7 Interrupts

7.1 Introduction

The ARM has two types of interrupt sources:

1. Interrupts coming from the GPU peripherals.
2. Interrupts coming from local ARM control peripherals.

The ARM processor gets three types of interrupts:

1. Interrupts from ARM specific peripherals.
2. Interrupts from GPU peripherals.
3. Special events interrupts.

to enable timer interrupts, we want to enable / check "basic interrupts" / "basic pending". the rest, specifically UART and GPIO, you will worry about later.

The ARM specific interrupts are:

- One timer.
- One Mailbox.
- Two Doorbells.
- Two GPU halted interrupts.
- Two Address/access error interrupt

The Mailbox and Doorbell registers are not for general usage.

For each interrupt source (ARM or GPU) there is an interrupt enable bit (read/write) and an interrupt pending bit (Read Only). All interrupts generated by the arm control block are level sensitive interrupts. Thus all interrupts remain asserted until disabled or the interrupt source is cleared.

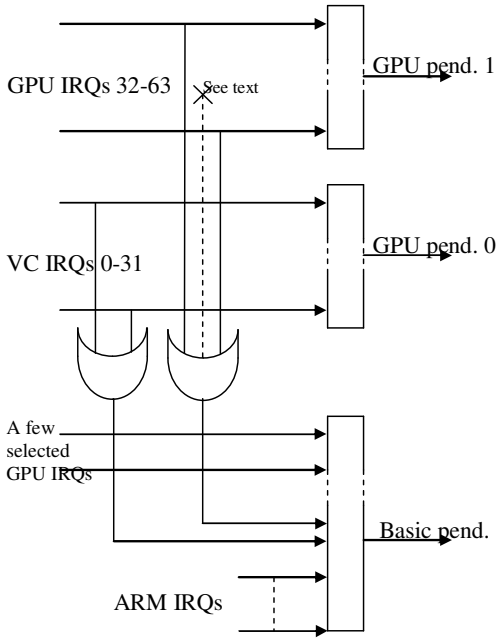
Default the interrupts from doorbell 0,1 and mailbox 0 go to the ARM this means that these resources should be written by the GPU and read by the ARM. The opposite holds for doorbells 2, 3 and mailbox 1.

so, to enable timer interrupts: have to figure out how to enable them, clear them, and --- since there are many possible interrupts --- how to check that they occured when in the interrupt handler.

in general, when we enable CPU state for the first time we have to clear it (since it will typically allowed to be in an indeterminate state). so we also have to figure out how to disable all interrupts before doing anything, and then only enable those we care about.

7.2 Interrupt pending.

An interrupt vector module has NOT been implemented. To still have adequate interrupt processing the interrupt pending bits are organized as follows:



confusingly, when you get an interrupt on the R/pi it could have been from a GPU source, even though you didn't enable these --- these interrupts are not for you, and should be ignored.

There are three interrupt pending registers.

One basic pending register and two GPU pending registers.

Basic pending register.

The basic pending register has interrupt pending bits for the ARM specific interrupts .

To speed up the interrupt processing it also has a number of selected GPU interrupts which are deemed most likely to be required in ARM drivers.

Further there are two special GPU pending bits which tell if any of the two other pending registers has bits set, one bit if a GPU interrupt 0-31 is pending, a second bit if a GPU interrupt 32-63 is pending. The 'selected GPU interrupts' on the basic pending registers are NOT taken into account for these two status bits. So the two pending 0,1 status bits tell you that 'there are more interrupt which you have not seen yet'.

GPU pending registers.

There are two GPU pending registers with one bit per GPU interrupt source.

7.3 Fast Interrupt (FIQ).

The ARM also supports a Fast Interrupt (FIQ). One interrupt sources can be selected to be connected to the ARM FIQ input. There is also one FIQ enable. An interrupt which is selected as FIQ should have its normal interrupt enable bit cleared. Otherwise an normal and a FIQ interrupt will be fired at the same time. Not a good idea!

7.4 Interrupt priority.

There is no priority for any interrupt. If one interrupt is much more important then all others it can be routed to the FIQ. Any remaining interrupts have to be processed by polling the pending

registers. It is up to the ARM software to device a strategy. e.g. First start looking for specific pending bits or process them all shifting one bit at a time.

As interrupt may arrive whilst this process is ongoing the usual care for any 'race-condition critical' code must be taken. The following ARM assembly code has been proven to work:

```
  .macro get_irqnr_preamble, base, tmp
  ldr  \base, =IO_ADDRESS(ARMCTRL_IC_BASE)
  .endm

  .macro get_irqnr_and_base, irqnr, irqstat, base, tmp
  ldr  \irqstat, [\base, #(ARM_IRQ_PEND0 - ARMCTRL_IC_BASE)] @ get masked status
  mov  \irqnr, #(ARM_IRQ0_BASE + 31)
  and  \tmp, \irqstat, #0x300           @ save bits 8 and 9
  bics \irqstat, \irqstat, #0x300       @ clear bits 8 and 9, and test
  bne  1010f

  tst  \tmp, #0x100
  ldrne \irqstat, [\base, #(ARM_IRQ_PEND1 - ARMCTRL_IC_BASE)]
  movne \irqnr, #(ARM_IRQ1_BASE + 31)
  @ Mask out the interrupts also present in PEND0 - see SW-5809
  bicne \irqstat, #((1<<7) | (1<<9) | (1<<10))
  bicne \irqstat, #((1<<18) | (1<<19))
  bne  1010f

  tst  \tmp, #0x200                                                  ignore
  ldrne \irqstat, [\base, #(ARM_IRQ_PEND2 - ARMCTRL_IC_BASE)]
  movne \irqnr, #(ARM_IRQ2_BASE + 31)
  @ Mask out the interrupts also present in PEND0 - see SW-5809
  bicne \irqstat, #((1<<21) | (1<<22) | (1<<23) | (1<<24) | (1<<25))
  bicne \irqstat, #((1<<30))
  beq 1020f

1010:
  @ For non-zero x, LSB(x) = 31 - CLZ(x^(x-1))
  @ N.B. CLZ is an ARM5 instruction.
  sub  \tmp, \irqstat, #1
  eor  \irqstat, \irqstat, \tmp
  clz  \tmp, \irqstat
  sub  \irqnr, \tmp

1020: @ EQ will be set if no irqs pending
```

.endm

7.5 Registers

these are what we care about for timer interrupts

The base address for the ARM interrupt register is 0x7E00B000.

base is 0x2000B000, so registers start at 0x2000B200

Registers overview:

Address offset[7]	Name	Notes
0x200	IRQ basic pending	
0x204	IRQ pending 1	
0x208	IRQ pending 2	
0x20C	FIQ control	
0x210	Enable IRQs 1	
0x214	Enable IRQs 2	
0x218	Enable Basic IRQs	
0x21C	Disable IRQs 1	
0x220	Disable IRQs 2	
0x224	Disable Basic IRQs	

these are what we need to disable at the start to put the r/pi in a known, clean state.

The following is a table which lists all interrupts which can come from the peripherals which can be handled by the ARM.

[7] This is the offset which needs to be added to the base address to get the full hardware address.

to catch these, you need to enable "basic interrupt 2" (p117) since they are > 32. having multiple means you can bind different events to different interrupt lines.

ARM peripherals interrupts table.

#	IRQ 0-15	#	IRQ 16-31	#	IRQ 32-47	#	IRQ 48-63
0		16		32		48	smi
1		17		33		49	gpio_int[0]
2		18		34		50	gpio_int[1]
3		19		35		51	gpio_int[2]
4		20		36		52	gpio_int[3]
5		21		37		53	i2c_int
6		22		38		54	spi_int
7		23		39		55	pcm_int
8		24		40		56	
9		25		41		57	uart_int
10		26		42		58	
11		27		43	i2c_spi_slv_int	59	
12		28		44		60	
13		29	Aux int	45	pwa0	61	
14		30		46	pwa1	62	
15		31		47		63	

later!

The table above has many empty entries. These should not be enabled as they will interfere with the GPU operation.

confusing table, since not attached to anything ("did i miss something?") is a reverse sort of the "IRQ pend base" register on the next page.

ARM peripherals interrupts table.

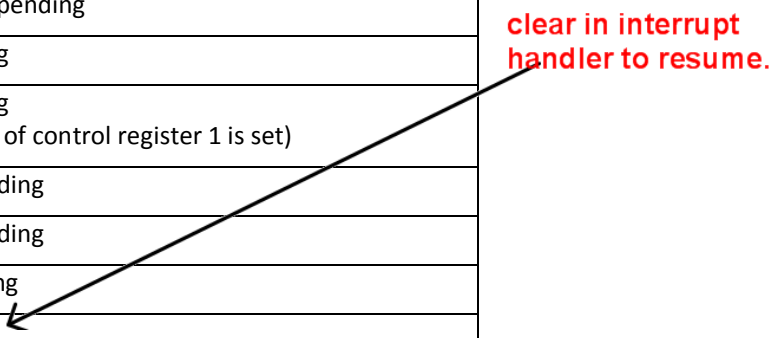
0	ARM Timer
1	ARM Mailbox
2	ARM Doorbell 0
3	ARM Doorbell 1
4	GPU0 halted (Or GPU1 halted if bit 10 of control register 1 is set)
5	GPU1 halted
6	Illegal access type 1
7	Illegal access type 0

Basic pending register.

The basic pending register shows which interrupt are pending. To speed up interrupts processing, a number of 'normal' interrupt status bits have been added to this register. This makes the *'IRQ pending base'* register different from the other *'base'* interrupt registers

Name: IRQ pend base	Address: 0x200	Reset: 0x000
Bit(s)	**R/W**	**Function**
31:21	-	<unused>
20	R	GPU IRQ 62

Name: IRQ pend base		Address: 0x200	Reset: 0x000
19	R	GPU IRQ 57	
18	R	GPU IRQ 56	
17	R	GPU IRQ 55	
16	R	GPU IRQ 54	
15	R	GPU IRQ 53	
14	R	GPU IRQ 19	
13	R	GPU IRQ 18	
12	R	GPU IRQ 10	
11	R	GPU IRQ 9	
10	R	GPU IRQ 7	
9	R	One or more bits set in pending register 2	
8	R	One or more bits set in pending register 1	
7	R	Illegal access type 0 IRQ pending	
6	R	Illegal access type 1 IRQ pending	
5	R	GPU1 halted IRQ pending	
4	R	GPU0 halted IRQ pending (Or GPU1 halted if bit 10 of control register 1 is set)	
3	R	ARM Doorbell 1 IRQ pending	
2	R	ARM Doorbell 0 IRQ pending	
1	R	ARM Mailbox IRQ pending	
0	R	ARM Timer IRQ pending	



GPU IRQ x (10,11..20)

These bits are direct interrupts from the GPU. They have been selected as interrupts which are most likely to be useful to the ARM. The GPU interrupt selected are 7, 9, 10, 18, 19, 53,54,55,56,57,62. For details see the *GPU interrupts table*.

Bits set in pending registers (8,9)

These bits indicates if there are bits set in the pending 1/2 registers. The pending 1/2 registers hold ALL interrupts 0..63 from the GPU side. Some of these 64 interrupts are also connected to the basic pending register. Any bit set in pending register 1/2 which is NOT connected to the basic pending register causes bit 8 or 9 to set. Status bits 8 and 9 should be seen as "There are some interrupts pending which you don't know about. They are in pending register 1 /2."

Illegal access type-0 IRQ (7)

This bit indicate that the address/access error line from the ARM processor has generated an interrupt. That signal is asserted when either an address bit 31 or 30 was high or when an access was

seen on the ARM Peripheral bus. The status of that signal can be read from Error/HALT status register bit 2.

Illegal access type-1 IRQ (6)

This bit indicates that an address/access error is seen in the ARM control has generated an interrupt. That can either be an address bit 29..26 was high or when a burst access was seen on the GPU Peripheral bus. The status of that signal can be read from Error/HALT status register bits 0 and 1.

GPU-1 halted IRQ (5)

This bit indicate that the GPU-1 halted status bit has generated an interrupt. The status of that signal can be read from Error/HALT status register bits 4.

GPU-0 (or any GPU) halted IRQ (4)

This bit indicate that the GPU-0 halted status bit has generated an interrupt. The status of that signal can be read from Error/HALT status register bits 3.

In order to allow a fast interrupt (FIQ) routine to cope with GPU 0 OR GPU-1 there is a bit in control register 1 which, if set will also route a GPU-1 halted status on this bit.

Standard peripheral IRQs (0,1,2,3)

These bits indicate if an interrupt is pending for one of the ARM control peripherals.

can check these to see how often random GPU stuff gets triggered.

GPU pending 1 register.

Name: IRQ pend base		Address: 0x204	Reset: 0x000
Bit(s)	**R/W**	**Function**	
31:0	R	IRQ pending source 31:0 (See IRQ table above)	

This register holds ALL interrupts 0..31 from the GPU side. Some of these interrupts are also connected to the basic pending register. Any interrupt status bit in here which is NOT connected to the basic pending will also cause bit 8 of the basic pending register to be set. That is all bits except 7, 9, 10, 18, 19.

GPU pending 2 register.

Name: IRQ pend base		Address: 0x208	Reset: 0x000
Bit(s)	**R/W**	**Function**	
31:0	R	IRQ pending source 63:32 (See IRQ table above)	

This register holds ALL interrupts 32..63 from the GPU side. Some of these interrupts are also connected to the basic pending register. Any interrupt status bit in here which is NOT connected to the basic pending will also cause bit 9 of the basic pending register to be set. That is all bits except . register bits 21..25, 30 (Interrupts 53..57,62).

FIQ register.

The FIQ register control which interrupt source can generate a FIQ to the ARM. Only a single interrupt can be selected.

Name: FIQ		Address: 0x20C	Reset: 0x000
Bit(s)	**R/W**	**Function**	
31:8	R	<unused>	
7	R	FIQ enable. Set this bit to 1 to enable FIQ generation. If set to 0 bits 6:0 are don't care.	
6:0	R/W	Select FIQ Source	

FIQ Source.

The FIQ source values 0-63 correspond to the GPU interrupt table. (See above)

The following values can be used to route ARM specific interrupts to the FIQ vector/routine:

FIQ index	Source
0-63	GPU Interrupts (See GPU IRQ table)
64	ARM Timer interrupt
65	ARM Mailbox interrupt
66	ARM Doorbell 0 interrupt
67	ARM Doorbell 1 interrupt
68	GPU0 Halted interrupt (Or GPU1)
69	GPU1 Halted interrupt
70	Illegal access type-1 interrupt
71	Illegal access type-0 interrupt
72-127	Do Not Use

Interrupt enable register 1.

Name: IRQ enable 1		Address: 0x210	Reset: 0x000
Bit(s)	**R/W**	**Function**	
31:0	R/Wbs	Set to enable IRQ source 31:0 (See IRQ table above)	

Writing a 1 to a bit will set the corresponding IRQ enable bit. All other IRQ enable bits are unaffected. Only bits which are enabled can be seen in the interrupt pending registers. There is no provision here to see if there are interrupts which are pending but not enabled.

Interrupt enable register 2.

Name: IRQ enable 2		Address: 0x214	Reset: 0x000
Bit(s)	**R/W**	**Function**	
31:0	R/Wbs	Set to enable IRQ source 63:32 (See IRQ table above)	

Writing a 1 to a bit will set the corresponding IRQ enable bit. All other IRQ enable bits are unaffected. Only bits which are enabled can be seen in the interrupt pending registers. There is no provision here to see if there are interrupts which are pending but not enabled.

Base Interrupt enable register.

Name: IRQ enable 3		Address: 0x218	Reset: 0x000
Bit(s)	**R/W**	**Function**	
31:8	R/Wbs	<Unused>	
7	R/Wbs	Set to enable Access error type -0 IRQ	
6	R/Wbs	Set to enable Access error type -1 IRQ	
5	R/Wbs	Set to enable GPU 1 Halted IRQ	
4	R/Wbs	Set to enable GPU 0 Halted IRQ	
3	R/Wbs	Set to enable ARM Doorbell 1 IRQ	
2	R/Wbs	Set to enable ARM Doorbell 0 IRQ	
1	R/Wbs	Set to enable ARM Mailbox IRQ	
0	R/Wbs	Set to enable ARM Timer IRQ	

Writing a 1 to a bit will set the corresponding IRQ enable bit. All other IRQ enable bits are unaffected. Again only bits which are enabled can be seen in the basic pending register. There is no provision here to see if there are interrupts which are pending but not enabled.

Interrupt disable register 1.

Name: IRQ disable 1		Address: 0x21C	Reset: 0x000
Bit(s)	**R/W**	**Function**	
31:0	R/Wbc	Set to disable IRQ source 31:0 (See IRQ table above)	

Writing a 1 to a bit will clear the corresponding IRQ enable bit. All other IRQ enable bits are unaffected.

Interrupt disable register 2.

Name: IRQ disable 2		Address: 0x220	Reset: 0x000
Bit(s)	**R/W**	**Function**	
31:0	R/Wbc	Set to disable IRQ source 63:32 (See IRQ table above)	

Writing a 1 to a bit will clear the corresponding IRQ enable bit. All other IRQ enable bits are unaffected.

Base disable register.

Name: IRQ disable 3		Address: 0x224	Reset: 0x000
Bit(s)	**R/W**	**Function**	
31:8	-	<Unused>	
7	R/Wbc	Set to disable Access error type -0 IRQ	
6	R/Wbc	Set to disable Access error type -1 IRQ	
5	R/Wbc	Set to disable GPU 1 Halted IRQ	
4	R/Wbc	Set to disable GPU 0 Halted IRQ	
3	R/Wbc	Set to disable ARM Doorbell 1 IRQ	
2	R/Wbc	Set to disable ARM Doorbell 0 IRQ	
1	R/Wbc	Set to disable ARM Mailbox IRQ	
0	R/Wbc	Set to disable ARM Timer IRQ	

Writing a 1 to a bit will clear the corresponding IRQ enable bit. All other IRQ enable bits are unaffected.

make sure you write a 1, NOT 0.