Sistemas Ubicuos, Empotrados y Móviles

Práctica 2: Diseño e implementación de un sistema

Máster Universitario en Ingeniería Informática
junio de 2021

Autor
Miguel Cabezas Puerto
70911497J
Óscar Sánchez Juanes
70918894G

El presente documento recoge el informe desarrollado por los alumnos Miguel Cabezas Puerto y Óscar Sánchez Juanes acerca de la segunda práctica de la asignatura Sistemas Ubicuos, Empotrados y Móviles en el seno del Máster en Ingeniería Informática de la Universidad de Salamanca en el curso 2020-2021, consistente en el diseño e implementación de un sistema de libre elección en el cual se empleen los conocimientos adquiridos en las diferentes sesiones.

Tabla de contenido

Tabla de ilustraciones

Introducción

El objetivo de la realización de este trabajo es aplicar los conocimientos adquiridos a lo largo de la asignatura. Para ello se ha diseñado un sistema que controla la temperatura de un reactor. Dicho sistema incluye dos placas, componentes de E/S digitales y analógicas, interrupciones, contadores y temporizadores, comunicaciones CAN y serie con terminal vía UART cableado y un módulo LCD.

Descripción del trabajo

Diseño del sistema

El sistema consiste en el control de la temperatura de un reactor. Se asume que existe una sala del reactor y una sala desde la que el operario monitoriza las variaciones de temperatura de dicho reactor. En el momento en el que se detecta que la temperatura medida es superior o inferior al rango seguro, se notifica, mediante una alerta auditiva y visual, al operario para que solvente la situación. En caso de que no se realice una corrección manual en un periodo establecido, el reactor será capaz de autorregularse.

Atendiendo a esto, el sistema se puede esquematizar de la siguiente manera:

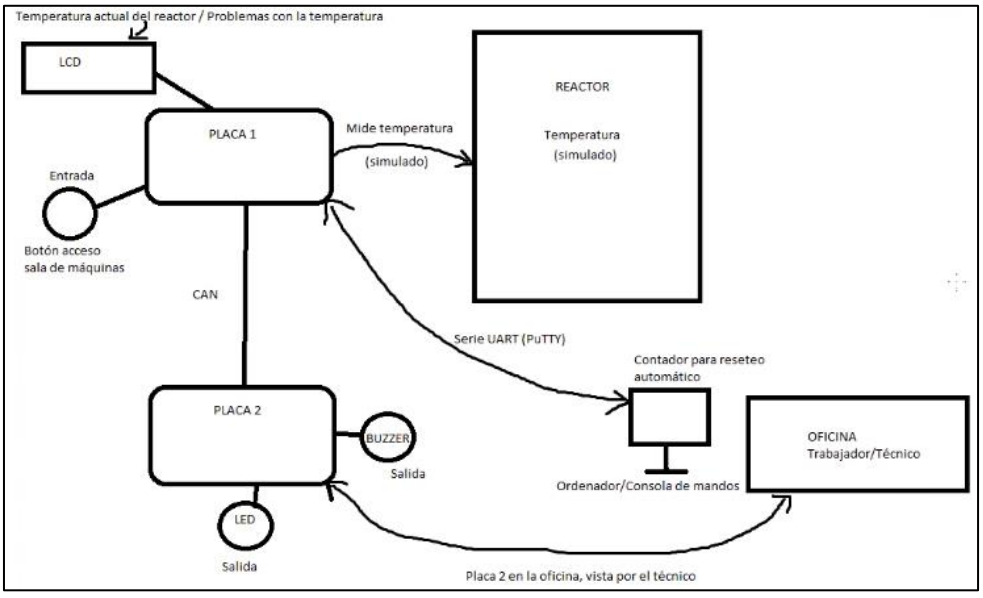

Ilustración 1: Esquema del sistema

El reactor tiene conectada una placa que simula la temperatura del mismo. A su vez, dicha placa contiene un LCD que muestra la temperatura en cada momento y, en caso de que ocurra una situación de peligro, informa sobre ello. Además, la placa se conecta con la consola del

equipo monitor a través de una comunicación serie UART cableada para enviar información sobre el tiempo transcurrido desde que se produjo la alerta.

Por otro lado, esta placa se comunica con la segunda a través de un protocolo de comunicación CAN. Esta segunda placa se encuentra situada en la sala del operario, de forma que, cuando se produce una alerta, la primera placa envía información a la segunda y ésta, a su vez, notifica al operario mediante sonido (*buzzer*) y luz (LED).

Finalmente, el operario puede solucionar manualmente el problema dirigiéndose a la sala del reactor. Para entrar a ella, debe pulsar el botón de acceso (conectado a la primera placa).

Componentes hardware

Para implementar el sistema de forma completa, se han hecho uso de los siguientes componentes hardware:

- Dos placas FRDM-KE06Z: Microcontrolador Kinetis E Series de 5V construido sobre el núcleo ARM® Cortex™-M0+ core.
  En la primera se simula la temperatura del reactor y notifica las situaciones de peligro que acontezcan. Además, permite solucionar de forma manual dichas situaciones.
  La segunda placa recibe información de peligro desde la primera y notifica al operario mediante luz y sonido.

- LCD (*Liquid-crystal display*): Pantalla que, en condiciones reales, se situaría en la puerta de la sala del reactor. En ella se muestra la información relativa a la temperatura de la sala y las posibles incidencias.

- Botonera: Componente que incluye una serie de botones, uno de los cuales se ha implementado como sistema de acceso a la sala del reactor.

- Cable PL2303HXD: Utilizado para la comunicación serie UART de la placa del reactor con el equipo en el que el operario monitoriza la temperatura desde su sala.

- *Buzzer*: Su funcionalidad dentro del sistema es emitir un sonido de alerta en la sala del operario cuando se detecta una incidencia en la temperatura del reactor.

Comunicaciones empleadas

Para enviar información a lo largo del sistema, se han empleado varios protocolos de comunicación diferentes:

- Protocolo de comunicación I2C (*Inter-Integrated Circuit*): Es un protocolo de comunicación serie síncrono que requiere dos líneas (más señal de tierra) y se utiliza para comunicar circuitos integrados dentro de un sistema. Se basa en la relación maestro-esclavo, donde el maestro controla el uso del bus.
  Este protocolo se emplea para la comunicación de una de las placas con el módulo LCD, en la que la placa adopta el rol de maestro y el módulo el LCD el de esclavo.

- Protocolo de comunicación CAN (*Controller Area Network*): Está basado en un protocolo del tipo CSMA/CD+AMP. Es un protocolo *multicast* en el que cada nodo debe monitorizar el bus CAN. Además, también es multimaestro, es decir, cualquier dispositivo conectado al bus puede enviar mensajes.
  En nuestro sistema se utiliza para la comunicación entre las dos placas, aunque sólo envía mensajes una de ellas.

  Nota: También se podría haber utilizado el protocolo de comunicación SPI al haber solo un maestro.

- Protocolo de comunicación serie UART (*Universal Asynchronous Receiver-Transmitter*): Es un protocolo que permite comunicaciones serie asíncronas del microcontrolador con una variedad de dispositivos.
  Se utiliza para la comunicación de una de las placas con la consola del equipo monitor.

Componentes lógicos Kinetis

Cabe destacar que, al emplear dos placas, cada una de ellas debe tener su propio proyecto, por lo que, a su vez, cada una va a tener sus propios componentes lógicos.

Antes de detallar los componentes lógicos de cada placa, es necesario incluir la configuración realizada para el procesador de cada placa (MKE06Z128VLK4).

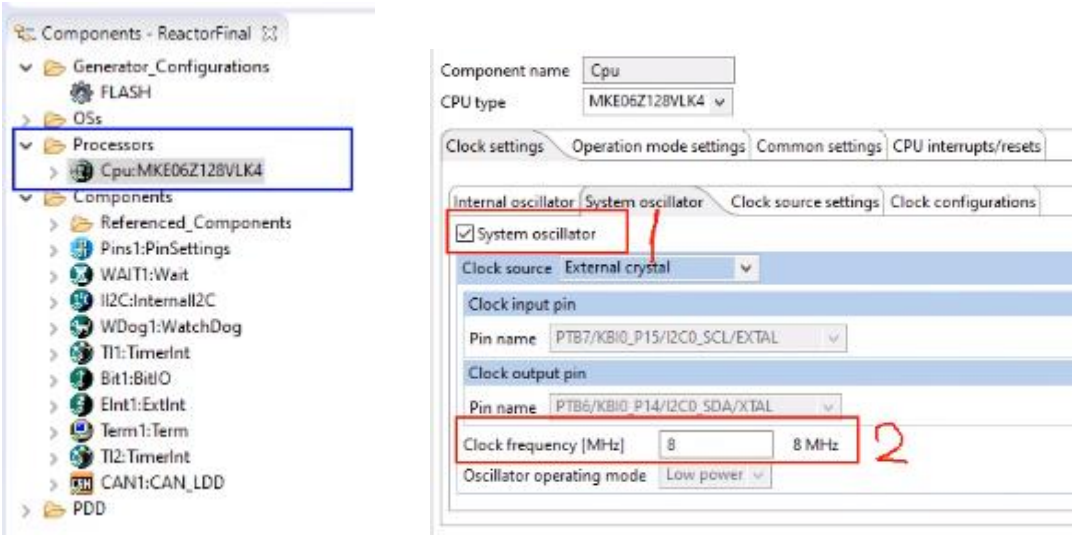

Ilustración 2: Configuración del procesador

Tal y como se puede apreciar en la *Ilustración 2*, hay que activar el *checkbox* "*System oscillator*", que se corresponde con el reloj externo. Una vez hecho esto, se establece la frecuencia de dicho reloj a 8MHz.

Componentes lógicos Kinetis Placa 1 (sala del reactor)

A continuación, se detallan los componentes lógicos de Kinetis utilizados en el proyecto ReactorFinal, correspondiente a la placa situada en la sala del reactor:

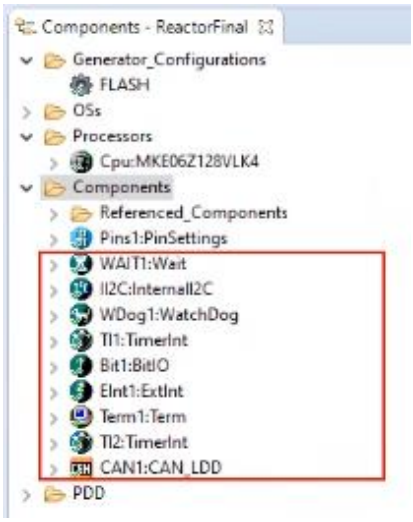

Ilustración 3: Componentes proyecto ReactorFinal

- *Wait*: Su función es implementar esperas dentro del programa (a modo de *sleep*).

- II2C (*Internal Inter-Integrated Circuit*): Gestiona la comunicación I2C de la placa con el módulo LCD.

En su configuración hay que establecer la frecuencia interna y los bits del 0 al 5 del registro divisor de frecuencia.

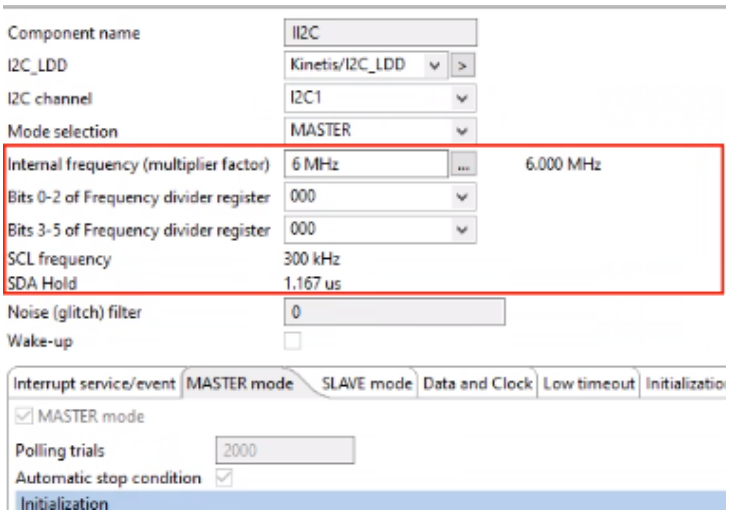

Ilustración 4: Configuración II2C

- *WatchDog Timer*: Es un contador que, al llegar a 0, lanza una interrupción y se ejecuta la acción establecida en su configuración. Por tanto, para poder usarlo de manera adecuada hay que determinar tanto el tiempo como la acción a realizar.

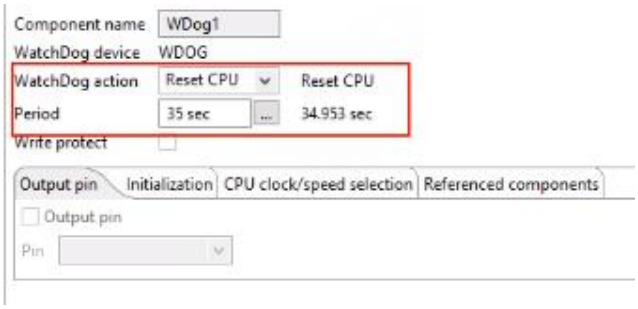

Ilustración 5: Configuración WatchDog Timer

- TimerInt: Es un componente para realizar interrupciones de forma periódica. Se han utilizado dos: uno para actualizar segundo a segundo el tiempo transcurrido desde que se produjo la alerta y otro para ajustar automáticamente la temperatura del reactor una vez finalizado el tiempo establecido.
  La configuración para ambos se realiza de forma análoga. Únicamente hay que configurar el periodo de interrupción.

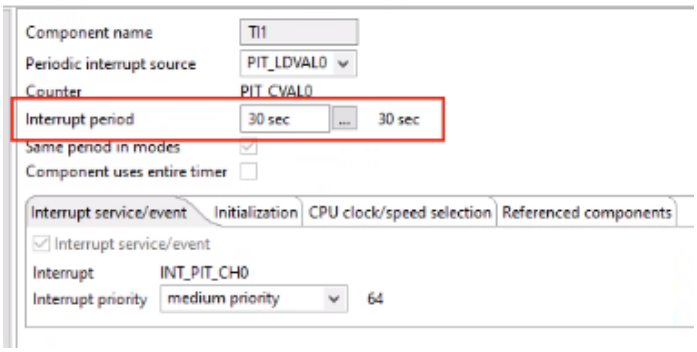

Ilustración 6: Configuración TimerInt

- ExtInt: Módulo que gestiona la pulsación de los botones de la botonera.
  En su configuración hay que determinar el pin y cuándo se genera la interrupción (en nuestro caso con flanco de bajada).

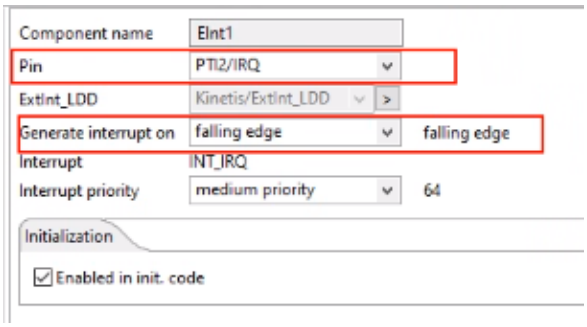

Ilustración 7: Configuración ExtInt

- Term: Componente para la comunicación con el equipo. Consta de un controlador para el SCI/UART, el AsynchroSerial, que permite leer y escribir caracteres de uno en uno o en bloque [1].
  En este caso hay que configurar dicho componente, en lugar del Term como tal.

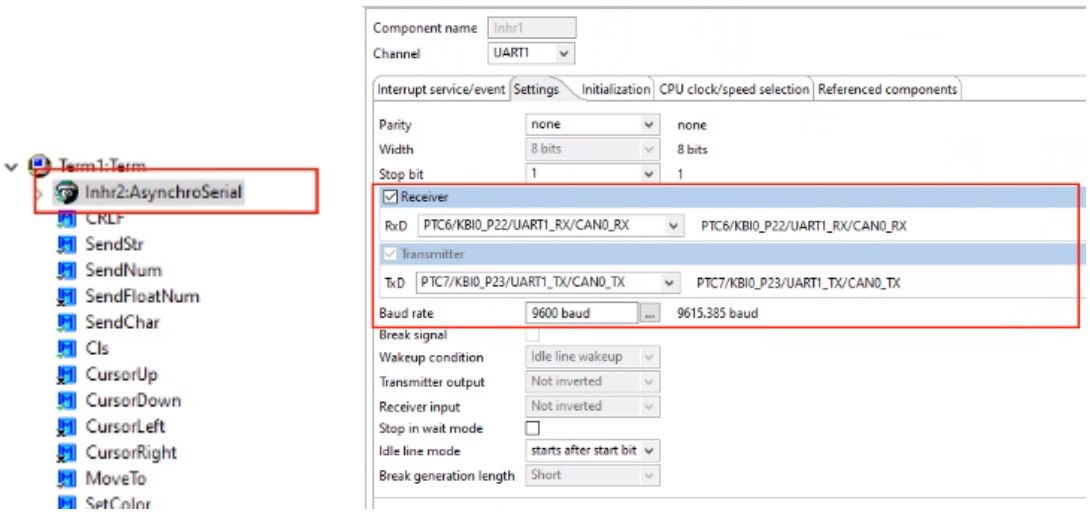

Ilustración 8: Configuración AsynchroSerial

- CAN_LDD: Componente utilizado para la comunicación CAN entre las dos placas. Durante su configuración hay que establecer el Bit rate y los pines de transmisión y recepción.

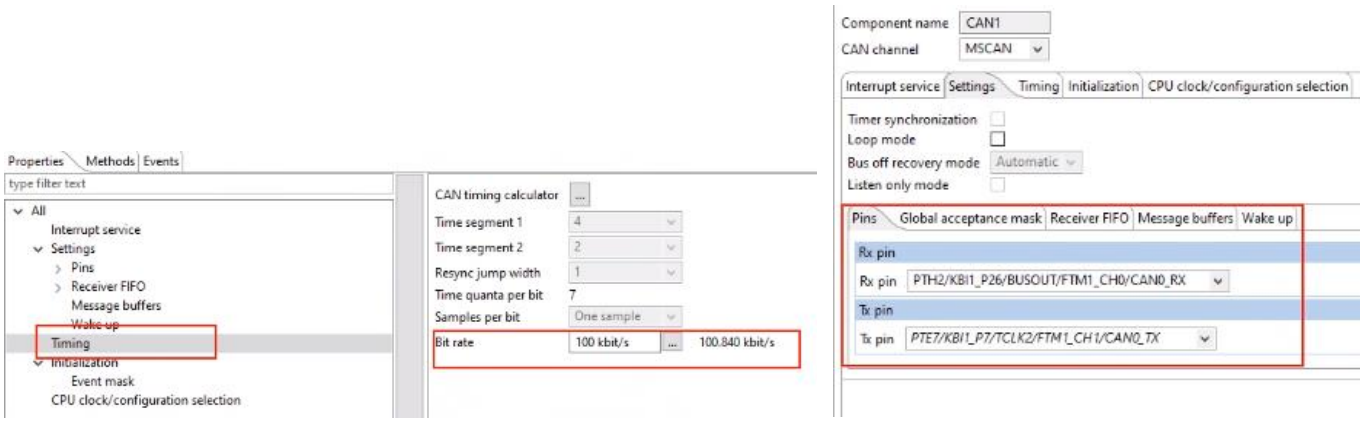

Ilustración 9: Configuración CAN_LDD

Componentes lógicos Kinetis Placa 2 (sala del operario)

A continuación, se detallan los componentes lógicos de Kinetis utilizados en el proyecto ReactorReception, correspondiente a la placa situada en la sala del operario:

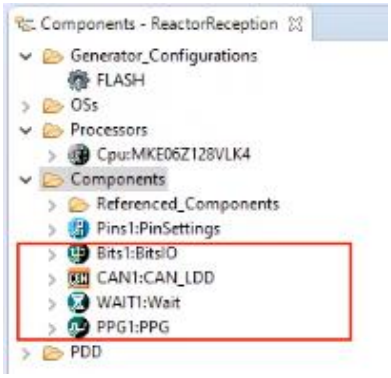

Ilustración 10: Componentes proyecto ReactorReception

- BitsIO: Utilizado para gestionar los colores del LED de la placa. Para su configuración hay que establecer el puerto, una dirección (salida, concretamente) y un pin.

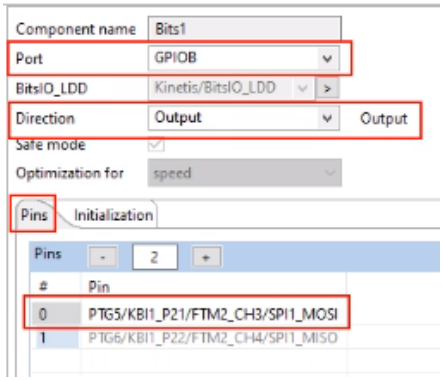

Ilustración 11: Configuración BitsIO

- CAN_LDD: Tanto su explicación como su configuración es análoga a la descrita para la placa anterior.

- Wait: Su función es implementar esperas dentro del programa (a modo de *sleep*).

- PPG – *Programable Pulse Generator*: Sirve para generar pulso con un determinado periodo y un ancho de pulso. Es utilizado para gestionar el funcionamiento del *buzzer*. Dentro de su configuración hay que establecer el FTM (*FlexTimer Module*) que se va a utilizar, el pin de salida, el periodo en modo intervalo y el ancho de pulso inicial.

  Nota: FTM es un temporizador de dos a ocho canales que admite captura de entrada, comparación de salida y generación de señales PWM para controlar aplicaciones de administración de energía y motores eléctricos [2].

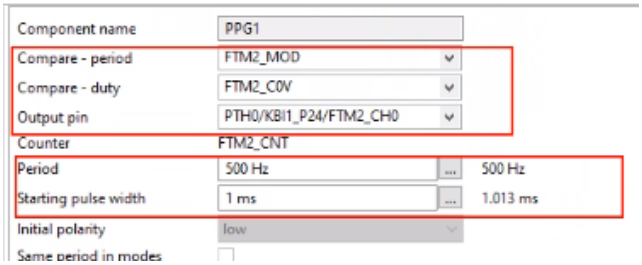

Ilustración 12: Configuración PPG (I)

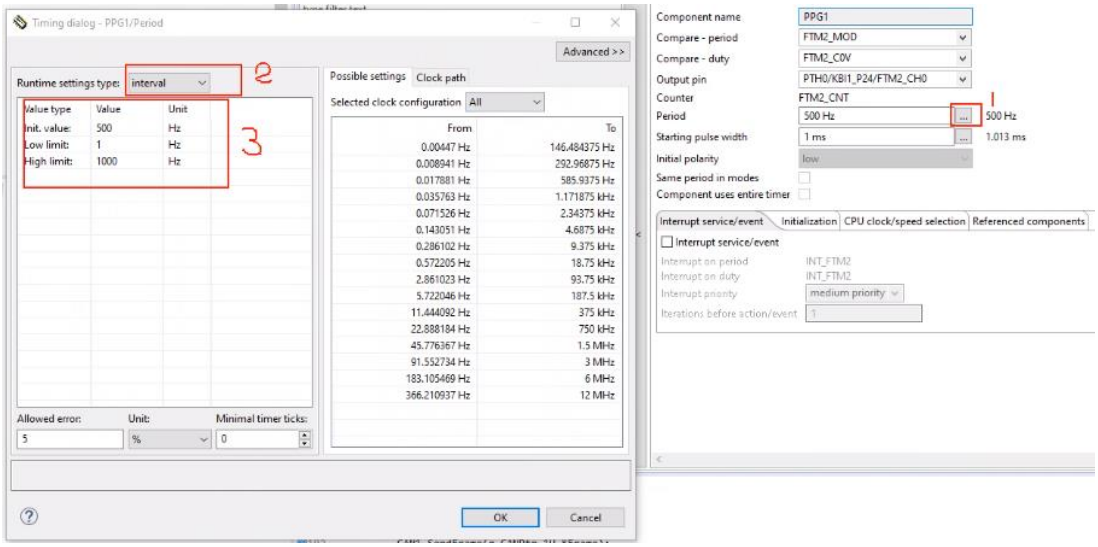

Ilustración 13: Configuración PPG (II)

Implementación

Al igual que para los componentes lógicos, al haber dos placas, cada una de ellas va a tener su propio código asociado.

A continuación se muestran los aspectos relevantes del código del programa de cada placa.

Si se desea consultar detalladamente el código, éste se encuentra disponible en el *Anexo 1: Código de los programas*.

Proyecto ReactorFinal (placa de la sala del reactor)

Main.c

Describe el flujo principal del programa correspondiente a la placa situada en la sala del reactor.

Seguidamente, se detallan los aspectos más importantes del mismo:

- Generación aleatoria de la temperatura: En la siguiente línea se muestra la forma en que se genera la temperatura cuando se sigue el curso normal del programa, es decir, no se ha producido ninguna incidencia.

```
97        temperatura=rand () % (LIMITE_SUPERIOR-LIMITE_INFERIOR+1) + LIMITE_INFERIOR;
```

Ilustración 14: Generación aleatoria de la temperatura

- Envío de información a la otra placa mediante el protocolo CAN: En las siguientes líneas se muestra el código asociado al envío de información cuando ocurre una incidencia (MessageID=0x123U), pero el procedimiento sería el mismo para el envío de información de vuelta a la normalidad (MessageID=0x124U).

```
99          Frame.MessageID = (0x123U); //LUZ ROJA, REACTOR EN PELIGRO
100         Frame.FrameType = LDD_CAN_DATA_FRAME;
101         uint8_t Message[] = {'T','O','F'};
102         Frame.Length = sizeof(Message);
103         Frame.Data = Message;
104         CAN1_SendFrame(g_CANPtr,1U,&Frame);
```

Ilustración 15: Envío de información protocolo CAN

Se puede apreciar como se crea el mensaje que se va a enviar (líneas 99-103) y el envío del mismo (línea 104).

Events.c

Describe los eventos e interrupciones que ocurren a lo largo del programa correspondiente a la placa situada en la sala del reactor.

- TI1_OnInterrupt(): Interrupción que se lanza cuando se acaba el tiempo dado para corregir manualmente la temperatura del reactor cuando existe una situación de peligro. En ella, se activa el *flag* de haber alcanzado el tiempo límite de reacción. Esto provoca que se solucione la incidencia automáticamente.

```
74  extern volatile timeReached;
75  void TI1_OnInterrupt(void)
76  {
77    /* Write your code here ... */
78      timeReached=1;
79  }
```

Ilustración 16: TI1_OnInterrupt()

- TI2_OnInterrupt(): Actualiza el tiempo transcurrido desde que se produjo la alarma. El valor correspondiente lo va mostrando por consola.

```
154 extern volatile int timeToEnd;
155 void TI2_OnInterrupt(void)
156 {
157   /* Write your code here ... */
158     Term1_SetColor(clWhite, clBlack);
159     Term1_Cls();
160     Term1_MoveTo(0,1);
161     Term1_SendStr("Tiempo transcurrido:");
162     Term1_MoveTo(22,1);
163     Term1_SendNum(timeToEnd);
164     Term1_MoveTo(0,2);
165     timeToEnd++;
166 }
```

Ilustración 17: TI2_OnInterrupt()

- Eint1_OnInterrupt(): Detecta la pulsación del botón de la botonera, de tal forma que sólo tendrá funcionalidad asociada cuando el programa se encuentre dentro del periodo en el que el operario puede restaurar manualmente la normalidad en la sala del reactor.

```
131 extern volatile manual_config;
132 extern volatile timeConcluded;
133 void EInt1_OnInterrupt(void)
134 {
135   /* Write your code here ... */
136     if(timeReached==0 && timeConcluded==1)
137         manual_config=1;
138 }
```

Ilustración 18: EInt1_OnInterrupt()

Proyecto ReactorReception (placa de la sala del operario)

Main.c

Describe el flujo principal del programa correspondiente a la placa situada en la sala del operario.

Seguidamente, se detallan los aspectos más importantes del mismo:

- Recepción de información a través del bus CAN: Comprueba continuamente, por sondeo, si recibe algún mensaje de la otra placa. En caso afirmativo, determina si el mensaje de alarma o de vuelta a la normalidad en la sala del reactor. Por el contrario, si no detecta ningún mensaje, mantiene su funcionamiento previo.

```
88  for(;;){
89      error=CAN1_ReadFrame(g_CANPtr,0U,&Frame);
90      if(error==ERR_OK){
91          if(Frame.MessageID == (0x123U)){
92              danger=1;
93              Bits1_PutVal(RGB_RED);
94              tone(TONO_LAB);
95          }else if(Frame.MessageID == (0x124U)){
96              danger=0;
97              Bits1_PutVal(RGB_GREEN);
98              notone();
99          }
100     }else{
101         if(danger==0){
102             Bits1_PutVal(RGB_GREEN);
103             notone();
104         }
105         else{
106             Bits1_PutVal(RGB_RED);
107             tone(TONO_LAB);
108         }
109     }
110 }
```

Ilustración 19: Recepción de información protocolo CAN

Events.c

Para la placa situada en la sala del operario, este archivo no contiene contenido relevante y únicamente está compuesto de código autogenerado por el IDE.

Aplicaciones externas

Tal y como se comentó al inicio del documento, el propósito principal del sistema es monitorizar la temperatura de un reactor. Sin embargo, podría aplicarse a otros casos donde se desee alertar a un operario acerca de una variación indebida en un parámetro deseado (temperatura, humedad, presión…). Para ello, bastaría con sustituir la generación aleatoria de los valores del parámetro, en nuestro caso la temperatura, por la medición recibida del sensor correspondiente, incluyendo éste en el sistema.

Bibliografía

[1] E. Styger, «Tutorial: Using a Terminal Input and Output; *without* printf() and scanf(),» 16 Noviembre 2013. [En línea]. Available: https://mcuoneclipse.com/2013/11/16/tutorial-using-a-terminal-input-and-output-without-printf-and-scanf/.

[2] M. Plachy, «Configuring the FlexTimer for position and speed measurement,» Diciembre 2011. [En línea]. Available: https://os.mbed.com/media/uploads/GregC/an4381-flextimer_configuration.pdf.