

UTMC Application Note

UT80CRH196KD Interrupt Priority Architecture

The UT80CRH196KD delivers a very powerful, multilevel interrupt priority architecture. This application note is intended to explain the differences between the UT80CRH196KD and the industry standard 8XC196KD interrupt architectures. Further, it will discuss how to use the UT80CRH196KD's multilevel interrupt architecture to simplify pending interrupt prioritization.

1.0 UT80CRH196KD vs. Industry Standard 80196 Interrupt Architecture

1.1 Industry Standard 8XC196KD Interrupt Architecture

The industry standard 8XC196KD uses a 4 level interrupt priority scheme. The 4 levels of this interrupt architecture include Non-Prioritized Interrupts, Non-Maskable Interrupt (NMI), Peripheral Transaction Server (PTS) Interrupts, and Standard (maskable) Interrupts. The two Non-Prioritized Interrupts are (a) the Unimplemented Opcode, and (b) the Software Trap. When either of these two interrupts occur, the interrupt controller will service them ahead of any other pending interrupt. The next highest priority interrupt is the NMI. The NMI has higher priority than the PTS Interrupts, and the Standard (maskable) Interrupts. The PTS Interrupts hold the next level of priority following the NMI. As long as there are no NMI, Unimplemented Opcode, or Software Trap interrupts pending, then the interrupt controller gives precedence to the PTS interrupts. All other prioritized interrupts reside below the PTS Interrupts in the Standard Interrupt level. Increase the priority of these interrupts by assigning them to a PTS interrupt; however, you have to rely on the microcoded PTS routines to service these interrupts. Table 1 shows the default interrupt priorities (15 is highest and 0 is the lowest) for both, the UT80CRH196KD, and the industry standard 8XC196KD.

Table 1: Interrupt Vector Sources, Locations, and Priorities

Number	Interrupt Vector	Source(s)	Interrupt Vector Location	PTS Vector Location	Priority ¹ (0 is the Lowest Priority)
Special	Unimplemented Opcode	Unimplemented Opcode	2012h	N/A	N/A
Special	Software Trap	Software Trap	2010h	N/A	N/A
INT 15	NMI ²	NMI	203Eh	N/A	15
INT 14	HSI FIFO Full	HSI FIFO Full	203Ch	205Ch	14
INT 13	EXTINT 1 ²	Port 2.2	203Ah	205Ah	13
INT 12	Timer 2 Overflow	Timer 2 Overflow	2038h	2058h	12
INT 11	Timer 2 Capture ²	Timer 2 Capture	2036h	2056h	11

Table 1: Interrupt Vector Sources, Locations, and Priorities

Number	Interrupt Vector	Source(s)	Interrupt Vector Location	PTS Vector Location	Priority ¹ (0 is the Lowest Priority)
INT 10	HSI FIFO 4	HSI FIFO Fourth Entry	2034h	2054h	10
INT 9	Receive	RI Flag ³	2032h	2052h	9
INT 8	Transmit	TI Flag ³	2030h	2050h	8
INT 7	EXTINT ²	Port 2.2 or Port 0.7	200Eh	204Eh	7
INT 6	Serial Port	RI Flag and TI Flag ⁴	200Ch	204Ch	6
INT 5	Software Timer	Software Timer 0-3 Timer 2 Reset	200Ah	204Ah	5
INT 4	HSI.0 ²	HSI.0 Pin	2008h	2048h	4
INT 3	High Speed Outputs	Events on HSO.0 thru HSO.5 Lines	2006h	2046h	3
INT 2	HSI Data Available	HSI FIFO Full or HSI Holding Reg. Loaded	2004h	2044h	2
INT 1	EDAC Bit Error	Single Bit Error Single Bit Error OVF Double Bit Error	2002h	2042h	1
INT 0	Timer Overflow	Timer 1 or Timer 2	2000h	2040h	0
<p>All of the previous maskable interrupts can be assigned to the PTS. Any PTS interrupt has priority over all other maskable interrupts.</p>					

Notes:

1. The Unimplemented Opcode and Software Trap interrupts are not prioritized. The Interrupt Controller immediately services these interrupts when they are asserted. NMI has the highest priority of all prioritized interrupts. Any PTS interrupt has priority over lower priority interrupts, and over all other maskable interrupts. The standard maskable interrupts are serviced according to their priority number with INT0 has the lowest priority of all interrupts.
2. These interrupts can be configured to function as independent, external interrupts.
3. If the Serial interrupt is masked and the Receive and Transmit interrupts are enabled, the RI flag and TI flag generate separate Receive and Transmit interrupts.
4. If the Receive and Transmit interrupts are masked and the Serial interrupt is enabled, both RI flag and TI flag generate a Serial Port interrupt.

There are two ways to increase the priority of Standard Interrupts. The first way to increase interrupt priority is to assign the Standard Interrupt to a PTS interrupt. Thus, allowing the PTS controller to service the specific interrupt ahead of all other Standard Interrupts. Secondly, place lower priority interrupts ahead of higher priority Standard Interrupts via software service routines. The following example shows how software can change interrupt priorities during an interrupt service routine.

Example 1:

```

Timer_2_Overflow_ISR:

pusha                ; Store the PSW, INT_MASK, INT_MASK1, and WSR onto the stack.
                    ; The pusha instruction also masks all interrupts.

ldb int_mask, #40h  ; Masks all interrupts associated with INT_MASK except the Serial
                    ; Port interrupt.

ei                   ; Enables all interrupts not masked by the interrupt mask registers.

.
.                   ; Timer 2 Overflow interrupt service routine body
.

popa                 ; Restore the PSW, INT_MASK, INT_MASK1, and WSR
                    ; from the stack.

ret                  ; Return from the interrupt service routine.

```

The microcontroller enters this subroutine after receiving a Timer 2 Overflow interrupt (Interrupt Priority 12). However, during the interrupt service routine, the Serial Port interrupt (Interrupt Priority 6) is enabled. As a result, a Serial Port interrupt, following the execution of the “ei” instruction, may interrupt the Timer 2 Overflow ISR at any time. Therefore, the Serial Port has effectively been given a higher priority than the Timer 2 Overflow.

1.2 UT80CRH196KD Interrupt Architecture

The interrupt architecture of the UT80CRH196KD has one distinct advantage over the industry standard 8XC196KD interrupt architecture. In addition to all the functionality of the industry standard 8XC196KD described in Section 1.1, the UT80CRH196KD offers two specific Standard Interrupt levels. The two separate interrupt levels increase flexibility and ease prioritization of pending interrupt events. The multilevel priority is set by writing to the Interrupt Priority Register (INT_PRI) located at address 0Ah of HWindow 1. The INT_PRI register has the same mapping as INT_MASK in the lower byte, and INT_MASK1 in the upper byte. A logic 1 in a position of the INT_PRI register increases the priority of that interrupt above priority 14, but lower than the PTS Interrupt priorities. Therefore, a Standard Level 1 interrupt will take priority over all Standard Level 0 (default) interrupts. Furthermore, if the Interrupt Priority Register is set to 0000h

(all zeroes) or FFFFh (all ones), the interrupt controller will operate the same as the industry standard 8XC196KD. The following example shows how to increase the priority level of several Standard Interrupts.

Example 2:

Assuming that you would like to have HSI FIFO Full, Serial Port, Software Timer, EDAC, and Timer Overflow interrupts to have a higher priority than the rest of the Standard Interrupts, then you would use the following line of code:

```
ld      int_pri , #4063h      ; (0100 0000 0110 0011 b)
```

By performing this instruction you would be placing these 5 interrupts into a higher level of interrupt priority. The interrupts placed in Standard Level 1, have the default prioritization scheme amongst themselves. Therefore, if the above instruction was executed, and all interrupt pending bits were simultaneously set, the following service order occurs:

1) NMI - **NMI will always be serviced first, regardless of the bit value of INT_PRI.**

Standard Level 1 Interrupts:

- 2) HSI FIFO Full
- 3) Serial Port
- 4) Software Timer
- 5) EDAC
- 6) Timer Overflow

Standard Level 0 Interrupts:

- 7) EXTINT 1
- 8) Timer 2 Overflow
- 9) Timer 2 Capture
- 10) HSI FIFO 4
- 11) Serial Receive
- 12) Serial Transmit
- 13) EXTINT
- 14) HSI.0 Pin
- 15) High Speed Outputs
- 16) HSI Data Available

Note, you may still want to use the software prioritization scheme shown in example 1 for long interrupt service routines. Additionally, you should be aware that other interrupts can not be serviced after the “pusha” or “pushf” instructions have been executed, unless they are re-enabled by setting their interrupt mask bits and executing the “ei” instruction. The “pusha” or “pushf” instructions mask all interrupts because they clear the INT_MASK, and INT_MASK1 registers.