// CONFIG

#pragma config FOSC = HS // Oscillator Selection bits (HS oscillator)

#pragma config WDTE = OFF // Watchdog Timer Enable bit (WDT disabled)

#pragma config PWRTE = OFF // Power-up Timer Enable bit (PWRT disabled)

#pragma config BOREN = ON // Brown-out Reset Enable bit (BOR enabled)

#pragma config LVP = OFF // Low-Voltage (Single-Supply) In-Circuit Serial Programming Enable bit (RB3 is digital I/O, HV on MCLR must be used for programming)

#pragma config CPD = OFF // Data EEPROM Memory Code Protection bit (Data EEPROM code protection off)

#pragma config WRT = OFF // Flash Program Memory Write Enable bits (Write protection off; all program memory may be written to by EECON control)

#pragma config CP = OFF // Flash Program Memory Code Protection bit (Code protection off)

#include <pic16f877a.h>

#define \_XTAL\_FREQ 8000000

#include "tolling.h"

unsigned char garbage;

unsigned char free\_va;

unsigned char sig;

unsigned char reg;

unsigned char log;

unsigned char reg\_status;

unsigned char byte\_message;

unsigned char other\_byte\_message;

unsigned char log\_message;

unsigned char pay\_message;

unsigned char phone\_no[15];

unsigned char meter[5];

unsigned char ROM\_address\_tally[2];

unsigned char UR\_name[5];

unsigned char bitwise\_variable;

unsigned const short tally[] = {0, 55, 110, 165};

unsigned short enable\_ROM;

unsigned short write\_address; //RAM index number // RAM to ROM transfer authority byte

unsigned short pg;

unsigned short ii; // loop variable for loading ROM

unsigned short reg\_sms; // try initialzing plus\_bit=0;

unsigned short reg\_swipe;

unsigned short ident\_sms;

unsigned short counter;

unsigned short page;

unsigned short charge;

unsigned short he, kd, me, acquire;

signed short sf;

unsigned int index;

unsigned int word\_size;

unsigned int other\_word\_size;

unsigned int LM\_size;

unsigned int LMS\_size;

unsigned int pay\_size;

unsigned int ql;

unsigned int exx;

unsigned int reg\_count; //counter for reg array

unsigned int short\_id;

unsigned int dd;

unsigned int LM;

unsigned int PY;

unsigned int dev;

unsigned int collect;

signed int tick;

void interrupt service(void) {

if (INTCONbits.INTF == 1) { //INT RB0 trigger response

PORTAbits.RA0 = 0; //USART selection bit

INTCONbits.INTF = 0;

}

if (PIR1bits.RCIF == 1) {

PORTCbits.RC1 = 1; // yellow LED

if (PORTAbits.RA0 == 0) { //Switch to Em\_18 USART if True

reg\_count = 12;

pg = store\_in\_RAM(9,229); //Was on (7,229);

}

//=======================================================RCREG Interrupt

if (PORTAbits.RA0 == 1) { //Remain at SIM\_800L USART if selection bit is off

fix();

if (RCREG == byte\_message) {

ql++;

if (ql == word\_size) { //REG SMS route

index++;

ql = 0;

}

garbage = RCREG;

} else ql = 0;

if (RCREG == other\_byte\_message) {

dd++;

if (dd == other\_word\_size) { //Reload route

index++;

if(other\_word\_size == 5){

charge = 5;//Signifies that we are taking reload route instead.

}

if(other\_word\_size == 6){

sig = 7;//Signifies that we are taking time and date check route instead.

}

dd = 0;

}

garbage = RCREG;

} else dd = 0;

if (RCREG == log\_message) { //Samples logs condition

LM++;

if (LM == LM\_size) { //Single Log route

free\_va = 3;

//exx = 0;

//LM = 0;

}

if (LM == LMS\_size) { //Multiple Logs route

free\_va = 5;

charge = 3;

//exx = 0; uncomment if anything usual is noticed

index++;

LM = 0;

}

garbage = RCREG;

} else LM = 0;

if(free\_va == 3 && RCREG == 44){ //takes care of the free\_va = 3 route

charge = 3;

index++;

}

if (RCREG == pay\_message) {

PY++;

if (PY == pay\_size) { //REG SMS route

charge = 7;

index++;

PY = 0;

}

garbage = RCREG;

} else PY = 0;

//============================================================================End of RCREG

switch (index) {

case 2:reg\_count = 15;

if(sig!=7){ //normal route

acquire = store\_in\_RAM(0,229);

}else if(sig == 7){ //clock check route

acquire = store\_in\_RAM(7,255); //return value not needed by reg\_sms

}

break;

case 4:acquire = store\_in\_RAM(charge,229);

break;

}

}

}

PORTCbits.RC1 = 0; // yellow LED

PIR1bits.RCIF = 0;

}

void main(void) {

ADCON1 = 7;

INTCON = 208;

OPTION\_REG = 0; //previously 192;

PIE1 = 32;

//PORTA = 0;

TRISA = 18;// was on 2

TRISB = 17;//was on 1 before

TRISC = 144;

TRISD = 0;

PORTB = 0;

PORTCbits.RC0 = 1;

PORTAbits.RA0 = 1;

//PORTAbits.RA3 = 1;

my\_init\_usart(9600); // Initialize UART module at 9600 bps

\_\_delay\_ms(20);

gsm\_config(); // Wait for UART module to stabilize

my\_SPI\_Master\_Init();

Lcd\_Start();

Lcd\_Clear();

Lcd\_Set\_Cursor(1,2);

Lcd\_Print\_String(P\_NAME);

Lcd\_Set\_Cursor(2,6);

Lcd\_Print\_String("SYSTEM");

enable\_ROM = 0;

index = 0;

ql = 0;

dd = 0;

LM = PY = 0;

exx = 0;

free\_va = 0;

reg\_sms = 0;

ident\_sms = 0;

reg\_swipe = 0;

charge = 0;

sig = 0;

page = 0;

log = 0;

while(page<2){

write\_to\_25LC320\_from\_RAM(255,217,0); //255,217. Erase: 0,page\*38 page<254

page++;

}

write\_address = Read\_eeprom(255); // Reads the last occupied address of ROM, which was written at the last byte in the ROM

\_\_delay\_ms(50);

garbage = 0;

while(garbage < 2){ //gets the next empty address in external ROM

ROM\_address\_tally[garbage] = Read\_eeprom(253+garbage);

garbage++;

}

while (1) {

PORTAbits.RA3 = 1;

\_\_delay\_ms(200);

// PORTCbits.RC1 = 0; //YELLOW LED // watch this very close

//HANDLES ACCOUNT RECHARGE=================================================================================

if (charge == 1){

PORTCbits.RC0 = 1; //RED LED

PORTCbits.RC1 = 1; //YELLOW LED

PORTCbits.RC2 = 0; //GREEN LED

short\_id\_to\_hex(288);

//IF THERE IS A MATCH IN DB

for (he = 0; he <= 3; he++) {

reg = Read\_eeprom(tally[he]); //search EEPROM from parameter

if (reg ==35 || reg == 42) { //if reg = \* or #

me = 0;

for (kd = 0; kd <= 3; kd++) {

reg = Read\_eeprom(tally[he] + kd + 34);

if (reg == meter[kd]) { //Searches for the short tag\_Id in detected reg swipe or full reg

me++;

if (me == 4) { //copy full tag ID

counter = 0;

garbage = 0;

while(counter <= 7){//Check if the reload process code is accurate.

STATUSbits.IRP = 1;

FSR = 294 + counter; //RAM address

collect = INDF;

STATUSbits.IRP = 0;

if(collect == LOAD\_PIN[counter]){//Checking process code

garbage++;

}

if (garbage == 6 && counter == 6) { //if process code is accurate

ii = 0;

if (collect > 64 && collect < 71) { //convert hex letters to decimal integer

collect -= 55;

} else if (collect > 47 & collect < 58) {

collect -= 48;

}

collect \*= 16; //convert hex to integer

ii += collect;

garbage++;

}

if(garbage == 7 && counter == 7){

if (collect >64 && collect < 71) {

collect -= 55;

} else if (collect > 47 & collect < 58) {

collect -= 48;

}

ii += collect; //convert hex to int

my\_inttoStr(ii); //handles the credit dinomination to send in the feedback message

collect = Read\_eeprom(tally[he] + 50); // get the current balance.

ii += collect; //adds new balance to existing balance

Write\_eeprom(tally[he] + 50, ii);

free\_va = 0;

while(free\_va<=13){// get number that was used to recharge

STATUSbits.IRP = 1;

FSR = 273 + free\_va; //RAM address

phone\_no[free\_va] = INDF;

STATUSbits.IRP = 0;

free\_va++;

}

free\_va = 0; //nulled to avoid affecting operations where it was reused

send\_message(phone\_no,'C');// Send credit feed back

}

counter++;

}

he = 4;

kd = 4;

}

}

}

}

}

//IF THERE ISNT A MATCH

if (me<4 || garbage<6){

free\_va = 0;

while(free\_va<=13){// get number that was used to recharge

STATUSbits.IRP = 1;

FSR = 273 + free\_va; //RAM address

phone\_no[free\_va] = INDF;

STATUSbits.IRP = 0;

free\_va++;

}

free\_va = 0; //nulled to avoid affecting operations where it was reused

send\_message(phone\_no,'M');// Send mismatch feedback

}

charge = 3;

PORTCbits.RC0 = 1; //RED LED

PORTCbits.RC1 = 0; //YELLOW LED

PORTCbits.RC2 = 0; //GREEN LED

}

//INITIATION OF LOG REQUEST=============================================================================

if(log == 1){

PORTCbits.RC0 = 1; //RED LED

PORTCbits.RC1 = 1; //YELLOW LED

PORTCbits.RC2 = 0; //GREEN LED

short\_id\_to\_hex(288);

garbage = ROM\_address\_tally[0];//garbage was reused here

sig = ROM\_address\_tally[1];//sig was reused here

tick = 0;

tick = 256 \* garbage; //gets external memory tally byte

tick += sig; //stores the 16 bit address in dev

if (free\_va == 3) {

log\_request\_processing(5,0); //terminates after an AND operation with tick

free\_va = 0;

}

if(free\_va == 5){

log\_request\_processing(5,65535); //continues after an AND operation with tick

free\_va = 0;

}

PORTCbits.RC0 = 1; //RED LED

PORTCbits.RC1 = 0; //YELLOW LED

PORTCbits.RC2 = 0; //GREEN LED

log = 0;

}

//SIGNIFIES THE INITIATION OF REG SWIPE======================================================================

if (PORTAbits.RA1 == 1) { //Button to initiate REG swipe

\_\_delay\_ms(10);

if (PORTAbits.RA1 == 1) {

reg\_swipe = 1; //REG swipe initiation byte.

PORTCbits.RC0 = 1; //RED LED

PORTCbits.RC1 = 1; //YELLOW LED

PORTCbits.RC2 = 0; //GREEN LED

}

}

//HANDLES REG SWIPES AND IDENT SWIPES

if (pg == 1) {//Take action on recently received Tag ID

if (reg\_swipe == 0) { //Identification time. Might make this routine stand alone.

PORTCbits.RC0 = 1; //RED LED

PORTCbits.RC1 = 1; //YELLOW LED

PORTCbits.RC2 = 0; //GREEN LED

//LCD MESSAGED

Lcd\_Clear();

Lcd\_Set\_Cursor(1,1);

Lcd\_Print\_String(SYS\_STATUS);

Lcd\_Set\_Cursor(1,5);

Lcd\_Print\_String(PROC);

Lcd\_Set\_Cursor(2,1);

Lcd\_Print\_String(U\_NAME); //short\_id name eh 7530

Lcd\_Set\_Cursor(2,9);

Lcd\_Print\_String(BALANCE);

for (he = 0; he <= 3; he++) {

reg\_status = Read\_eeprom(tally[he]);

//reg\_status = reg;

me = 0;

for (kd = 0; kd <= 3; kd++) {

reg = Read\_eeprom(tally[he] + kd + 34);

UR\_name[kd] = reg;

STATUSbits.IRP = 1;

FSR = 278 + kd;

if (reg == INDF) {

me++;

if (me == 4) {

reg = Read\_eeprom(tally[he] + 50);

//Open gate

ident\_match\_sequence(); //normal ident routine

//Deduct toll fee

Write\_eeprom(tally[he] + 50, reg);

//Handle low balance

if (reg==20){ //when balance is N20

free\_va = 0;

ii = 0;

while(free\_va<=13){//get phone number of user

phone\_no[free\_va] = Read\_eeprom(tally[he] + free\_va + 1);

if(phone\_no[free\_va] == 120){//if x was found in database, instead of a number

free\_va = 14;//terminate while loop and stop searching for a number

ii = 5; //avoids a low balance message from getting sent, if x was found.

}

free\_va++;

}

free\_va = 0; //nulled to avoid affecting operations where it was reused

if(ii == 0){

send\_message(phone\_no, 'L');//Send message to user if balance is low

}

}

//Creates and stores log in external memory

create\_log(tally[he]);

//LCD User name and Balance message

Lcd\_Set\_Cursor(2,4);

Lcd\_Print\_String(UR\_name); //Username

Lcd\_Set\_Cursor(2,13);

Lcd\_Print\_Char('N');

Lcd\_Set\_Cursor(2,14);

Lcd\_Print\_String(meter); //Balance message

PORTCbits.RC1 = 1; //YELLOW LED

if (reg\_status == 34) { //is reg = " . Was a matching REG SMS detected

reg\_swipe\_sequence(); //" SMS. Do a reg swipe for the customer

} // Ignore if Complete REG and REG swipe

he = 3;

kd = 3;

}

}

}

}

if (me < 4) {

//Prompt the customer to do a REG swipe at least

//Do the above in an instruction manual.

PORTCbits.RC1 = 1; //YELLOW LED

PORTCbits.RC2 = 0; //GREEN LED

//LCD SUCCESS message

Lcd\_Set\_Cursor(1,5);

Lcd\_Print\_String(DEN);

garbage = 0;

while (garbage <= 5) {

PORTCbits.RC0 = ~PORTCbits.RC0; //Blinks RED LED for denied access

\_\_delay\_ms(200);

garbage++;

}

PORTCbits.RC0 = 1; //RED LED

}

PORTCbits.RC0 = 1; //RED LED

PORTCbits.RC1 = 0; //YELLOW LED

PORTCbits.RC2 = 0; //GREEN LED

}

//HANDLES REG SWIPE

if (reg\_swipe == 1) { //Analyse recieved REG swipe bytes before transferring bytes from RAM to ROM

reg\_swipe\_sequence(); //reg swipe method

//LCD REG SWIPE COMPLETE MESSAGE

Lcd\_Clear();

Lcd\_Set\_Cursor(1,4);

Lcd\_Print\_String("REG SWIPE");

Lcd\_Set\_Cursor(2,3);

Lcd\_Print\_String(SUC);

reg\_swipe = 0;

}

pg = 0;

}

//HANDLES REG SMS===========================================================================================

if (reg\_sms == 1) { //Analyse recieved REG SMS bytes before transfering bytes from RAM to ROM

PORTCbits.RC0 = 1; //RED LED

PORTCbits.RC1 = 1; //YELLOW LED

PORTCbits.RC2 = 0; //GREEN LED

free\_va = 0;

while(free\_va <= 13){//get phone number of user

STATUSbits.IRP = 1;

FSR = 273 + free\_va;

acquire = INDF;

STATUSbits.IRP = 0;

phone\_no[free\_va] = acquire;

free\_va++;

}

page = 0;

STATUSbits.IRP = 1;

FSR = 288;

while (page != 44) { //Determines the character size of the user's name

FSR++; //and make it fit into DB

page = INDF;

}

FSR++;

page = FSR - 255; //page store the word size of a user's name and the immediate comma

//255 was used to convert FSR value to the 8bit realm

short\_id\_to\_hex(255 + page);

STATUSbits.IRP = 1;

//IntToHex(short\_id, meter);

FSR = 325; //The long list of characters is

INDF = meter[0]; // to move the value of meter[] to an

FSR = 326; //easily accessible location in IRP 1

INDF = meter[1];

FSR = 327;

INDF = meter[2];

FSR = 328;

INDF = meter[3];

STATUSbits.IRP = 0;

//IF THERE IS MATCH

for (he = 0; he <= 3; he++) { //Reconciling REG SMS with REG swipe bytes previously stored in RAM

reg = Read\_eeprom(tally[he]); //search EEPROM from parameter

if (reg == 42) { //terminate the for loop if reg = \*

me = 0;

for (kd = 0; kd <= 3; kd++) {

reg = Read\_eeprom(tally[he] + kd + 34); //was on 31 before

if (reg == meter[kd]) { //Searches for the short tag\_Id in detected reg swipe

me++;

if (me == 4) { //copy full tag ID

counter = 0;

while (counter <= 12) {

if (counter == 0) { //shifting plate number

sf = 0;

while (sf <= 8) {

STATUSbits.IRP = 1;

FSR = (268 + page) - sf;

acquire = INDF; //shifting U in RAM

FSR += 52 - page; // FSR+7+300-page

INDF = acquire;

STATUSbits.IRP = 0;

sf++;

}

}

acquire = Read\_eeprom(tally[he] + counter + 28); // acquire is reused her because INDF was acting.

STATUSbits.IRP = 1;

FSR = 300 + counter; //288+((page+255)-288)+(300-(page+255))+ft; //RAM address

INDF = acquire;

STATUSbits.IRP = 0;

counter++;

}

STATUSbits.IRP = 1;

FSR = 321; //RAM address to insert ,

INDF = 44;

FSR = 312; //(page+255)+(300-(page+255))+12; //MIGHT REMOVE LATER

INDF = 44;

FSR = 299; //287+((page+255)-288)+(300-(page+255));

INDF = 44;

FSR = 272; //RAM address to insert #

INDF = 35;

STATUSbits.IRP = 0;

acquire = he; //Here i re-used the variable "acquire", while it is free, to avoid creating new variables

collect = 6;

ii = 0;

sf = 0;

he = 4;

kd = 4;

reg\_count = 50;//reduced to 50 to avoid copying 0 to credit allocation

}

}

}

}

}

//IF THERE ISNT A MATCH

if (me < 4) { //when no REG swipe was found in DB

counter = 0;

bitwise\_variable = 57 - page; //bitwise\_variable is used because counter was misbehaving

while (counter < bitwise\_variable) {//(57-page)=10+2+(300-(page+255))//Loop to load xxxx

if (counter == 0) {

sf = 0;

while (sf <= 8) { //moves the plate number to avoid getting deleted

STATUSbits.IRP = 1;

FSR = (268 + page) - sf; //(255+page+13)-sf

acquire = INDF;

FSR += 52 - page; //FSR +7+(12-((page+255)-288))

INDF = acquire;

STATUSbits.IRP = 0;

sf++;

}

}

STATUSbits.IRP = 1;

FSR = 255 + counter + page;

INDF = 120; //inserts the "x" symbol (120)

STATUSbits.IRP = 0;

counter++;

bitwise\_variable = 51 - page;

if (counter == bitwise\_variable) {//(51 - page)=18-((page+255)-288))...12+6=18

bitwise\_variable = 54 - page;

while (counter <= bitwise\_variable) {//(54 - page)=21-((page+255)-288)

STATUSbits.IRP = 1;

FSR = 274 + page + counter; //325+ft-(18-((page+255)-288))

//this was used instead of directly addressing meter[kd]

acquire = INDF;

FSR -= 19; //RAM address to insert

INDF = acquire;

STATUSbits.IRP = 0;

counter++;

}

}

}

STATUSbits.IRP = 1;

FSR = 321; //RAM address

INDF = 44;

FSR = 299;

INDF = 44;

STATUSbits.IRP = 0;

collect = 1; //nothing was detected

reload\_100\_naira(); //Watch out for this in real time

reg\_count = 51;

}

enable\_ROM = 1;

//send a success feedback message to the user

send\_message(phone\_no, 'R');//Send registration confirmation

reg\_sms = 0;

}

//HANDLES IDENT SMS===========================================================================================

if (ident\_sms == 1){

//PORTCbits.RC2 = 1;

short\_id\_to\_hex(288);

PY = 0;

while(PY <= 3){

collect = meter[PY];

STATUSbits.IRP = 1;

FSR = 278 + PY;

INDF = collect;

STATUSbits.IRP = 0;

PY++;

}

collect = 0;

PY = 0;

pg = 1; //Setting this variable makes the ident swipe routine to run at the next loop, which saves up space than having to put ident swipe in a method of its own.

ident\_sms = 0;

}

//STORES NEW REG ENTRY IN DATABASE=================================================================================

if (enable\_ROM == 1) {// when something needs to be stored in ROM

//A MATCH IN DB READ

if (collect == 6) {

ii = 0;

while (ii < reg\_count) { // Fill data buffer

STATUSbits.IRP = 1;

FSR = 272 + ii; //RAM address

if (sf == 28 && ii == 12) {

ii = 1;

sf = -1;

}

counter = INDF;//counter is reused here for a different purpose from above .

Write\_eeprom(tally[acquire] + ii + sf, counter); // Write data to address 0x80+ii

STATUSbits.IRP = 0;

if (sf == -1 && ii == 1) {

ii = 12;

sf = 28;

}

ii++;

}

PORTCbits.RC0 = 0; //RED LED

PORTCbits.RC1 = 0; //YELLOW LED

garbage = 0;

while (garbage <= 5) {

PORTCbits.RC2 = ~PORTCbits.RC2; //GREEN LED //alternate green LED

\_\_delay\_ms(200);

garbage++;

}

PORTCbits.RC0 = 1; //RED LED

PORTCbits.RC2 = 0; //GREEN LED

collect = 0;

}

//NO MATCH IN DB READ

if (collect == 1) {

ii = 0;

for (ii = 0; ii < reg\_count; ii++) {// Fill data buffer. +2 is to accommodate naira balance

STATUSbits.IRP = 1;

FSR = 272 + ii; //RAM address was on 272

acquire = INDF;

Write\_eeprom(tally[write\_address] + ii, acquire);//Write data to address tally

STATUSbits.IRP = 0;

}

write\_address++;

if (write\_address == 4) {

write\_address = 0;

}

Write\_eeprom(255, write\_address);

PORTCbits.RC0 = 0; //RED LED

PORTCbits.RC1 = 0; //YELLOW LED

garbage = 0;

while (garbage <= 5) {

PORTCbits.RC2 = ~PORTCbits.RC2; //GREEN LED //alternate green LED

\_\_delay\_ms(200);

garbage++;

}

PORTCbits.RC0 = 1; //RED LED

PORTCbits.RC1 = 0; //YELLOW LED

PORTCbits.RC2 = 0; //GREEN LED

collect = 0;

}

enable\_ROM = 0;

pg = 0;

}

//SIGNIFIES THE INITIATION OF ALL MEMORY ERASE======================================================================

if (PORTAbits.RA4 == 1) { //Button to initiate REG swipe

\_\_delay\_ms(10);

if (PORTAbits.RA4 == 1) {

//Begin memory erase.

PORTCbits.RC0 = 1; //RED LED

PORTCbits.RC1 = 1; //YELLOW LED

PORTCbits.RC2 = 0; //GREEN LED

//Handles internal ROM erasing

Write\_eeprom(255,0); //DB tally eraser

he=0;

while(he<253){//D.B eraser

Write\_eeprom(he, 255);

he++;

}

write\_address = Read\_eeprom(255); //modifies write\_address variable, after erasing

//Handles external ROM erasing

reg\_status = 0;

garbage = 0;

while(reg\_status <= ROM\_address\_tally[0] && garbage <= ROM\_address\_tally[1]){ //start erasing until it encounters the address in ROM\_address\_tally

write\_to\_25LC320\_from\_RAM(reg\_status,garbage,4);

dev = garbage;

garbage += 42;

dev += 42;

if(dev > 255){ //checks to see if there was a carry over to MSB after adding 38

dev -= 255;

garbage = dev;

reg\_status++;

}

}

if(reg\_status >= ROM\_address\_tally[0] && garbage >= ROM\_address\_tally[1]){ //if the above while loop is complete

Write\_eeprom(253,0); //External EEPROM

Write\_eeprom(254,0); //address eraser

garbage = 0;

while(garbage < 2){ //modifies ROM\_address\_tally, after erasing

ROM\_address\_tally[garbage] = Read\_eeprom(253+garbage);

garbage++;

}

}

}

garbage = reg\_status = dev = 0; //initializes all for reuse

PORTCbits.RC0 = 1; //RED LED

PORTCbits.RC1 = 0; //YELLOW LED

}

PORTAbits.RA3 = 0;

\_\_delay\_ms(200);

}

}

//====================================================================================

void ident\_match\_sequence(void) {

reg -= 5; //Deduct toll fee

PORTCbits.RC0 = 0; //RED LED

PORTCbits.RC1 = 0; //YELLOW LED

PORTCbits.RC2 = 1; //turn on green LED change back to red after the gate closes.

//LCD SUCCESS message

Lcd\_Set\_Cursor(1,5);

Lcd\_Print\_String(SUC);

//Gate controls

PORTBbits.RB1 = 1; //Start opening the gate. port b .b2 is already zero.

//free\_va = 0;

\_\_delay\_ms(1000);// the will be replaced with a voltage drop polling command that monitors back EMF, after the circuit has been built

PORTBbits.RB1 = 0; // Stop opening the gate. Meant to be PORTBbits.RB1 = 1;

free\_va = PORTBbits.RB4; //This reads from the gate proximity sensor. It waits for only 1 vehicle to pass.

while (free\_va == PORTBbits.RB4 && free\_va == 0);

free\_va = PORTBbits.RB4;

while (free\_va == PORTBbits.RB4 && free\_va == 1);

//PORTBbits.RB1 = 0;

PORTBbits.RB2 = 1; //Start closing the gate.

\_\_delay\_ms(1000);// the will be replaced with a voltage drop polling command when the circuit has been built

PORTBbits.RB2 = 0; //Stop closing the gate.

//End of gate controls

PORTCbits.RC0 = 1; //Returns LED to active RED after the gate closes.

PORTCbits.RC1 = 1; //YELLOW LED

PORTCbits.RC2 = 0; //turn off green LED change back to red after the gate closes.

}

//===================================================================================

void reg\_swipe\_sequence(void) {

//IF THERE IS A MATCH

PORTCbits.RC0 = 1; //RED LED

PORTCbits.RC1 = 1; //YELLOW LED

PORTCbits.RC2 = 0; //GREEN LED

for (he = 0; he <= 3; he++){ //Reconciling REG swipe with REG SMS bytes previously stored in RAM

reg = Read\_eeprom(tally[he]);

if (reg == 34) {//if = "

me = 0;

for (kd = 0; kd <= 3; kd++) {

reg = Read\_eeprom(tally[he] + kd + 34);

STATUSbits.IRP = 1;

FSR = 278 + kd;

if (reg == INDF) {

me++;

if (me == 4) { // store tag ID in database and reconcile with REG SMS.

STATUSbits.IRP = 1;

FSR = 284;

INDF = 35; //for #

STATUSbits.IRP = 0;

acquire = he;

collect = 6;

ii = 0; // This byte sets the start address in ROM to byte 28 after tally[]

sf = 28;

reg\_count = 13;

he = 4; //Assigned to terminate once there is a match

kd = 4;

}

}

}

}

}

//IF THERE ISNT A MATCH

if (me < 4) { //when no REG swipe was detected

sf = 0;

while (sf <= 12) { //Create details with recieved ID and store in DB

STATUSbits.IRP = 1;

FSR = 272 + sf;

acquire = INDF;

FSR += 28; //Move Tag id to the allocated section in a DB packet.

INDF = acquire;

sf++;

}

counter = 0;

while (counter <= 48) {

STATUSbits.IRP = 1;

FSR = 272 + counter; //RAM address

INDF = 120;

counter++;

if (counter == 28) {

STATUSbits.IRP = 1;

FSR = 321; //RAM address to insert ,

INDF = 44;

STATUSbits.IRP = 1;

FSR = 272; //RAM address

INDF = 42; //insert \* in RAM to indicate REG swipe

counter = 40;

}

}

collect = 1;

reg\_count = 51;

reload\_100\_naira();

}

enable\_ROM = 1;

}

//=================================================================

void short\_id\_to\_hex(unsigned int RAM\_address\_offset){ //takes the customer's short\_id received, and converts it to hex

he = 0;

while (he < 2) { // To help efficiently convert to hex

short\_id = 0;

dd = 10000;

for (pg = 0; pg < 5; pg++) { //loop to extract short tag ID from Reg SMS Might change to while loop

STATUSbits.IRP = 1;

FSR = RAM\_address\_offset + pg; //RAM address

collect = INDF;

STATUSbits.IRP = 0;

collect -= 48;

short\_id += collect\*dd; //converts character to integer

dd /= 10;

if (dd < 1) {

dd = 10000;

}

}

he++;

}

dec\_to\_hex(short\_id); //Transforms short tag\_id to hex

}

//============================================================================

void log\_request\_processing(unsigned short terminator\_1, unsigned int terminator\_2){

kd = 0;

while (kd <= 13) {// get number that requested for logs

STATUSbits.IRP = 1;

FSR = 273 + kd; //RAM address

acquire = INDF;

STATUSbits.IRP = 0;

phone\_no[kd] = acquire;

kd++;

}

while (tick > 0) {

me = 0;

for (he = 0; he < 4; he++) {

acquire = read\_from\_25LC320(garbage, (sig - 36 + he));

if (meter[he] == acquire) {//read from external memory and search for short\_id

me++;

} else he = 5; //jumps to the next for loop of he

if (me == 4) {

kd = 0;

while (kd < 42) {

acquire = read\_from\_25LC320(garbage, (sig - 42 + kd));

STATUSbits.IRP = 1;

FSR = 272 + kd; //RAM address

INDF = acquire;

STATUSbits.IRP = 0;

kd++;

}

send\_message(phone\_no, 'U'); // Send log feed back

he = terminator\_1;

tick &= terminator\_2;

//me<4 was neglected.

}

}

//prepares for the next loop

sf = 0;

sf = sig;

sig -= 42;

sf -= 42;

if (sf < 0) { //checks to see if there was a carry over to MSB after adding 38

sig = 255 + sf; //255+(-x)

garbage--;

} //its finally done transferring the 16bit address to 2x8bits address

tick -= 42;

}

}

//============================================================================

//RF TOLLING source code

#include <xc.h>

#include <pic16f877a.h>

#define \_XTAL\_FREQ 8000000

#include "tolling.h" //might remove

#define RS PORTDbits.RD2

#define EN PORTDbits.RD3

#define D4 PORTDbits.RD4

#define D5 PORTDbits.RD5

#define D6 PORTDbits.RD6

#define D7 PORTDbits.RD7

unsigned char hold;

unsigned char SPI\_input;

unsigned short X;

extern unsigned char byte\_message;

extern unsigned char other\_byte\_message;

extern unsigned char log\_message;

extern unsigned char pay\_message;

extern unsigned char DEVE\_message;

extern unsigned char sig;

extern unsigned char log;

extern unsigned short charge;

extern unsigned short reg\_sms;

extern unsigned short ident\_sms;

extern unsigned int index;

extern unsigned int word\_size;

extern unsigned int other\_word\_size;

extern unsigned int LM\_size;

extern unsigned int LMS\_size;

extern unsigned int pay\_size;

extern unsigned int DEVE\_size;

extern unsigned int ql;

extern unsigned int dd;

extern unsigned int LM;

extern unsigned int PY;

extern unsigned int dev;

extern unsigned int exx;

extern unsigned int reg\_count;

extern unsigned char ROM\_address\_tally[2];

extern char meter[5];

//===================================================================End of initialization

//====================================

void my\_init\_usart(unsigned int baud\_rate){

//This function also incoperates USART reception.

TXSTAbits.BRGH = 1; //High Speed, Low Speed select

baud\_rate/=8;// divided by 8 to fit into 16bits

baud\_rate = 62500/baud\_rate;

X = baud\_rate - 1;

SPBRG = X; //BR generator register

TXSTAbits.SYNC = 0;

RCSTAbits.SPEN = 1;

//USART recive routine

PIE1bits.RCIE = 1;

RCSTAbits.CREN = 1; // Start recieving

}

void my\_usart\_write(unsigned char byte){

// Since the USART read function is already active, deactivate it and

//reactivate after Transmission

TXSTAbits.BRGH = 1;

SPBRG = X; //BR generator register

TXSTAbits.SYNC = 0; //USART mode select bit

RCSTAbits.SPEN = 1; //Serial port enable bit

TXREG = byte;

TXSTAbits.TXEN = 1;

while(TXSTAbits.TRMT==0); //waits for each byte to get sent.

}

void my\_usart\_write\_text(unsigned char \*text){

unsigned char y;

y = 0;

while(text[y]!= '\0'){

my\_usart\_write(text[y]);

y++;

}

}

//================================================

void gsm\_config(){

//delay\_ms(2000); //remove comment when its time to program the MCU

my\_usart\_write\_text(ATE0); // AT command for Echo OFF

my\_usart\_write(13);

my\_usart\_write(10);

\_\_delay\_ms(50);

my\_usart\_write\_text(AT\_CMGF);

my\_usart\_write(13);

my\_usart\_write(10);

\_\_delay\_ms(50);

my\_usart\_write\_text(AT\_CNMI);

my\_usart\_write(13);

my\_usart\_write(10);

}

//===================================================================

//=======================================

void my\_SPI\_Master\_Init(void){

SSPCON = 32; //SSPEN\_bit = 1 Configures SCK, SDO, SDI & SS as serial port pins.

//CKP\_bit = 0 Idle state for clock is low level

//SSPMs For Master clock rate selection at Fosc/4.

SSPSTAT = 0; //CKE\_bit = 0 Transmit occurs from idle to active.

TRISC|= 16; //Sets RC3(SCK), RC4(SDI) & RC5(SDO) signal direction for Master.

//TRISA = 0; //Sets RA5(SS) signal direction for Master.

}

void my\_SPI\_write\_read(unsigned char SPI\_data){

while(SSPSTATbits.BF);

SSPBUF = SPI\_data;

\_\_delay\_ms(5);

SPI\_input = SSPBUF;

}

//======================================================================

//==================================================

void write\_to\_25LC320\_from\_RAM(unsigned char MSB\_address, unsigned char LSB\_address, unsigned char erase\_cmd){

unsigned char catch, page\_check = 0, snow;

unsigned int address\_segment;

snow = 0;

dev = 0;

dev = 256 \* MSB\_address; //dev was reused here

dev += LSB\_address; //stores the 16 bit address in dev

address\_segment = dev / 32; //76/32 = 2.375; 2+1 = 3; 32\*3 = 96-72 = 24.

address\_segment += 1;

address\_segment \*= 32;

address\_segment -= dev; //distance from end of page

while (page\_check == 0) {

page\_check = 1;

PORTAbits.RA5 = 0; //CS=0;

my\_SPI\_write\_read(6); //disable write protect

PORTAbits.RA2 = 1; //disable WP pin to enable write

PORTAbits.RA5 = 1; //CS=1;

PORTAbits.RA5 = 0; //CS=0;

my\_SPI\_write\_read(2); //write 8-bit write instruction to 23x256 SRAM IC

my\_SPI\_write\_read(MSB\_address); //Send the 16bit address

my\_SPI\_write\_read(LSB\_address);

while (snow < 43 && page\_check == 1) { //was on 38 before

STATUSbits.IRP = 1; //Bank select together with FSR

FSR = 272 + snow; //RAM address where log details are kept

catch = INDF; //RAM data

STATUSbits.IRP = 0;

if(erase\_cmd == 4){

catch = 255;//send FF to erase external RAM

}

my\_SPI\_write\_read(catch);

snow++;

if(snow == address\_segment){

page\_check = 0; //this helps to terminate the second while loop, but ensures there is a re entry through the first while loop

dev = LSB\_address;

LSB\_address += snow;

dev += snow;

if(dev > 255){ //checks to see if there was a carry over to MSB after adding 38

dev -= 255;

LSB\_address = dev;

MSB\_address++;

}

}

}

PORTAbits.RA5 = 1; //CS=1;

}

PORTAbits.RA2 = 0;//enables write protect

}

unsigned char read\_from\_25LC320(unsigned char MSB\_address, unsigned char LSB\_address){

unsigned char result;

PORTAbits.RA5 = 0;//CS=0;

my\_SPI\_write\_read(3); //write 8-bit read instruction to 23x256 SRAM IC

my\_SPI\_write\_read(MSB\_address); //Send the 16bit address

my\_SPI\_write\_read(LSB\_address);

my\_SPI\_write\_read(15);

result = SPI\_input;

PORTAbits.RA5 = 1;//CS=1;

return result;

}

//=======================================================================

//==================================================

void Write\_eeprom(unsigned short add\_ress, unsigned short da\_ta){

char INTCON\_save = INTCON, PIE1\_save = PIE1, PIE2\_save = PIE2;

EEADR = add\_ress;

EEDATA = da\_ta; //EEPGD = 0; WREN = 1; // clearing this bit grants access to data EEPROM. Also enable write cycle.

EECON1bits.EEPGD = 0;//EECON1 = 4;

EECON1bits.WREN = 1;

INTCON = 0;

PIE1 = 0;

PIE2 = 0;

EECON2 = 85;

EECON2 = 170; //WR = 1; cleared by hardware

EECON1bits.WR = 1;//EECON1 = 6;

INTCON = INTCON\_save;

PIE1 = PIE1\_save;

while(EECON1bits.WR);//\_\_delay\_ms(500);

//PIE2 = 16; // ENABLE Interupts if you were using them previously.

EECON1 = 0;

EECON1bits.WREN = 0;

//\_\_delay\_ms(100);

PIE2 = PIE2\_save;

PIR2 = 0;

}

//==================================================================

//==========================================

unsigned short Read\_eeprom(unsigned short add\_ress){

EEADR = add\_ress; //EEPGD = 0; RD = 1;

EECON1 = 1;

\_\_delay\_ms(10);

return EEDATA;

}

//===============================================================

//================================

void send\_message(unsigned char number, unsigned char message\_select){

unsigned char increase = 0;

my\_usart\_write(13);

my\_usart\_write(10);//REMOVE WHEN TESTING THE MAIN CIRCUIT

my\_usart\_write\_text(AT\_CMGF);

my\_usart\_write(13);

my\_usart\_write(10);

\_\_delay\_ms(50);// change delay to 500ms when doing main testing

my\_usart\_write\_text(AT\_CMGS);

my\_usart\_write(34);

my\_usart\_write\_text(number);;

my\_usart\_write(34);

my\_usart\_write(13);

my\_usart\_write(10);

\_\_delay\_ms(50);// change delay to 500ms when doing main testing

switch (message\_select) {

case 67: my\_usart\_write(78);

my\_usart\_write\_text(meter);

my\_usart\_write\_text(LOAD\_COMP); //C for Credited

break;

case 76: my\_usart\_write\_text(LOW\_BALANCE); //L for LOW balance

break;

case 77: my\_usart\_write\_text(MISMATCH); //M for process code mismatch

break;

case 79: my\_usart\_write\_text(CREDIT); //O

break;

case 82: my\_usart\_write\_text(REG\_COMP); //R for REG SMS complete

break;

case 85: while(increase < 42){ //U for LOG feedback

STATUSbits.IRP = 1;

FSR = 272 + increase; //RAM address

hold = INDF;

STATUSbits.IRP = 0;

my\_usart\_write(hold);

increase++;

}

break;

}

my\_usart\_write(26);

}

//============================================================

//==================================

unsigned char store\_in\_RAM(unsigned short reload\_check, unsigned short start\_add){ //From USART to RAM

unsigned char test;

unsigned short set=0; //change back to unsigned int if not working

test = RCREG;

if(test == 46){reg\_count = exx;} //used to stop copying characters after '.' is detected in the SMS

STATUSbits.IRP = 1; //Bank select together with FSR

FSR = 43 + start\_add + exx; //RAM address is modified with date and time once received

INDF = test; //RAM data

STATUSbits.IRP = 0;

if(test!=46){exx++;}

if(exx == reg\_count){

if(index==2){

if(reload\_check == 0 || reload\_check == 5){//normal route

index++;

}

if(reload\_check == 7){ //clock check route

index = 0;

exx = 0;

hold = 9;// switches control back to create log

sig = 0;

}

}

if(index==4 || PORTAbits.RA0==0){

set = 1;

PORTAbits.RA0 = 1; //Reset USART selection bit

sig = 0;

exx = 0;

index=0; //monitor continuously

if (reload\_check == 0) {

reg\_sms = 1; // To enable the reg sms process

log = 0;

ident\_sms = 0;

charge = 0;

}

if (reload\_check == 3) {

log = 1; // To enable the log request process

reg\_sms = 0;

charge = 0;

ident\_sms = 0;

}

if (reload\_check == 5) {

charge = 1; // To enable the reload process

reg\_sms = 0;

ident\_sms = 0;

log = 0;

}

if (reload\_check == 7){

ident\_sms = 1; // To enable the ident\_sms process

charge = 0;

reg\_sms = 0;

log = 0;

}

}

}

return set;

}

//=========================================================================

//==================================

void reload\_100\_naira(){

STATUSbits.IRP = 1;

FSR = 322; //RAM address

INDF = 25;//100; // Tag currency. Load tag with 100naira after REG SMS reconcilation

STATUSbits.IRP = 0;

}

//====================================================================

//=================================

void dec\_to\_hex(unsigned int decimal){

unsigned char hex;

unsigned char count;

count = 4;

while(count>0 && decimal!=0){

hex = decimal % 16;

decimal/=16;

if(hex>9 && hex<16){

hex+=55;

}else if(hex>=0 && hex<=9){

hex+=48;

}

meter[count-1] = hex;

count-=1;

}

}

//==================================================================

//====================================

void fix(){

switch (index){

case 0: byte\_message = CMT\_notify[ql];

other\_byte\_message = CCLK\_notify[dd];

log\_message = 0;

word\_size = 5;

other\_word\_size = 6;

break;

case 1: byte\_message = 34;

word\_size = 1;

break;

case 3: byte\_message = re\_gis\_ter[ql];

other\_byte\_message = RELOAD[dd];

log\_message = LOG[LM];

pay\_message = PAY[PY];

DEVE\_message = DEVE[dev];

word\_size = 4;

other\_word\_size = 5;

LM\_size = 3;

LMS\_size = 5;

pay\_size = 4;

DEVE\_size = 4;

break;

}}

//================================================================

//========================================================

void my\_inttoStr(unsigned int integer){ //The function handles five characters max

char va;

unsigned int y;

unsigned int count;

int div;

int f\_ans;

int sub;

float ans;

va=0;

while(va<=3){ //erasing value array before use.

meter[va] = 0;

va++;

}

div=10000;

sub = 0;

y=5;

while(div>integer){

div/=10;

y-=1;

}

for(count=1;count<=y;count++){

ans = integer/div;

f\_ans = (int)ans;

va = f\_ans - sub;

div/=10;

va+= 48;

meter[count-1] = va;

sub = f\_ans\*10;

}

}//===================================================

void create\_log(unsigned short tally\_select){

unsigned int naira;

unsigned char plus;

plus = 0;

//Copies relevant data from DB and stores in RAM

while(plus<11){

hold = naira = Read\_eeprom(tally\_select + 40 + plus);

STATUSbits.IRP = 1;

FSR = 284 + plus; //RAM address

INDF = hold;

STATUSbits.IRP = 0;

plus++;

}

//converts currency to a displayable form

STATUSbits.IRP = 1;

FSR = 294;

INDF = 78; //N for naira sign

STATUSbits.IRP = 0;

my\_inttoStr(naira);

naira = 0;

while(naira < 3){

hold = meter[naira];

STATUSbits.IRP = 1;

FSR = 295 + naira;

INDF = hold; //balance in characters

STATUSbits.IRP = 0;

naira++;

}

//Get date and time

hold = 0;

my\_usart\_write(13); //remove both before loading

my\_usart\_write(10);//into the real device.

my\_usart\_write\_text(AT\_CCLK); //Requests real time form GSM module

my\_usart\_write(13);

my\_usart\_write(10);

while(!hold); //Wait until the request has been granted by the GSM module

STATUSbits.IRP = 1;

FSR = 313; //RAM address

INDF = 46; //adds full stop to the end of log data in RAM.

STATUSbits.IRP = 0;

//saves data in external memory

write\_to\_25LC320\_from\_RAM(ROM\_address\_tally[0], ROM\_address\_tally[1],0);

//save external memory tallY to internal ROM

dev = ROM\_address\_tally[1]; //puts the LSB of the char variable into an int

ROM\_address\_tally[1] += 42; //adds the write tally completed by 25LC320

dev += 42;

if(dev > 255){ //checks to see if there was a carry over to MSB after adding 38

dev -= 255;

ROM\_address\_tally[1] = dev;

ROM\_address\_tally[0]++;

}

hold = 0;

while(hold < 2){ //gets the next empty address in external ROM

Write\_eeprom(253+hold, ROM\_address\_tally[hold]);

hold++;

}

}

//===========================================================

void Lcd\_SetBit(char data\_bit) { //Based on the Hex value Set the Bits of the Data Lines

if (data\_bit & 1)

D4 = 1;

else

D4 = 0;

if (data\_bit & 2)

D5 = 1;

else

D5 = 0;

if (data\_bit & 4)

D6 = 1;

else

D6 = 0;

if (data\_bit & 8)

D7 = 1;

else

D7 = 0;

}

void Lcd\_Cmd(char a) {

RS = 0;

Lcd\_SetBit(a); //Incoming Hex value

EN = 1;

\_\_delay\_ms(4);

EN = 0;

}

void Lcd\_Clear(void) {

Lcd\_Cmd(0); //Clear the LCD

Lcd\_Cmd(1); //Move the curser to first position

}

void Lcd\_Set\_Cursor(char a, char b) {

char temp, z, y;

if (a == 1) {

temp = 0x80 + b - 1; //80H is used to move the curser

z = temp >> 4; //Lower 8-bits

y = temp & 0x0F; //Upper 8-bits

Lcd\_Cmd(z); //Set Row

Lcd\_Cmd(y); //Set Column

} else if (a == 2) {

temp = 0xC0 + b - 1;

z = temp >> 4; //Lower 8-bits

y = temp & 0x0F; //Upper 8-bits

Lcd\_Cmd(z); //Set Row

Lcd\_Cmd(y); //Set Column

}

}

void Lcd\_Start(void) {

int i;

Lcd\_SetBit(0x00);

//for (i = 1065244; i <= 0; i--)

//NOP();

Lcd\_Cmd(0x03);

\_\_delay\_ms(5);

Lcd\_Cmd(0x03);

\_\_delay\_ms(11);

Lcd\_Cmd(0x03);

Lcd\_Cmd(0x02); //02H is used for Return home -> Clears the RAM and initializes the LCD

Lcd\_Cmd(0x02); //02H is used for Return home -> Clears the RAM and initializes the LCD

Lcd\_Cmd(0x08); //Select Row 1

Lcd\_Cmd(0x00); //Clear Row 1 Display

Lcd\_Cmd(0x0C); //Select Row 2

Lcd\_Cmd(0x00); //Clear Row 2 Display

Lcd\_Cmd(0x06);

}

void Lcd\_Print\_Char(unsigned char da\_ta){ //Send 8-bits through 4-bit mode

int i;

char Lower\_Nibble, Upper\_Nibble;

Lower\_Nibble = da\_ta & 0x0F;

Upper\_Nibble = da\_ta & 0xF0;

RS = 1; // => RS = 1

Lcd\_SetBit(Upper\_Nibble >> 4); //Send upper half by shifting by 4

EN = 1;

for (i = 2130483; i <= 0; i--)

//NOP();

EN = 0;

Lcd\_SetBit(Lower\_Nibble); //Send Lower half

EN = 1;

for (i = 2130483; i <= 0; i--)

//NOP();

EN = 0;

}

void Lcd\_Print\_String(unsigned char \*a){

int i;

for (i = 0; a[i] != '\0'; i++)

Lcd\_Print\_Char(a[i]); //Split the string using pointers and call the Char function

}

//==============================================================================

//===================================

TOLLING .h

void my\_init\_usart(unsigned int baud\_rate);

void my\_usart\_write(unsigned char byte);

void my\_usart\_write\_text(unsigned char \*text);

void gsm\_config(void);

void my\_SPI\_Master\_Init(void);

void my\_SPI\_write\_read(unsigned char SPI\_data);

void write\_to\_25LC320\_from\_RAM(unsigned char MSB\_address, unsigned char LSB\_address, unsigned char erase\_cmd);

unsigned char read\_from\_25LC320(unsigned char MSB\_address, unsigned char LSB\_address);

void reload\_100\_naira(void);

void short\_id\_to\_hex(unsigned int RAM\_address\_offset);

void log\_request\_processing(unsigned short terminator\_1, unsigned int terminator\_2);

void Write\_eeprom(unsigned short add\_ress, unsigned short da\_ta);

unsigned short Read\_eeprom(unsigned short add\_ress);

unsigned char store\_in\_RAM(unsigned short reload\_check, unsigned short start\_add);

void send\_message(unsigned char number, unsigned char message\_select);

void fix(void);

void ident\_match\_sequence(void);

void reg\_swipe\_sequence(void);

void dec\_to\_hex(unsigned int decimal);

void my\_inttoStr(unsigned int interger);

void create\_log(unsigned short tally\_select);

void Lcd\_SetBit(char data\_bit);

void Lcd\_Cmd(char a);

void Lcd\_Clear(void);

void Lcd\_Set\_Cursor(char a, char b);

void Lcd\_Start(void);

void Lcd\_Print\_Char(unsigned char da\_ta);

void Lcd\_Print\_String(unsigned char \*a);

//============================================================

//===============================

unsigned const char P\_NAME[] = "RFID TOLL GATE";

unsigned const char SYS\_STATUS[] = "STA:";

unsigned const char U\_NAME[] = "UR:";

unsigned const char BALANCE[] = "Bal:";

unsigned const char PROC[] = " PROCESSING..";

unsigned const char SUC[] = " SUCCESSFUL!";

unsigned const char DEN[] = " DENIED! ";

unsigned const char CMT\_notify[] = "+CMT:";

unsigned const char CCLK\_notify[] = "+CCLK:";

unsigned const char re\_gis\_ter[] = "REG,";

unsigned const char RELOAD[] = "LOAD,";

unsigned const char LOAD\_PIN[] = "194CDB";

unsigned const char LOG[] = "LOGS,";

unsigned const char PAY[] = "PAY,";

unsigned const char USERS[] = "USERS";

unsigned const char ATE0[] = "ATE0";

unsigned const char AT\_CMGF[] = "AT+CMGF=1";

unsigned const char AT\_CNMI[] = "AT+CNMI=2,2,0,0,0";

unsigned const char AT\_CMGS[] ="AT+CMGS=";

unsigned const char AT\_CCLK[] ="AT+CCLK?";

unsigned const char LOW\_BALANCE[] = "YOUR TOLL TAG BALANCE IS LOW! PURCHASE TAG CREDITS ONLINE OR THROUGH A VENDOR.";

unsigned const char REG\_COMP[] = "YOU ARE NOW REGISTERED!";

unsigned const char LOAD\_COMP[] = " has been credited to your toll tag account.";

unsigned const char MISMATCH[] = "RECHARGE FAILED! Please type in the correct characters or send a REG SMS.";

COMMANDS.

SMS REGISTRATION FORMAT: REG,NAME,Tag-no,vehicle-no.

eg: REG,Tobi,45741,KEV123ZG.

SMS PAYMENT FORMAT: PAY,Tag-no.

eg: PAY,45741.

SMS ACCOUNT RECHARGE FORMAT: LOAD,Tag-no,Recharge-code.

eg: LOAD,45741,194CDB64.

// RFID\_No = Tag-no

// User\_Name = NAME