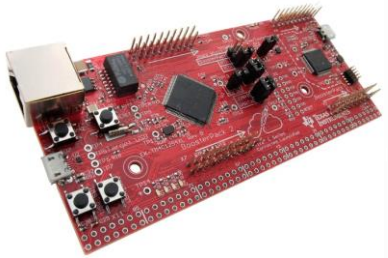


Tema 4. Periféricos del TM4C1294

4º Grado de Ingeniería en Electrónica, Robótica y Mecatrónica
Andalucía Tech

Índice

- Introducción
- Inicialización básica
- GPIO’s
- Interrupciones
- Timers / pwm
- Comunicaciones serie
- Canales analógicos
- USB
- Ethernet

Objetivos

- Mostrar el manejo de la librería de periféricos
- Aprender el funcionamiento de los periféricos básicos
- Sentar las bases para el *aprendizaje autónomo* del resto de periféricos

Introducción

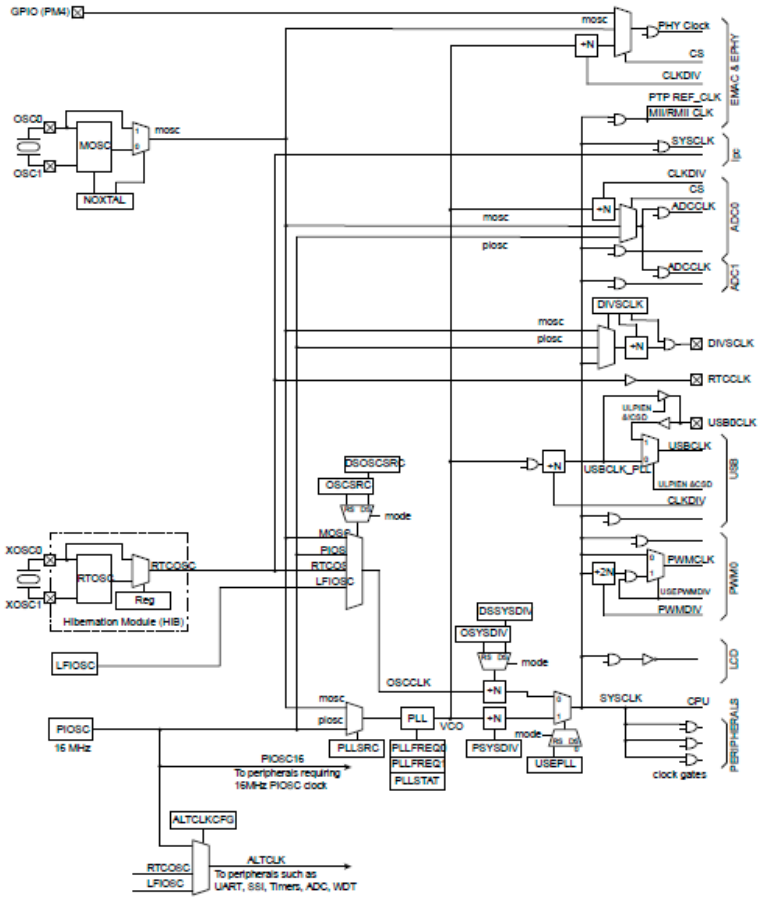
- Multitud de periféricos
  - Primarios: (gpio's, adc, spi, i2c…)
  - Secundarios (usb, ethernet)
- Manejo de la librería Driverlib.lib
- Fuentes de información
  - Ejemplos (TivaWare)
  - Workshop
  - Manual de la librería

Inicialización básica

- Al crear un nuevo proyecto, dos ficheros:
- tm4c1294ncpdt_startup_ccs.c
  - Definiciones de la tabla de vectores y reset
  - Modificar para incluir rutinas de interrupción
- main.c
  - Incluir las librerías a usar
  - Función main(void)

Inicialización del reloj

- Múltiples fuentes de reloj:
  - Oscilador interno (16MHz)
  - Osc. Principal con cristal externo
  - Osc. Interno a 30kHz (±50%)
  - Reloj para hibernación y RTC, a 32.768 Hz
  - PLL a 320 ó 480MHz, para dividir el osc. Interno o externo



Inicialización del reloj

- `uint32_t SysCtlClockFreqSet(uint32_t ui32Config, uint32_t ui32SysClock)`
  - ui32Config: configuración:
    - Elegir oscilador:
      - SYSCTL_OSC_MAIN, oscilador externo (definir frecuencia)
      - SYSCTL_OSC_INT, oscilador interno de 16MHz
      - SYSCTL_OSC_INT30, osc. Interno de 30kHz
      - SYSCTL_OSC_EXT32, osc. del módulo de hibernación
    - Usar PLL: SYSCTL_USE_PLL/ No usarlo: SYSCLT_USE_OSC
    - SYSCTL_CFG_VCO_xxx, para elegir la frecuencia del pll (480 ó 320 MHz)
  - ui32SysClock: frecuencia deseada (en Hz)
  - devuelve la frecuencia real, una vez ajustada

Inicialización del reloj

- Ejemplos:
  - Usar oscilador de 16MHz y generar el reloj interno a 40MHz con el PLL:
    - `frec=SysCtlClockFreqSet(SYSCTL_OSC_INT | SYSCTL_USE_PLL | SYSCTL_CFG_VCO_320, 40000000);`
  - Usar oscilador externo de 25MHz y generar el reloj interno a 120MHz con el PLL:
    - `frec=SysCtlClockFreqSet(SYSCTL_OSC_MAIN |SYSCLT_XTAL_25MHZ | SYSCTL_USE_PLL | SYSCTL_CFG_VCO_480, 120000000);`
  - Frecuencias posibles:
    - $F_{max}=F_{PLL}/4$
    - $F=F_{PLL}/N$, con N par>2

Pines de E/S

- 15 puertos (de PORTA a PORTQ, menos I y O)
- hasta 90 pines programables
- corriente máxima programable (2..12mA)
- Entradas Schmitt-trigger, con posibilidad de resistencia de pull up o down.
- Configurables a mano o con PinMux.

Multiplexión de pines

- Cada pin puede tener varias funciones (12)
- Los periféricos, mapeables a pines.
- Por defecto, *todos* los pines están configurados como entradas digitales
- Tabla de pin-out: pp 743-746 del manual

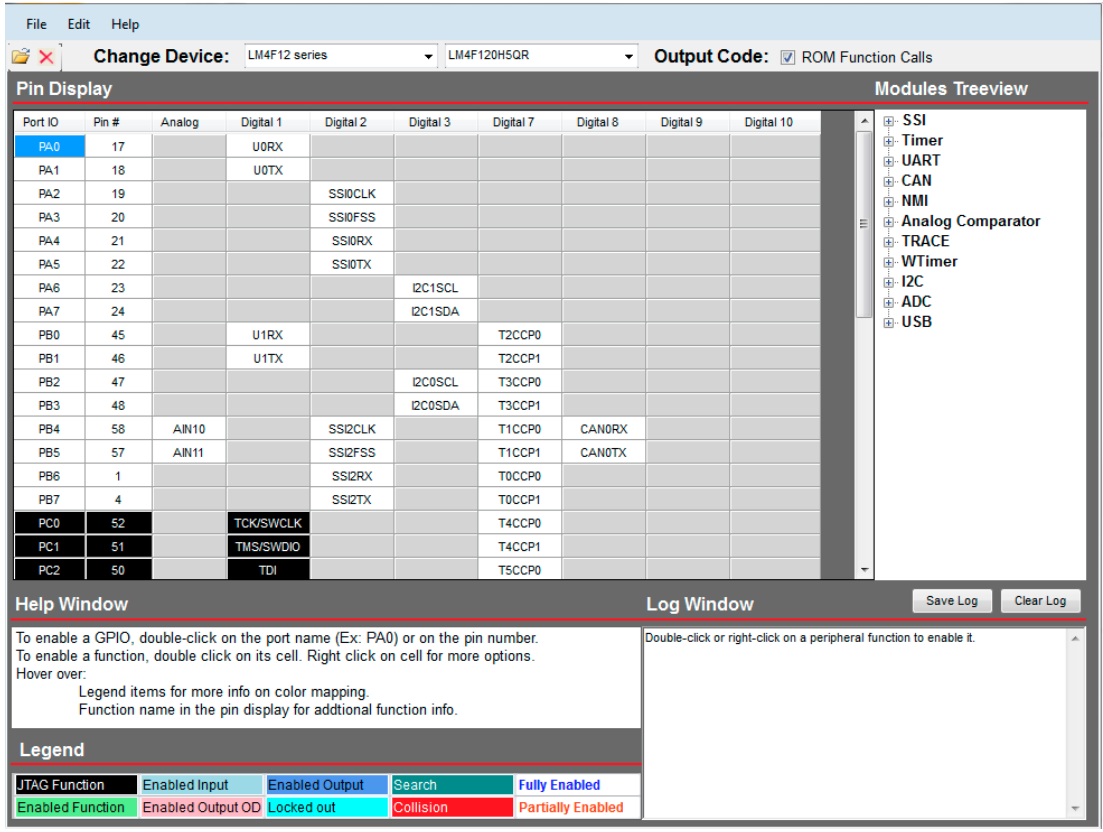
Multiplexión de pines

- Se repiten las funciones. Elegir dónde:

IO	Pin	Analog or Special Function[a]	Digital Function (GPIOPCTL PMCx Bit Field Encoding)[b]											
			1	2	3	4	5	6	7	8	11	13	14	15
PA5	38	-	U3Tx	I2C7SDA	T2CCP1	-	-	-	-	-	-	-	-	SSI0XDAT1
PA6	40	-	U2Rx	I2C6SCL	T3CCP0	-	USB0EPEN	-	-	-	-	SSI0XDAT2	-	EPI0S8
PA7	41	-	U2Tx	I2C6SDA	T3CCP1	-	USB0PFLT	-	-	-	USB0EPEN	SSI0XDAT3	-	EPI0S9
PB0	95	USB0ID	U1Rx	I2C5SCL	T4CCP0	-	-	-	CAN1Rx	-	-	-	-	-
PB1	96	USB0VBUS	U1Tx	I2C5SDA	T4CCP1	-	-	-	CAN1Tx	-	-	-	-	-
PB2	91	-	-	I2C0SCL	T5CCP0	-	-	-	-	-	-	-	USB0STP	EPI0S27
PB3	92	-	-	I2C0SDA	T5CCP1	-	-	-	-	-	-	-	USB0CLK	EPI0S28
PD4	125	AIN7	U2Rx	-	T3CCP0	-	-	-	-	-	-	-	-	SSI1XDAT2
PD5	126	AIN6	U2Tx	-	T3CCP1	-	-	-	-	-	-	-	-	SSI1XDAT3
PD6	127	AIN5	U2RTS	-	T4CCP0	-	USB0EPEN	-	-	-	-	-	-	SSI2XDAT3
PD7	128	AIN4	U2CTS	-	T4CCP1	-	USB0PFLT	-	-	NMI	-	-	-	SSI2XDAT2
PE0	15	AIN3	U1RTS	-	-	-	-	-	-	-	-	-	-	-

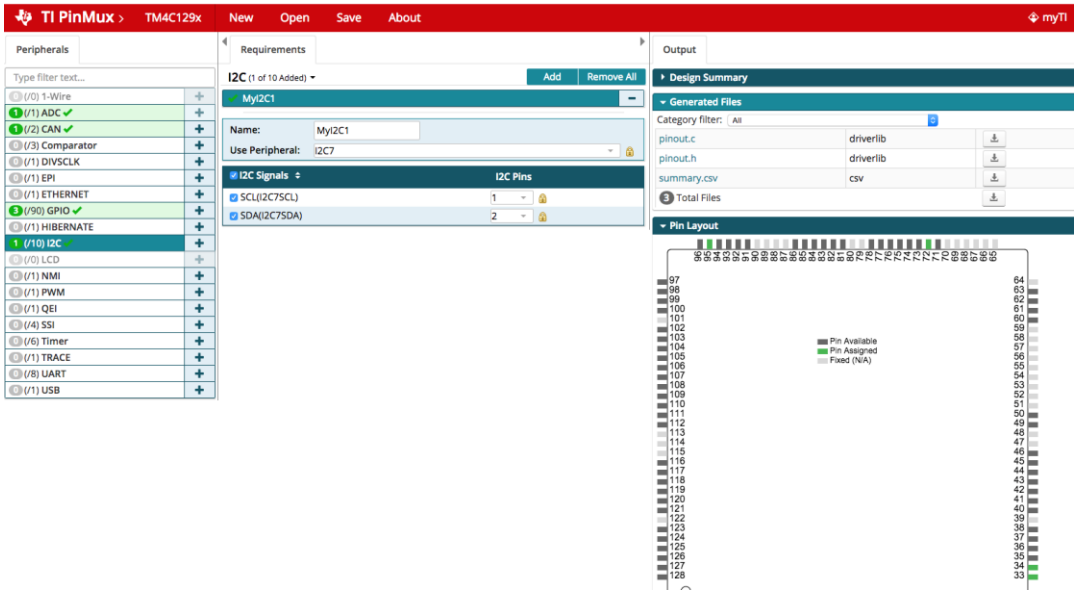
Utilidad PinMux

- Configurar *automáticamente*
- Función del pin
- Genera *.c y *.h
- Puede necesitar configuración adicional
  - Resistencias de pull up, corriente máxima…



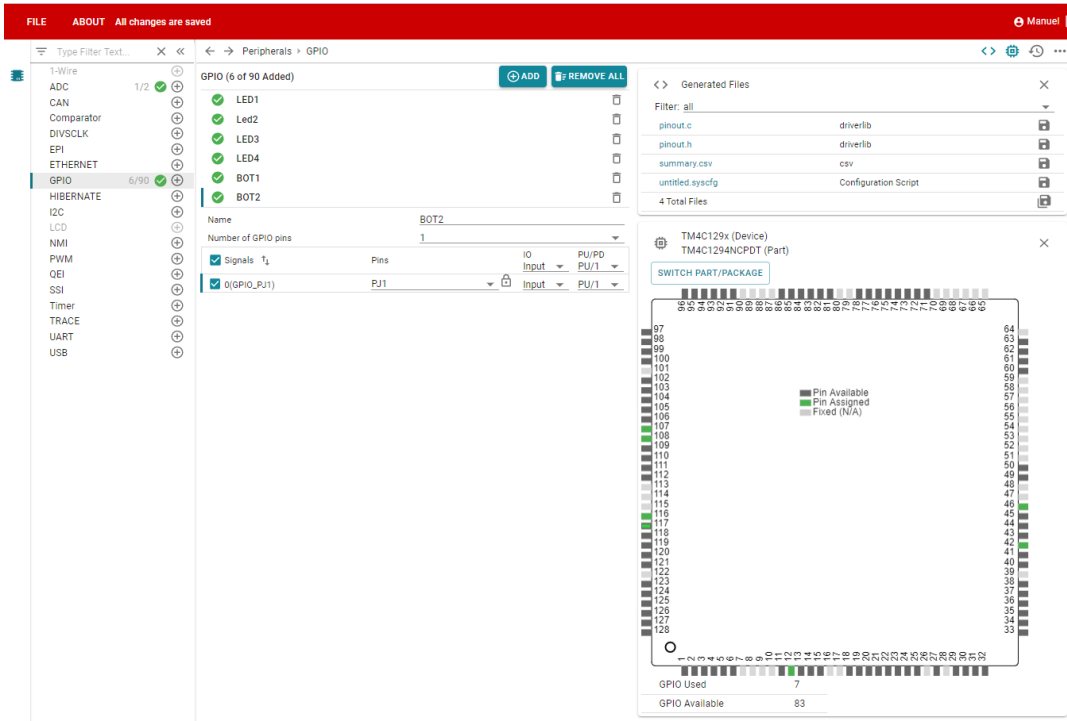
PinMux V4

- Interfaz más moderno
- Capacidades más completas
  - Pull up-down, carga de pines…
- Herramienta in-the-cloud
  - Colaboración e integración con CCS Cloud
  - Necesidad de Internet



SysConfig

- Nueva versión, misma funcionalidad
- Herramienta in-the-cloud
- Nuevos procesadores añadidos



Configuración manual

- 1.-Habilitar reloj para el periférico (puerto):
  - `SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOJ);`
- 2.-Definir pines como E/S:
  - GPIOPinTypeGPIOInput(*puerto,pines*):
    - `GPIOPinTypeGPIOInput(GPIO_PORTJ_BASE, GPIO_PIN0|GPIO_PIN1);`
  - GPIOPinTypeGPIOOutput(*puerto, pines)*
    - `GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, GPIO_PIN0|GPIO_PIN1);`
- 3.-Definir características de los pines (corriente máxima, Pull-ups…):
  - GPIOPadConfigSet(*puerto, pines, corriente, resistencia*):
    - `GPIOPadConfigSet(GPIO_PORTJ_BASE, GPIO_PIN0|GPIO_PIN1, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);`

Otras funciones de GPIO

- Tipo de pin:

```
void GPIOPinTypeADC(uint32_t ui32Port, uint8_t ui8Pins);
void GPIOPinTypeCAN(uint32_t ui32Port, uint8_t ui8Pins);
void GPIOPinTypeComparator(uint32_t ui32Port, uint8_t ui8Pins);
void GPIOPinTypeEPI(uint32_t ui32Port, uint8_t ui8Pins);
void GPIOPinTypeEthernetLED(uint32_t ui32Port, uint8_t ui8Pins);
void GPIOPinTypeEthernetMII(uint32_t ui32Port, uint8_t ui8Pins);
void GPIOPinTypeGPIOInput(uint32_t ui32Port, uint8_t ui8Pins);
void GPIOPinTypeGPIOOutput(uint32_t ui32Port, uint8_t ui8Pins);
void GPIOPinTypeGPIOOutputOD(uint32_t ui32Port, uint8_t ui8Pins);
void GPIOPinTypeI2C(uint32_t ui32Port, uint8_t ui8Pins);
void GPIOPinTypeI2CSCL(uint32_t ui32Port, uint8_t ui8Pins);
void GPIOPinTypeLCD(uint32_t ui32Port, uint8_t ui8Pins);
void GPIOPinTypePWM(uint32_t ui32Port, uint8_t ui8Pins);
void GPIOPinTypeQEI(uint32_t ui32Port, uint8_t ui8Pins);
void GPIOPinTypeSSI(uint32_t ui32Port, uint8_t ui8Pins);
void GPIOPinTypeTimer(uint32_t ui32Port, uint8_t ui8Pins);
void GPIOPinTypeUART(uint32_t ui32Port, uint8_t ui8Pins);
void GPIOPinTypeUSBAnalog(uint32_t ui32Port, uint8_t ui8Pins);
void GPIOPinTypeUSBDigital(uint32_t ui32Port, uint8_t ui8Pins);
void GPIOPinTypeWakeHigh(uint32_t ui32Port, uint8_t ui8Pins);
void GPIOPinTypeWakeLow(uint32_t ui32Port, uint8_t ui8Pins);
```

Otras funciones de GPIO

- Manejo de interrupciones:

```
void GPIOIntEnable(uint32_t ui32Port, uint32_t ui32IntFlags);
void GPIOIntDisable(uint32_t ui32Port, uint32_t ui32IntFlags);
uint32_t GPIOIntStatus(uint32_t ui32Port, bool bMasked);
void GPIOIntClear(uint32_t ui32Port, uint32_t ui32IntFlags);
void GPIOIntRegister(uint32_t ui32Port, void (*pfnIntHandler)(void));
void GPIOIntUnregister(uint32_t ui32Port);
void GPIOIntTypeSet(uint32_t ui32Port, uint8_t ui8Pins, uint32_t ui32IntType);
```

- Lectura y escritura de pines:

```
int32_t GPIOPinRead(uint32_t ui32Port, uint8_t ui8Pins);
void GPIOPinWrite(uint32_t ui32Port, uint8_t ui8Pins, uint8_t ui8Val);
```

  - OJO: para poner un pin a 1, escribir un 1 en su posición:

```
GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_4, 0);           //APAGA PF4
GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_4, GPIO_PIN_4);  //ENCIENDE PF4
```

Ejemplo 2

- Configurar los 4 pines de salida de los LED
  - PN1, PN0, PF4, PF0
- Configurar los 2 pines de entrada de los botones (con Resistencia de Pull-Up)
  - PJ0, PJ1
- Encender / apagar los led en una secuencia, al pulsar los botones:
  - 0001-0011-0111-1111-1110-1100-1000

Ejemplo 2

- Definimos una matriz con los led que se quieren encender en cada caso:

```
#define MSEC 40000 //Valor para 1ms con SysCtlDelay()
#define MaxEst 7

int LED[MaxEst][4]=
{0,0,0,1,
 0,0,1,1,
 0,1,1,1,
 1,1,1,1,
 1,1,1,0,
 1,1,0,0,
 1,0,0,0};
```

Ejemplo 2

- Inicialización:

```
 //Fijar velocidad a 120MHz
SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ | SYSCTL_OSC_MAIN | SYSCTL_USE_PLL |
SYSCTL_CFG_VCO_480),
120000000);

    //Habilitar los periféricos implicados: GPIOF, J y N (llevar el reloj)
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOJ);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);

    //Definir tipo de pines
GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_0 |GPIO_PIN_4); //F0 y F4: salidas
GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, GPIO_PIN_0 |GPIO_PIN_1); //N0 y N1: salidas
GPIOPinTypeGPIOInput(GPIO_PORTJ_BASE, GPIO_PIN_0|GPIO_PIN_1);   //J0 y J1: entradas
GPIOPadConfigSet(GPIO_PORTJ_BASE,GPIO_PIN_0|GPIO_PIN_1,GPIO_STRENGTH_2MA,
GPIO_PIN_TYPE_STD_WPU);     //Pullup en J0 y J1
```

Ejemplo 2

- Bucle principal:

```
i=0;
   while(1){
       if(!(GPIOPinRead(GPIO_PORTJ_BASE,GPIO_PIN_0))){     //Si se aprieta el boton 1
           SysCtlDelay(10*MSEC);
           while(!(GPIOPinRead(GPIO_PORTJ_BASE,GPIO_PIN_0)));  //Debouncing...
           SysCtlDelay(10*MSEC);
           i++; if(i==MaxEst) i=MaxEst-1; //Incrementa el estado. Si máximo, satura
       }
       if( !(GPIOPinRead(GPIO_PORTJ_BASE,GPIO_PIN_1))){    //Si se aprieta el botón 2
           SysCtlDelay(10*MSEC);
           while( !(GPIOPinRead(GPIO_PORTJ_BASE,GPIO_PIN_1)));   //Debouncing..
           SysCtlDelay(10*MSEC);
           i--; if(i==-1) i=0;   //Decrementa el estado. Si menor que cero, satura.
       }

       GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_1, GPIO_PIN_1*LED[i][0]);
       GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_0, GPIO_PIN_0*LED[i][1]);
       GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_4, GPIO_PIN_4*LED[i][2]);
       GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, GPIO_PIN_0*LED[i][3]);

   }
```

Manejo de interrupciones

- NVIC: Nested Vectored Interrupt Controller
- Tres niveles de habilitación/deshabilitación:
  - Global
  - Del periférico
  - De la función concreta
- Para estar seguros, habilitar las 3.

```
GPIOIntEnable(GPIO_PORTJ_BASE, GPIO_PIN_0|GPIO_PIN_1);
IntEnable(INT_GPIOJ);
IntMasterEnable();
```

Manejo de interrupciones

- La rutina de interrupción se puede situar *a mano* en el fichero ...startup_ccs.c:
- Sustituir por el nombre de la función que se desee

```
//*****************************************************************************
#pragma DATA_SECTION(g_pfnVectors, ".intvecs")
void (* const g_pfnVectors[])(void) =
{
    (void (*)(void))((uint32_t)&__STACK_TOP),
                                            // The initial stack pointer
    ResetISR,                               // The reset handler
    NmiSR,                                  // The NMI handler
    FaultISR,                               // The hard fault handler
    IntDefaultHandler,                      // The MPU fault handler
    IntDefaultHandler,                      // The bus fault handler
    IntDefaultHandler,                      // The usage fault handler
    0,                                      // Reserved
    0,                                      // Reserved
    0,                                      // Reserved
    0,                                      // Reserved
    IntDefaultHandler,                      // SVCall handler
    IntDefaultHandler,                      // Debug monitor handler
    0,                                      // Reserved
    IntDefaultHandler,                      // The PendSV handler
    IntDefaultHandler,                      // The SysTick handler
    Rutina_Pto_A,                           // GPIO Port A
    IntDefaultHandler,                      // GPIO Port B
```

Manejo de interrupciones

- La rutina también se puede cambiar dinámicamente con la función IntRegister:

```
GPIOIntRegister(GPIO_PORTJ_BASE, rutina_1);  //Registrar la rutina de interrupción
...
GPIOIntUnregister(GPIO_PORTJ_BASE)           //Eliminar el registro
GPIOIntRegister(GPIO_PORTJ_BASE, rutina_2);  //Registrar la nueva rutina
```

- OJO: al registrar, se habilita la interrupción del periférico

Ejemplo 2_4

- Versión del ejemplo 2, usando interrupciones.
- Configuración:

```
GPIOIntTypeSet(GPIO_PORTJ_BASE,GPIO_PIN_0|GPIO_PIN_1,GPIO_FALLING_EDGE); // Definir tipo int: flanco bajada
GPIOIntEnable(GPIO_PORTJ_BASE, GPIO_PIN_0|GPIO_PIN_1);  // Habilitar pines de interrupción J0, J1
GPIOIntRegister(GPIO_PORTJ_BASE, rutina_interrupcion);  //Registrar (definir) la rutina de interrupción
IntEnable(INT_GPIOJ);                                   //Habilitar interrupción del pto J
IntMasterEnable();                                      // Habilitar globalmente las ints
```

- Interrupción:

```
void rutina_interrupcion(void)
{
    if(B1_ON)
    {   while(B1_ON);
        SysCtlDelay(20*MSEC);
        estado++;
        if(estado==MaxEst) estado=MaxEst-1;
        GPIOIntClear(GPIO_PORTJ_BASE, GPIO_PIN_0);
    }
    if(B2_ON)
      {   while(B2_ON);
          SysCtlDelay(20*MSEC);
          estado--; if(estado==-1) estado=0;
          GPIOIntClear(GPIO_PORTJ_BASE, GPIO_PIN_1);
      }
}
```

tm4c1294ncpdt_startup_ccs.c

- Contiene tabla de vectores de interrupción.
- Definición de 4 funciones básicas:
  - ResetISR: tras el reset, ejecuta c_int00 (asm)
  - NmiSR: int. no enmascarable.
  - FaultISR: si error hw, por ejemplo si se accede a periférico no inicializado
  - IntDefaultHandler: int. por defecto

```
void ResetISR(void);
static void NmiSR(void);
static void FaultISR(void);
static void IntDefaultHandler(void);
```

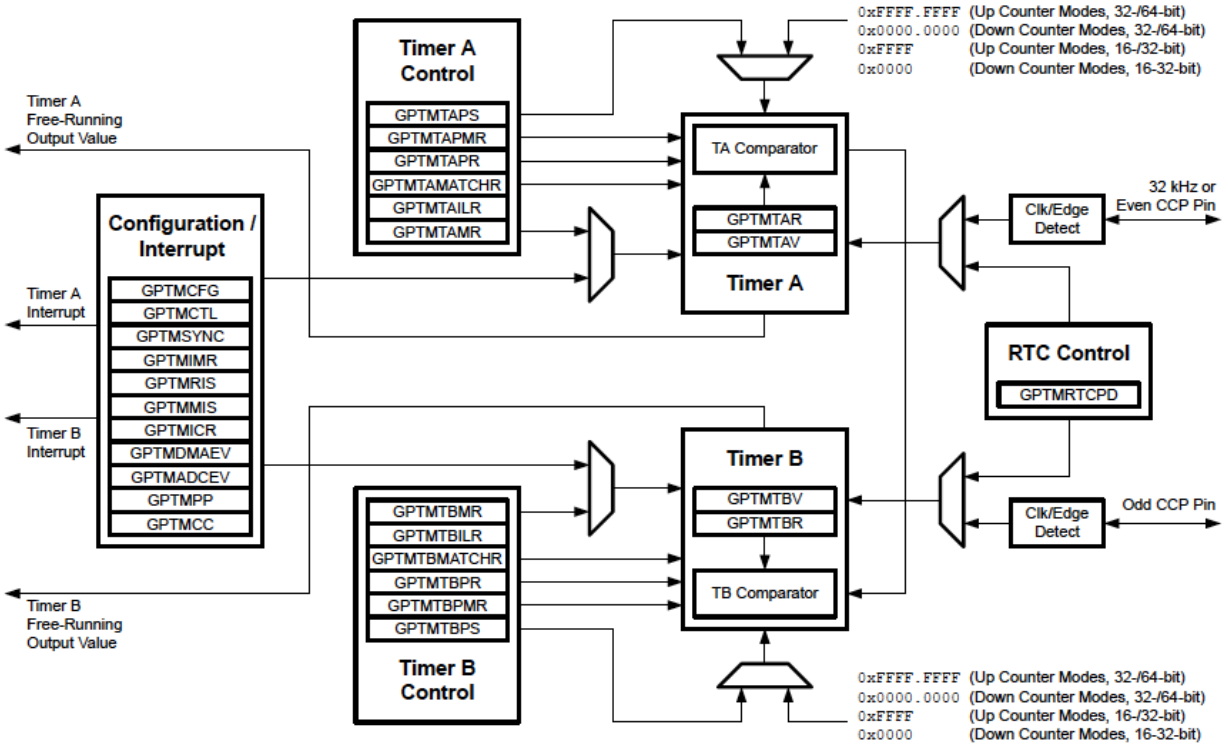
Temporizadores

- 8 timers de 32 bits
  - o hasta 16 timers de 16 bits con preescalado
- Generación de señales pwm
- 16 interrupciones asociadas

Mode	Timer Use	Count Direction	Counter Size	Prescaler Size[a]
One-shot	Individual	Up or Down	16-bit	8-bit
	Concatenated	Up or Down	32-bit	-
Periodic	Individual	Up or Down	16-bit	8-bit
	Concatenated	Up or Down	32-bit	-
RTC	Concatenated	Up	32-bit	-
Edge Count	Individual	Up or Down	16-bit	8-bit
Edge Time	Individual	Up or Down	16-bit	8-bit
PWM	Individual	Down	16-bit	8-bit

Temporizadores

- Esquema general



Funciones de configuración

- TimerConfigure(Base, Config)
  - Seleccionar modo (up, periodic, 32/16 bits…)
  - Multitud de máscaras de bit, combinables
- Ejemplos:
  - Periódico, modo 32 bits:
    - `TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);`
  - Periódico, 2 timers de 16 bits:
    - `TimerConfigure(TIMER0_BASE, TIMER_CFG_SPLIT_PAIR | TIMER_CFG_A_PERIODIC | TIMER_CFG_B_PERIODIC);`

Funciones de configuración

- TimerLoadSet(Base, Timer, valor)
  - Seleccionar Timer (TIMER_A, TIMER_B)
  - Depende de la entrada de reloj y el preescalado (en modo 16 bits)
- Ejemplo:
  - Cargar el Timer_A con *Periodo*:
    - `TimerLoadSet(TIMER0_BASE, TIMER_A, Periodo);`

Funciones de configuración

- TimerClockSourceSet(Base, fuente)
  - *fuente* puede ser:
    - TIMER_CLOCK_SYSTEM (el de sistema: 120MHz, externo)
    - TIMER_CLOCK_PIOSC (16MHz, interno)
- Ejemplos:
  - Timer_0 usa reloj de 120MHz:
    - `TimerClockSourceSet(TIMER0_BASE, TIMER_CLOCK_SYSTEM );`
  - Timer_3 usa reloj de 16MHz:
    - `TimerClockSourceSet(TIMER3_BASE, TIMER_CLOCK_PIOSC );`

Funciones de configuración

- TimerPrescaleSet(Base, Timer, Valor)
  - Sólo en modo *split* (16 bits)
  - *Valor* entre 0 y 255
- Ejemplo:
  - Timer_0_A preescalado a 120:
    - `TimerPrescaleSet(TIMER0_BASE, TIMER_A,119 );`

Funciones de configuración

- Para configurar la interrupción:
  - IntEnable(INT_TIMERXY)
    - *X=0..7, Y=A,B*
  - TimerIntEnable(Base, Tipo(s) de Int(s))
    - Para cada Timer, qué eventos provocan interrupciones
- Ejemplos:
  - Interrupción en el Timeout de Timer1_A:
    - `IntEnable(INT_TIMER1A);`
    - `TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);`
  - Interrupción en los Timeouts de Timer2_A y Timer2_B:
    - `IntEnable(INT_TIMER2A); IntEnable(INT_TIMER2B);`
    - `TimerIntEnable(TIMER2_BASE,TIMER_TIMA_TIMEOUT| TIMER_TIMB_TIMEOUT);`

Ejemplo 3

- Configurar los timers de la siguiente forma:
  - TIMER_0, 32 bits, periodo 0.5s
  - Timer_1, 32 bits, periodo 0.25s
  - Timer 2, 16 bits (dos contadores):
    - Timer_2_A con periodo de 0.5s
    - Timer_2_B con periodo de 0.25s
  - Generar las 4 interrupciones de TimeOut
- Conmutar los LED de la placa en las diferentes interrupciones

Ejemplo 3

- Inicializar reloj y periféricos:

```
Reloj = SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ | SYSCTL_OSC_MAIN |
SYSCTL_USE_PLL | SYSCTL_CFG_VCO_480), 120000000);

SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);       //Habilita T0
SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);       //Habilita T1
SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER2);       //Habilita T2

GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, GPIO_PIN_0|GPIO_PIN_1);
GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_0|GPIO_PIN_4);
```

Ejemplo 3

- Configurar funcionamiento de los Timers:

```
TimerClockSourceSet(TIMER0_BASE, TIMER_CLOCK_SYSTEM);   //T0 a 120MHz
TimerClockSourceSet(TIMER1_BASE, TIMER_CLOCK_SYSTEM);   //T1 a 120MHz
TimerClockSourceSet(TIMER2_BASE, TIMER_CLOCK_PIOSC);    //T2 a 16MHz


TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);  //T0 periodico y 32bits
TimerConfigure(TIMER1_BASE, TIMER_CFG_PERIODIC);  //T1 igual...
TimerConfigure(TIMER2_BASE, TIMER_CFG_SPLIT_PAIR| TIMER_CFG_A_PERIODIC |
TIMER_CFG_B_PERIODIC);
//T2 configurado como Split (16bit), periodicos ambas mitades

TimerPrescaleSet(TIMER2_BASE, TIMER_A,255); // T2 preescalado a 256: 62.5kHz
TimerPrescaleSet(TIMER2_BASE, TIMER_B,255); // ambas mitades
```

Ejemplo 3

- Calcular y fijar periodos:
  - 0.5s para el 1 y 4; 0.25s para el 2 y 3

```
periodo1 = Reloj/2; //0.5s
periodo2 = Reloj/4; //0.25s
periodo3 = 15625;    // T/4:0.25s
periodo4 = 31250;    // T/2: 0.5s

TimerLoadSet(TIMER0_BASE, TIMER_A, periodo1 -1);
TimerLoadSet(TIMER1_BASE, TIMER_A, periodo2 -1);
TimerLoadSet(TIMER2_BASE, TIMER_A, periodo3 -1);
TimerLoadSet(TIMER2_BASE, TIMER_B, periodo4 -1);
```

Ejemplo 3

- Gestionar las interrupciones:

```
TimerIntRegister(TIMER0_BASE,TIMER_A,IntTimer0);
TimerIntRegister(TIMER1_BASE,TIMER_A,IntTimer1);
TimerIntRegister(TIMER2_BASE,TIMER_A,IntTimer2);
TimerIntRegister(TIMER2_BASE,TIMER_B,IntTimer3);

IntEnable(INT_TIMER0A); //Habilitar las interrupciones de los timers
IntEnable(INT_TIMER1A);
IntEnable(INT_TIMER2A);
IntEnable(INT_TIMER2B);

TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);   // ints timeout
TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
TimerIntEnable(TIMER2_BASE, TIMER_TIMA_TIMEOUT |TIMER_TIMB_TIMEOUT);

IntMasterEnable();  //Habilitacion global de interrupciones
```

Ejemplo 3

- Habilitar los timers, y bucle vacío:

```
TimerEnable(TIMER0_BASE, TIMER_A);  //Habilitar Timer0, 1, 2A y 2B
TimerEnable(TIMER1_BASE, TIMER_A);
TimerEnable(TIMER2_BASE, TIMER_A);
TimerEnable(TIMER2_BASE, TIMER_B);
while(1)
{
    //Bucle infinito en el que no se hace nada
}
```

Rutinas de Interrupción

- Las 4 muy parecidas:
  - Borrar flag
  - Conmutar pin

```
void IntTimer0(void)
{
TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT); // Borra flag
// Si el pin está a 1 lo pone a 0, y viceversa
if(GPIOPinRead(GPIO_PORTN_BASE, GPIO_PIN_1))
   {
      GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_1, 0);
   }
else
   {
      GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_1, GPIO_PIN_1);
   }
}
```

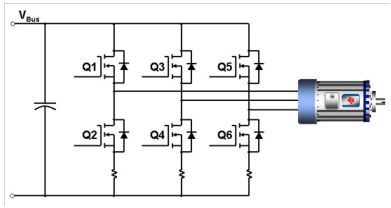
Ejemplo 3 (modificado)

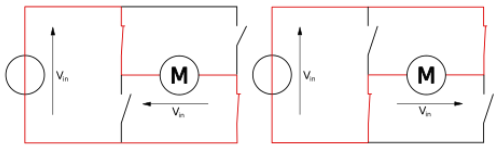
- Si se usa solamente el reloj de 120MHz para todas, no se observa desfase
  - No depende del preescalado sino de la fuente
- Interrupción: cuenta a 5

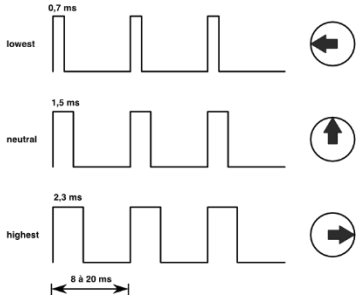
```
TimerClockSourceSet(TIMER2_BASE, TIMER_CLOCK_SYSTEM);    //T2 a 120MHz
TimerConfigure(TIMER2_BASE, TIMER_CFG_SPLIT_PAIR| TIMER_CFG_A_PERIODIC | TIMER_CFG_B_PERIODIC);

TimerPrescaleSet(TIMER2_BASE, TIMER_A,239);     // T2 preescalado a 240: 500kHz
TimerPrescaleSet(TIMER2_BASE, TIMER_B,239);

periodo3 = 25000;    // 0.05s
periodo4 = 50000;    // 0.1s

TimerLoadSet(TIMER2_BASE, TIMER_A, periodo3 -1);
TimerLoadSet(TIMER2_BASE, TIMER_B, periodo4 -1);
```

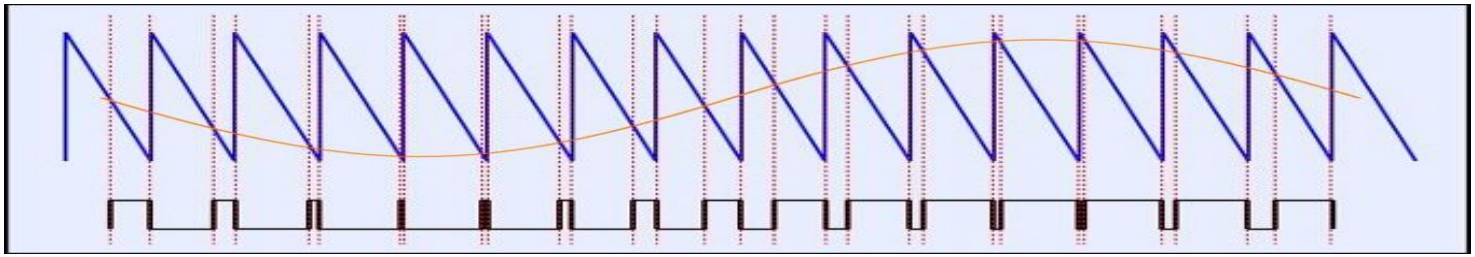
Modulación PWM

- Pulse width modulation
- Fundamental en control de potencia
  - Inversores
  - Control de motores DC
- Referencia para servos
- En esencia, un contador y un comparador



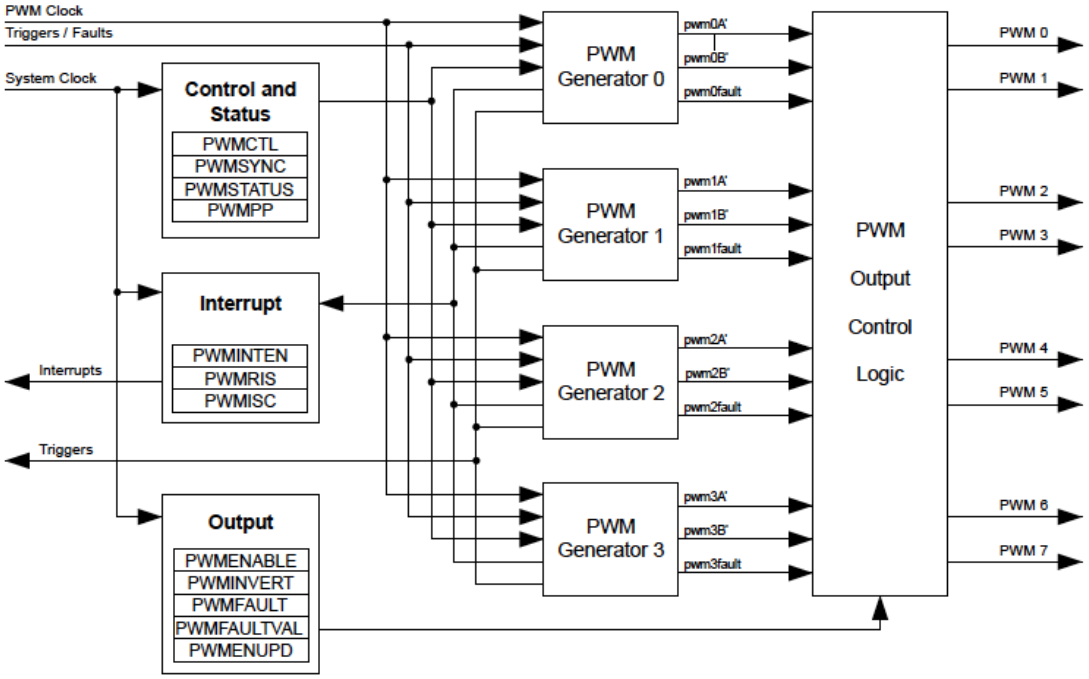






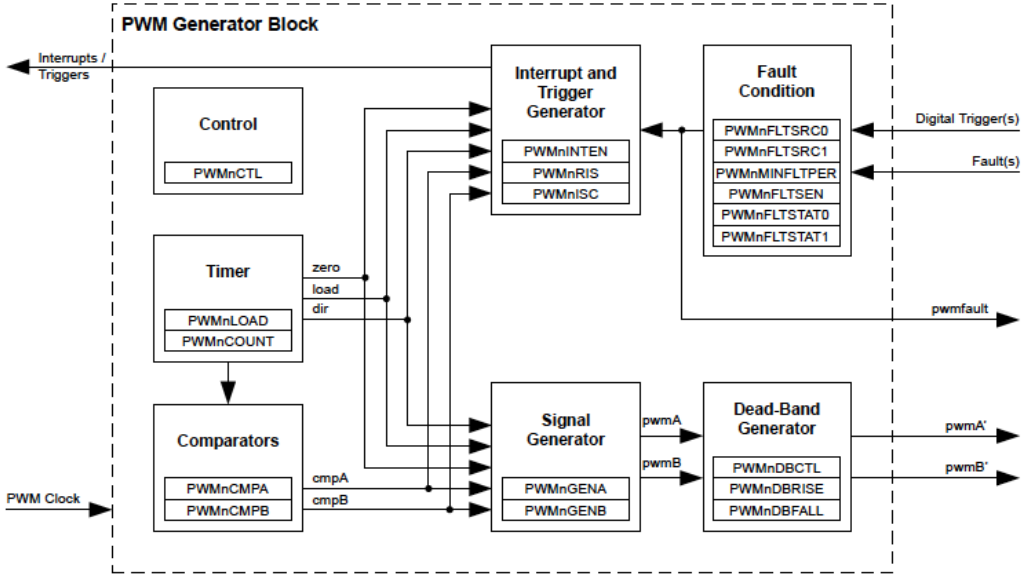
PWM en TM4C1294

- 4 generadores PWM independientes
- Bloque de control para polaridad y pines



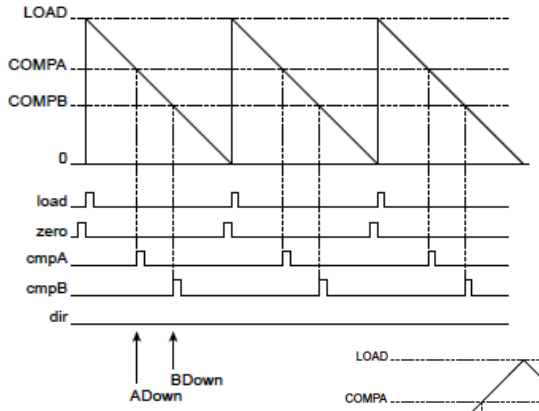
Generador PWM en TM4C1294

- Dos señales independientes o complementarias
- Modos de protección Hardware (interrupciones)
- 1 Timer y 2 comparadores de 16 bits por bloque

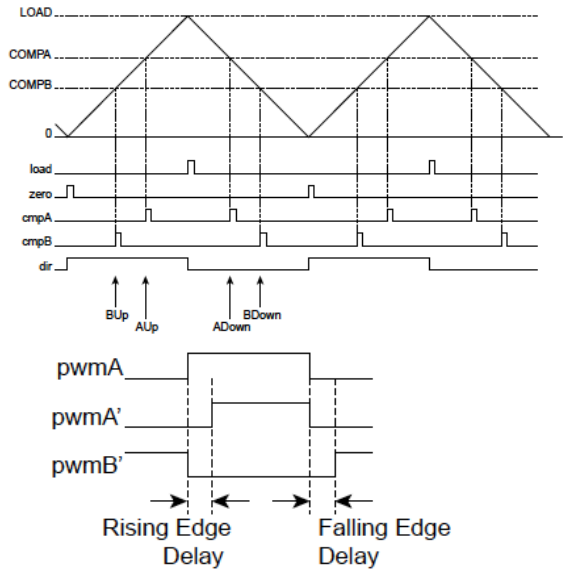


Carácterísticas del PWM

- Contador Down:
  - Modo más simple



- Contador UP/DOWN:
  - Señales simétricas
- Dead Band:
  - Evitar cortos en puentes H



Manejo con Driverlib

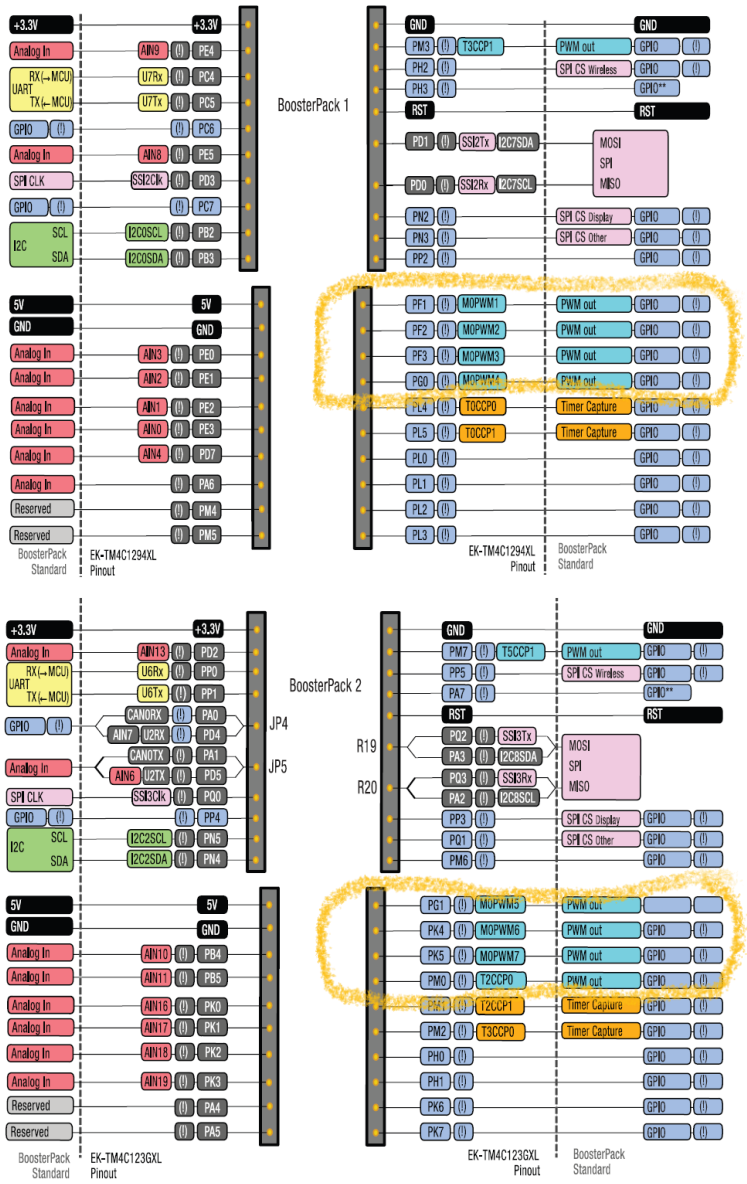
- Incluir el fichero pwm.h
- Manejo:
  - Conectar reloj a los puertos y al módulo pwm
  - Definir los pines como de PWM
  - Configurar el Generador de PWM
  - Fijar periodo y ancho de pulso
  - Habilitar el generador
  - Definir salida (activa a nivel alto, bajo…)

Pines asociados (TM4C1294NCPDT)

Pin Name	Pin Number	Pin Mux / Pin Assignment	Pin Type	Buffer Type	Description
M0FAULT0	46	PF4 (6)	I	TTL	Motion Control Module 0 PWM Fault 0.
M0FAULT1	61	PK6 (6)	I	TTL	Motion Control Module 0 PWM Fault 1.
M0FAULT2	60	PK7 (6)	I	TTL	Motion Control Module 0 PWM Fault 2.
M0FAULT3	81	PL0 (6)	I	TTL	Motion Control Module 0 PWM Fault 3.
M0PWM0	42	PF0 (6)	O	TTL	Motion Control Module 0 PWM 0. This signal is controlled by Module 0 PWM Generator 0.
M0PWM1	43	PF1 (6)	O	TTL	Motion Control Module 0 PWM 1. This signal is controlled by Module 0 PWM Generator 0.
M0PWM2	44	PF2 (6)	O	TTL	Motion Control Module 0 PWM 2. This signal is controlled by Module 0 PWM Generator 1.
M0PWM3	45	PF3 (6)	O	TTL	Motion Control Module 0 PWM 3. This signal is controlled by Module 0 PWM Generator 1.
M0PWM4	49	PG0 (6)	O	TTL	Motion Control Module 0 PWM 4. This signal is controlled by Module 0 PWM Generator 2.
M0PWM5	50	PG1 (6)	O	TTL	Motion Control Module 0 PWM 5. This signal is controlled by Module 0 PWM Generator 2.
M0PWM6	63	PK4 (6)	O	TTL	Motion Control Module 0 PWM 6. This signal is controlled by Module 0 PWM Generator 3.
M0PWM7	62	PK5 (6)	O	TTL	Motion Control Module 0 PWM 7. This signal is controlled by Module 0 PWM Generator 3.

En el Launchpad

- En ambos Boosterpack:
  - 4 en BP1
  - 3 en BP2, más PWM del Timer2



Funciones básicas(I)

- Configuración del reloj:
  - divisor del reloj del sistema (1, 2, 4, 8, 16, 32, 64)
  - Ejemplo: usar un reloj de 30MHz
    - supuesto 120MHz, dividirlo por 4
  - `PWMClockSet(PWM0_BASE, PWM_SYSCLK_DIV_4);`
- Configuración del pin a usar:
  - Dos funciones, la general y una particular.
  - Ejemplo: configurar el pin G0 como PWM4

```
GPIOPinConfigure(GPIO_PG0_M0PWM4);              //Conectar el pin a PWM
GPIOPinTypePWM(GPIO_PORTG_BASE, GPIO_PIN_0);    //Tipo de pin PWM
```

Funciones básicas(II)

- Configuración del generador pwm:
  - múltiples opciones (máscaras de bit: mirar pwm.h)
  - Básico: contador down o up/down
  - Ejemplo: configurar el PWM4 como Down:
  - `PWMGenConfigure(PWM0_BASE,PWM_GEN_2, PWM_GEN_MODE_DOWN);`
- Configuración del periodo:
  - Especificar el periodo (16 bits).
  - Ejemplo: configurar el periodo a 1kHz
    - Sup. 30MHz: periodo= (30M/1k)-1 = 29999
  - `PWMGenPeriodSet(PWM0_BASE,PWM_GEN_2, 29999);`

Funciones básicas(III)

- Habilitar la salida pwm
  - valor booleano (true/false)
  - Ejemplo: habilitar PWM4 :
  - `PWMOutputState(PWM0_BASE,PWM_OUT_4_BIT,true);`
- Signo de la salida:
  - Posibilidad de invertirla. Por defecto, activa a nivel alto.
  - Ejemplo: configurar la salida PWM4 como activa a nivel bajo
  - `PWMOutputInvert(PWM0_BASE, PWM_OUT_4_BIT,true);`

Funciones básicas(y IV)

- Habilitar el generador globalmente
  - valor booleano (true/false)
  - Ejemplo: habilitar el generador 2 (donde está PWM4) :
  - `PWMGenEnable(PWM0_BASE,PWM_GEN2);`
- Fijar la anchura del pulso:
  - La actualización puede ser instantánea o sincronizada (opción de configuración)
  - Ejemplo: fijar la anchura a un 60% del caso anterior
    - como el periodo era 30000, el 60% es 18000
  - `PWMPulseWidthSet(PWM0_BASE, PWM_OUT_4,18000);`

Otras funciones

- Funciones secundarias. Mirar documentación
  - `void PWMDeadBandEnable(uint32_t ui32Base, uint32_t ui32Gen, uint16_t ui16Rise, uint16_t ui16Fall);`
  - `void PWMDeadBandDisable(uint32_t ui32Base, uint32_t ui32Gen);`
  - `void PWMGenIntTrigEnable(uint32_t ui32Base, uint32_t ui32Gen, uint32_t ui32IntTrig);`
  - `void PWMGenIntTrigDisable(uint32_t ui32Base, uint32_t ui32Gen, uint32_t ui32IntTrig);`
  - `uint32_t PWMGenIntStatus(uint32_t ui32Base, uint32_t ui32Gen, bool bMasked);`
  - `void PWMGenIntClear(uint32_t ui32Base, uint32_t ui32Gen, uint32_t ui32Ints);`
  - `void PWMIntEnable(uint32_t ui32Base, uint32_t ui32GenFault);`
  - `void PWMIntDisable(uint32_t ui32Base, uint32_t ui32GenFault);`

Ejemplo 3(BIS)

- Modular la intensidad luminosa del led de PF0
- PF0 puede ser controlado con PWM0
- Usar los botones para subir o bajar la intensidad
  - Incrementar o decrementar un 10%
  - Saturar al 90% y al 10% de periodo activo

Ejemplo 3 (bis)

- Configuración de los periféricos:

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOJ);
SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0);
```

- Configuración del PWM:

```
PWMClockSet(PWM0_BASE,PWM_SYSCLK_DIV_4); // al PWM le llega un reloj de 30MHz
GPIOPinConfigure(GPIO_PF0_M0PWM0);//Configurar el pin a PWM
GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_0);
PWMGenConfigure(PWM0_BASE, PWM_GEN_0, PWM_GEN_MODE_DOWN | PWM_GEN_MODE_NO_SYNC);
PeriodoPWM=29999; // 1kHz a 30M
PWMGenPeriodSet(PWM0_BASE, PWM_GEN_0, PeriodoPWM); //frec:1kHz
PWMPulseWidthSet(PWM0_BASE, PWM_OUT_0, PeriodoPWM/2); //Inicialmente, un 50%
PWMGenEnable(PWM0_BASE, PWM_GEN_0);//Habilita el generador 0
PWMOutputState(PWM0_BASE, PWM_OUT_0_BIT , true);//Habilita la salida 0
```

Ejemplo 3 (BIS)

- Bucle de control similar a los anteriores.
  - Usar la función PWMPulseWidthSet

```
i=5;//Inicialmente, 50%
while(1){
if(B1_ON){//Si se pulsa B1
          SysCtlDelay(10*MSEC);
          while(B1_ON);
          SysCtlDelay(10*MSEC);
          i++; if(i==MaxEst) i=MaxEst-1; //Incrementa saturando a 90%
          PWMPulseWidthSet(PWM0_BASE, PWM_OUT_0, PeriodoPWM*i/10);
          }
if(B2_ON){//Si se pulsa B2
          SysCtlDelay(10*MSEC);
          while(B2_ON);
          SysCtlDelay(10*MSEC);
          i--; if(i==0) i=1; //Decrementa el periodo, saturando al 10%
          PWMPulseWidthSet(PWM0_BASE, PWM_OUT_0, PeriodoPWM*i/10);
          }
```

Ejemplo 3 (3)

- Manejar un servo con un pwm.
  - Conectado a PF1 (M0PWM1)
  - Periodo de 20ms, duty cycle entre 1 y 2ms.
- Configuración:

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0);
PWMClockSet(PWM0_BASE,PWM_SYSCLK_DIV_64);// al PWM le llega un reloj de 1.875MHz
GPIOPinConfigure(GPIO_PF1_M0PWM1);//Configurar el pin a PWM
GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_1);
//pwm0, contador descendente y sin sincronización (actualización automática)
PWMGenConfigure(PWM0_BASE, PWM_GEN_0, PWM_GEN_MODE_DOWN | PWM_GEN_MODE_NO_SYNC);
PeriodoPWM=37499; // 50Hz  a 1.875MHz
pos_100=50; //Inicialmente, 50%
posicion=Min_pos+((Max_pos-Min_pos)*pos_100)/100;
PWMGenPeriodSet(PWM0_BASE, PWM_GEN_0, PeriodoPWM); //frec:50Hz
PWMPulseWidthSet(PWM0_BASE, PWM_OUT_1, posicion); //Inicialmente, 1ms
PWMGenEnable(PWM0_BASE, PWM_GEN_0);//Habilita el generador 0
PWMOutputState(PWM0_BASE, PWM_OUT_1_BIT , true);//Habilita la salida 1
```

Ejemplo 3 (3)

- Operación continua (mientras esté pulsado)

```
while(1){
        if(B1_ON){//Si B1 pulsado incrementa, saturando
                SysCtlDelay(10*MSEC);
                if(pos_100<100) pos_100++;
                }
        if(B2_ON){//Si B2 pulsado decrementa, saturando
                SysCtlDelay(10*MSEC);
                if(pos_100>0) pos_100--;
                }
        posicion=Min_pos+((Max_pos-Min_pos)*pos_100)/100;
       PWMPulseWidthSet(PWM0_BASE, PWM_OUT_1, posicion);
        }
```