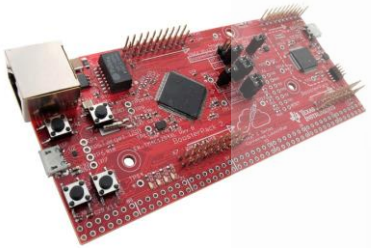


Tema 5. Periféricos del TM4C1294 (II)

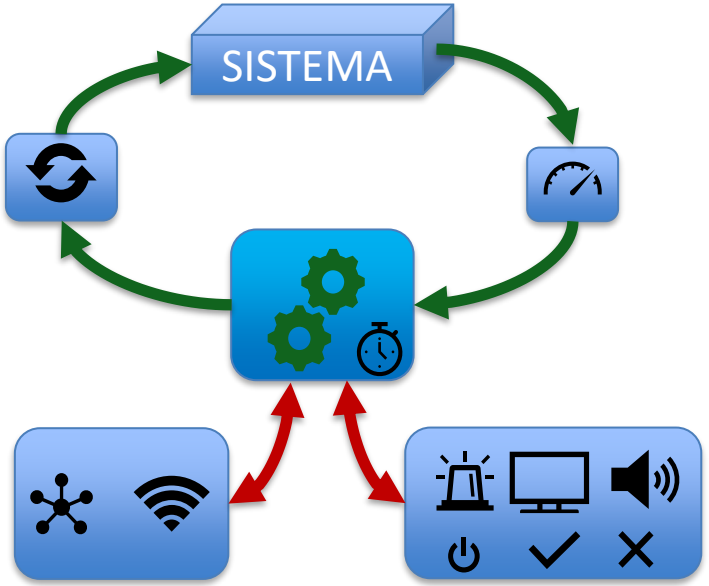
4º Grado de Ingeniería en Electrónica, Robótica y Mecatrónica
Andalucía Tech

Índice

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

Introducción

- El sistema necesita comunicaciones
  - Con otros periféricos
  - Con supervisión
- Necesidad de puertos serie de diversos tipos, según sus usos
- Además, puertos de comunicación de alto nivel (USB, ETHERNET)

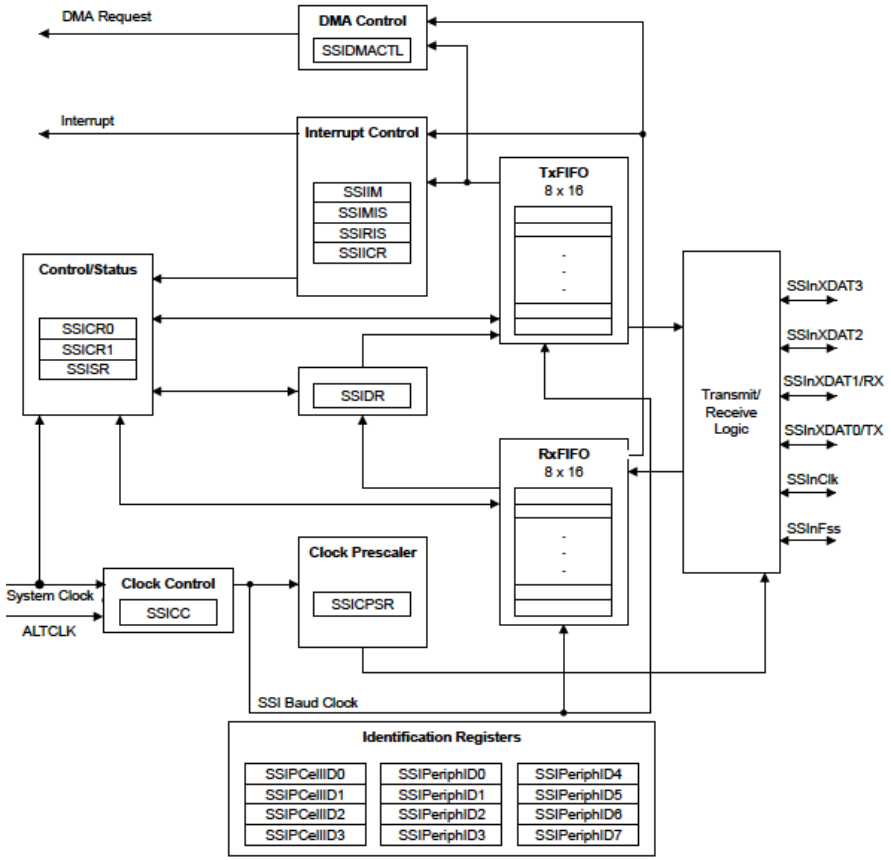


Puertos serie

- 4 puertos serie Síncronos: QSSI
  - Modo SPI (Freescale) y modo *TI-SSI*
  - de 1 a 4 líneas simultáneas con un reloj
- 8 puertos serie UART
  - Hasta 15MBPS
  - Control de modem en algunos
- 10 puertos I2C
- 2 buses CAN completos

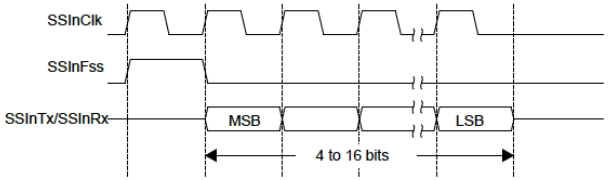
QSSI (SPI)

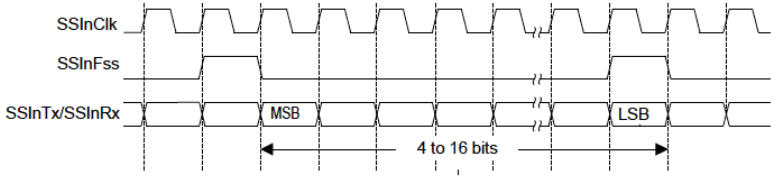
- QSSI: evolución del SPI.
- FIFOs de 8 x 16bits
- Señales:
  - 4 líneas de datos
  - 1 clk
  - Frame signal (CS)



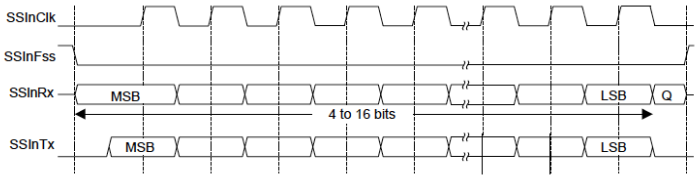
TI-SSI vs. SPI

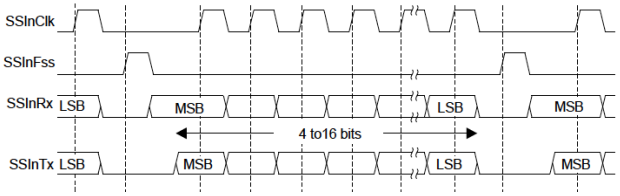
- Formas de onda:
  - TI-SSI (simple y continua)





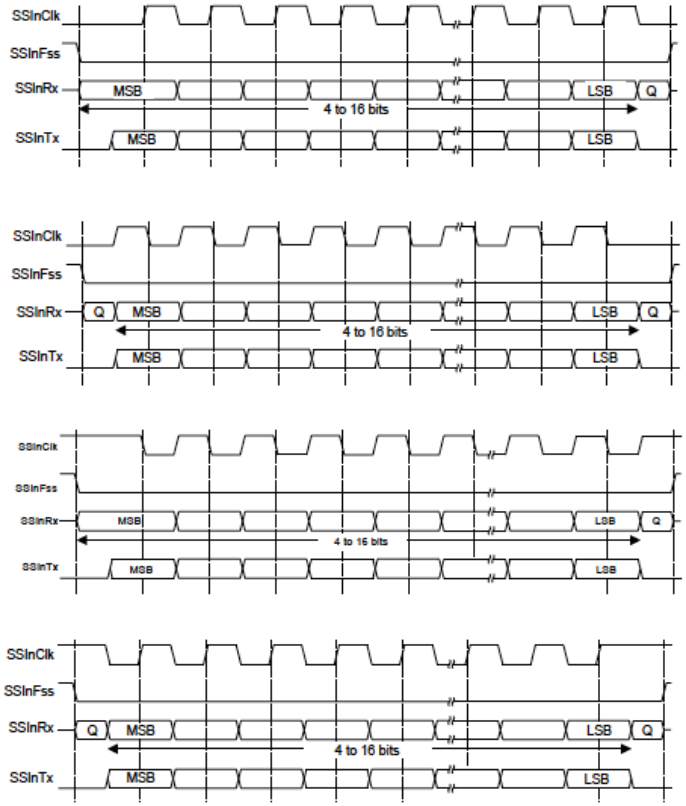
  - SPI (simple y continua)





QSSI en modo SPI

- Interés por la presencia en el mercado
- Estándar muy implantado
- 4 modos de reloj.
  - SSI_FRF_MOTO_MODE_0…3
- Master o Slave
  - SSI_MODE_MASTER / SLAVE



Funciones API para configuración

- **SSIConfigSetExpClk**(Base, Frec, Protocolo, Modo, BPS, bits):
  - BPS tiene que ser menor de Frec/2, en modo máster, y Frec /12 en Slave
- Ejemplo: configuración en modo Master, SPI, modo 0, a 1MHz, 8 bits:
  - `SSIConfigSetExpClk(SSI0_BASE, reloj,SSI_FRF_MOTO_MODE_0, SSI_MODE_MASTER, 1000000, 8);`
- **SSIEnable**(Base);
  - Habilita el puerto SSI correspondiente

Configuración de los pines

- Doble configuración:
  - Configurar el pin como pin de SS:
    - `GPIOPinConfigure(TIPO)`
  - Configurar pin con la función específica:
    - `GPIOPinTypeSSI(BASE_PTO, Pin)`
  - Mirar en pin_map.h las combinaciones posibles
  - SSI3 con pines *dobles*
- Pines SPI:
  - SSInCLK: SPI_CLK
  - SSInXDAT0: SPI_TX
  - SSInXDAT1: SPI_RX
- Ejemplo: configurar el pin PD3 como reloj de SSI2:

```
GPIOPinConfigure(GPIO_PD3_SSI2CLK);
GPIOPinTypeSSI(GPIO_PORTD_BASE, GPIO_PIN_3);
```

Pin Name	Pin Number	Pin Mux / Pin Assignment
SSI0Clk	35	PA2 (15)
SSI0Fss	36	PA3 (15)
SSI0XDAT0	37	PA4 (15)
SSI0XDAT1	38	PA5 (15)
SSI0XDAT2	40	PA6 (13)
SSI0XDAT3	41	PA7 (13)
SSI1Clk	120	PB5 (15)
SSI1Fss	121	PB4 (15)
SSI1XDAT0	123	PE4 (15)
SSI1XDAT1	124	PE5 (15)
SSI1XDAT2	125	PD4 (15)
SSI1XDAT3	126	PD5 (15)
SSI2Clk	4	PD3 (15)
SSI2Fss	3	PD2 (15)
SSI2XDAT0	2	PD1 (15)
SSI2XDAT1	1	PD0 (15)
SSI2XDAT2	128	PD7 (15)
SSI2XDAT3	127	PD6 (15)
SSI3Clk	5 45	PQ0 (14) PF3 (14)
SSI3Fss	6 44	PQ1 (14) PF2 (14)
SSI3XDAT0	11 43	PQ2 (14) PF1 (14)
SSI3XDAT1	27 42	PQ3 (14) PF0 (14)
SSI3XDAT2	46 118	PF4 (14) PP0 (15)
SSI3XDAT3	119	PP1 (15)

Transferencia de datos

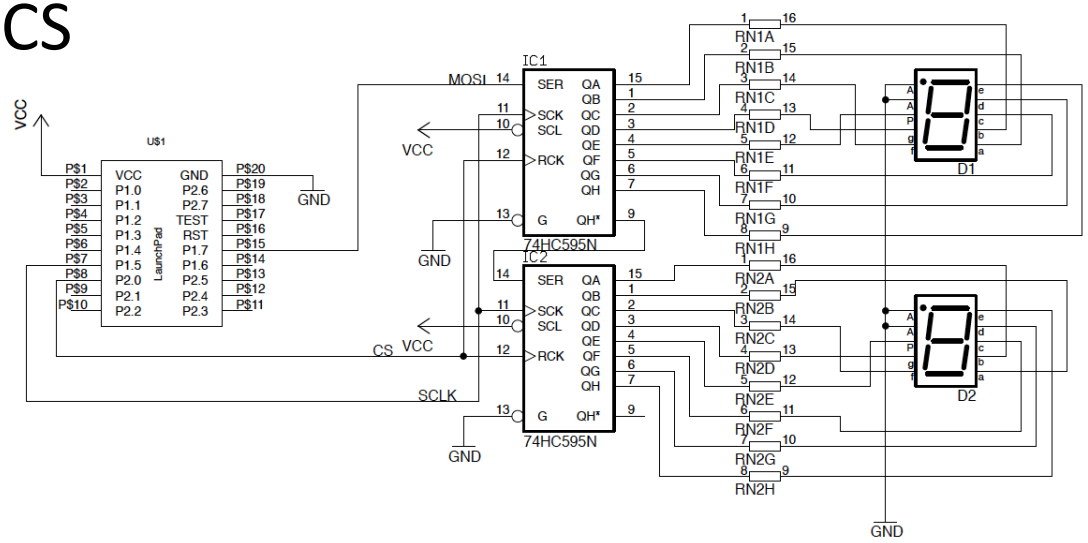
- Funciones *Blocking* y *NonBlocking:*
  - Esperan a poder transmitir o haber recibido, o no.
  - OJO con la FIFO: Poder transmitir no significa haber terminado la anterior y la Int.Rx no salta inmediatamente
- Transmitir: SSIDataPut y SSIDataPutNonBlocking
  - `SSIDataPut(SSI0_BASE, ui32Data);`
  - `uint32_t SSIDataPutNonBlocking(SSI0_BASE, ui32Data);`
    - `Si no puede, devuelve un 0`
- Recibir: SSIDataGet y SSIDataGetNonBlocking
  - `SSIDataGet(SSI0_BASE, *ui32Data);`
  - `uint32_t SSIDataGetNonBlocking(SSI0_BASE, *ui32Data);`
    - `Devuelve el número de datos recogidos (0 si nada)`

Gestión de interrupciones

- `SSIIntEnable(Base, flags);`
  - Usuales: SSI_TXFF y SSI_RXFF
- `SSIIntRegister(Base, void(*funcion)(void));`
  - Especificar la rutina de interrupción.
- Para usar las interrupciones, es necesario tener en cuenta la FIFO
  - Interrupción de FIFO RX llena
  - Interrupción de FIFO TX vacía (menos de la mitad)
  - Interrupción de overrun FIFO RX

Ejemplo 4

- Control de dos displays con SPI
  - Placa realizada para el MSP430, adaptada
- Conectados a un par de 74hc595
  - Registros de desplazamiento
- Contador de 1s de periodo
- Usar SSI2, y PC7 para CS
  - Si se conecta en BP2: cambiar por SSI3 y PP4



Configuración SSI2:

- Habilitar periférico y puerto
- Configurar pines
- Configurar SPI

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI2);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);

GPIOPinConfigure(GPIO_PD3_SSI2CLK);
GPIOPinTypeSSI(GPIO_PORTD_BASE, GPIO_PIN_3);
GPIOPinConfigure(GPIO_PD1_SSI2XDAT0);
GPIOPinTypeSSI(GPIO_PORTD_BASE, GPIO_PIN_1);

SSIConfigSetExpClk(SSI2_BASE, RELOJ,
SSI_FRF_MOTO_MODE_0,
SSI_MODE_MASTER, 1000000, 8);
SSIEnable(SSI2_BASE);
```

Configurar pines E/S y timer

- Pin PC7 como salida (CS)
- Timer0A como contador de 1s

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, GPIO_PIN_7);

SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
TimerClockSourceSet(TIMER0_BASE, TIMER_CLOCK_SYSTEM);
TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
TimerLoadSet(TIMER0_BASE, TIMER_A, RELOJ -1); //RELOJ=120Meg
TimerIntRegister(TIMER0_BASE, TIMER_A,Timer0IntHandler);
IntEnable(INT_TIMER0A);
TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
IntMasterEnable();
TimerEnable(TIMER0_BASE, TIMER_A);
```

Mandar datos en la Interrupción

- Incrementar índices y mandar por SPI

```
TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT); //Borra flag
Unidades++;//Incrementa unidades
if(Unidades==10){
        Unidades=0;
        Decenas++;//Incrementa decenas
        if(Decenas==10)Decenas=0;
        }
CSL;//CS=0
SSIDataPut(SSI2_BASE, disp[Decenas]); //Manda MSB
SSIDataPut(SSI2_BASE, disp[Unidades]);//Manda LSB
while(SSIBusy(SSI2_BASE)) {} //Espera fin de Tx
CSH;//CS=1
```

- Defines: CSH, CSL y disp:

```
#define CSL GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, 0);
#define CSH GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, GPIO_PIN_7);
const char disp[]={0xE7, 0x21, 0xCB, 0x6B, 0x2D, 0x6E, 0xEe, 0x23,
0xEF, 0x2F, 0x00};
```

Posible variación

- Configurar la transmisión para 16 bits:

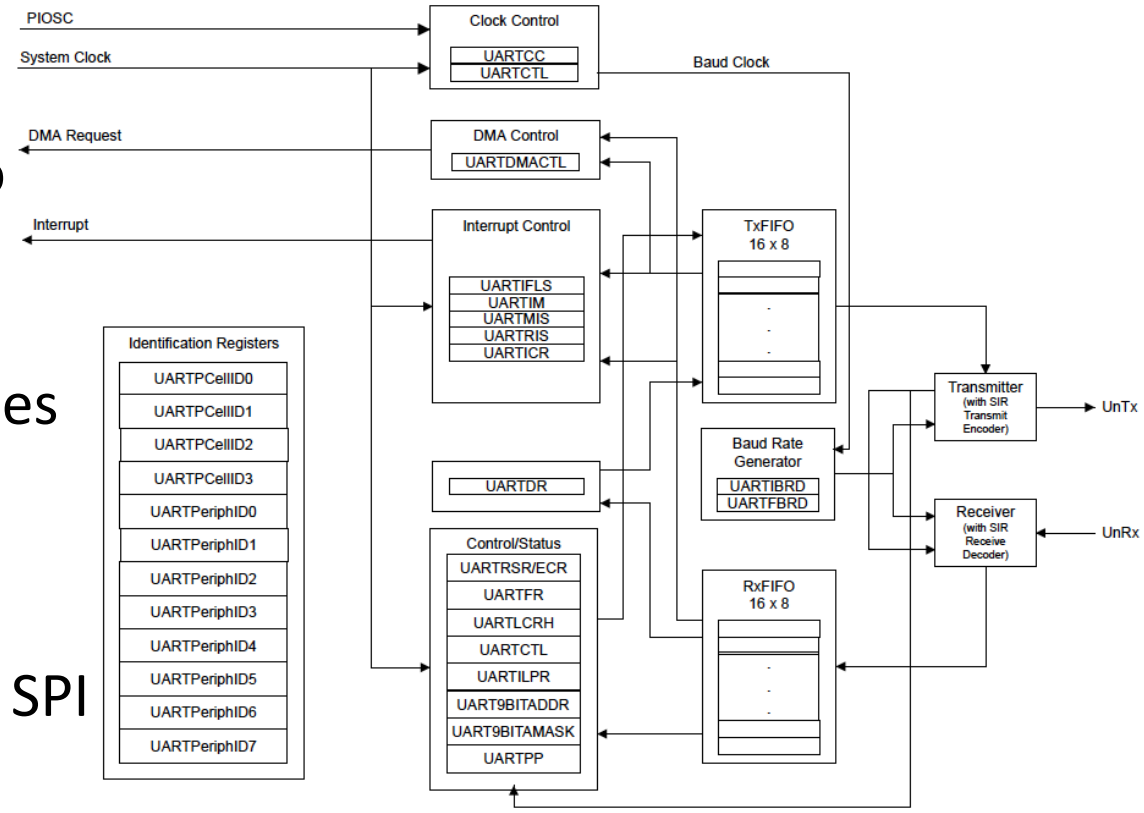
```
SSIConfigSetExpClk(SSI2_BASE, RELOJ, SSI_FRF_MOTO_MODE_0,
SSI_MODE_MASTER, 1000000, 16);
```

- Mandar un solo dato de 16 bits:

```
SSIDataPut(SSI2_BASE, ((disp[Decenas]<<8)+disp[Unidades]));
while(SSIBusy(SSI2_BASE))//Espera fin de Tx
```

UART

- 8 UART's
- 16x8 fifos
- Disparo de 1/8 a lleno
- De 5 a 8 bits de datos
- Modo 9 bit (dirección)
- Múltiples interrupciones
- Modo IrDA
- Uso de µDMA en conjunto con la FIFO.
- Manejo muy similar al SPI
- Líneas de control de Modem (uart0..4)



Configuración de la UART

- **UARTConfigSetExpClk**(Base, Reloj, Baud, Config);
- Reloj: el del sistema. Baud: la frec. Deseada
- Config: OR del nº de bits de datos, de stop y paridad:
  - UART_CONFIG_WLEN5…8
  - UART_CONFIG_STOP_ONE..TWO
  - UART_CONFIG_PAR_{NONE, ODD, EVEN, ONE, ZERO}
- Ejemplo: configuración a 115200bps, sin paridad, 1 bit de stop y 8 de datos, :
  - `UARTConfigSetExpClk(UART0_BASE, reloj,115200,UART_CONFIG_WLEN8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE);`
- **UARTEnable**(Base);
  - Habilita la UART

Configuración de los pines

- Doble configuración:
  - Configurar el pin como pin de UART:
    - `GPIOPinConfigure(TIPO)`
  - Configurar pin con la función específica:
    - `GPIOPinTypeUART(BASE_PTO, Pin)`
  - Mirar en pin_map.h las combinaciones posibles
- Ejemplo: configurar los pines A0, A1 como UART0:

```
GPIOPinConfigure(GPIO_PA0_U0RX);
GPIOPinConfigure(GPIO_PA1_U0TX);
GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 |
GPIO_PIN_1);
```

Pin Name	Pin Number	Pin Mux / Pin Assignment
U0Rx	33	PA0 (1)
U0Tx	34	PA1 (1)
U1Rx	95 102	PB0 (1) PQ4 (1)
U1Tx	96	PB1 (1)
U2Rx	40 125	PA6 (1) PD4 (1)
U2Tx	41 126	PA7 (1) PD5 (1)
U3Rx	37 116	PA4 (1) PJ0 (1)
U3Tx	38 117	PA5 (1) PJ1 (1)
U4Rx	18 35	PK0 (1) PA2 (1)
U4Tx	19 36	PK1 (1) PA3 (1)
U5Rx	23	PC6 (1)
U5Tx	22	PC7 (1)
U6Rx	118	PP0 (1)
U6Tx	119	PP1 (1)
U7Rx	25	PC4 (1)
U7Tx	24	PC5 (1)

Transferencia de datos

- Funciones *Blocking* y *NonBlocking:*
  - Esperan a poder transmitir o haber recibido, o no.
  - OJO con la FIFO: Poder transmitir no significa haber terminado la anterior, y la Int.Rx no salta inmediatamente
- Transmitir: UARTCharPut y UARTCharPutNonBlocking
  - `UARTCharPut(BASE, ui32Data);`
  - `bool UARTCharPutNonBlocking(BASE, ui32Data);`
    - Si no puede, devuelve un *false*
- Recibir: UARTCharGet y UARTCharGetNonBlocking
  - `Uint32_t UARTCharGet(BASE);`
  - `uint32_t UARTCharGetNonBlocking(BASE);`
    - Devuelve el número de datos recogidos (-1 si nada)

Gestión de interrupciones

- `UARTIntEnable(Base, flags);`
  - Usuales: UART_INT_RX, UART_INT_TX
- `UARTIntRegister(Base, void(*funcion)(void));`
  - Especificar la rutina de interrupción.
- Para usar las interrupciones, es necesario tener en cuenta la FIFO (o deshabilitarla)

Gestión de la FIFO

- **UARTFIFOEnable**(uint32_t ui32Base);
  - Habilita la FIFO
- **UARTFIFODisable**(uint32_t ui32Base);
  - Deshabilita la FIFO
- **UARTFIFOLevelSet**(uint32_t ui32Base, uint32_t ui32TxLevel, uint32_t ui32RxLevel);
  - Fija el nivel de disparo de las interrupciones
  - Valores fijos: 1, 2, 4, 6, 7:
    - **UART_FIFO_TX1_8… UART_FIFO_TX7_8**
    - **UART_FIFO_RX1_8… UART_FIFO_RX7_8**

Utilidad *uartstdio.c*

- En **`${TIVA_INSTALL}\utils\`**
- Funciones clásicas para consola
  - **`UARTStdioConfig`**`(Num, Baud, reloj)`
    - Configura la Uart para stdio
  - **`UARTgets`**`(*cadena, tamaño)`
    - Recoge caracteres hasta recibir un CR, o llegar a 'tamaño', haciendo eco de lo que llega
  - **`UARTprintf`**`("Texto y números %d", dato…)`
    - Formato clásico de *printf*.
- OJO: al usar uartstdio, se *deshabilita* la interrupción de recepción

Ejemplo 5

- Menú de varias opciones por terminal
- Manejo de las funciones de stdio
- Configuración básica (sin interrupciones)

Configuración UART:

- Habilitar periférico y puerto
  - UART0 en pines PA0, PA1
- Configurar pines
- Configurar UART y UARTstdio:

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
GPIOPinConfigure(GPIO_PA0_U0RX);
GPIOPinConfigure(GPIO_PA1_U0TX);
GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    UARTStdioConfig(0, 115200, g_ui32SysClock);
```

Escribir el Menú

- Función UARTprintf()

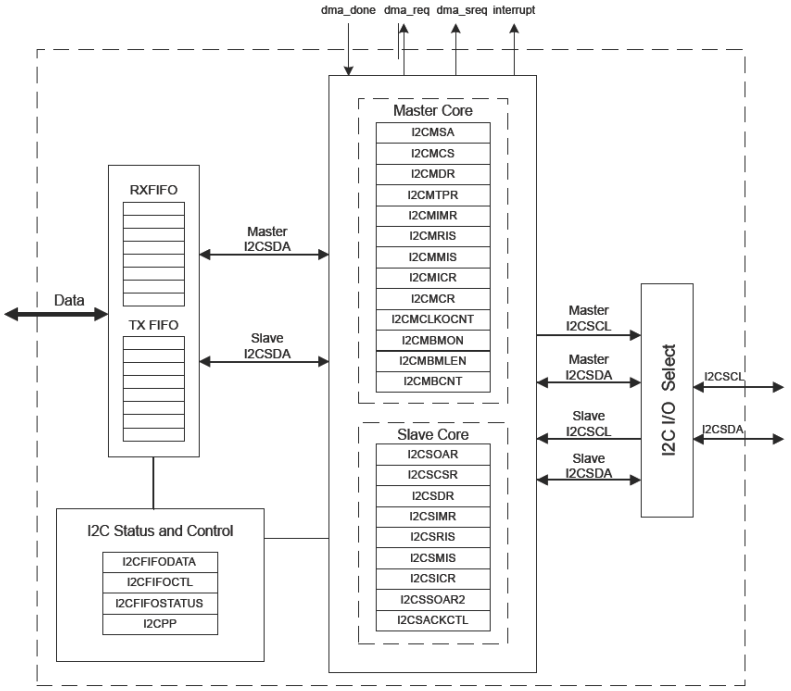
```
UARTprintf("----MENU de PRUEBA---------- \n");
UARTprintf("| - Elige una opción (1..3)-| \n");
UARTprintf("| - 1.Huevo frito          -| \n");
UARTprintf("| - 2.Tortilla francesa    -| \n");
UARTprintf("| - 3.Revuelto             -| \n");
UARTprintf("--------------------------- \n");
```

- Leer elección y presentar resultado:

```
while(1){    // \033[8;1H Ve a la octava linea, primera columna
    UARTprintf("\033[8;1H ELECCION: ");
    UARTgets(respuesta,2);    // Espera 2 teclas (o un CR)
    switch(respuesta[0]){    //Switch con el primer dato recibido
        case '1': UARTprintf("Has elegido Huevo Frito  \n");
        break;
        …
        default:  UARTprintf("Elige una opción (1..3)  \n");
        break;   o//Si distinto de 1, 2, 3, repite
        }
}
```

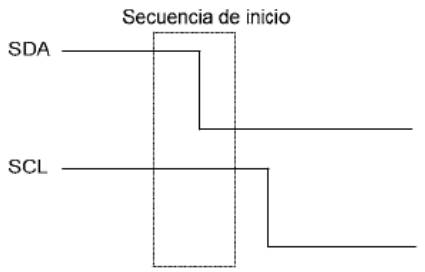
I2C

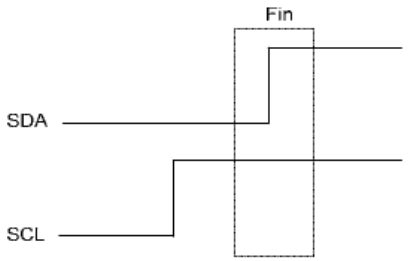
- 10 buses I2C independientes
- Modo Master y Slave
- Modo simple y *burst*
- FIFO de Tx y de Rx y DMA
- Velocidades estándar hasta 3.3MBPS

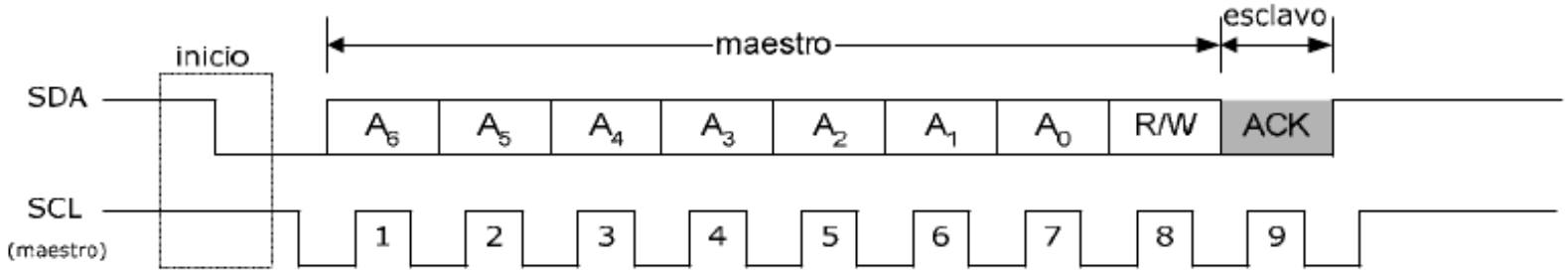


Inicio/fin de transmisión

- SDA cambia siempre con SCL a 0, excepto en Start y Stop.
- Tras Start, se manda la dirección del esclavo.
- Si éste está disponible, contesta con ACK=0



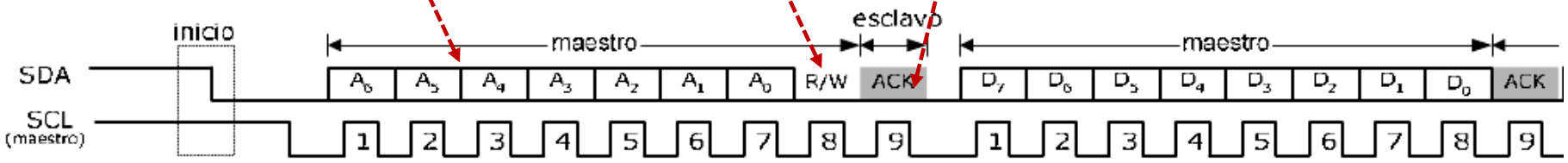


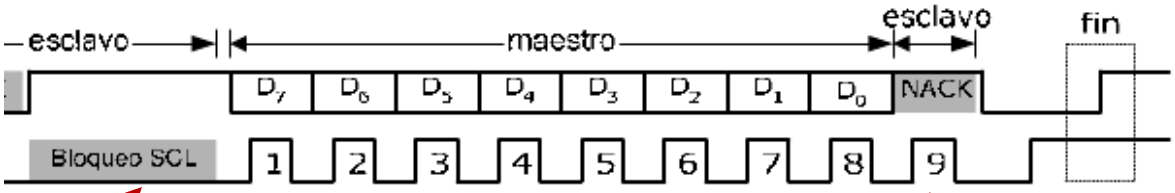


Escritura de un dato:

Se manda la dirección y el “sentido” :W

El esclavo contesta ACK=0: se mandan datos





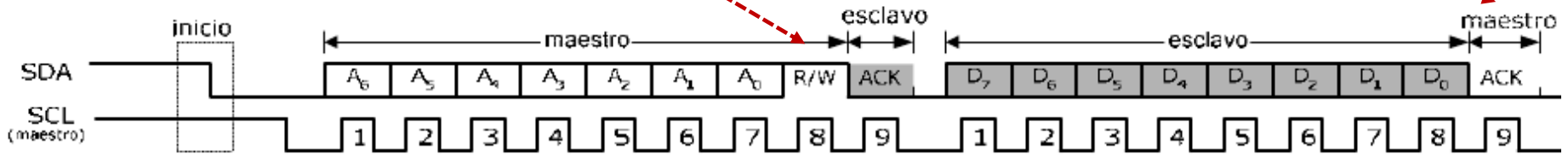
Bloqueo de SCL forzando un 0

ACK=1 no transmitir más

Lectura de un dato:

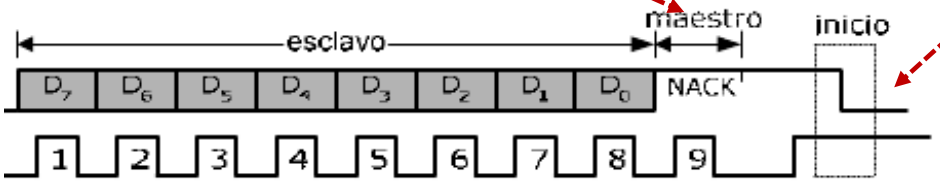
Selección de esclavo, R=1

Esclavo manda dato, master manda ACK



Master manda NACK, fin de transmisión

Nuevo inicio



Configuración del I2C

- **`I2CMasterInitExpClk`**`(Base, Reloj, AV);`
  - Reloj: el del sistema.
  - AV: *Boolean* que indica si alta velocidad o no:
    - AV=0: 100kHz; AV=1:400kHz
    - Para 1MBPS y 3.3MBPS, hay que configurarlo *a mano*
- Ejemplo: configuración a 100k, con reloj a 120M :
  - `I2CMasterInitExpClk(I2C0_BASE,120000000,false);`
- **`I2CMasterEnable`**`(Base);`
  - Habilita el I2C en modo Master

Configuración de los pines

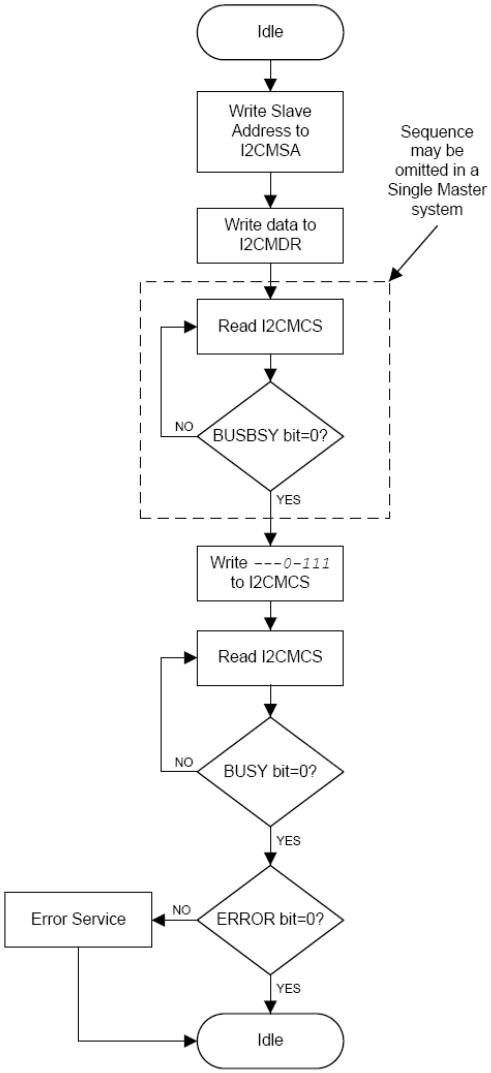
- Doble configuración:
  - Configurar el pin como pin de I2C:
    - `GPIOPinConfigure(TIPO)`
  - Configurar pin con la función específica:
    - `GPIOPinTypeI2C(BASE_PTO, Pin)`
  - Mirar en pin_map.h las combinaciones posibles
- Ejemplo: configurar el pin D1 como SDA de I2C-7:

```
GPIOPinConfigure(GPIO_PD1_I2C7SDA);
GPIOPinTypeI2C(GPIO_PORTD_BASE, GPIO_PIN_1);
```

Pin Name	Pin Number	Pin Mux / Pin Assignment
I2C0SCL	91	PB2 (2)
I2C0SDA	92	PB3 (2)
I2C1SCL	49	PG0 (2)
I2C1SDA	50	PG1 (2)
I2C2SCL	82 106 112	PL1 (2) PP5 (2) PN5 (3)
I2C2SDA	81 111	PL0 (2) PN4 (3)
I2C3SCL	63	PK4 (2)
I2C3SDA	62	PK5 (2)
I2C4SCL	61	PK6 (2)
I2C4SDA	60	PK7 (2)
I2C5SCL	95 121	PB0 (2) PB4 (2)
I2C5SDA	96 120	PB1 (2) PB5 (2)
I2C6SCL	40	PA6 (2)
I2C6SDA	41	PA7 (2)
I2C7SCL	1 37	PD0 (2) PA4 (2)
I2C7SDA	2 38	PD1 (2) PA5 (2)
I2C8SCL	3 35	PD2 (2) PA2 (2)
I2C8SDA	4 36	PD3 (2) PA3 (2)
I2C9SCL	33	PA0 (2)

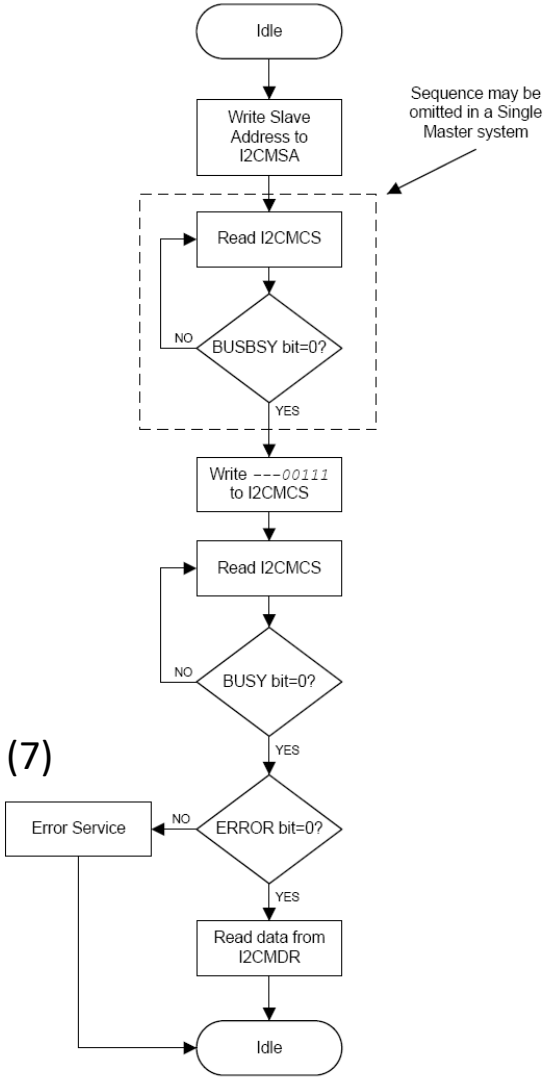
Transferencia de datos

- Proceso de escritura (master)*:*
  - Escribir la dirección del esclavo (con LSB=0)
  - Escribir el dato a mandar
  - Comprobar que el bus esté libre (sólo en multimaster)
  - Escribir 0x0007 en el registro I2CMCS
  - Esperar al fin de la comunicación (BUSY=0)
  - Mirar (si acaso) el bit de error
- Funciones para la transferencia:
- **`I2CMasterSlaveAddrSet(Base,Dir,rw);`**
  - Dir: la dirección del slave (7bit). rw=0, escribir; rw=1, leer
- **`I2CMasterDataPut(Base,Dato);`**
- **`I2CMasterControl(Base,Comando);`**
  - Comando: en este caso: I2C_MASTER_CMD_SINGLE_SEND (7)
- **`I2CMasterBusy(Base);`**
  - **`0 si libre, 1 si Busy`**



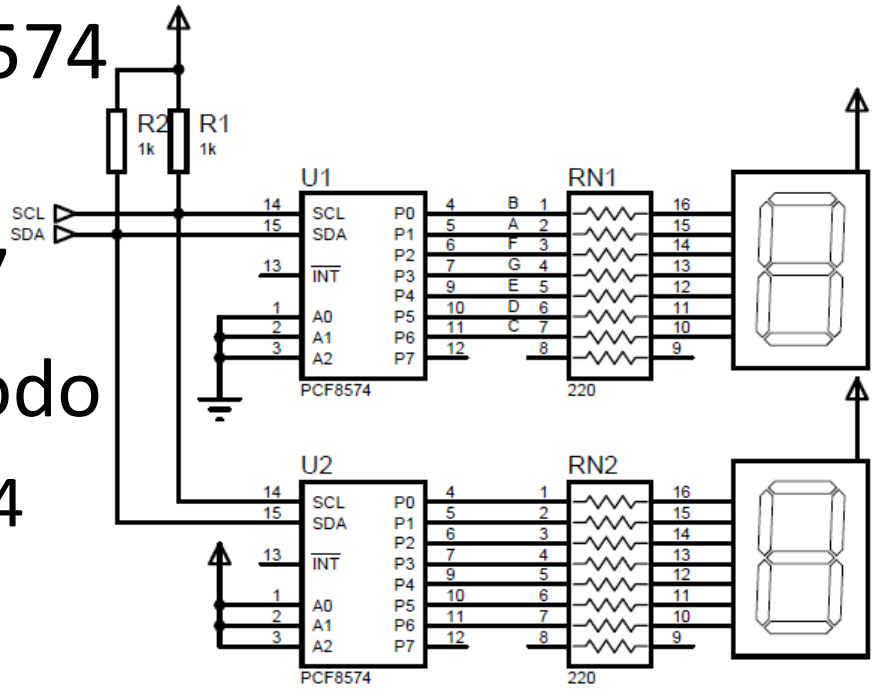
Transferencia de datos

- Proceso de lectura(master):
  - Escribir la dirección del esclavo (con LSB=1)
  - Comprobar que el bus esté libre (sólo en multimaster)
  - Escribir 0x0007 en el registro I2CMCS
  - Esperar al fin de la comunicación (BUSY=0)
  - Mirar (si acaso) el bit de error
  - Recoger el dato del registro de datos
- Funciones para la transferencia:
- `I2CMasterSlaveAddrSet(Base,Dir,rw);`
  - Dir: la dirección del slave (7bit). rw=0, escribir; rw=1, leer
- `Dato=I2CMasterDataGet(Base);`
- `I2CMasterControl(Base,Comando);`
  - Comando: en este caso: I2C_MASTER_CMD_SINGLE_RECEIVE (7)
- `I2CMasterBusy(Base);`
  - `0 si libre, 1 si Busy`



Ejemplo 6

- Control de dos displays con I2C
  - Placa realizada para el MSP430
- Conectados a dos PCF8574
  - Expansores de bus
  - Direcciones: 0x20 y 0x27
- Contador de 1s de periodo
  - Igual que en el ejemplo 4
- Sólo dos líneas (I2C7)



Configuración I2C7:

- Habilitar periférico y puerto
- Configurar pines
- Configurar I2C

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C7);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);

GPIOPinConfigure(GPIO_PD0_I2C7SCL);
GPIOPinTypeI2CSCL(GPIO_PORTD_BASE, GPIO_PIN_0);

GPIOPinConfigure(GPIO_PD1_I2C7SDA);
GPIOPinTypeI2C(GPIO_PORTD_BASE, GPIO_PIN_1);

I2CMasterInitExpClk(I2C7_BASE, RELOJ, false );
I2CMasterEnable(I2C7_BASE);
```

Bucle principal

- Mandar el MSB y el LSB

```
I2CMasterSlaveAddrSet(I2C7_BASE, I2CSAMSD, 0);
I2CMasterDataPut(I2C7_BASE, display[Decenas]);
I2CMasterControl(I2C7_BASE, I2C_MASTER_CMD_SINGLE_SEND);
while(!(I2CMasterBusBusy(I2C7_BASE)));
while((I2CMasterBusBusy(I2C7_BASE)));

I2CMasterSlaveAddrSet(I2C7_BASE, I2CSALSD, 0);
I2CMasterDataPut(I2C7_BASE, display[Unidades]);
I2CMasterControl(I2C7_BASE, I2C_MASTER_CMD_SINGLE_SEND);
while(!(I2CMasterBusBusy(I2C7_BASE)));
while((I2CMasterBusBusy(I2C7_BASE)));
```

- Doble comprobación del IDLE:
  - Si no se hace así, comprueba el bus *antes* de iniciar la comunicación, con lo que *siempre* está IDLE

Ejemplo 6(BIS)

- Manejo del sensor de luz y temperatura del Sensors Boosterpack
- Funciones de lectura y escritura de datos de 16 bits en I2C
- Librerías existentes para otro microcontrolador, adaptadas al TM4C1294:
  - HAL_I2C.C: Funciones de bajo nivel (leer / escribir 16 bits)
  - HAL_OPT3001.C: Funciones de nivel intermedio (leer /escribir el dispositivo concreto)
  - HAL_TMP007.C: Funciones para el manejo del sensor de temperatura

HAL_I2C

- Detección de Sensor Boosterpack:
  - Configurar pines como entradas y leerlos
  - Si BP, los pines I2C tendrán Pull-ups (1)

```
uint8_t Detecta_BP(int pos){
    int resul=0;
    switch (pos){
    case 1:
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
        GPIOPinTypeGPIOInput(GPIO_PORTB_BASE, GPIO_PIN_2|GPIO_PIN_3);
           GPIOPadConfigSet(GPIO_PORTB_BASE,GPIO_PIN_2|GPIO_PIN_3,GPIO_STRENGTH_2MA,
           GPIO_PIN_TYPE_STD_WPD);
        resul+=GPIOPinRead(GPIO_PORTB_BASE,GPIO_PIN_2 |GPIO_PIN_3);
        break;
    case 2:
//[... LO MISMO PARA LA SEGUNDA POSICIÓN]
return resul;
}
```

HAL_I2C

- Configura BP en la posición detectada:
  - I2C_Base: variable global (I2C0_BASE si BP en pos. 1, I2C2_BASE si BP en pos. 2)
  - Configuración GPIO's según posición

```
void Conf_Boosterpack(int pos, int RELOJ){
    switch (pos){
    case 1:
        I2C_Base=I2C0_BASE;
        SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
        GPIOPinConfigure(GPIO_PB2_I2C0SCL);
        GPIOPinTypeI2CSCL(GPIO_PORTB_BASE, GPIO_PIN_2);
        GPIOPinConfigure(GPIO_PB3_I2C0SDA);
        GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_3);
        I2CMasterInitExpClk(I2C0_BASE, RELOJ, true );
        I2CMasterEnable(I2C0_BASE);
        break;
    case 2:
 //[Lo mismo para el BP2]
```

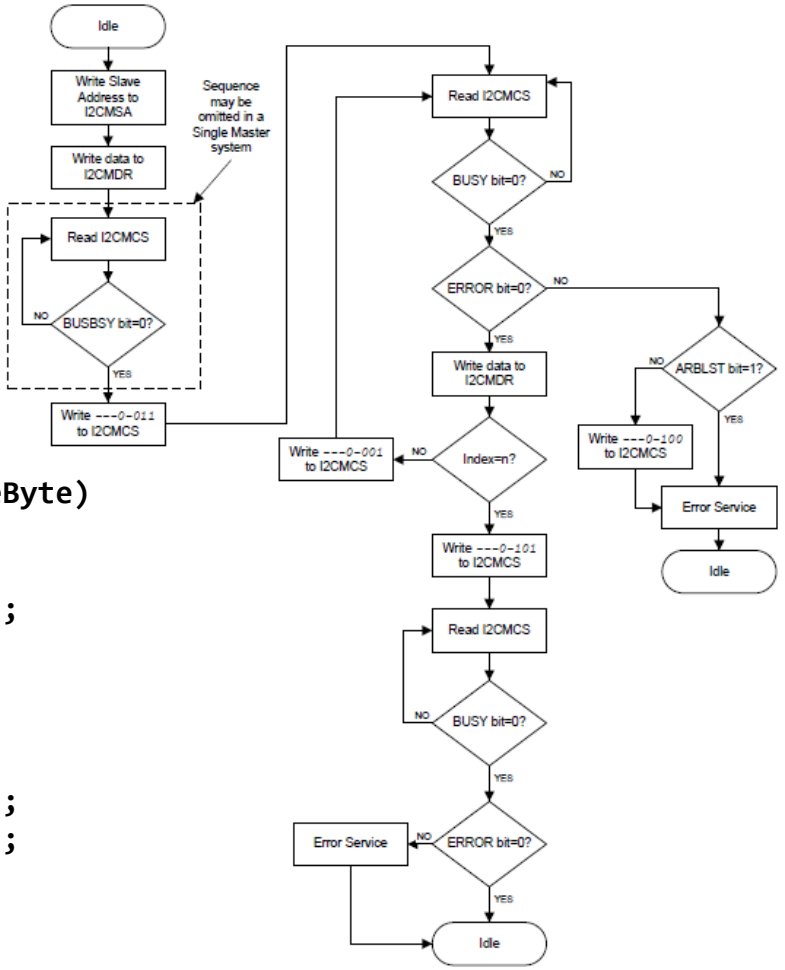
HAL_I2C

- Test_I2C_Dir: Para comprobar si hay un determinado elemento en una dirección I2C
  - Intentar escribir.
  - Leer si hay error de ACK
- Necesario para ver si hay TMP007

```
uint8_t Test_I2C_Dir(uint8_t DIR)
{
    I2CMasterSlaveAddrSet(I2C_Base, DIR, 1);
    I2CMasterControl(I2C_Base, I2C_MASTER_CMD_SINGLE_SEND);
    SysCtlDelay(20*MSEC);
    return !(I2CMasterErr(I2C_Base));
}
```

HAL_I2C

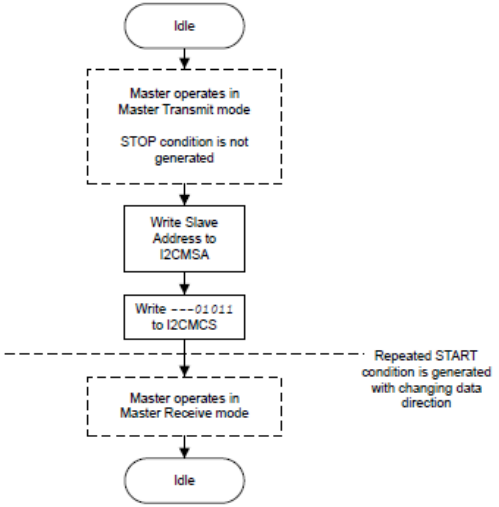
- Función para escribir:
  - Mandar dirección del registro
  - Mandar los datos
  - Seguir el protocolo:

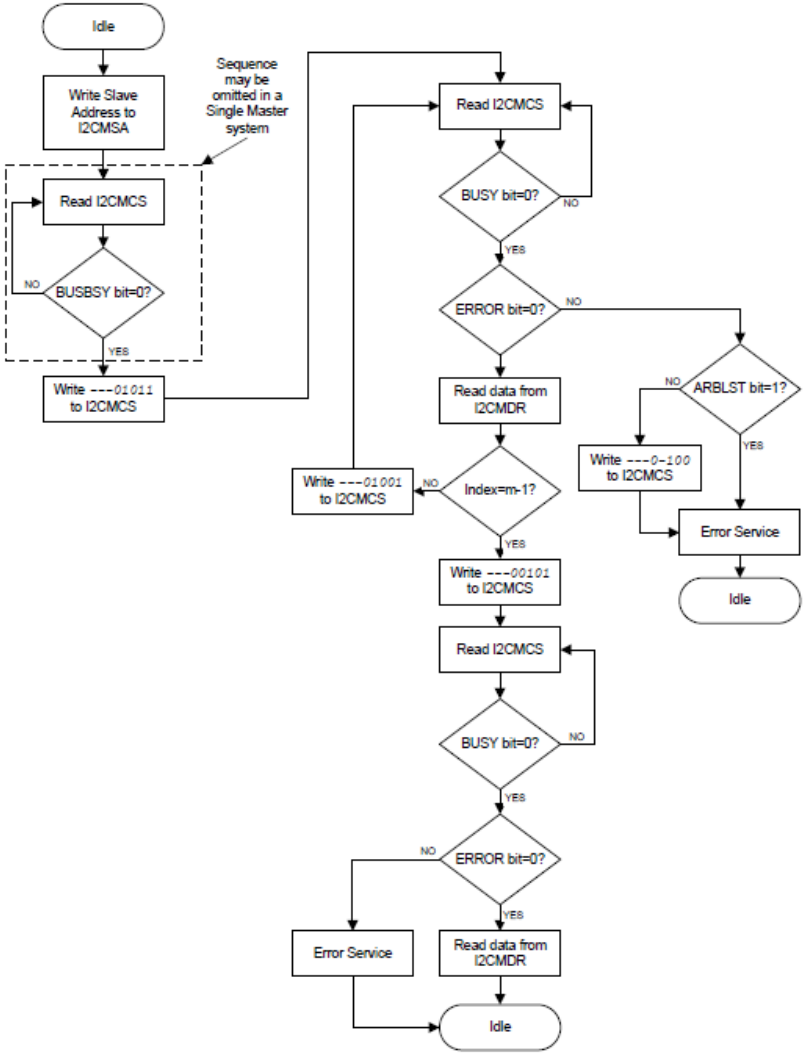



```
void I2C_write16 (unsigned char pointer, unsigned int writeByte)
{
I2CMasterDataPut(I2C_Base, pointer);
I2CMasterControl(I2C_Base, I2C_MASTER_CMD_BURST_SEND_START);
    Espera_I2C(I2C_Base);
I2CMasterDataPut(I2C_Base,(unsigned char)(writeByte>>8) );
I2CMasterControl(I2C_Base, I2C_MASTER_CMD_BURST_SEND_CONT);
    Espera_I2C(I2C_Base);
I2CMasterDataPut(I2C_Base, (unsigned char)(writeByte&0xFF));
I2CMasterControl(I2C_Base,I2C_MASTER_CMD_BURST_SEND_FINISH);
    Espera_I2C(I2C_Base);
}
```

HAL_I2C

- Para Leer:
  - En modo escritura se manda la dirección del registro a leer
  - Se pasa a modo lectura y se leen 2 bytes





HAL_I2C

- Para Leer:
  - En modo escritura se manda la dirección del registro a leer
  - Se pasa a modo lectura y se leen 2 bytes

```
int I2C_read16(unsigned int slaveAdr, unsigned char writeByte)
{
    volatile int val = 0;
    volatile int valScratch = 0;
    I2CMasterSlaveAddrSet(I2C_Base, slaveAdr, 0);//Modo ESCRITURA
    I2CMasterDataPut(I2C_Base, writeByte);
    I2CMasterControl(I2C_Base, I2C_MASTER_CMD_BURST_SEND_START); //esc. Multiple
    Espera_I2C(I2C_Base);
    I2CMasterSlaveAddrSet(I2C_Base, slaveAdr, 1);  //MODO LECTURA
    I2CMasterControl(I2C_Base, I2C_MASTER_CMD_BURST_RECEIVE_START); //Lect. Multiple
    Espera_I2C(I2C_Base);
    val=I2CMasterDataGet(I2C_Base);
    I2CMasterControl(I2C_Base,I2C_MASTER_CMD_BURST_RECEIVE_FINISH); //Lect. Final
    Espera_I2C(I2C_Base);
    val = (val << 8);
    valScratch=I2CMasterDataGet(I2C_Base);
    return(val|valScratch);
}
```

HAL_OPT3001

- Funciones básicas:
  - Configuración inicial:
  - Lectura del resultado:
    - Lee el sensor y calcula el valor (viene comprimido)

```
void OPT3001_init()
{
    /* Specify slave address for OPT3001 */
    I2C_setslave(OPT3001_SLAVE_ADDRESS);
    /* Set Default configuration for OPT3001*/
    I2C_write16(CONFIG_REG, DEFAULT_CONFIG_100);
}
```

```
float OPT3001_getLux()
{
    uint16_t exponent = 0;
    float result = 0;
    int16_t raw;
    raw = I2C_read16(OPT3001_SLAVE_ADDRESS,RESULT_REG);
    /*Convert to LUX*/
    //extract result & exponent data from raw readings
    result = raw&0x0FFF;
    exponent = (raw>>12)&0x000F;
    //convert raw readings to LUX
 result=result*(0.01*exp2(exponent));

    return result;
}
```

HAL_TMP007

- Funciones básicas:
  - Configuración:

```
bool sensorTmp007Enable(bool enable)
{
    I2C_setslave(TMP007_I2C_ADDRESS);
    if (enable){ val = TMP007_VAL_CONFIG_ON; }
    else { val = TMP007_VAL_CONFIG_OFF;}
I2C_write16(TMP007_REG_ADDR_CONFIG, val);
I2C_write16(TMP007_REG_ADDR_TC0_COEFFICIENT, TMP007_VAL_TC0);
I2C_write16(TMP007_REG_ADDR_TC1_COEFFICIENT, TMP007_VAL_TC1);
return (true);
}
```

  - Lectura:
    - T ambiente
    - T Objeto

```
bool sensorTmp007Read(int16_t *rawTemp, int16_t *rawObjTemp)
{
    int16_t T_O, T_L;
bool success;
val = I2C_read16(TMP007_I2C_ADDRESS, TMP007_REG_ADDR_STATUS);
success = val & CONV_RDY_BIT;
if (success)
{
T_L=(int) I2C_read16(TMP007_I2C_ADDRESS, TMP007_REG_ADDR_LOCAL_TEMP);
T_O =(int) I2C_read16(TMP007_I2C_ADDRESS, TMP007_REG_ADDR_OBJ_TEMP);
}
*rawTemp = T_L>>2;
*rawObjTemp = T_O>>2;
return (success);
}
```

Programa principal

- Detección de BP:

```
if(Detecta_BP(1))
   {
   UARTprintf("\n  BOOSTERPACK detectado en posicion 1");
   UARTprintf("\n   Configurando puerto I2C0");
       Conf_Boosterpack(1, RELOJ);
   }
   else if(Detecta_BP(2))
       {
       UARTprintf("\n  BOOSTERPACK detectado en posicion 2");
       UARTprintf("\n   Configurando puerto I2C2");
       Conf_Boosterpack(2, RELOJ);
       }
   else
       {
       UARTprintf("\n  Ningun BOOSTERPACK detectado   :-/  ");
       UARTprintf("\n              Saliendo");
       return 0;
       }
```

Programa principal

- Inicializa los sensores
- Comprueba el ID del OPT3001 y del TMP007
- Muestra información

```
Hay_OPT=Test_I2C_Dir(OPT3001_SLAVE_ADDRESS);
if(Hay_OPT){
    UARTprintf("\n\n Inicializando OPT3001...\n");
    OPT3001_init();
    UARTprintf("Hecho!\n");
    UARTprintf("Leyendo DevID...\n");
    DevID=OPT3001_readDeviceId();
    UARTprintf("DevID= 0X%x \n", DevID);
    UARTprintf("Leyendo Manufacturer ID...\n");
    DevID=OPT3001_readManufacturerId();
    }
else
    {
    UARTprintf("OPT3001 no encontrado\n");
    }
```

Programa principal

- Bucle:
  - Espera a interrupción de timer
  - lee los valores y los manda a la UART

```
while(1)
{
SysCtlSleep();
if(Hay_OPT){
I2C_setslave(OPT3001_SLAVE_ADDRESS);
lux=OPT3001_getLux();
sprintf(string," %5.3fLux\011 ||",lux);
UARTprintf(string);
}
if(Hay_TMP)
{
sensorTmp007Read(&T_amb, &T_obj);
sensorTmp007Convert(T_amb, T_obj, &Tf_obj, &Tf_amb);
sprintf(string," T_a:%2.4f, T_o:%2.4f ", Tf_amb, Tf_obj);
UARTprintf(string);
}
UARTprintf("\n");
}
```