









3. Especificación de *mikro_STR*

Desglosa los pasos del método, la notación, el desarrollo de requerimientos lógicos y temporales del sistema, y las directrices para la codificación del programa en el microcontrolador bajo el acercamiento, orientada por el flujo de datos. Vale la pena destacar que lo contemplado aquí forma parte de un trabajo de grado en el área de Sistemas Digitales y Control de la carrera de Ingeniería Eléctrica de la Universidad de Oriente (Alfonsi, 2011).

3.1 Notación Gráfica

Los componentes que se deben utilizar en la elaboración de los DFD para la EGS, son las notaciones originales de los DFD, propuestas por Yourdon y De Marco, con las extensiones de Ward y Mellor para sistemas de tiempo real (Wieringa, 2003) presentadas en la tabla 2.

Tabla 2. Notación gráfica de los DFD (op cit.)

Componentes	Notación
a) Flujo de Datos discretos	(a) 
b) Flujo de Datos continuos	(b) 
c) Flujo de Control	(c) 
a) Almacén de datos	(a) 
b) Almacén de control	(b) 
Proceso de Datos	
Proceso de Control	
Entidad Externa	

Flujo de datos: se usa para describir el movimiento de bloques o paquetes de información (datos) de una parte del sistema a otra. Se diferencian dos tipos, flujo de datos discreto y flujo de datos continuo.

Flujo de control: representa señales o interrupciones; no aporta datos con valores. Se puede decir que un flujo de control provoca un cambio de estado en el sistema.

Almacén de datos: representa información del sistema almacenados de forma temporal. Son depósitos lógicos de almacenamiento y pueden representar cualquier dato almacenado o en reposo. Pueden aparecer de manera repetida en el diagrama para mejorar la legibilidad.

Almacenamiento de control: es un depósito de eventos de control que se guardan para ser usados por los procesos.









Proceso de datos: es el elemento fundamental del comportamiento funcional del sistema. Representa una función que transforma los flujos de datos de entrada en uno o varios flujos de datos de salida. Debe disponer de todos los datos de entrada suficientes para llevar a cabo el proceso y generar los datos de salida. Cada proceso debe llevar un nombre representativo, breve y único.

Proceso de control: su única labor es coordinar y sincronizar las actividades de las otras burbujas del diagrama de flujo de datos. Sus entradas y salidas consisten en sólo flujos de control. Se describen cómo máquinas de estado, cuyas acciones se toman cuando ocurre una transición desde un estado a otro. Por lo general se le asignan nombres como comenzar, controlar, entre otros.

Entidad externa: Es aquella que se encuentra fuera del control del sistema que se está modelando, es decir, son externos al sistema. Representa un generador o consumidor de información del sistema. Generalmente, sólo aparecerán en el diagrama de contexto.

Además de éstos, se incorpora una nueva representación, el proceso *tarea*, el cual se agregó con el propósito de tener una función con características tanto lógicas como temporales dentro del DFD. Posee todas las propiedades funcionales de un proceso de datos, con la diferencia de poseer también, atributos y características propias del concepto de tareas de tiempo real, que a simple vista caracterizan al bloque, tabla 3. Dentro de los atributos que debe poseer internamente el bloque se encuentran el tiempo de cómputo (Ci), el plazo de finalización (Di), el período (Ti) y el tiempo de liberación (ri).

Tabla 3. Notación gráfica de los DFD para la Tarea en *mikro_STR*

Bloque Funcional	Elementos	Notación
Característica	Crítica	
	Opcional	
Tarea Tipo	Periódica	
	Esporádica	
	Aperiódica	
	Combinada	
	Crítica-periódica	
	Opcional-aperiódica.	

3.2 Diseño de la Arquitectura Lógica

En las metodologías de análisis y diseño estructurado estos pasos vienen dados por la descomposición funcional del sistema en DFD. El sistema debe ser derivado en su modelo ambiental y de comportamiento (op cit.).

El modelo ambiental hace referencia a los elementos que son parte del sistema y los que no lo son. Consta de los siguientes componentes: declaración de propósitos, identificación de entidades externas (*EE*), diagrama de contexto (*DC*).

El modelo de comportamiento, describe la estructura interna que del sistema se requiere para que interactúe de manera exitosa con el ambiente. Esto se logra realizando una descomposición jerárquica/funcional de niveles en la forma de DFD, de forma iterativa, partiendo desde el diagrama de contexto del sistema. En esta etapa se debe tener en consideración las heurísticas para un diseño modular efectivo, como lo son la cohesión modular y el acoplamiento (Labrosse, Ganssle y Oshana, 2008; Laplante, 2004).

A continuación se realiza el desarrollo de la arquitectura lógica.

Declaración de propósitos

La estructura global de software *mikro_STR* tiene la finalidad de servir como una guía para la construcción de algoritmos de control estructurados en forma de tareas, en aplicaciones de tiempo real con microcontroladores de gama media. Cada tarea estará formada por un lazo independiente de control, siendo cada una de ellas responsable de manejar una planta o proceso en particular del sistema.

Identificación de entidades externas

A continuación se presentan las entidades externas elementales con las cuales el sistema propuesto tendrá interacción:

- Pulsador de inicio: señal de control que indica el inicio de operaciones.
- Pulsador de reinicio: señal de control externa que indica al sistema que se debe reiniciar desde el comienzo la ejecución del programa.

- Sensores: tienen la finalidad de suministrar al sistema la magnitud de la variable de la planta o proceso del lazo de control asociado; esta es proporcionada en forma continua en el tiempo.
- Actuadores: elementos finales de control que modifican las variables del proceso del lazo de control asociado, según la respuesta que le proporcione el sistema.

Las entidades externas que se nombran son opcionales dentro de la estructura general, ya que dependerán de la aplicación específica que se vaya a desarrollar y del propósito de la misma:

- Punto de consigna (i): es la entidad correspondiente de proporcionar al sistema el valor deseado en estado estacionario de las variables a controlar en los diferentes lazos de control. Este dato es suministrado por el usuario. A pesar de suministrar un dato indispensable para correcto funcionamiento del sistema, la existencia o no de esta entidad externa dependerá de las características del proceso, ya que si éste no requiere de constantes modificaciones en el punto de consigna durante su funcionamiento, puede reservarse un espacio en la memoria del sistema para almacenar estos datos, y reducir la complejidad del sistema.
- Reloj del sistema: encargado de llevar el histórico del tiempo de ejecución del sistema, y es proporcionado de forma continua en el tiempo. Su presencia dentro de la estructura global va a depender de la política de planificación que se decida implementar, como por ejemplo aquellas donde las prioridades de las tareas varían en forma dinámica.
- Temporizadores: generan interrupciones periódicas al sistema cuando se cumple el tiempo para el cual fueron programados. Sirven como apoyo para el cumplimiento de los requerimientos temporales, y la presencia de éstos dentro del sistema también va a depender de la política de planificación seleccionada y las características temporales de las tareas.
- Led de tarea (i): su propósito es indicar la tarea que actualmente se está ejecutando dentro del sistema.

Diagrama de contexto del sistema

En la Fig. 3 se muestra el diagrama de contexto o diagrama de flujo de datos de nivel 0 para la estructura global de software *mikro_STR*.

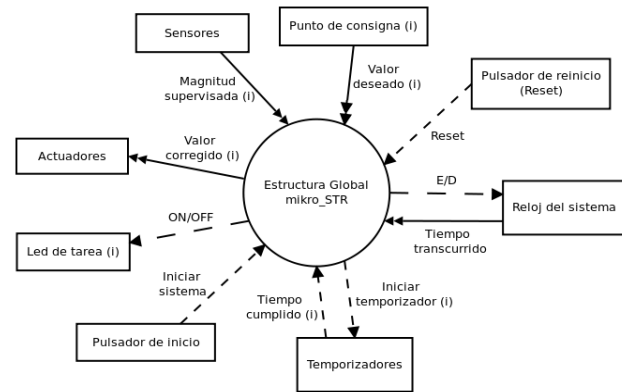


Figura 3. Diagrama de contexto para *mikro_STR*.

Descomposición del diagrama de contexto: DFD de nivel 1 para *mikro_STR*

En la Fig. 4 se presenta la descomposición de nivel 1, mostrando los bloques básicos internos del sistema: planificador de tareas, controlador de inicio y tarea de control (Ti).

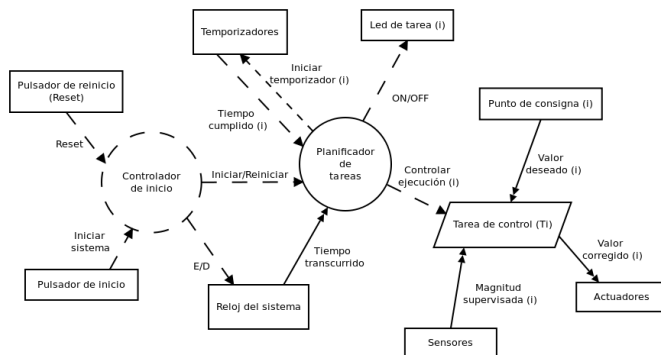


Figura 4. Diagrama de flujo de datos de nivel 1 para *mikro_STR*.

Alfonsi A., Alfonsi A.R. y Yáñez R. Estructura Global de Software para Aplicaciones de Control Empotrado con Restricciones Temporales en Microcontroladores con Recursos Limitados. Informe Final Proyecto de Investigación CI-UDO. Universidad de Oriente, Barcelona, 2015.

El proceso *Planificador de tareas* es el más importante de todos, debido a que es el encargado de controlar la ejecución de todas las tareas del sistema con tal que satisfagan en todo momento los requerimientos temporales del mismo. En la Fig. 5 se presenta la descomposición funcional elemental de este proceso. Se tienen tres subfunciones: *Identificación de interrupciones*, que se encarga de determinar qué temporizador generó la interrupción, y proporciona esta información a la política de planificación; *Gestión de atributos*, el cual lee periódicamente el tiempo transcurrido y actualiza los atributos de las tareas del sistema; *Política de planificación*, lee del almacén los atributos de las tareas, y dependiendo de la interrupción que haya recibido del bloque de identificación y de la política de planificación seleccionada, toma las decisiones correspondientes en cuanto a la ejecución de las tareas.

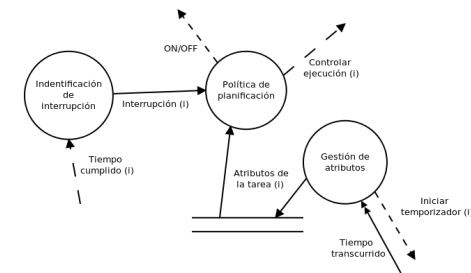


Figura 5. Diagrama de flujo de datos nivel 2 para planificador de tareas.

DFD de nivel 2 para Tarea de control (Ti)

En la Fig. 6 se presenta la descomposición funcional del proceso *Tarea de control (Ti)* del DFD nivel 1. Se muestran las funciones lógicas elementales que debe contener el bloque de manera de cumplir con la definición de tarea de control, sin dejar de lado sus propiedades temporales.

Alfonsi A., Alfonsi A.R. y Yáñez R. Estructura Global de Software para Aplicaciones de Control Empotrado con Restricciones Temporales en Microcontroladores con Recursos Limitados. Informe Final Proyecto de Investigación CI-UDO. Universidad de Oriente, Barcelona, 2015.

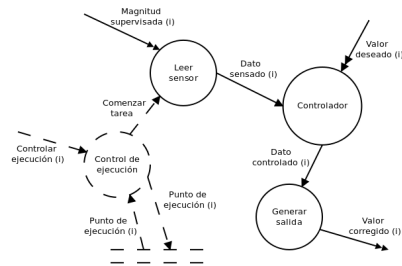


Figura 6. Diagrama de flujo de datos de nivel 2 para Tarea de control

Dentro de la estructura global, este proceso es el que se encarga de leer la variable del lazo de control correspondiente, calcular la acción de control necesaria para lograr el punto de consigna o referencia establecida y generar la respuesta apropiada a dicho lazo. Se divide en cuatro subfunciones: *Leer sensor*, encargado de tomar la muestra de la magnitud leída por el sensor y convertirla en un dato discreto. La subfunción *Generar salida*, que recibe el valor corregido de la variable controlada del lazo, y produce una señal directa sobre el actuador para llevar a la variable de la planta o proceso al punto calculado. Aquí se alojará el algoritmo de conversión necesario para que el actuador haga su trabajo; dependerá del tipo de actuador que se esté usando en el proceso. Por último, la subfunción *Controlador*, que calcula la acción de control del lazo al cual pertenece. Su algoritmo está depende directamente del tipo de controlador que se seleccione.

Tarea de control (Ti) también posee un proceso de control, encargado controlar la ejecución del bloque funcional. Recibe una señal de control proveniente del planificador, la cual indica si la tarea debe ser iniciada, suspendida o reanudada; su funcionamiento se describe a continuación: el proceso *Control de ejecución* recibe la señal codificada del planificador, éste la interpreta y decide cuál de las tres vías debe tomar. Si el proceso identifica que debe iniciar la tarea, envía un evento a la subfunción leer sensor para que inicie operaciones, y a partir de allí se ejecuta de forma secuencial el código de la tarea de control (leer sensor, calcular acción de control, generar señal corregida). Si la tarea se está ejecutando y se identifica una señal de suspensión, *Control de ejecución* guarda en el almacén el punto de ejecución actual de la tarea y pasa el control del sistema de nuevo al planificador, en otras palabras, se desaloja la tarea guardando el último punto de ejecución. Y por último, en caso de tratarse de una reanudación, se

extrae del almacén el último punto de ejecución de la tarea que fue almacenado, y se continúa con el proceso de ésta desde donde había quedado.

3.3 Diseño de la Arquitectura Física

Planificador de tareas

Para poder predecir el comportamiento de un sistema de tiempo real es necesario imponer restricciones temporales explícitas a todos sus componentes. En la estructura global de software *mikro_STR*, estas restricciones se manejan a través del proceso *Planificador de tareas*, por medio de la subfunción *Política de planificación*.

El diseño temporal del sistema se fundamenta en el establecimiento de un conjunto de pasos que permitan predecir el comportamiento y garantizar la ejecución del sistema en el peor caso, cuando se aplica un algoritmo de planificación. Es en este punto entonces, donde se debe seleccionar la política de planificación que regirá la ejecución de las tareas en el procesador y las pruebas de planificabilidad, donde se podrá comprobar si el sistema cumple o no las restricciones temporales que se le imponen.

Existen dos vertientes en cuanto a políticas de planificación, las guiadas por prioridades y los ejecutivos cíclicos (Buttazzo, 2011).

Pruebas de Planificabilidad

Es un método de análisis que se utiliza para comprobar si el sistema cumple o no con las restricciones temporales antes de implementarlo, también son llamados análisis de planificabilidad (op cit.). Si se escoge un algoritmo de planificación basado en prioridades, ya sean estáticas o dinámicas, existen dos formas de comprobar la planificabilidad del sistema, ya sea por análisis basado en la utilización, o el tiempo de respuesta de peor caso. Si por el contrario se decide utilizar un algoritmo de planificación ejecutivo cíclico, se debe emplear lo que se denomina prueba por construcción. Para esta etapa es necesario conocer como mínimo, el valor de los tiempos de cómputo de peor caso de las tareas.

Reloj del Sistema, Temporizadores e Interrupciones

Se propone utilizar temporizadores e interrupciones para garantizar el correcto comportamiento temporal del sistema, ya que como es bien sabido, la forma nativa de trabajar los microcontroladores es secuencial.

El *Reloj del sistema*, definido como una entidad externa opcional, es para algoritmos de planificación basados en prioridades dinámicas, donde constantemente se requiera estar actualizando los atributos de las tareas. Periódicamente, se producirá una interrupción en el sistema, para que la subfunción *Gestión de atributos* (Fig. 5) lea el valor del tiempo transcurrido y pueda actualizar en el almacén de datos, los valores de los atributos de las tareas. Constantemente, estos atributos son leídos por el algoritmo de planificación para decidir el orden de ejecución de las tareas.

Los *Temporizadores*, pudiendo utilizar los incorporados internamente en la arquitectura del microcontrolador, o bien, externos a éste, conectados como periféricos al mismo. El empleo de los temporizadores tiene como finalidad asignar por cada tarea T_i del sistema, un temporizador individual, el cual será programado para generar una interrupción en la ejecución normal del sistema, para indicar al planificador cada vez que una instancia de una tarea es activada (tiempo de levantamiento). Es aquí donde la subfunción *Identificación de interrupciones* (Fig. 5) entra en funcionamiento, identificando qué interrupción fue generada, pasando esta información al algoritmo de planificación, el cual debe decidir la acción a tomar.

En cuanto a la configuración de los temporizadores, para su inicialización, la subfunción *Gestión de atributos* (Fig. 5) indica mediante un evento (señal de control) el momento en el cual debe ser iniciado cada temporizador. El tiempo de desborde es programando al inicio del sistema en el caso de los temporizadores internos del microcontrolador, y en el caso de los temporizadores externos, dicho tiempo debe ser programado de antemano (*off-line*) en cada uno de ellos.

- Atributos temporales de las tareas: Definidos como tipo arreglo (*array*) de números enteros. El tamaño de cada uno será igual al número de tareas del sistema, donde cada índice del arreglo será asociado a una tarea. Dentro de las variables implicadas en esta definición se encuentran: el tiempo de cómputo, plazo de finalización y período de la tarea.

Alfonsi A., Alfonsi A.R. y Yáñez R. Estructura Global de Software para Aplicaciones de Control Empotrado con Restricciones Temporales en Microcontroladores con Recursos Limitados. Informe Final Proyecto de Investigación CI-UDO. Universidad de Oriente, Barcelona, 2015.

- Byte de interrupción de tarea (*B_interrupt*): A cada tarea le es asignado un temporizador externo el cual periódicamente estará generando interrupciones. La función de esta variable es, identificar qué temporizador generó la interrupción al sistema, de manera que el planificador pueda tomar acciones al respecto. A cada interrupción le es asignada una codificación dentro de esta variable. Al producirse alguna de ellas, el programa irá a la respectiva función de atención de dicha interrupción, cuya finalidad es simplemente modificar la variable con el código de la interrupción que se produjo, y pasar el control del sistema a la política de planificación para que esta decida qué hacer. Esta función está representada como *interrupcioni* dentro de la fuente *Planificador.c*. Se debe recordar también que estas entidades externas (temporizadores) fueron previstas de manera opcional para el sistema, por lo tanto, el uso del *Byte de interrupción de tarea* va a depender de la existencia de ellas, no obstante, la variable está definida dentro de la librería propuesta.
- Número de tareas: Definida como un entero corto (*int*), y representa al número total de tareas del sistema.

3.4 Diseño Detallado y Codificación

Se presenta la codificación de los módulos de la estructura global de software, usando como referencia el microcontrolador *AVR ATmega328* (Atmel AVR, 2014) bajo el entorno de desarrollo *Arduino IDE (Interface Development Environment)* (Arduino, 2014). A pesar de programar para este microcontrolador y este entorno de desarrollo, es posible utilizar a *mikro_STR* como base para implementarlo en cualquier otro microcontrolador de similares prestaciones y bajo cualquier otro entorno de desarrollo.

El código está dividido en dos partes: el módulo de planificación y el cuerpo principal del programa (*main*), sección del código que generalmente el usuario modifica dependiendo de las características del sistema.

Planificador de tareas

El módulo se realizó por separado, de manera de centralizar las definiciones y funciones del planificador de tareas en un archivo, dejando al programa principal con menos líneas de

Alfonsi A., Alfonsi A.R. y Yáñez R. Estructura Global de Software para Aplicaciones de Control Empotrado con Restricciones Temporales en Microcontroladores con Recursos Limitados. Informe Final Proyecto de Investigación CI-UDO. Universidad de Oriente, Barcelona, 2015.

código. Se creó el archivo fuente del planificador, con extensión *.c (*Planificador.c*), el cual contiene un listado con la declaración de ciertas variables y funciones, además de poseer el cuerpo de dichas funciones, las cuales representan a los submódulos del proceso *Planificador de tareas*.

En el archivo fuente *Planificador.c*, también se incluyen las declaraciones de las funciones que representan a las tareas del sistema. Esta sección del código debe ser modificada por el usuario dependiendo del número de tareas existentes dentro de la aplicación. A diferencia de las otras funciones de este archivo, propias del planificador, el cuerpo de las tareas no se incluye en el mismo, de tal manera que el código fuente del planificador sea lo más genérico posible. El cuerpo de las tareas será especificado por cada usuario en el archivo principal del proyecto (*main*). A continuación se presenta el código del archivo fuente *Planificador.c*:

```
//archivo: Planificador.c
//Librerías necesarias para el fuente:
#include "WProgram.h"
//#include otras...
//Declaración de variables globales del cuerpo del //programa:
#define N n; //n = número de tareas
static byte B_interrupt;//Byte interrupción de tarea
static byte bite;
static byte B_tarea1; //Byte de tarea, tarea 1
static byte B_tarea2; //Byte de tarea, tarea 2
//...
static byte B_tareaN; //Byte de tarea, tarea N
static int computo[N] = {tarea1, tarea2,tareaN};
static int periodo[N] = {tarea1, tarea2,tareaN};
static int plazo[N] = {tarea1, tarea2,tareaN};
//otras variables...
//Subfunciones del módulo de planificación:
void politica()
{
  /* Fragmento del código asociado al algoritmo de planificación que se desee implementar. */
}
void interrupcion_i() //identificación interrupción
{
  B_interrupt = //código de la tarea asociada
  B_tareai = (B_tareai&0x3F)|0x80; //Ti lista
  politica();
}
void gestion() //gestión atributos, int. periódica
{
  static int i;
  static int tiempo;
  tiempo = //leer tiempo transcurrido
//inicializar temporizadores:
/* rutina de inicialización de temporizadores.
se puede usar una estructura de control "if"
```

Alfonsi A., Alfonsi A.R. y Yáñez R. Estructura Global de Software para Aplicaciones de Control Empotrado con Restricciones Temporales en Microcontroladores con Recursos Limitados. Informe Final Proyecto de Investigación CI-UDO. Universidad de Oriente, Barcelona, 2015.

```
para cada caso cuando:tiempo de referencia (tareai) =tiempo transcurrido */
//actualizar atributos:
for(i=0; i<N; i++){
  switch(i){
    case 0: //tarea 1
      bite = B_tarea1;
      break;
    case 1: //tarea 2
      bite = B_tarea2;
      break;
    /*
    ...
    case N-1: //tarea N
      bite = B_tareaN;
      break;
    */
  }
  bite = (bite&0xC0)>>6;
  if(bite == 3){}
  else{
    plazo[i]=plazo[i]-tiempo;//o realizar con
    //período, depende de la política empleada
  }
}
//Declaración de las tareas del sistema
void tarea_1();
void tarea_2();
//...
void tarea_N();
```

En el código presentado anteriormente, se puede observar el archivo de cabecera *WProgram.h*. Éste debe incluirse cuando en el código estén presentes funciones propias del lenguaje Arduino.

La función *gestión* es activada periódicamente, cada vez que se cumpla el tiempo del temporizador interno (si se llega a utilizar). La configuración de este temporizador se debe realizar en el archivo principal. Se propone la utilización de una librería para el entorno de desarrollo Arduino (Arduino, 2014), *MsTimer2*, la cual trabaja con el temporizador *Timer2* del microcontrolador. Ésta genera una interrupción cada *n* milisegundos, enviando el control del programa hacia la función que se le indique, donde *n* es especificado por el usuario. En el cuerpo principal del programa se incluyeron las rutinas de configuración e inicio para este temporizador con la librería *MsTimer2*.

Como no todas las tareas necesariamente son activadas en el mismo instante, en la función de gestión de atributos, se encuentra una sección a la inicialización de los temporizadores externos. Como esta función se ejecuta cada cierto tiempo, de forma periódica, se puede

Alfonsi A., Alfonsi A.R. y Yáñez R. Estructura Global de Software para Aplicaciones de Control Empotrado con Restricciones Temporales en Microcontroladores con Recursos Limitados. Informe Final Proyecto de Investigación CI-UDO. Universidad de Oriente, Barcelona, 2015.

comparar el valor del tiempo transcurrido con un valor de referencia (previamente definido), y dependiendo del resultado, se inicia o no, el conteo del temporizador asociado.

En cuanto al control de ejecución de tareas por parte del planificador, se definen cuatro posibles estados para una tarea T_i del sistema, Fig. 7.

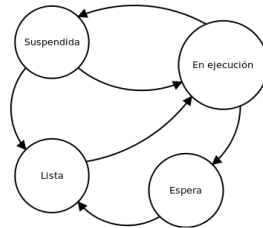


Figura 7. Diagrama de estados para una tarea T_i

- **Espera:** estado inicial o *stand-by* de la tarea. La tarea se encuentra a la espera de su i -ésima activación, código 00.
- **Lista:** estado de T_i cada vez que se produce una nueva activación, código 10. Es a partir de este momento, cuando el planificador toma en cuenta a la tarea en cuestión.
- **En ejecución:** indica que T_i está haciendo uso del procesador en ese momento, código 01.
- **Suspendida:** se utiliza para indicar que la tarea fue expulsada por el planificador. Tiene dos posibles transiciones, código 00.

Si alguna tarea se encuentra en el estado de espera, la subfunción *Gestión de atributos* no debe actualizar los atributos temporales de la misma, para no poner en conflicto al algoritmo de planificación. Sólo actualizarán los atributos de aquellas tareas que se encuentren en cualquiera de los otros 3 estados, las cuales son en ese momento.

Cuerpo principal del programa (main)

Se desarrolla bajo el entorno de programación de Arduino. La estructura básica del lenguaje de programación Arduino se organiza como mínimo en dos funciones obligatorias que encierran bloques de declaraciones, éstas son la función *setup* y la función *loop*. Adicional a estos

dos bloques obligatorios, es posible escribir nuevas funciones, que para el caso de estudio, serán las diferentes tareas del sistema.

La función *setup* contiene la declaración de cualquier variable al comienzo del programa. Es ejecutada una vez, al inicio, y es usada para la configuración del sistema en general: configurar los modos de los pines del microcontrolador, módulos de interrupciones, temporizadores, entre otros. La función *loop* incluye el código que será ejecutado continuamente en el microcontrolador como bucle principal.

Introduciendo el concepto de tareas y planificación de tiempo real, es necesario realizar ciertos cambios en la forma habitual de programación. En un sistema de tiempo real, el módulo encargado de llevar el control de ejecución de las tareas es el planificador, por lo que la diferencia principal será la sustitución del programa principal dentro la función *loop* (en este caso) por un simple bucle infinito, dando la posibilidad al planificador de tomar el control del sistema, ante la aparición de eventos temporales en forma de interrupciones.

A continuación se muestra el código para el archivo principal (*main*) de la estructura global de software, bajo el entorno Arduino *IDE*, con extensión **.pde*:

```

//archivo: main_mikroSTR.pde
//Librerías y archivos de cabecera para el sistema:
#include "Planificador.c"
#include <MsTimer2.h>
//include otras...
//Declaración de variables globales del cuerpo del //programa:
boolean ON = 0;
boolean inicio;
//otras variables...
void setup()
{
  /* Función de configuración del sistema. En esta sección del código se definen los modos de los pines,
  se inicializan las comunicaciones, interrupciones, temporizadores, entre otros. */
  attachInterrupt(0,interrupcion_i,modo);
  //activar interrupciones externas donde i=1,2,3
  //otras interrupciones
  MsTimer2::set(n,gestion);
  //configurar interrupción //periódica donde n, //cantidad de milisegundos
}
void loop()
{
  while (ON == 0){ //proceso "Controlador de Inicio"
    inicio = digitalRead(pin); //pin asociado con el //pulsador de inicio
    if (inicio == LOW) ON = 0;
    else if (inicio == HIGH){
      ON = 1;
      MsTimer2::start();//iniciar interrupción
    }
  }
}
  
```

```

        //periódica ir a planificador
        //No se especifica la función
    }
}
while (ON == 1){} //bucle infinito
}
void tarea_1()
{
    //Fragmento del código asociado a esta tarea
}
void tarea_2()
{
    //Fragmento del código asociado a esta tarea
}
//...
void tarea_N()
{
    //Fragmento del código asociado a esta tarea
}

```

Otros archivos fuente y librerías pueden ser incluidos, por lo general, se recomienda que los archivos fuentes que se incluyan sean guardados en la misma ruta del proyecto principal, *main_mikroSTR.pde*. Las librerías pueden estar almacenadas en la misma carpeta del proyecto, o también, en la ruta. En la función *setup*, estarán presentes las configuraciones de las interrupciones de los temporizadores a utilizar para las tareas. Se colocó a manera de ejemplo la configuración de la interrupción externa 0, asociada al pin digital 2 de la placa de desarrollo Arduino. El cuerpo de la función destino de la interrupción, está incluida en el archivo fuente del planificador. Para cada interrupción externa, debe existir una función *interrupción_i* de atención de interrupción diferente.

Como se pudo apreciar en el código, en la sección de cabeceras se incluyen el archivo fuente del planificador (*Planificador.c*) y la librería de interrupción interna con el *Timer2* del microcontrolador (*MsTimer2.h*). El uso de la librería de interrupción es opcional, se coloca a manera de ejemplo; el usuario tiene la potestad de decidir si usará esta librería, o alguna otra que desee.

Cuerpo de las tareas de control

A continuación se muestra la codificación del cuerpo básico de una tarea de control para *mikro_STR*.

```

void tarea_i()
{
    //declaración de variables internas de la función

```

Alfonsi A., Alfonsi A.R. y Yáñez R. Estructura Global de Software para Aplicaciones de Control Empotrado con Restricciones Temporales en Microcontroladores con Recursos Limitados. Informe Final Proyecto de Investigación CI-UDO. Universidad de Oriente, Barcelona, 2015.

```

static int consigna_Ti,valor_corregido;
static float uT; //respuesta del controlador
//otras variables internas...
B_tareai = (B_tareai&0x3F)|0x40;//Ti en ejecución
consigna_Ti = analogRead(pin1);//subfunción "Leer
                //sensor"
/* Fragmento del código asociado al algoritmo de control que se desee implementar. */
uT = constrain(uT, min, max);//acondicionar salida
valor_corregido = map(uT, max, min, 0, 255);
analogWrite(pin2,valor_corregido);//subfunción
                //"Generar salida"
B_tareai = (B_tareai&0x3F)|0xC0; //Ti finalizada,
                //espera nueva activación
}

```

Las variables *pin1* y *pin2*, las cuales están dentro de las funciones *analogRead* y *analogWrite* respectivamente, son de tipo entero (*int*), y el valor de éstas corresponde al número del pin con el que se haya asociado; no es necesario declararlas si el usuario sustituye dicho campo por el número específico del pin.

Las funciones *constrain* y *map*, definidas en el lenguaje propio de Arduino, son utilizadas para restringir y acondicionar la salida calculada por el controlador hacia el mundo exterior, si se utiliza el módulo de conversión Digital/Análogo, por medio de la función *analogWrite*. El módulo de conversión D/A, que trabaja por modulación de ancho de pulsos (*PWM* por sus siglas en inglés), sólo permite valores enteros comprendidos entre 0 y 255. Las constantes *min* y *max* deben ser declaradas al inicio de la función, y su valor está sujeto al tipo de proceso que se esté manejando.

El campo del controlador fue omitido a propósito, dejando a potestad del usuario la implementación del algoritmo de control que desee. Sin embargo, se define una variable, *uT*, con fines ilustrativos, que representa el valor de la respuesta del controlador.

Variables especiales

- Byte de tarea (*B_tareai*): variable de identificación para cada tarea, su estructura se puede ver en la Fig. 8. Sus dos bits más significativos contienen el estado actual de la tarea. Los otros seis bits restantes serán utilizados para asignarle un código único a cada tarea dentro del sistema. Esta variable será de mucha utilidad para el planificador de tareas a la hora de tomar sus decisiones. A esta variable sólo podrán ser modificados durante la ejecución del sistema los dos bits correspondientes al estado de la tarea asociada, cuando así se

Alfonsi A., Alfonsi A.R. y Yáñez R. Estructura Global de Software para Aplicaciones de Control Empotrado con Restricciones Temporales en Microcontroladores con Recursos Limitados. Informe Final Proyecto de Investigación CI-UDO. Universidad de Oriente, Barcelona, 2015.

requiera. El código de la tarea será asignado sólo una vez, y se realizará antes de colocar el sistema en funcionamiento.

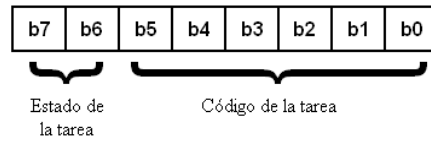


Figura 8. Byte de tarea

- Byte de interrupción de tarea (*B_interrupt*): A cada tarea le es asignado un temporizador externo el cual periódicamente estará generando interrupciones. La función de esta variable es, identificar qué temporizador generó la interrupción al sistema, de manera que el planificador pueda tomar acciones al respecto. A cada interrupción le es asignada una codificación dentro de esta variable. Al producirse alguna de ellas, el programa irá a la respectiva función de atención de dicha interrupción, cuya finalidad es simplemente modificar la variable con el código de la interrupción que se produjo, y pasar el control del sistema a la política de planificación para que esta decida qué hacer. Esta función está representada como *interrupcion* dentro de la fuente *Planificador.c*. Se debe recordar también que estas entidades externas (temporizadores) fueron previstas de manera opcional para el sistema, por lo tanto, el uso del *Byte de interrupción de tarea* va a depender de la existencia de ellas, no obstante, la variable está definida dentro de la librería propuesta.
- Número de tareas: Definida como un entero corto (*int*), y representa al número total de tareas del sistema.
- Atributos temporales de las tareas: Definidos como tipo arreglo (*array*) de números enteros. El tamaño de cada uno será igual al número de tareas del sistema, donde cada índice del arreglo será asociado a una tarea. Dentro de las variables implicadas en esta definición se encuentran: el tiempo de cómputo, plazo de finalización y período de la tarea.

3.5 Ejemplos de Aplicación: Diseño y Simulación de un Sistema de Control PID

Se aplicó *mikro_STR* a un sistema de control PID de calefacción presentado en Ruge (2012), con un planificador de tiempo real ejecutivo cíclico (Baker y Shaw, 1989) en un microcontrolador *Atmega328* (Atmel AVR, 2014). Las herramientas computacionales fueron, el entorno de desarrollo *Arduino IDE* (Arduino, 2014), de amplia utilización en proyectos empotrados de hardware libre, y para la simulación del sistema se utilizó *ISIS Proteus*, en su versión *Lite* (Proteus Lite, 2011).

Diseño de la Arquitectura Lógica

Para la elaboración de la estructura del software del sistema de control de calefacción, se desarrolló la arquitectura lógica del sistema, Fig. 9.

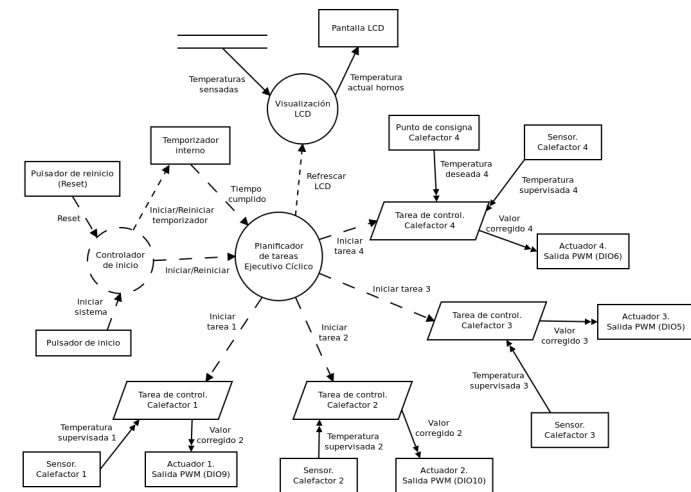


Figura 9. Diagrama de flujo de datos de nivel 1.

Diseño de la Arquitectura Física

Las características temporales de las tareas se tomaron de Alfonsi y Alfonsi (2012), y se procedió a realizar la prueba por construcción para obtener el diseño del planificador ejecutivo cíclico del sistema. Ésta se realizó mediante el uso de la herramienta *CICLIC*, la determinó seis

posibles diseños factibles. Se tomó el de ciclos secundarios igual a 100 ms, que además, fue el recomendado por el programa. El resumen de la planificación se muestra a continuación:

Duración del hiperperíodo (s) = 1
Duración de los ciclos secundarios (ms) = 100
Tiempo ocioso (ms) = 975
Instancias totales = 41
Utilización del procesador = 2,5%

Para la sincronización de los ciclos secundarios, se utilizó el temporizador *Timer2* del microcontrolador, configurado de manera tal que generara una llamada a la función de planificación cada 100 ms. Para ello se usó la librería *MsTimer2.h* de Arduino (Arduino, 2014). En la Fig. 10, se pueden apreciar los resultados del análisis digital realizado en el simulador, incorporando el planificador ejecutivo cíclico.

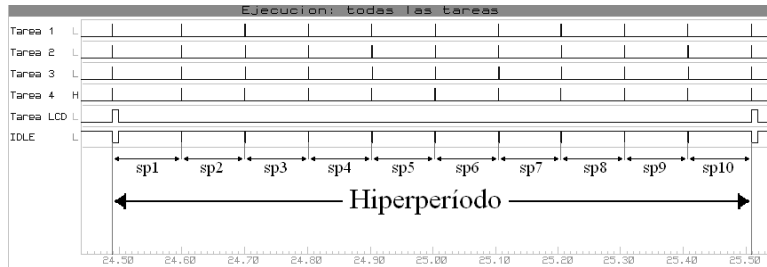


Figura 10. Ejecución Tareas.

Codificación

Después de haber confeccionado el plan de ejecución de las tareas, se procedió al código detallado de los módulos que conforman el programa que fue almacenado en el microcontrolador, usando como estructura principal las directrices propuestas en *mikro_STR*.

Respuesta

En cuanto al análisis de la respuesta del sistema a lazo cerrado, se observa que los controladores PID responden de manera muy eficiente, siguiendo las referencias impuestas. Las cuatro temperaturas lograron posicionarse en el valor deseado transcurrido cierto tiempo, a pesar de tener como entrada fija un escalón de 2 V que lleva la temperatura de los hornos a 276 °C de Alfonsi A., Alfonsi A.R. y Yáñez R. Estructura Global de Software para Aplicaciones de Control Empotrado con Restricciones Temporales en Microcontroladores con Recursos Limitados. Informe Final Proyecto de Investigación CI-UDO. Universidad de Oriente, Barcelona, 2015.

forma natural (Fig. 11). Se puede decir entonces que además de cumplir con las restricciones temporales, se lograron concretar todas las acciones de control del sistema.

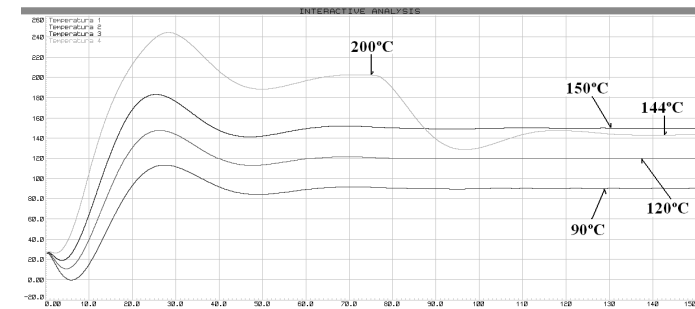


Figura 11. Respuesta lazo cerrado del sistema de calefacción con el ejecutivo cíclico.

4. Especificación *mikro_STR* con Extensión *UML*

Como segundo acercamiento se introduce la EGS *mikro_STR* con extensión *UML*. Se sigue aplicando el proceso iterativo (Fig. 1), ahora introduciendo el desarrollo de software con extensiones de tiempo real del *UML* (Herrera, 2011; Douglass, 2014).

La idea en esta nueva etapa es que el desarrollador, pueda tener acceso sin complicaciones metodológicas, por ser un único curso a la herramienta *UML*, pudiendo adaptar los conceptos impartidos en *mikro_STR*, al diseñar una variedad de sistemas de tiempo real, a partir de pequeños sistemas de microcontroladores de bajos requerimientos, extendiendo a grandes sistemas en red de varios SE. Así podrá prepararse para experiencias complejas como la utilización de sistemas operativos de tiempo real como MaRTE u otro, tomando ventaja de los métodos modernos de ingeniería basados en modelos y normas de la industria correspondientes para superar estas limitaciones (Selic y Gerad, 2013; Douglass, 2014).

4.1 Notación Gráfica

Es importante representar las características del sistema apoyados en diagramas *UML* con las extensiones para sistemas de tiempo real (Herrera, 2011; Douglass, 2014). El *UML* fue desarrollado en respuesta a la necesidad de un lenguaje de modelado de objetos estandarizados. Puede ser adaptado para diseñar una variedad de sistemas de tiempo real, a partir de pequeños

Alfonsi A., Alfonsi A.R. y Yáñez R. Estructura Global de Software para Aplicaciones de Control Empotrado con Restricciones Temporales en Microcontroladores con Recursos Limitados. Informe Final Proyecto de Investigación CI-UDO. Universidad de Oriente, Barcelona, 2015.