uc3m

Departamento Tecnología Electrónica

dte UC3M

Tema 12: Funciones Especiales y Desarrollo de Proyectos

Sistemas Digitales Basados en Microprocesadores

Universidad Carlos III de Madrid

Dpto. Tecnología Electrónica

Nota: Las figuras utilizadas para ilustrar las características y funcionalidades del microcontrolador del curso se han obtenido de la documentación técnica disponible en https://www.st.com/en/microcontrollers-microprocessors/stm32l151-152.html

Índice

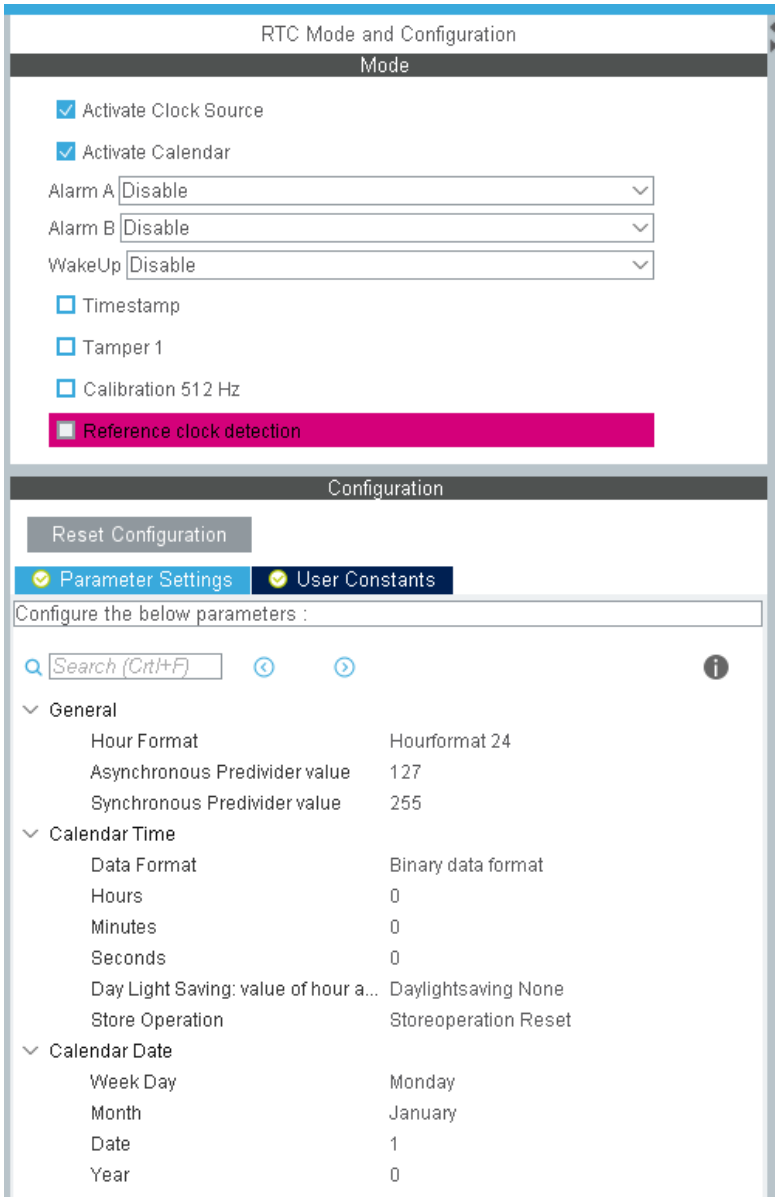
Características Especiales: Reloj en Tiempo Real (RTC)

Características y Funcionamiento

- El Reloj en Tiempo Real (RTC) es un dispositivo que permite medir el tiempo para mantener un calendario y un reloj
  - Evitando el uso de un temporizador para esta función.
- El del STM32L152 ha sido diseñado para un consumo mínimo, pensado para sistemas alimentados por batería
  - Proporciona segundos, minutos, horas, día del mes, mes, año y día de la semana
  - Posee un divisor de reloj para que se ajuste a distintas frecuencias de oscilador
    - Aunque por defecto está configurado para ser usado a través del LSE
  - No tiene ningún mecanismo para mantener la hora tras una pérdida de energía
  - Tiene un mecanismo de protección para evitar modificaciones no voluntarias de los registros de control
  - Contiene funcionalidades más avanzadas que no se van a ver en este curso

Guía de Funcionamiento con HAL

- En CubeMX:
  - Activa la fuente de reloj
  - Activa el Calendario
  - Decide si los datos se utilizan en Binario o en BCD
  - Pueden escogerse los valores iniciales de la hora y de la fecha
  - Si se selecciona una alarma, se da la opción de activar la interrupción
- En Programación:
  - Inicialización:
    - HAL_RTC_Init()
    - Se necesita crear una variable de tipo RTC_TimeTypeDef y otra RTC_DateTypeDef, que tendrán la estructura para la hora y la fecha



Guía de Funcionamiento con HAL

- En Programación (cont.):
  - Fecha y Hora:
    - Parámetros: h – handler; d – datos; f – formato (RTC_FORMAT_BIN, RTC_FORMAT_BCD)
    - HAL_RTC_SetTime(h, d, f)
    - HAL_RTC_SetDate(h, d, f)
    - HAL_RTC_GetTime(h, d, f)
    - HAL_RTC_GetDate(h, d, f)
  - Alarma:
    - Se necesita crear una variable tipo RTC_AlarmTypeDef para tener la estructura con la información de la alarma
    - Parámetros: h – handler; a – alarma; f – formato (RTC_FORMAT_BIN, RTC_FORMAT_BCD); w – timeout
    - HAL_RTC_SetAlarm(h, a, f)
    - HAL_RTC_SetAlarm_IT(h, a, f)
    - HAL_RTC_GetAlarm(h, a, f)
    - HAL_RTC_DeactivateAlarm(h)
    - HAL_RTC_PollForAlarmAEvent (h, w)

Ejemplo: Configuración y Uso RTC con HAL

```
/* USER CODE BEGIN 1 */
RTC_TimeTypeDef my_time;
RTC_DateTypeDef my_date;
uint8_t text[7];
uint16_t number;
/* USER CODE END 1 */
```

```
/* USER CODE BEGIN 2 */
my_date.Month = 3;
my_date.Date = 24;
my_date.WeekDay = RTC_WEEKDAY_SUNDAY;
my_date.Year = 19;
my_time.DayLightSaving = RTC_DAYLIGHTSAVING_NONE;
my_time.Hours = 23;
my_time.Minutes = 10;
my_time.Seconds = 0;
my_time.TimeFormat = RTC_HOURFORMAT_24;
HAL_RTC_SetDate(&hrtc, &my_date, RTC_FORMAT_BIN);
HAL_RTC_SetTime(&hrtc, &my_time, RTC_FORMAT_BIN);
BSP_LCD_GLASS_Init();
BSP_LCD_GLASS_BarLevelConfig(0);
BSP_LCD_GLASS_Clear();
/* USER CODE END 2 */
```

```
/* USER CODE BEGIN WHILE */
while (1) {
  HAL_RTC_GetTime(&hrtc, &my_time, RTC_FORMAT_BIN);
  number = 0;
  number = (my_time.Hours * 10000) + (my_time.Minutes * 100) + my_time.Seconds;
  Bin2Ascii((unsigned short)number, text);
  BSP_LCD_GLASS_DisplayString((uint8_t *)text);
  HAL_Delay(1000);
  HAL_RTC_GetDate(&hrtc, &my_date, RTC_FORMAT_BIN);
  number = 0;
  number = (my_date.Date * 10000) + (my_date.Month * 100) + my_date.Year;
  Bin2Ascii((unsigned short)number, text);
  BSP_LCD_GLASS_DisplayString((uint8_t *)text);
  HAL_Delay(1000);
  /* USER CODE END WHILE */
```

Guía de Funcionamiento con Registros

- Procedimiento de inicialización:
  - Se desprotegen los registros de control (**RTC→WPR**)
    - Se escribe primero 0xCA y seguidamente 0x53
    - Escribir cualquier cosa vuelve a bloquear los registros
    - El desbloqueo se mantiene
  - Se pone a 1 el bit INIT (**RTC→ISR**)
  - Se espera a que se ponga a 1 el bit INITF (**RTC→ISR**)
  - Se programa el divisor de reloj, tanto síncrono como asíncrono (**RTC→PRER**)
    - Se hace en dos pasos. Primero el síncrono (16 lsb) y luego el asíncrono (16msb)
    - Para el uso con el LSE se ponen de valores 255 para el síncrono y 127 para el asíncrono
  - Se escribe el valor de la hora inicial (**RTC→TR**)
  - Se escribe el valor de la fecha inicial (**RTC→DR**)
  - Se selecciona el modo de 12 o 24 horas en el bit FMT (**RTC→CR**)
  - Se pone a 0 el bit INIT (**RTC→ISR**)
  - Se protegen los registros (**RTC→WPR** = 0)
- Procedimiento de consulta de la hora:
  - Se leen en cualquier momento los registros **RTC→TR** para la hora y **RTC→DR** para la fecha.

RTC: Registros de Control

- **RTC→CR** – Control Register:
  - Sólo se va a utilizar el bit 6 (FMT) para indicar si es formato 24h (con un '0') o si es formato 12h (con un '1').
  - El resto de bits se deja a '0'

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								COE	OSEL[1:0]		POL	Reserv-ed	BKP	SUB1H	ADD1H
								rw	rw	rw	rw		rw	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TSIE	WUTIE	ALRBIE	ALRAIE	TSE	WUTE	ALRBE	ALRAE	DCE	FMT	Reser ved	REFCKON	TSEDGE	WUCKSEL[2:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw

- **RTC->PRER** – Prescaler Register:
  - Configura el preescalado asíncrono (PREDIV_A) y el síncrono (PREDIV-S)
  - La frecuencia de funcionamiento del RTC será:

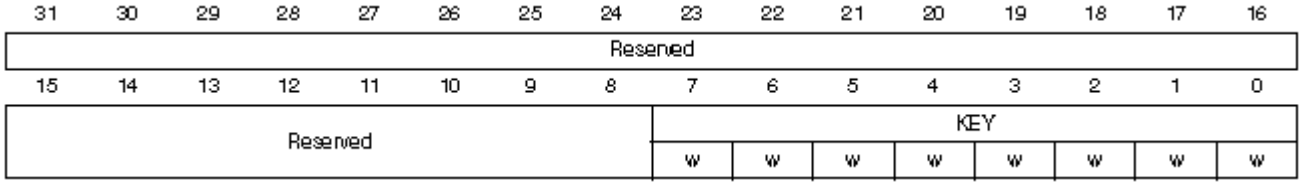
$$f_{CK\_SPRE} = \frac{f_{RTCCLK}}{(PREDIV\_S + 1) \times (PREVID\_A + 1)}$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Resened									PREDIV_A[6:0]						
									rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Resened			PREDIV_S[12:0]												
			rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

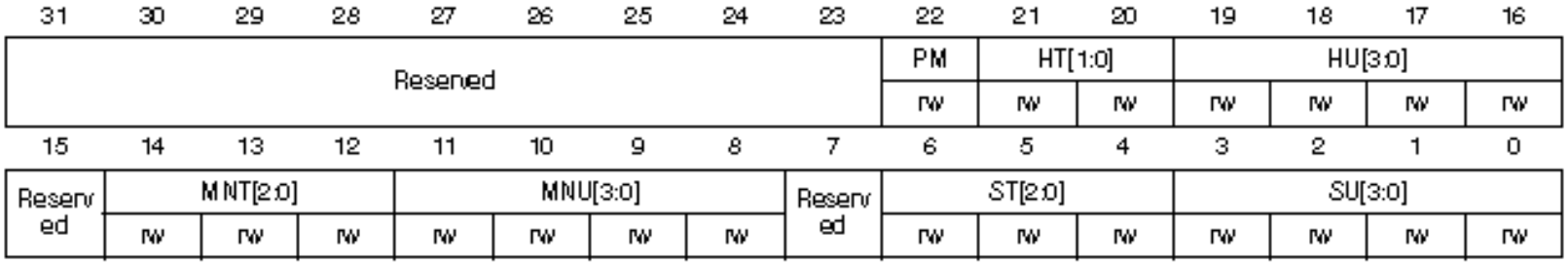
RTC: Registros de Control

- **RTC→WPR** – Write Protection Register:
  - Se utiliza como se ha comentado anteriormente



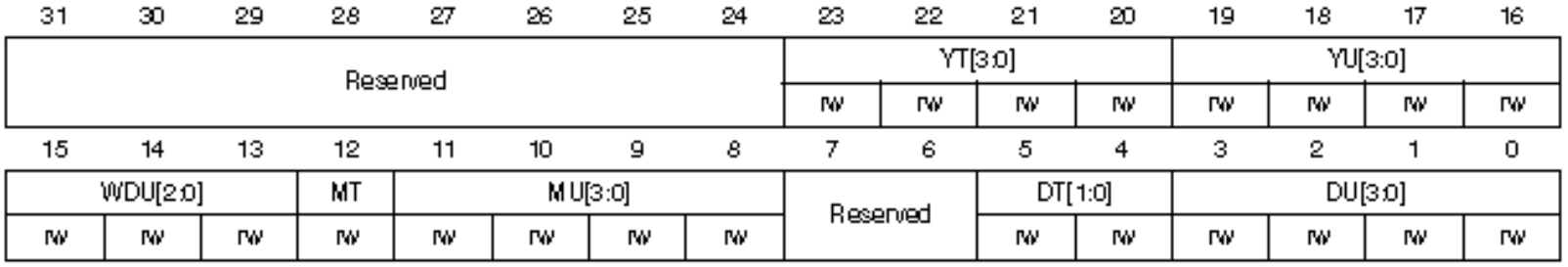
RTC: Registros de Datos

- **RTC→TR** – Time Register:
  - Contiene el valor de la hora, en la siguiente estructura (formato BCD):
    - PM: Con un 0 es formato de 24h o AM, y con un 1 es PM
    - HT: Decenas de hora
    - HU: Unidades de hora
    - MINT: Decenas de minuto
    - MINU: Unidades de minuto
    - ST: Decenas de segundo
    - SU: Unidades de segundo



RTC: Registros de Datos

- **RTC→DR** – Date Register:
  - Contiene el valor de la fecha, en la siguiente estructura (formato BCD):
    - YT: Decenas de año
    - YU: Unidades de año
    - WDU: Tres bits que indican el día de la semana (000-prohibido; 001-lunes; …; 111-domingo)
    - MT: Decenas de mes
    - MU: Unidades de mes
    - DT: Decenas de día
    - DU: Unidades de día



RTC: Registros de Estado

- **RTC→ISR** – Initialization and Status Register:
  - De todos los bits sólo interesan:
    - INIT, bit de control:
      - 0 – se quita el modo de inicialización, actualizando los nuevos valores y arrancando el RTC
      - 1 – se pone el RTC en modo inicialización
    - INITF, bit de estado:
      - 0 – el modo de inicialización está desactivado.
      - 1 – el modo de inicialización está activado y ya se puede escribir en los registros.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Resened															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	TAMP1F	TSOVF	TSF	WUTF	ALRBF	ALRAF	INIT	INITF	RSF	INITS	Res.	WUTWF	ALRBWF	ALRAWF
		rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rw	r	rc_w0	r		r	r	r

Ejemplo: Configuración y Uso RTC

```
#include "stm32l1xx.h"
#include "..\Biblioteca_SDM.h"
#include "..\Utiles_SDM.h"

int main(void){
   unsigned char cadena[7];
   unsigned valor;
   Init_SDM();
   Init_LCD();
   LCD_Limpia();

   RTC->WPR=0xCA;
   RTC->WPR=0x53;
   RTC->ISR |= (1<<7);
   while ((RTC->ISR & (1<<6))==0);
   RTC->PRER=255;
   RTC->PRER|=127<<16;
   RTC->TR = 0x00123456;
   RTC->DR = 0x00126314;
   RTC->CR = 0x00000000;
   RTC->ISR &= ~(1<<7);
   RTC->WPR=0;
   while ((RTC->ISR & (1<<6))!=0);
```

```
   while (1) {
      valor = RTC->TR;
      cadena[5]=(valor & 0x0000000F)+'0';
      valor = valor >> 4;
      cadena[4]=(valor & 0x00000007)+'0';
      valor = valor >> 4;
      cadena[3]=(valor & 0x0000000F)+'0';
      valor = valor >> 4;
      cadena[2]=(valor & 0x00000007)+'0';
      valor = valor >> 4;
      cadena[1]=(valor & 0x0000000F)+'0';
      valor = valor >> 4;
      cadena[0]=(valor & 0x00000003)+'0';
      valor = valor >> 4;
      LCD_Texto(cadena);
      espera(10000000);
      valor = RTC->DR;
      cadena[1]=(valor & 0x0000000F)+'0';
      valor = valor >> 4;
      cadena[0]=(valor & 0x00000003)+'0';
      valor = valor >> 4;
      cadena[3]=(valor & 0x0000000F)+'0';
      valor = valor >> 4;
      cadena[2]=(valor & 0x00000001)+'0';
      valor = valor >> 4;
      cadena[5]=(valor & 0x0000000F)+'0';
      valor = valor >> 4;
      cadena[4]=(valor & 0x0000000F)+'0';
      valor = valor >> 4;
      LCD_Texto(cadena);
      espera(10000000);
      }
   }
```

Características Especiales: Watchdog

Características Generales

- Un Watchdog es un mecanismo interno de control del microcontrolador, para provocar el reset del sistema en caso de que se detecte que el micro ha perdido el control
- El sistema de Watchdog tiene las siguientes características:
  - Resetea el chip internamente si no se actualiza periódicamente
    - Es decir, el programa debe, de forma periódica y antes de que pase un determinado tiempo límite, escribir en un registro del watchdog una secuencia de valores, de tal forma que le indica al watchdog que sigue teniendo el control del sistema
  - Se habilita por software, pero sólo se resetea por reset o por una interrupción de Watchdog
  - Un uso incorrecto o incompleto, provoca también el reset
  - Utiliza internamente un temporizador

Características Especiales: Consumo

Bajo Consumo

- Al crear un sistema electrónico, uno de los requisitos principales es que su consumo sea mínimo:
  - Por motivos de calificación energética
  - Cuando el sistema es portátil y ha de ser alimentado con baterías
- Todos los microcontroladores actuales tienen la posibilidad de definir distintos modos de bajo consumo:
  - Totalmente operativo (máximo consumo)
  - Determinados periféricos desconectados
  - Todos los periféricos desconectados (salvo algún PIN)
  - La mayor parte de la CPU desconectada (mínimo consumo)
- El ubicar al dispositivo en uno de los modos de bajo consumo, lo realiza el programador de la aplicación cuando se cumplen determinados requisitos
- El “despertar” al micro de un estado de bajo consumo se realiza mediante una IRQ (por ejemplo EINT)

Integración de Periféricos en un Proyecto

Pasos para el Diseño de un Proyecto

1. Analizar con detalle el problema a resolver
2. Localizar funcionalidades necesarias por parte del micro
    1. Realizar el **Diagrama de Bloques**, seleccionando los recursos (p.ej. pines) a utilizar
3. Definir la funcionalidad de cada uno de los periféricos del micro involucrados
4. Determinar la prioridad de cada uno de ellos y realizar un análisis temporal de su petición de uso por parte del micro
    1. **Determinar las IRQ a usar**, y las prioridades entre ellas
    2. Definir las RAI correspondientes
5. Diseñar la solución completa utilizando **diagramas de flujo**
6. Implementar la solución

Puntos a tener muy en cuenta

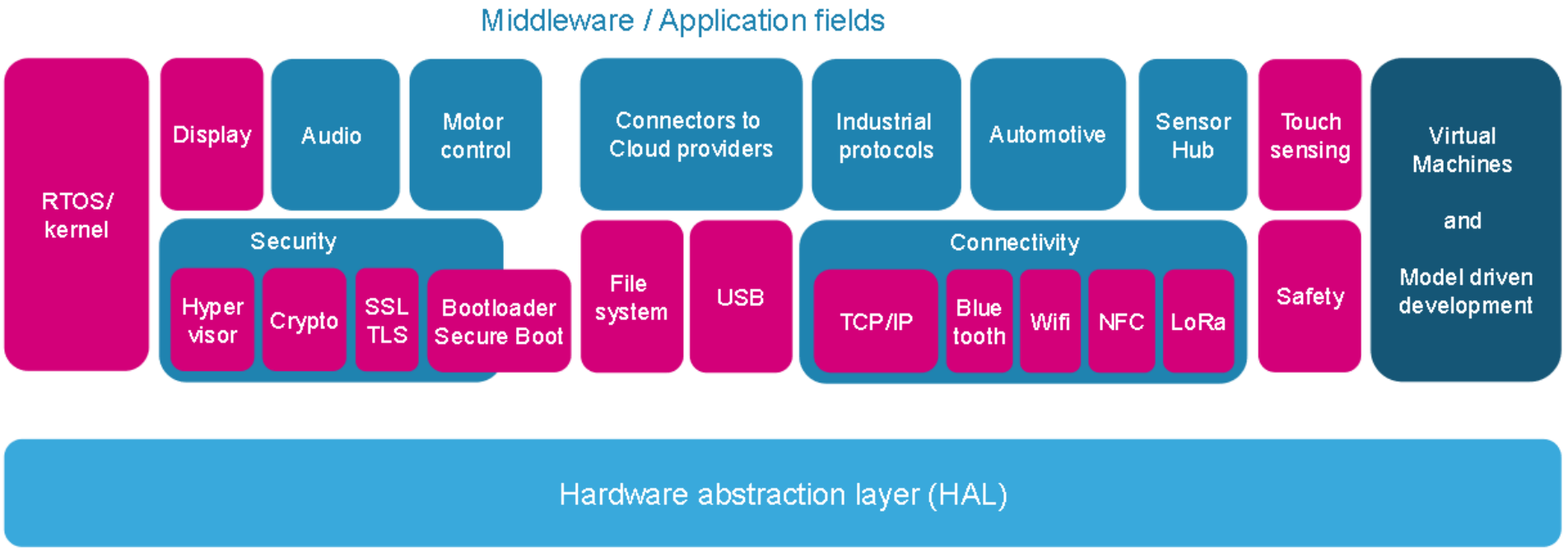
- Temporización:
  - Muchas veces hay necesidades de temporización que no están específicamente planteadas en el problema a resolver:
    - Suelen tener que ver con usabilidad (interacción con el usuario final), o con compartición del tiempo.
- Recursos comunes:
  - No sólo los pines, sino también otros periféricos
    - Especialmente los temporizadores
- Desarrollo:
  - Se recomienda siempre utilizar una estrategia de Divide y Vencerás. Dos posibilidades:
    - **Aproximación Top-Down:**
      - Se hace el programa principal con las llamadas a las distintas funciones, implementando estas como simples funciones vacías.
      - Se implementa cada una de las funciones por separado
    - **Aproximación Bottom-Up:**
      - Se desarrollan las funciones individuales
      - Posteriormente se implementa el programa principal uniendo las funciones individuales
  - Cada vez que se desarrolla una parte, es importante depurarla en profundidad, antes de que se integre con otros elementos.

Uso de APIs en Programación

Uso de APIs en Programación

- Cuando la CPU o los periféricos son excesivamente complicados, o simplemente la aplicación a desarrollar requiere de programación avanzada, suele ser necesario utilizar **Application Programming Interfaces (APIs)** para simplificar nuestra tarea
- También pueden ser conocidas como **Libraries** (traducido como Biblioteca o como Librería)
- Las APIs pueden ser de muy distinto tipo, e incluso tener nombres distintos al de API dependiendo del sector.
  - Por ejemplo, en el caso de microcontroladores, se habla de:
    - HAL – Hardware Abstraction Layer: para hablar de aquellas funciones definidas que simplifiquen la configuración y el acceso a los periféricos
      - También llamada Standard Peripheral Library
      - Muchas veces suele ser proporcionado por el mismo fabricante, pero a veces hay algunas de terceras partes
    - Middleware: que se refiere a aquellas APIs que intentan dar soporte a una funcionalidad compleja que todavía no se encuentra a nivel de aplicación final (por ejemplo, la pila de TCP/IP, el interfaz completo de USB, o un sistema de ficheros)
      - Pueden haber varias disponibles, procedentes de terceras partes
    - Language based library: Por ejemplo la C-library que incluye funciones estándar de ANSI C99, como el malloc, el espacio std, etc.

Uso de APIs en Programación



Uso de APIs en Programación

STM32 – Crypto

Provider	Solution name	Model	Cost	Availability									
				F0	F1	F2	F3	F4	F7	H7	L0	L1	L4
Cypherbridge	uVPN SDK IKE v1 /IKE v2/IPsec	Source	License	N	N	N	N	Y	Y	N[3]	N	N	N
HCC	Verifiable Encryption manager AES, 3DES, DSS, EDH, MD5, RSA, SHA1, SHA256	Source	License	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Rowebots	UNISON SSL/TLS Stack AES, Blowfish, Triple-DES (3DES), DES, ARC4, Camellia, XTEA ECB, CBC, CFB, CTR, GCM, CCM MD2, MD4, MD5, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, RIPEMD-160 ECC	Source	License	N	Y	Y	Y	Y	Y	Y	N	Y	Y
SEGGER	emSecure signatures	Source	License	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
SEGGER	emLib AES and emLib DES	Source	License	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
SEGGER	emFile encryption	Source	License	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
ST	STM32 Cryptographic library[1,2] AES, DES, 3DES, ARC4, MD5, SHA1, SHA2, RSAsig, ECC Keygen, ECDSA, ...	Binaries	Free	N	Y	Y	Y	Y	N	N	N	Y	N
ST	X-CUBE-CRYPTOLIB	Binaries	Free	Y	Y	Y	Y	Y	Y	N	Y	Y	Y
wolfSSL	wolfCrypt[1], part of wolfSSL MD2, MD4, MD5, SHA-1, SHA-256, SHA-384, SHA-512, BLAKE2b, RIPEMD-160, Poly1305 AES (CBC, CTR, GCM, CCM), Camellia, DES, 3DES, ARC4, RABBIT, HC-128, ChaCha20 RSA, DSS (DSA), DH, EDH, NTRU ECDH-ECDSA, ECDHE-ECDSA, ECDH-RSA, ECDHE-RSA	Open source (GPL2) or Source	Free or license	N	N	Y	N	Y	Y	Y	Y	Y	Y

Uso de APIs en Programación

STM32 – USB solutions (1/2)

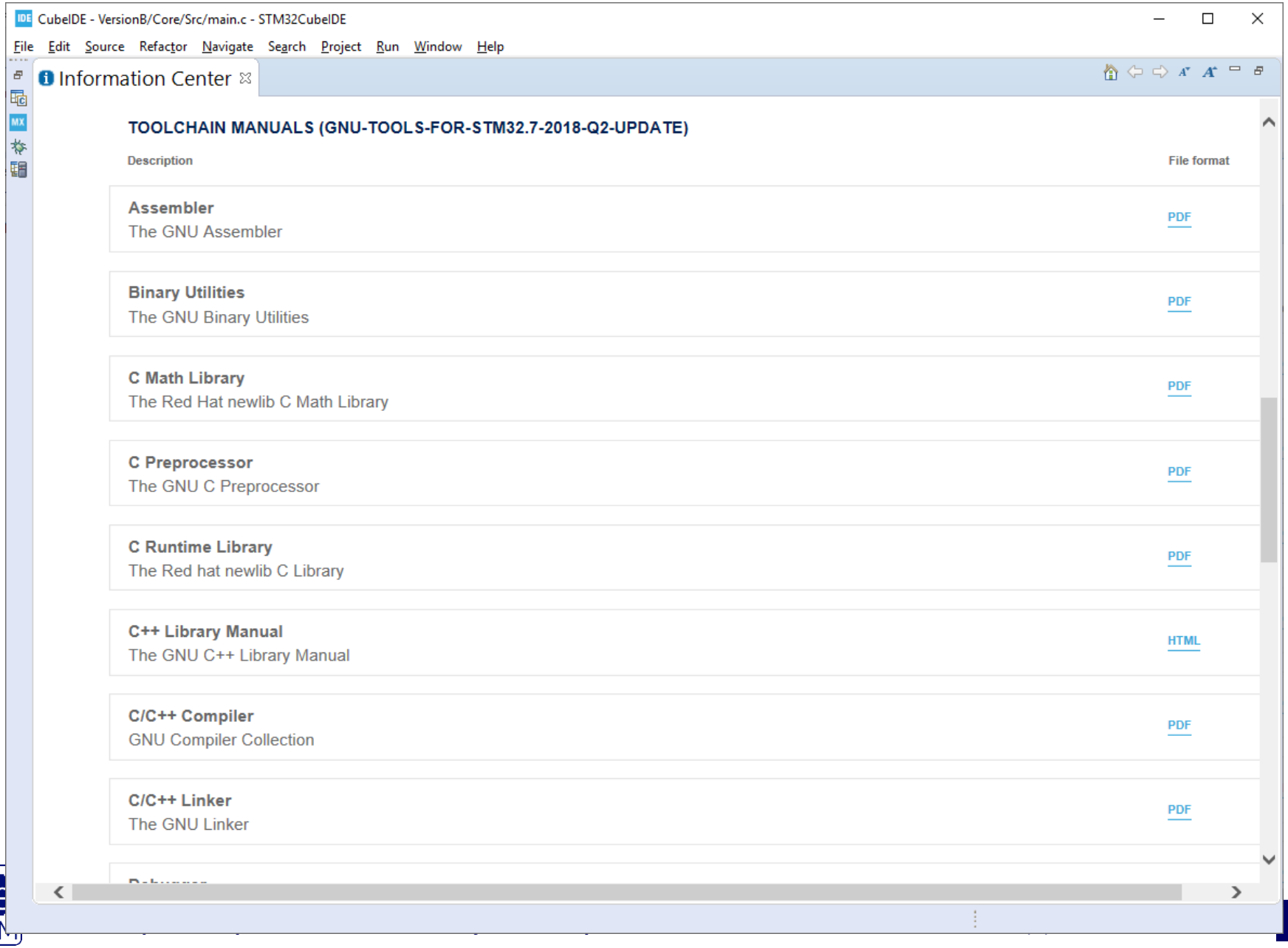
Provider	Solution name	Model	Cost	Availability									
				F0	F1	F2	F3	F4	F7	H7	L0	L1	L4
Chibios	ChibiOS/HAL	Open source (GPL3) or Source	Free or License	Y	Y	Y	Y	Y	Y	N[1]	Y	Y	Y
CMX	CMX-USB Device, Host	Source	License	Y	Y	Y	Y	Y	Y	N[1]	Y	Y	N
eCosCentric	eCosPro-Host, Device	Source	License	N	Y	Y	Y	Y	Y	N[1]	N	Y	Y
EUROS	USB Host & Device	Binaries	License	N	Y	Y	Y	Y	Y	N[1]	N	Y	Y
EmCraft	Linux USB Host	Open source (GPL)	Free	N	N	Y	N	Y	N	N[1]	N	N	N
Express Logic	USBX	Source	License	Y	Y	Y	Y	Y	Y	N[1]	Y	Y	Y
HCC	HCC-USB Host, Device	Source	License	Y	Y	Y	Y	Y	Y	N[1]	Y	Y	Y
Wittenstein - High Integrity Systems	CONNECT USB Device, USB Host	Source	License	Y	Y	Y	Y	Y	Y	N[1]	Y	Y	Y
Keil / arm	MDK-ARM USB	Source	License	Y	Y	Y	Y	Y	Y	N[1]	Y	Y	Y
Mentor Embedded	Nucleus USB	Source	License	N	Y	Y	Y	Y	Y	N[1]	N	Y	Y
Micrium	USB Host, USB Device	Source	License	Y	Y	Y	Y	Y	Y	N[1]	Y	Y	Y

Uso de APIs en Programación

STM32 – USB solutions (2/2)

Provider	Solution name	Model	Cost	Availability										
				F0	F1		F2	F3	F4	F7	H7	L0	L1	L4
					Others	F105 F107								
Micro Digital	smxUSB	Source	License	Y	Y	Y	Y	Y	Y	Y	N[1]	Y	Y	Y
Quadros	RTXCusb	Source	License	N[1]	Y	Y	Y	Y	N[1]	N[1]	N[1]	N[1]	N[1]	N[1]
Rowebots	Unison USB System	Source	License	N	Y	Y	Y	Y	Y	N	N[1]	Y	N	Y
SEGGER	emUSB Device, emUSB Host	Source	License	Y	Y	Y	Y	Y	Y	Y	N[1]	Y	Y	Y
ST	USB FS device library	Source	Free	Y	Y	N	N	Y	N	N	N[1]	N	Y	N
ST	USB FS&HS Host&Device lib	Source	Free	N	N	Y	Y	N	Y	N	N[1]	N	N	N
ST	STM32Cube – USB Host&Device	Source	Free	Y[2]	Y[2]		Y	Y[2]	Y	Y	N[1]	Y[2]	Y[2]	Y[2]
Thesycon	Embedded USB Device	Source	License	N[1]	N[1]		Y	N[1]	Y	Y	N[1]	N[1]	N[1]	N[1]
Zephyr	USB device stack	Source	Free[3]	Y	Y		N	Y	Y	N	N	N	N	Y

Bibliotecas de C y C++ en CubeIDE



La Standard Peripheral Library de STM32L1

- Estos son los módulos HAL para STM32L1:
  - misc.h
  - stm32l1xx_adc.h
  - stm32l1xx_aes.h
  - stm32l1xx_comp.h
  - stm32l1xx_crc.h
  - stm32l1xx_dac.h
  - stm32l1xx_dbgmcu.h
  - stm32l1xx_dma.h
  - stm32l1xx_exti.h
  - stm32l1xx_flash.h
  - stm32l1xx_fsmc.h
  - stm32l1xx_gpio.h
  - stm32l1xx_i2c.h
  - stm32l1xx_iwdg.h
  - stm32l1xx_lcd.h
  - stm32l1xx_opamp.h
  - stm32l1xx_pwr.h
  - stm32l1xx_rcc.h
  - stm32l1xx_rtc.h
  - stm32l1xx_sdio.h
  - stm32l1xx_spi.h
  - stm32l1xx_syscfg.h
  - stm32l1xx_tim.h
  - stm32l1xx_usart.h
  - stm32l1xx_wwdg.h