// Serial Example

// Jason Losh

//-----------------------------------------------------------------------------

// Hardware Target

//-----------------------------------------------------------------------------

// Target Platform: EK-TM4C123GXL Evaluation Board

// Target uC: TM4C123GH6PM

// System Clock: 40 MHz

// Hardware configuration:

// Red Backlight LED:

// PB5 drives an NPN transistor that powers the red LED

// Green Backlight LED:

// PE4 drives an NPN transistor that powers the green LED

// Blue Backlight LED:

// PE5 drives an NPN transistor that powers the blue LED

// Red LED:

// PF1 drives an NPN transistor that powers the red LED

// Green LED:

// PF3 drives an NPN transistor that powers the green LED

// Pushbutton:

// SW1 pulls pin PF4 low (internal pull-up is used)

// UART Interface:

// U0TX (PA1) and U0RX (PA0) are connected to the 2nd controller

// The USB on the 2nd controller enumerates to an ICDI interface and a virtual COM port

// Configured to 115,200 baud, 8N1

//-----------------------------------------------------------------------------

// Device includes, defines, and assembler directives

//-----------------------------------------------------------------------------

**#include** <stdint.h>

**#include** <stdio.h>

**#include** <stdbool.h>

**#include** <string.h>

**#include** <math.h>

**#include** <stdlib.h>

**#include** "tm4c123gh6pm.h"

**#define** DATA\_EN (\*((**volatile** uint32\_t \*)(0x42000000 + (0x400063FC-0x40000000)\*32 + 6\*4)))

// #define RED\_LED (\*((volatile uint32\_t \*)(0x42000000 + (0x400253FC-0x40000000)\*32 + 1\*4)))

// #define GREEN\_LED (\*((volatile uint32\_t \*)(0x42000000 + (0x400253FC-0x40000000)\*32 + 3\*4)))

// #define BLUE\_LED (\*((volatile uint32\_t \*)(0x42000000 + (0x400253FC-0x40000000)\*32 + 2\*4)))

**#define** RED\_LED PWM1\_2\_CMPB\_R

**#define** BLUE\_LED PWM1\_3\_CMPA\_R

**#define** GREEN\_LED PWM1\_3\_CMPB\_R

**#define** RED\_LED\_B (\*((**volatile** uint32\_t \*)(0x42000000 + (0x400043FC-0x40000000)\*32 + 3\*4)))

**#define** GREEN\_LED\_B (\*((**volatile** uint32\_t \*)(0x42000000 + (0x400043FC-0x40000000)\*32 + 2\*4)))

**#define** PUSH\_BUTTON (\*((**volatile** uint32\_t \*)(0x42000000 + (0x400253FC-0x40000000)\*32 + 4\*4)))

**#define** MAX\_CHARS 80

**#define** MAX\_FIELDS 10

**char** str[MAX\_CHARS+1];

**#define** MAX\_MESSAGE 30

**#define** MAX\_DATA 50

**#define** MAX\_RETRY 5

uint8\_t SRC\_ADDR= 1;

**#define** BROADCAST\_ADDR 255

**#define** PACKET\_SIZE 7

**#define** RX\_BUFFER\_SIZE 50

**#define** TX\_BUFFER\_SIZE 50

**#define** SET\_size 1

**#define** ACK\_size 1

**#define** RGB\_size 3

**#define** Poll\_Res\_size 1

**#define** Data\_Report\_size 1

**#define** Set\_Addr\_size 1

// Defining the command values

**#define** CmdSET 0

**#define** CmdRGB 0x48

**#define** CmdACK 0x70

**#define** CmdPOLLREQ 0x78

**#define** CmdPOLLRES 0x79

**#define** CmdSETADDR 0x7A

**#define** CmdRESET 0x7F

**#define** CmdDATAREQ 0x20

**#define** CmdDATAREP 0x21

**#define** CmdPULSE 0x02

**#define** CmdSQUARE 0x03

**#define** CmdSAWT 0x04

**#define** CmdTRI 0x05

// Defining the table for data

uint8\_t DST\_ADDRESS[MAX\_MESSAGE];

uint8\_t SEQ\_ID[MAX\_MESSAGE];

uint8\_t COMMAND[MAX\_MESSAGE];

uint8\_t CHANNEL[MAX\_MESSAGE];

uint8\_t SIZE[MAX\_MESSAGE];

uint8\_t CHECKSUM[MAX\_MESSAGE];

uint8\_t DATA[MAX\_MESSAGE][MAX\_DATA];

bool valid[MAX\_MESSAGE];

bool ackreq[MAX\_MESSAGE];

uint8\_t retranscount[MAX\_MESSAGE];

uint16\_t timeout[MAX\_MESSAGE];

uint8\_t field=0;

uint8\_t type[MAX\_FIELDS];

uint8\_t pos[MAX\_FIELDS];

**char**\* str2;

**char** str4[100];

**char** str5[50];

uint8\_t i,j,addr, channel, value[50];

uint8\_t sequence=0;

uint8\_t valid1=0, val=0;

uint16\_t rx\_test\_data;

bool ackon;

bool random;

bool cs=false;

bool csEnable;

bool end=false;

uint8\_t currentIndex, txPhase=0, rxPhase=0;

uint16\_t rxData[RX\_BUFFER\_SIZE],d=0;

uint8\_t txData[TX\_BUFFER\_SIZE];

bool inProgress=false;

uint16\_t led\_timeout;

bool blink=false;

bool time=false;

uint8\_t check\_data=0;

uint8\_t check\_data\_1=0,n;

uint16\_t N=0,M, T0=100, T=500;

**char**\* uart[50];

uint16\_t old\_rxPhase,dead\_rxtimeout,dead\_limit=300,old\_txPhase,dead\_txtimeout;

uint8\_t rx\_dest\_addr,rx\_src\_addr,rx\_seq\_id,rx\_cmd,rx\_channel,rx\_size,rx\_data\_value[20],rx\_checksum;

uint16\_t pulse\_timeout;

uint8\_t pulse\_amplitude=0;

uint16\_t high,low;

uint8\_t square\_high\_amp=0,square\_low\_amp=0;

uint16\_t pulse\_timeout\_on,pulse\_timeout\_off,cy,cycles;

uint8\_t saw\_high\_amp,saw\_low\_amp,delta;

uint16\_t dwell;

uint8\_t tri\_low\_amp, tri\_high\_amp ,delta1;

bool pul=false,squ=false,saw=false,tri=false;

uint8\_t **ran**();

**void** **processmsg**();

**void** **square**();

**void** **pulse**();

**void** **sawtooth**();

**void** **triangle**();

**void** **command**();

**void** **transmitdata**();

**void** **receivedata**();

**void** **deadlocks**();

//-----------------------------------------------------------------------------

// Subroutines

//-----------------------------------------------------------------------------

// Initialize Hardware

**void** **initHw**()

{

// Configure HW to work with 16 MHz XTAL, PLL enabled, system clock of 40 MHz

SYSCTL\_RCC\_R = SYSCTL\_RCC\_XTAL\_16MHZ | SYSCTL\_RCC\_OSCSRC\_MAIN | SYSCTL\_RCC\_USESYSDIV | (4 << SYSCTL\_RCC\_SYSDIV\_S)| SYSCTL\_RCC\_USEPWMDIV | SYSCTL\_RCC\_PWMDIV\_2;;

// Set GPIO ports to use APB (not needed since default configuration -- for clarity)

// Note UART on port A must use APB

SYSCTL\_GPIOHBCTL\_R = 0;

// Enable GPIO port A,F and C peripherals

SYSCTL\_RCGC2\_R = SYSCTL\_RCGC2\_GPIOA | SYSCTL\_RCGC2\_GPIOF |SYSCTL\_RCGC2\_GPIOC;

// Configure LED and pushbutton pins

GPIO\_PORTA\_DIR\_R = 0x0C; // bits 1 and 3 are outputs, other pins are inputs

GPIO\_PORTA\_DR2R\_R = 0x0C; // set drive strength to 2mA (not needed since default configuration -- for clarity)

GPIO\_PORTF\_DEN\_R = 0x1E; // enable LEDs and pushbuttons(red, blue, green)

GPIO\_PORTA\_DEN\_R = 0x0C; // enable LEDs and pushbuttons

GPIO\_PORTF\_PUR\_R = 0x10; // enable internal pull-up for push button

GPIO\_PORTF\_DIR\_R = 0x0E; // bits 1,2 and 3 are outputs, other pins are inputs

GPIO\_PORTF\_DR2R\_R = 0x0E; // set drive strength to 2mA (not needed since default configuration -- for clarity)

GPIO\_PORTF\_AFSEL\_R |= 0x0E;

GPIO\_PORTF\_PCTL\_R = GPIO\_PCTL\_PF1\_M1PWM5 | GPIO\_PCTL\_PF2\_M1PWM6|GPIO\_PCTL\_PF3\_M1PWM7;

// Configure GPIO pins for UART0

SYSCTL\_RCGCUART\_R |= SYSCTL\_RCGCUART\_R0; // turn-on UART0, leave other uarts in same status

GPIO\_PORTA\_DEN\_R |= 3; // default, added for clarity

GPIO\_PORTA\_AFSEL\_R |= 3; // default, added for clarity

GPIO\_PORTA\_PCTL\_R = GPIO\_PCTL\_PA1\_U0TX | GPIO\_PCTL\_PA0\_U0RX;

// Configure UART0 to 115200 baud, 8N1 format (must be 3 clocks from clock enable and config writes)

UART0\_CTL\_R = 0; // turn-off UART0 to allow safe programming

UART0\_CC\_R = UART\_CC\_CS\_SYSCLK; // use system clock (40 MHz)

UART0\_IBRD\_R = 21; // r = 40 MHz / (Nx115.2kHz), set floor(r)=21, where N=16

UART0\_FBRD\_R = 45; // round(fract(r)\*64)=45

UART0\_LCRH\_R = UART\_LCRH\_WLEN\_8 | UART\_LCRH\_FEN; // configure for 8N1 w/ 16-level FIFO

UART0\_CTL\_R = UART\_CTL\_TXE | UART\_CTL\_RXE | UART\_CTL\_UARTEN; // enable TX, RX, and module

// Configure UART1 pins(GPIO)

SYSCTL\_RCGCUART\_R |= SYSCTL\_RCGCUART\_R1; // turn-on UART1, leave other UART's in same status

GPIO\_PORTC\_DIR\_R |= 0x60;

GPIO\_PORTC\_DEN\_R |= 0x70; //

GPIO\_PORTC\_AFSEL\_R |= 0x30; //

GPIO\_PORTC\_PCTL\_R |= GPIO\_PCTL\_PC5\_U1TX | GPIO\_PCTL\_PC4\_U1RX;

// Configure UART1 to 38400 baud, 8N1 format (must be 3 clocks from clock enable and config writes)

UART1\_CTL\_R = 0; // turn-off UART0 to allow safe programming

UART1\_CC\_R = UART\_CC\_CS\_SYSCLK; // use system clock (40 MHz)

UART1\_IBRD\_R = 65; // r = 40 MHz / (Nx38.4kHz), set floor(r)=21, where N=16

UART1\_FBRD\_R = 7; // round(fract(r)\*64)=45

UART1\_LCRH\_R = UART\_LCRH\_WLEN\_8 | UART\_LCRH\_FEN; // configure for 8N1 w/ 16-level FIFO

UART1\_CTL\_R = UART\_CTL\_TXE | UART\_CTL\_RXE | UART\_CTL\_UARTEN; // enable TX, RX, and module

// Configure Timer 1 as the time base

SYSCTL\_RCGCTIMER\_R |= SYSCTL\_RCGCTIMER\_R1; // turn-on timer

TIMER1\_CTL\_R &= ~TIMER\_CTL\_TAEN; // turn-off timer before reconfiguring

TIMER1\_CFG\_R = TIMER\_CFG\_32\_BIT\_TIMER; // configure as 32-bit timer (A+B)

TIMER1\_TAMR\_R = TIMER\_TAMR\_TAMR\_PERIOD; // configure for periodic mode (count down)

TIMER1\_TAILR\_R = 0x9C40; // set load value to 40e3 for 1 kHz interrupt rate 0x9C40;

TIMER1\_IMR\_R = TIMER\_IMR\_TATOIM; // turn-on interrupts

NVIC\_EN0\_R |= 1 << (INT\_TIMER1A-16); // turn-on interrupt 37 (TIMER1A)

TIMER1\_CTL\_R |= TIMER\_CTL\_TAEN; // turn-on timer

//Configure PWM

SYSCTL\_RCGC0\_R |= SYSCTL\_RCGC0\_PWM0;

SYSCTL\_RCGCPWM\_R |= SYSCTL\_RCGCPWM\_R1;

**\_\_asm**(" NOP");

**\_\_asm**(" NOP");

**\_\_asm**(" NOP");

SYSCTL\_SRPWM\_R = SYSCTL\_SRPWM\_R1;

SYSCTL\_SRPWM\_R = 0;

PWM1\_2\_CTL\_R = 0;

PWM1\_3\_CTL\_R = 0;

PWM1\_2\_GENB\_R = PWM\_1\_GENB\_ACTCMPBD\_ZERO | PWM\_1\_GENB\_ACTLOAD\_ONE;

PWM1\_3\_GENA\_R = PWM\_1\_GENA\_ACTCMPAD\_ZERO | PWM\_1\_GENA\_ACTLOAD\_ONE;

PWM1\_3\_GENB\_R = PWM\_1\_GENB\_ACTCMPBD\_ZERO | PWM\_1\_GENB\_ACTLOAD\_ONE;

PWM1\_2\_LOAD\_R = 256;

PWM1\_3\_LOAD\_R = 256;

PWM1\_INVERT\_R = PWM\_INVERT\_PWM5INV | PWM\_INVERT\_PWM6INV | PWM\_INVERT\_PWM7INV;

PWM1\_2\_CMPB\_R = 0;

PWM1\_3\_CMPB\_R = 0;

PWM1\_3\_CMPA\_R = 0;

PWM1\_2\_CTL\_R = PWM\_1\_CTL\_ENABLE;

PWM1\_3\_CTL\_R = PWM\_1\_CTL\_ENABLE;

PWM1\_ENABLE\_R = PWM\_ENABLE\_PWM5EN | PWM\_ENABLE\_PWM6EN | PWM\_ENABLE\_PWM7EN;

}

// Blocking function that writes a serial character when the UART buffer is not full

**void** **putcUart0**(**char** c)

{

**while** (UART0\_FR\_R & UART\_FR\_TXFF);

UART0\_DR\_R = c;

}

// Blocking function that writes a string when the UART buffer is not full

**void** **putsUart0**(**char**\* str)

{

uint8\_t i;

**for** (i = 0; i < **strlen**(str); i++)

putcUart0(str[i]);

}

// Blocking function that returns with serial data once the buffer is not empty

**char** **getcUart0**()

{

**while** (UART0\_FR\_R & UART\_FR\_RXFE);

**return** UART0\_DR\_R & 0xFF;

}

// Approximate busy waiting (in units of microseconds), given a 40 MHz system clock

**void** **waitMicrosecond**(uint32\_t us)

{

**\_\_asm**("WMS\_LOOP0: MOV R1, #6"); // 1

**\_\_asm**("WMS\_LOOP1: SUB R1, #1"); // 6

**\_\_asm**(" CBZ R1, WMS\_DONE1"); // 5+1\*3

**\_\_asm**(" NOP"); // 5

**\_\_asm**(" NOP"); // 5

**\_\_asm**(" B WMS\_LOOP1"); // 5\*2 (speculative, so P=1)

**\_\_asm**("WMS\_DONE1: SUB R0, #1"); // 1

**\_\_asm**(" CBZ R0, WMS\_DONE0"); // 1

**\_\_asm**(" NOP"); // 1

**\_\_asm**(" B WMS\_LOOP0"); // 1\*2 (speculative, so P=1)

**\_\_asm**("WMS\_DONE0:"); // ---

// 40 clocks/us + error

}

// Function that returns string that is compatible with backspace, space and carriage return

**void** **getsUart0**()

{

uint8\_t count=0;

**char** c;

start: c= getcUart0() ;

**if** (c==8)

{

**if**(count>0)

{

count--;

**goto** start ;

}

**goto** start ;

}

**else** **if** (c==13)

{

x: str[count++]=0;

**return**;

}

**if**(c>=' ')

{

str[count++]=c;

}

**else**

{

**goto** start;

}

**if**(count> MAX\_CHARS)

{

**goto** x;

}

**else**

{

**goto** start ;

}

}

//Function that defines the differentiates delimiters and alphanumeric characters and defines fields, position and type of string

**void** **isstring**(**char**\* str)

{

field=0;

uint8\_t i, flag=0;

uint8\_t l=**strlen**(str);

**for**(i=0; i<l; i++)

{

// Condition for Alphanumeric characters

**if**((str[i]>='0' && str[i]<= '9') || (str[i]>='A' && str[i]<='Z')||(str[i]>='a' && str[i]<='z'))

{

**if**((str[i]>='a' && str[i]<='z')) //defining case insensitivity

{

str[i]=str[i]-32;

}

flag++;

**if**(flag==1)

{

field++; // Setting Field value

pos[field-1]=i; // Setting position value

// Defining the type of string

**if** ((str[i]>='A' && str[i]<='Z')||(str[i]>='a' && str[i]<='z'))

type[field-1]='a';

**else**

type[field-1]='n';

}

}

// Condition for delimiters and converting them to zero

**else**

{

flag=0;

str[i]=0;

}

}

}

// Function for checking the entered data

bool **iscommand**(**char** strcmd[], uint8\_t min\_args)

{

**if**((**strcmp**(&str[pos[0]], strcmd)==0) && (field>= min\_args))

**return** true;

**else**

**return** false;

}

// Function to get the entered string

**char**\* **getstring**(uint8\_t field)

{

**if** (type[field]=='a')

**return** &str[pos[field]];

**else**

valid1 =1;

**return** 0;

}

// Function to get the entered number

int16\_t **getnumber**(uint8\_t field)

{

uint8\_t x;

**if** (type[field]=='n')

{

x= **atoi**(&str[pos[field]]);

**return** x;

}

**else**

valid1 =1;

**return** valid1;

}

// Function to get the entered number(16 bits)

int16\_t **getnumber2**(uint8\_t field)

{

uint16\_t x;

**if** (type[field]=='n')

{

x= **atoi**(&str[pos[field]]);

**return** x;

}

**else**

valid1 =1;

**return** valid1;

}

//Function to store the data in the table

**void** **sendPacket**(uint8\_t destAddr, uint8\_t command, uint8\_t channel, uint8\_t size, uint8\_t data[])

{

**if**(valid[i]== false)

{

uint8\_t sum[50] ,sum1[50],l;

DST\_ADDRESS[i]=destAddr;

COMMAND[i]=command;

ackreq[i]=ackon ;

**if**(ackreq[i]==1 && COMMAND[i]!=0x70 )

COMMAND[i]|= 0x80;

CHANNEL[i]=channel;

SIZE[i]=size;

**for**(l=0;l<size;l++)

{

DATA[i][l]= value[i+l];

}

SEQ\_ID[i]=sequence;

sequence++;

sum[i]=0;

**for**( l=0;l<size; l++)

{

sum[i]+= DATA[i][l];

}

sum1[i]= SRC\_ADDR+DST\_ADDRESS[i]+COMMAND[i]+CHANNEL[i]+SIZE[i]+sum[i]+SEQ\_ID[i];

CHECKSUM[i]=~sum1[i];

retranscount[i]=0;

timeout[i]=0;

valid[i]= true;

}

**else**

putsUart0("ERROR \n");

}

//Timer function that runs every one milisecond

**void** **Timer1Isr**()

{

uint8\_t x;

**if**(!inProgress) //Routine for searching the valid message

{

**for**(x=0;x<MAX\_MESSAGE;x++)

{

**if**((valid[x]==true)&&(timeout[x]==0))

{

inProgress= true;

currentIndex=x;

txPhase=0;

old\_txPhase=0;

}

}

}

transmitdata(); // Function to transmit the data

**if**(time) // Timeout Routine

{

**if**(timeout[currentIndex]>0)

{

timeout[currentIndex]--;

}

**if**(timeout[currentIndex]==0)

{

**sprintf**(str5,"Transmitting Msg %u, Attempt %u \r\n",SEQ\_ID[currentIndex],retranscount[currentIndex]);

putsUart0(str5);

}

}

**if**(!(UART1\_FR\_R & UART\_FR\_BUSY))

DATA\_EN=0; //Turn off transmit pin after transmitting

receivedata(); // Function to receive the data

**if**(blink) //Routine for led timeout

{

**if**(led\_timeout>0)

{

led\_timeout--;

}

**if**(led\_timeout==0)

{

GREEN\_LED\_B=0;

RED\_LED\_B=0;

}

}

**if**(squ)

square(); //calling square function

**if**(pul)

pulse(); // calling pulse function

**if**(saw)

sawtooth(); // calling sawtooth function

**if**(tri)

triangle(); // calling triangle function

deadlocks(); //calling deadlock function

TIMER1\_ICR\_R = TIMER\_ICR\_TATOCINT; // clear interrupt flag

}

//Function for transmitting data.................................................................................................................................................................

**void** **transmitdata**()

{

uint8\_t x,l;

**if**(inProgress)

{

DATA\_EN=1; //Turning on transmit data pin

**if** (csEnable) // Checking for carrier sense

{

**if**(!(UART1\_FR\_R & UART\_FR\_RXFE))

{

putsUart0("Carrier is busy \n\r");

}

}

**switch**(txPhase)

{

**case** 0: //case 0: Transmitting destination address

**if**((UART1\_FR\_R & UART\_FR\_TXFE)^(csEnable==false|rxPhase==0))

{

UART1\_LCRH\_R &= ~UART\_LCRH\_EPS ;

UART1\_DR\_R = DST\_ADDRESS[currentIndex];

txData[txPhase]=DST\_ADDRESS[currentIndex];

txPhase++;

time=false;

**break**;

}

**case** 1: //case 1: Transmitting source address

**if**(!(UART1\_FR\_R & UART\_FR\_BUSY))

{

UART1\_LCRH\_R |= UART\_LCRH\_SPS|UART\_LCRH\_PEN|UART\_LCRH\_EPS ;

UART1\_DR\_R = SRC\_ADDR;

txData[txPhase]= SRC\_ADDR;

txPhase++;

time=false;

}

**case** 2: //case 2: Transmitting sequence ID address

UART1\_DR\_R = SEQ\_ID[currentIndex];

txData[txPhase]=SEQ\_ID[currentIndex];

txPhase++;

time=false;

**case** 3: //case 3: Transmitting command address

UART1\_DR\_R = COMMAND[currentIndex];

txData[txPhase]=COMMAND[currentIndex];

txPhase++;

time=false;

**case** 4: //case 4: Transmitting channel address

UART1\_DR\_R = CHANNEL[currentIndex];

txData[txPhase]=CHANNEL[currentIndex];

txPhase++;

time=false;

**case** 5: //case 5: Transmitting size address

UART1\_DR\_R = SIZE[currentIndex];

txData[txPhase]=SIZE[currentIndex];

txPhase++;

time=false;

**case** 6: //case 6: Transmitting data address

**for**( l=0;l<SIZE[currentIndex];l++)

{

UART1\_DR\_R = DATA[currentIndex][l];

txData[txPhase]= DATA[currentIndex][l];

time=false;

txPhase++;

}

**default**: //case 7: Transmitting checksum address

UART1\_DR\_R = CHECKSUM[currentIndex];

txData[txPhase]=CHECKSUM[currentIndex];

time=false;

txPhase++;

end=true;

**break**;

}

**if**(SIZE[currentIndex]==0)

x=PACKET\_SIZE+1;

**else**

x=PACKET\_SIZE;

**if**(end)

{

**if**(txPhase==x + SIZE[currentIndex]) //End of message

{

inProgress=false;

RED\_LED\_B=1;

led\_timeout=500;

blink=true;

txPhase=0;

**sprintf**(str5,"Queuing Msg %u \r\n",SEQ\_ID[currentIndex]);

putsUart0(str5);

**if**(!ackon || COMMAND[currentIndex]==0x70)

{

valid[currentIndex]=false;

time=false;

**if**(COMMAND[currentIndex]==0x70)

{

**sprintf**(str5,"Acknowledgement sent \r\n");

putsUart0(str5);

}

**else**

{

**sprintf**(str5,"Transmitting Msg %u, Attempt %u \r\n",SEQ\_ID[currentIndex],retranscount[currentIndex]);

putsUart0(str5);

}

}

**else**

{

retranscount[currentIndex]++;

**if**(retranscount[currentIndex]> MAX\_RETRY)

{

valid[currentIndex]=false;

RED\_LED\_B=1;

blink=false;

time=false;

**sprintf**(str5,"Error sending message %u \r\n",SEQ\_ID[currentIndex]);

putsUart0(str5);

}

**else**

{

**if**(random) //random retransmission

{

timeout[currentIndex]= T0 + ran()\*T;

time=true;

}

**else**

{

timeout[currentIndex]= T0 + **pow**(2,N)\*T;

N++;

**if**(N>4)

N=0;

time=true;

}

}

}

}

end=false;

}

}

}

//...Function to receive the data..................................................................................................................................................................

**void** **receivedata**()

{

UART1\_LCRH\_R |=UART\_LCRH\_SPS|UART\_LCRH\_PEN|UART\_LCRH\_EPS ;

**if**((!(UART1\_FR\_R & UART\_FR\_RXFE))&&(txPhase==0))

{

d = UART1\_DR\_R ;

**if**((d & 0x200)&& txPhase==0)

{

rxPhase=0;

old\_rxPhase=0;

rxData[rxPhase]=d & 0xFF;

**if**((rxData[rxPhase]==SRC\_ADDR)|| (rxData[rxPhase]== BROADCAST\_ADDR))

rxPhase++;

}

**else** **if**(rxPhase!=0)

{

rxData[rxPhase++]=d;

}

**if**(rxPhase==PACKET\_SIZE +rxData[5] )

{

processmsg();

rxPhase=0;

}

}

}

// Function to avoid deadlocks

**void** **deadlocks**()

{

**if**(old\_rxPhase != rxPhase)

dead\_rxtimeout++;

**if**(dead\_rxtimeout>dead\_limit)

{

rxPhase=0;

putsUart0("RECEIVE DEADLOCK\n\r");

dead\_rxtimeout=0;

}

**if**(old\_txPhase != txPhase)

dead\_txtimeout++;

**if**(dead\_txtimeout>dead\_limit)

{

txPhase=0;

putsUart0("TRANSMIT DEADLOCK\n\r");

dead\_rxtimeout=0;

}

}

//Random sequence function

uint8\_t **ran**()

{

uint8\_t k;

uint8\_t ran[20]={1,3,4,6,4,2,5,4,1,9,1,5,9,3,1,2,2,3,1,6};

k=ran[i];

i++;

**if**(i>19)

i=0;

**return** k;

}

//..Function to process the received message packet..................................................................................................................................................

**void** **processmsg**()

{

check\_data=0;

check\_data\_1=0;

uint8\_t x;

rx\_dest\_addr=rxData[0];

rx\_src\_addr=rxData[1];

rx\_seq\_id=rxData[2];

rx\_cmd=rxData[3];

rx\_channel=rxData[4];

rx\_size=rxData[5];

**for**(x=0;x<rx\_size;x++)

rx\_data\_value[x]=rxData[6+x];

rx\_checksum=rxData[6+rx\_size];

**for**(n=0; n<PACKET\_SIZE-1+rx\_size;n++)

{

check\_data+= rxData[n];

}

check\_data\_1=~check\_data;

**if**(check\_data\_1==rx\_checksum) // checking for checksum error

{

GREEN\_LED\_B=1;

led\_timeout=500;

blink=true;

**if**((rx\_cmd & 0x7F)==0x00) //set routine

{

**if**(rx\_channel==0x01)

RED\_LED=rx\_data\_value[0];

**if**(rx\_channel==0x02)

BLUE\_LED=rx\_data\_value[0];

**if**(rx\_channel==0x03)

GREEN\_LED=rx\_data\_value[0];

**if**(!ackon)

{

**sprintf**(str4,"DestAddr=%u,SrcAddr=%u,SeqID=%u,Command=%u,channel=%u,size=%u,data=%u,checksum=%u \n\r",rx\_dest\_addr,rx\_src\_addr,rx\_seq\_id,rx\_cmd,rx\_channel,rx\_size,rx\_data\_value[0],rx\_checksum);

putsUart0(str4);

}

}

**if**((rx\_cmd & 0x7F) ==0x48) //RGB receive routine

{

**if**(rx\_channel==0x04)

{

RED\_LED=rx\_data\_value[0];

BLUE\_LED=rx\_data\_value[1];

GREEN\_LED= rx\_data\_value[2];

}

}

**if**((rx\_cmd & 0x80) ==0x80) // Sending ACK

{

addr=rx\_src\_addr;

channel=rx\_channel;

value[i]=rx\_seq\_id;

**sprintf**(str4,"ACK:Received Address=%u, channel=%u, data=%u, SEQ\_ID=%u \n\r",rx\_src\_addr,rx\_channel,rx\_data\_value[0],rx\_seq\_id);

putsUart0(str4);

sendPacket(addr,CmdACK,channel,ACK\_size,&value[i]);

}

**if**((rx\_cmd & 0x7F) ==0x70) // Receiving ACK

{

**if**(SEQ\_ID[currentIndex]==rx\_data\_value[0])

{

**sprintf**(str4,"Acknowledgment received with SEQ\_ID= %u \n\r",rx\_data\_value[0]);

putsUart0(str4);

valid[currentIndex]=false;

time=false;

}

}

**if** ((rx\_cmd & 0x7F) ==0x78) //poll request

{

addr=rx\_src\_addr;

channel=rx\_channel;

value[i]=SRC\_ADDR;

valid[currentIndex]=false;

sendPacket(addr,CmdPOLLRES,channel,Poll\_Res\_size,&value[i]);

putsUart0("Poll response sent \n\r");

}

**if** ((rx\_cmd & 0x7F) ==0x79) // poll response

{

**sprintf**(str4,"POLL RESPONSE RECEIVED FROM ADDRESS %u \n\r",rx\_data\_value[0]);

putsUart0(str4);

}

**if** ((rx\_cmd & 0x7F) == 0x20) //data request

{

addr=rx\_src\_addr;

channel=rx\_channel;

**if**(rx\_channel==40)

value[i]=PUSH\_BUTTON;

**if**(rx\_channel==1)

value[i]=RED\_LED;

**if**(rx\_channel==2)

value[i]=BLUE\_LED;

**if**(rx\_channel==3)

value[i]=GREEN\_LED;

valid[currentIndex]=false;

sendPacket(addr,CmdDATAREP,channel,Data\_Report\_size,&value[i]);

}

**if** ((rx\_cmd & 0x7F) == 0x21) //data report

{

**sprintf**(str4,"DATA REPORT: CHANNEL=%u, VALUE=%u \n\r",rx\_channel,rx\_data\_value[0]);

putsUart0(str4);

}

**if**((rx\_cmd & 0x7F) == 0x7F) //reset

{

NVIC\_APINT\_R= NVIC\_APINT\_VECTKEY | NVIC\_APINT\_SYSRESETREQ;

}

**if**((rx\_cmd & 0x7F) == 0x7A) //set source address

{

SRC\_ADDR=rx\_data\_value[0];

**sprintf**(str4,"NEW ADDRESS= %u \n\r",SRC\_ADDR);

putsUart0(str4);

}

**if**((rx\_cmd & 0x7F) == 0x02) //pulse signal

{

**if**(rx\_channel==10)

{

pulse\_amplitude=rx\_data\_value[0];

high=rx\_data\_value[1]\*256;

pulse\_timeout= (high|rx\_data\_value[2])\*10;

RED\_LED=pulse\_amplitude;

pul=true;

}

}

**if**((rx\_cmd & 0x7F) == 0x03) //square signal

{

**if**(rx\_channel==11)

{

square\_high\_amp=rx\_data\_value[0];

square\_low\_amp=rx\_data\_value[1];

high=rx\_data\_value[2]\*256;

pulse\_timeout\_on=(high|rx\_data\_value[3]);

GREEN\_LED=square\_high\_amp;

low=rx\_data\_value[4]\*256;

pulse\_timeout\_off=(high|rx\_data\_value[5]);

cy=rx\_data\_value[6]\*256;

cycles=(cy|rx\_data\_value[7]);

squ=true;

}

}

**if**((rx\_cmd & 0x7F) == 0x04) //sawtooth signal

{

**if**(rx\_channel==12)

{

saw\_low\_amp=rx\_data\_value[0];

saw\_high\_amp=rx\_data\_value[1];

delta=rx\_data\_value[2];

high=rx\_data\_value[3]\*256;

dwell=(high|rx\_data\_value[4])\*10;

cy=rx\_data\_value[5]\*256;

cycles=(cy|rx\_data\_value[6]);

saw=true;

BLUE\_LED=saw\_low\_amp;

}

}

**if**((rx\_cmd & 0x7F) == 0x05) //triangle signal

{

**if**(rx\_channel==13)

{

tri\_low\_amp=rx\_data\_value[0];

tri\_high\_amp=rx\_data\_value[1];

delta=rx\_data\_value[2];

delta1=rx\_data\_value[3];

high=rx\_data\_value[4]\*256;

dwell=(high|rx\_data\_value[5])\*10;

cy=rx\_data\_value[6]\*256;

cycles=(cy|rx\_data\_value[7]);

tri=true;

RED\_LED=tri\_low\_amp;

}

}

}

**else**

{

GREEN\_LED\_B=1;

blink=false;

}

}

// Function to process the received the triangle command

**void** **triangle**()

{

**if**(cycles!=0)

{

**if**(RED\_LED==tri\_low\_amp)

{

**if**(dwell>0)

dwell--;

**if**(dwell==0)

{

tri\_low\_amp=tri\_low\_amp+delta;

dwell=(high|rx\_data\_value[5])\*10;

RED\_LED=tri\_low\_amp;

}

}

**if**(RED\_LED>=tri\_high\_amp-20)

{

// tri\_high\_amp=tri\_high\_amp-delta1;

RED\_LED=tri\_high\_amp;

**if**(dwell>0)

dwell--;

**if**(dwell==0)

{

tri\_high\_amp=tri\_high\_amp-delta1;

dwell=(high|rx\_data\_value[5])\*10;

RED\_LED=tri\_high\_amp;

}

**if**(RED\_LED<=rx\_data\_value[0]+20)

{

RED\_LED=rx\_data\_value[0];

tri\_low\_amp=rx\_data\_value[0];

tri\_high\_amp=rx\_data\_value[1];

**if**(cycles>0)

cycles--;

}

}

}

**else**

{

RED\_LED=0;

tri=false;

}

}

// Function to process the received the sawtooth command

**void** **sawtooth**()

{

**if**(cycles!=0)

{

**if**(BLUE\_LED==saw\_low\_amp)

{

**if**(dwell>0)

dwell--;

**if**(dwell==0)

{

saw\_low\_amp=saw\_low\_amp+delta;

dwell=(high|rx\_data\_value[4])\*10;

BLUE\_LED=saw\_low\_amp;

}

}

**if**(BLUE\_LED>=saw\_high\_amp-10)

{

saw\_low\_amp=rx\_data\_value[0];

BLUE\_LED=saw\_low\_amp;

**if**(cycles>0)

{

cycles--;

}

}

}

**else**

{

BLUE\_LED=0;

saw=false;

}

}

// Function to process the received the pulse command

**void** **pulse**()

{

**if**(pulse\_timeout>0)

pulse\_timeout--;

**if**(pulse\_timeout==0)

{

RED\_LED=0;

pul=false;

}

}

// Function to process the received the square command

**void** **square**()

{

**if**(cycles!=0)

{

**if**(GREEN\_LED==square\_high\_amp)

{

**if**(pulse\_timeout\_on>0)

pulse\_timeout\_on--;

**if**(pulse\_timeout\_on==0)

{

GREEN\_LED=square\_low\_amp;

pulse\_timeout\_on=(high|rx\_data\_value[3]);

}

}

**if**(GREEN\_LED==square\_low\_amp)

{

**if**(pulse\_timeout\_off>0)

pulse\_timeout\_off--;

**if**(pulse\_timeout\_off==0)

{

GREEN\_LED=square\_high\_amp;

pulse\_timeout\_off=(high|rx\_data\_value[5]);

**if**(cycles>0)

cycles--;

}

}

}

**else**

{

GREEN\_LED=0;

squ=false;

}

}

// Function to find and process the entered command

**void** **command**()

{

**if** (iscommand("SET", 3))

{

str2= getstring(0);

addr= getnumber(1);

channel= getnumber(2);

value[i]= getnumber(3);

sendPacket(addr,CmdSET,channel,SET\_size,&value[i]);

val=1;

}

**if**(iscommand("RGB",4))

{

addr= getnumber(1);

channel= getnumber(2);

value[i]= getnumber(3);

value[i+1]= getnumber(4);

value[i+2]= getnumber(5);

sendPacket(addr,CmdRGB,channel,RGB\_size,&value[i]);

val=1;

}

**if** (iscommand ("CS",1))

{

**if**(**strcmp**(getstring(1),"ON")==0)

{

csEnable=true;

putsUart0("carrier sense detection is enabled \n\r");

}

**else** **if**(**strcmp**(getstring(1),"OFF")==0)

{

csEnable=false;

putsUart0("carrier sense detection is disabled \n\r");

}

**else**

putsUart0("Error in 'cs' command \n\r");

val=1;

}

**if** (iscommand ("ACK",1))

{

**if**(**strcmp**(getstring(1),"ON")==0)

{

ackon=true;

putsUart0("data acknowledgment is enabled \n\r");

}

**else** **if**(**strcmp**(getstring(1),"OFF")==0)

{

ackon=false;

putsUart0("data acknowledgment is disabled \n\r");

}

**else**

putsUart0("Error in 'ack' command \n\r");

val=1;

}

**if** (iscommand ("RANDOM",1))

{

**if**(**strcmp**(getstring(1),"ON")==0)

{

random=true;

putsUart0("random retransmissions is enabled \n\r");

}

**else** **if**(**strcmp**(getstring(1),"OFF")==0)

{

random=false;

putsUart0("random retransmissions is disabled \n\r");

}

**else**

putsUart0("Error in 'random' command \n\r");

val=1;

}

**if** (iscommand("POLL", 0))

{

addr=BROADCAST\_ADDR;

channel=0;

value[i]=0;

sendPacket(addr,CmdPOLLREQ,channel,0,&value[i]);

val=1;

}

**if** (iscommand("RESET", 1))

{

addr=getnumber(1);

channel=0;

value[i]=0;

sendPacket(addr,CmdRESET,channel,0,&value[i]);

val=1;

}

**if** (iscommand("GET", 1))

{

addr=getnumber(1);

channel=getnumber(2);

value[i]=0;

sendPacket(addr,CmdDATAREQ,channel,0,&value[i]);

val=1;

}

**if** (iscommand("SA", 1))

{

addr=getnumber(1);

channel=0;

value[i]=getnumber(2);

sendPacket(addr,CmdSETADDR,channel,1,&value[i]);

val=1;

}

**if** (iscommand("PULSE", 3))

{

addr=getnumber(1);

channel=getnumber(2);

value[i]= getnumber(3);

value[i+1]=getnumber2(4)/0xFF;

value[i+2]=getnumber2(4)&0xFF;

sendPacket(addr,CmdPULSE,channel,3,&value[i]);

val=1;

}

**if** (iscommand("SQUARE", 7))

{

addr=getnumber(1);

channel=getnumber(2);

value[i]= getnumber(3);

value[i+1]=getnumber(4);

value[i+2]=getnumber2(5)/0xFF;

value[i+3]=getnumber2(5)&0xFF;

value[i+4]=getnumber2(6)/0xFF;

value[i+5]=getnumber2(6)&0xFF;

value[i+6]=getnumber2(7)/0xFF;

value[i+7]=getnumber2(7)&0xFF;

sendPacket(addr,CmdSQUARE,channel,8,&value[i]);

val=1;

}

**if** (iscommand("SAWTOOTH", 6))

{

addr=getnumber(1);

channel=getnumber(2);

value[i]= getnumber(3);

value[i+1]=getnumber(4);

value[i+2]=getnumber(5);

value[i+3]=getnumber2(6)/0xFF;

value[i+4]=getnumber2(6)&0xFF;

value[i+5]=getnumber2(7)/0xFF;

value[i+6]=getnumber2(7)&0xFF;

sendPacket(addr,CmdSAWT,channel,7,&value[i]);

val=1;

}

**if** (iscommand("TRIANGLE", 7))

{

addr=getnumber(1);

channel=getnumber(2);

value[i]= getnumber(3);

value[i+1]=getnumber(4);

value[i+2]=getnumber(5);

value[i+3]=getnumber(6);

value[i+4]=getnumber2(7)/0xFF;

value[i+5]=getnumber2(7)&0xFF;

value[i+6]=getnumber2(8)/0xFF;

value[i+7]=getnumber2(8)&0xFF;

sendPacket(addr,CmdTRI,channel,8,&value[i]);

val=1;

}

}

//-- ---------------------------------------------------------------------------

// Main

//-----------------------------------------------------------------------------

**int** **main**(**void**)

{

// Initialize hardware

initHw();

// Blinking on board green LED on for 500 ms

GREEN\_LED\_B = 1;

waitMicrosecond(500000);

GREEN\_LED\_B = 0;

waitMicrosecond(500000);

putsUart0("READY \n\r"); // Display status

**sprintf**(str4,"DEVICE ADDRESS: %u \n\r",SRC\_ADDR);

putsUart0(str4);

**for**(i=0;i<MAX\_MESSAGE;i++)

{

valid[i]= false;

val=0;

valid1=0;

// Receive. display and configure string

getsUart0();

putsUart0(str);

putcUart0('\n');

putcUart0('\r');

isstring(str);

command();

**if**(valid1==1)

{

putsUart0("Error \n\r");

}

**if**(!val)

{

putsUart0("Error \n\r");

}

}

**while**(1);

}