Título del Trabajo Final

Nombre Estudiante: José Alberto Rosales Navarro
Plan de Estudios del Estudiante: Master Interuniversitario en Ingeniería de Telecomunicación
Área del trabajo final: Aplicaciones Electrónicas

Nombre Consultor/a: Aléix López Antón
Nombre Profesor/a responsable de la asignatura: Carlos Monzo Sánchez

Fecha de Entrega: 23/06/2016

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Nunchuk RoboArm*
Nombre del autor:	*José Alberto Rosales Navarro*
Nombre del consultor/a:	*Aléix López Antón*
Nombre del PRA:	*Carlos Monzo Sánchez*
Fecha de entrega (mm/aaaa):	*06/2016*
Titulación::	*Máster Interuniversitario en Ingeniería de Telecomunicación*
Área del Trabajo Final:	*Aplicaciones electrónicas.*
Idioma del trabajo:	*Español*
Palabras clave	1. ***Nunchuck**: Elemento de control utilizado que encapsula acelerómetros, joystick y botones necesarios para la totalidad de movimientos que el brazo robot dispone. Este dispositivo proviene de la video consola Wii.* 2. ***Microcontrolador**: Dispositivo encargado de ejecutar el código necesario para transformar las señales eléctricas provenientes de los acelerómetros en señales de control para mover los servos del brazo.* 3. ***Brazo robotico**: Elemento que simula la extremidad humana con el mismo nombre. Está compuesto por servos, articulaciones y una pinza. La totalidad de sus movimientos va a ser controlada en los seis grados de libertad que dispone.*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

Abstract (in English, 250 words or less):

Abstracto

Motivación

El otro motivo principal para la realización de este proyecto viene derivada por la pasión del autor que tiene por la robótica y la automática.
Aunque esta pasión no es algo nuevo, bien es decir que ha sido reactivada o renovada recientemente a partir de haber cursado ciertas asignaturas del presente Master Universitario en Ingeniería de Telecomunicaciones.
Igualmente el autor pretenden que este sea el comienzo de estudio del campo de la robótica y automática y con el tiempo conseguir crear un pequeño autómata.

Problemas encontrados
Debido a la gran madurez que hay sobre el uso de microcontroladores para todo tipo de aplicaciones y servos sobre todo en radiocontrol no se han encontrado problemas en estos aspectos ya que existe multitud de información y ejemplos en internet.

El único problema reseñable es la conexión, configuración y lectura del Nunchuk con la placa de desarrollo STM32F4 Discovery.

Solución

La solución adoptada para la resolución de los problemas de lectura de los datos de un Nunchuk ha sido el estudio de varias librerías desarrolladas para otras plataformas para obtener la información necesaria y poder crear una librería propia para el uso con la placa de desarrollo STM32F4 Discovery.

Resultado

Conclusiones

Gracias al gran auge que los microcontroladores están teniendo, la electrónica y robótica pueden estar al alcance de cualquiera siempre y cuando se tenga la pasión suficiente e imaginación para abordar un proyecto.

Resumen

El proyecto consiste en la elaboración de una plataforma que permita mover un brazo robot mediante un Nunchuck y que explote la totalidad de movimientos que el brazo puede proporcionar.

El brazo robot dispondrá de 6 grados de libertad:

1.- Hombro movimiento vertical: Permite aumentar y disminuir la altura del resto de articulaciones del brazo. Este movimiento abarcará el rango de inclinación comprendido entre 0 y 90 grados.

2.- Hombro movimiento horizontal: Permite mover el brazo horizontalmente. El rango de movimiento será de 180 grados, 90 grados hacia la izquierda y 90 grados hacia la derecha desde la posición central.

3.- Codo : Permite aumentar y disminuir la altura de la muñeca y la pinza. No se trata de un elemento que configure un movimiento completo en el brazo, sino que compone un movimiento completo junto al hombro (vertical y la muñeca).

4.- Muñeca movimiento vertical: Permite aumentar disminuir la posición de la pinza. Al igual que ocurre con el codo, el movimiento vertical de la muñeca no compone un movimiento independiente por si mismo y forma parte de un movimiento mas complejo junto al hombro y el codo.

5.- Muñeca movimiento rotatorio: Permite modificar la inclinación horizontal de la pinza.

6.- Pinza (movimiento de apertura y cierre): Se trata de un movimiento que produce la apertura y cierre de la pinza lo que permitirá coger objetos.

El dispositivo de control a utilizar será un Nunchuck. Este dispositivo integra un conjunto de acelerómetros para codificar el movimiento en las tres dimensiones, un joystick que codifica el movimiento en el plano y aportará la precisión al movimiento del brazo que los acelerómetros no permitan y dos botones los cuales se utilizarán para abrir y cerrar la pinza.

Un nunchuck, aunque parezca fácil pensarlo, no se trata de un conjunto de acelerómetros, joystick y dos botones. El acceso a estos es a través de solamente 4 cables, de los cuales dos son para alimentación, con lo que sólo quedan dos para comunicarse con el chip que hace de pasarela entre los cables y los elementos y la elaboración de la librería puede proporcionar mucha potencia para futuros proyectos.

La electrónica a utilizar para orquestar ambos subsistemas (control y brazo) está basada en la familia de microcontrolador STM32F4 en su presentación denominada STM32F407-Discovery. Este microcontrolador dispone de la memoria, entradas y salidas y potencia de calculo necesario para ejecutar el código necesario que convierta el movimiento del nunchuck en precisos movimientos del brazo.

Esta plataforma proporciona multitud de pines de entrada y salida, así como leds, botones e incluso un acelerómetro, lo cual lo hace perfecto para abordar este proyecto.

Agradecimientos

Los principales agradecimientos, de este proyecto fin de grado, van dirigidos a sus padres José Rosales Hernández y Encarnación Navarro López por la infinita paciencia que han tenido durante todo el tiempo que la elaboración de este proyecto ha durado.

A parte de paciencia han sido un pilar fundamental de apoyo en los momentos en los que no se encontraba la luz para encauzar la dirección del proyecto, momentos los cuales han sido muchos, en los que el autor ha perdido toda esperanza de alcanzar los objetivos planteados.

En segundo lugar y no por ello menos importante es necesario hacer reseña al apoyo y animo mostrado por los amigos del autor los cuales han sabido estar junto al autor en los momentos más duros y han sido capaces de proporcionar momentos divertidos y joviales cuando han notado que este lo necesitaba.

Índice

Capítulo1: Introducción

La electrónica es un campo que en la época que nos toca vivir está en auge sin perspectivas de que esta tendencia vaya a decaer.
El objetivo de este documento es recoger los distintos entregables definidos en el Trabajo Fin de Master asociado a la propuesta **Control de brazo robótico de 6 grados de libertad mediante Nunchuck y microcontrolador.**
El nombre que el producto final tendrá es el de **Nunchuk RoboArm**
Los distintos entregables se dividirán en dos grupos bien diferenciados Diseño e Implementación.

Capítulo 2: Estado del Arte

Subámbitos del Trabajo fin de Master.

En este proyecto surge a partir de la unión de tres campos, microcontroladores, servomotores e interfaces de control.
Por lo que es interesante hacer un pequeño análisis a modo de recordatorio de cada uno de ellos:

Microcontroladores

Desde el nacimiento de la electrónica digital se ha perseguido que los circuitos integrados crezcan en capacidades de proceso, desde los integrados de puertas lógicas (operaciones lógicas básicas como AND, OR, NAND, XOR,..) hasta los potentes microprocesadores de propósito general que hoy en día se pueden encontrar en el mercado, lo que hace que los precios vayan desde pocos céntimos hasta los miles de euros.
Como solución intermedia nacieron los microcontroladores, los cuales son computadores completos pero de limitadas capacidades, ideales para trabajos donde no es necesaria una excesiva capacidad de cálculo y prima el bajo precio.
Estos suelen ser pequeños, consumir poca energía y son fácil de programar para ellos en comparación de un microprocesador.
Por regla general no necesitan de un sistema operativo que los gobierne y son perfectos para tareas repetitivas.
En la actualidad la gran mayoría de los microcontroladores que se producen son consumidos por la industria de la informática, tanto es así que la mayoría de electrodomésticos llevan uno de algún tipo (frigoríficos, lavadoras, microondas, lavavajillas, etc.).
El gran avance que actualmente se está produciendo es la aparición de microcontroladores sobre placas de prueba. Efecto que hay beneficiado de forma importante a profesionales ya que les ahora tener que desarrollar su propia placa de pruebas cuando se quiere hacer un estudio de un microcontrolador.
En el ámbito educativo la llegada de los microcontroladores también ha tenido un impacto significativo haciendo mucho mas sencillo y cercano el conocimiento de estos dispositivos a los alumnos que pueden adquirirlos por apenas unas decenas de euros, cosa que era imposible antaño con las carísimas placas de microprocesadores que había en los laboratorios (tales como la PROMAX que integraba un Motorola 68000)
A esto se suma la aparición de compiladores cargadores de código muy baratos e incluso gratuitos ha provocado que la electrónica esté cada al alcance de todo aquel que quiera iniciarse en el tema. Cada vez son más los IDEs que permiten programar en lenguajes de programación conocidos como C, C++, java, Phyton, etc y que son capaces de compilarlos a código objeto capaz de ser subido y ejecutado en un microcontrolador.

Servos

Un servo es un motor eléctrico de corriente continua, pero a diferencia de estos tienen la capacidad de posicionarse un en rango de posiciones posibles y mantener esta posición.
Aparecen como la necesidad de controlar articulaciones de forma precisa sin la necesidad de usar poleas o multitud de engranajes. Es por esto que los servos se utilizan en multitud de proyectos que requieran movilidad controlada.
El uso de servos toma mayor importancia al usarlos en modelismo, donde son el elemento principal ya que permiten simular el funcionamiento de los vehículos reales.
La existencia de los servos es en gran parte gracias a la modulación de anchura de pulso o PWM. Esta modulación hace que la anchura del pulso que se envía a un servo es variable y codifica la posición que se quiere que este alcance. Este tipo de modulación se comentará con mayor profundidad en posteriores entregas de este TFM.

Nunchuk

Este dispositivo apareció como extensión del mando inalámbrico original Wiimote de la video consola de Nintendo Wii.
Esta extensión se conecta por cable a la parte inferior del mencionado Wiimote y aporta control adicional y proporciona la posibilidad de desarrollar juegos de control tradicional que requieren de un joystick y botones para ser controlados.
Este dispositivo aúna tres tipos de dispositivos para realizar este control: tres acelerómetros, uno para cada eje (X, Y, Z), un joystick analógico para controlar con el dedo pulgar y que codifica el plano comprendido entre su eje X e Y, y por último pero no por ello mas importante dos botones denominados C y Z.
Esta combinación lo hace lo hace muy atractivo para tenerlo en cuenta a la hora de emprender un proyecto en el que exista la necesidad de controlar algún vehículo o como en este caso un brazo robótico.
La comunicación se realiza mediante dos líneas una de datos SDA y otra de reloj SCL a parte de las dos líneas de alimentación (3.3v o 5v y GND). Para ello el Nunchuk tiene un microcontrolador interno que captura y procesa la señales de los acelerómetros, joystick y botones.
Aunque se trata de un dispositivo barato que se puede adquirir hasta por 2.5 euros hay que hacer reseña a que provee de bastante precisión y sensibilidad en sus acelerómetros y joystick.
Este dispositivo ha levantado mucho interés entre los aficionados a la electrónica y se puede encontrar gran variedad de enlaces en internet de ejemplos de proyectos en los que se usa este mando Nunchuk como base de control.

Éxitos existentes en el ámbito del Trabajo fin de Master.

Algunos ejemplo tangibles del éxito de este tipo de brazos robotizados controlados por un usuario humano son:

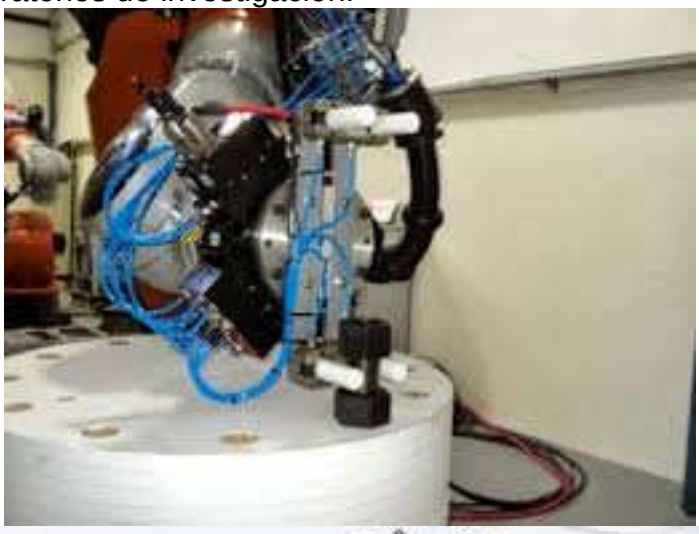
Carga de mercancías

En la industria donde se trabaja con materiales de grandes volúmenes o pesos, este tipo de herramientas es utilizado para cargar, transportar o mover estos

objetos. Este tipo de dispositivos son usados en puntos donde se requiere de un cuidado especial por la naturaleza del objeto a mover o donde los montacargas no pueden acceder.

Manipulación de elementos peligrosos

En la naturaleza hay multitud de elementos que son peligrosos para la vida humana, pero que por otro lado tienen un gran valor para el desarrollo de la vida humana, como por ejemplo son los elementos radioactivos, en centrales nucleares, laboratorios de investigación.



Robot policía

En el marco policial este tipo de robots, una vez dotados sobre un dispositivo que le proporcione automoción y provisto de algún sistema de monitorización para el usuario humano que lo maneje, es una herramienta perfecta para varias tareas como pueden ser:

Desactivación, extracción y/o manejo de explosivos.

Este dispositivo es perfecto para manipular explosivos de forma segura para la vida humana ya que con ello no se pone en peligro la vida de ningún ser humano.

Cizallas mecánicas.

En muchas ocasiones la policía tiene la necesidad de cortar alguna estructura la cual es difícil o imposible para un ser humano o puede poner en peligro la seguridad del operario. Para ello se pueden usar robots con un brazo articulado en los que se sustituye la pinza por unas cizallas para cortar metales.

Abrir puertas de habitáculos comprometidos para la seguridad.

En muchas ocasiones la policía tiene la necesidad de acceder a un habitáculo cerrado al cual se interpone una puerta y existe el peligro de

que habitáculo sea un ambiente explosivo, alta temperatura de la puerta o pomo, existencia de fuego tanto en el habitáculo al que se quiere acceder como en el ambiente anterior y desde el cual se quiere acceder.

Ámbito médico quirúrgico

Este tipo de brazos robot es actualmente utilizado en quirófanos en los que se requiere de una precisión milimétrica.

El dispositivo de control de estos brazos robot actualmente no son un nunchuk, sino dispositivos basados en joysticks y botones. El autor de este TFM piensa que el uso de sistemas de control que incluyan acelerómetros pueden añadir profundidad y precisión en el control de estos brazos.

Problemas en los que se trabaja actualmente.

La utilización de microcontroladores y servos se encuentra en un estadio muy maduro y no se consideran problemas reseñables.
Esta madurez ha sido resultado del amplio espectro de ámbitos en los que estos dispositivos son utilizados haciendo especial mención al modelismo, el cual cada vez tiene más adeptos.

Posibles aplicaciones

Como se ha comentado las aplicaciones de los tres componentes por separado son incontables.
Pero en el caso de este trabajo, o sea un brazo robótico controlado con microcontrolador y cuyo interfaz humano-máquina sea un nunchuk, las aplicaciones son varias y bastante importantes en distintos campos:

Entretenimiento infantil

Uno de los primeros usos que este proyecto puede tener es el de juguete para niños. Estos pueden usarlo para jugar a atrapar cosas y cambiarlas de ubicación. También puede ser usado como grúa en juegos que simulan entornos de construcción.

Ámbito educacional

Este proyecto puede ser beneficioso para alumnos con problemas psicomotrices y puede ayudar a mejorar la precisión de sus movimientos, así como mejorar el pulso

Ámbito médico

También puede ser utilizado para la recuperación de la movilidad de pacientes que hayan sufrido algún tipo de accidente y/u operación en las extremidades superiores, ayudando a la mejora de la movilidad y precisión de esta.

Software y Hardware

Software

El software solamente concierne al microcontrolador a utilizar en el proyecto.

Para un mismo microcontrolador y dependiendo del modelo existen gran variedad de lenguajes de programación. Desde código ensamblador o lenguaje a bajo nivel, hasta lenguajes de altísimo nivel como son C, C++, java, python, etc.

Del mismo modo existen gran número de entornos de trabajo y/o programación para estos dispositivos, algunos simples como sencillos editores de texto junto a comandos de carga de código en los microcontroladores. Otros entornos mas complejos cuyo propietario es el fabricante de los microcontroladores o del ensamblador de la placa donde el microcontrolador viene integrado. Estos entornos interpretan y compilan lenguajes de alto nivel, generan código objeto para el microcontrolador en concreto y tienen la capacidad de subir dicho código al microcontrolador.

Los terceros y más completos son entornos de programación de terceras compañías los cuales son válidos en su mayoría para una gran variedad de microcontroladores y plataformas, los cuales son capaces de realizar análisis sintácticos, semánticos, así como compilar el código fuente y generar código objeto capaz de ser interpretado por el microcontrolador destino. Algunos de estos entornos de programación tienen la capacidad de subir el código en el microcontrolador.

Hardware

En este apartado se van a comentar los tres componentes del TFM por separado para con ello dar un mayor visión de la gran variedad de posibilidades que en ellos existe.

Nunchuk

Los modelos de nunchuk son reducidos, existen los modelos:

- Original: Viene incluido en el pack base de la videoconsola Wii, aunque se puede comprar por separado por un precio que ronda los veinte euros.

- Compatible: Se puede encontrar en numerosos proveedores y tiendas. Suele tener un precio que abarca desde los 2.5 hasta los 10 euros.

Las características son similares aunque los grandes jugadores y expertos en este tipo de controles insisten en que la calidad y precisión de los original es sumamente superior a la de los compatibles.

Microcontroladores

Existen multitud de microcontroladores en el mercado, la diferencia entre ellos viene por el las capacidades que proporcionan, distribución de pines, encapsulado, fabricante.

Esto hace que la elección del microcontrolador se convierta en uno de los procesos mas importantes en la vida de un proyecto. Esta elección condicionará factores importantes como coste del proyecto, velocidad y precisión de procesado y capacidad de futuras ampliaciones.

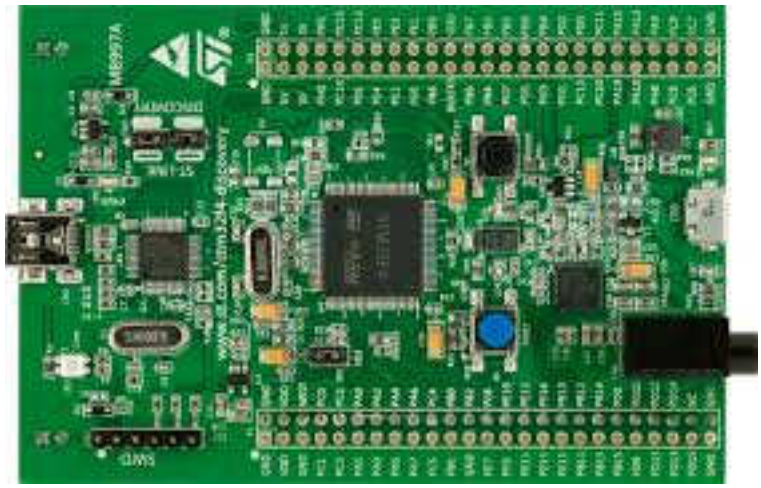
Ejemplos de microcontroladores y sus placas son:

- **Atmel en su presentación Arduino**

  Perfecto para ámbito educacional y aficionados a la electrónica.
  Se presenta en numerosos formatos y como ejemplos:
  - Arduino R3 UNO: baja capacidad de procesado, entradas analógicas, y entradas y salidas digitales. Capacidad de salidas PWM.

- Arduino Mega: igual que su hermano pequeño el arduino R3 UNO, pero con mayor capacidad de procesado y mayor numero de pines de entrada y salida tanto analógicas como digitales.

- **Stm32f4 en su presentación Stm32f4 discovery**.
  También es utilizado para el ámbito educacional, su presentación más común es en la placa discovery.
  Pertenece a la familia Cortex-M
  Dispone de 100 pines de entrada salida.
  La placa discovery está provista de leds de distintos colores, un acelerómetro, un codificador de audio entre otras capacidades.



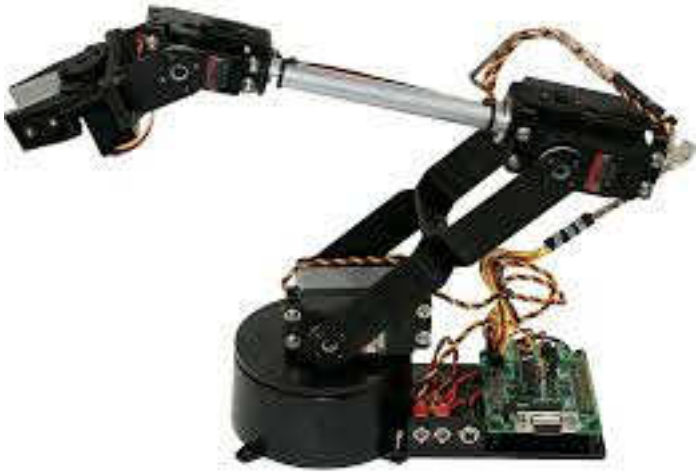
Como se ha dicho anteriormente existe una gran variedad de microcontroladores y presentaciones y enumerarlos todos haría muy difícil la finalización de este documento.

Brazos Robóticos

En el mercado existe un amplio abanico de brazos articulados cuyas articulaciones están formadas por servos.
Los hay dependiendo del tamaño, material y tipo de extremo.

- El tamaño sólo depende de la longitud de los segmentos que lo componen.

- Los tipos de material más comunes son, plástico y metal.
- En cuanto al extremo del brazo los hay de multitud de tipos como son pinzas de dos segmentos, pinzas de tres segmentos, un elemento punzante (perfecto para colocar un imán o un destornillador).



Otra diferencia que se puede encontrar en los brazos comerciales es el tipo de servo que forman parte de el

En este aspecto existen una amplísima variedad de servos los cuales se diferencia entre otras cosas por el tamaño, voltaje (3-9v) de funcionamiento, peso que soporta (7-20kg), material de las coronas que componen su mecanismo (plástico y metal) .

Capítulo 3: Diseño

Esquema
El diseño del proyecto en bloques es sencillo y solo está constituido por tres módulos:

Esto se corresponde con una conexión física entre el Nunchuk y la placa de desarrollo y otra conexión entre la placa de desarrollo y el brazo mecánico.

Intefaz de Control Humano

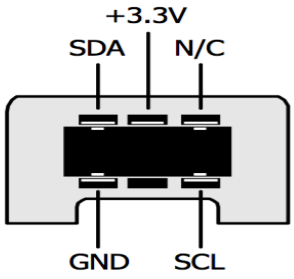
El primer módulo se corresponde con el Interfaz de Control Humano:
El sistema control está compuesto básicamente por el periférico de control de la video consola Wii de Nintendo, Nunchuk.

Este periférico consiste en una empuñadura sobre la que se ha montado:

- Un joystick analógico de dos dimensiones X e Y.
- Un acelerómetro de 3 dimensiones X, Y, Z.
- Un botón serigrafiado con la letra C y otro con la letra Z.

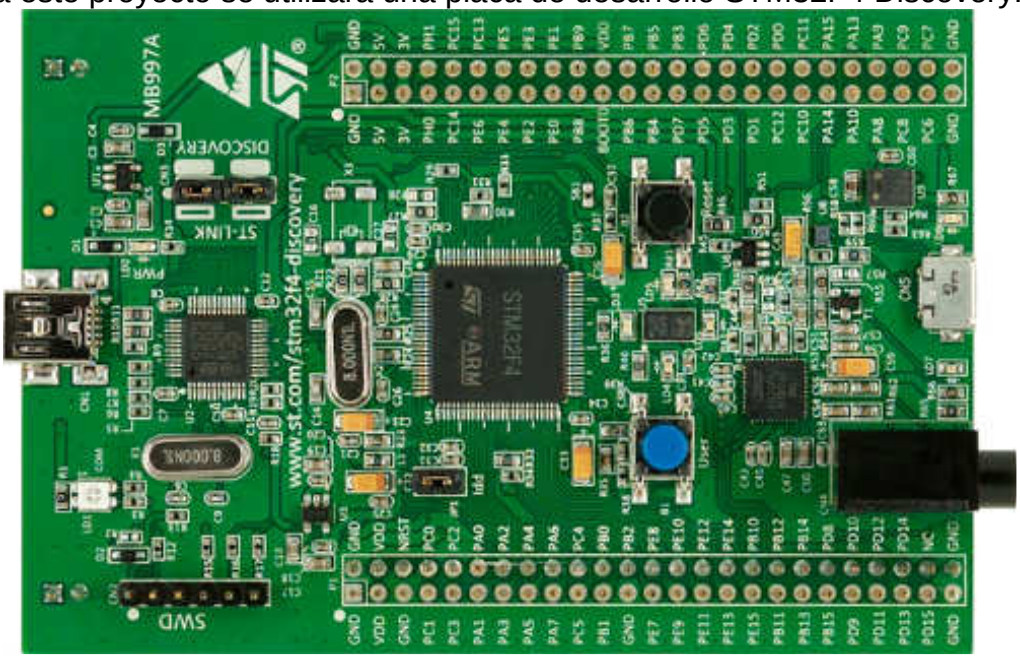
Para conseguir utilizar la rotación del mando sobre el eje Y (rotación lateral), denominado como Roll, así como la rotación sobre el eje X (inclinación hacia adelante y hacia atrás), denominado Pitch es necesario combinar en los valores procedentes de los acelerómetros.

Este mando tiene como punto de conexión un conector propietario de Nintendo de 6 pines de los cuales sólo se usan 4, quedando los otros dos para futuras revisiones.



Sistema de Procesado

El segundo módulo se corresponde con el sistema de procesamiento y control.
El sistema de procesamiento y control estará compuesto por un microcontrolador.
Para este proyecto se utilizará una placa de desarrollo STM32F4 Discovery.



Esta placa será la encargada de:

a.) Obtener la información del interfaz de control humano (Nunchuk) a través de los puertos de entrada y buses de comunicación que dispone.
b.) Procesar la información obtenida y calcular las nuevas posiciones que los distintos servos deben tener, mediante fórmulas independientes para cada servo.
c.) Actualizar la posición de los servos que harán que se mueva el brazo alcanzando la posición deseada por el usuario.

Sistema Mecánico

El tercer y último módulo se corresponde con el sistema mecánico.
Este sistema está compuesto por un brazo robótico articulado por seis servos, los cuales controlan los seis grados de libertad que proporciona.
Estos seis grados de libertad son:

- Movimiento horizontal del hombro: Este movimiento controla la dirección en la que el brazo puede actuar.

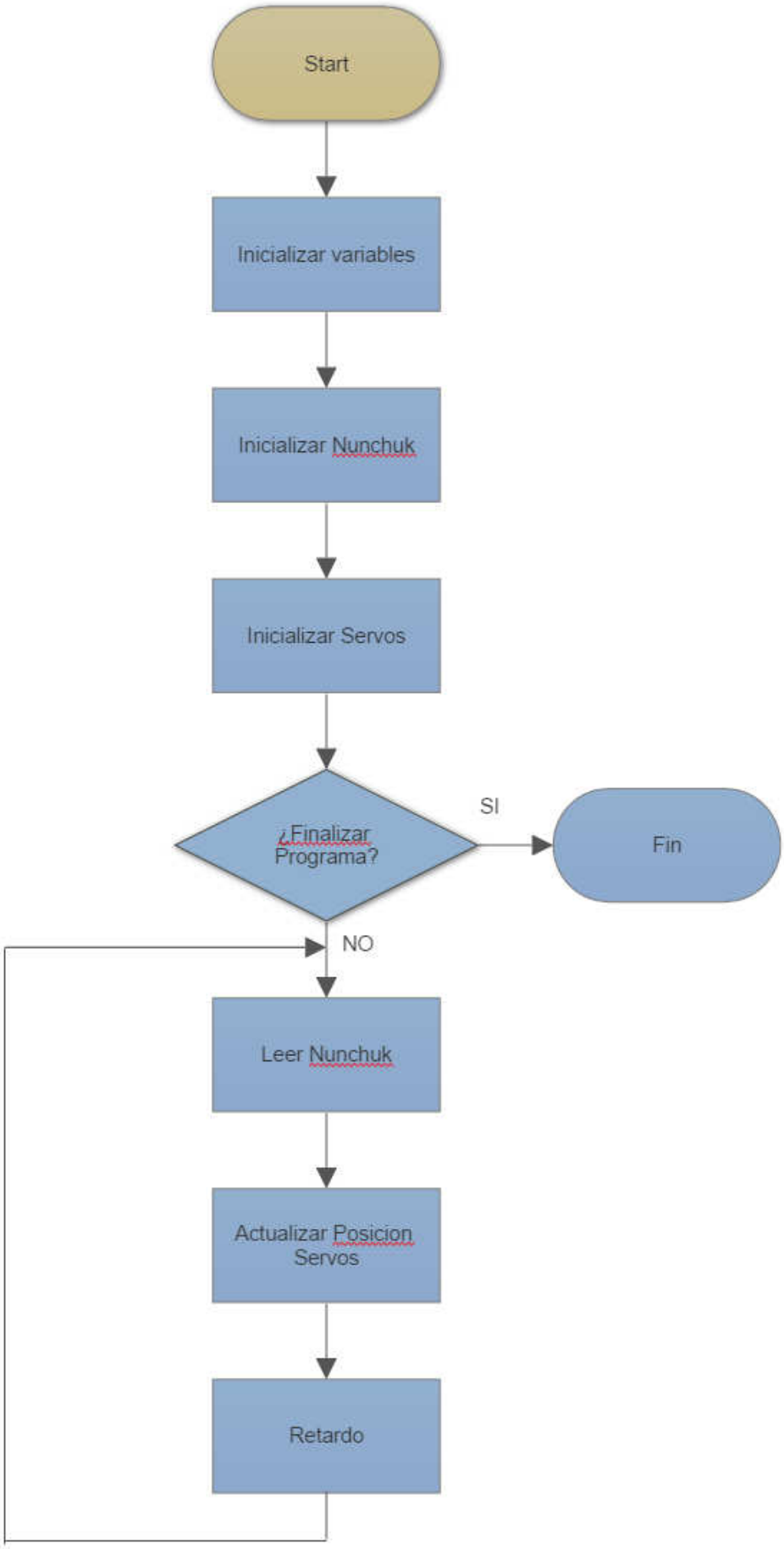
- Movimiento vertical del hombro: Este movimiento controla la altura del brazo en la que este puede actuar.
- Movimiento de expansión del brazo o codo: Este movimiento controla el radio de acción del brazo desde la misma base hasta la longitud completa del brazo.
- Movimiento rotatorio de la muñeca: Este movimiento permite alinear el elemento de agarre del brazo en ángulos comprendidos entre 0 y 180º.
- Movimiento vertical de la muñeca: Este momento permite modificar la inclinación del elemento de agarre del brazo permitiendo ampliar la precisión para coger y soltar cosas.
- Movimiento de apertura y cierre del elemento de agarre: Este elemento de agarre se corresponde con una pinza de dos dedos y el movimiento permite agarrar objetos de distintos tamaños.

Flujograma de acción

El flujograma de ejecución es sencillo, aunque esto no conlleva que el diseño del hardware y la programación del código lo sea.

1.- Este consiste en una fase inicial de configuración del sistema
    1.1.- Inicialización de variables.
    1.2.- Registro e inicialización del Nunchuk.
    1.3.- Registro e inicialización de los servos.

2.- Bucle infinito que ejecuta el grueso de la funcionalidad
    2.1.- Lectura de los valores almacenados en el Nunchuk
    2.2.- Actualización de servos.
    2.3.- Retardo.


Start
Inicializar variables
Inicializar Nunchuk
Inicializar Servos
¿Finalizar Programa?
SI
Fin
NO
Leer Nunchuk
Actualizar Posicion Servos
Retardo

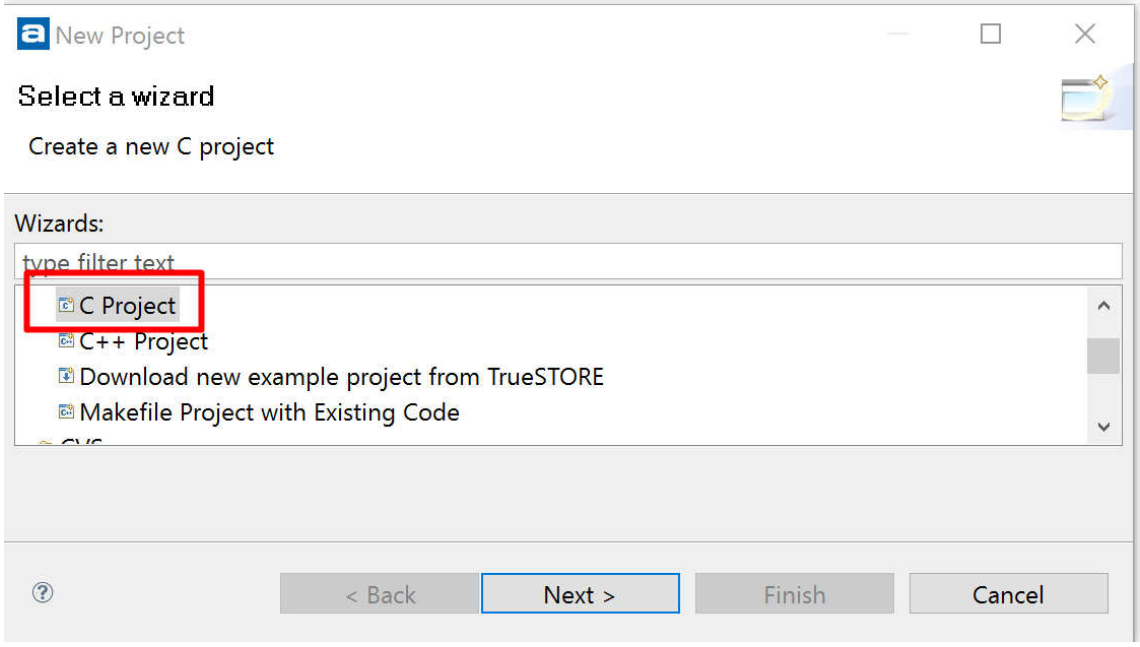
IDE de Programación

Selección de IDE de programación

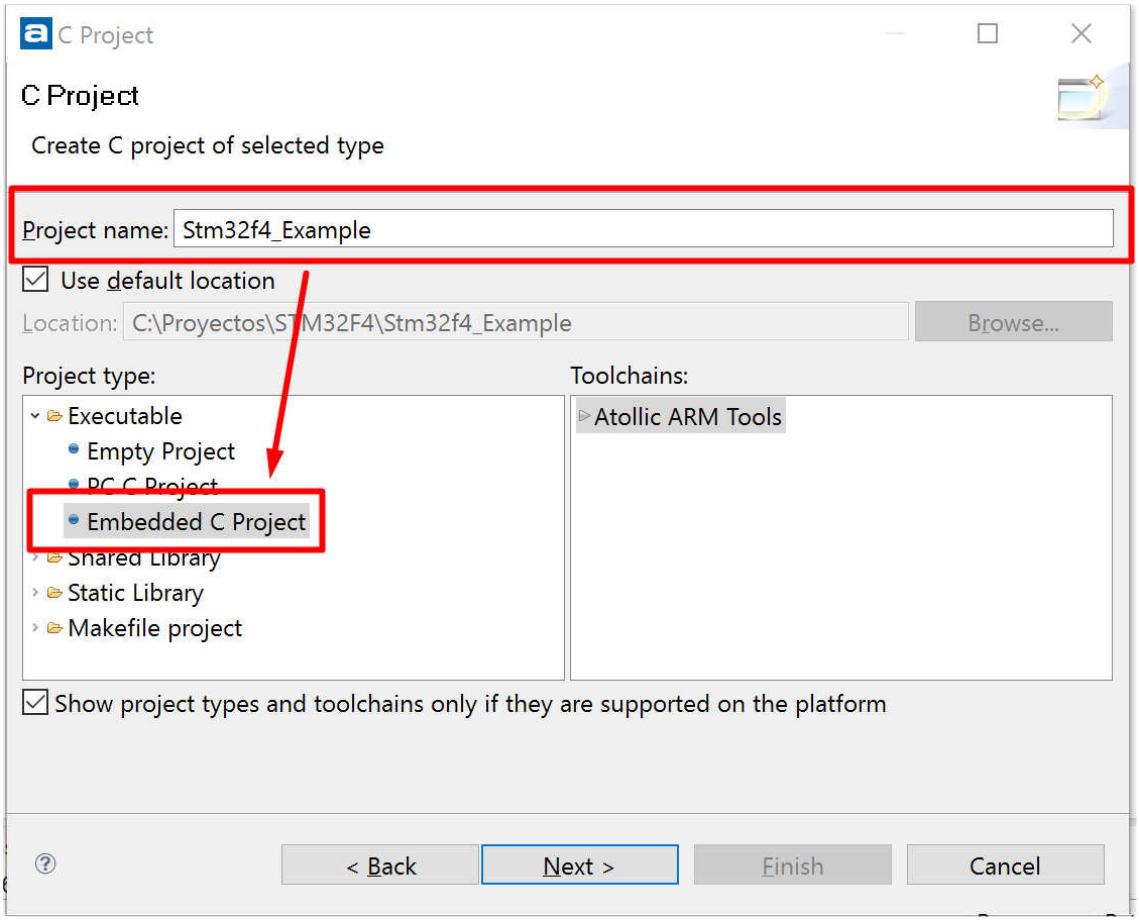
En el mercado existen varios IDEs que proporcionan las herramientas necesarias para trabajar con esta placa de desarrollo.
El propio empaquetado de la placa sugiere varias opciones:

- Keil: MDK-ARM
- CooCox
- IAR: EWARM
- IDEs basados en GCC: Atollic TrueStudio
- ARM mbed online

Si bien la página oficial de la placa http://stm32f4-discovery.net/ recomienda la utilización de Keil µStudio o CooCox, existen multitud de tutoriales en la red que recomiendan el uso de Atollic TrueStudio.
Tras seguir diferentes tutoriales y manuales para crear nuevos proyectos, así como compilarlos y ejecutarlos en la placa se ha llegado a la conclusión de que Atollic TrueStudio es el IDE perfecto para este proyecto ya que los parámetros de configuración son mínimos, mas intuitivo y que menos conocimientos requiere para las necesidades que este proyecto tiene.
Con este IDE los pasos para crear un nuevo proyecto para la placa de desarrollo STM32F4 Discovery son los siguientes:
1.- Pulsar sobre la opción Nuevo -> Proyecto y seleccionar C Project o C++ Project.
Pulsar Siguiente.

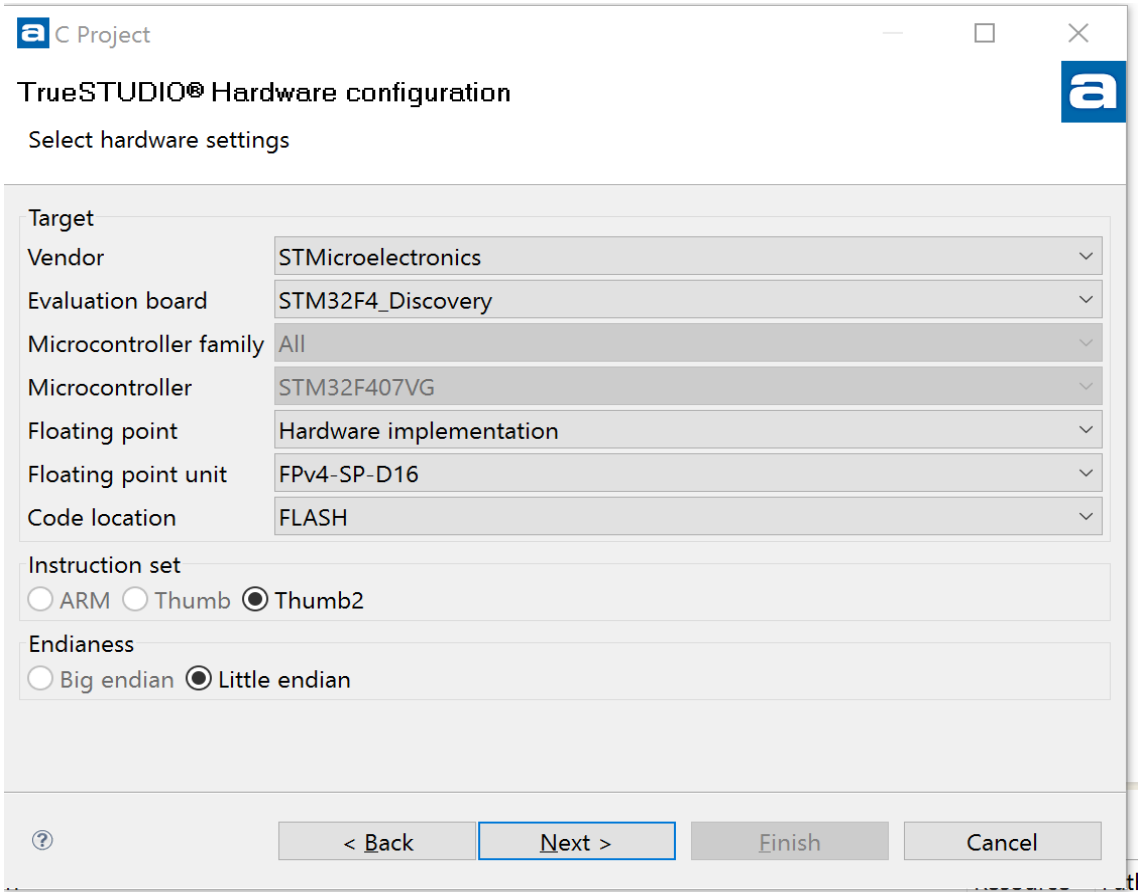


2.- Darle nombre y seleccionar Embebed Project
Esto se hace así ya que se va a ejecutar en un microcontrolador y no en un sistema operativo gobernado por un procesador.
Pulsar Siguiente.

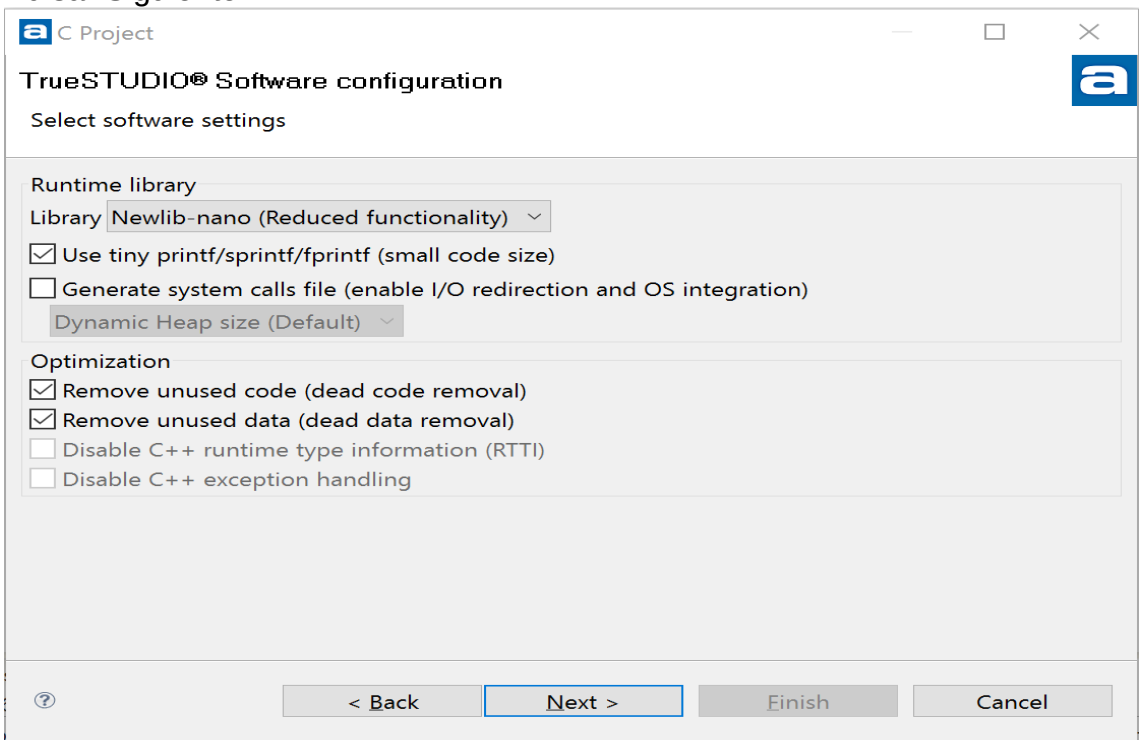


3.- Seleccionar los siguientes parámetros de la placa donde se ejecutará el proyecto
- Fabricante: STM Electronics.
- Placa de evaluación: STM32F4 Discovery.
- Punto Flotante: Implementacion por Hardware.
- Localización del Código: Flash

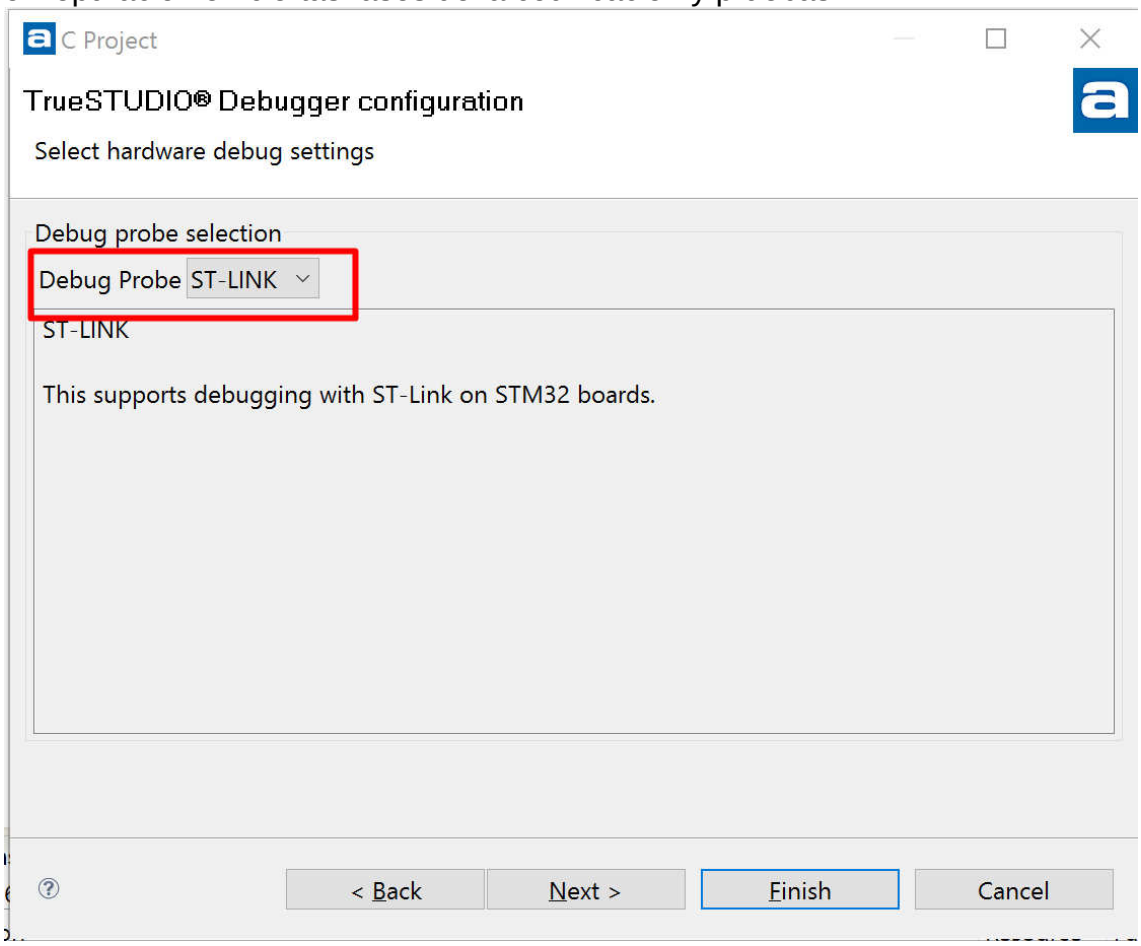
Pulsar Siguiente



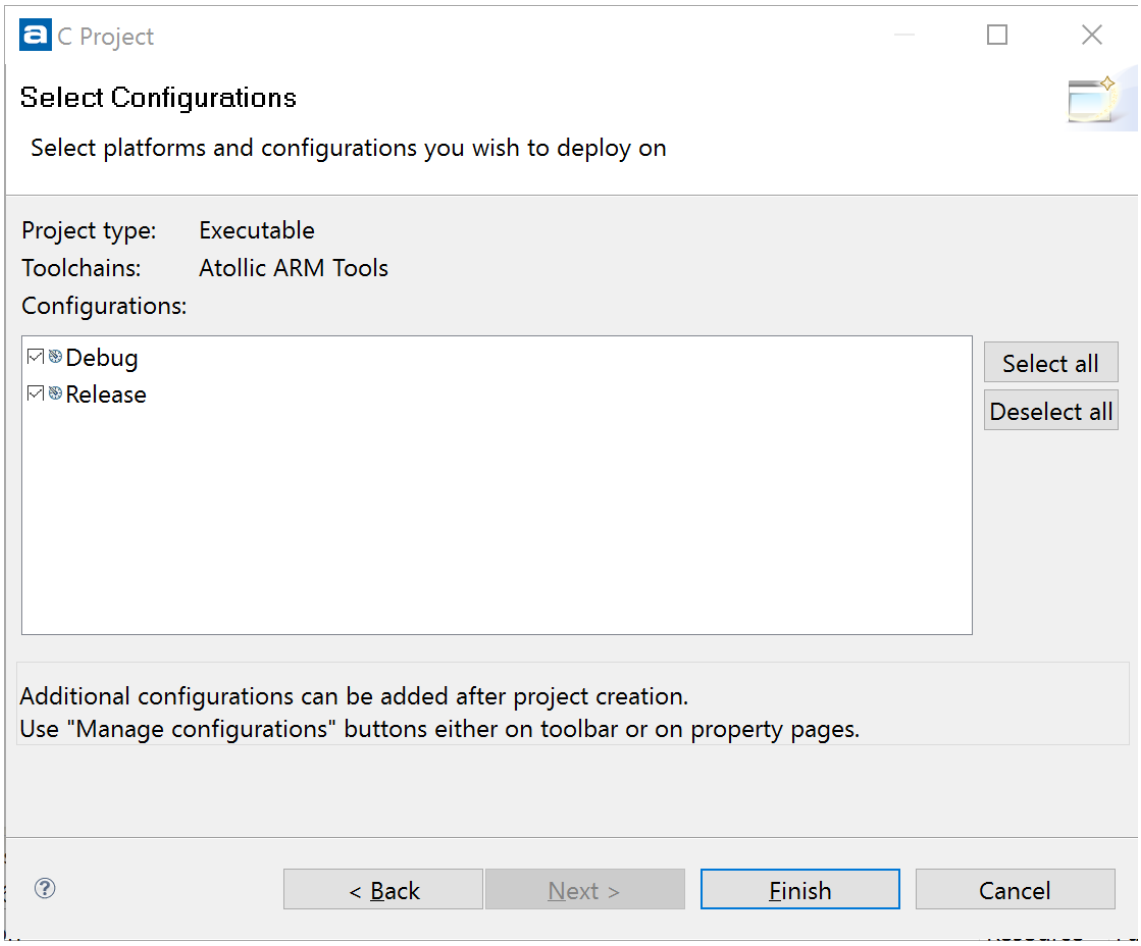
4.- Selección de tipo de librería.
Se debe seleccionar Newlib-nano debido a que las librerías deben ocupar el mínimo espacio ya que tenemos un tamaño de flash muy reducido.
Pulsar Siguiente



5.- Selección del modo de Debug
Este apartado es muy importante debido a que será necesario recurrir al Debug o Depuración en ciertas fases de la codificación y pruebas.



Se debe seleccionar el tipo de prueba ST-LINK ya que es el interfaz que la placa lleva integrado.
6.- Selección de generación de objetos.
En este paso el IDE propone que tipos de objetos generar cuando se compila el código fuente.
Se deben seleccionar los dos tipos que propone debug y reléase.
Debug genera un código intermedio entre el código fuente de alto nivel escrito por el programador y el código objeto maquina a ejecutar en la placa. De este modo el IDE puede hacer debug cuando se ejecute en este modo en la placa.
Release: El objeto que se genera en modo reléase (liberación) es un código a bajo nivel que se ejecuta en el microcontrolador.



Tras pulsar Aceptar el asistente de creación de nuevo proyecto habrá finalizado.
El IDE creara la estructura básica de un proyecto para la placa de desarrollo STM32F4 Discovery

- Stm32f4_Example
  - Binaries
  - Includes
  - Libraries
  - Utilities
  - src
    - main.c
    - startup_stm32f40xx.s
    - stm32f4xx_conf.h
    - stm32f4xx_it.c
    - stm32f4xx_it.h
    - system_stm32f4xx.c
    - tiny_printf.c
  - Debug
  - stm32f4_flash.ld

La clase principal que la placa ejecutará es la clase main.c la cual tiene la siguiente estructura:

```
/* Includes */
#include "stm32f4xx.h"
#include "stm32f4_discovery.h"
```

```
/* Private macro */
/* Private variables */
/* Private function prototypes */
/* Private functions */

/**
**===========================================================================
=======
**
**  Abstract: main program
**
**===========================================================================
=======
*/
int main(void)
{
  int i = 0;


  /* Infinite loop */
  while (1)
  {
      i++;
  }
}
```

Las demás clases creadas para seguir una buena práctica de programación deben estar en la misma carpeta scr que main.c y deben ser importadas para ser utilizadas.

Primer Proyecto

Siguiendo las instrucciones de la página http://stm32f4-discovery.net/2014/04/stm32f429-discovery-gpio-tutorial-with-onboard-leds-and-button/ y utilizando el proyecto creado en el punto anterior se rellena la clase main.c con el siguiente código:

```
#include "stm32f4xx.h"
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_gpio.h"

int main(void) {
        /*******************************************************/
        /*                    BLOQUE 1                         */
        /*******************************************************/
        GPIO_InitTypeDef GPIO_InitDef;
        RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOG, ENABLE);
        GPIO_InitDef.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15;
        GPIO_InitDef.GPIO_Mode = GPIO_Mode_OUT;
        GPIO_InitDef.GPIO_OType = GPIO_OType_PP;
        GPIO_InitDef.GPIO_PuPd = GPIO_PuPd_NOPULL;
        GPIO_InitDef.GPIO_Speed = GPIO_Speed_100MHz;
        //Initialize pins
        GPIO_Init(GPIOD, &GPIO_InitDef);
        /*******************************************************/
        /*                    FIN BLOQUE 1                     */
        /*******************************************************/
```

```
        /*******************************************************/
        /*                    BLOQUE 2                         */
        /*******************************************************/

        RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);

        GPIO_InitDef.GPIO_Pin = GPIO_Pin_0;
        GPIO_InitDef.GPIO_Mode = GPIO_Mode_IN;
        GPIO_InitDef.GPIO_OType = GPIO_OType_PP;
        GPIO_InitDef.GPIO_PuPd = GPIO_PuPd_DOWN;
        GPIO_InitDef.GPIO_Speed = GPIO_Speed_100MHz;
        //Initialize pins
        GPIO_Init(GPIOA, &GPIO_InitDef);
        /*******************************************************/
        /*                    FIN BLOQUE 2                     */
        /*******************************************************/

        //volatile int i;
        /*******************************************************/
        /*                    BLOQUE 3                         */
        /*******************************************************/
        while (1) {
           if (GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0)) {
                 GPIO_SetBits(GPIOD, GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14 |
                 GPIO_Pin_15);
           } else {
                 GPIO_ResetBits(GPIOD, GPIOD, GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14 |
                 GPIO_Pin_15);
          }
        }
        /*******************************************************/
        /*                    FIN BLOQUE 3                     */
        /*******************************************************/

}
```

Explicación del código.

- En la zona etiquetada como BLOQUE 1 se registra los pines 12,13,14 y 15 del puerto “D” de la placa de desarrollo como pines de salida. Como se indica en el estudio inicial de la placa estos pines están conectados físicamente a los leds rojo, verde, naranja y azul que se encuentran en el centro de la placa.

- En la zona etiquetada como BLOQUE 2 se registra el pin 0 del puerto “A” como pin de entrada. Como se indica en el estudio inicial de la placa este pin esta físicamente conectado con el botón de usuario de color azul.

- La zona de código etiquetada como BLOQUE 3 se trata de un bucle infinito en el cual se consulta el valor del pin 0 del puerto “A” y si este vale 1 (5 voltios) o lo que es lo mismo el botón azul está pulsado se pondrán los pines 12, 13, 14 y 15 del puerto “D” a nivel alto (5 voltios), o lo que es lo mismo se encenderán los 4 leds.

  Si por lo contrario el botón azul no está pulsado los pines 12, 13, 14 y 15 del puerto “D” a nivel bajo (0 voltios), o lo que equivale a apagar los 4 leds.

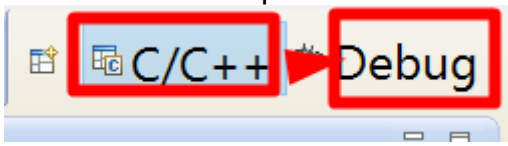
En conclusión, lo que el código hace es que mientras el botón azul se mantenga pulsado los cuatro leds se mantendrán encendidos y cuando no esté pulsado los leds estarán apagados.
Este es un ejemplo sencillo pero muy representativo del uso de puertos como entradas y salidas.

Una vez finalizada la codificación, el proyecto resultante deber ser compilado para que el IDE analice si se ha producido algún tipo de error sintáctico o semántico, para ello se debe el botón de la barra de herramientas del IDE cuyo icono es un martillo:

El resultado de la compilación , así como los errores, avisos e información se muestra en la pestaña Problemas que hay en la parte inferior (por defecto) del IDE:

Por último solo falta ejecutar el código creado y compilado en la plataforma de desarrollo, y para eso es suficiente con tener conectado la placa a un puerto USB del ordenador y pulsar sobre el botón cuya imagen es un escarabajo verde (debug)

Al comenzar este proceso el IDE cambia de perspectiva de la C/C++ a Debug



Cuando el código empieza a ejecutarse en la placa, se detiene en la primera línea con código escrito, no comentado que exista dentro del método int

main(void). Por lo que será necesario pulsar el botón Resume (continuar) para que la ejecución prosiga.

STM32F4 Discovery

En esta fase del proyecto se realiza un estudio minucioso de la placa de pruebas STM32F4 Discovery, para con ello intentar descubrir sus principales capacidades.

Este estudio de la laca de pruebas STM32F4 desvela muchos datos relevantes.

- Esta gobernado por un microcontrolador STM32F407 de la familia Cortex-M.
- Conexiones físicas: Dispone de 100 pines de conexión:
  - 16 pines de los cuales:
    - 5 son GNDs
    - 2 son VDD
    - 2 son 5 voltios (para alimentar los dispositivos que a ella se conecten)
    - 2 son 3 voltios (para alimentar aquellos dispositivos requieran esta tensión).
  - 84 pines para entradas y salidas, configurables por software, distribuidos en puertos de 16bits, etiquetados de la A a la H.
    La nomenclatura de los puertos es PAx, donde x es el bit comprendido entre el 0 y el 15.
    La excepción de esta regla es el puerto H que solo dispone de 2 bits PH0 y PH1.
- 4 leds rojo, verde, azul y naranja, cada uno de un color, conectados físicamente a los pines PD12, PD13, PD14 y PD15.
- 2 botones
  - 1 botón para resetear y reiniciar el sistema (Reset) de color negro.
  - 1 botón azul, denominado botón de usuario, que está disponible para que el usuario lo use en sus programas y está físicamente conectado con el pin PA0.

- 1 acelerómetro para las 3 dimensiones X, Y, Z, para el uso del usuario.
- 1 puerto mini USB para uso del usuario en los programas que requieran el uso de dispositivos USB.
- 1 mini Jack para la conexión de altavoces y/o auriculares.
- Dispone de 6 buses SPI (Serial Peripheral Interface), bus de comunicaciones maestro-esclavo, para conectar dispositivos tales como lectores de tarjetas SD, lectores RFID, etc…

  La configuración de los pines a usar en cada bus es la que se muestra en la siguiente tabla.

	PINS PACK 1			PINS PACK 2			PINS PACK 3			
SPIx	**MOSI**	**MISO**	**SCK**	**MOSI**	**MISO**	**SCK**	**MOSI**	**MISO**	**SCK**	**APB**
SPI1	PA7	PA6	PA5	PB5	PB4	PB3				2
SPI2	PC3	PC2	PB10	PB15	PB14	PB13	PI3	PI2	PI0	1
SPI3	PB5	PB4	PB3	PC12	PC11	PC10				1

SPI4	PE6	PE5	PE2	PE14	PE13	PE12				2
SPI5	PF9	PF8	PF7	PF11	PH7	PH6				2
SPI6	PG14	PG12	PG13							2

Configuración pines para bus SPI

Esta información ha sido extraída de la página http://stm32f4-discovery.net/2014/04/library-05-spi-for-stm32f4xx/ en la cual se presenta la librería para el manejo de forma fácil de los buses SPI y ejemplos de su utilización.

- De igual modo dispone de 3 buses I2C (Inter-Integrated Circuit) o bus de comunicación para dispositivos maestros.

	PINS PACK 2		**PINS PACK 3**		**PINS PACK 1**		
I2Cx	**SCL**	**SDA**	**SCL**	**SDA**	**SCL**	**SDA**	**APB**
I2C1	PB6	PB7	PB8	PB9	PB6	PB9	1
I2C2	PB10	PB11	PF1	PF0	PH4	PH5	1
I2C3	PA8	PC9	PH7	PH8			1

Configuración de pines para el bus I2C

Esta información ha sido extraída de http://stm32f4-discovery.net/2014/05/library-09-i2c-for-stm32f4xx/ donde se presenta una librería para acceso y manejo de forma sencilla del bus I2C, así como ejemplos prácticos de su utilización.

- Una de las grandes ventajas encontradas en esta placa ha sido la enorme cantidad de temporizadores que posee 14 ni más ni menos.
  Los temporizadores son muy necesarios para generar PWM o modulaciones por anchura del pulso.
  La modulación PWM es necesario para el manejo de servos entre otras aplicaciones como puede ser juego de luces o generación de sonidos, ya que con ello se controla el tiempo en que dura una señal.

  La placa STM32F4 proporciona multitud de formas de seleccionar los pines en función del temporizador que se quiera utilizar y se debe hacer la selección en función de la siguiente tabla:

Timer	**Channel 1**			**Channel 2**			**Channel 3**			**Channel 4**		
	PP1	**PP2**	**PP3**	**PP1**	**PP2**	**PP3**	**PP1**	**PP2**	**PP3**	**PP1**	**PP2**	**PP3**
TIM 1	PA8	PE9		PA9	PE10		PA10	PE13		PA11	PE14	

TIM 2	PA0	PA5	PA15	PA1	PB3		PA2	PB10		PA3	PB11	
TIM 3	PA6	PB4	PC6	PA7	PB5	PC7	PB0	PC8		PB1	PC9	
TIM 4	PB6	PD12		PB7	PD13		PB8	PD14		PB9	PD15	
TIM 5	PA0	PH10		PA1	PH11		PA2	PH12		PA3	PI0	
TIM 8	PC6	PI5		PC7	PI6		PC8	PI7		PC9	PI2	
TIM 9	PA2	PE5		PA3	PE6							
TIM 10	PB8	PF6										
TIM 11	PB9	PF7										
TIM 12	PB14	PH6		PB15	PH9							
TIM 13	PA6	PF8										
TIM 14	PA7	PF9										

Configuración de pines según el Timer seleccionado

Esta tabla ha sido obtenida de la página http://stm32f4-discovery.net/2014/09/library-33-pwm-stm32f4xx/ la cual ofrece una librería para la configuración de los distintos Timers que la placa de desarrollo proporciona, así como ejemplos prácticos de ello.

Servos

El elemento estrella de los objetos articulados electrónicos son los servos.

Los servos son un tipo especial de motor de corriente continua los cuales tienen capacidad de alcanzar de forma inmediata en cualquier posición dentro de su intervalo de operación. Para ello, el servomotor espera un tren de pulsos que se corresponde con el movimiento a realizar. Por regla general tienen un margen de operación de 180° aproximadamente.

Un servo es un dispositivo con un eje de rendimiento controlado ya que puede ser llevado a posiciones angulares específicas al enviar una señal codificada. Mientras se mantenga la señal codificada a la entrada, el servo mantendrá la posición angular del engranaje. Cuando la señal codificada cambia, la posición angular del eje cambia.

En la práctica, se usan servos para posicionar elementos de control como palancas, pequeños ascensores y timones. También se usan en radio-control, marionetas y, por supuesto, en robots. Los Servos son sumamente útiles en robótica. Los motores son pequeños sumamente potentes para su tamaño. La corriente que requiere depende del tamaño del servo. Normalmente el fabricante indica cual es la corriente que consume. Eso no significa mucho si

todos los servos van a estar moviéndose todo el tiempo. La corriente depende principalmente del par, y puede exceder un amperio si el servo está enclavado.

Funcionamiento del servo. Control PWM

La **modulación por anchura de pulso**, PWM (*Pulse Width Modulation*), es uno de los sistemas más empleados para el control de servos. Este sistema consiste en generar una onda cuadrada en la que se varía el tiempo que el pulso está a nivel alto, manteniendo el mismo período (normalmente), con el objetivo de modificar la posición del servo según se desee.

Para la generación de una onda PWM en un microcontrolador, lo más habitual es usar un *timer* y un comparador (interrupciones asociadas), de modo que el microcontrolador quede libre para realizar otras tareas, y la generación de la señal sea automática y más efectiva. El mecanismo consiste en programar el *timer* con el ancho del pulso (el período de la señal) y al comparador con el valor de duración del pulso a nivel alto. Cuando se produce una interrupción de *overflow* del *timer*, la subrutina de interrupción debe poner la señal PWM a nivel alto y cuando se produzca la interrupción del comparador, ésta debe poner la señal PWM a nivel bajo. En la actualidad, muchos microcontroladores, como el 68HC08, disponen de hardware específico para realizar esta tarea, eso sí, consumiendo los recursos antes mencionados (*timer* y comparador).

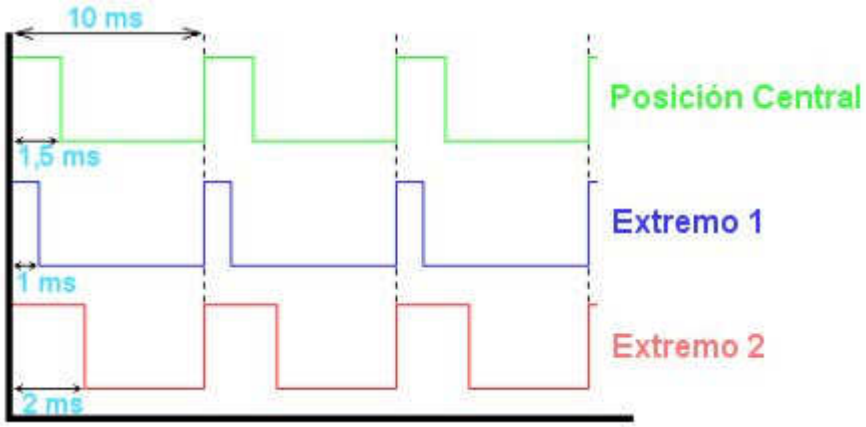
PWM para recorrer todo el rango de operación del servo

El sistema de control de un servo se limita a indicar en qué posición se debe situar. Esto se lleva a cabo mediante una serie de pulsos tal que la duración del pulso indica el ángulo de giro del motor. Cada servo tiene sus márgenes de operación, que se corresponden con el ancho del pulso máximo y mínimo que el servo entiende. Los valores más generales se corresponden con pulsos de entre 1 ms y 2 ms de anchura, que dejarían al motor en ambos extremos (0º y 180º). El valor 1.5 ms indicaría la posición central o neutra (90º), mientras que otros valores del pulso lo dejan en posiciones intermedias. Estos valores suelen ser los recomendados, sin embargo, es posible emplear pulsos menores de 1 ms o mayores de 2 ms, pudiéndose conseguir ángulos mayores de 180°. Si se sobrepasan los límites de movimiento del servo, éste comenzará a emitir un zumbido, indicando que se debe cambiar la longitud del pulso. El factor limitante es el tope del potenciómetro interno y los límites mecánicos constructivos.

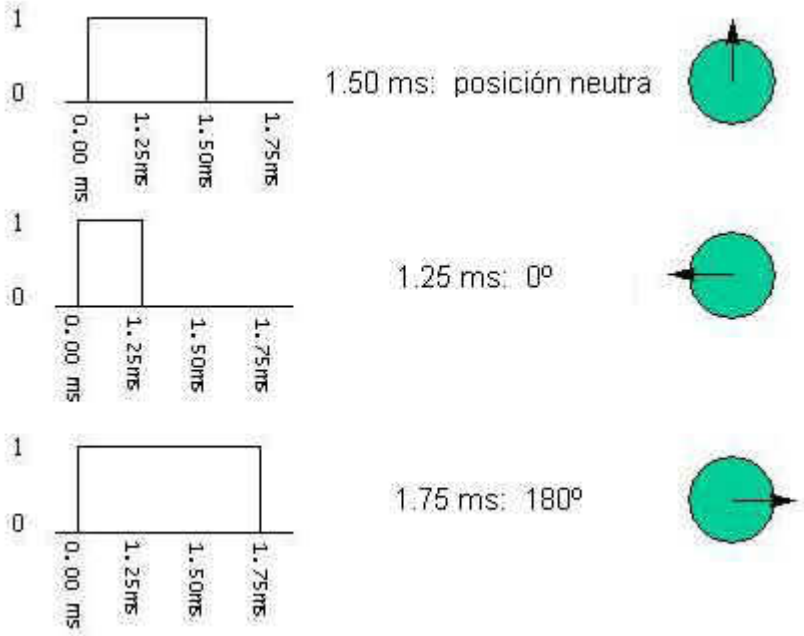
El período entre pulso y pulso (tiempo de OFF) no es crítico, e incluso puede ser distinto entre uno y otro pulso. Se suelen emplear valores ~ 20 ms (entre 10 ms y 30 ms). Si el intervalo entre pulso y pulso es inferior al mínimo, puede interferir con la temporización interna del servo, causando un zumbido, y la vibración del eje de salida. Si es mayor que el máximo, entonces el servo

pasará a estado dormido entre pulsos. Esto provoca que se mueva con intervalos pequeños.

Para que un servo se mantenga en la misma posición durante un cierto tiempo, es necesario enviarle continuamente el pulso correspondiente. De este modo, si existe alguna fuerza que le obligue a abandonar esta posición, intentará resistirse. Si se deja de enviar pulsos (o el intervalo entre pulsos es mayor que el máximo) entonces el servo perderá fuerza y dejará de intentar mantener su posición, de modo que cualquier fuerza externa podría desplazarlo.



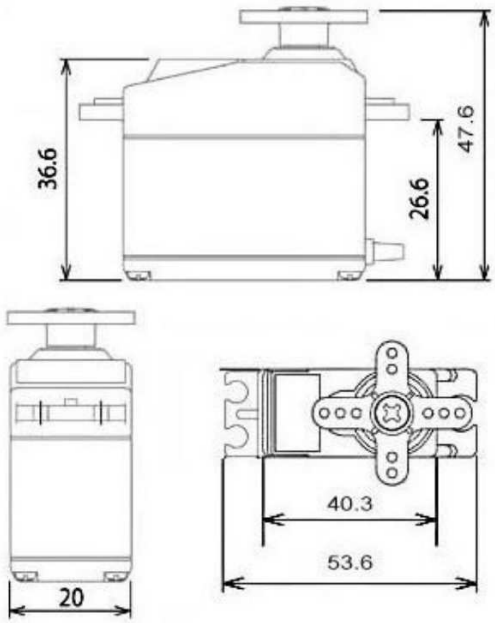
Tren de pulsos para control del servo



Para la ejecución de este proyecto se ha seleccionado el modelo de servo Futaba MG66R.
Este servo es un modelo muy común en radio modelismo.
Sus especificaciones son:

- Peso: 55 g
- Dimensiones: 40.7 x 19.7 x 42.9 mm aproximadamente.

- Par de bloqueo: 9.4 kgf·cm (4.8 V), 11 kgf·cm (6 V)
- Velocidad de Operación: 0.17 s/60º (4.8 V), 0.14 s/60º (6 V)
- Voltaje de funcionamiento: 4.8 V a 7.2 V
- Corriente de funcionamiento 500 mA – 900mA
- Corriente de mantenimiento 2.5 A (6V)
- Ancho de banda de muerto: 5 µs (este es el tiempo que un servo puede mantener la posición entre trenes de pulsos)
- Rango de temperatura de funcionamiento:: 0 ºC a 55 ºC

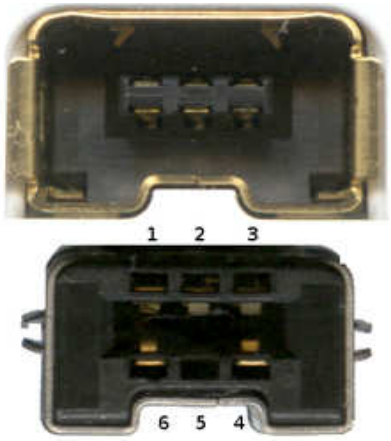


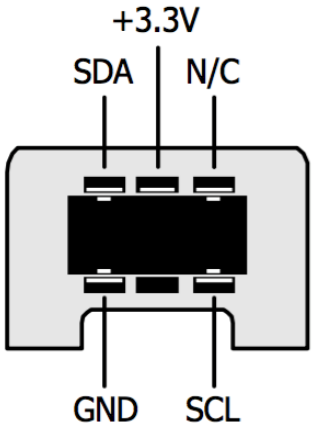
Capítulo 4: Implementación

La implementación se ha dividido en dos fases.
La primera fase es la adquisición de los datos procedentes del Nunchuk.
La segunda fase es la de actualización de Adquisición de Datos

La adquisición de datos se corresponde a la comunicación entre el Nunchuk o Interfaz de control humano y el sistema de adquisición de datos que se corresponde con la placa de desarrollo STM32F4 Discovery.
El objetivo es leer mediante la placa de desarrollo los datos procedentes del Nunchuk los cuales indican el estado de los botones, acelerómetro y joystick.
Esta fase es dividida entre la fase de decisión de la conexión física entre el Nunchuk y la placa de desarrollo y una segunda de codificación del software encargado de la adquisición de datos.

Conexión Física Nunchuk-Stm32f4
El Nunchuk dispone de un conector con seis pines para la comunicación con el sistema de control, como se muestra en la siguiente imagen, pero de ellos sólo se utilizan 4 ya que dos están reservados para futuras revisiones.

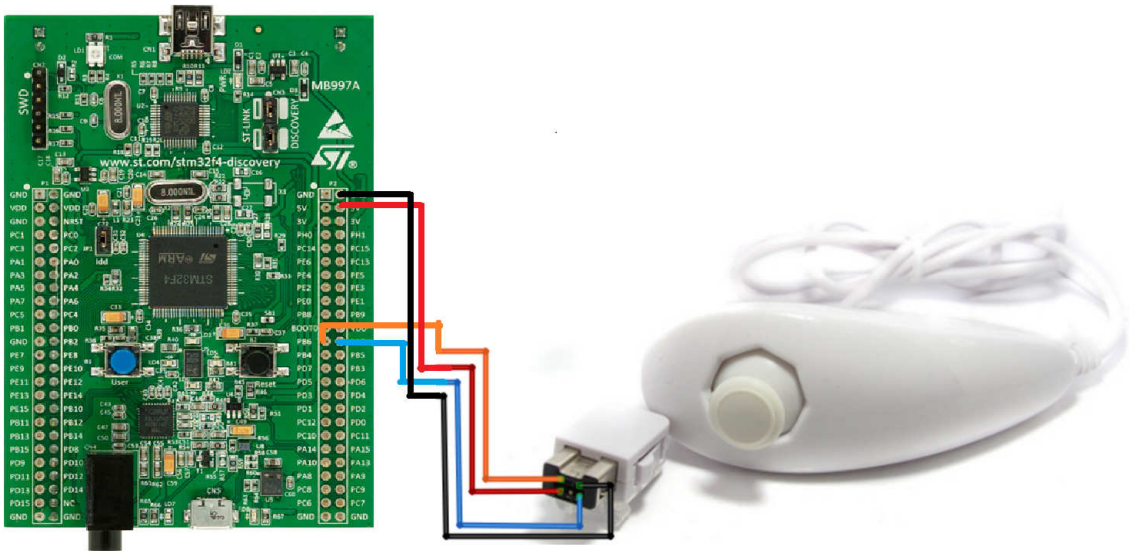




Para impedir realizar soldaduras, cortar dicho conector para acceder a los cables o realizar conexiones poco estables se utiliza un conector como el que se muestra en la siguiente imagen.

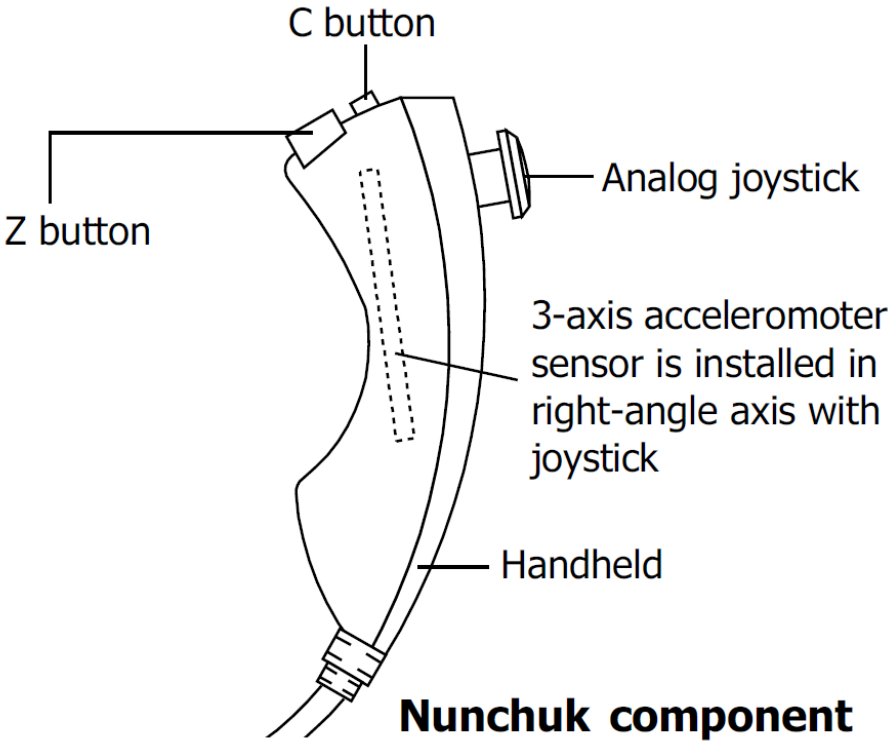
Este conector facilita la conexión o soldadura de cables a usar para conectar el Nunchuk con la placa de desarrollo.
Una vez se tiene el mecanismo que permite la conexión entre el nunchuk y la placa de desarrollo se identifica por las siglas de los pines, y por el datasheet del dispositivo, que se corresponde con un dispositivo que se comunica por bus I2C. (SCL, SDA).
De la tabla de la página http://stm32f4-discovery.net/2014/05/library-09-i2c-for-stm32f4xx/ se puede seleccionar cualquier par de pines SCL y SDA, de los cuales se selecciona el bus I2C1 y el PinsPack 1 lo cual impone el pin PC6 como SCL y el pin PC7 como SDA. Por lo que la conexión final queda:

- El pin marcado con el símbolo “+” se conecta a 5 voltios de la placa de desarrollo. (El datasheet indica que se puede conectar tanto a 3.3 como a 5 voltios).
- El pin marcado con el símbolo “–“ se conecta a la tensión de referencia de la placa de desarrollo.
- El pin marcado como “d” que se corresponde con el pin del conector SDA, se conecta con el pin de la placa de desarrollo etiquetado como PB7.
- Por último el pin marcado como “c” que se corresponde con el pin del conector SCL, se conecta con el pin de la placa de desarrollo etiquetado como PB6.



Desarrollo Software

Del datasheet del nunchuk se extrae información sobre la distribución de los datos en los registros dentro del controlador interno.



Según este datasheet la distribución de los datos está repartida en seis registros consecutivos de un byte (8 bits) cada uno de la siguiente forma:

Data byte receive								Address
Joystick X								0x00
Joystick Y								0x01
Accelerometer X (bit 9 to bit 2 for 10-bit resolution)								0x02
Accelerometer Y (bit 9 to bit 2 for 10-bit resolution)								0x03
Accelerometer Z (bit 9 to bit 2 for 10-bit resolution)								0x04
Accel. Z bit 1	Accel. Z bit 0	Accel. Y bit 1	Accel. Y bit 0	Accel. X bit 1	Accel. X bit 0	C-button	Z-button	0x05

Del datasheet también se obtiene la información que indica que la dirección del microcontrolador interno al Nunchuk es 52 que en hexadecimal es 4A.
Es en esta dirección de donde se debe leer y escribir para acceder a los datos que almacenan las posiciones del acelerómetro, joystick y botones.
Conociendo estos datos se crea una librería de acceso al Nunchuk con las siguientes funciones:

1. Inicialización del bus datos I2C con los datos número de I2C, dirección y frecuencia de funcionamiento.
2. Función para solicitar al Nunchuk que actualice sus registros con los valores actuales. Esto se hace enviando el valor 0x00 a la dirección en la que el nunchuk está registrado (4ª). Esta función ha sido definida con el nombre void nunchuk_sendZero().
3. Función de lectura de los nuevos valores almacenados en los registros del Nunchuk tras. Esta función se ha codificado con el nombre void nunchuk_update();
   Esta función se debe ejecutar después de la función del punto anterior nunchuk_sendZero(). Por lo que la secuencia siempre debe ser:
   - nunchuk_sendZero();

     Esta función envía un valor 0x0 al nuchuck lo que hace que este actualice los registros correspondientes con los valores actuales de su acelerómetro, joystick y botones.

   - nunchuk_update();

     Esta función recupera los valores de los registros internos del nunchuck dejándolos preparados para su consumo mediante las funciones desarrolladas a tal efecto.
4. Funciones para el acceso a cada uno de los valores individuales del Nunchuk correspondientes a los Joysticks:
   Acelerómetro Y, Acelerómetro X, Acelerómetro Z, Joystick X, Joystick Y, Boton C y Boton Z.
   - int nunchuk_getJoystickY();

     Esta función devuelve el valor actualizado del Joystick en el eje Y del plano, que se corresponde con el movimiento hacia adelante y hacia atrás del Joystick.

   - int nunchuk_getJoystickX();

Esta función devuelve el valor actualizado del Joystick en el eje X del plano, que se corresponde con el movimiento hacia la izquierda o la derecha.

Los datos extraídos del Nunchuk pertenecientes a los joysticks están en el rango [-123, 130] y debido a que no es un intervalo simétrico se trunca para que las funciones que devuelven los valores asociados a los joysticks como son nunchuk_getJoystickY() y nunchuk_getJoystickX() valores del rango [-123, 123].

5. Funciones para el acceso a cada uno de los valores individuales de los acelerómetros:
   - int nunchuk_getAccelerometerX();

   Esta función devuleve el valor actualizado del acelerómetro en su eje X.
   - int nunchuk_getAccelerometerY();

   Esta función devuleve el valor actualizado del acelerómetro en su eje Y.
   - int nunchuk_getAccelerometerZ();

   Esta función devuleve el valor actualizado del acelerómetro en su eje Z.

6. Funciones para la obtención de si los botones C y Z han sido pulsados: Del Nunchuk se obtiene un 1 cuando los botones están en reposo, o sea que no están siendo pulsados. Debido a que es más intuitivo que el valor sea 1 cuando se ha producido una pulsación sobre cualquier botón, se hace dicha modificación para que las funciones devuelvan un 1 cuando el botón en cuestión es pulsado y un 0 cuando no está siendo pulsado.
   - int nunchuk_getC();

   Esta función devuelve un 1 si el botón C está siendo pulsado o un 0 en caso contrario.
   - int nunchuk_getZ();

   Esta función devuelve un 1 si el botón Z está siendo pulsado o un 0 en caso contrario.

7. Funciones complejas, a partir de los valores de los acelerómetros, para obtención de la inclinación del Nunchuk respecto a su eje X o su eje Y, los cuales son llamados PITCH y ROLL respectivamente.
   - $pitch = \frac{cos^{-1}\left(\frac{AcelerometroY}{Radius}\right)}{\pi*180}$ -65
   - $roll = \frac{\tan^{-1}\left(AcelerometroX/_{AcelerometroZ}\right)}{\pi*180}$

   Donde Radius = 210.

   Estas funciones han sido codificadas con los nombres:
   - int nunchuk_getRoll();

Esta función devuelve el valor de la inclinación del nunchuk hacia la izquierda o hacia la derecha.

- int nunchuk_getPitch();

Esta función devuelve el valor de la inclinación del nunchuk hacia adelante o hacia atrás.

Estas ecuaciones están acotadas para valores dentro del rango [-65, 70]. Del mismo modo que ocurre con las funciones para obtener los valores de los joysticks, las funciones nunchuk_getRoll() y nunchuk_getPitch() son normalizadas para devolver valores en el rango [-65, 65]

Estas funciones quedan codificadas en los archivos **nunchuk.h** y **nunchuk.c** y serán librería para el uso en la implementación del proyecto completo.

Pruebas de Nunchuk

Para realizar las pruebas se debe utilizar el debug.

Colocando un punto de ruptura justo en el que ya han sido leidos todos los valores del nunchuk y utilizando la vista de variables puede verse el valor obtenido.

Código	Acción	Valor esperado	Valor obtenido
NUN000001	Dejar el Joystick X en su posición de reposo.	El valor obtenido del Joystick X debe ser 0	Se obtiene valor 0 en el Joystick X
NUN000002	Llevar el Joystick X hasta la posición más extrema hacia la izquierda	El valor obtenido del Joystick X debe ser -123	Se obtiene un valor -123 en la variable sobre la que se ha leído el Joystick X.
NUN000003	Llevar el Joystick X hasta la posición más extrema hacia la derecha	El valor obtenido del Joystick X debe ser 123	Se obtiene un valor 123 en la variable sobre la que se ha leído el Joystick X.
NUN000004	Llevar el Joystick X hasta la posición intermedia entre el centro y el extremo izquierdo.	El valor obtenido del Joystick X debe ser un valor intermedio al rango [-123, 0]	Se obtiene un valor entre -123 y 0 en el Joystick X proporcional a la inclinación de este.
NUN000005	Llevar el Joystick X hasta la posición intermedia entre el centro y el extremo derecho.	El valor obtenido del Joystick X debe ser un valor intermedio al rango [0, 123]	Se obtiene un valor entre 0 y 123 proporcional a la inclinación del Joystick.
NUN000006	Dejar el Joystick Y en su posición de reposo.	El valor obtenido del Joystick Y debe ser 0	Se obtiene valor 0 en el Joystick Y
NUN000007	Llevar el Joystick Y hasta la posición más extrema hacia la izquierda	El valor obtenido del Joystick Y debe ser -123	Se obtiene un valor -123 en la variable sobre la que se ha leído el Joystick Y.
NUN000008	Llevar el Joystick Y hasta la posición más extrema	El valor obtenido del Joystick Y debe	Se obtiene un valor 123 en la variable

	hacia la derecha	ser 123	sobre la que se ha leído el Joystick Y.
NUN000009	Llevar el Joystick Y hasta la posición intermedia entre el centro y el extremo izquierdo.	El valor obtenido del Joystick Y debe ser un valor intermedio al rango [-123, 0]	Se obtiene un valor entre -123 y 0 proporcional a la inclinación del Joystick.
NUN000010	Llevar el Joystick Y hasta la posición intermedia entre el centro y el extremo derecho.	El valor obtenido del Joystick Y debe ser un valor intermedio al rango [0, 123]	Se obtiene un valor entre 0 y 123 proporcional a la inclinación del Joystick.
NUN000011	Poner el Nunchuk en posición horizontal	El valor obtenido del Pitch debe ser 0	Se obtiene un valor 0 en la variable donde se ha obtenido el valor Pitch.
NUN000012	Inclinar el Nunchuk hasta ponerlo perpendicular al suelo con los botones hacia abajo.	El valor obtenido del Pitch debe ser -65	Se obtiene un valor -65 en la variable donde se ha obtenido el valor Pitch.
NUN000013	Inclinar el Nunchuk hasta ponerlo perpendicular al suelo con los botones hacia arriba.	El valor obtenido del Pitch debe ser 65	Se obtiene un valor 65 en la variable donde se ha obtenido el valor Pitch.
NUN000014	Inclinar el Nunchuk hasta ponerlo en una posición a medio camino entre la horizontal y la perpendicular con los botones hacia adelante.	El valor obtenido del Pitch debe ser un valor comprendido entre -65 y 0.	Se obtiene un valor dentro del rango [-65, 0] en la variable donde se ha obtenido el valor Pitch proporcional a la inclinación del Nunchuk.
NUN000015	Inclinar el Nunchuk hasta ponerlo en una posición a medio camino entre la horizontal y la perpendicular con los botones hacia atrás.	El valor obtenido del Pitch debe ser un valor comprendido entre 0 y 65	Se obtiene un valor dentro del rango [0, 65] en la variable donde se ha obtenido el valor Pitch proporcional a la inclinación del Nunchuk.
NUN000016	Poner el Nunchuk en posición horizontal	El valor obtenido del Roll debe ser 0	Se obtiene un valor 0 en la variable donde se ha obtenido el valor Roll.
NUN000017	Girar el Nunchuk hacia la izquierda rotando la muñeca 90 grados.	El valor obtenido del Roll debe ser -65	Se obtiene un valor -65 en la variable donde se ha obtenido el valor Roll.
NUN000018	Girar el Nunchuk hacia la	El valor obtenido	Se obtiene un valor

	derecha rotando la muñeca 90 grados.	del Pitch debe ser 65	65 en la variable donde se ha obtenido el valor Roll.
NUN000019	Girar el Nunchuk hacia la izquierda, rotando la muñeca, sin llegar a los 90 grados.	El valor obtenido del Roll debe ser un valor comprendido entre -65 y 0.	Se obtiene un valor dentro del rango [-65, 0] en la variable donde se ha obtenido el valor Roll proporcional a la rotación del Nunchuk.
NUN000020	Girar el Nunchuk hacia la derecha, rotando la muñeca, sin llegar a los 90 grados.	El valor obtenido del Roll debe ser un valor comprendido entre 0 y 65	Se obtiene un valor dentro del rango [0, 65] en la variable donde se ha obtenido el valor Roll proporcional a la rotación del Nunchuk.
NUN000021	Dejar el botón C sin pulsar.	El valor obtenido de la pulsación del botón C debe ser 0.	Se obtiene el valor 0 en la variable donde se ha volcado el estado del botón C.
NUN000022	Pulsar el botón C.	El valor obtenido de la pulsación del botón C debe ser 1.	Se obtiene el valor 1 en la variable donde se ha volcado el estado del botón C.
NUN000022	Dejar el botón Z sin pulsar.	El valor obtenido de la pulsación del botón Z debe ser 0.	Se obtiene el valor 0 en la variable donde se ha volcado el estado del botón Z.
NUN000024	Pulsar el botón Z.	El valor obtenido de la pulsación del botón Z debe ser 1.	Se obtiene el valor 1 en la variable donde se ha volcado el estado del botón Z.

Estas pruebas impiden comprobar la velocidad de actualización de los valores en el nunchuk así como la depuración de errores y la comprobación de la precisión de los movimientos.

LCD 1602

A causa de esta dificultad para comprobar el correcto funcionamiento de la lectura del Nunchuk se llega a la decisión de utilizar un LCD 1602 (LCD de 16 columnas y 2 filas) así como la programación para el uso de esta, para visualizar en tiempo real los valores obtenidos de la lectura del Nunchuk

El resultado de la codificación de la librería para el uso del LCD 1602 son los archivo **lcd_hd1602.h** y **lcd_hd1602.c**.

Las funcionalidades que se codifican en estas librerias son:

- lcd_init()

Inicialización del puerto a usar para escribir en el LCD.

- lcd_locate(X, Y)

Esta función posiciona el cursor dentro de la fila Y, columna X. Esta posición es donde el LCD escribirá cuando la siguiente función de escritura sea invocada.

- lcd_str(cad)

Esta función escribirá la cadena str donde el cursor esté situado, o en la fila 1 columna 1 por defecto.
Si la cadena de caracteres str tiene más de dieciséis caracteres será truncada en la pantalla pudiendo verse solo los caracteres que quepan en la fila dependiendo de la posición inicial. En ningún caso el LCD saltará de línea para mostrar la cadena completa.

- lcd_int(num)

Esta función escribirá el numero entero num donde el cursor esté situado, o en la fila 1 columna 1 por defecto.
Si el número num tiene más de dieciséis dígitos, incluyendo el signo, será truncado en la pantalla pudiendo verse solo los dígitos que quepan en la fila dependiendo de la posición inicial. En ningún caso el LCD saltará de línea para mostrar el número completo.

- lcd_strxy(X, Y, str);

Esta es una función que combina la función de posicionamiento con la de escribir cadenas en el LCD, escribiendo la cadena str en la columna Y fila X.

- lcd_int(X, Y, num);

Esta es una función que combina la función de posicionamiento con la de escribir número en el LCD, escribiendo el número num en la columna Y fila X.

Adicionalmente en el fichero **lcd_hd1602.h** se configura el puerto a utilizar para así poder elegir que puerto conectar al LCD, que por defecto se establece el puerto “D” usando los pines 0,2,4 para reinicio (reset), activación (CE) y escritura/lectura (W/R) y los pines 1,3,5,7 para los transmitir los datos a mostrar en el LCD. Se usan solo cuatro pines para los datos ya que sólo se van a mostrar caracteres del alfanuméricos.

Pruebas LCD

Una vez finalizada la librería de manejo del LCD se pasa a la fase de pruebas, para las cuales se define el siguiente plan de pruebas:

Código	Acción	Valor esperado	Valor obtenido
LCD000001	Usar la función lcd_str(“Hola”);	El LCD debe	El LCD muestra la

		mostrar la palabra “Hola” en la primera fila primera columna	palabra “Hola” a partir de la primera columna de la primera fila.
LCD000002	Usar la función lcd_str(“Hola que tal estas?”);	El LCD debe mostrar la cadena “Hola que tal est” en la primera fila a partir de la primera columna	El LCD muestra la cadena “Hola que tal est” a partir de la primera columna de la primera fila
LCD000003	Usar las funciones lcd_locate(5, 1); lcd_str(“Hola”);	El LCD debe mostrar la palabra “Hola” en la primera fila quinta columna	El LCD muestra la palabra “Hola” a partir de la quinta columna de la primera fila.
LCD000004	Usar las funciones lcd_locate(5, 2); lcd_str(“Hola”);	El LCD debe mostrar la palabra “Hola” en la segunda fila quinta columna	El LCD muestra la palabra “Hola” a partir de la quinta columna de la segunda fila.
LCD000005	Usar las funciones lcd_locate(5, 1); lcd_str(“Hola que tal estas?”);	El LCD debe mostrar la cadena “Hola que tal” en la primera fila a partir de la quinta columna	El LCD muestra la cadena “Hola que tal” a partir de la quinta columna de la primera fila
LCD000006	Usar las funciones lcd_locate(5, 2); lcd_str(“Hola que tal estas?”);	El LCD debe mostrar la cadena “Hola que tal” en la segunda fila a partir de la quinta columna	El LCD muestra la cadena “Hola que tal” a partir de la quinta columna de la segunda fila
LCD000007	Usar la función lcd_int(123456);	El LCD debe mostrar el número 123456 en la primera fila primera columna	El LCD muestra el número 123456 a partir de la primera columna de la primera fila.
LCD000008	Usar la función lcd_int(1234567890123456789);	El LCD debe mostrar el número 1234567890123456 en la primera fila a partir de la primera columna	El LCD muestra mostrar el número 1234567890123456 a partir de la primera columna de la primera fila
LCD000009	Usar las funciones lcd_locate(5, 1); lcd_int(1234567890123456789);	El LCD debe mostrar el número 123456789012 en la primera fila quinta columna	El LCD muestra el número 123456789012 a partir de la quinta columna de la primera fila.
LCD000010	Usar las funciones lcd_locate(5, 2); lcd_int(1234567890123456789);	El LCD debe mostrar el número 123456789012 en la segunda fila quinta columna	El LCD muestra el número 123456789012 a partir de la quinta columna de la

			segunda fila.
LCD000011	Usar la función lcd_int(-123456);	El LCD debe mostrar el número -123456 en la primera fila primera columna	El LCD muestra el número -123456 a partir de la primera columna de la primera fila.
LCD000012	Usar la función lcd_int(-1234567890123456789);	El LCD debe mostrar el número -123456789012345 en la primera fila a partir de la primera columna	El LCD muestra mostrar el número -123456789012345 a partir de la primera columna de la primera fila
LCD000013	Usar las funciones lcd_locate(5, 1); lcd_int(-1234567890123456789);	El LCD debe mostrar el número -12345678901 en la primera fila quinta columna	El LCD muestra el número -12345678901 a partir de la quinta columna de la primera fila.
LCD000014	Usar las funciones lcd_locate(5, 2); lcd_int(-1234567890123456789);	El LCD debe mostrar el número -12345678901 en la segunda fila quinta columna	El LCD muestra el número -12345678901 a partir de la quinta columna de la segunda fila.
LCD000015	Usar la función lcd_strxy(6,2,"Hola");	El LCD debe mostrar la palabra "Hola" en la segunda fila sexta columna	El LCD muestra la palabra "Hola" a partir de la sexta columna de la segunda fila.
LCD000016	Usar la función lcd_strxy(6,1,"Hola que tal estas");	El LCD debe mostrar la palabra "Hola que ta" en la primera fila sexta columna	El LCD muestra la palabra "Hola que ta" de la sexta columna de la primera fila.
LCD000017	Usar la función lcd_intxy(3,2,1234567890123456);	El LCD debe mostrar el número 12345678901234 en la segunda fila tercera columna	El LCD muestra l número 12345678901234 a partir de la tercera columna de la segunda fila.
LCD000018	Usar la función lcd_intxy(4,1,-123456789012345);	El LCD debe mostrar el número -123456789012 en la primera fila cuarta columna	El LCD muestra l número -123456789012 a partir de la cuarta columna de la primera fila.

Pruebas Nunchuk-LCD

El uso combinado de la librería para el manejo del LCD y la librería de adquisición de datos del Nunchuk posibilita ver en tiempo real la evolución los distintos valores obtenidos del nunchuk y de este modo ajustar de forma más precisa el tiempo de retardo a utilizar entre lectura y lectura del nunchuk.

En el LCD se añaden las etiquetas X:, Y:, C, R:, P:, Z para acompañar a los valores del joystickX, joystickY, botón C, Roll, Pitch y botón Z respectivamente. Pasando de nuevo las pruebas se obtiene el siguiente plan de pruebas:

Código	Acción	Valor esperado	Valor obtenido
NUNLCD001	Dejar el Joystick X en su posición de reposo.	El LCD debe mostrar el valor 0 junto a la etiqueta X:	Junto a la etiqueta X: aparece el valor 0
NUNLCD002	Llevar el Joystick X hasta la posición más extrema hacia la izquierda	El LCD debe mostrar el valor -123 junto a la etiqueta X:	Junto a la etiqueta X: aparece el valor -123
NUNLCD003	Llevar el Joystick X hasta la posición más extrema hacia la derecha	El LCD debe mostrar el valor 123 junto a la etiqueta X:	Junto a la etiqueta X: aparece el valor 123
NUNLCD004	Llevar el Joystick X hasta la posición intermedia entre el centro y el extremo izquierdo.	El LCD debe mostrar un valor entre -123 y 0 junto a la etiqueta X:	Junto a la etiqueta X: aparece un valor entre -123 y 0
NUNLCD005	Llevar el Joystick X hasta la posición intermedia entre el centro y el extremo derecho.	El LCD debe mostrar un valor entre 0 y 123 junto a la etiqueta X:	Junto a la etiqueta X: aparece un valor entre 0 y 123.
NUNLCD006	Dejar el Joystick Y en su posición de reposo.	El LCD debe mostrar el valor 0 junto a la etiqueta Y:	Junto a la etiqueta Y: aparece el valor 0
NUNLCD007	Llevar el Joystick Y hasta la posición más extrema hacia la izquierda	El LCD debe mostrar el valor -123 junto a la etiqueta Y:	Junto a la etiqueta Y: aparece el valor -123
NUNLCD008	Llevar el Joystick Y hasta la posición más extrema hacia la derecha	El LCD debe mostrar el valor 123 junto a la etiqueta Y:	Junto a la etiqueta Y: aparece el valor 123
NUNLCD009	Llevar el Joystick Y hasta la posición intermedia entre el centro y el extremo izquierdo.	El LCD debe mostrar un valor entre -123 y 0 junto a la etiqueta Y:	Junto a la etiqueta Y: aparece un valor entre -123 y 0
NUNLCD010	Llevar el Joystick Y hasta la posición intermedia entre el centro y el extremo derecho.	El LCD debe mostrar un valor entre 0 y 123 junto a la etiqueta Y:	Junto a la etiqueta Y: aparece un valor entre 0 y 123.
NUNLCD011	Dejar el Nunchuk en posición horizontal	El LCD debe mostrar valor 0 junto a la etiqueta P:	Junto a la etiqueta P: aparece el valor 0.
NUNLCD012	Inclinar el Nunchuk hasta ponerlo perpendicular al suelo con los botones hacia abajo.	El LCD debe mostrar valor -65 junto a la etiqueta P:	Junto a la etiqueta P: aparece el valor -65.
NUNLCD013	Inclinar el Nunchuk hasta ponerlo perpendicular al suelo con los botones hacia arriba.	El LCD debe mostrar valor 65 junto a la etiqueta P:	Junto a la etiqueta P: aparece el valor 65.
NUNLCD014	Inclinar el Nunchuk hasta ponerlo en una posición a medio camino entre la horizontal y la	El LCD debe mostrar un valor entre -65 y 0 junto a la etiqueta P:	Junto a la etiqueta P: aparece un valor entre -65 y 0.

	perpendicular con los botones hacia adelante.		
NUNLCD015	Inclinar el Nunchuk hasta ponerlo en una posición a medio camino entre la horizontal y la perpendicular con los botones hacia atrás.	El LCD debe mostrar un valor entre 0 y 65 junto a la etiqueta P:	Junto a la etiqueta P: aparece un valor entre 0 y 65.
NUNLCD016	Dejar el Nunchuk en posición horizontal	El LCD debe mostrar valor 0 junto a la etiqueta R:	Junto a la etiqueta R: aparece el valor 0.
NUNLCD017	Girar el Nunchuk hacia la izquierda rotando la muñeca 90 grados.	El LCD debe mostrar valor -65 junto a la etiqueta R:	Junto a la etiqueta R: aparece el valor -65.
NUNLCD018	Girar el Nunchuk hacia la derecha rotando la muñeca 90 grados.	El LCD debe mostrar valor 65 junto a la etiqueta R:	Junto a la etiqueta R: aparece el valor 65.
NUNLCD019	Girar el Nunchuk hacia la izquierda, rotando la muñeca, sin llegar a los 90 grados.	El LCD debe mostrar un valor entre -65 y 0 junto a la etiqueta R:	Junto a la etiqueta R: aparece un valor entre -65 y 0.
NUNLCD020	Girar el Nunchuk hacia la derecha, rotando la muñeca, sin llegar a los 90 grados.	El LCD debe mostrar un valor entre 0 y 65 junto a la etiqueta R:	Junto a la etiqueta R: aparece un valor entre 0 y 65.
NUNLCD021	Dejar el botón C sin pulsar.	El LCD debe mostrar el valor 0 junto a la etiqueta C.	Junto a la etiqueta C aparece el valor 0.
NUNLCD022	Pulsar el botón C.	El LCD debe mostrar el valor 1 junto a la etiqueta C.	Junto a la etiqueta C aparece el valor 1.
NUNLCD022	Dejar el botón Z sin pulsar.	El LCD debe mostrar el valor 0 junto a la etiqueta Z.	Junto a la etiqueta Z aparece el valor 0.
NUNLCD024	Pulsar el botón Z.	El LCD debe mostrar el valor 1 junto a la etiqueta Z.	Junto a la etiqueta Z aparece el valor 1.

Control del Servos

En el enlace http://stm32f4-discovery.net/2014/10/library-42-control-rc-servo-stm32f4/ hay una extensa explicación y demostración de trabajo con servos.
En este enlace se propone un ejemplo en el cual se inicializan dos servos, uno conectado al pin PA5 y otro al PB3 y ambos usan el mismo Timer.
Una vez inicializados dentro de un bucle infinito se posicionan ambos servos en los angulos 0º, 90º y 180º haciendo una pausa de dos segundos entre movimiento y movimiento.

```
#include "stm32f4xx.h"
/* Include my libraries here */
#include "defines.h"
#include "tm_stm32f4_delay.h"
#include "tm_stm32f4_servo.h"
```

```
int main(void) {
  /* Servo structs */
  TM_SERVO_t Servo1, Servo2;

  /* Initialize system */
  SystemInit();

  /* Initialize delay */
  TM_DELAY_Init();

  /* Initialize servo 1, TIM2, Channel 1, Pinspack 2 = PA5 */
  TM_SERVO_Init(&Servo1, TIM2, TM_PWM_Channel_1, TM_PWM_PinsPack_2);

  /* Initialize servo 2, TIM2, Channel 2, Pinspack 2 = PB3 */
  TM_SERVO_Init(&Servo2, TIM2, TM_PWM_Channel_2, TM_PWM_PinsPack_2);

  while (1) {
    /* 0 degrees rotation on servo 1 */
    TM_SERVO_SetDegrees(&Servo1, 0);
    /* 180 degrees on servo 2 */
    TM_SERVO_SetDegrees(&Servo2, 180);
    /* 2s delay */
    Delayms(2000);

    /* 90 degrees rotation */
    TM_SERVO_SetDegrees(&Servo1, 90);
    /* 90 degrees on servo 2 */
    TM_SERVO_SetDegrees(&Servo2, 90);
    /* 2s delay */
    Delayms(2000);

    /* 180 degrees rotation */
    TM_SERVO_SetDegrees(&Servo1, 180);
    /* 0 degrees on servo 2 */
    TM_SERVO_SetDegrees(&Servo2, 0);
    /* 2s delay */
    Delayms(2000);
  }
}
```

Para ello se deben descargar las librerías que este ejemplo usa **tm_stm32f4_delay** y **tm_stm32f4_servo.**
Como se puede comprobar esta librería permite trabajar con grados en lugar de anchura de pulsos mediante la operación **TM_SERVO_SetDegrees(&Servo, Grados)**, esto hace que su utilización sea más sencilla e intuitiva.
Cabe destacar que en la ejecución de este ejemplo efectivamente los servos se movían entre tres posiciones distintas, pero en lugar de ser 0º, 90º y 180º se trataba de 0º, 45º y 90º.
Para subsanar este problema fue necesario editar el archivo tm_stm32f4_servo.h y modificar las constantes SERVO_MICROS_MIN que

originalmente tenía un valor 1000 por el valor 500 y la constante SERVO_MICROS_MAX cuyo valor original era 2000 por el valor 2300.
Con esta modificación se consigue que el rango de operación del servo esté entre 0º y 180º.

Una vez funciona el ejemplo que el tutorial propone, el siguiente paso es ampliar el número servos hasta conseguir controlar 6 servos de forma independiente.
Atendiendo a la tabla que en la misma página contiene

Timer	Channel 1			Channel 2			Channel 3			Channel 4		
	PP1	PP2	PP3	PP1	PP2	PP3	PP1	PP2	PP3	PP1	PP2	PP3
TIM 1	PA8	PE9		PA9	PE10		PA10	PE13		PA11	PE14	
TIM 2	PA0	PA5	PA15	PA1	PB3		PA2	PB10		PA3	PB11	
TIM 3	PA6	PB4	PC6	PA7	PB5	PC7	PB0	PC8		PB1	PC9	
TIM 4	PB6	PD12		PB7	PD13		PB8	PD14		PB9	PD15	
TIM 5	PA0	PH10		PA1	PH11		PA2	PH12		PA3	PI0	
TIM 8	PC6	PI5		PC7	PI6		PC8	PI7		PC9	PI2	
TIM 9	PA2	PE5		PA3	PE6							
TIM 10	PB8	PF6										
TIM 11	PB9	PF7										
TIM 12	PB14	PH6		PB15	PH9							
TIM 13	PA6	PF8										
TIM 14	PA7	PF9										

Tras muchas pruebas infructíferas se llega a la conclusión de que pines de un mismo puerto no pueden ser utilizados con distintos Timers.
Esto significa, que si, por ejemplo, se usan los pines PA0, PA1, PA2 y PA3 con Timer 2 con la configuración PinsPack 1 y en los canales 1, 2, 3 y 4 respectivamente ya no sería posible utilizar los pines PA6 y PA7 con el Timer 3 con la configuración PinsPack 1 ya que se tendrían pines de un mismo puerto en dos timers distintos.
Eso hace que las posibilidades para manejar simultáneamente seis servos de forma independiente quedan drásticamente reducidas.

La siguiente prueba que se realiza es la de configurar los pines PA0 y PA1 con el Timer 5 en la configuración PinsPack 1 y el canal 1 y dos respectivamente.
Y los pines PD12, PD13, PD14 y PD5 con el Timer 4 y en su configuración PinsPack 2 y los canales 1, 2, 3 y 4 respectivamente.
Esta prueba es satisfactoria y su configuración es candidata a ser utilizada para el proyecto final. Solo se detectan vibraciones en el servo conectado al pin PD14.

También se hace la prueba para afianzar conocimientos configurar los pines PA0 y PA1 con el Timer 5 en la configuración PinsPack 1 y el canal 1 y dos respectivamente.
Y los pines PD12 y PD13 con el Timer 4 y en su configuración PinsPack 2 y los canales 1 y 2 respectivamente.
Y por último los pines PC6 y PC7 con el Timer 8 en la configuración PinsPack 1 y los canales 1 y 2 respectivamente.
Esta prueba es igualmente fructífera y se observa ausencia vibraciones en quinto servo anteriormente conectado al pin PD14 y ahora conectado al pin PC6.

Por esto se selecciona la configuración en la que se usan los pines PA0, PA1, PD12, PD13, PC6 y PC7 en la siguiente configuración:

Timer	Channel 1			Channel 2		
	PP1	PP2	PP3	PP1	PP2	PP3
TIM 4		PD12			PD13	
TIM 5	PA0			PA1		
TIM 8	PC6			PC7		

Pruebas de los Servos

Para comenzar las pruebas los servos se inicializan en 90º mediante las instrucciones.

```
#include "stm32f4_discovery.h"
TM_SERVO_t ShoulderHorizontal, ShoulderVertical, Cubit,
            WristVertical, WristRotation, Claw;
TM_SERVO_Init(&ShoulderHorizontal,        TIM5,        TM_PWM_Channel_1,
TM_PWM_PinsPack_1);     //PA0  Green -> Shoulder Horizontal
TM_SERVO_Init(&ShoulderVertical,         TIM5,        TM_PWM_Channel_2,
TM_PWM_PinsPack_1);     //PA1  Yellow -> Shoulder Vertical

TM_SERVO_Init(&Cubit, TIM4, TM_PWM_Channel_1, TM_PWM_PinsPack_2);
                    //PD12  Orange -> Cubit
TM_SERVO_Init(&WristVertical,          TIM4,          TM_PWM_Channel_2,
TM_PWM_PinsPack_2);           //PD13  Red -> Wrist Vertical
```

```
TM_SERVO_Init(&WristRotation,          TIM8,          TM_PWM_Channel_1,
TM_PWM_PinsPack_1);          //PC6 Grey -> Wrist Rotation
TM_SERVO_Init(&Claw, TIM8, TM_PWM_Channel_2, TM_PWM_PinsPack_1);
                  //PC7 Black -> Claw
TM_SERVO_SetDegrees(&ShoulderHorizontal, 90);
TM_SERVO_SetDegrees(&ShoulderVertical, 90);

TM_SERVO_SetDegrees(&Cubit, 90);

TM_SERVO_SetDegrees(&WristRotation, 90);
TM_SERVO_SetDegrees(&WristVertical, 90);
TM_SERVO_SetDegrees(&Claw, 90);
```

Código	Acción	Valor esperado	Valor obtenido
SER000001	Posicionar el servo conectado al pin PA0 a la posición 0º	El servo conectado al pin PA0 debe posicionarse en el ángulo 0º.	El servo conectado al pin PA0 se posiciona la posición correspondiente a 0º.
SER000002	Posicionar el servo conectado al pin PA0 a la posición 45º	El servo conectado al pin PA0 debe posicionarse en el ángulo 45º.	El servo conectado al pin PA0 se posiciona la posición correspondiente a 45º.
SER000003	Posicionar el servo conectado al pin PA0 a la posición 90º	El servo conectado al pin PA0 debe posicionarse en el ángulo 90º.	El servo conectado al pin PA0 se posiciona la posición correspondiente a 90º.
SER000004	Posicionar el servo conectado al pin PA0 a la posición 135º	El servo conectado al pin PA0 debe posicionarse en el ángulo 135º.	El servo conectado al pin PA0 se posiciona la posición correspondiente a 135º.
SER000005	Posicionar el servo conectado al pin PA0 a la posición 180º	El servo conectado al pin PA0 debe posicionarse en el ángulo 180º.	El servo conectado al pin PA0 se posiciona la posición correspondiente a 180º.
SER000006	Posicionar el servo conectado al pin PA1 a la posición 0º	El servo conectado al pin PA1 debe posicionarse en el ángulo 0º.	El servo conectado al pin PA1 se posiciona la posición correspondiente a 0º.
SER000007	Posicionar el servo conectado al pin PA1 a la posición 45º	El servo conectado al pin PA1 debe posicionarse en el ángulo 45º.	El servo conectado al pin PA1 se posiciona la posición correspondiente a

			45º.
SER000008	Posicionar el servo conectado al pin PA1 a la posición 90º	El servo conectado al pin PA1 debe posicionarse en el ángulo 90º.	El servo conectado al pin PA1 se posiciona la posición correspondiente a 90º.
SER000009	Posicionar el servo conectado al pin PA1 a la posición 135º	El servo conectado al pin PA1 debe posicionarse en el ángulo 135º.	El servo conectado al pin PA1 se posiciona la posición correspondiente a 135º.
SER000010	Posicionar el servo conectado al pin PA1 a la posición 180º	El servo conectado al pin PA1 debe posicionarse en el ángulo 180º.	El servo conectado al pin PA1 se posiciona la posición correspondiente a 180º.
SER000011	Posicionar el servo conectado al pin PD12 a la posición 0º	El servo conectado al pin PD12 debe posicionarse en el ángulo 0º.	El servo conectado al pin PD12 se posiciona la posición correspondiente a 0º.
SER000012	Posicionar el servo conectado al pin PD12 a la posición 45º	El servo conectado al pin PD12 debe posicionarse en el ángulo 45º.	El servo conectado al pin PD12 se posiciona la posición correspondiente a 45º.
SER000013	Posicionar el servo conectado al pin PD12 a la posición 90º	El servo conectado al pin PD12 debe posicionarse en el ángulo 90º.	El servo conectado al pin PD12 se posiciona la posición correspondiente a 90º.
SER000014	Posicionar el servo conectado al pin PD12 a la posición 135º	El servo conectado al pin PD12 debe posicionarse en el ángulo 135º.	El servo conectado al pin PD12 se posiciona la posición correspondiente a 135º.
SER000015	Posicionar el servo conectado al pin PD12 a la posición 180º	El servo conectado al pin PD12 debe posicionarse en el ángulo 180º.	El servo conectado al pin PD12 se posiciona la posición correspondiente a 180º.
SER000016	Posicionar el servo conectado al pin PD13 a la posición 0º	El servo conectado al pin PD13 debe posicionarse en el ángulo 0º.	El servo conectado al pin PD13 se posiciona la posición correspondiente a

			0º.
SER000017	Posicionar el servo conectado al pin PD13 a la posición 45º	El servo conectado al pin PD13 debe posicionarse en el ángulo 45º.	El servo conectado al pin PD13 se posiciona la posición correspondiente a 45º.
SER000018	Posicionar el servo conectado al pin PD13 a la posición 90º	El servo conectado al pin PD13 debe posicionarse en el ángulo 90º.	El servo conectado al pin PD13 se posiciona la posición correspondiente a 90º.
SER000019	Posicionar el servo conectado al pin PD13 a la posición 135º	El servo conectado al pin PD13 debe posicionarse en el ángulo 135º.	El servo conectado al pin PD13 se posiciona la posición correspondiente a 135º.
SER000020	Posicionar el servo conectado al pin PD13 a la posición 180º	El servo conectado al pin PD13 debe posicionarse en el ángulo 180º.	El servo conectado al pin PD13 se posiciona la posición correspondiente a 180º.
SER000021	Posicionar el servo conectado al pin PC6 a la posición 0º	El servo conectado al pin PC6 debe posicionarse en el ángulo 0º.	El servo conectado al pin PC6 se posiciona la posición correspondiente a 0º.
SER000022	Posicionar el servo conectado al pin PC6 a la posición 45º	El servo conectado al pin PC6 debe posicionarse en el ángulo 45º.	El servo conectado al pin PC6 se posiciona la posición correspondiente a 45º.
SER000023	Posicionar el servo conectado al pin PC6 a la posición 90º	El servo conectado al pin PC6 debe posicionarse en el ángulo 90º.	El servo conectado al pin PC6 se posiciona la posición correspondiente a 90º.
SER000024	Posicionar el servo conectado al pin PC6 a la posición 135º	El servo conectado al pin PC6 debe posicionarse en el ángulo 135º.	El servo conectado al pin PC6 se posiciona la posición correspondiente a 135º.
SER000025	Posicionar el servo conectado al pin PC6 a la posición 180º	El servo conectado al pin PC6 debe posicionarse en el ángulo 180º.	El servo conectado al pin PC6 se posiciona la posición correspondiente a

			180º.
SER000026	Posicionar el servo conectado al pin PC7 a la posición 0º	El servo conectado al pin PC7 debe posicionarse en el ángulo 0º.	El servo conectado al pin PC7 se posiciona la posición correspondiente a 0º.
SER000027	Posicionar el servo conectado al pin PC7 a la posición 45º	El servo conectado al pin PC7 debe posicionarse en el ángulo 45º.	El servo conectado al pin PC7 se posiciona la posición correspondiente a 45º.
SER000028	Posicionar el servo conectado al pin PC7 a la posición 90º	El servo conectado al pin PC7 debe posicionarse en el ángulo 90º.	El servo conectado al pin PC7 se posiciona la posición correspondiente a 90º.
SER000029	Posicionar el servo conectado al pin PC7 a la posición 135º	El servo conectado al pin PC7 debe posicionarse en el ángulo 135º.	El servo conectado al pin PC7 se posiciona la posición correspondiente a 135º.
SER000030	Posicionar el servo conectado al pin PC7 a la posición 180º	El servo conectado al pin PC7 debe posicionarse en el ángulo 180º.	El servo conectado al pin PC7 se posiciona la posición correspondiente a 180º.

Problemas encontrados

Problema 1

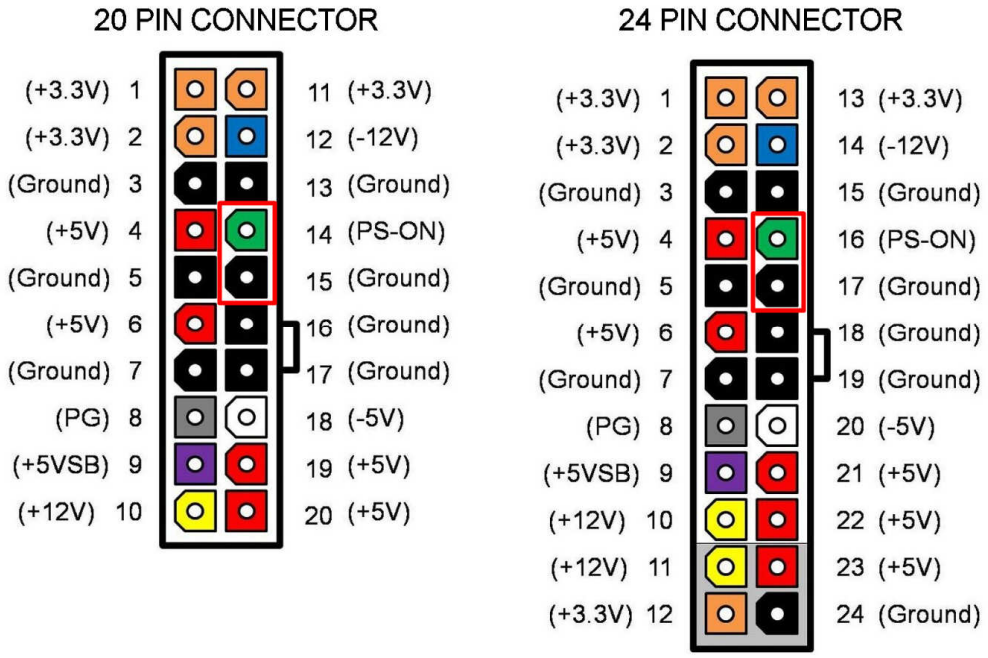
Durante las pruebas con los servos se descubrió que a medida que se añaden servos a las pruebas para su movimiento simultáneo, estos se volvían más lentos.
Tal era el efecto que cuando se llega a los 6 servos estos se movían de forma aparentemente errática. Finalmente se observa que no son movimientos erráticos sino retrasados en el tiempo respecto al momento de ejecución o incluso movimientos realizados en varios tramos en lugar de una sola vez.

Estudio Problema 1

Consultado la hoja de características de este servo se observa que tienen un consumo de corriente entre 500 y 900mA, lo que significa que cuando todos están en movimiento simultáneamente el consumo es de 900mA x 6 o lo que es lo mismo 5.4A. Esta corriente no es capaz de ser suministrada por la placa de pruebas STM32F4 Discovery.

Solución 1

Para solucionar este problema se usa una fuente de corriente externa capaz de dar 5 voltios y una corriente superior a los 7A para asegurar el correcto funcionamiento. Como opción a fuente de tensión/corriente se toma la decisión de usar una fuente de corriente de ordenador que siendo de 500W es capaz de alcanzar los 30A por las salidas de 5 voltios.
Esta fuente es posible utilizarla independientemente de un ordenador personal si se cortocircuitan los cables verde y negro.



Se conectan las líneas por donde se transmite las señales de los servos directamente a la placa de prueba y las líneas de Vcc y GND de cada servo a la fuente de alimentación de PC, líneas rojas y negras simultáneamente y se vuelve a realizar la prueba.

Problema 2

Tras realizar todas las conexiones y volver a realizar las pruebas de mover simultáneamente los 6 servos se comprueba que ninguno de estos se mueven, pero el código se está ejecutando correctamente en el microcontrolador de la placa.

Solución 2

Tras hacer varias pruebas para asegurar que las conexiones son correctas e incluso quitando todos los servos excepto uno se comprueba que tampoco funciona.
Para intentar solucionar el problema se conecta la fuente de alimentación de PC a los pines 5v y Gnd de la placa de pruebas y se observa que vuelve a funcionar el servo.
Se conectan todos los servos y siguen funcionando.

Conclusión Problema 2

Para que los periféricos conectados a la placa de pruebas funcionen correctamente, la placa de pruebas y los periféricos deben tener las mismas tensiones de referencia.

Montaje de Brazo Robot

Una vez se ha completado el estudio del uso de servos y solucionados los problemas encontrados en el trascurso de este estudio y pruebas, es el momento de montar el brazo robot.

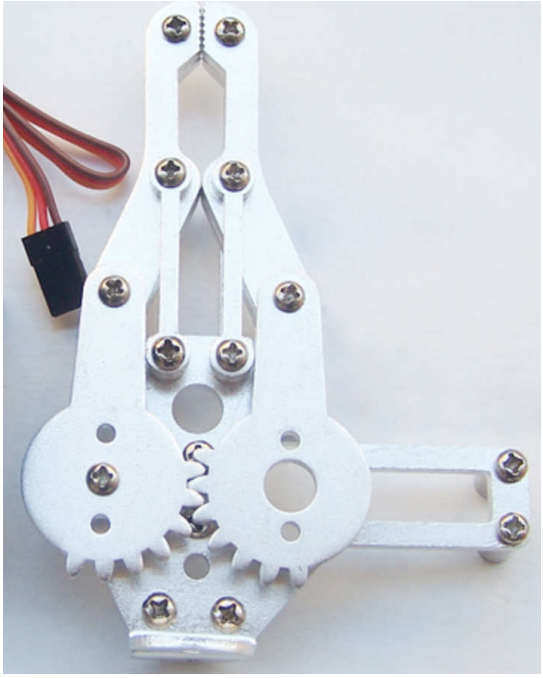
Para esta tarea no basta con atornillar las piezas y los servos a la estructura, sino que se debe atornillar en la posición correcta para con ello permitir que las articulaciones puedan tener el mayor recorrido posible de forma simétrica.

Para realizar esta unión mediante tornillos se utilizan estos fijadores de metal que disponen de 4 tornillos, dispuestos en cruz, para fijarlo a la estructura del brazo.

Pare realizar la correcta instalación de los servos en el brazo, antes de comenzar con el anclaje lo que se hace es posicionarlos en su posición central, mediante un pequeño programa a ejecutar en la placa mientras están todos los servos conectados. Esta posición central se corresponde con 90º.

Una vez hecho esto, y para conseguir que el rango de acción de los servos sea mayor y simétrico, se atornillan los servos a las articulaciones haciendo que estas formen ángulos rectos (90º) entre las dos partes que coinciden en la articulación.

Esta configuración es aplicable a todos los servos excepto el que va a ser utilizado para articular la pinza de agarre.



Este servo de ser inicializado a 0º en lugar de 90 y atornillarlo a la pinza manteniendo está cerrada. Con esto se consigue que su rango de funcionamiento esté comprendido entre 0º que equivale a la pinza cerrada y 90º que es la posición necesaria para que la pinza esté completamente abierta.

Del mismo modo, el servo encargado de realizar el movimiento horizontal del hombro se monta sobre la base atornillando el resto del brazo al servo recto respecto a la línea recta de los 90º que debe formar este servo respecto del rango de movimiento.

Una vez realizado todos los anclajes mediante los tornillos y las fijaciones entre el servo y las articulaciones el brazo queda montado y listo para comenzar las pruebas.

Pruebas del Brazo

Para comenzar las pruebas los servos de las distintas articulaciones se inicializan en 90º mediante las instrucciones.

```
#include "stm32f4_discovery.h"
TM_SERVO_t ShoulderHorizontal, ShoulderVertical, Cubit,
            WristVertical, WristRotation, Claw;
TM_SERVO_Init(&ShoulderHorizontal,        TIM5,        TM_PWM_Channel_1,
TM_PWM_PinsPack_1);     //PA0  Green -> Shoulder Horizontal
TM_SERVO_Init(&ShoulderVertical,         TIM5,        TM_PWM_Channel_2,
TM_PWM_PinsPack_1);     //PA1  Yellow -> Shoulder Vertical

TM_SERVO_Init(&Cubit, TIM4, TM_PWM_Channel_1, TM_PWM_PinsPack_2);
                  //PD12  Orange -> Cubit
TM_SERVO_Init(&WristVertical,          TIM4,          TM_PWM_Channel_2,
TM_PWM_PinsPack_2);           //PD13  Red -> Wrist Vertical


TM_SERVO_Init(&WristRotation,          TIM8,          TM_PWM_Channel_1,
TM_PWM_PinsPack_1);           //PC6 Grey -> Wrist Rotation
```

```
TM_SERVO_Init(&Claw, TIM8, TM_PWM_Channel_2, TM_PWM_PinsPack_1);
                  //PC7 Black -> Claw
TM_SERVO_SetDegrees(&ShoulderHorizontal, 90);
TM_SERVO_SetDegrees(&ShoulderVertical, 90);

TM_SERVO_SetDegrees(&Cubit, 90);

TM_SERVO_SetDegrees(&WristRotation, 90);
TM_SERVO_SetDegrees(&WristVertical, 90);
TM_SERVO_SetDegrees(&Claw, 90);
```

Como se puede apreciar en este ejemplo las principales funciones para controlar un servo son solamente dos:

- TM_SERVO_Init(Servo, Timer, Channel, PinsPack)

  Esta función es utilizada para inicializar un servo, lo cual se hace seleccionado el timer a utilizer, el canal que cada timer proporciona y el de pin a utilizer.

- TM_SERVO_SetDegrees(Servo, int)

  Esta funcion es utilizada para posicionar un servo en un ángulo comprendido entre 0 y 180 grados.

Código	Acción	Valor esperado	Valor obtenido
ARM000001	Posicionar el servo hombro horizontal a la posición 0º	El brazo gira hasta su tope hacia la derecha.	El brazo se posiciona en su posición extrema hacia la derecha de la base.
ARM000002	Posicionar el hombro horizontal a la posición 90º	El brazo gira hasta posicionarse en su posición central respecto a la base.	El brazo se posiciona en el centro de la base.
ARM000003	Posicionar el servo hombro a la posición 180º	El brazo gira hasta su tope hacia la izquierda.	El brazo se posiciona en su posición extrema hacia la izquierda de la base.
ARM000004	Posicionar el servo hombro vertical a la posición 0º	La parte del brazo comprendida entre el hombro y le codo se posiciona paralelo a la base y el codo hacia atrás.	El brazo se coloca paralelo a la base en su zona comprendida entre el hombro y el codo con el codo señalando la parte anterior de la base.
ARM000005	Posicionar el servo hombro vertical a la posición 90º	El brazo se sitúa de modo que el codo está en perpendicular al hombro.	El brazo se mueve hasta que la zona comprendida entre el hombro y el codo está perpendicular a la base.
ARM00006	Posicionar el servo hombro vertical a la posición 180º	La parte del brazo comprendida entre el hombro y le codo se posiciona paralelo a la base y el codo hacia	El brazo se coloca paralelo a la base en su zona comprendida entre el hombro y el codo con el codo señalando la parte

		adelante.	posterior de la base.
ARM000007	Posicionar el servo codo a la posición 0º	El brazo y ante brazo se alinean formando un ángulo de 180º.	El brazo se estira formando una línea recta el brazo y el antebrazo.
ARM000008	Posicionar el servo codo a la posición 90º	El brazo y ante brazo forman un ángulo de 90º.	El antebrazo gira respecto al antebrazo hasta formar un ángulo de 90º con el.
ARM000009	Posicionar el servo codo a la posición 180º	El brazo y antebrazo forman un ángulo de 0º.	El antebrazo gira respecto al brazo intentando formar un ángulo de 0º pero se lo impide la propia estructura.
ARM000010	Posicionar el servo muñeca rotación a la posición 0º	La muñeca rota hacia la derecha poniéndose vertical.	La muñeca rota 90º hacia la derecha hasta ponerse perpendicular a la base.
ARM000011	Posicionar el servo muñeca rotación a la posición 90º	La muñeca rota hasta ponerse horizontal respecto a la base.	La muñeca rota hasta la posición central estableciéndose horizontal a la base.
ARM000012	Posicionar el servo muñeca rotación a la posición 180º	La muñeca rota hacia la izquierda poniéndose vertical.	La muñeca rota 90º hacia la izquierda hasta ponerse perpendicular a la base.
ARM000013	Posicionar el servo muñeca vertical a la posición 0º	La muñeca gira verticalmente hasta colocarse en línea recta con el antebrazo.	La muñeca realiza un movimiento vertical ascendente hasta alinearse con el antebrazo.
ARM000014	Posicionar el servo muñeca vertical a la posición 90º	La muñeca gira verticalmente hasta formar un ángulo recto respecto al antebrazo.	La muñeca realiza un movimiento vertical hasta colocarse en perpendicular al antebrazo.
ARM000015	Posicionar el servo muñeca vertical a la posición 180º	La muñeca gira verticalmente hasta formar un ángulo de 0º con el antebrazo.	La muñeca gira verticalmente intentado llegar a la posición en la que formaría 0º con el antebrazo pero se lo impide la propia estructura.
ARM000016	Posicionar el servo pinza a la posición 0º	La pinza se mueve hasta cerrarse completamente.	La pinza realiza un movimiento de cerrado hasta cerrarse por completo.
ARM000017	Posicionar el servo pinza a la posición 45º	La pinza realiza un movimiento hasta	La pinza se abre o cierra hasta la

		alcanzar la la posición central de apertura.	posición central del recorrido.
ARM000018	Posicionar el servo pinza a la posición 90º	La pinza se abre completamente.	La pinza realiza un recorrido hasta abrirse completamente.

Unión Nunchuk-STM32F4-Brazo

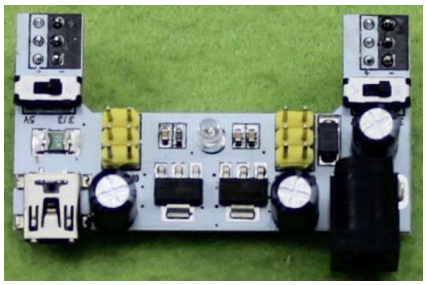
Una vez finalizado el estudio de los diferentes compontes hardware del proyecto como son el Nunchuk, LCD 16x2 (16 columnas y 2 filas), la plataforma de desarrollo STM32F4 Discovery, los servos y el brazo, así como los componentes software Atollic TrueStudio como IDE de desarrollo, librerías proporcionadas por el proveedor y librerías desarrolladas, llega el momento de unirlos para dar lugar al proyecto completo y con ello alcanzar el objetivo de este.

Conexionado

Como se ha comentado en puntos anteriores:

- El nunchuk queda conectado a 5voltios, Gnd y a los puertos PB6 y PB7.
- El LCD es conectado a los pines PD0, PD2 y PD4 para control de este así como PD1, PD3, PD5 y PD7 para transmitir datos entre la placa de desarrollo y el LCD
- Los servos ya emplazados en el brazo en la siguiente configuración:
    - Servo hombro movimiento horizontal:
      Conectado al pin PA0.
      Para ello se utiliza un cable verde.
    - Servo hombro movimiento vertical:
      Conectado al pin PA1.
      Para ello se utiliza un cable amarillo.
    - Servo codo:
      Conectado al pin PD12.
      Para ello se utiliza un cable de color naranja.
    - Servo muñeca movimiento vertical:
      Conectado al pin PD13.
      Para ello se utiliza un cable de color rojo.
    - Servo muñeca movimiento de rotación:
      Conectado al pin PC6.
      Para ello se utiliza un cable de color gris.
    - Servo de pinza:
      Conectado al pin PC7.
      Para ello se utiliza un cable de color negro.

Para conectar los servos a la placa de desarrollo al tiempo la fuente de alimentación externa se usa el siguiente componente:




Aunque el propósito de esta placa es la de conectar las líneas de alimentación a una placa de prototipos, su estructura es perfecta para proporcionar 6 conexiones a dispositivos que necesiten Vcc y Gnd quedando espacio para un tercer pin.
En la parte superior irá conectado un par de cables para llevar alimentación desde la fuente de alimentación externa hasta esta placa y otro par de cables para llevar la alimentación hasta la placa de desarrollo. Con ello se consigue que las tensiones de referencia de la fuente externa y la placa de desarrollo sea la misma.
Con esto quedan establecidas todas las conexiones necesarias.

Desarrollo Software

Una finalizadas las conexiones se pasa al desarrollo del software encargado de orquestar las distintas funciones desarrolladas en las librerías descargadas y desarrolladas.
La primera decisión que se toma en este punto es determinar que valores leídos del Nunchuk forman parte de cada uno de los movimientos de los distintos servos.
El Nunchuk proporciona seis eventos distintos e independientes:

- Movimiento lateral del Joystick. (Empujar hacia la izquierda o empujar hacia la derecha)
- Movimiento frontal del Joystick. (Empujar hacia adelante o tirar de él)
- Inclinación hacia adelante o hacia atrás del Nunchuk.
- Rotación del Nunchuk: Inclinación hacia la izquierda y hacia la derecha.
- Pulsación de botón C.
- Pulsación del botón Z.

No se tienen en cuenta el valor de los acelerómetros de forma independiente debido a que para usarlo habría que estudiar no solo como varía sino en la forma que lo hace (más rápido o más lento), lo cual los hace difícil de manejar.
Si se tiene en cuenta que la pulsación de los botones C y Z van a ser usados para cerrar y abrir la pinza respectivamente, sólo quedan cuatro eventos para actualizar los cinco servos que quedan.
En este momento surgen dos opciones para resolver este problema de tener menos eventos independientes que servos:

- La primera opción consiste en realizar funciones en las que se utilicen varios eventos simultáneamente para definir calcular la posición que un servo debe tener.
  Esta opción provoca que los servos no puedan ser movidos de forma independiente provocando que se pierdan zonas a las que se puede acceder.

- La segunda opción consiste utilizar un evento por cada servo y dos o más eventos, de modo que se genere un nuevo evento, para el último servo d. Esta opción de este modo. Esta solución permite alcanzar todas las posiciones a las que por dimensiones físicas el brazo puede alcanzar.

La opción que se toma es la de desarrollar es la segunda.
Con esta decisión tomada lo siguiente es asignar los distintos eventos individuales a los servos y después diseñar el nuevo evento para el servo que queda sin asignar. De esta forma se realiza la siguiente asociación:

- Pulsar el botón C controlará el cierre de la pinza
- Pulsar el botón Z controlará la apertura de la pinza.
- La inclinación del Nunchuk controlará el servo destinado a ser el movimiento vertical de la muñeca.
- La rotación del Nunchuk controlará el servo destinado a ser el movimiento rotatorio de la muñeca
- El movimiento frontal del joystick controlará el servo destinado a ser el codo del brazo.
- El movimiento lateral del joystick controlará el servo destinado a ser el movimiento lateral del hombro.

De esta manera todos los eventos individuales quedan asignados a un y solamente un servo y queda el servo destinado a ser el movimiento vertical del hombro el que no tiene evento asociado.
Para generar un nuevo evento se estudian las posibilidades de unión de varios eventos que no influya en los demás servos como efecto lateral.
Por ello se crea el vento de inclinación del Nunchuk + botón C + botón Z para controlar el servo asociado al movimiento vertical del hombro y con esto quedan todos los servos asociados a eventos de modo que ninguno interfiere en los demás.

El proceso es realizar un bucle donde dentro de él se sucedan las siguientes acciones:

1.- Lectura de los valores del Nunchuk.
2.- Calcular las nuevas posiciones de los distintos servos a partir de los valores leídos del Nunchuk.
3.- Actualizar la posición de los servos.
4.- Preparar nueva lectura.
5.- Realizar una pausa pequeña.
6.- Ir al paso 1

Primera Aproximación

En la primera aproximación se decide hacer un mapeo exacto del rango que tiene cada evento al rango posible del servo que controla.
Esta función map viene definida como $salida = \frac{d_{max}\text{-}d_{min}}{o_{max}\text{-}o_{min}} * entrada + o_{min}$
Donde:

- [o_{min}, o_{max}] es el rango posible del evento [-65, 65] para los valoes de Roll y Pitch y [-123, 123] para los valores del joystick.
- [d_{min}, d_{max}] es el rango de movimiento posible del servo en cuestión.

Con esta función para calcular la nueva posición de cada servo basta con hacer el cálculo:

- int posición = map(valor_evento, -65, 65, 0, 180)
  para los servos gobernados por los eventos Roll y Pitch.

- int posicion = map(valor_evento, -123, 123, 0, 180)
  para los servos gobernados por los eventos del joystick.

Tras bastantes pruebas se consigue un control bastante bueno del brazo, pero el uso es bastante incómodo por varias causas:

- Los movimientos combinados requieren de la unión de movimientos como presionar simultáneamente el joystick hacia adelante y la izquierda e inclinar el nunchuk hacia adelante y rotarlo para alcanzar una posición.
- Para acceder a una posición exacta es necesario mantener los eventos constantes, cosa que es difícil tanto para acelerómetros como el joystick.
- El movimiento se realiza de forma demasiada abrupta resultando incomodo tener exactitud.

Es por estos motivos que surge la necesidad de diseñar un mecanismo para optimizar los movimientos y exactitud.

Segunda Aproximación

Con el objetivo de conseguir mayor suavidad en el movimiento del brazo se definen las siguientes constantes:

- LOOP_DELAY: que son los milisegundos a esperar entre una ejecución del bucle y el siguiente.
- SECONDS_TO_MAX: segundos que se quiere para que un servo alcance un extremo de su movimiento desde el centro. Por ejemplo cuantos segundos se quieren que la rotación del hombro tarde en llegar desde el centro hasta la izquierda.

Combinando estas dos constantes se obtiene el número de veces que se tiene que ejecutar un bucle para completar un movimiento completo como: **NUMBER_ITERATIONS = (1000/LOOP_DELAY) * SECONDS_TO_MAX**.

El siguiente paso es definir el rango máximo en grados que un servo puede tener de forma simétrica.
Por ejemplo, el movimiento horizontal del hombro va de 0 a 180º por lo que el rango máximo es de 180 se tendría un paso máximo definido como
pasoServo = 180/(2 * NUMBER_ITERATIONS).

Este valor de paso máximo indica cuantos grados avanza un servo en cada iteración si el evento está en su máximo valor, para que el movimiento completo dure SECONDS_TO_MAX.

Con este cálculo en el que se define un paso máximo dependiendo del rango de movimiento de cada servo la función que actualiza la posición de cada servo queda de forma:

- Int grados =map(|valor_evento|, 0, 65, 0, pasoServo)
  para los servos gobernados por los eventos Roll y Pitch

- Int grados = map(|valor_evento|, 0, 123, 0, pasoServo)
  para los servos gobernados por los eventos del joystick.

Si el valor del evento es negativo se debe invertir el valor de la variable grados.
Y finalmente la nueva posición del servo es la posición actual del servo más el valor de la variable grados.
Por último, falta enviar el nuevo valor calculado al servo.
El uso de este algoritmo ofrece mejores resultados en la práctica:

- Una considerable mejora en el control de las distintas articulaciones.
- Permite volver la posición de los acelerómetros y joystick a su posición central para que el servo mantenga su posición sin volver a la posición de reposo.

Nota:
Cabe destacar que ya que tanto la muñeca en su movimiento vertical como el hombro en su movimiento igualmente vertical comparten el evento de la inclinación frontal del nunchuk.
Para hacerlos totalmente independiente se controla mediante la comprobación de que ambos botones C y Z estén pulsados, y en cambio para que el evento de inclinación actue sobre el movimiento vertical de la muñeca se comprueba que alguno de los dos botones C o Z o ambos están sin pulsar.

Problema 1

Aun así, se encuentran problemas en el control de tres servos en concreto. Los servos correspondientes a la muñeca vertical y rotatorio, y el movimiento vertical del hombro dependen de los eventos Roll y Pitch, lo que los condicionan al pulso que el usuario humano tenga.

Solución 1

Por ello para impedir vibraciones y un movimiento similar al Parkinson, que hacen imposible para la mayoría de usuario hacer movimientos precisos se modifica un poco el algoritmo que actualiza la posición de los servos.
Esta modificación consiste en establecer un margen alrededor de la posición de reposo de los eventos en el cual se establece que el valor del evento es 0 para imponer que el movimiento se detecte a partir de un mínimo, eliminado de esta forma las vibraciones.
Quedando el algoritmo para estos servos de la siguiente forma:
Grados =map(|valor_evento|, MIN, 65, 0, pasoServo);

Si el **valor del evento** es negativo se debe invertir el valor de la variable **grados**.

Y finalmente la nueva posición del servo es la posición actual del servo más el valor de la variable grados.
Donde MIN es el valor que se prefije a partir de la que la que se quiere hacer sensible los movimientos de los servos gobernados por los eventos Roll y Pitch.
Se ha definido un valor MIN distinto para cada servo.

Problema 2

Una vez finalizado el desarrollo y comienzan las prueba con el sistema completo se comprueba que la ejecución del programa en la placa de desarrollo es inmediata a la finalización de la carga del programa en el microcontrolador, provocando que durante un medianamente corto espacio de tiempo el brazo se mueve erráticamente dependiendo de la posición que este tenga sobre la mesa o la mano del usuario.

Solución 2

Se inicializa una variable que indica que el sistema está apagado, y si en cada iteración la variable indica que el sistema está apagado se comprueba si los botones C y Z son pulsados durante dos segundos.
En caso de que la comprobación indique que los botones C y Z no han sido pulsados durante 2 segundos volverá al bucle principal volviendo a comprobar si el sistema esta encendido o no.
En caso de que la comprobación indique que los botones C y Z si han sido pulsados durante 2 segundos consecutivos, el LCD parpadeará durante 3 segundos indicando que el sistema está siendo inicializado. Esta acción da tiempo al usuario a que se prepare.
Tras estos 3 segundos el brazo se posicionará en su posición inicial y comenzará a actualizar la posición del brazo en función de los eventos del Nunchuk.

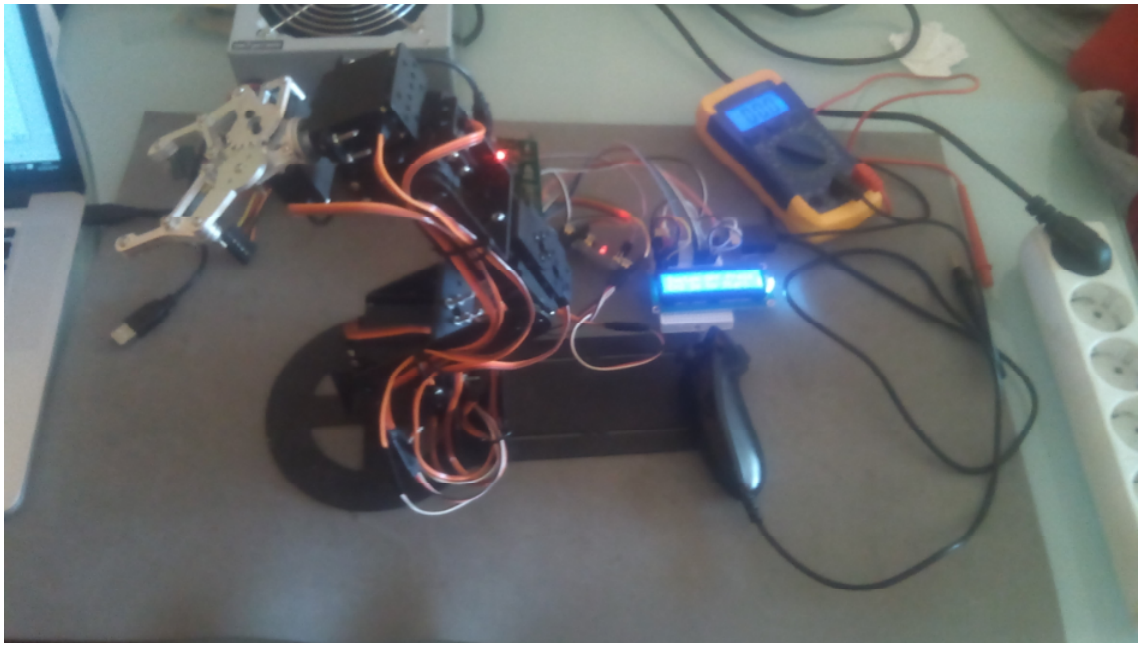
Pruebas Nunchuk-STM32F4-Brazo
Las pruebas siempre cumplen como premisa que los movimientos parten de la posición inicial del brazo tras inicializar el sistema.

Código	Acción	Valor esperado	Valor obtenido
NUNARM001	Dejar el Nunchuk en su posición de reposo.	El brazo no debe realizar ningún movimiento.	El brazo no se mueve
NUNARM002	Llevar el Joystick X hasta la posición más extrema hacia la izquierda	El brazo debe girar a su velocidad máxima hasta alcanzar la posición más izquierda de la base.	El brazo completo gira hacia la izquierda tardando 2 segundos en alcanzar el extremo.
NUNARM003	Llevar el Joystick X hasta la posición más extrema hacia la derecha	El brazo debe girar a su velocidad máxima hasta alcanzar la posición más derecha de la base.	El brazo completo gira hacia la derecha tardando 2 segundos en alcanzar el extremo.

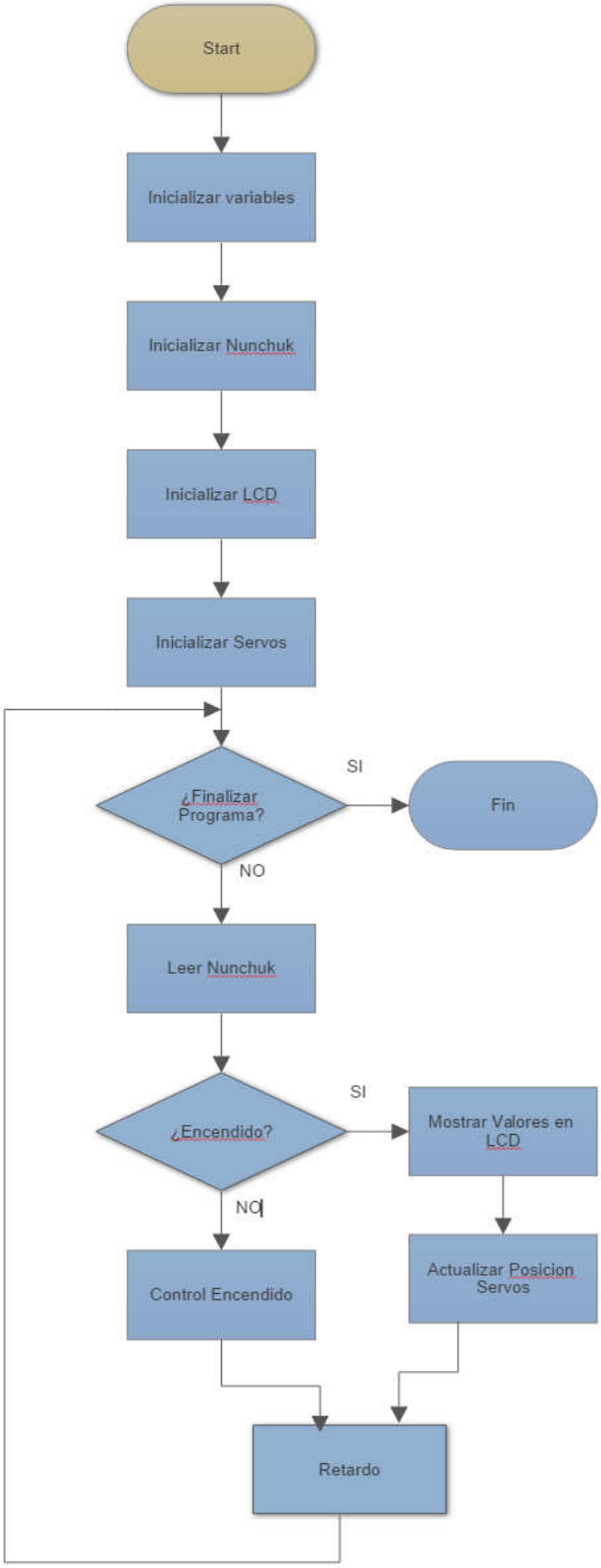
NUNARM004	Llevar el Joystick X hasta la posición intermedia entre el centro y el extremo izquierdo.	El brazo debe girar hacia la izquierda a su velocidad media y alcanzar el extremo en el doble de tiempo.	El brazo completo gira hacia la izquierda tardando 4 segundos en alcanzar el extremo.
NUNARM005	Llevar el Joystick X hasta la posición intermedia entre el centro y el extremo derecho.	El brazo debe girar hacia la derecha a su velocidad media y alcanzar el extremo en el doble de tiempo.	El brazo completo gira hacia la derecha tardando 4 segundos en alcanzar el extremo.
NUNARM006	Dejar el Joystick Y en su posición de reposo.	El brazo debe de dejar de moverse tanto horizontal como en altura	El brazo detiene su movimiento horizontal y en altura.
NUNARM007	Llevar el Joystick Y hasta la posición más extrema hacia la adelante	El antebrazo debe girar respecto del brazo y ponerse recto respecto a él.	El antebrazo gira respecto del brazo y ponerse recto respecto a él.
NUNARM008	Llevar el Joystick Y hasta la posición más extrema hacia la atrás.	El antebrazo debe girar respecto del antebrazo e intentar crear un ángulo de 0 grados con él.	El antebrazo gira respecto del antebrazo intentando crear un ángulo de 0 grados con él pero se lo impide la estructura.
NUNARM009	Llevar el Joystick Y hasta la posición intermedia entre el centro y el extremo izquierdo.	El brazo debe girar horizontalmente hacia la izquierda a una velocidad inferior a la máxima.	El brazo girar horizontalmente hacia la izquierda a una velocidad inferior a la máxima.
NUNARM010	Llevar el Joystick Y hasta la posición intermedia entre el centro y el extremo derecho.	El brazo debe girar horizontalmente hacia la derecha a una velocidad inferior a la máxima.	El brazo girar horizontalmente hacia la derecha a una velocidad inferior a la máxima.
NUNARM011	Inclinar el Nunchuk hasta ponerlo perpendicular al suelo con los botones hacia arriba. Con los botones C y Z pulsados.	El brazo debe inclinarse hacia adelante desde el hombro	El brazo se inclina hacia adelante desde un movimiento que surge del hombro.
NUNARM012	Inclinar el Nunchuk hasta ponerlo perpendicular al suelo con los botones hacia arriba. Con los botones C y Z pulsados.	El brazo debe inclinarse hacia adelante desde el hombro	La muñeca se inclina hacia abajo.
NUNARM013	Inclinar el Nunchuk hasta ponerlo perpendicular al suelo con los botones hacia arriba.	La muñeca debe inclinarse verticalmente hacia abajo.	El brazo se inclina hacia adelante desde un movimiento que surge del hombro.

	Co los botones C y Z sin pulsar.		
NUNARM014	Inclinar el Nunchuk hasta ponerlo perpendicular al suelo con los botones hacia arriba. Con los botones C y Z sin pulsar.	La muñeca debe ascender verticalmente hasta ponerse en línea con el antebrazo.	La muñeca asciende verticalmente hasta ponerse en línea con el antebrazo.
NUNARM015	Girar el Nunchuk hacia la izquierda rotando la muñeca 90 grados.	La muñeca del brazo se debe inclinar hacia la izquierda a la velocidad máxima.	La muñeca del brazo inclina hacia la izquierda a la velocidad máxima.
NUNARM016	Girar el Nunchuk hacia la derecha rotando la muñeca 90 grados.	La muñeca del brazo se debe inclinar hacia la derecha a la velocidad máxima.	La muñeca del brazo inclina hacia la derecha a la velocidad máxima.
NUNARM017	Girar el Nunchuk hacia la izquierda, rotando la muñeca, sin llegar a los 90 grados.	La muñeca del brazo se debe inclinar hacia la izquierda a inferior velocidad de la máxima.	La muñeca del brazo inclina hacia la izquierda a la velocidad inferior a la máxima.
NUNARM018	Girar el Nunchuk hacia la derecha, rotando la muñeca, sin llegar a los 90 grados.	La muñeca del brazo se debe inclinar hacia la derecha a inferior velocidad de la máxima.	La muñeca del brazo inclina hacia la derecha a la velocidad inferior a la máxima.
NUNARM019	Pulsar el botón C.	La pinza debe comenzar a cerrarse.	La pinza se cierra mientras esté pulsado el botón C.
NUNARM020	Pulsar el botón Z.	La pinza debe abrirse.	La pinza se abre mientras esté pulsado el botón Z.

Resultado final



Flujograma final del proyecto



En el flujograma final del proyecto se aprecia la inclusión del control de encendido, así como el de la visualización de los valores del Nunchuk en el LCD respecto al flujograma del Diseño.

Conclusiones

Anexos

En este apartado se proporcionan las librerías que han sido desarrolladas por el autor.

Código Desarrollado

nunchuk.h

```
/**
 * @defgroup TM_I2C
 * @brief    I2C library for STM32F4xx - http://stm32f4-discovery.com/2014/05/library-09-i2c-for-stm32f4xx/
 * @{
 *
 *            \par Pinout
 *
@verbatim
          |PINSPACK 1   |PINSPACK 2   |PINSPACK 3
I2CX   |SCL   SDA       |SCL   SDA       |SCL   SDA
-----------------------------------------------------------------------------
I2C1   |PB6   PB7       |PB8   PB9       |PB6   PB9
I2C2   |PB10  PB11      |PF1   PF0       |PH4   PH5
I2C3   |PA8   PC9       |PH7   PH8       |-     -
@endverbatim
 *
 * \par Custom pinout
*/
#include "stm32f4xx.h"
#include "tm_stm32f4_i2c.h"
#include "attributes.h"
#include "defines.h"
#include "tm_stm32f4_gpio.h"

/**
 * @defgroup TM_I2C_Macros
 * @brief    Library defines
 * @{
 */

/**
 * @brief  Timeout for I2C
 */
#define ZEROX 510
#define ZEROY 490
#define ZEROZ 460
#define RADIUS 210  // probably pretty universal

#define DEFAULT_ZERO_JOY_X 124
#define DEFAULT_ZERO_JOY_Y 132

void nunchuk_Init(I2C_TypeDef* I2Cx, uint8_t address, TM_I2C_PinsPack_t pinsPack, uint32_t speed);

void nunchuk_SendZero();

int nunchuk_Update();

int nunchuk_GetC();

int nunchuk_GetZ();

int nunchuk_GetJoystickX();

int nunchuk_GetJoystickY();

int nunchuk_GetPitch();

int nunchuk_GetRoll();

int nunchuk_GetAcelerometerX();
```

```
int nunchuk_GetAcelerometerY();

int getAcelerometerZ();
```

nunchuk.c

```
#include "nunchuk.h"
#include "tm_stm32f4_i2c.h"
#include "Math.h"

#define MAX_JOY 123
#define MAX_ACC 65

/* private variables */
float AX = 0;
float AY = 0;
float AZ = 0;
int JX = 0;
int JY = 0;
int BC = 0;
int BZ = 0;

uint8_t data[] = {0, 1, 2, 3, 4, 5};

I2C_TypeDef* I2C;
uint8_t nunchuk_address;
/* fin private variables */


void nunchuk_Init(I2C_TypeDef* I2Cx, uint8_t address, TM_I2C_PinsPack_t pinsPack, uint32_t speed) {
	I2C = I2Cx;
	nunchuk_address = address;

	//Initialize I2C, SCL: PB6 and SDA: PB7 with 100kHt serial clock
	TM_I2C_Init(I2C, pinsPack, speed);

	TM_I2C_Start(I2C, nunchuk_address, I2C_TRANSMITTER_MODE, I2C_ACK_DISABLE);
	TM_I2C_WriteData(I2C, 0xF0);
	TM_I2C_WriteData(I2C, 0x55);
	TM_I2C_Stop(I2C);
	int i = 1000;
	while(i > 0) {
		i--;
	}
	TM_I2C_Start(I2C, nunchuk_address, I2C_TRANSMITTER_MODE, I2C_ACK_DISABLE);
	TM_I2C_WriteData(I2C, 0xFB);
	TM_I2C_WriteData(I2C, 0x01);
	//TM_I2C_WriteData(I2C, (uint8_t)0x00);
	TM_I2C_Stop(I2C);
}

void nunchuk_SendZero() {
	TM_I2C_Start(I2C, nunchuk_address, I2C_TRANSMITTER_MODE, I2C_ACK_DISABLE);
	TM_I2C_WriteData(I2C, (uint8_t)0x00);
	TM_I2C_Stop(I2C);
}

int nunchuk_Update() {

	TM_I2C_ReadMultiNoRegister(I2C, nunchuk_address, data, 6);

	//const uint8_t buf[] = {0};
	//I2C_Write(I2C1 , buf , 1, NUNCHUK_ADDRESS);
	//I2C_Read(I2C1 , data , 6, NUNCHUK_ADDRESS);

	int resultado = (//data[0] != 0x1 &&
						data[1] != 0x1 && data[2] != 0x1 &&
						data[3] != 0x1 && data[4] != 0x1
						&& data[5] != 0x1)? 1 : 0;

	//data = (data ^ 0x17) + 0x17;
	//uint8_t resto = (data[5] >> 2) & 0x03;
```

```
		//AX = ((float)(((uint16_t)data[2] << 2) | resto)) - ZEROX;
		AX = (float) ((data[2] << 2) + ((data[5] & (00000011 << 2)) >> 2)) - ZEROX;

		//resto = (data[5] >> 4)& 0x03;
		//AY = ((float)(((uint16_t)data[3] << 2) | resto)) - ZEROY;
		AY = (float) ((data[3] << 2) + ((data[5] & (00000011 << 4)) >> 4)) - ZEROY;

		//resto = (data[5] >> 6) & 0x03;;
		//AZ = ((float)(((uint16_t)data[4] << 2) | resto))  - ZEROZ;
		AZ = (float) ((data[4] << 2) + ((data[5] & (00000011 << 6)) >> 6)) - ZEROZ;

		JX = (int) data[0] - DEFAULT_ZERO_JOY_X;
		JY = (int) data[1] - DEFAULT_ZERO_JOY_Y;

		BC = abs((int)((data[5] >> 1) & 0x01) - 1);
		BZ = abs((int) (data[5] & 0x01) - 1);
		return resultado;
}

int normalize(int value, int max) {
		if (value > max) {
			return max;
		} else if (value < -max) {
			return -max;
		}
		return value;
}

int nunchuk_GetC() {
		return BC;
}

int nunchuk_GetZ() {
		return BZ;
}

int nunchuk_GetJoystickX() {
		return normalize(JX, MAX_JOY);
}

int nunchuk_GetJoystickY() {
		return normalize(JY, MAX_JOY);
}

int nunchuk_GetPitch() {			// -65 <-> 65
		//return (float) atan(AX/sqrt(pow(AY,2) + pow(AZ,2)));
		int pitch = (int) (acos(AY/RADIUS)/ M_PI * 180.0) - 65;
		return normalize(pitch, MAX_ACC);
}

int nunchuk_GetRoll() {				// -65 <-> 65
		//return (float) atan(AY/sqrt(pow(AX,2) + pow(AZ,2)));
		int roll = (int)(atan2(AX,AZ)/ M_PI * 180.0);
		return normalize(roll, MAX_ACC);
}

int nunchuk_GetAcelerometerX() {
		return AX;
}

int nunchuk_GetAcelerometerY() {
		return AY;
}

int nunchuk_GetAcelerometerZ() {
		return AZ;
}
```

lcd_hd44780.h

```
//*****************************************************************************
//    THE SOFTWARE INCLUDED IN THIS FILE IS FOR GUIDANCE ONLY.
//    AUTHOR SHALL NOT BE HELD LIABLE FOR ANY DIRECT, INDIRECT
//    OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING
//    FROM USE OF THIS SOFTWARE.
//*****************************************************************************

//###############################################################################
// lcd_hd44780.h
//###############################################################################

//*****************************************************************************//
//#include "main.h"

#define LCD_GPIO GPIOD
#define LCD_CLK_LINE RCC_AHB1Periph_GPIOD

#define LCD_D4 GPIO_Pin_1
#define LCD_D5 GPIO_Pin_3
#define LCD_D6 GPIO_Pin_5
#define LCD_D7 GPIO_Pin_7

#define LCD_RS GPIO_Pin_0
#define LCD_RW GPIO_Pin_2
#define LCD_EN GPIO_Pin_4
//*****************************************************************************//


#define HD44780_CLEAR                                   0x01

#define HD44780_HOME                                    0x02

#define HD44780_ENTRY_MODE                              0x04
#define HD44780_EM_SHIFT_CURSOR                         0
#define HD44780_EM_SHIFT_DISPLAY                        1
#define HD44780_EM_DECREMENT                            0
#define HD44780_EM_INCREMENT                            2

#define HD44780_DISPLAY_ONOFF                           0x08
#define HD44780_DISPLAY_OFF                             0
#define HD44780_DISPLAY_ON                              4
#define HD44780_CURSOR_OFF                              0
#define HD44780_CURSOR_ON                               2
#define HD44780_CURSOR_NOBLINK                          0
#define HD44780_CURSOR_BLINK                            1

#define HD44780_DISPLAY_CURSOR_SHIFT            0x10
#define HD44780_SHIFT_CURSOR                            0
#define HD44780_SHIFT_DISPLAY                           8
#define HD44780_SHIFT_LEFT                              0
#define HD44780_SHIFT_RIGHT                             4

#define HD44780_FUNCTION_SET                            0x20
#define HD44780_FONT5x7                                 0
#define HD44780_FONT5x10                                4
#define HD44780_ONE_LINE                                0
#define HD44780_TWO_LINE                                8
#define HD44780_4_BIT                                   0
#define HD44780_8_BIT                                   16

#define HD44780_CGRAM_SET                               0x40

#define HD44780_DDRAM_SET                               0x80

//#################################################################
void lcd_init(void);
void lcd_cls(void);
void lcd_str(unsigned char * text);
void lcd_strxy(unsigned char * text, unsigned char x, unsigned char y);
void lcd_locate(unsigned char x, unsigned char y);
void lcd_int(int n);
```

```
void lcd_intxy(int n, unsigned char x, unsigned char y);

//#####################################################################

void lcd_writedata(unsigned char dataToWrite);
void lcd_writecommand(unsigned char commandToWrite);
void lcd_writebinary(unsigned int var, unsigned char bitCount);
void lcd_addchar (unsigned char chrNum, unsigned char n, const unsigned char *p);
```

lcd_hd44780.c

```
#include "lcd_hd44780.h"
#include "stm32f4xx_gpio.h"
#include "string.h"
#include "stdlib.h"
#include "stm32f4_discovery.h"
// C program for implementation of ftoa()
#include <stdio.h>
#include<math.h>

GPIO_InitTypeDef GPIO_InitStructure;

//-----------------------------------------------------------------------------
void lcd_writenibble(unsigned char nibbleToWrite) {
	GPIO_WriteBit(LCD_GPIO, LCD_EN, Bit_SET);
	GPIO_WriteBit(LCD_GPIO, LCD_D4, (BitAction) (nibbleToWrite & 0x01));
	GPIO_WriteBit(LCD_GPIO, LCD_D5, (BitAction) (nibbleToWrite & 0x02));
	GPIO_WriteBit(LCD_GPIO, LCD_D6, (BitAction) (nibbleToWrite & 0x04));
	GPIO_WriteBit(LCD_GPIO, LCD_D7, (BitAction) (nibbleToWrite & 0x08));
	GPIO_WriteBit(LCD_GPIO, LCD_EN, Bit_RESET);
}

//-----------------------------------------------------------------------------
unsigned char LCD_ReadNibble(void) {
	unsigned char tmp = 0;
	GPIO_WriteBit(LCD_GPIO, LCD_EN, Bit_SET);
	tmp |= (GPIO_ReadInputDataBit(LCD_GPIO, LCD_D4) << 0);
	tmp |= (GPIO_ReadInputDataBit(LCD_GPIO, LCD_D5) << 1);
	tmp |= (GPIO_ReadInputDataBit(LCD_GPIO, LCD_D6) << 2);
	tmp |= (GPIO_ReadInputDataBit(LCD_GPIO, LCD_D7) << 3);
	GPIO_WriteBit(LCD_GPIO, LCD_EN, Bit_RESET);
	return tmp;
}

//-----------------------------------------------------------------------------
unsigned char LCD_ReadStatus(void) {
	unsigned char status = 0;

	GPIO_InitStructure.GPIO_Pin = LCD_D4 | LCD_D5 | LCD_D6 | LCD_D7;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
	GPIO_Init(LCD_GPIO, &GPIO_InitStructure);

	GPIO_WriteBit(LCD_GPIO, LCD_RW, Bit_SET);
	GPIO_WriteBit(LCD_GPIO, LCD_RS, Bit_RESET);

	status |= (LCD_ReadNibble() << 4);
	status |= LCD_ReadNibble();

	GPIO_InitStructure.GPIO_Pin = LCD_D4 | LCD_D5 | LCD_D6 | LCD_D7;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
	GPIO_Init(LCD_GPIO, &GPIO_InitStructure);

	return status;
}

//-----------------------------------------------------------------------------
void lcd_writedata(unsigned char dataToWrite) {
	GPIO_WriteBit(LCD_GPIO, LCD_RW, Bit_RESET);
	GPIO_WriteBit(LCD_GPIO, LCD_RS, Bit_SET);

	lcd_writenibble(dataToWrite >> 4);
```

```
		lcd_writenibble(dataToWrite & 0x0F);

		while (LCD_ReadStatus() & 0x80)
			;
}

//-----------------------------------------------------------------------------
void lcd_writecommand(unsigned char commandToWrite) {
		GPIO_WriteBit(LCD_GPIO, LCD_RW | LCD_RS, Bit_RESET);
		lcd_writenibble(commandToWrite >> 4);
		lcd_writenibble(commandToWrite & 0x0F);

		while (LCD_ReadStatus() & 0x80)
			;
}

//-----------------------------------------------------------------------------
void lcd_str(unsigned char * text) {
		while (*text)
			lcd_writedata(*text++);
}

//-----------------------------------------------------------------------------
void lcd_locate(unsigned char x, unsigned char y) {
		lcd_writecommand(HD44780_DDRAM_SET | (x + (0x40 * y)));
}

//-----------------------------------------------------------------------------
void lcd_strxy(unsigned char * text, unsigned char x, unsigned char y) {
		lcd_locate(x, y);
		while (*text)
			lcd_writedata(*text++);
}

//-----------------------------------------------------------------------------
void lcd_writebinary(unsigned int var, unsigned char bitCount) {
		signed char i;

		for (i = (bitCount - 1); i >= 0; i--) {
			lcd_writedata((var & (1 << i)) ? '1' : '0');
		}
}

//-----------------------------------------------------------------------------
void LCD_ShiftLeft(void) {
		lcd_writecommand(
				HD44780_DISPLAY_CURSOR_SHIFT | HD44780_SHIFT_LEFT
						| HD44780_SHIFT_DISPLAY);
}

//-----------------------------------------------------------------------------
void LCD_ShiftRight(void) {
		lcd_writecommand(
				HD44780_DISPLAY_CURSOR_SHIFT | HD44780_SHIFT_RIGHT
						| HD44780_SHIFT_DISPLAY);
}

//-----------------------------------------------------------------------------
void lcd_init(void) {
		volatile unsigned char i = 0;
		volatile unsigned int delayCnt = 0;
		RCC_AHB1PeriphClockCmd(LCD_CLK_LINE, ENABLE);
		GPIO_InitStructure.GPIO_Pin = LCD_D4 | LCD_D5 | LCD_D6 | LCD_D7 | LCD_RS
				| LCD_RW | LCD_EN;
		GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
		GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;

		GPIO_Init(LCD_GPIO, &GPIO_InitStructure);

		GPIO_ResetBits(LCD_GPIO, LCD_RS | LCD_EN | LCD_RW);

		for (delayCnt = 0; delayCnt < 300000; delayCnt++)
			;

		for (i = 0; i < 3; i++) {
			lcd_writenibble(0x03);
```

```
		for (delayCnt = 0; delayCnt < 30000; delayCnt++)
			;
	}

	lcd_writenibble(0x02);

	for (delayCnt = 0; delayCnt < 6000; delayCnt++)
		;

	lcd_writecommand(HD44780_FUNCTION_SET |
	HD44780_FONT5x7 |
	HD44780_TWO_LINE |
	HD44780_4_BIT);

	lcd_writecommand(HD44780_DISPLAY_ONOFF |
	HD44780_DISPLAY_OFF);

	lcd_writecommand(HD44780_CLEAR);

	lcd_writecommand(HD44780_ENTRY_MODE |
	HD44780_EM_SHIFT_CURSOR |
	HD44780_EM_INCREMENT);

	lcd_writecommand(HD44780_DISPLAY_ONOFF |
	HD44780_DISPLAY_ON |
	HD44780_CURSOR_OFF |
	HD44780_CURSOR_NOBLINK);

}

//-----------------------------------------------------------------------------
void lcd_addchar(unsigned char chrNum, unsigned char n, const unsigned char *p) {
	lcd_writecommand(HD44780_CGRAM_SET | chrNum * 8);
	n *= 8;
	do
		lcd_writedata(*p++);
	while (--n);
}
//-----------------------------------------------------------------------------
void lcd_cls(void) {
	lcd_writecommand(HD44780_CLEAR);
}

void lcd_int(int a) {

	char* str = malloc(4);
	snprintf(str, 10, "%d", a);
	lcd_str(str);
}

void lcd_intxy(int n, unsigned char x, unsigned char y) {
	lcd_locate(x, y);
	lcd_int(n);
}
```

utilidades.h

```
void delay(uint32_t delay);
int map(int value, int oMin, int oMax, int dMin, int dMax);
int normalizeValue(int value, int min, int max);
char* intToStr(int value);
```

utilidades.c

```
#include "stm32f4xx.h"
#include <stdio.h>

void delay(uint16_t delay) {
	while (delay > 0) {
		delay--;
	}
}

int map(int value, int oMin, int oMax, int dMin, int dMax) {
	int resultado;
	int numerador = dMax - dMin;
	int denominador = oMax - oMin;
```

```
	float division = ((float)numerador / (float)denominador);
	int diferencia = value - oMin;
	resultado = (division * diferencia) + dMin;
	//resultado = (float)(((int)(dMax - dMin)/(oMax - oMin))*(value - oMin)) + dMin;
	return (int) resultado;
}

int normalizeValue(int value, int min, int max) {
	if (value < min) {
		value = min;
	} else if (value > max) {
		value = max;
	}
	return value;
}

char* intToStr(int value) {
	char* str = malloc(4);
	snprintf(str, 10, "%d", value);
	return str;
}
```

main.c

```
/**
 ******************************************************************************
 **
 **  File        : main.c
 **
 **  Abstract    : main function.
 **
 **  Functions   : main
 **
 **  Environment : Atollic TrueSTUDIO(R)
 **                STMicroelectronics STM32F4xx Standard Peripherals Library
 **
 **  Distribution: The file is distributed "as is", without any warranty
 **                of any kind.
 **
 **  (c)Copyright Atollic AB.
 **  You may use this file as-is or modify it according to the needs of your
 **  project. This file may only be built (assembled or compiled and linked)
 **  using the Atollic TrueSTUDIO(R) product. The use of this file together
 **  with other tools than Atollic TrueSTUDIO(R) is not permitted.
 **
 ******************************************************************************
 */

/* Includes */
#include "stdio.h"
#include "defines.h"
#include "tm_stm32f4_i2c.h"
#include "nunchuk.h"
#include "utilidades.h"

#include "stm32f4xx.h"
#include "stm32f4_discovery.h"
#include "tm_stm32f4_delay.h"
#include "tm_stm32f4_servo.h"

#include "lcd_hd44780.h"

//Slave address
#define NUNCHUK_ADDRESS		0xA4		// 0xA4 Nunchuk Address
#define CLOCK_SPEED		100000		// Speed for I2C
#define LOOP_DELAY		50		// Delay
to wait between iterations
#define SECONDS_TO_MAX		2
#define CYCLES		2
#define MAX_JOY		123		// Max value for
Joysticks
```

```
#define MAX_ACC                         65                                      // Max value for Roll
and Pitch

#define INIT_MESSAGE_BLINK 3
#define WAIT_SERVO 0

#define SHOULDER_HMIN 0                                 // 0º is right position
#define SHOULDER_HMAX 180                               // 180º is left position

#define SHOULDER_VMIN 5
#define SHOULDER_VMAX 120

#define CUBIT_MIN 0
#define CUBIT_MAX 180

#define WRIST_VMIN 65
#define WRIST_VMAX 175

#define WRIST_RMIN 0
#define WRIST_RMAX 180

#define CLAW_MIN 0
#define CLAW_MAX 90

#define MIN_VWRIST 10
#define MIN_RWRIST 10

/* Private macro */
/* Private variables */
TM_SERVO_t ShoulderHorizontal, ShoulderVertical, Cubit,
                        WristVertical, WristRotation, Claw;

int SHOULDER_HSTEP = 0;
int SHOULDER_VSTEP = 0;
int CUBIT_STEP  = 10;
int WRIST_VSTEP = 5;
int WRIST_RSTEP = 5;
int CLAW_STEP = 1;                                              // Step for opening and closing the claw

int shoulderHPosition = 90;
int shoulderVPosition = 45;
int clawAperture = 45;
int cubitPosition = 90;
int wristRPosition = 90;
int wristVPosition = 180;
int encendido = 0;
/* Private function prototypes */

/* Private functions */

/**
 * Method to control if C + Z buttoms have been pressed to Start System.
 */
int powerOnControl(int cPressed, int zPressed) {
            int resultado = 0;
            if (encendido == 0 && cPressed == 1 && zPressed == 1) {
                        nunchuk_SendZero();
                        Delay(2000);
                        nunchuk_Update();
                        int cPressed2 = nunchuk_GetC();
                        int zPressed2 = nunchuk_GetZ();
                        if (cPressed2 == 1 && zPressed2 == 1) {
                                    resultado = 1;
                                    encendido = 1;
                                    int i = 0;
                                    for (i = 0; i < INIT_MESSAGE_BLINK; i++) {
                                                lcd_cls();
                                                Delayms(1000);
                                                lcd_strxy("Initializing ",0,0);

                                                lcd_strxy("System...",0,1);
                                                Delayms(1000);
                                    }
                                    lcd_cls();
                                    lcd_PrintLabels();
                                    /* Set initial position */
```

```
                                TM_SERVO_SetDegrees(&ShoulderHorizontal, shoulderHPosition);   // 90º centers arm
in the center.
                                TM_SERVO_SetDegrees(&ShoulderVertical, shoulderVPosition);                    //        45º
centers arm in the center of elevation.

                                TM_SERVO_SetDegrees(&Cubit, cubitPosition);
            // 90º puts arm in the middle of lenght

                                TM_SERVO_SetDegrees(&WristRotation, wristRPosition);                          //        90º
centers arm in the center of rotation
                                TM_SERVO_SetDegrees(&WristVertical, wristVPosition);                          //      180º
puts the claw straight

                                TM_SERVO_SetDegrees(&Claw, clawAperture);
            // 45º puts claw aperture in the middle of total aperture
                        }
            }
            return resultado;
}

/**
 * Method to declarate and initialize arm servos.
 */
void initServos() {
            TM_SERVO_Init(&ShoulderHorizontal, TIM5, TM_PWM_Channel_1, TM_PWM_PinsPack_1);        //PA0
Green -> Shoulder Horizontal
            TM_SERVO_Init(&ShoulderVertical, TIM5, TM_PWM_Channel_2, TM_PWM_PinsPack_1);          //PA1
Yellow -> Shoulder Vertical

            TM_SERVO_Init(&Cubit, TIM4, TM_PWM_Channel_1, TM_PWM_PinsPack_2);
            //PD12  Orange -> Cubit
            TM_SERVO_Init(&WristVertical, TIM4, TM_PWM_Channel_2, TM_PWM_PinsPack_2);                //PD13
Red -> Wrist Vertical

            //TM_SERVO_Init(&WristRotation, TIM4, TM_PWM_Channel_3, TM_PWM_PinsPack_2);
            //PD14  Grey -> Wrist Rotation
            //TM_SERVO_Init(&Claw, TIM4, TM_PWM_Channel_4, TM_PWM_PinsPack_2);
            //PD15  Black -> Claw

            TM_SERVO_Init(&WristRotation, TIM8, TM_PWM_Channel_1, TM_PWM_PinsPack_1);                //PC6
            Grey -> Wrist Rotation
            TM_SERVO_Init(&Claw, TIM8, TM_PWM_Channel_2, TM_PWM_PinsPack_1);
            //PC7       Black -> Claw
}

/**
 * Method to update horizontal position of shoulder servo.
 */
void updateShoulderHorizontal(int value) {
//          int degrees = map(value, -MAX_JOY, MAX_JOY, 180, 0);
//          TM_SERVO_SetDegrees(&ShoulderHorizontal, degrees);
            int ant = shoulderHPosition;
            int degrees = map(value, -MAX_JOY, MAX_JOY, SHOULDER_HSTEP, -SHOULDER_HSTEP);
            shoulderHPosition += degrees;
            shoulderHPosition = normalizeValue(shoulderHPosition, SHOULDER_HMIN, SHOULDER_HMAX);
            if (shoulderHPosition != ant) {
                        TM_SERVO_SetDegrees(&ShoulderHorizontal, shoulderHPosition);
                        //Delayms(WAIT_SERVO);
            }
}

/**
 * Method to update vertical position of shoulder servo.
 */
void updateShoulderVertical(int value, int c, int z) {
            if (c == 1 && z == 1) {
//                      int degrees = map(value, -MAX_ACC, MAX_ACC, 180, 0);
//                      TM_SERVO_SetDegrees(&ShoulderVertical, degrees);
                        int ant = shoulderVPosition;

                        int degrees = map(value, -MAX_ACC, MAX_ACC, SHOULDER_VSTEP, -SHOULDER_VSTEP);
                        shoulderVPosition += degrees;
                        shoulderVPosition = normalizeValue(shoulderVPosition, SHOULDER_VMIN, SHOULDER_VMAX);
                        if (shoulderVPosition != ant) {
                                TM_SERVO_SetDegrees(&ShoulderVertical, shoulderVPosition);
                                //Delayms(WAIT_SERVO);
```

```
		}
	}
}

/**
 * Method to update position of cubit servo.
 */
void updateCubit(int value) {
//	int degrees = map(value, -MAX_JOY, MAX_JOY, 180, 0);
//	TM_SERVO_SetDegrees(&Cubit, degrees);
	int ant = cubitPosition;
	int degrees = map(value, -MAX_JOY, MAX_JOY, -CUBIT_STEP, CUBIT_STEP);
	cubitPosition += degrees;
	cubitPosition = normalizeValue(cubitPosition, CUBIT_MIN, CUBIT_MAX);
	if (cubitPosition != ant) {
		TM_SERVO_SetDegrees(&Cubit, cubitPosition);
		//Delayms(WAIT_SERVO);
	}
}

/**
 * Method to update vertical position of wrist servo
 */
void updateWristVertical(int value, int c, int z) {

	if (c == 0 || z == 0) {
		int ant = wristVPosition;
//		int degrees = map(value, -MAX_ACC, MAX_ACC, 180, 0);
//		TM_SERVO_SetDegrees(&WristVertical, degrees);
		int degrees = 0;
		//map(value, -MAX_ACC, MAX_ACC, WRIST_VSTEP, -WRIST_VSTEP);
		if (abs(value) < MIN_VWRIST) {
			degrees = 0;
		} else {
			degrees = map(abs(value), MIN_VWRIST, MAX_ACC, 0, WRIST_VSTEP);
		}
		if (value < 0) {
			degrees = -degrees;
		}

		wristVPosition = wristVPosition + degrees;
		wristVPosition = normalizeValue(wristVPosition, WRIST_VMIN, WRIST_VMAX);
		if (wristVPosition != ant) {
			TM_SERVO_SetDegrees(&WristVertical, wristVPosition);
			//Delayms(WAIT_SERVO);
		}
	}
}

/**
 * Method to update rotation position of wrist servo
 */
void updateWristRotation(int value) {

	//int degrees = map(value, -MAX_ACC, MAX_ACC, 0, 180);
	//TM_SERVO_SetDegrees(&WristRotation, degrees);
	int ant = wristRPosition;
	int degrees = 0;
	//map(value, -MAX_ACC, MAX_ACC, -WRIST_RSTEP, WRIST_RSTEP);

	if (abs(value) < MIN_RWRIST) {
		degrees = 0;
	} else {
		degrees = map(abs(value), MIN_RWRIST, MAX_ACC, 0, WRIST_RSTEP);
	}
	if (value < 0) {
		degrees = -degrees;
	}
	wristRPosition = wristRPosition + degrees;
	wristRPosition = normalizeValue(wristRPosition, WRIST_RMIN, WRIST_RMAX);
	if (wristRPosition != ant) {
		TM_SERVO_SetDegrees(&WristRotation, wristRPosition);
		//Delayms(WAIT_SERVO);
	}
}
```

```
/**
 * Method to update apperture of claw.
 */
void updateClaw(int cPressed, int zPressed) {

		if ((cPressed == 0 || zPressed == 0) && (cPressed == 1 || zPressed == 1)) {
			int ant = clawAperture;
			if (cPressed == 1) {
				clawAperture += CLAW_STEP;
			} else if (zPressed == 1) {
				clawAperture -= CLAW_STEP;
			}
			clawAperture = normalizeValue(clawAperture, CLAW_MIN, CLAW_MAX);
			if (clawAperture != ant) {
				TM_SERVO_SetDegrees(&Claw, clawAperture);
				//Delayms(WAIT_SERVO);
			}
		}
}

/**
 * Method to update servo position
 */
void updateArm(int joystickX, int joystickY, int roll, int pitch, int cPressed,		int zPressed) {
		updateShoulderHorizontal(joystickX);
		updateShoulderVertical(pitch, cPressed, zPressed);
		updateCubit(joystickY);
		updateWristVertical(pitch, cPressed, zPressed);
		updateWristRotation(roll);
		updateClaw(cPressed, zPressed);
}

/**
 * Method to print labels on LCD
 */
void lcd_PrintLabels() {

		lcd_strxy("X:",0,0);
		lcd_strxy("Y:",7,0);
		lcd_strxy("C",14,0);

		lcd_strxy("R:",0,1);
		lcd_strxy("P:",7,1);
		lcd_strxy("Z",14,1);
}

/**
 * Method to print nunchuk values on LCD
 */
void lcd_ShowNunchukValues(int joystickX, int joystickY, int roll, int pitch,
				int cPressed, int zPressed) {

		lcd_strxy("    ",2,0);
		lcd_intxy(joystickX,2,0);
		lcd_strxy("    ",9,0);
		lcd_intxy(joystickY,9,0);
		lcd_intxy(cPressed,15,0);

		lcd_strxy("    ",2,1);
		lcd_intxy(roll,2,1);
		lcd_strxy("    ",9,1);
		lcd_intxy(pitch,9,1);
		lcd_intxy(zPressed,15,1);
}

/**
 * Method to calculate servos step.
 */
void initSteps() {
		int iterationSecond = SECONDS_TO_MAX * (1000/LOOP_DELAY);
		SHOULDER_HSTEP = SHOULDER_HMAX/(2 * iterationSecond);
		SHOULDER_VSTEP = SHOULDER_VMAX/(2 * iterationSecond);;
		CUBIT_STEP  = CUBIT_MAX/(2 * iterationSecond);;
		WRIST_VSTEP = WRIST_VMAX/(2 * iterationSecond);
		WRIST_RSTEP = WRIST_RMAX/(2 * iterationSecond);;
		CLAW_STEP = CLAW_MAX/(2 * iterationSecond);;
```

```
}

/**
 **===========================================================================
 **
 **  Abstract: main program
 **
 **===========================================================================
 */
int main(void) {
	//Initialize system
	SystemInit();

	/* Initialize delay */
	TM_DELAY_Init();

	/* Initialize I2C for write/read to/from Nunchuk */
	nunchuk_Init(I2C1, NUNCHUK_ADDRESS, TM_I2C_PinsPack_1, CLOCK_SPEED); // SCL PB6 and SDA
PB7

	/* Initialize servos */
	initServos();

	/* Initialize LCD */
	lcd_init();

	initSteps();

	/* Print Labels on LCD */
	//lcd_PrintLabels();
	lcd_strxy("System OFF Press",0,0);
	lcd_strxy("C+Z 2s to Start",0,1);

	while (1) {		// Infinite Loop
		/* Update Nunchuk values */
		if (nunchuk_Update() == 1) {

			/* Read each values from Nunchuk */
			int joystickX = nunchuk_GetJoystickX();
			int joystickY = nunchuk_GetJoystickY();
			int roll = nunchuk_GetRoll();
			int pitch = nunchuk_GetPitch();
			int cPressed = nunchuk_GetC();
			int zPressed = nunchuk_GetZ();

			if (!encendido) {
				powerOnControl(cPressed, zPressed);
			} else {
				/* Update positions of all servos in arm */
				updateArm(joystickX, joystickY, roll, pitch, cPressed, zPressed);

				/* Show Nunchuk values on LCD */
				lcd_ShowNunchukValues(joystickX, joystickY, roll, pitch, cPressed, zPressed);
			}
		}

		/* Prepare Nunchuk for new read */
		nunchuk_SendZero();

		/* Wait for DELAY milliseconds for a new execution */
		//delay(2500000);
		Delayms(LOOP_DELAY);
	}
}

/*
 * Callback used by stm32f4_discovery_audio_codec.c.
 * Refer to stm32f4_discovery_audio_codec.h for more info.
 */
void EVAL_AUDIO_TransferComplete_CallBack(uint32_t pBuffer, uint32_t Size) {
	/* TODO, implement your code here */
	return;
}

/*
 * Callback used by stm324xg_eval_audio_codec.c.
```

```
 * Refer to stm324xg_eval_audio_codec.h for more info.
 */
uint16_t EVAL_AUDIO_GetSampleCallBack(void) {
	/* TODO, implement your code here */
	return -1;
}
```

utilidades.h

```
void delay(uint32_t delay);
int map(int value, int oMin, int oMax, int dMin, int dMax);
int normalizeValue(int value, int min, int max);
char* intToStr(int value);
```

utilidades.c

```
#include "stm32f4xx.h"
#include <stdio.h>

void delay(uint16_t delay) {
	while (delay > 0) {
		delay--;
	}
}

int map(int value, int oMin, int oMax, int dMin, int dMax) {
	int resultado;
	int numerador = dMax - dMin;
	int denominador = oMax - oMin;
	float division = ((float)numerador / (float)denominador);
	int diferencia = value - oMin;
	resultado = (division * diferencia) + dMin;
	//resultado = (float)(((int)(dMax - dMin)/(oMax - oMin))*(value - oMin)) + dMin;
	return (int) resultado;
}

int normalizeValue(int value, int min, int max) {
	if (value < min) {
		value = min;
	} else if (value > max) {
		value = max;
	}
	return value;
}

char* intToStr(int value) {
	char* str = malloc(4);
	snprintf(str, 10, "%d", value);
	return str;
}
```

Librerías Descargadas

tm_stm32f4_delay.h

```
/**
 * @author  Tilen Majerle
 * @email   tilen@majerle.eu
 * @website http://stm32f4-discovery.com
 * @link    http://stm32f4-discovery.com/2014/04/library-03-stm32f429-discovery-system-clock-and-pretty-precise-delay-library/
 * @version v2.4
 * @ide     Keil uVision
 * @license GNU GPL v3
```

```
 * @brief   Pretty accurate delay functions with SysTick or any other timer
 *
@verbatim
  ----------------------------------------------------------------------
   Copyright (C) Tilen Majerle, 2015

   This program is free software: you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation, either version 3 of the License, or
   any later version.

   This program is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.

   You should have received a copy of the GNU General Public License
   along with this program.  If not, see <http://www.gnu.org/licenses/>.
  ----------------------------------------------------------------------
@endverbatim
 */
#ifndef TM_DELAY_H
#define TM_DELAY_H 240

/* C++ detection */
#ifdef __cplusplus
extern "C" {
#endif

/**
 * @addtogroup TM_STM32F4xx_Libraries
 * @{
 */

/**
 * @defgroup TM_DELAY
 * @brief Pretty accurate delay functions with SysTick or any other timer - http://stm32f4-
discovery.com/2014/04/library-03-stm32f429-discovery-system-clock-and-pretty-precise-delay-library/
 * @{
 *
@verbatim
If you are using GCC compiler, then your microseconds delay is probably totally inaccurate.
USE TIMER FOR DELAY, otherwise your delay will not be accurate.

Another way is to use ARM compiler.
@endverbatim
 *
 * As of version 2.0 you have now two possible ways to make a delay.
 * The first (and default) is Systick timer. It makes interrupts every 1ms.
 * If you want delay in "us" accuracy, there is simple pooling (variable) mode.
 *
 *
 * The second (better) options is to use one of timers on F4xx MCU.
 * Timer also makes an interrupts every 1ms (for count time) instead of 1us as it was before.
 * For "us" delay, timer's counter is used to count ticks. It makes a new tick each "us".
 * Not all MCUs have all possible timers, so this lib has been designed that you select your own.
 *
 * \par Select custom TIM for delay functions
 *
 * By default, Systick timer is used for delay. If you want your custom timer,
 * open defines.h file, add lines below and edit for your needs.
 *
\code{.c}
//Select custom timer for delay, here is TIM2 selected.
//If you want custom TIMx, just replace number "2" for your TIM's number.
#define TM_DELAY_TIM                    TIM2
#define TM_DELAY_TIM_IRQ                TIM2_IRQn
#define TM_DELAY_TIM_IRQ_HANDLER        TIM2_IRQHandler
\endcode
 *
 *
 * With this setting (using custom timer) you have better accuracy in "us" delay.
 * Also, you have to know, that if you want to use timer for delay, you have to include additional files:
 *
 *      - CMSIS:
 *              - STM32F4xx TIM
```

```
 *                        - MISC
 *            - TM:
 *                        TM TIMER PROPERTIES
 *
 * Delay functions (Delay, Delayms) are now Inline functions.
 * This allows faster execution and more accurate delay.
 *
 * If you are working with Keil uVision and you are using Systick for delay,
 * then set KEIL_IDE define in options for project:
 *            - Open "Options for target"
 *            - Tab "C/C++"
 *            - Under "Define" add "KEIL_IDE", without quotes
 *
 * \par Custom timers
 *
 * Custom timers are a way to make some tasks in a periodic value.
 * As of version 2.4, delay library allows you to create custom timer which count DOWN and when it reaches zero,
callback is called.
 *
 * You can use variable settings for count, reload value and auto reload feature.
 *
 * \par Changelog
 *
@verbatim
 Version 2.4
  - May 26, 2015
  - Added support for custom timers which can be called periodically

 Version 2.3
  - April 18, 2015
  - Fixed support for internal RC clock

 Version 2.2
  - January 12, 2015
  - Added support for custom function call each time 1ms interrupt happen
  - Function is called TM_DELAY_1msHandler(void), with __weak parameter
  - attributes.h file needed

 Version 2.1
  - GCC compiler fixes
  - Still prefer that you use TIM for delay if you are working with ARM-GCC compiler

 Version 2.0
  - November 28, 2014
  - Delay library has been totally rewritten. Because Systick is designed to be used
      in RTOS, it is not compatible to use it at the 2 places at the same time.
      For that purpose, library has been rewritten.
  - Read full documentation above

 Version 1.0
  - First release
@endverbatim
 *
 * \par Dependencies
 *
@verbatim
 - STM32F4xx
 - STM32F4xx RCC:      Only if you want to use TIMx for delay instead of Systick
 - STM32F4xx TIM:      Only if you want to use TIMx for delay instead of Systick
 - MISC
 - defines.h
 - TM TIMER PROPERTIES: Only if you want to use TIMx for delay instead of Systick
 - attribute.h
@endverbatim
 */
#include "stm32f4xx.h"
#include "stm32f4xx_rcc.h"
#include "defines.h"
#include "attributes.h"
/* If user selectable timer is selected for delay */
#if defined(TM_DELAY_TIM)
#include "misc.h"
#include "stm32f4xx_tim.h"
#include "tm_stm32f4_timer_properties.h"
#endif
#include "stdlib.h"
```

```
/**
 * @defgroup TM_DELAY_Typedefs
 * @brief    Library Typedefs
 * @{
 */

/**
 * @brief  Custom timer structure
 */
typedef struct {
        uint32_t ARR;           /*!< Auto reload value */
        uint32_t AutoReload;    /*!< Set to 1 if timer should be auto reloaded when it reaches zero */
        uint32_t CNT;           /*!< Counter value, counter counts down */
        uint8_t Enabled;        /*!< Set to 1 when timer is enabled */
        void (*Callback)(void *); /*!< Callback which will be called when timer reaches zero */
        void* UserParameters;   /*!< Pointer to user parameters used for callback function */
} TM_DELAY_Timer_t;

/**
 * @}
 */

/**
 * @defgroup TM_DELAY_Macros
 * @brief    Library Macros
 * @{
 */

/**
 * @brief  Number of allowed custom timers
 * @note   Should be changes in defines.h file if necessary
 */
#ifndef DELAY_MAX_CUSTOM_TIMERS
#define DELAY_MAX_CUSTOM_TIMERS   5
#endif

/* Memory allocation function */
#ifndef LIB_ALLOC_FUNC
#define LIB_ALLOC_FUNC    malloc
#endif

/* Memory free function */
#ifndef LIB_FREE_FUNC
#define LIB_FREE_FUNC     free
#endif

/**
 * @}
 */

/**
 * @defgroup TM_DELAY_Variables
 * @brief    Library Variables
 * @{
 */

/**
 * This variable can be used in main
 * It is automatically increased every time systick make an interrupt
 */
extern __IO uint32_t TM_Time;
extern __IO uint32_t TM_Time2;
extern __IO uint32_t mult;

/**
 * @}
 */

/**
 * @defgroup TM_DELAY_Functions
 * @brief    Library Functions
 * @{
 */

/**
```

```
 * @param  Delays for specific amount of microseconds
 * @param  micros: Time in microseconds for delay
 * @retval None
 * @note   Declared as static inline
 */
static __INLINE void Delay(uint32_t micros) {
#if defined(TM_DELAY_TIM)
	volatile uint32_t timer = TM_DELAY_TIM->CNT;

	do {
		/* Count timer ticks */
		while ((TM_DELAY_TIM->CNT - timer) == 0);

		/* Increase timer */
		timer = TM_DELAY_TIM->CNT;

		/* Decrease microseconds */
	} while (--micros);
#else
	uint32_t amicros;

	/* Multiply micro seconds */
	amicros = (micros) * (mult);

	#ifdef __GNUC__
		if (SystemCoreClock == 180000000 || SystemCoreClock == 100000000) {
			amicros -= mult;
		}
	#endif

	/* If clock is 100MHz, then add additional multiplier */
	/* 100/3 = 33.3 = 33 and delay wouldn't be so accurate */
	#if defined(STM32F411xE)
	amicros += mult;
	#endif

	/* While loop */
	while (amicros--);
#endif /* TM_DELAY_TIM */
}

/**
 * @param  Delays for specific amount of milliseconds
 * @param  millis: Time in milliseconds for delay
 * @retval None
 * @note   Declared as static inline
 */
static __INLINE void Delayms(uint32_t millis) {
	volatile uint32_t timer = TM_Time;

	/* Called from thread */
	if (!__get_IPSR()) {
		/* Wait for timer to count milliseconds */
		while ((TM_Time - timer) < millis) {
#ifdef DELAY_SLEEP
			/* Go sleep, wait systick interrupt */
			__WFI();
#endif
		}
	} else {
		/* Called from interrupt */
		while (millis) {
			if (SysTick->CTRL & SysTick_CTRL_COUNTFLAG_Msk) {
				millis--;
			}
		}
	}
}

/**
 * @brief  Initializes timer settings for delay
 * @note   This function will initialize Systick or user timer, according to settings
 * @param  None
 * @retval None
 */
void TM_DELAY_Init(void);
```

```
/**
 * @brief  Gets the TM_Time variable value
 * @param  None
 * @retval Current time in milliseconds
 */
#define TM_DELAY_Time()                                    (TM_Time)

/**
 * @brief  Sets value for TM_Time variable
 * @param  time: Time in milliseconds
 * @retval None
 */
#define TM_DELAY_SetTime(time)                     (TM_Time = (time))

/**
 * @brief  Re-enables delay timer It has to be configured before with TM_DELAY_Init()
 * @note   This function enables delay timer. It can be systick or user selectable timer.
 * @param  None
 * @retval None
 */
void TM_DELAY_EnableDelayTimer(void);

/**
 * @brief  Disables delay timer
 * @note   This function disables delay timer. It can be systick or user selectable timer.
 * @param  None
 * @retval None
 */
void TM_DELAY_DisableDelayTimer(void);

/**
 * @brief  Gets the TM_Time2 variable value
 * @param  None
 * @retval Current time in milliseconds
 * @note   This is not meant for public use
 */
#define TM_DELAY_Time2()                           (TM_Time2)

/**
 * @brief  Sets value for TM_Time variable
 * @param  time: Time in milliseconds
 * @retval None
 * @note   This is not meant for public use
 */
#define TM_DELAY_SetTime2(time)                    (TM_Time2 = (time))

/**
 * @brief  Creates a new custom timer which has 1ms resolution
 * @note   It uses @ref malloc for memory allocation for timer structure
 * @param  ReloadValue: Number of milliseconds when timer reaches zero and callback function is called
 * @param  AutoReload: If set to 1, timer will start again when it reaches zero and callback is called
 * @param  StartTimer: If set to 1, timer will start immediately
 * @param  *TM_DELAY_CustomTimerCallback: Pointer to callback function which will be called when timer reaches
zero
 * @param  *UserParameters: Pointer to void pointer to user parameters used as first parameter in callback function
 * @retval Pointer to allocated timer structure
 */
TM_DELAY_Timer_t* TM_DELAY_TimerCreate(uint32_t ReloadValue, uint8_t AutoReload, uint8_t StartTimer, void
(*TM_DELAY_CustomTimerCallback)(void *), void* UserParameters);

/**
 * @brief  Deletes already allocated timer
 * @param  *Timer: Pointer to @ref TM_DELAY_Timer_t structure
 * @retval None
 */
void TM_DELAY_TimerDelete(TM_DELAY_Timer_t* Timer);

/**
 * @brief  Stops custom timer from counting
 * @param  *Timer: Pointer to @ref TM_DELAY_Timer_t structure
 * @retval Pointer to @ref TM_DELAY_Timer_t structure
 */
TM_DELAY_Timer_t* TM_DELAY_TimerStop(TM_DELAY_Timer_t* Timer);

/**
```

```
 * @brief  Starts custom timer counting
 * @param  *Timer: Pointer to @ref TM_DELAY_Timer_t structure
 * @retval Pointer to @ref TM_DELAY_Timer_t structure
 */
TM_DELAY_Timer_t* TM_DELAY_TimerStart(TM_DELAY_Timer_t* Timer);

/**
 * @brief  Resets custom timer counter value
 * @param  *Timer: Pointer to @ref TM_DELAY_Timer_t structure
 * @retval Pointer to @ref TM_DELAY_Timer_t structure
 */
TM_DELAY_Timer_t* TM_DELAY_TimerReset(TM_DELAY_Timer_t* Timer);

/**
 * @brief  Sets auto reload feature for timer
 * @note   Auto reload features is used for timer which starts again when zero is reached if auto reload active
 * @param  *Timer: Pointer to @ref TM_DELAY_Timer_t structure
 * uint8_t AutoReload: Set to 1 if you want to enable AutoReload or 0 to disable
 * @retval Pointer to @ref TM_DELAY_Timer_t structure
 */
TM_DELAY_Timer_t* TM_DELAY_TimerAutoReload(TM_DELAY_Timer_t* Timer, uint8_t AutoReload);

/**
 * @brief  Sets auto reload value for timer
 * @param  *Timer: Pointer to @ref TM_DELAY_Timer_t structure
 * @param  AutoReloadValue: Value for timer to be set when zero is reached and callback is called
 * @note   AutoReload feature must be enabled for timer in order to get this to work properly
 * @retval Pointer to @ref TM_DELAY_Timer_t structure
 */
TM_DELAY_Timer_t* TM_DELAY_TimerAutoReloadValue(TM_DELAY_Timer_t* Timer, uint32_t AutoReloadValue);

/**
 * @brief  User function, called each 1ms when interrupt from timer happen
 * @note   Here user should put things which has to be called periodically
 * @param  None
 * @retval None
 * @note   With __weak parameter to prevent link errors if not defined by user
 */
__weak void TM_DELAY_1msHandler(void);


/**
 * @}
 */

/**
 * @}
 */

/**
 * @}
 */

/* C++ detection */
#ifdef __cplusplus
}
#endif

#endif
```

tm_stm32f4_delay.c

```
/**
 * |----------------------------------------------------------------------
 * | Copyright (C) Tilen Majerle, 2014
 * |
 * | This program is free software: you can redistribute it and/or modify
 * | it under the terms of the GNU General Public License as published by
 * | the Free Software Foundation, either version 3 of the License, or
 * | any later version.
 * |
 * | This program is distributed in the hope that it will be useful,
 * | but WITHOUT ANY WARRANTY; without even the implied warranty of
 * | MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
```

```
 * | GNU General Public License for more details.
 * |
 * | You should have received a copy of the GNU General Public License
 * | along with this program.  If not, see <http://www.gnu.org/licenses/>.
 * |----------------------------------------------------------------------
 */
#include "tm_stm32f4_delay.h"

__IO uint32_t TM_TimingDelay = 0;
__IO uint32_t TM_Time = 0;
__IO uint32_t TM_Time2 = 0;
volatile uint32_t mult;
uint8_t TM_DELAY_Initialized = 0;

/* Private structure */
typedef struct {
	uint8_t Count;
	TM_DELAY_Timer_t* Timers[DELAY_MAX_CUSTOM_TIMERS];
} TM_DELAY_Timers_t;

/* Custom timers structure */
static TM_DELAY_Timers_t CustomTimers;

#if defined(TM_DELAY_TIM)
void TM_DELAY_INT_InitTIM(void);
#endif

#if defined(TM_DELAY_TIM)
void TM_DELAY_TIM_IRQ_HANDLER(void) {
	TM_DELAY_TIM->SR = ~TIM_IT_Update;
#elif defined(KEIL_IDE)
void TimingDelay_Decrement(void) {
#else
void SysTick_Handler(void) {
#endif
	uint8_t i;

	TM_Time++;
	if (TM_Time2 != 0x00) {
		TM_Time2--;
	}

	/* Call user function */
	TM_DELAY_1msHandler();

	/* Check custom timers */
	for (i = 0; i < CustomTimers.Count; i++) {
		/* Check if timer is enabled */
		if (
			CustomTimers.Timers[i] &&	/*!< Pointer exists */
			CustomTimers.Timers[i]->Enabled && /*!< Timer is enabled */
			CustomTimers.Timers[i]->CNT > 0	/*!< Counter is not NULL */
		) {
			/* Decrease counter */
			CustomTimers.Timers[i]->CNT--;

			/* Check if count is zero */
			if (CustomTimers.Timers[i]->CNT == 0) {
				/* Call user callback function */
				CustomTimers.Timers[i]->Callback(CustomTimers.Timers[i]->UserParameters);

				/* Set new counter value */
				CustomTimers.Timers[i]->CNT = CustomTimers.Timers[i]->ARR;

				/* Disable timer if auto reload feature is not used */
				if (!CustomTimers.Timers[i]->AutoReload) {
					/* Disable counter */
					CustomTimers.Timers[i]->Enabled = 0;
				}
			}
		}
	}
}

void TM_DELAY_Init(void) {
#if defined(TM_DELAY_TIM)
```

```
	TM_DELAY_INT_InitTIM();
#else
	/* Set Systick interrupt every 1ms */
	if (SysTick_Config(SystemCoreClock / 1000)) {
		/* Capture error */
		while (1);
	}

	#ifdef __GNUC__
		/* Set multiplier for delay under 1us with pooling mode = not so accurate */
		mult = SystemCoreClock / 7000000;
	#else
		/* Set multiplier for delay under 1us with pooling mode = not so accurate */
		mult = SystemCoreClock / 3000000;
	#endif
#endif

	/* Set initialized flag */
	TM_DELAY_Initialized = 1;
}

void TM_DELAY_EnableDelayTimer(void) {
	/* Check if library is even initialized */
	if (!TM_DELAY_Initialized) {
		return;
	}

#if defined(TM_DELAY_TIM)
	/* Enable TIMER for delay, useful when you wakeup from sleep mode */
	TM_DELAY_TIM->CR1 |= TIM_CR1_CEN;
#else
	/* Enable systick interrupts, useful when you wakeup from sleep mode */
	SysTick->CTRL |= SysTick_CTRL_TICKINT_Msk;
#endif
}

void TM_DELAY_DisableDelayTimer(void) {
#if defined(TM_DELAY_TIM)
	/* Disable TIMER for delay, useful when you go to sleep mode */
	TM_DELAY_TIM->CR1 &= ~TIM_CR1_CEN;
#else
	/* Disable systick, useful when you go to sleep mode */
	SysTick->CTRL &= ~SysTick_CTRL_TICKINT_Msk;
#endif
}

/* Internal functions */
#if defined(TM_DELAY_TIM)
void TM_DELAY_INT_InitTIM(void) {
	TIM_TimeBaseInitTypeDef TIM_TimeBaseStruct;
	NVIC_InitTypeDef NVIC_InitStruct;
	TM_TIMER_PROPERTIES_t TIM_Data;

	/* Get timer properties */
	TM_TIMER_PROPERTIES_GetTimerProperties(TM_DELAY_TIM, &TIM_Data);

	/* Generate timer properties, 1us ticks */
	TM_TIMER_PROPERTIES_GenerateDataForWorkingFrequency(&TIM_Data, 1000000);

	/* Enable clock for TIMx */
	TM_TIMER_PROPERTIES_EnableClock(TM_DELAY_TIM);

	/* Set timer settings */
	TIM_TimeBaseStruct.TIM_ClockDivision = 0;
	TIM_TimeBaseStruct.TIM_CounterMode = TIM_CounterMode_Up;
	TIM_TimeBaseStruct.TIM_Period = 999; /* 1 millisecond */
	TIM_TimeBaseStruct.TIM_Prescaler = SystemCoreClock / (1000000 * (SystemCoreClock /
TIM_Data.TimerFrequency)) - 1; /* With prescaler for 1 microsecond tick */
	TIM_TimeBaseStruct.TIM_RepetitionCounter = 0;

	/* Initialize timer */
	TIM_TimeBaseInit(TM_DELAY_TIM, &TIM_TimeBaseStruct);

	/* Enable interrupt each update cycle */
	TM_DELAY_TIM->DIER |= TIM_IT_Update;
```

```
	/* Set NVIC parameters */
	NVIC_InitStruct.NVIC_IRQChannel = TM_DELAY_TIM_IRQ;
	NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
	NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0;
	NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0;

	/* Add to NVIC */
	NVIC_Init(&NVIC_InitStruct);

	/* Start timer */
	TM_DELAY_TIM->CR1 |= TIM_CR1_CEN;
}
#endif

TM_DELAY_Timer_t* TM_DELAY_TimerCreate(uint32_t ReloadValue, uint8_t AutoReload, uint8_t StartTimer, void
(*TM_DELAY_CustomTimerCallback)(void *), void* UserParameters) {
	TM_DELAY_Timer_t* tmp;

	/* Check if available */
	if (CustomTimers.Count >= DELAY_MAX_CUSTOM_TIMERS) {
		return NULL;
	}

	/* Try to allocate memory for timer structure */
	tmp = (TM_DELAY_Timer_t *) LIB_ALLOC_FUNC(sizeof(TM_DELAY_Timer_t));

	/* Check if allocated */
	if (tmp == NULL) {
		return NULL;
	}

	/* Fill settings */
	tmp->ARR = ReloadValue;
	tmp->CNT = tmp->ARR;
	tmp->AutoReload = AutoReload;
	tmp->Enabled = StartTimer;
	tmp->Callback = TM_DELAY_CustomTimerCallback;
	tmp->UserParameters = UserParameters;

	/* Increase number of timers in memory */
	CustomTimers.Timers[CustomTimers.Count++] = tmp;

	/* Return pointer to user */
	return tmp;
}

void TM_DELAY_TimerDelete(TM_DELAY_Timer_t* Timer) {
	uint8_t i;
	uint32_t irq;
	TM_DELAY_Timer_t* tmp;

	/* Get location in array of pointers */
	for (i = 0; i < CustomTimers.Count; i++) {
		if (Timer == CustomTimers.Timers[i]) {
			break;
		}
	}

	/* Check for valid input */
	if (i == CustomTimers.Count) {
		return;
	}

	/* Save pointer to timer */
	tmp = CustomTimers.Timers[i];

	/* Get interrupt status */
	irq = __get_PRIMASK();

	/* Disable interrupts */
	__disable_irq();

	/* Shift array up */
	for (; i < (CustomTimers.Count - 1); i++) {
		/* Shift data to the left */
		CustomTimers.Timers[i] = CustomTimers.Timers[i + 1];
```

```
		}

		/* Decrease count */
		CustomTimers.Count--;

		/* Enable IRQ if necessary */
		if (!irq) {
			__enable_irq();
		}

		/* Free timer */
		LIB_FREE_FUNC(tmp);
}

TM_DELAY_Timer_t* TM_DELAY_TimerStop(TM_DELAY_Timer_t* Timer) {
		/* Disable timer */
		Timer->Enabled = 0;

		/* Return pointer */
		return Timer;
}

TM_DELAY_Timer_t* TM_DELAY_TimerStart(TM_DELAY_Timer_t* Timer) {
		/* Enable timer */
		Timer->Enabled = 1;

		/* Return pointer */
		return Timer;
}

TM_DELAY_Timer_t* TM_DELAY_TimerReset(TM_DELAY_Timer_t* Timer) {
		/* Reset timer */
		Timer->CNT = Timer->ARR;

		/* Return pointer */
		return Timer;
}

TM_DELAY_Timer_t* TM_DELAY_TimerAutoReload(TM_DELAY_Timer_t* Timer, uint8_t AutoReload) {
		/* Reset timer */
		Timer->AutoReload = AutoReload;

		/* Return pointer */
		return Timer;
}

TM_DELAY_Timer_t* TM_DELAY_TimerAutoReloadValue(TM_DELAY_Timer_t* Timer, uint32_t AutoReloadValue) {
		/* Reset timer */
		Timer->ARR = AutoReloadValue;

		/* Return pointer */
		return Timer;
}
```

tm_stm32f4_timer_properties.h

```
/**
 *	Timer properties for all STM32F4xx timers
 *
 *	@author  Tilen Majerle
 *	@email		tilen@majerle.eu
 *	@websitehttp://stm32f4-discovery.com
 *	@link
 *	@version	v1.0
 *	@ide		Keil uVision
 *	@license GNU GPL v3
 *
 * |----------------------------------------------------------------------
 * | Copyright (C) Tilen Majerle, 2014
 * |
 * | This program is free software: you can redistribute it and/or modify
 * | it under the terms of the GNU General Public License as published by
```

```
 * | the Free Software Foundation, either version 3 of the License, or
 * | any later version.
 * |
 * | This program is distributed in the hope that it will be useful,
 * | but WITHOUT ANY WARRANTY; without even the implied warranty of
 * | MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * | GNU General Public License for more details.
 * |
 * | You should have received a copy of the GNU General Public License
 * | along with this program.  If not, see <http://www.gnu.org/licenses/>.
 * |----------------------------------------------------------------------
 *
 * This library allows you to enable/disable clock for specific timer,
 * and returns you settings for timer, ex. max period, max prescaler and timer's clock
 *
 * Also, it is used in every library where timers are used.
 */
#ifndef TM_TIMER_PROPERTIES_H
#define TM_TIMER_PROPERTIES_H   100
/**
 * Dependencies
 * - STM32F4xx
 * - STM32F4xx RCC
 * - STM32F4xx TIM
 * - defines.h
 */
/**
 * Includes
 */
#include "stm32f4xx.h"
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_tim.h"
#include "defines.h"

/**
 * Result enumeration
 *
 * Parameters:
 *		- TM_TIMER_PROPERTIES_Result_Ok:
 *			Everything OK
 *		- TM_TIMER_PROPERTIES_Result_Error:
 *			An error occured
 *		- TM_TIMER_PROPERTIES_Result_TimerNotValid:
 *			Timer is not valid
 *		- TM_TIMER_PROPERTIES_Result_FrequencyTooHigh:
 *			Frequency for timer is too high
 *		- TM_TIMER_PROPERTIES_Result_FrequencyTooLow:
 *			Frequency for timer is too low
 */
typedef enum {
		TM_TIMER_PROPERTIES_Result_Ok,
		TM_TIMER_PROPERTIES_Result_Error,
		TM_TIMER_PROPERTIES_Result_TimerNotValid,
		TM_TIMER_PROPERTIES_Result_FrequencyTooHigh,
		TM_TIMER_PROPERTIES_Result_FrequencyTooLow
} TM_TIMER_PROPERTIES_Result_t;

/**
 * Struct for timer data
 *
 * Parameters:
 *		- uint32_t TimerFrequency:
 *			timer's working frequency
 *		- uint32_t MaxPeriod:
 *			Max timer period
 *		- uint32_t MaxPrescaler:
 *			Max timer prescaler
 *		- uint32_t Period:
 *			Timer's working period
 *		- uint32_t Prescaler:
 *			Timer's working prescaler
 *		- uint32_t Frequency:
 *			Timer's reload frequency
 */
typedef struct {
		uint32_t TimerFrequency;
```

```
	uint32_t MaxPeriod;
	uint32_t MaxPrescaler;
	uint32_t Period;
	uint32_t Prescaler;
	uint32_t Frequency;
} TM_TIMER_PROPERTIES_t;

/**
 * Returns you a timer properties
 *
 * Parameters:
 *	- TIM_TypeDef* TIMx:
 *		Timer used to get settings for
 *	- TM_TIMER_PROPERTIES_t* Timer_Data:
 *		Pointer to TM_TIMER_PROPERTIES_t struct to store data to
 *
 * Member of TM_TIMER_PROPERTIES_Result_t is returned
 */
extern TM_TIMER_PROPERTIES_Result_t TM_TIMER_PROPERTIES_GetTimerProperties(TIM_TypeDef* TIMx,
TM_TIMER_PROPERTIES_t* Timer_Data);

/**
 * Generate period and prescaller for given timer frequency
 *
 * Parameters:
 *	- TM_TIMER_PROPERTIES_t* Timer_Data:
 *		Pointer for timer data
 *	- uint32_t frequency:
 *		Frequency used
 *
 * Member of TM_TIMER_PROPERTIES_Result_t is returned
 */
extern TM_TIMER_PROPERTIES_Result_t
TM_TIMER_PROPERTIES_GenerateDataForWorkingFrequency(TM_TIMER_PROPERTIES_t* Timer_Data, double
frequency);

/**
 * Enable timer's clock
 *
 * Parameters:
 *	- TIM_TypeDef* TIMx:
 *		Timer to enable clock for
 *
 * Member of TM_TIMER_PROPERTIES_Result_t is returned
 */
extern TM_TIMER_PROPERTIES_Result_t TM_TIMER_PROPERTIES_EnableClock(TIM_TypeDef* TIMx);

/**
 * Disable timer's clock
 *
 * Parameters:
 *	- TIM_TypeDef* TIMx:
 *		Timer to disable clock for
 *
 * Member of TM_TIMER_PROPERTIES_Result_t is returned
 */
extern TM_TIMER_PROPERTIES_Result_t TM_TIMER_PROPERTIES_DisableClock(TIM_TypeDef* TIMx);

#endif
```

tm_stm32f4_timer_properties.c

```
/**
 * |----------------------------------------------------------------------
 * | Copyright (C) Tilen Majerle, 2014
 * |
 * | This program is free software: you can redistribute it and/or modify
 * | it under the terms of the GNU General Public License as published by
 * | the Free Software Foundation, either version 3 of the License, or
 * | any later version.
 * |
 * | This program is distributed in the hope that it will be useful,
 * | but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
 * | MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * | GNU General Public License for more details.
 * |
 * | You should have received a copy of the GNU General Public License
 * | along with this program.  If not, see <http://www.gnu.org/licenses/>.
 * |----------------------------------------------------------------------
 */
#include "tm_stm32f4_timer_properties.h"

TM_TIMER_PROPERTIES_Result_t	TM_TIMER_PROPERTIES_GetTimerProperties(TIM_TypeDef*	TIMx,
TM_TIMER_PROPERTIES_t* Timer_Data) {
	RCC_ClocksTypeDef RCC_ClocksStruct;

	/* Get clocks */
	RCC_GetClocksFreq(&RCC_ClocksStruct);

	/* All timers have 16-bit prescaler */
	Timer_Data->MaxPrescaler = 0xFFFF;

	if ( /* 32bit timers with PCLK2 max frequency */
		TIMx == TIM2 ||
		TIMx == TIM5
	) {
		Timer_Data->TimerFrequency = RCC_ClocksStruct.PCLK2_Frequency;	/* Clock */
		Timer_Data->MaxPeriod = 0xFFFFFFFF;
	/* Max period */

		return TM_TIMER_PROPERTIES_Result_Ok;
	} else if (  /* 16bit timers with HCLK clock frequency */
		TIMx == TIM1 ||
		TIMx == TIM8 ||
		TIMx == TIM9 ||
		TIMx == TIM10 ||
		TIMx == TIM11
	) {
		Timer_Data->TimerFrequency = RCC_ClocksStruct.HCLK_Frequency;	/* Clock */
		Timer_Data->MaxPeriod = 0xFFFF;
	/* Max period */

		return TM_TIMER_PROPERTIES_Result_Ok;
	} else if (  /* 16bit timers with PCLK2 clock frequency */
		TIMx == TIM3 ||
		TIMx == TIM4 ||
		TIMx == TIM6 ||
		TIMx == TIM7 ||
		TIMx == TIM12 ||
		TIMx == TIM13 ||
		TIMx == TIM14
	) {
		Timer_Data->TimerFrequency = RCC_ClocksStruct.PCLK2_Frequency;	/* Clock */
		Timer_Data->MaxPeriod = 0xFFFF;
	/* Max period */

		return TM_TIMER_PROPERTIES_Result_Ok;
	}
	/* Timer is not valid */
	return TM_TIMER_PROPERTIES_Result_TimerNotValid;
}

TM_TIMER_PROPERTIES_Result_t
TM_TIMER_PROPERTIES_GenerateDataForWorkingFrequency(TM_TIMER_PROPERTIES_t*	Timer_Data,	double
frequency) {
	if (frequency > Timer_Data->TimerFrequency) {
		/* Reset values */
		Timer_Data->Prescaler = 0;
		Timer_Data->Period = 0;
		Timer_Data->Frequency = 0;

		/* Frequency too high */
		return TM_TIMER_PROPERTIES_Result_FrequencyTooHigh;
	} else if (frequency == 0) {
		/* Reset values */
		Timer_Data->Prescaler = 0;
		Timer_Data->Period = 0;
		Timer_Data->Frequency = 0;
```

```
		/* Not valid frequency */
		return TM_TIMER_PROPERTIES_Result_FrequencyTooLow;
	}

	/* Fix for 16/32bit timers */
	if (Timer_Data->MaxPeriod <= 0xFFFF) {
		Timer_Data->MaxPeriod++;
	}

	/* Get minimum prescaler and maximum resolution for timer */
	Timer_Data->Prescaler = 0;
	do {
		/* Get clock */
		Timer_Data->Period = (Timer_Data->TimerFrequency / (Timer_Data->Prescaler + 1));
		/* Get period */
		Timer_Data->Period = (Timer_Data->Period / frequency);
		/* Increase prescaler value */
		Timer_Data->Prescaler++;
	} while (Timer_Data->Period > (Timer_Data->MaxPeriod) && Timer_Data->Prescaler <= (Timer_Data-
>MaxPrescaler + 1));
	/* Check for too low frequency */
	if (Timer_Data->Prescaler > (Timer_Data->MaxPrescaler + 1)) {
		/* Reset values */
		Timer_Data->Prescaler = 0;
		Timer_Data->Period = 0;
		Timer_Data->Frequency = 0;

		/* Prescaler too high, frequency is too low for use */
		return TM_TIMER_PROPERTIES_Result_FrequencyTooLow;
	}

	/* Set frequency */
	Timer_Data->Frequency = frequency;

	/* Return OK */
	return TM_TIMER_PROPERTIES_Result_Ok;
}

TM_TIMER_PROPERTIES_Result_t TM_TIMER_PROPERTIES_EnableClock(TIM_TypeDef* TIMx) {
	if (TIMx == TIM1) {
		RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1, ENABLE);
	} else if (TIMx == TIM2) {
		RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
	} else if (TIMx == TIM3) {
		RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);
	} else if (TIMx == TIM4) {
		RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);
	} else if (TIMx == TIM5) {
		RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM5, ENABLE);
	} else if (TIMx == TIM6) {
		RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM6, ENABLE);
	} else if (TIMx == TIM7) {
		RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM7, ENABLE);
	} else if (TIMx == TIM8) {
		RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM8, ENABLE);
	} else if (TIMx == TIM9) {
		RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM9, ENABLE);
	} else if (TIMx == TIM10) {
		RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM10, ENABLE);
	} else if (TIMx == TIM11) {
		RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM11, ENABLE);
	} else if (TIMx == TIM12) {
		RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM12, ENABLE);
	} else if (TIMx == TIM13) {
		RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM13, ENABLE);
	} else if (TIMx == TIM14) {
		RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM14, ENABLE);
	}
	/* Return OK */
	return TM_TIMER_PROPERTIES_Result_Ok;
}

TM_TIMER_PROPERTIES_Result_t TM_TIMER_PROPERTIES_DisableClock(TIM_TypeDef* TIMx) {
	if (TIMx == TIM1) {
		RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1, DISABLE);
	} else if (TIMx == TIM2) {
```

```
		RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, DISABLE);
	} else if (TIMx == TIM3) {
		RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, DISABLE);
	} else if (TIMx == TIM4) {
		RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, DISABLE);
	} else if (TIMx == TIM5) {
		RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM5, DISABLE);
	} else if (TIMx == TIM6) {
		RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM6, DISABLE);
	} else if (TIMx == TIM7) {
		RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM7, DISABLE);
	} else if (TIMx == TIM8) {
		RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM8, DISABLE);
	} else if (TIMx == TIM9) {
		RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM9, DISABLE);
	} else if (TIMx == TIM10) {
		RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM10, DISABLE);
	} else if (TIMx == TIM11) {
		RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM11, DISABLE);
	} else if (TIMx == TIM12) {
		RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM12, DISABLE);
	} else if (TIMx == TIM13) {
		RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM13, DISABLE);
	} else if (TIMx == TIM14) {
		RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM14, DISABLE);
	}
	/* Return OK */
	return TM_TIMER_PROPERTIES_Result_Ok;
}
```

tm_stm32f4_pwm.h

```
/**
 * @author  Tilen Majerle
 * @email   tilen@majerle.eu
 * @website http://stm32f4-discovery.com
 * @link    http://stm32f4-discovery.com/2014/09/library-33-pwm-for-stm32f4xx/
 * @version v2.1
 * @ide     Keil uVision
 * @license GNU GPL v3
 * @brief   PWM library for STM32F4xx devices, supporting all possible timers with PWM feature
 *
@verbatim
   ----------------------------------------------------------------------
    Copyright (C) Tilen Majerle, 2015

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
   ----------------------------------------------------------------------
@endverbatim
 */
#ifndef TM_PWM_H
#define TM_PWM_H 210
/**
 * @addtogroup TM_STM32F4xx_Libraries
 * @{
 */

/**
 * @defgroup TM_PWM
 * @brief    PWM library for STM32F4xx devices, supporting all possible timers with PWM feature - http://stm32f4-discovery.com/2014/09/library-33-pwm-for-stm32f4xx/
```

```
 * @{
 *
 * This library allows you to use PWM feature on any timer with supported PWM output
 *
 * \par Pinout
 *
 * PWM pins are connected to fixed possible pins
 *
@verbatim
TIMER   |CHANNEL 1          |CHANNEL 2          |CHANNEL 3          |CHANNEL 4
      |PP1    PP2    PP3    |PP1    PP2    PP3    |PP1    PP2    PP3    |PP1    PP2    PP3

TIM 1   |PA8    PE9    -      |PA9    PE10   -      |PA10   PE13   -      |PA11   PE14   -
TIM 2   |PA0    PA5    PA15   |PA1    PB3    -      |PA2    PB10   -      |PA3    PB11   -
TIM 3   |PA6    PB4    PC6    |PA7    PB5    PC7    |PB0    PC8    -      |PB1    PC9    -
TIM 4   |PB6    PD12   -      |PB7    PD13   -      |PB8    PD14   -      |PB9    PD15   -
TIM 5   |PA0    PH10   -      |PA1    PH11   -      |PA2    PH12   -      |PA3    PI0    -
TIM 8   |PC6    PI5    -      |PC7    PI6    -      |PC8    PI7    -      |PC9    PI2    -
TIM 9   |PA2    PE5    -      |PA3    PE6    -      |-      -      -      |-      -      -
TIM 10  |PB8    PF6    -      |-      -      -      |-      -      -      |-      -      -
TIM 11  |PB9    PF7    -      |-      -      -      |-      -      -      |-      -      -
TIM 12  |PB14   PH6    -      |PB15   PH9    -      |-      -      -      |-      -      -
TIM 13  |PA6    PF8    -      |-      -      -      |-      -      -      |-      -      -
TIM 14  |PA7    PF9    -      |-      -      -      |-      -      -      |-      -      -
@endverbatim
 *
 *          - PPx: Pins Pack 1 to 3, for 3 possible channel outputs on timer.
 *
 * Notes on table above:
 *          - Not all timers are available on all STM32F4xx devices
 *          - All timers have 16-bit prescaler
 *          - TIM6 and TIM7 don't have PWM feature, they are only basic timers
 *          - TIM2 and TIM5 are 32bit timers
 *          - TIM9 and TIM12 have two PWM channels
 *          - TIM10, TIM11, TIM13 and TIM14 have only one PWM channel
 *          - All channels at one timer have the same PWM frequency!
 *
 * \par Changelog
 *
@verbatim
 Version 2.1
  - March 15, 2015
  - Added support for new GPIO library

 Version 2.0
  - January 03, 2015
  - Changed parameters for functions

 Version 1.1
  - August 15, 2014
  - Split timer properties with enable/disable clocks into new library. This library will be used
              in each project where timers are included.

 Version 1.0
  - First release
@endverbatim
 *
 * \par Dependencies
 *
@verbatim
 - STM32F4xx
 - STM32F4xx RCC
 - STM32F4xx GPIO
 - STM32F4xx TIM
 - defines.h
 - TM TIMER PROPERTIES
 - TM GPIO
@endverbatim
 */
#include "stm32f4xx.h"
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_tim.h"
#include "tm_stm32f4_timer_properties.h"
#include "tm_stm32f4_gpio.h"
#include "defines.h"
```

```
/**
 * @defgroup TM_PWM_Typedefs
 * @brief    Library Typedefs
 * @{
 */

/**
 * @brief  PWM Result enumeration
 */
typedef enum {
	TM_PWM_Result_Ok = 0,		/*!< Everything OK */
	TM_PWM_Result_FrequencyTooHigh, /*!< You select too high frequency for timer for PWM */
	TM_PWM_Result_FrequencyTooLow,  /*!< Prescaler value is too big for selected frequency */
	TM_PWM_Result_PulseTooHigh,	/*!< Pulse for Output compare is larger than timer period */
	TM_PWM_Result_TimerNotValid,	/*!< Selected timer is not valid. This happens when you select TIM6 or TIM7,
				because they don't have PWM capability */
	TM_PWM_Result_ChannelNotValid,  /*!< Channel is not valid. Some timers don't have all 4 timers available for PWM */
	TM_PWM_Result_PinNotValid	/*!< Selected pin is not valid. Most channels have only 2 possible pins for PWM,
				but some 3. If you select pin 3 on channel that don't have 3rd pin available
				for PWM, this will be returned */
} TM_PWM_Result_t;

/**
 * @brief  PWM Timer data
 */
typedef struct {
	TIM_TypeDef* TIM;	/*!< Pointer to timer used */
	uint32_t Period;	/*!< Period used, set on initialization for PWM */
	uint32_t Prescaler;	/*!< Prescaler used for PWM frequency */
	uint32_t Frequency;	/*!< PWM frequency used */
	uint32_t Micros;	/*!< Microseconds used for one period.
				This is not useful in large pwm frequency, but good for controlling servos or similar,
				Where you need exact time of pulse high */
	uint32_t CH_Periods[4]; /*!< Array of periods for PWM compare */
	uint8_t CH_Init;	/*!< Flag to check if specific channel is already initialized */
} TM_PWM_TIM_t;

/**
 * @brief  Channel selection for PWM on specific timer
 */
typedef enum {
	TM_PWM_Channel_1 = 0x00, /*!<  Operate with channel 1 */
	TM_PWM_Channel_2 = 0x01, /*!<  Operate with channel 2 */
	TM_PWM_Channel_3 = 0x02, /*!<  Operate with channel 3 */
	TM_PWM_Channel_4 = 0x03  /*!<  Operate with channel 4 */
} TM_PWM_Channel_t;

/**
 * @brief  Pin selected for corresponding channel on specific channel
 */
typedef enum {
	TM_PWM_PinsPack_1 = 0x00, /*!< Pinspack 1 from pinout table */
	TM_PWM_PinsPack_2,	/*!< Pinspack 2 from pinout table */
	TM_PWM_PinsPack_3	/*!< Pinspack 3 from pinout table */
} TM_PWM_PinsPack_t;

/**
 * @}
 */

/**
 * @defgroup TM_PWM_Functions
 * @brief    Library Functions
 * @{
 */

/**
 * @brief  Initializes specific timer for PWM capability
 * @param  *TIMx: Pointer to selected timer, you want to use for PWM
 * @param  *TIM_Data: Pointer to blank @ref TM_PWM_TIM_t structure. Here will init function save all data for specific timer
 * @param  PWMFrequency: Select custom frequency for PWM
```

```
 * @retval Member of @ref TM_PWM_Result_t
 */
TM_PWM_Result_t TM_PWM_InitTimer(TIM_TypeDef* TIMx, TM_PWM_TIM_t* TIM_Data, double PWMFrequency);

/**
 * @brief  Initializes channel used for specific timer
 * @param  *TIM_Data: Pointer to struct with already initialized timer for PWM
 * @param  Channel: Select channel you will use on specific timer. This parameter can be a value of @ref
TM_PWM_Channel_t
 * @param  PinsPack: Select which pinspack you will use for pin. This parameter can be a value of @ref
TM_PWM_PinsPack_t
 * @retval Member of @ref TM_PWM_Result_t
 */
TM_PWM_Result_t TM_PWM_InitChannel(TM_PWM_TIM_t* TIM_Data, TM_PWM_Channel_t Channel,
TM_PWM_PinsPack_t PinsPack);

/**
 * @brief  Sets PWM value for specific timer and channel
 * @param  *TIM_Data: Pointer to struct with already initialized timer for PWM
 * @param  Channel: Select channel you will use on specific timer. This parameter can be a value of @ref
TM_PWM_Channel_t
 * @param  Pulse: Pulse, to be set for compare match
 * @retval Member of @ref TM_PWM_Result_t
 */
TM_PWM_Result_t TM_PWM_SetChannel(TM_PWM_TIM_t* TIM_Data, TM_PWM_Channel_t Channel, uint32_t
Pulse);

/**
 * @brief  Sets PWM value for specific timer and channel with percentage feature
 * @param  *TIM_Data: Pointer to struct with already initialized timer for PWM
 * @param  Channel: Select channel you will use on specific timer. This parameter can be a value of @ref
TM_PWM_Channel_t
 * @param  percent: Percentage from 0 to 100, to set PWM duty cycle
 * @retval Member of @ref TM_PWM_Result_t
 */
TM_PWM_Result_t TM_PWM_SetChannelPercent(TM_PWM_TIM_t* TIM_Data, TM_PWM_Channel_t Channel, float
percent);

/**
 * @brief  Sets PWM value for specific timer and channel with pulse high time feature.
 * @note   You have to specify amount of micro seconds pulse will be high for specific timer and channel.
 *         This method is not so good on PWM large than 100k, because you cannot set nice and correct settings, with
microseconds
 *        accuracy. It is perfect, and was designed for low frequencies, eg. use with servos, where you have exact amount
of time
 *        for servo's rotation.
 * @param  *TIM_Data: Pointer to struct with already initialized timer for PWM
 * @param  Channel: Select channel you will use on specific timer. This parameter can be a value of @ref
TM_PWM_Channel_t
 * @param  micros: Microseconds for pulse high on PWM. Cannot be large than timer period in micros.
 *                 PWM 1kHz = Timer period = 1000000 / 1000 = 1000us. This parameter can not be greater than
1000us in this case.
 * @retval Member of @ref TM_PWM_Result_t
 */
TM_PWM_Result_t TM_PWM_SetChannelMicros(TM_PWM_TIM_t* TIM_Data, TM_PWM_Channel_t Channel,
uint32_t micros);

/**
 * @}
 */

/**
 * @}
 */

/**
 * @}
 */

#endif
```

tm_stm32f4_pwm.c

```
/**
 * |----------------------------------------------------------------------
 * | Copyright (C) Tilen Majerle, 2014
 * |
 * | This program is free software: you can redistribute it and/or modify
 * | it under the terms of the GNU General Public License as published by
 * | the Free Software Foundation, either version 3 of the License, or
 * | any later version.
 * |
 * | This program is distributed in the hope that it will be useful,
 * | but WITHOUT ANY WARRANTY; without even the implied warranty of
 * | MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * | GNU General Public License for more details.
 * |
 * | You should have received a copy of the GNU General Public License
 * | along with this program.  If not, see <http://www.gnu.org/licenses/>.
 * |----------------------------------------------------------------------
 */
#include "tm_stm32f4_pwm.h"

/* Private functions */
void TM_PWM_INT_EnableClock(TIM_TypeDef* TIMx);
TM_PWM_Result_t TM_PWM_INT_GetTimerProperties(TIM_TypeDef* TIMx, uint32_t* frequency, uint32_t* period);

TM_PWM_Result_t TM_PWM_INT_InitTIM1Pins(TM_PWM_Channel_t Channel, TM_PWM_PinsPack_t PinsPack);
TM_PWM_Result_t TM_PWM_INT_InitTIM2Pins(TM_PWM_Channel_t Channel, TM_PWM_PinsPack_t PinsPack);
TM_PWM_Result_t TM_PWM_INT_InitTIM3Pins(TM_PWM_Channel_t Channel, TM_PWM_PinsPack_t PinsPack);
TM_PWM_Result_t TM_PWM_INT_InitTIM4Pins(TM_PWM_Channel_t Channel, TM_PWM_PinsPack_t PinsPack);
TM_PWM_Result_t TM_PWM_INT_InitTIM5Pins(TM_PWM_Channel_t Channel, TM_PWM_PinsPack_t PinsPack);
TM_PWM_Result_t TM_PWM_INT_InitTIM8Pins(TM_PWM_Channel_t Channel, TM_PWM_PinsPack_t PinsPack);
TM_PWM_Result_t TM_PWM_INT_InitTIM9Pins(TM_PWM_Channel_t Channel, TM_PWM_PinsPack_t PinsPack);
TM_PWM_Result_t TM_PWM_INT_InitTIM10Pins(TM_PWM_Channel_t Channel, TM_PWM_PinsPack_t PinsPack);
TM_PWM_Result_t TM_PWM_INT_InitTIM11Pins(TM_PWM_Channel_t Channel, TM_PWM_PinsPack_t PinsPack);
TM_PWM_Result_t TM_PWM_INT_InitTIM12Pins(TM_PWM_Channel_t Channel, TM_PWM_PinsPack_t PinsPack);
TM_PWM_Result_t TM_PWM_INT_InitTIM13Pins(TM_PWM_Channel_t Channel, TM_PWM_PinsPack_t PinsPack);
TM_PWM_Result_t TM_PWM_INT_InitTIM14Pins(TM_PWM_Channel_t Channel, TM_PWM_PinsPack_t PinsPack);

TM_PWM_Result_t TM_PWM_InitTimer(TIM_TypeDef* TIMx, TM_PWM_TIM_t* TIM_Data, double PWMFrequency) {
	TIM_TimeBaseInitTypeDef TIM_BaseStruct;
	TM_TIMER_PROPERTIES_t Timer_Data;

	/* Check valid timer */
	if (0
#ifdef TIM6
			|| TIMx == TIM6
#endif
#ifdef TIM7
			|| TIMx == TIM7
#endif
	) {
			/* Timers TIM6 and TIM7 can not provide PWM feature */
			return TM_PWM_Result_TimerNotValid;
	}

	/* Save timer */
	TIM_Data->TIM = TIMx;

	/* Get timer properties */
	TM_TIMER_PROPERTIES_GetTimerProperties(TIMx, &Timer_Data);

	/* Check for maximum timer frequency */
	if (PWMFrequency > Timer_Data.TimerFrequency) {
		/* Frequency too high */
		return TM_PWM_Result_FrequencyTooHigh;
	} else if (PWMFrequency == 0) {
		/* Not valid frequency */
		return TM_PWM_Result_FrequencyTooLow;
	}

	/* Generate settings */
	TM_TIMER_PROPERTIES_GenerateDataForWorkingFrequency(&Timer_Data, PWMFrequency);
```

```
	/* Check valid data */
	if (Timer_Data.Period == 0) {
		/* Too high frequency */
		return TM_PWM_Result_FrequencyTooHigh;
	}

	/* Tests are OK */
	TIM_Data->Frequency = PWMFrequency;
	TIM_Data->Micros = 1000000 / PWMFrequency;
	TIM_Data->Period = Timer_Data.Period;
	TIM_Data->Prescaler = Timer_Data.Prescaler;

	/* Enable clock for Timer */
	TM_TIMER_PROPERTIES_EnableClock(TIMx);

	/* Set timer options */
	TIM_BaseStruct.TIM_Prescaler = Timer_Data.Prescaler - 1;
	TIM_BaseStruct.TIM_CounterMode = TIM_CounterMode_Up;
	TIM_BaseStruct.TIM_Period = Timer_Data.Period - 1;
	TIM_BaseStruct.TIM_ClockDivision = TIM_CKD_DIV1;
	TIM_BaseStruct.TIM_RepetitionCounter = 0;

	/* Initialize timer */
	TIM_TimeBaseInit(TIMx, &TIM_BaseStruct);

	/* Preload enable */
	TIM_ARRPreloadConfig(TIMx, ENABLE);

	if (0
#ifdef TIM1
		|| TIMx == TIM1
#endif
#ifdef TIM8
		|| TIMx == TIM8
#endif
	) {
		/* Enable PWM outputs */
		TIM_CtrlPWMOutputs(TIMx, ENABLE);
	}

	/* Start timer */
	TIMx->CR1 |= TIM_CR1_CEN;

	/* Set default values */
	TIM_Data->CH_Init = 0;

	/* Return OK */
	return TM_PWM_Result_Ok;
}


TM_PWM_Result_t	TM_PWM_InitChannel(TM_PWM_TIM_t*	TIM_Data,	TM_PWM_Channel_t	Channel,
TM_PWM_PinsPack_t PinsPack) {
#ifdef TIM1
	if (TIM_Data->TIM == TIM1) {
		return TM_PWM_INT_InitTIM1Pins(Channel, PinsPack);
	}
#endif
#ifdef TIM2
	if (TIM_Data->TIM == TIM2) {
		return TM_PWM_INT_InitTIM2Pins(Channel, PinsPack);
	}
#endif
#ifdef TIM3
	if (TIM_Data->TIM == TIM3) {
		return TM_PWM_INT_InitTIM3Pins(Channel, PinsPack);
	}
#endif
#ifdef TIM4
	if (TIM_Data->TIM == TIM4) {
		return TM_PWM_INT_InitTIM4Pins(Channel, PinsPack);
	}
#endif
#ifdef TIM5
	if (TIM_Data->TIM == TIM5) {
		return TM_PWM_INT_InitTIM5Pins(Channel, PinsPack);
```

```
		}
#endif
#ifdef TIM8
		if (TIM_Data->TIM == TIM8) {
			return TM_PWM_INT_InitTIM8Pins(Channel, PinsPack);
		}
#endif
#ifdef TIM9
		if (TIM_Data->TIM == TIM9) {
			return TM_PWM_INT_InitTIM9Pins(Channel, PinsPack);
		}
#endif
#ifdef TIM10
		if (TIM_Data->TIM == TIM10) {
			return TM_PWM_INT_InitTIM10Pins(Channel, PinsPack);
		}
#endif
#ifdef TIM11
		if (TIM_Data->TIM == TIM11) {
			return TM_PWM_INT_InitTIM11Pins(Channel, PinsPack);
		}
#endif
#ifdef TIM12
		if (TIM_Data->TIM == TIM12) {
			return TM_PWM_INT_InitTIM12Pins(Channel, PinsPack);
		}
#endif
#ifdef TIM13
		if (TIM_Data->TIM == TIM13) {
			return TM_PWM_INT_InitTIM13Pins(Channel, PinsPack);
		}
#endif
#ifdef TIM14
		if (TIM_Data->TIM == TIM14) {
			return TM_PWM_INT_InitTIM14Pins(Channel, PinsPack);
		}
#endif

		/* Timer is not valid */
		return TM_PWM_Result_TimerNotValid;
}

TM_PWM_Result_t TM_PWM_SetChannel(TM_PWM_TIM_t* TIM_Data, TM_PWM_Channel_t Channel, uint32_t
Pulse) {
		TIM_OCInitTypeDef TIM_OCStruct;
		uint8_t ch = (uint8_t)Channel;

		/* Check pulse length */
		if (Pulse >= TIM_Data->Period) {
			/* Pulse too high */
			return TM_PWM_Result_PulseTooHigh;
		}

		/* Common settings */
		TIM_OCStruct.TIM_Pulse = Pulse;
		TIM_OCStruct.TIM_OCMode = TIM_OCMode_PWM2;
		TIM_OCStruct.TIM_OutputState = TIM_OutputState_Enable;
		TIM_OCStruct.TIM_OCPolarity = TIM_OCPolarity_Low;

		/* Save pulse value */
		TIM_Data->CH_Periods[ch] = Pulse;

		/* Go to bits */
		ch = 1 << ch;

		/* Select proper channel */
		switch (Channel) {
			case TM_PWM_Channel_1:
				/* Check if initialized */
				if (!(TIM_Data->CH_Init & ch)) {
					TIM_Data->CH_Init |= ch;

					/* Init channel */
					TIM_OC1Init(TIM_Data->TIM, &TIM_OCStruct);
					TIM_OC1PreloadConfig(TIM_Data->TIM, TIM_OCPreload_Enable);
				}
```

```
		/* Set pulse */
		TIM_Data->TIM->CCR1 = Pulse;
		break;
	case TM_PWM_Channel_2:
		/* Check if initialized */
		if (!(TIM_Data->CH_Init & ch)) {
			TIM_Data->CH_Init |= ch;

			/* Init channel */
			TIM_OC2Init(TIM_Data->TIM, &TIM_OCStruct);
			TIM_OC2PreloadConfig(TIM_Data->TIM, TIM_OCPreload_Enable);
		}

		/* Set pulse */
		TIM_Data->TIM->CCR2 = Pulse;
		break;
	case TM_PWM_Channel_3:
		/* Check if initialized */
		if (!(TIM_Data->CH_Init & ch)) {
			TIM_Data->CH_Init |= ch;

			/* Init channel */
			TIM_OC3Init(TIM_Data->TIM, &TIM_OCStruct);
			TIM_OC3PreloadConfig(TIM_Data->TIM, TIM_OCPreload_Enable);
		}

		/* Set pulse */
		TIM_Data->TIM->CCR3 = Pulse;
		break;
	case TM_PWM_Channel_4:
		/* Check if initialized */
		if (!(TIM_Data->CH_Init & ch)) {
			TIM_Data->CH_Init |= ch;

			/* Init channel */
			TIM_OC4Init(TIM_Data->TIM, &TIM_OCStruct);
			TIM_OC4PreloadConfig(TIM_Data->TIM, TIM_OCPreload_Enable);
		}

		/* Set pulse */
		TIM_Data->TIM->CCR4 = Pulse;
		break;
	default:
		break;
	}

	/* Return everything OK */
	return TM_PWM_Result_Ok;
}

TM_PWM_Result_t TM_PWM_SetChannelPercent(TM_PWM_TIM_t* TIM_Data, TM_PWM_Channel_t Channel, float percent) {
	/* Check input value */
	if (percent > 100) {
		return TM_PWM_SetChannel(TIM_Data, Channel, TIM_Data->Period);
	} else if (percent <= 0) {
		return TM_PWM_SetChannel(TIM_Data, Channel, 0);
	}

	/* Set channel value */
	return TM_PWM_SetChannel(TIM_Data, Channel, (uint32_t)((float)(TIM_Data->Period - 1) * percent) / 100);
}

TM_PWM_Result_t TM_PWM_SetChannelMicros(TM_PWM_TIM_t* TIM_Data, TM_PWM_Channel_t Channel, uint32_t micros) {
	/* If we choose too much micro seconds that we have valid */
	if (micros > TIM_Data->Micros) {
		/* Too high pulse */
		return TM_PWM_Result_PulseTooHigh;
	}

	/* Set PWM channel */
	return TM_PWM_SetChannel(TIM_Data, Channel, (uint32_t)((TIM_Data->Period - 1) * micros) / TIM_Data->Micros);
}
```

```
/* Private functions */
TM_PWM_Result_t TM_PWM_INT_InitTIM1Pins(TM_PWM_Channel_t Channel, TM_PWM_PinsPack_t PinsPack) {
	TM_PWM_Result_t result = TM_PWM_Result_PinNotValid;

	switch (Channel) {
		case TM_PWM_Channel_1:
			switch (PinsPack) {
				case TM_PWM_PinsPack_1:
#ifdef GPIOA
					TM_GPIO_InitAlternate(GPIOA,		GPIO_PIN_8,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM1);
					result = TM_PWM_Result_Ok;
#endif
					break;
				case TM_PWM_PinsPack_2:
#ifdef GPIOE
					TM_GPIO_InitAlternate(GPIOE,		GPIO_PIN_9,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM1);
					result = TM_PWM_Result_Ok;
#endif
					break;
				default:
					result = TM_PWM_Result_PinNotValid;
					break;
			}
			break;
		case TM_PWM_Channel_2:
			switch (PinsPack) {
				case TM_PWM_PinsPack_1:
#ifdef GPIOA
					TM_GPIO_InitAlternate(GPIOA,		GPIO_PIN_9,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM1);
					result = TM_PWM_Result_Ok;
#endif
					break;
				case TM_PWM_PinsPack_2:
#ifdef GPIOE
					TM_GPIO_InitAlternate(GPIOE,		GPIO_PIN_10,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM1);
					result = TM_PWM_Result_Ok;
#endif
					break;
				default:
					result = TM_PWM_Result_PinNotValid;
					break;
			}
			break;
		case TM_PWM_Channel_3:
			switch (PinsPack) {
				case TM_PWM_PinsPack_1:
#ifdef GPIOA
					TM_GPIO_InitAlternate(GPIOA,		GPIO_PIN_10,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM1);
					result = TM_PWM_Result_Ok;
#endif
					break;
				case TM_PWM_PinsPack_2:
#ifdef GPIOE
					TM_GPIO_InitAlternate(GPIOE,		GPIO_PIN_13,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM1);
					result = TM_PWM_Result_Ok;
#endif
					break;
				default:
					result = TM_PWM_Result_PinNotValid;
					break;
			}
			break;
		case TM_PWM_Channel_4:
			switch (PinsPack) {
				case TM_PWM_PinsPack_1:
#ifdef GPIOA
					TM_GPIO_InitAlternate(GPIOA,		GPIO_PIN_11,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM1);
					result = TM_PWM_Result_Ok;
```

```
#endif
                                        break;
                                case TM_PWM_PinsPack_2:
#ifdef GPIOE
                                        TM_GPIO_InitAlternate(GPIOE,                    GPIO_PIN_14,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM1);
                                        result = TM_PWM_Result_Ok;
#endif
                                        break;
                                default:
                                        result = TM_PWM_Result_PinNotValid;
                                        break;
                        }
                        break;
                default:
                        result = TM_PWM_Result_ChannelNotValid;
                        break;
        }

        return result;
}

TM_PWM_Result_t TM_PWM_INT_InitTIM2Pins(TM_PWM_Channel_t Channel, TM_PWM_PinsPack_t PinsPack) {
        TM_PWM_Result_t result = TM_PWM_Result_PinNotValid;

        switch (Channel) {
                case TM_PWM_Channel_1:
                        switch (PinsPack) {
                                case TM_PWM_PinsPack_1:
#ifdef GPIOA
                                        TM_GPIO_InitAlternate(GPIOA,                    GPIO_PIN_0,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM2);
                                        result = TM_PWM_Result_Ok;
#endif
                                        break;
                                case TM_PWM_PinsPack_2:
#ifdef GPIOA
                                        TM_GPIO_InitAlternate(GPIOA,                    GPIO_PIN_5,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM2);
                                        result = TM_PWM_Result_Ok;
#endif
                                        break;
                                case TM_PWM_PinsPack_3:
#ifdef GPIOA
                                        TM_GPIO_InitAlternate(GPIOA,                    GPIO_PIN_15,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM2);
                                        result = TM_PWM_Result_Ok;
#endif
                                        break;
                                default:
                                        result = TM_PWM_Result_PinNotValid;
                                        break;
                        }
                        break;
                case TM_PWM_Channel_2:
                        switch (PinsPack) {
                                case TM_PWM_PinsPack_1:
#ifdef GPIOA
                                        TM_GPIO_InitAlternate(GPIOA,                    GPIO_PIN_1,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM2);
                                        result = TM_PWM_Result_Ok;
#endif
                                        break;
                                case TM_PWM_PinsPack_2:
#ifdef GPIOB
                                        TM_GPIO_InitAlternate(GPIOB,                    GPIO_PIN_3,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM2);
                                        result = TM_PWM_Result_Ok;
#endif
                                        break;
                                default:
                                        result = TM_PWM_Result_PinNotValid;
                                        break;
                        }
                        break;
                case TM_PWM_Channel_3:
```

```
                        switch (PinsPack) {
                                case TM_PWM_PinsPack_1:
#ifdef GPIOA
                                        TM_GPIO_InitAlternate(GPIOA,                    GPIO_PIN_2,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM2);
                                        result = TM_PWM_Result_Ok;
#endif
                                        break;
                                case TM_PWM_PinsPack_2:
#ifdef GPIOB
                                        TM_GPIO_InitAlternate(GPIOB,                    GPIO_PIN_10,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM2);
                                        result = TM_PWM_Result_Ok;
#endif
                                        break;
                                default:
                                        result = TM_PWM_Result_PinNotValid;
                                        break;
                        }
                        break;
                case TM_PWM_Channel_4:
                        switch (PinsPack) {
                                case TM_PWM_PinsPack_1:
#ifdef GPIOA
                                        TM_GPIO_InitAlternate(GPIOA,                    GPIO_PIN_3,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM2);
                                        result = TM_PWM_Result_Ok;
#endif
                                        break;
                                case TM_PWM_PinsPack_2:
#ifdef GPIOB
                                        TM_GPIO_InitAlternate(GPIOB,                    GPIO_PIN_11,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM2);
                                        result = TM_PWM_Result_Ok;
#endif
                                        break;
                                default:
                                        result = TM_PWM_Result_PinNotValid;
                                        break;
                        }
                        break;
                default:
                        result = TM_PWM_Result_ChannelNotValid;
                        break;
        }

        return result;
}

TM_PWM_Result_t TM_PWM_INT_InitTIM3Pins(TM_PWM_Channel_t Channel, TM_PWM_PinsPack_t PinsPack) {
        TM_PWM_Result_t result = TM_PWM_Result_PinNotValid;

        switch (Channel) {
                case TM_PWM_Channel_1:
                        switch (PinsPack) {
                                case TM_PWM_PinsPack_1:
#ifdef GPIOA
                                        TM_GPIO_InitAlternate(GPIOA,                    GPIO_PIN_6,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM3);
                                        result = TM_PWM_Result_Ok;
#endif
                                        break;
                                case TM_PWM_PinsPack_2:
#ifdef GPIOB
                                        TM_GPIO_InitAlternate(GPIOB,                    GPIO_PIN_4,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM3);
                                        result = TM_PWM_Result_Ok;
#endif
                                        break;
                                case TM_PWM_PinsPack_3:
#ifdef GPIOC
                                        TM_GPIO_InitAlternate(GPIOC,                    GPIO_PIN_10,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM3);
                                        result = TM_PWM_Result_Ok;
#endif
                                        break;
```

```
                        default:
                                result = TM_PWM_Result_PinNotValid;
                                break;
                }
                break;
        case TM_PWM_Channel_2:
                switch (PinsPack) {
                        case TM_PWM_PinsPack_1:
#ifdef GPIOA
                                TM_GPIO_InitAlternate(GPIOA,                    GPIO_PIN_7,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM3);
                                result = TM_PWM_Result_Ok;
#endif
                                break;
                        case TM_PWM_PinsPack_2:
#ifdef GPIOB
                                TM_GPIO_InitAlternate(GPIOB,                    GPIO_PIN_5,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM3);
                                result = TM_PWM_Result_Ok;
#endif
                                break;
                        case TM_PWM_PinsPack_3:
#ifdef GPIOC
                                TM_GPIO_InitAlternate(GPIOC,                    GPIO_PIN_7,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM3);
                                result = TM_PWM_Result_Ok;
#endif
                                break;
                        default:
                                result = TM_PWM_Result_PinNotValid;
                                break;
                }
                break;
        case TM_PWM_Channel_3:
                switch (PinsPack) {
                        case TM_PWM_PinsPack_1:
#ifdef GPIOB
                                TM_GPIO_InitAlternate(GPIOB,                    GPIO_PIN_0,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM3);
                                result = TM_PWM_Result_Ok;
#endif
                                break;
                        case TM_PWM_PinsPack_2:
#ifdef GPIOC
                                TM_GPIO_InitAlternate(GPIOC,                    GPIO_PIN_8,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM3);
                                result = TM_PWM_Result_Ok;
#endif
                                break;
                        default:
                                result = TM_PWM_Result_PinNotValid;
                                break;
                }
                break;
        case TM_PWM_Channel_4:
                switch (PinsPack) {
                        case TM_PWM_PinsPack_1:
#ifdef GPIOB
                                TM_GPIO_InitAlternate(GPIOB,                    GPIO_PIN_1,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM3);
                                result = TM_PWM_Result_Ok;
#endif
                                break;
                        case TM_PWM_PinsPack_2:
#ifdef GPIOC
                                TM_GPIO_InitAlternate(GPIOC,                    GPIO_PIN_9,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM3);
                                result = TM_PWM_Result_Ok;
#endif
                                break;
                        default:
                                result = TM_PWM_Result_PinNotValid;
                                break;
                }
                break;
        default:
```

```
                result = TM_PWM_Result_ChannelNotValid;
                break;
    }

    return result;
}

TM_PWM_Result_t TM_PWM_INT_InitTIM4Pins(TM_PWM_Channel_t Channel, TM_PWM_PinsPack_t PinsPack) {
    TM_PWM_Result_t result = TM_PWM_Result_PinNotValid;

    switch (Channel) {
        case TM_PWM_Channel_1:
            switch (PinsPack) {
                case TM_PWM_PinsPack_1:
#ifdef GPIOB
                    TM_GPIO_InitAlternate(GPIOB, GPIO_PIN_6,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM4);
                    result = TM_PWM_Result_Ok;
#endif
                    break;
                case TM_PWM_PinsPack_2:
#ifdef GPIOD
                    TM_GPIO_InitAlternate(GPIOD, GPIO_PIN_12,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM4);
                    result = TM_PWM_Result_Ok;
#endif
                    break;
                default:
                    result = TM_PWM_Result_PinNotValid;
                    break;
            }
            break;
        case TM_PWM_Channel_2:
            switch (PinsPack) {
                case TM_PWM_PinsPack_1:
#ifdef GPIOB
                    TM_GPIO_InitAlternate(GPIOB, GPIO_PIN_7,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM4);
                    result = TM_PWM_Result_Ok;
#endif
                    break;
                case TM_PWM_PinsPack_2:
#ifdef GPIOD
                    TM_GPIO_InitAlternate(GPIOD, GPIO_PIN_13,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM4);
                    result = TM_PWM_Result_Ok;
#endif
                    break;
                default:
                    result = TM_PWM_Result_PinNotValid;
                    break;
            }
            break;
        case TM_PWM_Channel_3:
            switch (PinsPack) {
                case TM_PWM_PinsPack_1:
#ifdef GPIOB
                    TM_GPIO_InitAlternate(GPIOB, GPIO_PIN_8,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM4);
                    result = TM_PWM_Result_Ok;
#endif
                    break;
                case TM_PWM_PinsPack_2:
#ifdef GPIOD
                    TM_GPIO_InitAlternate(GPIOD, GPIO_PIN_14,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM4);
                    result = TM_PWM_Result_Ok;
#endif
                    break;
                default:
                    result = TM_PWM_Result_PinNotValid;
                    break;
            }
            break;
        case TM_PWM_Channel_4:
            switch (PinsPack) {
```

```
                                        case TM_PWM_PinsPack_1:
#ifdef GPIOB
                                                TM_GPIO_InitAlternate(GPIOB,                    GPIO_PIN_9,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM4);
                                                result = TM_PWM_Result_Ok;
#endif
                                                break;
                                        case TM_PWM_PinsPack_2:
#ifdef GPIOD
                                                TM_GPIO_InitAlternate(GPIOD,                    GPIO_PIN_15,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM4);
                                                result = TM_PWM_Result_Ok;
#endif
                                                break;
                                        default:
                                                result = TM_PWM_Result_PinNotValid;
                                                break;
                                }
                                break;
                        default:
                                result = TM_PWM_Result_ChannelNotValid;
                                break;
        }

        return result;
}

TM_PWM_Result_t TM_PWM_INT_InitTIM5Pins(TM_PWM_Channel_t Channel, TM_PWM_PinsPack_t PinsPack) {
        TM_PWM_Result_t result = TM_PWM_Result_PinNotValid;

        switch (Channel) {
                case TM_PWM_Channel_1:
                        switch (PinsPack) {
                                        case TM_PWM_PinsPack_1:
#ifdef GPIOA
                                                TM_GPIO_InitAlternate(GPIOA,                    GPIO_PIN_0,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM5);
                                                result = TM_PWM_Result_Ok;
#endif
                                                break;
                                        case TM_PWM_PinsPack_2:
#ifdef GPIOH
                                                TM_GPIO_InitAlternate(GPIOH,                    GPIO_PIN_10,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM5);
                                                result = TM_PWM_Result_Ok;
#endif
                                                break;
                                        default:
                                                result = TM_PWM_Result_PinNotValid;
                                                break;
                                }
                                break;
                case TM_PWM_Channel_2:
                        switch (PinsPack) {
                                        case TM_PWM_PinsPack_1:
#ifdef GPIOA
                                                TM_GPIO_InitAlternate(GPIOA,                    GPIO_PIN_1,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM5);
                                                result = TM_PWM_Result_Ok;
#endif
                                                break;
                                        case TM_PWM_PinsPack_2:
#ifdef GPIOH
                                                TM_GPIO_InitAlternate(GPIOH,                    GPIO_PIN_11,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM5);
                                                result = TM_PWM_Result_Ok;
#endif
                                                break;
                                        default:
                                                result = TM_PWM_Result_PinNotValid;
                                                break;
                                }
                                break;
                case TM_PWM_Channel_3:
                        switch (PinsPack) {
                                        case TM_PWM_PinsPack_1:
```

```
#ifdef GPIOA
                                        TM_GPIO_InitAlternate(GPIOA,                    GPIO_PIN_2,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM5);
                                        result = TM_PWM_Result_Ok;
#endif
                                        break;
                                case TM_PWM_PinsPack_2:
#ifdef GPIOH
                                        TM_GPIO_InitAlternate(GPIOH,                    GPIO_PIN_12,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM5);
                                        result = TM_PWM_Result_Ok;
#endif
                                        break;
                                default:
                                        result = TM_PWM_Result_PinNotValid;
                                        break;
                        }
                        break;
                case TM_PWM_Channel_4:
                        switch (PinsPack) {
                                case TM_PWM_PinsPack_1:
#ifdef GPIOA
                                        TM_GPIO_InitAlternate(GPIOA,                    GPIO_PIN_3,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM5);
                                        result = TM_PWM_Result_Ok;
#endif
                                        break;
                                case TM_PWM_PinsPack_2:
#ifdef GPIOI
                                        TM_GPIO_InitAlternate(GPIOI, GPIO_PIN_0, TM_GPIO_OType_PP,
TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM5);
                                        result = TM_PWM_Result_Ok;
#endif
                                        break;
                                default:
                                        result = TM_PWM_Result_PinNotValid;
                                        break;
                        }
                        break;
                default:
                        result = TM_PWM_Result_ChannelNotValid;
                        break;
        }

        return result;
}

TM_PWM_Result_t TM_PWM_INT_InitTIM8Pins(TM_PWM_Channel_t Channel, TM_PWM_PinsPack_t PinsPack) {
        TM_PWM_Result_t result = TM_PWM_Result_PinNotValid;

        switch (Channel) {
                case TM_PWM_Channel_1:
                        switch (PinsPack) {
                                case TM_PWM_PinsPack_1:
#ifdef GPIOC
                                        TM_GPIO_InitAlternate(GPIOC,                    GPIO_PIN_6,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM8);
                                        result = TM_PWM_Result_Ok;
#endif
                                        break;
                                case TM_PWM_PinsPack_2:
#ifdef GPIOI
                                        TM_GPIO_InitAlternate(GPIOI, GPIO_PIN_5, TM_GPIO_OType_PP,
TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM8);
                                        result = TM_PWM_Result_Ok;
#endif
                                        break;
                                default:
                                        result = TM_PWM_Result_PinNotValid;
                                        break;
                        }
                        break;
                case TM_PWM_Channel_2:
                        switch (PinsPack) {
                                case TM_PWM_PinsPack_1:
#ifdef GPIOC
```

```
                                        TM_GPIO_InitAlternate(GPIOC,                    GPIO_PIN_7,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM8);
                                        result = TM_PWM_Result_Ok;
#endif
                                        break;
                                case TM_PWM_PinsPack_2:
#ifdef GPIOI
                                        TM_GPIO_InitAlternate(GPIOI, GPIO_PIN_6, TM_GPIO_OType_PP,
TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM8);
                                        result = TM_PWM_Result_Ok;
#endif
                                        break;
                                default:
                                        result = TM_PWM_Result_PinNotValid;
                                        break;
                        }
                        break;
                case TM_PWM_Channel_3:
                        switch (PinsPack) {
                                case TM_PWM_PinsPack_1:
#ifdef GPIOC
                                        TM_GPIO_InitAlternate(GPIOC,                    GPIO_PIN_8,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM8);
                                        result = TM_PWM_Result_Ok;
#endif
                                        break;
                                case TM_PWM_PinsPack_2:
#ifdef GPIOI
                                        TM_GPIO_InitAlternate(GPIOI, GPIO_PIN_7, TM_GPIO_OType_PP,
TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM8);
                                        result = TM_PWM_Result_Ok;
#endif
                                        break;
                                default:
                                        result = TM_PWM_Result_PinNotValid;
                                        break;
                        }
                        break;
                case TM_PWM_Channel_4:
                        switch (PinsPack) {
                                case TM_PWM_PinsPack_1:
#ifdef GPIOC
                                        TM_GPIO_InitAlternate(GPIOC,                    GPIO_PIN_9,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM8);
                                        result = TM_PWM_Result_Ok;
#endif
                                        break;
                                case TM_PWM_PinsPack_2:
#ifdef GPIOI
                                        TM_GPIO_InitAlternate(GPIOI, GPIO_PIN_8, TM_GPIO_OType_PP,
TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM8);
                                        result = TM_PWM_Result_Ok;
#endif
                                        break;
                                default:
                                        result = TM_PWM_Result_PinNotValid;
                                        break;
                        }
                        break;
                default:
                        result = TM_PWM_Result_ChannelNotValid;
                        break;
        }

        return result;
}

TM_PWM_Result_t TM_PWM_INT_InitTIM9Pins(TM_PWM_Channel_t Channel, TM_PWM_PinsPack_t PinsPack) {
        TM_PWM_Result_t result = TM_PWM_Result_PinNotValid;

        switch (Channel) {
                case TM_PWM_Channel_1:
                        switch (PinsPack) {
                                case TM_PWM_PinsPack_1:
#ifdef GPIOA
```

```
                                        TM_GPIO_InitAlternate(GPIOA,                    GPIO_PIN_2,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM9);
                                        result = TM_PWM_Result_Ok;
#endif
                                        break;
                                case TM_PWM_PinsPack_2:
#ifdef GPIOE
                                        TM_GPIO_InitAlternate(GPIOE,                    GPIO_PIN_5,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM9);
                                        result = TM_PWM_Result_Ok;
#endif
                                        break;
                                default:
                                        result = TM_PWM_Result_PinNotValid;
                                        break;
                        }
                        break;
                case TM_PWM_Channel_2:
                        switch (PinsPack) {
                                case TM_PWM_PinsPack_1:
#ifdef GPIOA
                                        TM_GPIO_InitAlternate(GPIOA,                    GPIO_PIN_3,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM9);
                                        result = TM_PWM_Result_Ok;
#endif
                                        break;
                                case TM_PWM_PinsPack_2:
#ifdef GPIOE
                                        TM_GPIO_InitAlternate(GPIOE,                    GPIO_PIN_6,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM9);
                                        result = TM_PWM_Result_Ok;
#endif
                                        break;
                                default:
                                        result = TM_PWM_Result_PinNotValid;
                                        break;
                        }
                        break;
                default:
                        result = TM_PWM_Result_ChannelNotValid;
                        break;
        }

        return result;
}

TM_PWM_Result_t TM_PWM_INT_InitTIM10Pins(TM_PWM_Channel_t Channel, TM_PWM_PinsPack_t PinsPack) {
        TM_PWM_Result_t result = TM_PWM_Result_PinNotValid;

        switch (Channel) {
                case TM_PWM_Channel_1:
                        switch (PinsPack) {
                                case TM_PWM_PinsPack_1:
#ifdef GPIOB
                                        TM_GPIO_InitAlternate(GPIOB,                    GPIO_PIN_8,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM10);
                                        result = TM_PWM_Result_Ok;
#endif
                                        break;
                                case TM_PWM_PinsPack_2:
#ifdef GPIOF
                                        TM_GPIO_InitAlternate(GPIOF, GPIO_PIN_6, TM_GPIO_OType_PP,
TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM10);
                                        result = TM_PWM_Result_Ok;
#endif
                                        break;
                                default:
                                        result = TM_PWM_Result_PinNotValid;
                                        break;
                        }
                        break;
                default:
                        result = TM_PWM_Result_ChannelNotValid;
                        break;
        }
```

```
	return result;
}

TM_PWM_Result_t TM_PWM_INT_InitTIM11Pins(TM_PWM_Channel_t Channel, TM_PWM_PinsPack_t PinsPack) {
	TM_PWM_Result_t result = TM_PWM_Result_PinNotValid;

	switch (Channel) {
		case TM_PWM_Channel_1:
			switch (PinsPack) {
				case TM_PWM_PinsPack_1:
#ifdef GPIOB
					TM_GPIO_InitAlternate(GPIOB,		GPIO_PIN_9,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM11);
					result = TM_PWM_Result_Ok;
#endif
					break;
				case TM_PWM_PinsPack_2:
#ifdef GPIOF
					TM_GPIO_InitAlternate(GPIOF, GPIO_PIN_7, TM_GPIO_OType_PP,
TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM11);
					result = TM_PWM_Result_Ok;
#endif
					break;
				default:
					result = TM_PWM_Result_PinNotValid;
					break;
			}
			break;
		default:
			result = TM_PWM_Result_ChannelNotValid;
			break;
	}

	return result;
}

TM_PWM_Result_t TM_PWM_INT_InitTIM12Pins(TM_PWM_Channel_t Channel, TM_PWM_PinsPack_t PinsPack) {
	TM_PWM_Result_t result = TM_PWM_Result_PinNotValid;

	switch (Channel) {
		case TM_PWM_Channel_1:
			switch (PinsPack) {
				case TM_PWM_PinsPack_1:
#ifdef GPIOB
					TM_GPIO_InitAlternate(GPIOB,		GPIO_PIN_14,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM12);
					result = TM_PWM_Result_Ok;
#endif
					break;
				case TM_PWM_PinsPack_2:
#ifdef GPIOH
					TM_GPIO_InitAlternate(GPIOH,		GPIO_PIN_6,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM12);
					result = TM_PWM_Result_Ok;
#endif
					break;
				default:
					result = TM_PWM_Result_PinNotValid;
					break;
			}
			break;
		case TM_PWM_Channel_2:
			switch (PinsPack) {
				case TM_PWM_PinsPack_1:
#ifdef GPIOB
					TM_GPIO_InitAlternate(GPIOB,		GPIO_PIN_15,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM12);
					result = TM_PWM_Result_Ok;
#endif
					break;
				case TM_PWM_PinsPack_2:
#ifdef GPIOH
					TM_GPIO_InitAlternate(GPIOH,		GPIO_PIN_9,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM12);
					result = TM_PWM_Result_Ok;
#endif
```

```
                                        break;
                                default:
                                        result = TM_PWM_Result_PinNotValid;
                                        break;
                                }
                                break;
                        default:
                                result = TM_PWM_Result_ChannelNotValid;
                                break;
                }

                return result;
}

TM_PWM_Result_t TM_PWM_INT_InitTIM13Pins(TM_PWM_Channel_t Channel, TM_PWM_PinsPack_t PinsPack) {
                TM_PWM_Result_t result = TM_PWM_Result_PinNotValid;

                switch (Channel) {
                        case TM_PWM_Channel_1:
                                switch (PinsPack) {
                                        case TM_PWM_PinsPack_1:
#ifdef GPIOA
                                                TM_GPIO_InitAlternate(GPIOA,                    GPIO_PIN_1,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM13);
                                                result = TM_PWM_Result_Ok;
#endif
                                                break;
                                        case TM_PWM_PinsPack_2:
#ifdef GPIOF
                                                TM_GPIO_InitAlternate(GPIOF, GPIO_PIN_8, TM_GPIO_OType_PP,
TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM13);
                                                result = TM_PWM_Result_Ok;
#endif
                                                break;
                                        default:
                                                result = TM_PWM_Result_PinNotValid;
                                                break;
                                }
                                break;
                        default:
                                result = TM_PWM_Result_ChannelNotValid;
                                break;
                }

                return result;
}

TM_PWM_Result_t TM_PWM_INT_InitTIM14Pins(TM_PWM_Channel_t Channel, TM_PWM_PinsPack_t PinsPack) {
                TM_PWM_Result_t result = TM_PWM_Result_PinNotValid;

                switch (Channel) {
                        case TM_PWM_Channel_1:
                                switch (PinsPack) {
                                        case TM_PWM_PinsPack_1:
#ifdef GPIOA
                                                TM_GPIO_InitAlternate(GPIOA,                    GPIO_PIN_7,
TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM14);
                                                result = TM_PWM_Result_Ok;
#endif
                                                break;
                                        case TM_PWM_PinsPack_2:
#ifdef GPIOF
                                                TM_GPIO_InitAlternate(GPIOF, GPIO_PIN_9, TM_GPIO_OType_PP,
TM_GPIO_PuPd_NOPULL, TM_GPIO_Speed_High, GPIO_AF_TIM14);
                                                result = TM_PWM_Result_Ok;
#endif
                                                break;
                                        default:
                                                result = TM_PWM_Result_PinNotValid;
                                                break;
                                }
                                break;
                        default:
                                result = TM_PWM_Result_ChannelNotValid;
                                break;
                }
```

```
        return result;
}
```

tm_stm32f4_servo.h

```
/**
 * @author  Tilen Majerle
 * @email   tilen@majerle.eu
 * @website http://stm32f4-discovery.com
 * @link    http://stm32f4-discovery.com/2014/10/library-42-control-rc-servo-stm32f4
 * @version v1.1
 * @ide     Keil uVision
 * @license GNU GPL v3
 * @brief   Servo library for STM32F4xx
 *
@verbatim
   ----------------------------------------------------------------------
    Copyright (C) Tilen Majerle, 2015

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
   ----------------------------------------------------------------------
@endverbatim
 */
#ifndef TM_SERVO_H
#define TM_SERVO_H 110

/* C++ detection */
#ifdef __cplusplus
extern "C" {
#endif

/**
 * @addtogroup TM_STM32F4xx_Libraries
 * @{
 */

/**
 * @defgroup TM_SERVO
 * @brief     SERVO library for STM32F4xx devices - http://stm32f4-discovery.com/2014/10/library-42-control-rc-servo-
stm32f4
 * @{
 *
 * This library allows you to control RC servo with STM32F4xx.
 *
 * Basically, only limitation for number of servos is number of timers in MCU.
 *
 * \par Calculations
 *
 * Equation for pulse length to get specific rotation is:
\code
Pulse length (degrees) = (MAX – MIN) * degrees / 180 + MIN
\endcode
 *
 * where:
 *  - MAX: maximum pulse length for servo, 2000us
 *  - MIN: minimum pulse length for servo, 1000us
 *
 * If you want to get rotation in degrees from know pulse length:
 *
\code
```

```
Degrees (pulse_length) = (pulse_length - MIN) * 180 / (MAX - MIN)
\endcode
 *
 *
 * \par Changelog
 *
@verbatim
 Version 1.1
  - January 03, 2015
  - Added "micros" and "degrees" values to structure for user feedback where servo is located
  - Supports TM PWM 2.0 version library

 Version 1.0
  - First release
@endverbatim
 *
 * \par Dependencies
 *
@verbatim
 - STM32F4xx
 - STM32F4xx RCC
 - STM32F4xx GPIO
 - STM32F4xx TIM
 - defines.h
 - TM PWM
 - TM TIMER PROPERTIES
@endverbatim
 */

#include "stm32f4xx.h"
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_tim.h"
#include "defines.h"
#include "tm_stm32f4_pwm.h"
#include "tm_stm32f4_timer_properties.h"

/**
 * @defgroup TM_SERVO_Macros
 * @brief    Library defines
 * @{
 */

/**
 * @brief  Minimal pulse length for servo in micros
 */
#ifndef SERVO_MICROS_MIN
#define SERVO_MICROS_MIN            500
#endif

/**
 * @brief  Maximal pulse length for servo in micros
 */
#ifndef SERVO_MICROS_MAX
#define SERVO_MICROS_MAX            2300
#endif

/**
 * @}
 */

/**
 * @defgroup TM_SERVO_Typedefs
 * @brief    Library Typedefs
 * @{
 */

/**
 * @brief  Servo structure
 */
typedef struct {
        TM_PWM_TIM_t PWM;           /*!< PWM settings */
        TIM_TypeDef* TIM;           /*!< Pointer to specific timer you will use for servo */
        TM_PWM_Channel_t Channel;   /*!< Output channel on specific timer */
        TM_PWM_PinsPack_t Pinspack; /*!< Pinspack for specific channel */
        float Degrees;              /*!< Position of servo in degrees */
```

```
	uint16_t Micros;		/*!< Pulse length in micro seconds for current servo position */
} TM_SERVO_t;

/**
 * @brief Results enumeration
 */
typedef enum {
	TM_SERVO_Result_Ok = 0, /*!< Everything OK */
	TM_SERVO_Result_Error	/*!< An error occurred somewhere */
} TM_SERVO_Result_t;

/**
 * @}
 */

/**
 * @defgroup TM_SERVO_Functions
 * @brief    Library Functions
 * @{
 */

/**
 * @brief  Initializes TIM and PWM for servo motor purpose
 * @param  *ServoStruct: Pointer to an empty @ref TM_SERVO_t servo structure
 * @param  *TIMx: Pointer to TIMx you will use for servo
 * @param  PWMChannel: Channel you will use for timer. This parameter can be a value of @ref TM_PWM_Channel_t
enumeration
 * @param  Pinspack: Pinspack for channel. This parameter can be a value of @ref TM_PWM_PinsPack_t enumeration
 * @retval Member of TM_SERVO_Result_t
 */
TM_SERVO_Result_t	TM_SERVO_Init(TM_SERVO_t*	ServoStruct,	TIM_TypeDef*	TIMx,	TM_PWM_Channel_t
PWMChannel, TM_PWM_PinsPack_t Pinspack);

/**
 * @brief  Set rotation degrees for servo
 * @note   Degrees can be between 0 and 180
 * @param  *ServoStruct: Pointer to an @ref TM_SERVO_t servo structure
 * @param  degrees: Rotation in degrees, between 0 and 180
 * @retval Member of TM_SERVO_Result_t
 */
TM_SERVO_Result_t TM_SERVO_SetDegrees(TM_SERVO_t* ServoStruct, float degrees);

/**
 * @brief  Sets pulse length in microseconds
 * @param  *ServoStruct: Pointer to an @ref TM_SERVO_t servo structure
 * @param  micros: pulse length in microseconds
 * @retval Member of TM_SERVO_Result_t
 */
TM_SERVO_Result_t TM_SERVO_SetMicros(TM_SERVO_t* ServoStruct, uint16_t micros);

/**
 * @}
 */

/**
 * @}
 */

/**
 * @}
 */

/* C++ detection */
#ifdef __cplusplus
}
#endif

#endif
```

tm_stm32f4_servo.c

```
/**
```

```
 * |----------------------------------------------------------------------
 * | Copyright (C) Tilen Majerle, 2014
 * |
 * | This program is free software: you can redistribute it and/or modify
 * | it under the terms of the GNU General Public License as published by
 * | the Free Software Foundation, either version 3 of the License, or
 * | any later version.
 * |
 * | This program is distributed in the hope that it will be useful,
 * | but WITHOUT ANY WARRANTY; without even the implied warranty of
 * | MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * | GNU General Public License for more details.
 * |
 * | You should have received a copy of the GNU General Public License
 * | along with this program.  If not, see <http://www.gnu.org/licenses/>.
 * |----------------------------------------------------------------------
 */
#include "tm_stm32f4_servo.h"

TM_SERVO_Result_t TM_SERVO_Init(TM_SERVO_t* ServoStruct, TIM_TypeDef* TIMx, TM_PWM_Channel_t
PWMChannel, TM_PWM_PinsPack_t Pinspack) {
	/* Initialize timer with 50Hz frequency for PWM */
	if (TM_PWM_InitTimer(TIMx, &ServoStruct->PWM, 50) != TM_PWM_Result_Ok) {
		/* Return error */
		return TM_SERVO_Result_Error;
	}

	/* Initialize channel */
	if (TM_PWM_InitChannel(&ServoStruct->PWM, PWMChannel, Pinspack) != TM_PWM_Result_Ok) {
		/* Return Error */
		return TM_SERVO_Result_Error;
	}

	/* Fill settings */
	ServoStruct->TIM = TIMx;
	ServoStruct->Channel = PWMChannel;
	ServoStruct->Pinspack = Pinspack;

	/* Return OK */
	return TM_SERVO_Result_Ok;
}

TM_SERVO_Result_t TM_SERVO_SetDegrees(TM_SERVO_t* ServoStruct, float degrees) {
	/* Set PWM value */
	uint16_t micros;

	/* Filter */
	//if (degrees < 0 || degrees > 180) {
		/* Return error */
	//	return TM_SERVO_Result_Error;
	//}

	/* Generate micros value from degrees */
	float intermedio = (float)(SERVO_MICROS_MAX - SERVO_MICROS_MIN) * (float)(degrees / 180) +
SERVO_MICROS_MIN;
	micros = (uint16_t) intermedio;
	/* Save micros and degrees values to struct */
	ServoStruct->Micros = micros;
	ServoStruct->Degrees = degrees;

	/* Set micros */
	TM_PWM_SetChannelMicros(
		&ServoStruct->PWM,
		ServoStruct->Channel,
		micros
	);

	/* Return OK */
	return TM_SERVO_Result_Ok;
}

TM_SERVO_Result_t TM_SERVO_SetMicros(TM_SERVO_t* ServoStruct, uint16_t micros) {
	/* Set PWM value */
	float degrees;

	/* Filter */
```

```
	if (micros < SERVO_MICROS_MIN || micros > SERVO_MICROS_MAX) {
		/* Return error */
		return TM_SERVO_Result_Error;
	}

	/* Generate micros value from degrees */
	degrees = (float)(micros - SERVO_MICROS_MIN) * (float)180 / (float)(SERVO_MICROS_MAX -
SERVO_MICROS_MIN);

	/* Save micros and degrees values to struct */
	ServoStruct->Micros = micros;
	ServoStruct->Degrees = degrees;

	/* Set micros */
	TM_PWM_SetChannelMicros(
		&ServoStruct->PWM,
		ServoStruct->Channel,
		micros
	);

	/* Return OK */
	return TM_SERVO_Result_Ok;
}
```

Librerías Proporcionadas por el IDE

startup_stm32f40xx.s

```
/**
  ******************************************************************************
  * @file      startup_stm32f40xx.s
  * @author    MCD Application Team
  * @version   V1.1.0
  * @date      11-January-2013
  * @brief     STM32F40xx/41xx Devices vector table for Atollic TrueSTUDIO toolchain.
  *            This module performs:
  *                - Set the initial SP
  *                - Set the initial PC == Reset_Handler,
  *                - Set the vector table entries with the exceptions ISR address
  *                - Configure the clock system and the external SRAM mounted on
  *                  STM324xG-EVAL board to be used as data memory (optional,
  *                  to be enabled by user)
  *                - Branches to main in the C library (which eventually
  *                  calls main()).
  *            After Reset the Cortex-M4 processor is in Thread mode,
  *            priority is Privileged, and the Stack is set to Main.
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; COPYRIGHT 2013 STMicroelectronics</center></h2>
  *
  * Licensed under MCD-ST Liberty SW License Agreement V2, (the "License");
  * You may not use this file except in compliance with the License.
  * You may obtain a copy of the License at:
  *
  *        http://www.st.com/software_license_agreement_liberty_v2
  *
  * Unless required by applicable law or agreed to in writing, software
  * distributed under the License is distributed on an "AS IS" BASIS,
  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  * See the License for the specific language governing permissions and
  * limitations under the License.
  *
  ******************************************************************************
  */

  .syntax unified
  .cpu cortex-m3
  .fpu softvfp
  .thumb

.global  g_pfnVectors
.global  Default_Handler

/* start address for the initialization values of the .data section.
defined in linker script */
.word  _sidata
/* start address for the .data section. defined in linker script */
.word  _sdata
/* end address for the .data section. defined in linker script */
.word  _edata
/* start address for the .bss section. defined in linker script */
.word  _sbss
/* end address for the .bss section. defined in linker script */
.word  _ebss
/* stack used for SystemInit_ExtMemCtl; always internal RAM used */

/**
 * @brief  This is the code that gets called when the processor first
 *         starts execution following a reset event. Only the absolutely
 *         necessary set is performed, after which the application
 *         supplied main() routine is called.
 * @param  None
 * @retval : None
*/
```

```
    .section  .text.Reset_Handler
  .weak  Reset_Handler
  .type  Reset_Handler, %function
Reset_Handler:
  ldr   sp, =_estack    /* Atollic update: set stack pointer */

/* Copy the data segment initializers from flash to SRAM */
  movs  r1, #0
  b  LoopCopyDataInit

CopyDataInit:
  ldr  r3, =_sidata
  ldr  r3, [r3, r1]
  str  r3, [r0, r1]
  adds  r1, r1, #4

LoopCopyDataInit:
  ldr  r0, =_sdata
  ldr  r3, =_edata
  adds  r2, r0, r1
  cmp  r2, r3
  bcc  CopyDataInit
  ldr  r2, =_sbss
  b  LoopFillZerobss
/* Zero fill the bss segment. */
FillZerobss:
  movs  r3, #0
  str  r3, [r2], #4

LoopFillZerobss:
  ldr  r3, = _ebss
  cmp  r2, r3
  bcc  FillZerobss

/* Call the clock system intitialization function.*/
  bl  SystemInit
/* Call static constructors */
    bl __libc_init_array
/* Call the application's entry point.*/
  bl  main
  bx  lr
.size  Reset_Handler, .-Reset_Handler

/**
 * @brief  This is the code that gets called when the processor receives an
 *         unexpected interrupt.  This simply enters an infinite loop, preserving
 *         the system state for examination by a debugger.
 * @param  None
 * @retval None
*/
    .section  .text.Default_Handler,"ax",%progbits
Default_Handler:
Infinite_Loop:
  b  Infinite_Loop
  .size  Default_Handler, .-Default_Handler
/*******************************************************************************
*
* The minimal vector table for a Cortex M3. Note that the proper constructs
* must be placed on this to ensure that it ends up at physical address
* 0x0000.0000.
*
*******************************************************************************/
    .section  .isr_vector,"a",%progbits
  .type  g_pfnVectors, %object
  .size  g_pfnVectors, .-g_pfnVectors


g_pfnVectors:
  .word  _estack
  .word  Reset_Handler
  .word  NMI_Handler
  .word  HardFault_Handler
  .word  MemManage_Handler
  .word  BusFault_Handler
  .word  UsageFault_Handler
  .word  0
```

```
.word  0
.word  0
.word  0
.word  SVC_Handler
.word  DebugMon_Handler
.word  0
.word  PendSV_Handler
.word  SysTick_Handler

/* External Interrupts */
.word     WWDG_IRQHandler                  /* Window WatchDog              */
.word     PVD_IRQHandler                   /* PVD through EXTI Line detection */
.word     TAMP_STAMP_IRQHandler            /* Tamper and TimeStamps through the EXTI line */
.word     RTC_WKUP_IRQHandler              /* RTC Wakeup through the EXTI line */
.word     FLASH_IRQHandler                 /* FLASH                        */
.word     RCC_IRQHandler                   /* RCC                          */
.word     EXTI0_IRQHandler                 /* EXTI Line0                   */
.word     EXTI1_IRQHandler                 /* EXTI Line1                   */
.word     EXTI2_IRQHandler                 /* EXTI Line2                   */
.word     EXTI3_IRQHandler                 /* EXTI Line3                   */
.word     EXTI4_IRQHandler                 /* EXTI Line4                   */
.word     DMA1_Stream0_IRQHandler          /* DMA1 Stream 0                */
.word     DMA1_Stream1_IRQHandler          /* DMA1 Stream 1                */
.word     DMA1_Stream2_IRQHandler          /* DMA1 Stream 2                */
.word     DMA1_Stream3_IRQHandler          /* DMA1 Stream 3                */
.word     DMA1_Stream4_IRQHandler          /* DMA1 Stream 4                */
.word     DMA1_Stream5_IRQHandler          /* DMA1 Stream 5                */
.word     DMA1_Stream6_IRQHandler          /* DMA1 Stream 6                */
.word     ADC_IRQHandler                   /* ADC1, ADC2 and ADC3s         */
.word     CAN1_TX_IRQHandler               /* CAN1 TX                      */
.word     CAN1_RX0_IRQHandler              /* CAN1 RX0                     */
.word     CAN1_RX1_IRQHandler              /* CAN1 RX1                     */
.word     CAN1_SCE_IRQHandler              /* CAN1 SCE                     */
.word     EXTI9_5_IRQHandler               /* External Line[9:5]s          */
.word     TIM1_BRK_TIM9_IRQHandler         /* TIM1 Break and TIM9          */
.word     TIM1_UP_TIM10_IRQHandler         /* TIM1 Update and TIM10        */
.word     TIM1_TRG_COM_TIM11_IRQHandler    /* TIM1 Trigger and Commutation and TIM11 */
.word     TIM1_CC_IRQHandler               /* TIM1 Capture Compare         */
.word     TIM2_IRQHandler                  /* TIM2                         */
.word     TIM3_IRQHandler                  /* TIM3                         */
.word     TIM4_IRQHandler                  /* TIM4                         */
.word     I2C1_EV_IRQHandler               /* I2C1 Event                   */
.word     I2C1_ER_IRQHandler               /* I2C1 Error                   */
.word     I2C2_EV_IRQHandler               /* I2C2 Event                   */
.word     I2C2_ER_IRQHandler               /* I2C2 Error                   */
.word     SPI1_IRQHandler                  /* SPI1                         */
.word     SPI2_IRQHandler                  /* SPI2                         */
.word     USART1_IRQHandler                /* USART1                       */
.word     USART2_IRQHandler                /* USART2                       */
.word     USART3_IRQHandler                /* USART3                       */
.word     EXTI15_10_IRQHandler             /* External Line[15:10]s        */
.word     RTC_Alarm_IRQHandler             /* RTC Alarm (A and B) through EXTI Line */
.word     OTG_FS_WKUP_IRQHandler           /* USB OTG FS Wakeup through EXTI line */
.word     TIM8_BRK_TIM12_IRQHandler        /* TIM8 Break and TIM12         */
.word     TIM8_UP_TIM13_IRQHandler         /* TIM8 Update and TIM13        */
.word     TIM8_TRG_COM_TIM14_IRQHandler    /* TIM8 Trigger and Commutation and TIM14 */
.word     TIM8_CC_IRQHandler               /* TIM8 Capture Compare         */
.word     DMA1_Stream7_IRQHandler          /* DMA1 Stream7                 */
.word     FSMC_IRQHandler                  /* FSMC                         */
.word     SDIO_IRQHandler                  /* SDIO                         */
.word     TIM5_IRQHandler                  /* TIM5                         */
.word     SPI3_IRQHandler                  /* SPI3                         */
.word     UART4_IRQHandler                 /* UART4                        */
.word     UART5_IRQHandler                 /* UART5                        */
.word     TIM6_DAC_IRQHandler              /* TIM6 and DAC1&2 underrun errors */
.word     TIM7_IRQHandler                  /* TIM7                         */
.word     DMA2_Stream0_IRQHandler          /* DMA2 Stream 0                */
.word     DMA2_Stream1_IRQHandler          /* DMA2 Stream 1                */
.word     DMA2_Stream2_IRQHandler          /* DMA2 Stream 2                */
.word     DMA2_Stream3_IRQHandler          /* DMA2 Stream 3                */
.word     DMA2_Stream4_IRQHandler          /* DMA2 Stream 4                */
.word     ETH_IRQHandler                   /* Ethernet                     */
.word     ETH_WKUP_IRQHandler              /* Ethernet Wakeup through EXTI line */
.word     CAN2_TX_IRQHandler               /* CAN2 TX                      */
.word     CAN2_RX0_IRQHandler              /* CAN2 RX0                     */
.word     CAN2_RX1_IRQHandler              /* CAN2 RX1                     */
```

```
  .word     CAN2_SCE_IRQHandler             /* CAN2 SCE                    */
  .word     OTG_FS_IRQHandler               /* USB OTG FS                  */
  .word     DMA2_Stream5_IRQHandler         /* DMA2 Stream 5               */
  .word     DMA2_Stream6_IRQHandler         /* DMA2 Stream 6               */
  .word     DMA2_Stream7_IRQHandler         /* DMA2 Stream 7               */
  .word     USART6_IRQHandler               /* USART6                      */
  .word     I2C3_EV_IRQHandler              /* I2C3 event                  */
  .word     I2C3_ER_IRQHandler              /* I2C3 error                  */
  .word     OTG_HS_EP1_OUT_IRQHandler       /* USB OTG HS End Point 1 Out   */
  .word     OTG_HS_EP1_IN_IRQHandler        /* USB OTG HS End Point 1 In    */
  .word     OTG_HS_WKUP_IRQHandler          /* USB OTG HS Wakeup through EXTI */
  .word     OTG_HS_IRQHandler               /* USB OTG HS                  */
  .word     DCMI_IRQHandler                 /* DCMI                        */
  .word     CRYP_IRQHandler                 /* CRYP crypto                 */
  .word     HASH_RNG_IRQHandler             /* Hash and Rng                */
  .word     FPU_IRQHandler                  /* FPU                         */

/*******************************************************************************
*
* Provide weak aliases for each Exception handler to the Default_Handler.
* As they are weak aliases, any function with the same name will override
* this definition.
*
*******************************************************************************/
   .weak      NMI_Handler
   .thumb_set NMI_Handler,Default_Handler

   .weak      HardFault_Handler
   .thumb_set HardFault_Handler,Default_Handler

   .weak      MemManage_Handler
   .thumb_set MemManage_Handler,Default_Handler

   .weak      BusFault_Handler
   .thumb_set BusFault_Handler,Default_Handler

   .weak      UsageFault_Handler
   .thumb_set UsageFault_Handler,Default_Handler

   .weak      SVC_Handler
   .thumb_set SVC_Handler,Default_Handler

   .weak      DebugMon_Handler
   .thumb_set DebugMon_Handler,Default_Handler

   .weak      PendSV_Handler
   .thumb_set PendSV_Handler,Default_Handler

   .weak      SysTick_Handler
   .thumb_set SysTick_Handler,Default_Handler

   .weak      WWDG_IRQHandler
   .thumb_set WWDG_IRQHandler,Default_Handler

   .weak      PVD_IRQHandler
   .thumb_set PVD_IRQHandler,Default_Handler

   .weak      TAMP_STAMP_IRQHandler
   .thumb_set TAMP_STAMP_IRQHandler,Default_Handler

   .weak      RTC_WKUP_IRQHandler
   .thumb_set RTC_WKUP_IRQHandler,Default_Handler

   .weak      FLASH_IRQHandler
   .thumb_set FLASH_IRQHandler,Default_Handler

   .weak      RCC_IRQHandler
   .thumb_set RCC_IRQHandler,Default_Handler

   .weak      EXTI0_IRQHandler
   .thumb_set EXTI0_IRQHandler,Default_Handler

   .weak      EXTI1_IRQHandler
   .thumb_set EXTI1_IRQHandler,Default_Handler

   .weak      EXTI2_IRQHandler
```

```
.thumb_set EXTI2_IRQHandler,Default_Handler

.weak      EXTI3_IRQHandler
.thumb_set EXTI3_IRQHandler,Default_Handler

.weak      EXTI4_IRQHandler
.thumb_set EXTI4_IRQHandler,Default_Handler

.weak      DMA1_Stream0_IRQHandler
.thumb_set DMA1_Stream0_IRQHandler,Default_Handler

.weak      DMA1_Stream1_IRQHandler
.thumb_set DMA1_Stream1_IRQHandler,Default_Handler

.weak      DMA1_Stream2_IRQHandler
.thumb_set DMA1_Stream2_IRQHandler,Default_Handler

.weak      DMA1_Stream3_IRQHandler
.thumb_set DMA1_Stream3_IRQHandler,Default_Handler

.weak      DMA1_Stream4_IRQHandler
.thumb_set DMA1_Stream4_IRQHandler,Default_Handler

.weak      DMA1_Stream5_IRQHandler
.thumb_set DMA1_Stream5_IRQHandler,Default_Handler

.weak      DMA1_Stream6_IRQHandler
.thumb_set DMA1_Stream6_IRQHandler,Default_Handler

.weak      ADC_IRQHandler
.thumb_set ADC_IRQHandler,Default_Handler

.weak      CAN1_TX_IRQHandler
.thumb_set CAN1_TX_IRQHandler,Default_Handler

.weak      CAN1_RX0_IRQHandler
.thumb_set CAN1_RX0_IRQHandler,Default_Handler

.weak      CAN1_RX1_IRQHandler
.thumb_set CAN1_RX1_IRQHandler,Default_Handler

.weak      CAN1_SCE_IRQHandler
.thumb_set CAN1_SCE_IRQHandler,Default_Handler

.weak      EXTI9_5_IRQHandler
.thumb_set EXTI9_5_IRQHandler,Default_Handler

.weak      TIM1_BRK_TIM9_IRQHandler
.thumb_set TIM1_BRK_TIM9_IRQHandler,Default_Handler

.weak      TIM1_UP_TIM10_IRQHandler
.thumb_set TIM1_UP_TIM10_IRQHandler,Default_Handler

.weak      TIM1_TRG_COM_TIM11_IRQHandler
.thumb_set TIM1_TRG_COM_TIM11_IRQHandler,Default_Handler

.weak      TIM1_CC_IRQHandler
.thumb_set TIM1_CC_IRQHandler,Default_Handler

.weak      TIM2_IRQHandler
.thumb_set TIM2_IRQHandler,Default_Handler

.weak      TIM3_IRQHandler
.thumb_set TIM3_IRQHandler,Default_Handler

.weak      TIM4_IRQHandler
.thumb_set TIM4_IRQHandler,Default_Handler

.weak      I2C1_EV_IRQHandler
.thumb_set I2C1_EV_IRQHandler,Default_Handler

.weak      I2C1_ER_IRQHandler
.thumb_set I2C1_ER_IRQHandler,Default_Handler

.weak      I2C2_EV_IRQHandler
.thumb_set I2C2_EV_IRQHandler,Default_Handler
```

```
.weak      I2C2_ER_IRQHandler
.thumb_set I2C2_ER_IRQHandler,Default_Handler

.weak      SPI1_IRQHandler
.thumb_set SPI1_IRQHandler,Default_Handler

.weak      SPI2_IRQHandler
.thumb_set SPI2_IRQHandler,Default_Handler

.weak      USART1_IRQHandler
.thumb_set USART1_IRQHandler,Default_Handler

.weak      USART2_IRQHandler
.thumb_set USART2_IRQHandler,Default_Handler

.weak      USART3_IRQHandler
.thumb_set USART3_IRQHandler,Default_Handler

.weak      EXTI15_10_IRQHandler
.thumb_set EXTI15_10_IRQHandler,Default_Handler

.weak      RTC_Alarm_IRQHandler
.thumb_set RTC_Alarm_IRQHandler,Default_Handler

.weak      OTG_FS_WKUP_IRQHandler
.thumb_set OTG_FS_WKUP_IRQHandler,Default_Handler

.weak      TIM8_BRK_TIM12_IRQHandler
.thumb_set TIM8_BRK_TIM12_IRQHandler,Default_Handler

.weak      TIM8_UP_TIM13_IRQHandler
.thumb_set TIM8_UP_TIM13_IRQHandler,Default_Handler

.weak      TIM8_TRG_COM_TIM14_IRQHandler
.thumb_set TIM8_TRG_COM_TIM14_IRQHandler,Default_Handler

.weak      TIM8_CC_IRQHandler
.thumb_set TIM8_CC_IRQHandler,Default_Handler

.weak      DMA1_Stream7_IRQHandler
.thumb_set DMA1_Stream7_IRQHandler,Default_Handler

.weak      FSMC_IRQHandler
.thumb_set FSMC_IRQHandler,Default_Handler

.weak      SDIO_IRQHandler
.thumb_set SDIO_IRQHandler,Default_Handler

.weak      TIM5_IRQHandler
.thumb_set TIM5_IRQHandler,Default_Handler

.weak      SPI3_IRQHandler
.thumb_set SPI3_IRQHandler,Default_Handler

.weak      UART4_IRQHandler
.thumb_set UART4_IRQHandler,Default_Handler

.weak      UART5_IRQHandler
.thumb_set UART5_IRQHandler,Default_Handler

.weak      TIM6_DAC_IRQHandler
.thumb_set TIM6_DAC_IRQHandler,Default_Handler

.weak      TIM7_IRQHandler
.thumb_set TIM7_IRQHandler,Default_Handler

.weak      DMA2_Stream0_IRQHandler
.thumb_set DMA2_Stream0_IRQHandler,Default_Handler

.weak      DMA2_Stream1_IRQHandler
.thumb_set DMA2_Stream1_IRQHandler,Default_Handler

.weak      DMA2_Stream2_IRQHandler
.thumb_set DMA2_Stream2_IRQHandler,Default_Handler
```

```
.weak      DMA2_Stream3_IRQHandler
.thumb_set DMA2_Stream3_IRQHandler,Default_Handler

.weak      DMA2_Stream4_IRQHandler
.thumb_set DMA2_Stream4_IRQHandler,Default_Handler

.weak      ETH_IRQHandler
.thumb_set ETH_IRQHandler,Default_Handler

.weak      ETH_WKUP_IRQHandler
.thumb_set ETH_WKUP_IRQHandler,Default_Handler

.weak      CAN2_TX_IRQHandler
.thumb_set CAN2_TX_IRQHandler,Default_Handler

.weak      CAN2_RX0_IRQHandler
.thumb_set CAN2_RX0_IRQHandler,Default_Handler

.weak      CAN2_RX1_IRQHandler
.thumb_set CAN2_RX1_IRQHandler,Default_Handler

.weak      CAN2_SCE_IRQHandler
.thumb_set CAN2_SCE_IRQHandler,Default_Handler

.weak      OTG_FS_IRQHandler
.thumb_set OTG_FS_IRQHandler,Default_Handler

.weak      DMA2_Stream5_IRQHandler
.thumb_set DMA2_Stream5_IRQHandler,Default_Handler

.weak      DMA2_Stream6_IRQHandler
.thumb_set DMA2_Stream6_IRQHandler,Default_Handler

.weak      DMA2_Stream7_IRQHandler
.thumb_set DMA2_Stream7_IRQHandler,Default_Handler

.weak      USART6_IRQHandler
.thumb_set USART6_IRQHandler,Default_Handler

.weak      I2C3_EV_IRQHandler
.thumb_set I2C3_EV_IRQHandler,Default_Handler

.weak      I2C3_ER_IRQHandler
.thumb_set I2C3_ER_IRQHandler,Default_Handler

.weak      OTG_HS_EP1_OUT_IRQHandler
.thumb_set OTG_HS_EP1_OUT_IRQHandler,Default_Handler

.weak      OTG_HS_EP1_IN_IRQHandler
.thumb_set OTG_HS_EP1_IN_IRQHandler,Default_Handler

.weak      OTG_HS_WKUP_IRQHandler
.thumb_set OTG_HS_WKUP_IRQHandler,Default_Handler

.weak      OTG_HS_IRQHandler
.thumb_set OTG_HS_IRQHandler,Default_Handler

.weak      DCMI_IRQHandler
.thumb_set DCMI_IRQHandler,Default_Handler

.weak      CRYP_IRQHandler
.thumb_set CRYP_IRQHandler,Default_Handler

.weak      HASH_RNG_IRQHandler
.thumb_set HASH_RNG_IRQHandler,Default_Handler

.weak      FPU_IRQHandler
.thumb_set FPU_IRQHandler,Default_Handler
```

stm32f4xx_conf.h

```
/********************** (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
/**
  ******************************************************************************
  * @file    Project/STM32F4xx_StdPeriph_Templates/stm32f4xx_conf.h
  * @author  MCD Application Team
  * @version V1.1.0
  * @date    18-January-2013
  * @brief   Library configuration file.
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; COPYRIGHT 2013 STMicroelectronics</center></h2>
  *
  * Licensed under MCD-ST Liberty SW License Agreement V2, (the "License");
  * You may not use this file except in compliance with the License.
  * You may obtain a copy of the License at:
  *
  *        http://www.st.com/software_license_agreement_liberty_v2
  *
  * Unless required by applicable law or agreed to in writing, software
  * distributed under the License is distributed on an "AS IS" BASIS,
  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  * See the License for the specific language governing permissions and
  * limitations under the License.
  *
  ******************************************************************************
  */

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __STM32F4xx_CONF_H
#define __STM32F4xx_CONF_H

/* Includes ------------------------------------------------------------------*/
/* Uncomment the line below to enable peripheral header file inclusion */
#include "stm32f4xx_adc.h"
#include "stm32f4xx_can.h"
#include "stm32f4xx_crc.h"
#include "stm32f4xx_cryp.h"
#include "stm32f4xx_dac.h"
#include "stm32f4xx_dbgmcu.h"
#include "stm32f4xx_dcmi.h"
#include "stm32f4xx_dma.h"
#include "stm32f4xx_exti.h"
#include "stm32f4xx_flash.h"
#include "stm32f4xx_fsmc.h"
#include "stm32f4xx_hash.h"
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_i2c.h"
#include "stm32f4xx_iwdg.h"
#include "stm32f4xx_pwr.h"
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_rng.h"
#include "stm32f4xx_rtc.h"
#include "stm32f4xx_sdio.h"
#include "stm32f4xx_spi.h"
#include "stm32f4xx_syscfg.h"
#include "stm32f4xx_tim.h"
#include "stm32f4xx_usart.h"
#include "stm32f4xx_wwdg.h"
#include "misc.h" /* High level functions for NVIC and SysTick (add-on to CMSIS functions) */

/* Exported types ------------------------------------------------------------*/
/* Exported constants --------------------------------------------------------*/

/* If an external clock source is used, then the value of the following define
   should be set to the value of the external clock source, else, if no external
   clock is used, keep this define commented */
/*#define I2S_EXTERNAL_CLOCK_VAL   12288000 */ /* Value of the external clock in Hz */


/* Uncomment the line below to expanse the "assert_param" macro in the
```

```
  Standard Peripheral Library drivers code */
/* #define USE_FULL_ASSERT    1 */

/* Exported macro ------------------------------------------------------------*/
#ifdef  USE_FULL_ASSERT

/**
  * @brief  The assert_param macro is used for function's parameters check.
  * @param  expr: If expr is false, it calls assert_failed function
  *   which reports the name of the source file and the source
  *   line number of the call that failed.
  *   If expr is true, it returns no value.
  * @retval None
  */
  #define assert_param(expr) ((expr) ? (void)0 : assert_failed((uint8_t *)__FILE__, __LINE__))
/* Exported functions ------------------------------------------------------- */
  void assert_failed(uint8_t* file, uint32_t line);
#else
  #define assert_param(expr) ((void)0)
#endif /* USE_FULL_ASSERT */

#endif /* __STM32F4xx_CONF_H */
```

system_stm32f4xx.c

```
/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
/**
  ******************************************************************************
  * @file    system_stm32f4xx.c
  * @author  MCD Application Team
  * @version V1.0.0
  * @date    19-September-2011
  * @brief   CMSIS Cortex-M4 Device Peripheral Access Layer System Source File.
  *          This file contains the system clock configuration for STM32F4xx devices,
  *          and is generated by the clock configuration tool
  *          stm32f4xx_Clock_Configuration_V1.0.0.xls
  *
  * 1.  This file provides two functions and one global variable to be called from
  *     user application:
  *      - SystemInit(): Setups the system clock (System clock source, PLL Multiplier
  *                      and Divider factors, AHB/APBx prescalers and Flash settings),
  *                      depending on the configuration made in the clock xls tool.
  *                      This function is called at startup just after reset and
  *                      before branch to main program. This call is made inside
  *                      the "startup_stm32f4xx.s" file.
  *
  *      - SystemCoreClock variable: Contains the core clock (HCLK), it can be used
  *                                  by the user application to setup the SysTick
  *                                  timer or configure other parameters.
  *
  *      - SystemCoreClockUpdate(): Updates the variable SystemCoreClock and must
  *                                 be called whenever the core clock is changed
  *                                 during program execution.
  *
  * 2. After each device reset the HSI (16 MHz) is used as system clock source.
  *    Then SystemInit() function is called, in "startup_stm32f4xx.s" file, to
  *    configure the system clock before to branch to main program.
  *
  * 3. If the system clock source selected by user fails to startup, the SystemInit()
  *    function will do nothing and HSI still used as system clock source. User can
  *    add some code to deal with this issue inside the SetSysClock() function.
  *
  * 4. The default value of HSE crystal is set to 8 MHz, refer to "HSE_VALUE" define
  *    in "stm32f4xx.h" file. When HSE is used as system clock source, directly or
  *    through PLL, and you are using different crystal you have to adapt the HSE
  *    value to your own configuration.
  *
  * 5. This file configures the system clock as follows:
  *=============================================================================
  *=============================================================================
  *       Supported STM32F4xx device revision    | Rev A
  *-----------------------------------------------------------------------------
```

```
  *       System Clock source                    | PLL (HSE)
  *-----------------------------------------------------------------------------
  *       SYSCLK(Hz)                             | 168000000
  *-----------------------------------------------------------------------------
  *       HCLK(Hz)                               | 168000000
  *-----------------------------------------------------------------------------
  *       AHB Prescaler                          | 1
  *-----------------------------------------------------------------------------
  *       APB1 Prescaler                         | 4
  *-----------------------------------------------------------------------------
  *       APB2 Prescaler                         | 2
  *-----------------------------------------------------------------------------
  *       HSE Frequency(Hz)                      | 8000000
  *-----------------------------------------------------------------------------
  *       PLL_M                                  | 8
  *-----------------------------------------------------------------------------
  *       PLL_N                                  | 336
  *-----------------------------------------------------------------------------
  *       PLL_P                                  | 2
  *-----------------------------------------------------------------------------
  *       PLL_Q                                  | 7
  *-----------------------------------------------------------------------------
  *       PLLI2S_N                               | NA
  *-----------------------------------------------------------------------------
  *       PLLI2S_R                               | NA
  *-----------------------------------------------------------------------------
  *       I2S input clock                        | NA
  *-----------------------------------------------------------------------------
  *       VDD(V)                                 | 3.3
  *-----------------------------------------------------------------------------
  *       High Performance mode                  | Enabled
  *-----------------------------------------------------------------------------
  *       Flash Latency(WS)                      | 5
  *-----------------------------------------------------------------------------
  *       Prefetch Buffer                        | OFF
  *-----------------------------------------------------------------------------
  *       Instruction cache                      | ON
  *-----------------------------------------------------------------------------
  *       Data cache                             | ON
  *-----------------------------------------------------------------------------
  *       Require 48MHz for USB OTG FS,          | Enabled
  *       SDIO and RNG clock                     |
  *-----------------------------------------------------------------------------
  *=============================================================================
  ******************************************************************************
  * @attention
  *
  * THE PRESENT FIRMWARE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS
  * WITH CODING INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE
  * TIME. AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY
  * DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING
  * FROM THE CONTENT OF SUCH FIRMWARE AND/OR THE USE MADE BY CUSTOMERS OF THE
  * CODING INFORMATION CONTAINED HEREIN IN CONNECTION WITH THEIR PRODUCTS.
  *
  * <h2><center>&copy; COPYRIGHT 2011 STMicroelectronics</center></h2>
  ******************************************************************************
  */

/** @addtogroup CMSIS
  * @{
  */

/** @addtogroup stm32f4xx_system
  * @{
  */

/** @addtogroup STM32F4xx_System_Private_Includes
  * @{
  */

#include "stm32f4xx.h"

/**
  * @}
  */
```

```
/** @addtogroup STM32F4xx_System_Private_TypesDefinitions
  * @{
  */

/**
  * @}
  */

/** @addtogroup STM32F4xx_System_Private_Defines
  * @{
  */

/*!< Uncomment the following line if you need to use external SRAM mounted
     on STM324xG_EVAL board as data memory  */
/* #define DATA_IN_ExtSRAM */

/*!< Uncomment the following line if you need to relocate your vector Table in
     Internal SRAM. */
/* #define VECT_TAB_SRAM */
#define VECT_TAB_OFFSET  0x00 /*!< Vector Table base offset field.
                                  This value must be a multiple of 0x200. */


/* PLL_VCO = (HSE_VALUE or HSI_VALUE / PLL_M) * PLL_N */
#define PLL_M      8
#define PLL_N      336

/* SYSCLK = PLL_VCO / PLL_P */
#define PLL_P      2

/* USB OTG FS, SDIO and RNG Clock =  PLL_VCO / PLLQ */
#define PLL_Q      7

/**
  * @}
  */

/** @addtogroup STM32F4xx_System_Private_Macros
  * @{
  */

/**
  * @}
  */

/** @addtogroup STM32F4xx_System_Private_Variables
  * @{
  */

  uint32_t SystemCoreClock = 168000000;

  __I uint8_t AHBPrescTable[16] = {0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 6, 7, 8, 9};

/**
  * @}
  */

/** @addtogroup STM32F4xx_System_Private_FunctionPrototypes
  * @{
  */

static void SetSysClock(void);
#ifdef DATA_IN_ExtSRAM
  static void SystemInit_ExtMemCtl(void);
#endif /* DATA_IN_ExtSRAM */

/**
  * @}
  */

/** @addtogroup STM32F4xx_System_Private_Functions
  * @{
  */

/**
  * @brief  Setup the microcontroller system
```

```
  *        Initialize the Embedded Flash Interface, the PLL and update the
  *        SystemFrequency variable.
  * @param  None
  * @retval None
  */
void SystemInit(void)
{
  /* FPU settings ------------------------------------------------------------*/
  #if (__FPU_PRESENT == 1) && (__FPU_USED == 1)
              SCB->CPACR |= ((3UL << 10*2)|(3UL << 11*2));  /* set CP10 and CP11 Full Access */
  #endif

  /* Reset the RCC clock configuration to the default reset state ------------*/
  /* Set HSION bit */
  RCC->CR |= (uint32_t)0x00000001;

  /* Reset CFGR register */
  RCC->CFGR = 0x00000000;

  /* Reset HSEON, CSSON and PLLON bits */
  RCC->CR &= (uint32_t)0xFEF6FFFF;

  /* Reset PLLCFGR register */
  RCC->PLLCFGR = 0x24003010;

  /* Reset HSEBYP bit */
  RCC->CR &= (uint32_t)0xFFFBFFFF;

  /* Disable all interrupts */
  RCC->CIR = 0x00000000;

#ifdef DATA_IN_ExtSRAM
  SystemInit_ExtMemCtl();
#endif /* DATA_IN_ExtSRAM */

  /* Configure the System clock source, PLL Multiplier and Divider factors,
     AHB/APBx prescalers and Flash settings ----------------------------------*/
  SetSysClock();

  /* Configure the Vector Table location add offset address ------------------*/
#ifdef VECT_TAB_SRAM
  SCB->VTOR = SRAM_BASE | VECT_TAB_OFFSET; /* Vector Table Relocation in Internal SRAM */
#else
  SCB->VTOR = FLASH_BASE | VECT_TAB_OFFSET; /* Vector Table Relocation in Internal FLASH */
#endif
}

/**
   * @brief  Update SystemCoreClock variable according to Clock Register Values.
  *        The SystemCoreClock variable contains the core clock (HCLK), it can
  *        be used by the user application to setup the SysTick timer or configure
  *        other parameters.
  *
  * @note   Each time the core clock (HCLK) changes, this function must be called
  *        to update SystemCoreClock variable value. Otherwise, any configuration
  *        based on this variable will be incorrect.
  *
  * @note   - The system frequency computed by this function is not the real
  *          frequency in the chip. It is calculated based on the predefined
  *          constant and the selected clock source:
  *
  *          - If SYSCLK source is HSI, SystemCoreClock will contain the HSI_VALUE(*)
  *
  *          - If SYSCLK source is HSE, SystemCoreClock will contain the HSE_VALUE(**)
  *
  *          - If SYSCLK source is PLL, SystemCoreClock will contain the HSE_VALUE(**)
  *            or HSI_VALUE(*) multiplied/divided by the PLL factors.
  *
  *        (*) HSI_VALUE is a constant defined in stm32f4xx.h file (default value
  *            16 MHz) but the real value may vary depending on the variations
  *            in voltage and temperature.
  *
  *        (**) HSE_VALUE is a constant defined in stm32f4xx.h file (default value
  *             25 MHz), user has to ensure that HSE_VALUE is same as the real
  *             frequency of the crystal used. Otherwise, this function may
  *             have wrong result.
```

```
  *
  *        - The result of this function could be not correct when using fractional
  *          value for HSE crystal.
  *
  * @param  None
  * @retval None
  */
void SystemCoreClockUpdate(void)
{
  uint32_t tmp = 0, pllvco = 0, pllp = 2, pllsource = 0, pllm = 2;

  /* Get SYSCLK source -------------------------------------------------------*/
  tmp = RCC->CFGR & RCC_CFGR_SWS;

  switch (tmp)
  {
    case 0x00:  /* HSI used as system clock source */
      SystemCoreClock = HSI_VALUE;
      break;
    case 0x04:  /* HSE used as system clock source */
      SystemCoreClock = HSE_VALUE;
      break;
    case 0x08:  /* PLL used as system clock source */

      /* PLL_VCO = (HSE_VALUE or HSI_VALUE / PLL_M) * PLL_N
         SYSCLK = PLL_VCO / PLL_P
         */
      pllsource = (RCC->PLLCFGR & RCC_PLLCFGR_PLLSRC) >> 22;
      pllm = RCC->PLLCFGR & RCC_PLLCFGR_PLLM;

      if (pllsource != 0)
      {
        /* HSE used as PLL clock source */
        pllvco = (HSE_VALUE / pllm) * ((RCC->PLLCFGR & RCC_PLLCFGR_PLLN) >> 6);
      }
      else
      {
        /* HSI used as PLL clock source */
        pllvco = (HSI_VALUE / pllm) * ((RCC->PLLCFGR & RCC_PLLCFGR_PLLN) >> 6);
      }

      pllp = (((RCC->PLLCFGR & RCC_PLLCFGR_PLLP) >>16) + 1 ) *2;
      SystemCoreClock = pllvco/pllp;
      break;
    default:
      SystemCoreClock = HSI_VALUE;
      break;
  }
  /* Compute HCLK frequency --------------------------------------------------*/
  /* Get HCLK prescaler */
  tmp = AHBPrescTable[((RCC->CFGR & RCC_CFGR_HPRE) >> 4)];
  /* HCLK frequency */
  SystemCoreClock >>= tmp;
}

/**
  * @brief  Configures the System clock source, PLL Multiplier and Divider factors,
  *         AHB/APBx prescalers and Flash settings
  * @Note   This function should be called only once the RCC clock configuration
  *         is reset to the default reset state (done in SystemInit() function).
  * @param  None
  * @retval None
  */
static void SetSysClock(void)
{
/******************************************************************************/
/*            PLL (clocked by HSE) used as System clock source                */
/******************************************************************************/
  __IO uint32_t StartUpCounter = 0, HSEStatus = 0;

  /* Enable HSE */
  RCC->CR |= ((uint32_t)RCC_CR_HSEON);

  /* Wait till HSE is ready and if Time out is reached exit */
  do
  {
```

```
    HSEStatus = RCC->CR & RCC_CR_HSERDY;
    StartUpCounter++;
  } while((HSEStatus == 0) && (StartUpCounter != HSE_STARTUP_TIMEOUT));

  if ((RCC->CR & RCC_CR_HSERDY) != RESET)
  {
    HSEStatus = (uint32_t)0x01;
  }
  else
  {
    HSEStatus = (uint32_t)0x00;
  }

  if (HSEStatus == (uint32_t)0x01)
  {
    /* Enable high performance mode, System frequency up to 168 MHz */
    RCC->APB1ENR |= RCC_APB1ENR_PWREN;
    PWR->CR |= PWR_CR_PMODE;

    /* HCLK = SYSCLK / 1*/
    RCC->CFGR |= RCC_CFGR_HPRE_DIV1;

    /* PCLK2 = HCLK / 2*/
    RCC->CFGR |= RCC_CFGR_PPRE2_DIV2;

    /* PCLK1 = HCLK / 4*/
    RCC->CFGR |= RCC_CFGR_PPRE1_DIV4;

    /* Configure the main PLL */
    RCC->PLLCFGR = PLL_M | (PLL_N << 6) | (((PLL_P >> 1) -1) << 16) |
                   (RCC_PLLCFGR_PLLSRC_HSE) | (PLL_Q << 24);

    /* Enable the main PLL */
    RCC->CR |= RCC_CR_PLLON;

    /* Wait till the main PLL is ready */
    while((RCC->CR & RCC_CR_PLLRDY) == 0)
    {
    }

    /* Configure Flash prefetch, Instruction cache, Data cache and wait state */
    FLASH->ACR = FLASH_ACR_ICEN |FLASH_ACR_DCEN |FLASH_ACR_LATENCY_5WS;

    /* Select the main PLL as system clock source */
    RCC->CFGR &= (uint32_t)((uint32_t)~(RCC_CFGR_SW));
    RCC->CFGR |= RCC_CFGR_SW_PLL;

    /* Wait till the main PLL is used as system clock source */
    while ((RCC->CFGR & (uint32_t)RCC_CFGR_SWS ) != RCC_CFGR_SWS_PLL);
    {
    }
  }
  else
  { /* If HSE fails to start-up, the application will have wrong clock
       configuration. User can add here some code to deal with this error */
  }

}

/**
  * @brief  Setup the external memory controller. Called in startup_stm32f4xx.s
  *         before jump to __main
  * @param  None
  * @retval None
  */
#ifdef DATA_IN_ExtSRAM
/**
  * @brief  Setup the external memory controller.
  *         Called in startup_stm32f4xx.s before jump to main.
  *         This function configures the external SRAM mounted on STM324xG_EVAL board
  *         This SRAM will be used as program data memory (including heap and stack).
  * @param  None
  * @retval None
  */
void SystemInit_ExtMemCtl(void)
{
```

```
/*-- GPIOs Configuration -----------------------------------------------------*/
/*
 +-------------------+--------------------+------------------+------------------+
 +                    SRAM pins assignment                                      +
 +-------------------+--------------------+------------------+------------------+
 | PD0  <-> FSMC_D2  | PE0  <-> FSMC_NBL0 | PF0  <-> FSMC_A0 | PG0 <-> FSMC_A10 |
 | PD1  <-> FSMC_D3  | PE1  <-> FSMC_NBL1 | PF1  <-> FSMC_A1 | PG1 <-> FSMC_A11 |
 | PD4  <-> FSMC_NOE | PE3  <-> FSMC_A19  | PF2  <-> FSMC_A2 | PG2 <-> FSMC_A12 |
 | PD5  <-> FSMC_NWE | PE4  <-> FSMC_A20  | PF3  <-> FSMC_A3 | PG3 <-> FSMC_A13 |
 | PD8  <-> FSMC_D13 | PE7  <-> FSMC_D4   | PF4  <-> FSMC_A4 | PG4 <-> FSMC_A14 |
 | PD9  <-> FSMC_D14 | PE8  <-> FSMC_D5   | PF5  <-> FSMC_A5 | PG5 <-> FSMC_A15 |
 | PD10 <-> FSMC_D15 | PE9  <-> FSMC_D6   | PF12 <-> FSMC_A6 | PG9 <-> FSMC_NE2 |
 | PD11 <-> FSMC_A16 | PE10 <-> FSMC_D7   | PF13 <-> FSMC_A7 |------------------+
 | PD12 <-> FSMC_A17 | PE11 <-> FSMC_D8   | PF14 <-> FSMC_A8 |
 | PD13 <-> FSMC_A18 | PE12 <-> FSMC_D9   | PF15 <-> FSMC_A9 |
 | PD14 <-> FSMC_D0  | PE13 <-> FSMC_D10  |------------------+
 | PD15 <-> FSMC_D1  | PE14 <-> FSMC_D11  |
 |                   | PE15 <-> FSMC_D12  |
 +-------------------+--------------------+
*/
  /* Enable GPIOD, GPIOE, GPIOF and GPIOG interface clock */
  RCC->AHB1ENR   = 0x00000078;

  /* Connect PDx pins to FSMC Alternate function */
  GPIOD->AFR[0]  = 0x00cc00cc;
  GPIOD->AFR[1]  = 0xcc0ccccc;
  /* Configure PDx pins in Alternate function mode */
  GPIOD->MODER   = 0xaaaa0a0a;
  /* Configure PDx pins speed to 100 MHz */
  GPIOD->OSPEEDR = 0xffff0f0f;
  /* Configure PDx pins Output type to push-pull */
  GPIOD->OTYPER  = 0x00000000;
  /* No pull-up, pull-down for PDx pins */
  GPIOD->PUPDR   = 0x00000000;

  /* Connect PEx pins to FSMC Alternate function */
  GPIOE->AFR[0]  = 0xc00cc0cc;
  GPIOE->AFR[1]  = 0xcccccccc;
  /* Configure PEx pins in Alternate function mode */
  GPIOE->MODER   = 0xaaaa828a;
  /* Configure PEx pins speed to 100 MHz */
  GPIOE->OSPEEDR = 0xffffc3cf;
  /* Configure PEx pins Output type to push-pull */
  GPIOE->OTYPER  = 0x00000000;
  /* No pull-up, pull-down for PEx pins */
  GPIOE->PUPDR   = 0x00000000;

  /* Connect PFx pins to FSMC Alternate function */
  GPIOF->AFR[0]  = 0x00cccccc;
  GPIOF->AFR[1]  = 0xcccc0000;
  /* Configure PFx pins in Alternate function mode */
  GPIOF->MODER   = 0xaa000aaa;
  /* Configure PFx pins speed to 100 MHz */
  GPIOF->OSPEEDR = 0xff000fff;
  /* Configure PFx pins Output type to push-pull */
  GPIOF->OTYPER  = 0x00000000;
  /* No pull-up, pull-down for PFx pins */
  GPIOF->PUPDR   = 0x00000000;

  /* Connect PGx pins to FSMC Alternate function */
  GPIOG->AFR[0]  = 0x00cccccc;
  GPIOG->AFR[1]  = 0x000000c0;
  /* Configure PGx pins in Alternate function mode */
  GPIOG->MODER   = 0x00080aaa;
  /* Configure PGx pins speed to 100 MHz */
  GPIOG->OSPEEDR = 0x000c0fff;
  /* Configure PGx pins Output type to push-pull */
  GPIOG->OTYPER  = 0x00000000;
  /* No pull-up, pull-down for PGx pins */
  GPIOG->PUPDR   = 0x00000000;

/*-- FSMC Configuration ------------------------------------------------------*/
  /* Enable the FSMC interface clock */
  RCC->AHB3ENR         = 0x00000001;

  /* Configure and enable Bank1_SRAM2 */
```

```
  FSMC_Bank1->BTCR[2]  = 0x00001015;
  FSMC_Bank1->BTCR[3]  = 0x00010603;//0x00010400;
  FSMC_Bank1E->BWTR[2] = 0x0fffffff;
/*
  Bank1_SRAM2 is configured as follow:

  p.FSMC_AddressSetupTime = 3;//0;
  p.FSMC_AddressHoldTime = 0;
  p.FSMC_DataSetupTime = 6;//4;
  p.FSMC_BusTurnAroundDuration = 1;
  p.FSMC_CLKDivision = 0;
  p.FSMC_DataLatency = 0;
  p.FSMC_AccessMode = FSMC_AccessMode_A;

  FSMC_NORSRAMInitStructure.FSMC_Bank = FSMC_Bank1_NORSRAM2;
  FSMC_NORSRAMInitStructure.FSMC_DataAddressMux = FSMC_DataAddressMux_Disable;
  FSMC_NORSRAMInitStructure.FSMC_MemoryType = FSMC_MemoryType_PSRAM;
  FSMC_NORSRAMInitStructure.FSMC_MemoryDataWidth = FSMC_MemoryDataWidth_16b;
  FSMC_NORSRAMInitStructure.FSMC_BurstAccessMode = FSMC_BurstAccessMode_Disable;
  FSMC_NORSRAMInitStructure.FSMC_AsynchronousWait = FSMC_AsynchronousWait_Disable;
  FSMC_NORSRAMInitStructure.FSMC_WaitSignalPolarity = FSMC_WaitSignalPolarity_Low;
  FSMC_NORSRAMInitStructure.FSMC_WrapMode = FSMC_WrapMode_Disable;
  FSMC_NORSRAMInitStructure.FSMC_WaitSignalActive = FSMC_WaitSignalActive_BeforeWaitState;
  FSMC_NORSRAMInitStructure.FSMC_WriteOperation = FSMC_WriteOperation_Enable;
  FSMC_NORSRAMInitStructure.FSMC_WaitSignal = FSMC_WaitSignal_Disable;
  FSMC_NORSRAMInitStructure.FSMC_ExtendedMode = FSMC_ExtendedMode_Disable;
  FSMC_NORSRAMInitStructure.FSMC_WriteBurst = FSMC_WriteBurst_Disable;
  FSMC_NORSRAMInitStructure.FSMC_ReadWriteTimingStruct = &p;
  FSMC_NORSRAMInitStructure.FSMC_WriteTimingStruct = &p;
*/

}
#endif /* DATA_IN_ExtSRAM */


/**
  * @}
  */

/**
  * @}
  */

/**
  * @}
  */
/****************** (C) COPYRIGHT 2011 STMicroelectronics *****END OF FILE****/
```

tm_stm32f4_gpio.c

```
/**
 * |----------------------------------------------------------------------
 * | Copyright (C) Tilen Majerle, 2015
 * |
 * | This program is free software: you can redistribute it and/or modify
 * | it under the terms of the GNU General Public License as published by
 * | the Free Software Foundation, either version 3 of the License, or
 * | any later version.
 * |
 * | This program is distributed in the hope that it will be useful,
 * | but WITHOUT ANY WARRANTY; without even the implied warranty of
 * | MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * | GNU General Public License for more details.
 * |
 * | You should have received a copy of the GNU General Public License
 * | along with this program.  If not, see <http://www.gnu.org/licenses/>.
 * |----------------------------------------------------------------------
 */
#include "tm_stm32f4_gpio.h"

/* Private function */
static uint16_t GPIO_UsedPins[11] = {0,0,0,0,0,0,0,0,0,0,0};
```

```
/* Private functions */
void TM_GPIO_INT_EnableClock(GPIO_TypeDef* GPIOx);
void TM_GPIO_INT_DisableClock(GPIO_TypeDef* GPIOx);
void TM_GPIO_INT_Init(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, TM_GPIO_Mode_t GPIO_Mode,
TM_GPIO_OType_t GPIO_OType, TM_GPIO_PuPd_t GPIO_PuPd, TM_GPIO_Speed_t GPIO_Speed);

void TM_GPIO_Init(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, TM_GPIO_Mode_t GPIO_Mode, TM_GPIO_OType_t
GPIO_OType, TM_GPIO_PuPd_t GPIO_PuPd, TM_GPIO_Speed_t GPIO_Speed) {
	/* Check input */
	if (GPIO_Pin == 0x00) {
		return;
	}

	/* Enable clock for GPIO */
	TM_GPIO_INT_EnableClock(GPIOx);

	/* Do initialization */
	TM_GPIO_INT_Init(GPIOx, GPIO_Pin, GPIO_Mode, GPIO_OType, GPIO_PuPd, GPIO_Speed);
}

void TM_GPIO_InitAlternate(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, TM_GPIO_OType_t GPIO_OType,
TM_GPIO_PuPd_t GPIO_PuPd, TM_GPIO_Speed_t GPIO_Speed, uint8_t Alternate) {
	uint32_t pinpos;

	/* Check input */
	if (GPIO_Pin == 0x00) {
		return;
	}

	/* Enable GPIOx clock */
	TM_GPIO_INT_EnableClock(GPIOx);

	/* Set alternate functions for all pins */
	for (pinpos = 0; pinpos < 0x10; pinpos++) {
		/* Check pin */
		if ((GPIO_Pin & (1 << pinpos)) == 0) {
			continue;
		}

		/* Set alternate function */
		GPIOx->AFR[pinpos >> 0x03] = (GPIOx->AFR[pinpos >> 0x03] & ~(0x0F << (4 * (pinpos & 0x07))))
| (Alternate << (4 * (pinpos & 0x07)));
	}

	/* Do initialization */
	TM_GPIO_INT_Init(GPIOx, GPIO_Pin, TM_GPIO_Mode_AF, GPIO_OType, GPIO_PuPd, GPIO_Speed);
}

void TM_GPIO_DeInit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin) {
	uint8_t i;
	uint8_t ptr = TM_GPIO_GetPortSource(GPIOx);

	/* Go through all pins */
	for (i = 0x00; i < 0x10; i++) {
		/* Pin is set */
		if (GPIO_Pin & (1 << i)) {
			/* Set 11 bits combination for analog mode */
			GPIOx->MODER |= (0x03 << (2 * i));

			/* Pin is not used */
			GPIO_UsedPins[ptr] &= ~(1 << i);
		}
	}
}

void TM_GPIO_SetPinAsInput(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin) {
	uint8_t i;
	/* Go through all pins */
	for (i = 0x00; i < 0x10; i++) {
		/* Pin is set */
		if (GPIO_Pin & (1 << i)) {
			/* Set 00 bits combination for input */
			GPIOx->MODER &= ~(0x03 << (2 * i));
		}
	}
```

```
}

void TM_GPIO_SetPinAsOutput(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin) {
	uint8_t i;
	/* Go through all pins */
	for (i = 0x00; i < 0x10; i++) {
		/* Pin is set */
		if (GPIO_Pin & (1 << i)) {
			/* Set 01 bits combination for output */
			GPIOx->MODER = (GPIOx->MODER & ~(0x03 << (2 * i))) | (0x01 << (2 * i));
		}
	}
}

void TM_GPIO_SetPinAsAnalog(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin) {
	uint8_t i;
	/* Go through all pins */
	for (i = 0x00; i < 0x10; i++) {
		/* Pin is set */
		if (GPIO_Pin & (1 << i)) {
			/* Set 11 bits combination for analog mode */
			GPIOx->MODER |= (0x03 << (2 * i));
		}
	}
}

void TM_GPIO_SetPinAsAlternate(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin) {
	uint8_t i;

	/* Set alternate functions for all pins */
	for (i = 0; i < 0x10; i++) {
		/* Check pin */
		if ((GPIO_Pin & (1 << i)) == 0) {
			continue;
		}

		/* Set alternate mode */
		GPIOx->MODER = (GPIOx->MODER & ~(0x03 << (2 * i))) | (0x02 << (2 * i));
	}
}

void TM_GPIO_SetPullResistor(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, TM_GPIO_PuPd_t GPIO_PuPd) {
	uint8_t pinpos;

	/* Go through all pins */
	for (pinpos = 0; pinpos < 0x10; pinpos++) {
		/* Check if pin available */
		if ((GPIO_Pin & (1 << pinpos)) == 0) {
			continue;
		}

		/* Set GPIO PUPD register */
		GPIOx->PUPDR = (GPIOx->PUPDR & ~(0x03 << (2 * pinpos))) | ((uint32_t)(GPIO_PuPd << (2 *
pinpos)));
	}
}

void TM_GPIO_Lock(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin) {
	uint32_t d;

	/* Set GPIO pin with 16th bit set to 1 */
	d = 0x00010000 | GPIO_Pin;

	/* Write to LCKR register */
	GPIOx->LCKR = d;
	GPIOx->LCKR = GPIO_Pin;
	GPIOx->LCKR = d;

	/* Read twice */
	(void)GPIOx->LCKR;
	(void)GPIOx->LCKR;
}

uint16_t TM_GPIO_GetPinSource(uint16_t GPIO_Pin) {
	uint16_t pinsource = 0;
```

```
	/* Get pinsource */
	while (GPIO_Pin > 1) {
		/* Increase pinsource */
		pinsource++;
		/* Shift right */
		GPIO_Pin >>= 1;
	}

	/* Return source */
	return pinsource;
}

uint16_t TM_GPIO_GetPortSource(GPIO_TypeDef* GPIOx) {
	/* Get port source number */
	/* Offset from GPIOA		Difference between 2 GPIO addresses */
	return ((uint32_t)GPIOx - (GPIOA_BASE)) / ((GPIOB_BASE) - (GPIOA_BASE));
}

/* Private functions */
void TM_GPIO_INT_EnableClock(GPIO_TypeDef* GPIOx) {
	/* Set bit according to the 1 << portsourcenumber */
	RCC->AHB1ENR |= (1 << TM_GPIO_GetPortSource(GPIOx));
}

void TM_GPIO_INT_DisableClock(GPIO_TypeDef* GPIOx) {
	/* Clear bit according to the 1 << portsourcenumber */
	RCC->AHB1ENR &= ~(1 << TM_GPIO_GetPortSource(GPIOx));
}

void TM_GPIO_INT_Init(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, TM_GPIO_Mode_t GPIO_Mode,
TM_GPIO_OType_t GPIO_OType, TM_GPIO_PuPd_t GPIO_PuPd, TM_GPIO_Speed_t GPIO_Speed) {
	uint8_t pinpos;
	uint8_t ptr = TM_GPIO_GetPortSource(GPIOx);

	/* Go through all pins */
	for (pinpos = 0; pinpos < 0x10; pinpos++) {
		/* Check if pin available */
		if ((GPIO_Pin & (1 << pinpos)) == 0) {
			continue;
		}

		/* Pin is used */
		GPIO_UsedPins[ptr] |= 1 << pinpos;

		/* Set GPIO PUPD register */
		GPIOx->PUPDR = (GPIOx->PUPDR & ~(0x03 << (2 * pinpos))) | ((uint32_t)(GPIO_PuPd << (2 *
pinpos)));

		/* Set GPIO MODE register */
		GPIOx->MODER = (GPIOx->MODER & ~((uint32_t)(0x03 << (2 * pinpos)))) |
((uint32_t)(GPIO_Mode << (2 * pinpos)));

		/* Set only if output or alternate functions */
		if (GPIO_Mode == TM_GPIO_Mode_OUT || GPIO_Mode == TM_GPIO_Mode_AF) {
			/* Set GPIO OTYPE register */
			GPIOx->OTYPER = (GPIOx->OTYPER & ~(uint16_t)(0x01 << pinpos)) |
((uint16_t)(GPIO_OType << pinpos));

			/* Set GPIO OSPEED register */
			GPIOx->OSPEEDR = (GPIOx->OSPEEDR & ~((uint32_t)(0x03 << (2 * pinpos)))) |
((uint32_t)(GPIO_Speed << (2 * pinpos)));
		}
	}
}

uint16_t TM_GPIO_GetUsedPins(GPIO_TypeDef* GPIOx) {
	/* Return used */
	return GPIO_UsedPins[TM_GPIO_GetPortSource(GPIOx)];
}

uint16_t TM_GPIO_GetFreePins(GPIO_TypeDef* GPIOx) {
	/* Return free pins */
	return ~GPIO_UsedPins[TM_GPIO_GetPortSource(GPIOx)];
}
```

tm_stm32f4_gpio.h

```
/**
 * @author  Tilen Majerle
 * @email   tilen@majerle.eu
 * @website http://stm32f4-discovery.com
 * @link    http://stm32f4-discovery.com/2015/03/library-53-gpio-for-stm32f4
 * @version v1.5
 * @ide     Keil uVision
 * @license GNU GPL v3
 * @brief   GPIO Library for STM32F4xx devices
 *
@verbatim
   ----------------------------------------------------------------------
    Copyright (C) Tilen Majerle, 2015

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
   ----------------------------------------------------------------------
@endverbatim
 */
#ifndef TM_GPIO_H
#define TM_GPIO_H 150

/* C++ detection */
#ifdef __cplusplus
extern "C" {
#endif

/**
 * @addtogroup TM_STM32F4xx_Libraries
 * @{
 */

/**
 * @defgroup TM_GPIO
 * @brief    TM GPIO Library for STM32F4xx - http://stm32f4-discovery.com/2015/03/library-53-gpio-for-stm32f4
 * @{
 *
 * GPIO library can be used for GPIO pins.
 *
 * It features fast initialization methods as well pin input/output methods.
 *
 * It can be used as replacement for STD/HAL drivers GPIO library.
 *
 * \par Changelog
 *
@verbatim
 Version 1.5
  - June 10 2015
  - Added 2 new functions for getting used GPIO pins

 Version 1.4
  - April 28, 2015
  - Added support for PORT locking

 Version 1.3
  - March 23, 2015
  - Totally independent from HAL / SPD drivers
  - Library can be used with any drivers or totally itself

 Version 1.2
  - March 10, 2015
  - Added functions TM_GPIO_SetPinAsInput and TM_GPIO_SetPinAsOutput
```

```
  - Added functions TM_GPIO_GetPortSource and TM_GPIO_GetPinSource
0
 Version 1.1
  - March 09, 2015
  - Added function to deinit pin. Pin is set to analog input which allows lowest current consumption

 Version 1.0
  - March 08, 2015
  - Initial release
@endverbatim
 *
 * \par Dependencies
 *
@verbatim
 - STM32F4xx
 - STM32F4xx GPIO
 - defines.h
@endverbatim
 */

#include "stm32f4xx.h"
#include "stm32f4xx_gpio.h"
#include "defines.h"

/**
 * @defgroup TM_GPIO_Macros
 * @brief    GPIO Library macros
 * @{
 */

/**
 * @brief GPIO Pins declarations
 * @note  For HAL drivers compatibility
 */

#ifndef GPIO_PIN_0
#define GPIO_PIN_0                      ((uint16_t)0x0001)
#define GPIO_PIN_1                      ((uint16_t)0x0002)
#define GPIO_PIN_2                      ((uint16_t)0x0004)
#define GPIO_PIN_3                      ((uint16_t)0x0008)
#define GPIO_PIN_4                      ((uint16_t)0x0010)
#define GPIO_PIN_5                      ((uint16_t)0x0020)
#define GPIO_PIN_6                      ((uint16_t)0x0040)
#define GPIO_PIN_7                      ((uint16_t)0x0080)
#define GPIO_PIN_8                      ((uint16_t)0x0100)
#define GPIO_PIN_9                      ((uint16_t)0x0200)
#define GPIO_PIN_10                     ((uint16_t)0x0400)
#define GPIO_PIN_11                     ((uint16_t)0x0800)
#define GPIO_PIN_12                     ((uint16_t)0x1000)
#define GPIO_PIN_13                     ((uint16_t)0x2000)
#define GPIO_PIN_14                     ((uint16_t)0x4000)
#define GPIO_PIN_15                     ((uint16_t)0x8000)
#define GPIO_PIN_ALL          ((uint16_t)0xFFFF)
#endif

/**
 * @brief GPIO Pins declarations
 * @note  For STD Periph drivers compatibility
 */

#ifndef GPIO_Pin_0
#define GPIO_Pin_0                      ((uint16_t)0x0001)
#define GPIO_Pin_1                      ((uint16_t)0x0002)
#define GPIO_Pin_2                      ((uint16_t)0x0004)
#define GPIO_Pin_3                      ((uint16_t)0x0008)
#define GPIO_Pin_4                      ((uint16_t)0x0010)
#define GPIO_Pin_5                      ((uint16_t)0x0020)
#define GPIO_Pin_6                      ((uint16_t)0x0040)
#define GPIO_Pin_7                      ((uint16_t)0x0080)
#define GPIO_Pin_8                      ((uint16_t)0x0100)
#define GPIO_Pin_9                      ((uint16_t)0x0200)
#define GPIO_Pin_10                     ((uint16_t)0x0400)
#define GPIO_Pin_11                     ((uint16_t)0x0800)
#define GPIO_Pin_12                     ((uint16_t)0x1000)
#define GPIO_Pin_13                     ((uint16_t)0x2000)
#define GPIO_Pin_14                     ((uint16_t)0x4000)
```

```
#define GPIO_Pin_15                 ((uint16_t)0x8000)
#define GPIO_Pin_All        ((uint16_t)0xFFFF)
#endif

/**
 * @}
 */

/**
 * @defgroup TM_GPIO_Typedefs
 * @brief    GPIO Typedefs used for GPIO library for initialization purposes
 * @{
 */

/**
 * @brief GPIO Mode enumeration
 */
typedef enum {
        TM_GPIO_Mode_IN = 0x00,  /*!< GPIO Pin as General Purpose Input */
        TM_GPIO_Mode_OUT = 0x01, /*!< GPIO Pin as General Purpose Output */
        TM_GPIO_Mode_AF = 0x02,  /*!< GPIO Pin as Alternate Function */
        TM_GPIO_Mode_AN = 0x03,  /*!< GPIO Pin as Analog */
} TM_GPIO_Mode_t;

/**
 * @brief GPIO Output type enumeration
 */
typedef enum {
        TM_GPIO_OType_PP = 0x00, /*!< GPIO Output Type Push-Pull */
        TM_GPIO_OType_OD = 0x01  /*!< GPIO Output Type Open-Drain */
} TM_GPIO_OType_t;

/**
 * @brief  GPIO Speed enumeration
 */
typedef enum {
        TM_GPIO_Speed_Low = 0x00,    /*!< GPIO Speed Low */
        TM_GPIO_Speed_Medium = 0x01, /*!< GPIO Speed Medium */
        TM_GPIO_Speed_Fast = 0x02,   /*!< GPIO Speed Fast */
        TM_GPIO_Speed_High = 0x03    /*!< GPIO Speed High */
} TM_GPIO_Speed_t;

/**
 * @brief GPIO pull resistors enumeration
 */
typedef enum {
        TM_GPIO_PuPd_NOPULL = 0x00, /*!< No pull resistor */
        TM_GPIO_PuPd_UP = 0x01,     /*!< Pull up resistor enabled */
        TM_GPIO_PuPd_DOWN = 0x02    /*!< Pull down resistor enabled */
} TM_GPIO_PuPd_t;

/**
 * @} TM_GPIO_Typedefs
 */

/**
 * @defgroup TM_GPIO_Functions
 * @brief    GPIO Functions
 * @{
 */

/**
 * @brief  Initializes GPIO pins(s)
 * @note   This function also enables clock for GPIO port
 * @param  GPIOx: Pointer to GPIOx port you will use for initialization
 * @param  GPIO_Pin: GPIO pin(s) you will use for initialization
 * @param  GPIO_Mode: Select GPIO mode. This parameter can be a value of @ref TM_GPIO_Mode_t enumeration
 * @param  GPIO_OType: Select GPIO Output type. This parameter can be a value of @ref TM_GPIO_OType_t
enumeration
 * @param  GPIO_PuPd: Select GPIO pull resistor. This parameter can be a value of @ref TM_GPIO_PuPd_t
enumeration
 * @param  GPIO_Speed: Select GPIO speed. This parameter can be a value of @ref TM_GPIO_Speed_t enumeration
 * @retval None
 */
void TM_GPIO_Init(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, TM_GPIO_Mode_t GPIO_Mode, TM_GPIO_OType_t
GPIO_OType, TM_GPIO_PuPd_t GPIO_PuPd, TM_GPIO_Speed_t GPIO_Speed);
```

```
/**
 * @brief  Initializes GPIO pins(s) as alternate function
 * @note   This function also enables clock for GPIO port
 * @param  GPIOx: Pointer to GPIOx port you will use for initialization
 * @param  GPIO_Pin: GPIO pin(s) you will use for initialization
 * @param  GPIO_OType: Select GPIO Output type. This parameter can be a value of @ref TM_GPIO_OType_t
enumeration
 * @param  GPIO_PuPd: Select GPIO pull resistor. This parameter can be a value of @ref TM_GPIO_PuPd_t
enumeration
 * @param  GPIO_Speed: Select GPIO speed. This parameter can be a value of @ref TM_GPIO_Speed_t enumeration
 * @param  Alternate: Alternate function you will use
 * @retval None
 */
void TM_GPIO_InitAlternate(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, TM_GPIO_OType_t GPIO_OType,
TM_GPIO_PuPd_t GPIO_PuPd, TM_GPIO_Speed_t GPIO_Speed, uint8_t Alternate);

/**
 * @brief  Deinitializes pin(s)
 * @note   Pins(s) will be set as analog mode to get low power consumption
 * @param  GPIOx: GPIOx PORT where you want to set pin as input
 * @param  GPIO_Pin: Select GPIO pin(s). You can select more pins with | (OR) operator to set them as input
 * @retval None
 */
void TM_GPIO_DeInit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);

/**
 * @brief  Sets pin(s) as input
 * @note   Pins HAVE to be initialized first using @ref TM_GPIO_Init() or @ref TM_GPIO_InitAlternate() function
 * @note   This is just an option for fast input mode
 * @param  GPIOx: GPIOx PORT where you want to set pin as input
 * @param  GPIO_Pin: Select GPIO pin(s). You can select more pins with | (OR) operator to set them as input
 * @retval None
 */
void TM_GPIO_SetPinAsInput(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);

/**
 * @brief  Sets pin(s) as output
 * @note   Pins HAVE to be initialized first using @ref TM_GPIO_Init() or @ref TM_GPIO_InitAlternate() function
 * @note   This is just an option for fast output mode
 * @param  GPIOx: GPIOx PORT where you want to set pin as output
 * @param  GPIO_Pin: Select GPIO pin(s). You can select more pins with | (OR) operator to set them as output
 * @retval None
 */
void TM_GPIO_SetPinAsOutput(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);

/**
 * @brief  Sets pin(s) as analog
 * @note   Pins HAVE to be initialized first using @ref TM_GPIO_Init() or @ref TM_GPIO_InitAlternate() function
 * @note   This is just an option for fast analog mode
 * @param  GPIOx: GPIOx PORT where you want to set pin as analog
 * @param  GPIO_Pin: Select GPIO pin(s). You can select more pins with | (OR) operator to set them as analog
 * @retval None
 */
void TM_GPIO_SetPinAsAnalog(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);

/**
 * @brief  Sets pin(s) as alternate function
 * @note   For proper alternate function, you should first init pin using @ref TM_GPIO_InitAlternate() function.
 *         This functions is only used for changing GPIO mode
 * @param  GPIOx: GPIOx PORT where you want to set pin as alternate
 * @param  GPIO_Pin: Select GPIO pin(s). You can select more pins with | (OR) operator to set them as alternate
 * @retval None
 */
void TM_GPIO_SetPinAsAlternate(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);

/**
 * @brief  Sets pull resistor settings to GPIO pin(s)
 * @note   Pins HAVE to be initialized first using @ref TM_GPIO_Init() or @ref TM_GPIO_InitAlternate() function
 * @param  *GPIOx: GPIOx PORT where you want to select pull resistor
 * @param  GPIO_Pin: Select GPIO pin(s). You can select more pins with | (OR) operator to set them as output
 * @param  GPIO_PuPd: Pull resistor option. This parameter can be a value of @ref TM_GPIO_PuPd_t enumeration
 * @retval None
 */
void TM_GPIO_SetPullResistor(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, TM_GPIO_PuPd_t GPIO_PuPd);
```

```
/**
 * @brief  Sets pin(s) low
 * @note   Defined as macro to get maximum speed using register access
 * @param  GPIOx: GPIOx PORT where you want to set pin low
 * @param  GPIO_Pin: Select GPIO pin(s). You can select more pins with | (OR) operator to set them low
 * @retval None
 */
#define TM_GPIO_SetPinLow(GPIOx, GPIO_Pin)                          ((GPIOx)->BSRRH = (GPIO_Pin))

/**
 * @brief  Sets pin(s) high
 * @note   Defined as macro to get maximum speed using register access
 * @param  GPIOx: GPIOx PORT where you want to set pin high
 * @param  GPIO_Pin: Select GPIO pin(s). You can select more pins with | (OR) operator to set them high
 * @retval None
 */
#define TM_GPIO_SetPinHigh(GPIOx, GPIO_Pin)                         ((GPIOx)->BSRRL = (GPIO_Pin))

/**
 * @brief  Sets pin(s) value
 * @note   Defined as macro to get maximum speed using register access
 * @param  GPIOx: GPIOx PORT where you want to set pin value
 * @param  GPIO_Pin: Select GPIO pin(s). You can select more pins with | (OR) operator to set them value
 * @param  val: If parameter is 0 then pin will be low, otherwise high
 * @retval None
 */
#define TM_GPIO_SetPinValue(GPIOx, GPIO_Pin, val)       ((val) ? TM_GPIO_SetPinHigh(GPIOx, GPIO_Pin) :
TM_GPIO_SetPinLow(GPIOx, GPIO_Pin))

/**
 * @brief  Toggles pin(s)
 * @note   Defined as macro to get maximum speed using register access
 * @param  GPIOx: GPIOx PORT where you want to toggle pin value
 * @param  GPIO_Pin: Select GPIO pin(s). You can select more pins with | (OR) operator to toggle them all at a time
 * @retval None
 */
#define TM_GPIO_TogglePinValue(GPIOx, GPIO_Pin)                     ((GPIOx)->ODR ^= (GPIO_Pin))

/**
 * @brief  Sets value to entire GPIO PORT
 * @note   Defined as macro to get maximum speed using register access
 * @param  GPIOx: GPIOx PORT where you want to set value
 * @param  value: Value for GPIO OUTPUT data
 * @retval None
 */
#define TM_GPIO_SetPortValue(GPIOx, value)                          ((GPIOx)->ODR = (value))

/**
 * @brief  Gets input data bit
 * @note   Defined as macro to get maximum speed using register access
 * @param  GPIOx: GPIOx PORT where you want to read input bit value
 * @param  GPIO_Pin: GPIO pin where you want to read value
 * @retval 1 in case pin is high, or 0 if low
 */
#define TM_GPIO_GetInputPinValue(GPIOx, GPIO_Pin)     (((GPIOx)->IDR & (GPIO_Pin)) == 0 ? 0 : 1)

/**
 * @brief  Gets output data bit
 * @note   Defined as macro to get maximum speed using register access
 * @param  GPIOx: GPIOx PORT where you want to read output bit value
 * @param  GPIO_Pin: GPIO pin where you want to read value
 * @retval 1 in case pin is high, or 0 if low
 */
#define TM_GPIO_GetOutputPinValue(GPIOx, GPIO_Pin)  (((GPIOx)->ODR & (GPIO_Pin)) == 0 ? 0 : 1)

/**
 * @brief  Gets input value from entire GPIO PORT
 * @note   Defined as macro to get maximum speed using register access
 * @param  GPIOx: GPIOx PORT where you want to read input data value
 * @retval Entire PORT INPUT register
 */
#define TM_GPIO_GetPortInputValue(GPIOx)                            ((GPIOx)->IDR)

/**
 * @brief  Gets output value from entire GPIO PORT
 * @note   Defined as macro to get maximum speed using register access
```

```
 * @param  GPIOx: GPIOx PORT where you want to read output data value
 * @retval Entire PORT OUTPUT register
 */
#define TM_GPIO_GetPortOutputValue(GPIOx)                           ((GPIOx)->ODR)

/**
 * @brief  Gets port source from desired GPIOx PORT
 * @note   Meant for private use, unless you know what are you doing
 * @param  GPIOx: GPIO PORT for calculating port source
 * @retval Calculated port source for GPIO
 */
uint16_t TM_GPIO_GetPortSource(GPIO_TypeDef* GPIOx);

/**
 * @brief  Gets pin source from desired GPIO pin
 * @note   Meant for private use, unless you know what are you doing
 * @param  GPIO_Pin: GPIO pin for calculating port source
 * @retval Calculated pin source for GPIO pin
 */
uint16_t TM_GPIO_GetPinSource(uint16_t GPIO_Pin);

/**
 * @brief  Locks GPIOx register for future changes
 * @note   You are not able to config GPIO registers until new MCU reset occurs
 * @param  *GPIOx: GPIOx PORT where you want to lock config registers
 * @param  GPIO_Pin: GPIO pin(s) where you want to lock config registers
 * @retval None
 */
void TM_GPIO_Lock(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);

/**
 * @brief  Gets bit separated pins which were used at least once in library and were not deinitialized
 * @param  *GPIOx: Pointer to GPIOx peripheral where to check used GPIO pins
 * @retval Bit values for used pins
 */
uint16_t TM_GPIO_GetUsedPins(GPIO_TypeDef* GPIOx);

/**
 * @brief  Gets bit separated pins which were not used at in library or were deinitialized
 * @param  *GPIOx: Pointer to GPIOx peripheral where to check used GPIO pins
 * @retval Bit values for free pins
 */
uint16_t TM_GPIO_GetFreePins(GPIO_TypeDef* GPIOx);

/**
 * @}
 */
/**
 * @}
 */
/**
 * @}
 */

/* C++ detection */
#ifdef __cplusplus
}
#endif

#endif
```

tm_stm32f4_i2c.c

```
/**
 * |----------------------------------------------------------------------
 * | Copyright (C) Tilen Majerle, 2014
 * |
 * | This program is free software: you can redistribute it and/or modify
 * | it under the terms of the GNU General Public License as published by
 * | the Free Software Foundation, either version 3 of the License, or
 * | any later version.
 * |
 * | This program is distributed in the hope that it will be useful,
```

```
 * | but WITHOUT ANY WARRANTY; without even the implied warranty of
 * | MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * | GNU General Public License for more details.
 * |
 * | You should have received a copy of the GNU General Public License
 * | along with this program.  If not, see <http://www.gnu.org/licenses/>.
 * |----------------------------------------------------------------------
 */
#include "tm_stm32f4_i2c.h"

/* Private variables */
static uint32_t TM_I2C_Timeout;
static uint32_t TM_I2C_INT_Clocks[3] = {0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF};

/* Private defines */


/* Private functions */
void TM_I2C1_INT_InitPins(TM_I2C_PinsPack_t pinspack);
void TM_I2C2_INT_InitPins(TM_I2C_PinsPack_t pinspack);
void TM_I2C3_INT_InitPins(TM_I2C_PinsPack_t pinspack);

void TM_I2C_Init(I2C_TypeDef* I2Cx, TM_I2C_PinsPack_t pinspack, uint32_t clockSpeed) {
	I2C_InitTypeDef I2C_InitStruct;

	if (I2Cx == I2C1) {
		/* Enable clock */
		RCC->APB1ENR |= RCC_APB1ENR_I2C1EN;

		/* Enable pins */
		TM_I2C1_INT_InitPins(pinspack);

		/* Check clock, set the lowest clock your devices support on the same I2C but */
		if (clockSpeed < TM_I2C_INT_Clocks[0]) {
			TM_I2C_INT_Clocks[0] = clockSpeed;
		}

		/* Set values */
		I2C_InitStruct.I2C_ClockSpeed = TM_I2C_INT_Clocks[0];
		I2C_InitStruct.I2C_AcknowledgedAddress = TM_I2C1_ACKNOWLEDGED_ADDRESS;
		I2C_InitStruct.I2C_Mode = TM_I2C1_MODE;
		I2C_InitStruct.I2C_OwnAddress1 = TM_I2C1_OWN_ADDRESS;
		I2C_InitStruct.I2C_Ack = TM_I2C1_ACK;
		I2C_InitStruct.I2C_DutyCycle = TM_I2C1_DUTY_CYCLE;
	} else if (I2Cx == I2C2) {
		/* Enable clock */
		RCC->APB1ENR |= RCC_APB1ENR_I2C2EN;

		/* Enable pins */
		TM_I2C2_INT_InitPins(pinspack);

		/* Check clock, set the lowest clock your devices support on the same I2C but */
		if (clockSpeed < TM_I2C_INT_Clocks[1]) {
			TM_I2C_INT_Clocks[1] = clockSpeed;
		}

		/* Set values */
		I2C_InitStruct.I2C_ClockSpeed = TM_I2C_INT_Clocks[1];
		I2C_InitStruct.I2C_AcknowledgedAddress = TM_I2C2_ACKNOWLEDGED_ADDRESS;
		I2C_InitStruct.I2C_Mode = TM_I2C2_MODE;
		I2C_InitStruct.I2C_OwnAddress1 = TM_I2C2_OWN_ADDRESS;
		I2C_InitStruct.I2C_Ack = TM_I2C2_ACK;
		I2C_InitStruct.I2C_DutyCycle = TM_I2C2_DUTY_CYCLE;
	} else if (I2Cx == I2C3) {
		/* Enable clock */
		RCC->APB1ENR |= RCC_APB1ENR_I2C3EN;

		/* Enable pins */
		TM_I2C3_INT_InitPins(pinspack);

		/* Check clock, set the lowest clock your devices support on the same I2C but */
		if (clockSpeed < TM_I2C_INT_Clocks[2]) {
			TM_I2C_INT_Clocks[2] = clockSpeed;
		}

		/* Set values */
```

```
		I2C_InitStruct.I2C_ClockSpeed = TM_I2C_INT_Clocks[2];
		I2C_InitStruct.I2C_AcknowledgedAddress = TM_I2C3_ACKNOWLEDGED_ADDRESS;
		I2C_InitStruct.I2C_Mode = TM_I2C3_MODE;
		I2C_InitStruct.I2C_OwnAddress1 = TM_I2C3_OWN_ADDRESS;
		I2C_InitStruct.I2C_Ack = TM_I2C3_ACK;
		I2C_InitStruct.I2C_DutyCycle = TM_I2C3_DUTY_CYCLE;
	}

	/* Disable I2C first */
	I2Cx->CR1 &= ~I2C_CR1_PE;

	/* Initialize I2C */
	I2C_Init(I2Cx, &I2C_InitStruct);

	/* Enable I2C */
	I2Cx->CR1 |= I2C_CR1_PE;
}

uint8_t TM_I2C_Read(I2C_TypeDef* I2Cx, uint8_t address, uint8_t reg) {
	uint8_t received_data;
	TM_I2C_Start(I2Cx, address, I2C_TRANSMITTER_MODE, I2C_ACK_DISABLE);
	TM_I2C_WriteData(I2Cx, reg);
	TM_I2C_Stop(I2Cx);
	TM_I2C_Start(I2Cx, address, I2C_RECEIVER_MODE, I2C_ACK_DISABLE);
	received_data = TM_I2C_ReadNack(I2Cx);
	return received_data;
}


void TM_I2C_ReadMulti(I2C_TypeDef* I2Cx, uint8_t address, uint8_t reg, uint8_t* data, uint16_t count) {
	uint8_t i;
	TM_I2C_Start(I2Cx, address, I2C_TRANSMITTER_MODE, I2C_ACK_ENABLE);
	TM_I2C_WriteData(I2Cx, reg);
	//TM_I2C_Stop(I2Cx);
	TM_I2C_Start(I2Cx, address, I2C_RECEIVER_MODE, I2C_ACK_ENABLE);
	for (i = 0; i < count; i++) {
		if (i == (count - 1)) {
			/* Last byte */
			data[i] = TM_I2C_ReadNack(I2Cx);
		} else {
			data[i] = TM_I2C_ReadAck(I2Cx);
		}
	}
}

uint8_t TM_I2C_ReadNoRegister(I2C_TypeDef* I2Cx, uint8_t address) {
	uint8_t data;
	TM_I2C_Start(I2Cx, address, I2C_RECEIVER_MODE, I2C_ACK_ENABLE);
	/* Also stop condition happens */
	data = TM_I2C_ReadNack(I2Cx);
	return data;
}

void TM_I2C_ReadMultiNoRegister(I2C_TypeDef* I2Cx, uint8_t address, uint8_t* data, uint16_t count) {
	uint8_t i;
	TM_I2C_Start(I2Cx, address, I2C_RECEIVER_MODE, I2C_ACK_ENABLE);
	for (i = 0; i < count; i++) {
		if (i == (count - 1)) {
			/* Last byte */
			data[i] = TM_I2C_ReadNack(I2Cx);
		} else {
			data[i] = TM_I2C_ReadAck(I2Cx);
		}
	}
}



void TM_I2C_Write(I2C_TypeDef* I2Cx, uint8_t address, uint8_t reg, uint8_t data) {
	TM_I2C_Start(I2Cx, address, I2C_TRANSMITTER_MODE, I2C_ACK_DISABLE);
	TM_I2C_WriteData(I2Cx, reg);
	TM_I2C_WriteData(I2Cx, data);
	TM_I2C_Stop(I2Cx);
}

void TM_I2C_WriteMulti(I2C_TypeDef* I2Cx, uint8_t address, uint8_t reg, uint8_t* data, uint16_t count) {
```

```
	uint8_t i;
	TM_I2C_Start(I2Cx, address, I2C_TRANSMITTER_MODE, I2C_ACK_DISABLE);
	TM_I2C_WriteData(I2Cx, reg);
	for (i = 0; i < count; i++) {
		TM_I2C_WriteData(I2Cx, data[i]);
	}
	TM_I2C_Stop(I2Cx);
}

void TM_I2C_WriteNoRegister(I2C_TypeDef* I2Cx, uint8_t address, uint8_t data) {
	TM_I2C_Start(I2Cx, address, I2C_TRANSMITTER_MODE, I2C_ACK_DISABLE);
	TM_I2C_WriteData(I2Cx, data);
	TM_I2C_Stop(I2Cx);
}

void TM_I2C_WriteMultiNoRegister(I2C_TypeDef* I2Cx, uint8_t address, uint8_t* data, uint16_t count) {
	uint8_t i;
	TM_I2C_Start(I2Cx, address, I2C_TRANSMITTER_MODE, I2C_ACK_DISABLE);
	for (i = 0; i < count; i++) {
		TM_I2C_WriteData(I2Cx, data[i]);
	}
	TM_I2C_Stop(I2Cx);
}

/* Private functions */
int16_t TM_I2C_Start(I2C_TypeDef* I2Cx, uint8_t address, uint8_t direction, uint8_t ack) {
	/* Generate I2C start pulse */
	I2Cx->CR1 |= I2C_CR1_START;

	/* Wait till I2C is busy */
	TM_I2C_Timeout = TM_I2C_TIMEOUT;
	while (!(I2Cx->SR1 & I2C_SR1_SB)) {
		if (--TM_I2C_Timeout == 0x00) {
			return 1;
		}
	}

	/* Enable ack if we select it */
	if (ack) {
		I2Cx->CR1 |= I2C_CR1_ACK;
	}

	/* Send write/read bit */
	if (direction == I2C_TRANSMITTER_MODE) {
		/* Send address with zero last bit */
		I2Cx->DR = address & ~I2C_OAR1_ADD0;

		/* Wait till finished */
		TM_I2C_Timeout = TM_I2C_TIMEOUT;
		while (!(I2Cx->SR1 & I2C_SR1_ADDR)) {
			if (--TM_I2C_Timeout == 0x00) {
				return 1;
			}
		}
	}
	if (direction == I2C_RECEIVER_MODE) {
		/* Send address with 1 last bit */
		I2Cx->DR = address | I2C_OAR1_ADD0;

		/* Wait till finished */
		TM_I2C_Timeout = TM_I2C_TIMEOUT;
		while (!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED)) {
			if (--TM_I2C_Timeout == 0x00) {
				return 1;
			}
		}
	}

	/* Read status register to clear ADDR flag */
	I2Cx->SR2;

	/* Return 0, everything ok */
	return 0;
}

void TM_I2C_WriteData(I2C_TypeDef* I2Cx, uint8_t data) {
```

```
	/* Wait till I2C is not busy anymore */
	TM_I2C_Timeout = TM_I2C_TIMEOUT;
	while (!(I2Cx->SR1 & I2C_SR1_TXE) && TM_I2C_Timeout) {
		TM_I2C_Timeout--;
	}

	/* Send I2C data */
	I2Cx->DR = data;
}

uint8_t TM_I2C_ReadAck(I2C_TypeDef* I2Cx) {
	uint8_t data;

	/* Enable ACK */
	I2Cx->CR1 |= I2C_CR1_ACK;

	/* Wait till not received */
	TM_I2C_Timeout = TM_I2C_TIMEOUT;
	while (!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_BYTE_RECEIVED)) {
		if (--TM_I2C_Timeout == 0x00) {
			return 1;
		}
	}

	/* Read data */
	data = I2Cx->DR;

	/* Return data */
	return data;
}

uint8_t TM_I2C_ReadNack(I2C_TypeDef* I2Cx) {
	uint8_t data;

	/* Disable ACK */
	I2Cx->CR1 &= ~I2C_CR1_ACK;

	/* Generate stop */
	I2Cx->CR1 |= I2C_CR1_STOP;

	/* Wait till received */
	TM_I2C_Timeout = TM_I2C_TIMEOUT;
	while (!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_BYTE_RECEIVED)) {
		if (--TM_I2C_Timeout == 0x00) {
			return 1;
		}
	}

	/* Read data */
	data = I2Cx->DR;

	/* Return data */
	return data;
}

uint8_t TM_I2C_Stop(I2C_TypeDef* I2Cx) {
	/* Wait till transmitter not empty */
	TM_I2C_Timeout = TM_I2C_TIMEOUT;
	while (((!(I2Cx->SR1 & I2C_SR1_TXE)) || (!(I2Cx->SR1 & I2C_SR1_BTF)))) {
		if (--TM_I2C_Timeout == 0x00) {
			return 1;
		}
	}

	/* Generate stop */
	I2Cx->CR1 |= I2C_CR1_STOP;

	/* Return 0, everything ok */
	return 0;
}

uint8_t TM_I2C_IsDeviceConnected(I2C_TypeDef* I2Cx, uint8_t address) {
	uint8_t connected = 0;
	/* Try to start, function will return 0 in case device will send ACK */
	if (!TM_I2C_Start(I2Cx, address, I2C_TRANSMITTER_MODE, I2C_ACK_ENABLE)) {
		connected = 1;
```

```
	}

	/* STOP I2C */
	TM_I2C_Stop(I2Cx);

	/* Return status */
	return connected;
}

__weak void TM_I2C_InitCustomPinsCallback(I2C_TypeDef* I2Cx, uint16_t AlternateFunction) {
	/* Custom user function. */
	/* In case user needs functionality for custom pins, this function should be declared outside this library */
}

/* Private functions */
void TM_I2C1_INT_InitPins(TM_I2C_PinsPack_t pinspack) {
	/* Init pins */
#if defined(GPIOB)
	if (pinspack == TM_I2C_PinsPack_1) {
		TM_GPIO_InitAlternate(GPIOB, GPIO_PIN_6 | GPIO_PIN_7, TM_GPIO_OType_OD,
TM_GPIO_PuPd_UP, TM_GPIO_Speed_Medium, GPIO_AF_I2C1);
	}
#endif
#if defined(GPIOB)
	if (pinspack == TM_I2C_PinsPack_2) {
		TM_GPIO_InitAlternate(GPIOB, GPIO_PIN_8 | GPIO_PIN_9, TM_GPIO_OType_OD,
TM_GPIO_PuPd_UP, TM_GPIO_Speed_Medium, GPIO_AF_I2C1);
	}
#endif
#if defined(GPIOB)
	if (pinspack == TM_I2C_PinsPack_3) {
		TM_GPIO_InitAlternate(GPIOB, GPIO_PIN_6 | GPIO_PIN_9, TM_GPIO_OType_OD,
TM_GPIO_PuPd_UP, TM_GPIO_Speed_Medium, GPIO_AF_I2C1);
	}
#endif
	if (pinspack == TM_I2C_PinsPack_Custom) {
		/* Init custom pins, callback function */
		TM_I2C_InitCustomPinsCallback(I2C1, GPIO_AF_I2C1);
	}
}

void TM_I2C2_INT_InitPins(TM_I2C_PinsPack_t pinspack) {
	/* Init pins */
#if defined(GPIOB)
	if (pinspack == TM_I2C_PinsPack_1) {
		TM_GPIO_InitAlternate(GPIOB, GPIO_PIN_10 | GPIO_PIN_11, TM_GPIO_OType_OD,
TM_GPIO_PuPd_UP, TM_GPIO_Speed_Medium, GPIO_AF_I2C2);
	}
#endif
#if defined(GPIOF)
	if (pinspack == TM_I2C_PinsPack_2) {
		TM_GPIO_InitAlternate(GPIOF, GPIO_PIN_0 | GPIO_PIN_1, TM_GPIO_OType_OD,
TM_GPIO_PuPd_UP, TM_GPIO_Speed_Medium, GPIO_AF_I2C2);
	}
#endif
#if defined(GPIOH)
	if (pinspack == TM_I2C_PinsPack_3) {
		TM_GPIO_InitAlternate(GPIOH, GPIO_PIN_4 | GPIO_PIN_5, TM_GPIO_OType_OD,
TM_GPIO_PuPd_UP, TM_GPIO_Speed_Medium, GPIO_AF_I2C2);
	}
#endif
	if (pinspack == TM_I2C_PinsPack_Custom) {
		/* Init custom pins, callback function */
		TM_I2C_InitCustomPinsCallback(I2C2, GPIO_AF_I2C2);
	}
}

void TM_I2C3_INT_InitPins(TM_I2C_PinsPack_t pinspack) {
	/* Init pins */
#if defined(GPIOA) && defined(GPIOC)
	if (pinspack == TM_I2C_PinsPack_1) {
		TM_GPIO_InitAlternate(GPIOA, GPIO_PIN_8, TM_GPIO_OType_OD, TM_GPIO_PuPd_UP,
TM_GPIO_Speed_Medium, GPIO_AF_I2C3);
		TM_GPIO_InitAlternate(GPIOC, GPIO_PIN_9, TM_GPIO_OType_OD, TM_GPIO_PuPd_UP,
TM_GPIO_Speed_Medium, GPIO_AF_I2C3);
	}
```

```
#endif
#if defined(GPIOH)
		if (pinspack == TM_I2C_PinsPack_2) {
			TM_GPIO_InitAlternate(GPIOH, GPIO_PIN_7 | GPIO_PIN_8, TM_GPIO_OType_OD,
TM_GPIO_PuPd_UP, TM_GPIO_Speed_Medium, GPIO_AF_I2C3);
		}
#endif
		if (pinspack == TM_I2C_PinsPack_Custom) {
			/* Init custom pins, callback function */
			TM_I2C_InitCustomPinsCallback(I2C3, GPIO_AF_I2C3);
		}
}
```

tm_stm32f4_i2c.h

```
/**
 * @author  Tilen Majerle
 * @email   tilen@majerle.eu
 * @website http://stm32f4-discovery.com
 * @link    http://stm32f4-discovery.com/2014/05/library-09-i2c-for-stm32f4xx/
 * @version v1.6.1
 * @ide     Keil uVision
 * @license GNU GPL v3
 * @brief   I2C library for STM32F4xx
 *
@verbatim
   ----------------------------------------------------------------------
    Copyright (C) Tilen Majerle, 2015

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
   ----------------------------------------------------------------------
@endverbatim
 */
#ifndef TM_I2C_H
#define TM_I2C_H 161
/**
 * @addtogroup TM_STM32F4xx_Libraries
 * @{
 */

/**
 * @defgroup TM_I2C
 * @brief    I2C library for STM32F4xx - http://stm32f4-discovery.com/2014/05/library-09-i2c-for-stm32f4xx/
 * @{
 *
 *            \par Pinout
 *
@verbatim
      |PINSPACK 1   |PINSPACK 2   |PINSPACK 3
I2CX   |SCL   SDA    |SCL   SDA    |SCL   SDA
      |            |            |
I2C1   |PB6   PB7    |PB8   PB9    |PB6   PB9
I2C2   |PB10  PB11   |PF1   PF0    |PH4   PH5
I2C3   |PA8   PC9    |PH7   PH8    |-     -
@endverbatim
 *
 * \par Custom pinout
 *
 * In case these pins are not good for you, you can use
 * @ref TM_I2C_PinsPack_Custom in @ref TM_I2C_Init() function and callback function will be called,
 * where you can initialize your custom pinout for your I2C peripheral
 *
```

```
 * Possible changes in your defines.h file:
 * Change x to your I2C used, 1-3
 *
@verbatim
//By default library support only 7bit long address
#define TM_I2Cx_ACKNOWLEDGED_ADDRESS   I2C_AcknowledgedAddress_7bit
//Library supports I2C mode
#define TM_I2Cx_MODE                   I2C_Mode_I2C
//Own address, if slave mode
#define TM_I2Cx_OWN_ADDRESS            0x00
//By default, disable ack
#define TM_I2Cx_ACK                    I2C_Ack_Disable
//Duty cycle 2, 50%
#define TM_I2Cx_DUTY_CYCLE             I2C_DutyCycle_2
@endverbatim
 *
 * \par Changelog
 *
@verbatim
 Version 1.6.1
  - March 31, 2015
  - Fixed I2C issue when sometime it didn't send data

 Version 1.6
  - March 13, 2015
  - Added new function to write multi bytes to device without specify register address

 Version 1.5
  - March 10, 2015
  - Updated to be more independent of HAL/STD drivers.

 Version 1.4
  - March 08, 2015
  - Added support for new GPIO settings

 Version 1.3
  - December 22, 2014
  - Added option to read multi bytes from device without setting register from where

 Version 1.2
  - August 14, 2014
  - If you connect more devices on one I2C with different max SCL speed, low speed will be always selected.
  - Added some additional pins for I2C

 Version 1.1
  - September 08, 2014
  - Added support to check if device is connected to I2C bus

 Version 1.0
  - First release
@endverbatim
 *
 * \par Dependencies
 *
@verbatim
 - STM32F4xx
 - STM32F4xx I2C
 - defines.h
 - attributes.h
 - TM GPIO
@endverbatim
 */
#include "stm32f4xx.h"
#include "stm32f4xx_i2c.h"
#include "attributes.h"
#include "defines.h"
#include "tm_stm32f4_gpio.h"

/**
 * @defgroup TM_I2C_Macros
 * @brief    Library defines
 * @{
 */

/**
 * @brief  Timeout for I2C
```

```
 */
#ifndef TM_I2C_TIMEOUT
#define TM_I2C_TIMEOUT                          20000
#endif

/* I2C1 settings, change them in defines.h project file */
#ifndef TM_I2C1_ACKNOWLEDGED_ADDRESS
#define TM_I2C1_ACKNOWLEDGED_ADDRESS            I2C_AcknowledgedAddress_7bit
#endif
#ifndef TM_I2C1_MODE
#define TM_I2C1_MODE                            I2C_Mode_I2C
#endif
#ifndef TM_I2C1_OWN_ADDRESS
#define TM_I2C1_OWN_ADDRESS                     0x00
#endif
#ifndef TM_I2C1_ACK
#define TM_I2C1_ACK                                     I2C_Ack_Disable
#endif
#ifndef TM_I2C1_DUTY_CYCLE
#define TM_I2C1_DUTY_CYCLE                      I2C_DutyCycle_2
#endif

/* I2C2 settings, change them in defines.h project file */
#ifndef TM_I2C2_ACKNOWLEDGED_ADDRESS
#define TM_I2C2_ACKNOWLEDGED_ADDRESS            I2C_AcknowledgedAddress_7bit
#endif
#ifndef TM_I2C2_MODE
#define TM_I2C2_MODE                            I2C_Mode_I2C
#endif
#ifndef TM_I2C2_OWN_ADDRESS
#define TM_I2C2_OWN_ADDRESS                     0x00
#endif
#ifndef TM_I2C2_ACK
#define TM_I2C2_ACK                                     I2C_Ack_Disable
#endif
#ifndef TM_I2C2_DUTY_CYCLE
#define TM_I2C2_DUTY_CYCLE                      I2C_DutyCycle_2
#endif

/* I2C3 settings, change them in defines.h project file */
#ifndef TM_I2C3_ACKNOWLEDGED_ADDRESS
#define TM_I2C3_ACKNOWLEDGED_ADDRESS            I2C_AcknowledgedAddress_7bit
#endif
#ifndef TM_I2C3_MODE
#define TM_I2C3_MODE                            I2C_Mode_I2C
#endif
#ifndef TM_I2C3_OWN_ADDRESS
#define TM_I2C3_OWN_ADDRESS                     0x00
#endif
#ifndef TM_I2C3_ACK
#define TM_I2C3_ACK                                     I2C_Ack_Disable
#endif
#ifndef TM_I2C3_DUTY_CYCLE
#define TM_I2C3_DUTY_CYCLE                      I2C_DutyCycle_2
#endif

#define TM_I2C_CLOCK_STANDARD                   100000  /*!< I2C Standard speed */
#define TM_I2C_CLOCK_FAST_MODE                  400000  /*!< I2C Fast mode speed */
#define TM_I2C_CLOCK_FAST_MODE_PLUS             1000000 /*!< I2C Fast mode plus speed */
#define TM_I2C_CLOCK_HIGH_SPEED                 3400000 /*!< I2C High speed */

#define I2C_TRANSMITTER_MODE   0
#define I2C_RECEIVER_MODE      1
#define I2C_ACK_ENABLE         1
#define I2C_ACK_DISABLE        0

 /**
 * @}
 */

/**
 * @defgroup TM_I2C_Typedefs
 * @brief    Library Typedefs
 * @{
 */
```

```
/**
 * @brief  I2C pinspack enumeration
 */
typedef enum {
		TM_I2C_PinsPack_1,	/*!< Use Pinspack1 from Pinout table for I2Cx */
		TM_I2C_PinsPack_2,	/*!< Use Pinspack2 from Pinout table for I2Cx */
		TM_I2C_PinsPack_3,	/*!< Use Pinspack3 from Pinout table for I2Cx */
		TM_I2C_PinsPack_Custom  /*!< Use custom pins for I2Cx */
} TM_I2C_PinsPack_t;

/**
 * @}
 */

/**
 * @defgroup TM_I2C_Functions
 * @brief    Library Functions
 * @{
 */

/**
 * @brief  Initializes I2C
 * @param  *I2Cx: I2C used
 * @param  pinspack: Pins used. This parameter can have a value of @ref TM_I2C_PinsPack_t enumeration
 * @param  clockSpeed: Clock speed for SCL in Hertz
 * @retval None
 */
void TM_I2C_Init(I2C_TypeDef* I2Cx, TM_I2C_PinsPack_t pinspack, uint32_t clockSpeed);

/**
 * @brief  Reads single byte from slave
 * @param  *I2Cx: I2C used
 * @param  address: 7 bit slave address, left aligned, bits 7:1 are used, LSB bit is not used
 * @param  reg: register to read from
 * @retval Data from slave
 */
uint8_t TM_I2C_Read(I2C_TypeDef* I2Cx, uint8_t address, uint8_t reg);

/**
 * @brief  Reads multi bytes from slave
 * @param  *I2Cx: I2C used
 * @param  uint8_t address: 7 bit slave address, left aligned, bits 7:1 are used, LSB bit is not used
 * @param  uint8_t reg: register to read from
 * @param  uint8_t *data: pointer to data array to store data from slave
 * @param  uint8_t count: how many bytes will be read
 * @retval None
 */
void TM_I2C_ReadMulti(I2C_TypeDef* I2Cx, uint8_t address, uint8_t reg, uint8_t *data, uint16_t count);

/**
 * @brief  Reads byte from slave without specify register address
 * @param  *I2Cx: I2C used
 * @param  address: 7 bit slave address, left aligned, bits 7:1 are used, LSB bit is not used
 * @retval Data from slave
 */
uint8_t TM_I2C_ReadNoRegister(I2C_TypeDef* I2Cx, uint8_t address);

/**
 * @brief  Reads multi bytes from slave without setting register from where to start read
 * @param  *I2Cx: I2C used
 * @param  address: 7 bit slave address, left aligned, bits 7:1 are used, LSB bit is not used
 * @param  *data: pointer to data array to store data from slave
 * @param  count: how many bytes will be read
 * @retval None
 */
void TM_I2C_ReadMultiNoRegister(I2C_TypeDef* I2Cx, uint8_t address, uint8_t* data, uint16_t count);

/**
 * @brief  Writes single byte to slave
 * @param  *I2Cx: I2C used
 * @param  address: 7 bit slave address, left aligned, bits 7:1 are used, LSB bit is not used
 * @param  reg: register to write to
 * @param  data: data to be written
 * @retval None
 */
void TM_I2C_Write(I2C_TypeDef* I2Cx, uint8_t address, uint8_t reg, uint8_t data);
```

```
/**
 * @brief  Writes multi bytes to slave
 * @param  *I2Cx: I2C used
 * @param  address: 7 bit slave address, left aligned, bits 7:1 are used, LSB bit is not used
 * @param  reg: register to write to
 * @param  *data: pointer to data array to write it to slave
 * @param  count: how many bytes will be written
 * @retval None
 */
void TM_I2C_WriteMulti(I2C_TypeDef* I2Cx, uint8_t address, uint8_t reg, uint8_t *data, uint16_t count);

/**
 * @brief  Writes byte to slave without specify register address
 *
 *         Useful if you have I2C device to read like that:
 *           - I2C START
 *           - SEND DEVICE ADDRESS
 *           - SEND DATA BYTE
 *           - I2C STOP
 *
 * @param  *I2Cx: I2C used
 * @param  address: 7 bit slave address, left aligned, bits 7:1 are used, LSB bit is not used
 * @param  data: data byte which will be send to device
 * @retval None
 */
void TM_I2C_WriteNoRegister(I2C_TypeDef* I2Cx, uint8_t address, uint8_t data);

/**
 * @brief  Writes multi bytes to slave without setting register from where to start write
 * @param  *I2Cx: I2C used
 * @param  address: 7 bit slave address, left aligned, bits 7:1 are used, LSB bit is not used
 * @param  *data: pointer to data array to write data to slave
 * @param  count: how many bytes you want to write
 * @retval None
 */
void TM_I2C_WriteMultiNoRegister(I2C_TypeDef* I2Cx, uint8_t address, uint8_t* data, uint16_t count);

/**
 * @brief  Checks if device is connected to I2C bus
 * @param  *I2Cx: I2C used
 * @param  address: 7 bit slave address, left aligned, bits 7:1 are used, LSB bit is not used
 * @retval Device status:
 *           - 0: Device is not connected
 *           - > 0: Device is connected
 */
uint8_t TM_I2C_IsDeviceConnected(I2C_TypeDef* I2Cx, uint8_t address);

/**
 * @brief  I2C Start condition
 * @param  *I2Cx: I2C used
 * @param  address: slave address
 * @param  direction: master to slave or slave to master
 * @param  ack: ack enabled or disabled
 * @retval Start condition status
 * @note   For private use
 */
int16_t TM_I2C_Start(I2C_TypeDef* I2Cx, uint8_t address, uint8_t direction, uint8_t ack);

/**
 * @brief  Stop condition on I2C
 * @param  *I2Cx: I2C used
 * @retval Stop condition status
 * @note   For private use
 */
uint8_t TM_I2C_Stop(I2C_TypeDef* I2Cx);

/**
 * @brief  Reads byte without ack
 * @param  *I2Cx: I2C used
 * @retval Byte from slave
 * @note   For private use
 */
uint8_t TM_I2C_ReadNack(I2C_TypeDef* I2Cx);

/**
```

```
 * @brief  Reads byte with ack
 * @param  *I2Cx: I2C used
 * @retval Byte from slave
 * @note   For private use
 */
uint8_t TM_I2C_ReadAck(I2C_TypeDef* I2Cx);

/**
 * @brief  Writes to slave
 * @param  *I2Cx: I2C used
 * @param  data: data to be sent
 * @retval None
 * @note   For private use
 */
void TM_I2C_WriteData(I2C_TypeDef* I2Cx, uint8_t data);

/**
 * @brief  Callback for custom pins initialization.
 *
 *         When you call TM_I2C_Init() function, and if you pass TM_I2C_PinsPack_Custom to function,
 *         then this function will be called where you can initialize custom pins for I2C peripheral.
 * @param  *I2Cx: I2C for which initialization will be set
 * @param  AlternateFunction: Alternate function which should be used for GPIO initialization
 * @retval None
 * @note   With __weak parameter to prevent link errors if not defined by user
 */
void TM_I2C_InitCustomPinsCallback(I2C_TypeDef* I2Cx, uint16_t AlternateFunction);

/**
 * @}
 */

/**
 * @}
 */

/**
 * @}
 */

#endif
```

Recursos

[1]. Entorno de trabajo para STM32F4
http://timor.atollic.com/resources/downloads/

[2]. Ejemplos STM32F4XX
https://github.com/MaJerle/stm32f429

[3]. Nunchuck y STM32F4
https://acassis.wordpress.com/2015/06/26/using-the-nintendo-wii-nunchuk-joystick-in-the-stm32f4discovery-running-nuttx/

[4]. Librerías bus I2C para STM32F4 Discovery
http://stm32f4-discovery.net/2014/05/library-09-i2c-for-stm32f4xx/

[5]. Librerías PWM para STM32F4 Discovery
http://stm32f4-discovery.com/2014/09/library-33-pwm-stm32f4xx/

[6]. Librería Servo para STM32F4 Discovery
http://stm32f4-discovery.net/2014/10/library-42-control-rc-servo-stm32f4/