PIC32 Interrupts

2015-08-19

Content

1 Purpose of this document

This document will identify and elucidate the items that are related to interrupts on the PIC32. All in this document was experienced/discovered during the writing of RTOS for PIC32, see http://www.libstock.com/projects/view/1427/rtos-for-pic32.

Thanks also to
JPC: see http://www.mikroe.com/forum/viewtopic.php?f=173&t=64127&p=255475#p255475,
Rotary_Ed and Jim Keuneman see http://www.mikroe.com/forum/viewtopic.php?f=172&t=57488&p=229272.

2 Overview

The interrupt related issues are:

- Enabling and disabling interrupts, see section 4
- Interrupt priorities, see section 5
- Context saving and restoration, see section 6
- The shadow register set, see section 7
- Usage of the interrupt assistant, see section 8
- Interrupt flags, see section 9 (including how to prevent persistent interrupts)

3 Order of activities

Order in which interrupt related activities should be done:

1. **In the configuration settings**
    a. Select the SRS priority level (if SRS present), see section 7.
2. **In the interrupt's initialisation code**
    a. Set the interrupt priority level bits, see section 5
    b. Clear the interrupt flag, see section 9
    c. Set the interrupt specific enable bit, see section 4
3. **Create the interrupt procedure body**
    a. Use the Interrupt Assistant Tool (see section 8) or the Timer Calculator tool to do this.
4. **In the interrupt procedure itself**
    a. Clear the interrupt flag, see section 9
    b. Make sure the interrupt flag is not immediately set again, see section 9.2
5. **Right before the main program loop**
    a. Globally enable interrupts, see section 4

4 Enabling and disabling Interrupts

4.1 The interrupt specific enabling

Each interrupt source has an 'interrupt enable' bit, named **xxIE_bit**, where xxx is the (short) name of the interrupt source.
For example: Timer1 (T1 for short) has the **T1IE_bit**, which, when set to "1", enables the Timer1 interrupt.


Do not forget to set this bit to '1' if you want the interrupt to fire.

4.2 The global interrupt enabling

There is also a 'global' interrupt enable mechanism, which does not work via an enable bit, but with calling procedures:
`EnableInterrupts` and `DisableInterrupts.`

If the interrupts are not globally enabled then all interrupts are disabled.


Do not forget to add 'EnableInterrupts' in your code if you want interrupts to work.

5 Setting Interrupt Priorities

5.1 Description

Interrupt priorities range from 0..7, **7** being the **highest** one, **0** being the interrupt **disabled**.
An interrupt routine can only be interrupted itself by an interrupt of a higher priority.

The interrupt priority is set in in the **xxIP0_bit ... xxIP2_bit**, where xx is the (short) name of the interrupt source.
For example: Timer1 (T1 for short) has 3 bits which define its interrupt priority:
`T1IP0_bit`, `T1IP1_bit` and `T1IP2_bit.`


Do not forget to set the interrupt priority in your code.

The interrupt priority should be set in e.g. an "init" routine before the interrupt is enabled.
A simple, easy way is the following:

```
var Priority: byte;
...
  Priority  :=  4;          // wanted priority
  xxIP0_bit := Priority.0; // set the interrupt priority bits for "xx"
  xxIP1_bit := Priority.1; //    "
  xxIP2_bit := Priority.2; //    "
```

Above code sets the interrupt for interrupt source "xx" to priority 4.

Notes:

- ‘xx’ can have more than 2 characters. Find the correct bit names in the ‘defs’ file of your processor (crtl-alt D in the IDE)

5.2 Important

When using the Timer calculator tool, the delivered initialisation code will always set the priority of the timer to 7 (highest priority). Change this priority if needed (see above).

It is necessary to know which interrupt priority level the interrupt has, and set the parameters of the Interrupt Assistant tool correctly (see section 8.2).

6 Context Saving

6.1 Description

When servicing an interrupt the context of the interrupted process has to be saved.
The context of a process consists of all the processor’s general purpose registers (R1..R31) plus some other stuff. Normally the saving and restoring of the context is done by the compiler, unless one selects that no context should be saved (see section 8). In the latter case one has to write own code for saving and restoring the context (if wanted of course).

6.2 Important

It is necessary to know which interrupt priority will use the SRS (see section below), and set the parameters of the Interrupt Assistant tool correctly (see section 8.2).

7 The Shadow Register set (SRS)

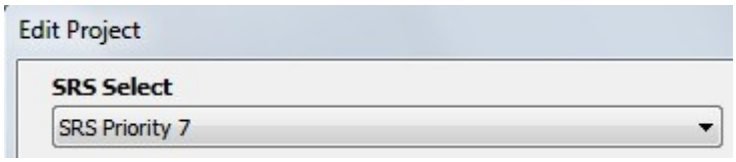
7.1 Description

In previous section was stated that R1..R31 is part of the context. The saving of his part of the context can be avoided by using a “Shadow Register Set”. This is simply an extra register set, used by the interrupt service routine, so that the register set contents of the interrupted process is not disturbed.
Using the SRS results minimal latency of the interrupt service routine invocation, because R1..R31 do not have to be saved or restored.

7.2 Important

Usage of the SRS is <u>priority dependent</u>. Only one interrupt priority level can use the Shadow Register Set. Of course more than one interrupt can be of the same priority level.

In the project setup (Edit Project in the IDE) one can choose which interrupt priority will use the SRS (this is default priority 7):



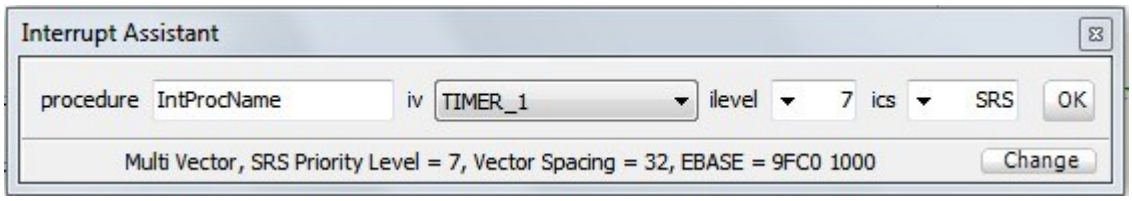
Above the default priority (7) is selected to use the SRS. The selected priority to use the SRS is also called the "**SRS priority Level**".

Take into account the "SRS priority level" in the selections to be made in the interrupt assistant tool (see next section).

8 The interrupt assistant tool

8.1 Description

The interrupt assistant tool provides the body of the interrupt service routine and is the most reliable manner to generate such. It can be started up from within the IDE: menu Tools -> Interrupt Assistant:



It has 4 fields:

- *procedure*: This is the name of the interrupt procedure, to be chosen freely
- *iv*: This is the **I**nterrupt **V**ector, the source of the interrupt
- *ilevel*: 1..7; This is the interrupt priority level you have chosen for the interrupt, see section 5
- *ics*: This is the selection of the Interrupt Context Saving, see sections 6 and 7

8.2 Important

"**ilevel**" does NOT define the interrupt priority of the interrupt (that is defined in the interrupt priority bits, see section 5). This item is only needed for the compiler, so it can prevent interrupts with the same or lower priority to break in in the interrupt service routine. The value set here should be the same as the interrupt priority used.

"**ics**" does NOT define the usage of the SRS (Shadow register set) or not. The usage of the SRS is priority dependent, see section 7. This item is only needed to warn the compiler that the context to be saved is much smaller than usually when using the SRS (or bigger when not using the SRS).

The possible values of 'ics' are:

- SRS: to be selected if (and only if) the priority of the interrupt is the one using the SRS (see section 7)
- SOFT: to be selected in case context saving is wanted and the SRS is not used.
- AUTO: I suppose this is an automatic selection between SRS and SOFT.
- OFF: when no context saving is wanted (the user has to write one then if needed).

9 Interrupt Flags

9.1 Description

Each interrupt source has an interrupt flag called xxIF_bit, wherein 'xx' is the (short) name of the interrupt source.

Example: Timer1 (in short T1) has its **T1IF_bit.**
Note: 'xx' can have more than 2 characters. Find the correct bit names in the 'defs' file of your processor (crtl-alt D in the IDE)

The interrupt flag causes the interrupt to fire provided the source's interrupt enable bit is set and 'EnableInterrupts' is called (see section 4).

 Interrupt flags must be cleared in the interrupt service routine, otherwise, when leaving the interrupt service routine, the interrupt will fire again. See also the important remarks below.

 The Interrupt flag must be cleared before the interrupt is enabled, otherwise the interrupt will fire immediately after enabling if the flag was set.

9.2 Important

 Ensure that the buffers of some modules (such as the IC capture) must be emptied OR you get the infamous persistent interrupt (interrupt flag set immediately again and interrupt firing accordingly).
Thanks Rotary_Ed for the tip.

 Be aware of the behavior of the mP compiler when using a 'dummy' (not further used) variable to clear the buffer (see above). In this case the compiler will not translate the mP statement to assembler, which means the buffer will not be cleared at all, resulting in persistent interrupt behavior. Thanks Rotary_Ed for the tip.
See this URL's: http://www.mikroe.com/forum/viewtopic.php?f=172&t=63265&p=258778#p258778 and http://www.mikroe.com/forum/viewtopic.php?f=172&t=62535&p=248553#p248553.

10 Examples

10.1 Simple 1 ms timer

This is a simple 1 milliseconds timer, using Timer1, interrupt priority 7 and the context saving using the Shadow register set. The example is made for the Mini32 board (P32MX534F064H with CPU clock of 80 Mhz).

```
program Test_Timer1_Interrupt;

{ Declarations section }

var DATA_LED : sbit at LATD6_bit;
var STAT_LED : sbit at LATG6_bit;

//Timer1
//Prescaler 1:8; PR1 Preload = 10000; Actual Interrupt Time = 1 ms
//Code generated by the Timer Calculator tool, statement order adapted manually
procedure InitTimer1();
begin
  T1CON       := 0x8010;

  T1IP0_bit   := 1;                          // interrupt priority level 7
  T1IP1_bit   := 1;                          //        "
  T1IP2_bit   := 1;                          //        "

  T1IF_bit    := 0;                          // Clear the Timer1 interrupt flag
```

```
  T1IE_bit    := 1;                          // Enable Timer1 interrupt

  PR1         := 10000;
  TMR1        := 0
end;

//Interrupt procedure body generated by the Interrupt Assistant tool
procedure Timer1Interrupt(); iv IVT_TIMER_1; ilevel 7; ics ICS_SRS;
begin
  T1IF_bit    := 0;
  //Enter your code here
  DATA_LED := not DATA_LED;
end;

begin
  { Main program }
  AD1PCFG := 0xFFFF;                         // configure AN pins as digital
  TrisD6_bit := 0;
  DATA_LED := 0;

  InitTimer1;
  EnableInterrupts();                        // Enable all interrupts
end.
```

[end of document]