**Embedded Computational Systems**

**Monitoring and Alarm System**

Group nº3

Eduardo Costa nº84037

Eduardo Melo nº84038

João Sebastião nº84087

1-Data Structures

typedef struct \_register

{

    uint8\_t h, m, s;

    uint8\_t temp;

    uint8\_t lum;

} Register;

Structure used to store data registers that are transferred from the board to the computer.

typedef struct \_ring\_buffer

{

    uint8\_t nr;

    uint8\_t ri;

    uint8\_t wi;

    bool empty;

    Register buffer [NRBUF];

} RingBuffer;

Structure used to store the registers transferred from the board to the computer. Note that *nr* is the number of valid registers (i.e. number of registers that contain information from the board, regardless of already being read or not).

2-Main Blocks

The code that runs on *eCos* environment is divided into 4 threads, represented by the 4 functions below:

Note: We represent only the loop that reads the messages for convenience, since the actual functions have considerable size and would bloat the document.

void pr\_procedure(cyg\_addrword\_t data)

{

…

while(1)

     {

m = cyg\_mbox\_get(pr\_box2\_h);

}

…

}

The processing thread represented above by the function pr\_procedure() is responsible for the processing of registers either when they are received through transference or when the user asks for processing through the *pr* command. This thread communicates through mailboxes with the receiving and the user interface threads that will be represented bellow.

void tx\_procedure(cyg\_addrword\_t data)

{

…

while(1)

     {

        m = cyg\_mbox\_get(tx\_box1\_h); //get message from ui/pr thread

        n = 1;

        buf[0] = m;

        err = cyg\_io\_write(serH, buf, &n); //sent message to PIC

     }

…

}

The transmitting thread represented above by the function tx\_procedure() is responsible for sending the messages coming from the user interface and processing threads to the PIC board using the *cyg\_io* library. This thread is pretty simple since it doesn’t even need to know what command it is sending so it simply redirects the messages from the mailbox to the serial channel.

void rx\_procedure(cyg\_addrword\_t data)

{

    …

    while(1)

    {

        n = 1;

        err = cyg\_io\_read(serH, buf, &n);//get message from board

        cmd\_count = 1;

/\*loop to read the command until EOM is reached\*/

        while(buf[cmd\_count-1] != EOM)

        {

            n = 1;

            err = cyg\_io\_read(serH, buf+cmd\_count, &n);

            cmd\_count++;

        }

        cmd = buf[1];

…

{

…

{

The receiving thread represented above by the rx\_procedure() function is used to receive the command replies from the PIC board and redirect them to the thread that issued the request. The reading loop is more complicated in this thread since it needs to know the command that in receives in order to process it properly (if for example it receives a transference response, the thread needs to save the registers in the local ring buffer). This thread communicates with the user interface and processing threads through mailboxes.

int main(void)

{

  cmd\_ini(0, NULL);

//init global variables, threads, synchronization variables

…

monitor();

//free used memory

…

{

The user interface thread represented above by the main() function is responsible for reading commands from the command line and calling the respective functions that then act on the command and send the appropriate messages to the other threads. It is also responsible for representing the results of the commands in the command line. This thread communicates with the transmitting and processing threads through mailboxes.