Bizkaiko Foru Aldundia
Diputación Foral de Bizkaia
Berrikuntza eta Ekonomi Sustapen Saila
Departamento de Innovación y Promoción Económica

Programa Ekinberri 2006

FLEXEO

FLEXible remote sEnsing and Operation architecture

E2.2B-Diseño del prototipo

FlexContainer

Tecnológico Fundación Deusto
Teknologikoa Deustu Fundazioa

ESI European Software Institute
tecnalia

HISTORIAL DE CAMBIOS

Versión	Descripción	Autor	Fecha	Comentarios
V0.1	Definición de la plantilla	Igor Santos	15/09/2006	
V0.3	Cambios menores	Iñaki Vázquez	13/11/2006	

TABLA DE CONTENIDOS

Historial de cambios 2

Tabla de contenidos 3

1 Introducción.....4

2 Algortimo 5

3 Formato de mensajes.....12

1 INTRODUCCIÓN

El FlexContainer controla que el nivel de líquido de un tanque esté entre dos límites, es decir, entre el límite superior e inferior. En caso de que se detecte alguna anomalía en el nivel enviará un mensaje de alarma a la base. Resumiendo si el nivel del agua está por debajo del nivel bajo, transmitirá a la base el mensaje de alarma de que está por debajo del nivel inferior. Si por el contrario el nivel del agua está por encima del nivel superior se envía un mensaje de alarma a la base con el identificativo de nivel alto sobrepasado.

Para ello tenemos dos variables booleanas (una para cada nivel) en las que mientras su estado es el correspondiente al funcionamiento normal rellenaremos con un cero dichas variables y en caso contrario enviaremos un `1´.

Para chequear dichos niveles miraremos el valor de las entradas analógicas 2 y 3 (ADC2 Y ADC3) cada 3 segundos. Una vez chequeados los valores de estas entradas asignamos los valores adecuados a las dos variables. Junto a los valores de los niveles también enviamos la temperatura, la luz y el nivel de batería. Por último enviaremos el paquete a la base.

2 ALGORTIMO

```
#define EMPEZAR 0X55

#define BORRAR

#include "appFeatures.h"

//DataType* pointerToUsefulData;

//pointerToUsefulData = (DataType*) MsgPtr->data;

//TOS_BCAST_ADDR ES PARA ENVIAR A TODOS LOS NODOS

includes sensorboard;

includes AM;

module FlexContainerM {

  provides {

    interface StdControl;

  }

  uses {

      interface Leds;

      interface MhopSend as Send;

      interface RouteControl;

      interface ADC as ADCBATT;

            interface StdControl as BattControl;


    interface StdControl as TempControl;

    interface ADC as Temperature;

    interface StdControl as PhotoControl;

    interface ADC as Light;
```

```
     interface ADC as ADC2;
    interface ADC as ADC3;
    interface ADCControl;
     interface HPLUART;
     interface Receive;
    interface Timer;
  }
}
implementation {
  #define MSG_LEN  29
   TOS_Msg msg_buf;
   TOS_MsgPtr msg_ptr;
   norace bool sending_packet;
   norace uint8_t state;
   int j,cont1=0,cont2=0;
   sample mensaje;
static void start()
    {
       call BattControl.start();
       call TempControl.start();
       call PhotoControl.start();
    }
task void stop()
    {
       call StdControl.stop();
    }
task void battstop()
```

```
    {
       call BattControl.stop();
    }
task void tempstop()
    {
       call TempControl.stop();
    }
task void photostop()
    {
       call PhotoControl.stop();
    }





task void send_datos(){
    uint16_t  len;
       sample *data;
    uint8_t i;
       if(sending_packet) return;
     atomic sending_packet=TRUE;


    data = (sample*)call Send.getBuffer(msg_ptr, &len);
       for (i=0; i<= sizeof(sample)-1; i++)
              ((uint8_t*) data)[i] = ((uint8_t*)&mensaje)[i];
    data->xMeshHeader.board_id =0xA2;
    data->xMeshHeader.packet_id =0x01;
    data->xMeshHeader.node_id = TOS_LOCAL_ADDRESS;
```

```
    data->xMeshHeader.parent    = call RouteControl.getParent();


    if(call                          Send.send(TOS_BCAST_ADDR,MODE_ANY2ANY,msg_ptr,
sizeof(sample))!=SUCCESS)

                    atomic sending_packet = FALSE;

    return;

  }

command result_t StdControl.init() {

       atomic {

    msg_ptr = &msg_buf;

    }

    call HPLUART.init();

    call BattControl.init();

    call Leds.init();

    call TempControl.init();

    call PhotoControl.init();

    call ADCControl.bindPort(TOS_ADC2_PORT, TOSH_ACTUAL_ADC2_PORT);

    call ADCControl.bindPort(TOS_ADC3_PORT, TOSH_ACTUAL_ADC3_PORT);

    call ADCControl.bindPort(TOS_ADC4_PORT, TOSH_ACTUAL_ADC4_PORT);

    call ADCControl.bindPort(TOS_ADC5_PORT, TOSH_ACTUAL_ADC5_PORT);

    call ADCControl.bindPort(TOS_ADC6_PORT, TOSH_ACTUAL_ADC6_PORT);

    call ADCControl.init();

       return SUCCESS;

  }

 command result_t StdControl.start(){

       call HPLUART.init();

         call ADCBATT.getData();

         call Timer.start(TIMER_REPEAT,3000);
```

```
    return SUCCESS;
  }


  command result_t StdControl.stop() {
    call BattControl.stop();
    call TempControl.stop();
    call PhotoControl.stop();
    call HPLUART.stop();
    return SUCCESS;
  }
event result_t Timer.fired() {
            call ADCBATT.getData();
        return SUCCESS;
  }
async event result_t ADCBATT.dataReady(uint16_t data){
  atomic mensaje.xData.datos.bateria=data;
      post battstop();
      call Temperature.getData();
      return SUCCESS;
  }
      async event result_t Temperature.dataReady(uint16_t data) {
       atomic mensaje.xData.datos.temperatura=data;
       post tempstop();
       call Light.getData();
       return SUCCESS;
  }
  async event result_t Light.dataReady(uint16_t data) {
```

```
  atomic mensaje.xData.datos.luz=data;
        call ADC2.getData();
        return SUCCESS;
  }
  async event result_t ADC2.dataReady(uint16_t data) {//nivel bajo
if(data>=920){
       atomic cont1=0;
        atomic mensaje.xData.datos.level_l=1;
       }
else{
       atomic mensaje.xData.datos.level_l=0;
       }
          call ADC3.getData();
        return SUCCESS;
  }
 async event result_t ADC3.dataReady(uint16_t data){//nivel alto
if(data>=910){
       atomic mensaje.xData.datos.level_h=1;
       }
else{
              atomic mensaje.xData.datos.level_h=0;
              atomic cont2=0;
       }
          post send_datos();
        return SUCCESS;
  }
event result_t Send.sendDone(TOS_MsgPtr msg, result_t success)
```

```
  {
        call Leds.yellowToggle();
      atomic {
         msg_ptr = msg;
         sending_packet = FALSE;
      }
       return SUCCESS;
  }
async event result_t HPLUART.putDone(){
 return SUCCESS;
 }
async event result_t HPLUART.get(uint8_t data){
 return SUCCESS;
 }


event TOS_MsgPtr Receive.receive(TOS_MsgPtr msg, void* payload, uint16_t
payloadLen){
       call Leds.redToggle();
return msg;
 }
}
```

3 FORMATO DE MENSAJES

```
typedef struct XMeshHeader{
  uint8_t  board_id;
  uint8_t  packet_id; // 3
  uint8_t  node_id; // 3
  uint16_t parent;
}__attribute__ ((packed)) XMeshHeader;

typedef struct Sensores {
 uint16_t bateria;
 uint16_t temperatura;
 uint16_t luz;
 bool level_l; //bool
 bool level_h; //bool
} __attribute__ ((packed)) Sensores;


typedef struct sample {
  XMeshHeader xMeshHeader;
  union {
  Sensores datos;
  }xData;
} __attribute__ ((packed)) sample;
```