Práctica 2

Segundo día

ec

¿Qué más hay que hacer?

- Utilizar las **interrupciones vectorizadas**

1. Completar las rutinas de iniciacion del dispositivo boton, y su rutina de tratamiento de interrupcion en **boton.c**
2. Completar las rutinas de iniciacion del dispositivo temporizador, y su rutina de tratamiento de interrupcion en **timer.c**
3. Completar la rutina para mostrar el valor del contador en el display de 8 segmentos en el archivo **8seg.c**

¿Qué más hay que hacer?

- Todo se va a programar en C
  - Es necesario estudiar en detalle el archivo 44blib.c
  - Es necesario estudiar en detalle el archivo 44b.h

Interrupciones vectorizadas en ARM

Instrucción	Vector
ldr pc,=HandlerEINT0	0x20
ldr pc,=HandlerEINT1	0x24
…	…
ldr pc,=HandlerEINT4567	0x30
…	…
ldr pc,=HandlerTIMER0	0x60
ldr pc,=HandlerTIMER1	0x64

- La gestión es similar a la gestión de excepciones:
  - Cuando un periférico interrumpe, el controlador de interrupciones genera su vector
  - Esa dirección de memoria (ROM) salta a un Handler que accede a una tabla donde está almacenado el puntero a la subrutina de tratamiento de ese periférico

Interrupciones vectorizadas

- ¿Cómo se rellena la tabla de punteros a subrutinas en C?
  1. Hay que definir que esa función de C se corresponde con una interrupción

```
void Eint4567_ISR(void) __attribute__ ((interrupt ("IRQ")));

void Eint4567_init(void);
```

boton.c

  2. Hay que definir internamente lo que hace esa interrupción (seguir los mismo pasos que en la parte A, pero todo en C) e incializar el controlador.
  3. Una vez que tenemos la función definida se puede rellenar la tabla

```
pISR_EINT4567 = (int)Eint4567_ISR;
```

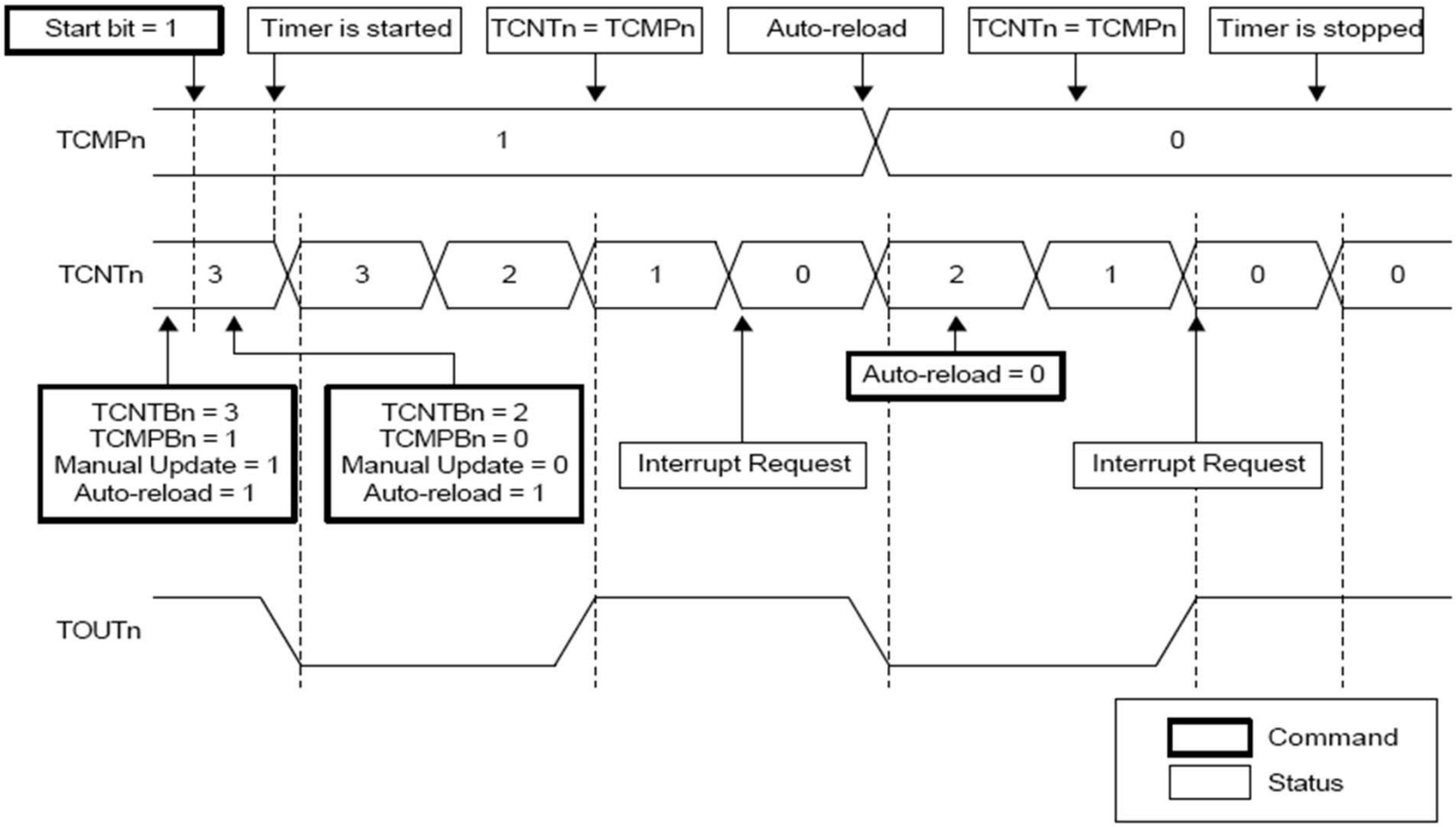
(*Se hace lo mismo para el timer p**ISR_TIMER0** = **(unsigned)timer_ISR***;)

- ¿En qué dirección está la tabla? ¿Es continuación de la tabla de excepciones?
- ¿Qué posiciones de memoria en la tabla se corresponden con ISR_EINT4567 e ISR_TIMER0?

Gestión del Timer

- Registros
  - TCFG0 (Timer configuration register 0)
    - Permite configurar los módulos de pre-escalado
  - TCFG1 (Timer configuration register 1)
    - Permite configurar cual es el temporizador que usará el DMA y para cada temporizador permite seleccionar la salida del divisor de frecuencia
  - TCON (Timer control register)
    - Permite controlar el comportamiento de los temporizadores (start/stop, auto-reload, etc.)
  - TCNTB0-5 (Count buffer register 0-5)
    - Registro de buffer del valor de inicialización
  - TCMPB0-5 (Compare buffer register 0-5)
    - Registro de buffer del valor de comparación
  - TCNTO0-5 (Count observation register 0-5)
    - Registro que permite consultar el valor actual del temporizador

Gestión del Timer



Configuración de las interrupciones

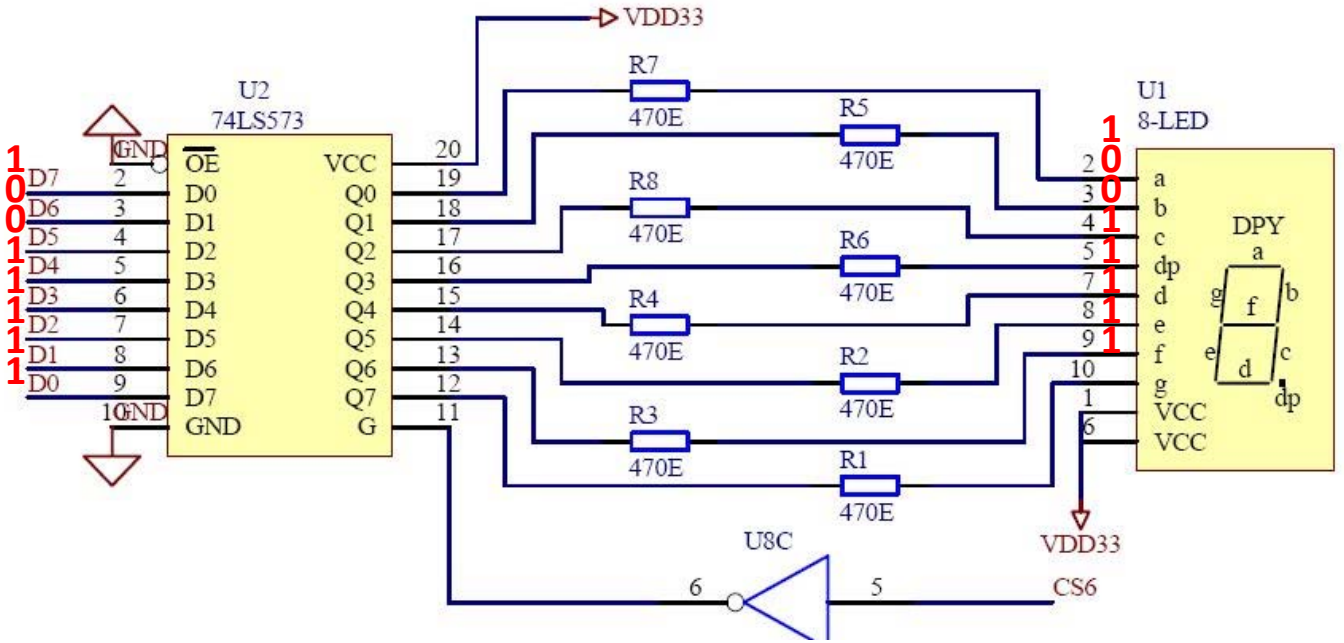
Esquema de la función timer_init (**timer.c**)

```
void timer_init() {
  // Habilitamos las interrupciones, IRQ con vectores
      rINTMOD = ¿?;
      rINTCON = ¿?;
  /// Desenmascaramos EINT1 y TIMER0
  /// Establecemos la función timer_int como ISR para TIMER0
  pISR_TIMER0=(unsigned)timer_ISR;
  // Configuramos el timer 0
  rTCFG0 = 63; rTCFG1 = 0x0; rTCNTB0 = 65535; rTCMPB0 = 12800;
  // Actualizamos TCON0
  rTCON = 0x2;
  // Habilitamos el TIMER
  rTCON  = 0x09;
  rI_ISPC = BIT_TIMER0; // Limpiamos INT pendientes
}
```

Hay que programar algo parecido también para *EINT4567_init* en *boton.c*

Display de 8 segmentos

- En el banco 6 del espacio de direcciones: 0x02140000 – 0x0217FFFF
  - #define LED8ADDR (*(volatile unsigned char *)(0x2140000))
- Para que se ilumine un segmento hay que escribir 0 en el bit correspondiente.
  - LED8ADDR = 0x0 enciende todos los segmentos
  - LED8ADDR = 0x9F enciende los segmentos b y c



Display de 8 segmentos

Definición de segmentos

```
// ...
#define SEGMENT_A 0x80
#define SEGMENT_B 0x40
/** Definir desde SEGMENT_C a SEGMENT_F **/
#define SEGMENT_G 0x01
#define SEGMENT_P 0x10

#define DIGIT_F  (SEGMENT_A | SEGMENT_G | SEGMENT_E | SEGMENT_F)
/** Definir desde DIGIT_E hasta DIGIT_1 **/
#define DIGIT_0  (SEGMENT_A | SEGMENT_B | SEGMENT_C | SEGMENT_D |
SEGMENT_E | SEGMENT_G)
```