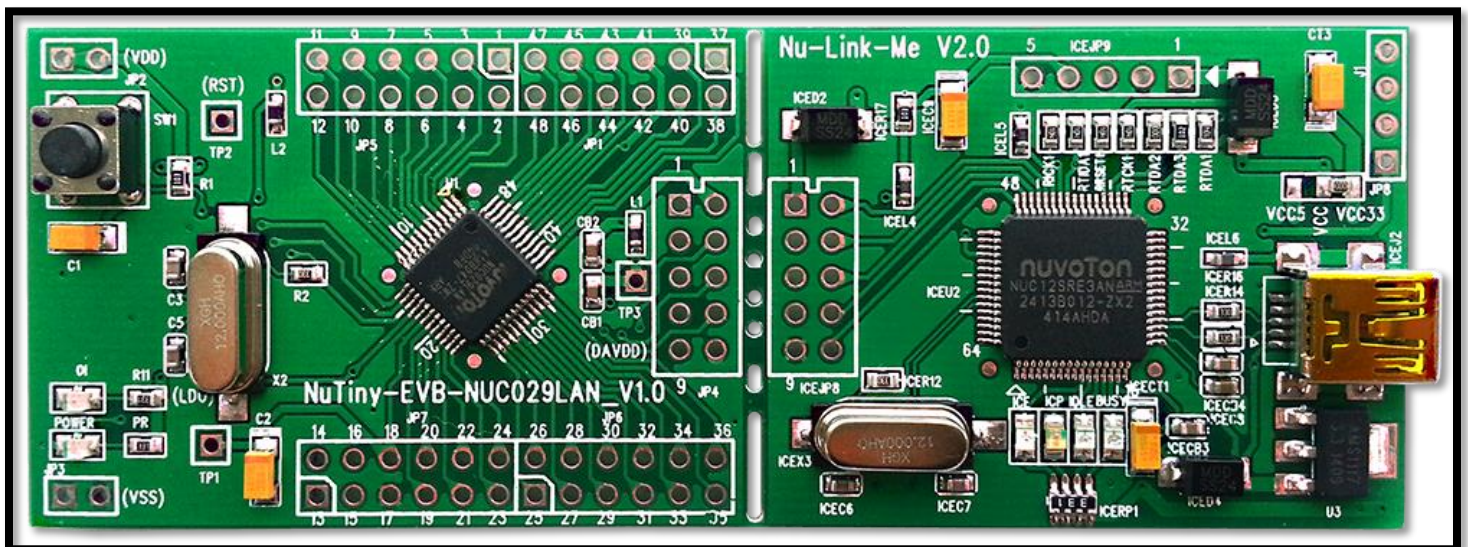
As in pro mini we have limited GPIOs, we used 8bit I/O expander for I2C Bus which uses I2C protocol.

9. LCD working-

Note- There must be at least 4V DC (4.5V DC) potential difference between VCC and VEE pins
(Clashes between 3.3V and 5V logic levels)

MIGRATING TO NUVOTON M0516LDN-

1. Nuvoton M0516LDN

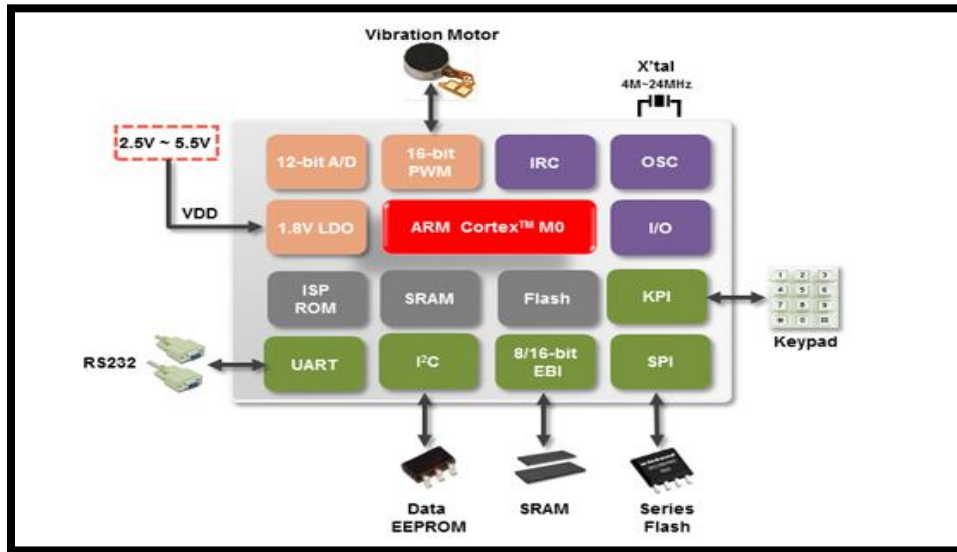


1. ARM Cortex M0 processor

ARM-Advanced RISC (Reduced Instruction Set Computer) Machine

Cortex M0- Most primitive core, and cheaper.

2. 32 bit controller/ Architecture



3. Learning Nuvoton M0516LDN

We need to have some system startup files before we use the controller/ development board-

These are-

CMSIS (Cortex Microcontroller Software Interface Standard) files-

You will find those .c or .s or .h files in the controller firmware.

Also you need to have specific piece of code for the controller's relevant functions such as system clock configurations and all, you must include those functions.

e.g.

```
57 /* SYS_Init()- System Initialisation, configures system registers */
58 void SYS_Init(void)
59 {
60     /* Enable Internal RC 22.1184MHz clock */
61     CLK->PWRCON |= CLK_PWRCON_OSC22M_EN_Msk;
62
63     /* Waiting for Internal RC clock ready */
64     while(!(CLK->CLKSTATUS & CLK_CLKSTATUS_OSC22M_STB_Msk));
65
66     /* Switch HCLK clock source to Internal RC and HCLK source divide 1 */
67     CLK->CLKSELO = (CLK->CLKSELO & (~CLK_CLKSELO_HCLK_S_Msk)) | CLK_CLKSELO_HCLK_S_HIRC;
68     CLK->CLKDIV = (CLK->CLKDIV & (~CLK_CLKDIV_HCLK_N_Msk)) | CLK_CLKDIV_HCLK(1);
69
70     /* Set PLL to Power down mode and HW will also clear PLL_STB bit in CLKSTATUS register */
71     CLK->PLLCON |= CLK_PLLCON_PD_Msk;
72
73     /* Set core clock as PLL_CLOCK from PLL */
74     CLK->PLLCON = CLK_PLLCON_50MHz_HIRC;
75     while(!(CLK->CLKSTATUS & CLK_CLKSTATUS_PLL_STB_Msk));
76     CLK->CLKSELO = (CLK->CLKSELO & (~CLK_CLKSELO_HCLK_S_Msk)) | CLK_CLKSELO_HCLK_S_PLL;
77
78     /* Update System Core Clock */
79     /* User can use SystemCoreClockUpdate() to calculate PllClock, SystemCoreClock and CyclesPerUs automatically. */
80     PllClock = PLL_CLOCK; // PLL
81     SystemCoreClock = PLL_CLOCK / 1; // HCLK
82     CyclesPerUs = PLL_CLOCK / 1000000; // For CLK_SysTickDelay()
83
84 }
```

1. Basic GPIO programming

1. LED blinking using registers (Register based programming)- Individual Registers were configured by referring the datasheet.

2. LED blinking using C programs (Std driver based programming)

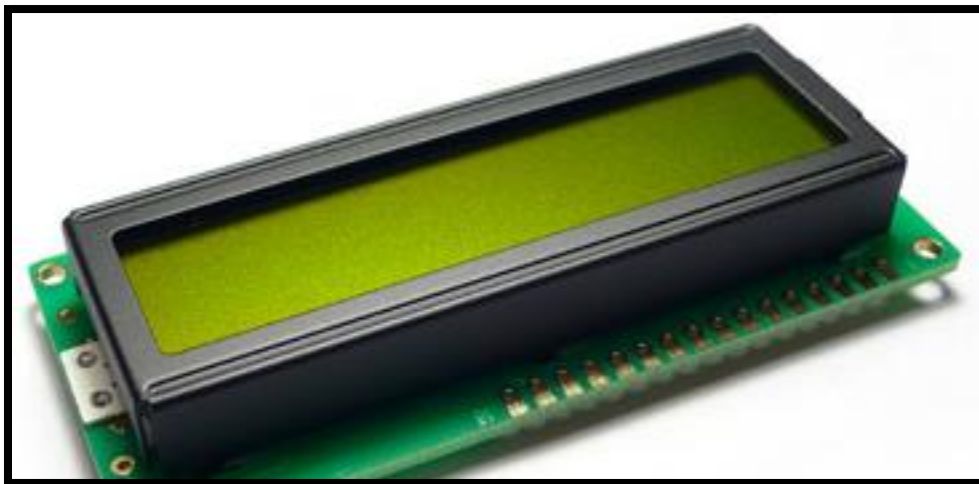
3. LCD display
4. 4 bit LCD display
5. 4*3 keypad
2. TIMERS programming-
3. UART
4. I2C
5. Interrupts- While programming with interrupts, if you need to use certain variable again and again, use global variables against local ones.
6. Capture
7. ADC

(* Like Arduino serial monitor, you can use UART1 as serial monitor where you can use printf() functions to print data for Nuvoton M0516LDN by using SEMIHOST methodology, it is mentioned in the firmware)

4. Programming

1. LCD display-
8 bit LCD display- Works fine (refer the code)
4 bit LCD display- Works fine (refer the code)
(* 4bit LCD- Code was modified to only toggle the specified bits or pins rather than a port, use bits logic and left and right shift operators)

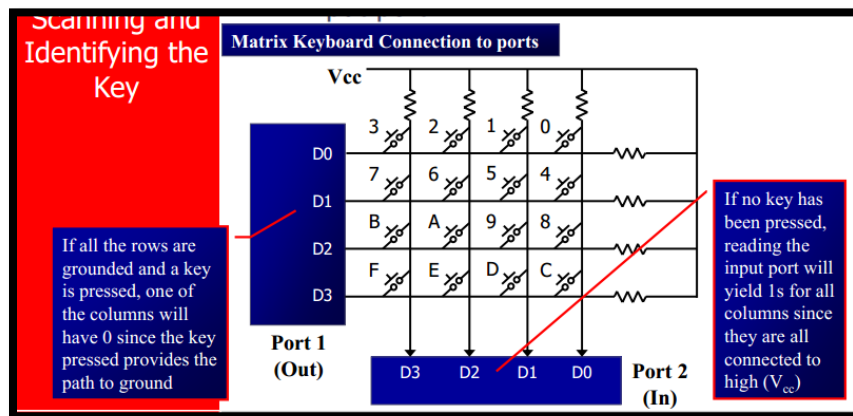
It strictly needs 5V DC voltage, doesn't work on 3.3V levels.



2. 4*3 keypad-

In Keypads, rows are o/p, columns are inputs

1. We need to use external pull ups to make columns high, as the i/p configuration makes pins low
2. If we reverse logic, i.e. instead of grounding rows, make them high and then see if column goes high, this logic is ambiguous as it has no reference to the ground i.e. common voltage. (The high state of the column is the floating one)



3. Ultrasonic Sensor

1. Use echo pin as interrupt.
2. Refer datasheet to program the sensor.
3. * Use timeout condition (else system hangs), why? Answer- If we don't give timeout, there is a case where signal isn't returned i.e. echo pin doesn't go low, it still waits for the time and eventually system gets hung.
How did we realize- The code used to get hang up the system often, googled for other codes, checked Arduino header files for ping library, analyzed the available resources and codes and came up with the timeout condition.

```

GPIO_EnableInt(P1, 3, GPIO_INT_RISING);
/* Waiting for interrupts */
while(!flag);
flag=FALSE;
GPIO_EnableInt(P1, 3, GPIO_INT_FALLING);
while(!flag);
TIMER_Close(TIMER1);
c=0;
flag=FALSE;
GPIO_DisableInt(P1, 3);

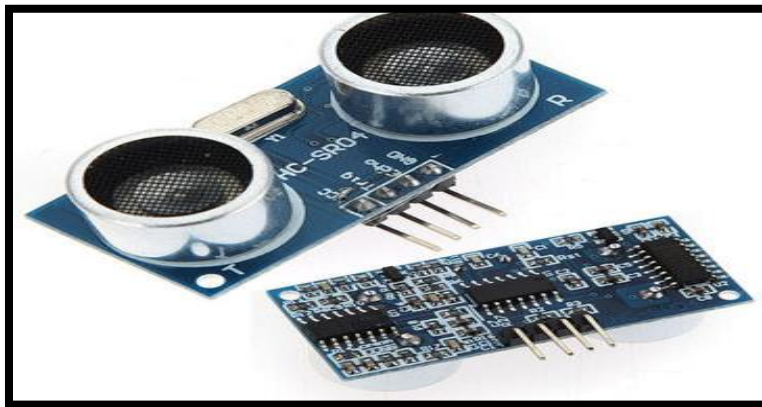
```

```

GPIO_EnableInt(P1, 3, GPIO_INT_RISING);
/* Waiting for interrupts */
while(!flag);
flag = FALSE;
GPIO_EnableInt(P1, 3, GPIO_INT_FALLING);
while(!flag && TIMER1->TDR-c1<500000);
flag = FALSE;
GPIO_DisableInt(P1, 3);

```

4. It is kind of polling only but we use ISRs to get timing values i.e. pulse duration using TIMER
5. It works on 5V only, needs voltage divider for 3.3V logic level
6. Use flag variables to avoid faulty detection



Issues-

1. Tried implementing TIMER CAPTURE feature, but it didn't work, debugged using debugger, checked the flag values, still couldn't figure out the bug, hence opted for an alternative solution.
2. There was no timeout so system used to get hung up.
3. Also, never use printf() in the ISR, as printf() execution timings delay the working of hamper the working of the system.
4. Also hampering wires of the sensor cause it fail i.e. working fails
5. Make sure you give VCC as 5V from the controller as trigger and echo also needs 5V and use controller GND. (These VCC-GND connections can also be the cause/ is the cause for the sinister working of the sensor)

4. I2C LCD

1. Basic I2C communication
2. Use oring and anding properly i.e (|| and &&)



Issues-

1. There was no ORING operator used and hence the voltage levels changed continuously- How did we realize? Answer- Checked voltage levels using multimeter and read datasheet for more understanding.
2. Also referred Arduino I2C library for reference, here we encountered fully different approach, tried to implement it, but didn't work, probably logic in my code was wrong. But latter by reading datasheet and thinking about bugs in the code, got the solution.
3. Read the datasheet effectively for efficient writing of the code.

5. ADC- SHARP IR Sensor

1. Program using ADC
2. Refer datasheet for the program



Issues-

1. Difference in the theoretical and practical values
2. Range Issue
3. Why these all issues? Don't Know, instead of debugging the cause-effect relationship, migrated to Ultrasonic Sensor as it was easily available, well suited and cheaper.

(* All debugging for failures was done using debugger, using breakpoints though software i.e. KEIL All hardware debugging was done using multimeter, checking voltage levels and values.)

6. UART

1. GSM Communication-

Basically we are using GSM i.e. 2G network i.e. internet to share or transmit data to the server
We are using TCP

Here we send some AT commands to the Quectel MC60 module and Receive the response.

This is all done using UART

The AT commands used can be referred in the code and Quectel documentation.

2. UART programming

1. Initially did basic transmission and reception of characters using M0516LDN and FTDI
2. Used UART as an interrupt
3. Stored received data into buffer

4. For GSM programming-

1. Transmitted a command
2. Enabled Interrupt
3. Waited for 10-50 ms
4. Disabled the Interrupt

(The above mentioned worked, but not always)

Solution

5. Transmitted a command
6. Waited for first 2 bytes of data to be received
7. Enabled Interrupt
8. Waited for 10-50 ms
9. Disabled the interrupt

(Worked for almost all cases)

This was done for basic GSM commands

The screenshot shows an IDE with three main panels. On the left, a 'Registers' panel lists core registers (R0-R15) and system registers (xPSR). The middle panel displays assembly code for lines 550 to 561. Line 550 is highlighted: `550: SYS->REGWRPROT = 0x59; MOVs r0,#0x59`. Line 551: `551: SYS->REGWRPROT = 0x16; MOVs r1,#0x16`. Line 552: `552: SYS->REGWRPROT = 0x88; MOVs r1,#0x88`. Line 553: `553: while(SYS->REGWRPROT != SYS_REGWRPROT_REGPROTDIS_Msk) {`. Line 554: `554: }`. Line 555: `555: void SYS_ClearResetSrc(uint32_t u32Src);`. Line 556: `556: uint32_t SYS_GetBODStatus(void);`. Line 557: `557: uint32_t SYS_GetResetSrc(void);`. The right panel shows UART data for UART #1, including 'AT', 'OK', 'AT+QIFGCNT=0', 'AT+QICSGP=1,"CMNET"', 'AT+QIOPEN="TCP","130.211.246.202","12339"', 'ALREADY CONNECT', 'ERROR', 'AT+QISEND', '> Hello', 'SEND OK', 'receivedAT+QISEND', '> Mumbai', 'SEND OK', 'receivedAT+QISEND', '> Pune', 'SEND OK'.

Here data was sent to server/cloud and was acknowledged by receiving “received”.

For time stamping and location details, used GNSS commands
 GNSS commands take longer time to execute i.e. for reception of data

The screenshot shows an IDE with three main panels. On the left, a 'Registers' panel lists core registers (R0-R15) and system registers (xPSR). The middle panel displays assembly code for lines 149 to 163. Line 149: `149: {`. Line 150: `150: SUB sp,sp,#0x68`. Line 151: `151: unsigned char a[100]="0,1234"; //packet a`. Line 152: `152: //unsigned char c[100]="2,1234"; //packet c`. Line 153: `153: /* Unlock protected registers */`. Line 154: `154: SYS_UnlockReg();`. Line 155: `155: /* Init System, peripheral clock and multi-function I/O */`. Line 156: `156: SYS_Init();`. Line 157: `157: /* Lock protected registers */`. Line 158: `158: SYS_LockReg();`. Line 159: `159: /* Program Initialisation */`. Line 160: `160: UART0_Init(); //GSM and GNSS initialisation`. Line 161: `161: GPIO_init(); //KEYPAD Initialisation`. Line 162: `162:` . Line 163: `163:` . The right panel shows UART data for UART #1, including 'AT+QGNSSC?', '+QGNSSC: 1', 'OK', 'AT+QIFGCNT=0', 'AT+QICSGP=1,"CMNET"', 'AT+QIOPEN="TCP","130.211.246.202","12339"', 'ALREADY CONNECT', 'ERROR', 'AT+QGNSSRD="NMEA/RMC"', '+QGNSSRD: \$GNRMC,125526.000,A,1833.9745,N,07346.5219,E,0.00,27', 'OK', 'AT+QISEND', '> 0,1234,125526.000,1833.9745,N,07346.5219,E,050218', 'SEND OK'.

We need only specific information or data for further processing or Transmission

Also data received in the buffer cannot be always reliable, hence use pointers

Use-

`strstr(buffer,string);` - It basically finds the instance of string in buffer and provides a pointer to the first character of the string

Ref- https://www.tutorialspoint.com/c_standard_library/c_function_strstr.htm

After reception of data, extract the relevant data and store it in the buffer

Use-

`sprintf(buffer, “ %s %d %s”,data,integer,string);` - It basically formats or fills string buffer with other data structures like strings, integers and floats.

Ref- https://www.tutorialspoint.com/c_standard_library/c_function_sprintf.htm

** Sometimes “received” or “OK” is not received or full response of the system is not received, hence we send a blank Command to Quectel MC60, In this if it doesn’t receive anything(i.e. “received”), it just hangs up, hence use a timeout condition too (like wait max for 2-5 s and then break from the system). But eventually we got rid of “received” response from server and also blank commands, because it gave me unpredictable results sometimes and the buffer wouldn’t receive relevant data and I better thought to avoid them.

(All UART debugging was done using debugger, reading the buffer array and also using SEMIHOST)

Issue-

1. `printf()`- This function didn’t work aptly at many times, might be its buffers wasn’t cleared
Solution- Use debuggers and read buffer string.
2. Sometimes AT commands didn’t give intended response, this hampered system working.
Solution- Found causes for such responses and eliminated them, referred documentation for AT commands.

7. Watchdog Timer-

Why WDT? What is WDT?

A watchdog timer (WDT) is a hardware timer that automatically generates a system reset if the main program neglects to periodically service it. It is often used to automatically reset an embedded device that hangs because of a software or hardware fault.

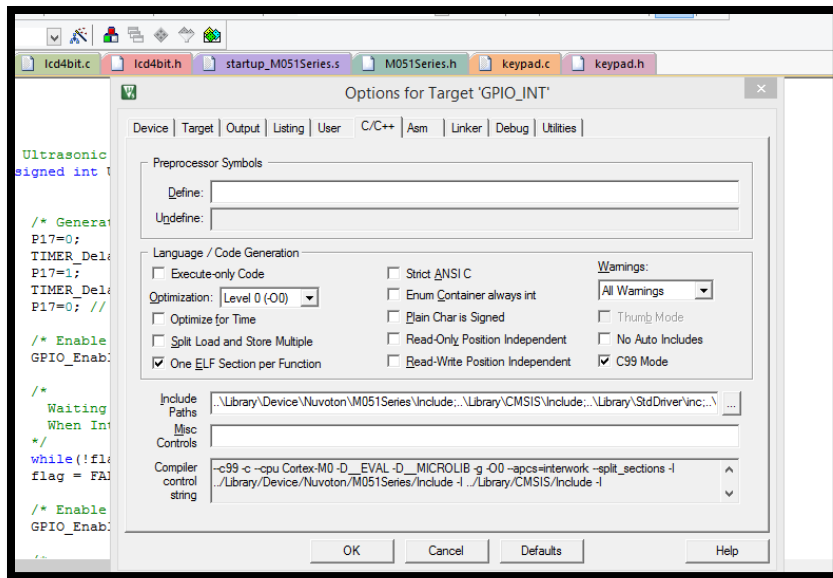
In our case, system could get hung up, if it didn’t receive the intended data, as we use interrupts, even for ultrasonic sensor also. Hence use WDT

1. Programmed watchdog reset after 25-26 seconds

C optimization and C99 compiler-

C Optimization Level 0-

This is done through Target Window



Ref- <https://www.geeksforgeeks.org/compiler-design-code-optimization/>
<https://en.wikipedia.org/wiki/C99>

BASIC IDEOLOGY OF PRODUCT/PROJECT-

1. Send 3 data packets
 - i. <0,device ID,time+date,lat,long>
This packet will be sent only once when the device boots
 - ii. <1,device ID,time+date,objectcount>
This packet is send in every 5 s if the object is detected
 - iii. <2,device ID,time+date,keypadentry,objectcount>
This packet is send when user enters keypad data and presses send/enter key

```
(data received', '0,1234,113829.000,1833.9758,N,07346.5063,E,200218\n')
(data received', '1,1234,200218113839.000,1\n')
(data received', '2,1234,200218113901.000,9420699906\n')
(data received', '1,1234,200218113909.000,2\n')
(data received', '2,1234,200218113921.000,9420699906\n')
(data received', '1,1234,200218113934.000,3\n')
(data received', '1,1234,200218113939.000,4\n')
(data received', '1,1234,200218113944.000,5\n')
(data received', '1,1234,200218113949.000,6\n')
(data received', '2,1234,200218113955.000,9420699906\n')
```

2. Ultrasonic- used for object detection and Keypad- Data entry are polled continuously in while(1) loop

ISSUES-

1. *UART didn't work in TIMER ISR? (Same priority ISRs can't be invoked in each other)*

Solution- Make TIMER ISR priority lower to UART as, higher priority ISR can be called in lower priority ISR

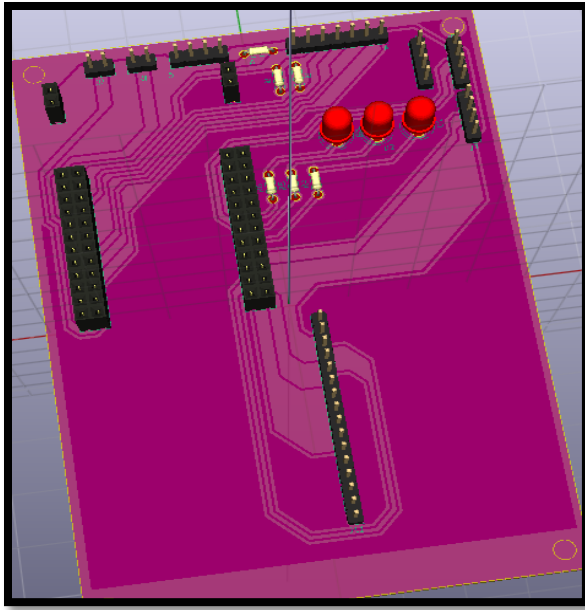
- 2. For same issue, we enabled Hardware Flow Control- (It is beneficial, but how for above case, not sure) One has to do settings for Hardware Flow Control for the hardware too i.e. Quectel MC60 using AT commands.*
- 3. Even used UART Interrupt Status was used, if it is then only receive the data (I don't think even this is the case)*
- 4. Sometimes GNSS data wasn't received fully- Increased time delay, worked well but not 100 %. It was 1 in 15 times that response wasn't received.*
- 5. GPRS connectivity is lost after 30 mins, hence there is no TCP connection, hence data is not sent to the server and we get error, in such case establish GPRS connection and again establish TCP connection.*
- 6. URC- Unsolicited Result Codes
Use delay of 15-20 s before executing other commands*

INTEGRATING ALL CODES-

1. LCD command issues- there was problem in the logic of the code
2. UART and TIMER ISR issue- lower the priority of TIMER ISR
3. Sending blank commands- system used to get hung up, Solution- Use Timeout condition
4. For unpredicted system hanging, use watchdog timer.
5. False detection of ultrasonic sensor
6. Improper connection of the ultrasonic sensor, i.e. wiring issues

PCB DESIGNING-

1. Chemical Etching PCB- It was done using Kicad
2. Trace width and clearance values (In kicad files design rules)
3. Single layered PCB was made
4. **for chemical etching give the mirror image as pdf file to the vendor
5. Check if footprints are proper (Use general 2.54 mm headers)
6. Use clearance of 0.5mm for etching PCBs



SOLDERING-

1. Learnt Soldering
2. Soldered the chemical etching PCB

DRILLING and CASING-

1. Drilled the acrylic casing
2. Screwed the EVB board, Quectel MC60 board, Chemical Etching PCB

TESTING FOR SAFETY-

1. Tested the device by riding it in the bike's closet on a road of potholes and speed breakers

PROPOSED CHANGES BY CLIENT-

1. *Doesn't require latitude ,longitude data- Remove GNSS setup*
2. *Hence use only M66 rather than MC60*

MIGRATION TO M66-

Quectel M66-

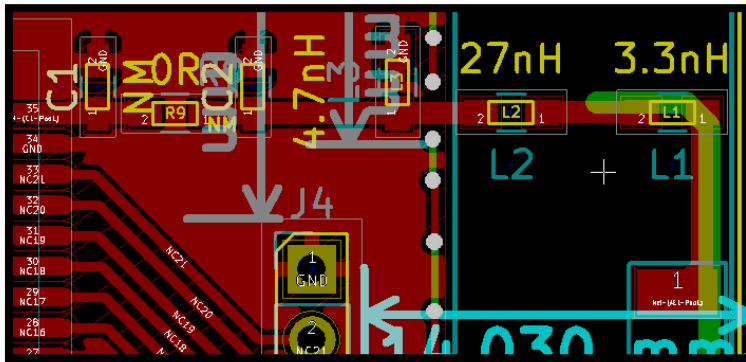
1. GSM/GPRS module
2. Supply Voltage- 3.3 to 4.6V (4 V typically)

PCB DESIGNING-

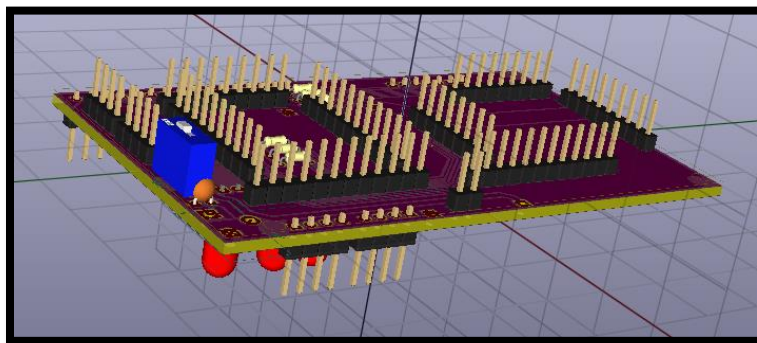
1. Modular PCB designing was done
2. M0516LDN PCB

3. Quectel M66 PCB

1. VBAT plane
2. Designing for chip antenna
 1. There shouldn't be any traces at right angle or forming right angle.
 2. Use Coplanar Wave Guide Calculator to calculate trace width of the antenna trace, impedance of the trace must be 50 ohms.
 3. Use the document of application note for ceramic chip antenna.
3. Shielding for SIMCARD pins
4. UART voltage divider
5. Remember about Power Key. It has been grounded.
6. Connect the both GND planes on upper and bottom Cu layer by numerous via joined by the traces to shield the PCB and for better grounding.



4. Combined main PCB with LCD, keypad and Ultrasonic sensor (Double Layered PCB)
5. For 5V-4V DC conversion, use buck converter LM2596 rather than LDO as variable output LDO are costlier.



ISSUES- 1. Power Issues

2. Foot print issues- made custom footprints

UART programming-

(* removed "received", it was sent by server after reception of the data)

IN APPLICATION PROGRAMMING (IAP)-

It is basically to program the controller or device when it is running or executing an application, i.e. jump to the specified vector page. In-Application Programming (IAP) capability which allows them to be programmed under firmware control of the embedded application. In-Application Programming means that the application itself can re-program the on-chip Flash ROM.

*We need to change the RO address in the linker settings to create the desired bin file for executing IAP i.e. for application code other than boot loader

Use FMC- Flash Memory Controller-
APROM with IAP

1. APROM- 64 KB (Application Program ROM)
2. LDROM- 4 KB (Loader Program ROM)
3. FLASH- 4KB
4. PAGE SIZE- 512 bytes

Issue-

1. After system reset, program in the vector page mapping doesn't get executed, but the same program gets executed. Why? Answer- NVIC Reset.
2. Solution- Do proper CPU Reset rather than NVIC Reset/ Do Software reset (Try It?)
3. ***Also disabling the all interrupts i.e primask() in the current code was the issue. Why? Answer- When executed the primask() disables all interrupts and the ResetFunc() used in the code is actually kinda CPU reset interrupt.***
4. WDT issue- If system resets in bootloader, its fine but when system is executing the application system reset jumps to bootloader. To avoid this, use FLASH storage to store application and bootloader independent data and access flag variable from FLASH.

HTTP-

1. HTTP (Hypertext Transfer Protocol) is the set of rules for transferring files (text, graphic images, sound, video, and other multimedia files) on the World Wide Web.
2. We use http to download code from the server to the MC60
3. MC60 has memory where it can store the data
4. Query the available memory using FILE AT commands
5. Refer HTTP application note for the process
6. We use UFS(User File Storage)
<https://www.tutorialspoint.com/http/index.htm>

FILE HANDLING-

1. We need to open the downloaded file in the memory i.e. UFS
2. We need to read that file
3. Reading and writing to the memory of the controller depends on the flash page size, for instance here it is 512 bytes.
4. Hence, read the file in the chunks of multiples or factors of the page size
5. A byte is read, but we have 32 bits system, hence read 4 bytes and convert them into 32 bits
6. Once file is opened, it must be closed after the use, else gives error in the next iteration.

Issues-

1. It took a lot of time to write the file data to the controller- about 8-10 mins.

2. Sometimes, full bytes not received, e.g. to receive 128 bytes, sometimes received only 35-40 bytes or sometimes 6 bytes or sometimes 120 bytes only.
3. Solution- removed sent command in the response i.e. if at => at ok was received, now at=> ok is received. (ATE0) i.e. disabled the echo.
4. This wasn't issue of the code
5. *****It was the firmware issue with Quectel MC60***
6. ***(Use AT+GMI), some version issue. (* AR01A07)***

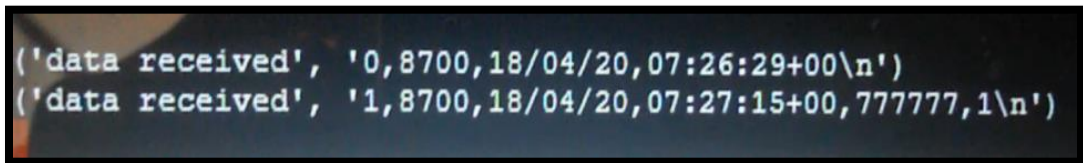
FINAL PIECE OF CODE-

It has 2 project files

1. boot-
 1. We delete all files from the UFS and then download the file from the server.
 2. We read that file to write the data to the controller memory starting at specified address
 3. We do IAP, i.e. jump to the desired address, where the downloaded file was written.
2. main-

Here we do main application programming

 1. data packet 0 is sent to server
 2. data packet 1 is sent to server



```
('data received', '0,8700,18/04/20,07:26:29+00\n')
('data received', '1,8700,18/04/20,07:27:15+00,777777,1\n')
```

ISSUES-

1. LCD improper working- Why? Answer- Same function gets called when it is being used/executed i.e function is busy (eg. lcd();). It gives garbage values often
2. ***System also gets hung due to LCD***
3. ***System gets hung due to Ultrasonic Sensor- Its wiring i.e. VCC and GND***

(**Keil Debugger, Multimeter and Datasheets were used for debugging)

TOOLS-

1. Arduino IDE
2. Kicad
3. Keil
4. NUTool pinview
5. Nu ISP
6. Gitlab (** Not used efficiently)

How to use tools/ software efficiently

1. Understand its use first, don't do donkey work
2. Use shortcut keys, wherever possible

CHANGES IN UART-

1. *GPRS connectivity is lost after 30 minutes, hence there is no TCP connection, hence data is not sent to the server and we get error, in such case establish GPRS connection and again establish TCP connection*
2. *If TCP connection already exists, it will give already connect error, its fine, not to worry*
3. *If TCP connection gives connect fail, we get send fail for AT+QISEND. Solution- close the connection and reestablish the connection.*
4. *Also do not wait for response rather than delay between enable and disable interrupts, use response and timeout condition.*
5. *Even after disable interrupt, read the buffer if there is any data available.*

FEW INSIGHTS-

1. When you are not using external crystal, do the changes in the SYS_Init() function where one configures the clock source and enables External Crystal.
2. Ensure that PLL clock source is apt i.e. either External XTAL or Internal XTAL, because Clk_SysTick() function is dependent on PLL
3. While dealing with INTERRUPTS using polling method, use a timeout condition too.
4. If you have enabled Hardware Flow Control in Hardware, you need to use Flow Control in UART too i.e. software/application code.
5. You cannot change the ISR function names
6. NVIC_EnableXYZ() is must for an interrupt to function

AS FOR NOW-

1. *If system hangs up, its mainly due to Ultrasonic Sensor.*
2. *If Sensor is fine ,its due to UART i.e. network connectivity or data sent issue.*

TIMELINE-

January

- Arduino Pro and UNO programming
- Nuvoton M0516LDN programming

February

- Integrating modular codes
- Demo to the client
- PCB designing
- Created a setup
- Shipped device/setup to client
- Manual
- Soldering
- Drilling

March

- OpenCV programming
- Tensor Flow primitive programming
- PCB designing
- IAP Programming
- Received Setup for further upgradation
- Resolving Issues

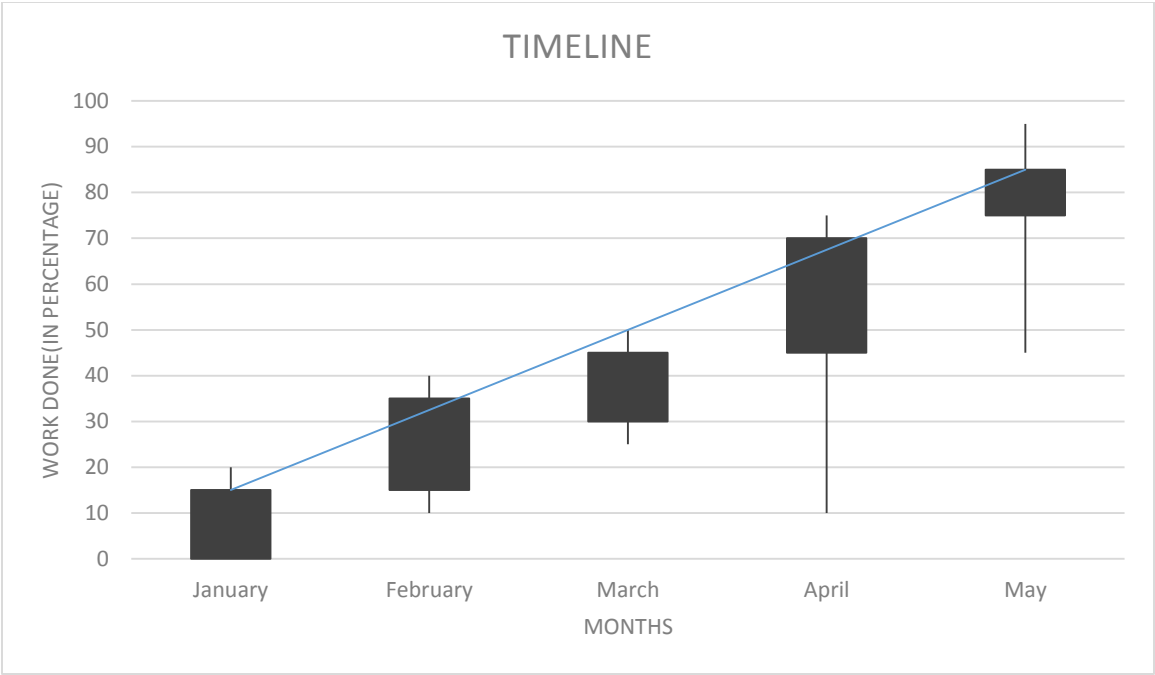
April

- IAP Programming
- File Handling
- Remote Programming
- Ultrasonics debouncing
- Shipped device/setup to client
- Documentation

May

- Documentation
- Resolving bugs/issues

TIMELINE



Change in the flow.-

Using TIMER ISR, a timeout of 20s is created-

Why?

1. When object is detected, lcd displays “Object Detected”, say for example a user is entering his phone number and at that instant object is being detected, “Object Detected” is displayed followed by “Enter your phone number”. This causes clashes between lcd function used in keypad and ultrasonic sensors timer ISR, we get gibberish or unwanted response.
2. To avoid system form hanging up
3. Halt object detection for 20s

References-

1. Quectel MC60
2. Quectel M66
3. Nuvoton M0516LDN
4. Kicad

Datasheets

1. Ceramic Dual Band Monopole Antenna
2. Nuvoton M0516LDN
3. HC-SR04 Ultrasonic Sensor
4. SMS05C- ESD Protection Diode Array
5. GP2Y0A21YK0F- IR SHARP sensor
6. 16*2 LCD
7. PCF8574- I2C GPIO Expander

Tools-

1. NuTool-PinConfigure
2. NuTool-PinView
3. NuMicro ISP Programming Tool