Interrupts in C

Main Points to Remember:

1. The interrupt vector is set by the linker through the use of the Linker Parameter (.PRM) file.

2. The keyword "interrupt" must precede each of the interrupt routine declarations.

Setting up a new CodeWarrior C project

- **The following example implements a simple system that responds to a falling edge IRQ interrupt that is produced by a bounceless pushbutton connected to the PE1/IRQ input pin (Pin 2) of our CSMB12C128 module.**

- **In response to this falling edge, and LED (on PT1) is toggled and a counter (on Port M) is incremented.**

Complete Example of a C program that incorporates an interrupt service routine

- First open CodeWarrior, and click
  File – New – HC(S)12 New Project Wizard
- Enter Project name and desired path
- Click OK and Next
- Enter MC9S12C128 and Next
- Check C and uncheck Assembly and Next
- Check NO to Processor Expert and Next

- Check NO PC-lint and Next
- Check ANSI startup code and Next
- Check None for floating point and Next
- Check Small for memory model
- Check both “Full-Chip Simulation” and
  also “P&E Multilink/Cyclone Pro” and Finish
- Once the “C” project is created, click on “PRM”. Then select (double left click on) the “P&E Multilink/Cyclone Pro” Linker Parameter file.

Now add the desired interrupt vector initialization to the .PRM file

- In our case we want the function we shall call IRQISR to be entered when an IRQ interrupt occurs, which is Vector 6.
- Therefore, add the following line to the end of the .PRM file:

VECTOR 6 IRQISR

Note: Vector 6 is the IRQ vector, initialize it with address of IRQISR

Copy the following program into the main.c function

```
//Example of using C with interrupts
//Bounceless SW connected to Pin 2 (PE1/IRQ input)
//LED connected to PT1
//Interrupt count set up on Port A.
//Interrupt vector for IRQ is initialized to
// point to the IRQISR routine at the end of
// the .PRM file.
#include <hidef.h>      /* common defines and macros */
#include <mc9s12c128.h>     /* derivative information */
#pragma LINK_INFO DERIVATIVE "mc9s12c128"
void          IRQISR(void);
unsigned char cnt;      /* This is global variable is used for
                  communicating between main program
                  and ISR, so it MUST be defined outside
                  of the main program   */
void main(void) {
              DDRA        = 0xFF;
              cnt         = 0;
              PORTA       = 0;
              DDRT        = 0x02;       /* configure PT1 pin for output */
              PTT         = 0x02;       /* enable LEDs to light */
              INTCR       = 0xC0;     /* enable IRQ (PE1/IRQ, Pin2) interrupt
                                         on falling edge */
              asm("cli");               /* enable interrupt globally */
                                        /* Alternately you could invoke “EnableInterrupts;” */
              while(1);                 /* wait for interrupt forever */
}
```

```
interrupt void IRQISR(void)
{
        cnt++;
        PORTA= cnt;   //Increment count on Port A
        PTT = ~PTT;  // Toggle LED on PT1
        //Since IRQ input is made edge sensitive,
        //The IRQ interrupt is automatically
        //relaxed after the interrupt routine is
        //entered, so there is no need to "shut the
        //baby up" as with most other"I-bit related"
        //interrupts.  Upon return from IRQISR, the
        //interrupt is already inactive.
}
```


Configuring PE1/IRQ pin for interrupt input

- **Note that the INTCR (interrupt control register) must be set to $C0 in order to configure pin PE1 as a “falling edge” sensitive IRQ input pin instead of a PORT E digital I/O pin.**

INTCR Register

(From 9S12C128DGV1.PDF Design Guide Document)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$001E	INTCR	Read:	IRQE	IRQEN	0	0	0	0	0	0
		Write:								

2.3.14 PE1 / $\overline{\text{IRQ}}$ — Port E input Pin [1] / Maskable Interrupt Pin

The $\overline{\text{IRQ}}$ input provides a means of applying asynchronous interrupt requests to the MCU. Either falling edge-sensitive triggering or level-sensitive triggering is program selectable (INTCR register). $\overline{\text{IRQ}}$ is always enabled and configured to level-sensitive triggering out of reset. It can be disabled by clearing IRQEN bit (INTCR register). When the MCU is reset the $\overline{\text{IRQ}}$ function is masked in the condition code register. This pin is always an input and can always be read. There is an active pull-up on this pin while in reset and immediately out of reset. The pull-up can be turned off by clearing PUPEE in the PUCR register.

Denoting Interrupt Routines in C

- **Note how the interrupt routine begins with the keyword "interrupt". It must also be given a name, so that the interrupt vector can be initialized using the name of the interrupt routine. Use of this keyword ensures that the routine will end with an RTI instead of just an RTS.**
- **There should be as many interrupt routines defined in this way as you have interrupt sources.**

This program will be interrupted to increment PORT A and then toggle the LED on PT1 with each falling edge on PE1/IRQ pin!