

```

1: |-----|
2: |-----|
3: | File Name       : IARM_Nest_Interrupts_Info.txt
4: | Author         : Pride Embedded, LLC
5: | Copyright      : Copyright © 2005, Pride Embedded, LLC
6: | Creation Date  : 6/28/06
7: | Version        : 1.00
8: | Spaces per tab : 2
9: | Description     : Project information
10: | Revision       : Initial
11: |-----|
12: |-----|
13:
14: -----
15: Description
16: -----
17:
18: This demo project implements ARM FIQ (Fast Interrupt) and IRQ (Normal Interrupt) interrupts and
19: shows how a custom startup assembly file is used to branch to an FIQ interrupt handler. The FIQ
20: interrupt handler will interrupt the IRQ interrupt handler showing how hardware interrupt nesting
21: is implemented on the LPC2103.
22:
23: Before describing the details of the demo project, it's important to understand just how the
24: Vectored Interrupt Controller (VIC) works at a high level. There are many resources out there
25: that can describe this in much greater detail (LPC2103 datasheet, ARM website, etc.), but here's
26: some basics.
27:
28: You can assign an interrupt to be either an FIQ, vectored IRQ, or non-vectored IRQ. In this demo project,
29: the FIQ interrupt is assigned to Timer-0 and a vectored IRQ is assigned to EINT1 (External interrupt one)
30: which is the ISP dip switch on the iARM-2103 board. Non-vectored interrupts are not used in this demo project.
31:
32: When dealing with VIC FIQ's and IRQ's, the FIQ's are a higher priority interrupt than the IRQ's.
33: That means if an IRQ handler is executing, and an FIQ interrupt occurs, the FIQ interrupt handler
34: will take control and execute. Once the FIQ handler is done executing, the IRQ can then resume, and
35: once the IRQ handler is done executing, the main application can resume. So, it's easy to understand
36: that any interrupt assigned to an FIQ has priority over an interrupt assigned to an IRQ.
37:
38: Now, something that's usually learned the hard way, is that although you can assign the vectored IRQ's
39: to the 16 available IRQ slots of the VIC (Prioritized 0 to 15, with 0 being the highest priority), this
40: does not mean that one IRQ can interrupt another in a nested fashion. Although there are 16 assignable
41: priorities, they are cooperative in nature. Let's say a timer interrupt is assigned to priority 0; if
42: before the timer interrupt occurs a UART interrupt with priority 5 fires off. During this interrupt, if
43: the timer interrupt fires off, it cannot execute until the UART interrupt handler is finished. Once
44: the handler is finished, the next highest priority interrupt can handle, which is the timer interrupt in
45: this example. To test this, simply modify the provided demo project to use just IRQ's. Then, adjust
46: the priorities and see what happens. Simply stated, the vectored IRQ's are not hardware nestable. But,
47: using an FIQ with an IRQ let's hardware nesting occur.
48:
49: There are of course clever software tricks that can be used to work around the cooperative IRQ issue, like
50: setting flags in the interrupt handler, quickly exiting, and then executing the handler in the main application.
51: Or, creating a software stack in assembly code for the IRQ's, but if you just need one level of hardware nesting,
52: then simply use an FIQ to interrupt the IRQ handlers. In many cases, you'll have just one or two high priority
53: interrupts which can be assigned as an FIQ. This lets these interrupts have priority over any IRQ assigned
54: interrupt.
55:
56: Finally, we can describe what exactly the demo project does. In the main application, the IARM's red LED blinks
57: continuously. A 270 mS timer interrupt, assigned to be an FIQ, is also firing off continuously through our custom
58: startup file. The timer FIQ handler just turns off the IARM green LED and waits via a software delay for about 100
59: mS.
60: If you watch the red LED that blinks in the main application, you can see it occasionally gets interrupted and goes
61: out
62: of sync due to the FIQ timer interrupt firing off. Now, to show how the nesting happens, an external interrupt
63: (EINT1
64: setup as level sensitive which means interrupts occur whenever a low is on that pin) is assigned to be an IRQ.
65: This
66: external interrupt can be accessed via the iARM-2103 dip switch with the "ISP" label (Note, make sure the ISP dip
67: switch is off when booting the board, or else you'll go into ISP programming mode).
68:
69: When the ISP dip switch is off, the IARM red LED just blinks all the time with an occasional blip from the timer
70: FIQ firing off. Now, once the ISP dip switch is switched on (EINT1 goes low), you'll notice that the IARM red LED
71: no longer blinks. What now happens is the IARM green LED blinks. As mentioned previously, the timer FIQ interrupt
72: continuously turns the IARM green LED off, but now the EINT1 IRQ interrupt is active and it is turning the IARM
73: green LED on. So, why does the green LED blink then? The reason for this is that the EINT1 interrupt is active
74: all the time, this is why the main application no longer executes. That means with the EINT1 interrupt always
75: active,
76: it's interrupt handler is always turning the IARM green LED on...but, since we have a timer interrupt assigned to
77: an FIQ interrupt, the IRQ handler is interrupted by this FIQ interrupt which results in the IARM green LED being
78: turned off for a bit. That's why the IARM green LED blinks...it's all about the nesting!
79:
80:
81:
82:
83:
84:
85:
86:
87:
88:
89:
90:
91:
92:
93:
94:
95:
96:
97:
98:
99:
100:

```

75: In short, the main application can be interrupted by an FIQ and an IRQ, and an IRQ can be interrupted by an FIQ.  
76: This demonstrates the FIQ and IRQ interrupt nesting capability of the iARM-2103.  
77:  
78: Obviously, the above info is just one interpretation on the VIC, so below is more info to help describe the VIC.  
79:  
80: -----  
81: Info on the VIC from the Philips LPC2103 Datasheet  
82: -----  
83: The Vectored Interrupt Controller (VIC) takes 32 interrupt request inputs and  
84: programmably assigns them into 3 categories, FIQ, vectored IRQ, and non-vectored IRQ.  
85: The programmable assignment scheme means that priorities of interrupts from the  
86: various peripherals can be dynamically assigned and adjusted.  
87:  
88: Fast Interrupt reQuest (FIQ) requests have the highest priority. If more than one request is  
89: assigned to FIQ, the VIC ORs the requests to produce the FIQ signal to the ARM  
90: processor. The fastest possible FIQ latency is achieved when only one request is  
91: classified as FIQ because then the FIQ service routine can simply start dealing with that  
92: device. But if more than one request is assigned to the FIQ class, the FIQ service routine  
93: can read a word from the VIC that identifies which FIQ source(s) is (are) requesting an  
94: interrupt.  
95:  
96: Vectored IRQs have the middle priority, but only 16 of the 32 requests can be assigned to  
97: this category. Any of the 32 requests can be assigned to any of the 16 vectored IRQ slots  
98: among which slot 0 has the highest priority and slot 15 has the lowest.  
99: Non-vectored IRQs have the lowest priority.  
100:  
101: The VIC ORs the requests from all the vectored and non-vectored IRQs to produce the  
102: IRQ signal to the ARM processor. The IRQ service routine can start by reading a register  
103: from the VIC and jumping there. If any of the vectored IRQs are requesting, the VIC  
104: provides the address of the highest-priority requesting IRQs service routine, otherwise it  
105: provides the address of a default routine that is shared by all the non-vectored IRQs. The  
106: default routine can read another VIC register to see what IRQs are active.  
107: All registers in the VIC are word registers. Byte and halfword reads and write are not  
108: supported.  
109:  
110: Additional information on the Vectored Interrupt Controller is available in the ARM  
111: PrimeCell™ Vectored Interrupt Controller (PL190) documentation.  
112: -----  
113: -----