

Chapter 6

Parallel Input/Output Control

This section explains software controls related to parallel input/output (I/O) and pin control. The MC9S08SH32 has three parallel I/O ports which include a total of 23 I/O pins and one output-only pin. See [Chapter 2, “Pins and Connections,”](#) for more information about pin assignments and external hardware considerations of these pins.

Many of these pins are shared with on-chip peripherals such as timer systems, communication systems, or pin interrupts as shown in [Table 2-1](#). The peripheral modules have priority over the general-purpose I/O functions so that when a peripheral is enabled, the I/O functions associated with the shared pins are disabled.

After reset, the shared peripheral functions are disabled and the pins are configured as inputs ($PTxDDn = 0$). The pin control functions for each pin are configured as follows: slew rate disabled ($PTxSEn = 0$), low drive strength selected ($PTxDSn = 0$), and internal pull-ups disabled ($PTxPEN = 0$).

NOTE

Not all general-purpose I/O pins are available on all packages. To avoid extra current drain from floating input pins, the user's reset initialization routine in the application program must either enable on-chip pull-up devices or change the direction of unconnected pins to outputs so the pins do not float.

6.1 Port Data and Data Direction

Reading and writing of parallel I/Os are performed through the port data registers. The direction, either input or output, is controlled through the port data direction registers. The parallel I/O port function for an individual pin is illustrated in the block diagram shown in [Figure 6-1](#).

The data direction control bit ($PTxDDn$) determines whether the output buffer for the associated pin is enabled, and also controls the source for port data register reads. The input buffer for the associated pin is always enabled unless the pin is enabled as an analog function or is an output-only pin.

When a shared digital function is enabled for a pin, the output buffer is controlled by the shared function. However, the data direction register bit will continue to control the source for reads of the port data register.

When a shared analog function is enabled for a pin, both the input and output buffers are disabled. A value of 0 is read for any port data bit where the bit is an input ($PTxDDn = 0$) and the input buffer is disabled. In general, whenever a pin is shared with both an alternate digital function and an analog function, the analog function has priority such that if both the digital and analog functions are enabled, the analog function controls the pin.

It is a good programming practice to write to the port data register before changing the direction of a port pin to become an output. This ensures that the pin will not be driven momentarily with an old data value that happened to be in the port data register.

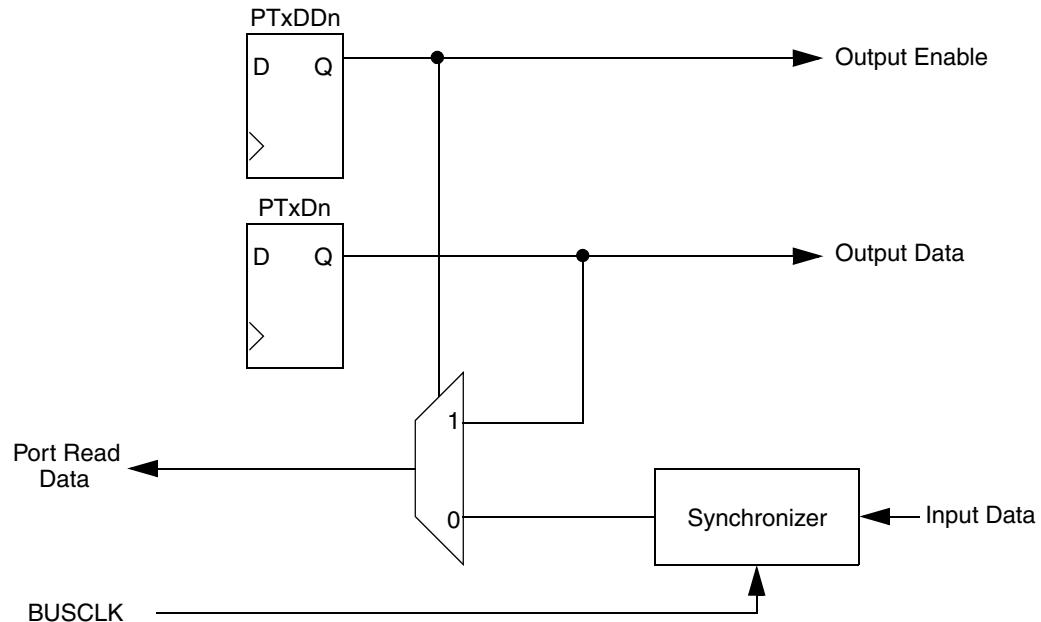


Figure 6-1. Parallel I/O Block Diagram

6.2 Pull-up, Slew Rate, and Drive Strength

Associated with the parallel I/O ports is a set of registers located in the high page register space that operate independently of the parallel I/O registers. These registers are used to control pull-ups, slew rate, and drive strength for the pins.

An internal pull-up device can be enabled for each port pin by setting the corresponding bit in the pull-up enable register (PTxPEN). The pull-up device is disabled if the pin is configured as an output by the parallel I/O control logic or any shared peripheral function regardless of the state of the corresponding pull-up enable register bit. The pull-up device is also disabled if the pin is controlled by an analog function.

Slew rate control can be enabled for each port pin by setting the corresponding bit in the slew rate control register (PTxSEn). When enabled, slew control limits the rate at which an output can transition in order to reduce EMC emissions. Slew rate control has no effect on pins that are configured as inputs.

An output pin can be selected to have high output drive strength by setting the corresponding bit in the drive strength select register (PTxDSn). When high drive is selected, a pin is capable of sourcing and sinking greater current. Even though every I/O pin can be selected as high drive, the user must ensure that the total current source and sink limits for the MCU are not exceeded. Drive strength selection is intended to affect the DC behavior of I/O pins. However, the AC behavior is also affected. High drive allows a pin to drive a greater load with the same switching speed as a low drive enabled pin into a smaller load. Because of this, the EMC emissions may be affected by enabling pins as high drive.

6.3 Ganged Output

The MC9S08SH32 Series devices contain a feature that allows for up to eight port pins to be tied together externally to allow higher output current drive. The ganged output drive control register (GNGC) is a write-once register that is used to enable the ganged output feature and select which port pins will be used as ganged outputs. The GNGEN bit in GNGC enables ganged output. The GNGPS[7:1] bits are used to select which pin will be part of the ganged output.

When GNGEN is set, any pin that is enabled as a ganged output will be automatically configured as an output and follow the data, drive strength and slew rate control of PTC0. The ganged output drive pin mapping is shown in [Table 6-1](#).

NOTE

See the DC characteristics in the electrical section for maximum Port I/O currents allowed for this MCU.

When a pin is enabled as ganged output, this feature will have priority over any digital module. An enabled analog function will have priority over the ganged output pin. See [Table 2-1](#) for information on pin priority.

Table 6-1. Ganged Output Pin Enable

	GNGC Register Bits							
	GNGPS7	GNGPS6	GNGPS5	GNGPS4	GNGPS3	GNGPS2	GNGPS1	GNGEN ¹
Port Pin ²	PTB5	PTB4	PTB3	PTB2	PTC3	PTC2	PTC1	PTC0
Data Direction Control	Pin is automatically configured as output when pin is enabled as ganged output.							
Data Control	PTCD0 in PTCD controls data value of output							
Drive Strength Control	PTCDS0 in PTCDS controls drive strength of output							
Slew Rate Control	PTCSE0 in PTCSE controls slew rate of output							

¹ Ganged output on PTC3-PTC0 not available on 16-pin packages, however PTC0 control registers are still used to control ganged output.

² When GNGEN = 1, PTC0 is forced to an output, regardless of the value in PTCD0 in PTCD.

6.4 Pin Interrupts

Port A[3:0] and port B[3:0] pins can be configured as external interrupt inputs and as an external means of waking the MCU from stop3 or wait low-power modes.

The block diagram for the pin interrupts is shown [Figure 6-2](#).

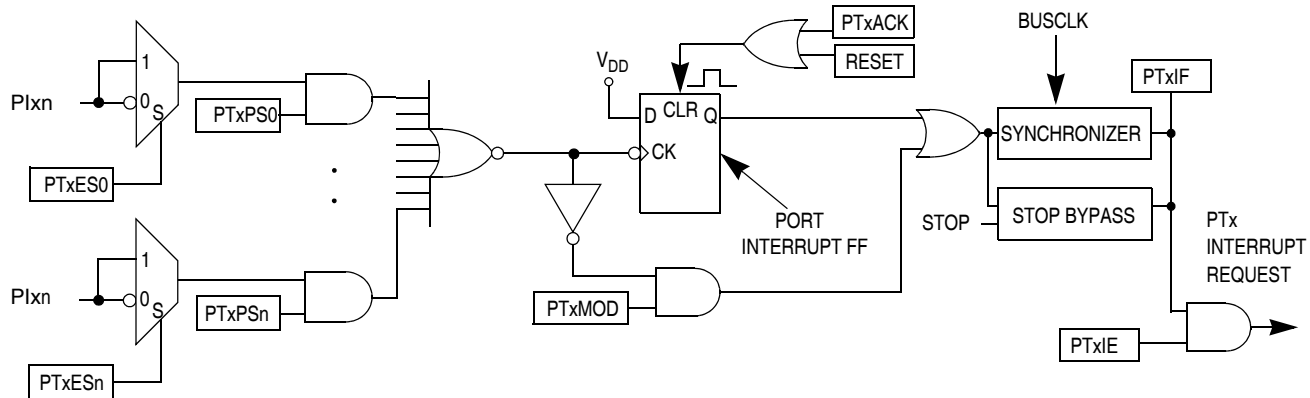


Figure 6-2. Pin Interrupt Block Diagram

Writing to the PTxPSn bits in the port interrupt pin enable register (PTxPS) independently enables or disables each port pin interrupt. Each port can be configured as edge sensitive or edge and level sensitive based on the PTxMOD bit in the port interrupt status and control register (PTxSC). Edge sensitivity can be software programmed to be either falling or rising; the level can be either low or high. The polarity of the edge or edge and level sensitivity is selected using the PTxESn bits in the port interrupt edge select register (PTxES).

Synchronous logic is used to detect edges. Prior to detecting an edge, enabled pin interrupt inputs must be at the deasserted logic level. A falling edge is detected when an enabled port input signal is seen as a logic 1 (the deasserted level) during one bus cycle and then a logic 0 (the asserted level) during the next cycle. A rising edge is detected when the input signal is seen as a logic 0 during one bus cycle and then a logic 1 during the next cycle.

6.4.1 Edge-Only Sensitivity

A valid edge on an enabled pin interrupt sets PTxIF in PTxSC. If PTxIE in PTxSC is set, an interrupt request is presented to the CPU. To clear PTxIF, write a 1 to PTxACK in PTxSC.

NOTE

If a pin is enabled for interrupt on edge-sensitive only, a falling (or rising) edge on the pin does not latch an interrupt request if another pin interrupt is already asserted.

To prevent losing an interrupt request on one pin because another pin is asserted, software can disable the asserted pin interrupt while having the unasserted pin interrupt enabled. The asserted status of a pin is reflected by its associated I/O general purpose data register.

6.4.2 Edge and Level Sensitivity

A valid edge or level on an enabled pin interrupt sets PTxIF in PTxSC. If PTxIE in PTxSC is set, an interrupt request is presented to the CPU. To clear PTxIF, write a 1 to PTxACK in PTxSC provided all enabled pin interrupt inputs are at their de-asserted levels. PTxIF remains set if any enabled pin interrupt is asserted while attempting to clear by writing a 1 to PTxACK.

6.4.3 Pull-up/Pull-down Resistors

The pin interrupts can be configured to use an internal pull-up/pull-down resistor using the associated I/O port pull-up enable register. If an internal resistor is enabled, the PTxES register is used to select whether the resistor is a pull-up (PTxESn = 0) or a pull-down (PTxESn = 1).

6.4.4 Pin Interrupt Initialization

When a pin interrupt is first enabled, it is possible to get a false interrupt flag. To prevent a false interrupt request during pin interrupt initialization, the user should do the following:

1. Mask interrupts by clearing PTxIE in PTxSC.
2. Select the pin polarity by setting the appropriate PTxESn bits in PTxES.
3. If using internal pull-up/pull-down device, configure the associated pull enable bits in PTxPE.
4. Enable the interrupt pins by setting the appropriate PTxPSn bits in PTxPS.
5. Write to PTxACK in PTxSC to clear any false interrupts.
6. Set PTxIE in PTxSC to enable interrupts.

6.5 Pin Behavior in Stop Modes

Pin behavior following execution of a STOP instruction depends on the stop mode that is entered. An explanation of pin behavior for the various stop modes follows:

- Stop2 mode is a partial power-down mode, whereby I/O latches are maintained in their state as before the STOP instruction was executed. CPU register status and the state of I/O registers should be saved in RAM before the STOP instruction is executed to place the MCU in stop2 mode. Upon recovery from stop2 mode, before accessing any I/O, the user should examine the state of the PPDF bit in the SPMSC2 register. If the PPDF bit is 0, I/O must be initialized as if a power on reset had occurred. If the PPDF bit is 1, I/O data previously stored in RAM, before the STOP instruction was executed, peripherals may require being initialized and restored to their pre-stop condition. The user must then write a 1 to the PPDACK bit in the SPMSC2 register. Access to I/O is now permitted again in the user application program.
- In stop3 mode, all I/O is maintained because internal logic circuitry stays powered up. Upon recovery, normal I/O function is available to the user.

6.6 Parallel I/O and Pin Control Registers

This section provides information about the registers associated with the parallel I/O ports. The data and data direction registers are located in page zero of the memory map. The pull up, slew rate, drive strength, and interrupt control registers are located in the high page section of the memory map.

Refer to tables in [Chapter 4, “Memory,”](#) for the absolute address assignments for all parallel I/O and their pin control registers. This section refers to registers and control bits only by their names. A Freescale Semiconductor-provided equate or header file normally is used to translate these names into the appropriate absolute addresses.

6.6.1 Port A Registers

Port A is controlled by the registers listed below.

The pins PTA4 and PTA5 are unique. PTA4 is output-only, so the control bits for the input function will not have any effect on this pin. PTA5, when configured as an output, is open drain.

NOTE

This PTA5 pin does not contain a clamp diode to V_{DD} and should not be driven above V_{DD} .

When the internal pullup device is enabled on PTA5 when used as an input or open drain output the voltage measured on PTA5 will not be pulled to V_{DD} . The internal gates connected to this pin are pulled to V_{DD} . If the PTA5 pin is required to drive to a V_{DD} level an external pullup should be used.

6.6.1.1 Port A Data Register (PTAD)

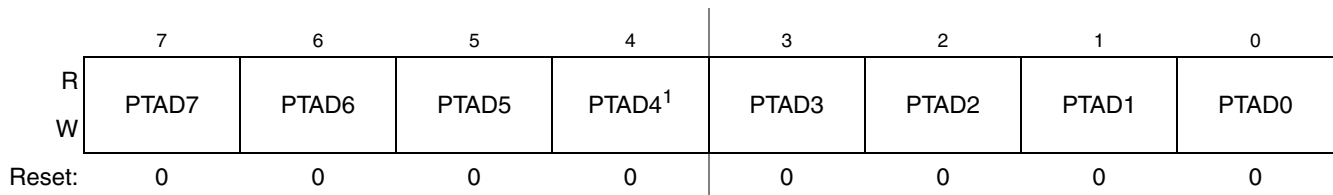


Figure 6-3. Port A Data Register (PTAD)

¹ Reads of bit PTAD4 always return the contents of PTAD4, regardless of the value stored in bit PTADD4.

Table 6-2. PTAD Register Field Descriptions

Field	Description
7:0 PTAD[7:0]	<p>Port A Data Register Bits — For port A pins that are inputs, reads return the logic level on the pin. For port A pins that are configured as outputs, reads return the last value written to this register.</p> <p>Writes are latched into all bits of this register. For port A pins that are configured as outputs, the logic level is driven out the corresponding MCU pin.</p> <p>Reset forces PTAD to all 0s, but these 0s are not driven out the corresponding pins because reset also configures all port pins as high-impedance inputs with pull-ups/pull-downs disabled.</p>

6.6.1.2 Port A Data Direction Register (PTADD)

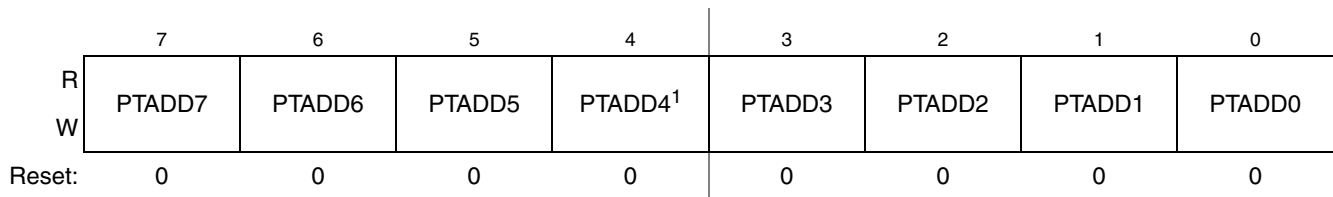


Figure 6-4. Port A Data Direction Register (PTADD)

¹ PTADD4 has no effect on the output-only PTA4 pin.

Table 6-3. PTADD Register Field Descriptions

Field	Description
7:0 PTADD[7:0]	Data Direction for Port A Bits — These read/write bits control the direction of port A pins and what is read for PTAD reads. 0 Input (output driver disabled) and reads return the pin value. 1 Output driver enabled for port A bit n and PTAD reads return the contents of PTADn.

6.6.1.3 Port A Pull Enable Register (PTAPE)

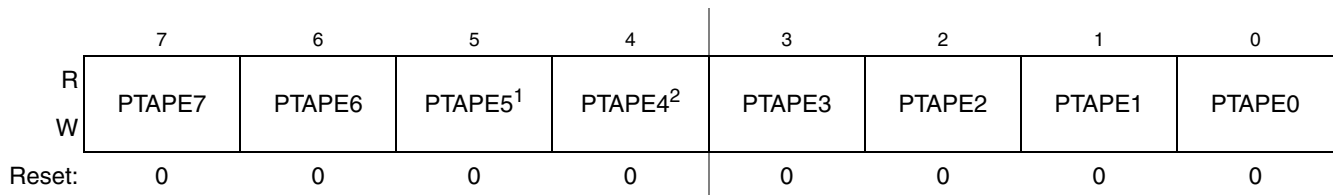


Figure 6-5. Internal Pull Enable for Port A Register (PTAPE)

¹ PTAPE5 can be used to pullup PTA5 when configured as open drain output pin, however pullup will not pull pin all the way to V_{DD}. An external pullup should be used if applications requires PTA5 to be driven to V_{DD}.

² PTAPE4 has no effect on the output-only PTA4 pin.

Table 6-4. PTAPE Register Field Descriptions

Field	Description
7:0 PTAPE[7:0]	Internal Pull Enable for Port A Bits — Each of these control bits determines if the internal pull-up or pull-down device is enabled for the associated PTA pin. For port A pins (except for PTA5) that are configured as outputs, these bits have no effect and the internal pull devices are disabled. 0 Internal pull-up/pull-down device disabled for port A bit n. 1 Internal pull-up/pull-down device enabled for port A bit n.

NOTE

Pull-down devices only apply when using pin interrupt functions, when corresponding edge select and pin select functions are configured to detect rising edges.

6.6.1.4 Port A Slew Rate Enable Register (PTASE)

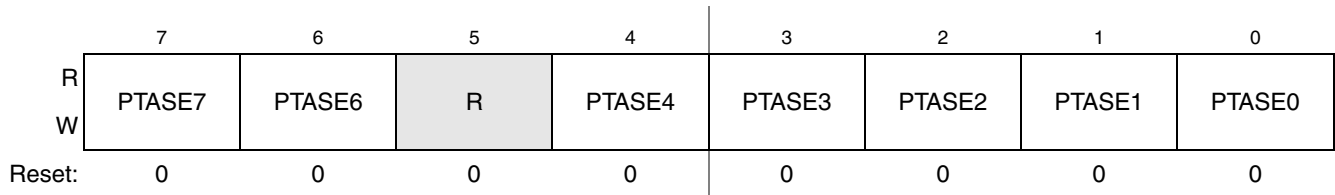


Figure 6-6. Slew Rate Enable for Port A Register (PTASE)

Table 6-5. PTASE Register Field Descriptions

Field	Description
7:6,4:0 PTASE [7:6, 4:0]	Output Slew Rate Enable for Port A Bits — Each of these control bits determines if the output slew rate control is enabled for the associated PTA pin. For port A pins that are configured as inputs, these bits have no effect. 0 Output slew rate control disabled for port A bit n. 1 Output slew rate control enabled for port A bit n.
5 Reserved	Reserved Bits — These bits are unused on this MCU, writes have no affect and could read as 1s or 0s.

6.6.1.5 Port A Drive Strength Selection Register (PTADS)

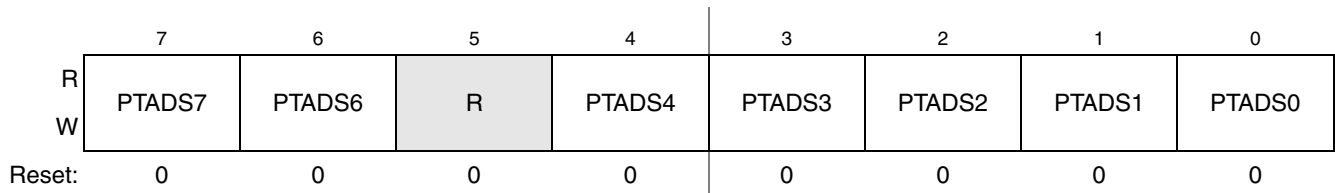


Figure 6-7. Drive Strength Selection for Port A Register (PTADS)

Table 6-6. PTADS Register Field Descriptions

Field	Description
7:6, 4:0 PTADS [7:6, 4:0]	Output Drive Strength Selection for Port A Bits — Each of these control bits selects between low and high output drive for the associated PTA pin. For port A pins that are configured as inputs, these bits have no effect. 0 Low output drive strength selected for port A bit n. 1 High output drive strength selected for port A bit n.
5 Reserved	Reserved Bits — These bits are unused on this MCU, writes have no affect and could read as 1s or 0s.

6.6.1.6 Port A Interrupt Status and Control Register (PTASC)

	7	6	5	4	3	2	1	0
R	0	0	0	0	PTAIF	0	PTAIE	PTAMOD
W						PTAACK		
Reset:	0	0	0	0	0	0	0	0

Figure 6-8. Port A Interrupt Status and Control Register (PTASC)

Table 6-7. PTASC Register Field Descriptions

Field	Description
3 PTAIF	Port A Interrupt Flag — PTAIF indicates when a port A interrupt is detected. Writes have no effect on PTAIF. 0 No port A interrupt detected. 1 Port A interrupt detected.
2 PTAACK	Port A Interrupt Acknowledge — Writing a 1 to PTAACK is part of the flag clearing mechanism. PTAACK always reads as 0.
1 PTAIE	Port A Interrupt Enable — PTAIE determines whether a port A interrupt is enabled. 0 Port A interrupt request not enabled. 1 Port A interrupt request enabled.
0 PTAMOD	Port A Detection Mode — PTAMOD (along with the PTAES bits) controls the detection mode of the port A interrupt pins. 0 Port A pins detect edges only. 1 Port A pins detect both edges and levels.

6.6.1.7 Port A Interrupt Pin Select Register (PTAPS)

	7	6	5	4	3	2	1	0
R	0	0	0	0	PTAPS3	PTAPS2	PTAPS1	PTAPS0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 6-9. Port A Interrupt Pin Select Register (PTAPS)

Table 6-8. PTAPS Register Field Descriptions

Field	Description
3:0 PTAPS[3:0]	Port A Interrupt Pin Selects — Each of the PTAPSn bits enable the corresponding port A interrupt pin. 0 Pin not enabled as interrupt. 1 Pin enabled as interrupt.

6.6.1.8 Port A Interrupt Edge Select Register (PTAES)

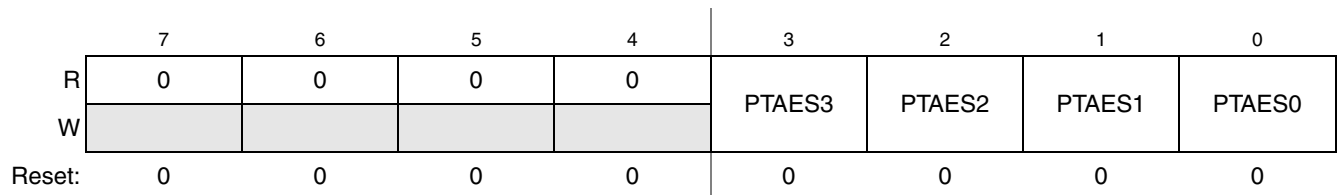


Figure 6-10. Port A Edge Select Register (PTAES)

Table 6-9. PTAES Register Field Descriptions

Field	Description
3:0 PTAES[3:0]	<p>Port A Edge Selects — Each of the PTAESn bits serves a dual purpose by selecting the polarity of the active interrupt edge as well as selecting a pull-up or pull-down device if enabled.</p> <p>0 A pull-up device is connected to the associated pin and detects falling edge/low level for interrupt generation.</p> <p>1 A pull-down device is connected to the associated pin and detects rising edge/high level for interrupt generation.</p>

6.6.2 Port B Registers

Port B is controlled by the registers listed below.

6.6.2.1 Port B Data Register (PTBD)

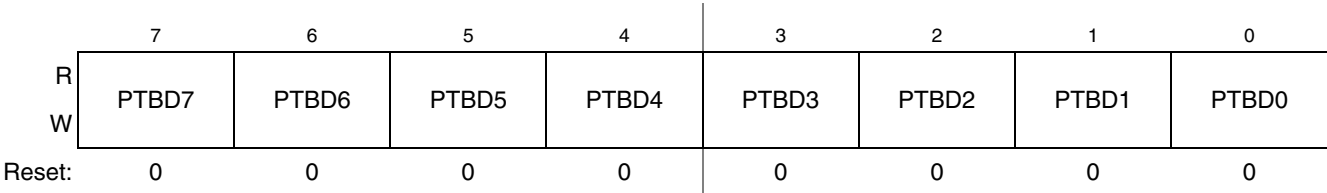


Figure 6-11. Port B Data Register (PTBD)

Table 6-10. PTBD Register Field Descriptions

Field	Description
7:0 PTBD[7:0]	Port B Data Register Bits — For port B pins that are inputs, reads return the logic level on the pin. For port B pins that are configured as outputs, reads return the last value written to this register. Writes are latched into all bits of this register. For port B pins that are configured as outputs, the logic level is driven out the corresponding MCU pin. Reset forces PTBD to all 0s, but these 0s are not driven out the corresponding pins because reset also configures all port pins as high-impedance inputs with pull-ups/pull-downs disabled.

6.6.2.2 Port B Data Direction Register (PTBDD)

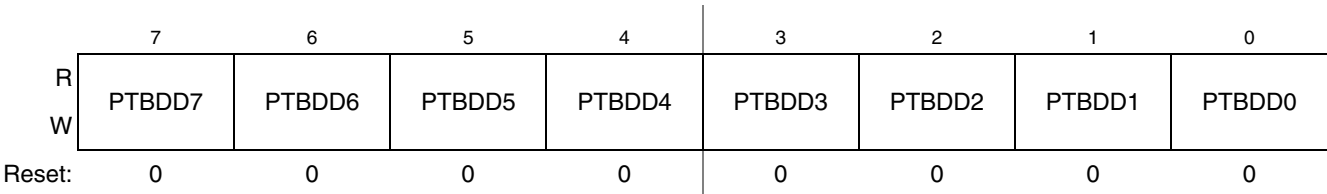


Figure 6-12. Port B Data Direction Register (PTBDD)

Table 6-11. PTBDD Register Field Descriptions

Field	Description
7:0 PTBDD[7:0]	Data Direction for Port B Bits — These read/write bits control the direction of port B pins and what is read for PTBD reads. 0 Input (output driver disabled) and reads return the pin value. 1 Output driver enabled for port B bit n and PTBD reads return the contents of PTBDn.

6.6.2.3 Port B Pull Enable Register (PTBPE)

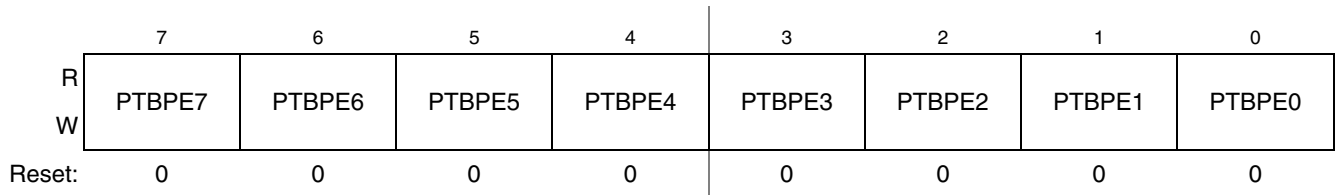


Figure 6-13. Internal Pull Enable for Port B Register (PTBPE)

Table 6-12. PTBPE Register Field Descriptions

Field	Description
7:0 PTBPE[7:0]	Internal Pull Enable for Port B Bits — Each of these control bits determines if the internal pull-up or pull-down device is enabled for the associated PTB pin. For port B pins that are configured as outputs, these bits have no effect and the internal pull devices are disabled. 0 Internal pull-up/pull-down device disabled for port B bit n. 1 Internal pull-up/pull-down device enabled for port B bit n.

NOTE

Pull-down devices only apply when using pin interrupt functions, when corresponding edge select and pin select functions are configured to detect rising edges.

6.6.2.4 Port B Slew Rate Enable Register (PTBSE)

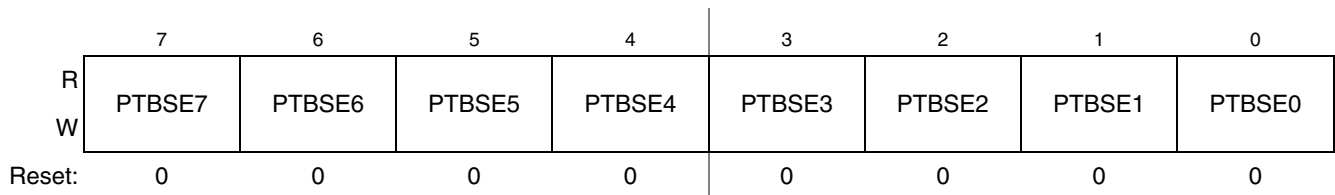


Figure 6-14. Slew Rate Enable for Port B Register (PTBSE)

Table 6-13. PTBSE Register Field Descriptions

Field	Description
7:0 PTBSE[7:0]	Output Slew Rate Enable for Port B Bits — Each of these control bits determines if the output slew rate control is enabled for the associated PTB pin. For port B pins that are configured as inputs, these bits have no effect. 0 Output slew rate control disabled for port B bit n. 1 Output slew rate control enabled for port B bit n.

6.6.2.5 Port B Drive Strength Selection Register (PTBDS)

	7	6	5	4	3	2	1	0
R	PTBDS7	PTBDS6	PTBDS5	PTBDS4	PTBDS3	PTBDS2	PTBDS1	PTBDS0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 6-15. Drive Strength Selection for Port B Register (PTBDS)

Table 6-14. PTBDS Register Field Descriptions

Field	Description
7:0 PTBDS[7:0]	Output Drive Strength Selection for Port B Bits — Each of these control bits selects between low and high output drive for the associated PTB pin. For port B pins that are configured as inputs, these bits have no effect. 0 Low output drive strength selected for port B bit n. 1 High output drive strength selected for port B bit n.

6.6.2.6 Port B Interrupt Status and Control Register (PTBSC)

	7	6	5	4	3	2	1	0
R	0	0	0	0	PTBIF	0	PTBIE	PTBMOD
W						PTBACK		
Reset:	0	0	0	0	0	0	0	0

Figure 6-16. Port B Interrupt Status and Control Register (PTBSC)

Table 6-15. PTBSC Register Field Descriptions

Field	Description
3 PTBIF	Port B Interrupt Flag — PTBIF indicates when a Port B interrupt is detected. Writes have no effect on PTBIF. 0 No Port B interrupt detected. 1 Port B interrupt detected.
2 PTBACK	Port B Interrupt Acknowledge — Writing a 1 to PTBACK is part of the flag clearing mechanism. PTBACK always reads as 0.
1 PTBIE	Port B Interrupt Enable — PTBIE determines whether a port B interrupt is enabled. 0 Port B interrupt request not enabled. 1 Port B interrupt request enabled.
0 PTBMOD	Port B Detection Mode — PTBMOD (along with the PTBES bits) controls the detection mode of the port B interrupt pins. 0 Port B pins detect edges only. 1 Port B pins detect both edges and levels.

6.6.2.7 Port B Interrupt Pin Select Register (PTBPS)

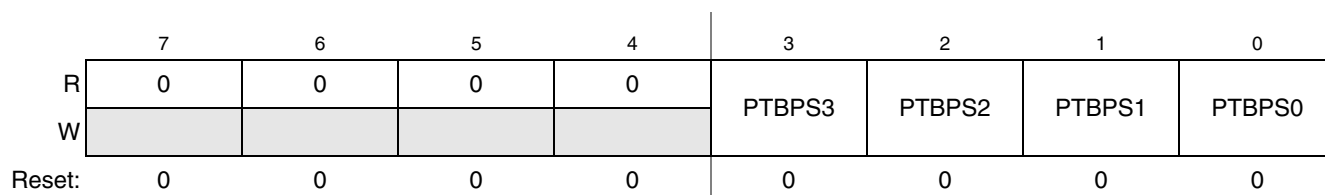


Figure 6-17. Port B Interrupt Pin Select Register (PTBPS)

Table 6-16. PTBPS Register Field Descriptions

Field	Description
3:0 PTBPS[3:0]	Port B Interrupt Pin Selects — Each of the PTBPSn bits enable the corresponding port B interrupt pin. 0 Pin not enabled as interrupt. 1 Pin enabled as interrupt.

6.6.2.8 Port B Interrupt Edge Select Register (PTBES)

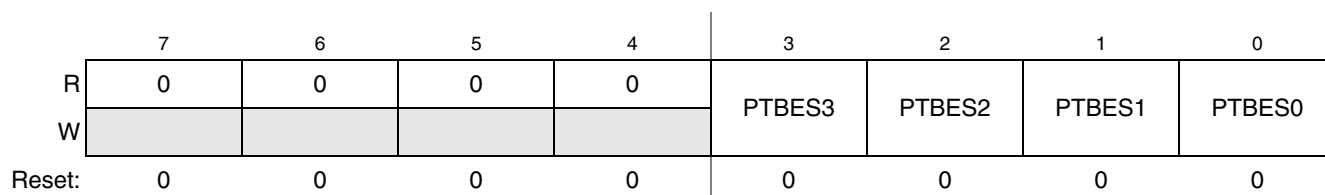


Figure 6-18. Port B Edge Select Register (PTBES)

Table 6-17. PTBES Register Field Descriptions

Field	Description
3:0 PTBES[3:0]	Port B Edge Selects — Each of the PTBESn bits serves a dual purpose by selecting the polarity of the active interrupt edge as well as selecting a pull-up or pull-down device if enabled. 0 A pull-up device is connected to the associated pin and detects falling edge/low level for interrupt generation. 1 A pull-down device is connected to the associated pin and detects rising edge/high level for interrupt generation.

6.6.3 Port C Registers

Port C is controlled by the registers listed below.

6.6.3.1 Port C Data Register (PTCD)

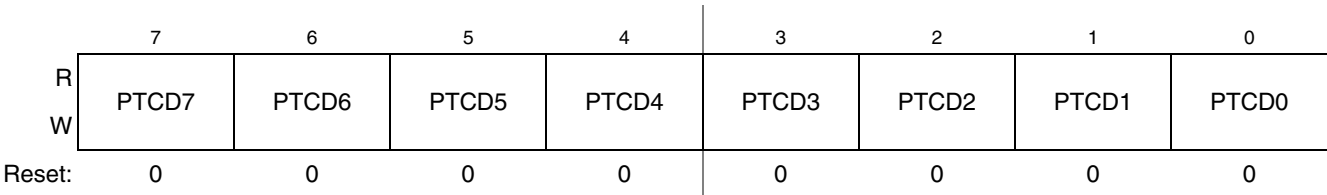


Figure 6-19. Port C Data Register (PTCD)

Table 6-18. PTCD Register Field Descriptions

Field	Description
7:0 PTCD[7:0]	Port C Data Register Bits — For port C pins that are inputs, reads return the logic level on the pin. For port C pins that are configured as outputs, reads return the last value written to this register. Writes are latched into all bits of this register. For port C pins that are configured as outputs, the logic level is driven out the corresponding MCU pin. Reset forces PTCD to all 0s, but these 0s are not driven out the corresponding pins because reset also configures all port pins as high-impedance inputs with pull-ups disabled.

6.6.3.2 Port C Data Direction Register (PTCDD)

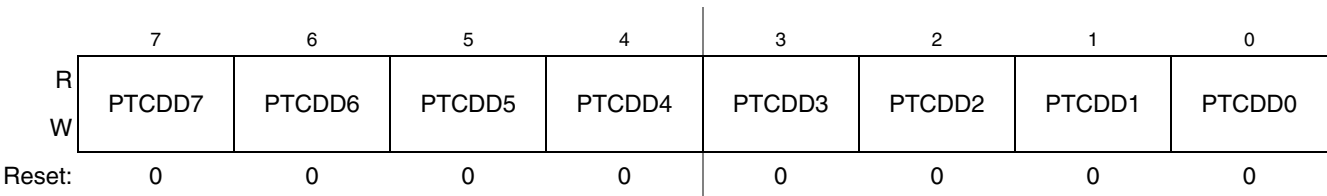


Figure 6-20. Port C Data Direction Register (PTCDD)

Table 6-19. PTCDD Register Field Descriptions

Field	Description
7:0 PTCDD[7:0]	Data Direction for Port C Bits — These read/write bits control the direction of port C pins and what is read for PTCD reads. 0 Input (output driver disabled) and reads return the pin value. 1 Output driver enabled for port C bit n and PTCD reads return the contents of PTCDn.

6.6.3.3 Port C Pull Enable Register (PTCPE)

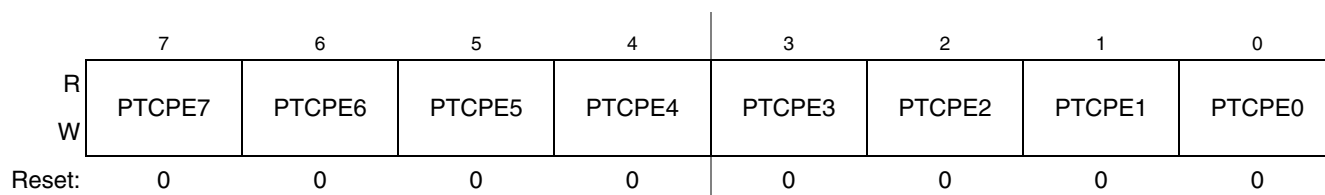


Figure 6-21. Internal Pull Enable for Port C Register (PTCPE)

Table 6-20. PTCPE Register Field Descriptions

Field	Description
7:0 PTCPE[7:0]	Internal Pull Enable for Port C Bits — Each of these control bits determines if the internal pull-up device is enabled for the associated PTC pin. For port C pins that are configured as outputs, these bits have no effect and the internal pull devices are disabled. 0 Internal pull-up device disabled for port C bit n. 1 Internal pull-up device enabled for port C bit n.

6.6.3.4 Port C Slew Rate Enable Register (PTCSE)

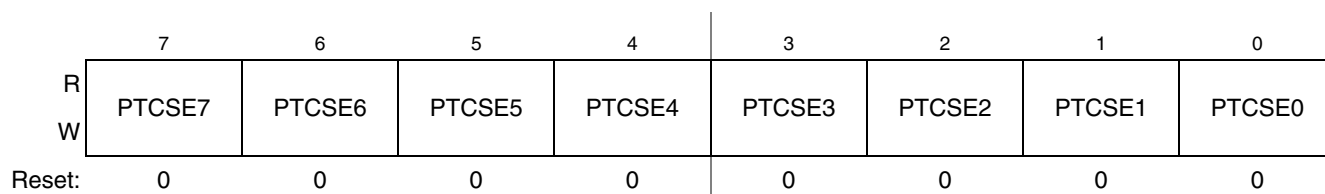


Figure 6-22. Slew Rate Enable for Port C Register (PTCSE)

Table 6-21. PTCSE Register Field Descriptions

Field	Description
7:0 PTCSE[7:0]	Output Slew Rate Enable for Port C Bits — Each of these control bits determines if the output slew rate control is enabled for the associated PTC pin. For port C pins that are configured as inputs, these bits have no effect. 0 Output slew rate control disabled for port C bit n. 1 Output slew rate control enabled for port C bit n.

6.6.3.5 Port C Drive Strength Selection Register (PTCDS)

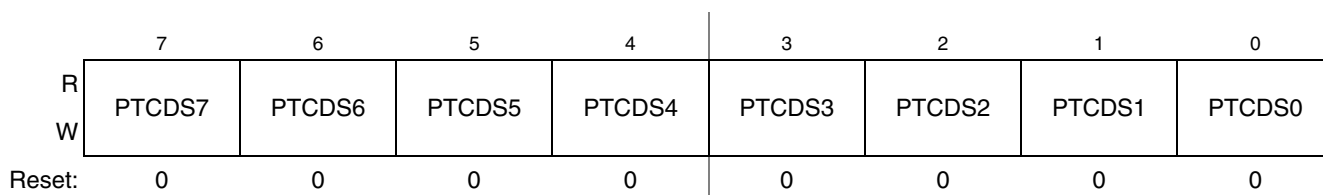


Figure 6-23. Drive Strength Selection for Port C Register (PTCDS)

Table 6-22. PTCDS Register Field Descriptions

Field	Description
7:0 PTCDS[7:0]	Output Drive Strength Selection for Port C Bits — Each of these control bits selects between low and high output drive for the associated PTC pin. For port C pins that are configured as inputs, these bits have no effect. 0 Low output drive strength selected for port C bit n. 1 High output drive strength selected for port C bit n.

6.6.3.6 Ganged Output Drive Control Register (GNGC)

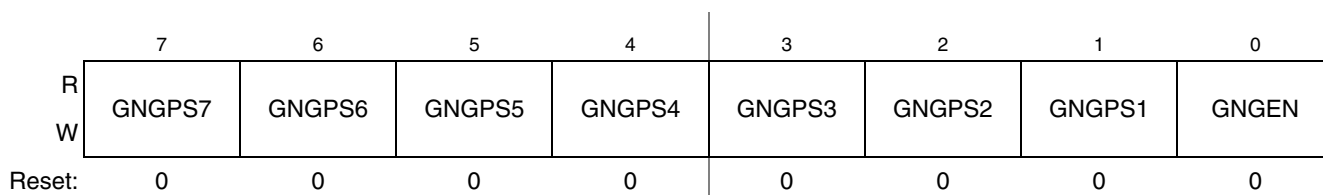


Figure 6-24. Ganged Output Drive Control Register (GNGC)

Table 6-23. GNGC Register Field Descriptions

Field	Description
7:1 GNGP[7:1]	Ganged Output Pin Select Bits — These write-once control bits selects whether the associated pin (see Table 6-1 for pins available) is enabled for ganged output. When NGEN = 1, all enabled ganged output pins will be controlled by the data, drive strength and slew rate settings for PTCO. 0 Associated pin is not part of the ganged output drive. 1 Associated pin is part of the ganged output drive. Requires NGEN = 1.
0 NGEN	Ganged Output Drive Enable Bit — This write-once control bit selects whether the ganged output drive feature is enabled. 0 Ganged output drive disabled. 1 Ganged output drive enabled. PTC0 forced to output regardless of the value of PTCDD0 in PTCDD.

```

* Programa EC01_21-1
* Recibe un byte por el puerto serie y lo transmite de regreso
* Ejecutable en RAM del MCU MC9S08SH32
* marzo 10 de 2021-1

```

```

**** Equ's del sci1 ****

```

```

scibdh equ $38
scibdl equ $39
scic2  equ $3b
scis1  equ $3c
scid   equ $3f

```

```

*****

```

```

    org $0100

```

```

; Inicializa puerto serie
    ldhx #$0082 ;BR=130 para
    sthx scibdh ;9600 bps, @Fbus=20 MHz

```

```

    mov #$0c,scic2 ;habilita tx y rx

```

```

;.....
ciclo:    bsr rxsci
          bsr txsci
          bra ciclo

```

```

*****

```

```

** Subrutina txsci
** Antes de invocar:
** a <-- byte a transmitir
** Al retornar:
** Se ha iniciado la transmisión del byte implicado
txsci:    brclr 6,scis1,txsci
          sta scid
          rts

```

```

** Subrutina rxsci
** Recibe un byte por el puerto serie (SCI)
** Al retornar:
** a <-- byte recibido
rxsci:    brclr 5,scis1,rxsci
          lda scid
          rts

```

INTERRUPCIONES EN LOS MICROCONTROLADORES

Clase virtual para la asignatura
MICROCOMPUTADORAS
Profesor: Antonio Salvá Calleja
Abril de 2020

INTERRUPCIONES EN LOS MICROCONTROLADORES

En lo fundamental, las interrupciones en los microcontroladores pueden ser de dos tipos :

Interrupciones de hardware

Interrupciones de software

Las interrupciones de hardware. Consisten en el cambio en la secuencia de ejecución de un programa, esto cuando se da un determinado evento de hardware, para pasar a ejecutar una rutina asociada con el evento de hardware implicado, a ésta frecuentemente se le denomina “***rutina de servicio***”. ***Una vez que concluye la ejecución de la rutina de servicio, el microcontrolador deberá retornar a la secuencia de ejecución que fue interrumpida.***

Para un determinado MCU, eventos de hardware podrán ser entre otros: **que el puerto serie asíncrono haya recibido un byte y éste esté listo para ser leído, que se de un flanco de bajada en un determinado bit de puerto, etc.**

En esta exposición, a los diversos eventos de hardware que pueden hacer que se de una interrupción, se les denominará como **INSTANCIAS DE INTERRUPCIÓN** y **cuando éstas llegan a presentarse, se dice que se tiene un REQUERIMIENTO DE INTERRUPCIÓN (RI)**.

INTERRUPCIONES EN LOS MICROCONTROLADORES

En lo fundamental, las interrupciones en los microcontroladores pueden ser de dos tipos :

Interrupciones de hardware

Interrupciones de software

Las interrupciones de software. Se invocan no porque se de un determinado evento de hardware, sino porque se coloque en el código del programa una instrucción que invoque a la rutina de servicio asociada, siguiendo el MCU los mismos pasos que se llevan a cabo cuando suceden interrupciones de hardware. Por ejemplo, para el MCU MC9S08SH32, solo existe una interrupción de software, y ésta se da cuando se ejecuta la instrucción SWI. En esta exposición se tratará solo con interrupciones de hardware.

INTERRUPCIONES EN EL MCU MC9S08SH32

CONCEPTOS GENERALES

Todos los microcontroladores de la familia HCS08 siguen el mismo patrón de respuesta a los diversos requerimientos de interrupción que se les pudieran presentar, lo que varía de uno a otro miembro de la familia es, en esencia, el número de instancias de interrupción soportables.

El MCU MC9S08SH32 soporta 19 instancias de interrupción de hardware y una sola interrupción de software.

En la tabla 5-2, presente en la página 64 del documento MC9S08SH32.pdf, se muestra información relevante acerca de cada una de las instancias de interrupción soportadas por el MCU MC9S08SH32. Ahí se aprecia que cada una de ellas tiene un número. Por ejemplo, la instancia de interrupción 23 se denomina como **Vadc y el evento de hardware asociado es que el convertidor analógico digital haya completado una conversión, y el dato binario generado esté listo para ser leído.**

INTERRUPCIONES EN EL MCU MC9S08SH32

PRIORIDADES

Si se llegaran a presentar simultáneamente dos o más requerimientos de interrupción, el MCU atiende primero el requerimiento que tenga asociado el número más pequeño, para después proseguir a atender los subsecuentes requerimientos en orden ascendente de sus números asociados. Por ejemplo, si se dan simultáneamente las instancias de interrupción digamos 23 y 11, el MCU ejecuta primero la rutina de servicio asociada con la instancia 11, y una vez que retorna de ésta, pasa a ejecutar la rutina de servicio asociada con la instancia 23, y después procede a seguir ejecutando el programa que fue interrumpido.

INTERRUPCIONES EN EL MCU MC9S08SH32

HABILITACIÓN GLOBAL DE INTERRUPCIONES

El que el MCU responda a los diversos requerimientos de interrupción está gobernado por el bit “I” presente en el registro de banderas (CCR) y denominado “mascara global de interrupciones”. Si éste es 1, el MCU no responde a los requerimientos de interrupción que se pudieran presentar; por otra parte si el bit “I” es cero, el MCU puede responder a los requerimientos de interrupción que se pudieran llegar a presentar.

INTERRUPCIONES EN EL MCU MC9S08SH32

HABILITACIÓN GLOBAL DE INTERRUPCIONES

El que el MCU responda a los diversos requerimientos de interrupción está gobernado por el bit “I” presente en el registro de banderas (CCR) y denominado **“mascara global de interrupciones”**. Si éste es 1, el MCU no responde a los requerimientos de interrupción que se pudieran presentar; por otra parte si el bit “I” es cero, el MCU puede responder a los requerimientos de interrupción que se pudieran llegar a presentar.

Al reset, el bit “I” es inicializado en uno lógico; por lo tanto, por defecto el MCU no responde a requerimientos de interrupción.

INTERRUPCIONES EN EL MCU MC9S08SH32

HABILITACIÓN GLOBAL DE INTERRUPCIONES

El que el MCU responda a los diversos requerimientos de interrupción está gobernado por el bit “I” presente en el registro de banderas (CCR) y denominado “mascara global de interrupciones”. Si éste es 1, el MCU no responde a los requerimientos de interrupción que se pudieran presentar; por otra parte si el bit “I” es cero, el MCU puede responder a los requerimientos de interrupción que se pudieran llegar a presentar.

Al reset, el bit “I” es inicializado en uno lógico; por lo tanto, por defecto el MCU no responde a requerimientos de interrupción.

A grandes rasgos, lo que sucede es que si el bit “I” es cero lógico, cada vez que el MCU termina la ejecución de una instrucción, checa si hay algún requerimiento de interrupción, si es el caso, procede a llevar a cabo una serie de pasos que conducen a que se pase a ejecutar la rutina de servicio asociada. Si no hay requerimiento de interrupción, el MCU procede a la ejecución de la instrucción subsecuente en el programa. Por otra parte, si el bit “I” es uno lógico, cada vez que se termina la ejecución de una instrucción, el MCU no checa si hay un requerimiento de interrupción y procede a la ejecución de la instrucción siguiente presente en el programa.

INTERRUPCIONES EN EL MCU MC9S08SH32

HABILITACIÓN GLOBAL DE INTERRUPCIONES

El que el MCU responda a los diversos requerimientos de interrupción está gobernado por el bit “I” presente en el registro de banderas (CCR) y denominado **“mascara global de interrupciones”**. Si éste es 1, el MCU no responde a los requerimientos de interrupción que se pudieran presentar; por otra parte si el bit “I” es cero, el MCU puede responder a los requerimientos de interrupción que se pudieran llegar a presentar.

Al reset, el bit “I” es inicializado en uno lógico; por lo tanto, por defecto el MCU no responde a requerimientos de interrupción.

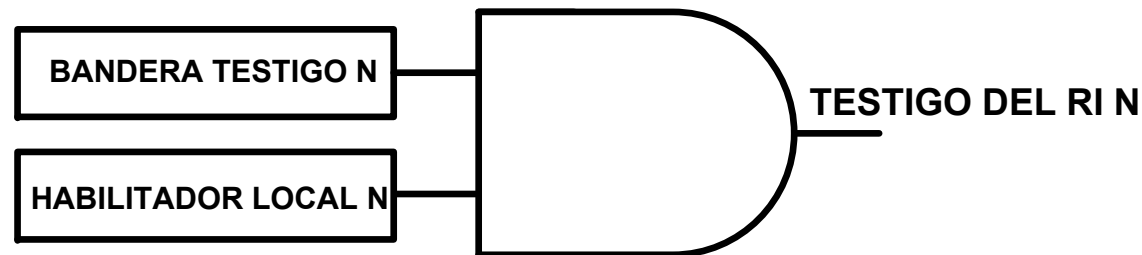
A grandes rasgos, lo que sucede es que si el bit “I” es cero lógico, cada vez que el MCU termina la ejecución de una instrucción, checha si hay algún requerimiento de interrupción, si es el caso, procede a llevar a cabo una serie de pasos que conducen a que se pase a ejecutar la rutina de servicio asociada, si no hay requerimiento de interrupción, el MCU procede a la ejecución de la instrucción subsecuente en el programa. Por otra parte, si el bit “I” es uno lógico, cada vez que se termina la ejecución de una instrucción, el MCU no checa si hay un requerimiento de interrupción y procede a la ejecución de la instrucción siguiente presente en el programa.

Si una determinada aplicación va usar interrupciones, al inicio del programa asociado con ésta, se debe hacer que el nivel del bit “I” sea cero, esto puede lograrse colocando, al final del bloque inicial del programa después del código que inicializa los recursos del MCU que se van a emplear, la instrucción CLI.

INTERRUPCIONES EN EL MCU MC9S08SH32

TESTIFICACIÓN DE LOS REQUERIMIENTOS DE INTERRUPCIÓN (explicación genérica)

Como se ha visto en láminas anteriores, cada requerimiento de interrupción (RI), tiene asociado un número, que aquí denotamos como “N”. El RI N es testificado con el que un bit asociado con éste, denominado aquí como, “**TESTIGO DEL RI N**”, sea uno lógico. Si es el caso, el MCU procederá a la ejecución de los pasos que lo llevarán a la ejecución de la rutina de servicio asociada con el RI cuyo número asociado es N. Dicho bit es la salida de una compuerta AND de dos entradas. Una entrada a la compuerta es un bit, denominado aquí genéricamente como “**BANDERA TESTIGO N**”, que se pone en uno si se da el evento de hardware asociado con el RI implicado. La otra entrada es un bit que aquí denominamos como “**HABILITADOR LOCAL N**”. Para cada instancia de interrupción, existe dentro del MCU una compuerta AND asociada con las diversas instancias de interrupción que soporta el MCU. Cabe señalar que al reset, todos los habilitadores locales de las diversas instancias de interrupción *amanecen* en cero lógico. **Nótese que si el habilitador local es cero lógico, aún cuando se haya dado el evento de hardware, el requerimiento de interrupción no se dará; por lo tanto, si una determinada aplicación va a usar interrupciones, los habilitadores locales de cada una de las instancias de interrupción implicadas, deberán inicializarse a uno lógico.**



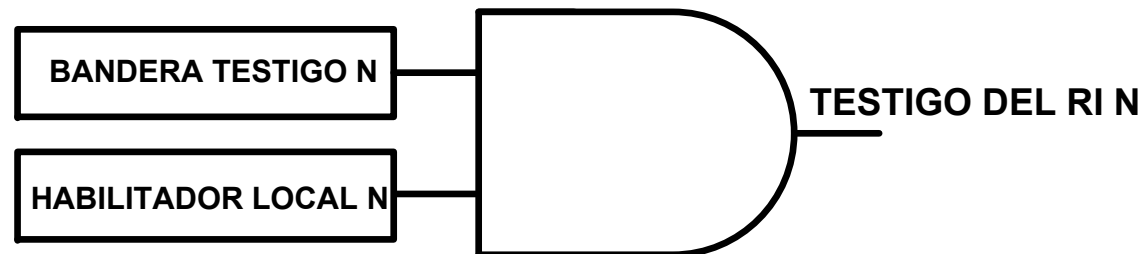
INTERRUPCIONES EN EL MCU MC9S08SH32

TESTIFICACIÓN DE LOS REQUERIMIENTOS DE INTERRUPCIÓN

(explicación genérica)

Es importante destacar que la bandera testigo está validada en cada caso por un LATCH, esto implica que una vez que se pone en uno, debido a la ocurrencia en cada caso del evento de hardware asociado, **deberá ser retornada a cero lógico; esto debe hacerse en la rutina de servicio asociada**. Si esto no se hace, al retornar de la interrupción la salida de la compuerta lógica que testifica el RI, seguirá en uno lógico, lo cual hará que el MCU entre a un ciclo infinito de entradas y salidas de la rutina de servicio, **sin atender absolutamente nada más**.

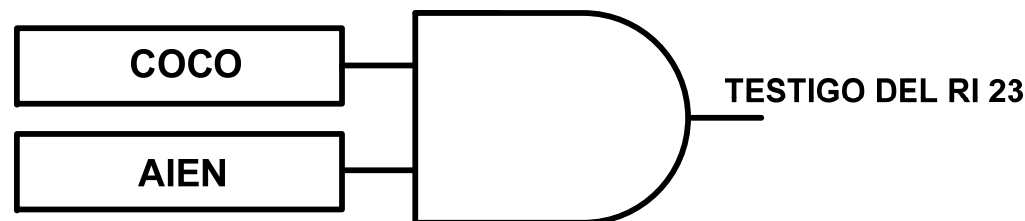
Los pasos a seguir para retornar a cero las banderas testigo, se detallan en las explicaciones propias de éstas, presentes en el manual del MCU.



INTERRUPCIONES EN EL MCU MC9S08SH32

TESTIFICACIÓN DE LOS REQUERIMIENTOS DE INTERRUPCIÓN (caso explícito)

La denotación de los bits explícitos para la bandera testigo y el habilitador local que corresponden a cada una de las instancias de interrupción, soportadas por el MCU MC9S08SH32, puede verse en las columnas “source” y “enable” de la tabla 5-2 de la página 64 del documento mc9s08sh.pdf. En cada caso, tales bits forman parte de registros asociados con el control y operación del periférico del MCU asociado con el evento de hardware que generaría el requerimiento de interrupción. Por ejemplo, a partir de la tabla puede verse que para la instancia 23 de interrupción, que genera un RI cuando el convertidor analógico digital ha completado una conversión y el dato binario asociado está listo para ser leído, la bandera testigo 23 se denomina “COCO” y el habilitador local 23 se denota como “AIEN”. El registro donde están estos dos bits se denomina ADCSC1 y la descripción de la funcionalidad de sus bits está en la página 128 del documento mc9s08sh.pdf.



INTERRUPCIONES EN EL MCU MC9S08SH32

VECTORES DE INTERRUPCIÓN

Al par de bytes que denotan la dirección de memoria donde inician cada una de las rutinas de servicio, asociadas con cada una de las instancias de interrupción, que use una determinada aplicación, se le denomina “**vector de interrupción**” propio de la instancia implicada.

Para cada una de las instancias de interrupción existe un par de direcciones de memoria donde se deben colocar este par de bytes, ya que cuando el MCU responde a un RI, en algún momento pasa a leer de esas localidades la dirección de memoria a donde debe saltar para iniciar la ejecución de la rutina de servicio asociada con el RI. Las direcciones de memoria donde el programador debe colocar cada uno de los vectores de interrupción propios de cada una de las instancias que use su aplicación, puede verse en la columna ADDRESS (High/Low) de la tabla 5-2 aquí multicitada.

Por ejemplo, para la instancia de interrupción 23 las direcciones de colocación del vector de interrupción son \$FFD0 y \$FFD1.

INTERRUPCIONES EN EL MCU MC9S08SH32

DIRECCIONES DE COLOCACIÓN DE VECTORES DE INTERRUPCIÓN PARA UN DISPOSITIVO CHIPBAS8SH

Las direcciones de colocación de los vectores de interrupción mostradas en la tabla 5-2 del manual, son validas siempre que el MCU no contenga firmware de base (FB). Si en el MCU existe firmware de base, como es el caso es el caso del dispositivo CHIPBAS8SH presente en la tarjeta FACIL_08SH, las direcciones de colocación de los vectores de interrupción estarán fuera del intervalo de 10k que ocupa el FB (\$D800 a \$FFFF); de hecho, el FB configura el que las direcciones de colocación de los vectores estén 10k por debajo de las indicadas en la tabla 5-2. Por ejemplo, para un dispositivo CHIPBAS8SH, las direcciones de colocación del vector asociado con la instancia de interrupción 23 serán \$D7D0 y \$D7D1 y las de la instancia 11 serán \$D7E8 y \$D7E9.

Se aprecia que para obtener la direcciones de colocación de los vectores de interrupción para un dispositivo CHIPBAS8SH, simplemente el byte alto del par de direcciones que se muestran en la tabla 5-2 debe ser cambiado de “FF” a “D7”.

INTERRUPCIONES EN EL MCU MC9S08SH32

PASOS QUE SIGUE EL MCU AL RESPONDER A UN REQUERIMIENTO DE INTERRUPCIÓN

Una vez que el MCU ha reconocido un requerimiento de interrupción, los pasos que sigue para proceder a ejecutar la rutina de servicio asociada, se describen a continuación

INTERRUPCIONES EN EL MCU MC9S08SH32

PASOS QUE SIGUE EL MCU AL RESPONDER A UN REQUERIMIENTO DE INTERRUPCIÓN

PASO 1: Guarda en el stack cinco bytes en el siguiente orden:

PCL (byte bajo del PC de retorno)

PCH (byte alto del PC de retorno)

Registro X

Registro A

Registro CCR

Nótese que no se guarda en el stack el registro H, entonces si el código de la rutina de servicio modifica H, es responsabilidad del programador el poner código en la rutina de servicio, que al entrar lo guarde, y rescate antes de salir de ésta, empleando para esto las instrucciones PSHH y PULH.

INTERRUPCIONES EN EL MCU MC9S08SH32

PASOS QUE SIGUE EL MCU AL RESPONDER A UN REQUERIMIENTO DE INTERRUPCIÓN

PASO 2: El bit “I” se pone en uno lógico

$$I \leftarrow 1$$

Nótese que el efecto de esto es que una rutina de servicio de interrupción no podrá, a su vez, ser interrumpida.

INTERRUPCIONES EN EL MCU MC9S08SH32

PASOS QUE SIGUE EL MCU AL RESPONDER A UN REQUERIMIENTO DE INTERRUPCIÓN

PASO 3: Lee el valor del vector de interrupción asociado, esto desde el par de direcciones, propias de la instancia de interrupción implicada.

INTERRUPCIONES EN EL MCU MC9S08SH32

PASOS QUE SIGUE EL MCU AL RESPONDER A UN REQUERIMIENTO DE INTERRUPCIÓN

PASO 4: Salta a la dirección denotada por el vector de interrupción capturado en el paso anterior, esto es, inicia la ejecución de la rutina de servicio asociada con la instancia de interrupción para la cual se está respondiendo.

INTERRUPCIONES EN EL MCU MC9S08SH32

TIEMPO DE LATENCIA DE INTERRUPCIÓN

El tiempo que transcurre entre el instante en que se da un requerimiento de interrupción, y el instante en que el MCU inicia la ejecución de la rutina de servicio asociada, se denomina TIEMPO DE LATENCIA DE INTERRUPCIÓN, que aquí se denota como “ T_{lat} ”. Puede verse que para microcontroladores de la familia hcs08, T_{lat} estará en el intervalo definido por la siguiente desigualdad:

$$1.1T_b \leq T_{lat} \leq 2.2T_b$$

Donde T_b es el periodo del reloj de bus. Por ejemplo, si éste es de 20 MHz, T_{lat} podrá estar entre 0.55 μ s y 1.1 μ s

INTERRUPCIONES EN EL MCU MC9S08SH32

FORMATO DE UN PROGRAMA QUE USE INTERRUPCIONES

En general, un programa que maneje interrupciones, estará integrado por cuatro bloques funcionales, denominados aquí como BLOQUE1, BLOQUE2, BLOQUE3 y BLOQUE4.

ESTRUCTURA DE UN PROGRAMA QUE USE INTERRUPCIONES

BLOQUE 1

BLOQUE 2

BLOQUE 3

BLOQUE 4

A continuación se menciona que deben contener cada uno de los bloques.

INTERRUPCIONES EN EL MCU MC9S08SH32

FORMATO DE UN PROGRAMA QUE USE INTERRUPCIONES

En general, el BLOQUE 1 contendrá:

- Código que inicialice los periféricos del MCU que se usen, para fines de la funcionalidad de éstos en la aplicación
- Código que inicialice en uno lógico a todos los habilitadores locales, propios de cada una de las instancias de interrupción, que se usen en la aplicación
- Al final de este bloque deberá estar la instrucción **CLI**, lo cual hará que el MCU pueda responder a los requerimientos de interrupción implicados en la aplicación

INTERRUPCIONES EN EL MCU MC9S08SH32

FORMATO DE UN PROGRAMA QUE USE INTERRUPCIONES

En general, el BLOQUE 2 contendrá:

- Código que por lo regular es un lazo donde el MCU efectúa acciones propias de la funcionalidad básica de la aplicación.

INTERRUPCIONES EN EL MCU MC9S08SH32

FORMATO DE UN PROGRAMA QUE USE INTERRUPCIONES

En general, el BLOQUE 3 contendrá:

- Código de cada una de las rutinas de servicio asociadas con cada una de las instancias de interrupción implicadas en la aplicación.

Es importante destacar que las rutinas de servicio de interrupción, deben concluirse con la instrucción RTI, y no con la instrucción RTS.

- Código de cada una de las rutinas llamables con la instrucción JSR que se usen en la aplicación, éstas desde luego que se concluyen con la instrucción RTS.

INTERRUPCIONES EN EL MCU MC9S08SH32

FORMATO DE UN PROGRAMA QUE USE INTERRUPCIONES

En general, el BLOQUE 4 contendrá en el orden indicado:

1. Si es necesario, código que declare tablas de datos que pudieran requerirse en la aplicación.
2. Código que declare la colocación de cada uno de los vectores de interrupción, asociados con cada una de las instancias de interrupción implicadas en la aplicación.

INTERRUPCIONES EN EL MCU MC9S08SH32

FORMATO DE LAS DECLARACIONES PARA COLOCAR VECTORES DE INTERRUPCIÓN

Esto se ilustra aquí con un ejemplo explícito.

Supóngase que una aplicación usa las instancias de interrupción 11 y 20, y que el programador denota el inicio de las dos rutinas de servicio con las etiquetas “servint11:” y “servint20:”.

De acuerdo con la tabla 5-2 de la página 64 del documento MC9S08SH32.PDF, las direcciones de colocación de los vectores asociados son: \$ffe8:\$ffe9 para la instancia 11, y \$ffd6:\$ffd7 para la instancia 20.

Dado que para un dispositivo CHIPBAS8SH, se sabe que las direcciones de colocación de los vectores se obtienen cambiando en cada caso el byte alto FF por el byte D7, en las direcciones obtenidas directamente de la tabla 5-2; las direcciones de colocación de los vectores de las instancias 11 y 20 respectivamente serían \$d7e8:\$d7e9 y \$d7d6:\$d7d7.

INTERRUPCIONES EN EL MCU MC9S08SH32

FORMATO DE LAS DECLARACIONES PARA COLOCAR VECTORES DE INTERRUPCIÓN

Considerando las direcciones recabadas, la declaración de la colocación del vector asociado con la instancia 11 sería:

```
org $d7e8  
dw servint11
```

Para la instancia 20 la declaración de colocación del vector sería:

```
org $d7d6  
dw servint20
```

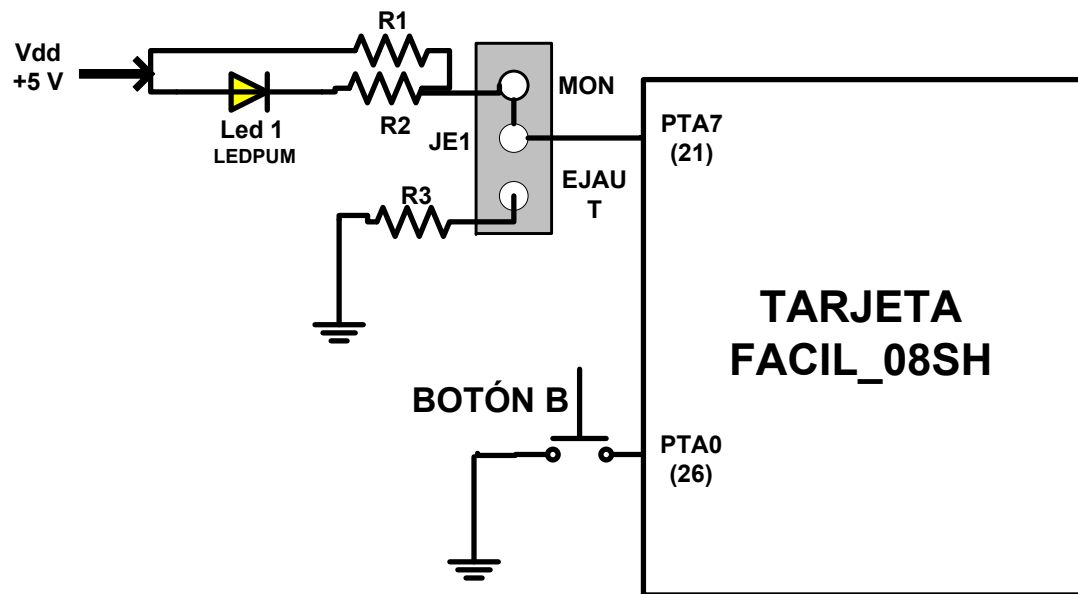
INTERRUPCIONES EN EL MCU MC9S08SH32

EJEMPLO ILUSTRATIVO

Para aclarar ideas se plantea aquí un programa ilustrativo que usa la instancia 20 de interrupción. El evento de hardware asociado es la detección de un flanco de bajada en el bit pta0. El programa está contenido en el archivo “**ejemplo1_intflpta0.asm**”.

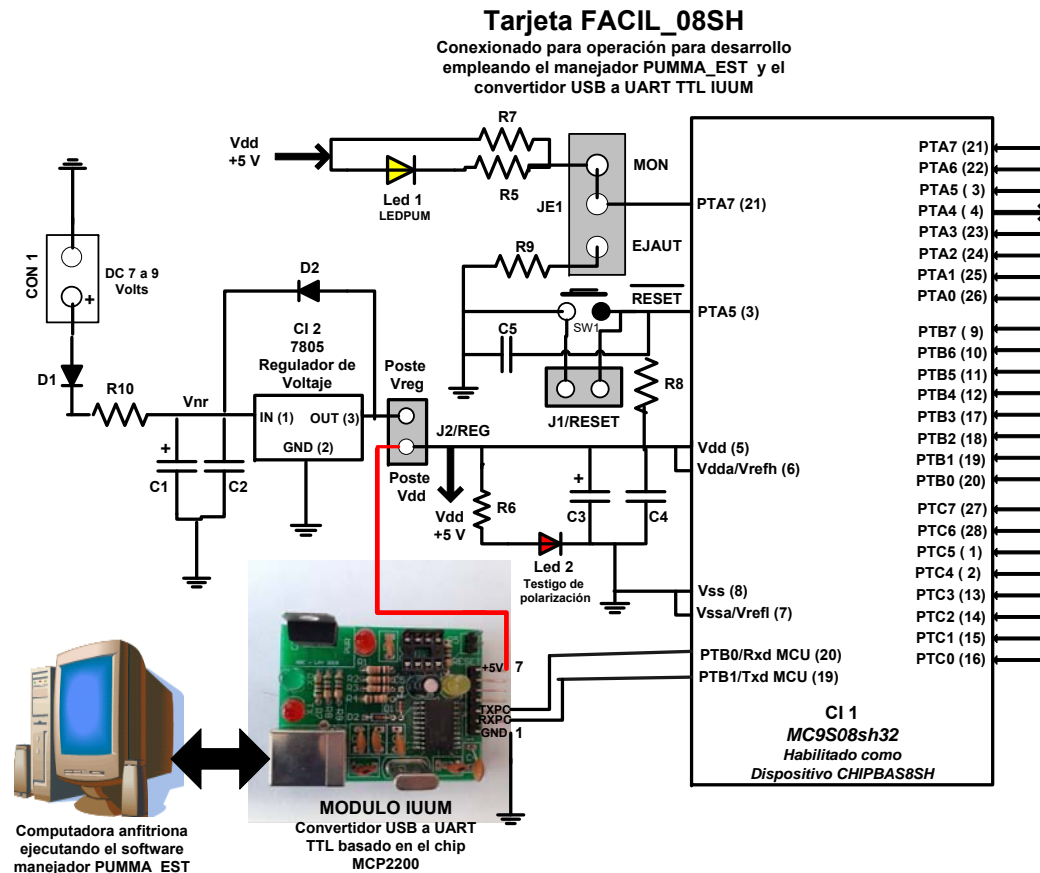
Al ejecutarse éste, deberá observarse un parpadeo del ledpum, presente en la tarjeta FACIL_08SH con una cadencia (apagado un segundo)/(encendido 25 ms), el código asociado con esta acción estaría en el bloque 2.

Por otra parte, al oprimirse y soltarse el botón B, el MCU pasaría a ejecutar la rutina de servicio, la cual haría que el ledpum permanezca encendido durante un segundo, para después retornar a la cadencia original. **Es importante que se lean los comentarios explicativos presentes en el programa.**



**PASOS A SEGUIR PARA EJECUTAR EN LA TARJETA FACIL_08SH EL
PROGRAMA ILUSTRATIVO
CONTENIDO EN EL ARCHIVO ejemplo1_intflpta0.asm**

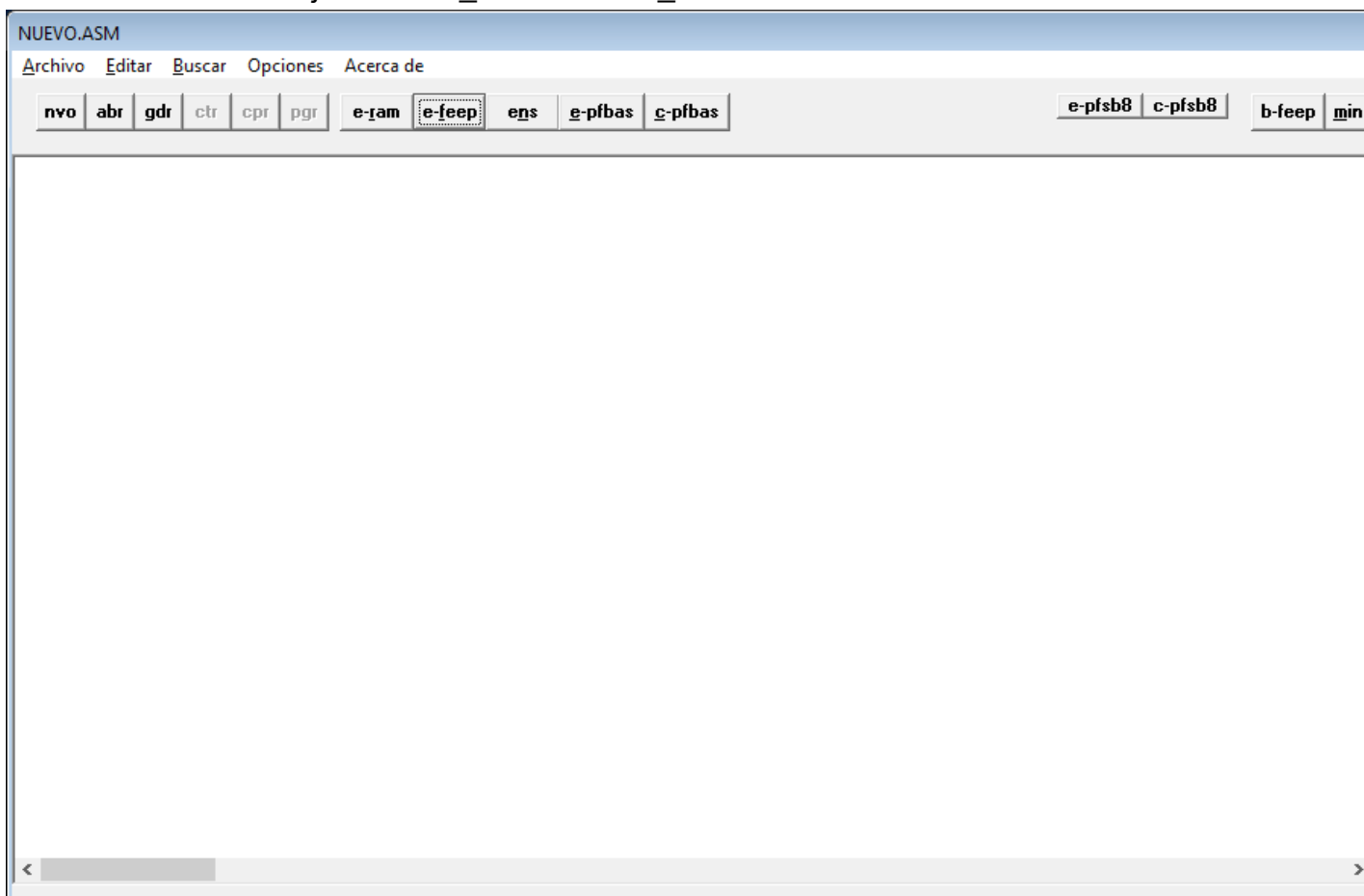
Paso 1: Conectar a la PC, empleando el modulo IUUM como se muestra en la figura, la tarjeta FACIL_08SH. El ledpum en las tarjeta deberá parpadear, indicando que ésta está en posibilidad de recibir comandos desde el software manejador PUMMA_EST



**PASOS A SEGUIR PARA EJECUTAR EN LA TARJETA FACIL_08SH EL
PROGRAMA ILUSTRATIVO
CONTENIDO EN EL ARCHIVO ejemplo1_intflpta0.asm**

Paso 2: Ejecutar en la PC el software PUMMA_EST. Una vez que se haya inicializado PUMMA_EST, y el usuario haya verificado la comunicación leyendo una página de memoria, deberá aparecer la ventana del editor de PUMMA_EST como se muestra en la figura.

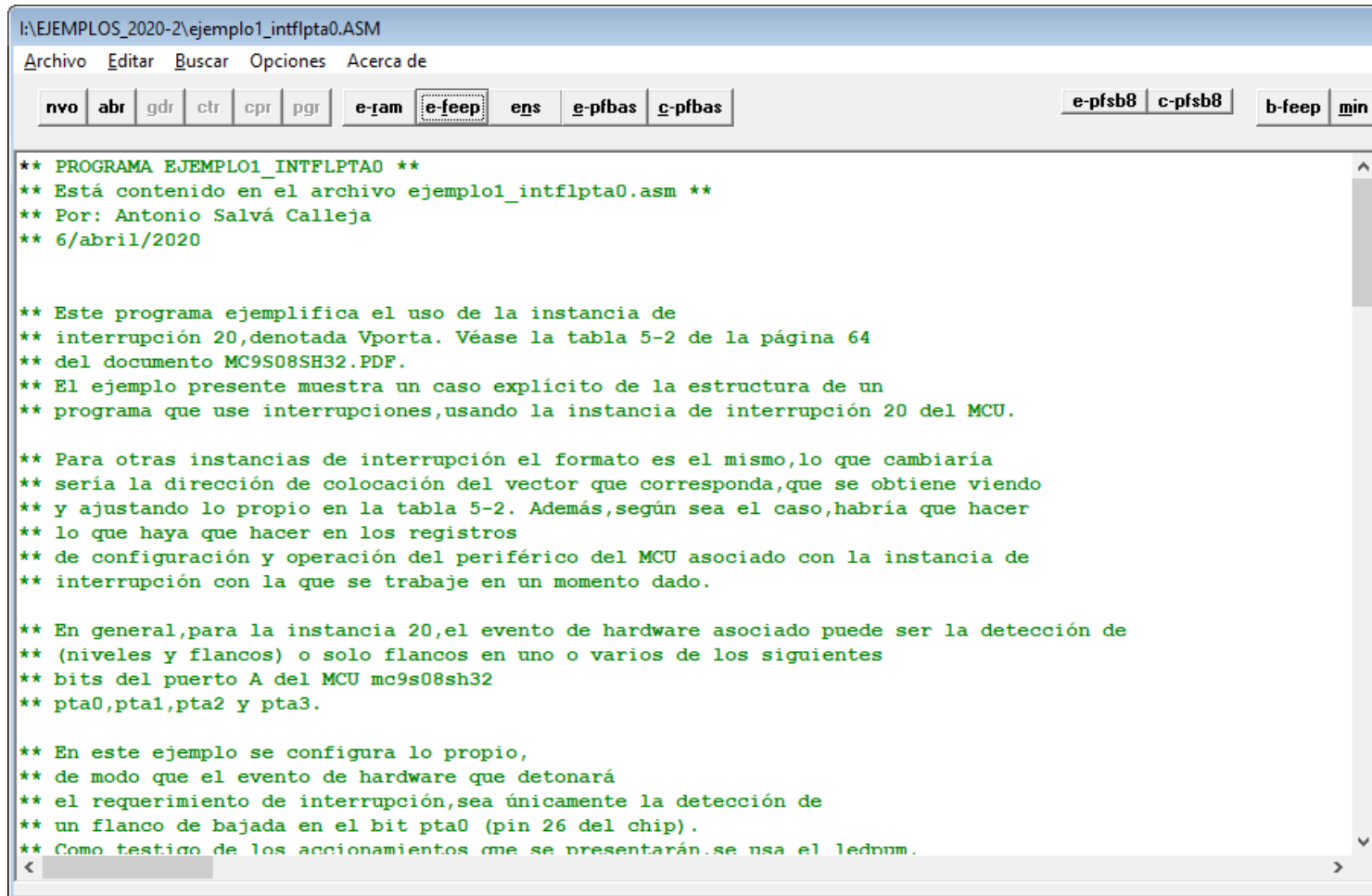
De ser necesario, pueden verse en la presentación Inisespum-facil_08sh.pdf, los pasos a seguir para iniciar una sesión de trabajo PUMMA_EST – FACIL_08SH.



PASOS A SEGUIR PARA EJECUTAR EN LA TARJETA FACIL_08SH EL PROGRAMA ILUSTRATIVO

CONTENIDO EN EL ARCHIVO ejemplo1_intflpta0.asm

Paso 3: Abrir el archivo **ejemplo1_intflpta0.asm**. Después de esto, oprimir el botón <b-feep>, lo cual borrará la memoria FLASH de usuario. El ledpum se apagará unos segundos, para regresar al parpadeo una vez que se haya concluido con el borrado de la memoria no volátil disponible para el usuario.



```
I:\EJEMPLOS_2020-2\ejemplo1_intflpta0.ASM
Archivo  Editar  Buscar  Opciones  Acerca de

nvo  abr  gdr  ctr  cpr  pgr  e-ram  e-feep  ens  e-pfbas  c-pfbas  e-pfsb8  c-pfsb8  b-feep  min

** PROGRAMA EJEMPLO1_INTFLPTA0 **
** Está contenido en el archivo ejemplo1_intflpta0.asm **
** Por: Antonio Salvá Calleja
** 6/abril/2020

** Este programa ejemplifica el uso de la instancia de
** interrupción 20,denotada Vporta. Véase la tabla 5-2 de la página 64
** del documento MC9S08SH32.PDF.
** El ejemplo presente muestra un caso explícito de la estructura de un
** programa que use interrupciones,usando la instancia de interrupción 20 del MCU.

** Para otras instancias de interrupción el formato es el mismo,lo que cambiaría
** sería la dirección de colocación del vector que corresponda,que se obtiene viendo
** y ajustando lo propio en la tabla 5-2. Además,según sea el caso,habría que hacer
** lo que haya que hacer en los registros
** de configuración y operación del periférico del MCU asociado con la instancia de
** interrupción con la que se trabaje en un momento dado.

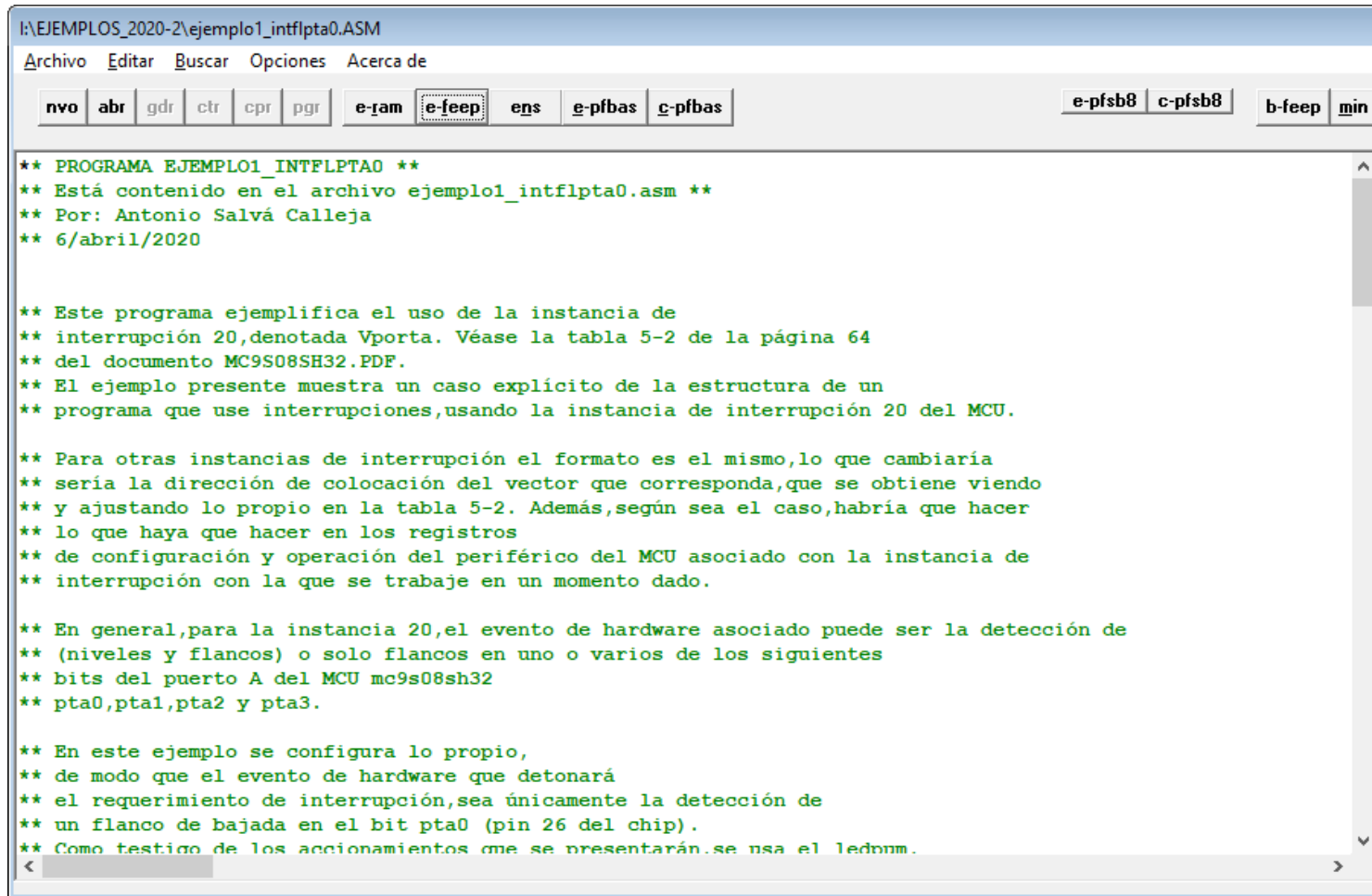
** En general,para la instancia 20,el evento de hardware asociado puede ser la detección de
** (niveles y flancos) o solo flancos en uno o varios de los siguientes
** bits del puerto A del MCU mc9s08sh32
** pta0,pta1,pta2 y pta3.

** En este ejemplo se configura lo propio,
** de modo que el evento de hardware que detonará
** el requerimiento de interrupción,sea únicamente la detección de
** un flanco de bajada en el bit pta0 (pin 26 del chip).
** Como testigo de los accionamientos que se presentarán,se usa el ledpum.
```

PASOS A SEGUIR PARA EJECUTAR EN LA TARJETA FACIL_08SH EL PROGRAMA ILUSTRATIVO

CONTENIDO EN EL ARCHIVO ejemplo1_intflpta0.asm

Paso 4: Oprimir el botón <e-feep>, lo cual hará que el programa contenido en la ventana del editor se cargue y ejecute en la memoria FLASH del usuario, a partir de la dirección \$8000.



```
I:\EJEMPLOS_2020-2\ejemplo1_intflpta0.ASM
Archivo  Editar  Buscar  Opciones  Acerca de
nvo  abr  gdr  ctr  cpr  pgr  e-ram  e-feep  ens  e-pfbas  c-pfbas  e-pfsb8  c-pfsb8  b-feep  min

** PROGRAMA EJEMPLO1_INTFLPTA0 **
** Está contenido en el archivo ejemplo1_intflpta0.asm **
** Por: Antonio Salvá Calleja
** 6/abril/2020

** Este programa ejemplifica el uso de la instancia de
** interrupción 20,denotada Vporta. Véase la tabla 5-2 de la página 64
** del documento MC9S08SH32.PDF.
** El ejemplo presente muestra un caso explícito de la estructura de un
** programa que use interrupciones,usando la instancia de interrupción 20 del MCU.

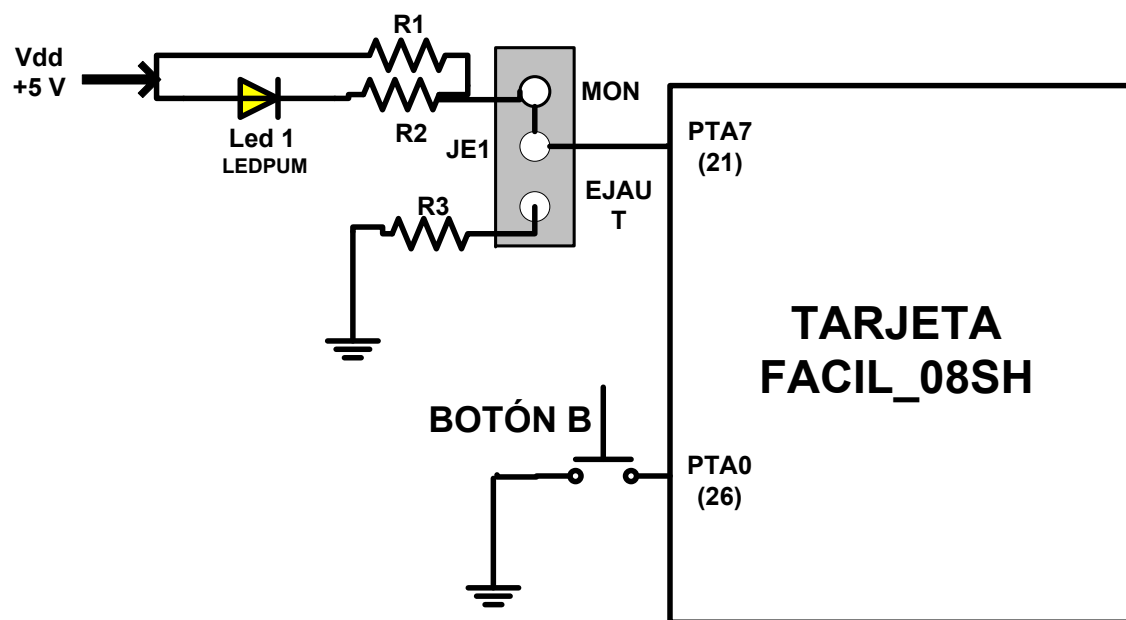
** Para otras instancias de interrupción el formato es el mismo,lo que cambiaría
** sería la dirección de colocación del vector que corresponda,que se obtiene viendo
** y ajustando lo propio en la tabla 5-2. Además,según sea el caso,habría que hacer
** lo que haya que hacer en los registros
** de configuración y operación del periférico del MCU asociado con la instancia de
** interrupción con la que se trabaje en un momento dado.

** En general,para la instancia 20,el evento de hardware asociado puede ser la detección de
** (niveles y flancos) o solo flancos en uno o varios de los siguientes
** bits del puerto A del MCU mc9s08sh32
** pta0,pta1,pta2 y pta3.

** En este ejemplo se configura lo propio,
** de modo que el evento de hardware que detonará
** el requerimiento de interrupción,sea únicamente la detección de
** un flanco de bajada en el bit pta0 (pin 26 del chip).
** Como testigo de los accionamientos que se presentarán,se usa el lednum.
```

**PASOS A SEGUIR PARA EJECUTAR EN LA TARJETA FACIL_08SH EL
PROGRAMA ILUSTRATIVO
CONTENIDO EN EL ARCHIVO ejemplo1_intflpta0.asm**

Paso 5: Una vez que el programa se esté ejecutando, se deberá observar un parpadeo del ledpum con la cadencia (apagado un segundo)/(encendido 25 ms).



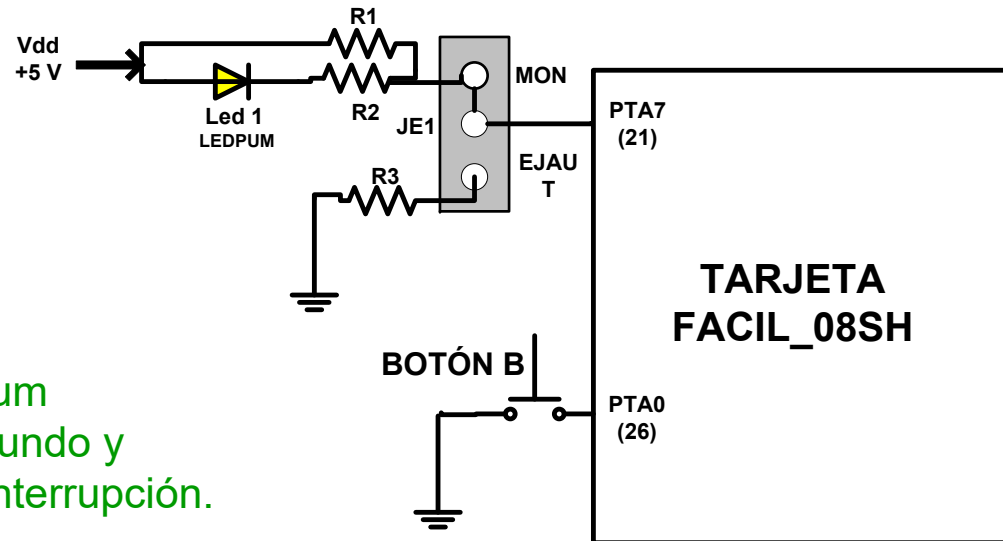
PASOS A SEGUIR PARA EJECUTAR EN LA TARJETA FACIL_08SH EL PROGRAMA ILUSTRATIVO

CONTENIDO EN EL ARCHIVO ejemplo1_intflpta0.asm

Paso 6: Oprimir y soltar el botón B, esto hará que se presente un flanco de bajada en el bit pta0, lo cual hará que el MCU pase a ejecutar la rutina de servicio, la cual hará que el ledpum permanezca prendido durante un segundo, después de esto se retorna de la interrupción, regresando el ledpum a la cadencia (apagado un segundo)/(encendido 25 ms). Como referencia didáctica, se muestra aquí la rutina de servicio, cuyo inicio fue denotado por el programador con la etiqueta “servfl:”.

```
servfl:  lda ptasc
         ora #$04
         sta ptasc ; ptaif ← 0

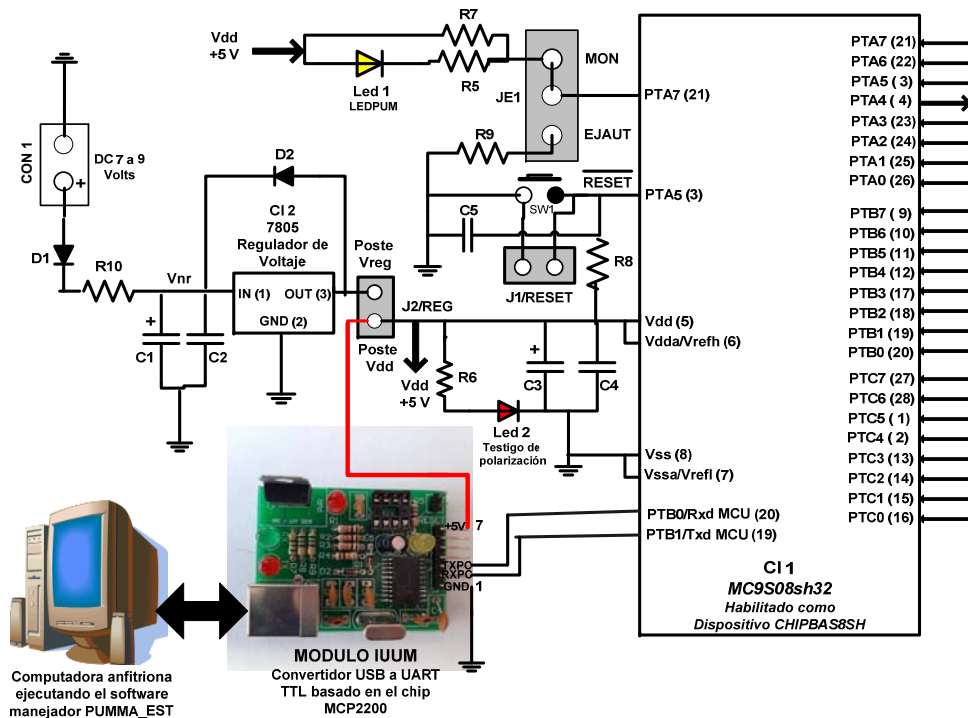
         bclr 7,ptad ;enciende ledpum
         bsr ret1seg ;espera un segundo y
         rti         ;retorna de la interrupción.
```



CONTENIDO EN EL ARCHIVO ejemplo1_intflpta0.asm

Tarjeta FACIL 08SH

Conexionado para operación para desarrollo empleando el manejador PUMMA_EST y el convertidor USB a UART TTL IUUM



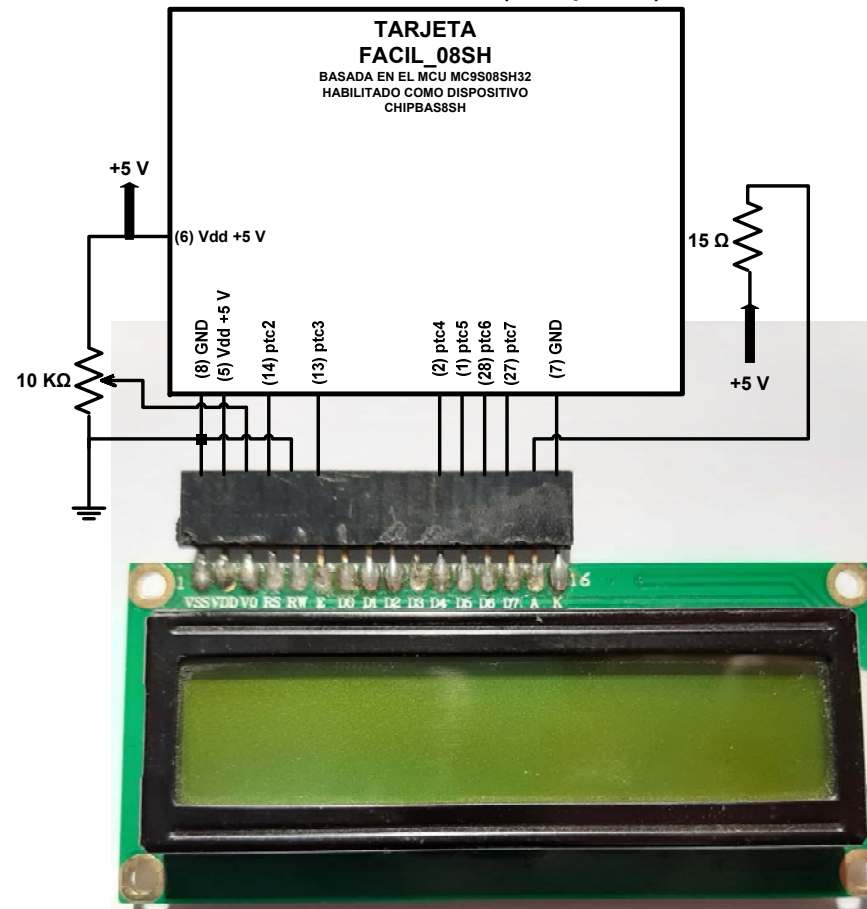
MANEJO DEL LCD CONECTADO AL PUERTO C DEL MCU PRESENTE EN LA TARJETA FACIL_08SH

Tema 2 virtual para la asignatura
MICROCOMPUTADORAS
Profesor: Antonio Salvá Calleja
Mayo de 2020

MANEJO DEL LCD CONECTADO AL MCU PRESENTE EN LA TARJETA FACIL_08SH

Para fines de manejo de un LCD de 16x2, existen subrutinas contenidas en el archivo **rutsswlcdsh32_20mhz.asm**, diseñadas para el manejo del mismo, cuando éste está conectado a la tarjeta FACIL_08SH empleando bits del puerto C, como se muestra en la figura.

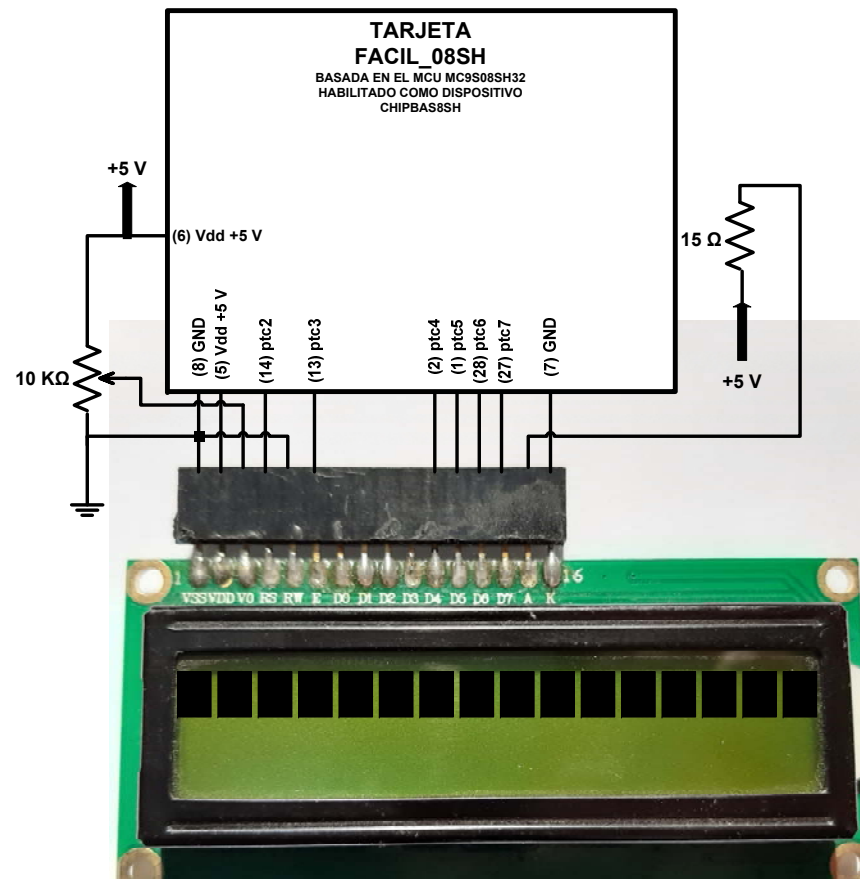
En un programa que use el LCD como se muestra aquí, el contenido del archivo aquí mencionado, debe ser incluido en la zona de colocación de subrutinas del mismo (bloque 3).



MANEJO DEL LCD CONECTADO AL MCU PRESENTE EN LA TARJETA FACIL_08SH

CALIBRACIÓN PREVIA AL USO DEL LCD

Antes de usar el LCD en un programa, el contraste del mismo debe ser calibrado. Para esto, con el LCD conectado y la tarjeta FACIL_08SH energizada, se debe ajustar el potenciómetro hasta que aparezcan en el primer renglón 16 rectángulos como aquí se muestra. De no hacerse este ajuste previo, aún cuando estén correctos el software usado y el conexionado, no se desplegará nada en el LCD.



MANEJO DEL LCD CONECTADO AL MCU PRESENTE EN LA TARJETA FACIL_08SH

ESTRUCTURA DE UN PROGRAMA QUE INTERACTÚE CON EL LCD EMLEANDO RUTINAS CONTENIDAS EN EL ARCHIVO RUTSSWLCDSH32_20MHZ.ASM

Los componentes, y orden de colocación de éstos, requeridos en un programa que use un LCD controlado con líneas del puerto C, de acuerdo con el conexionado mostrado en láminas previas de esta presentación son:

1. Al inicio del programa, como parte de la inicialización requerida por el mismo, invocación de la subrutina **'inilcd'** contenida en el archivo **rutsswlcdsh32_20mhz.asm**.
2. Código acorde con las características funcionales del programa, esto podrá incluir la escritura al LCD de bytes comando, mediante la subrutina **'escom4'**; de bytes dato, mediante la subrutina **'escdat4'**; o bien la escritura de renglones de texto mediante la subrutina **'copiadis'**. Las tres subrutinas mencionadas en este párrafo están contenidas en el archivo **rutsswlcdsh32_20mhz.asm**.
3. Inclusión en la zona de colocación de subrutinas del programa (Bloque 3), el contenido del archivo **rutsswlcdsh32_20mhz.asm**.

MANEJO DEL LCD CONECTADO AL MCU PRESENTE EN LA TARJETA FACIL_08SH

BYTES COMANDO Y BYTES DATO

En lo básico, la interacción con el LCD se lleva acabo mediante la escritura al mismo de bytes que pueden ser comandos, o bien, datos.

Los **bytes comando** indican al LCD que lleve acabo una determinada acción, como podrían ser entre otras: borrar su contenido, hacer que aparezca un cursor, posicionar la colocación del siguiente caracter a desplegar en un renglón y columna determinados.

Por otra parte los **bytes dato**, al ser escritos al LCD, harán que se despliegue un carácter, cuyo código ASCII es el valor del byte dato escrito, esto en el renglón y columna que corresponda.

MANEJO DEL LCD CONECTADO AL MCU PRESENTE EN LA TARJETA FACIL_08SH

**Funcionalidad de las subrutinas para manejo del LCD contenidas en el archivo
rutsswlcdsh32_20mhz.asm**

Subrutina INILCD

Al invocarse esta subrutina el LCD es inicializado, de modo que:

- Queda configurada la interfaz de datos de 4 bits
- La escritura de caracteres subsecuentes es de izquierda a derecha
- No aparecerá cursor
- Es posible usar los caracteres: á, é, í, ó, ú, Ñ, ñ y ü; propios de la lengua española

MANEJO DEL LCD CONECTADO AL MCU PRESENTE EN LA TARJETA FACIL_08SH

**Funcionalidad de las subrutinas para manejo del LCD contenidas en el archivo
rutsswlcdsh32_20mhz.asm**

Subrutina ESCOM4

Esta subrutina se usa para escribir un byte comando al LCD

Antes de invocar:

El acumulador A del MCU se debe cargar con el valor del byte comando por escribir al LCD.

Por ejemplo, se sabe que 0x01 es valor del byte comando para borrar el contenido de los dos renglones del LCD, entonces la ejecución de las siguientes dos líneas de código, hará que la pantalla del LCD quede vacía.

Ida #\$01

jsr escom4 ; Se escribe al LCD el byte comando 0x01 que borra la pantalla

MANEJO DEL LCD CONECTADO AL MCU PRESENTE EN LA TARJETA FACIL_08SH

**Funcionalidad de las subrutinas para manejo del LCD contenidas en el archivo
rutsswlcdsh32_20mhz.asm**

Subrutina ESCDAT4

Esta subrutina se usa para escribir un byte dato al LCD

Antes de invocar:

El acumulador A del MCU se debe cargar con el valor del byte dato por escribir al LCD.

Por ejemplo, se sabe que 0x42 es el valor del código ASCII del caracter 'B'; entonces al ejecutarse las dos líneas de código mostradas, se desplegará en el LCD el caracter 'B', esto en la siguiente posición disponible.

Ida #\$42

jsr escdat4 ; Se despliega en el LCD el carácter 'B'

MANEJO DEL LCD CONECTADO AL MCU PRESENTE EN LA TARJETA FACIL_08SH

**Funcionalidad de las subrutinas para manejo del LCD contenidas en el archivo
rutsswlcdsh32_20mhz.asm**

Subrutina COPIADIS

Esta subrutina se usa para escribir un renglón de texto al LCD

Antes de invocar:

El acumulador A del MCU se debe cargar con 0x80, si se desea que el texto se despliegue en el renglón uno, en otro caso, el acumulador A se debe cargar con 0xC0.

El par H:X se debe cargar con la dirección de memoria donde inicia la lista de bytes que representan a los caracteres que integran el texto a desplegar.

Como ejemplo, aquí se muestra un programa completo, ejecutable en RAM, que despliega en el segundo renglón del LCD el texto “ HOLA EN EL LCD ” . Después de esto se para.

org \$0100

jsr inilcd ;Inicializa el LCD

lda #\$c0 ;se indica que el texto aparecerá en el renglón 2

ldhx #mensaje ;h:x ← dir inicial de lista que representa el texto a colocar

jsr copiadis ;despliega el texto en el renglón 2

fin: bra fin ; Finaliza. Esto equivale a un end

\$include "i:\ejemplos_2019-2\rutsswlcdsh32_20mhz.asm" ; Se incluye archivo con rutinas para el LCD

mensaje: fcc “ HOLA EN EL LCD “

MANEJO DEL LCD CONECTADO AL MCU PRESENTE EN LA TARJETA FACIL_08SH

COMANDOS BASICOS DE UN LCD DE 16x2

Los bytes comando básicos de un LCD se muestran en la tabla aquí presentada.

Byte comando (hex)	Acción que se realiza en el LCD
0x01	Se borra el LCD
0x02	Regresa posición de siguiente carácter a desplegar al renglón 1 columna 1
0x04	La escritura de caracteres subsecuentes es de derecha a izquierda
0x06	La escritura de caracteres subsecuentes es de izquierda a derecha
0x0C	LCD encendido, cursor no visible
0x0E	LCD encendido, cursor visible
0x0F	LCD encendido. cursor visible y parpadeante

MANEJO DEL LCD CONECTADO AL MCU PRESENTE EN LA TARJETA FACIL_08SH

COMANDOS DE POSICIONAMIENTO DE UN LCD DE 16x2

El valor de los bytes comando, para hacer que la colocación del siguiente carácter que se ha de desplegar en el LCD, sea en un renglón y columna determinados, puede obtenerse a partir de la siguiente ecuación

$$cmdpos = 128 + (nren - 1)64 + ncol - 1 \dots \dots (1)$$

Donde nren es el número del renglón (1 ó 2), y ncol (1 a 16), es el número de la columna, a donde se va a colocar el siguiente carácter que se envíe al LCD como byte dato, cuyo valor es el código ascii del carácter que se va a desplegar.

Por ejemplo, definir líneas de código en ensamblador, que al ejecutarse, hagan que el string "APOLO" se escriba en el LCD a partir del renglón 2 y la columna 3.

A partir de la ecuación (1), el valor del byte comando requerido para el posicionamiento deseado es cmdpos = 194 = 0xC2. Por lo tanto el código en ensamblador podría ser:

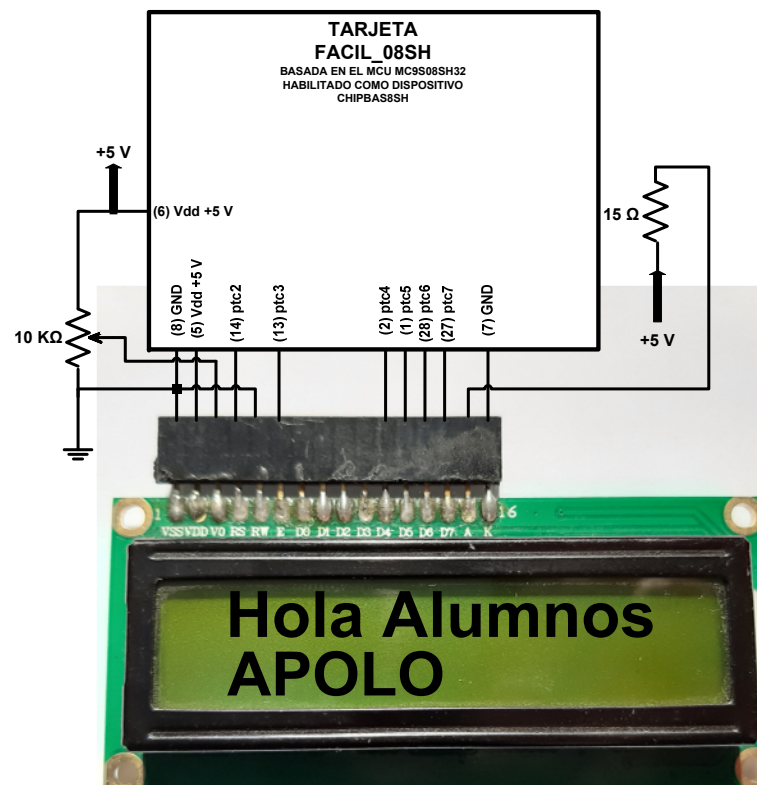
```
lda #c2
jsr escom4 ;escribe al LCD comando para posicionar despliegue a partir de renglón 2 y columna 3
lda #$41
jsr escdat4 ;escribe al LCD el carácter cuyo código ascii es $41 (A)
lda #$50
jsr escdat4 ;escribe al LCD el carácter cuyo código ascii es $50 (P)
lda #$4f
jsr escdat4 ;escribe al LCD el carácter cuyo código ascii es $4f (O)
lda #$4c
jsr escdat4 ;escribe al LCD el carácter cuyo código ascii es $4c (L)
lda #$4f
jsr escdat4 ;escribe al LCD el carácter cuyo código ascii es $4f (O)
```

MANEJO DEL LCD CONECTADO AL MCU PRESENTE EN LA TARJETA FACIL_08SH

EJEMPLOS ILUSTRATIVOS DE USO DEL LCD EMPLEANDO LAS RUTINAS CONTENIDAS EN EL ARCHIVO RUTSSWLCDSH32_20MHZ.ASM

Para complementar lo expuesto en esta presentación pueden verse los programas **EJ1_LCD_FACIL_08SH** y **MENS16X2SH32_SCROLL_2020-2** contenidos respectivamente en los archivos **ej1_lcd_facil_08sh.asm** y **mens16x2sh32_scroll_2020-2.asm**

Aquí se muestra lo que despliega el LCD al ejecutarse el programa contenido en el archivo **ej1_lcd_facil_08sh.asm**. Al ejecutarse el segundo programa se muestra en el LCD un mensaje largo contenido en renglones de 16 columnas y con accionamiento de scroll al desplegarse. Véanse los comentarios en el propio programa.



INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

Tema 3 virtual para la asignatura
MICROCOMPUTADORAS
Profesor: Antonio Salvá Calleja
Mayo de 2020

INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

Todos los microcontroladores cuentan con la capacidad de poder ser configurados para que se generen interrupciones que se dan a un cierto intervalo de tiempo, esto es, el MCU entraría a la rutina de servicio periódicamente, siendo el intervalo entre entradas sucesivas a ésta programable. Esta funcionalidad es de gran utilidad en Instrumentación y Control; por ejemplo cuando se muestrea una señal. En este caso la rutina de servicio simplemente leería la salida del canal del convertidor analógico digital, a cuya entrada estuviera conectada la señal que se muestrea. El periodo entre interrupciones, obviamente sería el periodo de muestreo que se requiera en un momento dado.

INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

El hardware básico para la funcionalidad aquí esbozada, consiste en un contador binario, frecuentemente de 16 bits, para el cual:

1. Se puede configurar cual es la cuenta tope a la que puede llegar, para pasar a cero de nuevo, a esto se le llama **evento de overflow**.
2. Se conoce la frecuencia de la señal cuadrada TTL que acciona el contador.
3. Existe un bit testigo que se pone en uno lógico cuando se da el overflow. Ésto sería la bandera testigo de una instancia de interrupción, que se daría cada vez que hay un overflow del contador.

Por ejemplo, si se sabe que la cuenta tope del contador es 999, y que la frecuencia de la señal a la entrada del contador es 1 MHz, el intervalo de tiempo entre overflows sería de 1 mS. Ya que las cuentas del contador serían: 0,1,2,3,.....999,999; o sea, se darían mil periodos de 1 uS de la señal cuadrada de entrada al contador. **Nótese que la cuenta tope es el número de cuentas asociado menos 1.**

Para los microcontroladores, el contador aquí mencionado es parte de un periférico o módulo funcional denominado **Temporizador**. Un determinado MCU puede tener uno o varios temporizadores, cada uno con su respectivo contador.

INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

En la práctica, frecuentemente lo que se requiere es determinar cual es la cuenta tope (CT) requerida, para que el periodo entre overflows (Tovf), tenga un valor deseado, esto para un determinado valor de la frecuencia (Fck) de la señal que acciona el contador.

De acuerdo con lo explicado en la lámina anterior el número de cuentas requerido (NC) y la cuenta tope requerida estarían dados por:

$$NC = \frac{T_{ovf}}{T_{ck}} \dots\dots(1)$$

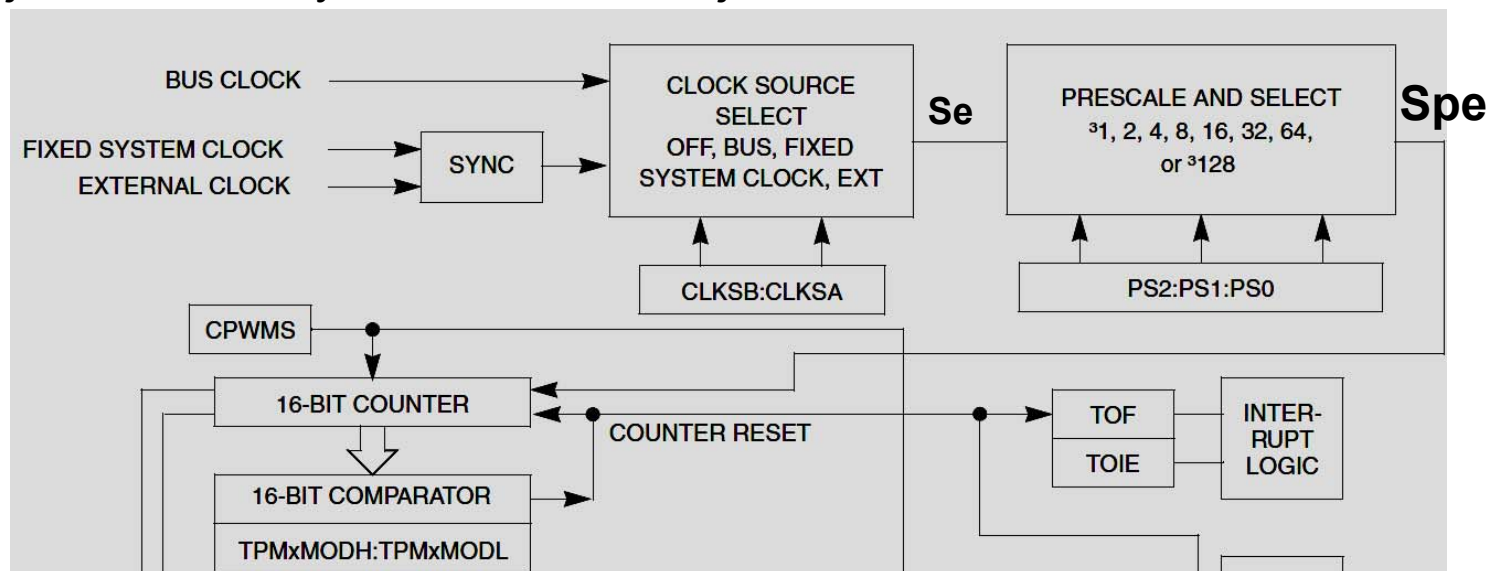
$$CT = NC - 1 \dots\dots(2)$$

Donde $T_{ck} = 1/F_{ck}$ es el periodo de la señal aplicada al contador. Si el valor de NC obtenido no es entero, se debe considerar para éste al entero más próximo, esto haría que se produzca un error en el valor de Tovf respecto al valor requerido. En muchos casos prácticos este error suele ser pequeño y su efecto no afecta la funcionalidad de la aplicación implicada.

INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

TEMPORIZADORES DEL MCU MC9S08SH32

El MCU mc9s08sh32 cuenta con dos temporizadores que el fabricante denota como TPM1 y TPM2. En la figura mostrada, tomada de la página 241 del documento mc9s08sh32.pdf, se muestra un esquema genérico de éstos, donde se aprecia únicamente lo relacionado con la generación de interrupciones periódicas. Ahí se aprecia que la señal de reloj, denotada como '**Spe**', que entra al contador, tiene una frecuencia que es el resultado de dividir entre uno de ocho posibles divisores, la frecuencia de otra señal, aquí denotada como '**Se**', la cual puede escogerse entre una de tres señales que son: el reloj de bus, una señal que el fabricante denomina reloj fijo del sistema y una señal de reloj externa.



INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

TEMPORIZADORES DEL MCU MC9S08SH32

Selección de preescalamiento

El divisor asociado con el preescalamiento es determinado con tres bits, denotados PS2, PS1 y PS0. En la tabla aquí mostrada, tomada de la página 247 del documento mc9s08sh32.pdf, se aprecia el valor que tiene el divisor de la frecuencia, que aquí denotamos como 'pe', para cada una de las ocho posibilidades asociadas con los tres bits aquí mencionados.

PS2:PS1:PS0	TPM Clock Source Divided-by (pe)
000	1
001	2
010	4
011	8
100	16
101	32
110	64
111	128

INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

TEMPORIZADORES DEL MCU MC9S08SH32

Selección de preescalamiento

Los tres bits PS2, PS1 y PS0, son parte de un registro del MCU denominado genéricamente como TPMxSC.

En la figura mostrada, tomada de la página 246 del documento mc9s08sh32.pdf, se muestra este registro con sus bits componentes. Se aprecia que los tres bits definitorios del preescalamiento son los tres bits menos significativos.

Al reset estos tres bits se inicializan como 000; por lo tanto, de acuerdo con la tabla mostrada en la lámina anterior, el valor por default para el preescalamiento es uno.

	7	6	5	4	3	2	1	0
R	TOF	TOIE	CPWMS	CLKSB	CLKSA	PS2	PS1	PS0
W	0							
Reset	0	0	0	0	0	0	0	0

Figure 16-7. TPM Status and Control Register (TPMxSC)

INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

TEMPORIZADORES DEL MCU MC9S08SH32

Selección de señal de entrada 'Se'

La selección de la señal de entrada 'Se' está gobernada por los bits CLKSB y CLKSA. En la tabla mostrada, tomada de la página 247 del documento mc9s08sh32.pdf, se aprecia que valores deben tener estos bits, de acuerdo con la señal que se desee sea 'Se', en un momento dado.

Al reset, estos dos bits se inicializan como 00; y el contador del temporizador mostrará 0x0000 como cuenta, sin modificarse.

En los ejemplos ilustrativos de esta presentación, se usará como señal 'Se' al reloj de bus; por lo tanto, los bits CLKSB y CLKSA siempre se pondrán como 01.

CLKSB:CLKSA	TPM Clock Source to Prescaler Input (Se)
00	No clock selected (TPM counter disable)
01	Bus rate clock
10	Fixed system clock
11	External source

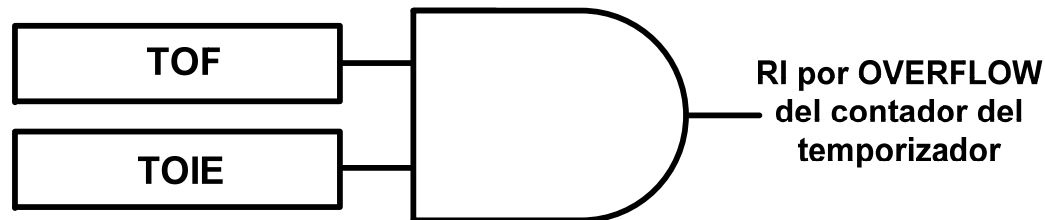
INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

TEMPORIZADORES DEL MCU MC9S08SH32

Habilitación de interrupciones cada que se da un overflow

Cada que se da un overflow se puede generar un requerimiento de interrupción, esto hará que el MCU entre periódicamente a la rutina de servicio asociada; esto es, el accionamiento asociado con el código colocado en ésta, se ejecutará repetitivamente, con una separación entre accionamientos igual al periodo de tiempo entre overflows, que en esta presentación se denota como **Tovf**.

En la figura se muestra la compuerta lógica, interna al MCU, cuya salida en uno lógico testifica el requerimiento de interrupción por overflow. Se aprecia que la bandera testigo asociada se denomina **TOF**, y que el habilitador local asociado se denomina como **TOIE**, ambos son respectivamente los bits 7 y 6 del registro TPMxSC. Véanse las láminas 10 a 13 de la presentación **Tema1_concepto_int.pdf**.

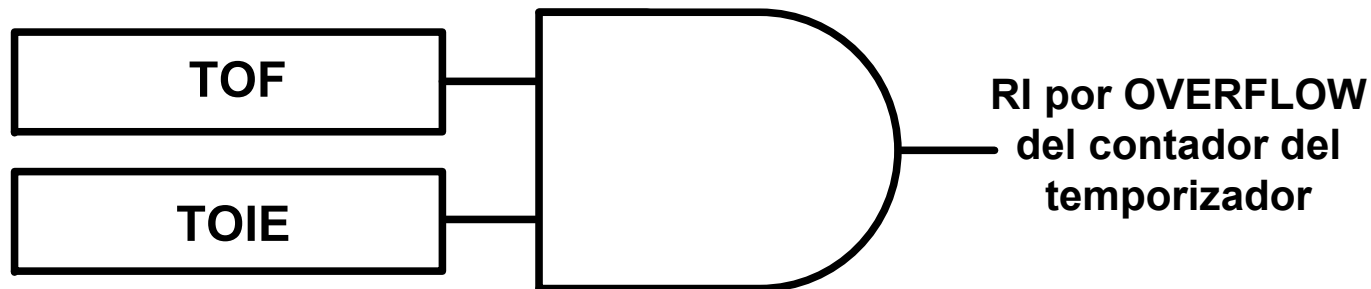


INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

TEMPORIZADORES DEL MCU MC9S08SH32

Habilitación de interrupciones cada que se da un overflow

De acuerdo con lo expresado en la lámina anterior, si un determinado programa va a usar interrupciones periódicas, en el bloque 1 de éste, deberá haber código que haga que el habilitador local TOIE tome el valor de uno lógico, además, en algún punto se debe hacer que la máscara global de interrupciones 'I' sea cero lógico, mediante la instrucción **CLI**. Para una mejor comprensión de lo expuesto en esta lámina, véanse las láminas 6 a 9 y 21 a 25 de la presentación **Tema1_concepto_int.pdf**.



INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

TEMPORIZADORES DEL MCU MC9S08SH32

Cálculo y colocación de la cuenta tope (CT) asociada con un determinado T_{ovf}

Asumiendo que la señal de entrada 'Se' al preescalador es el reloj de bus, $T_{ck} = T_b$, véase la figura presente en la lámina 7, y que se usa un determinado preescalamiento 'pe'. Considerando las ecuaciones (1) y (2) mostradas en la lámina 4, la cuenta tope 'CT' requerida para un determinado valor del tiempo entre overflows ' T_{ovf} ', se determina mediante la siguiente ecuación:

$$CT = Int\left\{\frac{T_{ovf}}{peT_b}\right\} - 1 \dots\dots (3)$$

Donde T_b es el periodo asociado con la frecuencia del reloj de bus, además, se considera el hecho de que CT debe ser un entero.

INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

TEMPORIZADORES DEL MCU MC9S08SH32

Cálculo y colocación de la cuenta tope (CT) asociada con un determinado T_{ovf}

El valor de la cuenta tope obtenida mediante la ecuación (3), debe colocarse en un par de registros del temporizador denominados genéricamente como **TPMxMODH** y **TPMxMODL**, los cuales deberán contener respectivamente los bytes alto y bajo del valor de la CT requerida para un determinado T_{ovf} .

Nótese que el valor de la cuenta tope CT deberá caber en 16 bits.

INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

TEMPORIZADORES DEL MCU MC9S08SH32

Ejemplo 1: cálculo del valor de CT para un determinado Tovf

Supóngase que se desea que el tiempo entre overflows Tovf para el contador del temporizador sea 50 mS, si el reloj de bus tiene una frecuencia $F_b = 20 \text{ MHz}$ ($T_b = 50 \text{ nS}$), determinar los posibles pares de valores para la cuenta tope CT y el preescalamiento 'pe'.

Primero calculamos con $pe = 1$, usando la ecuación (3) considerando los valores explícitos implicados:

$$CT = Int\left\{\frac{50 \times 10^{-3}}{(50 \times 10^{-9}) \times (1)}\right\} - 1 = 999999$$

Dado que el valor obtenido no cabe en 16 bits, es claro que no se puede usar un preescalamiento unitario para los valores de Tovf y Tb implicados.

Por lo tanto, se debe aumentar el valor de 'pe'. Puede verse que 16, es el primer valor hacia arriba de 'pe', que conduce a un valor de CT que cabe en 16 bits, este valor se obtiene aplicando la ecuación (3) con $pe = 16$ y el detalle de su cálculo se muestra en la siguiente lámina

INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

TEMPORIZADORES DEL MCU MC9S08SH32

Ejemplo 1: cálculo del valor de CT para un determinado Tovf

Con $pe = 16$ se tiene:

$$CT = Int\left\{\frac{50 \times 10^{-3}}{(50 \times 10^{-9}) \times (16)}\right\} - 1 = 62499 = 0xF423$$

Puede verse que si usamos $pe = 32$ ó 64 ó 128 , los valores de CT obtenidos cabrían en 16 bits, y serían respectivamente: 31249, 15624 y 7811.

De hecho, cualquiera de los 4 pares de valores para CT y 'pe' obtenidos, podrían usarse para que el tiempo entre overflows sea 50 mS, cuando

Fbus = 20 MHz

INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

TEMPORIZADORES DEL MCU MC9S08SH32

Direcciones de los registros asociados con la generación de interrupciones periódicas con los temporizadores del MCU mc9s08sh32

Los tres registros asociados con la generación de interrupciones periódicas con un temporizador del MCU mc9s08sh32, genéricamente el fabricante los denota como: TPMxSC, TPMxMODH y TPMxMODL. La literal 'x' denota el número de temporizador de que se trate en un momento dado. Dado que el MCU mc9s08sh32 tiene dos temporizadores, la literal 'x' será 1 para el temporizador uno, y 2 para el temporizador dos.

INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

TEMPORIZADORES DEL MCU MC9S08SH32

Direcciones de los registros asociados con la generación de interrupciones periódicas con los temporizadores del MCU mc9s08sh32

Así, del mapa de memoria del MCU aquí mencionado, las direcciones y denotación de los dos tríos de bytes asociados con la funcionalidad de overflow, para cada uno de los dos temporizadores, presentes en el MCU son:

Temporizador	Registro	Dir (hex)
1	TPM1SC	0x20
1	TPM1MODH	0x23
1	TPM1MODL	0x24
2	TPM2SC	0x60
2	TPM2MODH	0x63
2	TPM2MODL	0x64

INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

TEMPORIZADORES DEL MCU MC9S08SH32

Inicialización de los registros del temporizador 1 para que éste genere tiempos entre overflows T_{ovf} determinados

Pasos a seguir:

1. Emplendo la ecuación (3), determinar un par de valores viables para el preescalamiento 'pe' y la cuenta tope CT, considerando el valor del tiempo entre overflows T_{ovf} deseado y el periodo de bus T_b del reloj del MCU.
2. A partir del valor del preescalamiento 'pe' determinado en el paso anterior, usando la tabla 16-5 de la página 247 del manual del MCU, determinar el valor npe que representan los 3 dígitos binarios PS2, PS1 y PS0. Por ejemplo, si $pe = 16$, $npe = 4$, si $pe = 8$, $npe = 3$. Nótese que npe es el exponente al que hay que elevar el número dos para obtener el valor de pe.
3. Obtener el valor 'valsc' del byte que se debe cargar en el registro TPM1SC empleando la ecuación:

•

$$valsc = 64h + 8 + npe \cdots \cdots (4)$$

Donde h debe ser uno, si se desea que cada que se de un overflow, se genere un requerimiento de interrupción; en otro caso h debe ser cero.

Nota: Puede verse, aplicando un poco de aritmética binaria elemental, que si $h = 1$, el valor del bit 6 del byte que se va a cargar en el registro TPM1SC es uno lógico, por lo tanto, TOIE se pondrá en uno lógico, habilitándose localmente las interrupciones por overflow; por otra parte, si $h = 0$, TOIE quedará en cero lógico, no estando habilitadas interrupciones por overflow.

INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

TEMPORIZADORES DEL MCU MC9S08SH32

Inicialización de los registros del temporizador 1 para que éste genere tiempos entre overflows T_{ovf} determinados

Ejemplo 2: Se desea que el temporizador 1 genere interrupciones periódicas cada 100 mS, suponiendo que $F_{bus} = 20$ MHz ($T_b = 50$ nS), determinar:

- Un par de valores viable para la cuenta tope CT y el preescalamiento 'pe'.
- El valor valsc que debe cargarse en el registro TPM1SC.
- Esbozar un código en ensamblador que configure el temporizador 1 para fines de lo especificado en el enunciado de este ejemplo.

INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

TEMPORIZADORES DEL MCU MC9S08SH32

Inicialización de los registros del temporizador 1 para que éste genere tiempos entre overflows Tovf determinados

Ejemplo 2: Solución

Paso 1: Empleando la ecuación 3 puede verse que el mínimo valor de 'pe' para que CT tenga un valor que quepa en 16 bits es 32, siendo $CT = 62499 = 0xF423$.

Paso 2: Dado que $pe = 32$, a partir de la tabla 16- 2 de la página 247 del manual del MCU, se ve que $npe = 5$.

Paso 3: Empleando la ecuación 4, considerando que se han de generar interrupciones ($h=1$), cada vez que se de un overflow, el valor valsc que ha de cargarse en el registro TPM1SC es
 $valsc = 64(1) + 8 + npe = 77 = 0x4D = 01001101$.

Nótese que el bit 6 que corresponde al habilitador local TOIE se pondrá en uno lógico, lo que habilita localmente la instancia de interrupción por overflow del temporizador 1.

Un posible tramo de código en ensamblador para los fines de este ejemplo es:

```
mov #$f4,tpm1modh
mov #$23,tpm1modl ;carga cuenta tope CT

mov #$4d,tpm1sc ;pe = 32, toie = 1, Se = reloj de bus
```

INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

TEMPORIZADORES DEL MCU MC9S08SH32

Ejemplo 3: Se desea que empleando el temporizador 1 se produzca la siguiente cadencia en el ledpum de la tarjeta FACIL_08SH, (250 ms encendido/250 mS apagado), suponiendo que $F_{bus} = 20 \text{ MHz}$ ($T_b = 50 \text{ nS}$), escribir un programa en ensamblador que realice esto empleando para ello el que se generen interrupciones periódicas cada 250 mS y en la rutina de servicio asociada, simplemente se complementa el bit PTA7, que es a donde está conectado el **ledpum**, véase la figura 4 del documento Gb_facil_08sh.pdf.

Empleando la ecuación (3) puede verse que el único par viable (CT, pe) es: $pe = 128$ y $CT = 39061 = 0x9895$.

Dado el valor de 'pe' requerido, $n_{pe} = 7$, y ya que se requiere que se de un requerimiento de interrupción cada que haya un overflow, debemos usar $h=1$ en la ecuación (4), por lo tanto, el valor de valsc que se debe cargar en el registro TPM1SC es:

$$valsc = 64(1) + 8 + 7 = 79 = 0x4F$$

INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

TEMPORIZADORES DEL MCU MC9S08SH32

Ejemplo 3: Continuación

De lo obtenido en la lámina anterior, el bloque 1 del programa podría ser:

```
bset 7,ptadd ;ptad7 es salida
```

```
mov #$98,tpm1modh
```

```
mov #$95,tpm1modl ;carga cuenta tope CT
```

```
mov #$4f,tpm1sc ;pe = 128, toie = 1, Se = reloj de bus, Tovf = 250 mS
```

```
cli ;habilita globalmente interrupciones
```

INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

TEMPORIZADORES DEL MCU MC9S08SH32

Ejemplo 3: Continuación

Dado que en este ejemplo todo lo hará la rutina de servicio de interrupción, el bloque 2 del programa será simplemente un salto a la misma línea.

BLOQUE 2

fin

bra fin

INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

TEMPORIZADORES DEL MCU MC9S08SH32

Ejemplo 3: Continuación

Estructura del BLOQUE 3

El bloque 3 contendrá únicamente la rutina de servicio de interrupción, el inicio de ésta se denota con la etiqueta 'servparp:'. El MCU pasará a ejecutarla cada 250 mS, el código en ésta debe:

- Regresar a cero la bandera testigo TOF. Cómo se hace esto, puede verse en la explicación alusiva al bit TOF, presente en la página 246 del manual del MCU.
Ahí dice que para limpiar la bandera TOF se debe hacer una lectura del registro TPM1SC, seguido esto por una puesta a cero del bit 7 del mismo registro.
- Efectuar el accionamiento que se desea se haga cada 250 mS. Éste será simplemente complementar el bit pta7 del puerto A, sin afectar a los otros bits del mismo puerto. Esto se logra efectuando una operación or exclusivo con 1 lógico para el bit 7 del puerto y con cero lógico para los demás bits, empleando para ello a la instrucción EOR del MCU.

INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

TEMPORIZADORES DEL MCU MC9S08SH32

Ejemplo 3: Continuación
Estructura del BLOQUE 3

La rutina de servicio podría ser:

```
servparp:      lda tpm1sc
                bclr 7,tpm1sc ; TOF ← 0

                lda ptad
                eor #$80
                sta ptad ;complementa el bit pta7

                rti ;Retorna de la interrupción
```

INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

TEMPORIZADORES DEL MCU MC9S08SH32

Ejemplo 3: Continuación
Estructura del BLOQUE 4

En la tabla 5-2 de la página 64 del manual del MCU, se ve que la instancia de interrupción por overflow para el temporizador 1 es la número 11. Ahí se aprecia que las direcciones naturales para la carga del vector de interrupción son FFE8 y FFE9, por lo tanto, para un dispositivo CHIPBAS8SH, las direcciones de carga del vector son D7E8 y D7E9. Entonces, el bloque 4 podría quedar como:

```
org $d7e8  
dw servparp
```

El programa fuente de este ejemplo está en el archivo **parp250ms.asm**

INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

TEMPORIZADORES DEL MCU MC9S08SH32

Ejemplo 4: Se desea emplear el temporizador 1 para efectuar cada cinco segundos un accionamiento, que consiste en complementar el bit pta7 del puerto A. Considérese que $F_{bus} = 20 \text{ Mhz}$ ($T_b = 50 \text{ nS}$). En este caso se requiere que T_{ovf} sea 5 segundos. Empleando la ecuación 3 se puede ver que aún cuando se use $pe = 128$, el valor de CT es 156249,

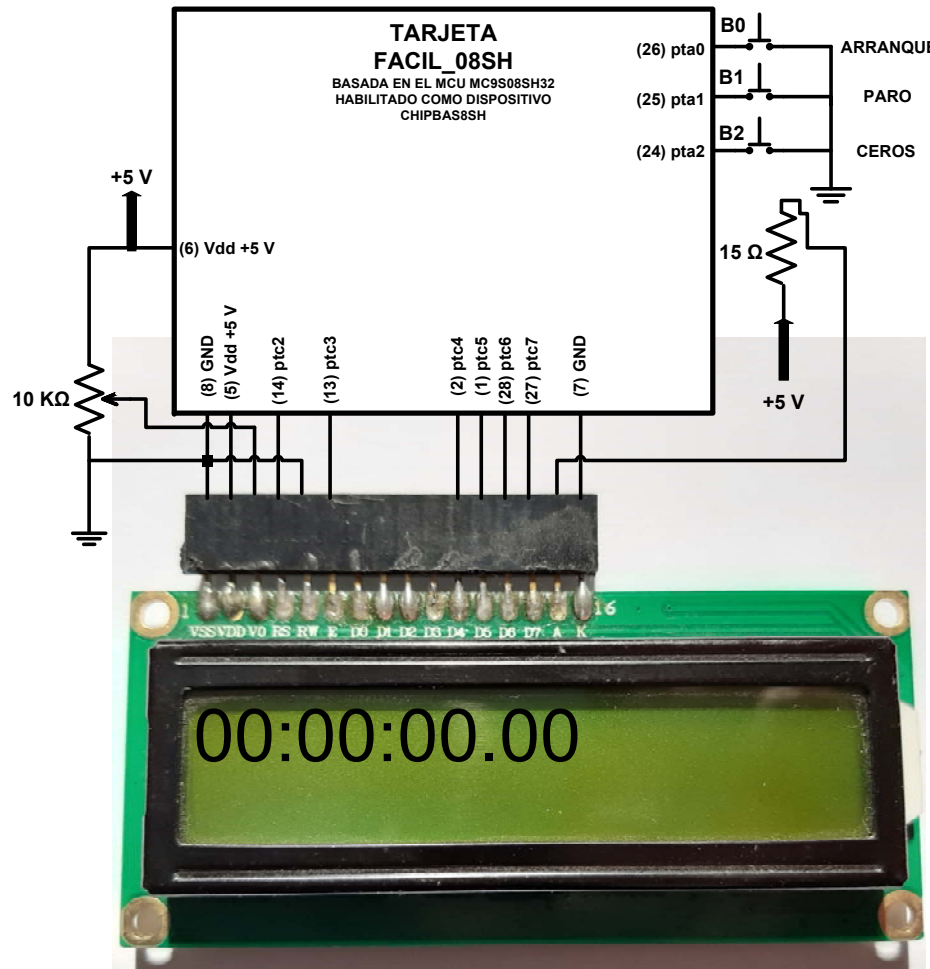
¡que no cabe en 16 bits!

Dado que no se puede lograr un par (CT,pe) para que T_{ovf} sea 5 segundos, para lograr el tiempo entre accionamientos deseado, se configura el temporizador para un intervalo entre overflows que sea un submúltiplo exacto del tiempo entre accionamientos deseado. En este ejemplo se usó $T_{ovf} = 250 \text{ mS}$, y se usa un contador base 20 en RAM, que se incrementa cada que el MCU entra a la rutina de servicio asociada. Dado que esto sucede cada 250ms, al llegarse a la cuenta 20 habrán transcurrido cinco segundos, es entonces que se efectúa el accionamiento deseado y se regresa a cero el contador auxiliar, para que quede listo para el transcurso de otros cinco segundos.

El programa fuente para este ejemplo está en el archivo **parp5seg.asm**

INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

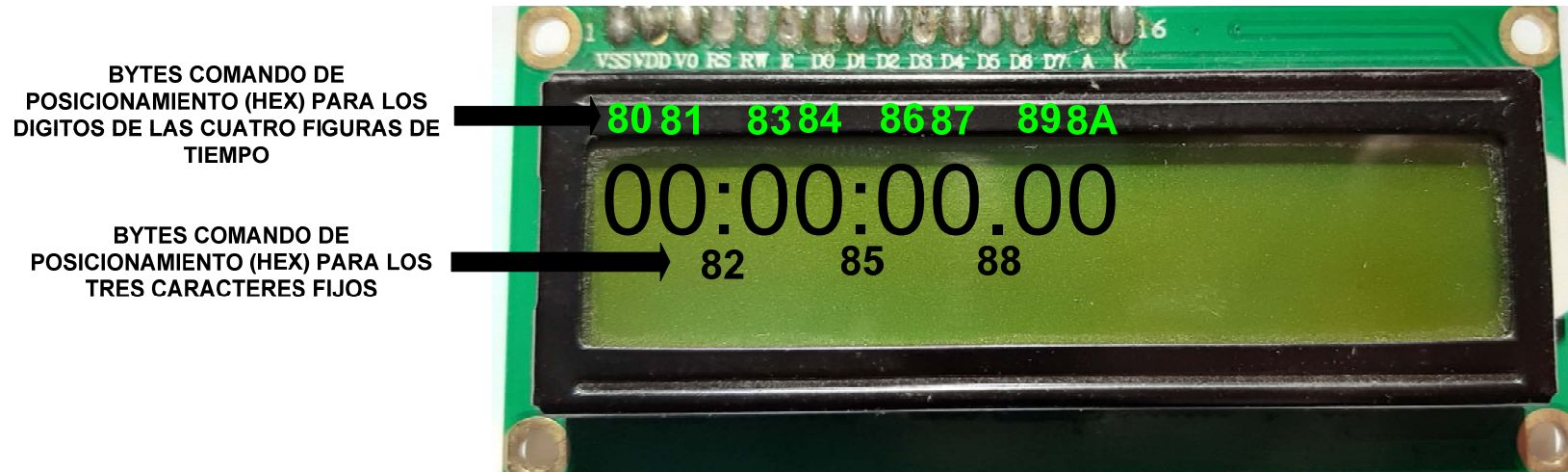
Ejemplo 5: Escribir un programa en ensamblador, que implique al temporizador 1 del MCU, de modo que al ejecutarse, se valide la funcionalidad del cronómetro mostrado en la figura. Ahí se aprecia que éste deberá contar con un botón de arranque, ligado con pta0; un botón de paro, ligado con pta1; y un botón de puesta a cero, ligado con pta2. La resolución deberá ser de horas, minutos, segundos y centésimas. El tiempo deberá correr de las 00:00:00.00 a las 23:59:59.99. Las cuatro figuras de tiempo se deben desplegar en un LCD de 16x2 a partir de la columna 1 del renglón 1. El programa fuente asociado deberá almacenarse en el archivo **cronobotb.asm**



INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

Ejemplo 5

Por ser necesario para fines del despliegado de los caracteres fijos, y de los asociados con las cuatro figuras de tiempo, aquí se muestran los bytes comando de posicionamiento asociados con éstos.



Nota: A partir de la ecuación (1) mostrada en la lámina 11 del archivo Tema2_manejo_de_lcd.pdf, puede verse fácilmente que los bytes de posicionamiento para las columnas 1 a 16 del renglón 1 respectivamente son: 0x80, 0x81, 0x82, ..., 0x8E y 0x8F. En lo que toca al renglón 2, los bytes de posicionamiento para las columnas 1 a 16 respectivamente son: 0xC0, 0xC1, 0xC2, ..., 0xCE y 0xCF.

INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

Ejemplo 5: Código básico que deberá ejecutarse cada centésima de segundo

Las cuatro figuras de tiempo se asocian con sendos contadores en la página cero de la RAM del MCU, denominados como: **conthor** para las horas, que deberá contar de 0 a 24; **contmin** para los minutos, que deberá contar de 0 a 60; **contseg** para los segundos, que deberá contar de 0 a 60; y **cont100** para las centésimas que deberá contar de 0 a 100.

Dado que el mínimo intervalo de tiempo implicado es una centésima de segundo, o sea, 10 mS, se configura el temporizador 1 de modo que se de una interrupción por overflow cada 10 mS.

Las cuentas de los cuatro contadores deberán iniciarse a cero en el bloque 1 del programa y deberán actualizarse **cada centésima (10 mS)**, por lo tanto, la rutina de servicio de interrupción deberá:

1. Regresar a cero la bandera testigo TOF
2. Se incrementa el contador de centésimas (cont100), si éste no ha llegado 100, se pasa al paso 6, pero si éste llega a 100, esto es, transcurrió ya un segundo más, se regresa a cero cont100 y se pasa al paso 3
3. Se incrementa el contador de segundos (contseg), si éste no ha llegado 60, se pasa al paso 6, pero si éste llega a 60, esto es, transcurrió ya un minuto más, se regresa a cero contseg y se pasa al paso 4
4. Se incrementa el contador de minutos (contmin), si éste no ha llegado 60, se pasa al paso 6, pero si éste llega a 60, esto es, transcurrió ya una hora más, se regresa a cero contmin y se pasa al paso 5
5. Se incrementa el contador de horas (conthor), si éste no ha llegado 24, se pasa al paso 6, pero si éste llega a 24, esto es, transcurrió ya un día más, se regresa a cero conthor y se pasa al paso 6
6. Retorna de la interrupción

INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

Ejemplo 5: Código de la rutina de servicio de interrupción

Aquí se muestra un posible código para validar la subrutina de servicio de interrupción, que es parte del bloque 3 del programa , y cuyo inicio se denota con la etiqueta 'sertof:'

**** Subrutina de servicio interrupción ****

```
sertof:  lda tpm1sc
        bclr 7,tpm1sc ;tof <-- 0
        inc cont100
        lda cont100
        cmp #$64 ;Si cont100 no ha llegado a 100 sale,
        bne salir ;si cont100 llega a 100,limpia cont100 y pasa a incrementar contseg
        clr cont100
        inc contseg
        lda contseg
        cmp #$3C ;Si contseg no ha llegado a 60 sale,
        bne salir ;si contseg llega a 60,limpia contseg y pasa a incrementar contmin
        clr contseg
        inc contmin
        lda contmin
        cmp #$3C ;Si contmin no ha llegado a 60 sale,
        bne salir ;si contmin llega a 60,limpia contmin y pasa a incrementar conthor
        clr contmin
        inc conthor
        lda conthor
        cmp #$18 ;Si conthor no ha llegado a 24 sale,
        bne salir ;si conthor llega a 24,limpia conthor
        clr conthor
salir:   rti
```


INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

Ejemplo 5

Aplicando la ecuación (3), puede verse que, para que T_{ovf} sea 10 mS, un par viable (p_e, CT) es $p_e=8$, lo que implica que n_{pe} sea 3, y $CT = 24999 = 0x61a7$.

Por otra parte, al inicializar el temporizador no se debe habilitar la interrupción, sólo se debe configurar lo propio, para que T_{ovf} sea 10 mS, ya que será la detección de la opresión del botón de arranque (B0), el evento que habilite localmente la interrupción haciendo **TOIE = 1**, y la opresión del botón paro (B1), el evento que deshabilite localmente la interrupción haciendo **TOIE = 0**. Por lo tanto, para fines de la inicialización del temporizador usado, el valor del byte **valsc** que se debe cargar en el registro **TPM1SC**, se obtiene empleando la ecuación (4) con $n_{pe} = 3$ y $h = 0$; o sea , **valsc = 11 = 0x0b**.

INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

Ejemplo 5

Sentencias equ requeridas

El programa a desarrollar implica datos a usar como pueden ser entre otros: los códigos ASCII de los caracteres fijos, la cuenta tope, las localidades de RAM donde residirán los cuatro contadores asociados con las figuras de tiempo. Esta información se declara en las sentencias equ presentes en el encabezado del programa, éstas se muestran a continuación:

```
asciip equ $2e ;Código ASCII del caracter '.'  
ascii2p equ $3a ;Código ASCII del caracter ':'  
conthor equ $90  
contmin equ $91  
contseg equ $92  
cont100 equ $93  
tpm1sc equ $20  
tpm1modh equ $23  
ctope equ $61a7 ;Cuenta tope (CT)  
pardig equ $95  
pardig2 equ $96
```

```
ptad equ $00  
ptadd equ $01  
ptape equ $1840
```

INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

Ejemplo 5 Continuación

:

***** Sentencias equ para definir bytes de posicionamiento requeridos *****

bypos2p1 equ \$82 ;byte comando para posición de caracter ':' entre horas y minutos

bypos2p2 equ \$85 ;byte comando para posición de caracter ':' entre minutos y segundos

bypospto equ \$88 ;byte comando para posición de caracter '.' entre segundos y centésimas

byposhor equ \$80 ;byte comando de posicionamiento para las horas

byposmin equ \$83 ;byte comando de posicionamiento para los minutos

byposseg equ \$86 ;byte comando de posicionamiento para los segundos

byposcent equ \$89 ;byte comando de posicionamiento para las centésimas

INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

Ejemplo 5: Estructura del bloque 1 del programa asociado con la realización del cronómetro mostrado en la lámina 27

Este bloque deberá efectuar las siguientes acciones de inicialización y configuración de recursos del MCU, empleados para fines de la realización del cronómetro

1. Habilitar resistencias de pull-up internas para los tres bits pta0, pta1 y pta2 que reciben el nivel lógico de los botones B0, B1 y B2
2. Inicializa el desplegado (LCD)
3. Colocar en las columnas del LCD que corresponda los caracteres fijos ':' y '.'
4. Configura temporizador 1 de modo que, $T_{ovf} = 10 \text{ mS}$ sin habilitar interrupciones por overflow
5. Inicializa a cero las cuatro figuras de tiempo, horas, minutos, segundos y centésimas
6. Habilita globalmente las interrupciones

En las siguientes dos láminas se detalla el código que corresponde a las acciones de inicialización

INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

Ejemplo 5: Estructura del bloque 1 del programa asociado con la realización del cronómetro mostrado en la lámina 27

Código en el bloque 1, asociado con las acciones de inicialización 1, 2 y 3

```
lda #$07 ;Pta2,pta1 y pta0  
sta ptape ;tienen 'pull-ups' internos.
```

```
jsr inilcd ;Inicializa LCD
```

**** Coloca caracteres fijos ':' y '.' *******

```
lda #bypos2p1  
jsr escom4 ;Posiciona siguiente escritura en renglón 1 columna 3  
lda #ascii2p  
jsr escdat4 ;Coloca caracter ':' en renglón 1 columna 3
```

```
lda #bypos2p2  
jsr escom4 ;Posiciona siguiente escritura en renglón 1 columna 6  
lda #ascii2p  
jsr escdat4 ;Coloca caracter ':' en renglón 1 columna 6
```

```
lda #bypospto  
jsr escom4 ;Posiciona siguiente escritura en renglón 1 columna 9  
lda #asciip  
jsr escdat4 ;Coloca caracter '.' en renglón 1 columna 9
```

INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

Ejemplo 5: Estructura del bloque 1 del programa asociado con la realización del cronómetro mostrado en la lámina 27

Código en el bloque 1, asociado con las acciones de inicialización 4, 5 y 6

**** Configura temporizador 1 para que $T_{ovf} = 10$ mS sin habilitar interrupciones por overflow *******

```
ldhx #modc
sthx tpm1modh
mov #$0b,tpm1sc ;toie--0,clksb:clksa <--01, pe=8,tovf=10 ms.
```

***** Inicializa a cero los cuatro contadores en RAM,**

***** asociados con las cuatro figuras de tiempo.**

***** Horas está en conthor,minutos está en contmin**

***** segundos está en contseg y centésimas está en cont100**

```
clr conthor
clr contmin
clr contseg
clr cont100
```

```
cli ; habilita interrupciones globalmente
```

INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

Ejemplo 5

Estructura del bloque 2 del programa asociado con la realización del cronómetro mostrado en la lámina 27

Las acciones que debe efectuarse mediante el código que integra este bloque son:

1. Checar si alguno de los tres botones está oprimido, si es el caso llevar a cabo la acción que corresponda acorde con el botón que se oprimió, después de esto, pasar al paso 2. Si ningún botón está oprimido, pasar directamente al paso 2.
2. Desplegar en el LCD las cuatro figuras de tiempo expresadas en decimal. Para esto se invoca una subrutina cuyo inicio se denota con la etiqueta 'desptemp:'.
3. Regresar al paso 1

INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

Ejemplo 5

Estructura del bloque 2 del programa asociado con la realización del cronómetro mostrado en la lámina 27

Aquí se muestra el código que integra al bloque 2 del programa

lazo: brclr 0,ptad,arranque

brclr 1,ptad,paro**brclr 2,ptad,ceros**

desple: bsr desptemp ;Esta subrutina despliega las figuras de tiempo en el LCD. Se ejecuta en aprox 42.5 mS

bra lazo

```
arranque: bset 6,tpm1sc ;toie <-- 1
```

bra desple

```
paro:  bclr 6,tpm1sc ;toie <-- 0
```

bra desple

ceros: clr conthor

clr contmin

clr contseg

```
clr cont100
```

bra desple

INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

Ejemplo 5

Estructura del bloque 3 del programa asociado con la realización del cronómetro mostrado en la lámina 27

El bloque 3 del programa está integrado por los siguientes componentes:

- **Subrutina de servicio de interrupción por overflow**, ésta se mostró previamente en la lamina 30.
- **Subrutina desptemp**, que despliega las cuatro figuras de tiempo en el LCD. Se auxilia de una subrutina auxiliar denominada auxdesp.
- **Subrutina con_a**, que convierte un byte menor que 100 a su representación en decimal, como un par de códigos ASCII. Esta subrutina se invoca cuatro veces desde la subrutina desptemp, para poder desplegar las cuatro figuras de tiempo en decimal en el LCD.
- **Subrutinas inilcd, escom4, escdat4 y copiadis**, contenidas en el archivo **rutsswlcsh32_20mhz.asm**, que se incluye con una sentencia `$include`.

El detalle y funcionalidad de las subrutinas que integran el bloque 3 puede verse en el archivo **coronobotb.asm**, que contiene el programa que valida el cronómetro.

INTERRUPCIONES PERIÓDICAS EN LOS MICROCONTROLADORES

Ejemplo 5

Estructura del bloque 4 del programa asociado con la realización del cronómetro mostrado en la lámina 27

El bloque 4 del programa está integrado únicamente por la declaración que valida la colocación del vector asociado con la instancia 11 de interrupción del MCU, cuyo evento asociado es el overflow del temporizador 1. A partir de la tabla 5-2 de la página 64 del manual del MCU, y considerando la ubicación de los vectores de interrupción para un dispositivo CHIPBAS8SH, es fácil ver que la declaración de la colocación del vector de interrupción, es la que se muestra a continuación:

```
org $d7e8 ;Colocación del vector de interrupción  
dw sertof ;propio del evento de overflow del temporizador 1
```

- **El programa completo del ejemplo 5, está en el archivo cronobotb.asm**

SURUTINA DE RETARDO BÁSICA

```
ret:  pshh  ;[2]
      pshx  ;[2]
      ldhx  #Xr      ;Xr es un número de 16 bits ;[3]
```

```
vuelta: nop    ;[1]                <<
        nop    ;[1]
        aix  #$ff ;h:x <-- h:x - 1 ;[2]
        cphx  #$0000      ;[3]
        bne  vuelta      ;[3]      >>
```

```
        nop    ;[1]
        pulx   ;[3]
        pulh   ;[3]
        rts    ;[5]
```

$$Tr = NC.Tb$$

$$NC = 19 + 10Xr + 6 = 25 + 10 Xr$$

$$Tr = (25+10Xr)Tb, \text{ si } Xr \geq 1$$

$$Tr = 655385Tb, \text{ si } Xr = 0$$

$$Tr_{min} = 35.Tb = 1.75 \text{ uS}, \text{ si } Tb = 50 \text{ nS}$$

$$Tr_{max} = 655385Tb = 32.76925 \text{ mS}, \text{ si } Tb = 50 \text{ nS}$$

$$Xr = ((Tr/Tb)-25)/10$$

Ejemplo 1

Se desea que la rutina de retardo aquí explicada genere un retardo de 10 mS. Suponer Tb = 50 nS. Determinar Xr en hexadecimal.

$$Xr = 19997.5 \text{ aprox a } 19997 = \$4e1d$$

Como hubo truncamiento de la parte fraccionaria de Xr, El retardo real es: $Tr = (25 + 199970)50nS = 9.99975 \text{ mS}$

Ejemplo 2

Se desea que la rutina de retardo aquí explicada genere un retardo de dos segundos. Suponer Tb = 50 nS. Determinar Xr en hexadecimal.

$$Xr = ((2/50 \times 10^{(-9)})-25)/10$$

$$Xr = 39999997.5 \text{ aprox a } 39999997 = \$3d08fd$$

¡ QUE NO CABE EN 16 BITS !

Dado que Xr no cabe en 16 bits,el retardo de dos segundos no puede realizarse con la subrutina básica aquí expuesta.

El retardo mínimo (Trmin) que se podría generar es:

$$\text{Si } Tb = 50 \text{ nS}$$

$$Tr_{min} = 1.75 \text{ uS } (Xr = 1)$$

$$Tr_{max} = (25 + 655360)50 \text{ nS} = 32.76 \text{ mS}$$