QA7

ARM Quad A7 core

Gert van Loo, 18 August 2014

Date	Auth.	Rev	Changes
28-06-2013	GVL	1.0	Initial version
19-07-2013	GVL	1.1	
19-07-2013	GVL	1.2	
24-07-2013	GVL	2.0	
25-07-2013	GVL	2.1	
2-08-2013	GVL	2.2	
21-08-2013	GVL	3.0	
12-09-2013	GVL	3.1	
11-10-2013	GVL	3.2	
30-06-2014	GVL	3.3	
18-08-2014	GVL	3.4	

Contents

1 Introduction

The document describes the implementation of the Quad-A7 core on top of the 2708 project resulting in a new (BCM2836) product.

2 System overview.

The ARM address map is split according to the following table:

Address	Device(s)
0x0000_0000 .. 0x3FFF_FFFF	GPU access
0x3E00_0000 .. 0x3FFF_FFFF	GPU peripheral access[1]
0x4000_0000 .. 0xFFFF_FFFF	Local peripherals
0x4000_0000 .. 0x4001_FFFF	ARM timer, IRQs, mailboxes
0x4002_0000 .. 0x4002_FFFF	Debug ROM
0x4003_0000 .. 0x4003_FFFF	DAP
0x4004_0000 .. 0xFFFF_FFFF	<Unused>

3 ARM control

The ARM control logic is a module which performs the following functions:

- Provide a 64-bit Timer
- Route the various interrupts to the cores
- Route the GPU interrupts to the core
- Provide mailboxes between the processors
- Provide extra interrupt timer

3.1 64-bit Timer

The A7-core requires a 64-bit timing input signal which is used to implement the four timers internally to each processor core. There is only one 64-bit timer signal going to all four cores. This means that any changes to the timer will affect all cores.

3.1.1 Timer clock

I have not found any information on how fast this timer should run. It seems common to run it of the processor clock. However that would not give a reliably timing signal when the frequency of the processor is variable. Therefore the source of the timer can come from either the external crystal or from a CPU related clock.

To give maximum flexibility to the timer speed there is a 32-bit pre-scaler. This prescaler can provide integer as well as fractional division ratios.

$$divider = \frac{2\text{^}31}{prescaler\_value}$$

(*prescaler_value*<=2^31)

Thus setting the prescaler to 0x8000_0000 gives a divider ratio of 1. Setting the prescaler to 0 will stop the timer.

To get a divider ratio of 19.2 use: 2^31/19.2 = 0x06AA_AAAB. The value is rounded upwards and introduces an error of 8.9E-9 which is much lower than any ordinary crystal oscillator produces.

Do not use timer values >2^31 (2147483648)

To simplify the design the timer is running from the local peripheral (APB) clock. That clock is half the speed of the ARM thus the timer cannot represent the ARM clock frequency exactly. To give a nearest number you can increment the timer by 2 thus giving a value close to the ARM clock frequency. Beware that if the timer increment is set to 2 you will only get either all even or all odd values (Depending on if the initial value you write to it is even or odd).

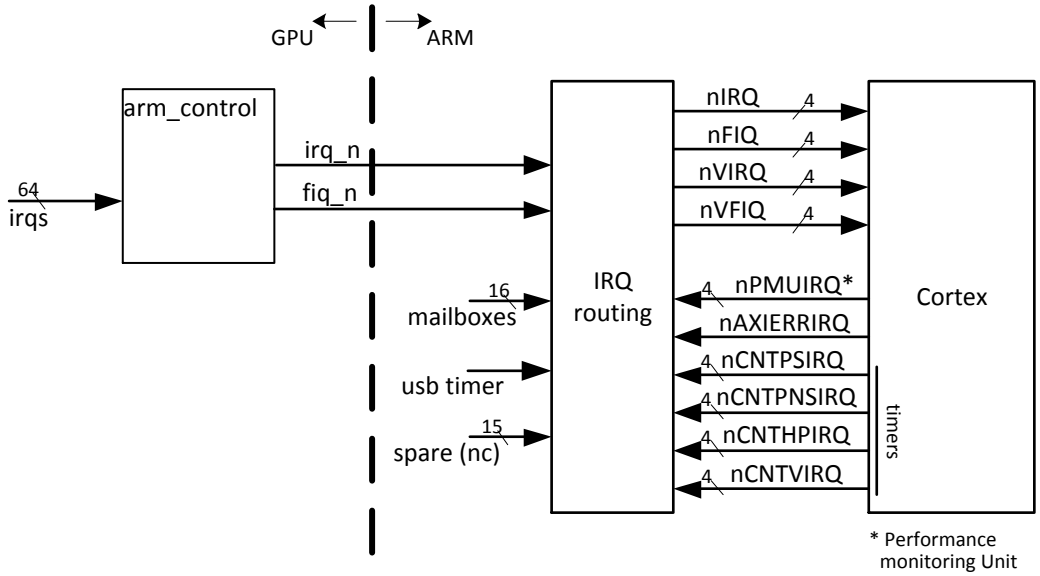
3.1.2 64-bit timer read/write

The ARM has 32-bit registers and the main timer has a 64-bits wide changing value. There is the potential for reading the wrong value if the timer value changes between reading the LS and MS 32 bits. Therefore it has dedicated logic to give trouble free access to the 64-bit value.

When reading the 64-bit timer value the user must always read the LS 32 bits first. At the same time the LS-32 bits are read, internal a copy is made of the MS-32 bits into a timer-read-hold register. When reading the timer MS-32 bits actually the timer-read-hold register is read.

When writing the 64-bit timer value the user must always write the LS 32 bits first. The LS 32-bits are stored in a timer-write-hold register. When the user writes the MS 32-bit, the stored LS bits are written as well.
The timer pre-scaler register is set to zero when the timer value (MS) is written.

3.2 Interrupt routing



There are numerous interrupts which need to be routed. The interrupt routing logic has the following input signals:

- Core related interrupts
- Core un-related interrupts

3.2.1 Core related interrupts

Core related interrupts are interrupts which are bound to one specific core. Most of these are interrupts generated by that core itself like the four timer interrupt signals. Additionally each core has four mailboxes assigned to it. These are the core related interrupts:

- Four timer interrupts (64-bit timer)
- One performance monitor interrupts
- Four Mailbox interrupts

For each of these interrupts you can only choose to send them to either the IRQ pin or to the FIQ pin of one core. (or not pass at all) The following table shows the truth table:

FIQ bit	IRQ Bit	Destination
0	0	None (disabled)
0	1	IRQ
1	0	FIQ
1	1	FIQ

The mailbox interrupts do not have a separate interrupt enable/disable bit. The routing bits have to be used for that. Unfortunately this enables or disables all 32 bits of a mailbox.

After a reset all bits are zero so none of the interrupts is enabled.

3.2.2 Core un-related interrupts

Core unrelated interrupts are interrupts which can be send to any of the four cores and to either the interrupt or the fast-interrupt of that core. These are the core unrelated interrupts:

- GPU IRQ (As generated by the ARM Control logic)
- GPU FIQ (As generated by the ARM Control logic)
- Local timer interrupt
- AXI error
- (unused: Fifteen local peripheral interrupts)

For each of these interrupts you can choose to send them to either the IRQ pin or to the FIQ pin of any of the four cores. The following table shows the truth table:

Routing code	Destination
000	IRQ Core 0
001	IRQ Core 1
010	IRQ Core 2
011	IRQ Core 3
100	FIQ Core 0
101	FIQ Core 1
110	FIQ Core 2
111	FIQ Core 3

Note that these interrupts do not have a 'disable' code. They are expected to have an enable/disable bit at the source where they are generated. After a reset all bits are zero thus all interrupts are send to the IRQ of core 0.

3.3 Mailboxes

A mailbox is a 32-bit wide write-bits-high-to-set and write-bits-high-to-clear register. The write-set register addresses are write only. The write-clear register addresses can be read as well. A mailbox generates and interrupt as long as it is non-zero.

The system has sixteen mailboxes, four for each core. The system has no doorbells. You can use the mailboxes as doorbells instead.

Mailbox 0-3 are dedicated to core 0, mailbox 4-7 are dedicated to core 1 etc. Each mailbox has two interrupt routing bits as described in *3.2.1 Core related interrupts*.

There is no difference between any of the four mailboxes assigned to a core. It is left to the programmer to decide how to use them.

4 Registers

Address	Register
0x4000_0000	Control register
0x4000_0004	<unused>
0x4000_0008	Core timer prescaler
0x4000_000C	GPU interrupts routing
0x4000_0010	Performance Monitor Interrupts routing-set
0x4000_0014	Performance Monitor Interrupts routing-clear
0x4000_0018	<unused>
0x4000_001C	Core timer access LS 32 bits
0x4000_0020	Core timer access MS 32 bits
0x4000_0024	Local Interrupt 0 [1-7] routing
~~0x4000_0028~~	~~Local Interrupts 8-15 routing~~
0x4000_002C	Axi outstanding counters
0x4000_0030	Axi outstanding IRQ
0x4000_0034	Local timer control & status
0x4000_0038	Local timer write flags
0x4000_003C	<unused>
0x4000_0040	Core0 timers Interrupt control
0x4000_0044	Core1 timers Interrupt control
0x4000_0048	Core2 timers Interrupt control
0x4000_004C	Core3 timers Interrupt control
0x4000_0050	Core0 Mailboxes Interrupt control
0x4000_0054	Core1 Mailboxes Interrupt control
0x4000_0058	Core2 Mailboxes Interrupt control
0x4000_005C	Core3 Mailboxes Interrupt control
0x4000_0060	Core0 IRQ Source
0x4000_0064	Core1 IRQ Source
0x4000_0068	Core2 IRQ Source
0x4000_006C	Core3 IRQ Source
0x4000_0070	Core0 FIQ Source
0x4000_0074	Core1 FIQ Source
0x4000_0078	Core2 FIQ Source
0x4000_007C	Core3 FIQ Source
0x4000_0080	Core 0 Mailbox 0 write-set (WO)
0x4000_0084	Core 0 Mailbox 1 write-set (WO)
0x4000_0088	Core 0 Mailbox 2 write-set (WO)
0x4000_008C	Core 0 Mailbox 3 write-set (WO)
0x4000_0090	Core 1 Mailbox 0 write-set (WO)
0x4000_0094	Core 1 Mailbox 1 write-set (WO)
0x4000_0098	Core 1 Mailbox 2 write-set (WO)
0x4000_009C	Core 1 Mailbox 3 write-set (WO)
0x4000_00A0	Core 2 Mailbox 0 write-set (WO)
0x4000_00A4	Core 2 Mailbox 1 write-set (WO)
0x4000_00A8	Core 2 Mailbox 2 write-set (WO)
0x4000_00AC	Core 2 Mailbox 3 write-set (WO)
0x4000_00B0	Core 3 Mailbox 0 write-set (WO)
0x4000_00B4	Core 3 Mailbox 1 write-set (WO)

Address	Register
0x4000_00B8	Core 3 Mailbox 2 write-set (WO)
0x4000_00BC	Core 3 Mailbox 3 write-set (WO)
0x4000_00C0	Core 0 Mailbox 0 read & write-high-to-clear
0x4000_00C4	Core 0 Mailbox 1 read & write-high-to-clear
0x4000_00C8	Core 0 Mailbox 2 read & write-high-to-clear
0x4000_00CC	Core 0 Mailbox 3 read & write-high-to-clear
0x4000_00D0	Core 1 Mailbox 0 read & write-high-to-clear
0x4000_00D4	Core 1 Mailbox 1 read & write-high-to-clear
0x4000_00D8	Core 1 Mailbox 2 read & write-high-to-clear
0x4000_00DC	Core 1 Mailbox 3 read & write-high-to-clear
0x4000_00E0	Core 2 Mailbox 0 read & write-high-to-clear
0x4000_00E4	Core 2 Mailbox 1 read & write-high-to-clear
0x4000_00E8	Core 2 Mailbox 2 read & write-high-to-clear
0x4000_00EC	Core 2 Mailbox 3 read & write-high-to-clear
0x4000_00F0	Core 3 Mailbox 0 read & write-high-to-clear
0x4000_00F4	Core 3 Mailbox 1 read & write-high-to-clear
0x4000_00F8	Core 3 Mailbox 2 read & write-high-to-clear
0x4000_00FC	Core 3 Mailbox 3 read & write-high-to-clear

4.1 Write-set / Write-clear registers

To allow atomic operations a number of register are split into a write-set register and a write-clear register.

A **write-set register** allows you to set additional bits **high**. Bits which were already high are not affected. You set a bit high by writing the value '1' to it. Everywhere you write a '0' bit the original register contents remains unchanged.

Old bit value	Write bit value	Result bit value
0	0	0
0	1	1
1	0	1
1	1	1

Thus writing 0xFC060014 to a register containing 0x30840008 gives 0xFC86001C.

A **write-clear register** allows you to set additional bits **low**. Bits which were already low are not affected. You set a bit **low** by writing the value '1' to it. Everywhere you write a '0' bit the original register contents remains unchanged. Beware that you write a **one** to get a **zero!**

Old bit value	Write bit value	Result bit value
0	0	0
0	1	0
1	0	1
1	1	0

Thus writing 0xFC060014 to a register containing 0x30840008 gives 0x00800008.

This system is particular useful when dealing with asynchronous events like interrupts. Interrupt status bits can appear at any time. Thus it is possible that the CPU reads the interrupt-pending register and sees bit 4 high. The next clock cycle an interrupt comes in and sets additionally bit 2 high but the processor is

no aware of this. The processor handles the interrupt bit 4. To clear the interrupt, the processor writes back the value it has read. (which has bit 4 set) and thus it guarantees that only bit 4 gets cleared and any missed, non -processed, interrupt bits remain pending.

4.2 Control register

The control register is currently only used to control the 64-bit core timer.

Address: 0x4000_0000 **Reset: 0x0000_0000**	
Bits	**Description**
31-10	<Reserved>
9	1 : 64-bit Core timer increments by 2 0 : 64-bit Core timer increments by1
8	Core timer clock source 1 : 64-bit Core timer runs from the APB clock 0 : 64-bit Core timer runs from the Crystal clock
7-0	<Reserved>

Bit 8: Core timer clock source
This bit controls what the source clock is of the 64-bit Core timer . Actually it selects the source clock of the Core timer prescaler but that amounts to the same end-result.

- If set the 64-bit core timer pre-scaler is running of the APB clock.
- If clear the 64-bit core timer pre-scaler is running of the Crystal clock.

Note that the APB clock is running at half the speed of the ARM clock. Thus the pre-scaler is only changing every second CPU clock cycle.

Bit 9: Timer increment
This bit controls the step size of the 64-bit core timer . This may be important if you want the core timer to accurate represent the number of CPU cycles.
The core timer pre-scaler is running of the APB clock. As the APB clock is running at half the speed of the ARM clock, you cannot get a timer value equal to the ARM clock cycles, even if the pre-scaler is set to divide-by-one. This bit provides a means of getting close to the actual number of CPU cycles.

- If set the 64-bit core timer is incremented by 2.
- If clear the 64-bit core timer is incremented by 1.

This will still not get you the exact number of CPU cycles but will get you close to plus/minus one. Beware that if the core timer increment is set to 2 you will only get either all even or all odd values (Depending on if the initial value you write to it is even or odd).

4.3 Core timer register

There is a core timer pre-scaler registers and two timer read/write registers.

~~**Address: 0x4000_0004**~~ **<Deprecated>** ~~**Reset: 0x0000_0000**~~	
~~**Bits**~~	~~**Description**~~
~~31-0~~	~~Timer prescaler subtract~~

This register is deprecated

Address: 0x4000_008 Core timer pre-scaler. **Reset: 0x0000_0000**	
Bits	**Description**
31-0	Core timer prescaler

timer_frequency = (2^{31}/prescaler) * input frequency
(Pre-scaler <= 2^{31})

For details see §3.1.1 Timer clock.

Address: 0x4000_01C: Core timer read: LS 32 bits., Write: LS-32 holding register **Reset: 0x0000_0000**	
Bits	**Description**
31-0	64-bit core timer read/write, LS 32 bits When reading returns the current 32 LS bit of the 64 timer and triggers storing a copy of the MS 32 bits. When writing: stores a copy of the 32 bits written. That copy is transferred to the timer when the MS 32 bits are written.

Address: 0x4000_020: Core timer read: Stored MS 32 bits register, Write: MS-32 bits **Reset: 0x0000_0000**	
Bits	**Description**
31-0	64-bit core timer read/write, MS 32 bits When reading returns the status of the core timer-read-hold register. That register is loaded when the user does a read of the LS-32 timer bits. There is little sense in reading this register without first doing a read from the LS-32 bit register. When writing the value is written to the timer, as well as the value previously written to the LS-32 write-holding bit register. There is little sense in writing this register without first doing a write to the LS-32 bit register.

4.4 GPU interrupts routing

The GPU interrupt routing register controls where the IRQ and FIQ of the GPU are routed to.

Address: 0x4000_000C GPU interrupt routing **Reset: 0x0000_0000**	
Bits	**Description**
31-4	<Reserved>
3:2	GPU FIQ routing: 00 : GPU FIQ goes to FIQ input of core 0 01 : GPU FIQ goes to FIQ input of core 1 10 : GPU FIQ goes to FIQ input of core 2 11 : GPU FIQ goes to FIQ input of core 3
1:0	GPU IRQ routing: 00 : GPU IRQ goes to IRQ input of core 0 01 : GPU IRQ goes to IRQ input of core 1 10 : GPU IRQ goes to IRQ input of core 2 11 : GPU IRQ goes to IRQ input of core 3

The IRQ /FIQ can be connected to one processor core only. This also means that there is only one possible GPU-IRQ/GPU-FIQ interrupt outstanding bit.

4.5 Performance monitors interrupts

The performance monitors have two registers:

- PMU interrupt routing write-set.
- PMU interrupt routing write-clear.

The performance interrupt routing registers control where the nPMUIRQ signal are routed to. There are two registers: One is a write-set and the other is a write-clear register (See *4.1 Write-set / Write-clear registers*). The PMU interrupts are dedicated routed. Thus a PMU from a certain core can only generate interrupts to that core, not to any of the other cores. The register allow you to enable or disable an IRQ or FIQ interrupt. They cannot clear an pending interrupts. For that and other details of the PMUs read the Cortex-A7-coprocessor description.

Address: 0x4000_0010 PMU interrupt routing write-set **Address: 0x4000_0014 PMU interrupt routing write-clear** **Reset: 0x0000_0000**	
Bits	**Description**
31-8	<Reserved>
7	nPMUIRQ[3] FIQ control. If set, this bit overrides the IRQ bit (3). 0 : FIQ disabled 1 : FIQ Enabled
6	nPMUIRQ[2] FIQ control. If set, this bit overrides the IRQ bit (2). 0 : FIQ disabled 1 : FIQ Enabled
5	nPMUIRQ[1] FIQ control. If set, this bit overrides the IRQ bit (1). 0 : FIQ disabled 1 : FIQ Enabled
4	nPMUIRQ[0] FIQ control. If set, this bit overrides the IRQ bit (0). 0 : FIQ disabled 1 : FIQ Enabled
3	nPMUIRQ[3] IRQ control. This bit is only valid if bit 7 is clear otherwise it is ignored. 0 : IRQ disabled 1 : IRQ Enabled
2	nPMUIRQ[2] IRQ control. This bit is only valid if bit 6 is clear otherwise it is ignored. 0 : IRQ disabled 1 : IRQ Enabled
1	nPMUIRQ[1] IRQ control. This bit is only valid if bit 5 is clear otherwise it is ignored. 0 : IRQ disabled 1 : IRQ Enabled
0	nPMUIRQ[0] IRQ control. This bit is only valid if bit 4 is clear otherwise it is ignored. 0 : IRQ disabled 1 : IRQ Enabled

4.6 Core timers interrupts

There are four core timer control and four core timer status registers. The registers allow you to enable or disable an IRQ or FIQ interrupt. They cannot clear an pending interrupts. For that and other details of the timers, read the Cortex-A7-coprocessor description.

Address: 0x4000_0040 Core 0 Timers interrupt control **Address: 0x4000_0044 Core 1 Timers interrupt control** **Address: 0x4000_0048 Core 2 Timers interrupt control** **Address: 0x4000_004C Core 3 Timers interrupt control** **Reset: 0x0000_0000**	
Bits	**Description**
31-8	<Reserved>
7	nCNTVIRQ FIQ control. If set, this bit overrides the IRQ bit (3). 0 : FIQ disabled 1 : FIQ Enabled
6	nCNTHPIRQ FIQ control. If set, this bit overrides the IRQ bit (2). 0 : FIQ disabled 1 : FIQ Enabled
5	nCNTPNSIRQ FIQ control. If set, this bit overrides the IRQ bit (1). 0 : FIQ disabled 1 : FIQ Enabled
4	nCNTPSIRQ FIQ control. If set, this bit overrides the IRQ bit (0). 0 : FIQ disabled 1 : FIQ Enabled
3	nCNTVIRQ IRQ control. This bit is only valid if bit 7 is clear otherwise it is ignored. 0 : IRQ disabled 1 : IRQ Enabled
2	nCNTHPIRQ IRQ control. This bit is only valid if bit 6 is clear otherwise it is ignored. 0 : IRQ disabled 1 : IRQ Enabled
1	nCNTPNSIRQ IRQ control. This bit is only valid if bit 5 is clear otherwise it is ignored. 0 : IRQ disabled 1 : IRQ Enabled
0	nCNTPSIRQ IRQ control. This bit is only valid if bit 4 is clear otherwise it is ignored. 0 : IRQ disabled 1 : IRQ Enabled

4.7 Core Mailboxes interrupts

There are four Mailbox interrupt control registers.

Address: 0x4000_0050 Core0 Mailboxes interrupt control **Address: 0x4000_0054 Core1 Mailboxes interrupt control** **Address: 0x4000_0058 Core2 Mailboxes interrupt control** **Address: 0x4000_005C Core3 Mailboxes interrupt control** **Reset: 0x0000_0000**	
Bits	**Description**
31-8	<Reserved>
7	Mailbox-3 FIQ control. If set, this bit overrides the IRQ bit (3). 0 : FIQ disabled 1 : FIQ Enabled
6	Mailbox-2 FIQ control. If set, this bit overrides the IRQ bit (2). 0 : FIQ disabled 1 : FIQ Enabled
5	Mailbox-1 FIQ control. If set, this bit overrides the IRQ bit (1). 0 : FIQ disabled 1 : FIQ Enabled
4	Mailbox-0 FIQ control. If set, this bit overrides the IRQ bit (0). 0 : FIQ disabled 1 : FIQ Enabled
3	Mailbox-3 IRQ control. This bit is only valid if bit 7 is clear otherwise it is ignored. 0 : IRQ disabled 1 : IRQ Enabled
2	Mailbox-2 IRQ control. This bit is only valid if bit 6 is clear otherwise it is ignored. 0 : IRQ disabled 1 : IRQ Enabled
1	Mailbox-1 IRQ control. This bit is only valid if bit 4 is clear otherwise it is ignored. 0 : IRQ disabled 1 : IRQ Enabled
0	Mailbox-0 IRQ control. This bit is only valid if bit 4 is clear otherwise it is ignored. 0 : IRQ disabled 1 : IRQ Enabled

4.8 Core Mailboxes

The system has sixteen mailboxes. They are accessed through 32 register.

- There are 16 write-to-set-bits registers
  These registers are write-only
- There are 16 write-to-clear-bits registers
  These register can also be read.

Address	Mailbox-set
0x4000_0080	Core 0 Mailbox 0 Set
0x4000_0084	Core 0 Mailbox 1 Set
0x4000_0088	Core 0 Mailbox 2 Set
0x4000_008C	Core 0 Mailbox 3 Set
0x4000_0090	Core 1 Mailbox 0 Set
0x4000_0094	Core 1 Mailbox 1 Set
0x4000_0098	Core 1 Mailbox 2 Set
0x4000_009C	Core 1 Mailbox 3 Set
0x4000_00A0	Core 2 Mailbox 0 Set
0x4000_00A4	Core 2 Mailbox 1 Set
0x4000_00A8	Core 2 Mailbox 2 Set
0x4000_00AC	Core 2 Mailbox 3 Set
0x4000_00B0	Core 3 Mailbox 0 Set
0x4000_00B4	Core 3 Mailbox 1 Set
0x4000_00B8	Core 3 Mailbox 2 Set
0x4000_00BC	Core 3 Mailbox 3 Set

Address	Mailbox-clear
0x4000_00C0	Core 0 Mailbox 0 Rd/Clr
0x4000_00C4	Core 0 Mailbox 1 Rd/Clr
0x4000_00C8	Core 0 Mailbox 2 Rd/Clr
0x4000_00CC	Core 0 Mailbox 3 Rd/Clr
0x4000_00D0	Core 1 Mailbox 0 Rd/Clr
0x4000_00D4	Core 1 Mailbox 1 Rd/Clr
0x4000_00D8	Core 1 Mailbox 2 Rd/Clr
0x4000_00DC	Core 1 Mailbox 3 Rd/Clr
0x4000_00E0	Core 2 Mailbox 0 Rd/Clr
0x4000_00E4	Core 2 Mailbox 1 Rd/Clr
0x4000_00E8	Core 2 Mailbox 2 Rd/Clr
0x4000_00EC	Core 2 Mailbox 3 Rd/Clr
0x4000_00F0	Core 3 Mailbox 0 Rd/Clr
0x4000_00F4	Core 3 Mailbox 1 Rd/Clr
0x4000_00F8	Core 3 Mailbox 2 Rd/Clr
0x4000_00FC	Core 3 Mailbox 3 Rd/Clr

4.9 Axi outstanding

The core has a splitter which keeps track of outstanding reads and writes. You can read these counters. Beware that the outstanding reads counter can never be zero as it will always see the read of itself.

Additionally there is a outstanding IRQ mechanism. This generates an interrupt (to core0 only!) if the outstanding counters have been zero for a certain amount of time. The idea is that it gives you reasonable confidence that the none of the ARM core should have any more outstanding AXI transactions.

Address: 0x4000_002C AXI outstanding counters **Reset: 0x0000_0000**	
Bits	**Description**
31-26	0
15:16	Outstanding writes counter
15:10	0
9:0	Outstanding reads counter

Address: 0x4000_0030 AXI outstanding interrupt **Reset: 0x0000_0000**	
Bits	**Description**
31-21	0
20	AXI-outstanding interrupt enable
19:0	AXI-outstanding time-out MS24 bits

The counter is loaded with the 20 bits time-out value concatenated with 4'hF on the LS side. Thus the real-timeout value is time_out*16+15. (Setting it to zero gives a time out of 15). The ARM APB clock is used as time-out clock. The AXI timer is re-loaded as soon as a read or write is outstanding. Thus setting the time-out value to e.g. 0x1000 gives a interrupt if there has been seen no outstanding reads or write for ~65551 APB clocks.

To idea is to disable all interrupts (even the timers) except the time-out IRQ, then execute a WFI. You will get an interrupt if the AXI bus has been idle for the specified time.

If you do not want any new AXI transactions but still use the CPUs, you must make sure that all code/data is present in the L1 and L2 caches.

4.10 Core interrupt sources

The cores can get an interrupt or fast interrupt from many places. In order to speed up interrupt processing the interrupt source registers shows what the source bits are for the IRQ/FIQ. As is usual there is a register for each processor.

There are four interrupt source registers.

Address: 0x4000_0060 Core0 interrupt source **Address: 0x4000_0064 Core1 interrupt source** **Address: 0x4000_0068 Core2 interrupt source** **Address: 0x4000_006C Core3 interrupt source** **Reset: 0x0000_0000**	
Bits	**Description**
31-28	<Reserved>
17:12	Peripheral 1..15 interrupt (Currently not used)
11	Local timer interrupt
10	AXI-outstanding interrupt <For core 0 only!> all others are 0
9	PMU interrupt
8	GPU interrupt <Can be high in one core only>
7	Mailbox 3 interrupt
6	Mailbox 2 interrupt
5	Mailbox 1 interrupt
4	Mailbox 0 interrupt
3	CNTVIRQ interrupt
2	CNTHPIRQ interrupt
1	CNTPNSIRQ interrupt
0	CNTPSIRQ interrupt (Physical Timer -1)

There are four fast- interrupt source registers.

Address: 0x4000_0070 Core0 fast interrupt source **Address: 0x4000_0074 Core1 fast interrupt source** **Address: 0x4000_0078 Core2 fast interrupt source** **Address: 0x4000_007C Core3 fast interrupt source** **Reset: 0x0000_0000**	
Bits	**Description**
31-28	<Reserved>
17:12	Peripheral 1..15 fast interrupt (Currently not used)
11	Local timer fast interrupt
10	Always 0
9	PMU fast interrupt
8	GPU fast interrupt <Can be high in one core only>
7	Mailbox 3 fast interrupt
6	Mailbox 2 fast interrupt
5	Mailbox 1 fast interrupt
4	Mailbox 0 fast interrupt
3	CNTVIRQ fast interrupt
2	CNTHPIRQ fast interrupt
1	CNTPNSIRQ fast interrupt
0	CNTPSIRQ fast interrupt

4.11 Local timer

The code has a single local timer which can generate interrupts. The local timer ALWAYS gets its timing pulses from the Crystal clock. You get a 'timing pulse' every clock EDGE. Thus a 19.2 MHz crystal gives 38.4 M pulses/second.

The local timer has a 28-bit programmable divider which gives a lowest frequency of 38.4/2^28 = 0.14 Hz.

The local timer counts down and re-loads when it gets to zero. At the same time an interrupt-flag is set. The user must clear the interrupt flag. There is no detection if the interrupt flag is still set when the next time the local timer re-loads.

Address: 0x4000_0034 local timer control & status **Reset: 0x0000_0000**	
Bits	**Description**
31	Interrupt flag (Read-Only)
30	Unused
29	Interrupt enable (1= enabled)
28	Timer enable (1 = enabled)
0:27	Re-load value

When disabled the local timer loads the re-load value. Bit 32 is the status of the interrupt flag. The interrupt flag is always set upon a re-load and is independent of the interrupt enable bit. An interrupt is generated as long as the interrupt flag is set and the interrupt -enable bit is set.

The interrupt flag is clear by writing bit 31 high of the local timer IRQ clear & reload register.

Address: 0x4000_0038 local timer IRQ clear & reload (write-only) **Reset: -**	
Bits	**Description**
31	Interrupt flag clear when written as 1 (write-Only)
30	Local timer-reloaded when written as 1 (write only)
0:27	(Unused)

The IRQ clear & reload register has one extra bit: when writing bit 30 high, the local timer is immediately reloaded without generating an interrupt. As such it can also be used as a watchdog timer.

The local interrupt routing register is described here as the local time is the only local interrupt source present.

Address: 0x4000_0024 local interrupt routing **Reset: -**	
Bits	**Description**
31:3	(Unused)
0:2	000: Local timer interrupt goes to Core 0 IRQ 001: Local timer interrupt goes to Core 1 IRQ 010: Local timer interrupt goes to Core 2 IRQ 011: Local timer interrupt goes to Core 3 IRQ 100: Local timer interrupt goes to Core 0 FIQ 101: Local timer interrupt goes to Core 1 FIQ 110: Local timer interrupt goes to Core 2 FIQ 111: Local timer interrupt goes to Core 3 FIQ

Appendix A: Defines

These are not complete yet. The mailbox defines are fully tested (used)

```
// Timers interrupt control registers
#define CORE0_TIMER_IRQCNTL  0x40000040
#define CORE1_TIMER_IRQCNTL  0x40000044
#define CORE2_TIMER_IRQCNTL  0x40000048
#define CORE3_TIMER_IRQCNTL  0x4000004C

// Where to route timer interrupt to, IRQ/FIQ
// Setting both the IRQ and FIQ bit gives an FIQ
#define TIMER0_IRQ 0x01
#define TIMER1_IRQ 0x02
#define TIMER2_IRQ 0x04
#define TIMER3_IRQ 0x08
#define TIMER0_FIQ 0x10
#define TIMER1_FIQ 0x20
#define TIMER2_FIQ 0x40
#define TIMER3_FIQ 0x80

// Mailbox interrupt control registers
#define CORE0_MBOX_IRQCNTL   0x40000050
#define CORE1_MBOX_IRQCNTL   0x40000054
#define CORE2_MBOX_IRQCNTL   0x40000058
#define CORE3_MBOX_IRQCNTL   0x4000005C

// Where to route mailbox interrupt to, IRQ/FIQ
// Setting both the IRQ and FIQ bit gives an FIQ
#define MBOX0_IRQ 0x01
#define MBOX1_IRQ 0x02
#define MBOX2_IRQ 0x04
#define MBOX3_IRQ 0x08
#define MBOX0_FIQ 0x10
#define MBOX1_FIQ 0x20
#define MBOX2_FIQ 0x40
#define MBOX3_FIQ 0x80

// IRQ & FIQ source registers
#define CORE0_IRQ_SOURCE 0x40000060
#define CORE1_IRQ_SOURCE 0x40000064
#define CORE2_IRQ_SOURCE 0x40000068
#define CORE3_IRQ_SOURCE 0x4000006C
#define CORE0_FIQ_SOURCE 0x40000070
#define CORE1_FIQ_SOURCE 0x40000074
#define CORE2_FIQ_SOURCE 0x40000078
#define CORE3_FIQ_SOURCE 0x4000007C

// Interrupt source bits
// IRQ and FIQ are the same
// GPU bits can be set for one core only
#define INT_SRC_TIMER0   0x00000001
#define INT_SRC_TIMER1   0x00000002
#define INT_SRC_TIMER2   0x00000004
#define INT_SRC_TIMER3   0x00000008
#define INT_SRC_MBOX0    0x00000010
#define INT_SRC_MBOX1    0x00000020
#define INT_SRC_MBOX2    0x00000040
#define INT_SRC_MBOX3    0x00000080
#define INT_SRC_GPU      0x00000100
#define INT_SRC_PMU      0x00000200


// Mailbox write-set registers (Write only)
#define CORE0_MBOX0_SET 0x40000080
#define CORE0_MBOX1_SET 0x40000084
#define CORE0_MBOX2_SET 0x40000088
#define CORE0_MBOX3_SET 0x4000008C
```

```
#define CORE1_MBOX0_SET 0x40000090
#define CORE1_MBOX1_SET 0x40000094
#define CORE1_MBOX2_SET 0x40000098
#define CORE1_MBOX3_SET 0x4000009C
#define CORE2_MBOX0_SET 0x400000A0
#define CORE2_MBOX1_SET 0x400000A4
#define CORE2_MBOX2_SET 0x400000A8
#define CORE2_MBOX3_SET 0x400000AC
#define CORE3_MBOX0_SET 0x400000B0
#define CORE3_MBOX1_SET 0x400000B4
#define CORE3_MBOX2_SET 0x400000B8
#define CORE3_MBOX3_SET 0x400000BC

// Mailbox write-clear registers (Read & Write)
#define CORE0_MBOX0_RDCLR 0x400000C0
#define CORE0_MBOX1_RDCLR 0x400000C4
#define CORE0_MBOX2_RDCLR 0x400000C8
#define CORE0_MBOX3_RDCLR 0x400000CC
#define CORE1_MBOX0_RDCLR 0x400000D0
#define CORE1_MBOX1_RDCLR 0x400000D4
#define CORE1_MBOX2_RDCLR 0x400000D8
#define CORE1_MBOX3_RDCLR 0x400000DC
#define CORE2_MBOX0_RDCLR 0x400000E0
#define CORE2_MBOX1_RDCLR 0x400000E4
#define CORE2_MBOX2_RDCLR 0x400000E8
#define CORE2_MBOX3_RDCLR 0x400000EC
#define CORE3_MBOX0_RDCLR 0x400000F0
#define CORE3_MBOX1_RDCLR 0x400000F4
#define CORE3_MBOX2_RDCLR 0x400000F8
#define CORE3_MBOX3_RDCLR 0x400000FC
```