LAB4

Generated by Doxygen 1.8.13

Contents

Chapter 1

File Index

1.1 File List

Here is a list of all files with brief descriptions:

Chapter 2

File Documentation

2.1 src/main.cpp File Reference

```
#include <Arduino.h>
```

Functions

- void setup ()
- void loop ()

Variables

- int cont =0
- char letra
- int flag =0
- int pulso =0
- int entrada =2
- int rojo =3
- int verde =4
- int amarillo =5
- int contLed =6

2.1.1 Function Documentation

2.1.1.1 loop()

```
void loop ( )
```

Definition at line 43 of file main.cpp.

```
{

if(Serial.available()>0)
{  //Serial.available para ver si hay un dato
   letra=Serial.read(); //a letra le asigna lo que entra del pulsador

  switch(letra)
  { //case para alumbrar los leds y asignar la bandera segun el caso

    case 's': //start

    digitalWrite(amarillo, HIGH);
    digitalWrite(verde,LOW);
    digitalWrite(rojo,LOW);
    flag='s';
    break;

    case 'r': //ready

    digitalWrite(amarillo,LOW);
    digitalWrite(verde,HIGH);
    digitalWrite(rojo,LOW);
    flag='r'; //Bandera para pasar al otro codigo
    break;

    case 'S': //STOP

    digitalWrite(amarillo,LOW);
    digitalWrite(verde,LOW);
    digitalWrite(rojo,HIGH);
    flag='S';
    entrada=0;
    Serial.write(cont); //Envia el valor de los pulsos contados
    break;
   }
}

// este else entra si la bandera esta en r para contar
else
{

  if(flag=='r')
  {
    pulso=digitalRead(entrada);

    if(pulso==1)
    {
      cont++;
    }

    //este es antirrebote de internet
    while(digitalRead(entrada)==1)
    {
      digitalWrite(contLed,HIGH); //para led contLed
    }
    while(digitalRead(entrada)==0)
    {
      digitalWrite(contLed,LOW); //para led contLed
    }
  }
}
}
```

2.1.1.2 setup()

```
void setup ( )
```

Definition at line 20 of file main.cpp.

```
{

  //Velocidad
    Serial.begin(9600);
  //ENTRADA CONTADOR
  pinMode(entrada,INPUT);

  //LED CONTADOR
  pinMode(contLed,OUTPUT);

  // PIN ROJO
  pinMode(rojo,OUTPUT);

  //PIN VERDE
  pinMode(verde,OUTPUT);

  //PIN AMARILLO
  pinMode(amarillo,OUTPUT);

}
```

2.1.2 Variable Documentation

2.1.2.1 amarillo

`int amarillo =5`

Definition at line 14 of file main.cpp.

2.1.2.2 cont

`int cont =0`

Definition at line 5 of file main.cpp.

2.1.2.3 contLed

`int contLed =6`

Definition at line 16 of file main.cpp.

2.1.2.4 entrada

`int entrada =2`

Definition at line 11 of file main.cpp.

2.1.2.5 flag

```
int flag =0
```

Definition at line 7 of file main.cpp.

2.1.2.6 letra

```
char letra
```

Definition at line 6 of file main.cpp.

2.1.2.7 pulso

```
int pulso =0
```

Definition at line 8 of file main.cpp.

2.1.2.8 rojo

```
int rojo =3
```

Definition at line 12 of file main.cpp.

2.1.2.9 verde

```
int verde =4
```

Definition at line 13 of file main.cpp.

2.2 test/arduinoprueba.cpp File Reference

```
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <errno.h>
#include <termios.h>
#include <unistd.h>
#include <iostream>
#include <sys/file.h>
```

Macros

- #define BAUD_RATE B9600

Functions

- void config_tty (const char ∗tty_port, struct termios ∗tty, unsigned int baud, int ∗serial_port)
- int main ()

Variables

- const char ∗ SERIAL_PORT = "/dev/ttyS0"

2.2.1 Macro Definition Documentation

2.2.1.1 BAUD_RATE

```
#define BAUD_RATE B9600
```

Definition at line 22 of file arduinoprueba.cpp.

2.2.2 Function Documentation

2.2.2.1 config_tty()

```
void config_tty (
             const char * tty_port,
             struct termios * tty,
             unsigned int baud,
             int * serial_port )
```

Definition at line 31 of file arduinoprueba.cpp.

```
{

    *serial_port = open(tty_port, O_RDWR);

    // Check for errors
    if (*serial_port < 0) {
        printf("Error %i from open: %s\n", errno, strerror(errno));
    }


    // Create new termios struct, we call it 'tty' for convention
    // No need for "= {0}" at the end as we'll immediately write the existing
    // config to this struct
    //struct termios tty;//no needed here as is received in function argument

    // Read in existing settings, and handle any error
    // NOTE: This is important! POSIX states that the struct passed to tcsetattr()
    // must have been initialized with a call to tcgetattr() overwise behaviour
    // is undefined
    if(tcgetattr(*serial_port, tty) != 0) {
        printf("Error %i from tcgetattr: %s\n", errno, strerror(errno));
    }

    tty->c_cflag &= ~PARENB; // Clear parity bit, disabling parity (most common)
    //tty->c_cflag |= PARENB;  // Set parity bit, enabling parity

    tty->c_cflag &= ~CSTOPB; // Clear stop field, only one stop bit used in communication (most common)
    tty->c_cflag |= CSTOPB;  // Set stop field, two stop bits used in communication


    tty->c_cflag &= ~CSIZE; // Clear all the size bits, then use one of the statements below
    tty->c_cflag |= CS5; // 5 bits
    tty->c_cflag |= CS6; // 6 bits
    tty->c_cflag |= CS7; // 7 bits
    tty->c_cflag |= CS8; // 8 bits (most common)


    tty->c_cflag &= ~CRTSCTS; // Disable RTS/CTS hardware flow control (most common)
    //tty->c_cflag |= CRTSCTS;  // Enable RTS/CTS hardware flow control

    tty->c_cflag |= CREAD | CLOCAL; // Turn on READ & ignore ctrl lines (CLOCAL = 1)

    //In canonical mode, input is processed when a new line character is received.
    tty->c_lflag &= ~ICANON; // non-canonical

    //If this bit is set, sent characters will be echoed back.
    tty->c_lflag &= ~ECHO; // Disable echo
    tty->c_lflag &= ~ECHOE; // Disable erasure
    tty->c_lflag &= ~ECHONL; // Disable new-line echo

    tty->c_lflag &= ~ISIG; // Disable interpretation of INTR, QUIT and SUSP

    tty->c_iflag &= ~(IXON | IXOFF | IXANY); // Turn off s/w flow ctrl

    tty->c_iflag &= ~(IGNBRK|BRKINT|PARMRK|ISTRIP|INLCR|IGNCR|ICRNL); // Disable any special handling of received bytes

    tty->c_oflag &= ~OPOST; // Prevent special interpretation of output bytes (e.g. newline chars)
    tty->c_oflag &= ~ONLCR; // Prevent conversion of newline to carriage return/line feed
    // tty->c_oflag &= ~OXTABS; // Prevent conversion of tabs to spaces (NOT PRESENT IN LINUX)
    // tty->c_oflag &= ~ONOEOT; // Prevent removal of C-d chars (0x004) in output (NOT PRESENT IN LINUX)


  /*VMIN = 0, VTIME = 0: No blocking, return immediately with what is available
    VMIN > 0, VTIME = 0: This will make read() always wait for bytes (exactly how many is determined by VMIN), so read() could block indefinitely.
    VMIN = 0, VTIME > 0: This is a blocking read of any number of chars with a maximum timeout (given by VTIME). read() will block until either any amount of data is available, or the timeout occurs. This happens to be my favourite mode (and the one I use the most).
    VMIN > 0, VTIME > 0: Block until either VMIN characters have been received, or VTIME after first character has elapsed. Note that the timeout for VTIME does not begin until the first character is received.
    type of VMIN and VTIME: cc_t (1B)*/
    tty->c_cc[VTIME] = 0;
    tty->c_cc[VMIN] = 1; // wait one byte

    //B0,  B50,  B75,  B110,  B134,  B150,  B200, B300, B600, B1200, B1800, B2400, B4800, B9600, B19200, B38400, B57600, B115200, B230400, B460800
    // Set in/out baud rate to be 9600
    cfsetispeed(tty, baud);
    cfsetospeed(tty, baud);
    //cfsetspeed(tty, B9600); //set both input and output

    //cfsetispeed(tty, 104560); //Specifying a custom baud rate when using GNU C
    //cfsetospeed(tty, 104560);

    /*Other option for custom baud rate*/
    /*
113         // #include <termios.h> This must be removed!
114         // Otherwise we'll get "redefinition of  struct termios " errors
115         #include <sys/ioctl.h> // Used for TCGETS2/TCSETS2, which is required for custom baud rates
116         struct termios2 tty;
117         // Read in the terminal settings using ioctl instead
118         // of tcsetattr (tcsetattr only works with termios, not termios2)
119         ioctl(fd, TCGETS2, tty);
120         // Set everything but baud rate as usual
121         // ...
122         // ...
123         // Set custom baud rate
124         tty->c_cflag &= ~CBAUD;
125         tty->c_cflag |= CBAUDEX;
126         // On the internet there is also talk of using the "BOTHER" macro here:
127         // tty->c_cflag |= BOTHER;
128         // I never had any luck with it, so omitting in favour of using
129         // CBAUDEX
130         tty->c_ispeed = 123456; // What a custom baud rate!
131         tty->c_ospeed = 123456;
132         // Write terminal settings to file descriptor
133         ioctl(*serial_port, TCSETS2, tty);
134     */
135 
136     // Save tty settings, also checking for error
137     if (tcsetattr(*serial_port, TCSANOW, tty) != 0) {
138         printf("Error %i from tcsetattr: %s\n", errno, strerror(errno));
139     }
140     /*********/
141     /*WRITING*/
142     /*********/
143     //unsigned char msg[] = { 'H', 'e', 'l', 'l', 'o', '\r' };
144     //write(*serial_port, msg, sizeof(msg));
145 
146     /*********/
147     /*READING*/
148     /*********/
149       // Allocate memory for read buffer, set size according to your needs
150     //char read_buf [256];
151 
152     // Normally you wouldn't do this memset() call, but since we will just receive
153     // ASCII data for this example, we'll set everything to 0 so we can
154     // call printf() easily.
155     //memset(&read_buf, '\0', sizeof(read_buf));
156 
157     // Read bytes. The behaviour of read() (e.g. does it block?,
158     // how long does it block for?) depends on the configuration
159     // settings above, specifically VMIN and VTIME
160     //int num_bytes = read(*serial_port, &read_buf, sizeof(read_buf));
161 
162     // n is the number of bytes read. n may be 0 if no bytes were received, and can also be -1 to signal an
   error.
163     //if (num_bytes < 0) {
164     //  printf("Error reading: %s", strerror(errno));
165     //  return 1;
166     //}
167 
168     // Here we assume we received ASCII data, but you might be sending raw bytes (in that case, don't try
   and
169     // print it to the screen like this!)
170     //printf("Read %i bytes. Received message: %s", num_bytes, read_buf);
171 
172     //close(serial_port);
173 
174 }
```

2.2.2.2 main()

```
int main ( )
```

Definition at line 182 of file arduinoprueba.cpp.

```
182            {
183 
184     // ... get file descriptor here
185     // Acquire non-blocking exclusive lock
186    //  if(flock(fd, LOCK_EX | LOCK_NB) == -1) {
  //   throw std::runtime_error("Serial port with file descriptor " +
    //     std::to_string(fd) + " is already locked by another process.");

struct termios tty;
int serial_port;
int read_buf;
int num_bytes;




config_tty("/dev/ttyS0",&tty,B9600,&serial_port);
sleep(3);
write(serial_port,"s",sizeof(char));
sleep(3);
write(serial_port,"r",sizeof(char));
sleep(3);
//write(serial_port,"S",sizeof(char));

num_bytes=read(serial_port,&read_buf,sizeof(read_buf));
sleep(3);
write(serial_port,"S",sizeof(char));
cout<<serial_port;
close(serial_port);
}
```

2.2.3 Variable Documentation

2.2.3.1 SERIAL_PORT

```
const char* SERIAL_PORT = "/dev/ttyS0"
```

Definition at line 26 of file arduinoprueba.cpp.