Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

AUTOMATIZACIÓN DE UN PANEL SOLAR

Trabajo Final de Grado

Grado en Ingeniería Eléctrica

ESCUELA TÉCNICA SUPERIOR DEL DISEÑO
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Autor: Benjamín Adán Cháfer Ferrando
Director: Francisco Rodríguez Ballester

05/12/2016

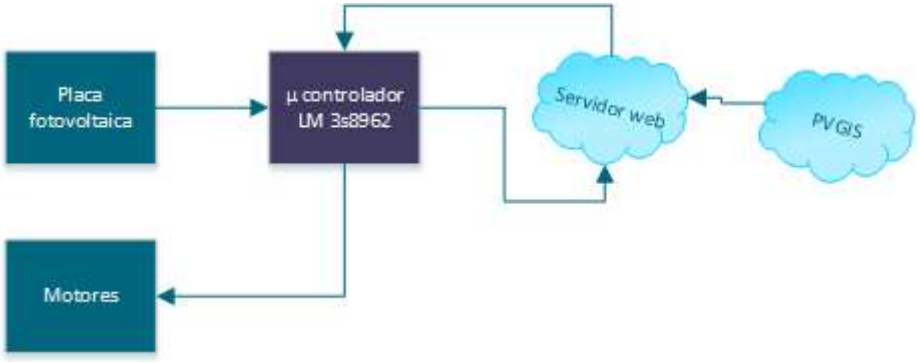
Índice

Capítulo 1 – Introducción 2
1.1. Objeto del proyecto 2
1.2. Antecedentes 2
1.3. Justificación del proyecto 2
1.4. Especificación del encargo 2
1.4.1. Problema 2
1.4.2. Posibles soluciones 2
Capítulo 2 – Descripción del hardware 4
2.1. Esquema de bloques 4
2.2. Conversor Analógico-Digital (ADC) 5
2.3. Trasmisores y receptores asíncronos universales (UART) 7
2.3.1. Breve introducción a la comunicación RS-232 7
2.3.2. Configuración del proyecto 8
2.4. Temporizadores de propósito general (GPTM) 9
2.5. Modulación por ancho de pulso (PWM) 11
2.6. Entradas y salidas de propósito general (GPIO) 12
2.7. Módulo externo GSM 13
Capítulo 3 – Descripción del software 14
3.1. Esquema de bloques 14
3.2. Estructura modular 14
3.2.1. ADC 15
3.2.2. Entradas y salidas 16
3.2.3. Depuración 18
3.2.4. GSM 20
3.2.5. Muestreo 22
3.2.6. PWM 23
3.2.7. Programa principal 23
Capítulo 4 – Herramientas utilizadas 24
4.1. Entorno de programación 24
4.2. Emulador de terminal 24
Capítulo 5 – Características técnicas 26
5.1. Características del microcontrolador LM3S8962 26
Capítulo 6 – Conclusiones 29
Capítulo 7 – Presupuesto 30
Capítulo 8 – Bibliografía 31
8.1. Bibliografía 31

Capítulo 1 – Introducción

1.1. Objeto del proyecto

El objetivo de este proyecto es la creación de un demostrador de viabilidad a partir de la programación de una placa electrónica de Texas Instruments modelo LM3S8962 [1] para que controle, la orientación de una placa fotovoltaica, mediante un conjunto de motores y la eficiencia de la placa mediante medidas periódicas. La placa fotovoltaica será reorientada según el ángulo óptimo ofrecido por la base de datos de la página web PVGIS que será transmitido a través de un servidor web desarrollado de forma externa.



1.2. Antecedentes

Actualmente se encuentran en el mercado gran cantidad de mecanismos similares a los que a continuación se justifican en este documento, no obstante, no siempre consiguen la máxima eficiencia pese a la mayor inversión económica.

1.3. Justificación del proyecto

Debido a la rotación de la Tierra, el ángulo de recepción de la radiación del Sol no es constante. Esto supone un inconveniente a la hora de realizar una instalación de placas solares fotovoltaicas, pues el rendimiento de estas disminuye en gran medida si no recibe la radiación solar en su ángulo óptimo.

1.4. Especificación del encargo

1.4.1. Problema

El problema radica en la importancia de mantener el ángulo óptimo de recepción de la radiación solar para aumentar el rendimiento de las instalaciones fotovoltaicas.

1.4.2. Posibles soluciones

Durante el planteamiento del proyecto se propusieron diferentes ideas para solucionar el problema anteriormente mencionado.

Una de ellas fue la implementación de un módulo de geoposicionamiento de forma que, dependiendo del lugar de la instalación, el programa utilizara los motores para ubicar las placas en su ángulo óptimo en tiempo real. Esta opción fue descartada tanto por su complejidad como por su incapacidad de variar el comportamiento si no se detecta suficiente luz solar.

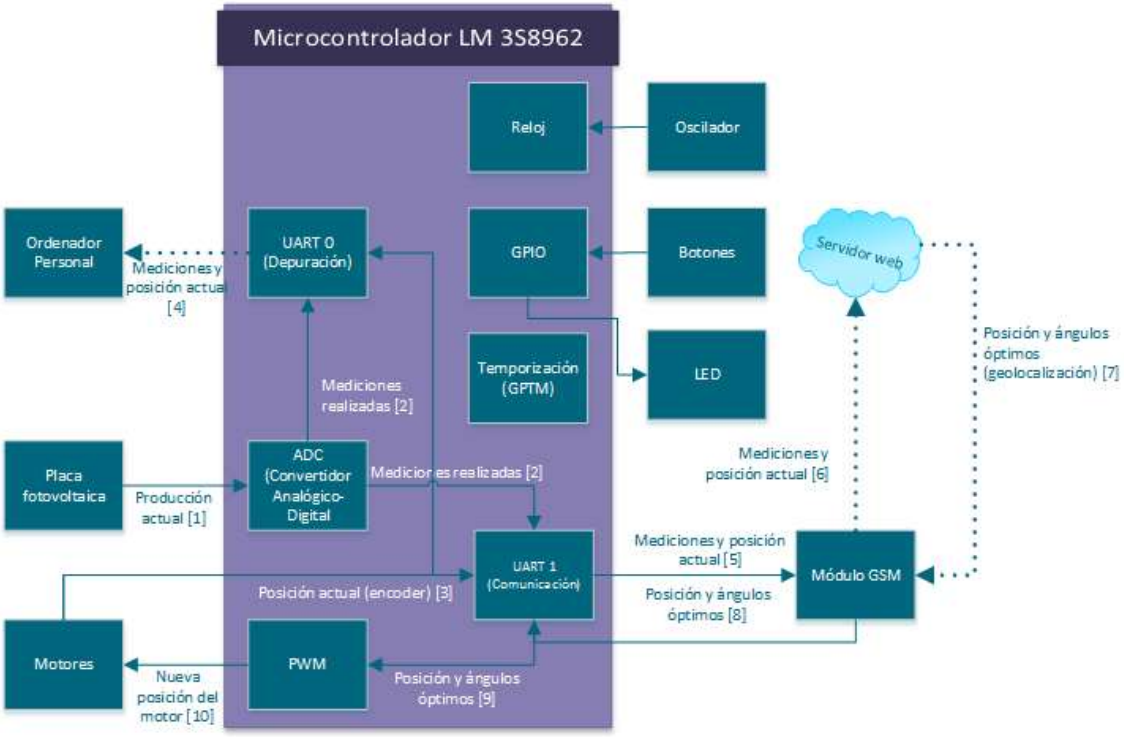
Otra opción era la inclusión de sensores lumínicos para conducir las placas mediante los motores hacia una fuente de luz, concretamente la luz solar. En este caso, se usarían luxómetros instalados a cada extremo de la placa, conectados a un comparador digital para calcular qué posición tomarán los motores.

La anterior propuesta fue desechada en pos de un sistema de monitorización del potencial generado por cada placa fotovoltaica. El programa envía cada cierto tiempo el rendimiento de la placa a un servidor web, el cual también envía instrucciones al sistema para movilizar la placa hacia un ángulo óptimo.

Microcontrolador LM3s8962

Capítulo 2 – Descripción del hardware

2.1. Esquema de bloques



En este esquema podemos observar la forma en que el hardware del proyecto está relacionado entre sí y su función dentro del mismo.

Concretamente, la placa fotovoltaica tendrá los bornes siempre conectados a los pines de la entrada del ADC de forma que este siempre podrá tomar medidas (1). Una vez tomadas las medidas de la placa, tanto el ADC como los encoders de los motores, enviarán las mediciones y la posición actual a cada una de las unidades UART (2) y (3) para que, en caso de la unidad de depuración, sean mostrados en una línea de comandos a un ordenador personal a modo de verificación del funcionamiento (4) y, en el caso de la unidad de comunicación, envíe los datos vía GSM, a través de un módulo externo (5) conectado vía puerto serie al microcontrolador, a un servidor web (6). Desde el servidor se determinará la localización del sistema mediante geolocalización vía IP y, en consecuencia, este devolverá, según la hora del día, una nueva posición óptima para la placa fotovoltaica, que se recibirá en el sistema mediante GSM nuevamente (7). La unidad UART de comunicación se encarga de procesar la información recibida por el módulo GSM externo (8) y entonces podrá manejar el sistema de motores mediante PWM (9) para posicionarlos en su nueva posición (10).

2.2. Conversor Analógico-Digital (ADC)

El ADC es un elemento del hardware del microcontrolador que, como indica su nombre, es capaz de tomar señales analógicas y transformarlas en digitales. De esta forma pueden ser interpretadas por la programación del microcontrolador.

Nuestro modelo de microcontrolador posee cuatro entradas para medir señales analógicas, más un sensor de temperatura. A su vez, cada señal tiene un secuenciador con las siguientes muestras:

Secuenciador	Número de muestras	Longitud de la cola de datos
0	8	8
1	4	4
2	4	4
3	1	1

El número de muestras indica la longitud de la secuencia de medida. Esto es, un orden de mediciones donde se indica para cada medición si ésta se trata del último elemento, si va a usar el sensor de temperatura interno o si va a disparar una interrupción.

Para configurar la secuencia, disponemos de seis bits por elemento de la forma que se detalla en la siguiente tabla:

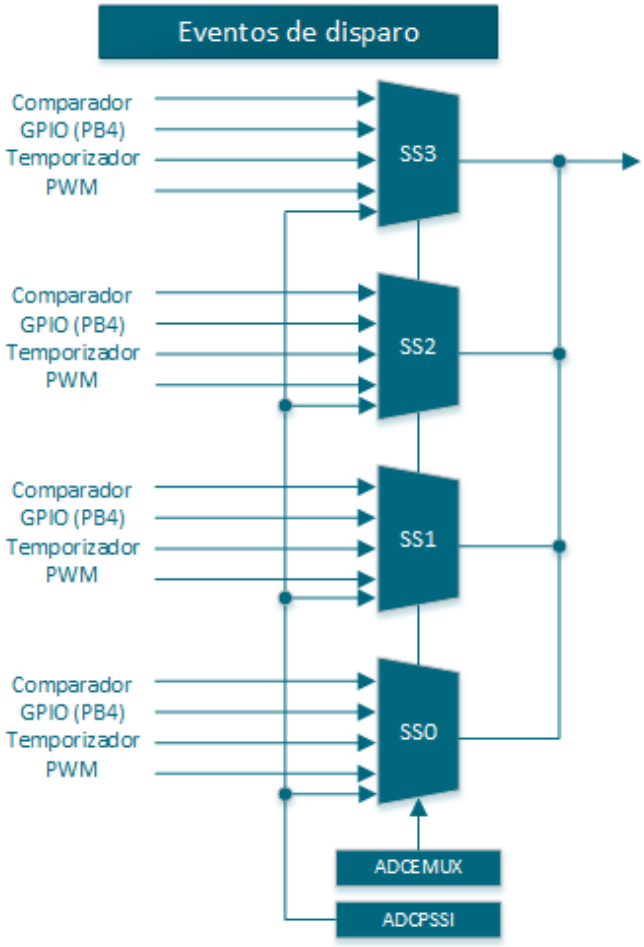
Bits de configuración para el elemento 'z' del secuenciador (SS) 'n':

Registro	Bit	Descripción
ADCSSMUXn	MUXz	Selección del pin de entrada analógica (2 bits).
ADCSSCTLn	Dz	Al tener como valor 0x1 indica que la medida se tomará de forma diferencial. No es compatible con el sensor de temperatura interno.
	ENDz	Cuando el valor es 0x1 indica que es el último elemento de la secuencia.
	IEz	Si el valor es 0x1 activa una interrupción al tomar la medida actual.
	TSz	Cuando toma el valor 0x1 selecciona como entrada el sensor de temperatura interno.

El ADC permite realizar una interrupción al tomar una medida. Como ya veremos más adelante, esto nos servirá para contar el número de medidas tomadas por el ADC.

El ADC también cuenta con la posibilidad de que un temporizador sea el encargado de disparar una medida, de forma que se pueda realizar un muestreo

periódico. Esto consiste en que cada vez que un temporizador concreto termina su cuenta atrás, el ADC toma una medida y la almacena en una cola de datos.



Aquí podemos observar la variedad de eventos que pueden disparar una medida de un secuenciador (SS). Esto se configura mediante el registro ADCEMUX, que engloba la configuración de los cuatro secuenciadores. Concretamente, dispone de cuatro campos de configuración, de cuatro bits para cada uno: EM0, EM1, EM2 y EM3. Cada uno de ellos puede tener los siguientes valores:

- Si es 0x0, el disparo lo lleva a cabo el controlador.
- Si es 0x1, el disparo lo ejecuta el comparador analógico.
- Si es 0x4, el disparo es realizado por una entrada del GPIO (Pin B4).
- Cuando es 0x5, el disparo lo causa un temporizador (el cual debe estar configurado para tal acción).
- Si es 0x6, el disparo corre a cargo del módulo PWM0, cuando está configurado para tal.
- Si el valor es 0x7, el disparo corre a cargo del módulo PWM1, cuando está configurado para tal.
- Cuando es 0x8, el disparo corre a cargo del módulo PWM2, cuando está configurado para tal.
- Si es 0xF, el disparo es continuo.

Además, observamos la capacidad del registro ADCPSSI para ejecutar el disparo de forma forzada, mediante software.

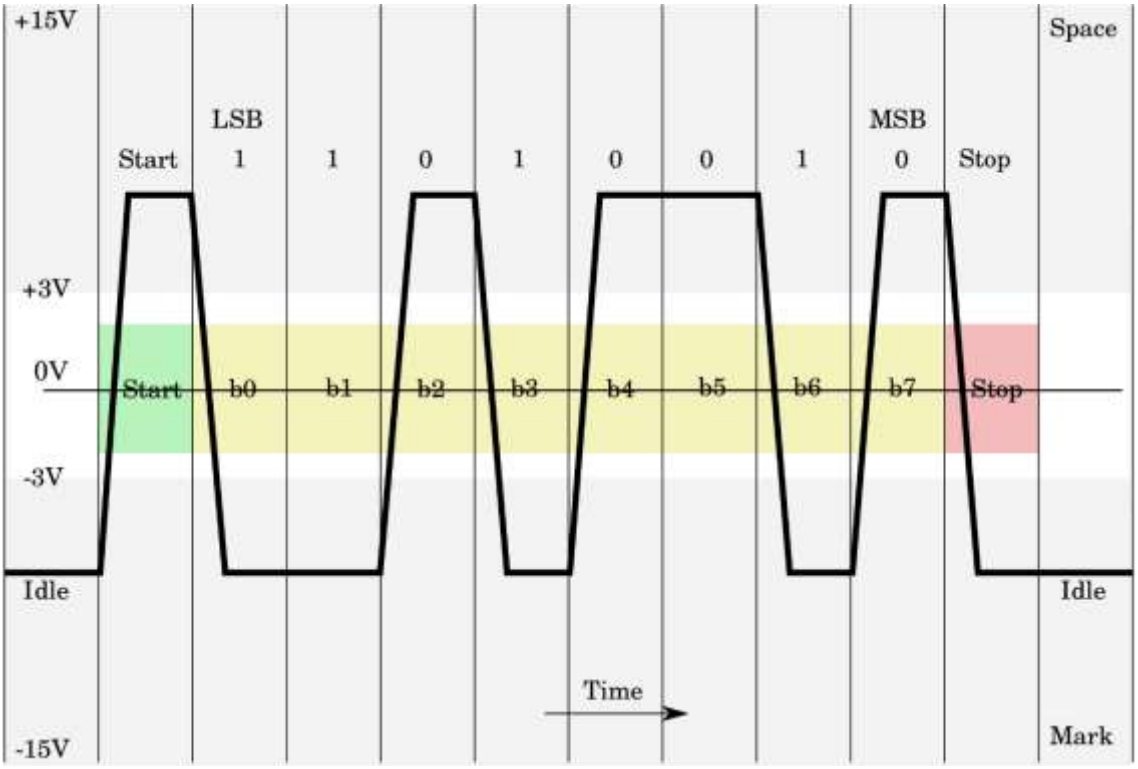
2.3. Trasmisores y receptores asíncronos universales (UART)

Este microcontrolador cuenta con dos unidades UART, las cuales sirven para realizar comunicaciones vía puerto serie (RS-232). Las comunicaciones RS-232 son un estándar para el intercambio de datos binarios.

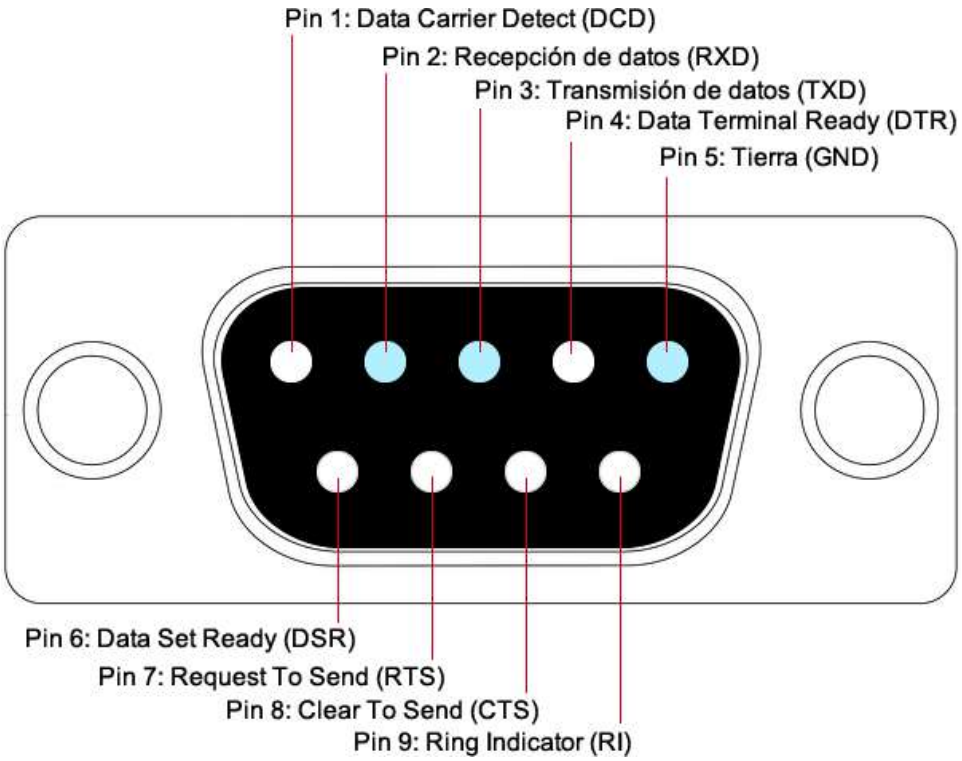
2.3.1. Breve introducción a la comunicación RS-232

El estándar R-232 define unos niveles de voltaje que corresponden al *cero lógico* y al *uno lógico* a la hora de ser transmitidos. Estos niveles de voltaje siempre serán, para un flanco negativo (uno lógico), de entre -15 y -3 V, y, para un flanco positivo (cero lógico) de entre 3 y 15 V. Durante la transmisión, el cero lógico se conoce como *espacio* y el uno lógico como *marca*.

A continuación, se muestra una gráfica [6] que ejemplifica la transmisión de datos mediante RS-232:



Como podemos ver, la transmisión se efectúa cuando el transmisor se encuentra en estado *Idle* lo cual significa que está inactivo, o sea, sin ninguna orden. Una vez empieza la transmisión, se envía el *bit de Start*, con el cual se señaliza el principio del mensaje. A continuación, se envía el primer bit del mensaje (LSB, bit menos significativo). Uno a uno, se van enviando los siguientes bits, hasta enviar el último bit del mensaje (MSB, bit más significativo). Para finalizar la transmisión se envía el *bit de Stop* y el transmisor vuelve al estado inicial *Idle*.



En la imagen anterior podemos observar qué función tiene cada pin en el estándar RS-232, no obstante, actualmente solo se suelen utilizar los pines 2, 3 y 5, que son los que intervienen en la transmisión de datos.

En nuestro microcontrolador las señales son las siguientes:

Dato	Voltaje
0 (espacio)	0
1 (marca)	3.3

2.3.2. Configuración del proyecto

En nuestro proyecto asignaremos una UART al proceso de depuración y otra a la comunicación vía GSM.

Para realizar la conexión ambos terminales deben estar configurados a la misma velocidad de transmisión (baudrate). En nuestro proyecto configuraremos la transmisión a 9600 bps con un reloj de 50 MHz de frecuencia y, para ello, usaremos los registros UARTIBRD y UARTFBRD. Nuestro microcontrolador utiliza estos dos registros para generar el baudrate.

En el registro de 16 bits UARTIBRD insertaremos la parte entera del baudrate, que se calcula de la siguiente manera:

BAUDRATE = 9600, SYSCLK = 50 MHz

$$Baudrate_{int} = Int\left[\frac{SYSCLK}{16 \cdot BAUDRATE}\right] = Int[325{,}52] = 325$$

En el registro de 16 bits UARTIBRD insertaremos la parte fraccionaria del baudrate, que se calcula de la siguiente manera:

BAUDRATE = 9600, SYSCLK = 50 MHz

$$Baudrate_{frac} = Int\left[Frac\left[\frac{SYSCLK}{16 \cdot BAUDRATE}\right] \cdot 64 + 0.5\right] = Int[33.83] = 33$$

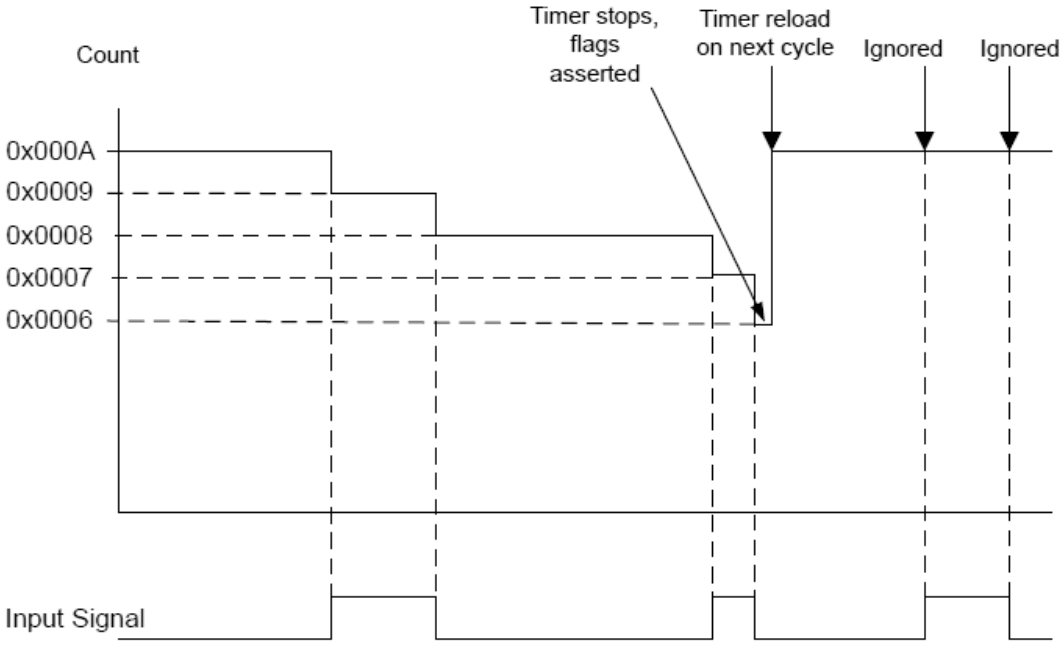
2.4. Temporizadores de propósito general (GPTM)

Como extra añadido por el fabricante, este microcontrolador dispone de cuatro bloques de temporización. Cada bloque está dividido en dos subtemporizadores llamados A y B. Estos subtemporizadores pueden funcionar como dos temporizadores de 16 bits (dependiendo de la frecuencia de entrada, hasta 65.000 veces el periodo de entrada) o como uno de 32 bits (4.294.967.296 veces el periodo de entrada).

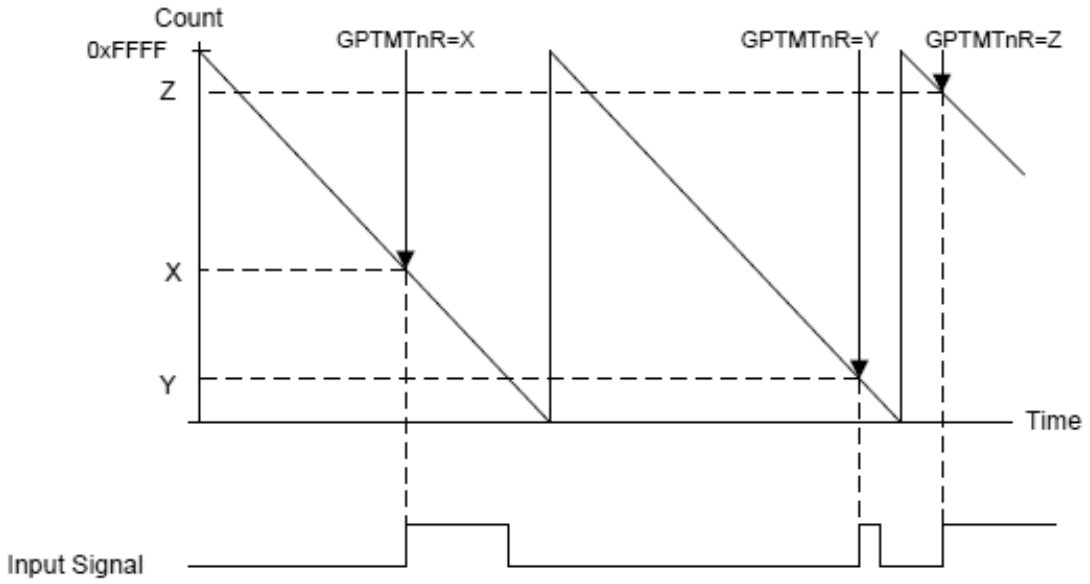
Además, cada subtemporizador en modo de 16 bits tiene una preescala de 8 bits, lo cual significa que, por cada división del temporizador, tiene 256 divisiones más como mejora de la precisión.

Por otro lado, los temporizadores tienen diferentes tipos de cuenta atrás:

- *One-Shot*: en este modo, cuando el temporizador llegue al final de su cuenta atrás, este se detendrá.



- *Free-running* o periódico: A diferencia del modo anterior, cuando acabe la cuenta atrás, automáticamente volverá a su valor inicial y seguirá con la cuenta atrás.



- RTC: El temporizador emula un reloj real. (Solo en modo de 32 bits)
- PWM: En este modo, el temporizador crea una señal PWM con un periodo y amplitud dado. (Solo en modo de 16 bits). Más adelante ampliaremos este modo, pues se utilizará para el movimiento de los motores.

A la hora de configurar un temporizador, deberemos usar los siguientes bits:

Registro	Bit	Descripción
GPTMCFG	GPTMCFG	Indica el modo del temporizador (3 bits): ▪ Si el valor es 0x0, el temporizador será de 32 bits. ▪ Si el valor es 0x1, el temporizador será de 32 bits con RTC. ▪ Si el valor es 0x4, el temporizador será de 16 bits.
GPTMTAMR	TAAMS	Controla el modo alternativo del temporizador A: ▪ Si el valor es 0x0, el temporizador funciona en modo captura de eventos. ▪ Si por el contrario toma el valor 0x1, generará una señal PWM.
	TACMR	Indica el modo de captura del temporizador A: ▪ Si el valor es 0x0 realizará captura de flancos. ▪ Si el valor es 0x1, se realizará captura de tiempo.
	TAMR	Selecciona el tipo de cuenta atrás del temporizador A (2 bits): ▪ Si el valor es 0x1, la cuenta atrás será One-Shot.

		▪ Cuando el valor es 0x2, la cuenta atrás será Free-Running. ▪ Cuando el valor es 0x3, activa el modo de captura configurado en TACMR.
GPTMTBMR	TBAMS	Controla el modo alternativo del temporizador B: ▪ Si el valor es 0x0, el temporizador funciona en modo captura de eventos. ▪ Si por el contrario toma el valor 0x1, generará una señal PWM.
	TBCMR	Indica el modo de captura del temporizador B: ▪ Si el valor es 0x0 realizará captura de flancos. ▪ Si el valor es 0x1, se realizará captura de tiempo.
	TBMR	Selecciona el tipo de cuenta atrás del temporizador B (2 bits): ▪ Si el valor es 0x1, la cuenta atrás será One-Shot. ▪ Cuando el valor es 0x2, la cuenta atrás será Free-Running. ▪ Cuando el valor es 0x3, activa el modo de captura configurado en TBCMR.

Si el modo elegido en GPTMCFG es el de 32 bits, únicamente se tiene en cuenta el registro GPTMTAMR.

2.5. Modulación por ancho de pulso (PWM)

La modulación por ancho de pulso (PWM) es una técnica de control que se puede usar en máquinas eléctricas. Consiste en el uso de una señal digital de periodo fijo y ratio variable.

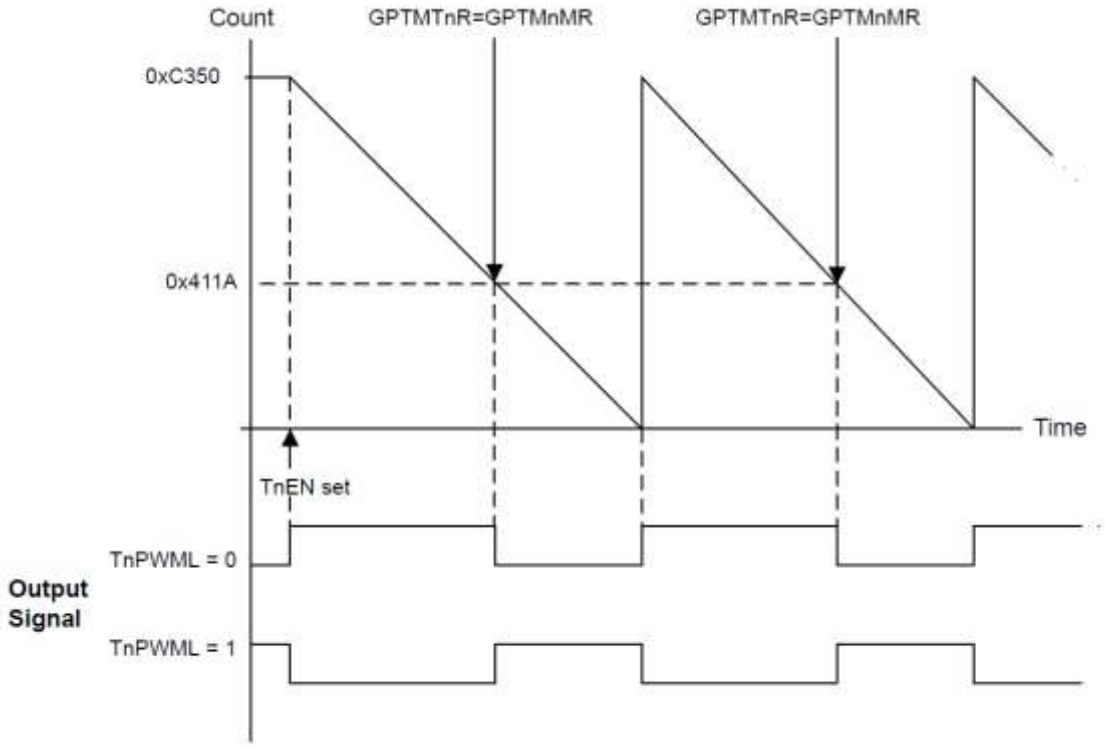
Para la modulación por ancho de pulso, usaremos dos subtemporizadores del mismo temporizador. Para ello, deberemos escribir los bits TAAMS y TBAMS para activar el modo PWM en ambos temporizadores.

Para definir el periodo de la señal PWM escribiremos el valor deseado en los registros de 16 bits GPTMTAIRL y GPTMTBILR los cuales contienen el valor del periodo de la señal PWM para los temporizadores A y B respectivamente.

A continuación, para determinar el ratio (tiempo en OFF de la señal PWM en porcentaje), introduciremos el valor deseado en los campos TAMRL del registro GPTMTAMATCHR y TBMRL del registro GPTMTBMATCHR.

Al utilizar temporizadores como generadores PWM, tenemos un inconveniente añadido: Los periodos en OFF y en ON están invertidos, ya que el temporizador realiza una cuenta atrás. Para poder invertir la polaridad de los periodos,

realizaremos una escritura en los bits TAPWML y TBPWML del registro GTPMCTL.



En la imagen anterior podemos observar la señal PWM generada por el temporizador, con y sin la inversión de la señal producida por el bit TnPWML (donde 'n' significa el subtemporizador A o B).

2.6. Entradas y salidas de propósito general (GPIO)

El microcontrolador LM 3s8962 cuenta con cierta cantidad de entradas y salidas, entre las cuales se encuentran un conjunto de botones y pilotos LED programables.

Tanto las entradas y salidas normales (patillas) como los botones y los LEDs, pueden configurarse cambiando una serie de bits del registro correspondiente al puerto en el que se encuentra. Por ejemplo, si vamos a configurar el LED STATUS que se encuentra en el Puerto F, deberemos dirigirnos al registro GPIO_PORTF y editar los bits convenientes para la configuración que se va a realizar.

A continuación, se muestra una tabla con las configuraciones más comunes en el proyecto para estas entradas y salidas:

Configuración	Valores de los bits del registro GPIO									
	AFSEL	DIR	ODR	DEN	PUR	PDR	DR2R	DR4R	DR8R	SLR
Entrada digital	0	0	0	1	?	?	X	X	X	X
Salida digital	0	1	0	1	?	?	?	?	?	?

Entrada/salida digital (UART)	1	X	0	1	?	?	?	?	?	?
Salida digital (Temporizador PWM)	1	X	0	1	?	?	?	?	?	?
Entrada analógica	0	0	0	0	0	0	X	X	X	X

Leyenda:

X – No influye en la configuración

? – Puede ser 1 o 0 dependiendo de la configuración

2.7. Módulo externo GSM

El módulo externo GSM elegido para este proyecto es el modelo GL865-Quad [4] del fabricante Telit.

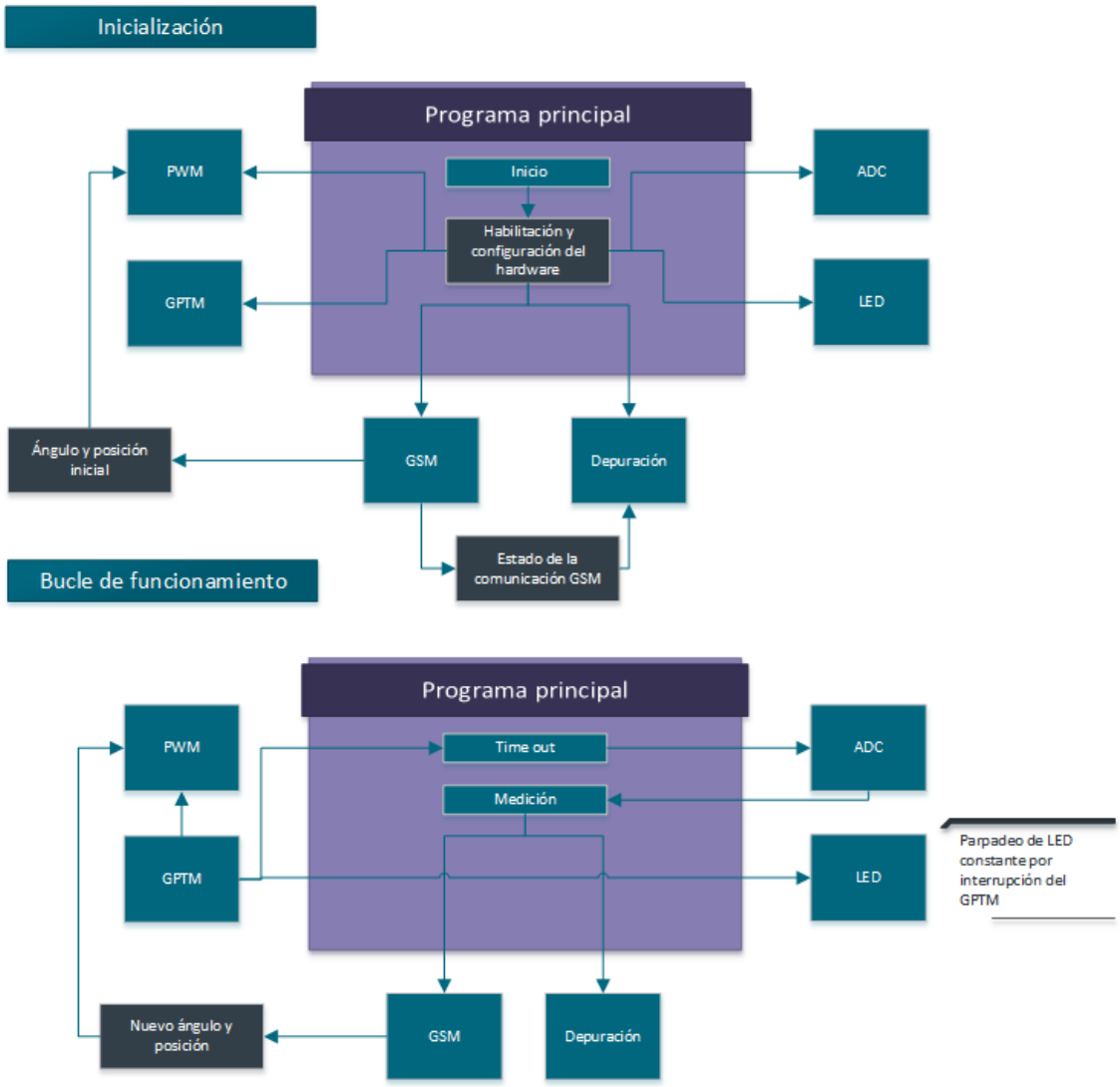
Se trata de un módulo que realiza la comunicación con el microcontrolador vía puerto serie con el protocolo RS-232. Para poder usar esta comunicación, dedicaremos una UART a esta tarea.

Foto extraída de adaptivemodules.co.uk

Este módulo trabaja con comandos AT [5], los cuales son específicos para realizar tareas de red o relativas al módulo externo GSM. Estos comandos se envían mediante la UART (en formato texto) y, en su caso, el módulo también enviará su respuesta.

Capítulo 3 – Descripción del software

3.1. Esquema de bloques



3.2. Estructura modular

Este proyecto se ha estructurado en distintos módulos de software, los cuales se dedican cada uno a un grupo de tareas.

Conforme se puede ver en el esquema, el programa principal se encarga de realizar las pertinentes configuraciones en los bits de cada elemento de hardware para poder realizar las funciones del proyecto.

A continuación, se explicará detalladamente las instrucciones que contiene cada módulo de software.

3.2.1. ADC

El ADC es el elemento de hardware que se ocupa de realizar las mediciones en la placa fotovoltaica. Para que realice dicha función hemos programado unas funciones que permiten inicializar y activar un secuenciador.

La inicialización se realiza mediante la función `void adcInit(enum adc_oversampling oversampling, uint32_t cpuHz)`, en primer lugar se activa la señal de reloj hacia el hardware y se desactivan los secuenciadores para evitar problemas durante la escritura de bits. A continuación, se configura el número de tomas de datos por muestra, esto permite realizar una media por hardware de forma que el dato que se almacena definitivamente es más preciso que si se tomara una única medida.

```
void adcInit(enum adc_oversampling oversampling,
uint32_t cpuHz)
```

Una vez inicializado el hardware del ADC, se procede a inicializar los secuenciadores con la función `void adcSeqInit(enum adc_sequencer seq, enum adc_trigger trigger, uint8_t temp_sensor, uint8_t n_items_seq, uint32_t samplingRate_us)`. Esto implica usar los seis bits de configuración para realizar la configuración del número de elementos de la secuencia (número de muestras para almacenar en la cola), la señal de entrada, la señal de disparo y la velocidad de muestreo.

```
void adcSeqInit(enum adc_sequencer seq, enum
adc_trigger trigger, uint8_t temp_sensor, uint8_t
n_items_seq, uint32_t samplingRate_us)
```

Finalmente, después de inicializar el hardware y los secuenciadores, se puede activar la medida de cada secuenciador con una sencilla función llamada `void adcSeqEnable(enum adc_sequencer seq)` que activa las interrupciones del microcontrolador para un secuenciador dado. Esto permite, mediante una lectura del campo SSFIFOn (donde n es el secuenciador utilizado) almacenar el valor de la última muestra captada por el ADC al realizarse esta.

```
void adcSeqEnable(enum adc_sequencer seq)
```

Para cada secuenciador, se ha programado una función que se ejecuta mediante una interrupción del secuenciador. En ella se ejecuta el almacenamiento de las medidas, como hemos explicado anteriormente, y un mecanismo de seguimiento por depuración.

Este mecanismo se ocupa de contar el número de medidas realizadas y de medidas procesadas para poder llevar un seguimiento de los errores surgidos. Este balance se mostrará, mediante las funciones de depuración, en el PC del personal técnico.

```
void adcSeqn_isr(void)
```

El módulo tiene también una función para desactivar un secuenciador dado desactivando las interrupciones y otra que genera una medida de forma forzada mediante software.

```
void adcSeqDisable(enum adc_sequencer seq)
```

```
void adcSeqSwTrigger(enum adc_sequencer seq)
```

3.2.2. Entradas y salidas

En este módulo se configuran las entradas y salidas pertinentes para poder interactuar con el exterior del microcontrolador.

En cuanto a salidas, usaremos tres en este proyecto: una para el LED incorporado y otras dos para cada motor controlado por PWM.

Para la salida del LED se ha implementado la función `void initLed(void)`, que se encarga de la inicialización del pin del LED, esto es, activar la señal de reloj para el puerto de este pin y configurar el pin como salida digital. Finalmente, apaga la luz.

```
void initLed(void)
```

Siguiendo con el LED, tenemos las funciones `void setLed(void)` y `void resetLed(void)` que se usan para encender y apagar el LED respectivamente. Cada una asigna un valor a la variable `stateLed`, que utilizaremos para comprobar el estado del LED.

```
void setLed(void)
```

```
void resetLed(void)
```

`uint8_t getLed(void)` es la función que devuelve el estado del LED (variable `stateLed`): cuando devuelve el valor 1, el LED está encendido y cuando devuelve el valor 0, está apagado.

```
uint8_t getLed(void)
```

La función que controla el apagado o encendido del LED es `void switchLed(void)` y comprueba el valor devuelto por `uint8_t getLed(void)` y dependiendo del estado del LED, enciende o apaga el LED, es decir, si está apagado, se enciende y viceversa.

```
void switchLed(void)
```

Una vez terminadas las funciones específicas del LED, es el turno de las funciones genéricas de los pines:

En primer lugar, tenemos `void init_GPIO_port(enum port)`, que es la función que habilita el puerto del pin a configurar. Esta función se encarga de proporcionar señal de reloj al puerto del pin.

```
void init_GPIO_port(enum port)
```

Una vez inicializado el puerto, tenemos la posibilidad de activar o desactivar las interrupciones ocasionadas por un pin de entrada

```
void GPIO_IntEn(int pin, enum port)
```

```
void GPIO_IntDis(int pin, enum port)
```

A continuación, deberemos configurar el pin, para ello se ha creado una función para cada configuración, según se puede ver en 2.1. Entradas y Salidas de Propósito General.

Para configurar un pin como salida digital tenemos la función `void GPIO_setPinAsDigitalOutput(int pin, enum port)`, la cual recibe como argumentos el pin y el puerto para realizar las operaciones necesarias para configurar el pin como salida digital según 2.1. Entradas y Salidas de Propósito General.

Si por el contrario, queremos configurar un pin como salida digital PWM, la función a utilizar será `void GPIO_setPinAsPWMOutput(int pin, enum port)`.

De la misma forma, tenemos todas las configuraciones disponibles:

```
void GPIO_setPinAsDigitalOutput(int pin, enum port)
```

```
void GPIO_setPinAsPWMOutput(int pin, enum port)
```

```
void GPIO_setPinAsDigitalInput(int pin, enum port)
```

```
void GPIO_setPinAsAnalogInput(int pin, enum port)
```

Una vez hemos configurado todas las entradas y las salidas, tenemos la posibilidad de asociar interrupciones a cada entrada. Para ello, insertaremos la función `void GPIO_ISR(void)` que se ejecutará con cualquier interrupción de una entrada y según la entrada que provoca la interrupción, se ejecutará una función u otra.

```
void GPIO_ISR(void)
```

En nuestro proyecto, se establecerá la siguiente configuración de entradas y salidas:

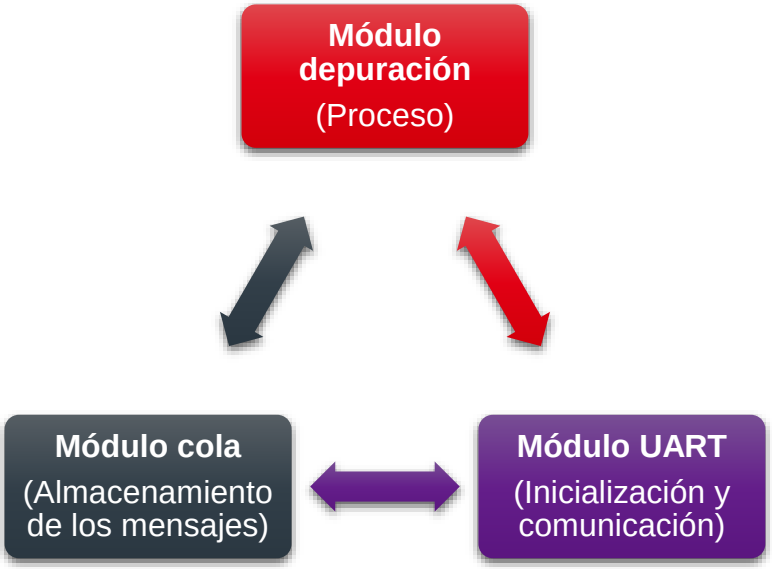
- Como entradas digitales, disponemos de los encoders de los motores y los finales de carrera que evitarán una posición indeseada.
- Como entradas analógicas, las salidas de la placa para poder realizar las mediciones.

- Como salidas digitales de PWM, las señales PWM de los temporizadores en modo PWM.
- Como salida LED, el pin del LED STATUS.

3.2.3. Depuración

El sistema de depuración se encarga de mostrar paso a paso el funcionamiento del programa mediante la comunicación vía puerto serie. Este proceso se ha incluido para poder detectar errores y defectos en el programa.

En este proyecto se ha decidido incluir tres módulos de código para realizar esta tarea. Esto es debido a que tenemos una parte del código que depende del hardware, correspondiente a la inicialización y la comunicación (Módulo UART), otra que es independiente, ya que se encarga de realizar el procesamiento de los mensajes (Módulo Depuración) y otra, auxiliar, que almacena los mensajes en la cola (Módulo Cola) para ser tratados por el resto de la programación.



Módulo UART

Se ocupa de las comunicaciones puerto serie a bajo nivel. Este módulo además contiene la función `void debug_uart_init(which_uart_t which_uart)`, necesaria para inicializar la Unidad UART que se va a usar para realizar la depuración.

Para ello, se deberán desactivar las interrupciones de transmisión, para que no exista inconsistencia en los mensajes, en caso de que se esté ejecutando la transmisión. A continuación, se deberá desactivar la unidad UART, siguiendo el manual del fabricante y esperar a que termine la transmisión en curso, si la hubiera.

El siguiente paso a realizar es la configuración de la unidad UART para la depuración. Se realizarán las operaciones necesarias para establecer la siguiente configuración:

- Activar recepción y transmisión

- Desactivar la comunicación por infrarrojos (IrDA)
- No evitar Break
- Desactivar paridad
- Establecer un bit de stop
- Parte entera del Baud-rate establecida a 325 (véase 2.1.2. Configuración del proyecto)
- Parte fraccionaria del Baud-rate establecida a 33 (véase 2.1.2. Configuración del proyecto)
- Activar unidad UART
- Activar interrupciones por recepción y transmisión
- Activar interrupciones de la unidad UART en el NVIC

```
void debug_uart_init(which_uart_t which_uart)
```

La siguiente función es `void debug_uart_send (uint8_t data)`, la cual recibe un carácter que es escrito en el campo DATA del registro UARTDR. Acto seguido, el microcontrolador transmite ese carácter hacia el terminal.

```
void debug_uart_send (uint8_t data)
```

Para activar las salidas de la unidad UART, deberemos configurar los pines 0 y 1 del puerto A del GPIO como entrada y salida UART (véase 2.1. Entradas y Salidas de Propósito General). Para llevar a cabo esta configuración se ha implementado la función `void _uart_enable_hw(which_uart_t which_uart)`.

```
void _uart_enable_hw(which_uart_t which_uart)
```

Finalmente, tenemos las funciones `void _uart_enable_interrupts_nvic (which_uart_t which_uart)` y `void _uart_disable_interrupts_nvic (which_uart_t which_uart)` que activan y desactivan las interrupciones de la unidad UART en el NVIC respectivamente. Cabe decir que estas funciones son utilizadas en varias ocasiones por los demás módulos.

```
void _uart_enable_interrupts_nvic (which_uart_t which_uart)
```

```
void _uart_disable_interrupts_nvic (which_uart_t which_uart)
```

Módulo Depuración

El Módulo Depuración contiene la función `void DebugInit(void)`, la cual prepara la cola que se va a utilizar en la depuración, inicializa la unidad UART de depuración, establece el estado de la transmisión en IDLE (libre) y guarda el siguiente mensaje en la cola, esperando que sea mostrado:

```
Mensaje desde Debug ... inicializando
```

```
void DebugInit(void)
```

Este módulo también contiene la función `void DebugNoTask(void)`, la cual extrae un mensaje de la cola, lo divide en caracteres y, mediante el Módulo UART, envía hacia el terminal dichos caracteres utilizando las interrupciones del puerto serie. Acto seguido establece el estado de la transmisión en BUSY (ocupado).

```
void DebugNoTask(void)
```

Módulo Cola

Este módulo contiene las operaciones pertinentes al almacenamiento de los mensajes en la cola, las cuales son cadenas de texto que recogen un mensaje para ser transmitido al terminal posteriormente.

El Módulo Cola contiene tres funciones: la primera es `int cola_init(t_cola *p)`, la cual establece una cola definida (`*p`) en un buffer vacío, de una longitud definida en el código de 4096 caracteres como máximo.

```
int cola_init(t_cola *p)
```

La segunda es `int cola_guardar(t_cola *p, char *msg)`, la cual recibe una cola (donde se guardará el mensaje) y el mensaje (`*msg`) que será analizado para comprobar si cabe en la cola. Antes de ser introducido el mensaje, se deshabilitarán las interrupciones de la unidad UART para evitar inconsistencia en los datos mientras se escribe el mensaje actual. Esta medida conseguirá que la interrupción no modifique los datos de la cola mientras un mensaje se está almacenando. Por último se activarán las interrupciones, anteriormente desactivadas.

```
int cola_guardar (t_cola *p, char *msg)
```

Finalmente, tenemos la función `int cola_leer (t_cola *p, char *msg, uint32_t l)` la cual se encarga de extraer un mensaje de la cola e introducirlo en una cadena de texto que puede procesar el programa. Durante el proceso, de igual forma que en la función anterior, las interrupciones de la transmisión serán deshabilitadas y posteriormente rehabilitadas al acabar el proceso.

```
int cola_leer (t_cola *p, char *msg, uint32_t l)
```

Las funciones de este módulo son utilizadas por los demás módulos para poder enviar mensajes y datos al terminal.

3.2.4. GSM

Para utilizar el módulo GSM externo, se ha utilizado un código implementado anteriormente por el Departamento de Informática de Sistemas y Computadores de la Universitat Politècnica de València. Este código incluye las funciones necesarias para realizar la comunicación con el módulo de GSM externo utilizando una segunda UART en la comunicación.

No entraremos en detalle en este apartado, pero sí hablaremos de los algunos de los Comandos AT [5] más utilizados en este proyecto:

- ATE: Activa o desactiva el eco, es decir, repite los comandos que envía el usuario, o en este caso, la unidad UART.
  - Al enviar "ATE0" desactiva el eco (1).
  - Al enviar "ATE1" activa el eco
- AT#QSS: Comprueba el estado de la tarjeta SIM
  - El comando "AT#QSS?" solicita información del estado de la tarjeta SIM (2).
  - La respuesta al comando anterior es "#QSS?: x,y": donde x es el modo, y es el estado (3).
- AT+CPIN: Comprueba el PIN de la tarjeta SIM
  - "AT+CPIN?" solicita al módulo GSM si la tarjeta SIM insertada requiere la introducción de alguna contraseña, ya sea el código PIN o el PUK (4).
  - La respuesta al comando anterior tiene esta forma "+CPIN: pin_status" donde pin_status puede ser
    - READY: Listo para usar.
    - SIM PIN: Esperando introducción del PIN.
    - SIM PUK: Esperando introducción del PUK.
  - Al enviar "AT+CPIN=pin" (donde pin es el valor del PIN de la SIM) se puede introducir el PIN (5).
- AT+CGDCONT: Estado de la conexión a la red de datos (6).
  - AL enviar "AT+CGDCONT=1,IP,APN,0.0.0.0,0,0": solicita la conexión a la red móvil identificada por el APN (ej: Jazztel)
- AT#USERID: Establece los datos de usuario para la conexión a la red de datos (7).
  - Al enviar "AT#USERID=usuario" se establece el usuario.
- AT#PASSW: Establece la contraseña de acceso a la red de datos (8).
  - Se envía "AT#PASSW=password" para establecer la contraseña.
- AT#GPRS: Estado del acceso a la red de datos
  - "AT#GPRS=1": solicita la conexión a la red de datos (9).
  - "AT#GPRS=0": solicita la desconexión de la red de datos.
  - "AT#GPRS?": solicita la información sobre el estado de la conexión a la red de datos (10). La respuesta puede ser:
    - "#GPRS=1": Se ha conectado a la red de datos. Se puede continuar.
    - "#GPRS=0": No se ha conectado a la red de datos.
- AT#SD: Apertura de una conexión de socket a un servidor remoto.
  - Al enviar el comando "AT#SD=1,0,puerto,servidor" se realiza una solicitud de apertura de un socket de conexión con el puerto "puerto" del servidor "servidor" (11).
    - La respuesta "CONNECT" indica que el socket está abierto. A partir de este momento toda la información enviada al módulo Telit es retransmitida al servidor y de la misma manera toda información procedente del servidor el módulo Telit la retransmite por la UART hacia el microcontrolador y por tanto, podemos seguir

A partir de aquí se envían y reciben mensajes intercambiados con el servidor web mediante el protocolo HTTP [7].

3.2.5. Muestreo

El módulo de muestreo es el que controla la temporización de todo el proyecto. Este está dividido en dos partes:

Una configura el oscilador, que controla, a su vez, la velocidad de transmisión de las unidades UART. Esto se consigue activando el oscilador principal y configurando el cristal a una frecuencia de 8 MHz mediante la función `initOsc()`.

```
void initOsc(void)
```

La otra parte, más extensa, se encarga de la configuración de los GPTM (temporizadores). Esto es, en primer lugar, la inicialización (función `GPTMinit(which_timer, freq, sysclk_mhz)`), o sea, activar la señal de reloj hacia el hardware del GPTM a configurar y a continuación realizar las escrituras oportunas en los registros de forma que establece la siguiente configuración:

- Modo de 16 bits
- Disparo de ADC
- Cuenta atrás Free-Running
- Sin PWM
- Preescala igual a la frecuencia en MHz para conseguir una cuenta cada 1µs

Finalmente, activa las interrupciones del temporizador con la función `GPTMenableISR()`.

```
void GPTMinit(which_timer, freq, sysclk_mhz)
```

```
void GPTMenableISR(which_timer)
```

Este módulo también contiene instrucciones para parar y activar la temporización de un temporizador (las funciones `void GPTMrun(void)` y `void GPTMstop(void)` respectivamente, así como las funciones `void GPTMn_ISR(void)` que se ejecutan al finalizar la cuenta atrás del temporizador correspondiente, donde ‘n’ significa el número de temporizador.

```
void GPTMrun(void)
```

```
void GPTMstop(void)
```

```
void GPTMn_ISR(void)
```

3.2.6. PWM

Mediante el Módulo PWM podremos activar la señal PWM de cada uno de los dos motores que utilizaremos. Como se ha explicado anteriormente, dedicaremos un temporizador, dividido en dos subtemporizadores de 16 bits en modo PWM con señal invertida.

El primer paso a realizar es la inicialización, esto es, la configuración del temporizador. La función `void PWM_init(void)` establece esta configuración en ambos subtemporizadores:

- Modo de 16 bits
- Sin disparo de ADC
- Con PWM
- Inversión de señal PWM

Finalmente, activa el temporizador.

```
void PWM_init(void)
```

A continuación, se deberá establecer el periodo y el ratio de cada señal PWM, para ello nos valemos de la función `void PWM_set(int subtimer, int ratio)`. Un ratio de 0 equivaldría a parar el motor.

```
void PWM_set(int subtimer, int ratio)
```

3.2.7. Programa principal

Todos los módulos y funciones anteriores son controlados por el programa principal y por las interrupciones asociadas a los diferentes eventos. En este programa principal se ejecutan todas las inicializaciones necesarias para que el conjunto de la programación realice correctamente su tarea, pues es el primer módulo que se ejecuta.

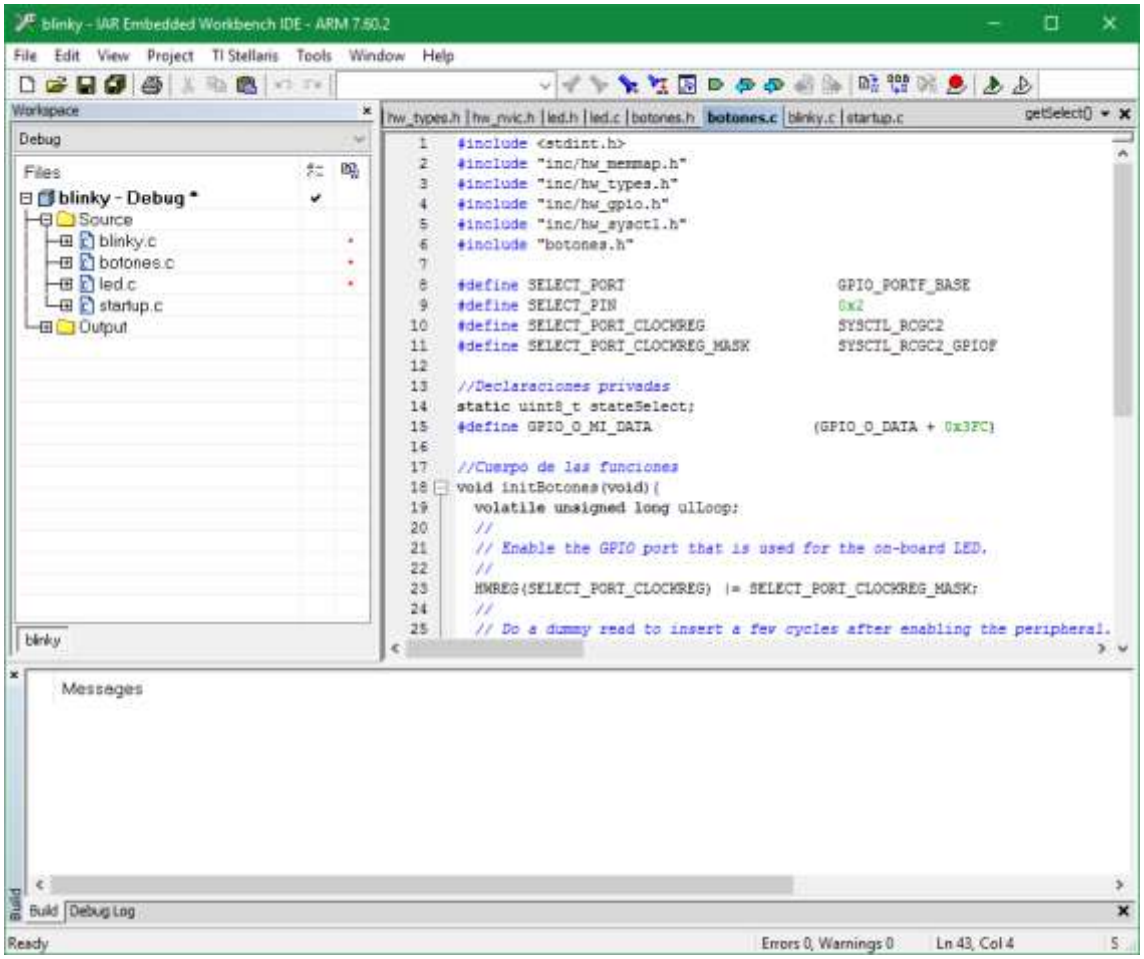
Una vez configurado todo, el programa principal acaba en un bucle vacío infinito, en el que se deja trabajar a las interrupciones.

Capítulo 4 – Herramientas utilizadas

4.1. Entorno de programación

En un principio se seleccionó el entorno CCS (Code Composer Studio) proporcionado por Texas Instruments, que es el fabricante del microcontrolador. Sin embargo, por razones diversas, no se logró hacer funcionar un programa con librerías externas necesarias para el desarrollo del proyecto, de modo que finalmente se eligió el entorno Embedded Workbench for ARM de IAR Systems.

Finalmente el proyecto ha sido desarrollado en lenguaje C (escrito, compilado y depurado) utilizando el entorno *Embedded Workbench for ARM* de *IAR*[3]



Este es un entorno con editor, compilador y enlazador con funciones de depuración, diseñado especialmente para programar sistemas empotrados.

4.2. Emulador de terminal

Un hiperterminal es un programa que simula una terminal de computadora, lo cual significa que, a través de ella, podremos recibir información, vía puerto serie, del microcontrolador.

Para recibir los mensajes de depuración provenientes del microcontrolador haremos uso del programa informático Putty. Este programa es capaz de emular un terminal conectado a un puerto serie que es lo que necesitamos para recibir los mensajes enviados por el microcontrolador.

Capítulo 5 – Características técnicas

5.1. Características del microcontrolador LM3S8962

El microcontrolador seleccionado tiene las siguientes características, las cuales están especificadas en su hoja de características: (entre otras cosas)

- **Microcontrolador de 32 bits RISC de altas prestaciones**
  - Arquitectura 32-bit ARM® Cortex™-M3 v7M optimizada para aplicaciones empotradas
  - Temporizador del sistema (SysTick), común a todos los modelos Cortex-M que ofrece un temporizador free-running de uso sencillo de 24 bits con cuenta descendente y generación de interrupciones al finalizar la cuenta
  - Funcionamiento a 50 MHz
  - Controlador de interrupciones integrado (NVIC – Integrated Nested Vectored Interrupt Controller – por sus siglas en inglés) que permite una gestión flexible de interrupciones con varios niveles de prioridad y un tiempo de ejecución determinista de los manejadores de interrupción
  - 36 interrupciones con ocho niveles de prioridad
  - Manipulación de bits individuales de memoria (bit-banding) lo que permite en la mayoría de los casos dejar de lado las costosas operaciones con máscaras durante el acceso a los registros de los periféricos
- **Memoria interna**
  - 256 KB de memoria flash no volátil para instrucciones y datos constantes
  - 64 KB de memoria volátil SRAM para datos
- **Entradas y salidas de propósito general (GPIOs)**
  - Entre 5 y 42 GPIOs, dependiendo de la configuración
  - Configuración de entradas tolerantes a 5V
  - Control programable de las interrupciones de las GPIOs
  - Enmascaramiento de bit en operaciones de escritura y lectura de los puertos a través de las líneas de dirección
  - Posibilidad de iniciar una secuencia de muestreo del ADC
  - Configuración programable de la electrónica del GPIO
  - Resistencias seleccionables de pull-up o pull-down
  - Potencia de salida seleccionable en 2mA, 4mA, y 8mA. Hasta cuatro GPIOs pueden ser configurados con una salida de potencia de 18mA
- **Temporizadores de propósito general (GPTMs)**
  - Cuatro GPTMs con dos temporizadores/contadores de 16 bits cada uno. Cada GPTM puede ser configurado para operar independientemente:
    - Como un único temporizador de 32 bits
    - Como dos subtemporizadores independientes de 16 bits
    - Para modulación de amplitud de pulso (PWM)
  - Modos de funcionamiento de 32 bits
    - Temporizador programable one-shot o free-running

    - RTC al usar un reloj externo de 32.768 KHz como entrada
    - Puede disparar eventos de conversión del ADC
  - Modos de funcionamiento de 16 bits
    - Temporizador programable one-shot o free-running con una preescala de 8 bits lo que permite disponer de un temporizador efectivo de 24 bits. En este modo el temporizador puede disparar eventos de conversión del ADC
    - Modo de captura de eventos para contabilizar flancos de una señal digital externa
    - Modo de temporización de eventos para determinar el instante en el que ocurren los flancos de una señal digital externa
    - Modo PWM en 16 bits capaz de generar una señal PWM con inversión opcional en la salida
- **ADC**
  - Cuatro canales de entrada analógica diferenciales o referidos a masa
  - Sensor de temperatura integrado
  - Ratio de muestreo a 500.000 muestras por segundo
  - Conversión analógica configurable y flexible: cuatro secuencias de conversión de muestreo programables de entre una y ocho entradas, con sus correspondientes colas FIFO para almacenar los resultados de la conversión.
  - Control flexible del disparo
    - Controlador (software)
    - Temporizadores
    - Comparadores analógicos
    - PWM
    - GPIO
  - Media calculada por hardware de hasta 64 muestras para mejorar la precisión
  - Referencia interna de 3V
  - El circuito entre tensión y tierra analógico está separado del circuito entre tensión y tierra digital para reducir el ruido en el resultado de la conversión
- **UART**
  - Dos UART totalmente programables de tipo 16C550 con soporte para comunicaciones por infrarrojos
  - Colas FIFO de 16x8 de recepción (RX) y transmisión (TX) separadas para reducir la carga del servicio de interrupciones de la CPU. La longitud de estas colas es configurable permitiendo una interfaz clásica de doble búfer de 1 byte
  - Generador de baud-rate programable que acepta velocidades de transmisión de hasta 3.125 Mbps
  - Niveles de disparo de la cola FIFO de 1/8, 1/4, 1/2, 3/4 y 7/8 de su capacidad
  - Bits de comunicación asíncrona para inicio, stop y paridad estándar
  - Características de interfaz serie totalmente programables
    - 5, 6, 7, u 8 bits de datos
    - Generación y detección de paridad fija, par, impar o ninguna

        - Generación de 1 o 2 bits de stop
    - Codificador/decodificador IrDA serial-IR (SIR) para comunicaciones mediante infrarrojos
        - Soporte para tasas de transferencia half-dúplex de hasta 115.2 Kbps
        - Soporte para el uso de duraciones de bits normales (3/16) y de bajo consumo (1.41-2.23 µs)
        - Generador interno de reloj programable para permitir la división del reloj de referencia entre 1 y 256 en aplicaciones de bajo consumo
- **Gestión de la alimentación**
    - Regulador de voltaje con poca pérdida (LDO – On-chip Low Drop-Out – por sus siglas en inglés) con salida programable entre 2.25V y 2.75V

Capítulo 6 – Conclusiones

Una vez finalizado el proyecto podemos extraer varias conclusiones y valoraciones al respecto.

La primera conclusión es que el desarrollo de este proyecto ha servido para orientarme en cuanto a la resolución de problemas, pues han surgido varios percances durante el desarrollo del software. Por ejemplo, diversos errores de compilación en el entorno *CCS de Texas Instruments* que llevaron a trasladar el código a un entorno distinto (*IAR Embedded Workbench for ARM*) o fallos en el hardware que obligaron a realizar un cambio de microcontrolador por otro del mismo modelo.

El proceso de realización de este proyecto también me ha permitido ahondar en el campo de la programación, descubriéndome varios comandos y tipos de datos, así como profundizar en la estructura de un sistema empotrado conociendo así las operaciones necesarias para modificarlo.

A todo esto, debo añadir que al seguir los pasos para completar este proyecto me pude iniciar en el desarrollo de proyectos de ingeniería, lo cual es positivo para mi futuro como ingeniero.

Finalmente, quiero mostrar mi satisfacción personal al haber realizado este proyecto.

Lamentablemente, no se pudo construir un prototipo pues debido a retrasos causados por los numerosos errores que surgieron, se agotó el tiempo que se hubiera dedicado al prototipo.

No obstante, hace unos años hubiera visto bastante improbable que pudiera ser capaz de llevar a cabo un proyecto de este tipo y resulta realmente gratificante haber conseguido finalmente poder presentar un proyecto que para mí ha sido realmente interesante.

Capítulo 7 – Presupuesto

Puesto que este proyecto se basa en la elaboración del código del sistema, el presupuesto contará únicamente con la mano de obra dedicada a la programación del código y la amortización del equipo utilizado.

El programa se diseñó entre los meses de abril y julio de 2016, dedicando a esta tarea 8 horas diarias durante los días laborables.

Horas trabajadas en abril:	8 h/día · 21 días laborables	=	168 h
Horas trabajadas en mayo:	8 h/día · 22 días laborables	=	176 h
Horas trabajadas en junio:	8 h/día · 22 días laborables	=	176 h
Horas trabajadas en julio:	8 h/día · 21 días laborables	=	168 h
	Total	=	688 h

Si tenemos en cuenta que el desarrollo del software se ha llevado a cabo por un Ingeniero Técnico Industrial asalariado según el Convenio del Metal vigente, el cual establece un salario estimado por mes de 1.563,02 €, al cual se le aplica un 30% más por gastos de Seguridad Social.

Salario base:	1.563,02 €/mes · 3 meses	=	4.689,06 €
Gastos en Seguridad Social:	4.689,06 € · 30%	=	1.406,72 €
	Total	=	6.095,78 €

En cuanto a la amortización del equipo utilizado, calculamos un precio de adquisición de 900€, con una vida útil de 8000 horas y un precio residual de 140 €. Este equipo será utilizado por unas 480 horas a un coste de uso de 45,60 €.

Por cada hora de uso, al valor inicial se le resta un valor de $^{45.60€}/_{480h} = 0{,}095\ €/h$.

En el tiempo que se ha utilizado el equipo, este se ha gastado un $^{45.60€}/_{900€ - 140€} = 0{,}06 \rightarrow 6\%$ de su vida útil.

Capítulo 8 – Bibliografía

8.1. Bibliografía

[1] Descripción del producto – http://www.ti.com/product/LM3S8962

[2] Hoja de características del Microcontrolador LM 3s8962 – http://www.ti.com/lit/ds/spms001g/spms001g.pdf

[3] Página web del entorno IAR Workbench – https://www.iar.com/iar-embedded-workbench/

[4] Manual de usuario del Módulo de GSM Externo Telit GL865 Quad – http://www.telit.com/fileadmin/user_upload/products/Downloads/2G/CL865/Telit_GL865-DUAL_QUAD_V3_Hardware_User_Guide_r11.pdf

[5] Guía de comandos AT de Telit – http://www.telit.com/fileadmin/user_upload/products/Downloads/2G/Telit_AT_Commands_Reference_Guide_r23.pdf

[6] Gráfica de osciloscopio de RS-232 – https://commons.wikimedia.org/wiki/File:Rs232_oscilloscope_trace.svg

[7] Información sobre el protocolo HTTP – https://es.wikipedia.org/wiki/Hypertext_Transfer_Protocol