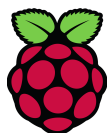


# **BCM2711 ARM Peripherals**

Raspberry Pi (Trading) Ltd.

Version 1, 5th February 2020

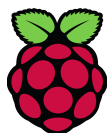


# Table of Contents

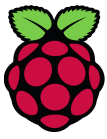
|  |    |
|--|----|
| Colophon .....   | 1  |
| List of Figures .....  | 2  |
| List of Tables .....   | 3  |
| 1. Introduction .....  | 8  |
| 1.1. Overview .....  | 8  |
| 1.2. Address map .....   | 8  |
| 1.2.1. Diagrammatic overview .....                                   | 8  |
| 1.2.2. Full 35-bit address map .....                                 | 9  |
| 1.2.3. ARM physical addresses .....                                  | 9  |
| 1.2.4. Legacy master addresses .....                                 | 10 |
| 1.3. Peripheral access precautions for correct memory ordering ..... | 10 |
| 2. Auxiliaries: UART1, SPI1 & SPI2 .....                             | 12 |
| 2.1. Overview .....  | 12 |
| 2.1.1. AUX registers .....   | 13 |
| 2.2. Mini UART .....   | 14 |
| 2.2.1. Mini UART implementation details .....                        | 15 |
| 2.2.2. Mini UART register details .....                              | 15 |
| 2.3. Universal SPI Master (2x) .....                                 | 23 |
| 2.3.1. SPI implementation details .....                              | 23 |
| 2.3.2. Interrupts .....  | 24 |
| 2.3.3. Long bit streams .....  | 25 |
| 2.3.4. SPI register details .....                                    | 25 |
| 3. BSC .....   | 31 |
| 3.1. Overview .....  | 31 |
| 3.2. Register View .....   | 31 |
| 3.3. 10-Bit Addressing .....   | 37 |
| 3.3.1. Writing .....   | 38 |
| 3.3.2. Reading .....   | 38 |
| 4. DMA Controller .....  | 40 |
| 4.1. Overview .....  | 40 |
| 4.2. DMA Controller Registers .....                                  | 41 |
| 4.2.1. DMA Channel Register Address Map .....                        | 41 |
| Control Block Data Structure .....                                   | 42 |
| Register Map .....   | 43 |
| Peripheral DREQ Signals .....  | 78 |
| 4.3. AXI Bursts .....  | 80 |
| 4.4. Error Handling .....  | 80 |



|   |     |
|---|-----|
| 4.5. DMA LITE Engines .....                 | 81  |
| 4.6. DMA4 Engines .....                     | 81  |
| 5. General Purpose I/O (GPIO) .....         | 82  |
| 5.1. Overview .....                         | 82  |
| 5.2. Register View .....                    | 83  |
| 5.3. Alternative Function Assignments ..... | 98  |
| 5.4. General Purpose GPIO Clocks .....      | 102 |
| 5.4.1. Operating Frequency .....            | 104 |
| 5.4.2. Register Definitions .....           | 104 |
| 6. Interrupts .....                         | 106 |
| 6.1. Overview .....                         | 106 |
| 6.2. Interrupt sources .....                | 107 |
| 6.2.1. ARM Core <i>n</i> interrupts .....   | 107 |
| 6.2.2. ARM_LOCAL interrupts .....           | 107 |
| 6.2.3. ARMC interrupts .....                | 107 |
| 6.2.4. VideoCore interrupts .....           | 108 |
| 6.2.5. ETH_PCIe interrupts .....            | 110 |
| 6.3. GIC-400 interrupt controller .....     | 110 |
| 6.4. Legacy interrupt controller .....      | 111 |
| 6.5. Registers .....                        | 114 |
| 6.5.1. GIC-400 .....                        | 114 |
| 6.5.2. ARM_LOCAL .....                      | 115 |
| 6.5.3. ARMC .....                           | 124 |
| 7. PCM / I2S Audio .....                    | 138 |
| 7.1. Overview .....                         | 138 |
| 7.2. Block Diagram .....                    | 139 |
| 7.3. Typical Timing .....                   | 139 |
| 7.4. Operation .....                        | 140 |
| 7.5. Software Operation .....               | 141 |
| 7.5.1. Operating in Polled mode .....       | 141 |
| 7.5.2. Operating in Interrupt mode .....    | 142 |
| 7.5.3. DMA .....                            | 142 |
| 7.6. Error Handling .....                   | 142 |
| 7.7. PDM Input Mode Operation .....         | 143 |
| 7.8. GRAY Code Input Mode Operation .....   | 143 |
| 7.9. PCM Register Map .....                 | 144 |
| 8. Pulse Width Modulator .....              | 156 |
| 8.1. Overview .....                         | 156 |
| 8.2. Block Diagram .....                    | 156 |



|   |     |
|---|-----|
| 8.3. PWM Implementation .....               | 156 |
| 8.4. Modes of Operation .....               | 157 |
| 8.5. Quick Reference .....                  | 158 |
| 8.6. Control and Status Registers .....     | 158 |
| 9. SPI .....                                | 164 |
| 9.1. Overview .....                         | 164 |
| 9.2. SPI Master Mode .....                  | 164 |
| 9.2.1. Standard mode .....                  | 164 |
| 9.2.2. Bidirectional mode .....             | 165 |
| 9.3. LoSSI mode .....                       | 166 |
| 9.3.1. Command write .....                  | 166 |
| 9.3.2. Parameter write .....                | 166 |
| 9.3.3. Byte read commands .....             | 166 |
| 9.3.4. 24-bit read command .....            | 167 |
| 9.3.5. 32-bit read command .....            | 167 |
| 9.4. Block Diagram .....                    | 167 |
| 9.5. SPI Register Map .....                 | 168 |
| 9.6. Software Operation .....               | 173 |
| 9.6.1. Polled .....                         | 173 |
| 9.6.2. Interrupt .....                      | 173 |
| 9.6.3. DMA .....                            | 173 |
| 9.6.4. Notes .....                          | 174 |
| 10. System Timer .....                      | 175 |
| 10.1. Overview .....                        | 175 |
| 10.2. System Timer Registers .....          | 175 |
| 11. UART .....                              | 178 |
| 11.1. Overview .....                        | 178 |
| 11.2. Variations from the 16C650 UART ..... | 179 |
| 11.3. Primary UART Inputs and Outputs ..... | 179 |
| 11.4. UART Interrupts .....                 | 180 |
| 11.5. Register View .....                   | 181 |
| 12. Timer (ARM side) .....                  | 197 |
| 12.1. Overview .....                        | 197 |
| 12.2. Timer Registers .....                 | 197 |
| 13. ARM Mailboxes .....                     | 202 |
| 13.1. Overview .....                        | 202 |
| 13.2. Registers .....                       | 202 |



# Colophon

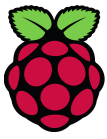
BCM2711 ARM Peripherals, based in large part on the earlier BCM2835 ARM Peripherals documentation.

© 2012 Broadcom Europe Ltd., 2020 Raspberry Pi (Trading) Ltd.  
All rights reserved.

| Release | Date       | Description   |
|---------|------------|---------------|
| 1       | 05/02/2020 | First release |

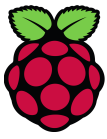
*Table 1. Release History*

The latest release of this document can be found at <https://www.raspberrypi.org>



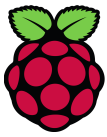
# List of Figures

|                     |   |
|---------------------|---|
| Figure 1 . . . . .  | BCM2711 Address Maps                          |
| Figure 2 . . . . .  | Write to a slave with 10-bit address          |
| Figure 3 . . . . .  | Read from slave with 10-bit address           |
| Figure 4 . . . . .  | GPIO Block Diagram                            |
| Figure 5 . . . . .  | Interrupt sources and destinations            |
| Figure 6 . . . . .  | Peripheral IRQ OR-ing                         |
| Figure 7 . . . . .  | GIC IRQ routing                               |
| Figure 8 . . . . .  | Legacy IRQ routing                            |
| Figure 9 . . . . .  | Legacy IRQ status registers                   |
| Figure 10 . . . . . | PCM Audio Interface Typical Timing            |
| Figure 11 . . . . . | PCM Audio Interface Block Diagram             |
| Figure 12 . . . . . | Timing at Start of Frame                      |
| Figure 13 . . . . . | Gray mode input format                        |
| Figure 14 . . . . . | PWM block diagram                             |
| Figure 15 . . . . . | Serial bit transmission when M/S Mode enabled |
| Figure 16 . . . . . | SPI Master Typical Usage                      |
| Figure 17 . . . . . | SPI Cycle                                     |
| Figure 18 . . . . . | Different Clock Polarity/Phase                |
| Figure 19 . . . . . | Bidirectional SPI Master Typical Usage        |
| Figure 20 . . . . . | LoSSI mode Typical usage                      |
| Figure 21 . . . . . | Serial interface Block Diagram                |



# List of Tables

|          |   |
|----------|---|
| Table 1  | Release History   |
| Table 2  | Auxiliary peripherals Address Map   |
| Table 3  | AUX_IRQ Register  |
| Table 4  | AUX_ENABLES Register  |
| Table 5  | AUX_MU_IO_REG Register  |
| Table 6  | AUX_MU_IER_REG Register   |
| Table 7  | AUX_MU_IIR_REG Register   |
| Table 8  | AUX_MU_LCR_REG Register   |
| Table 9  | AUX_MU_MCR_REG Register   |
| Table 10 | AUX_MU_LSR_REG Register   |
| Table 11 | AUX_MU_MSR_REG Register   |
| Table 12 | AUX_MU_SCRATCH Register   |
| Table 13 | AUX_MU_CNTL_REG Register  |
| Table 14 | AUX_MU_STAT_REG Register  |
| Table 15 | AUX_MU_BAUD_REG Register  |
| Table 16 | AUX_SPI1_CNTL0_REG & AUX_SPI2_CNTL0_REG Registers   |
| Table 17 | AUX_SPI1_CNTL1_REG & AUX_SPI2_CNTL1_REG Registers   |
| Table 18 | AUX_SPI1_STAT_REG & AUX_SPI2_STAT_REG Registers   |
| Table 19 | AUX_SPI1_PEEK_REG & AUX_SPI2_PEEK_REG Registers   |
| Table 20 | AUX_SPI1_IO_REG & AUX_SPI2_IO_REG Registers   |
| Table 21 | AUX_SPI1_TXHOLD_REG & AUX_SPI2_TXHOLD_REG Registers   |
| Table 22 | I2C Address Map   |
| Table 23 | C Register  |
| Table 24 | S Register  |
| Table 25 | DLEN Register   |
| Table 26 | A Register  |
| Table 27 | FIFO Register   |
| Table 28 | DIV Register  |
| Table 29 | DEL Register  |
| Table 30 | CLKT Register   |
| Table 31 | DMA Controller Register Address Map   |
| Table 32 | DMA Control Block Definition  |
| Table 33 | DMA Lite Control Block Definition   |
| Table 34 | DMA4 Control Block Definition   |
| Table 35 | DMA Controller Register Map   |
| Table 36 | 0_CS, 1_CS, 2_CS, 3_CS, 4_CS, 5_CS, 6_CS, 7_CS, 8_CS, 9_CS & 10_CS Registers  |
| Table 37 | 0_CONBLK_AD, 1_CONBLK_AD, 2_CONBLK_AD, 3_CONBLK_AD, 4_CONBLK_AD, 5_CONBLK_AD, 6_CONBLK_AD, 7_CONBLK_AD, 8_CONBLK_AD, 9_CONBLK_AD & 10_CONBLK_AD Registers |
| Table 38 | 0_TI, 1_TI, 2_TI, 3_TI, 4_TI, 5_TI & 6_TI Registers   |
| Table 39 | 0_SOURCE_AD, 1_SOURCE_AD, 2_SOURCE_AD, 3_SOURCE_AD, 4_SOURCE_AD, 5_SOURCE_AD, 6_SOURCE_AD, 7_SOURCE_AD, 8_SOURCE_AD, 9_SOURCE_AD & 10_SOURCE_AD Registers |
| Table 40 | 0_DEST_AD, 1_DEST_AD, 2_DEST_AD, 3_DEST_AD, 4_DEST_AD, 5_DEST_AD, 6_DEST_AD, 7_DEST_AD, 8_DEST_AD, 9_DEST_AD & 10_DEST_AD Registers                       |



7\_DEST\_AD, 8\_DEST\_AD, 9\_DEST\_AD & 10\_DEST\_AD Registers

Table 41 . . . . 0\_TXFR\_LEN, 1\_TXFR\_LEN, 2\_TXFR\_LEN, 3\_TXFR\_LEN, 4\_TXFR\_LEN, 5\_TXFR\_LEN & 6\_TXFR\_LEN Registers

Table 42 . . . . 0\_STRIDE, 1\_STRIDE, 2\_STRIDE, 3\_STRIDE, 4\_STRIDE, 5\_STRIDE & 6\_STRIDE Registers

Table 43 . . . . 0\_NEXTCONBK, 1\_NEXTCONBK, 2\_NEXTCONBK, 3\_NEXTCONBK, 4\_NEXTCONBK, 5\_NEXTCONBK, 6\_NEXTCONBK, 7\_NEXTCONBK, 8\_NEXTCONBK, 9\_NEXTCONBK & 10\_NEXTCONBK Registers

Table 44 . . . . 0\_DEBUG, 1\_DEBUG, 2\_DEBUG, 3\_DEBUG, 4\_DEBUG, 5\_DEBUG & 6\_DEBUG Registers

Table 45 . . . . 7\_TI, 8\_TI, 9\_TI & 10\_TI Registers

Table 46 . . . . 7\_TXFR\_LEN, 8\_TXFR\_LEN, 9\_TXFR\_LEN & 10\_TXFR\_LEN Registers

Table 47 . . . . 7\_DEBUG, 8\_DEBUG, 9\_DEBUG & 10\_DEBUG Registers

Table 48 . . . . 11\_CS, 12\_CS, 13\_CS & 14\_CS Registers

Table 49 . . . . 11\_CB, 12\_CB, 13\_CB & 14\_CB Registers

Table 50 . . . . 11\_DEBUG, 12\_DEBUG, 13\_DEBUG & 14\_DEBUG Registers

Table 51 . . . . 11\_TI, 12\_TI, 13\_TI & 14\_TI Registers

Table 52 . . . . 11\_SRC, 12\_SRC, 13\_SRC & 14\_SRC Registers

Table 53 . . . . 11\_SRCI, 12\_SRCI, 13\_SRCI & 14\_SRCI Registers

Table 54 . . . . 11\_DEST, 12\_DEST, 13\_DEST & 14\_DEST Registers

Table 55 . . . . 11\_DESTI, 12\_DESTI, 13\_DESTI & 14\_DESTI Registers

Table 56 . . . . 11\_LEN, 12\_LEN, 13\_LEN & 14\_LEN Registers

Table 57 . . . . 11\_NEXT\_CB, 12\_NEXT\_CB, 13\_NEXT\_CB & 14\_NEXT\_CB Registers

Table 58 . . . . 11\_DEBUG2, 12\_DEBUG2, 13\_DEBUG2 & 14\_DEBUG2 Registers

Table 59 . . . . INT\_STATUS Register

Table 60 . . . . ENABLE Register

Table 61 . . . . GPIO Register Assignment

Table 62 . . . . GPIO Alternate function select register 0

Table 63 . . . . GPIO Alternate function select register 1

Table 64 . . . . GPIO Alternate function select register 2

Table 65 . . . . GPIO Alternate function select register 3

Table 66 . . . . GPIO Alternate function select register 4

Table 67 . . . . GPIO Alternate function select register 5

Table 68 . . . . GPIO Output Set Register 0

Table 69 . . . . GPIO Output Set Register 1

Table 70 . . . . GPIO Output Clear Register 0

Table 71 . . . . GPIO Output Clear Register 1

Table 72 . . . . GPIO Level Register 0

Table 73 . . . . GPIO Level Register 1

Table 74 . . . . GPIO Event Detect Status Register 0

Table 75 . . . . GPIO Event Detect Status Register 1

Table 76 . . . . GPIO Rising Edge Detect Status Register 0

Table 77 . . . . GPIO Rising Edge Detect Status Register 1

Table 78 . . . . GPIO Falling Edge Detect Status Register 0

Table 79 . . . . GPIO Falling Edge Detect Status Register 1

Table 80 . . . . GPIO High Detect Status Register 0

Table 81 . . . . GPIO High Detect Status Register 1

Table 82 . . . . GPIO Low Detect Status Register 0





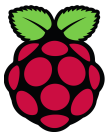
|           |   |
|-----------|---|
| Table 83  | GPIO Low Detect Status Register 1   |
| Table 84  | GPIO Asynchronous rising Edge Detect Status Register 0                    |
| Table 85  | GPIO Asynchronous rising Edge Detect Status Register 1                    |
| Table 86  | GPIO Asynchronous Falling Edge Detect Status Register 0                   |
| Table 87  | GPIO Asynchronous Falling Edge Detect Status Register 1                   |
| Table 88  | GPIO Pull-up / Pull-down Register 0                                       |
| Table 89  | GPIO Pull-up / Pull-down Register 1                                       |
| Table 90  | GPIO Pull-up / Pull-down Register 2                                       |
| Table 91  | GPIO Pull-up / Pull-down Register 3                                       |
| Table 92  | GPIO Pins Alternative Function Assignment                                 |
| Table 93  | GPIO Pins Alternative Function Legend                                     |
| Table 94  | Effect of MASH Filter on Frequency  |
| Table 95  | Example of Frequency Spread when using MASH Filtering                     |
| Table 96  | General Purpose Clocks Registers  |
| Table 97  | General Purpose Clocks Control  |
| Table 98  | General Purpose Clock Divisors  |
| Table 99  | ARMC peripheral IRQs  |
| Table 100 | VC peripheral IRQs  |
| Table 101 | ETH_PCIe L2 IRQs  |
| Table 102 | ARM_LOCAL Interrupt Registers   |
| Table 103 | ARM_CONTROL Register  |
| Table 104 | CORE_IRQ_CONTROL Register   |
| Table 105 | PMU_CONTROL_SET Register  |
| Table 106 | PMU_CONTROL_CLR Register  |
| Table 107 | PERI_IRQ_ROUTE0 Register  |
| Table 108 | AXI_QUIET_TIME Register   |
| Table 109 | LOCAL_TIMER_CONTROL Register  |
| Table 110 | LOCAL_TIMER_IRQ Register  |
| Table 111 | TIMER_CNTRL0, TIMER_CNTRL1, TIMER_CNTRL2 & TIMER_CNTRL3 Registers         |
| Table 112 | MAILBOX_CNTRL0, MAILBOX_CNTRL1, MAILBOX_CNTRL2 & MAILBOX_CNTRL3 Registers |
| Table 113 | IRQ_SOURCE0, IRQ_SOURCE1, IRQ_SOURCE2 & IRQ_SOURCE3 Registers             |
| Table 114 | FIQ_SOURCE0, FIQ_SOURCE1, FIQ_SOURCE2 & FIQ_SOURCE3 Registers             |
| Table 115 | ARMC Interrupt Registers  |
| Table 116 | IRQ0_PENDING0, IRQ1_PENDING0, IRQ2_PENDING0 & IRQ3_PENDING0 Registers     |
| Table 117 | IRQ0_PENDING1, IRQ1_PENDING1, IRQ2_PENDING1 & IRQ3_PENDING1 Registers     |
| Table 118 | IRQ0_PENDING2, IRQ1_PENDING2, IRQ2_PENDING2 & IRQ3_PENDING2 Registers     |
| Table 119 | IRQ0_SET_EN_0, IRQ1_SET_EN_0, IRQ2_SET_EN_0 & IRQ3_SET_EN_0 Registers     |
| Table 120 | IRQ0_SET_EN_1, IRQ1_SET_EN_1, IRQ2_SET_EN_1 & IRQ3_SET_EN_1 Registers     |
| Table 121 | IRQ0_SET_EN_2, IRQ1_SET_EN_2, IRQ2_SET_EN_2 & IRQ3_SET_EN_2 Registers     |
| Table 122 | IRQ0_CLR_EN_0, IRQ1_CLR_EN_0, IRQ2_CLR_EN_0 & IRQ3_CLR_EN_0 Registers     |
| Table 123 | IRQ0_CLR_EN_1, IRQ1_CLR_EN_1, IRQ2_CLR_EN_1 & IRQ3_CLR_EN_1 Registers     |
| Table 124 | IRQ0_CLR_EN_2, IRQ1_CLR_EN_2, IRQ2_CLR_EN_2 & IRQ3_CLR_EN_2 Registers     |
| Table 125 | IRQ_STATUS0 Register  |
| Table 126 | IRQ_STATUS1 Register  |
| Table 127 | IRQ_STATUS2 Register  |
| Table 128 | FIQ0_PENDING0, FIQ1_PENDING0, FIQ2_PENDING0 & FIQ3_PENDING0 Registers     |



|               |   |
|---------------|---|
| Table 129 ... | FIQ0_PENDING1, FIQ1_PENDING1, FIQ2_PENDING1 & FIQ3_PENDING1 Registers |
| Table 130 ... | FIQ0_PENDING2, FIQ1_PENDING2, FIQ2_PENDING2 & FIQ3_PENDING2 Registers |
| Table 131 ... | FIQ0_SET_EN_0, FIQ1_SET_EN_0, FIQ2_SET_EN_0 & FIQ3_SET_EN_0 Registers |
| Table 132 ... | FIQ0_SET_EN_1, FIQ1_SET_EN_1, FIQ2_SET_EN_1 & FIQ3_SET_EN_1 Registers |
| Table 133 ... | FIQ0_SET_EN_2, FIQ1_SET_EN_2, FIQ2_SET_EN_2 & FIQ3_SET_EN_2 Registers |
| Table 134 ... | FIQ0_CLR_EN_0, FIQ1_CLR_EN_0, FIQ2_CLR_EN_0 & FIQ3_CLR_EN_0 Registers |
| Table 135 ... | FIQ0_CLR_EN_1, FIQ1_CLR_EN_1, FIQ2_CLR_EN_1 & FIQ3_CLR_EN_1 Registers |
| Table 136 ... | FIQ0_CLR_EN_2, FIQ1_CLR_EN_2, FIQ2_CLR_EN_2 & FIQ3_CLR_EN_2 Registers |
| Table 137 ... | SWIRQ_SET Register  |
| Table 138 ... | SWIRQ_CLEAR Register  |
| Table 139 ... | PDM Input Mode Configuration  |
| Table 140 ... | PCM Register Map  |
| Table 141 ... | CS_A Register   |
| Table 142 ... | FIFO_A Register   |
| Table 143 ... | MODE_A Register   |
| Table 144 ... | RXC_A Register  |
| Table 145 ... | TXC_A Register  |
| Table 146 ... | DREQ_A Register   |
| Table 147 ... | INTEN_A Register  |
| Table 148 ... | INTSTC_A Register   |
| Table 149 ... | GRAY Register   |
| Table 150 ... | PWM Register Map  |
| Table 151 ... | CTL Register  |
| Table 152 ... | STA Register  |
| Table 153 ... | DMAC Register   |
| Table 154 ... | RNG1 & RNG2 Registers   |
| Table 155 ... | DAT1 & DAT2 Registers   |
| Table 156 ... | FIF1 Register   |
| Table 157 ... | SPI Register Map  |
| Table 158 ... | CS Register   |
| Table 159 ... | FIFO Register   |
| Table 160 ... | CLK Register  |
| Table 161 ... | DLEN Register   |
| Table 162 ... | LTOH Register   |
| Table 163 ... | DC Register   |
| Table 164 ... | System Timer Registers  |
| Table 165 ... | CS Register   |
| Table 166 ... | CLO Register  |
| Table 167 ... | CHI Register  |
| Table 168 ... | C0, C1, C2 & C3 Registers   |
| Table 169 ... | UART Assignment on the GPIO Pin map                                   |
| Table 170 ... | UART Registers  |
| Table 171 ... | DR Register   |
| Table 172 ... | RSRECR Register   |
| Table 173 ... | FR Register   |
| Table 174 ... | ILPR Register   |



|               |   |
|---------------|---|
| Table 175 ... | IBRD Register   |
| Table 176 ... | FBRD Register   |
| Table 177 ... | LCRH Register   |
| Table 178 ... | UART parity bits  |
| Table 179 ... | CR Register   |
| Table 180 ... | IFLS Register   |
| Table 181 ... | IMSC Register   |
| Table 182 ... | RIS Register  |
| Table 183 ... | MIS Register  |
| Table 184 ... | ICR Register  |
| Table 185 ... | DMACR Register  |
| Table 186 ... | ITCR Register   |
| Table 187 ... | ITIP Register   |
| Table 188 ... | ITOP Register   |
| Table 189 ... | TDR Register  |
| Table 190 ... | Timer Registers   |
| Table 191 ... | Timer Load register   |
| Table 192 ... | Timer Value register  |
| Table 193 ... | Timer control register  |
| Table 194 ... | Timer IRQ clear register  |
| Table 195 ... | Timer Raw IRQ register  |
| Table 196 ... | Timer Masked IRQ register   |
| Table 197 ... | Timer Reload register   |
| Table 198 ... | Timer pre-divider register  |
| Table 199 ... | Free running counter  |
| Table 200 ... | ARM Mailbox registers   |
| Table 201 ... | MBOX_SET00, MBOX_SET01, MBOX_SET02, MBOX_SET03, MBOX_SET04, MBOX_SET05,<br>MBOX_SET06, MBOX_SET07, MBOX_SET08, MBOX_SET09, MBOX_SET10, MBOX_SET11, MBOX_SET12,<br>MBOX_SET13, MBOX_SET14 & MBOX_SET15 Registers |
| Table 202 ... | MBOX_CLR00, MBOX_CLR01, MBOX_CLR02, MBOX_CLR03, MBOX_CLR04, MBOX_CLR05,<br>MBOX_CLR06, MBOX_CLR07, MBOX_CLR08, MBOX_CLR09, MBOX_CLR10, MBOX_CLR11, MBOX_CLR12,<br>MBOX_CLR13, MBOX_CLR14 & MBOX_CLR15 Registers |



# Chapter 1. Introduction

## 1.1. Overview

BCM2711 contains the following peripherals which may safely be accessed by the ARM:

- Timers
- Interrupt controller
- GPIO
- USB
- PCM / I2S
- DMA controller
- I2C masters
- SPI masters
- PWM
- UARTs

The purpose of this datasheet is to provide documentation for these peripherals in sufficient detail to allow a developer to port an operating system to BCM2711. Not all of these peripherals have been fully documented yet.

There are a number of peripherals which are intended to be controlled by the GPU. These are omitted from this datasheet. Accessing these peripherals from the ARM is not recommended.

## 1.2. Address map

### 1.2.1. Diagrammatic overview

The BCM2711 has two main addressing schemes: a "Full" 35-bit address bus and a 32-bit "Legacy Master" view as seen by the peripherals (except for "large address" masters). There's also a "Low Peripherals" mode which modifies the ARM's view of the peripheral addresses. [Figure 1](#) shows how these address maps inter-relate. Note that the relative sizes of the address blocks in the diagram are definitely **not** to scale! (The PCIe address range covers 8GB, but the Main peripherals address range only covers 64MB.)

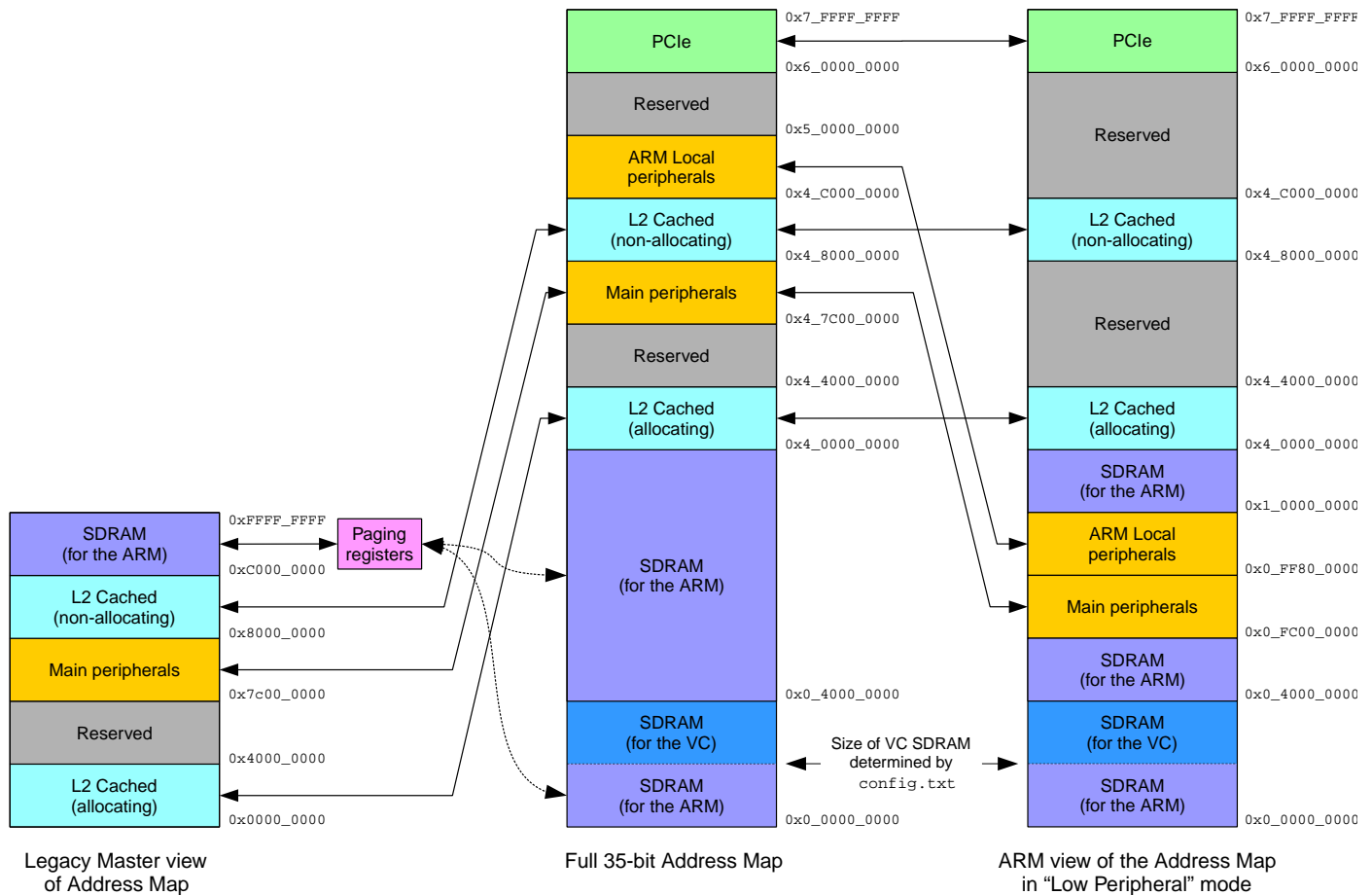


Figure 1. BCM2711 Address Maps

Addresses in ARM Linux are:

1. Issued as virtual addresses by the ARM core, then
2. Mapped into a physical address by the ARM MMU, then
3. Used to select the appropriate peripheral or location in RAM

### 1.2.2. Full 35-bit address map

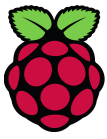
The full 35-bit address map is shown in Figure 1. This is seen by both "large address" masters (e.g. the DMA4 engines) and the ARM CPU.

It has two L2 cache aliases (one allocating, one non-allocating) which cache (only) the first 1GB of SDRAM.

### 1.2.3. ARM physical addresses

Physical addresses start at 0x0\_0000\_0000 for RAM.

- The ARM section of the RAM starts at 0x0\_0000\_0000 and extends up to the size of installed SDRAM.
- The VideoCore section of the RAM is mapped in from 0x0\_4000\_0000 downwards. The size of the VideoCore RAM is determined by a setting in `config.txt` - refer to [raspberrypi.org documentation](https://www.raspberrypi.org/documentation) for



further details.

The VideoCore maps the ARM physical address space directly to the bus address space seen by VideoCore. The bus addresses for RAM are set up to map onto the uncached<sup>[1]</sup> bus address range on the VideoCore starting at 0x0\_0000\_0000.

When running in 32-bit mode, the ARM uses LPAA mode to enable it to access the full 32GB address space.

Physical addresses range from 0x4\_7C00\_0000 to 0x4\_7FFF\_FFFF for Main peripherals, and from 0x4\_C000\_0000 to 0x4\_FFFF\_FFFF for ARM Local peripherals.

If the VPU enables "Low Peripheral" mode then the ARM (only) has Main peripherals available from 0x0\_FC00\_0000 to 0x0\_FF7F\_FFFF and ARM Local peripherals available from 0x0\_FF80\_0000 to 0x0\_FFFF\_FFFF.

#### 1.2.4. Legacy master addresses

**The peripheral addresses specified in this document are legacy master addresses.** Software accessing peripherals using the DMA engines must use 32-bit legacy master addresses. The Main peripherals are available from 0x7C00\_0000 to 0x7FFF\_FFFF. Behind the scenes, the VideoCore transparently translates these addresses to the 35-bit 0x4\_7nnn\_nnnn addresses.

So a peripheral described in this document as being at legacy address 0x7Enn\_nnnn is available in the 35-bit address space at 0x4\_7Enn\_nnnn, and visible to the ARM at 0x0\_FEnn\_nnnn if Low Peripheral mode is enabled.

Software accessing RAM using the DMA engines must use legacy addresses (between 0xC000\_0000 and 0xFFFF\_FFFF). This accesses a 1GB window within the full 16GB SDRAM address space. If the DMA engine needs to access RAM above the first 1GB, this window can be moved using the PAGE or PAGELITE bits - see [Chapter 4](#) for more details. Behind the scenes, the VideoCore transparently translates these addresses to the 35-bit 0x0\_nnnn\_nnnn addresses.

Software accessing the VPU L2 cache using the DMA engines must use legacy addresses starting at 0x0000\_0000 (for allocating cache) or 0x8000\_0000 (for non-allocating cache). Behind the scenes, the VideoCore transparently translates these addresses to the corresponding 35-bit 0x4\_nnnn\_nnnn addresses. These 1GB windows can't be moved, and are limited to the first 1GB of SDRAM.

### 1.3. Peripheral access precautions for correct memory ordering

The BCM2711 system uses an AMBA AXI-compatible interface structure. In order to keep the system complexity low and data throughput high, the BCM2711 AXI system does not always return read data in-order.<sup>[2]</sup> The GPU has special logic to cope with data arriving out-of-order; however the ARM core does not contain such logic. Therefore some precautions must be taken when using the ARM to access peripherals.



Accesses to the same peripheral will always arrive and return in-order. It is only when switching from one peripheral to another that data can arrive out-of-order. The simplest way to make sure that data is processed in-order is to place a memory barrier instruction at critical positions in the code. You should place:

- A memory write barrier before the first write to a peripheral
- A memory read barrier after the last read of a peripheral

It is **not** required to put a memory barrier instruction after **each** read or write access. Only at those places in the code where it is possible that a peripheral read or write may be followed by a read or write of a **different** peripheral. This is normally at the entry and exit points of the peripheral service code.

As interrupts can appear anywhere in the code, you should also safeguard those. If an interrupt routine reads from a peripheral the routine should start with a memory read barrier. If an interrupt routine writes to a peripheral the routine should end with a memory write barrier.

[1] BCM2711 provides a 1MB system L2 cache, which is used primarily by the GPU. Accesses to memory are routed either via or around the L2 cache depending on the address range being used.

[2] Normally a processor assumes that if it executes two read operations the data will arrive in order. So a read from location X followed by a read from location Y should return the data of location X first, followed by the data of location Y. Data arriving out of order can have disastrous consequences. For example:

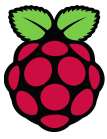
```
a_status = *pointer_to_peripheral_a;  
b_status = *pointer_to_peripheral_b;
```

Without precautions the values ending up in the variables `a_status` and `b_status` can be swapped around.

It is theoretically possible for writes to go ‘wrong’ but that is far more difficult to achieve. The AXI system makes sure the data always arrives in-order at its intended destination. So:

```
*pointer_to_peripheral_a = value_a;  
*pointer_to_peripheral_b = value_b;
```

will always give the expected result. The only time write data can arrive out-of-order is if two different peripherals are connected to the same external equipment.



## Chapter 2. Auxiliaries: UART1, SPI1 & SPI2

### 2.1. Overview

The BCM2711 device has three Auxiliary peripherals: One mini UART (UART1) and two SPI masters (SPI1 & SPI2). These three peripherals are grouped together as they share the same area in the peripheral register map and they share a common interrupt. Also all three are controlled by the Auxiliary enable register. The Auxiliary register base address is `0x7E21 5000`.

| Address Offset | Register Name                       | Description                  | Size |
|----------------|-------------------------------------|------------------------------|------|
| 0x00           | <a href="#">AUX_IRQ</a>             | Auxiliary Interrupt status   | 3    |
| 0x04           | <a href="#">AUX_ENABLES</a>         | Auxiliary enables            | 3    |
| 0x40           | <a href="#">AUX_MU_IO_REG</a>       | Mini UART I/O Data           | 8    |
| 0x44           | <a href="#">AUX_MU_IER_REG</a>      | Mini UART Interrupt Enable   | 8    |
| 0x48           | <a href="#">AUX_MU_IIR_REG</a>      | Mini UART Interrupt Identify | 8    |
| 0x4C           | <a href="#">AUX_MU_LCR_REG</a>      | Mini UART Line Control       | 8    |
| 0x50           | <a href="#">AUX_MU_MCR_REG</a>      | Mini UART Modem Control      | 8    |
| 0x54           | <a href="#">AUX_MU_LSR_REG</a>      | Mini UART Line Status        | 8    |
| 0x58           | <a href="#">AUX_MU_MSR_REG</a>      | Mini UART Modem Status       | 8    |
| 0x5C           | <a href="#">AUX_MU_SCRATCH</a>      | Mini UART Scratch            | 8    |
| 0x60           | <a href="#">AUX_MU_CNTL_REG</a>     | Mini UART Extra Control      | 8    |
| 0x64           | <a href="#">AUX_MU_STAT_REG</a>     | Mini UART Extra Status       | 32   |
| 0x68           | <a href="#">AUX_MU_BAUD_REG</a>     | Mini UART Baudrate           | 16   |
| 0x80           | <a href="#">AUX_SPI1_CNTL0_REG</a>  | SPI 1 Control register 0     | 32   |
| 0x84           | <a href="#">AUX_SPI1_CNTL1_REG</a>  | SPI 1 Control register 1     | 8    |
| 0x88           | <a href="#">AUX_SPI1_STAT_REG</a>   | SPI 1 Status                 | 32   |
| 0x8C           | <a href="#">AUX_SPI1_PEEK_REG</a>   | SPI 1 Peek                   | 16   |
| 0xA0           | <a href="#">AUX_SPI1_IO_REG</a>     | SPI 1 Data                   | 32   |
| 0xA4           | <a href="#">AUX_SPI1_IO_REG</a>     | SPI 1 Data                   | 32   |
| 0xA8           | <a href="#">AUX_SPI1_IO_REG</a>     | SPI 1 Data                   | 32   |
| 0xAC           | <a href="#">AUX_SPI1_IO_REG</a>     | SPI 1 Data                   | 32   |
| 0xB0           | <a href="#">AUX_SPI1_TXHOLD_REG</a> | SPI 1 Extended Data          | 32   |
| 0xB4           | <a href="#">AUX_SPI1_TXHOLD_REG</a> | SPI 1 Extended Data          | 32   |
| 0xB8           | <a href="#">AUX_SPI1_TXHOLD_REG</a> | SPI 1 Extended Data          | 32   |





| Address Offset | Register Name                       | Description              | Size |
|----------------|-------------------------------------|--------------------------|------|
| 0xBC           | <a href="#">AUX_SPI1_TXHOLD_REG</a> | SPI 1 Extended Data      | 32   |
| 0xC0           | <a href="#">AUX_SPI2_CNTL0_REG</a>  | SPI 2 Control register 0 | 32   |
| 0xC4           | <a href="#">AUX_SPI2_CNTL1_REG</a>  | SPI 2 Control register 1 | 8    |
| 0xC8           | <a href="#">AUX_SPI2_STAT_REG</a>   | SPI 2 Status             | 32   |
| 0xCC           | <a href="#">AUX_SPI2_PEEK_REG</a>   | SPI 2 Peek               | 16   |
| 0xE0           | <a href="#">AUX_SPI2_IO_REG</a>     | SPI 2 Data               | 32   |
| 0xE4           | <a href="#">AUX_SPI2_IO_REG</a>     | SPI 2 Data               | 32   |
| 0xE8           | <a href="#">AUX_SPI2_IO_REG</a>     | SPI 2 Data               | 32   |
| 0xEC           | <a href="#">AUX_SPI2_IO_REG</a>     | SPI 2 Data               | 32   |
| 0xF0           | <a href="#">AUX_SPI2_TXHOLD_REG</a> | SPI 2 Extended Data      | 32   |
| 0xF4           | <a href="#">AUX_SPI2_TXHOLD_REG</a> | SPI 2 Extended Data      | 32   |
| 0xF8           | <a href="#">AUX_SPI2_TXHOLD_REG</a> | SPI 2 Extended Data      | 32   |
| 0xFC           | <a href="#">AUX_SPI2_TXHOLD_REG</a> | SPI 2 Extended Data      | 32   |

Table 2. Auxiliary peripherals Address Map

### 2.1.1. AUX registers

There are two Auxiliary registers which control all three devices. One is the interrupt status register, the second is the Auxiliary enable register. The Auxiliary IRQ status register can help to hierarchically determine the source of an interrupt.

#### AUX\_IRQ Register

##### Synopsis

The AUX\_IRQ register is used to check any pending interrupts which may be asserted by the three Auxiliary sub blocks.

| Bit(s) | Field Name                               | Description                                       | Type | Reset |
|--------|--|---|------|-------|
| 31:3   | Reserved, write zero, read as don't care |   |      |       |
| 2      | SPI 2 IRQ                                | If set the SPI 2 module has an interrupt pending. | RO   | 0     |
| 1      | SPI 1 IRQ                                | If set the SPI1 module has an interrupt pending.  | RO   | 0     |
| 0      | Mini UART IRQ                            | If set the mini UART has an interrupt pending.    | RO   | 0     |

Table 3. AUX\_IRQ Register

#### AUX\_ENABLES Register



## Synopsis

The AUX\_ENABLES register is used to enable the three modules: UART1, SPI1, SPI2.

| Bit(s) | Field Name       | Description  | Type | Reset |
|--------|------------------|--|------|-------|
| 31:3   |                  | Reserved, write zero, read as don't care   |      |       |
| 2      | SPI2 enable      | If set the SPI 2 module is enabled.<br>If clear the SPI 2 module is disabled. That also disables any SPI 2 module register access  | RW   | 0     |
| 1      | SPI 1 enable     | If set the SPI 1 module is enabled.<br>If clear the SPI 1 module is disabled. That also disables any SPI 1 module register access  | RW   | 0     |
| 0      | Mini UART enable | If set the mini UART is enabled. The UART will immediately start receiving data, especially if the UART1_RX line is <i>low</i> .<br>If clear the mini UART is disabled. That also disables any mini UART register access | RW   | 0     |

Table 4. AUX\_ENABLES Register

If the enable bits are clear you will have **no access** to a peripheral. You can not even read or write the registers!

GPIO pins should be set up first before enabling the UART. The UART core is built to emulate 16550 behaviour. So when it is enabled any data at the inputs will immediately be received. If the UART1\_RX line is low (because the GPIO pins have not been set-up yet) that will be seen as a start bit and the UART will start receiving 0x00-characters.

Valid stops bits are not required for the UART. (See also Implementation details). Hence any bit status is acceptable as a stop bit, and is only used so there is a clean timing start for the next bit.

Immediately after a reset, the baudrate register will be zero and the system clock will be 250 MHz. So only 2.5 µseconds suffice to fill the receive FIFO. The result will be that if the UART is enabled without changing the default configuration, the FIFO will be full and overflowing in no time flat.

## 2.2. Mini UART

The mini UART is a secondary low throughput <sup>[3]</sup> UART intended to be used as a console. It needs to be enabled before it can be used. It is also recommended that the correct GPIO function mode is selected before enabling the mini UART (see [Chapter 5](#)).

The mini UART has the following features:

- 7-bit or 8-bit operation
- 1 start and 1 stop bit
- No parities



- Break generation
- 8 symbols deep FIFOs for receive and transmit
- SW controlled RTS, SW readable CTS
- Auto flow control with programmable FIFO level
- 16550 *like* registers
- Baudrate derived from system clock

This is a **mini** UART and it does NOT have the following capabilities:

- Break detection
- Framing errors detection
- Parity bit
- Receive Time-out interrupt
- DCD, DSR, DTR or RI signals

The implemented UART is **not** a 16550 compatible UART. However as far as possible the first 8 control and status registers are laid out **like** a 16550 UART. All 16550 register bits which are not supported can be written but will be ignored and read back as 0. All control bits for simple UART receive/transmit operations are available.

### 2.2.1. Mini UART implementation details

The UART1\_CTS and UART1\_RX inputs are synchronised and will take 2 system clock cycles before they are processed.

The module does not check for any framing errors. After receiving a start bit and 8 (or 7) data bits the receiver waits for one half-bit time and then starts scanning for the next start bit. The mini UART does *not* check if the stop bit is high or wait for the stop bit to appear. As a result of this, a UART1\_RX input line which is continuously low (a break condition or an error in connection or GPIO setup) causes the receiver to continuously receive 0x00 symbols.

The mini UART uses 8-times oversampling. The Baudrate can be calculated from:

$$\text{baudrate} = \frac{\text{system\_clock\_freq}}{8 * (\text{baudrate\_reg} + 1)}$$

If the system clock is 250 MHz and the baud register is zero the baudrate is 31.25 Mega baud. (25 Mbits/sec or 3.125 Mbytes/sec). The lowest baudrate with a 250 MHz system clock is 476 Baud.

When writing to the data register only the LS 8 bits are taken. All other bits are ignored.  
When reading from the data register only the LS 8 bits are valid. All other bits are zero.

### 2.2.2. Mini UART register details

#### AUX\_MU\_IO\_REG Register



## Synopsis

The AUX\_MU\_IO\_REG register is primarily used to write data to and read data from the UART FIFOs. If the DLAB bit in the line control register is set this register gives access to the LS 8 bits of the baud rate. (Note: there is easier access to the baud rate register in AUX\_MU\_BAUD\_REG)

| Bit(s) | Field Name                                     | Description   | Type | Reset |
|--------|--|---|------|-------|
| 31:8   |  | Reserved, write zero, read as don't care  |      |       |
| 7:0    | LS 8 bits<br>Baudrate<br>read/write,<br>DLAB=1 | Access to the LS 8 bits of the 16-bit baudrate register.<br>(Only if bit 7 of the line control register (DLAB bit) is set)              | RW   | 0     |
| 7:0    | Transmit data<br>write, DLAB=0                 | Data written is put in the transmit FIFO (Provided it is not full)<br>(Only if bit 7 of the line control register (DLAB bit) is clear)  | WO   | 0     |
| 7:0    | Receive data<br>read, DLAB=0                   | Data read is taken from the receive FIFO (Provided it is not empty)<br>(Only if bit 7 of the line control register (DLAB bit) is clear) | RO   | 0     |

Table 5. AUX\_MU\_IO\_REG Register

## AUX\_MU\_IER\_REG Register

### Synopsis

The AUX\_MU\_IER\_REG register is primarily used to enable interrupts. If the DLAB bit in the line control register is set this register gives access to the MS 8 bits of the baud rate. (Note: there is easier access to the baud rate register in AUX\_MU\_BAUD\_REG)

| Bit(s) | Field Name                                     | Description   | Type | Reset |
|--------|--|---|------|-------|
| 31:8   |  | Reserved, write zero, read as don't care  |      |       |
| 7:0    | MS 8 bits<br>Baudrate<br>read/write,<br>DLAB=1 | Access to the MS 8 bits of the 16-bit baudrate register.<br>(Only if bit 7 of the line control register (DLAB bit) is set)                                      | RW   | 0     |
| 7:2    |  | Reserved, write zero, read as don't care<br><i>Some of these bits have functions in a 16550 compatible UART but are ignored here</i>                            |      |       |
| 1      | Enable receive<br>interrupt<br>(DLAB=0)        | If this bit is set the interrupt line is asserted whenever the receive FIFO holds at least 1 byte.<br>If this bit is clear no receive interrupts are generated. | RW   | 0     |



| Bit(s) | Field Name                            | Description  | Type | Reset |
|--------|---------------------------------------|--|------|-------|
| 0      | Enable transmit interrupt<br>(DLAB=0) | If this bit is set the interrupt line is asserted whenever the transmit FIFO is empty.<br>If this bit is clear no transmit interrupts are generated. | RW   | 0     |

Table 6. AUX\_MU\_IER\_REG Register

## AUX\_MU\_IIR\_REG Register

### Synopsis

The AUX\_MU\_IIR\_REG register shows the interrupt status.

It also has two FIFO enable status bits and (when writing) FIFO clear bits.

| Bit(s) | Field Name  | Description   | Type | Reset |
|--------|---|---|------|-------|
| 31:8   |   | Reserved, write zero, read as don't care  |      |       |
| 7:6    | FIFO enables  | Both bits always read as 1 as the FIFOs are always enabled  | RO   | 11    |
| 5:4    | -   | Always read as zero   | RO   | 00    |
| 3      | -   | Always read as zero as the mini UART has no timeout function  | RO   | 0     |
| 2:1    | READ:<br>Interrupt ID bits<br>WRITE:<br>FIFO clear bits | On read this register shows the interrupt ID bit<br>00 : No interrupts<br>01 : Transmit holding register empty<br>10 : Receiver holds valid byte<br>11 : <Not possible><br>On write:<br>Writing with bit 1 set will clear the receive FIFO<br>Writing with bit 2 set will clear the transmit FIFO | RW   | 00    |
| 0      | Interrupt pending                                       | This bit is clear whenever an interrupt is pending  | RO   | 1     |

Table 7. AUX\_MU\_IIR\_REG Register

## AUX\_MU\_LCR\_REG Register

### Synopsis

The AUX\_MU\_LCR\_REG register controls the line data format and gives access to the baudrate register

| Bit(s) | Field Name  | Description   | Type | Reset |
|--------|-------------|---|------|-------|
| 31:8   |             | Reserved, write zero, read as don't care  |      |       |
| 7      | DLAB access | If set the first two Mini UART registers give access to the Baudrate register. During operation this bit must be cleared. | RW   | 0     |



| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 6      | Break      | If set high the UART1_TX line is pulled low continuously.<br>If held for at least 12 bits times that will indicate a break condition. | RW   | 0     |
| 5:1    |            | Reserved, write zero, read as don't care<br><i>Some of these bits have functions in a 16550 compatible UART but are ignored here</i>  |      | 0     |
| 0      | Data size  | If clear the UART works in 7-bit mode<br>If set the UART works in 8-bit mode  | RW   | 0     |

Table 8. AUX\_MU\_LCR\_REG Register

## AUX\_MU\_MCR\_REG Register

### Synopsis

The AUX\_MU\_MCR\_REG register controls the 'modem' signals.

| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 31:8   |            | Reserved, write zero, read as don't care   |      |       |
| 7:2    |            | Reserved, write zero, read as don't care<br><i>Some of these bits have functions in a 16550 compatible UART but are ignored here</i>   |      | 0     |
| 1      | RTS        | If clear the UART1_RTS line is high<br>If set the UART1_RTS line is low<br>This bit is ignored if the RTS is used for auto-flow control. See the Mini UART Extra Control register description) | RW   | 0     |
| 0      |            | Reserved, write zero, read as don't care<br><i>This bit has a function in a 16550 compatible UART but is ignored here</i>  |      | 0     |

Table 9. AUX\_MU\_MCR\_REG Register

## AUX\_MU\_LSR\_REG Register

### Synopsis

The AUX\_MU\_LSR\_REG register shows the data status.

| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 31:8   |            | Reserved, write zero, read as don't care  |      |       |
| 7      |            | Reserved, write zero, read as don't care<br><i>This bit has a function in a 16550 compatible UART but is ignored here</i> |      | 0     |



| Bit(s) | Field Name        | Description   | Type | Reset |
|--------|-------------------|---|------|-------|
| 6      | Transmitter idle  | This bit is set if the transmit FIFO is empty and the transmitter is idle. (Finished shifting out the last bit).  | RO   | 1     |
| 5      | Transmitter empty | This bit is set if the transmit FIFO can accept at least one byte.  | RO   | 0     |
| 4:2    |                   | Reserved, write zero, read as don't care<br><i>Some of these bits have functions in a 16550 compatible UART but are ignored here</i>  |      | 0     |
| 1      | Receiver Overrun  | This bit is set if there was a receiver overrun. That is: one or more characters arrived whilst the receive FIFO was full. The newly arrived characters have been discarded. This bit is cleared each time this register is read. To do a non-destructive read of this overrun bit use the Mini UART Extra Status register. | RC   | 0     |
| 0      | Data ready        | This bit is set if the receive FIFO holds at least 1 symbol.  | RO   | 0     |

Table 10. AUX\_MU\_LSR\_REG Register

## AUX\_MU\_MSR\_REG Register

### Synopsis

The AUX\_MU\_MSR\_REG register shows the 'modem' status.

| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 31:8   |            | Reserved, write zero, read as don't care   |      |       |
| 7:5    |            | Reserved, write zero, read as don't care<br><i>Some of these bits have functions in a 16550 compatible UART but are ignored here</i> |      | 0     |
| 4      | CTS status | This bit is the inverse of the UART1_CTS input. Thus:<br>If set the UART1_CTS pin is low<br>If clear the UART1_CTS pin is high       | RO   | 1     |
| 3:0    |            | Reserved, write zero, read as don't care<br><i>Some of these bits have functions in a 16550 compatible UART but are ignored here</i> |      | 0     |

Table 11. AUX\_MU\_MSR\_REG Register

## AUX\_MU\_SCRATCH Register

### Synopsis

The AUX\_MU\_SCRATCH is a single byte of temporary storage.



| Bit(s) | Field Name | Description                              | Type | Reset |
|--------|------------|--|------|-------|
| 31:8   |            | Reserved, write zero, read as don't care |      |       |
| 7:0    | Scratch    | A byte of temporary storage              | RW   | 0     |

Table 12. AUX\_MU\_SCRATCH Register

## AUX\_MU\_CNTL\_REG Register

### Synopsis

The AUX\_MU\_CNTL\_REG provides access to some extra useful and nice features not found on a normal 16550 UART.

| Bit(s) | Field Name                                  | Description  | Type | Reset |
|--------|---|--|------|-------|
| 31:8   |   | Reserved, write zero, read as don't care   |      |       |
| 7      | CTS assert level                            | This bit allows one to invert the CTS auto flow operation polarity.<br>If set the CTS auto flow assert level is low*<br>If clear the CTS auto flow assert level is high*   | RW   | 0     |
| 6      | RTS assert level                            | This bit allows one to invert the RTS auto flow operation polarity.<br>If set the RTS auto flow assert level is low*<br>If clear the RTS auto flow assert level is high*   | RW   | 0     |
| 5:4    | RTS AUTO flow level                         | These two bits specify at what receiver FIFO level the RTS line is de-asserted in auto-flow mode.<br>00 : De-assert RTS when the receive FIFO has 3 empty spaces left.<br>01 : De-assert RTS when the receive FIFO has 2 empty spaces left.<br>10 : De-assert RTS when the receive FIFO has 1 empty space left.<br>11 : De-assert RTS when the receive FIFO has 4 empty spaces left. | RW   | 0     |
| 3      | Enable transmit Auto flow-control using CTS | If this bit is set the transmitter will stop if the CTS line is de-asserted.<br>If this bit is clear the transmitter will ignore the status of the CTS line  | RW   | 0     |
| 2      | Enable receive Auto flow-control using RTS  | If this bit is set the RTS line will de-assert if the receive FIFO reaches its 'auto flow' level. In fact the RTS line will behave as an RTR (Ready To Receive) line.<br>If this bit is clear the RTS line is controlled by the AUX_MU_MCR_REG register bit 1.   | RW   | 0     |
| 1      | Transmitter enable                          | If this bit is set the mini UART transmitter is enabled.<br>If this bit is clear the mini UART transmitter is disabled   | RW   | 1     |





| Bit(s) | Field Name      | Description  | Type | Reset |
|--------|-----------------|--|------|-------|
| 0      | Receiver enable | If this bit is set the mini UART receiver is enabled.<br>If this bit is clear the mini UART receiver is disabled | RW   | 1     |

Table 13. AUX\_MU\_CNTL\_REG Register

### Receiver enable

If this bit is clear no new symbols will be accepted by the receiver. Any symbols in progress of reception will be finished.

### Transmitter enable

If this bit is clear no new symbols will be sent by the transmitter. Any symbols in progress of transmission will be finished.

### Auto flow control

Automatic flow control can be enabled independent for the receiver and the transmitter.

CTS auto flow control impacts the transmitter only. The transmitter will not send out new symbols when the CTS line is de-asserted. Any symbols in progress of transmission when the CTS line becomes de-asserted will be finished.

RTS auto flow control impacts the receiver only. In fact the name RTS for the control line is incorrect and should be RTR (Ready to Receive). The receiver will de-assert the RTS (RTR) line when its receive FIFO has a number of empty spaces left. Normally 3 empty spaces should be enough.

If looping back a mini UART using full auto flow control the logic is fast enough to allow the RTS auto flow level of '10' (De-assert RTS when the receive FIFO has 1 empty space left).

### Auto flow polarity

To offer full flexibility the polarity of the CTS and RTS (RTR) lines can be programmed. This should allow the mini UART to interface with any existing hardware flow control available.

## AUX\_MU\_STAT\_REG Register

### Synopsis

The AUX\_MU\_STAT\_REG provides a lot of useful information about the internal status of the mini UART not found on a normal 16550 UART.

| Bit(s) | Field Name               | Description  | Type | Reset |
|--------|--------------------------|--|------|-------|
| 31:28  |                          | Reserved, write zero, read as don't care   |      |       |
| 27:24  | Transmit FIFO fill level | These bits shows how many symbols are stored in the transmit FIFO<br>The value is in the range 0-8 | RO   | 0     |
| 23:20  |                          | Reserved, write zero, read as don't care   |      |       |



| Bit(s) | Field Name                               | Description   | Type | Reset |
|--------|--|---|------|-------|
| 19:16  | Receive FIFO fill level                  | These bits shows how many symbols are stored in the receive FIFO<br>The value is in the range 0-8   | RO   | 0     |
| 15:10  | Reserved, write zero, read as don't care |   |      |       |
| 9      | Transmitter done                         | This bit is set if the transmitter is idle and the transmit FIFO is empty.<br>It is a logic AND of bits 3 and 8   | RO   | 1     |
| 8      | Transmit FIFO is empty                   | If this bit is set the transmitter FIFO is empty. Thus it can accept 8 symbols.   | RO   | 1     |
| 7      | CTS line                                 | This bit shows the status of the UART1_CTS line.  | RO   | 0     |
| 6      | RTS status                               | This bit shows the status of the UART1_RTS line.  | RO   | 0     |
| 5      | Transmit FIFO is full                    | This is the inverse of bit 1  | RO   | 0     |
| 4      | Receiver overrun                         | This bit is set if there was a receiver overrun. That is: one or more characters arrived whilst the receive FIFO was full. The newly arrived characters have been discarded. This bit is cleared each time the AUX_MU_LSR_REG register is read. | RO   | 0     |
| 3      | Transmitter is idle                      | If this bit is set the transmitter is idle.<br>If this bit is clear the transmitter is busy.  | RO   | 1     |
| 2      | Receiver is idle                         | If this bit is set the receiver is idle.<br>If this bit is clear the receiver is busy.<br>This bit can change unless the receiver is disabled   | RO   | 1     |
| 1      | Space available                          | If this bit is set the mini UART transmitter FIFO can accept at least one more symbol.<br>If this bit is clear the mini UART transmitter FIFO is full   | RO   | 0     |
| 0      | Symbol available                         | If this bit is set the mini UART receive FIFO contains at least 1 symbol<br>If this bit is clear the mini UART receiver FIFO is empty   | RO   | 0     |

Table 14. AUX\_MU\_STAT\_REG Register

### Receiver is idle

This bit is only useful if the receiver is disabled. The normal use is to disable the receiver, then check (or wait) until the bit is set. Now you can be sure that no new symbols will arrive (e.g. now you can change the baudrate...).

### Transmitter is idle

This bit tells if the transmitter is idle. Note that the bit will set only for a short time if the transmit FIFO contains data. Normally you want to use bit 9: Transmitter done.



## RTS status

This bit is useful only in receive Auto flow-control mode as it shows the status of the RTS line.

## AUX\_MU\_BAUD\_REG Register

### Synopsis

The AUX\_MU\_BAUD\_REG register allows direct access to the 16-bit wide baudrate counter.

| Bit(s) | Field Name | Description                              | Type | Reset |
|--------|------------|--|------|-------|
| 31:16  |            | Reserved, write zero, read as don't care |      |       |
| 15:0   | Baudrate   | mini UART baudrate counter               | RW   | 0     |

Table 15. AUX\_MU\_BAUD\_REG Register

This is the same register as is accessed using the DLAB bit and the first two registers, but much easier to access.

## 2.3. Universal SPI Master (2x)

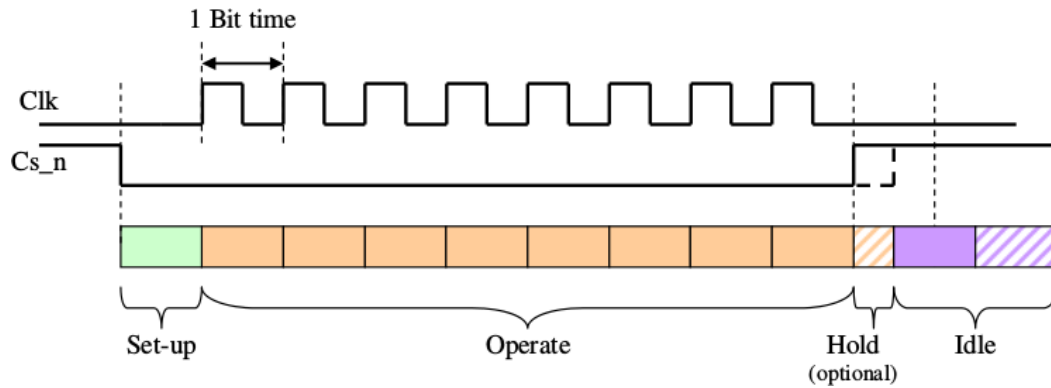
The two universal SPI masters are secondary low throughput <sup>[4]</sup> SPI interfaces. Like the mini UART the devices need to be enabled before they can be used. Each SPI master has the following features:

- Single-beat bit length between 1 and 32 bits
- Single-beat variable bit length between 1 and 24 bits
- Multi-beat infinite bit length
- 3 independent chip selects per master
- 4 entries 32-bit wide transmit and receive FIFOs
- Data out on rising or falling clock edge
- Data in on rising or falling clock edge
- Clock inversion (idle high or idle low)
- Wide clocking range
- Programmable data out hold time
- Shift in/out MS or LS bit first

A major issue with an SPI interface is that there is no SPI standard in any form. Because the SPI interface has been around for a long time some pseudo-standard rules have appeared mostly when interfacing with memory devices. The universal SPI master has been developed to work even with the most 'non-standard' SPI devices.

### 2.3.1. SPI implementation details

The following diagrams shows a typical SPI access cycle. In this case we have 8 SPI clocks.



One bit-time before any clock edge changes the **Cs\_n** will go low. This makes sure that the MOSI signal has a full bit-time of set-up against any changing clock edges.

The operation normally ends after the last clock cycle. Note that at the end there is one half-bit time where the clock does not change but which still is part of the operation cycle.

There is an option to add a half-bit cycle hold time. This makes sure that any MISO data has at least a full SPI bit-time to arrive. (Without this hold time, data clocked out of the SPI device on the last clock edge would have only half a bit-time to arrive).

Lastly there is a guarantee of at least a full bit-time where the SPI chip select is high. A longer **Cs\_n** high period can be programmed for another 1-7 cycles.

The SPI clock frequency is:

$$SPIx\_CLK = \frac{system\_clock\_freq}{2 * (speed\_field + 1)}$$

If the system clock is 250 MHz and the speed field is zero the SPI clock frequency is 125 MHz. The practical SPI clock will be lower as the I/O pads can not transmit or receive signals at such high speed. The lowest SPI clock frequency with a 250 MHz system clock is 30.5 KHz.

The hardware has an option to add hold time to the MOSI signal against the SPI clk. This is again done using the system clock. So a 250 MHz system clock will add hold times in units of 4 ns. Hold times of 0, 1, 4 and 7 system clock cycles can be used. (So at 250MHz an additional hold time of 0, 4, 16 and 28 ns can be achieved). The hold time is **additional** to the normal output timing as specified in the data sheet.

## 2.3.2. Interrupts

The SPI block has two interrupts: TX FIFO is empty, SPI is Idle.

### TX FIFO is empty

This interrupt will be asserted as soon as the last entry has been read from the transmit FIFO. At that time the interface will still be busy shifting out that data. This also implies that the receive FIFO will not yet contain the last received data. It is possible at that time to fill the TX FIFO again and read the receive FIFO entries which have been received. There is a RX FIFO level field which tells you exactly how many words are in the receive FIFO. In general at that time the receive FIFO should contain the number of TX items minus one (the last one still being received). Note that there is no "receive FIFO



full" interrupt as the number of entries received can never be more than the number of entries transmitted.

### SPI is Idle

This interrupt will be asserted when the transmit FIFO is empty and the SPI block has finished all actions (including the CS-high time). By this time the receive FIFO will have received all data as well.

### 2.3.3. Long bit streams

The SPI module works in bursts of up to 32 bits. Some SPI devices require data which is longer than 32 bits. To do this the user must make use of the two different data TX addresses: TX data written to one address causes the CS to remain asserted. TX data written to the other address causes the CS to be de-asserted at the end of the transmit cycle. So in order to exchange 96 bits you do the following: Write the first two data words to one address, then write the third word to the other address.

### 2.3.4. SPI register details

#### AUX\_SPI1\_CNTL0\_REG & AUX\_SPI2\_CNTL0\_REG Registers

##### Synopsis

The AUX\_SPIx\_CNTL0\_REG registers control many features of the SPI interfaces.

| Bit(s) | Field Name      | Description   | Type | Reset |
|--------|-----------------|---|------|-------|
| 31:20  | Speed           | Sets the SPI clock speed. $\text{spi\_clk\_freq} = \text{system\_clock\_freq}/2*(\text{speed}+1)$   | RW   | 0     |
| 19:17  | Chip Selects    | The pattern output on the CS pins when active.  | RW   | 111   |
| 16     | Post-input mode | If set the SPI input works in post-input mode.<br>For details see text further down   | RW   | 0     |
| 15     | Variable CS     | If 1 the SPI takes the CS pattern and the data from the TX FIFO<br>If 0 the SPI takes the CS pattern from bits 17-19 of this register<br>Set this bit only if bit 14 (variable width) is also set           | RW   | 0     |
| 14     | Variable width  | If 1 the SPI takes the shift length and the data from the TX FIFO<br>If 0 the SPI takes the shift length from bits 0-5 of this register   | RW   | 0     |
| 13:12  | DOUT Hold time  | Controls the extra DOUT hold time in system clock cycles.<br>00 : No extra hold time<br>01 : 1 system clock extra hold time<br>10 : 4 system clocks extra hold time<br>11 : 7 system clocks extra hold time | RW   | 0     |



| Bit(s) | Field Name             | Description   | Type | Reset |
|--------|------------------------|---|------|-------|
| 11     | Enable                 | Enables the SPI interface. Whilst disabled the FIFOs can still be written to or read from<br>This bit should be 1 during normal operation.  | RW   | 0     |
| 10     | In rising              | If 1 data is clocked in on the rising edge of the SPI clock<br>If 0 data is clocked in on the falling edge of the SPI clock                 | RW   | 0     |
| 9      | Clear FIFOs            | If 1 the receive and transmit FIFOs are held in reset (and thus flushed.)<br>This bit should be 0 during normal operation.                  | RW   | 0     |
| 8      | Out rising             | If 1 data is clocked out on the rising edge of the SPI clock<br>If 0 data is clocked out on the falling edge of the SPI clock               | RW   | 0     |
| 7      | Invert SPI CLK         | If 1 the 'idle' clock line state is high.<br>If 0 the 'idle' clock line state is low.   | RW   | 0     |
| 6      | Shift out MS bit first | If 1 the data is shifted out starting with the MS bit. (bit 31 or bit 23)<br>If 0 the data is shifted out starting with the LS bit. (bit 0) | RW   | 0     |
| 5:0    | Shift length           | Specifies the number of bits to shift<br>This field is ignored when using 'variable width' mode   | RW   | 0     |

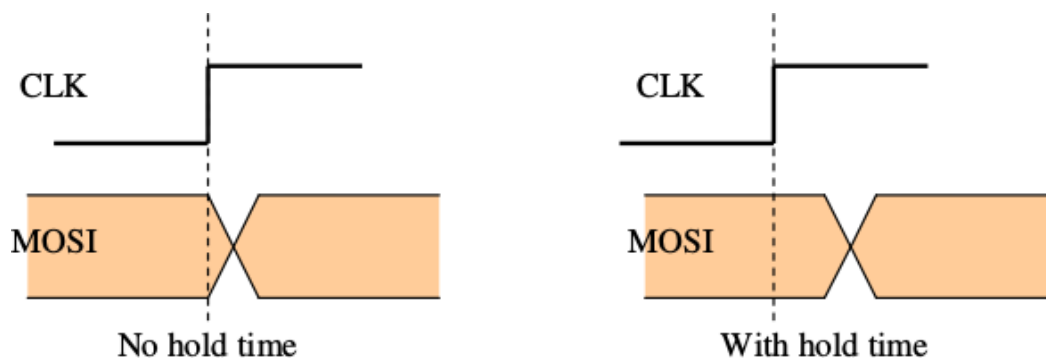
Table 16. AUX\_SPI1\_CNTLO\_REG & AUX\_SPI2\_CNTLO\_REG Registers

### Invert SPI CLK

Changing this bit will immediately change the polarity of the SPI clock output. It is recommended to not do this when the CS is active, as the connected devices will see this as a clock change.

### DOUT hold time

Because the interface runs off fast silicon the MOSI hold time against the clock will be very short. This can cause considerable problems on SPI slaves. To make it easier for the slave to see the data the hold time of the MOSI out against the SPI clock out is programmable.



### Variable width

In this mode the shift length is taken from the transmit FIFO. The transmit data bits 28:24 are used as shift length and the data bits 23:0 are the actual transmit data. If the option 'shift MS out first' is



selected the first bit shifted out will be bit 23. The receive data will arrive as normal.

## Variable CS

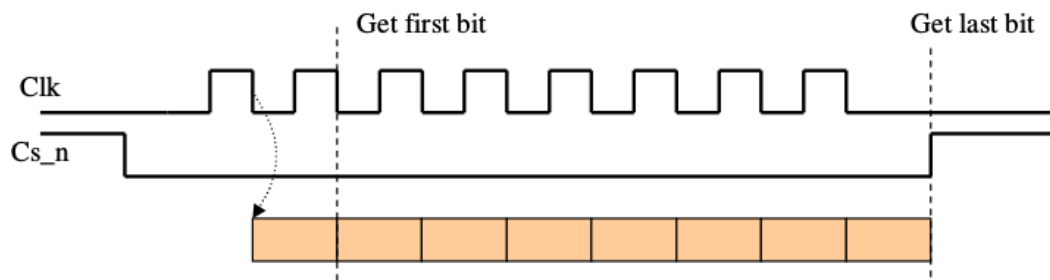
This mode is used together with the variable width mode. In this mode the CS pattern is taken from the transmit FIFO. The transmit data bits 31:29 are used as CS and the data bits 23:0 are the actual transmit data. This allows the CPU to write to different SPI devices without having to change the CS bits. However the data length is limited to 24 bits.

## Post-input mode

Some rare SPI devices output data on the falling clock edge which then has to be picked up on the next falling clock edge. There are two problems with this:

1. On the very first falling clock edge there is no valid data arriving
2. After the last clock edge there is one more 'dangling' bit to pick up

The post-input mode is specifically to deal with this sort of data. If the post-input mode bit is set, the data arriving at the first falling clock edge is ignored. Then after the last falling clock edge the CS remains asserted and after a full bit-time the last data bit is picked up. The following figure shows this behaviour:



In this mode the CS will go high 1 full SPI clock cycle after the last clock edge. This guarantees a full SPI clock cycle time for the data to settle and arrive at the MISO input.

## AUX\_SPI1\_CNTL1\_REG & AUX\_SPI2\_CNTL1\_REG Registers

### Synopsis

The AUX\_SPIx\_CNTL1\_REG registers control more features of the SPI interfaces.

| Bit(s) | Field Name            | Description   | Type | Reset |
|--------|-----------------------|---|------|-------|
| 31:11  |                       | Reserved, write zero, read as don't care  |      |       |
| 10:8   | CS high time          | Additional SPI clock cycles where the CS is high.   | RW   | 0     |
| 7      | TX empty IRQ          | If 1 the interrupt line is high when the transmit FIFO is empty   | RW   | 0     |
| 6      | Done IRQ              | If 1 the interrupt line is high when the interface is idle  | RW   | 0     |
| 5:2    |                       | Reserved, write zero, read as don't care  |      |       |
| 1      | Shift in MS bit first | If 1 the data is shifted in starting with the MS bit. (bit 15)<br>If 0 the data is shifted in starting with the LS bit. (bit 0) | RW   | 0     |



| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 0      | Keep input | If 1 the receiver shift register is NOT cleared. Thus new data is concatenated to old data.<br>If 0 the receiver shift register is cleared before each transaction. | RW   | 0     |

Table 17. *AUX\_SPI1\_CNTL1\_REG & AUX\_SPI2\_CNTL1\_REG Registers*

### Keep input

Setting the 'Keep input' bit will prevent the input shift register being cleared between transactions. However the contents of the shift register is still written to the receive FIFO at the end of each transaction. E.g. if you receive two 8-bit values 0x81 followed by 0x46 the receive FIFO will contain: 0x0081 in the first entry and 0x8146 in the second entry. This mode may save CPU time concatenating bits (4 bits followed by 12 bits).

### CS high time

The SPI CS will always be high for at least 1 SPI clock cycle. Some SPI devices need more time to process the data. This field will set a longer CS-high time. So the actual CS high time is (CS\_high\_time + 1) (in SPI clock cycles).

## AUX\_SPI1\_STAT\_REG & AUX\_SPI2\_STAT\_REG Registers

### Synopsis

The AUX\_SPIx\_STAT\_REG registers show the status of the SPI interfaces.

| Bit(s) | Field Name    | Description   | Type | Reset |
|--------|---------------|---|------|-------|
| 31:28  |               | Reserved, write zero, read as don't care  |      |       |
| 27:24  | TX FIFO level | The number of data units in the transmit data FIFO  | RO   | 0     |
| 23:20  |               | Reserved, write zero, read as don't care  |      |       |
| 19:16  | RX FIFO level | The number of data units in the receive data FIFO.  | RO   | 0     |
| 15:11  |               | Reserved, write zero, read as don't care  |      |       |
| 10     | TX Full       | If 1 the transmit FIFO is full<br>If 0 the transmit FIFO can accept at least 1 data unit. | RO   | 0     |
| 9      | TX Empty      | If 1 the transmit FIFO is empty<br>If 0 the transmit FIFO holds at least 1 data unit.     | RO   | 0     |
| 8      | RX Full       | If 1 the receiver FIFO is full<br>If 0 the receiver FIFO can accept at least 1 data unit. | RO   | 0     |
| 7      | RX Empty      | If 1 the receiver FIFO is empty<br>If 0 the receiver FIFO holds at least 1 data unit.     | RO   | 0     |
| 6      | Busy          | Indicates the module is busy transferring data.   | RO   | 0     |
| 5:0    | Bit count     | The number of bits still to be processed. Starts with 'shift-length' and counts down.     | RO   | 0     |





Table 18. AUX\_SPI1\_STAT\_REG & AUX\_SPI2\_STAT\_REG Registers

### Busy

This status bit indicates if the module is busy. It will be clear when the TX FIFO is empty and the module has finished all activities, including waiting the minimum CS high time.

## AUX\_SPI1\_PEEK\_REG & AUX\_SPI2\_PEEK\_REG Registers

### Synopsis

The AUX\_SPIx\_PEEK\_REG registers show received data of the SPI interfaces.

| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 31:16  |            | Reserved, write zero, read as don't care  |      |       |
| 15:0   | Data       | Reads from this address will show the top entry from the receive FIFO, but the data is not taken from the FIFO. This provides a means of inspecting the data but not removing it from the FIFO. | RO   | 0     |

Table 19. AUX\_SPI1\_PEEK\_REG & AUX\_SPI2\_PEEK\_REG Registers

## AUX\_SPI1\_IO\_REG & AUX\_SPI2\_IO\_REG Registers

### Synopsis

The AUX\_SPIx\_IO\_REG registers are the primary data port of the SPI interfaces.

These four addresses all write to the same FIFO.

**Writing to any of these addresses causes the SPI CS<sub>n</sub> pins to be de-asserted at the end of the access.**

| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 31:16  |            | Reserved, write zero, read as don't care   |      |       |
| 15:0   | Data       | Writes to this address range end up in the transmit FIFO. Data is lost when writing whilst the transmit FIFO is full.<br>Reads from this address will take the top entry from the receive FIFO. Reading whilst the receive FIFO is empty will return the last data received. | RW   | 0     |

Table 20. AUX\_SPI1\_IO\_REG & AUX\_SPI2\_IO\_REG Registers

## AUX\_SPI1\_TXHOLD\_REG & AUX\_SPI2\_TXHOLD\_REG Registers

### Synopsis

The AUX\_SPIx\_TXHOLD\_REG registers are the extended CS port of the SPI interfaces.

These four addresses all write to the same FIFO.

**Writing to these addresses causes the SPI CS<sub>n</sub> pins to remain asserted at the end of the access.**

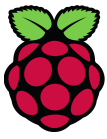


| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 31:16  |            | Reserved, write zero, read as don't care   |      |       |
| 15:0   | Data       | Writes to this address range end up in the transmit FIFO. Data is lost when writing whilst the transmit FIFO is full.<br>Reads from this address will take the top entry from the receive FIFO. Reading whilst the receive FIFO is empty will return the last data received. | RW   | 0     |

*Table 21. AUX\_SPI1\_TXHOLD\_REG & AUX\_SPI2\_TXHOLD\_REG Registers*

[3] The UART itself has no throughput limitations, in fact it can run up to 32 **Mega** baud. But doing so requires significant CPU involvement as it has shallow FIFOs and no DMA support.

[4] Again the SPIs themselves have no throughput limitations, in fact they can run with an SPI clock of 125 MHz. But doing so requires significant CPU involvement as they have shallow FIFOs and no DMA support.



## Chapter 3. BSC

### 3.1. Overview

The Broadcom Serial Control (BSC) controller is a master, fast-mode (400Kb/s) BSC controller. The Broadcom Serial Control bus is a proprietary bus compliant with the Philips® I<sup>2</sup>C bus/interface version 2.1 January 2000.

- I<sup>2</sup>C single master only operation (supports clock stretching wait states)
- Both 7-bit and 10-bit addressing is supported
- Timing completely software controllable via registers
- The BSC controller in the BCM2711 fixes the clock-stretching bug that was present in BCM283x devices

### 3.2. Register View

The BSC controller has eight memory-mapped registers. All accesses are assumed to be 32-bit. Note that the BSC2 and BSC7 masters are dedicated for use by the HDMI interfaces and should not be accessed by user programs.

There are eight BSC masters inside BCM2711. The user-accessible register addresses start from

- BSC0: 0x7E20 5000
- BSC1: 0x7E80 4000
- BSC3: 0x7E20 5600
- BSC4: 0x7E20 5800
- BSC5: 0x7E20 5A80
- BSC6: 0x7E20 5C00

The table below shows the addresses of the I<sup>2</sup>C registers, where the address is an offset from one of the base addresses listed above.

| Address Offset | Register Name | Description   | Size |
|----------------|---------------|---------------|------|
| 0x00           | C             | Control       | 32   |
| 0x04           | S             | Status        | 32   |
| 0x08           | DLEN          | Data Length   | 32   |
| 0x0C           | A             | Slave Address | 32   |
| 0x10           | FIFO          | Data FIFO     | 32   |
| 0x14           | DIV           | Clock Divider | 32   |



| Address Offset | Register Name        | Description           | Size |
|----------------|----------------------|-----------------------|------|
| 0x18           | <a href="#">DEL</a>  | Data Delay            | 32   |
| 0x1C           | <a href="#">CLKT</a> | Clock Stretch Timeout | 32   |

Table 22. I2C Address Map

## C Register

### Synopsis

The control register is used to enable interrupts, clear the FIFO, define a read or write operation and start a transfer.

The READ field specifies the type of transfer.

The CLEAR field is used to clear the FIFO. Writing to this field is a one-shot operation which will always read back as zero. The CLEAR bit can set at the same time as the start transfer bit, and will result in the FIFO being cleared just prior to the start of transfer. Note that clearing the FIFO during a transfer will result in the transfer being aborted.

The ST field starts a new BSC transfer. This is a one-shot action, and so the bit will always read back as 0.

The INTD field enables interrupts at the end of a transfer - the DONE condition. The interrupt remains active until the DONE condition is cleared by writing a 1 to the I2CS.DONE field. Writing a 0 to the INTD field disables interrupts on DONE.

The INTT field enables interrupts whenever the FIFO is  $\frac{1}{4}$  or more empty and needs writing (i.e. during a write transfer) - the TXW condition. The interrupt remains active until the TXW condition is cleared by writing sufficient data to the FIFO to complete the transfer. Writing a 0 to the INTT field disables interrupts on TXW.

The INTR field enables interrupts whenever the FIFO is  $\frac{3}{4}$  or more full and needs reading (i.e. during a read transfer) - the RXR condition. The interrupt remains active until the RXW condition is cleared by reading sufficient data from the FIFO. Writing a 0 to the INTR field disables interrupts on RXR.

The I2CEN field enables BSC operations. If this bit is 0 then transfers will not be performed. All register accesses are still permitted however.

| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 31:16  |            | Reserved - Write as 0, read as don't care   |      |       |
| 15     | I2CEN      | I2C Enable<br>0 = BSC controller is disabled<br>1 = BSC controller is enabled                               | RW   | 0x0   |
| 14:11  |            | Reserved - Write as 0, read as don't care   |      |       |
| 10     | INTR       | Interrupt on RX<br>0 = Don't generate interrupts on RXR condition.<br>1 = Generate interrupt while RXR = 1. | RW   | 0x0   |
| 9      | INTT       | Interrupt on TX<br>0 = Don't generate interrupts on TXW condition.<br>1 = Generate interrupt while TXW = 1. | RW   | 0x0   |



| Bit(s) | Field Name                                | Description   | Type | Reset |
|--------|---|---|------|-------|
| 8      | INTD                                      | Interrupt on DONE<br>0 = Don't generate interrupts on DONE condition.<br>1 = Generate interrupt while DONE = 1.   | RW   | 0x0   |
| 7      | ST  | Start Transfer<br>0 = No action.<br>1 = Start a new transfer. One-shot operation. Read back as 0.   | W1SC | 0x0   |
| 6      | Reserved - Write as 0, read as don't care |   |      |       |
| 5:4    | CLEAR                                     | FIFO Clear<br>00 = No action.<br>x1 = Clear FIFO. One-shot operation.<br>1x = Clear FIFO. One-shot operation.<br>If CLEAR and ST are both set in the same operation, the FIFO is cleared before the new frame is started. Read back as 0.<br>Note: 2 bits are used to maintain compatibility with the previous version. | W1SC | 0x0   |
| 3:1    | Reserved - Write as 0, read as don't care |   |      |       |
| 0      | READ                                      | Read Transfer<br>0 = Write Packet Transfer.<br>1 = Read Packet Transfer.  | RW   | 0x0   |

Table 23. C Register

## S Register

### Synopsis

The status register is used to record activity status, errors and interrupt requests.

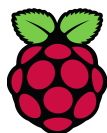
The TA field indicates the activity status of the BSC controller. This read-only field returns a 1 when the controller is in the middle of a transfer and a 0 when idle.

The DONE field is set when the transfer completes. The DONE condition can be used with I2CC.INTD to generate an interrupt on transfer completion. The DONE field is reset by writing a 1, writing a 0 to the field has no effect.

The read-only TXW bit is set during a write transfer and the FIFO is less than  $\frac{1}{4}$  full and needs writing. Writing sufficient data (i.e. enough data to either fill the FIFO more than  $\frac{1}{4}$  full or complete the transfer) to the FIFO will clear the field. When the I2CC.INTT control bit is set, the TXW condition can be used to generate an interrupt to write more data to the FIFO to complete the current transfer. If the I2C controller runs out of data to send, it will wait for more data to be written into the FIFO.

The read-only RXR field is set during a read transfer and the FIFO is  $\frac{3}{4}$  or more full and needs reading. Reading sufficient data to bring the depth below  $\frac{3}{4}$  will clear the field.

When I2CC.INTR control bit is set, the RXR condition can be used to generate an interrupt to read data from the FIFO before it becomes full. In the event that the FIFO does become full, all I2C operations will stall until data is removed from the FIFO.



The read-only TXD field is set when the FIFO has space for at least one byte of data.

TXD is clear when the FIFO is full. The TXD field can be used to check that the FIFO can accept data before any is written. Any writes to a full TX FIFO will be ignored.

The read-only RXD field is set when the FIFO contains at least one byte of data. RXD is cleared when the FIFO becomes empty. The RXD field can be used to check that the FIFO contains data before reading. Reading from an empty FIFO will return invalid data.

The read-only TXE field is set when the FIFO is empty. No further data will be transmitted until more data is written to the FIFO.

The read-only RXF field is set when the FIFO is full. No more clocks will be generated until space is available in the FIFO to receive more data.

The ERR field is set when the slave fails to acknowledge either its address or a data byte written to it. The ERR field is reset by writing a 1, writing a 0 to the field has no effect.

The CLKT field is set when the slave holds the SCL signal high for too long (clock stretching). The CLKT field is reset by writing a 1, writing a 0 to the field has no effect.

| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 31:10  |            | Reserved - Write as 0, read as don't care   |      |       |
| 9      | CLKT       | Clock Stretch Timeout<br>0 = No errors detected.<br>1 = Slave has held the SCL signal low (clock stretching) for longer and that specified in the I2CCLKT register.<br>Cleared by writing 1 to the field. | W1C  | 0x0   |
| 8      | ERR        | ACK Error<br>0 = No errors detected.<br>1 = Slave has not acknowledged its address. Cleared by writing 1 to the field.  | W1C  | 0x0   |
| 7      | RXF        | FIFO Full<br>0 = FIFO is not full.<br>1 = FIFO is full. If a read is underway, no further serial data will be received until data is read from FIFO.  | RO   | 0x0   |
| 6      | TXE        | FIFO Empty<br>0 = FIFO is not empty.<br>1 = FIFO is empty. If a write is underway, no further serial data can be transmitted until data is written to the FIFO.   | RO   | 0x1   |
| 5      | RXD        | FIFO contains Data<br>0 = FIFO is empty.<br>1 = FIFO contains at least 1 byte. Cleared by reading sufficient data from FIFO.  | RO   | 0x0   |
| 4      | TXD        | FIFO can accept Data<br>0 = FIFO is full. The FIFO cannot accept more data.<br>1 = FIFO has space for at least 1 byte.  | RO   | 0x1   |



| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 3      | RXR        | FIFO needs Reading ( $\frac{3}{4}$ full)<br>0 = FIFO is less than $\frac{3}{4}$ full and a read is underway.<br>1 = FIFO is $\frac{3}{4}$ or more full and a read is underway.<br>Cleared by reading sufficient data from the FIFO.                               | RO   | 0x0   |
| 2      | TXW        | FIFO needs Writing ( $\frac{1}{4}$ full)<br>0 = FIFO is at least $\frac{1}{4}$ full and a write is underway (or sufficient data to send).<br>1 = FIFO is less than $\frac{1}{4}$ full and a write is underway.<br>Cleared by writing sufficient data to the FIFO. | RO   | 0x0   |
| 1      | DONE       | Transfer Done<br>0 = Transfer not completed.<br>1 = Transfer complete. Cleared by writing 1 to the field.   | W1C  | 0x0   |
| 0      | TA         | Transfer Active<br>0 = Transfer not active.<br>1 = Transfer active.   | RO   | 0x0   |

Table 24. S Register

## DLEN Register

### Synopsis

The data length register defines the number of bytes of data to transmit or receive in the I2C transfer. Reading the register gives the number of bytes remaining in the current transfer.

The DLEN field specifies the number of bytes to be transmitted/received. Reading the DLEN field when a transfer is in progress (TA = 1) returns the number of bytes still to be transmitted or received. Reading the DLEN field when the transfer has just completed (DONE = 1) returns zero as there are no more bytes to transmit or receive.

Finally, reading the DLEN field when TA = 0 and DONE = 0 returns the last value written. The DLEN field can be left over multiple transfers.

| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 31:16  |            | Reserved - Write as 0, read as don't care   |      |       |
| 15:0   | DLEN       | Data Length.<br>Writing to DLEN specifies the number of bytes to be transmitted/received. Reading from DLEN when TA = 1 or DONE = 1, returns the number of bytes still to be transmitted or received.<br>Reading from DLEN when TA = 0 and DONE = 0, returns the last DLEN value written. DLEN can be left over multiple packets. | RW   | 0x0   |

Table 25. DLEN Register

## A Register



## Synopsis

The slave address register specifies the slave address and cycle type. The address register can be left across multiple transfers.

The ADDR field specifies the slave address of the I2C device.

| Bit(s) | Field Name | Description                               | Type | Reset |
|--------|------------|---|------|-------|
| 31:7   |            | Reserved - Write as 0, read as don't care |      |       |
| 6:0    | ADDR       | Slave Address.                            | RW   | 0x0   |

Table 26. A Register

## FIFO Register

### Synopsis

The Data FIFO register is used to access the FIFO. Write cycles to this address place data in the 16-byte FIFO, ready to transmit on the BSC bus. Read cycles access data received from the bus.

Data writes to a full FIFO will be ignored and data reads from an empty FIFO will result in invalid data. The FIFO can be cleared using the I2CC.CLEAR field.

The DATA field specifies the data to be transmitted or received.

| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 31:8   |            | Reserved - Write as 0, read as don't care  |      |       |
| 7:0    | DATA       | Writes to the register write transmit data to the FIFO.<br>Reads from register read received data from the FIFO. | RW   | 0x0   |

Table 27. FIFO Register

## DIV Register

### Synopsis

The clock divider register is used to define the clock speed of the BSC peripheral.

The CDIV field specifies the core clock divider used by the BSC.

| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 31:16  |            | Reserved - Write as 0, read as don't care   |      |       |
| 15:0   | CDIV       | Clock Divider<br>$SCL = \text{core\_clock} / \text{CDIV}$<br>Where core_clk is nominally 150 MHz. If CDIV is set to 0, the divisor is 32768. CDIV is always rounded down to an even number. The default value should result in a 100 kHz I2C clock frequency. | RW   | 0x5dc |

Table 28. DIV Register

## DEL Register





## Synopsis

The data delay register provides fine control over the sampling/launch point of the data.

The REDL field specifies the number core clocks to wait after the rising edge before sampling the incoming data.

The FEDL field specifies the number core clocks to wait after the falling edge before outputting the next data bit.

Note: Care must be taken in choosing values for FEDL and REDL as it is possible to cause the BSC master to malfunction by setting values of CDIV/2 or greater. Therefore the delay values should always be set to less than CDIV/2.

| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 31:16  | FEDL       | Falling Edge Delay<br>Number of core clock cycles to wait after the falling edge of SCL before outputting next bit of data. | RW   | 0x30  |
| 15:0   | REDL       | Rising Edge Delay<br>Number of core clock cycles to wait after the rising edge of SCL before reading the next bit of data.  | RW   | 0x30  |

Table 29. DEL Register

## CLKT Register

### Synopsis

The clock stretch timeout register provides a timeout on how long the master waits for the slave to stretch the clock before deciding that the slave has hung.

The TOUT field specifies the number I2C SCL clocks to wait after releasing SCL high and finding that the SCL is still low before deciding that the slave is not responding and moving the I2C machine forward. When a timeout occurs, the I2CS.CLKT bit is set.

Writing 0x0 to TOUT will result in the Clock Stretch Timeout being disabled.

| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 31:16  |            | Reserved - Write as 0, read as don't care  |      |       |
| 15:0   | TOUT       | Clock Stretch Timeout Value<br>Number of SCL clock cycles to wait after the rising edge of SCL before deciding that the slave is not responding. | RW   | 0x40  |

Table 30. CLKT Register

## 3.3. 10-Bit Addressing

10-bit addressing is an extension to the standard 7-bit addressing mode. This section describes in detail how to read/write using 10-bit addressing with this I<sup>2</sup>C controller.

10-bit addressing is compatible with, and can be combined with, 7-bit addressing. Using 10 bits for addressing exploits the reserved combination 1111 0xx for the first byte following a START (S) or



REPEATED START (Sr) condition.

The 10-bit slave address is formed from the first two bytes following a S or Sr condition.

The first seven bits of the first byte are the combination 11110XX of which the last two bits (XX) are the two most significant bits of the 10-bit address. The eighth bit of the first byte is the R/W bit. If the R/W bit is '0' (write) then the following byte contains the remaining 8 bits of the 10-bit address. If the R/W bit is '1' then the next byte contains data transmitted from the slave to the master.

### 3.3.1. Writing

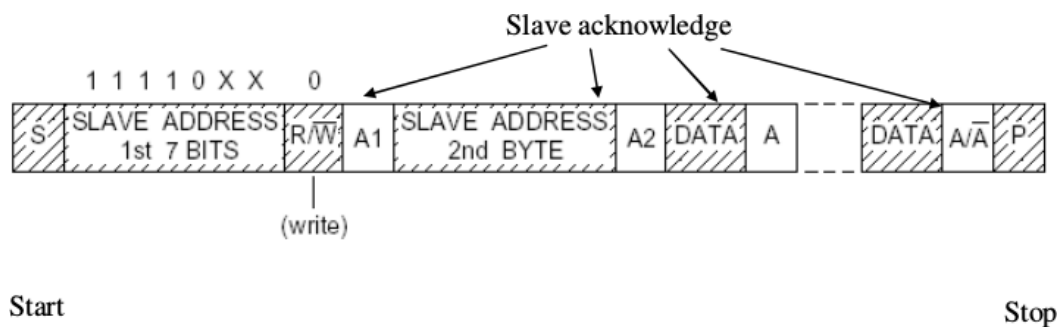


Figure 2. Write to a slave with 10-bit address

Figure 2 shows a write to a slave with a 10-bit address, to perform this using the controller one must do the following:

Assuming we are in the 'stop' state: (and the FIFO is empty)

1. Write the number of data bytes to written (plus one) to the **I2CDLEN** register
2. Write 'XXXXXXXX' to the FIFO where 'XXXXXXXX' are the least 8 significant bits of the 10-bit slave address
3. Write other data to be transmitted to the FIFO
4. Write '11110XX' to Slave Address Register where 'XX' are the two most significant bits of the 10-bit address
5. Set **I2CC.READ = 0** and **I2CC.ST = 1**, this will start a write transfer

### 3.3.2. Reading

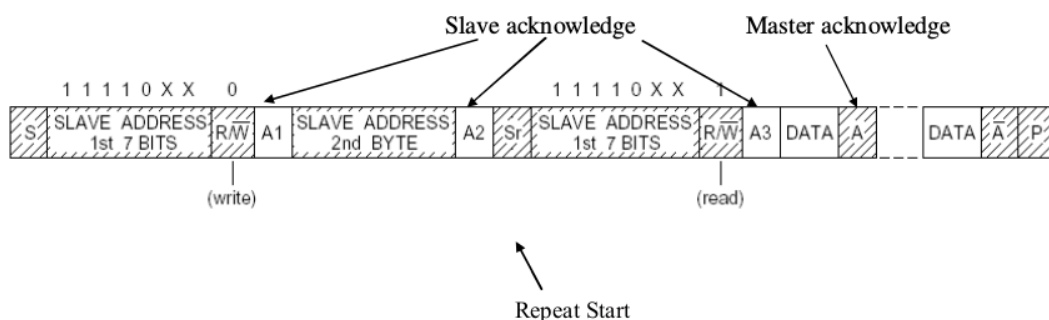
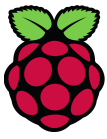


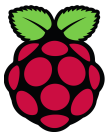
Figure 3. Read from slave with 10-bit address

Figure 3 shows how a read from a slave with a 10-bit address is performed. The following procedure



shows how to perform a read using the controller:

1. Write 1 to the `I2CDLEN` register
2. Write 'XXXXXXXX' to the FIFO where 'XXXXXXXX' are the least 8 significant bits of the 10-bit slave address
3. Write '11110XX' to the Slave Address Register where 'XX' are the two most significant bits of the 10-bit address
4. Set `I2CC.READ = 0` and `I2CC.ST = 1`, this will start a write transfer
5. Poll the `I2CS.TA` bit, waiting for the transfer to start
6. Write the number of data bytes to read to the `I2CDLEN` register
7. Set `I2CC.READ = 1` and `I2CC.ST = 1`, this will send the repeat start bit, the slave address and the R/W bit (which is '1'), initiating the read



## Chapter 4. DMA Controller

### 4.1. Overview

The majority of hardware pipelines and peripherals within the BCM2711 are bus masters, enabling them to efficiently satisfy their own data requirements. This reduces the requirements of the DMA controller to block-to-block memory transfers and supporting some of the simpler peripherals. In addition, the DMA controller provides a *read-only* prefetch mode to allow data to be brought into the L2 cache in anticipation of its later use.

Note that the DMA controller is directly connected to the peripherals. Therefore the DMA controller must be set-up to use the Legacy Master addresses of the peripherals.

The BCM2711 DMA Controller provides a total of 16 DMA channels. Four of these are *DMA Lite* channels (with reduced performance and features), and four of them are *DMA4* channels (with increased performance and a wider address range). Each channel operates independently from the others and is internally arbitrated onto one of the three system busses. This means that the amount of bandwidth that a DMA channel may consume can be controlled by the arbiter settings (although these are not publicly exposed).

Each DMA channel operates by loading a *Control Block* (CB) data structure from memory into internal registers. The Control Block defines the required DMA operation. Each Control Block can point to a further Control Block to be loaded and executed once the operation described in the current Control Block has completed. In this way a linked list of Control Blocks can be constructed in order to execute a sequence of DMA operations without software intervention.

The DMA supports AXI read bursts to ensure efficient external SDRAM use. The DMA Control Block contains a burst parameter which indicates the required burst size of certain memory transfers. In general the DMA doesn't do write bursts, although wide writes will be done in 2 beat bursts if possible.

Memory-to-Peripheral transfers can be paced by a *Data Request* (DREQ) signal which is generated by the peripheral. The DREQ signal is level sensitive and controls the DMA by gating its AXI bus requests.

A peripheral can also provide a *Panic* signal alongside the DREQ to indicate that there is an imminent danger of FIFO underflow or overflow or similar critical situation. The Panic is used to select the AXI apriority level which is then passed out onto the AXI bus so that it can be used to influence arbitration in the rest of the system.

The allocation of peripherals to DMA channels is programmable.

The DMA can deal with byte aligned transfers and will minimise bus traffic by buffering and packing misaligned accesses.

Each DMA channel can be fully disabled via a top level power register to save power.



## 4.2. DMA Controller Registers

The DMA Controller is comprised of several identical DMA Channels depending upon the required configuration. Each individual DMA channel has an identical register map (although LITE channels have fewer registers and DMA4 channels have more registers).

DMA Channel 0 is located at the address of `0x7E00 7000`, Channel 1 at `0x7E00 7100`, Channel 2 at `0x7E00 7200` and so on. Thus adjacent DMA Channels are offset by `0x100`.

DMA Channel 15 however, is physically removed from the other DMA Channels and so has a different address base of `0x7EE0 5000`. DMA Channel 15 is exclusively used by the VPU.

| Base Address       | DMA Channel                 |
|--------------------|-----------------------------|
| DMA0_BASE + 0x000  | DMA Channel 0 Register Set  |
| DMA0_BASE + 0x100  | DMA Channel 1 Register Set  |
| DMA0_BASE + 0x200  | DMA Channel 2 Register Set  |
| DMA0_BASE + 0x300  | DMA Channel 3 Register Set  |
| DMA0_BASE + 0x400  | DMA Channel 4 Register Set  |
| DMA0_BASE + 0x500  | DMA Channel 5 Register Set  |
| DMA0_BASE + 0x600  | DMA Channel 6 Register Set  |
| DMA0_BASE + 0x700  | DMA Channel 7 Register Set  |
| DMA0_BASE + 0x800  | DMA Channel 8 Register Set  |
| DMA0_BASE + 0x900  | DMA Channel 9 Register Set  |
| DMA0_BASE + 0xa00  | DMA Channel 10 Register Set |
| DMA0_BASE + 0xb00  | DMA Channel 11 Register Set |
| DMA0_BASE + 0xc00  | DMA Channel 12 Register Set |
| DMA0_BASE + 0xd00  | DMA Channel 13 Register Set |
| DMA0_BASE + 0xe00  | DMA Channel 14 Register Set |
| DMA15_BASE + 0x000 | DMA Channel 15 Register Set |

*Table 31. DMA Controller Register Address Map*

### 4.2.1. DMA Channel Register Address Map

Each DMA channel of a particular type has an identical register map, only the base address of each channel is different.

There is a global enable register at the top of the Address map that can disable each DMA for powersaving.

Only three registers in each channel's register set are directly writeable (CS, CONBLK\_AD and DEBUG).



The other registers (TI, SOURCE\_AD, DEST\_AD, TXFR\_LEN, STRIDE & NEXTCONBK) are automatically loaded from a Control Block data structure held in external memory.

### Control Block Data Structure

Control Blocks (CB) are 8 words (256 bits) in length and must start at a 256-bit aligned address. The format of the different CB data structures in memory, are shown below.

Each 32-bit word of the Control Block is automatically loaded into the corresponding 32-bit DMA Control Block register at the start of a DMA transfer. The descriptions of these registers also define the corresponding bit locations in the CB data structure in memory.

| 32-bit Word Offset | Description                | Associated Read-Only Register |
|--------------------|----------------------------|-------------------------------|
| 0                  | Transfer Information       | TI                            |
| 1                  | Source Address             | SOURCE_AD                     |
| 2                  | Destination Address        | DEST_AD                       |
| 3                  | Transfer Length            | TXFR_LEN                      |
| 4                  | 2D Mode Stride             | STRIDE                        |
| 5                  | Next Control Block Address | NEXTCONBK                     |
| 6-7                | Reserved – set to zero.    | N/A                           |

*Table 32. DMA Control Block Definition*

| 32-bit Word Offset | Description                | Associated Read-Only Register |
|--------------------|----------------------------|-------------------------------|
| 0                  | Transfer Information       | TI                            |
| 1                  | Source Address             | SOURCE_AD                     |
| 2                  | Destination Address        | DEST_AD                       |
| 3                  | Transfer Length            | TXFR_LEN                      |
| 4                  | Reserved – set to zero.    | N/A                           |
| 5                  | Next Control Block Address | NEXTCONBK                     |
| 6-7                | Reserved – set to zero.    | N/A                           |

*Table 33. DMA Lite Control Block Definition*

| 32-bit Word Offset | Description          | Associated Read-Only Register |
|--------------------|----------------------|-------------------------------|
| 0                  | Transfer Information | TI                            |



| 32-bit Word Offset | Description                | Associated Read-Only Register |
|--------------------|----------------------------|-------------------------------|
| 1                  | Source Address             | SRC                           |
| 2                  | Source Information         | SRCI                          |
| 3                  | Destination Address        | DEST                          |
| 4                  | Destination Information    | DETI                          |
| 5                  | Transfer Length            | LEN                           |
| 6                  | Next Control Block Address | NEXT_CB                       |
| 7                  | Reserved – set to zero.    | N/A                           |

*Table 34. DMA4 Control Block Definition*

The DMA is started by writing the address of a CB structure into the CONBLK\_AD register <sup>[5]</sup> and then setting the ACTIVE bit. The DMA will fetch the CB from the address set in the SCB\_ADDR field of the CONBLK\_AD register <sup>[6]</sup> and it will load it into the read-only registers described below. It will then begin a DMA transfer according to the information in the CB.

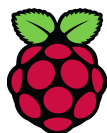
When it has completed the current DMA transfer (length  $\Rightarrow$  0) the DMA will update the CONBLK\_AD register with the contents of the NEXTCONBK register <sup>[7]</sup>, fetch a new CB from that address, and start the whole procedure once again.

The DMA will stop (and clear the ACTIVE bit) when it has completed a DMA transfer and the NEXTCONBK register is set to 0x0000\_0000. It will load this value into the CONBLK\_AD register and then stop.

Most of the Control Block registers cannot be written to directly as they are loaded automatically from memory. They can be read to provide status information, and to indicate the progress of the current DMA transfer. The value loaded into the NEXTCONBK / NEXT\_CB register can be overwritten so that the linked list of Control Block data structures can be dynamically altered. However it is only safe to do this when the DMA is paused.

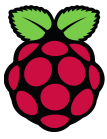
## Register Map

| Address Offset | Register Name               | Description                                    | Size |
|----------------|-----------------------------|--|------|
| 0x000          | <a href="#">0_CS</a>        | DMA Channel 0 Control and Status               | 32   |
| 0x004          | <a href="#">0_CONBLK_AD</a> | DMA Channel 0 Control Block Address            | 32   |
| 0x008          | <a href="#">0_TI</a>        | DMA Channel 0 CB Word 0 (Transfer Information) | 32   |
| 0x00C          | <a href="#">0_SOURCE_AD</a> | DMA Channel 0 CB Word 1 (Source Address)       | 32   |

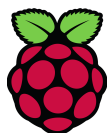


| Address Offset | Register Name               | Description                                    | Size |
|----------------|-----------------------------|--|------|
| 0x010          | <a href="#">0_DEST_AD</a>   | DMA Channel 0 CB Word 2 (Destination Address)  | 32   |
| 0x014          | <a href="#">0_TXFR_LEN</a>  | DMA Channel 0 CB Word 3 (Transfer Length)      | 32   |
| 0x018          | <a href="#">0_STRIDE</a>    | DMA Channel 0 CB Word 4 (2D Stride)            | 32   |
| 0x01C          | <a href="#">0_NEXTCONBK</a> | DMA Channel 0 CB Word 5 (Next CB Address)      | 32   |
| 0x020          | <a href="#">0_DEBUG</a>     | DMA Channel 0 Debug                            | 32   |
| 0x100          | <a href="#">1_CS</a>        | DMA Channel 1 Control and Status               | 32   |
| 0x104          | <a href="#">1_CONBLK_AD</a> | DMA Channel 1 Control Block Address            | 32   |
| 0x108          | <a href="#">1_TI</a>        | DMA Channel 1 CB Word 0 (Transfer Information) | 32   |
| 0x10C          | <a href="#">1_SOURCE_AD</a> | DMA Channel 1 CB Word 1 (Source Address)       | 32   |
| 0x110          | <a href="#">1_DEST_AD</a>   | DMA Channel 1 CB Word 2 (Destination Address)  | 32   |
| 0x114          | <a href="#">1_TXFR_LEN</a>  | DMA Channel 1 CB Word 3 (Transfer Length)      | 32   |
| 0x118          | <a href="#">1_STRIDE</a>    | DMA Channel 1 CB Word 4 (2D Stride)            | 32   |
| 0x11C          | <a href="#">1_NEXTCONBK</a> | DMA Channel 1 CB Word 5 (Next CB Address)      | 32   |
| 0x120          | <a href="#">1_DEBUG</a>     | DMA Channel 1 Debug                            | 32   |
| 0x200          | <a href="#">2_CS</a>        | DMA Channel 2 Control and Status               | 32   |
| 0x204          | <a href="#">2_CONBLK_AD</a> | DMA Channel 2 Control Block Address            | 32   |
| 0x208          | <a href="#">2_TI</a>        | DMA Channel 2 CB Word 0 (Transfer Information) | 32   |
| 0x20C          | <a href="#">2_SOURCE_AD</a> | DMA Channel 2 CB Word 1 (Source Address)       | 32   |
| 0x210          | <a href="#">2_DEST_AD</a>   | DMA Channel 2 CB Word 2 (Destination Address)  | 32   |
| 0x214          | <a href="#">2_TXFR_LEN</a>  | DMA Channel 2 CB Word 3 (Transfer Length)      | 32   |





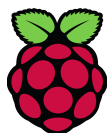
| Address Offset | Register Name               | Description                                    | Size |
|----------------|-----------------------------|--|------|
| 0x218          | <a href="#">2_STRIDE</a>    | DMA Channel 2 CB Word 4 (2D Stride)            | 32   |
| 0x21C          | <a href="#">2_NEXTCONBK</a> | DMA Channel 2 CB Word 5 (Next CB Address)      | 32   |
| 0x220          | <a href="#">2_DEBUG</a>     | DMA Channel 2 Debug                            | 32   |
| 0x300          | <a href="#">3_CS</a>        | DMA Channel 3 Control and Status               | 32   |
| 0x304          | <a href="#">3_CONBLK_AD</a> | DMA Channel 3 Control Block Address            | 32   |
| 0x308          | <a href="#">3_TI</a>        | DMA Channel 3 CB Word 0 (Transfer Information) | 32   |
| 0x30C          | <a href="#">3_SOURCE_AD</a> | DMA Channel 3 CB Word 1 (Source Address)       | 32   |
| 0x310          | <a href="#">3_DEST_AD</a>   | DMA Channel 3 CB Word 2 (Destination Address)  | 32   |
| 0x314          | <a href="#">3_TXFR_LEN</a>  | DMA Channel 3 CB Word 3 (Transfer Length)      | 32   |
| 0x318          | <a href="#">3_STRIDE</a>    | DMA Channel 3 CB Word 4 (2D Stride)            | 32   |
| 0x31C          | <a href="#">3_NEXTCONBK</a> | DMA Channel 3 CB Word 5 (Next CB Address)      | 32   |
| 0x320          | <a href="#">3_DEBUG</a>     | DMA Channel 0 Debug                            | 32   |
| 0x400          | <a href="#">4_CS</a>        | DMA Channel 4 Control and Status               | 32   |
| 0x404          | <a href="#">4_CONBLK_AD</a> | DMA Channel 4 Control Block Address            | 32   |
| 0x408          | <a href="#">4_TI</a>        | DMA Channel 4 CB Word 0 (Transfer Information) | 32   |
| 0x40C          | <a href="#">4_SOURCE_AD</a> | DMA Channel 4 CB Word 1 (Source Address)       | 32   |
| 0x410          | <a href="#">4_DEST_AD</a>   | DMA Channel 4 CB Word 2 (Destination Address)  | 32   |
| 0x414          | <a href="#">4_TXFR_LEN</a>  | DMA Channel 4 CB Word 3 (Transfer Length)      | 32   |
| 0x418          | <a href="#">4_STRIDE</a>    | DMA Channel 4 CB Word 4 (2D Stride)            | 32   |
| 0x41C          | <a href="#">4_NEXTCONBK</a> | DMA Channel 4 CB Word 5 (Next CB Address)      | 32   |
| 0x420          | <a href="#">4_DEBUG</a>     | DMA Channel 0 Debug                            | 32   |



| Address Offset | Register Name               | Description  | Size |
|----------------|-----------------------------|--|------|
| 0x500          | <a href="#">5_CS</a>        | DMA Channel 5 Control and Status                       | 32   |
| 0x504          | <a href="#">5_CONBLK_AD</a> | DMA Channel 5 Control Block Address                    | 32   |
| 0x508          | <a href="#">5_TI</a>        | DMA Channel 5 CB Word 0<br>(Transfer Information)      | 32   |
| 0x50C          | <a href="#">5_SOURCE_AD</a> | DMA Channel 5 CB Word 1<br>(Source Address)            | 32   |
| 0x510          | <a href="#">5_DEST_AD</a>   | DMA Channel 5 CB Word 2<br>(Destination Address)       | 32   |
| 0x514          | <a href="#">5_TXFR_LEN</a>  | DMA Channel 5 CB Word 3<br>(Transfer Length)           | 32   |
| 0x518          | <a href="#">5_STRIDE</a>    | DMA Channel 5 CB Word 4<br>(2D Stride)                 | 32   |
| 0x51C          | <a href="#">5_NEXTCONBK</a> | DMA Channel 5 CB Word 5<br>(Next CB Address)           | 32   |
| 0x520          | <a href="#">5_DEBUG</a>     | DMA Channel 5 Debug                                    | 32   |
| 0x600          | <a href="#">6_CS</a>        | DMA Channel 6 Control and Status                       | 32   |
| 0x604          | <a href="#">6_CONBLK_AD</a> | DMA Channel 6 Control Block Address                    | 32   |
| 0x608          | <a href="#">6_TI</a>        | DMA Channel 6 CB Word 0<br>(Transfer Information)      | 32   |
| 0x60C          | <a href="#">6_SOURCE_AD</a> | DMA Channel 6 CB Word 1<br>(Source Address)            | 32   |
| 0x610          | <a href="#">6_DEST_AD</a>   | DMA Channel 6 CB Word 2<br>(Destination Address)       | 32   |
| 0x614          | <a href="#">6_TXFR_LEN</a>  | DMA Channel 6 CB Word 3<br>(Transfer Length)           | 32   |
| 0x618          | <a href="#">6_STRIDE</a>    | DMA Channel 6 CB Word 4<br>(2D Stride)                 | 32   |
| 0x61C          | <a href="#">6_NEXTCONBK</a> | DMA Channel 6 CB Word 5<br>(Next CB Address)           | 32   |
| 0x620          | <a href="#">6_DEBUG</a>     | DMA Channel 6 Debug                                    | 32   |
| 0x700          | <a href="#">7_CS</a>        | DMA Lite Channel 7 Control and Status                  | 32   |
| 0x704          | <a href="#">7_CONBLK_AD</a> | DMA Lite Channel 7 Control Block Address               | 32   |
| 0x708          | <a href="#">7_TI</a>        | DMA Lite Channel 7 CB Word 0<br>(Transfer Information) | 32   |



| Address Offset | Register Name               | Description   | Size |
|----------------|-----------------------------|---|------|
| 0x70C          | <a href="#">7_SOURCE_AD</a> | DMA Lite Channel 7 CB Word 1 (Source Address)       | 32   |
| 0x710          | <a href="#">7_DEST_AD</a>   | DMA Lite Channel 7 CB Word 2 (Destination Address)  | 32   |
| 0x714          | <a href="#">7_TXFR_LEN</a>  | DMA Lite Channel 7 CB Word 3 (Transfer Length)      | 32   |
| 0x71C          | <a href="#">7_NEXTCONBK</a> | DMA Lite Channel 7 CB Word 5 (Next CB Address)      | 32   |
| 0x720          | <a href="#">7_DEBUG</a>     | DMA Lite Channel 7 Debug                            | 32   |
| 0x800          | <a href="#">8_CS</a>        | DMA Lite Channel 8 Control and Status               | 32   |
| 0x804          | <a href="#">8_CONBLK_AD</a> | DMA Lite Channel 8 Control Block Address            | 32   |
| 0x808          | <a href="#">8_TI</a>        | DMA Lite Channel 8 CB Word 0 (Transfer Information) | 32   |
| 0x80C          | <a href="#">8_SOURCE_AD</a> | DMA Lite Channel 8 CB Word 1 (Source Address)       | 32   |
| 0x810          | <a href="#">8_DEST_AD</a>   | DMA Lite Channel 8 CB Word 2 (Destination Address)  | 32   |
| 0x814          | <a href="#">8_TXFR_LEN</a>  | DMA Lite Channel 8 CB Word 3 (Transfer Length)      | 32   |
| 0x81C          | <a href="#">8_NEXTCONBK</a> | DMA Lite Channel 8 CB Word 5 (Next CB Address)      | 32   |
| 0x820          | <a href="#">8_DEBUG</a>     | DMA Lite Channel 8 Debug                            | 32   |
| 0x900          | <a href="#">9_CS</a>        | DMA Lite Channel 9 Control and Status               | 32   |
| 0x904          | <a href="#">9_CONBLK_AD</a> | DMA Lite Channel 9 Control Block Address            | 32   |
| 0x908          | <a href="#">9_TI</a>        | DMA Lite Channel 9 CB Word 0 (Transfer Information) | 32   |
| 0x90C          | <a href="#">9_SOURCE_AD</a> | DMA Lite Channel 9 CB Word 1 (Source Address)       | 32   |
| 0x910          | <a href="#">9_DEST_AD</a>   | DMA Lite Channel 9 CB Word 2 (Destination Address)  | 32   |
| 0x914          | <a href="#">9_TXFR_LEN</a>  | DMA Lite Channel 9 CB Word 3 (Transfer Length)      | 32   |
| 0x91C          | <a href="#">9_NEXTCONBK</a> | DMA Lite Channel 9 CB Word 5 (Next CB Address)      | 32   |
| 0x920          | <a href="#">9_DEBUG</a>     | DMA Lite Channel 9 Debug                            | 32   |



| Address Offset | Register Name                | Description   | Size |
|----------------|------------------------------|---|------|
| 0xA00          | <a href="#">10_CS</a>        | DMA Lite Channel 10 Control and Status                          | 32   |
| 0xA04          | <a href="#">10_CONBLK_AD</a> | DMA Lite Channel 10 Control Block Address                       | 32   |
| 0xA08          | <a href="#">10_TI</a>        | DMA Lite Channel 10 CB Word 0 (Transfer Information)            | 32   |
| 0xA0C          | <a href="#">10_SOURCE_AD</a> | DMA Lite Channel 10 CB Word 1 (Source Address)                  | 32   |
| 0xA10          | <a href="#">10_DEST_AD</a>   | DMA Lite Channel 10 CB Word 2 (Destination Address)             | 32   |
| 0xA14          | <a href="#">10_TXFR_LEN</a>  | DMA Lite Channel 10 CB Word 3 (Transfer Length)                 | 32   |
| 0xA1C          | <a href="#">10_NEXTCONBK</a> | DMA Lite Channel 10 CB Word 5 (Next CB Address)                 | 32   |
| 0xA20          | <a href="#">10_DEBUG</a>     | DMA Lite Channel 10 Debug                                       | 32   |
| 0xB00          | <a href="#">11_CS</a>        | DMA4 Channel 11 Control and Status                              | 32   |
| 0xB04          | <a href="#">11_CB</a>        | DMA4 Channel 11 Control Block Address                           | 32   |
| 0xB0C          | <a href="#">11_DEBUG</a>     | DMA4 Channel 11 Debug   | 32   |
| 0xB10          | <a href="#">11_TI</a>        | DMA4 Channel 11 CB Word 0 (Transfer Information)                | 32   |
| 0xB14          | <a href="#">11_SRC</a>       | DMA4 Channel 11 CB Word 1 (Source Address [31:0])               | 32   |
| 0xB18          | <a href="#">11_SRCI</a>      | DMA4 Channel 11 CB Word 2 (Source Address [40:32] and Info)     | 32   |
| 0xB1C          | <a href="#">11_DEST</a>      | DMA4 Channel 11 CB Word 3 (Destination Address[31:0])           | 32   |
| 0xB20          | <a href="#">11_DESTI</a>     | DMA4 Channel 11 CB Word 4 (Destination Address[40:32] and Info) | 32   |
| 0xB24          | <a href="#">11_LEN</a>       | DMA4 Channel 11 CB Word 5 (Transfer Length)                     | 32   |
| 0xB28          | <a href="#">11_NEXT_CB</a>   | DMA4 Channel 11 CB Word 6 (Next CB Address)                     | 32   |
| 0xB2C          | <a href="#">11_DEBUG2</a>    | DMA4 Channel 11 More Debug                                      | 32   |
| 0xC00          | <a href="#">12_CS</a>        | DMA4 Channel 12 Control and Status                              | 32   |
| 0xC04          | <a href="#">12_CB</a>        | DMA4 Channel 12 Control Block Address                           | 32   |
| 0xC0C          | <a href="#">12_DEBUG</a>     | DMA4 Channel 12 Debug   | 32   |



| Address Offset | Register Name              | Description   | Size |
|----------------|----------------------------|---|------|
| 0xC10          | <a href="#">12_TI</a>      | DMA4 Channel 12 CB Word 0 (Transfer Information)                | 32   |
| 0xC14          | <a href="#">12_SRC</a>     | DMA4 Channel 12 CB Word 1 (Source Address [31:0])               | 32   |
| 0xC18          | <a href="#">12_SRCI</a>    | DMA4 Channel 12 CB Word 2 (Source Address [40:32] and Info)     | 32   |
| 0xC1C          | <a href="#">12_DEST</a>    | DMA4 Channel 12 CB Word 3 (Destination Address[31:0])           | 32   |
| 0xC20          | <a href="#">12_DESTI</a>   | DMA4 Channel 12 CB Word 4 (Destination Address[40:32] and Info) | 32   |
| 0xC24          | <a href="#">12_LEN</a>     | DMA4 Channel 12 CB Word 5 (Transfer Length)                     | 32   |
| 0xC28          | <a href="#">12_NEXT_CB</a> | DMA4 Channel 12 CB Word 6 (Next CB Address)                     | 32   |
| 0xC2C          | <a href="#">12_DEBUG2</a>  | DMA4 Channel 12 More Debug                                      | 32   |
| 0xD00          | <a href="#">13_CS</a>      | DMA4 Channel 13 Control and Status                              | 32   |
| 0xD04          | <a href="#">13_CB</a>      | DMA4 Channel 13 Control Block Address                           | 32   |
| 0xD0C          | <a href="#">13_DEBUG</a>   | DMA4 Channel 13 Debug   | 32   |
| 0xD10          | <a href="#">13_TI</a>      | DMA4 Channel 13 CB Word 0 (Transfer Information)                | 32   |
| 0xD14          | <a href="#">13_SRC</a>     | DMA4 Channel 13 CB Word 1 (Source Address [31:0])               | 32   |
| 0xD18          | <a href="#">13_SRCI</a>    | DMA4 Channel 13 CB Word 2 (Source Address [40:32] and Info)     | 32   |
| 0xD1C          | <a href="#">13_DEST</a>    | DMA4 Channel 13 CB Word 3 (Destination Address[31:0])           | 32   |
| 0xD20          | <a href="#">13_DESTI</a>   | DMA4 Channel 13 CB Word 4 (Destination Address[40:32] and Info) | 32   |
| 0xD24          | <a href="#">13_LEN</a>     | DMA4 Channel 13 CB Word 5 (Transfer Length)                     | 32   |
| 0xD28          | <a href="#">13_NEXT_CB</a> | DMA4 Channel 13 CB Word 6 (Next CB Address)                     | 32   |
| 0xD2C          | <a href="#">13_DEBUG2</a>  | DMA4 Channel 13 More Debug                                      | 32   |
| 0xE00          | <a href="#">14_CS</a>      | DMA4 Channel 14 Control and Status                              | 32   |
| 0xE04          | <a href="#">14_CB</a>      | DMA4 Channel 14 Control Block Address                           | 32   |



| Address Offset | Register Name              | Description   | Size |
|----------------|----------------------------|---|------|
| 0xE0C          | <a href="#">14_DEBUG</a>   | DMA4 Channel 14 Debug   | 32   |
| 0xE10          | <a href="#">14_TI</a>      | DMA4 Channel 14 CB Word 0 (Transfer Information)                | 32   |
| 0xE14          | <a href="#">14_SRC</a>     | DMA4 Channel 14 CB Word 1 (Source Address [31:0])               | 32   |
| 0xE18          | <a href="#">14_SRCI</a>    | DMA4 Channel 14 CB Word 2 (Source Address [40:32] and Info)     | 32   |
| 0xE1C          | <a href="#">14_DEST</a>    | DMA4 Channel 14 CB Word 3 (Destination Address[31:0])           | 32   |
| 0xE20          | <a href="#">14_DESTI</a>   | DMA4 Channel 14 CB Word 4 (Destination Address[40:32] and Info) | 32   |
| 0xE24          | <a href="#">14_LEN</a>     | DMA4 Channel 14 CB Word 5 (Transfer Length)                     | 32   |
| 0xE28          | <a href="#">14_NEXT_CB</a> | DMA4 Channel 14 CB Word 6 (Next CB Address)                     | 32   |
| 0xE2C          | <a href="#">14_DEBUG2</a>  | DMA4 Channel 14 More Debug                                      | 32   |
| 0xFE0          | <a href="#">INT_STATUS</a> | Interrupt status of each DMA channel                            | 32   |
| 0xFF0          | <a href="#">ENABLE</a>     | Global enable bits for each DMA channel                         | 32   |

Table 35. DMA Controller Register Map

## 0\_CS, 1\_CS, 2\_CS, 3\_CS, 4\_CS, 5\_CS, 6\_CS, 7\_CS, 8\_CS, 9\_CS & 10\_CS Registers

### Synopsis

DMA Control and Status register contains the main control and status bits for this DMA channel.

| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 31     | RESET      | DMA Channel Reset<br>Writing a 1 to this bit will reset the DMA. The bit cannot be read, and will self clear.   | W1SC | 0x0   |
| 30     | ABORT      | Abort DMA<br>Writing a 1 to this bit will abort the current DMA CB. The DMA will load the next CB and attempt to continue. The bit cannot be read, and will self clear. | W1SC | 0x0   |
| 29     | DISDEBUG   | Disable debug pause signal<br>When set to 1, the DMA will not stop when the debug pause signal is asserted.   | RW   | 0x0   |



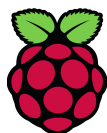
| Bit(s) | Field Name                                | Description  | Type | Reset |
|--------|---|--|------|-------|
| 28     | WAIT_FOR_OUTSTANDING_WRITES               | Wait for outstanding writes<br>When set to 1, the DMA will keep a tally of the AXI writes going out and the write responses coming in. At the very end of the current DMA transfer it will wait until the last outstanding write response has been received before indicating the transfer is complete. Whilst waiting it will load the next CB address (but will not fetch the CB), clear the active flag (if the next CB address = zero), and it will defer setting the END flag or the INT flag until the last outstanding write response has been received.<br>In this mode, the DMA will pause if it has more than 13 outstanding writes at any one time. | RW   | 0x0   |
| 27:24  | Reserved - Write as 0, read as don't care |  |      |       |
| 23:20  | PANIC_PRIORITY                            | AXI Panic Priority Level<br>Sets the priority of panicking AXI bus transactions. This value is used when the panic bit of the selected peripheral channel is 1.<br>Zero is the lowest priority.  | RW   | 0x0   |
| 19:16  | PRIORITY                                  | AXI Priority Level<br>Sets the priority of normal AXI bus transactions. This value is used when the panic bit of the selected peripheral channel is zero.<br>Zero is the lowest priority.  | RW   | 0x0   |
| 15:9   | Reserved - Write as 0, read as don't care |  |      |       |
| 8      | ERROR                                     | DMA Error<br>Indicates if the DMA has detected an error. The error flags are available in the debug register, and have to be cleared by writing to that register.<br>1 = DMA channel has an error flag set.<br>0 = DMA channel is OK.  | RO   | 0x0   |
| 7      | Reserved - Write as 0, read as don't care |  |      |       |
| 6      | WAITING_FOR_OUTSTANDING_WRITES            | DMA is Waiting for the Last Write to be Received<br>Indicates if the DMA is currently waiting for any outstanding writes to be received, and is not transferring data.<br>1 = DMA channel is waiting.  | RO   | 0x0   |



| Bit(s) | Field Name     | Description  | Type | Reset |
|--------|----------------|--|------|-------|
| 5      | DREQ_STOPS_DMA | DMA Paused by DREQ State<br>Indicates if the DMA is currently paused and not transferring data due to the DREQ being inactive.<br>1 = DMA channel is paused.<br>0 = DMA channel is running.  | RO   | 0x0   |
| 4      | PAUSED         | DMA Paused State<br>Indicates if the DMA is currently paused and not transferring data. This will occur if: the active bit has been cleared, the DMA is currently executing wait cycles, the debug_pause signal has been set by the debug block, or the number of outstanding writes has exceeded the max count.<br>1 = DMA channel is paused.<br>0 = DMA channel is running.  | RO   | 0x0   |
| 3      | DREQ           | DREQ State<br>Indicates the state of the selected DREQ (Data Request) signal, i.e. the DREQ selected by the PERMAP field of the transfer info.<br>1 = Requesting data. This will only be valid once the DMA has started and the PERMAP field has been loaded from the CB. It will remain valid, indicating the selected DREQ signal, until a new CB is loaded. If PERMAP is set to zero (un-paced transfer) then this bit will read back as 1.<br>0 = No data request. | RO   | 0x0   |
| 2      | INT            | Interrupt Status<br>This is set when the transfer for the CB ends and INTEN is set to 1. Once set it must be manually cleared down, even if the next CB has INTEN = 0.<br>Write 1 to clear.  | W1C  | 0x0   |
| 1      | END            | DMA End Flag<br>Set when the transfer described by the current Control Block is complete. Write 1 to clear.  | W1C  | 0x0   |
| 0      | ACTIVE         | Activate the DMA<br>This bit enables the DMA. The DMA will start if this bit is set and the CB_ADDR is non zero. The DMA transfer can be paused and resumed by clearing, then setting it again.<br>This bit is automatically cleared at the end of the complete DMA transfer, i.e. after a NEXTCONBK = 0x0000_0000 has been loaded.  | RW   | 0x0   |

Table 36. 0\_CS, 1\_CS, 2\_CS, 3\_CS, 4\_CS, 5\_CS, 6\_CS, 7\_CS, 8\_CS, 9\_CS & 10\_CS Registers





## 0\_CONBLK\_AD, 1\_CONBLK\_AD, 2\_CONBLK\_AD, 3\_CONBLK\_AD, 4\_CONBLK\_AD, 5\_CONBLK\_AD, 6\_CONBLK\_AD, 7\_CONBLK\_AD, 8\_CONBLK\_AD, 9\_CONBLK\_AD & 10\_CONBLK\_AD Registers

### Synopsis

DMA Control Block Address register.

| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 31:0   | SCB_ADDR   | <p>Control Block Address</p> <p>This tells the DMA where to find a Control Block stored in memory. When the ACTIVE bit is set and this address is non zero, the DMA will begin its transfer by loading the contents of the addressed CB into the relevant DMA channel registers.</p> <p>At the end of the transfer this register will be updated with the ADDR field of the NEXTCONBK Control Block register. If this field is zero, the DMA will stop. Reading this register will return the address of the currently active CB (in the linked list of CBs). The address must be 256-bit aligned, so the bottom 5 bits of the address must be zero.</p> | RW   | 0x0   |

Table 37. 0\_CONBLK\_AD, 1\_CONBLK\_AD, 2\_CONBLK\_AD, 3\_CONBLK\_AD, 4\_CONBLK\_AD, 5\_CONBLK\_AD, 6\_CONBLK\_AD, 7\_CONBLK\_AD, 8\_CONBLK\_AD, 9\_CONBLK\_AD & 10\_CONBLK\_AD Registers

## 0\_TI, 1\_TI, 2\_TI, 3\_TI, 4\_TI, 5\_TI & 6\_TI Registers

### Synopsis

DMA Transfer Information.

| Bit(s) | Field Name     | Description  | Type | Reset |
|--------|----------------|--|------|-------|
| 31:27  |                | Reserved - Write as 0, read as don't care  |      |       |
| 26     | NO_WIDE_BURSTS | <p>Don't do wide writes as a 2 beat burst</p> <p>This prevents the DMA from issuing wide writes as 2 beat AXI bursts. This is an inefficient access mode, so the default is to use the bursts.</p>                               | RW   | 0x0   |
| 25:21  | WAITS          | <p>Add Wait Cycles</p> <p>This slows down the DMA throughput by setting the number of dummy cycles burnt after each DMA read or write operation is completed.</p> <p>A value of 0 means that no wait cycles are to be added.</p> | RW   | 0x0   |



| Bit(s) | Field Name   | Description   | Type | Reset |
|--------|--------------|---|------|-------|
| 20:16  | PERMAP       | Peripheral Mapping<br>Indicates the peripheral number (1-31) whose ready signal shall be used to control the rate of the transfers, and whose panic signals will be output on the DMA AXI bus. Set to 0 for a continuous un-paced transfer.                         | RW   | 0x0   |
| 15:12  | BURST_LENGTH | Burst Transfer Length<br>Indicates the burst length of the DMA transfers. The DMA will attempt to transfer data as bursts of this number of words. A value of zero will produce a single transfer. Bursts are only produced for specific conditions, see main text. | RW   | 0x0   |
| 11     | SRC_IGNORE   | Ignore Reads<br>1 = Do not perform source reads. In addition, destination writes will zero all the write strobes. This is used for fast cache fill operations.<br>0 = Perform source reads.   | RW   | 0x0   |
| 10     | SRC_DREQ     | Control Source Reads with DREQ<br>1 = The DREQ selected by PERMAP will gate the source reads.<br>0 = DREQ has no effect.  | RW   | 0x0   |
| 9      | SRC_WIDTH    | Source Transfer Width<br>1 = Use 128-bit source read width.<br>0 = Use 32-bit source read width.  | RW   | 0x0   |
| 8      | SRC_INC      | Source Address Increment<br>1 = Source address increments after each read. The address will increment by 4, if SRC_WIDTH=0 else by 32.<br>0 = Source address does not change.   | RW   | 0x0   |
| 7      | DEST_IGNORE  | Ignore Writes<br>1 = Do not perform destination writes.<br>0 = Write data to destination.   | RW   | 0x0   |
| 6      | DEST_DREQ    | Control Destination Writes with DREQ<br>1 = The DREQ selected by PERMAP will gate the destination writes.<br>0 = DREQ has no effect.  | RW   | 0x0   |
| 5      | DEST_WIDTH   | Destination Transfer Width<br>1 = Use 128-bit destination write width.<br>0 = Use 32-bit destination write width.   | RW   | 0x0   |



| Bit(s) | Field Name                                | Description   | Type | Reset |
|--------|---|---|------|-------|
| 4      | DEST_INC                                  | Destination Address Increment<br>1 = Destination address increments after each write.<br>The address will increment by 4, if DEST_WIDTH=0 else by 32.<br>0 = Destination address does not change.   | RW   | 0x0   |
| 3      | WAIT_RESP                                 | Wait for a Write Response<br>When set this makes the DMA wait until it receives the AXI write response for each write. This ensures that multiple writes cannot get stacked in the AXI bus pipeline.<br>1= Wait for the write response to be received before proceeding.<br>0 = Don't wait; continue as soon as the write data is sent. | RW   | 0x0   |
| 2      | Reserved - Write as 0, read as don't care |   |      |       |
| 1      | TDMODE                                    | 2D Mode<br>1 = 2D mode interpret the TXFR_LEN register as YLENGTH number of transfers each of XLENGTH, and add the strides to the address after each transfer.<br>0 = Linear mode interpret the TXFR_LEN register as a single transfer of total length {YLENGTH, XLENGTH}.  | RW   | 0x0   |
| 0      | INTEN                                     | Interrupt Enable<br>1 = Generate an interrupt when the transfer described by the current Control Block completes.<br>0 = Do not generate an interrupt.  | RW   | 0x0   |

Table 38. 0\_TI, 1\_TI, 2\_TI, 3\_TI, 4\_TI, 5\_TI & 6\_TI Registers

## 0\_SOURCE\_AD, 1\_SOURCE\_AD, 2\_SOURCE\_AD, 3\_SOURCE\_AD, 4\_SOURCE\_AD, 5\_SOURCE\_AD, 6\_SOURCE\_AD, 7\_SOURCE\_AD, 8\_SOURCE\_AD, 9\_SOURCE\_AD & 10\_SOURCE\_AD Registers

### Synopsis

DMA Source Address

| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 31:0   | S_ADDR     | DMA Source Address<br>Source address for the DMA operation. Updated by the DMA engine as the transfer progresses. | RW   | 0x0   |

Table 39. 0\_SOURCE\_AD, 1\_SOURCE\_AD, 2\_SOURCE\_AD, 3\_SOURCE\_AD, 4\_SOURCE\_AD, 5\_SOURCE\_AD, 6\_SOURCE\_AD, 7\_SOURCE\_AD, 8\_SOURCE\_AD, 9\_SOURCE\_AD & 10\_SOURCE\_AD Registers



## 0\_DEST\_AD, 1\_DEST\_AD, 2\_DEST\_AD, 3\_DEST\_AD, 4\_DEST\_AD, 5\_DEST\_AD, 6\_DEST\_AD, 7\_DEST\_AD, 8\_DEST\_AD, 9\_DEST\_AD & 10\_DEST\_AD Registers

### Synopsis

DMA Destination Address

| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 31:0   | D_ADDR     | DMA Destination Address<br>Destination address for the DMA operation. Updated by the DMA engine as the transfer progresses. | RW   | 0x0   |

Table 40. 0\_DEST\_AD, 1\_DEST\_AD, 2\_DEST\_AD, 3\_DEST\_AD, 4\_DEST\_AD, 5\_DEST\_AD, 6\_DEST\_AD, 7\_DEST\_AD, 8\_DEST\_AD, 9\_DEST\_AD & 10\_DEST\_AD Registers

## 0\_TXFR\_LEN, 1\_TXFR\_LEN, 2\_TXFR\_LEN, 3\_TXFR\_LEN, 4\_TXFR\_LEN, 5\_TXFR\_LEN & 6\_TXFR\_LEN Registers

### Synopsis

DMA Transfer Length. This specifies the amount of data to be transferred in bytes.

In normal (non 2D) mode this specifies the amount of bytes to be transferred.

In 2D mode it is interpreted as an X and a Y length, and the DMA will perform Y transfers, each of length X bytes and add the strides onto the addresses after each X leg of the transfer.

The length register is updated by the DMA engine as the transfer progresses, so it will indicate the data left to transfer.

| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 31:30  |            | Reserved - Write as 0, read as don't care  |      |       |
| 29:16  | YLENGTH    | When in 2D mode, This is the Y transfer length, indicating how many xlength transfers are performed. When in normal linear mode this becomes the top bits of the XLENGTH | RW   | 0x0   |
| 15:0   | XLENGTH    | Transfer Length in bytes.  | RW   | 0x0   |

Table 41. 0\_TXFR\_LEN, 1\_TXFR\_LEN, 2\_TXFR\_LEN, 3\_TXFR\_LEN, 4\_TXFR\_LEN, 5\_TXFR\_LEN & 6\_TXFR\_LEN Registers

## 0\_STRIDE, 1\_STRIDE, 2\_STRIDE, 3\_STRIDE, 4\_STRIDE, 5\_STRIDE & 6\_STRIDE Registers

### Synopsis

DMA 2D Stride



| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 31:16  | D_STRIDE   | Destination Stride (2D Mode)<br>Signed (2 s complement) byte increment to apply to the destination address at the end of each row in 2D mode. | RW   | 0x0   |
| 15:0   | S_STRIDE   | Source Stride (2D Mode)<br>Signed (2 s complement) byte increment to apply to the source address at the end of each row in 2D mode.           | RW   | 0x0   |

Table 42. 0\_STRIDE, 1\_STRIDE, 2\_STRIDE, 3\_STRIDE, 4\_STRIDE, 5\_STRIDE & 6\_STRIDE Registers

## 0\_NEXTCONBK, 1\_NEXTCONBK, 2\_NEXTCONBK, 3\_NEXTCONBK, 4\_NEXTCONBK, 5\_NEXTCONBK, 6\_NEXTCONBK, 7\_NEXTCONBK, 8\_NEXTCONBK, 9\_NEXTCONBK & 10\_NEXTCONBK Registers

### Synopsis

DMA Next Control Block Address

The value loaded into this register can be overwritten so that the linked list of Control Block data structures can be altered. However it is only safe to do this when the DMA is paused. The address must be 256-bit aligned and so the bottom 5 bits cannot be set and will read back as zero.

| Bit(s) | Field Name | Description                                    | Type | Reset |
|--------|------------|--|------|-------|
| 31:0   | ADDR       | Address of next CB for chained DMA operations. | RW   | 0x0   |

Table 43. 0\_NEXTCONBK, 1\_NEXTCONBK, 2\_NEXTCONBK, 3\_NEXTCONBK, 4\_NEXTCONBK, 5\_NEXTCONBK, 6\_NEXTCONBK, 7\_NEXTCONBK, 8\_NEXTCONBK, 9\_NEXTCONBK & 10\_NEXTCONBK Registers

## 0\_DEBUG, 1\_DEBUG, 2\_DEBUG, 3\_DEBUG, 4\_DEBUG, 5\_DEBUG & 6\_DEBUG Registers

### Synopsis

DMA Debug register.

| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 31:29  |            | Reserved - Write as 0, read as don't care  |      |       |
| 28     | LITE       | DMA Lite<br>Set if the DMA is a reduced performance LITE engine.                                 | RO   | 0x0   |
| 27:25  | VERSION    | DMA Version<br>DMA version number, indicating control bit field changes.                         | RO   | 0x2   |
| 24:16  | DMA_STATE  | DMA State Machine State<br>Returns the value of the DMA engine's state machine for this channel. | RO   | 0x0   |



| Bit(s) | Field Name                                | Description  | Type | Reset |
|--------|---|--|------|-------|
| 15:8   | DMA_ID                                    | DMA ID<br>Returns the DMA AXI ID of this DMA channel.  | RO   | 0x0   |
| 7:4    | OUTSTANDING_WRITES                        | DMA Outstanding Writes Counter<br>Returns the number of write responses that have not yet been received.<br>This count is reset at the start of each new DMA transfer or with a DMA reset. | RO   | 0x0   |
| 3      | Reserved - Write as 0, read as don't care |  |      |       |
| 2      | READ_ERROR                                | Slave Read Response Error<br>Set if the read operation returned an error value on the read response bus. It can be cleared by writing a 1.   | W1C  | 0x0   |
| 1      | FIFO_ERROR                                | FIFO Error<br>Set if the optional read FIFO records an error condition. It can be cleared by writing a 1.  | W1C  | 0x0   |
| 0      | READ_LAST_NOT_SET_ERROR                   | Read Last Not Set Error<br>If the AXI read last signal was not set when expected, then this error bit will be set. It can be cleared by writing a 1.                                       | W1C  | 0x0   |

Table 44. 0\_DEBUG, 1\_DEBUG, 2\_DEBUG, 3\_DEBUG, 4\_DEBUG, 5\_DEBUG & 6\_DEBUG Registers

## 7\_TI, 8\_TI, 9\_TI & 10\_TI Registers

### Synopsis

DMA Lite Transfer Information.

| Bit(s) | Field Name                                | Description   | Type | Reset |
|--------|---|---|------|-------|
| 31:26  | Reserved - Write as 0, read as don't care |   |      |       |
| 25:21  | WAITS                                     | Add Wait Cycles<br>This slows down the DMA throughput by setting the number of dummy cycles burnt after each DMA read or write operation is completed.<br>A value of 0 means that no wait cycles are to be added.                           | RW   | 0x0   |
| 20:16  | PERMAP                                    | Peripheral Mapping<br>Indicates the peripheral number (1-31) whose ready signal shall be used to control the rate of the transfers, and whose panic signals will be output on the DMA AXI bus. Set to 0 for a continuous un-paced transfer. | RW   | 0x0   |



| Bit(s) | Field Name                                | Description   | Type | Reset |
|--------|---|---|------|-------|
| 15:12  | BURST_LENGTH                              | Burst Transfer Length<br>Indicates the burst length of the DMA transfers. The DMA will attempt to transfer data as bursts of this number of words. A value of zero will produce a single transfer. Bursts are only produced for specific conditions, see main text.   | RW   | 0x0   |
| 11     | Reserved - Write as 0, read as don't care |   |      |       |
| 10     | SRC_DREQ                                  | Control Source Reads with DREQ<br>1 = The DREQ selected by PERMAP will gate the source reads.<br>0 = DREQ has no effect.  | RW   | 0x0   |
| 9      | SRC_WIDTH                                 | Source Transfer Width<br>1 = Use 128-bit source read width.<br>0 = Use 32-bit source read width.  | RW   | 0x0   |
| 8      | SRC_INC                                   | Source Address Increment<br>1 = Source address increments after each read. The address will increment by 4, if SRC_WIDTH=0 else by 32.<br>0 = Source address does not change.   | RW   | 0x0   |
| 7      | Reserved - Write as 0, read as don't care |   |      |       |
| 6      | DEST_DREQ                                 | Control Destination Writes with DREQ<br>1 = The DREQ selected by PERMAP will gate the destination writes.<br>0 = DREQ has no effect.  | RW   | 0x0   |
| 5      | DEST_WIDTH                                | Destination Transfer Width<br>1 = Use 128-bit destination write width.<br>0 = Use 32-bit destination write width.   | RW   | 0x0   |
| 4      | DEST_INC                                  | Destination Address Increment<br>1 = Destination address increments after each write. The address will increment by 4, if DEST_WIDTH=0 else by 32.<br>0 = Destination address does not change.  | RW   | 0x0   |
| 3      | WAIT_RESP                                 | Wait for a Write Response<br>When set this makes the DMA wait until it receives the AXI write response for each write. This ensures that multiple writes cannot get stacked in the AXI bus pipeline.<br>1= Wait for the write response to be received before proceeding.<br>0 = Don't wait; continue as soon as the write data is sent. | RW   | 0x0   |



| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 2:1    |            | Reserved - Write as 0, read as don't care  |      |       |
| 0      | INTEN      | Interrupt Enable<br>1 = Generate an interrupt when the transfer described by the current Control Block completes.<br>0 = Do not generate an interrupt. | RW   | 0x0   |

Table 45. 7\_TI, 8\_TI, 9\_TI & 10\_TI Registers

## 7\_TXFR\_LEN, 8\_TXFR\_LEN, 9\_TXFR\_LEN & 10\_TXFR\_LEN Registers

### Synopsis

DMA Lite Transfer Length

| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 31:16  |            | Reserved - Write as 0, read as don't care  |      |       |
| 15:0   | XLENGTH    | Transfer Length<br>Length of transfer, in bytes. Updated by the DMA engine as the transfer progresses. | RW   | 0x0   |

Table 46. 7\_TXFR\_LEN, 8\_TXFR\_LEN, 9\_TXFR\_LEN & 10\_TXFR\_LEN Registers

## 7\_DEBUG, 8\_DEBUG, 9\_DEBUG & 10\_DEBUG Registers

### Synopsis

DMA Lite Debug register.

| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 31:29  |            | Reserved - Write as 0, read as don't care  |      |       |
| 28     | LITE       | DMA Lite<br>Set if the DMA is a reduced performance LITE engine.                                 | RO   | 0x1   |
| 27:25  | VERSION    | DMA Version<br>DMA version number, indicating control bit field changes.                         | RO   | 0x2   |
| 24:16  | DMA_STATE  | DMA State Machine State<br>Returns the value of the DMA engine's state machine for this channel. | RO   | 0x0   |
| 15:8   | DMA_ID     | DMA ID<br>Returns the DMA AXI ID of this DMA channel.  | RO   | 0x0   |





| Bit(s) | Field Name                                | Description  | Type | Reset |
|--------|---|--|------|-------|
| 7:4    | OUTSTANDING_WRITES                        | DMA Outstanding Writes Counter<br>Returns the number of write responses that have not yet been received.<br>This count is reset at the start of each new DMA transfer or with a DMA reset. | RO   | 0x0   |
| 3      | Reserved - Write as 0, read as don't care |  |      |       |
| 2      | READ_ERROR                                | Slave Read Response Error<br>Set if the read operation returned an error value on the read response bus. It can be cleared by writing a 1.   | W1C  | 0x0   |
| 1      | FIFO_ERROR                                | FIFO Error<br>Set if the optional read FIFO records an error condition. It can be cleared by writing a 1.  | W1C  | 0x0   |
| 0      | READ_LAST_NOT_SET_ERROR                   | Read Last Not Set Error<br>If the AXI read last signal was not set when expected, then this error bit will be set. It can be cleared by writing a 1.                                       | W1C  | 0x0   |

Table 47. 7\_DEBUG, 8\_DEBUG, 9\_DEBUG & 10\_DEBUG Registers

## 11\_CS, 12\_CS, 13\_CS & 14\_CS Registers

### Synopsis

DMA4 Control and Status register contains the main control and status bits for this DMA4 channel.

| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 31     | HALT       | Writing a 1 to this bit will cleanly halt the current DMA transfer. The halt will cause the DMA4 to zero its length counters and thus it will complete the current transfer and wait until all outstanding bus activity has finished. The DMA4 will then zero the active flag and return to idle, leaving the address of the aborted CB in the CB reg. The halt bit will self clear when the DMA4 has fully stopped.<br>The Halt bit can be automatically set if the DMA4 detects an error and the debug HALT_ON_ERROR bit is set. | W1SC | 0x0   |



| Bit(s) | Field Name                                | Description   | Type | Reset |
|--------|---|---|------|-------|
| 30     | ABORT                                     | <p>Abort DMA</p> <p>Writing a 1 to this bit will cleanly abort the current DMA transfer. The abort will cause the DMA4 to zero its length counters and thus it will complete the current transfer and wait until all outstanding bus activity has finished.</p> <p>The DMA4 will then check the NEXT_CB address and if it is non zero it will load it into the CB and attempt to continue.</p> <p>The abort bit will self clear when the abort has completed.</p> <p>The abort bit can be automatically set if the DMA4 detects an error and the debug ABORT_ON_ERROR bit is set.</p> | W1SC | 0x0   |
| 29     | DISDEBUG                                  | <p>Disable Debug Pause Signal</p> <p>When set to 1, the DMA4 will not pause when the debug pause signal is asserted.</p> <p>Normally the DMA4 will pause when the debugger asserts the debug_pause DMA control signal. This prevents the DMA4 from running on ahead when the processor is stopped by the debugger. Debug_pause will cleanly pause the DMA4 by preventing it from issuing new commands. Releasing the debug_pause will allow the DMA4 to carry on where it left off.</p>   | RW   | 0x0   |
| 28     | WAIT_FOR_OUTSTANDING_WRITES               | <p>Wait for outstanding writes.</p> <p>The DMA4 keeps a tally of the AXI writes requests going out and the write responses coming in.</p> <p>When set to 1, the DMA4 will complete the current transfer and then wait until the last outstanding write response has been received and the tally has returned to zero. Only then will it indicate that the transfer is complete and set the END flag or if required the INT and move on to the next CB.</p> <p>The number of outstanding writes will be limited by the FIFO size as set by the instantiation parameters.</p>           | RW   | 0x0   |
| 27:26  | Reserved - Write as 0, read as don't care |   |      |       |
| 25     | OUTSTANDING_TRANSACTIONS                  | <p>Indicates that there are outstanding AXI transfers, either outstanding read data or outstanding write responses</p> <p>This just indicates that the outstanding counters in DEBUG2 are &gt;0</p>   | RO   | 0x0   |



| Bit(s) | Field Name                                | Description  | Type | Reset |
|--------|---|--|------|-------|
| 24     | DMA_BUSY                                  | Indicates the DMA4 is BUSY.<br>This indicates that the DMA4 is operating or waiting for outstanding data or otherwise in use.<br>It can be used as an indicator of when it is safe to powersave the DMA4 and turn off all the clocks by using the global DMA_EN bits   | RO   | 0x0   |
| 23:20  | PANIC_QOS                                 | AXI Panic QOS Level<br>Sets the QOS level of AXI bus transactions when the DMA4 is panicking.<br>This value is used when the panic bit of the selected peripheral channel is 1 indicating that the peripheral is in panic mode.<br>Zero is the lowest QOS.   | RW   | 0x0   |
| 19:16  | QOS                                       | AXI QOS Level<br>Sets the QOS level of normal AXI bus transactions. This value is used when the panic bit of the selected peripheral channel is zero or when no peripheral is selected as in a memory to memory transfer.<br>Zero is the lowest QOS.   | RW   | 0x0   |
| 15:11  | Reserved - Write as 0, read as don't care |  |      |       |
| 10     | ERROR                                     | DMA Error<br>Indicates if the DMA4 has detected an error.<br>The error flags are available in the debug register, and are cleared by reading that register.<br>1 = there is error flag set.<br>0 = no errors.  | RO   | 0x0   |
| 9:8    | Reserved - Write as 0, read as don't care |  |      |       |
| 7      | WAITING_FOR_OUTSTANDING_WRITES            | The DMA4 is Waiting for all the Write Response to be returned.<br>If WAIT_FOR_OUTSTANDING_WRITES is enabled, the DMA4 will complete its transfer and then enter a waiting state where it waits for all the outstanding write responses to be returned. When they are all accounted for, the DMA4 will indicate that the transfer is complete and set the END or INT flags and move on to the next CB.<br>1 = The DMA4 is waiting for outstanding bresponses. | RO   | 0x0   |



| Bit(s) | Field Name     | Description   | Type | Reset |
|--------|----------------|---|------|-------|
| 6      | DREQ_STOPS_DMA | <p>DMA Paused by DREQ State</p> <p>This indicates that the DMA4 is currently paused and not transferring data due to the selected DREQ being inactive.</p> <p>The DMA4 has either src_dreq or dest_dreq set in its CB and the permap value will be indicating which of the DREQ line it should select.</p> <p>If this DREQ line is low then the DMA4 will be paused waiting for the peripheral to request more data.</p> <p>1 = DMA is paused.<br/>0 = DMA is running.</p>                | RO   | 0x0   |
| 5      | WR_PAUSED      | <p>DMA Write Paused State</p> <p>Indicates that the DMA4 is currently paused and not writing data.</p> <p>This will occur if: the active bit has been cleared, if the debug_pause signal has been set by the debug block, or the selected peripheral dreq input isn't set and writes are gated by dreq.</p> <p>1 = paused for writes.<br/>0 = running.</p>  | RO   | 0x0   |
| 4      | RD_PAUSED      | <p>DMA read Paused State</p> <p>Indicates that the DMA4 is currently paused and not reading data.</p> <p>This will occur if: the active bit has been cleared or if the debug_pause signal has been set by the debug block or the selected peripheral dreq isn't set and reads are gated by dreq.</p> <p>1 = paused for reads.<br/>0 = running.</p>  | RO   | 0x0   |
| 3      | DREQ           | <p>DREQ State</p> <p>Indicates the state of the selected DREQ (Data Request) signal, i.e. the DREQ selected by the PERMAP field of the transfer info.</p> <p>1 = Requesting data. This will only be valid once the DMA has started and the PERMAP field has been loaded from the CB. It will remain valid, indicating the selected DREQ signal, until a new CB is loaded. If PERMAP is set to zero (un-paced transfer) then this bit will read back as 1.</p> <p>0 = No data request.</p> | RO   | 0x1   |



| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 2      | INT        | <p>Interrupt Status</p> <p>If interrupts are enabled (INTEN is set to 1) the interrupt is set when the transfer for the current CB is completed.</p> <p>If WAIT_FOR_OUTSTANDING_WRITES is enabled, the DMA4 will wait for all the outstanding bresponses before setting the INT bit.</p> <p>Once set it must be manually cleared by writing a 1 to this bit, even if the next CB has INTEN = 0.</p> <p>The interrupt can also be set if the INT_ON_ERROR debug bit is set and an error is detected. An error interrupt won't be set until the current CB has completed.</p> <p>Write 1 to clear.</p> | W1C  | 0x0   |
| 1      | END        | <p>End Flag</p> <p>Set when the transfer described by the current Control Block is complete.</p> <p>If WAIT_FOR_OUTSTANDING_WRITES is enabled, the DMA4 will wait for all the outstanding bresponses before setting the end bit.</p> <p>Once set it must be manually cleared by writing a 1 to this bit.</p> <p>Write 1 to clear.</p>  | W1C  | 0x0   |
| 0      | ACTIVE     | <p>Activate the DMA4</p> <p>This bit enables the DMA4 to start transferring data. The DMA4 will start operating if this bit is set and the CB is non zero.</p> <p>The DMA transfer can be cleanly paused and re-started in mid transfer by clearing and setting this active bit. The DMA4 will pause at a safe AXI transaction point. This bit is automatically cleared at the end of the CB linked List, i.e. after a CB with a NEXTCONBK = 0x0000_0000 has been executed.</p>  | RW   | 0x0   |

Table 48. 11\_CS, 12\_CS, 13\_CS & 14\_CS Registers

## 11\_CB, 12\_CB, 13\_CB & 14\_CB Registers

### Synopsis

DMA4 Control Block Address register.



| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 31:0   | ADDR       | <p>Control Block Address [36:5]</p> <p>This tells the DMA4 where to find a Control Block (CB) stored in memory. The address must be 256-bit aligned, i.e. the bottom 5 address bits are 0.</p> <p>To support a larger address range, the bottom 5 bits of the CB address are not written here i.e. you should write <code>CB_byte_addr&gt;&gt;5</code></p> <p>The DMA4 will not start unless this address is non zero and the ACTIVE bit has been set.</p> <p>Once the CB and Active are set, the DMA4 will start by reading the CB from the given address and loading the data into the relevant CB registers</p> <p>It will then execute the DMA described by the CB regardless of what it is, so if garbage is read then it will execute it.</p> <p>At the end of the DMA transfer described by the CB, the NEXTCONBK field of the CB will be loaded into to this <code>CB_ADDR</code> register and if it is non zero another DMA sequence will begin anew.</p> <p>Reading this register will return the address of the currently active CB.</p> | RW   | 0x0   |

*Table 49. 11\_CB, 12\_CB, 13\_CB & 14\_CB Registers*

## 11\_DEBUG, 12\_DEBUG, 13\_DEBUG & 14\_DEBUG Registers

### Synopsis

DMA4 Debug register.

| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 31:28  | VERSION    | <p>DMA Version</p> <p>DMA version number, indicating control bit field changes.</p>   | RO   | 0x1   |
| 27:24  | ID         | <p>ID</p> <p>Returns the ID of this DMA4. This is also used as the AXI subid</p>  | RO   | 0x0   |
| 23     | RESET      | <p>DMA Reset</p> <p>This is a hard reset of the DMA4 state machine and certain internal registers.</p> <p>Writing a 1 to this bit will reset the DMA4. The bit cannot be read, and will self clear.</p> <p>Using this in the middle of a DMA transfer or when the AXI bus is active or has outstanding transactions will probably be fatal.</p> | W1SC | 0x0   |



| Bit(s) | Field Name       | Description   | Type | Reset |
|--------|------------------|---|------|-------|
| 22     |                  | Reserved - Write as 0, read as don't care   |      |       |
| 21:18  | W_STATE          | Write State Machine State<br>Returns the value of the DMA4 engine's write state machine.<br>W_IDLE = 0<br>W_PRELOAD = 1<br>W_CALC = 2<br>W_WRITE4K = 3<br>W_READFIFO_EMPTY = 4<br>W_WAIT_OUTSTANDING = 5  | RO   | 0x0   |
| 17:14  | R_STATE          | Read State Machine State<br>Returns the value of the DMA4 engine's read state machine.<br>R_IDLE = 0<br>R_WAIT_CB_DATA = 1<br>R_CALC = 2<br>R_READ4K = 3<br>R_READING = 4<br>R_READFIFO_FULL = 5<br>R_WAIT_WRITE_COMPLETE = 6   | RO   | 0x0   |
| 13:12  |                  | Reserved - Write as 0, read as don't care   |      |       |
| 11     | DISABLE_CLK_GATE | Disable the clock gating logic.   | RW   | 0x0   |
| 10     | ABORT_ON_ERROR   | Instruct the DMA4 to ABORT if it detects an error. If any of the error conditions are met then the CS_ABORT bit will be set causing the DMA4 to terminate the current CB and attempt to start the next one.<br>Clearing this bit will allow it to plough on regardless and probably trample over the entire memory. | RW   | 0x1   |
| 9      | HALT_ON_ERROR    | Instruct the DMA4 to HALT if it detects an error. If any of the error conditions are met then the CS_HALT bit will be set causing the DMA4 to stop.<br>This will override the abort on error behaviour.   | RW   | 0x0   |
| 8      | INT_ON_ERROR     | Generate an interrupt if an error is detected<br>This forces the DMA4 to generate an error even if the inten bit in the TI isn't set<br>The int will be generated when the DMA4 finishes the current CB   | RW   | 0x0   |
| 7:4    |                  | Reserved - Write as 0, read as don't care   |      |       |



| Bit(s) | Field Name    | Description   | Type | Reset |
|--------|---------------|---|------|-------|
| 3      | READ_CB_ERROR | Slave Read Response Error During Control Block Read<br>Set if the read operation returned an error value on the read response bus whilst reading the CB.<br>It is cleared by reading. | RC   | 0x0   |
| 2      | READ_ERROR    | Slave Read Response Error<br>Set if the read operation returned an error value on the read response bus during a data read.<br>It is cleared by reading.                              | RC   | 0x0   |
| 1      | FIFO_ERROR    | FIFO Error<br>Set if the optional read FIFO records an error condition (read when empty or write when full).<br>It is cleared by reading.   | RC   | 0x0   |
| 0      | WRITE_ERROR   | Slave Write Response Error<br>Set if a write operation returned an error value on the write response bus.<br>It is cleared by reading.  | RC   | 0x0   |

Table 50. 11\_DEBUG, 12\_DEBUG, 13\_DEBUG & 14\_DEBUG Registers

## 11\_TI, 12\_TI, 13\_TI & 14\_TI Registers

### Synopsis

DMA4 Transfer Information.

| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 31:24  | D_WAITS    | Write Wait Cycles<br>This slows down the DMA throughput by setting the number of dummy cycles before each AXI Write operation is started.<br>A value of 0 means that no wait cycles are to be added. Waits are counted down when the DMA4 wants to do a wait and the bus is available and the writes aren't paused because of DREQS or some other reason.        | RW   | 0x0   |
| 23:16  | S_WAITS    | Read Wait Cycles<br>This slows down the DMA throughput by setting the number of dummy cycles burnt before each DMA AXI read operation is started.<br>A value of 0 means that no wait cycles are to be added. Waits are counted down when the DMA4 wants to do a read and the bus is available and the reads aren't paused because of DREQS or some other reason. | RW   | 0x0   |





| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 15     | D_DREQ     | <p>Control Destination Writes with DREQ.</p> <p>This is used when writing to a peripheral that has a DREQ flow control available to control the data flow. The DMA4 will observe the DREQ input selected by the PERMAP value and pause writes when it is low. Care must be taken when using this as the DMA4 will only stop writing peripheral data a clock cycle or two after it sees a low DREQ at its input.</p> <p>However there may still be outstanding write data in the pipeline formed by the infrastructure between the DMA and the peripheral.</p> <p>The peripheral must take this into account when deciding when to drop its DREQ signal, and must have spare FIFO room to accommodate the data that's still in flight.</p> <p>The WAIT_RESP feature can be used to ensure there is only ever 1 outstanding write at any time for use with peripherals that can't provide any spare storage for any in-flight data.</p> <p>The D_WAITS feature can be used to add a delay before each write to allow the DREQ more time to make it back to the DMA.</p> <p>1 = The DREQ selected by PERMAP will gate the Destination writes.</p> <p>0 = DREQ has no effect.</p> | RW   | 0x0   |



| Bit(s) | Field Name                                | Description  | Type | Reset |
|--------|---|--|------|-------|
| 14     | S_DREQ                                    | <p>Control Source Reads with DREQ</p> <p>This is used when reading from a peripheral that has a DREQ flow control available.</p> <p>The DMA will observe the DREQ input selected by the permap value and pause reads when it is low.</p> <p>Care must be taken when using this as the DMA4 will only stop issuing peripheral read requests a clock cycle or two after it sees a low DREQ at its input.</p> <p>The AXI infrastructure will allow several read requests to become queued outside of the DMA4 engine so it's possible to request far more data than a peripheral can immediately supply.</p> <p>If this happens then the infrastructure may become locked until the data is available and this will adversely affect system performance.</p> <p>The WAIT_RD_RESP option prevents the DMA4 from issuing more than 1 read request at a time, so the amount of data requested can be governed by the burst size, and this allows more time for the peripheral to retract its DREQ when it runs out of data.</p> <p>The S_WAITS feature can be used to add a delay before each read to allow the DREQ more time to make it back to the DMA.</p> <p>1 = The DREQ selected by PERMAP will gate the source reads.</p> <p>0 = DREQ has no effect.</p> | RW   | 0x0   |
| 13:09  | PERMAP                                    | <p>Peripheral Mapping</p> <p>Indicates the DREQ of selected peripheral (1-31).</p> <p>The DMA4 will select the DREQ from this peripheral and use that to control the rate of read or write transfers.</p> <p>The DMA4 will also select the panic signals from this peripheral and use that to set the QOS level on the AXI bus.</p> <p>Setting a PERMAP of 0 selects a dummy peripheral that is always active for a continuous un-paced transfer.</p>  | RW   | 0x0   |
| 8:4    | Reserved - Write as 0, read as don't care |  |      |       |



| Bit(s) | Field Name   | Description   | Type | Reset |
|--------|--------------|---|------|-------|
| 3      | WAIT_RD_RESP | Wait for a Read Response<br>When set this makes the DMA4 wait until it receives all the data from each read. This ensures that multiple reads cannot get stacked in the AXI bus pipeline. This allows the amount of data to be controlled by the burst size, e.g. when reading for a peripheral FIFO<br>1 = Wait for the read data to be received before proceeding.<br>0 = Don't wait; allow multiple reads to be queued.  | RW   | 0x0   |
| 2      | WAIT_RESP    | Wait for a Write Response<br>When set this makes the DMA4 wait until it receives the AXI write response for each write. This ensures that multiple writes cannot get stacked in the AXI bus pipeline.<br>1 = Wait for the write response to be received before proceeding.<br>0 = Don't wait; continue as soon as the write data is sent.   | RW   | 0x0   |
| 1      | TDMODE       | 2D Mode - perform a 2D transfer instead of a normal linear transfer.<br>In 2D mode the DMA4 will interpret the length field as an X and a Y length. It will execute Y+1 transfers each of length X.<br>After each X transfer, the DMA4 will add the value in the STRIDE registers to the source and destination address.<br>If 2D mode isn't selected then the DMA4 interprets the X&Y lengths as a single 30bit length and performs one transfer of that number of bytes.<br>1 = 2D mode - perform Y+1 transfers of X bytes<br>0 = Linear mode interpret the LEN register as a single transfer of total length {YLENGTH, XLENGTH} bytes. | RW   | 0x0   |
| 0      | INTEN        | Interrupt Enable<br>1 = Generate an interrupt when the transfer described by the current Control Block completes.<br>0 = Do not generate an interrupt.  | RW   | 0x0   |

*Table 51. 11\_TI, 12\_TI, 13\_TI & 14\_TI Registers*

## 11\_SRC, 12\_SRC, 13\_SRC & 14\_SRC Registers

### Synopsis

Lower 32 bits of the DMA4 Source Address

The DMA4 can handle up to 40bit addresses so the full source address is split over 2 registers.



| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 31:0   | ADDR       | Lower bits of the Source Address [31:0]<br>This specifies the BYTE address that the DMA4 should read source data from.<br>The address is BYTE aligned allowing transfers from any byte address to any other byte address.<br>This reg value is automatically updated by the DMA4 engine as the transfer progresses, so it indicates the current address being read. | RW   | 0x0   |

Table 52. 11\_SRC, 12\_SRC, 13\_SRC & 14\_SRC Registers

## 11\_SRCI, 12\_SRCI, 13\_SRCI & 14\_SRCI Registers

### Synopsis

DMA4 Source Information

This contains the high bits of the source address[40:32] as well as other source control bits

| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 31:16  | STRIDE     | Source Stride<br>This is only used in 2D transfer mode (TDMODE).<br>In a 2D transfer the DMA4 will perform Y transfers each of X bytes. At the end of each X row, the source stride is added to the source address and this is used as the start address of the source data for the next X row.<br>The source stride is a signed (2 s complement) byte increment so negative values are allowed. | RW   | 0x0   |
| 15     | IGNORE     | Ignore Reads.<br>The DMA4 will perform a normal transfer except that it will not produce any reads. The DMA4 will write zero data.<br>1 = Do not perform source reads.<br>0 = Perform source reads.  | RW   | 0x0   |
| 14:13  | SIZE       | Source Transfer Width<br>The DMA4 will perform all AXI source reads with this AXI transfer width. Data will be fetched in bursts of this width and assembled into the correct data size inside the DMA4. On the BCM2711 the width cannot be set larger than 128 bits.<br>3 = 256<br>2 = 128<br>1 = 64<br>0 = 32  | RW   | 0x0   |



| Bit(s) | Field Name   | Description   | Type | Reset |
|--------|--------------|---|------|-------|
| 12     | INC          | Increment the Source Address<br>1 = Source address increments after each read. The address will increment by the number of bytes in the transfer width.<br>0 = Source address does not change. Data will always be read from the same source address with an AXI "Fixed" transfer. This is intended to be used to read from a peripheral FIFO type of source. | RW   | 0x0   |
| 11:08  | BURST_LENGTH | Burst Transfer Length<br>Indicates the maximum burst length of the source reads. The DMA4 will attempt to transfer data as bursts of this number of words unless it will cause a 4k crossing or there isn't enough data required.<br>A value of zero will produce a single-beat transfer.   | RW   | 0x0   |
| 7:0    | ADDR         | High Bits of the Source Address [40:32]<br>The source address is split over 2 registers, and together they give a 40-bit address  | RW   | 0x0   |

Table 53. 11\_SRCI, 12\_SRCI, 13\_SRCI & 14\_SRCI Registers

## 11\_DEST, 12\_DEST, 13\_DEST & 14\_DEST Registers

### Synopsis

Lower 32 bits of the DMA4 Destination Address

The DMA4 can handle up to 40bit addresses so the full address is split over 2 registers.

| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 31:0   | ADDR       | Destination Address<br>This specifies the BYTE address that the DMA4 should write data to.<br>The address is BYTE aligned allowing transfers from any byte address to any other byte address.<br>This register value is automatically updated by the DMA4 engine as the transfer progresses, so it indicates the current address being written. | RW   | 0x0   |

Table 54. 11\_DEST, 12\_DEST, 13\_DEST & 14\_DEST Registers

## 11\_DESTI, 12\_DESTI, 13\_DESTI & 14\_DESTI Registers

### Synopsis

DMA4 Destination Information

This contains the high bits of the destination address [40:32] and other information bits for the destination



| Bit(s) | Field Name   | Description   | Type | Reset |
|--------|--------------|---|------|-------|
| 31:16  | STRIDE       | Destination Stride<br>This is only used in 2D transfer mode.<br>In a 2D transfer the DMA4 will perform Y transfers each of X bytes. At the end of each X row, the destination stride is added to the destination address and this is used as the start address of the destination for the next X row.<br>The destination stride is a signed (2 s complement) byte increment so negative values are allowed. | RW   | 0x0   |
| 15     | IGNORE       | Ignore Destination Writes<br>1 = Do not perform destination Writes. The DMA4 will read the source data but not write it.<br>0 = Perform destination Writes.   | RW   | 0x0   |
| 14:13  | SIZE         | Destination Transfer Width<br>The DMA4 will perform all AXI destination writes with this AXI transfer width. Data will be written in bursts of this width. On the BCM2711 the width cannot be set larger than 128 bits.<br>3 = 256<br>2 = 128<br>1 = 64<br>0 = 32   | RW   | 0x0   |
| 12     | INC          | Destination Address Increment<br>1 = Destination address increments after each write.<br>The address will increment by by the number bytes in the transfer width.<br>0 = Destination address does not change.   | RW   | 0x0   |
| 11:08  | BURST_LENGTH | Burst Transfer Length<br>Indicates the maximum burst length of the destination writes. The DMA4 will attempt to transfer data as bursts of this number of words unless it will cause a 4k crossing or there isn't enough data required.<br>A value of zero will produce a single transfer.  | RW   | 0x0   |
| 7:0    | ADDR         | High Bits of the Destination Address [40:32]<br>The destination address is split over 2 registers, and together they give a 40-bit address  | RW   | 0x0   |

Table 55. 11\_DESTI, 12\_DESTI, 13\_DESTI & 14\_DESTI Registers

## 11\_LEN, 12\_LEN, 13\_LEN & 14\_LEN Registers

### Synopsis

DMA4 Transfer Length.



This specifies the amount of data to be transferred in bytes.

In normal (non 2D) mode the X&Y are combined to specifies the number of bytes to be transferred up to a max of  $2^{30}-1$ .

In 2D mode it is interpreted as an X and a Y length, and the DMA4 will perform Y+1 transfers, each of length X bytes.

In 2D mode the source and destination strides are added onto the source and destination addresses after each X leg of the transfer.

The length register is updated by the DMA4 engine as the transfer progresses, so it will indicate the data left to transfer.

| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 31:30  |            | Reserved - Write as 0, read as don't care   |      |       |
| 29:16  | YLENGTH    | When in 2D mode, This is the Y transfer length, indicating how many xlength transfers are performed. When in normal linear mode this becomes the top bits of the XLENGTH<br>In 2D mode a value of 0 will result in a single XLENGTH transfer and a value of 1 will result in 2 XLENGTH transfers. | RW   | 0x0   |
| 15:0   | XLENGTH    | Transfer Length in bytes.<br>A value of 1 will transfer 1 byte<br>A value of 0 is illegal   | RW   | 0x0   |

Table 56. 11\_LEN, 12\_LEN, 13\_LEN & 14\_LEN Registers

## 11\_NEXT\_CB, 12\_NEXT\_CB, 13\_NEXT\_CB & 14\_NEXT\_CB Registers

### Synopsis

DMA4 Next Control Block Address

When the current DMA transfer has completed, the Next Control Block address is transferred to the CB address register and if the active bit is still set the next DMA in the linked list of CBs is begun.

A CB with a Next Control Block Address of 0 indicates the end of the list. Once that CB is executed the zero next CB will be loaded and the DMA will stop (as the start condition for the DMA4 is (ACTIVE & CB!=0)).

The value loaded into this register can be overwritten so that the linked list of Control Block data structures can be dynamically altered. However it is only safe to do this when the DMA4 is paused.

The address must be 256-bit aligned and so the bottom 5 bits of the byte address are discarded, i.e. write `cb_byte_address[39:0]>>5` into the CB.

| Bit(s) | Field Name | Description                                    | Type | Reset |
|--------|------------|--|------|-------|
| 31:0   | ADDR       | Address of next CB for chained DMA operations. | RW   | 0x0   |

Table 57. 11\_NEXT\_CB, 12\_NEXT\_CB, 13\_NEXT\_CB & 14\_NEXT\_CB Registers

## 11\_DEBUG2, 12\_DEBUG2, 13\_DEBUG2 & 14\_DEBUG2 Registers



## Synopsis

DMA4 Debug2 register.

| Bit(s) | Field Name         | Description   | Type | Reset |
|--------|--------------------|---|------|-------|
| 31:25  |                    | Reserved - Write as 0, read as don't care   |      |       |
| 24:16  | OUTSTANDING_READS  | Outstanding read Words Count<br>This indicates the number of outstanding read words.<br>This keeps count of the number of read words that have been requested and the number that have actually been returned.<br>This should be zero at the end of every transfer                  | RO   | 0x0   |
| 15:09  |                    | Reserved - Write as 0, read as don't care   |      |       |
| 8:0    | OUTSTANDING_WRITES | Outstanding Write Response Count<br>This indicates the number of outstanding write responses.<br>This keeps count of the number of write requests that have been generated and the number of responses that have been returned.<br>This should be zero at the end of every transfer | RO   | 0x0   |

Table 58. 11\_DEBUG2, 12\_DEBUG2, 13\_DEBUG2 & 14\_DEBUG2 Registers

## INT\_STATUS Register

### Synopsis

Interrupt status of each DMA engine

| Bit(s) | Field Name | Description                               | Type | Reset |
|--------|------------|---|------|-------|
| 31:16  |            | Reserved - Write as 0, read as don't care |      |       |
| 15     | INT15      | Interrupt status of DMA engine 15         | RO   | 0x0   |
| 14     | INT14      | Interrupt status of DMA engine 14         | RO   | 0x0   |
| 13     | INT13      | Interrupt status of DMA engine 13         | RO   | 0x0   |
| 12     | INT12      | Interrupt status of DMA engine 12         | RO   | 0x0   |
| 11     | INT11      | Interrupt status of DMA engine 11         | RO   | 0x0   |
| 10     | INT10      | Interrupt status of DMA engine 10         | RO   | 0x0   |
| 9      | INT9       | Interrupt status of DMA engine 9          | RO   | 0x0   |
| 8      | INT8       | Interrupt status of DMA engine 8          | RO   | 0x0   |
| 7      | INT7       | Interrupt status of DMA engine 7          | RO   | 0x0   |
| 6      | INT6       | Interrupt status of DMA engine 6          | RO   | 0x0   |
| 5      | INT5       | Interrupt status of DMA engine 5          | RO   | 0x0   |





| Bit(s) | Field Name | Description                      | Type | Reset |
|--------|------------|----------------------------------|------|-------|
| 4      | INT4       | Interrupt status of DMA engine 4 | RO   | 0x0   |
| 3      | INT3       | Interrupt status of DMA engine 3 | RO   | 0x0   |
| 2      | INT2       | Interrupt status of DMA engine 2 | RO   | 0x0   |
| 1      | INT1       | Interrupt status of DMA engine 1 | RO   | 0x0   |
| 0      | INT0       | Interrupt status of DMA engine 0 | RO   | 0x0   |

Table 59. INT\_STATUS Register

## ENABLE Register

### Synopsis

Global enable bits for each channel.

Setting these to 0 will disable the DMA for power saving reasons. Disabling whilst the DMA is operating will be fatal.

| Bit(s) | Field Name                                | Description  | Type | Reset |
|--------|---|--|------|-------|
| 31:28  | PAGELITE                                  | Set the 1G SDRAM ram page that the DMA Lite engines (DMA7-10) will access when addressing the 1G uncached range C000_0000 → ffff_ffff<br>E.g. setting this to 1 will mean that when the DMA writes to C000_0000 (uncached) the final address in SDRAM will be 4000_0000 ( pagelite<<30   addr[29:0] )<br>This allows the 1G uncached page to be moved around the 16G SDRAM space | RW   | 0x0   |
| 27:24  | PAGE                                      | Set the 1G SDRAM ram page that the 30-bit DMA engines (DMA0-6) will access when addressing the 1G uncached range C000_0000 → ffff_ffff<br>E.g. setting this to 1 will mean that when the DMA writes to C000_0000 (uncached) the final address in SDRAM will be 4000_0000 ( page<<30   addr[29:0] )<br>This allows the 1G uncached page to be moved around the 16G SDRAM space    | RW   | 0x0   |
| 23:15  | Reserved - Write as 0, read as don't care |  |      |       |
| 14     | EN14                                      | Enable DMA engine 14   | RW   | 0x1   |
| 13     | EN13                                      | Enable DMA engine 13   | RW   | 0x1   |
| 12     | EN12                                      | Enable DMA engine 12   | RW   | 0x1   |
| 11     | EN11                                      | Enable DMA engine 11   | RW   | 0x1   |
| 10     | EN10                                      | Enable DMA engine 10   | RW   | 0x1   |
| 9      | EN9                                       | Enable DMA engine 9  | RW   | 0x1   |



| Bit(s) | Field Name | Description         | Type | Reset |
|--------|------------|---------------------|------|-------|
| 8      | EN8        | Enable DMA engine 8 | RW   | 0x1   |
| 7      | EN7        | Enable DMA engine 7 | RW   | 0x1   |
| 6      | EN6        | Enable DMA engine 6 | RW   | 0x1   |
| 5      | EN5        | Enable DMA engine 5 | RW   | 0x1   |
| 4      | EN4        | Enable DMA engine 4 | RW   | 0x1   |
| 3      | EN3        | Enable DMA engine 3 | RW   | 0x1   |
| 2      | EN2        | Enable DMA engine 2 | RW   | 0x1   |
| 1      | EN1        | Enable DMA engine 1 | RW   | 0x1   |
| 0      | EN0        | Enable DMA engine 0 | RW   | 0x1   |

*Table 60. ENABLE Register*

## Peripheral DREQ Signals

A DREQ (Data Request) mechanism is used to pace the data flow between the DMA and a peripheral.

Each peripheral is allocated a permanent DREQ signal. Each DMA channel can select which of the DREQ signals should be used to pace the transfer by controlling the DMA reads, DMA writes or both. Note that DREQ 0 is permanently enabled and can be used if no DREQ is required.

When a DREQ signal is being used to pace the DMA reads, the DMA will wait until it has sampled DREQ high before launching a single or burst read operation. It will then wait for all the read data to be returned before re-checking the DREQ and starting the next read. Thus once a peripheral receives the read request it should remove its DREQ as soon as possible to prevent the DMA from re-sampling the same DREQ assertion.

DREQs are not required when reading from AXI peripherals. In this case, the DMA will request data from the peripheral and the peripheral will only send the data when it is available. The DMA will not request data that it does not have room for, so no pacing of the data flow is required.

DREQs are required when reading from APB peripherals as the AXI-to-APB bridge will not wait for an APB peripheral to be ready and will just perform the APB read regardless. Thus an APB peripheral needs to make sure that it has all of its read data ready before it drives its DREQ high.

When writing to peripherals, a DREQ is always required to pace the data. However, due to the pipelined nature of the AXI bus system, several writes may be in flight before the peripheral receives any data and withdraws its DREQ signal. Thus the peripheral must ensure that it has sufficient room in its input FIFO to accommodate the maximum amount of data that it might receive. If the peripheral is unable to do this, the DMA WAIT\_RESP mechanism can be used to ensure that only one write is in flight at any one time, however this is a less efficient transfer mechanism.

The mapping of peripherals to DREQs is as follows:



| DREQ | Peripheral  |
|------|---|
| 0    | DREQ = 1<br>This is always on so use this channel if no DREQ is required. |
| 1    | DSIO / PWM1 **  |
| 2    | PCM TX  |
| 3    | PCM RX  |
| 4    | SMI   |
| 5    | PWM0  |
| 6    | SPI0 TX   |
| 7    | SPI0 RX   |
| 8    | BSC/SPI Slave TX  |
| 9    | BSC/SPI Slave RX  |
| 10   | HDMI0   |
| 11   | eMMC  |
| 12   | UART0 TX  |
| 13   | SD HOST   |
| 14   | UART0 RX  |
| 15   | DSI1  |
| 16   | SPI1 TX   |
| 17   | HDMI1   |
| 18   | SPI1 RX   |
| 19   | UART3 TX / SPI4 TX **   |
| 20   | UART3 RX / SPI4 RX **   |
| 21   | UART5 TX / SPI5 TX **   |
| 22   | UART5 RX / SPI5 RX **   |
| 23   | SPI6 TX   |
| 24   | Scaler FIFO 0 & SMI *   |
| 25   | Scaler FIFO 1 & SMI *   |
| 26   | Scaler FIFO 2 & SMI *   |
| 27   | SPI6 RX   |
| 28   | UART2 TX  |
| 29   | UART2 RX  |



| DREQ | Peripheral |
|------|------------|
| 30   | UART4 TX   |
| 31   | UART4 RX   |

\* The SMI element of the Scaler FIFO 0 & SMI DREQs can be disabled by setting the SMI\_DISABLE bit in the DMA\_DREQ\_CONTROL register in the system arbiter control block.

\*\* The alternate DREQs are available by changing the DMA\_CNTRL\_MUX bits in the PACTL\_CS register in the peri audio control block.

## 4.3. AXI Bursts

The DMA supports bursts under specific conditions. Up to 16 beat bursts can be accommodated.

Peripheral (32-bit wide) read bursts are supported. The DMA will generate the burst if there is sufficient room in its read buffer to accommodate all the data from the burst. This limits the burst size to a maximum of 8 beats.

Read bursts in destination ignore mode (DEST\_IGNORE) are supported as there is no need for the DMA to deal with the data. This allows wide bursts of up to 16 beats to be used for efficient L2 cache fills.

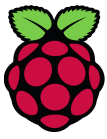
DMA channel 0 and 15 are fitted with an external 128-bit 8 word read FIFO. This enables efficient memory to memory transfers to be performed. This FIFO allows the DMA to accommodate a wide read burst up to the size of the FIFO. In practice this will allow a 128-bit wide read burst of 9 as the first word back will be immediately read into the DMA engine (or a 32-bit peripheral read burst of 16: 8 in the input buffer and 8 in the FIFO). On any DMA channel, if a read burst is selected that is too large, the AXI read bus will be stalled until the DMA has written out the data. This may lead to inefficient system operation, and possibly AXI lock up if it causes a circular dependency.

In general write bursts are not supported. However to increase the efficiency of L2 cache fills, source ignore (SRC\_IGNORE) transfers can be specified with a write burst. In this case the DMA will issue a write burst address sequence followed by the appropriate number of zero data, zero strobe write bus cycles, which will cause the cache to pre-fetch the data. To improve the efficiency of the 128-bit wide bus architecture, and to make use of the DMA's internal 256-bit registers, the DMA will generate 128-bit wide writes as 2 beat bursts wherever possible, although this behaviour can be disabled.

## 4.4. Error Handling

If the DMA detects a Read Response error it will record the fact in the READ\_ERROR flag in the debug register. This will remain set until it is cleared by writing a 1 to it. The DMA will clear its active flag and generate an interrupt. Any outstanding read data transactions (remainder of a burst) will be honoured. This allows the operator to either restart the DMA by clearing the error bit and setting the active bit, or to abort the DMA transfer by clearing the NEXTCONBK register and restarting the DMA with the ABORT bit set.

The DMA will also record any errors from an external read FIFO. These will be latched in the



FIFO\_ERROR bit in the debug register until they are cleared by writing a ‘1’ to the bit. (note that only DMA0 and 15 have an external read FIFO)

If the DMA detects that a read occurred without the AXI rlast signal being set as expected then it will set the READ\_LAST\_NOT\_SET\_ERROR bit in the debug register. This can be cleared by writing a ‘1’ to it.

The error bits are logically OR-ed together and presented as a general ERROR bit in the CS register.

## 4.5. DMA LITE Engines

Several of the DMA engines are of the LITE design. This is a reduced specification engine designed to save space. The engine behaves in the same way as a normal DMA engine except for the following differences:

1. The internal data structure is 128 bits instead of 256 bits. This means that if you do a 128-bit wide read burst of more than 1 beat, the DMA input register will be full and the read bus will be stalled. The normal DMA engine can accept a read burst of 2 without stalling. If you do a narrow 32-bit read burst from the peripherals then the lite engine can cope with a burst of 4 as opposed to a burst of 8 for the normal engine. Note that stalling the read bus will potentially reduce the overall system performance, and may possibly cause a system lockup if you end up with a conflict where the DMA cannot free the read bus as the read stall has prevented it writing out its data due to some circular system relationship.
2. The Lite engine does not support 2D transfers. The TDMODE, S\_STRIDE, D\_STRIDE and YLENGTH registers will all be removed. Setting these registers will have no effect.
3. The DMA length register is now 16 bits, limiting the maximum transferable length to 65536 bytes.
4. Source ignore (SRC\_IGNORE) and destination ignore (DEST\_IGNORE) modes are removed. The Lite engine will have about half the bandwidth of a normal DMA engine, and are intended for low bandwidth peripheral servicing.

## 4.6. DMA4 Engines

Several of the DMA engines are of the DMA4 design. These have higher performance due to their uncoupled read/write design and can access up to 40 address bits. Unlike the other DMA engines they are also capable of performing write bursts. Note that they directly access the full 35-bit address bus of the BCM2711 and so bypass the paging registers of the DMA and DMA Lite engines.

DMA channel 11 is additionally able to access the PCIe interface.

[5] Or the CB register in the DMA4 channels

[6] Or the ADDR field of the CB register in the DMA4 channels

[7] Or the NEXT\_CB register in the DMA4 channels





the event detect status register. Each bank has its own interrupt line with the fourth line shared between all bits.

The Alternate function table ([Table 92](#)) also has the pull state (pull-up/pull-down) which is applied after a power down.

## 5.2. Register View

The GPIO has the following registers. All accesses are assumed to be 32-bit. The GPIO register base address is ~~0x7E21 5000~~.

**0x7E20 0000**

| Address Offset | Register Name           | Description                           | Size |
|----------------|-------------------------|---------------------------------------|------|
| 0x00           | <a href="#">GPFSEL0</a> | GPIO Function Select 0                | 32   |
| 0x04           | <a href="#">GPFSEL1</a> | GPIO Function Select 1                | 32   |
| 0x08           | <a href="#">GPFSEL2</a> | GPIO Function Select 2                | 32   |
| 0x0C           | <a href="#">GPFSEL3</a> | GPIO Function Select 3                | 32   |
| 0x10           | <a href="#">GPFSEL4</a> | GPIO Function Select 4                | 32   |
| 0x14           | <a href="#">GPFSEL5</a> | GPIO Function Select 5                | 32   |
| 0x18           | -                       | Reserved                              | -    |
| 0x1C           | <a href="#">GPSET0</a>  | GPIO Pin Output Set 0                 | 32   |
| 0x20           | <a href="#">GPSET1</a>  | GPIO Pin Output Set 1                 | 32   |
| 0x24           | -                       | Reserved                              | -    |
| 0x28           | <a href="#">GPCLR0</a>  | GPIO Pin Output Clear 0               | 32   |
| 0x2C           | <a href="#">GPCLR1</a>  | GPIO Pin Output Clear 1               | 32   |
| 0x30           | -                       | Reserved                              | -    |
| 0x34           | <a href="#">GPLEV0</a>  | GPIO Pin Level 0                      | 32   |
| 0x38           | <a href="#">GPLEV1</a>  | GPIO Pin Level 1                      | 32   |
| 0x3C           | -                       | Reserved                              | -    |
| 0x40           | <a href="#">GPEDS0</a>  | GPIO Pin Event Detect Status 0        | 32   |
| 0x44           | <a href="#">GPEDS1</a>  | GPIO Pin Event Detect Status 1        | 32   |
| 0x48           | -                       | Reserved                              | -    |
| 0x4C           | <a href="#">GPREN0</a>  | GPIO Pin Rising Edge Detect Enable 0  | 32   |
| 0x50           | <a href="#">GPREN1</a>  | GPIO Pin Rising Edge Detect Enable 1  | 32   |
| 0x54           | -                       | Reserved                              | -    |
| 0x58           | <a href="#">GPFEN0</a>  | GPIO Pin Falling Edge Detect Enable 0 | 32   |



| Address Offset | Register Name                           | Description                           | Size |
|----------------|---|---------------------------------------|------|
| 0x5C           | <a href="#">GPFEN1</a>                  | GPIO Pin Falling Edge Detect Enable 1 | 32   |
| 0x60           | -                                       | Reserved                              | -    |
| 0x64           | <a href="#">GPHEN0</a>                  | GPIO Pin High Detect Enable 0         | 32   |
| 0x68           | <a href="#">GPHEN1</a>                  | GPIO Pin High Detect Enable 1         | 32   |
| 0x6C           | -                                       | Reserved                              | -    |
| 0x70           | <a href="#">GPLEN0</a>                  | GPIO Pin Low Detect Enable 0          | 32   |
| 0x74           | <a href="#">GPLEN1</a>                  | GPIO Pin Low Detect Enable 1          | 32   |
| 0x78           | -                                       | Reserved                              | -    |
| 0x7C           | <a href="#">GPAREN0</a>                 | GPIO Pin Async. Rising Edge Detect 0  | 32   |
| 0x80           | <a href="#">GPAREN1</a>                 | GPIO Pin Async. Rising Edge Detect 1  | 32   |
| 0x84           | -                                       | Reserved                              | -    |
| 0x88           | <a href="#">GPAFEN0</a>                 | GPIO Pin Async. Falling Edge Detect 0 | 32   |
| 0x8C           | <a href="#">GPAFEN1</a>                 | GPIO Pin Async. Falling Edge Detect 1 | 32   |
| 0x90           | -                                       | Reserved                              | -    |
| 0xE4           | <a href="#">GPIO_PUP_PDN_CNTRL_REG0</a> | GPIO Pull-up / Pull-down Register 0   | 32   |
| 0xE8           | <a href="#">GPIO_PUP_PDN_CNTRL_REG1</a> | GPIO Pull-up / Pull-down Register 1   | 32   |
| 0xEC           | <a href="#">GPIO_PUP_PDN_CNTRL_REG2</a> | GPIO Pull-up / Pull-down Register 2   | 32   |
| 0xF0           | <a href="#">GPIO_PUP_PDN_CNTRL_REG3</a> | GPIO Pull-up / Pull-down Register 3   | 32   |

Table 61. GPIO Register Assignment

## GPFSEL0 Register

### Synopsis

The function select registers are used to define the operation of the general-purpose I/O pins. Each of the 58 GPIO pins has at least two alternative functions as defined in [Section 5.3](#). The FSEL $n$  field determines the functionality of the  $n$ th GPIO pin. All unused alternative function lines are tied to ground and will output a “0” if selected. All pins reset to normal GPIO input operation.

| Bit(s) | Field Name | Description                               | Type | Reset |
|--------|------------|---|------|-------|
| 31:30  |            | Reserved - Write as 0, read as don't care |      |       |





| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 29:27  | FSEL9      | FSEL9 - Function Select 9<br>000 = GPIO Pin 9 is an input<br>001 = GPIO Pin 9 is an output<br>100 = GPIO Pin 9 takes alternate function 0<br>101 = GPIO Pin 9 takes alternate function 1<br>110 = GPIO Pin 9 takes alternate function 2<br>111 = GPIO Pin 9 takes alternate function 3<br>011 = GPIO Pin 9 takes alternate function 4<br>010 = GPIO Pin 9 takes alternate function 5 | RW   | 0     |
| 26:24  | FSEL8      | FSEL8 - Function Select 8  | RW   | 0     |
| 23:21  | FSEL7      | FSEL7 - Function Select 7  | RW   | 0     |
| 20:18  | FSEL6      | FSEL6 - Function Select 6  | RW   | 0     |
| 17:15  | FSEL5      | FSEL5 - Function Select 5  | RW   | 0     |
| 14:12  | FSEL4      | FSEL4 - Function Select 4  | RW   | 0     |
| 11:9   | FSEL3      | FSEL3 - Function Select 3  | RW   | 0     |
| 8:6    | FSEL2      | FSEL2 - Function Select 2  | RW   | 0     |
| 5:3    | FSEL1      | FSEL1 - Function Select 1  | RW   | 0     |
| 2:0    | FSEL0      | FSEL0 - Function Select 0  | RW   | 0     |

Table 62. GPIO Alternate function select register 0

## GPFSEL1 Register

| Bit(s) | Field Name                                | Description  | Type | Reset |
|--------|---|--|------|-------|
| 31:30  | Reserved - Write as 0, read as don't care |  |      |       |
| 29:27  | FSEL19                                    | FSEL19 - Function Select 19<br>000 = GPIO Pin 19 is an input<br>001 = GPIO Pin 19 is an output<br>100 = GPIO Pin 19 takes alternate function 0<br>101 = GPIO Pin 19 takes alternate function 1<br>110 = GPIO Pin 19 takes alternate function 2<br>111 = GPIO Pin 19 takes alternate function 3<br>011 = GPIO Pin 19 takes alternate function 4<br>010 = GPIO Pin 19 takes alternate function 5 | RW   | 0     |
| 26:24  | FSEL18                                    | FSEL18 - Function Select 18  | RW   | 0     |
| 23:21  | FSEL17                                    | FSEL17 - Function Select 17  | RW   | 0     |
| 20:18  | FSEL16                                    | FSEL16 - Function Select 16  | RW   | 0     |
| 17:15  | FSEL15                                    | FSEL15 - Function Select 15  | RW   | 0     |
| 14:12  | FSEL14                                    | FSEL14 - Function Select 14  | RW   | 0     |



| Bit(s) | Field Name | Description                 | Type | Reset |
|--------|------------|-----------------------------|------|-------|
| 11:9   | FSEL13     | FSEL13 - Function Select 13 | RW   | 0     |
| 8:6    | FSEL12     | FSEL12 - Function Select 12 | RW   | 0     |
| 5:3    | FSEL11     | FSEL11 - Function Select 11 | RW   | 0     |
| 2:0    | FSEL10     | FSEL10 - Function Select 10 | RW   | 0     |

Table 63. GPIO Alternate function select register 1

## GPFSEL2 Register

| Bit(s) | Field Name                                | Description  | Type | Reset |
|--------|---|--|------|-------|
| 31:30  | Reserved - Write as 0, read as don't care |  |      |       |
| 29:27  | FSEL29                                    | FSEL29 - Function Select 29<br>000 = GPIO Pin 29 is an input<br>001 = GPIO Pin 29 is an output<br>100 = GPIO Pin 29 takes alternate function 0<br>101 = GPIO Pin 29 takes alternate function 1<br>110 = GPIO Pin 29 takes alternate function 2<br>111 = GPIO Pin 29 takes alternate function 3<br>011 = GPIO Pin 29 takes alternate function 4<br>010 = GPIO Pin 29 takes alternate function 5 | RW   | 0     |
| 26:24  | FSEL28                                    | FSEL28 - Function Select 28  | RW   | 0     |
| 23:21  | FSEL27                                    | FSEL27 - Function Select 27  | RW   | 0     |
| 20:18  | FSEL26                                    | FSEL26 - Function Select 26  | RW   | 0     |
| 17:15  | FSEL25                                    | FSEL25 - Function Select 25  | RW   | 0     |
| 14:12  | FSEL24                                    | FSEL24 - Function Select 24  | RW   | 0     |
| 11:9   | FSEL23                                    | FSEL23 - Function Select 23  | RW   | 0     |
| 8:6    | FSEL22                                    | FSEL22 - Function Select 22  | RW   | 0     |
| 5:3    | FSEL21                                    | FSEL21 - Function Select 21  | RW   | 0     |
| 2:0    | FSEL20                                    | FSEL20 - Function Select 20  | RW   | 0     |

Table 64. GPIO Alternate function select register 2

## GPFSEL3 Register

| Bit(s) | Field Name                                | Description | Type | Reset |
|--------|---|-------------|------|-------|
| 31:30  | Reserved - Write as 0, read as don't care |             |      |       |



| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 29:27  | FSEL39     | FSEL39 - Function Select 39<br>000 = GPIO Pin 39 is an input<br>001 = GPIO Pin 39 is an output<br>100 = GPIO Pin 39 takes alternate function 0<br>101 = GPIO Pin 39 takes alternate function 1<br>110 = GPIO Pin 39 takes alternate function 2<br>111 = GPIO Pin 39 takes alternate function 3<br>011 = GPIO Pin 39 takes alternate function 4<br>010 = GPIO Pin 39 takes alternate function 5 | RW   | 0     |
| 26:24  | FSEL38     | FSEL38 - Function Select 38  | RW   | 0     |
| 23:21  | FSEL37     | FSEL37 - Function Select 37  | RW   | 0     |
| 20:18  | FSEL36     | FSEL36 - Function Select 36  | RW   | 0     |
| 17:15  | FSEL35     | FSEL35 - Function Select 35  | RW   | 0     |
| 14:12  | FSEL34     | FSEL34 - Function Select 34  | RW   | 0     |
| 11:9   | FSEL33     | FSEL33 - Function Select 33  | RW   | 0     |
| 8:6    | FSEL32     | FSEL32 - Function Select 32  | RW   | 0     |
| 5:3    | FSEL31     | FSEL31 - Function Select 31  | RW   | 0     |
| 2:0    | FSEL30     | FSEL30 - Function Select 30  | RW   | 0     |

Table 65. GPIO Alternate function select register 3

## GPFSEL4 Register

| Bit(s) | Field Name                                | Description  | Type | Reset |
|--------|---|--|------|-------|
| 31:30  | Reserved - Write as 0, read as don't care |  |      |       |
| 29:27  | FSEL49                                    | FSEL49 - Function Select 49<br>000 = GPIO Pin 49 is an input<br>001 = GPIO Pin 49 is an output<br>100 = GPIO Pin 49 takes alternate function 0<br>101 = GPIO Pin 49 takes alternate function 1<br>110 = GPIO Pin 49 takes alternate function 2<br>111 = GPIO Pin 49 takes alternate function 3<br>011 = GPIO Pin 49 takes alternate function 4<br>010 = GPIO Pin 49 takes alternate function 5 | RW   | 0     |
| 26:24  | FSEL48                                    | FSEL48 - Function Select 48  | RW   | 0     |
| 23:21  | FSEL47                                    | FSEL47 - Function Select 47  | RW   | 0     |
| 20:18  | FSEL46                                    | FSEL46 - Function Select 46  | RW   | 0     |
| 17:15  | FSEL45                                    | FSEL45 - Function Select 45  | RW   | 0     |
| 14:12  | FSEL44                                    | FSEL44 - Function Select 44  | RW   | 0     |



| Bit(s) | Field Name | Description                 | Type | Reset |
|--------|------------|-----------------------------|------|-------|
| 11:9   | FSEL43     | FSEL43 - Function Select 43 | RW   | 0     |
| 8:6    | FSEL42     | FSEL42 - Function Select 42 | RW   | 0     |
| 5:3    | FSEL41     | FSEL41 - Function Select 41 | RW   | 0     |
| 2:0    | FSEL40     | FSEL40 - Function Select 40 | RW   | 0     |

Table 66. GPIO Alternate function select register 4

## GPFSSEL5 Register

| Bit(s) | Field Name                                | Description  | Type | Reset |
|--------|---|--|------|-------|
| 31:24  | Reserved - Write as 0, read as don't care |  |      |       |
| 23:21  | FSEL57                                    | FSEL57 - Function Select 57<br>000 = GPIO Pin 57 is an input<br>001 = GPIO Pin 57 is an output<br>100 = GPIO Pin 57 takes alternate function 0<br>101 = GPIO Pin 57 takes alternate function 1<br>110 = GPIO Pin 57 takes alternate function 2<br>111 = GPIO Pin 57 takes alternate function 3<br>011 = GPIO Pin 57 takes alternate function 4<br>010 = GPIO Pin 57 takes alternate function 5 | RW   | 0     |
| 20:18  | FSEL56                                    | FSEL56 - Function Select 56  | RW   | 0     |
| 17:15  | FSEL55                                    | FSEL55 - Function Select 55  | RW   | 0     |
| 14:12  | FSEL54                                    | FSEL54 - Function Select 54  | RW   | 0     |
| 11:9   | FSEL53                                    | FSEL53 - Function Select 53  | RW   | 0     |
| 8:6    | FSEL52                                    | FSEL52 - Function Select 52  | RW   | 0     |
| 5:3    | FSEL51                                    | FSEL51 - Function Select 51  | RW   | 0     |
| 2:0    | FSEL50                                    | FSEL50 - Function Select 50  | RW   | 0     |

Table 67. GPIO Alternate function select register 5

## GPSET0 Register

### Synopsis

The output set registers are used to set a GPIO pin. The SET $n$  field defines the respective GPIO pin to set, writing a “0” to the field has no effect. If the GPIO pin is being used as an input (by default) then the value in the SET $n$  field is ignored. However, if the pin is subsequently defined as an output then the bit will be set according to the last set/clear operation. Separating the set and clear functions removes the need for read-modify-write operations



| Bit(s) | Field Name     | Description                         | Type | Reset |
|--------|----------------|-------------------------------------|------|-------|
| 31:0   | SETn (n=0..31) | 0 = No effect<br>1 = Set GPIO pin n | WO   | 0     |

Table 68. GPIO Output Set Register 0

## GPSET1 Register

| Bit(s) | Field Name      | Description                               | Type | Reset |
|--------|-----------------|---|------|-------|
| 31:26  |                 | Reserved - Write as 0, read as don't care |      |       |
| 25:0   | SETn (n=32..57) | 0 = No effect<br>1 = Set GPIO pin n.      | WO   | 0     |

Table 69. GPIO Output Set Register 1

## GPCLR0 Register

### Synopsis

The output clear registers are used to clear a GPIO pin. The CLRn field defines the respective GPIO pin to clear, writing a “0” to the field has no effect. If the GPIO pin is being used as an input (by default) then the value in the CLRn field is ignored. However, if the pin is subsequently defined as an output then the bit will be set according to the last set/clear operation. Separating the set and clear functions removes the need for read-modify-write operations.

| Bit(s) | Field Name     | Description                           | Type | Reset |
|--------|----------------|---------------------------------------|------|-------|
| 31:0   | CLRn (n=0..31) | 0 = No effect<br>1 = Clear GPIO pin n | WO   | 0     |

Table 70. GPIO Output Clear Register 0

## GPCLR1 Register

| Bit(s) | Field Name      | Description                               | Type | Reset |
|--------|-----------------|---|------|-------|
| 31:26  |                 | Reserved - Write as 0, read as don't care |      |       |
| 25:0   | CLRn (n=32..57) | 0 = No effect<br>1 = Clear GPIO pin n     | WO   | 0     |

Table 71. GPIO Output Clear Register 1

## GPLEV0 Register

### Synopsis

The pin level registers return the actual value of the pin. The LEVn field gives the value of the respective GPIO pin.



| Bit(s) | Field Name            | Description   | Type | Reset |
|--------|-----------------------|---|------|-------|
| 31:0   | LEV $n$ ( $n=0..31$ ) | 0 = GPIO pin $n$ is low<br>1 = GPIO pin $n$ is high | RO   | 0     |

Table 72. GPIO Level Register 0

## GPLEV1 Register

| Bit(s) | Field Name                                | Description   | Type | Reset |
|--------|---|---|------|-------|
| 31:26  | Reserved - Write as 0, read as don't care |   |      |       |
| 25:0   | LEV $n$ ( $n=32..57$ )                    | 0 = GPIO pin $n$ is low<br>1 = GPIO pin $n$ is high | RO   | 0     |

Table 73. GPIO Level Register 1

## GPEDS0 Register

### Synopsis

The event detect status registers are used to record level and edge events on the GPIO pins. The relevant bit in the event detect status registers is set whenever: 1) an edge is detected that matches the type of edge programmed in the rising/falling edge detect enable registers, or 2) a level is detected that matches the type of level programmed in the high/low level detect enable registers. The bit is cleared by writing a “1” to the relevant bit.

The interrupt controller can be programmed to interrupt the processor when any of the status bits are set. The GPIO peripheral has four dedicated interrupt lines.

Each GPIO bank can generate an independent interrupt. The fourth line generates a single interrupt whenever any bit is set.

| Bit(s) | Field Name            | Description  | Type | Reset |
|--------|-----------------------|--|------|-------|
| 31:0   | EDS $n$ ( $n=0..31$ ) | 0 = Event not detected on GPIO pin $n$<br>1 = Event detected on GPIO pin $n$ | W1C  | 0     |

Table 74. GPIO Event Detect Status Register 0

## GPEDS1 Register

| Bit(s) | Field Name                                | Description  | Type | Reset |
|--------|---|--|------|-------|
| 31:26  | Reserved - Write as 0, read as don't care |  |      |       |
| 25:0   | EDS $n$ ( $n=32..57$ )                    | 0 = Event not detected on GPIO pin $n$<br>1 = Event detected on GPIO pin $n$ | W1C  | 0     |

Table 75. GPIO Event Detect Status Register 1

## GPREN0 Register



## Synopsis

The rising edge detect enable registers define the pins for which a rising edge transition sets a bit in the event detect status registers (GPEDSn). When the relevant bits are set in both the GPRENn and GPFENn registers, any transition (1 to 0 and 0 to 1) will set a bit in the GPEDSn registers. The GPRENn registers use synchronous edge detection. This means the input signal is sampled using the system clock and then it is looking for a “011” pattern on the sampled signal. This has the effect of suppressing glitches.

| Bit(s) | Field Name     | Description   | Type | Reset |
|--------|----------------|---|------|-------|
| 31:0   | RENn (n=0..31) | 0 = Rising edge detect disabled on GPIO pin <i>n</i><br>1 = Rising edge on GPIO pin <i>n</i> sets corresponding bit in GPEDS0 | RW   | 0     |

Table 76. GPIO Rising Edge Detect Status Register 0

## GPREN1 Register

| Bit(s) | Field Name                                | Description   | Type | Reset |
|--------|---|---|------|-------|
| 31:26  | Reserved - Write as 0, read as don't care |   |      |       |
| 25:0   | RENn (n=32..57)                           | 0 = Rising edge detect disabled on GPIO pin <i>n</i><br>1 = Rising edge on GPIO pin <i>n</i> sets corresponding bit in GPEDS1 | RW   | 0     |

Table 77. GPIO Rising Edge Detect Status Register 1

## GPFEN0 Register

### Synopsis

The falling edge detect enable registers define the pins for which a falling edge transition sets a bit in the event detect status registers (GPEDSn). When the relevant bits are set in both the GPRENn and GPFENn registers, any transition (1 to 0 and 0 to 1) will set a bit in the GPEDSn registers. The GPFENn registers use synchronous edge detection. This means the input signal is sampled using the system clock and then it is looking for a “100” pattern on the sampled signal. This has the effect of suppressing glitches.

| Bit(s) | Field Name     | Description   | Type | Reset |
|--------|----------------|---|------|-------|
| 31:0   | FENn (n=0..31) | 0 = Falling edge detect disabled on GPIO pin <i>n</i><br>1 = Falling edge on GPIO pin <i>n</i> sets corresponding bit in GPEDS0 | RW   | 0     |

Table 78. GPIO Falling Edge Detect Status Register 0

## GPFEN1 Register

| Bit(s) | Field Name                                | Description | Type | Reset |
|--------|---|-------------|------|-------|
| 31:26  | Reserved - Write as 0, read as don't care |             |      |       |



| Bit(s) | Field Name             | Description   | Type | Reset |
|--------|------------------------|---|------|-------|
| 25:0   | FEN $n$ ( $n=32..57$ ) | 0 = Falling edge detect disabled on GPIO pin $n$<br>1 = Falling edge on GPIO pin $n$ sets corresponding bit in GPEDS1 | RW   | 0     |

Table 79. GPIO Falling Edge Detect Status Register 1

## GPHEN0 Register

### Synopsis

The high level detect enable registers define the pins for which a high level sets a bit in the event detect status register (GPEDS $n$ ). If the pin is still high when an attempt is made to clear the status bit in GPEDS $n$  then the status bit will remain set.

| Bit(s) | Field Name            | Description   | Type | Reset |
|--------|-----------------------|---|------|-------|
| 31:0   | HEN $n$ ( $n=0..31$ ) | 0 = High detect disabled on GPIO pin $n$<br>1 = High on GPIO pin $n$ sets corresponding bit in GPEDS0 | RW   | 0     |

Table 80. GPIO High Detect Status Register 0

## GPHEN1 Register

| Bit(s) | Field Name                                | Description   | Type | Reset |
|--------|---|---|------|-------|
| 31:26  | Reserved - Write as 0, read as don't care |   |      |       |
| 25:0   | HEN $n$ ( $n=32..57$ )                    | 0 = High detect disabled on GPIO pin $n$<br>1 = High on GPIO pin $n$ sets corresponding bit in GPEDS1 | RW   | 0     |

Table 81. GPIO High Detect Status Register 1

## GPLEN0 Register

### Synopsis

The low level detect enable registers define the pins for which a low level sets a bit in the event detect status register (GPEDS $n$ ). If the pin is still low when an attempt is made to clear the status bit in GPEDS $n$  then the status bit will remain set.

| Bit(s) | Field Name            | Description   | Type | Reset |
|--------|-----------------------|---|------|-------|
| 31:0   | LEN $n$ ( $n=0..31$ ) | 0 = Low detect disabled on GPIO pin $n$<br>1 = Low on GPIO pin $n$ sets corresponding bit in GPEDS0 | RW   | 0     |

Table 82. GPIO Low Detect Status Register 0

## GPLEN1 Register





| Bit(s) | Field Name             | Description   | Type | Reset |
|--------|------------------------|---|------|-------|
| 31:26  |                        | Reserved - Write as 0, read as don't care   |      |       |
| 25:0   | LEN $n$ ( $n=32..57$ ) | 0 = Low detect disabled on GPIO pin $n$<br>1 = Low on GPIO pin $n$ sets corresponding bit in GPEDS1 | RW   | 0     |

Table 83. GPIO Low Detect Status Register 1

## GPAREN0 Register

### Synopsis

The asynchronous rising edge detect enable registers define the pins for which an asynchronous rising edge transition sets a bit in the event detect status registers (GPEDS $n$ ).

Asynchronous means the incoming signal is not sampled by the system clock. As such rising edges of very short duration can be detected.

| Bit(s) | Field Name             | Description   | Type | Reset |
|--------|------------------------|---|------|-------|
| 31:0   | AREN $n$ ( $n=0..31$ ) | 0 = Asynchronous rising edge detect disabled on GPIO pin $n$<br>1 = Asynchronous rising edge on GPIO pin $n$ sets corresponding bit in GPEDS0 | RW   | 0     |

Table 84. GPIO Asynchronous rising Edge Detect Status Register 0

## GPAREN1 Register

| Bit(s) | Field Name              | Description   | Type | Reset |
|--------|-------------------------|---|------|-------|
| 31:26  |                         | Reserved - Write as 0, read as don't care   |      |       |
| 25:0   | AREN $n$ ( $n=32..57$ ) | 0 = Asynchronous rising edge detect disabled on GPIO pin $n$<br>1 = Asynchronous rising edge on GPIO pin $n$ sets corresponding bit in GPEDS1 | RW   | 0     |

Table 85. GPIO Asynchronous rising Edge Detect Status Register 1

## GPAFEN0 Register

### Synopsis

The asynchronous falling edge detect enable registers define the pins for which an asynchronous falling edge transition sets a bit in the event detect status registers (GPEDS $n$ ). Asynchronous means the incoming signal is not sampled by the system clock. As such falling edges of very short duration can be detected.



| Bit(s) | Field Name             | Description   | Type | Reset |
|--------|------------------------|---|------|-------|
| 31:0   | AFEN $n$ ( $n=0..31$ ) | 0 = Asynchronous falling edge detect disabled on GPIO pin $n$<br>1 = Asynchronous falling edge on GPIO pin $n$ sets corresponding bit in GPEDS0 | RW   | 0     |

Table 86. GPIO Asynchronous Falling Edge Detect Status Register 0

## GPAFEN1 Register

| Bit(s) | Field Name              | Description   | Type | Reset |
|--------|-------------------------|---|------|-------|
| 31:26  |                         | Reserved - Write as 0, read as don't care   |      |       |
| 25:0   | AFEN $n$ ( $n=32..57$ ) | 0 = Asynchronous falling edge detect disabled on GPIO pin $n$<br>1 = Asynchronous falling edge on GPIO pin $n$ sets corresponding bit in GPEDS1 | RW   | 0     |

Table 87. GPIO Asynchronous Falling Edge Detect Status Register 1

## GPIO\_PUP\_PDN\_CNTRL\_REG0 Register

### Synopsis

The GPIO Pull-up / Pull-down Registers control the actuation of the internal pull-up/down resistors. Reading these registers gives the current pull-state.

The Alternate function table also has the pull state which is applied after a power down.

| Bit(s) | Field Name           | Description   | Type | Reset |
|--------|----------------------|---|------|-------|
| 31:30  | GPIO_PUP_PDN_CNTRL15 | Resistor Select for GPIO15<br>00 = No resistor is selected<br>01 = Pull up resistor is selected<br>10 = Pull down resistor is selected<br>11 = Reserved | RW   | 0x2   |
| 29:28  | GPIO_PUP_PDN_CNTRL14 | Resistor Select for GPIO14  | RW   | 0x2   |
| 27:26  | GPIO_PUP_PDN_CNTRL13 | Resistor Select for GPIO13  | RW   | 0x2   |
| 25:24  | GPIO_PUP_PDN_CNTRL12 | Resistor Select for GPIO12  | RW   | 0x2   |
| 23:22  | GPIO_PUP_PDN_CNTRL11 | Resistor Select for GPIO11  | RW   | 0x2   |
| 21:20  | GPIO_PUP_PDN_CNTRL10 | Resistor Select for GPIO10  | RW   | 0x2   |



| Bit(s) | Field Name           | Description                | Type | Reset |
|--------|----------------------|----------------------------|------|-------|
| 19:18  | GPIO_PUP_PDN_CNTRL09 | Resistor Select for GPIO09 | RW   | 0x2   |
| 17:16  | GPIO_PUP_PDN_CNTRL08 | Resistor Select for GPIO08 | RW   | 0x2   |
| 15:14  | GPIO_PUP_PDN_CNTRL07 | Resistor Select for GPIO07 | RW   | 0x2   |
| 13:12  | GPIO_PUP_PDN_CNTRL06 | Resistor Select for GPIO06 | RW   | 0x2   |
| 11:10  | GPIO_PUP_PDN_CNTRL05 | Resistor Select for GPIO05 | RW   | 0x2   |
| 09:08  | GPIO_PUP_PDN_CNTRL04 | Resistor Select for GPIO04 | RW   | 0x2   |
| 07:06  | GPIO_PUP_PDN_CNTRL03 | Resistor Select for GPIO03 | RW   | 0x2   |
| 05:04  | GPIO_PUP_PDN_CNTRL02 | Resistor Select for GPIO02 | RW   | 0x2   |
| 03:02  | GPIO_PUP_PDN_CNTRL01 | Resistor Select for GPIO01 | RW   | 0x2   |
| 01:00  | GPIO_PUP_PDN_CNTRL00 | Resistor Select for GPIO00 | RW   | 0x2   |

Table 88. GPIO Pull-up / Pull-down Register 0

## GPIO\_PUP\_PDN\_CNTRL\_REG1 Register

| Bit(s) | Field Name           | Description   | Type | Reset |
|--------|----------------------|---|------|-------|
| 31:30  | GPIO_PUP_PDN_CNTRL31 | Resistor Select for GPIO31<br>00 = No resistor is selected<br>01 = Pull up resistor is selected<br>10 = Pull down resistor is selected<br>11 = Reserved | RW   | 0x2   |
| 29:28  | GPIO_PUP_PDN_CNTRL30 | Resistor Select for GPIO30  | RW   | 0x2   |
| 27:26  | GPIO_PUP_PDN_CNTRL29 | Resistor Select for GPIO29  | RW   | 0x2   |
| 25:24  | GPIO_PUP_PDN_CNTRL28 | Resistor Select for GPIO28  | RW   | 0x2   |
| 23:22  | GPIO_PUP_PDN_CNTRL27 | Resistor Select for GPIO27  | RW   | 0x2   |



| Bit(s) | Field Name               | Description                | Type | Reset |
|--------|--------------------------|----------------------------|------|-------|
| 21:20  | GPIO_PUP_PDN_C<br>NTRL26 | Resistor Select for GPIO26 | RW   | 0x2   |
| 19:18  | GPIO_PUP_PDN_C<br>NTRL25 | Resistor Select for GPIO25 | RW   | 0x2   |
| 17:16  | GPIO_PUP_PDN_C<br>NTRL24 | Resistor Select for GPIO24 | RW   | 0x2   |
| 15:14  | GPIO_PUP_PDN_C<br>NTRL23 | Resistor Select for GPIO23 | RW   | 0x2   |
| 13:12  | GPIO_PUP_PDN_C<br>NTRL22 | Resistor Select for GPIO22 | RW   | 0x2   |
| 11:10  | GPIO_PUP_PDN_C<br>NTRL21 | Resistor Select for GPIO21 | RW   | 0x2   |
| 09:08  | GPIO_PUP_PDN_C<br>NTRL20 | Resistor Select for GPIO20 | RW   | 0x2   |
| 07:06  | GPIO_PUP_PDN_C<br>NTRL19 | Resistor Select for GPIO19 | RW   | 0x2   |
| 05:04  | GPIO_PUP_PDN_C<br>NTRL18 | Resistor Select for GPIO18 | RW   | 0x2   |
| 03:02  | GPIO_PUP_PDN_C<br>NTRL17 | Resistor Select for GPIO17 | RW   | 0x2   |
| 01:00  | GPIO_PUP_PDN_C<br>NTRL16 | Resistor Select for GPIO16 | RW   | 0x2   |

Table 89. GPIO Pull-up / Pull-down Register 1

## GPIO\_PUP\_PDN\_CNTRL\_REG2 Register

| Bit(s) | Field Name               | Description   | Type | Reset |
|--------|--------------------------|---|------|-------|
| 31:30  | GPIO_PUP_PDN_C<br>NTRL47 | Resistor Select for GPIO47<br>00 = No resistor is selected<br>01 = Pull up resistor is selected<br>10 = Pull down resistor is selected<br>11 = Reserved | RW   | 0x2   |
| 29:28  | GPIO_PUP_PDN_C<br>NTRL46 | Resistor Select for GPIO46  | RW   | 0x2   |
| 27:26  | GPIO_PUP_PDN_C<br>NTRL45 | Resistor Select for GPIO45  | RW   | 0x2   |
| 25:24  | GPIO_PUP_PDN_C<br>NTRL44 | Resistor Select for GPIO44  | RW   | 0x2   |



| Bit(s) | Field Name           | Description                | Type | Reset |
|--------|----------------------|----------------------------|------|-------|
| 23:22  | GPIO_PUP_PDN_CNTRL43 | Resistor Select for GPIO43 | RW   | 0x2   |
| 21:20  | GPIO_PUP_PDN_CNTRL42 | Resistor Select for GPIO42 | RW   | 0x2   |
| 19:18  | GPIO_PUP_PDN_CNTRL41 | Resistor Select for GPIO41 | RW   | 0x2   |
| 17:16  | GPIO_PUP_PDN_CNTRL40 | Resistor Select for GPIO40 | RW   | 0x2   |
| 15:14  | GPIO_PUP_PDN_CNTRL39 | Resistor Select for GPIO39 | RW   | 0x2   |
| 13:12  | GPIO_PUP_PDN_CNTRL38 | Resistor Select for GPIO38 | RW   | 0x2   |
| 11:10  | GPIO_PUP_PDN_CNTRL37 | Resistor Select for GPIO37 | RW   | 0x2   |
| 09:08  | GPIO_PUP_PDN_CNTRL36 | Resistor Select for GPIO36 | RW   | 0x2   |
| 07:06  | GPIO_PUP_PDN_CNTRL35 | Resistor Select for GPIO35 | RW   | 0x2   |
| 05:04  | GPIO_PUP_PDN_CNTRL34 | Resistor Select for GPIO34 | RW   | 0x2   |
| 03:02  | GPIO_PUP_PDN_CNTRL33 | Resistor Select for GPIO33 | RW   | 0x2   |
| 01:00  | GPIO_PUP_PDN_CNTRL32 | Resistor Select for GPIO32 | RW   | 0x2   |

Table 90. GPIO Pull-up / Pull-down Register 2

## GPIO\_PUP\_PDN\_CNTRL\_REG3 Register

| Bit(s) | Field Name                                | Description   | Type | Reset |
|--------|---|---|------|-------|
| 31:20  | Reserved - Write as 0, read as don't care |   |      |       |
| 19:18  | GPIO_PUP_PDN_CNTRL57                      | Resistor Select for GPIO57<br>00 = No resistor is selected<br>01 = Pull up resistor is selected<br>10 = Pull down resistor is selected<br>11 = Reserved | RW   | 0x2   |
| 17:16  | GPIO_PUP_PDN_CNTRL56                      | Resistor Select for GPIO56  | RW   | 0x2   |



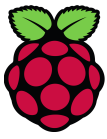
| Bit(s) | Field Name               | Description                | Type | Reset |
|--------|--------------------------|----------------------------|------|-------|
| 15:14  | GPIO_PUP_PDN_C<br>NTRL55 | Resistor Select for GPIO55 | RW   | 0x2   |
| 13:12  | GPIO_PUP_PDN_C<br>NTRL54 | Resistor Select for GPIO54 | RW   | 0x2   |
| 11:10  | GPIO_PUP_PDN_C<br>NTRL53 | Resistor Select for GPIO53 | RW   | 0x2   |
| 09:08  | GPIO_PUP_PDN_C<br>NTRL52 | Resistor Select for GPIO52 | RW   | 0x2   |
| 07:06  | GPIO_PUP_PDN_C<br>NTRL51 | Resistor Select for GPIO51 | RW   | 0x2   |
| 05:04  | GPIO_PUP_PDN_C<br>NTRL50 | Resistor Select for GPIO50 | RW   | 0x2   |
| 03:02  | GPIO_PUP_PDN_C<br>NTRL49 | Resistor Select for GPIO49 | RW   | 0x2   |
| 01:00  | GPIO_PUP_PDN_C<br>NTRL48 | Resistor Select for GPIO48 | RW   | 0x2   |

Table 91. GPIO Pull-up / Pull-down Register 3

## 5.3. Alternative Function Assignments

Every GPIO pin can carry an alternate function. Up to 6 alternate functions are available but not every pin has that many alternate functions. The table below gives a quick overview.

| GPIO  | Pull | ALT0       | ALT1             | ALT2      | ALT3         | ALT4 | ALT5 |
|-------|------|------------|------------------|-----------|--------------|------|------|
| GPIO0 | High | SDA0       | SA5              | PCLK      | SPI3_CE0_N   | TXD2 | SDA6 |
| GPIO1 | High | SCL0       | SA4              | DE        | SPI3_MISO    | RXD2 | SCL6 |
| GPIO2 | High | SDA1       | SA3              | LCD_VSYNC | SPI3_MOSI    | CTS2 | SDA3 |
| GPIO3 | High | SCL1       | SA2              | LCD_HSYNC | SPI3_SCLK    | RTS2 | SCL3 |
| GPIO4 | High | GPCLK0     | SA1              | DPI_D0    | SPI4_CE0_N   | TXD3 | SDA3 |
| GPIO5 | High | GPCLK1     | SA0              | DPI_D1    | SPI4_MISO    | RXD3 | SCL3 |
| GPIO6 | High | GPCLK2     | SOE_N / SE       | DPI_D2    | SPI4_MOSI    | CTS3 | SDA4 |
| GPIO7 | High | SPI0_CE1_N | SWE_N /<br>SRW_N | DPI_D3    | SPI4_SCLK    | RTS3 | SCL4 |
| GPIO8 | High | SPI0_CE0_N | SD0              | DPI_D4    | BSCSL / CE_N | TXD4 | SDA4 |
| GPIO9 | Low  | SPI0_MISO  | SD1              | DPI_D5    | BSCSL / MISO | RXD4 | SCL4 |



| GPIO   | Pull | ALT0       | ALT1             | ALT2       | ALT3             | ALT4                | ALT5       |
|--------|------|------------|------------------|------------|------------------|---------------------|------------|
| GPIO10 | Low  | SPI0_MOSI  | SD2              | DPI_D6     | BSCSL_SDA / MOSI | CTS4                | SDA5       |
| GPIO11 | Low  | SPI0_SCLK  | SD3              | DPI_D7     | BSCSL_SCL / SCLK | RTS4                | SCL5       |
| GPIO12 | Low  | PWM0_0     | SD4              | DPI_D8     | SPI5_CE0_N       | TXD5                | SDA5       |
| GPIO13 | Low  | PWM0_1     | SD5              | DPI_D9     | SPI5_MISO        | RXD5                | SCL5       |
| GPIO14 | Low  | TXD0       | SD6              | DPI_D10    | SPI5_MOSI        | CTS5                | TXD1       |
| GPIO15 | Low  | RXD0       | SD7              | DPI_D11    | SPI5_SCLK        | RTS5                | RXD1       |
| GPIO16 | Low  | <reserved> | SD8              | DPI_D12    | CTS0             | SPI1_CE2_N          | CTS1       |
| GPIO17 | Low  | <reserved> | SD9              | DPI_D13    | RTS0             | SPI1_CE1_N          | RTS1       |
| GPIO18 | Low  | PCM_CLK    | SD10             | DPI_D14    | SPI6_CE0_N       | SPI1_CE0_N          | PWM0_0     |
| GPIO19 | Low  | PCM_FS     | SD11             | DPI_D15    | SPI6_MISO        | SPI1_MISO           | PWM0_1     |
| GPIO20 | Low  | PCM_DIN    | SD12             | DPI_D16    | SPI6_MOSI        | SPI1_MOSI           | GPCLK0     |
| GPIO21 | Low  | PCM_DOUT   | SD13             | DPI_D17    | SPI6_SCLK        | SPI1_SCLK           | GPCLK1     |
| GPIO22 | Low  | SD0_CLK    | SD14             | DPI_D18    | SD1_CLK          | ARM_TRST            | SDA6       |
| GPIO23 | Low  | SD0_CMD    | SD15             | DPI_D19    | SD1_CMD          | ARM_RTCK            | SCL6       |
| GPIO24 | Low  | SD0_DAT0   | SD16             | DPI_D20    | SD1_DAT0         | ARM_TDO             | SPI3_CE1_N |
| GPIO25 | Low  | SD0_DAT1   | SD17             | DPI_D21    | SD1_DAT1         | ARM_TCK             | SPI4_CE1_N |
| GPIO26 | Low  | SD0_DAT2   | <reserved>       | DPI_D22    | SD1_DAT2         | ARM_TDI             | SPI5_CE1_N |
| GPIO27 | Low  | SD0_DAT3   | <reserved>       | DPI_D23    | SD1_DAT3         | ARM_TMS             | SPI6_CE1_N |
| GPIO28 | -    | SDA0       | SA5              | PCM_CLK    | <reserved>       | MII_A_RX_ER<br>R    | RGMIIMDIO  |
| GPIO29 | -    | SCL0       | SA4              | PCM_FS     | <reserved>       | MII_A_TX_ER<br>R    | RGMIIMDC   |
| GPIO30 | Low  | <reserved> | SA3              | PCM_DIN    | CTS0             | MII_A_CRS           | CTS1       |
| GPIO31 | Low  | <reserved> | SA2              | PCM_DOUT   | RTS0             | MII_A_COL           | RTS1       |
| GPIO32 | Low  | GPCLK0     | SA1              | <reserved> | TXD0             | SD_CARD_PR<br>ES    | TXD1       |
| GPIO33 | Low  | <reserved> | SA0              | <reserved> | RXD0             | SD_CARD_W<br>RPROT  | RXD1       |
| GPIO34 | High | GPCLK0     | SOE_N / SE       | <reserved> | SD1_CLK          | SD_CARD_LE<br>D     | RGMIIRQ    |
| GPIO35 | High | SPI0_CE1_N | SWE_N /<br>SRW_N |            | SD1_CMD          | RGMIISTAR<br>T_STOP |            |



| GPIO   | Pull | ALT0       | ALT1 | ALT2       | ALT3       | ALT4            | ALT5             |
|--------|------|------------|------|------------|------------|-----------------|------------------|
| GPIO36 | High | SPI0_CE0_N | SD0  | TXD0       | SD1_DAT0   | RGMII_RX_O<br>K | MII_A_RX_ER<br>R |
| GPIO37 | Low  | SPI0_MISO  | SD1  | RXD0       | SD1_DAT1   | RGMII_MDIO      | MII_A_TX_ER<br>R |
| GPIO38 | Low  | SPI0_MOSI  | SD2  | RTS0       | SD1_DAT2   | RGMII_MDC       | MII_A_CRS        |
| GPIO39 | Low  | SPI0_SCLK  | SD3  | CTS0       | SD1_DAT3   | RGMII_IRQ       | MII_A_COL        |
| GPIO40 | Low  | PWM1_0     | SD4  |            | SD1_DAT4   | SPI0_MISO       | TXD1             |
| GPIO41 | Low  | PWM1_1     | SD5  | <reserved> | SD1_DAT5   | SPI0_MOSI       | RXD1             |
| GPIO42 | Low  | GPCLK1     | SD6  | <reserved> | SD1_DAT6   | SPI0_SCLK       | RTS1             |
| GPIO43 | Low  | GPCLK2     | SD7  | <reserved> | SD1_DAT7   | SPI0_CE0_N      | CTS1             |
| GPIO44 | -    | GPCLK1     | SDA0 | SDA1       | <reserved> | SPI0_CE1_N      | SD_CARD_VO<br>LT |
| GPIO45 | -    | PWM0_1     | SCL0 | SCL1       | <reserved> | SPI0_CE2_N      | SD_CARD_P<br>WR0 |
| GPIO46 | High | <Internal> |      |            |            |                 |                  |
| GPIO47 | High | <Internal> |      |            |            |                 |                  |
| GPIO48 | High | <Internal> |      |            |            |                 |                  |
| GPIO49 | High | <Internal> |      |            |            |                 |                  |
| GPIO50 | High | <Internal> |      |            |            |                 |                  |
| GPIO51 | High | <Internal> |      |            |            |                 |                  |
| GPIO52 | High | <Internal> |      |            |            |                 |                  |
| GPIO53 | High | <Internal> |      |            |            |                 |                  |
| GPIO54 | High | <Internal> |      |            |            |                 |                  |
| GPIO55 | High | <Internal> |      |            |            |                 |                  |
| GPIO56 | High | <Internal> |      |            |            |                 |                  |
| GPIO57 | High | <Internal> |      |            |            |                 |                  |

Table 92. GPIO Pins Alternative Function Assignment

Entries which are white should **not** be used. These may have unexpected results as some of these have special functions used in test mode e.g. they may drive the output with high frequency signals.

Special function legend:

| Name | Function                              | See section         |
|------|---------------------------------------|---------------------|
| SDA0 | BSC <sup>[8]</sup> master 0 data line | <a href="#">BSC</a> |





| Name             | Function                                      | See section                                 |
|------------------|---|---|
| SCL0             | BSC master 0 clock line                       | <a href="#">BSC</a>                         |
| SDAx             | BSC master 1,3,4,5,6 <sup>[9]</sup> data line | <a href="#">BSC</a>                         |
| SCLx             | BSC master 1,3,4,5,6 clock line               | <a href="#">BSC</a>                         |
| GPCLKx           | General purpose Clock 0,1,2                   | <a href="#">General Purpose GPIO Clocks</a> |
| SPIx_CE2_N       | SPI 0,3,4,5,6 Chip select 2                   | <a href="#">SPI</a>                         |
| SPIx_CE1_N       | SPI 0,3,4,5,6 Chip select 1                   | <a href="#">SPI</a>                         |
| SPIx_CE0_N       | SPI 0,3,4,5,6 Chip select 0                   | <a href="#">SPI</a>                         |
| SPIx_MISO        | SPI 0,3,4,5,6 MISO                            | <a href="#">SPI</a>                         |
| SPIx_MOSI        | SPI 0,3,4,5,6 MOSI                            | <a href="#">SPI</a>                         |
| SPIx_SCLK        | SPI 0,3,4,5,6 Serial clock                    | <a href="#">SPI</a>                         |
| PWMx_0           | PWM 0,1 channel 0                             | <a href="#">Pulse Width Modulator</a>       |
| PWMx_1           | PWM 0,1 channel 1                             | <a href="#">Pulse Width Modulator</a>       |
| TXDx             | UART 0,2,3,4,5 Transmit Data                  | <a href="#">UART</a>                        |
| RXDx             | UART 0,2,3,4,5 Receive Data                   | <a href="#">UART</a>                        |
| CTSx             | UART 0,2,3,4,5 Clear To Send                  | <a href="#">UART</a>                        |
| RTSx             | UART 0,2,3,4,5 Request To Send                | <a href="#">UART</a>                        |
| PCM_CLK          | PCM clock                                     | <a href="#">PCM Audio</a>                   |
| PCM_FS           | PCM Frame Sync                                | <a href="#">PCM Audio</a>                   |
| PCM_DIN          | PCM Data in                                   | <a href="#">PCM Audio</a>                   |
| PCM_DOUT         | PCM data out                                  | <a href="#">PCM Audio</a>                   |
| SAX              | Secondary mem Address bus                     | <a href="#">Secondary Memory Interface</a>  |
| SOE_N / SE       | Secondary mem. Controls                       | <a href="#">Secondary Memory Interface</a>  |
| SWE_N / SRW_N    | Secondary mem. Controls                       | <a href="#">Secondary Memory Interface</a>  |
| SDx              | Secondary mem. data bus                       | <a href="#">Secondary Memory Interface</a>  |
| BSCSL SDA / MOSI | BSC slave Data, SPI slave MOSI                | <a href="#">BSC/SPI slave</a>               |
| BSCSL SCL / SCLK | BSC slave Clock, SPI slave clock              | <a href="#">BSC/SPI slave</a>               |
| BSCSL - / MISO   | BSC <not used>, SPI MISO                      | <a href="#">BSC/SPI slave</a>               |
| BSCSL - / CE_N   | BSC <not used>, SPI CSn                       | <a href="#">BSC/SPI slave</a>               |
| SPI1_CE2_N       | SPI 1 <sup>[10]</sup> Chip select 2           | <a href="#">Auxiliary I/O</a>               |
| SPI1_CE1_N       | SPI 1 Chip select 1                           | <a href="#">Auxiliary I/O</a>               |
| SPI1_CE0_N       | SPI 1 Chip select 0                           | <a href="#">Auxiliary I/O</a>               |
| SPI1_MISO        | SPI 1 MISO                                    | <a href="#">Auxiliary I/O</a>               |



| Name      | Function                   | See section                   |
|-----------|----------------------------|-------------------------------|
| SPI1_MOSI | SPI 1 MOSI                 | <a href="#">Auxiliary I/O</a> |
| SPI1_SCLK | SPI 1 Serial clock         | <a href="#">Auxiliary I/O</a> |
| TXD1      | UART 1 Transmit Data       | <a href="#">Auxiliary I/O</a> |
| RXD1      | UART 1 Receive Data        | <a href="#">Auxiliary I/O</a> |
| CTS1      | UART 1 Clear To Send       | <a href="#">Auxiliary I/O</a> |
| RTS1      | UART 1 Request To Send     | <a href="#">Auxiliary I/O</a> |
| ARM_TRST  | ARM JTAG reset             | <TBD>                         |
| ARM_RTCK  | ARM JTAG return clock      | <TBD>                         |
| ARM_TDO   | ARM JTAG Data out          | <TBD>                         |
| ARM_TCK   | ARM JTAG Clock             | <TBD>                         |
| ARM_TDI   | ARM JTAG Data in           | <TBD>                         |
| ARM_TMS   | ARM JTAG Mode select       | <TBD>                         |
| PCLK      | Display Parallel Interface | <TBD>                         |
| DE        | Display Parallel Interface | <TBD>                         |
| LCD_VSYNC | Display Parallel Interface | <TBD>                         |
| LCD_HSYNC | Display Parallel Interface | <TBD>                         |
| DPI_Dx    | Display Parallel Interface | <TBD>                         |

Table 93. GPIO Pins Alternative Function Legend

## 5.4. General Purpose GPIO Clocks

The General Purpose clocks can be output to GPIO pins. They run from the peripherals clock sources and use clock generators with noise-shaping MASH dividers. These allow the GPIO clocks to be used to drive audio devices. The fractional divider operates by periodically dropping source clock pulses, therefore the output frequency will periodically switch between:

$$\frac{\text{source\_frequency}}{DIVI}$$

and

$$\frac{\text{source\_frequency}}{DIVI + 1}$$

Jitter is therefore reduced by increasing the source clock frequency. In applications where jitter is a concern, the fastest available clock source should be used.

The General Purpose clocks have MASH noise-shaping dividers which push this fractional divider jitter out of the audio band.



MASH noise-shaping is incorporated to push the fractional divider jitter out of the audio band if required. The MASH can be programmed for 1, 2 or 3-stage filtering. When using the MASH filter, the frequency is spread around the requested frequency and the user must ensure that the module is not exposed to frequencies higher than 25MHz. Also, the MASH filter imposes a low limit on the range of DIVI.

| MASH           | min DIVI | min output freq       | average output freq             | max output freq       |
|----------------|----------|-----------------------|---------------------------------|-----------------------|
| 0 (int divide) | 1        | source / ( DIVI )     | source / ( DIVI )               | source / ( DIVI )     |
| 1              | 2        | source / ( DIVI + 1 ) | source / ( DIVI + DIVF / 1024 ) | source / ( DIVI )     |
| 2              | 3        | source / ( DIVI + 2 ) | source / ( DIVI + DIVF / 1024 ) | source / ( DIVI - 1 ) |
| 3              | 5        | source / ( DIVI + 4 ) | source / ( DIVI + DIVF / 1024 ) | source / ( DIVI - 3 ) |

*Table 94. Effect of MASH Filter on Frequency*

The following example illustrates the spreading of output clock frequency resulting from the use of the MASH filter. Note that the spread is greater for lower divisors.

| PLL freq (MHz) | target freq (MHz) | MASH | divisor | DIVI | DIVF | min freq (MHz) | ave freq (MHz) | max freq (MHz) | error |
|----------------|-------------------|------|---------|------|------|----------------|----------------|----------------|-------|
| 650            | 18.32             | 0    | 35.480  | 35   | 492  | 18.57          | 18.57          | 18.57          | ok    |
| 650            | 18.32             | 1    | 35.480  | 35   | 492  | 18.06          | 18.32          | 18.57          | ok    |
| 650            | 18.32             | 2    | 35.480  | 35   | 492  | 17.57          | 18.32          | 19.12          | ok    |
| 650            | 18.32             | 3    | 35.480  | 35   | 492  | 16.67          | 18.32          | 20.31          | ok    |
| 400            | 18.32             | 0    | 21.834  | 21   | 854  | 19.05          | 19.05          | 19.05          | ok    |
| 400            | 18.32             | 1    | 21.834  | 21   | 854  | 18.18          | 18.32          | 19.05          | ok    |
| 400            | 18.32             | 2    | 21.834  | 21   | 854  | 17.39          | 18.32          | 20.00          | ok    |
| 400            | 18.32             | 3    | 21.834  | 21   | 854  | 16.00          | 18.32          | 22.22          | ok    |
| 200            | 18.32             | 0    | 10.917  | 10   | 939  | 20.00          | 20.00          | 20.00          | ok    |
| 200            | 18.32             | 1    | 10.917  | 10   | 939  | 18.18          | 18.32          | 20.00          | ok    |
| 200            | 18.32             | 2    | 10.917  | 10   | 939  | 16.67          | 18.32          | 22.22          | ok    |
| 200            | 18.32             | 3    | 10.917  | 10   | 939  | 14.29          | 18.32          | 28.57          | error |

*Table 95. Example of Frequency Spread when using MASH Filtering*

It is beyond the scope of this specification to describe the operation of a MASH filter or to determine under what conditions the available levels of filtering are beneficial.



### 5.4.1. Operating Frequency

The maximum operating frequency of the General Purpose clocks is ~125MHz at 1.2V but this will be reduced if the GPIO pins are heavily loaded or have a capacitive load.

### 5.4.2. Register Definitions

The General Purpose clocks register base address is **0x7E10 1000**.

| Address Offset | Register Name             | Description                                  | Size |
|----------------|---------------------------|--|------|
| 0x70           | <a href="#">CM_GP0CTL</a> | Clock Manager General Purpose Clocks Control | 32   |
| 0x74           | <a href="#">CM_GP0DIV</a> | Clock Manager General Purpose Clock Divisors | 32   |
| 0x78           | <a href="#">CM_GP1CTL</a> | Clock Manager General Purpose Clocks Control | 32   |
| 0x7C           | <a href="#">CM_GP1DIV</a> | Clock Manager General Purpose Clock Divisors | 32   |
| 0x80           | <a href="#">CM_GP2CTL</a> | Clock Manager General Purpose Clocks Control | 32   |
| 0x84           | <a href="#">CM_GP2DIV</a> | Clock Manager General Purpose Clock Divisors | 32   |

Table 96. General Purpose Clocks Registers

### CM\_GP0CTL, CM\_GP1CTL & CM\_GP2CTL Registers

| Bit(s) | Field Name                                | Description   | Type | Reset |
|--------|---|---|------|-------|
| 31:24  | PASSWD                                    | Clock Manager password "5a"   | WO   | 0     |
| 23:11  | Reserved - Write as 0, read as don't care |   |      |       |
| 10:9   | MASH                                      | MASH control<br>0 = integer division<br>1 = 1-stage MASH (equivalent to non-MASH dividers)<br>2 = 2-stage MASH<br>3 = 3-stage MASH<br>To avoid lock-ups and glitches do not change this control while BUSY=1 and do not change this control at the same time as asserting ENAB. | RW   | 0     |
| 8      | FLIP                                      | Invert the clock generator output<br>This is intended for use in test/debug only. Switching this control will generate an edge on the clock generator output. To avoid output glitches do not switch this control while BUSY=1.   | RW   | 0     |
| 7      | BUSY                                      | Clock generator is running<br>Indicates the clock generator is running. To avoid glitches and lock-ups, clock sources and setups must not be changed while this flag is set.  | RO   | 0     |



| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 6      |            | Reserved - Write as 0, read as don't care  |      |       |
| 5      | KILL       | Kill the clock generator<br>0 = no action<br>1 = stop and reset the clock generator<br>This is intended for test/debug only. Using this control may cause a glitch on the clock generator output.  | RW   | 0     |
| 4      | ENAB       | Enable the clock generator<br>This requests the clock to start or stop without glitches. The output clock will not stop immediately because the cycle must be allowed to complete to avoid glitches. The BUSY flag will go low when the final cycle is completed.  | RW   | 0     |
| 3:0    | SRC        | Clock source<br>0 = GND<br>1 = oscillator<br>2 = testdebug0<br>3 = testdebug1<br>4 = PLLA per<br>5 = PLLC per<br>6 = PLLD per<br>7 = HDMI auxiliary<br>8-15 = GND<br>To avoid lock-ups and glitches do not change this control while BUSY=1 and do not change this control at the same time as asserting ENAB. | RW   | 0     |

Table 97. General Purpose Clocks Control

## CM\_GP0DIV, CM\_GP1DIV & CM\_GP2DIV Registers

| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 31:24  | PASSWD     | Clock Manager password "5a"   | WO   | 0     |
| 23:12  | DIVI       | Integer part of divisor<br>This value has a minimum limit determined by the MASH setting. See text for details. To avoid lock-ups and glitches do not change this control while BUSY=1. | RW   | 0     |
| 11:0   | DIVE       | Fractional part of divisor<br>To avoid lock-ups and glitches do not change this control while BUSY=1.   | RW   | 0     |

Table 98. General Purpose Clock Divisors

[8] The Broadcom Serial Control bus is a proprietary bus compliant with the Philips® I2C bus/interface

[9] BSC master 2 & 7 are not user-accessible

[10] SPI 2 is not user-accessible



# Chapter 6. Interrupts

## 6.1. Overview

The BCM2711 has a large number of interrupts from various sources, and a choice of two interrupt controllers. The GIC-400 interrupt controller is selected by default, but the legacy interrupt controller can be selected with a setting in `config.txt` - refer to [raspberrypi.org documentation](https://www.raspberrypi.org/documentation) for further details.

In [Figure 5](#) the orange boxes illustrate the various interrupt source blocks, the blue box covers the interrupt controller routing (explained later), and the green box shows the final interrupt destinations. The number underneath each slash through the thick arrows indicates how many signals that arrow contains (thin arrows without a number only contain one signal). The "ARM Core *n*" blocks in orange are actually the same as the "ARM Core *n*" blocks in green, they're just drawn as separate source and destination blocks for clarity. ARM\_LOCAL and ARMC are different hardware blocks within the chip, each with their own set of registers; ARMC is visible to both the VPU and CPU, but ARM\_LOCAL is only visible to the CPU (and corresponds to the "ARM Local peripherals" in [Chapter 1](#)).

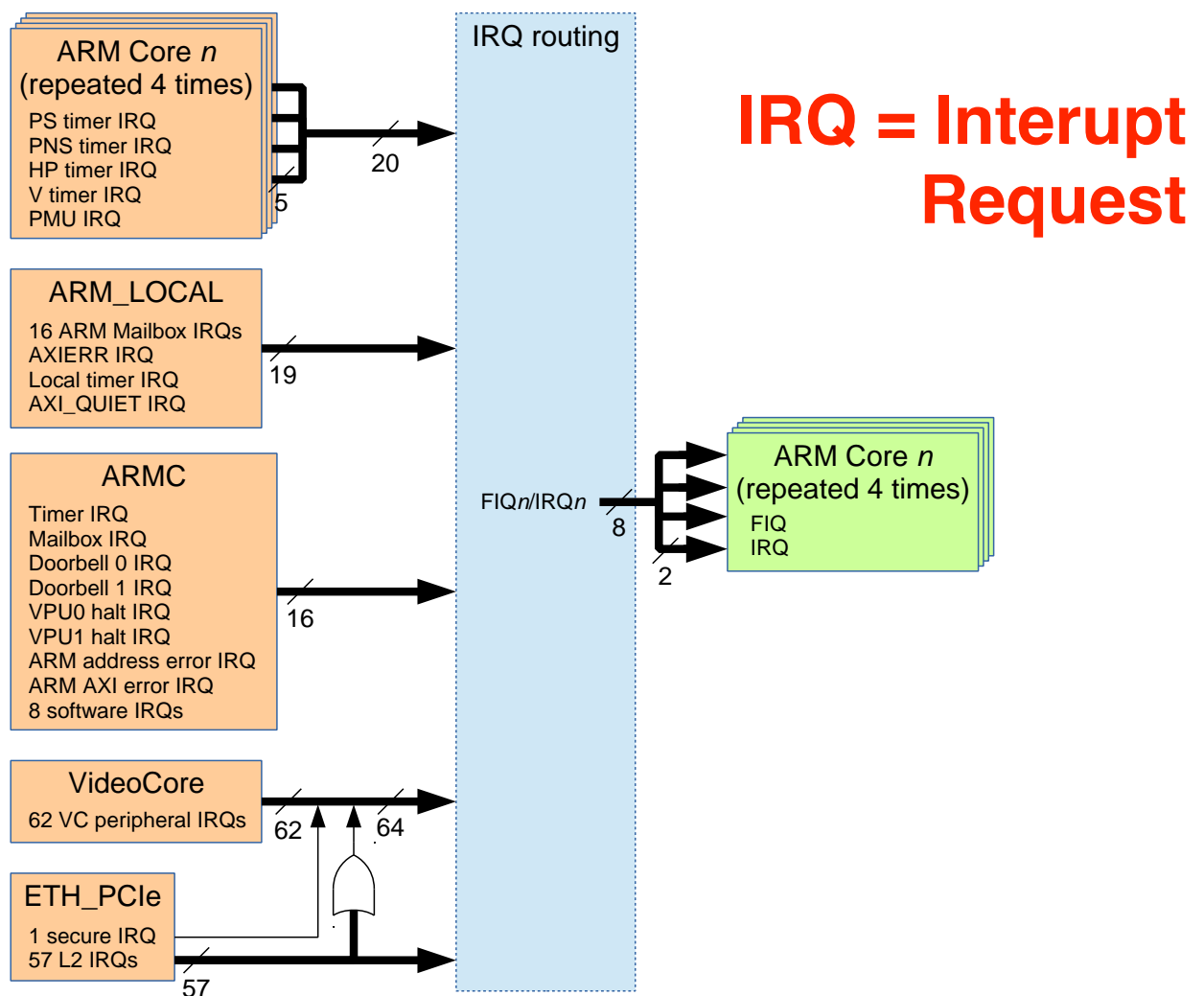


Figure 5. Interrupt sources and destinations

The final output from each interrupt controller is 8 separate signals - a FIQ (Fast Interrupt reQuest) and



an IRQ (Interrupt ReQuest) for each of the 4 ARM cores, i.e. FIQ0 and IRQ0 connected to ARM core 0, FIQ1 and IRQ1 connected to ARM core 1, FIQ2 and IRQ2 connected to ARM core 2, and FIQ3 and IRQ3 connected to ARM core 3. For convenience, this document will refer to those 8 signals as FIQ $n$ /IRQ $n$ .

To avoid confusion, note that the "ARM Mailbox IRQs" in the ARM\_LOCAL block are different from the "Mailbox IRQ" in the ARMC block. Similarly, the "Local timer IRQ" in the ARM\_LOCAL block is different to the "Timer IRQ" in the ARMC block, which are both different from the 4 timer IRQs in the "ARM Core  $n$ " block. The "AXIERR IRQ" in the ARM\_LOCAL block is also different from the "ARM AXI error IRQ" in the ARMC block.

## 6.2. Interrupt sources

### 6.2.1. ARM Core $n$ interrupts

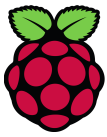
Each of the ARM Cores can raise a Secure Physical (PS) timer interrupt, a Non-Secure Physical (PNS) timer interrupt, a Hypervisor (HP) timer interrupt, a Virtual (V) timer interrupt and a Performance Monitoring Unit (PMU) interrupt. For more information, please refer to the ARM Cortex-A72 documentation on the [ARM Developer website](#).

### 6.2.2. ARM\_LOCAL interrupts

Further information about the ARM Mailboxes can be found in [Chapter 13](#). The AXIERR output is asserted by the ARM's L2 cache if an error response is received. Further information about the Local Timer and AXI\_QUIET can be found in the [Registers section](#) of this chapter.

### 6.2.3. ARMC interrupts

| #  | IRQ                  |
|----|----------------------|
| 0  | Timer                |
| 1  | Mailbox              |
| 2  | Doorbell 0           |
| 3  | Doorbell 1           |
| 4  | VPU0 halted          |
| 5  | VPU1 halted          |
| 6  | ARM address error    |
| 7  | ARM AXI error        |
| 8  | Software Interrupt 0 |
| 9  | Software Interrupt 1 |
| 10 | Software Interrupt 2 |
| 11 | Software Interrupt 3 |



| #  | IRQ                  |
|----|----------------------|
| 12 | Software Interrupt 4 |
| 13 | Software Interrupt 5 |
| 14 | Software Interrupt 6 |
| 15 | Software Interrupt 7 |

Table 99. ARMC peripheral IRQs

The Timer interrupt in [Table 99](#) comes from the "Timer (ARM side)" described in [Chapter 12](#).

The eight general-purpose software interrupts can be set by writing to the SWIRQ\_SET register and cleared by writing to the SWIRQ\_CLEAR register.

#### 6.2.4. VideoCore interrupts

| #  | IRQ 0-15         | #  | IRQ 16-31             | #  | IRQ 32-47                    | #  | IRQ 48-63                    |
|----|------------------|----|-----------------------|----|------------------------------|----|------------------------------|
| 0  | Timer 0          | 16 | DMA 0                 | 32 | HDMI CEC                     | 48 | SMI                          |
| 1  | Timer 1          | 17 | DMA 1                 | 33 | HVS                          | 49 | GPIO 0                       |
| 2  | Timer 2          | 18 | DMA 2                 | 34 | RPIVID                       | 50 | GPIO 1                       |
| 3  | Timer 3          | 19 | DMA 3                 | 35 | SDC                          | 51 | GPIO 2                       |
| 4  | H264 0           | 20 | DMA 4                 | 36 | DSI 0                        | 52 | GPIO 3                       |
| 5  | H264 1           | 21 | DMA 5                 | 37 | Pixel Valve 2                | 53 | <b>OR of all I2C</b>         |
| 6  | H264 2           | 22 | DMA 6                 | 38 | Camera 0                     | 54 | <b>OR of all SPI</b>         |
| 7  | JPEG             | 23 | <b>DMA 7 &amp; 8</b>  | 39 | Camera 1                     | 55 | PCM/I2S                      |
| 8  | ISP              | 24 | <b>DMA 9 &amp; 10</b> | 40 | HDMI 0                       | 56 | SDHOST                       |
| 9  | USB              | 25 | DMA 11                | 41 | HDMI 1                       | 57 | <b>OR of all PL011 UART</b>  |
| 10 | V3D              | 26 | DMA 12                | 42 | Pixel Valve 3                | 58 | <b>OR of all ETH_PCIe L2</b> |
| 11 | Transposer       | 27 | DMA 13                | 43 | SPI/BSC Slave                | 59 | VEC                          |
| 12 | Multicore Sync 0 | 28 | DMA 14                | 44 | DSI 1                        | 60 | CPG                          |
| 13 | MultiCore Sync 1 | 29 | <b>AUX</b>            | 45 | Pixel Valve 0                | 61 | RNG                          |
| 14 | MultiCore Sync 2 | 30 | ARM                   | 46 | <b>Pixel Valve 1 &amp; 4</b> | 62 | <b>EMMC &amp; EMMC2</b>      |
| 15 | MultiCore Sync 3 | 31 | DMA 15                | 47 | CPR                          | 63 | ETH_PCIe secure              |

Table 100. VC peripheral IRQs

The 4 timer interrupts in [Table 100](#) come from the "System Timer" described in [Chapter 10](#).

Because there are more peripherals than available VC peripheral IRQs, some of the VC peripheral





interrupts (highlighted in **bold** in Table 100) are the OR-ed version of multiple peripheral interrupts. The per-peripheral interrupt statuses for VC peripheral IRQs 29, 53, 54 & 57 can in turn be read from the AUX\_IRQ (documented in Chapter 2) and PACTL\_CS (at address `0x7E20 4E00`) registers. Figure 6 shows how this is logically connected, with the vertically-aligned numbers inside the grey boxes indicating bit-positions within the registers.

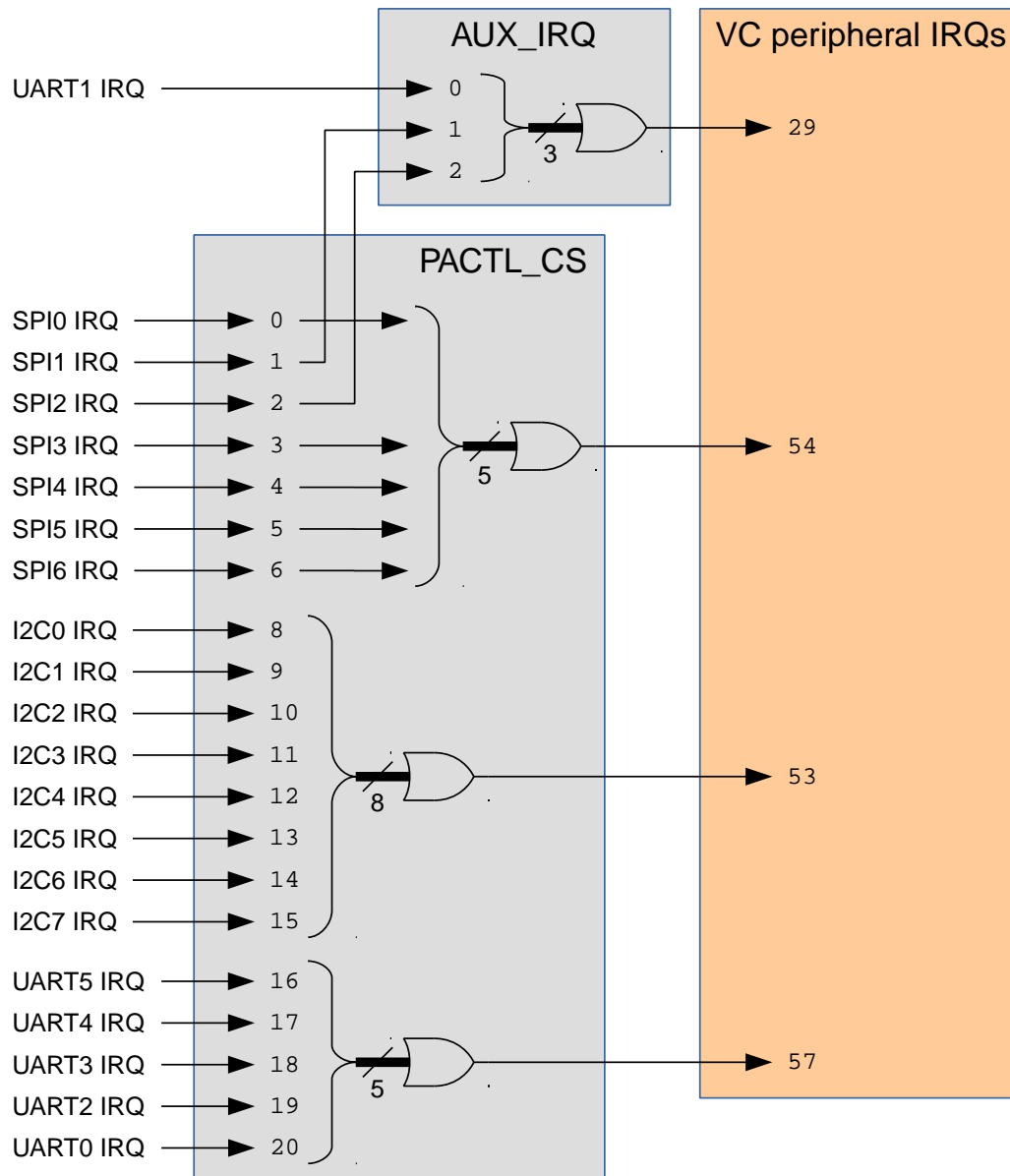


Figure 6. Peripheral IRQ OR-ing

For example if VC peripheral IRQ 53 is triggered, then you know at least one of the I2C peripherals has caused an interrupt. To find out exactly *which* I2C peripherals have interrupts pending, you can read bits 8 to 15 inclusive of PACTL\_CS (alternatively, you could simply read the Status register for each of the I2C peripherals).

There are also some VC peripheral interrupts (23, 24, 46 and 62) that are an OR-ed version of two peripheral interrupt signals - if these interrupts are received the only option is to read the status register for each of the peripherals concerned.



### 6.2.5. ETH\_PCIE interrupts

| #  | IRQ         |
|----|-------------|
| 9  | AVS         |
| 15 | PCIE_0_INTA |
| 16 | PCIE_0_INTB |
| 17 | PCIE_0_INTC |
| 18 | PCIE_0_INTD |
| 20 | PCIE_0_MSI  |
| 29 | GENET_0_A   |
| 30 | GENET_0_B   |
| 48 | USB0_XHCI_0 |

*Table 101. ETH\_PCIE L2 IRQs*

Any IRQ numbers not listed in the table above are reserved.

The secure IRQ output<sup>[11]</sup> from the ETH\_PCIE block is routed to VC peripheral IRQ 63, and all 57 ETH\_PCIE L2 IRQs are OR-ed together and routed to VC peripheral IRQ 58 - see [Figure 5](#) and [Table 100](#).

Note that the 57 individual ETH\_PCIE interrupts aren't routed to the legacy interrupt controller, only VC peripheral IRQ 58 (the OR-ed version) is available.

## 6.3. GIC-400 interrupt controller

The BCM2711 contains an ARM GIC-400 interrupt controller, which is enabled by default. For more information, please refer to the ARM GIC-400 documentation on the [ARM Developer website](#).

[Figure 7](#) shows how the interrupt sources described earlier are connected to the GIC. When the GIC-400 is selected as the interrupt controller, the eight "GIC FIQn/IRQn" outputs are routed to the FIQn/IRQn inputs of the ARM cores.

Note that even when the GIC-400 is selected as the interrupt controller, the outputs of the legacy interrupt controller (described later) are available as PPIs within the GIC.

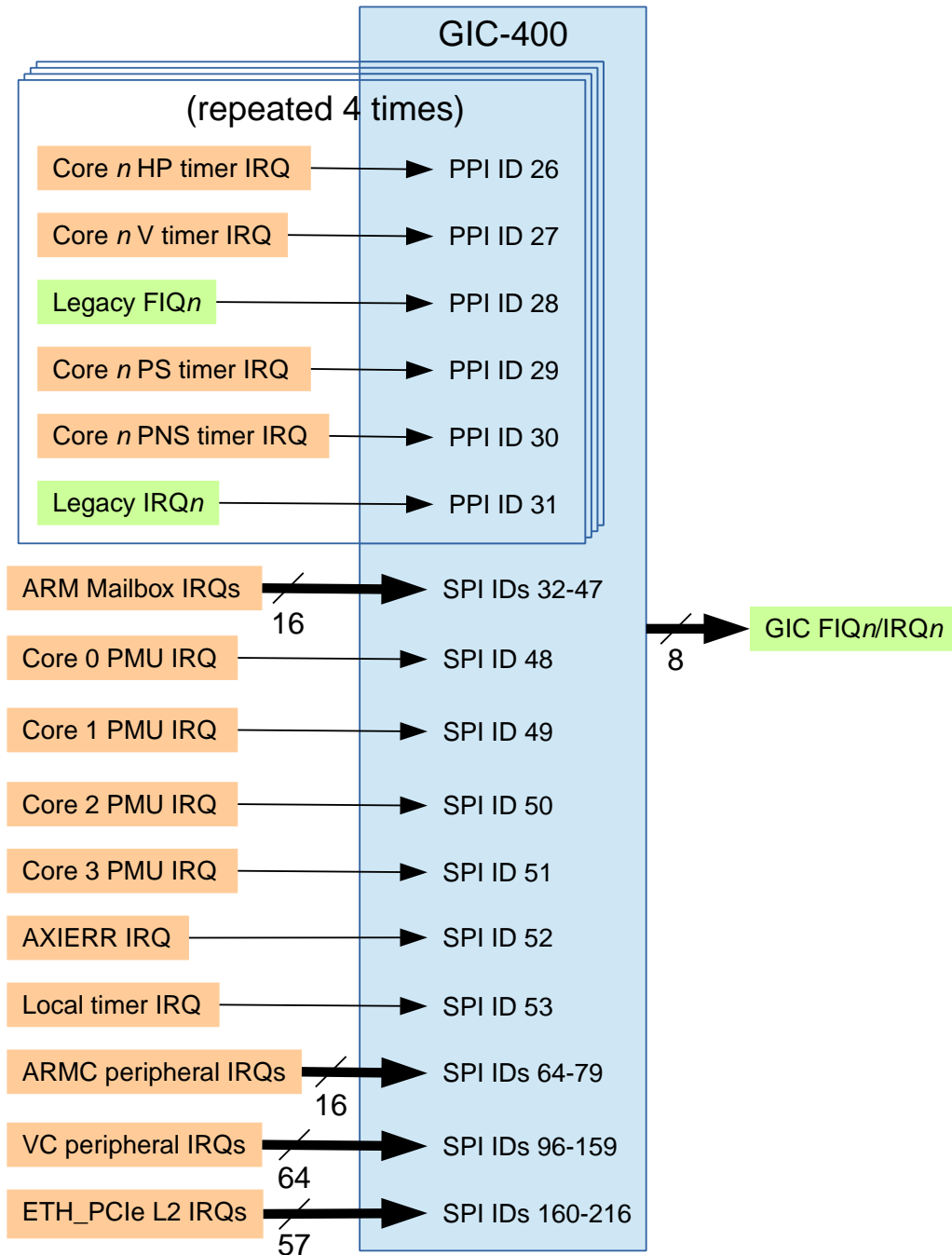


Figure 7. GIC IRQ routing

The GIC-400 also connects to the VFIQ (Virtual FIQ) and VIRQ (Virtual IRQ) input of each ARM core, but for brevity these signals are not shown here.

## 6.4. Legacy interrupt controller

The legacy interrupt controller in the BCM2711 has some similarities with the interrupt controllers used in earlier BCM283x chips, but also several differences. When the legacy interrupt controller is selected, the eight "Legacy FIQ $n$ /IRQ $n$ " outputs (shown in [Figure 9](#)) are routed to the FIQ $n$ /IRQ $n$  inputs of the ARM cores.

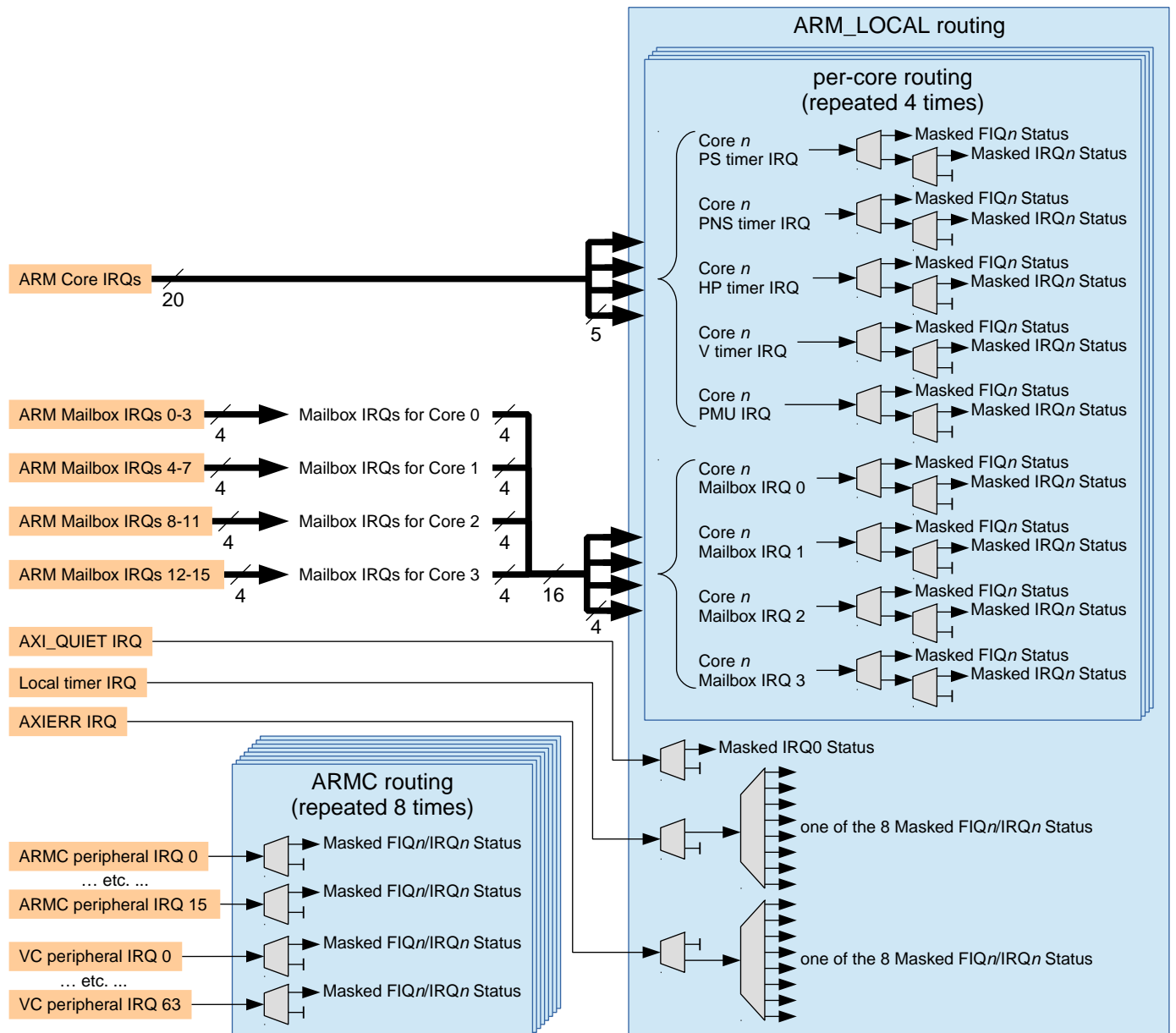
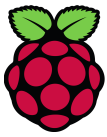


Figure 8. Legacy IRQ routing

The interrupts coming directly from each of the ARM cores (PS timer, PNS timer, HP timer, V timer and PMU) can only be routed to either the FIQ or IRQ of the core from which they originate. For example the PS timer and PMU IRQs from core 3 could be routed to FIQ3 and the PNS timer IRQ from core 2 could be routed to IRQ2. The masking of the ARM timer IRQs is controlled by the 4 `TIMER_CNTRL` registers (one for each core) and the masking of the PMU IRQs is controlled by the `PMU_CONTROL_SET` and `PMU_CONTROL_CLR` registers.

The sixteen ARM Mailbox interrupts are allocated so that four go to each core - ARM Mailbox IRQs 0 to 3 are routed to the four Mailbox IRQs on ARM core 0, and ARM Mailbox IRQs 12 to 15 are routed to the four Mailbox IRQs on ARM core 3, i.e. ARM Mailbox IRQ 13 appears to ARM Core 3 as Mailbox IRQ 1. Like the ARM Core interrupts, the ARM Mailbox IRQs can only be routed to the FIQ or IRQ of the core for which they are intended, for example the Mailbox 4 and 5 IRQs could be routed to FIQ1 and the Mailbox 10 IRQ could be routed to IRQ2. The masking of the ARM Mailbox IRQs is controlled by the four `MAILBOX_CNTRL` registers (one for each core).



The AXI\_QUIET IRQ is only available to the IRQ input on ARM core 0, and its masking is controlled by the AXI\_QUIET\_TIME register.

The Local timer and AXIERR IRQs can be routed to any one of the 8 FIQn/IRQn signals.

The masking of the Local timer IRQ is controlled by the LOCAL\_TIMER\_CONTROL and PERI\_IRQ\_ROUTE0 registers.

The masking of the AXIERR IRQ is controlled by the ARM\_CONTROL and CORE\_IRQ\_CONTROL registers.

The unmasked inputs to the "ARMC routing" block are readable from the IRQ\_STATUS0, IRQ\_STATUS1 and IRQ\_STATUS2 registers.

The masking within the "ARMC routing" block is controlled by the SET\_EN\_0, SET\_EN\_1, SET\_EN\_2, CLR\_EN\_0, CLR\_EN\_1 and CLR\_EN\_2 registers. Each of these registers is repeated for each of the eight FIQn/IRQn signals (48 registers in total).

Once the interrupts have been masked and routed, their statuses can be read from the 3 PENDING and 1 SOURCE registers (repeated 8 times for each of the FIQn/IRQn signals, for a total of 32 registers).

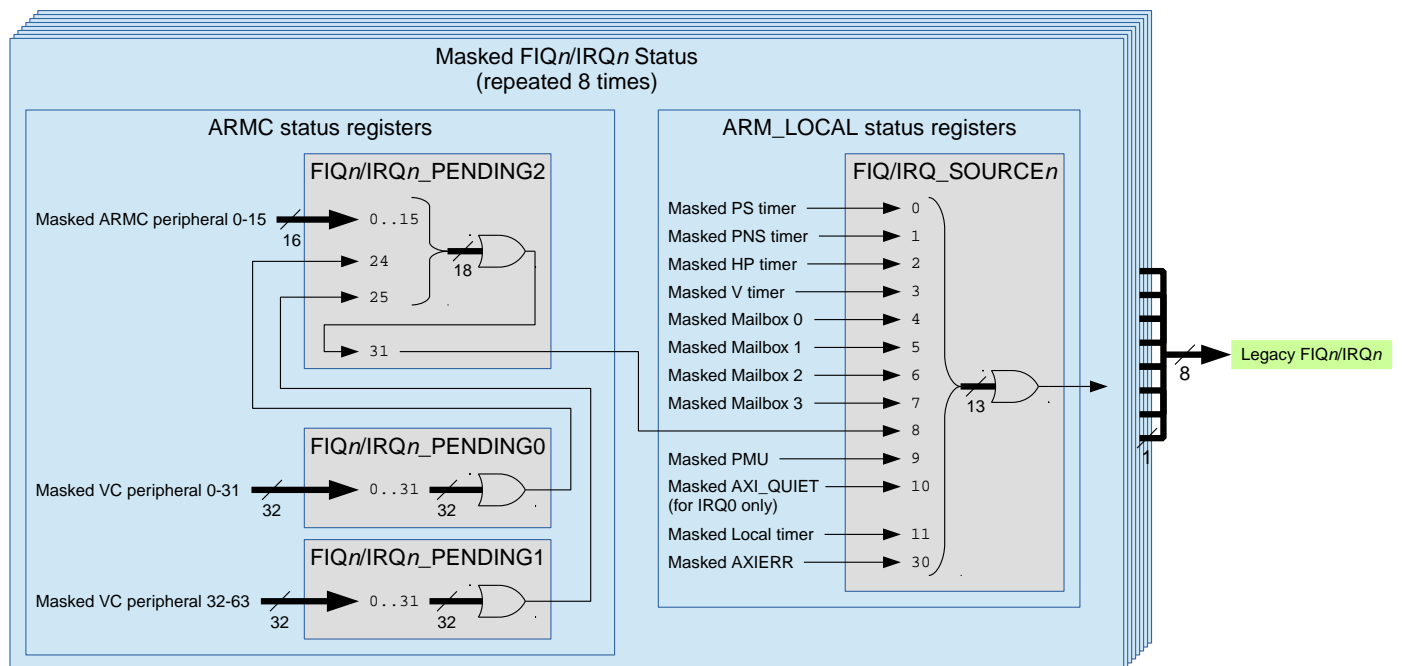


Figure 9. Legacy IRQ status registers

These are "nested" status registers, which means if bit 8 in the SOURCE register is set, you also need to read PENDING2 to see which bits are set there. If bit 24 in the PENDING2 register is set, then you also need to read PENDING0 to see which bits there are set.

As a more complete example, if the interrupt routing and masking is set up so that an interrupt from UART4 triggers a FIQ interrupt to ARM Core 3, the sequence (on ARM Core 3) would be:

1. Enter FIQ handler
2. Read FIQ\_SOURCE3
3. Find that FIQ\_SOURCE3[8] is set, so read FIQ3\_PENDING2
4. Find that FIQ3\_PENDING2[25] is set, so read FIQ3\_PENDING1



5. Find that FIQ3\_PENDING1[25] (i.e. VC peripheral IRQ 57) is set, so read PACTL\_CS[20:16] (see [Figure 6](#)) to see which UART triggered it
6. Find that PACTL\_CS[17] is set, so read UART4\_MIS to (finally) determine what caused the interrupt

## 6.5. Registers

To allow atomic operations (where only particular bits are modified, without modifying any of the other bits in the register), some registers are split into a write-set register and a write-clear register.

A **write-set register** allows you to set particular bits **high** (change them to 1). You set a bit high by writing a '1' to its bit-position - bits that were low get changed to high, and bits that were already high remain high. Any bit-positions written with a '0' retain their previous value.

| 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 write-set register containing 0x30840008 changes it to 0xFC86001C.

A **write-clear register** allows you to set particular bits **low** (change them to 0). You set a bit low by writing a '1' to its bit-position - bits that were low remain low, and bits that were high get changed to low. Any bit-positions written with a '0' retain their previous value. Note that you write a **one** to change a bit to **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 write-clear register containing 0x30840008 changes it to 0x00800008.

### 6.5.1. GIC-400

The base address of the GIC-400 is **0x4 C004 0000**. Note that, unlike other peripheral addresses in this document, this is an ARM-only address and not a legacy master address. If Low Peripheral mode is enabled this base address becomes **0x0 FF84 0000**.

The GIC-400 is configured with "NUM\_CPUS=4" and "NUM\_SPIS=192". For full register details, please refer to the ARM GIC-400 documentation on the [ARM Developer website](#).



## 6.5.2. ARM\_LOCAL

The ARM\_LOCAL register base address is **0x4 C000 0000**. Note that, unlike other peripheral addresses in this document, this is an ARM-only address and not a legacy master address. If Low Peripheral mode is enabled this base address becomes **0x0 FF80 0000**.

The PMU\_CONTROL\_SET / PMU\_CONTROL\_CLR registers are write-set / write-clear registers as described earlier.

| Address Offset | Register Name                       | Description                                      | Size |
|----------------|-------------------------------------|--|------|
| 0x00           | <a href="#">ARM_CONTROL</a>         | ARM Timer and AXI Error IRQ control              | 32   |
| 0x0C           | <a href="#">CORE_IRQ_CONTROL</a>    | VideoCore Interrupt Control                      | 32   |
| 0x10           | <a href="#">PMU_CONTROL_SET</a>     | PMU Bit Set                                      | 32   |
| 0x14           | <a href="#">PMU_CONTROL_CLR</a>     | PMU Bit Clear                                    | 32   |
| 0x24           | <a href="#">PERI_IRQ_ROUTE0</a>     | Peripheral Interrupt Routing (Bank 0)            | 32   |
| 0x30           | <a href="#">AXI_QUIET_TIME</a>      | AXI Outstanding Transaction Time and IRQ Control | 32   |
| 0x34           | <a href="#">LOCAL_TIMER_CONTROL</a> | Local Timer Control                              | 32   |
| 0x38           | <a href="#">LOCAL_TIMER_IRQ</a>     | Local Timer Reload and Interrupt                 | 32   |
| 0x40           | <a href="#">TIMER_CNTRL0</a>        | Timer Interrupt Control for ARM Core 0           | 32   |
| 0x44           | <a href="#">TIMER_CNTRL1</a>        | Timer Interrupt Control for ARM Core 1           | 32   |
| 0x48           | <a href="#">TIMER_CNTRL2</a>        | Timer Interrupt Control for ARM Core 2           | 32   |
| 0x4C           | <a href="#">TIMER_CNTRL3</a>        | Timer Interrupt Control for ARM Core 3           | 32   |
| 0x50           | <a href="#">MAILBOX_CNTRL0</a>      | Mailbox Interrupt Control for ARM Core 0         | 32   |
| 0x54           | <a href="#">MAILBOX_CNTRL1</a>      | Mailbox Interrupt Control for ARM Core 1         | 32   |
| 0x58           | <a href="#">MAILBOX_CNTRL2</a>      | Mailbox Interrupt Control for ARM Core 2         | 32   |
| 0x5C           | <a href="#">MAILBOX_CNTRL3</a>      | Mailbox Interrupt Control for ARM Core 3         | 32   |
| 0x60           | <a href="#">IRQ_SOURCE0</a>         | IRQ Source flags for ARM Core 0                  | 32   |
| 0x64           | <a href="#">IRQ_SOURCE1</a>         | IRQ Source flags for ARM Core 1                  | 32   |
| 0x68           | <a href="#">IRQ_SOURCE2</a>         | IRQ Source flags for ARM Core 2                  | 32   |
| 0x6C           | <a href="#">IRQ_SOURCE3</a>         | IRQ Source flags for ARM Core 3                  | 32   |
| 0x70           | <a href="#">FIQ_SOURCE0</a>         | FIQ Source flags for ARM Core 0                  | 32   |
| 0x74           | <a href="#">FIQ_SOURCE1</a>         | FIQ Source flags for ARM Core 1                  | 32   |
| 0x78           | <a href="#">FIQ_SOURCE2</a>         | FIQ Source flags for ARM Core 2                  | 32   |
| 0x7C           | <a href="#">FIQ_SOURCE3</a>         | FIQ Source flags for ARM Core 3                  | 32   |

Table 102. ARM\_LOCAL Interrupt Registers



## ARM\_CONTROL Register

### Synopsis

Main Timer and AXI Error Control.

| Bit(s) | Field Name      | Description  | Type | Reset |
|--------|-----------------|--|------|-------|
| 31:09  |                 | Reserved - Write as 0, read as don't care  |      |       |
| 08     | TIMER_INCREMENT | Main timer increment value selection<br>The main timer (driving the ARM core 'global system counter') is incremented by this amount each time the prescaler output is asserted. The ability to set the increment value to two allows the main timer to count ARM core clock cycles in the case where the AXI/APB clock frequency is half of the ARM core clock frequency and the prescaler ratio is unity.<br>1 = increment count by two.<br>0 = increment count by one. | RW   | 0x0   |
| 07     | PROC_CLK_TIMER  | Main timer clock selection<br>The main timer (driving the ARM core 'global system counter') may be driven either from the fast but variable AXI/APB bus clock or from the fixed-frequency but slower crystal reference clock.<br>1 = select AXI/APB clock.<br>0 = select crystal clock.  | RW   | 0x0   |
| 06     | AXIERRIRQ_EN    | When set to '1', this bit masks the AXI Error interrupt. An AXI error output is asserted by the ARM's L2 cache if an error response is received. If not masked, this causes an interrupt to be raised. If this bit is set, the interrupt is not raised.<br>Interrupt routing for this is controlled by the AXI_ERR_CORE field in the CORE_IRQ_CONTROL register.  | RW   | 0x0   |
| 05:00  |                 | Reserved - Write as 0, read as don't care  |      |       |

Table 103. ARM\_CONTROL Register

## CORE\_IRQ\_CONTROL Register

### Synopsis

VideoCore Interrupt Routing Control

| Bit(s) | Field Name | Description                               | Type | Reset |
|--------|------------|---|------|-------|
| 31:07  |            | Reserved - Write as 0, read as don't care |      |       |





| Bit(s) | Field Name                                | Description   | Type | Reset |
|--------|---|---|------|-------|
| 06:04  | AXI_ERR_CORE                              | Controls to which ARM core interrupt request pin the external error interrupt request signal from the ARM L2 cache is routed.<br>This interrupt is enabled in the AXIERRIRQ_EN field in the ARM_CONTROL register.<br>0 = CORE0_IRQ<br>1 = CORE1_IRQ<br>2 = CORE2_IRQ<br>3 = CORE3_IRQ<br>4 = CORE0_FIQ<br>5 = CORE1_FIQ<br>6 = CORE2_FIQ<br>7 = CORE3_FIQ | RW   | 0x0   |
| 03:00  | Reserved - Write as 0, read as don't care |   |      |       |

Table 104. CORE\_IRQ\_CONTROL Register

## PMU\_CONTROL\_SET Register

### Synopsis

Performance Monitoring Unit (PMU) control word. Each ARM core provides a PMUIRQ output; this control word specifies to which interrupt pins they are routed.

Writing a '1' to a bit position in this register causes the corresponding bit in the PMU control word to be set to 1.

| Bit(s) | Field Name                                | Description   | Type | Reset |
|--------|---|---|------|-------|
| 31:08  | Reserved - Write as 0, read as don't care |   |      |       |
| 07:04  | PMU_FIQ                                   | PMU to FIQ routing mask.<br>Setting bit N within this four bit field routes the PMUIRQ output from ARM core N to the FIQ interrupt request on that ARM core.  | RW   | 0x0   |
| 03:00  | PMU_IRQ                                   | PMU to IRQ routing mask<br>Setting bit N within this four bit field routes the PMUIRQ output from ARM core N to the IRQ interrupt request on that ARM core. Note that it is not possible to route the PMU interrupt to both the FIQ and IRQ request pins: setting a bit in the PMU_FIQ field causes the request to go to the core's FIQ pin only, irrespective of this field. | RW   | 0x0   |

Table 105. PMU\_CONTROL\_SET Register

## PMU\_CONTROL\_CLR Register



## Synopsis

Performance Monitoring Unit (PMU) control word. Each ARM core provides a PMUIRQ output; this control word specifies to which interrupt pins they are routed.

Writing a '1' to a bit position in this register causes the corresponding bit in the PMU control word to be cleared to 0.

| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 31:08  |            | Reserved - Write as 0, read as don't care   |      |       |
| 07:04  | PMU_FIQ    | PMU to FIQ routing mask.<br>Setting bit N within this four bit field routes the PMUIRQ output from ARM core N to the FIQ interrupt request on that ARM core.  | W1C  | 0x0   |
| 03:00  | PMU_IRQ    | PMU to IRQ routing mask<br>Setting bit N within this four bit field routes the PMUIRQ output from ARM core N to the IRQ interrupt request on that ARM core. Note that it is not possible to route the PMU interrupt to both the FIQ and IRQ request pins: setting a bit in the PMU_FIQ field causes the request to go to the core's FIQ pin only, irrespective of this field. | W1C  | 0x0   |

Table 106. PMU\_CONTROL\_CLR Register

## PERI\_IRQ\_ROUTE0 Register

### Synopsis

This register controls the routing of the Local timer interrupts.

| Bit(s) | Field Name      | Description   | Type | Reset |
|--------|-----------------|---|------|-------|
| 31:24  | WRITE_MASKS     | Interrupt routing field write mask bits.<br>This field must be written with 0x01, otherwise changes to LOCAL_TIMER_IRQ will be ignored.                     | RW   | 0x0   |
| 23:03  |                 | Reserved - Write as 0, read as don't care   |      |       |
| 02:00  | LOCAL_TIMER_IRQ | Local timer Routing<br>0 = CORE0_IRQ<br>1 = CORE1_IRQ<br>2 = CORE2_IRQ<br>3 = CORE3_IRQ<br>4 = CORE0_FIQ<br>5 = CORE1_FIQ<br>6 = CORE2_FIQ<br>7 = CORE3_FIQ | RW   | 0x0   |

Table 107. PERI\_IRQ\_ROUTE0 Register



## AXI\_QUIET\_TIME Register

### Synopsis

No outstanding AXI transactions for a while.

This register controls logic that is able to generate an interrupt to the IRQ interrupt pin of ARM core 0 if there has been no AXI bus traffic for a programmable time. The intention is that software can use this to have reasonable confidence that the bus traffic from the ARM cluster to VideoCore has ceased.

A 24-bit timer is loaded with a value equal to

$16 \times \text{AXI\_QUIET\_TIME} \cdot \text{AXI\_QUIET\_TIME} + 15$

whenever one or more AXI transactions are outstanding. The counter decrements on each AXI/APB clock rising edge when no transactions are outstanding. When the counter reaches zero, the interrupt request is generated if enabled.

| Bit(s) | Field Name        | Description   | Type | Reset |
|--------|-------------------|---|------|-------|
| 31:21  |                   | Reserved - Write as 0, read as don't care   |      |       |
| 20     | AXI_QUIET_IRQ_ENB | 1: Enable Core 0 IRQ on AXI quiet timer expiry<br>0: Disable Core 0 IRQ on AXI quiet timer expiry | RW   | 0x0   |
| 19:00  | AXI_QUIET_TIME    | Timer load value, in units of 16 AXI/APB clock cycles.  | RW   | 0x0   |

Table 108. AXI\_QUIET\_TIME Register

## LOCAL\_TIMER\_CONTROL Register

### Synopsis

Local Timer Configuration.

A free-running secondary timer is provided that can generate an interrupt each time it crosses zero. When it is enabled, the timer is decremented on each edge (positive or negative) of the crystal reference clock. It is automatically reloaded with the TIMER\_TIMEOUT value when it reaches zero and then continues to decrement.

Routing of the timer interrupt is controlled by the PERI\_IRQ\_ROUTE0 register.

| Bit(s) | Field Name     | Description  | Type | Reset |
|--------|----------------|--|------|-------|
| 31     | TIMER_IRQ_FLAG | This read-only field allows software to see the current state of the timer interrupt request. A '1' indicates a valid interrupt request.   | RO   | 0x0   |
| 30     |                | Reserved - Write as 0, read as don't care  |      |       |
| 29     | TIMER_IRQ_EN   | Interrupt request enable.<br>When set to '1', this bit causes the timer to request an interrupt as the timer crosses zero.   | RW   | 0x0   |
| 28     | TIMER_EN       | Timer Enable<br>When set to '1', this bit enables to the timer. When cleared to '0', timer operation is completely disabled: the timer does not decrement or raise interrupt requests. | RW   | 0x0   |



| Bit(s) | Field Name    | Description       | Type | Reset |
|--------|---------------|-------------------|------|-------|
| 27:00  | TIMER_TIMEOUT | Timer load value. | RW   | 0x0   |

Table 109. LOCAL\_TIMER\_CONTROL Register

## LOCAL\_TIMER\_IRQ Register

### Synopsis

Local Timer Interrupt Control

| Bit(s) | Field Name                                | Description   | Type | Reset |
|--------|---|---|------|-------|
| 31     | IRQ_CLEAR                                 | Write a '1' to this field to clear a timer interrupt request. If the timer crosses zero at the same time as the write, the clear operation will fail; interrupt request will remain asserted. This bit self-clears. | W1SC | 0x0   |
| 30     | RELOAD                                    | Write a '1' to this field to (re)load the timer with the timeout value. This bit self-clears.   | W1SC | 0x0   |
| 29:00  | Reserved - Write as 0, read as don't care |   |      |       |

Table 110. LOCAL\_TIMER\_IRQ Register

## TIMER\_CNTRL0, TIMER\_CNTRL1, TIMER\_CNTRL2 & TIMER\_CNTRL3 Registers

### Synopsis

This register allows software to determine the cause of a FIQ interrupt request received by an ARM core.

| Bit(s) | Field Name                                | Description  | Type | Reset |
|--------|---|--|------|-------|
| 31:08  | Reserved - Write as 0, read as don't care |  |      |       |
| 07     | CNT_V_IRQ_FIQ                             | When set to '1', this bit causes the 'Virtual Timer Event' output to be routed to the FIQ interrupt request.             | RW   | 0x0   |
| 06     | CNT_HP_IRQ_FIQ                            | When set to '1', this bit causes the 'Hypervisor Physical Timer Event' output to be routed to the FIQ interrupt request. | RW   | 0x0   |
| 05     | CNT_PNS_IRQ_FIQ                           | When set to '1', this bit causes the 'Nonsecure Physical Timer Event' output to be routed to the FIQ interrupt request.  | RW   | 0x0   |
| 04     | CNT_PS_IRQ_FIQ                            | When set to '1', this bit causes the 'Secure Physical Timer Event' output to be routed to the FIQ interrupt request.     | RW   | 0x0   |



| Bit(s) | Field Name  | Description   | Type | Reset |
|--------|-------------|---|------|-------|
| 03     | CNT_V_IRQ   | When set to '1', this bit causes the 'Virtual Timer Event' output to be routed to the IRQ interrupt request. Note that this is overridden by the corresponding FIQ bit: a particular event may be routed either to the FIQ or IRQ request pin, not both. If the FIQ bit is set, then the event will be routed to the FIQ request pin only, irrespective of the state of this bit.             | RW   | 0x0   |
| 02     | CNT_HP_IRQ  | When set to '1', this bit causes the 'Hypervisor Physical Timer Event' output to be routed to the IRQ interrupt request. Note that this is overridden by the corresponding FIQ bit: a particular event may be routed either to the FIQ or IRQ request pin, not both. If the FIQ bit is set, then the event will be routed to the FIQ request pin only, irrespective of the state of this bit. | RW   | 0x0   |
| 01     | CNT_PNS_IRQ | When set to '1', this bit causes the 'Nonsecure Physical Timer Event' output to be routed to the IRQ interrupt request. Note that this is overridden by the corresponding FIQ bit: a particular event may be routed either to the FIQ or IRQ request pin, not both. If the FIQ bit is set, then the event will be routed to the FIQ request pin only, irrespective of the state of this bit.  | RW   | 0x0   |
| 00     | CNT_PS_IRQ  | When set to '1', this bit causes the 'Secure Physical Timer Event' output to be routed to the IRQ interrupt request. Note that this is overridden by the corresponding FIQ bit: a particular event may be routed either to the FIQ or IRQ request pin, not both. If the FIQ bit is set, then the event will be routed to the FIQ request pin only, irrespective of the state of this bit.     | RW   | 0x0   |

Table 111. *TIMER\_CNTRL0, TIMER\_CNTRL1, TIMER\_CNTRL2 & TIMER\_CNTRL3 Registers*

## MAILBOX\_CNTRL0, MAILBOX\_CNTRL1, MAILBOX\_CNTRL2 & MAILBOX\_CNTRL3 Registers

### Synopsis

This register controls the routing of the mailbox interrupts to an ARM core's IRQ or FIQ interrupt request pins. Each ARM can receive interrupts from four of the sixteen mailbox registers. For ARM core 0, these are mailboxes 0-3; for ARM core 1, mailboxes 4-7 and so on.

| Bit(s) | Field Name | Description                               | Type | Reset |
|--------|------------|---|------|-------|
| 31:08  |            | Reserved - Write as 0, read as don't care |      |       |



| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 07     | MBOX3_FIQ  | When set to '1', this bit causes the fourth mailbox, i.e. mailbox 4C+3 for ARM core number C, (so mailbox 3 for ARM core 0, 7 for ARM core 1, etc.) to trigger a FIQ interrupt when any bit is set in the mailbox.   | RW   | 0x0   |
| 06     | MBOX2_FIQ  | When set to '1', this bit causes the third mailbox, i.e. mailbox 4C+2 for ARM core number C, (so mailbox 2 for ARM core 0, 6 for ARM core 1, etc.) to trigger a FIQ interrupt when any bit is set in the mailbox.  | RW   | 0x0   |
| 05     | MBOX1_FIQ  | When set to '1', this bit causes the second mailbox, i.e. mailbox 4C+1 for ARM core number C, (so mailbox 1 for ARM core 0, 5 for ARM core 1, etc.) to trigger a FIQ interrupt when any bit is set in the mailbox.   | RW   | 0x0   |
| 04     | MBOX0_FIQ  | When set to '1', this bit causes the first mailbox, i.e. mailbox 4C for ARM core number C, (so mailbox 0 for ARM core 0, 4 for ARM core 1, etc.) to trigger a FIQ interrupt when any bit is set in the mailbox.  | RW   | 0x0   |
| 03     | MBOX3_IRQ  | When set to '1', this bit causes the fourth mailbox, i.e. mailbox 4C+3 for ARM core number C, (so mailbox 3 for ARM core 0, 7 for ARM core 1, etc.) to trigger an IRQ interrupt when any bit is set in the mailbox. Note that this is overridden by the corresponding FIQ bit: a particular event may be routed either to the FIQ or IRQ request pin, not both. If the FIQ bit is set, then the event will be routed to the FIQ request pin only, irrespective of the state of this bit. | RW   | 0x0   |
| 02     | MBOX2_IRQ  | When set to '1', this bit causes the third mailbox, i.e. mailbox 4C+2 for ARM core number C, (so mailbox 2 for ARM core 0, 6 for ARM core 1, etc.) to trigger an IRQ interrupt when any bit is set in the mailbox. Note that this is overridden by the corresponding FIQ bit: a particular event may be routed either to the FIQ or IRQ request pin, not both. If the FIQ bit is set, then the event will be routed to the FIQ request pin only, irrespective of the state of this bit.  | RW   | 0x0   |



| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 01     | MBOX1_IRQ  | When set to '1', this bit causes the second mailbox, i.e. mailbox 4C+1 for ARM core number C, (so mailbox 1 for ARM core 0, 5 for ARM core 1, etc.) to trigger an IRQ interrupt when any bit is set in the mailbox. Note that this is overridden by the corresponding FIQ bit: a particular event may be routed either to the FIQ or IRQ request pin, not both. If the FIQ bit is set, then the event will be routed to the FIQ request pin only, irrespective of the state of this bit. | RW   | 0x0   |
| 00     | MBOX0_IRQ  | When set to '1', this bit causes the first mailbox, i.e. mailbox 4C for ARM core number C, (so mailbox 0 for ARM core 0, 4 for ARM core 1, etc.) to trigger an IRQ interrupt when any bit is set in the mailbox. Note that this is overridden by the corresponding FIQ bit: a particular event may be routed either to the FIQ or IRQ request pin, not both. If the FIQ bit is set, then the event will be routed to the FIQ request pin only, irrespective of the state of this bit.    | RW   | 0x0   |

Table 112. MAILBOX\_CNTRL0, MAILBOX\_CNTRL1, MAILBOX\_CNTRL2 & MAILBOX\_CNTRL3 Registers

## IRQ\_SOURCE0, IRQ\_SOURCE1, IRQ\_SOURCE2 & IRQ\_SOURCE3 Registers

### Synopsis

This register allows software to determine the cause of an IRQ interrupt request received by an ARM core.

| Bit(s) | Field Name                                | Description   | Type | Reset |
|--------|---|---|------|-------|
| 31     | Reserved - Write as 0, read as don't care |   |      |       |
| 30     | AXI_IRQ                                   | AXI error, as reported by the ARM L2 cache.   | RO   | 0x0   |
| 29:12  | Reserved - Write as 0, read as don't care |   |      |       |
| 11     | TIMER_IRQ                                 | Local timer interrupt.  | RO   | 0x0   |
| 10     | AXI_QUIET                                 | No AXI outstanding requests have been seen for the time-out period.<br><b>Present for Core 0 only. Reserved for others.</b> | RO   | 0x0   |
| 09     | PMU_IRQ                                   | Performance measurement unit interrupt.   | RO   | 0x0   |
| 08     | CORE_IRQ                                  | VideoCore interrupt request.  | RO   | 0x0   |
| 07:04  | MAILBOX_IRQ                               | Mailbox interrupts: bit 4 is the first of the core's mailboxes, bit 7 is the fourth.  | RO   | 0x0   |
| 03     | CNT_V_IRQ                                 | Virtual Timer Event interrupt.  | RO   | 0x0   |
| 02     | CNT_HP_IRQ                                | Hypervisor Physical Timer Event interrupt.  | RO   | 0x0   |



| Bit(s) | Field Name  | Description                               | Type | Reset |
|--------|-------------|---|------|-------|
| 01     | CNT_PNS_IRQ | Nonsecure Physical Timer Event interrupt. | RO   | 0x0   |
| 00     | CNT_PS_IRQ  | Secure Physical Timer Event interrupt.    | RO   | 0x0   |

Table 113. IRQ\_SOURCE0, IRQ\_SOURCE1, IRQ\_SOURCE2 & IRQ\_SOURCE3 Registers

## FIQ\_SOURCE0, FIQ\_SOURCE1, FIQ\_SOURCE2 & FIQ\_SOURCE3 Registers

### Synopsis

This register allows software to determine the cause of a FIQ interrupt request received by an ARM core.

| Bit(s) | Field Name                                | Description  | Type | Reset |
|--------|---|--|------|-------|
| 31     | Reserved - Write as 0, read as don't care |  |      |       |
| 30     | AXI_FIQ                                   | AXI error, as reported by the ARM L2 cache.  | RO   | 0x0   |
| 29:12  | Reserved - Write as 0, read as don't care |  |      |       |
| 11     | LOCAL_TIMER_FIQ                           | Local timer interrupt.   | RO   | 0x0   |
| 10     | Reserved - Write as 0, read as don't care |  |      |       |
| 09     | PMU_FIQ                                   | Performance measurement unit interrupt.  | RO   | 0x0   |
| 08     | CORE_FIQ                                  | VideoCore interrupt request.   | RO   | 0x0   |
| 07:04  | MAILBOX_FIQ                               | Mailbox interrupts: bit 4 is the first of the core's mailboxes, bit 7 is the fourth. | RO   | 0x0   |
| 03     | CNT_V_FIQ                                 | Virtual Timer Event interrupt.   | RO   | 0x0   |
| 02     | CNT_HP_FIQ                                | Hypervisor Physical Timer Event interrupt.   | RO   | 0x0   |
| 01     | CNT_PNS_FIQ                               | Nonsecure Physical Timer Event interrupt.  | RO   | 0x0   |
| 00     | CNT_PS_FIQ                                | Secure Physical Timer Event interrupt.   | RO   | 0x0   |

Table 114. FIQ\_SOURCE0, FIQ\_SOURCE1, FIQ\_SOURCE2 & FIQ\_SOURCE3 Registers

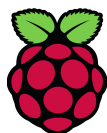
### 6.5.3. ARMC

The IRQn\_SET\_EN\_x / IRQn\_CLR\_EN\_x, FIQn\_SET\_EN\_x / FIQn\_CLR\_EN\_x and SWIRQ\_SET / SWIRQ\_CLEAR registers are write-set / write-clear registers as described earlier.

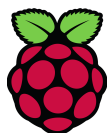
The ARMC register base address is **0x7E00 B000**.

| Address Offset | Register Name                 | Description  | Size |
|----------------|-------------------------------|--|------|
| 0x200          | <a href="#">IRQ0_PENDING0</a> | ARM Core 0 IRQ Enabled Interrupt Pending bits [31:0] | 32   |





| Address Offset | Register Name                 | Description   | Size |
|----------------|-------------------------------|---|------|
| 0x204          | <a href="#">IRQ0_PENDING1</a> | ARM Core 0 IRQ Enabled Interrupt pending bits [63:32] | 32   |
| 0x208          | <a href="#">IRQ0_PENDING2</a> | ARM Core 0 IRQ Enabled Interrupt pending bits [79:64] | 32   |
| 0x210          | <a href="#">IRQ0_SET_EN_0</a> | Write to Set ARM Core 0 IRQ enable bits [31:0]        | 32   |
| 0x214          | <a href="#">IRQ0_SET_EN_1</a> | Write to Set ARM Core 0 IRQ enable bits [63:32]       | 32   |
| 0x218          | <a href="#">IRQ0_SET_EN_2</a> | Write to Set ARM Core 0 IRQ enable bits[79:64]        | 32   |
| 0x220          | <a href="#">IRQ0_CLR_EN_0</a> | Write to Clear ARM Core 0 IRQ enable bits [31:0]      | 32   |
| 0x224          | <a href="#">IRQ0_CLR_EN_1</a> | Write to Clear ARM Core 0 IRQ enable bits [63:32]     | 32   |
| 0x228          | <a href="#">IRQ0_CLR_EN_2</a> | Write to Clear ARM Core 0 IRQ enable bits [79:64]     | 32   |
| 0x230          | <a href="#">IRQ_STATUS0</a>   | Interrupt Line bits [31:0]                            | 32   |
| 0x234          | <a href="#">IRQ_STATUS1</a>   | Interrupt Line bits [63:32]                           | 32   |
| 0x238          | <a href="#">IRQ_STATUS2</a>   | Interrupt Line bits [79:64]                           | 32   |
| 0x240          | <a href="#">IRQ1_PENDING0</a> | ARM Core 1 IRQ Enabled Interrupt pending bits [31:0]  | 32   |
| 0x244          | <a href="#">IRQ1_PENDING1</a> | ARM Core 1 IRQ Enabled Interrupt pending bits [63:32] | 32   |
| 0x248          | <a href="#">IRQ1_PENDING2</a> | ARM Core 1 IRQ Enabled Interrupt pending bits [79:64] | 32   |
| 0x250          | <a href="#">IRQ1_SET_EN_0</a> | Write to Set ARM Core 1 IRQ enable bits [31:0]        | 32   |
| 0x254          | <a href="#">IRQ1_SET_EN_1</a> | Write to Set ARM Core 1 IRQ enable bits [63:32]       | 32   |
| 0x258          | <a href="#">IRQ1_SET_EN_2</a> | Write to Set ARM Core 1 IRQ enable bits[79:64]        | 32   |
| 0x260          | <a href="#">IRQ1_CLR_EN_0</a> | Write to Clear ARM Core 1 IRQ enable bits [31:0]      | 32   |
| 0x264          | <a href="#">IRQ1_CLR_EN_1</a> | Write to Clear ARM Core 1 IRQ enable bits [63:32]     | 32   |
| 0x268          | <a href="#">IRQ1_CLR_EN_2</a> | Write to Clear ARM Core 1 IRQ enable bits [79:64]     | 32   |
| 0x280          | <a href="#">IRQ2_PENDING0</a> | ARM Core 2 IRQ Enabled Interrupt pending bits [31:0]  | 32   |
| 0x284          | <a href="#">IRQ2_PENDING1</a> | ARM Core 2 IRQ Enabled Interrupt pending bits [63:32] | 32   |
| 0x288          | <a href="#">IRQ2_PENDING2</a> | ARM Core 2 IRQ Enabled Interrupt pending bits [79:64] | 32   |
| 0x290          | <a href="#">IRQ2_SET_EN_0</a> | Write to Set ARM Core 2 IRQ enable bits [31:0]        | 32   |
| 0x294          | <a href="#">IRQ2_SET_EN_1</a> | Write to Set ARM Core 2 IRQ enable bits [63:32]       | 32   |



| Address Offset | Register Name                 | Description   | Size |
|----------------|-------------------------------|---|------|
| 0x298          | <a href="#">IRQ2_SET_EN_2</a> | Write to Set ARM Core 2 IRQ enable bits[79:64]        | 32   |
| 0x2A0          | <a href="#">IRQ2_CLR_EN_0</a> | Write to Clear ARM Core 2 IRQ enable bits [31:0]      | 32   |
| 0x2A4          | <a href="#">IRQ2_CLR_EN_1</a> | Write to Clear ARM Core 2 IRQ enable bits [63:32]     | 32   |
| 0x2A8          | <a href="#">IRQ2_CLR_EN_2</a> | Write to Clear ARM Core 2 IRQ enable bits [79:64]     | 32   |
| 0x2C0          | <a href="#">IRQ3_PENDING0</a> | ARM Core 3 IRQ Enabled Interrupt pending bits [31:0]  | 32   |
| 0x2C4          | <a href="#">IRQ3_PENDING1</a> | ARM Core 3 IRQ Enabled Interrupt pending bits [63:32] | 32   |
| 0x2C8          | <a href="#">IRQ3_PENDING2</a> | ARM Core 3 IRQ Enabled Interrupt pending bits [79:64] | 32   |
| 0x2D0          | <a href="#">IRQ3_SET_EN_0</a> | Write to Set ARM Core 3 IRQ enable bits [31:0]        | 32   |
| 0x2D4          | <a href="#">IRQ3_SET_EN_1</a> | Write to Set ARM Core 3 IRQ enable bits [63:32]       | 32   |
| 0x2D8          | <a href="#">IRQ3_SET_EN_2</a> | Write to Set ARM Core 3 IRQ enable bits[79:64]        | 32   |
| 0x2E0          | <a href="#">IRQ3_CLR_EN_0</a> | Write to Clear ARM Core 3 IRQ enable bits [31:0]      | 32   |
| 0x2E4          | <a href="#">IRQ3_CLR_EN_1</a> | Write to Clear ARM Core 3 IRQ enable bits [63:32]     | 32   |
| 0x2E8          | <a href="#">IRQ3_CLR_EN_2</a> | Write to Clear ARM Core 3 IRQ enable bits [79:64]     | 32   |
| 0x300          | <a href="#">FIQ0_PENDING0</a> | ARM Core 0 FIQ Enabled Interrupt pending bits [31:0]  | 32   |
| 0x304          | <a href="#">FIQ0_PENDING1</a> | ARM Core 0 FIQ Enabled Interrupt pending bits [63:32] | 32   |
| 0x308          | <a href="#">FIQ0_PENDING2</a> | ARM Core 0 FIQ Enabled Interrupt pending bits [79:64] | 32   |
| 0x310          | <a href="#">FIQ0_SET_EN_0</a> | Write to Set ARM Core 0 FIQ enable bits [31:0]        | 32   |
| 0x314          | <a href="#">FIQ0_SET_EN_1</a> | Write to Set ARM Core 0 FIQ enable bits [63:32]       | 32   |
| 0x318          | <a href="#">FIQ0_SET_EN_2</a> | Write to Set ARM Core 0 FIQ enable bits[79:64]        | 32   |
| 0x320          | <a href="#">FIQ0_CLR_EN_0</a> | Write to Clear ARM Core 0 FIQ enable bits [31:0]      | 32   |
| 0x324          | <a href="#">FIQ0_CLR_EN_1</a> | Write to Clear ARM Core 0 FIQ enable bits [63:32]     | 32   |
| 0x328          | <a href="#">FIQ0_CLR_EN_2</a> | Write to Clear ARM Core 0 FIQ enable bits [79:64]     | 32   |
| 0x340          | <a href="#">FIQ1_PENDING0</a> | ARM Core 1 FIQ Enabled Interrupt pending bits [31:0]  | 32   |
| 0x344          | <a href="#">FIQ1_PENDING1</a> | ARM Core 1 FIQ Enabled Interrupt pending bits [63:32] | 32   |
| 0x348          | <a href="#">FIQ1_PENDING2</a> | ARM Core 1 FIQ Enabled Interrupt pending bits [79:64] | 32   |



| Address Offset | Register Name                 | Description   | Size |
|----------------|-------------------------------|---|------|
| 0x350          | <a href="#">FIQ1_SET_EN_0</a> | Write to Set ARM Core 1 FIQ enable bits [31:0]        | 32   |
| 0x354          | <a href="#">FIQ1_SET_EN_1</a> | Write to Set ARM Core 1 FIQ enable bits [63:32]       | 32   |
| 0x358          | <a href="#">FIQ1_SET_EN_2</a> | Write to Set ARM Core 1 FIQ enable bits[79:64]        | 32   |
| 0x360          | <a href="#">FIQ1_CLR_EN_0</a> | Write to Clear ARM Core 1 FIQ enable bits [31:0]      | 32   |
| 0x364          | <a href="#">FIQ1_CLR_EN_1</a> | Write to Clear ARM Core 1 FIQ enable bits [63:32]     | 32   |
| 0x368          | <a href="#">FIQ1_CLR_EN_2</a> | Write to Clear ARM Core 1 FIQ enable bits [79:64]     | 32   |
| 0x380          | <a href="#">FIQ2_PENDING0</a> | ARM Core 2 FIQ Enabled Interrupt pending bits [31:0]  | 32   |
| 0x384          | <a href="#">FIQ2_PENDING1</a> | ARM Core 2 FIQ Enabled Interrupt pending bits [63:32] | 32   |
| 0x388          | <a href="#">FIQ2_PENDING2</a> | ARM Core 2 FIQ Enabled Interrupt pending bits [79:64] | 32   |
| 0x390          | <a href="#">FIQ2_SET_EN_0</a> | Write to Set ARM Core 2 FIQ enable bits [31:0]        | 32   |
| 0x394          | <a href="#">FIQ2_SET_EN_1</a> | Write to Set ARM Core 2 FIQ enable bits [63:32]       | 32   |
| 0x398          | <a href="#">FIQ2_SET_EN_2</a> | Write to Set ARM Core 2 FIQ enable bits[79:64]        | 32   |
| 0x3A0          | <a href="#">FIQ2_CLR_EN_0</a> | Write to Clear ARM Core 2 FIQ enable bits [31:0]      | 32   |
| 0x3A4          | <a href="#">FIQ2_CLR_EN_1</a> | Write to Clear ARM Core 2 FIQ enable bits [63:32]     | 32   |
| 0x3A8          | <a href="#">FIQ2_CLR_EN_2</a> | Write to Clear ARM Core 2 FIQ enable bits [79:64]     | 32   |
| 0x3C0          | <a href="#">FIQ3_PENDING0</a> | ARM Core 3 FIQ Enabled Interrupt pending bits [31:0]  | 32   |
| 0x3C4          | <a href="#">FIQ3_PENDING1</a> | ARM Core 3 FIQ Enabled Interrupt pending bits [63:32] | 32   |
| 0x3C8          | <a href="#">FIQ3_PENDING2</a> | ARM Core 3 FIQ Enabled Interrupt pending bits [79:64] | 32   |
| 0x3D0          | <a href="#">FIQ3_SET_EN_0</a> | Write to Set ARM Core 3 FIQ enable bits [31:0]        | 32   |
| 0x3D4          | <a href="#">FIQ3_SET_EN_1</a> | Write to Set ARM Core 3 FIQ enable bits [63:32]       | 32   |
| 0x3D8          | <a href="#">FIQ3_SET_EN_2</a> | Write to Set ARM Core 3 FIQ enable bits[79:64]        | 32   |
| 0x3E0          | <a href="#">FIQ3_CLR_EN_0</a> | Write to Clear ARM Core 3 FIQ enable bits [31:0]      | 32   |
| 0x3E4          | <a href="#">FIQ3_CLR_EN_1</a> | Write to Clear ARM Core 3 FIQ enable bits [63:32]     | 32   |
| 0x3E8          | <a href="#">FIQ3_CLR_EN_2</a> | Write to Clear ARM Core 3 FIQ enable bits [79:64]     | 32   |
| 0x3F0          | <a href="#">SWIRQ_SET</a>     | Write to Set Software Interrupt sources               | 32   |
| 0x3F4          | <a href="#">SWIRQ_CLEAR</a>   | Write to Clear Software Interrupt sources             | 32   |

Table 115. ARMC Interrupt Registers



## IRQ0\_PENDING0, IRQ1\_PENDING0, IRQ2\_PENDING0 & IRQ3\_PENDING0 Registers

### Synopsis

Shows the status of the Enabled interrupts [31:0] (that will be OR-ed into the Core's interrupt line)  
Only Interrupts that are enabled will show up here

| Bit(s) | Field Name  | Description                   | Type | Reset |
|--------|-------------|-------------------------------|------|-------|
| 31:00  | VC_IRQ_31_0 | VideoCore interrupts 31 to 0. | RO   | 0x0   |

Table 116. IRQ0\_PENDING0, IRQ1\_PENDING0, IRQ2\_PENDING0 & IRQ3\_PENDING0 Registers

## IRQ0\_PENDING1, IRQ1\_PENDING1, IRQ2\_PENDING1 & IRQ3\_PENDING1 Registers

### Synopsis

Shows the status of the Enabled interrupts [63:32] (that will be OR-ed into the Core's interrupt line)  
Only Interrupts that are enabled will show up here

| Bit(s) | Field Name   | Description                    | Type | Reset |
|--------|--------------|--------------------------------|------|-------|
| 31:00  | VC_IRQ_63_32 | VideoCore interrupts 63 to 32. | RO   | 0x0   |

Table 117. IRQ0\_PENDING1, IRQ1\_PENDING1, IRQ2\_PENDING1 & IRQ3\_PENDING1 Registers

## IRQ0\_PENDING2, IRQ1\_PENDING2, IRQ2\_PENDING2 & IRQ3\_PENDING2 Registers

### Synopsis

Shows the status of the Enabled interrupts [79:64] (that will be OR-ed into the Core's interrupt line)  
Only Interrupts that are enabled will show up here

| Bit(s) | Field Name                                | Description   | Type | Reset |
|--------|---|---|------|-------|
| 31     | IRQ                                       | This is the value of the ARM interrupt input  | RO   | 0x0   |
| 30:26  | Reserved - Write as 0, read as don't care |   |      |       |
| 25     | INT63_32                                  | This bit is the logical OR of all the interrupt pending bits for interrupts 63 to 32. If set, read the PENDING1 register to determine which interrupts are pending from this set. | RO   | 0x0   |
| 24     | INT31_0                                   | This bit is the logical OR of all the interrupt pending bits for interrupts 31 to 0. If set, read the PENDING0 register to determine which interrupts are pending from this set.  | RO   | 0x0   |
| 23:16  | Reserved - Write as 0, read as don't care |   |      |       |



| Bit(s) | Field Name     | Description   | Type | Reset |
|--------|----------------|---|------|-------|
| 15:08  | SW_TRIG_INT    | These eight bits are software-triggered interrupts. By writing to the SWIRQ_SET register, software may set interrupt trigger bits. These interrupts can only be cleared by clearing the trigger bit by writing to the SWIRQ_CLEAR register. | RO   | 0x0   |
| 07     | ARM_AXI_ERROR  | ARM AXI error interrupt. This is set if the logic in the ARM block detects that an AXI error has occurred. This interrupt cannot be cleared other than by resetting the ARM complex.  | RO   | 0x0   |
| 06     | ARM_ADDR_ERROR | ARM address range error. This interrupt is set if the ARM attempts an AXI burst (ALEN > 0) access to VideoCore peripheral space.  | RO   | 0x0   |
| 05     | VPU_C1_HALT    | VPU Core 1 halted in debug mode.  | RO   | 0x0   |
| 04     | VPU_C0_C1_HALT | VPU Core 0 halted in debug mode, or (if enabled by bit 10 of the config register) VPU Core 1 halted in debug mode.  | RO   | 0x0   |
| 03     | BELL_IRQ1      | Doorbell 1 interrupt. This interrupt can be cleared by reading the relevant doorbell register.  | RO   | 0x0   |
| 02     | BELL_IRQ0      | Doorbell 0 interrupt. This interrupt can be cleared by reading the relevant doorbell register.  | RO   | 0x0   |
| 01     | MAILBOX_IRQ0   | Mailbox 0 interrupt.  | RO   | 0x0   |
| 00     | TIMER_IRQ      | Timer interrupt. This interrupt can be cleared by writing to the IRQCNTL register.  | RO   | 0x0   |

Table 118. IRQ0\_PENDING2, IRQ1\_PENDING2, IRQ2\_PENDING2 & IRQ3\_PENDING2 Registers

## IRQ0\_SET\_EN\_0, IRQ1\_SET\_EN\_0, IRQ2\_SET\_EN\_0 & IRQ3\_SET\_EN\_0 Registers

### Synopsis

Writing a '1' to a bit position in this register enables the corresponding interrupt.  
A read returns the current state of this enable register.

| Bit(s) | Field Name  | Description                   | Type | Reset |
|--------|-------------|-------------------------------|------|-------|
| 31:00  | VC_IRQ_31_0 | VideoCore interrupts 31 to 0. | RW   | 0x0   |

Table 119. IRQ0\_SET\_EN\_0, IRQ1\_SET\_EN\_0, IRQ2\_SET\_EN\_0 & IRQ3\_SET\_EN\_0 Registers

## IRQ0\_SET\_EN\_1, IRQ1\_SET\_EN\_1, IRQ2\_SET\_EN\_1 & IRQ3\_SET\_EN\_1 Registers

### Synopsis

Writing a '1' to a bit position in this register enables the corresponding interrupt.  
A read returns the current state of this enable register.



| Bit(s) | Field Name   | Description                    | Type | Reset |
|--------|--------------|--------------------------------|------|-------|
| 31:00  | VC_IRQ_63_32 | VideoCore interrupts 63 to 32. | RW   | 0x0   |

Table 120. IRQ0\_SET\_EN\_1, IRQ1\_SET\_EN\_1, IRQ2\_SET\_EN\_1 & IRQ3\_SET\_EN\_1 Registers

## IRQ0\_SET\_EN\_2, IRQ1\_SET\_EN\_2, IRQ2\_SET\_EN\_2 & IRQ3\_SET\_EN\_2 Registers

### Synopsis

Writing a '1' to a bit position in this register enables the corresponding interrupt.

A read returns the current state of this enable register.

| Bit(s) | Field Name                                | Description  | Type | Reset |
|--------|---|--|------|-------|
| 31     | IRQ                                       | This is the value of the ARM interrupt input   | RW   | 0x0   |
| 30:16  | Reserved - Write as 0, read as don't care |  |      |       |
| 15:08  | SW_TRIG_INT                               | These eight bits are software-triggered interrupts. By writing to the SWIRQ_SET register, software may set interrupt trigger bits. | RW   | 0x0   |
| 07     | ARM_AXI_ERROR                             | ARM AXI error interrupt. This is set if the logic in the ARM block detects that an AXI error has occurred.                         | RW   | 0x0   |
| 06     | ARM_ADDR_ERROR                            | ARM address range error. This interrupt is set if the ARM attempts an AXI burst (ALEN > 0) access to VideoCore peripheral space.   | RW   | 0x0   |
| 05     | VPU_C1_HALT                               | VPU Core 1 halted in debug mode.   | RW   | 0x0   |
| 04     | VPU_C0_C1_HALT                            | VPU Core 0 halted in debug mode, or (if enabled by bit 10 of the config register) VPU Core 1 halted in debug mode.                 | RW   | 0x0   |
| 03     | BELL_IRQ1                                 | Doorbell 1 interrupt.  | RW   | 0x0   |
| 02     | BELL_IRQ0                                 | Doorbell 0 interrupt.  | RW   | 0x0   |
| 01     | MAILBOX_IRQ0                              | Mailbox 0 interrupt.   | RW   | 0x0   |
| 00     | TIMER_IRQ                                 | Timer interrupt.   | RW   | 0x0   |

Table 121. IRQ0\_SET\_EN\_2, IRQ1\_SET\_EN\_2, IRQ2\_SET\_EN\_2 & IRQ3\_SET\_EN\_2 Registers

## IRQ0\_CLR\_EN\_0, IRQ1\_CLR\_EN\_0, IRQ2\_CLR\_EN\_0 & IRQ3\_CLR\_EN\_0 Registers

### Synopsis

Writing a '1' to a bit position in this register disables the corresponding interrupt.

A read returns the current state of the IRQ enable register.

| Bit(s) | Field Name  | Description                   | Type | Reset |
|--------|-------------|-------------------------------|------|-------|
| 31:00  | VC_IRQ_31_0 | VideoCore interrupts 31 to 0. | W1C  | 0x0   |

Table 122. IRQ0\_CLR\_EN\_0, IRQ1\_CLR\_EN\_0, IRQ2\_CLR\_EN\_0 & IRQ3\_CLR\_EN\_0 Registers



## IRQ0\_CLR\_EN\_1, IRQ1\_CLR\_EN\_1, IRQ2\_CLR\_EN\_1 & IRQ3\_CLR\_EN\_1 Registers

### Synopsis

Writing a '1' to a bit position in this register disables the corresponding interrupt.  
A read returns the current state of the IRQ enable register.

| Bit(s) | Field Name   | Description                    | Type | Reset |
|--------|--------------|--------------------------------|------|-------|
| 31:00  | VC_IRQ_63_32 | VideoCore interrupts 63 to 32. | W1C  | 0x0   |

Table 123. IRQ0\_CLR\_EN\_1, IRQ1\_CLR\_EN\_1, IRQ2\_CLR\_EN\_1 & IRQ3\_CLR\_EN\_1 Registers

## IRQ0\_CLR\_EN\_2, IRQ1\_CLR\_EN\_2, IRQ2\_CLR\_EN\_2 & IRQ3\_CLR\_EN\_2 Registers

### Synopsis

Writing a '1' to a bit position in this register disables the corresponding interrupt.  
A read returns the current state of the IRQ enable register.

| Bit(s) | Field Name                                | Description  | Type | Reset |
|--------|---|--|------|-------|
| 31     | IRQ                                       | This is the value of the ARM interrupt input   | W1C  | 0x0   |
| 30:16  | Reserved - Write as 0, read as don't care |  |      |       |
| 15:08  | SW_TRIG_INT                               | These eight bits are software-triggered interrupts. By writing to the SWIRQ_SET register, software may set interrupt trigger bits. | W1C  | 0x0   |
| 07     | ARM_AXI_ERROR                             | ARM AXI error interrupt. This is set if the logic in the ARM block detects that an AXI error has occurred.                         | W1C  | 0x0   |
| 06     | ARM_ADDR_ERROR                            | ARM address range error. This interrupt is set if the ARM attempts an AXI burst (ALEN > 0) access to VideoCore peripheral space.   | W1C  | 0x0   |
| 05     | VPU_C1_HALT                               | VPU Core 1 halted in debug mode.   | W1C  | 0x0   |
| 04     | VPU_C0_C1_HALT                            | VPU Core 0 halted in debug mode, or (if enabled by bit 10 of the config register) VPU Core 1 halted in debug mode.                 | W1C  | 0x0   |
| 03     | BELL_IRQ1                                 | Doorbell 1 interrupt.  | W1C  | 0x0   |
| 02     | BELL_IRQ0                                 | Doorbell 0 interrupt.  | W1C  | 0x0   |
| 01     | MAILBOX_IRQ0                              | Mailbox 0 interrupt.   | W1C  | 0x0   |
| 00     | TIMER_IRQ                                 | Timer interrupt.   | W1C  | 0x0   |

Table 124. IRQ0\_CLR\_EN\_2, IRQ1\_CLR\_EN\_2, IRQ2\_CLR\_EN\_2 & IRQ3\_CLR\_EN\_2 Registers

## IRQ\_STATUS0 Register





## Synopsis

Shows the status of the actual Interrupts [31:0] before they are masked

| Bit(s) | Field Name  | Description                   | Type | Reset |
|--------|-------------|-------------------------------|------|-------|
| 31:00  | VC_IRQ_31_0 | VideoCore interrupts 31 to 0. | RO   | 0x0   |

Table 125. IRQ\_STATUS0 Register

## IRQ\_STATUS1 Register

### Synopsis

Shows the status of the actual Interrupts [63:32] before they are masked

| Bit(s) | Field Name   | Description                    | Type | Reset |
|--------|--------------|--------------------------------|------|-------|
| 31:00  | VC_IRQ_63_32 | VideoCore interrupts 63 to 32. | RO   | 0x0   |

Table 126. IRQ\_STATUS1 Register

## IRQ\_STATUS2 Register

### Synopsis

Shows the status of the actual Interrupts [79:64] before they are masked

| Bit(s) | Field Name                                | Description   | Type | Reset |
|--------|---|---|------|-------|
| 31     | IRQ                                       | This is the value of the ARM interrupt input  | RO   | 0x0   |
| 30:16  | Reserved - Write as 0, read as don't care |   |      |       |
| 15:08  | SW_TRIG_INT                               | These eight bits are software-triggered interrupts. By writing to the SWIRQ_SET register, software may set interrupt trigger bits. These interrupts can only be cleared by clearing the trigger bit by writing to the SWIRQ_CLEAR register. | RO   | 0x0   |
| 07     | ARM_AXI_ERROR                             | ARM AXI error interrupt. This is set if the logic in the ARM block detects that an AXI error has occurred. This interrupt cannot be cleared other than by resetting the ARM complex.  | RO   | 0x0   |
| 06     | ARM_ADDR_ERROR                            | ARM address range error. This interrupt is set if the ARM attempts an AXI burst (ALEN > 0) access to VideoCore peripheral space.  | RO   | 0x0   |
| 05     | VPU_C1_HALT                               | VPU Core 1 halted in debug mode.  | RO   | 0x0   |
| 04     | VPU_C0_C1_HALT                            | VPU Core 0 halted in debug mode, or (if enabled by bit 10 of the config register) VPU Core 1 halted in debug mode.  | RO   | 0x0   |





| Bit(s) | Field Name   | Description  | Type | Reset |
|--------|--------------|--|------|-------|
| 03     | BELL_IRQ1    | Doorbell 1 interrupt. This interrupt can be cleared by reading the relevant doorbell register. | RO   | 0x0   |
| 02     | BELL_IRQ0    | Doorbell 0 interrupt. This interrupt can be cleared by reading the relevant doorbell register. | RO   | 0x0   |
| 01     | MAILBOX_IRQ0 | Mailbox 0 interrupt.   | RO   | 0x0   |
| 00     | TIMER_IRQ    | Timer interrupt. This interrupt can be cleared by writing to the IRQCNTL register.             | RO   | 0x0   |

Table 127. IRQ\_STATUS2 Register

## FIQ0\_PENDING0, FIQ1\_PENDING0, FIQ2\_PENDING0 & FIQ3\_PENDING0 Registers

### Synopsis

Shows the status of the Enabled interrupts [31:0] (that will be OR-ed into the Core's interrupt line)  
Only Interrupts that are enabled will show up here

| Bit(s) | Field Name  | Description                   | Type | Reset |
|--------|-------------|-------------------------------|------|-------|
| 31:00  | VC_IRQ_31_0 | VideoCore interrupts 31 to 0. | RO   | 0x0   |

Table 128. FIQ0\_PENDING0, FIQ1\_PENDING0, FIQ2\_PENDING0 & FIQ3\_PENDING0 Registers

## FIQ0\_PENDING1, FIQ1\_PENDING1, FIQ2\_PENDING1 & FIQ3\_PENDING1 Registers

### Synopsis

Shows the status of the Enabled interrupts [63:32] (that will be OR-ed into the Core's interrupt line)  
Only Interrupts that are enabled will show up here

| Bit(s) | Field Name   | Description                    | Type | Reset |
|--------|--------------|--------------------------------|------|-------|
| 31:00  | VC_IRQ_63_32 | VideoCore interrupts 63 to 32. | RO   | 0x0   |

Table 129. FIQ0\_PENDING1, FIQ1\_PENDING1, FIQ2\_PENDING1 & FIQ3\_PENDING1 Registers

## FIQ0\_PENDING2, FIQ1\_PENDING2, FIQ2\_PENDING2 & FIQ3\_PENDING2 Registers

### Synopsis

Shows the status of the Enabled interrupts [79:64] (that will be OR-ed into the Core's interrupt line)  
Only Interrupts that are enabled will show up here

| Bit(s) | Field Name | Description                                  | Type | Reset |
|--------|------------|--|------|-------|
| 31     | IRQ        | This is the value of the ARM interrupt input | RO   | 0x0   |
| 30:26  |            | Reserved - Write as 0, read as don't care    |      |       |



| Bit(s) | Field Name                                | Description   | Type | Reset |
|--------|---|---|------|-------|
| 25     | INT63_32                                  | This bit is the logical OR of all the interrupt pending bits for interrupts 63 to 32. If set, read the PENDING1 register to determine which interrupts are pending from this set.   | RO   | 0x0   |
| 24     | INT31_0                                   | This bit is the logical OR of all the interrupt pending bits for interrupts 31 to 0. If set, read the PENDING0 register to determine which interrupts are pending from this set.  | RO   | 0x0   |
| 23:16  | Reserved - Write as 0, read as don't care |   |      |       |
| 15:08  | SW_TRIG_INT                               | These eight bits are software-triggered interrupts. By writing to the SWIRQ_SET register, software may set interrupt trigger bits. These interrupts can only be cleared by clearing the trigger bit by writing to the SWIRQ_CLEAR register. | RO   | 0x0   |
| 07     | ARM_AXI_ERROR                             | ARM AXI error interrupt. This is set if the logic in the ARM block detects that an AXI error has occurred. This interrupt cannot be cleared other than by resetting the ARM complex.  | RO   | 0x0   |
| 06     | ARM_ADDR_ERR OR                           | ARM address range error. This interrupt is set if the ARM attempts an AXI burst (ALEN > 0) access to VideoCore peripheral space.  | RO   | 0x0   |
| 05     | VPU_C1_HALT                               | VPU Core 1 halted in debug mode.  | RO   | 0x0   |
| 04     | VPU_C0_C1_HALT                            | VPU Core 0 halted in debug mode, or (if enabled by bit 10 of the config register) VPU Core 1 halted in debug mode.  | RO   | 0x0   |
| 03     | BELL_IRQ1                                 | Doorbell 1 interrupt. This interrupt can be cleared by reading the relevant doorbell register.  | RO   | 0x0   |
| 02     | BELL_IRQ0                                 | Doorbell 0 interrupt. This interrupt can be cleared by reading the relevant doorbell register.  | RO   | 0x0   |
| 01     | MAILBOX_IRQ0                              | Mailbox 0 interrupt.  | RO   | 0x0   |
| 00     | TIMER_IRQ                                 | Timer interrupt. This interrupt can be cleared by writing to the IRQCNTL register.  | RO   | 0x0   |

Table 130. FIQ0\_PENDING2, FIQ1\_PENDING2, FIQ2\_PENDING2 & FIQ3\_PENDING2 Registers

## FIQ0\_SET\_EN\_0, FIQ1\_SET\_EN\_0, FIQ2\_SET\_EN\_0 & FIQ3\_SET\_EN\_0 Registers

### Synopsis

Writing a '1' to a bit position in this register enables the corresponding interrupt.  
A read returns the current state of the FIQ enable register.



| Bit(s) | Field Name  | Description                   | Type | Reset |
|--------|-------------|-------------------------------|------|-------|
| 31:00  | VC_IRQ_31_0 | VideoCore interrupts 31 to 0. | RW   | 0x0   |

Table 131. *FIQ0\_SET\_EN\_0, FIQ1\_SET\_EN\_0, FIQ2\_SET\_EN\_0 & FIQ3\_SET\_EN\_0 Registers*

## FIQ0\_SET\_EN\_1, FIQ1\_SET\_EN\_1, FIQ2\_SET\_EN\_1 & FIQ3\_SET\_EN\_1 Registers

### Synopsis

Writing a '1' to a bit position in this register enables the corresponding interrupt.  
A read returns the current state of the FIQ enable register.

| Bit(s) | Field Name   | Description                    | Type | Reset |
|--------|--------------|--------------------------------|------|-------|
| 31:00  | VC_IRQ_63_32 | VideoCore interrupts 63 to 32. | RW   | 0x0   |

Table 132. *FIQ0\_SET\_EN\_1, FIQ1\_SET\_EN\_1, FIQ2\_SET\_EN\_1 & FIQ3\_SET\_EN\_1 Registers*

## FIQ0\_SET\_EN\_2, FIQ1\_SET\_EN\_2, FIQ2\_SET\_EN\_2 & FIQ3\_SET\_EN\_2 Registers

### Synopsis

Writing a '1' to a bit position in this register enables the corresponding interrupt.  
A read returns the current state of the FIQ enable register.

| Bit(s) | Field Name                                | Description  | Type | Reset |
|--------|---|--|------|-------|
| 31     | IRQ                                       | This is the value of the ARM interrupt input   | RW   | 0x0   |
| 30:16  | Reserved - Write as 0, read as don't care |  |      |       |
| 15:08  | SW_TRIG_INT                               | These eight bits are software-triggered interrupts. By writing to the SWIRQ_SET register, software may set interrupt trigger bits. | RW   | 0x0   |
| 07     | ARM_AXI_ERROR                             | ARM AXI error interrupt. This is set if the logic in the ARM block detects that an AXI error has occurred.                         | RW   | 0x0   |
| 06     | ARM_ADDR_ERROR                            | ARM address range error. This interrupt is set if the ARM attempts an AXI burst (ALEN > 0) access to VideoCore peripheral space.   | RW   | 0x0   |
| 05     | VPU_C1_HALT                               | VPU Core 1 halted in debug mode.   | RW   | 0x0   |
| 04     | VPU_C0_C1_HALT                            | VPU Core 0 halted in debug mode, or (if enabled by bit 10 of the config register) VPU Core 1 halted in debug mode.                 | RW   | 0x0   |
| 03     | BELL_IRQ1                                 | Doorbell 1 interrupt.  | RW   | 0x0   |
| 02     | BELL_IRQ0                                 | Doorbell 0 interrupt.  | RW   | 0x0   |
| 01     | MAILBOX_IRQ0                              | Mailbox 0 interrupt.   | RW   | 0x0   |
| 00     | TIMER_IRQ                                 | Timer interrupt.   | RW   | 0x0   |

Table 133. *FIQ0\_SET\_EN\_2, FIQ1\_SET\_EN\_2, FIQ2\_SET\_EN\_2 & FIQ3\_SET\_EN\_2 Registers*



## FIQ0\_CLR\_EN\_0, FIQ1\_CLR\_EN\_0, FIQ2\_CLR\_EN\_0 & FIQ3\_CLR\_EN\_0 Registers

### Synopsis

Writing a '1' to a bit position in this register disables the corresponding interrupt.  
A read returns the current state of the FIQ enable register.

| Bit(s) | Field Name  | Description                   | Type | Reset |
|--------|-------------|-------------------------------|------|-------|
| 31:00  | VC_IRQ_31_0 | VideoCore interrupts 31 to 0. | W1C  | 0x0   |

Table 134. FIQ0\_CLR\_EN\_0, FIQ1\_CLR\_EN\_0, FIQ2\_CLR\_EN\_0 & FIQ3\_CLR\_EN\_0 Registers

## FIQ0\_CLR\_EN\_1, FIQ1\_CLR\_EN\_1, FIQ2\_CLR\_EN\_1 & FIQ3\_CLR\_EN\_1 Registers

### Synopsis

Writing a '1' to a bit position in this register disables the corresponding interrupt.  
A read returns the current state of the FIQ enable register.

| Bit(s) | Field Name   | Description                    | Type | Reset |
|--------|--------------|--------------------------------|------|-------|
| 31:00  | VC_IRQ_63_32 | VideoCore interrupts 63 to 32. | W1C  | 0x0   |

Table 135. FIQ0\_CLR\_EN\_1, FIQ1\_CLR\_EN\_1, FIQ2\_CLR\_EN\_1 & FIQ3\_CLR\_EN\_1 Registers

## FIQ0\_CLR\_EN\_2, FIQ1\_CLR\_EN\_2, FIQ2\_CLR\_EN\_2 & FIQ3\_CLR\_EN\_2 Registers

### Synopsis

Writing a '1' to a bit position in this register disables the corresponding interrupt.  
A read returns the current state of the FIQ enable register.

| Bit(s) | Field Name                                | Description  | Type | Reset |
|--------|---|--|------|-------|
| 31     | IRQ                                       | This is the value of the ARM interrupt input   | W1C  | 0x0   |
| 30:16  | Reserved - Write as 0, read as don't care |  |      |       |
| 15:08  | SW_TRIG_INT                               | These eight bits are software-triggered interrupts. By writing to the SWIRQ_SET register, software may set interrupt trigger bits. | W1C  | 0x0   |
| 07     | ARM_AXI_ERROR                             | ARM AXI error interrupt. This is set if the logic in the ARM block detects that an AXI error has occurred.                         | W1C  | 0x0   |
| 06     | ARM_ADDR_ERROR                            | ARM address range error. This interrupt is set if the ARM attempts an AXI burst (ALEN > 0) access to VideoCore peripheral space.   | W1C  | 0x0   |
| 05     | VPU_C1_HALT                               | VPU Core 1 halted in debug mode.   | W1C  | 0x0   |
| 04     | VPU_C0_C1_HALT                            | VPU Core 0 halted in debug mode, or (if enabled by bit 10 of the config register) VPU Core 1 halted in debug mode.                 | W1C  | 0x0   |



| Bit(s) | Field Name   | Description           | Type | Reset |
|--------|--------------|-----------------------|------|-------|
| 03     | BELL_IRQ1    | Doorbell 1 interrupt. | W1C  | 0x0   |
| 02     | BELL_IRQ0    | Doorbell 0 interrupt. | W1C  | 0x0   |
| 01     | MAILBOX_IRQ0 | Mailbox 0 interrupt.  | W1C  | 0x0   |
| 00     | TIMER_IRQ    | Timer interrupt.      | W1C  | 0x0   |

Table 136. *FIQ0\_CLR\_EN\_2, FIQ1\_CLR\_EN\_2, FIQ2\_CLR\_EN\_2 & FIQ3\_CLR\_EN\_2 Registers*

## SWIRQ\_SET Register

### Synopsis

Software-triggered interrupts.

Writing a '1' to a bit position in this register sets the corresponding software interrupt source bit. A read returns the current state of the software interrupt bits.

| Bit(s) | Field Name | Description                               | Type | Reset |
|--------|------------|---|------|-------|
| 31:08  |            | Reserved - Write as 0, read as don't care |      |       |
| 07:00  | SW_INT     | Eight software-triggered interrupt bits.  | RW   | 0x0   |

Table 137. *SWIRQ\_SET Register*

## SWIRQ\_CLEAR Register

### Synopsis

Software-triggered interrupts.

Writing a '1' to a bit position in this register clears the corresponding software interrupt source bit. A read returns the current state of the software interrupt bits.

| Bit(s) | Field Name | Description                               | Type | Reset |
|--------|------------|---|------|-------|
| 31:08  |            | Reserved - Write as 0, read as don't care |      |       |
| 07:00  | SW_INT     | Eight software-triggered interrupt bits.  | W1C  | 0x0   |

Table 138. *SWIRQ\_CLEAR Register*

[11] which is only useful for the VPU and not the CPU



## Chapter 7. PCM / I2S Audio

### 7.1. Overview

The PCM (Pulse Code Modulation) audio interface is an APB peripheral providing input and output of telephony or high quality serial audio streams. It supports many classic PCM formats including I2S.

The PCM audio interface has 4 interface signals:

- PCM\_CLK - bit clock
- PCM\_FS - frame sync signal
- PCM\_DIN - serial data input
- PCM\_DOUT - serial data output

PCM is a serial format with a single-bit data\_in and single-bit data\_out. Data is always serialised MS-bit first.

The frame sync signal (PCM\_FS) is used to delimit the serial data into individual frames. The length of the frame, and the size and polarity of the frame sync, are fully programmable.

Frames can contain 1 or 2 audio/data channels in each direction. Each channel can be between 8 and 32 bits wide and can be positioned anywhere within the frame as long as the two channels don't overlap. The channel format is separately programmable for transmit and receive directions.

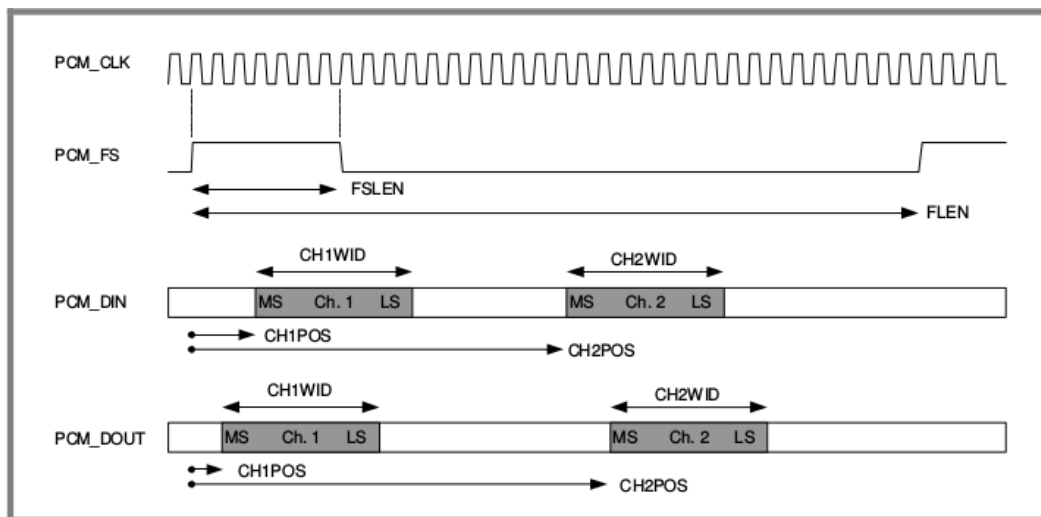


Figure 10. PCM Audio Interface Typical Timing

The PCM\_CLK can be asynchronous to the bus APB clock and can be logically inverted if required.

The direction of the PCM\_CLK and PCM\_FS signals can be individually selected, allowing the interface to act as a master or slave device.

The input interface is also capable of supporting up to 2 PDM (Pulse Density Modulation) microphones, as an alternative to the classic PCM input format, in conjunction with a PCM output.



## 7.2. Block Diagram

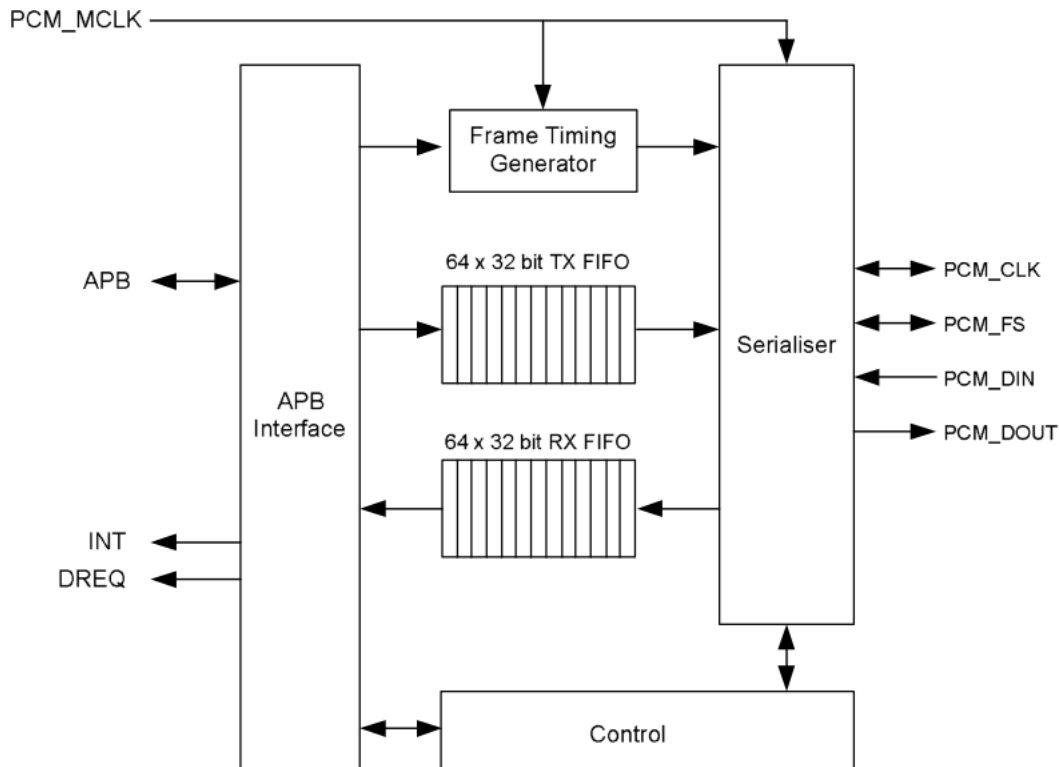


Figure 11. PCM Audio Interface Block Diagram

The PCM audio interface contains separate transmit and receive FIFOs. Note that if the frame contains two data channels, they must share the same FIFO and so the channel data will be interleaved. The block can be driven using simple polling, an interrupt based method or direct DMA control.

## 7.3. Typical Timing

Figure 10 shows typical interface timing and indicates the flexibility that the peripheral offers.

Normally PCM output signals change on the rising edge of PCM\_CLK and input signals are sampled on its falling edge. The frame sync is considered as a data signal and sampled in the same way.

The front end of the PCM audio interface is run off the PCM\_CLK and the PCM signals are timed against this clock. However, the polarity of the PCM\_CLK can be physically inverted, in which case the edges are reversed.

In clock master mode (CLKM=0), the PCM\_CLK is an output and is driven from the PCM\_MCLK clock input.

In clock slave mode (CLKM=1), the PCM\_CLK is an input, supplied by some external clock source.

In frame sync master mode (FSM=0), the PCM\_FS is internally generated and is treated as a data output that changes on the positive edge of the clock. The length and polarity of the frame sync is fully programmable and it can be used as a standard frame sync signal, or as an L-R signal for I2S.





In frame sync slave mode (FSM=1), the PCM\_FS is treated as a data input and is sampled on the negative edge of PCM\_CLK. The first clock of a frame is taken as the first clock period where PCM\_FS is sampled as a 1 following a period or periods where it was previously a 0. The PCM audio interface locks onto the incoming frame sync and uses this to indicate where the data channels are positioned. The precise timing at the start of frame is shown in [Figure 12](#).

Note that in frame sync slave mode there are two synchronising methods. The legacy method is used when the frame length = 0. In this case the internal frame logic has to detect the incoming PCM\_FS signal and reset the internal frame counter at the start of every frame. The logic relies on the PCM\_FS to indicate the length of the frame and so can cope with adjacent frames of different lengths. However, this creates a short timing path that will corrupt the PCM\_DOUT for one specific frame/channel setting.

The preferred method is to set the frame length to the expected length. Here the incoming PCM\_FS is used to resynchronise the internal frame counter and this eliminates the short timing path.

## 7.4. Operation

The PCM interface runs asynchronously at the PCM\_CLK rate and automatically transfers transmit and receive data across to the internal APB clock domain. The control registers (with the exception of INTSTC\_A and GRAY) are NOT synchronised and should be programmed before the device is enabled and should NOT be changed whilst the interface is running.

Only the EN, RXON and TXON bits of the PCMCS register are synchronised across the PCM - APB clock domain and are allowed to be changed whilst the interface is running.

The EN bit is a global power-saving enable. The TXON and RXON bits enable transmit and receive, and the interface is running whenever either TXON or RXON is enabled.

In operation, the PCM format is programmed by setting the appropriate frame length, frame sync, channel position values, and signal polarity controls. The transmit FIFO should be preloaded with data and the interface can then be enabled and started, and will run continuously until stopped. If the receive FIFO becomes full or the transmit FIFO becomes empty, the RXERR or TXERR error flags will be set, but the interface will just continue. If the RX FIFO overflows, new samples are discarded and if the TX FIFO underflows, zeros are transmitted.

Normally channel data is read or written into the appropriate FIFO as a single word. If the channel is less than 32 bits, the data is right justified and should be padded with zeros. If the RXSEX bit is set then the received data is sign extended up to the full 32 bits. When a frame is programmed to have two data channels, then each channel is written/read as a separate word in the FIFO, producing an interleaved data stream. When initialising the interface, the first word read out of the TX FIFO will be used for the first channel, and the data from the first channel on the first frame to be received will be the first word written into the RX FIFO.

If a FIFO error occurs in a two channel frame, then channel synchronisation may be lost which may result in a left-right audio channel swap. RXSYNC and TXSYNC status bits are provided to help determine if channel slip has occurred. They indicate if the number of words in the FIFO is a multiple of a full frame (taking into account where we are in the current frame being transferred). This assumes that an





integer number of frames data has been sent/read from the FIFOs.

If a frame is programmed to have two data channels and the packed mode bits are set (FRXP / FTXP) then the FIFOs are configured so that each word contains the data for both channels (2 x 16-bit samples). In this mode each word written to the TX FIFO contains two 16-bit samples, and the Least Significant sample is transmitted first. Each word read from the RX FIFO will contain the data received from two channels, the first channel received will be in the Least Significant half of the word. If the channel's size is less than 16 bits, the TX data will be truncated and RX data will be padded to 16 bits with zeros.

Note that data is always serialised MS-bit first. This is well-established behaviour in both PCM and I2S.

If the PDM input mode is enabled then channel 1 is sampled on the negative edge of PCM\_CLK whilst channel 2 is sampled on the positive edge of PCM\_CLK.

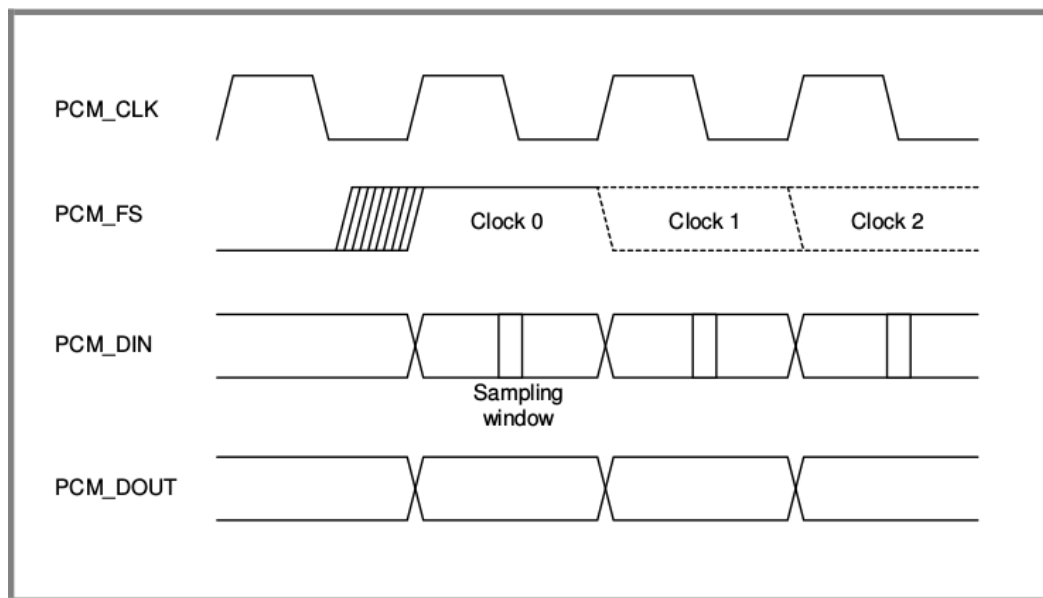


Figure 12. Timing at Start of Frame

Note that the precise timing of PCM\_FS (when it is an input) is not clearly defined and it may change state before or after the positive edge of the clock. Here the first clock of the frame is defined as the clock period where the PCM\_FS is sampled (on a negative edge of PCM\_CLK) as a 1 where it was previously sampled as a 0.

## 7.5. Software Operation

### 7.5.1. Operating in Polled mode

1. Set the EN bit to enable the PCM block. Set all operational values to define the frame and channel settings. Assert RXCLR and/or TXCLR and wait for 2 PCM clocks to ensure the FIFOs are reset. The SYNC bit can be used to determine when 2 clocks have passed. Set RXTHR/TXTHR to determine the FIFO thresholds.
2. If transmitting, ensure that sufficient sample words have been written to PCM FIFO before transmission is started. Set TXON and/or RXON to begin operation.



3. Poll TXW writing sample words to PCM FIFO and poll RXR reading sample words from PCM FIFO, until all data is transferred.

### 7.5.2. Operating in Interrupt mode

1. Set the EN bit to enable the PCM block. Set all operational values to define the frame and channel settings. Assert RXCLR and/or TXCLR and wait for 2 PCM clocks to ensure the FIFOs are reset. The SYNC bit can be used to determine when 2 clocks have passed. Set RXTHR/TXTHR to determine the FIFO thresholds.
2. Set INTR and/or INTT to enable interrupts.
3. If transmitting, ensure that sufficient sample words have been written to PCM FIFO before transmission is started. Set TXON and/or RXON to begin operation.
4. When an interrupt occurs, check RXR. If this is set then one or more sample words are available in PCM FIFO. If TXW is set then one or more sample words can be sent to PCM FIFO.

### 7.5.3. DMA

1. Set the EN bit to enable the PCM block. Set all operational values to define the frame and channel settings. Assert RXCLR and/or TXCLR and wait for 2 PCM clocks to ensure the FIFOs are reset. The SYNC bit can be used to determine when 2 clocks have passed.
2. Set DMAEN to enable DMA DREQ generation and set RX\_REQ/TX\_REQ to determine the FIFO thresholds for the DREQs. If required, set TX\_PANIC and RX\_PANIC to determine the level at which the DMA should increase its AXI priority,
3. In the DMA controllers set the correct DREQ channels, one for RX and one for TX. Start the DMA which should fill the TX FIFO.
4. Set TXON and/or RXON to begin operation.

## 7.6. Error Handling

In all software operational modes, the possibility of FIFO over- or under-run exists. Should this happen when using 2 channels per frame, there is a risk of losing sync with the channel data stored in the FIFO. If this happens and is not detected and corrected, then the data channels may become swapped.

The FIFOs will automatically detect an error condition caused by a FIFO over- or under-run and this will set the appropriate latching error bit in the control/status register. Writing a '1' back to this error bit will clear the latched flag.

In a system using a polled operation, the error bits can be checked manually. For an interrupt or DMA based system, setting the RXERR and/or TXERR bits in INTEN\_A will cause the PCM interface to generate an interrupt when an error is detected.

If a FIFO error occurs during operation in which 2 data channels are being used then the synchronisation of the data may be lost. This can be recovered by either of these two methods:



- Disable transmit and receive (set TXON and RXON to 0). Clear the FIFOs (set RXCLR and TXCLR to 1). Note that it may take up to 2 PCM clocks for the FIFOs to be physically cleared after initiating a clear. Then preload the transmit FIFO and restart transmission. This of course loses the data in the FIFO and further interrupts the data flow to the external device.
- Examine the TXSYNC and RXSYNC flags. These flags indicate if the amount of data in the FIFO is a whole number of frames, automatically taking into account where we are in the current frame being transmitted or received. Thus, providing an even number of samples was read or written to the FIFOs, then if the flags are set then this indicates that a single word needs to be written or read to adjust the data. Normal exchange of data can then proceed (where the first word in a data pair is for channel 1). This method should cause less disruption to the data stream.

## 7.7. PDM Input Mode Operation

The PDM input mode is capable of interfacing with two digital half-cycle PDM microphones and implements a 4<sup>th</sup> order CIC decimation filter with a selectable decimation factor. The clock input of the microphones is shared with the PCM output codec and it should be configured to provide the correct clock rate for the microphones. As a result it may be necessary to add a number of padding bits into the PCM output and configure the output codec to allow for this.

When using the PDM input mode the bit width and the rate of the data received will depend on the decimation factor used. Once the data has been read from the peripheral a further decimation and filtering stage will be required and can be implemented in software. The software filter should also correct the droop introduced by the CIC filter stage. Similarly a DC correction stage should also be employed.

| PDMN     | PCM_CLK (MHz) | Peripheral Output Format | OSR | Fs    |
|----------|---------------|--------------------------|-----|-------|
| 0 (N=16) | 3.072         | 16 bits unsigned         | 4   | 48kHz |
| 1 (N=32) | 3.072         | 20 bits unsigned         | 2   | 48kHz |

Table 139. PDM Input Mode Configuration

## 7.8. GRAY Code Input Mode Operation

GRAY mode is used for an incoming data stream only. GRAY mode is selected by setting the enable bit (EN) in the PCM\_GRAY register.

In this mode data is received on the PCM\_DIN (data) and the PCM\_FS (strobe) pins. The data is expected to be in data/strobe format. In this mode data is detected when either the data or the strobe change state. As each bit is received it is written into the RX buffer and when 32 bits are received they are written out to the RX FIFO as a 32-bit word. In order for this mode to work the user must program a PCM clock rate which is 4 times faster than the gray data rate. Also the gray coded data input signals should be clean.

The normal RX\_REQ and RXTHR FIFO levels will apply as for normal PCM received data.

If a message is received that is not a multiple of 32 bits, any data in the RX buffer can be flushed out by



setting the flush bit (FLUSH). Once set, this bit will read back as zero until the flush operation has completed. This may take several cycles as the APB clock may be many times faster than the PCM clock. Once the flush has occurred, the bits are packed up to 32 bits with zeros and written out to the RX FIFO. The flushed field (FLUSHED) will indicate how many of bits of this word are valid.

Note that to get an accurate indication of the number of bits currently in the RX shift register (RXLEVEL) the APB clock must be at least twice the PCM\_CLK.

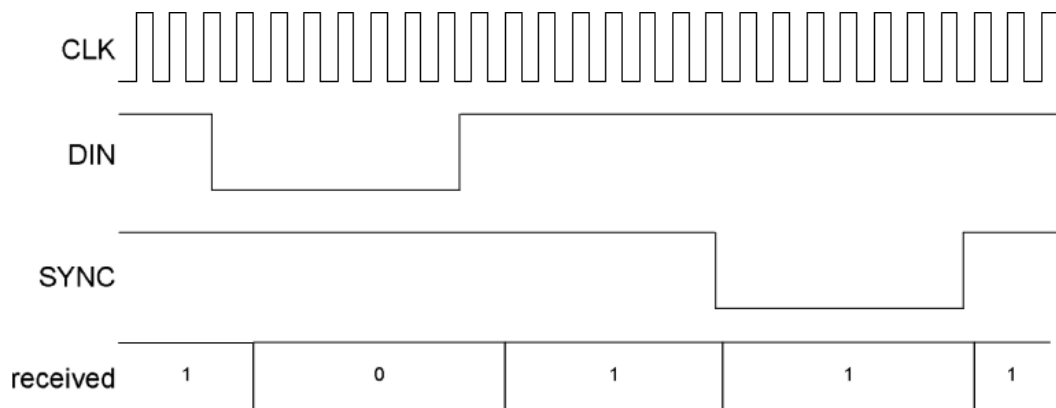


Figure 13. Gray mode input format

## 7.9. PCM Register Map

There is only one PCM module in the BCM2711. The PCM base address for the registers is **0x7E20 3000**.

| Address Offset | Register Name            | Description                  | Size |
|----------------|--------------------------|------------------------------|------|
| 0x00           | <a href="#">CS_A</a>     | PCM Control and Status       | 32   |
| 0x04           | <a href="#">FIFO_A</a>   | PCM FIFO Data                | 32   |
| 0x08           | <a href="#">MODE_A</a>   | PCM Mode                     | 32   |
| 0x0C           | <a href="#">RXC_A</a>    | PCM Receive Configuration    | 32   |
| 0x10           | <a href="#">TXC_A</a>    | PCM Transmit Configuration   | 32   |
| 0x14           | <a href="#">DREQ_A</a>   | PCM DMA Request Level        | 32   |
| 0x18           | <a href="#">INTEN_A</a>  | PCM Interrupt Enables        | 32   |
| 0x1C           | <a href="#">INTSTC_A</a> | PCM Interrupt Status & Clear | 32   |
| 0x20           | <a href="#">GRAY</a>     | PCM Gray Mode Control        | 32   |

Table 140. PCM Register Map

### CS\_A Register

#### Synopsis

This register contains the main control and status bits for the PCM. The bottom 3 bits of this register can be written to whilst the PCM is running. The remaining bits cannot.



| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 31:25  |            | Reserved - Write as 0, read as don't care   |      |       |
| 24     | SYNC       | PCM Clock sync helper.<br>This bit provides a software synchronisation mechanism to allow the software to detect when 2 PCM clocks have occurred. It takes 2 PCM clocks before the value written to this bit will be echoed back in the read value. | RW   | 0x0   |
| 23     | RXSEX      | RX Sign Extend<br>0 = No sign extension.<br>1 = Sign extend the RX data. When set, the MSB of the received data channel (as set by the CHxWID parameter) is repeated in all the higher data bits up to the full 32-bit data width.                  | RW   | 0x0   |
| 22     | RXF        | RX FIFO is Full<br>0 = RX FIFO can accept more data.<br>1 = RX FIFO is full and will overflow if more data is received.   | RO   | 0x0   |
| 21     | TXE        | TX FIFO is Empty<br>0 = TX FIFO is not empty.<br>1 = TX FIFO is empty and underflow will take place if no more data is written.   | RO   | 0x1   |
| 20     | RXD        | Indicates that the RX FIFO contains Data<br>0 = RX FIFO is empty.<br>1 = RX FIFO contains at least 1 sample.  | RO   | 0x0   |
| 19     | TXD        | Indicates that the TX FIFO can accept Data<br>0 = TX FIFO is full and so cannot accept more data.<br>1 = TX FIFO has space for at least 1 sample.   | RO   | 0x1   |
| 18     | RXR        | Indicates that the RX FIFO needs Reading<br>0 = RX FIFO is less than RXTHR full.<br>1 = RX FIFO is RXTHR or more full.<br>This is cleared by reading sufficient data from the RX FIFO.  | RO   | 0x0   |
| 17     | TXW        | Indicates that the TX FIFO needs Writing<br>0 = TX FIFO is at least TXTHR full.<br>1 = TX FIFO is less than TXTHR full.<br>This is cleared by writing sufficient data to the TX FIFO.   | RO   | 0x1   |
| 16     | RXERR      | RX FIFO Error<br>0 = FIFO has had no errors.<br>1 = FIFO has had an under or overflow error.<br>This flag is cleared by writing a 1.  | W1C  | 0x0   |



| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 15     | TXERR      | TX FIFO Error<br>0 = FIFO has had no errors.<br>1 = FIFO has had an under or overflow error.<br>This flag is cleared by writing a 1.  | W1C  | 0x0   |
| 14     | RXSYNC     | RX FIFO Sync<br>0 = FIFO is out of sync. The amount of data left in the FIFO is not a multiple of that required for a frame. This takes into account if we are halfway through the frame.<br>1 = FIFO is in sync.   | RO   | 0x0   |
| 13     | TXSYNC     | TX FIFO Sync<br>0 = FIFO is out of sync. The amount of data left in the FIFO is not a multiple of that required for a frame. This takes into account if we are halfway through the frame.<br>1 = FIFO is in sync.   | RO   | 0x0   |
| 12:10  |            | Reserved - Write as 0, read as don't care   |      |       |
| 9      | DMAEN      | DMA DREQ Enable<br>0 = Don't generate DMA DREQ requests.<br>1 = Generates a TX DMA DREQ request whenever the TX FIFO level is lower than TX_REQ or generates a RX DMA DREQ when the RX FIFO level is higher than RX_REQ.                                      | RW   | 0x0   |
| 8:7    | RXTHR      | Sets the RX FIFO threshold at which point the RXR flag is set<br>00 = set when we have a single sample in the RX FIFO<br>01 = set when the RX FIFO is at least ¼ full<br>10 = set when the RX FIFO is at least ¾ full<br>11 = set when the RX FIFO is full    | RW   | 0x0   |
| 6:5    | TXTHR      | Sets the TX FIFO threshold at which point the TXW flag is set<br>00 = set when the TX FIFO is empty<br>01 = set when the TX FIFO is less than ¼ full<br>10 = set when the TX FIFO is less than ¾ full<br>11 = set when the TX FIFO is full but for one sample | RW   | 0x0   |
| 4      | RXCLR      | Clear the RX FIFO.<br>Assert to clear RX FIFO. This bit is self clearing and is always read as clear<br>Note that it will take 2 PCM clocks for the FIFO to be physically cleared.  | W1SC | 0x0   |



| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 3      | TXCLR      | Clear the TX FIFO<br>Assert to clear TX FIFO. This bit is self clearing and is always read as clear.<br>Note that it will take 2 PCM clocks for the FIFO to be physically cleared.  | W1SC | 0x0   |
| 2      | TXON       | Enable transmission<br>0 = Stop transmission. This will stop immediately if possible or else at the end of the next frame. The TX FIFO can still be written to, to preload data.<br>1 = Start transmission. This will start transmitting at the start of the next frame. Once enabled, the first data read from the TX FIFO will be placed in the first channel of the frame, thus ensuring proper channel synchronisation.<br>The frame counter will be started whenever TXON or RXON are set.<br>This bit can be written whilst the interface is running. | RW   | 0x0   |
| 1      | RXON       | Enable reception.<br>0 = Disable reception. This will stop on the next available frame end. RX FIFO data can still be read.<br>1 = Enable reception. This will start receiving at the start of the next frame. The first channel to be received will be the first word written to the RX FIFO.<br>This bit can be written whilst the interface is running.  | RW   | 0x0   |
| 0      | EN         | Enable the PCM Audio Interface<br>0 = The PCM interface is disabled and most logic is gated off to save power.<br>1 = The PCM Interface is enabled.<br>This bit can be written whilst the interface is running.   | RW   | 0x0   |

Table 141. CS\_A Register

## FIFO\_A Register

### Synopsis

This is the FIFO port of the PCM. Data written here is transmitted, and received data is read from here.

| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 31:0   | FIFO       | Data written here is transmitted, and received data is read from here. | RW   | 0x0   |

Table 142. FIFO\_A Register





## MODE\_A Register

### Synopsis

This register defines the basic PCM Operating Mode. It is used to configure the frame size and format and whether the PCM is in master or slave modes for its frame sync or clock. This register cannot be changed whilst the PCM is running.

| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 31:29  |            | Reserved - Write as 0, read as don't care  |      |       |
| 28     | CLK_DIS    | PCM Clock Disable<br>1 = Disable the PCM Clock.<br>This cleanly disables the PCM clock. This enables glitch free clock switching between an internal and an uncontrollable external clock. The PCM clock can be disabled, and then the clock source switched, and then the clock re-enabled.<br>0 = Enable the PCM clock.  | RW   | 0x0   |
| 27     | PDMN       | PDM Decimation Factor (N)<br>0 = Decimation factor 16.<br>1 = Decimation factor 32.<br>Sets the decimation factor of the CIC decimation filter.  | RW   | 0x0   |
| 26     | PDME       | PDM Input Mode Enable<br>0 = Disable PDM (classic PCM input).<br>1 = Enable PDM input filter.<br>Enable CIC filter on input pin for PDM inputs. In order to receive data RXON must also be set.  | RW   | 0x0   |
| 25     | FRXP       | Receive Frame Packed Mode<br>0 = The data from each channel is written into the RX FIFO.<br>1 = The data from both RX channels is merged (1st channel is in the LS half) and then written to the RX FIFO as a single 2x16-bit packed mode word.<br>First received channel in the frame goes into the LS half word. If the received data is larger than 16 bits, the upper bits are truncated. The maximum channel size is 16 bits. | RW   | 0x0   |
| 24     | FTXP       | Transmit Frame Packed Mode<br>0 = Each TX FIFO word is written into a single channel.<br>1 = Each TX FIFO word is split into 2 16-bit words and used to fill both data channels in the same frame. The maximum channel size is 16 bits.<br>The LS half of the word is used in the first channel of the frame.  | RW   | 0x0   |





| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 23     | CLKM       | PCM Clock Mode<br>0 = Master mode. The PCM CLK is an output and drives at the MCLK rate.<br>1 = Slave mode. The PCM CLK is an input.   | RW   | 0x0   |
| 22     | CLKI       | Clock Invert<br>This logically inverts the PCM_CLK signal.<br>0 = Outputs change on rising edge of clock, inputs are sampled on falling edge.<br>1 = Outputs change on falling edge of clock, inputs are sampled on rising edge.   | RW   | 0x0   |
| 21     | FSM        | Frame Sync Mode<br>0 = Master mode. The PCM_FS is an output and we generate the frame sync.<br>1 = Slave mode. The PCM_FS is an input and we lock onto the incoming frame sync signal.   | RW   | 0x0   |
| 20     | FSI        | Frame Sync Invert<br>This logically inverts the frame sync signal.<br>0 = In master mode, FS is normally low and goes high to indicate frame sync. In slave mode, the frame starts with the clock where FS is a 1 after being a 0.<br>1 = In master mode, FS is normally high and goes low to indicate frame sync. In slave mode, the frame starts with the clock where FS is a 0 after being a 1. | RW   | 0x0   |
| 19:10  | FLEN       | Frame Length<br>Sets the frame length to (FLEN+1) clocks.<br>1 = frame length of 2 clocks.<br>2 = frame length of 3 clocks.<br>etc.  | RW   | 0x0   |
| 9:0    | FSLEN      | Frame Sync Length<br>Sets the frame sync length to (FSLEN) clocks. This is only used when FSM = 0.<br>PCM_FS will remain permanently active if FSLEN >= FLEN.<br>0 = frame sync pulse is off.<br>1 = frame sync pulse is 1 clock wide.<br>etc.   | RW   | 0x0   |

Table 143. MODE\_A Register

## RXC\_A Register

### Synopsis

Sets the Channel configurations for Receiving. This sets the position and width of the 2 receive



channels within the frame. The two channels cannot overlap, however channel 2 can come after channel 1, although the first data will always be from the first channel in the frame. Channels can also straddle the frame begin-end boundary (as set by the frame sync position). This register cannot be changed whilst the PCM is running.

| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 31     | CH1WEX     | Channel 1 Width Extension Bit<br>This is the MSB of the channel 1 width (CH1WID). It allows widths greater than 24 bits to be programmed and is added here to keep backwards compatibility with older versions of the PCM  | RW   | 0x0   |
| 30     | CH1EN      | Channel 1 Enable<br>0 = Channel 1 disabled and no data is received from channel 1 and written to the RX FIFO.<br>1 = Channel 1 enabled.  | RW   | 0x0   |
| 29:20  | CH1POS     | Channel 1 Position<br>This sets the bit clock at which the first bit (MS bit) of channel 1 data occurs in the frame.<br>0 indicates the first clock of frame.  | RW   | 0x0   |
| 19:16  | CH1WID     | Channel 1 Width<br>This sets the width of channel 1 in bit clocks. This field has been extended with the CH1WEX bit giving a total width of $(CH1WEX * 16) + CH1WID + 8$ . The maximum supported width is 32 bits.<br>0 = 8 bits wide<br>1 = 9 bits wide<br>etc. | RW   | 0x0   |
| 15     | CH2WEX     | Channel 2 Width Extension Bit<br>This is the MSB of the channel 2 width (CH2WID). It allows widths greater than 24 bits to be programmed and is added here to keep backwards compatibility with older versions of the PCM  | RW   | 0x0   |
| 14     | CH2EN      | Channel 2 Enable<br>0 = Channel 2 disabled and no data is received from channel 2 and written to the RX FIFO.<br>1 = Channel 2 enabled.  | RW   | 0x0   |
| 13:4   | CH2POS     | Channel 2 Position<br>This sets the bit clock at which the first bit (MS bit) of channel 2 data occurs in the frame.<br>0 indicates the first clock of frame.  | RW   | 0x0   |



| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 3:0    | CH2WID     | Channel 2 Width<br>This sets the width of channel 2 in bit clocks. This field has been extended with the CH2WEX bit giving a total width of $(CH2WEX * 16) + CH2WID + 8$ . The maximum supported width is 32 bits.<br>0 = 8 bits wide<br>1 = 9 bits wide<br>etc. | RW   | 0x0   |

Table 144. *RXC\_A Register*

## TXC\_A Register

### Synopsis

Sets the Channel configurations for Transmitting. This sets the position and width of the 2 transmit channels within the frame. The two channels cannot overlap, however channel 2 can come after channel 1, although the first data will always be used in the first channel in the frame. Channels can also straddle the frame begin-end boundary (as set by the frame sync position). This register cannot be changed whilst the PCM is running.

| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 31     | CH1WEX     | Channel 1 Width Extension Bit<br>This is the MSB of the channel 1 width (CH1WID). It allows widths greater than 24 bits to be programmed and is added here to keep backwards compatibility with older versions of the PCM  | RW   | 0x0   |
| 30     | CH1EN      | Channel 1 Enable<br>0 = Channel 1 disabled and no data is taken from the TX FIFO and transmitted on channel 1.<br>1 = Channel 1 enabled.   | RW   | 0x0   |
| 29:20  | CH1POS     | Channel 1 Position<br>This sets the bit clock at which the first bit (MS bit) of channel 1 data occurs in the frame.<br>0 indicates the first clock of frame.  | RW   | 0x0   |
| 19:16  | CH1WID     | Channel 1 Width<br>This sets the width of channel 1 in bit clocks. This field has been extended with the CH1WEX bit giving a total width of $(CH1WEX * 16) + CH1WID + 8$ . The maximum supported width is 32 bits.<br>0 = 8 bits wide<br>1 = 9 bits wide<br>etc. | RW   | 0x0   |



| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 15     | CH2WEX     | Channel 2 Width Extension Bit<br>This is the MSB of the channel 2 width (CH2WID). It allows widths greater than 24 bits to be programmed and is added here to keep backwards compatibility with older versions of the PCM  | RW   | 0x0   |
| 14     | CH2EN      | Channel 2 Enable<br>0 = Channel 2 disabled and no data is taken from the TX FIFO and transmitted on channel 2.<br>1 = Channel 2 enabled.   | RW   | 0x0   |
| 13:4   | CH2POS     | Channel 2 Position<br>This sets the bit clock at which the first bit (MS bit) of channel 2 data occurs in the frame.<br>0 indicates the first clock of frame.  | RW   | 0x0   |
| 3:0    | CH2WID     | Channel 2 Width<br>This sets the width of channel 2 in bit clocks. This field has been extended with the CH2WEX bit giving a total width of $(CH2WEX * 16) + CH2WID + 8$ . The maximum supported width is 32 bits.<br>0 = 8 bits wide<br>1 = 9 bits wide<br>etc. | RW   | 0x0   |

Table 145. TXC\_A Register

## DREQ\_A Register

### Synopsis

Set the DMA DREQ and Panic thresholds. The PCM drives 2 DMA controls back to the DMA, one for the TX channel and one for the RX channel. DMA DREQ is used to request the DMA to perform another transfer, and DMA Panic is used to tell the DMA to use its panic level of priority when requesting things on the AXI bus. This register cannot be changed whilst the PCM is running.

| Bit(s) | Field Name                                | Description  | Type | Reset |
|--------|---|--|------|-------|
| 31     | Reserved - Write as 0, read as don't care |  |      |       |
| 30:24  | TX_PANIC                                  | TX Panic Level<br>This sets the TX FIFO Panic level. When the level is below this the PCM will assert its TX DMA Panic signal. | RW   | 0x10  |
| 23     | Reserved - Write as 0, read as don't care |  |      |       |
| 22:16  | RX_PANIC                                  | RX Panic Level<br>This sets the RX FIFO Panic level. When the level is above this the PCM will assert its RX DMA Panic signal. | RW   | 0x30  |
| 15     | Reserved - Write as 0, read as don't care |  |      |       |



| Bit(s) | Field Name                                | Description   | Type | Reset |
|--------|---|---|------|-------|
| 14:8   | TX_REQ                                    | TX Request Level<br>This sets the TX FIFO DREQ level. When the level is below this the PCM will assert its DMA DREQ signal to request that more data is written to the TX FIFO.       | RW   | 0x30  |
| 7      | Reserved - Write as 0, read as don't care |   |      |       |
| 6:0    | RX_REQ                                    | RX Request Level<br>This sets the RX FIFO DREQ level. When the level is above this the PCM will assert its DMA DREQ signal to request that some more data is read out of the RX FIFO. | RW   | 0x20  |

Table 146. DREQ\_A Register

## INTEN\_A Register

### Synopsis

Set the reasons for generating an Interrupt. This register cannot be changed whilst the PCM is running.

| Bit(s) | Field Name                                | Description  | Type | Reset |
|--------|---|--|------|-------|
| 31:4   | Reserved - Write as 0, read as don't care |  |      |       |
| 3      | RXERR                                     | RX Error Interrupt<br>Setting this bit enables interrupts from PCM block when RX FIFO error occurs.  | RW   | 0x0   |
| 2      | TXERR                                     | TX Error Interrupt<br>Setting this bit enables interrupts from PCM block when TX FIFO error occurs.  | RW   | 0x0   |
| 1      | RXR                                       | RX Read Interrupt Enable<br>Setting this bit enables interrupts from PCM block when RX FIFO level is greater than or equal to the specified RXTHR level. | RW   | 0x0   |
| 0      | TXW                                       | TX Write Interrupt Enable<br>Setting this bit enables interrupts from PCM block when TX FIFO level is less than the specified TXTHR level.               | RW   | 0x0   |

Table 147. INTEN\_A Register

## INTSTC\_A Register

### Synopsis

This register is used to read and clear the PCM interrupt status. Writing a 1 to the asserted bit clears the bit. Writing a 0 has no effect.



| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 31:4   |            | Reserved - Write as 0, read as don't care  |      |       |
| 3      | RXERR      | RX Error Interrupt Status / Clear<br>This bit indicates an interrupt occurred on RX FIFO Error.<br>Writing 1 to this bit clears it. Writing 0 has no effect. | W1C  | 0x0   |
| 2      | TXERR      | TX Error Interrupt Status / Clear<br>This bit indicates an interrupt occurred on TX FIFO Error.<br>Writing 1 to this bit clears it. Writing 0 has no effect. | W1C  | 0x0   |
| 1      | RXR        | RX Read Interrupt Status / Clear<br>This bit indicates an interrupt occurred on RX Read.<br>Writing 1 to this bit clears it. Writing 0 has no effect.        | W1C  | 0x0   |
| 0      | TXW        | TX Write Interrupt Status / Clear<br>This bit indicates an interrupt occurred on TX Write.<br>Writing 1 to this bit clears it. Writing 0 has no effect.      | W1C  | 0x0   |

Table 148. INTSTC\_A Register

## GRAY Register

### Synopsis

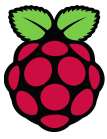
This register is used to control the gray mode generation. This is used to put the PCM into a special data/strobe mode. This mode is under 'best effort' contract.

| Bit(s) | Field Name  | Description   | Type | Reset |
|--------|-------------|---|------|-------|
| 31:22  |             | Reserved - Write as 0, read as don't care   |      |       |
| 21:16  | RXFIFOLEVEL | The current level of the RX FIFO<br>This indicates how many words are currently in the RX FIFO.   | RO   | 0x0   |
| 15:10  | FLUSHED     | The number of bits that were flushed into the RX FIFO<br>This indicates how many bits were valid when the flush operation was performed. The valid bits are from bit 0 upwards. Non-valid bits are set to zero. | RO   | 0x0   |
| 9:4    | RXLEVEL     | The current fill level of the RX Buffer<br>This indicates how many GRAY coded bits have been received. When 32 bits are received, they are written out into the RX FIFO.  | RO   | 0x0   |
| 3      |             | Reserved - Write as 0, read as don't care   |      |       |



| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 2      | FLUSH      | Flush the RX Buffer into the RX FIFO<br>This forces the RX Buffer to do an early write. This is necessary if we have reached the end of the message and we have bits left in the RX Buffer. Flushing will write these bits as a single 32-bit word, starting at bit zero. Empty bits will be packed with zeros. The number of bits written will be recorded in the FLUSHED field. This bit is written as a 1 to initiate a flush. It will read back as a zero until the flush operation has completed (as the PCM Clock may be very slow). | RW   | 0x0   |
| 1      | CLR        | Clear the GRAY Mode Logic<br>This bit will reset all the GRAY mode logic, and flush the RX buffer. It is not self clearing.  | RW   | 0x0   |
| 0      | EN         | Enable GRAY Mode<br>Setting this bit will put the PCM into GRAY mode. In gray mode the data is received on the data in and the frame sync pins. The data is expected to be in data/strobe format.  | RW   | 0x0   |

*Table 149. GRAY Register*



## Chapter 8. Pulse Width Modulator

### 8.1. Overview

This section specifies in detail the functionality provided by the device Pulse Width Modulator (PWM) peripherals.

Each PWM controller incorporates the following features:

- Two independent output bit-streams, clocked at a fixed frequency
- Bit-streams configured individually to output either PWM or a serialised version of a 32-bit word
- PWM outputs have variable output resolutions
- Serialise mode configured to read data from a FIFO storage block, which can store up to sixty-four 32-bit words
- Both modes **clocked by clk\_pwm** which is nominally 100MHz, but can be varied by the clock manager

### 8.2. Block Diagram

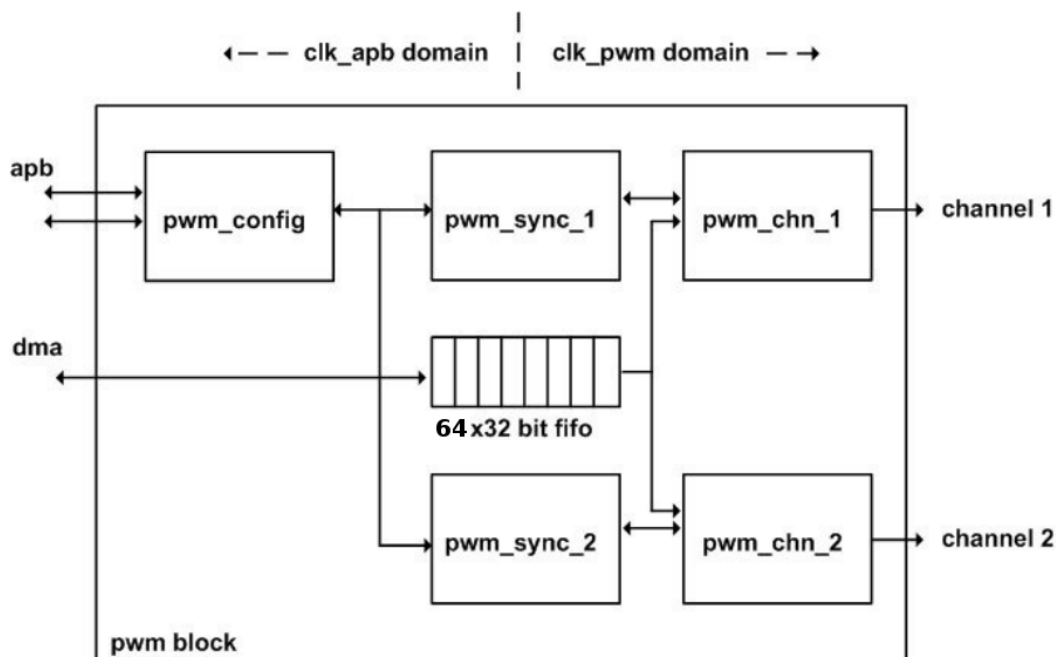


Figure 14. PWM block diagram

The BCM2711 device has two instances of this block, named PWM0 and PWM1 (each with two output channels).

### 8.3. PWM Implementation

A value represented as a ratio of N/M can be transmitted along a serial channel with pulse width





modulation, in which the value is represented by the duty cycle of the output signal. To send value  $N/M$  within a periodic sequence of  $M$  cycles, output should be 1 for  $N$  cycles and 0 for  $(M-N)$  cycles. The desired sequence should have 1s and 0s spread out as evenly as possible, so that during any arbitrary period of time the duty cycle achieves the closest approximation of the value. This can be shown in the following table where  $4/8$  is modulated ( $N=4$ ,  $M=8$ ).

|      |   |   |   |   |   |   |   |   |   |   |   |   |
|------|---|---|---|---|---|---|---|---|---|---|---|---|
| Bad  | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| Fair | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| Good | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

Sequence which gives the ‘good’ approximation from the table above can be achieved by the following algorithm:

```
1. Set context = 0
2. context = context + N
3. if (context >= M)
    context = context - M
    send 1
   else
    send 0
4. Repeat from step 2
```

where **context** is a register which stores the result of the additions/subtractions.

## 8.4. Modes of Operation

The PWM controller consists of two independent channels (`pwm_chn_i` in [Figure 14](#)) which implement the PWM algorithm explained in the previous section. Each channel can operate in either PWM mode or serialiser mode.

**PWM mode:** There are two sub-modes in PWM mode:  $MSEN=0$  and  $MSEN=1$ .

When  $MSEN=0$  (which is the default mode), data to be sent is interpreted as the value  $N$  of the algorithm explained above. The number of clock cycles (range) used to send data is the value  $M$  of the algorithm. Pulses are sent within this range so that the resulting duty cycle is  $N/M$ . The channel sends its output continuously as long as the data register is used ( $USEFi=0$ ), or the FIFO is used and it is not empty.

When  $MSEN=1$ , the PWM channel does not use the algorithm explained above, instead it sends serial data with the  $M/S$  ratio as in [Figure 15](#).  $M$  is the data to be sent, and  $S$  is the range. This mode may be preferred if high frequency modulation is not required or has negative effects. The channel sends its output continuously as long as the data register is used ( $USEFi=0$ ), or the FIFO is used and it is not empty.

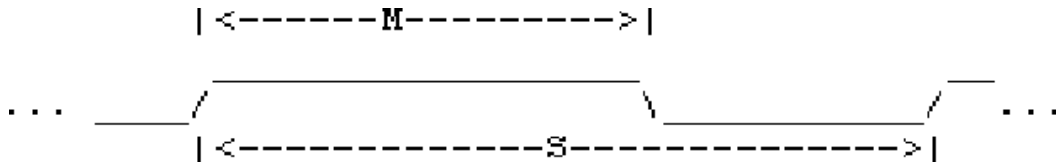


Figure 15. Serial bit transmission when M/S Mode enabled

**Serialiser mode:** Each channel is also capable of working as a serialiser. In this mode data written in the FIFO or the data register is sent serially.

## 8.5. Quick Reference

- PWM0 DMA is mapped to DMA channel 5.
- PWM1 DMA is mapped to DMA channel 1 (muxed with DSI0).
- GPIOs are assigned to PWM channels as below. Please refer to the [GPIO chapter](#) for further details:

| GPIO   | ALT0   | ALT1 | ALT2 | ALT3 | ALT4 | ALT5   |
|--------|--------|------|------|------|------|--------|
| GPIO12 | PWM0_0 |      |      |      |      |        |
| GPIO13 | PWM0_1 |      |      |      |      |        |
| GPIO18 |        |      |      |      |      | PWM0_0 |
| GPIO19 |        |      |      |      |      | PWM0_1 |
| GPIO40 | PWM1_0 |      |      |      |      |        |
| GPIO41 | PWM1_1 |      |      |      |      |        |
| GPIO45 | PWM0_1 |      |      |      |      |        |

- PWM clock source and frequency is controlled in CPRMAN.

## 8.6. Control and Status Registers

The PWM0 register base address is **0x7E20 C000** and the PWM1 register base address is **0x7E20 C800**.

| Address Offset | Register Name        | Description           | Size |
|----------------|----------------------|-----------------------|------|
| 0x00           | <a href="#">CTL</a>  | PWM Control           | 32   |
| 0x04           | <a href="#">STA</a>  | PWM Status            | 32   |
| 0x08           | <a href="#">DMAC</a> | PWM DMA Configuration | 32   |
| 0x10           | <a href="#">RNG1</a> | PWM Channel 1 Range   | 32   |
| 0x14           | <a href="#">DAT1</a> | PWM Channel 1 Data    | 32   |
| 0x18           | <a href="#">FIF1</a> | PWM FIFO Input        | 32   |
| 0x20           | <a href="#">RNG2</a> | PWM Channel 2 Range   | 32   |



| Address Offset | Register Name        | Description        | Size |
|----------------|----------------------|--------------------|------|
| 0x24           | <a href="#">DAT2</a> | PWM Channel 2 Data | 32   |

Table 150. PWM Register Map

## CTL Register

### Synopsis

PWEN<sub>i</sub> is used to enable/disable the corresponding channel. Setting this bit to 1 enables the channel and transmitter state machine. All registers and FIFOs are writeable without setting this bit.

MODE<sub>i</sub> bit is used to determine mode of operation. Setting this bit to 0 (the default) enables PWM mode. In this mode data stored in either PWM\_DAT<sub>i</sub> or FIFO is transmitted by pulse width modulation within the range defined by PWM\_RNG<sub>i</sub>. When this mode is used, MSEN<sub>i</sub> defines whether to use PWM algorithm or M/S transmission. Setting MODE<sub>i</sub> to 1 enables serial mode, in which data stored in either PWM\_DAT<sub>i</sub> or FIFO is transmitted serially within the range defined by PWM\_RNG<sub>i</sub>. Data is transmitted MSB first and truncated or zero-padded depending on PWM\_RNG<sub>i</sub>.

RPTL<sub>i</sub> is used to enable/disable repeating of the last data available in the FIFO just before it empties. When this bit is 1 and FIFO is used, the last available data in the FIFO is repeatedly sent. This may be useful in PWM mode to avoid duty cycle gaps. If the FIFO is not used this bit does not have any effect. Default operation is do-not-repeat.

SBIT<sub>i</sub> defines the state of the output when no transmission takes place. It also defines the zero polarity for the zero padding in serialiser mode. This bit is padded between two consecutive transfers as well as tail of the data when PWM\_RNG<sub>i</sub> is larger than bit depth of data being transferred. This bit is zero by default.

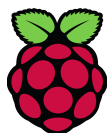
POLA<sub>i</sub> is used to configure the polarity of the output bit. When set to high the final output is inverted. Default operation is no inversion.

USEF<sub>i</sub> bit is used to enable/disable FIFO transfer. When this bit is high data stored in the FIFO is used for transmission. When it is low, data written to PWM\_DAT<sub>i</sub> is transferred. This bit is 0 by default.

CLRF is used to clear the FIFO. Writing a 1 to this bit clears the FIFO. Writing 0 has no effect. This is a one-shot operation and reading the bit always returns 0.

MSEN<sub>i</sub> is used to determine whether to use PWM algorithm or simple M/S ratio transmission. When this bit is high M/S transmission is used. This bit is zero by default. When MODE<sub>i</sub> is 1, this configuration bit has no effect.

| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 31:16  |            | Reserved - Write as 0, read as don't care   |      |       |
| 15     | MSEN2      | Channel 2 M/S Enable<br>0: PWM algorithm is used<br>1: M/S transmission is used.          | RW   | 0x0   |
| 14     |            | Reserved - Write as 0, read as don't care   |      |       |
| 13     | USEF2      | Channel 2 Use FIFO<br>0: Data register is transmitted<br>1: FIFO is used for transmission | RW   | 0x0   |



| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 12     | POLA2      | Channel 2 Polarity<br>0 : 0=low 1=high<br>1: 1=low 0=high   | RW   | 0x0   |
| 11     | SBIT2      | Channel 2 Silence Bit<br>Defines the state of the output when no transmission takes place   | RW   | 0x0   |
| 10     | RPTL2      | Channel 2 Repeat Last Data<br>0: Transmission interrupts when FIFO is empty<br>1: Last data in FIFO is transmitted repeatedly until FIFO is not empty | RW   | 0x0   |
| 9      | MODE2      | Channel 2 Mode<br>0: PWM mode<br>1: Serialiser mode   | RW   | 0x0   |
| 8      | PWEN2      | Channel 2 Enable<br>0: Channel is disabled<br>1: Channel is enabled   | RW   | 0x0   |
| 7      | MSEN1      | Channel 1 M/S Enable<br>0: PWM algorithm is used<br>1: M/S transmission is used.  | RW   | 0x0   |
| 6      | CLRF       | Clear FIFO<br>1: Clears FIFO<br>0: Has no effect<br>This is a one-shot operation. This bit always reads 0   | W1SC | 0x0   |
| 5      | USEF1      | Channel 1 Use FIFO<br>0: Data register is transmitted<br>1: FIFO is used for transmission   | RW   | 0x0   |
| 4      | POLA1      | Channel 1 Polarity<br>0 : 0=low 1=high<br>1: 1=low 0=high   | RW   | 0x0   |
| 3      | SBIT1      | Channel 1 Silence Bit<br>Defines the state of the output when no transmission takes place   | RW   | 0x0   |
| 2      | RPTL1      | Channel 1 Repeat Last Data<br>0: Transmission interrupts when FIFO is empty<br>1: Last data in FIFO is transmitted repeatedly until FIFO is not empty | RW   | 0x0   |
| 1      | MODE1      | Channel 1 Mode<br>0: PWM mode<br>1: Serialiser mode   | RW   | 0x0   |



| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 0      | PWEN1      | Channel 1 Enable<br>0: Channel is disabled<br>1: Channel is enabled | RW   | 0x0   |

Table 151. CTL Register

## STA Register

### Synopsis

FULL1 bit indicates the full status of the FIFO. If this bit is high the FIFO is full.

EMPT1 bit indicates the empty status of the FIFO. If this bit is high the FIFO is empty.

WERR1 bit is set to high when a write-when-full error occurs. Software must clear this bit by writing 1. Writing 0 to this bit has no effect.

RERR1 bit is set to high when a read-when-empty error occurs. Software must clear this bit by writing 1. Writing 0 to this bit has no effect.

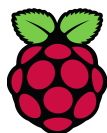
GAPOi bit indicates that there has been a gap between transmission of two consecutive data from FIFO. This may happen when the FIFO becomes empty after the state machine has sent a word and is waiting for the next word. If control bit RPTLi is set to high this event will not occur. Software must clear this bit by writing 1. Writing 0 to this bit has no effect.

BERR is set to high when an error has occurred while writing to registers via APB. This may happen if the bus tries to write successively to same set of registers faster than the synchroniser block can cope with. Multiple switching may occur and contaminate the data during synchronisation. Software should clear this bit by writing 1. Writing 0 to this bit has no effect.

STAi bit indicates the current state of the channel, which is useful for debugging purposes. 0 means the channel is not currently transmitting, 1 means channel is transmitting data.

| Bit(s) | Field Name | Description                               | Type | Reset |
|--------|------------|---|------|-------|
| 31:11  |            | Reserved - Write as 0, read as don't care |      |       |
| 10     | STA2       | Channel 2 State                           | RO   | 0x0   |
| 9      | STA1       | Channel 1 State                           | RO   | 0x0   |
| 8      | BERR       | Bus Error Flag                            | W1C  | 0x0   |
| 7:6    |            | Reserved - Write as 0, read as don't care |      |       |
| 5      | GAP02      | Channel 2 Gap Occurred Flag               | W1C  | 0x0   |
| 4      | GAP01      | Channel 1 Gap Occurred Flag               | W1C  | 0x0   |
| 3      | RERR1      | FIFO Read Error Flag                      | W1C  | 0x0   |
| 2      | WERR1      | FIFO Write Error Flag                     | W1C  | 0x0   |
| 1      | EMPT1      | FIFO Empty Flag                           | RO   | 0x1   |
| 0      | FULL1      | FIFO Full Flag                            | RO   | 0x0   |

Table 152. STA Register



## DMAC Register

### Synopsis

ENAB bit is used to start DMA.

PANIC bits are used to determine the threshold level for PANIC signal going active. Default value is 7.

DREQ bits are used to determine the threshold level for DREQ signal going active. Default value is 7.

| Bit(s) | Field Name                                | Description                                     | Type | Reset |
|--------|---|---|------|-------|
| 31     | ENAB                                      | DMA Enable<br>0: DMA disabled<br>1: DMA enabled | RW   | 0x0   |
| 30:16  | Reserved - Write as 0, read as don't care |   |      |       |
| 15:8   | PANIC                                     | DMA Threshold for PANIC signal                  | RW   | 0x7   |
| 7:0    | DREQ                                      | DMA Threshold for DREQ signal                   | RW   | 0x7   |

Table 153. DMAC Register

## RNG1 & RNG2 Registers

### Synopsis

This register is used to define the range for the corresponding channel. In PWM mode, evenly distributed pulses are sent within a period of length defined by this register. In serial mode, serialised data is transmitted within the same period. If the value in PWM\_RNG $i$  is less than 32, only the first PWM\_RNG $i$  bits are sent resulting in a truncation. If it is larger than 32, excess zero bits are padded at the end of data. Default value for this register is 32.

| Bit(s) | Field Name  | Description       | Type | Reset |
|--------|-------------|-------------------|------|-------|
| 31:0   | PWM_RNG $i$ | Channel $i$ Range | RW   | 0x20  |

Table 154. RNG1 & RNG2 Registers

## DAT1 & DAT2 Registers

### Synopsis

This register stores the 32-bit data to be sent by the PWM Controller when USEF $i$  is 0. In PWM mode, data is sent by pulse width modulation: the value of this register defines the number of pulses which are sent within the period defined by PWM\_RNG $i$ . In serialiser mode, data stored in this register is serialised and transmitted.

| Bit(s) | Field Name  | Description      | Type | Reset |
|--------|-------------|------------------|------|-------|
| 31:0   | PWM_DAT $i$ | Channel $i$ Data | RW   | 0x0   |

Table 155. DAT1 & DAT2 Registers

## FIF1 Register



## Synopsis

This register is the FIFO input for both the channels. Data written to this address is stored in the FIFO and if USEF $i$  is enabled for channel  $i$  it is used as data to be sent. This register is write-only, and reading this register will always return bus default return value, pwm0.

When more than one channel is enabled for FIFO usage, the data written into the FIFO is shared between these channels in turn. For example if the word series A B C D E F G H I .. is written to the FIFO and both channels are active and configured to use FIFO, then channel 1 will transmit words A C E G I .. and channel 2 will transmit words B D F H .. .

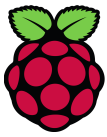
Note that requesting data from the FIFO is in locked-step manner and therefore requires tight coupling of state machines of the channels. If the channel range (period) value of one channel is different to the other, this will cause the channel with the smaller range value to wait between words, hence resulting in gaps between words. To avoid that, each channel sharing the FIFO should be configured to use the same range value.

Also note that the RPTL $i$  bits are not meaningful when the FIFO is shared between channels as there is no defined channel to own the last data in the FIFO. Therefore sharing channels must have their RPTL $i$  bit set to zero.

If the set of channels sharing the FIFO has been modified after a configuration change, the FIFO should be cleared before writing new data.

| Bit(s) | Field Name | Description        | Type | Reset |
|--------|------------|--------------------|------|-------|
| 31:0   | PWM_FIFO   | Channel FIFO Input | WO   | 0x0   |

*Table 156. FIF1 Register*



## Chapter 9. SPI

### 9.1. Overview

This serial interface peripheral supports the following features:

- Implements a 3 wire serial protocol, variously called Serial Peripheral Interface (SPI) or Synchronous Serial Protocol (SSP).
- Implements a 2 wire version of SPI that uses a single wire as a bidirectional data wire instead of one for each direction as in standard SPI.
- Implements a LoSSI Master (Low Speed Serial Interface).
- Provides support for polled, interrupt or DMA operation.

### 9.2. SPI Master Mode

#### 9.2.1. Standard mode

In standard SPI master mode the peripheral implements the standard 3 wire serial protocol described below.

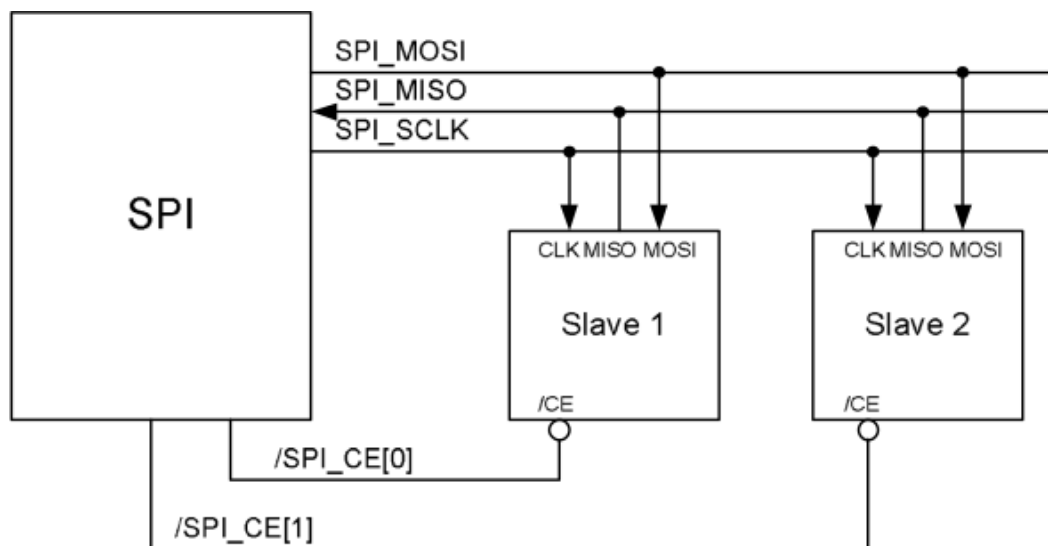
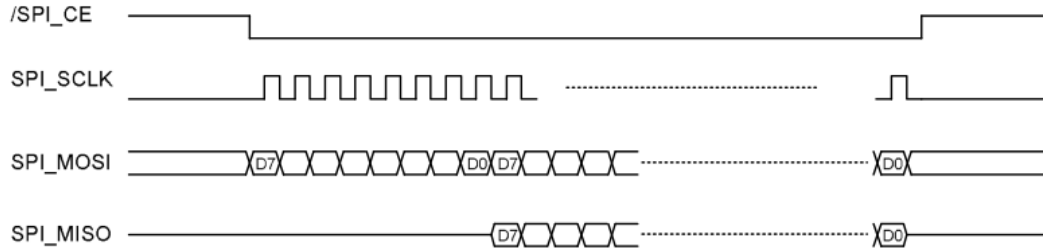


Figure 16. SPI Master Typical Usage





#### Notes

1. Slave enables itself onto SPI\_MISO only when it is outputting data. At other times, output is tristate.
2. Different SCLK polarity and phase values are possible. Case illustrated is CPOL = 0, CPHA = 0.
3. Data is transmitted MSB first.
4. Transactions can be from a single byte to hundreds of bytes.

Figure 17. SPI Cycle

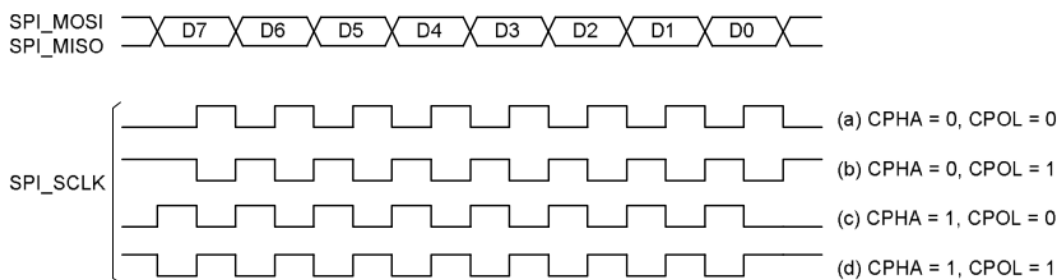


Figure 18. Different Clock Polarity/Phase

## 9.2.2. Bidirectional mode

In bidirectional SPI master mode the same SPI standard is implemented except that a single wire is used for the data (MIMO) instead of two as in standard mode (MISO and MOSI). Bidirectional mode is used in a similar way to standard mode, the only difference is that before attempting to read data from the slave, you must set the read enable (SPI\_REN) bit in the SPI control and status register (SPI\_CS). This will turn the bus around, and when you write to the SPI\_FIFO register (with junk) a read transaction will take place on the bus, and the read data will appear in the FIFO.

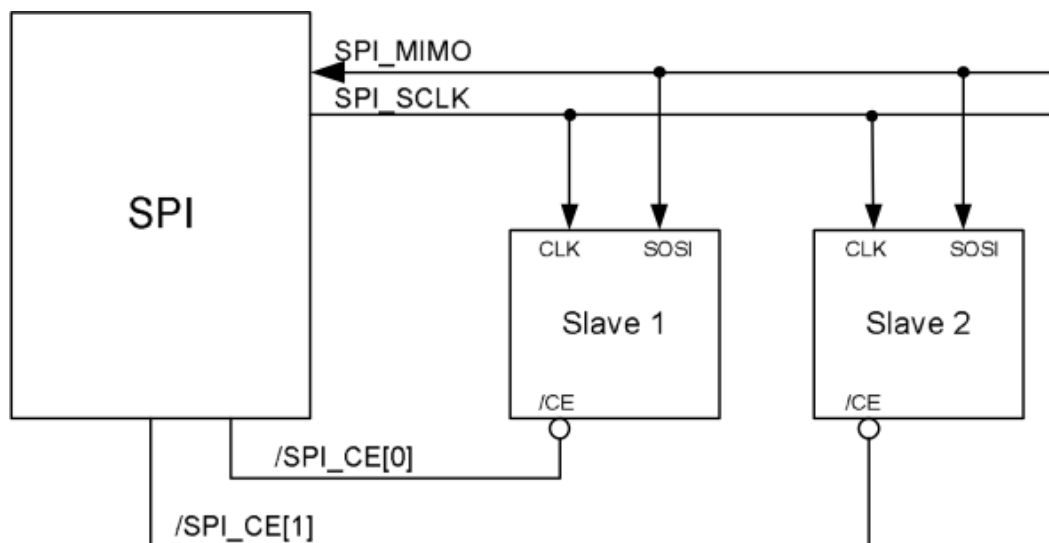


Figure 19. Bidirectional SPI Master Typical Usage



## 9.3. LoSSI mode

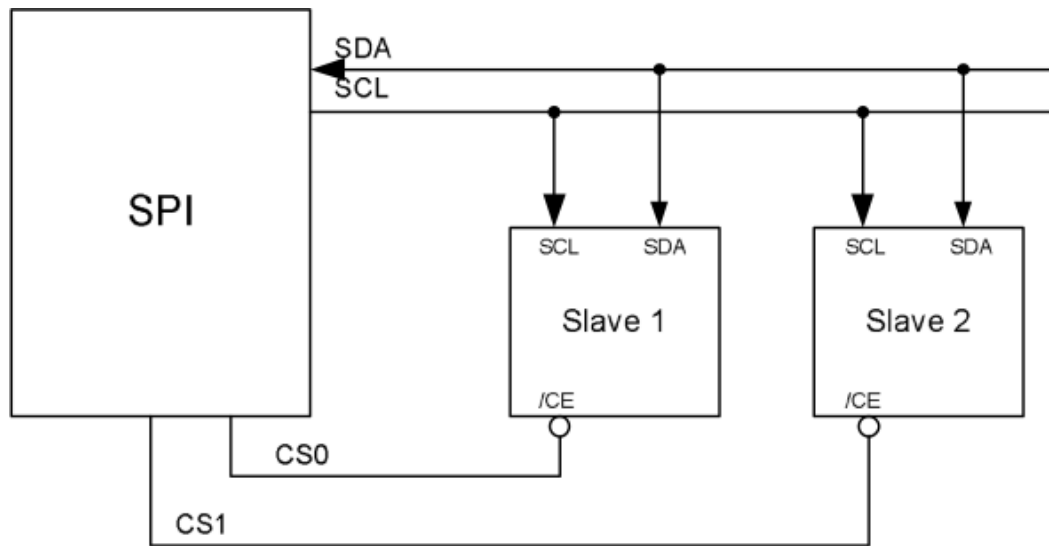
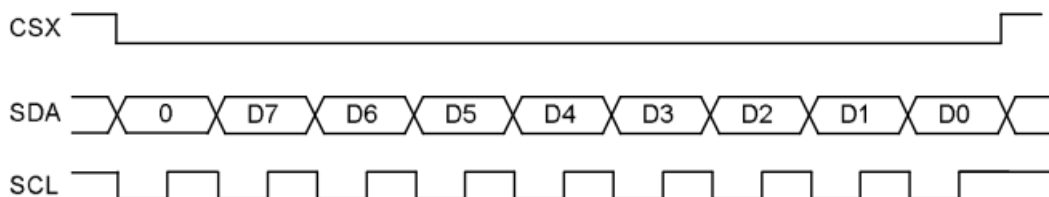


Figure 20. LoSSI mode Typical usage

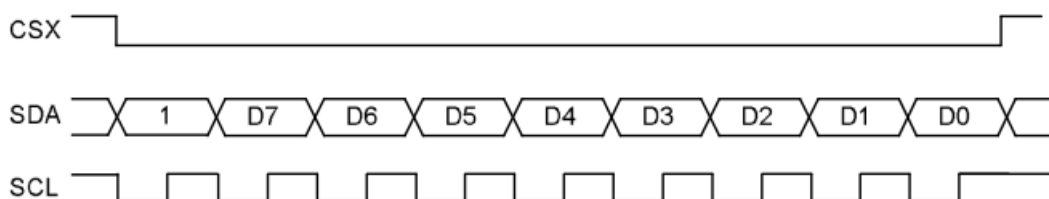
The LoSSI standard allows us to issue commands to peripherals and to transfer data to and from them. LoSSI commands and parameters are 8 bits long, but an extra bit is used to indicate whether the byte is a command or data. This extra bit is set high for a parameter and low for a command. The resulting 9-bit value is serialized to the output. When reading from a LoSSI peripheral the standard allows us to read bytes of data, as well as 24- and 32-bit words.

Commands and parameters are issued to a LoSSI peripheral by writing the 9-bit value of the command or data into the SPI\_FIFO register as you would for SPI mode. Reads are automated in that if the serial interface peripheral detects a read command being issued, it will issue the command and complete the read transaction, putting the received data into the FIFO.

### 9.3.1. Command write

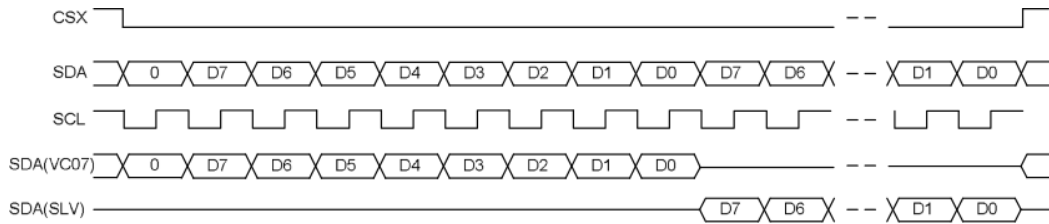


### 9.3.2. Parameter write



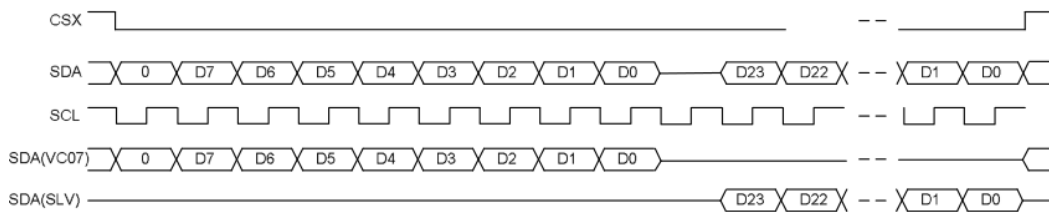
### 9.3.3. Byte read commands

Byte read commands are 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f, 0xda, 0xdb, 0xdc.



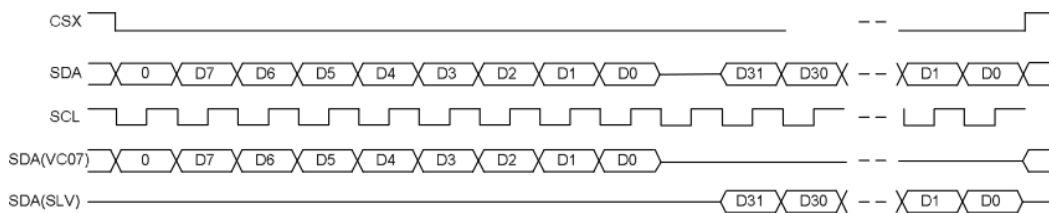
### 9.3.4. 24-bit read command

A 24-bit read can be achieved by using the command 0x04.



### 9.3.5. 32-bit read command

A 32-bit read can be achieved by using the command 0x09.



## 9.4. Block Diagram

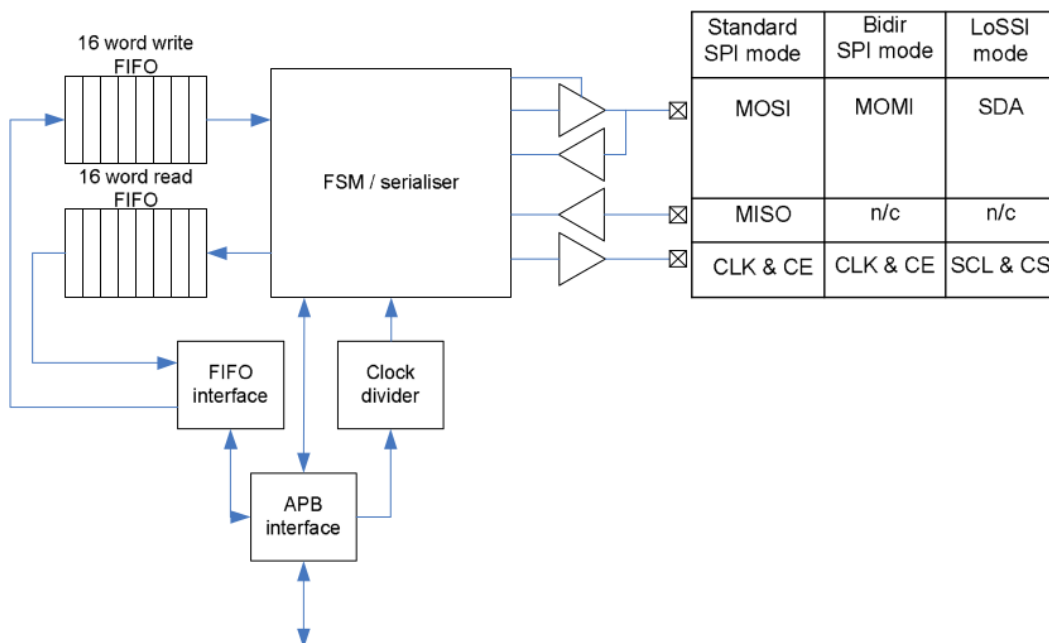


Figure 21. Serial interface Block Diagram



## 9.5. SPI Register Map

The BCM2711 device has five SPI interfaces of this type: SPI0, SPI3, SPI4, SPI5 & SPI6. It has two additional mini SPI interfaces (SPI1 and SPI2). The specification of those can be found under [Section 2.3](#).

The base addresses of these SPI interfaces are

- SPI0: **0x7E20 4000**
- SPI3: **0x7E20 4600**
- SPI4: **0x7E20 4800**
- SPI5: **0x7E20 4A00**
- SPI6: **0x7E20 4C00**

| Address Offset | Register Name        | Description                   | Size |
|----------------|----------------------|-------------------------------|------|
| 0x00           | <a href="#">CS</a>   | SPI Master Control and Status | 32   |
| 0x04           | <a href="#">FIFO</a> | SPI Master TX and RX FIFOs    | 32   |
| 0x08           | <a href="#">CLK</a>  | SPI Master Clock Divider      | 32   |
| 0x0C           | <a href="#">DLEN</a> | SPI Master Data Length        | 32   |
| 0x10           | <a href="#">LTOH</a> | SPI LoSSI mode TOH            | 32   |
| 0x14           | <a href="#">DC</a>   | SPI DMA DREQ Controls         | 32   |

Table 157. SPI Register Map

### CS Register

#### Synopsis

This register contains the main control and status bits for the SPI.

| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 31:26  |            | Reserved - Write as 0, read as don't care   |      |       |
| 25     | LEN_LONG   | Enable Long data word in LoSSI mode if DMA_LEN is set<br>0= writing to the FIFO will write a single byte<br>1= writing to the FIFO will write a 32-bit word | RW   | 0x0   |
| 24     | DMA_LEN    | Enable DMA mode in LoSSI mode   | RW   | 0x0   |
| 23     | CSPOL2     | Chip Select 2 Polarity<br>0= Chip select is active low.<br>1= Chip select is active high.   | RW   | 0x0   |



| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 22     | CSPOL1     | Chip Select 1 Polarity<br>0= Chip select is active low.<br>1= Chip select is active high.   | RW   | 0x0   |
| 21     | CSPOL0     | Chip Select 0 Polarity<br>0= Chip select is active low.<br>1= Chip select is active high.   | RW   | 0x0   |
| 20     | RXF        | RX FIFO Full<br>0 = RX FIFO is not full.<br>1 = RX FIFO is full. No further serial data will be sent / received until data is read from FIFO.   | RO   | 0x0   |
| 19     | RXR        | RX FIFO needs Reading ( $\frac{3}{4}$ full)<br>0 = RX FIFO is less than $\frac{3}{4}$ full (or not active TA = 0).<br>1 = RX FIFO is $\frac{3}{4}$ or more full. Cleared by reading sufficient data from the RX FIFO or setting TA to 0.            | RO   | 0x0   |
| 18     | TXD        | TX FIFO can accept Data<br>0 = TX FIFO is full and so cannot accept more data.<br>1 = TX FIFO has space for at least 1 byte.  | RO   | 0x1   |
| 17     | RXD        | RX FIFO contains Data<br>0 = RX FIFO is empty.<br>1 = RX FIFO contains at least 1 byte.   | RO   | 0x0   |
| 16     | DONE       | Transfer Done<br>0 = Transfer is in progress (or not active TA = 0).<br>1 = Transfer is complete. Cleared by writing more data to the TX FIFO or setting TA to 0.   | RO   | 0x0   |
| 15     | TE_EN      | Unused  | RW   | 0x0   |
| 14     | LMONO      | Unused  | RW   | 0x0   |
| 13     | LEN        | LoSSI enable<br>The serial interface is configured as a LoSSI master.<br>0 = The serial interface will behave as an SPI master.<br>1 = The serial interface will behave as a LoSSI master.  | RW   | 0x0   |
| 12     | REN        | Read Enable<br>Read enable if you are using bidirectional mode. If this bit is set, the SPI peripheral will be able to send data to this device.<br>0 = We intend to write to the SPI peripheral.<br>1 = We intend to read from the SPI peripheral. | RW   | 0x1   |



| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 11     | ADCS       | Automatically De-assert Chip Select<br>0 = Don't automatically de-assert chip select at the end of a DMA transfer; chip select is manually controlled by software.<br>1 = Automatically de-assert chip select at the end of a DMA transfer (as determined by SPIDLEN)   | RW   | 0x0   |
| 10     | INTR       | Interrupt on RXR<br>0 = Don't generate interrupts on RX FIFO condition.<br>1 = Generate interrupt while RXR = 1.  | RW   | 0x0   |
| 9      | INTD       | Interrupt on Done<br>0 = Don't generate interrupt on transfer complete.<br>1 = Generate interrupt when DONE = 1.  | RW   | 0x0   |
| 8      | DMAEN      | DMA Enable<br>0 = No DMA requests will be issued.<br>1 = Enable DMA operation.<br>Peripheral generates data requests. These will be taken in four-byte words until the SPIDLEN has been reached.  | RW   | 0x0   |
| 7      | TA         | Transfer Active<br>0 = Transfer not active. /CS lines are all high (assuming CSPOL = 0). RXR and DONE are 0. Writes to SPI_FIFO write data into bits 15:0 of SPIDLEN and bits 7:0 of SPICS allowing DMA data blocks to set mode before sending data.<br>1 = Transfer active. /CS lines are set according to CS bits and CSPOL. Writes to SPI_FIFO write data to TX FIFO.<br>TA is cleared by a dma_frame_end pulse from the DMA controller. | RW   | 0x0   |
| 6      | CSPOL      | Chip Select Polarity<br>0 = Chip select lines are active low<br>1 = Chip select lines are active high   | RW   | 0x0   |
| 5:4    | CLEAR      | FIFO Clear<br>00 = No action.<br>x1 = Clear TX FIFO. One-shot operation.<br>1x = Clear RX FIFO. One-shot operation.<br>If CLEAR and TA are both set in the same operation, the FIFOs are cleared before the new frame is started. Read back as 0.   | W1SC | 0x0   |
| 3      | CPOL       | Clock Polarity<br>0 = Rest state of clock = low.<br>1 = Rest state of clock = high.   | RW   | 0x0   |



| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 2      | CPHA       | Clock Phase<br>0 = First SCLK transition at middle of data bit.<br>1 = First SCLK transition at beginning of data bit. | RW   | 0x0   |
| 1:0    | CS         | Chip Select<br>00 = Chip select 0<br>01 = Chip select 1<br>10 = Chip select 2<br>11 = Reserved                         | RW   | 0x0   |

Table 158. CS Register

## FIFO Register

### Synopsis

This register allows TX data to be written to the TX FIFO and RX data to be read from the RX FIFO.

| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 31:0   | DATA       | DMA Mode (DMAEN set)<br>If TA is clear, the first 32-bit write to this register will control SPIDLEN and SPICS. Subsequent reads and writes will be taken as four-byte data words to be read/written to the FIFOs<br>Poll/Interrupt Mode (DMAEN clear, TA set)<br>Writes to the register write bytes to TX FIFO. Reads from register read bytes from the RX FIFO | RW   | 0x0   |

Table 159. FIFO Register

## CLK Register

### Synopsis

This register allows the SPI clock rate to be set.

| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 31:16  |            | Reserved - Write as 0, read as don't care   |      |       |
| 15:0   | CDIV       | Clock Divider<br>$SCLK = Core\ Clock / CDIV$<br>If CDIV is set to 0, the divisor is 65536. The divisor must be a multiple of 2. Odd numbers rounded down. The maximum SPI clock rate is of the APB clock. | RW   | 0x0   |

Table 160. CLK Register

## DLEN Register



## Synopsis

This register allows the SPI data length rate to be set.

| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 31:16  |            | Reserved - Write as 0, read as don't care   |      |       |
| 15:0   | LEN        | Data Length<br>The number of bytes to transfer.<br>This field is only valid for DMA mode (DMAEN set) and controls how many bytes to transmit (and therefore receive). | RW   | 0x0   |

Table 161. DLEN Register

## LTOH Register

### Synopsis

This register allows the LoSSI output hold delay to be set.

| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 31:4   |            | Reserved - Write as 0, read as don't care   |      |       |
| 3:0    | TOH        | This sets the Output Hold delay in APB clocks. A value of 0 causes a 1 clock delay. | RW   | 0x1   |

Table 162. LTOH Register

## DC Register

### Synopsis

This register controls the generation of the DREQ and Panic signals to an external DMA engine. The DREQ signals are generated when the FIFOs reach their defined levels and need servicing. The Panic signals instruct the external DMA engine to raise the priority of its AXI requests.

| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 31:24  | RPANIC     | DMA Read Panic Threshold.<br>Generate the Panic signal to the RX DMA engine whenever the RX FIFO level is greater than this amount.  | RW   | 0x30  |
| 23:16  | RDREQ      | DMA Read Request Threshold.<br>Generate a DREQ to the RX DMA engine whenever the RX FIFO level is greater than this amount (RX DREQ is also generated if the transfer has finished but the RX FIFO isn't empty). | RW   | 0x20  |





| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 15:8   | TPANIC     | DMA Write Panic Threshold.<br>Generate the Panic signal to the TX DMA engine whenever the TX FIFO level is less than or equal to this amount. | RW   | 0x10  |
| 7:0    | TDREQ      | DMA Write Request Threshold.<br>Generate a DREQ signal to the TX DMA engine whenever the TX FIFO level is less than or equal to this amount.  | RW   | 0x20  |

Table 163. DC Register

## 9.6. Software Operation

### 9.6.1. Polled

1. Set CS, CPOL, CPHA as required and set TA = 1
2. Poll TXD writing bytes to SPI\_FIFO, RXD reading bytes from SPI\_FIFO until all data written
3. Poll DONE until it goes to 1
4. Set TA = 0

### 9.6.2. Interrupt

1. Set INTR and INTD. These can be left set over multiple operations.
2. Set CS, CPOL, CPHA as required and set TA = 1. This will immediately trigger a first interrupt with DONE = 1.
3. On interrupt:
  - If DONE is set and data to write (this means it is the first interrupt), write up to 64 bytes to SPI\_FIFO. If DONE is set and no more data, set TA = 0. Read trailing data from SPI\_FIFO until RXD is 0.
  - If RXR is set read 48 bytes data from SPI\_FIFO and if more data to write, write up to 48 bytes to SPI\_FIFO.

### 9.6.3. DMA

**Note:** In order to function correctly, each DMA channel must be set to perform 32-bit transfers when communicating with the SPI. Either the Source or the Destination Transfer Width field in the DMA TI register must be set to 0 (i.e. 32-bit words) depending upon whether the channel is reading or writing to the SPI. Two DMA channels are required, one to read from and one to write to the SPI.

1. Enable DMA DREQs by setting the DMAEN bit and ADCS if required.
2. Program two DMA Control Blocks, one for each DMA controller.



3. DMA channel 1 Control Block should have its PERMAP set to SPI<sub>n</sub> TX and should be set to write ‘transfer length’ + 1 words to SPI\_FIFO. The data should comprise:
  - a. A word with the transfer length in bytes in the top sixteen bits, and the control register settings [7:0] in the bottom eight bits (i.e. TA = 1, CS, CPOL, CPHA as required.).
  - b. ‘Transfer length’ number in words of data to send.
4. DMA channel 2 Control Block should have its PERMAP set to SPI<sub>n</sub> RX and should be set to read ‘transfer length’ words from SPI\_FIFO.
5. Point each DMA channel at its CB and set its ACTIVE bit to 1.
6. On receipt of an interrupt from DMA channel 2, the transfer is complete.

#### 9.6.4. Notes

1. The SPI Master knows nothing of the peripherals it is connected to. It always both sends and receives bytes for every byte of the transaction.
2. SCLK is only generated during byte serial transfer. It pauses in the rest state if the next byte to send is not ready or RXF is set.
3. Setup and Hold times related to the automatic assertion and de-assertion of the CS lines when operating in DMA mode (DMAEN and ADCS set) are as follows:
  - The CS line will be asserted at least 3 core clock cycles before the MSB of the first byte of the transfer.
  - The CS line will be de-asserted no earlier than 1 core clock cycle after the trailing edge of the final clock pulse.
  - If these parameters are insufficient, software control should alleviate the problem. ADCS should be 0 allowing software to manually control the assertion and de-assertion of the CS lines.



## Chapter 10. System Timer

### 10.1. Overview

The System Timer peripheral provides four 32-bit timer channels and a single 64-bit free running counter. Each channel has an output compare register, which is compared against the 32 least significant bits of the free running counter values. When the two values match, the system timer peripheral generates a signal to indicate a match for the appropriate channel. The match signal is then fed into the interrupt controller. The interrupt service routine then reads the output compare register and adds the appropriate offset for the next timer tick. The free running counter is driven by the timer clock and stopped whenever the processor is stopped in debug mode.

The physical (hardware) base address for the system timers is `0x7E00 3000`.

### 10.2. System Timer Registers

| Address Offset | Register Name | Description                         | Size |
|----------------|---------------|-------------------------------------|------|
| 0x00           | CS            | System Timer Control/Status         | 32   |
| 0x04           | CLO           | System Timer Counter Lower 32 bits  | 32   |
| 0x08           | CHI           | System Timer Counter Higher 32 bits | 32   |
| 0x0C           | C0            | System Timer Compare 0              | 32   |
| 0x10           | C1            | System Timer Compare 1              | 32   |
| 0x14           | C2            | System Timer Compare 2              | 32   |
| 0x18           | C3            | System Timer Compare 3              | 32   |

Table 164. System Timer Registers

#### CS Register

##### Synopsis

System Timer Control / Status.

This register is used to record and clear timer channel comparator matches. The system timer match bits are routed to the interrupt controller where they can generate an interrupt.

The M0-3 fields contain the free-running counter match status. Write a one to the relevant bit to clear the match detect status bit and the corresponding interrupt request line.

| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 31:4   |            | Reserved - Write as 0, read as don't care   |      |       |
| 3      | M3         | System Timer Match 3<br>0 = No Timer 3 match since last cleared.<br>1 = Timer 3 match detected. | W1C  | 0x0   |



| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 2      | M2         | System Timer Match 2<br>0 = No Timer 2 match since last cleared.<br>1 = Timer 2 match detected. | W1C  | 0x0   |
| 1      | M1         | System Timer Match 1<br>0 = No Timer 1 match since last cleared.<br>1 = Timer 1 match detected. | W1C  | 0x0   |
| 0      | M0         | System Timer Match 0<br>0 = No Timer 0 match since last cleared.<br>1 = Timer 0 match detected. | W1C  | 0x0   |

Table 165. CS Register

## CLO Register

### Synopsis

System Timer Counter Lower bits.

The system timer free-running counter lower register is a read-only register that returns the current value of the lower 32-bits of the free running counter.

| Bit(s) | Field Name | Description                                      | Type | Reset |
|--------|------------|--|------|-------|
| 31:0   | CNT        | Lower 32-bits of the free running counter value. | RO   | 0x0   |

Table 166. CLO Register

## CHI Register

### Synopsis

System Timer Counter Higher bits.

The system timer free-running counter higher register is a read-only register that returns the current value of the higher 32-bits of the free running counter.

| Bit(s) | Field Name | Description                                       | Type | Reset |
|--------|------------|---|------|-------|
| 31:0   | CNT        | Higher 32-bits of the free running counter value. | RO   | 0x0   |

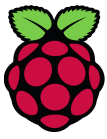
Table 167. CHI Register

## C0, C1, C2 & C3 Registers

### Synopsis

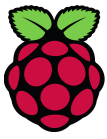
System Timer Compare.

The system timer compare registers hold the compare value for each of the four timer channels. Whenever the lower 32-bits of the free-running counter matches one of the compare values the corresponding bit in the system timer control/status register is set.



| Bit(s) | Field Name | Description                                | Type | Reset |
|--------|------------|--|------|-------|
| 31:0   | CMP        | Compare value for match channel <i>n</i> . | RW   | 0x0   |

*Table 168. C0, C1, C2 & C3 Registers*



# Chapter 11. UART

## 11.1. Overview

The BCM2711 device has six UARTs. One mini UART (UART1) and five PL011 UARTs (UART0, UART2, UART3, UART4 & UART5). This section describes the PL011 UARTs. For details of the mini UART see [Section 2.2](#).

The PL011 UART is a Universal Asynchronous Receiver/Transmitter. This is the ARM UART (PL011) implementation. The UART performs serial-to-parallel conversion on data characters received from an external peripheral device or modem, and parallel-to-serial conversion on data characters received from the Advanced Peripheral Bus (APB).

The ARM PL011 UART has some optional functionality which can be included or left out.

The following functionality is ***not supported*** :

- Infrared Data Association (IrDA)
- Serial InfraRed (SIR) protocol Encoder/Decoder (ENDEC)

The UARTs provide:

- Separate 32x8 transmit and 32x12 receive FIFO memory.
- Programmable baud rate generator.
- Standard asynchronous communication bits (start, stop and parity). These are added prior to transmission and removed on reception.
- False start bit detection.
- Line break generation and detection.
- Support of the modem control functions CTS and RTS. However DCD, DSR, DTR, and RI are not supported.
- Programmable hardware flow control.
- Fully-programmable serial interface characteristics:
  - data can be 5, 6, 7, or 8 bits.
  - even, odd, stick, or no-parity bit generation and detection.
  - 1 or 2 stop bit generation.
  - baud rate generation, up to UARTCLK/16.

The UART clock source and associated dividers are controlled by the Clock Manager.

For the in-depth UART overview, please refer to the ARM PrimeCell UART (PL011) Revision: r1p5 Technical Reference Manual.



## 11.2. Variations from the 16C650 UART

The UART varies from the industry-standard 16C650 UART device as follows:

- Receive FIFO trigger levels are 1/8, 1/4, 1/2, 3/4, and 7/8
- Transmit FIFO trigger levels are 1/8, 1/4, 1/2, 3/4, and 7/8
- The internal register map address space, and the bit function of each register differ
- The deltas of the modem status signals are not available

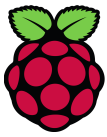
The following 16C650 UART features are not supported:

- 1.5 stop bits (1 or 2 stop bits only are supported)
- Independent receive clock

## 11.3. Primary UART Inputs and Outputs

The UARTs have two primary inputs (RXD, nCTS) and two primary outputs (TXD, nRTS). The remaining signals (like SRIN, SROUT, OUT1, OUT2, DSR, DTR, and RI) are not supported in this implementation. The following table shows how the various UART signals (including the mini UART) map on the General Purpose I/O (GPIO). For more details on how to select alternate functions refer to [Chapter 5](#).

|        | Pull | ALT0 | ALT1 | ALT2 | ALT3 | ALT4 | ALT5 |
|--------|------|------|------|------|------|------|------|
| GPIO0  | High |      |      |      |      | TXD2 |      |
| GPIO1  | High |      |      |      |      | RXD2 |      |
| GPIO2  | High |      |      |      |      | CTS2 |      |
| GPIO3  | High |      |      |      |      | RTS2 |      |
| GPIO4  | High |      |      |      |      | TXD3 |      |
| GPIO5  | High |      |      |      |      | RXD3 |      |
| GPIO6  | High |      |      |      |      | CTS3 |      |
| GPIO7  | High |      |      |      |      | RTS3 |      |
| GPIO8  | High |      |      |      |      | TXD4 |      |
| GPIO9  | High |      |      |      |      | RXD4 |      |
| GPIO10 | High |      |      |      |      | CTS4 |      |
| GPIO11 | High |      |      |      |      | RTS4 |      |
| GPIO12 | High |      |      |      |      | TXD5 |      |
| GPIO13 | High |      |      |      |      | RXD5 |      |
| GPIO14 | Low  | TXD0 |      |      |      | CTS5 |      |
| GPIO15 | Low  | RXD0 |      |      |      | RTS5 |      |



|        | <b>Pull</b> | <b>ALT0</b> | <b>ALT1</b> | <b>ALT2</b> | <b>ALT3</b> | <b>ALT4</b> | <b>ALT5</b> |
|--------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| GPIO16 | Low         |             |             |             | CTS0        |             |             |
| GPIO17 | Low         |             |             |             | RTS0        |             |             |
| GPIO30 | Low         |             |             |             | CTS0        |             | CTS1        |
| GPIO31 | Low         |             |             |             | RTS0        |             | RTS1        |
| GPIO32 | Low         |             |             |             | TXD0        |             | TXD1        |
| GPIO33 | Low         |             |             |             | RXD0        |             | RXD1        |
| GPIO36 | High        |             |             | TXD0        |             |             |             |
| GPIO37 | Low         |             |             | RXD0        |             |             |             |
| GPIO38 | Low         |             |             | RTS0        |             |             |             |
| GPIO39 | Low         |             |             | CTS0        |             |             |             |
| GPIO40 | Low         |             |             |             |             |             | TXD1        |
| GPIO41 | Low         |             |             |             |             |             | RXD1        |
| GPIO42 | Low         |             |             |             |             |             | RTS1        |
| GPIO43 | Low         |             |             |             |             |             | CTS1        |

Table 169. UART Assignment on the GPIO Pin map

## 11.4. UART Interrupts

Each UART has one intra-chip interrupt `UARTINTR` generated as the OR-ed function of its five individual interrupts.

`UARTINTR`, this is an OR function of the five individual masked outputs:

- `UARTRXINTR`
- `UARTTXINTR`
- `UARTRTINTR`
- `UARTMSINTR`, that can be caused by:
  - `UARTCTSINTR`, because of a change in the `nUARTCTS` modem status
  - `UARTDSRINTR`, because of a change in the `nUARTDSR` modem status
- `UARTEINTR`, that can be caused by an error in the reception:
  - `UARTOEINTR`, because of an overrun error
  - `UARTBEINTR`, because of a break in the reception
  - `UARTPEINTR`, because of a parity error in the received character
  - `UARTFEINTR`, because of a framing error in the received character





One can enable or disable the individual interrupts by changing the mask bits in the Interrupt Mask Set/Clear Register, UART\_IMSC. Setting the appropriate mask bit HIGH enables the interrupt.

### UARTTXINTR

The transmit interrupt changes state when one of the following events occurs:

- If the FIFOs are enabled and the transmit FIFO is equal to or lower than the programmed trigger level then the transmit interrupt is asserted HIGH. The transmit interrupt is cleared by writing data to the transmit FIFO until it becomes greater than the trigger level, or by clearing the interrupt.
- If the FIFOs are disabled (have a depth of one location) and there is no data present in the transmitter's single location, the transmit interrupt is asserted HIGH. It is cleared by performing a single write to the transmit FIFO, or by clearing the interrupt.

### UARTRXINTR

The receive interrupt changes state when one of the following events occurs:

- If the FIFOs are enabled and the receive FIFO reaches the programmed trigger level. When this happens, the receive interrupt is asserted HIGH. The receive interrupt is cleared by reading data from the receive FIFO until it becomes less than the trigger level, or by clearing the interrupt.
- If the FIFOs are disabled (have a depth of one location) and data is received thereby filling the location, the receive interrupt is asserted HIGH. The receive interrupt is cleared by performing a single read of the receive FIFO, or by clearing the interrupt.

## 11.5. Register View

The PL011 UARTs are mapped onto the following base addresses:

- UART0: 0x7E20 1000
- UART2: 0x7E20 1400
- UART3: 0x7E20 1600
- UART4: 0x7E20 1800
- UART5: 0x7E20 1A00

They have the following memory-mapped registers.

| Address Offset | Register Name | Description   | Size |
|----------------|---------------|---------------|------|
| 0x00           | DR            | Data Register | 32   |
| 0x04           | RSRECR        |               | 32   |
| 0x18           | FR            | Flag register | 32   |
| 0x20           | ILPR          | not in use    | 32   |



| Address Offset | Register Name         | Description                          | Size |
|----------------|-----------------------|--------------------------------------|------|
| 0x24           | <a href="#">IBRD</a>  | Integer Baud rate divisor            | 32   |
| 0x28           | <a href="#">FBRD</a>  | Fractional Baud rate divisor         | 32   |
| 0x2C           | <a href="#">LCRH</a>  | Line Control register                | 32   |
| 0x30           | <a href="#">CR</a>    | Control register                     | 32   |
| 0x34           | <a href="#">IFLS</a>  | Interrupt FIFO Level Select Register | 32   |
| 0x38           | <a href="#">IMSC</a>  | Interrupt Mask Set Clear Register    | 32   |
| 0x3C           | <a href="#">RIS</a>   | Raw Interrupt Status Register        | 32   |
| 0x40           | <a href="#">MIS</a>   | Masked Interrupt Status Register     | 32   |
| 0x44           | <a href="#">ICR</a>   | Interrupt Clear Register             | 32   |
| 0x48           | <a href="#">DMACR</a> | DMA Control Register                 | 32   |
| 0x80           | <a href="#">ITCR</a>  | Test Control register                | 32   |
| 0x84           | <a href="#">ITIP</a>  | Integration test input reg           | 32   |
| 0x88           | <a href="#">ITOP</a>  | Integration test output reg          | 32   |
| 0x8C           | <a href="#">TDR</a>   | Test Data reg                        | 32   |

Table 170. UART Registers

## DR Register

### Synopsis

The UART\_DR Register is the data register.

For words to be transmitted:

if the FIFOs are enabled, data written to this location is pushed onto the transmit FIFO.

if the FIFOs are not enabled, data is stored in the transmitter holding register (the bottom word of the transmit FIFO).

The write operation initiates transmission from the UART. The data is prefixed with a start bit, appended with the appropriate parity bit (if parity is enabled), and a stop bit. The resultant word is then transmitted.

For received words:

if the FIFOs are enabled, the data byte and the 4-bit status (break, frame, parity, and overrun) is pushed onto the 12-bit wide receive FIFO

if the FIFOs are not enabled, the data byte and status are stored in the receiving holding register (the bottom word of the receive FIFO).

| Bit(s) | Field Name | Description                               | Type | Reset |
|--------|------------|---|------|-------|
| 31:12  |            | Reserved - Write as 0, read as don't care |      |       |



| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 11     | OE         | Overrun error. This bit is set to 1 if data is received and the receive FIFO is already full.<br>This is cleared to 0 once there is an empty space in the FIFO and a new character can be written to it.   | RO   | 0x0   |
| 10     | BE         | Break error. This bit is set to 1 if a break condition was detected, indicating that the received data input was held LOW for longer than a full-word transmission time (defined as start, data, parity and stop bits).<br>In FIFO mode, this error is associated with the character at the top of the FIFO. When a break occurs, only one 0 character is loaded into the FIFO. The next character is only enabled after the receive data input goes to a 1 (marking state), and the next valid start bit is received. | RO   | 0x0   |
| 9      | PE         | Parity error. When set to 1, it indicates that the parity of the received data character does not match the parity that the EPS and SPS bits in the Line Control Register, UART_LCRH select.<br>In FIFO mode, this error is associated with the character at the top of the FIFO.  | RO   | 0x0   |
| 8      | FE         | Framing error. When set to 1, it indicates that the received character did not have a valid stop bit (a valid stop bit is 1).<br>In FIFO mode, this error is associated with the character at the top of the FIFO.   | RO   | 0x0   |
| 7:0    | DATA       | Receive (read) data character.<br>Transmit (write) data character.   | RW   | 0x0   |

Table 171. DR Register

## RSRECR Register

### Synopsis

The UART\_RSRECR Register is the receive status register/error clear register. If the status is read from this register, then the status information for break, framing and parity corresponds to the data character read from the Data Register, UART\_DR. The status information for overrun is set immediately when an overrun condition occurs. NOTE: The received data character must be read first from the Data Register UART\_DR, before reading the error status associated with that data character from this register.

| Bit(s) | Field Name | Description                               | Type | Reset |
|--------|------------|---|------|-------|
| 31:4   |            | Reserved - Write as 0, read as don't care |      |       |



| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 3      | OE         | Overrun error. This bit is set to 1 if data is received and the receive FIFO is already full.<br>This is cleared to 0 once there is an empty space in the FIFO and a new character can be written to it.   | RW   | 0x0   |
| 2      | BE         | Break error. This bit is set to 1 if a break condition was detected, indicating that the received data input was held LOW for longer than a full-word transmission time (defined as start, data, parity and stop bits).<br>In FIFO mode, this error is associated with the character at the top of the FIFO. When a break occurs, only one 0 character is loaded into the FIFO. The next character is only enabled after the receive data input goes to a 1 (marking state), and the next valid start bit is received. | RW   | 0x0   |
| 1      | PE         | Parity error. When set to 1, it indicates that the parity of the received data character does not match the parity that the EPS and SPS bits in the Line Control Register, UART_LCRH select.<br>In FIFO mode, this error is associated with the character at the top of the FIFO.  | RW   | 0x0   |
| 0      | FE         | Framing error. When set to 1, it indicates that the received character did not have a valid stop bit (a valid stop bit is 1).<br>In FIFO mode, this error is associated with the character at the top of the FIFO.   | RW   | 0x0   |

Table 172. RSRECR Register

## FR Register

### Synopsis

The UART\_FR Register is the flag register.

| Bit(s) | Field Name | Description                                 | Type | Reset |
|--------|------------|---|------|-------|
| 31:9   |            | Reserved - Write as 0, read as don't care   |      |       |
| 8      | RI         | Unsupported, write zero, read as don't care | RO   | 0x0   |



| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 7      | TXFE       | Transmit FIFO empty. The meaning of this bit depends on the state of the FEN bit in the Line Control Register, UART_LCRH.<br>If the FIFO is disabled, this bit is set when the transmit holding register is empty.<br>If the FIFO is enabled, the TXFE bit is set when the transmit FIFO is empty. This bit does not indicate if there is data in the transmit shift register. | RO   | 0x1   |
| 6      | RXFF       | Receive FIFO full. The meaning of this bit depends on the state of the FEN bit in the UART_LCRH Register.<br>If the FIFO is disabled, this bit is set when the receive holding register is full.<br>If the FIFO is enabled, the RXFF bit is set when the receive FIFO is full.   | RO   | 0x0   |
| 5      | TXFF       | Transmit FIFO full. The meaning of this bit depends on the state of the FEN bit in the UART_LCRH Register.<br>If the FIFO is disabled, this bit is set when the transmit holding register is full.<br>If the FIFO is enabled, the TXFF bit is set when the transmit FIFO is full.  | RO   | 0x0   |
| 4      | RXFE       | Receive FIFO empty. The meaning of this bit depends on the state of the FEN bit in the UART_LCRH Register.<br>If the FIFO is disabled, this bit is set when the receive holding register is empty.<br>If the FIFO is enabled, the RXFE bit is set when the receive FIFO is empty.  | RO   | 0x0   |
| 3      | BUSY       | UART busy. If this bit is set to 1, the UART is busy transmitting data. This bit remains set until the complete byte, including all the stop bits, has been sent from the shift register.<br>This bit is set as soon as the transmit FIFO becomes non-empty, regardless of whether the UART is enabled or not.   | RO   | 0x0   |
| 2      | DCD        | Unsupported, write zero, read as don't care  | RO   | 0x0   |
| 1      | DSR        | Unsupported, write zero, read as don't care  | RO   | 0x0   |
| 0      | CTS        | Clear to send. This bit is the complement of the UART clear to send, nUARTCTS, modem status input. That is, the bit is 1 when nUARTCTS is LOW.   | RO   | 0x0   |

Table 173. FR Register

## ILPR Register



## Synopsis

This is the disabled IrDA register, writing to it has no effect and reading returns 0.

| Bit(s) | Field Name | Description                                | Type | Reset |
|--------|------------|--|------|-------|
| 31:0   |            | Reserved - write zero, read as don't care. |      |       |

*Table 174. ILPR Register*

## IBRD Register

### Synopsis

The UART\_IBRD Register is the integer part of the baud rate divisor value.

| Bit(s) | Field Name | Description                               | Type | Reset |
|--------|------------|---|------|-------|
| 31:16  |            | Reserved - Write as 0, read as don't care |      |       |
| 15:0   | IBRD       | The integer baud rate divisor.            | RW   | 0x0   |

*Table 175. IBRD Register*

## FBRD Register

### Synopsis

The UART\_FBRD Register is the fractional part of the baud rate divisor value.

The baud rate divisor is calculated as follows:

Baud rate divisor BAUDDIV = (FUARTCLK/(16 \* Baud rate))

where FUARTCLK is the UART reference clock frequency. The BAUDDIV is comprised of the integer value IBRD and the fractional value FBRD.

NOTE: The contents of the IBRD and FBRD registers are not updated until transmission or reception of the current character is complete.

| Bit(s) | Field Name | Description                               | Type | Reset |
|--------|------------|---|------|-------|
| 31:6   |            | Reserved - Write as 0, read as don't care |      |       |
| 5:0    | FBRD       | The fractional baud rate divisor.         | RW   | 0x0   |

*Table 176. FBRD Register*

## LCRH Register

### Synopsis

The UART\_LCRH Register is the line control register.

NOTE: The UART\_LCRH, UART\_IBRD, and UART\_FBRD registers must not be changed:

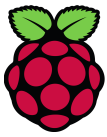
when the UART is enabled

when completing a transmission or a reception when it has been programmed to become disabled.



| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 31:8   |            | Reserved - Write as 0, read as don't care  |      |       |
| 7      | SPS        | Stick parity select.<br>0 = stick parity is disabled<br>1 = either:<br>if the EPS bit is 0 then the parity bit is transmitted and checked as a 1<br>if the EPS bit is 1 then the parity bit is transmitted and checked as a 0. See <a href="#">Table 178</a> .   | RO   | 0x0   |
| 6:5    | WLEN       | Word length. These bits indicate the number of data bits transmitted or received in a frame as follows:<br>b11 = 8 bits<br>b10 = 7 bits<br>b01 = 6 bits<br>b00 = 5 bits.   | RW   | 0x0   |
| 4      | FEN        | Enable FIFOs:<br>0 = FIFOs are disabled (character mode) that is, the FIFOs become 1-byte-deep holding registers<br>1 = transmit and receive FIFO buffers are enabled (FIFO mode).   | RW   | 0x0   |
| 3      | STP2       | Two stop bits select. If this bit is set to 1, two stop bits are transmitted at the end of the frame. The receive logic does not check for two stop bits being received.   | RW   | 0x0   |
| 2      | EPS        | Even parity select. Controls the type of parity the UART uses during transmission and reception:<br>0 = odd parity. The UART generates or checks for an odd number of 1s in the data and parity bits.<br>1 = even parity. The UART generates or checks for an even number of 1s in the data and parity bits.<br>This bit has no effect when the PEN bit disables parity checking and generation. See <a href="#">Table 178</a> . | RW   | 0x0   |
| 1      | PEN        | Parity enable:<br>0 = parity is disabled and no parity bit added to the data frame<br>1 = parity checking and generation is enabled. See <a href="#">Table 178</a> .   | RW   | 0x0   |
| 0      | BRK        | Send break. If this bit is set to 1, a low-level is continually output on the TXD output, after completing transmission of the current character.  | RW   | 0x0   |

Table 177. LCRH Register



| PEN | EPS | SPS | Parity bit (transmitted or checked) |
|-----|-----|-----|-------------------------------------|
| 0   | x   | x   | Not transmitted or checked          |
| 1   | 1   | 0   | Even parity                         |
| 1   | 0   | 0   | Odd parity                          |
| 1   | 0   | 1   | 1                                   |
| 1   | 1   | 1   | 0                                   |

Table 178. UART parity bits

## CR Register

### Synopsis

The UART\_CR Register is the control register.

NOTE: To enable transmission, the TXE bit and UARTEN bit must be set to 1. Similarly, to enable reception, the RXE bit and UARTEN bit, must be set to 1.

NOTE: Program the control registers as follows:

1. Disable the UART.
2. Wait for the end of transmission or reception of the current character.
3. Flush the transmit FIFO by setting the FEN bit to 0 in the Line Control Register, UART\_LCRH.
4. Reprogram the Control Register, UART\_CR.
5. Enable the UART.

| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 31:16  |            | Reserved - Write as 0, read as don't care  |      |       |
| 15     | CTSEN      | CTS hardware flow control enable. If this bit is set to 1, CTS hardware flow control is enabled. Data is only transmitted when the nUARTCTS signal is asserted.                        | RW   | 0x0   |
| 14     | RTSEN      | RTS hardware flow control enable. If this bit is set to 1, RTS hardware flow control is enabled. Data is only requested when there is space in the receive FIFO for it to be received. | RW   | 0x0   |
| 13     | OUT2       | Unsupported, write zero, read as don't care  | RO   | 0x0   |
| 12     | OUT1       | Unsupported, write zero, read as don't care  | RO   | 0x0   |
| 11     | RTS        | Request to send. This bit is the complement of the UART request to send, nUARTRTS, modem status output. That is, when the bit is programmed to a 1 then nUARTRTS is LOW.               | RW   | 0x0   |
| 10     | DTR        | Unsupported, write zero, read as don't care  | RO   | 0x0   |





| Bit(s) | Field Name                                | Description  | Type | Reset |
|--------|---|--|------|-------|
| 9      | RXE                                       | Receive enable. If this bit is set to 1, the receive section of the UART is enabled. Data reception occurs for UART signals. When the UART is disabled in the middle of reception, it completes the current character before stopping.                 | RW   | 0x1   |
| 8      | TXE                                       | Transmit enable. If this bit is set to 1, the transmit section of the UART is enabled. Data transmission occurs for UART signals. When the UART is disabled in the middle of transmission, it completes the current character before stopping.         | RW   | 0x1   |
| 7      | LBE                                       | Loopback enable. If this bit is set to 1, the UARTTXD path is fed through to the UARTRXD path. In UART mode, when this bit is set, the modem outputs are also fed through to the modem inputs. This bit is cleared to 0 on reset, to disable loopback. | RW   | 0x0   |
| 6:3    | Reserved - Write as 0, read as don't care |  |      |       |
| 2      | SIRLP                                     | Unsupported, write zero, read as don't care  | RO   | 0x0   |
| 1      | SIREN                                     | Unsupported, write zero, read as don't care  | RO   | 0x0   |
| 0      | UARTEN                                    | UART enable:<br>0 = UART is disabled. If the UART is disabled in the middle of transmission or reception, it completes the current character before stopping.<br>1 = the UART is enabled.  | RW   | 0x0   |

Table 179. CR Register

## IFLS Register

### Synopsis

The UART\_IFLS Register is the interrupt FIFO level select register. You can use this register to define the FIFO level that triggers the assertion of the combined interrupt signal.

The interrupts are generated based on a transition through a level rather than being based on the level. That is, the interrupts are generated when the fill level progresses through the trigger level.

The bits are reset so that the trigger level is when the FIFOs are at the half-way mark.

| Bit(s) | Field Name                                | Description                                 | Type | Reset |
|--------|---|---|------|-------|
| 31:12  | Reserved - Write as 0, read as don't care |   |      |       |
| 11:9   | RXIFPSEL                                  | Unsupported, write zero, read as don't care | RO   | 0x0   |
| 8:6    | TXIFPSEL                                  | Unsupported, write zero, read as don't care | RO   | 0x0   |



| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 5:3    | RXIFLSEL   | Receive interrupt FIFO level select. The trigger points for the receive interrupt are as follows:<br>b000 = Receive FIFO becomes 1/8 full<br>b001 = Receive FIFO becomes 1/4 full<br>b010 = Receive FIFO becomes 1/2 full<br>b011 = Receive FIFO becomes 3/4 full<br>b100 = Receive FIFO becomes 7/8 full<br>b101-b111 = reserved.        | RW   | 0x2   |
| 2:0    | TXIFLSEL   | Transmit interrupt FIFO level select. The trigger points for the transmit interrupt are as follows:<br>b000 = Transmit FIFO becomes 1/8 full<br>b001 = Transmit FIFO becomes 1/4 full<br>b010 = Transmit FIFO becomes 1/2 full<br>b011 = Transmit FIFO becomes 3/4 full<br>b100 = Transmit FIFO becomes 7/8 full<br>b101-b111 = reserved. | RW   | 0x2   |

Table 180. IFLS Register

## IMSC Register

### Synopsis

The UART\_IMSC Register is the interrupt mask set/clear register. It is a read/write register. On a read this register returns the current value of the mask on the relevant interrupt. On a write of 1 to the particular bit, it sets the corresponding mask of that interrupt. A write of 0 clears the corresponding mask.

| Bit(s) | Field Name                                | Description  | Type | Reset |
|--------|---|--|------|-------|
| 31:11  | Reserved - Write as 0, read as don't care |  |      |       |
| 10     | OEIM                                      | Overrun error interrupt mask. A read returns the current mask for the interrupt. On a write of 1, the mask of the UARTOEINTR interrupt is set. A write of 0 clears the mask. | RW   | 0x0   |
| 9      | BEIM                                      | Break error interrupt mask. A read returns the current mask for the UARTBEINTR interrupt. On a write of 1, the mask of the interrupt is set. A write of 0 clears the mask.   | RW   | 0x0   |
| 8      | PEIM                                      | Parity error interrupt mask. A read returns the current mask for the UARTPEINTR interrupt. On a write of 1, the mask of the interrupt is set. A write of 0 clears the mask.  | RW   | 0x0   |



| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 7      | FEIM       | Framing error interrupt mask. A read returns the current mask for the UARTFEINTR interrupt. On a write of 1, the mask of the interrupt is set. A write of 0 clears the mask.   | RW   | 0x0   |
| 6      | RTIM       | Receive timeout interrupt mask. A read returns the current mask for the UARTRTINTR interrupt. On a write of 1, the mask of the interrupt is set. A write of 0 clears the mask. | RW   | 0x0   |
| 5      | TXIM       | Transmit interrupt mask. A read returns the current mask for the UARTTXINTR interrupt. On a write of 1, the mask of the interrupt is set. A write of 0 clears the mask.        | RW   | 0x0   |
| 4      | RXIM       | Receive interrupt mask. A read returns the current mask for the UARTRXINTR interrupt. On a write of 1, the mask of the interrupt is set. A write of 0 clears the mask.         | RW   | 0x0   |
| 3      | DSRMIM     | Unsupported, write zero, read as don't care  | RO   | 0x0   |
| 2      | DCDMIM     | Unsupported, write zero, read as don't care  | RO   | 0x0   |
| 1      | CTSMIM     | nUARTCTS modem interrupt mask. A read returns the current mask for the UARTCTSINTR interrupt. On a write of 1, the mask of the interrupt is set. A write of 0 clears the mask. | RW   | 0x0   |
| 0      | RIMIM      | Unsupported, write zero, read as don't care  | RO   | 0x0   |

Table 181. IMSC Register

## RIS Register

### Synopsis

The UART\_RIS Register is the raw interrupt status register. It is a read-only register. This register returns the current raw status value, prior to masking, of the corresponding interrupt.

NOTE: All the bits, except for the modem status interrupt bits (bits 3 to 0), are cleared to 0 when reset. The modem status interrupt bits are undefined after reset.

| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 31:11  |            | Reserved - Write as 0, read as don't care  |      |       |
| 10     | OERIS      | Overflow error interrupt status. Returns the raw interrupt state of the UARTOERINTR interrupt. | RO   | 0x0   |
| 9      | BERIS      | Break error interrupt status. Returns the raw interrupt state of the UARTBEINTR interrupt.     | RO   | 0x0   |



| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 8      | PERIS      | Parity error interrupt status. Returns the raw interrupt state of the UARTPEINTR interrupt.    | RO   | 0x0   |
| 7      | FERIS      | Framing error interrupt status. Returns the raw interrupt state of the UARTFEINTR interrupt.   | RO   | 0x0   |
| 6      | RTRIS      | Receive timeout interrupt status. Returns the raw interrupt state of the UARTRTINTR interrupt. | RO   | 0x0   |
| 5      | TXRIS      | Transmit interrupt status. Returns the raw interrupt state of the UARTTXINTR interrupt.        | RO   | 0x0   |
| 4      | RXRIS      | Receive interrupt status. Returns the raw interrupt state of the UARTRXINTR interrupt.         | RO   | 0x0   |
| 3      | DSRRMIS    | Unsupported, write zero, read as don't care  | RO   | 0x0   |
| 2      | DCDRMIS    | Unsupported, write zero, read as don't care  | RO   | 0x0   |
| 1      | CTSRMIS    | nUARTCTS modem interrupt status. Returns the raw interrupt state of the UARTCTSINTR interrupt. | RO   | 0x0   |
| 0      | RIRMIS     | Unsupported, write zero, read as don't care  | RO   | 0x0   |

Table 182. RIS Register

## MIS Register

### Synopsis

The UART\_MIS Register is the masked interrupt status register. This register returns the current masked status value of the corresponding interrupt.

NOTE: All the bits, except for the modem status interrupt bits (bits 3 to 0), are cleared to 0 when reset. The modem status interrupt bits are undefined after reset.

| Bit(s) | Field Name                                | Description  | Type | Reset |
|--------|---|--|------|-------|
| 31:11  | Reserved - Write as 0, read as don't care |  |      |       |
| 10     | OEMIS                                     | Overflow error masked interrupt status. Returns the masked interrupt state of the UARTOEINTR interrupt.  | RO   | 0x0   |
| 9      | BEMIS                                     | Break error masked interrupt status. Returns the masked interrupt state of the UARTBEINTR interrupt.     | RO   | 0x0   |
| 8      | PEMIS                                     | Parity error masked interrupt status. Returns the masked interrupt state of the UARTPEINTR interrupt.    | RO   | 0x0   |
| 7      | FEMIS                                     | Framing error masked interrupt status. Returns the masked interrupt state of the UARTFEINTR interrupt.   | RO   | 0x0   |
| 6      | RTMIS                                     | Receive timeout masked interrupt status. Returns the masked interrupt state of the UARTRTINTR interrupt. | RO   | 0x0   |



| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 5      | TXMIS      | Transmit masked interrupt status. Returns the masked interrupt state of the UARTTXINTR interrupt.        | RO   | 0x0   |
| 4      | RXMIS      | Receive masked interrupt status. Returns the masked interrupt state of the UARTRXINTR interrupt.         | RO   | 0x0   |
| 3      | DSRMMIS    | Unsupported, write zero, read as don't care  | RO   | 0x0   |
| 2      | DCDMMIS    | Unsupported, write zero, read as don't care  | RO   | 0x0   |
| 1      | CTSMMIS    | nUARTCTS modem masked interrupt status. Returns the masked interrupt state of the UARTCTSINTR interrupt. | RO   | 0x0   |
| 0      | RIMMIS     | Unsupported, write zero, read as don't care  | RO   | 0x0   |

Table 183. MIS Register

## ICR Register

### Synopsis

The UART\_ICR Register is the interrupt clear register.

| Bit(s) | Field Name                                | Description   | Type | Reset |
|--------|---|---|------|-------|
| 31:11  | Reserved - Write as 0, read as don't care |   |      |       |
| 10     | OEIC                                      | Overrun error interrupt clear. Clears the UARTOEINTR interrupt.   | WO   | 0x0   |
| 9      | BEIC                                      | Break error interrupt clear. Clears the UARTBEINTR interrupt.     | WO   | 0x0   |
| 8      | PEIC                                      | Parity error interrupt clear. Clears the UARTPEINTR interrupt.    | WO   | 0x0   |
| 7      | FEIC                                      | Framing error interrupt clear. Clears the UARTFEINTR interrupt.   | WO   | 0x0   |
| 6      | RTIC                                      | Receive timeout interrupt clear. Clears the UARTRTINTR interrupt. | WO   | 0x0   |
| 5      | TXIC                                      | Transmit interrupt clear. Clears the UARTTXINTR interrupt.        | WO   | 0x0   |
| 4      | RXIC                                      | Receive interrupt clear. Clears the UARTRXINTR interrupt.         | WO   | 0x0   |
| 3      | DSRMIC                                    | Unsupported, write zero, read as don't care                       | WO   | 0x0   |
| 2      | DCDMIC                                    | Unsupported, write zero, read as don't care                       | WO   | 0x0   |
| 1      | CTSMIC                                    | nUARTCTS modem interrupt clear. Clears the UARTCTSINTR interrupt. | WO   | 0x0   |
| 0      | RIMIC                                     | Unsupported, write zero, read as don't care                       | WO   | 0x0   |



Table 184. ICR Register

## DMACR Register

### Synopsis

The UART\_DMCCR Register is the DMA control register.

| Bit(s) | Field Name                                | Description  | Type | Reset |
|--------|---|--|------|-------|
| 31:3   | Reserved - Write as 0, read as don't care |  |      |       |
| 2      | DMAONERR                                  | DMA on error. If this bit is set to 1, the DMA receive request outputs are disabled when the UART error interrupt is asserted. | RW   | 0x0   |
| 1      | TXDMAE                                    | Transmit DMA enable. If this bit is set to 1, DMA for the transmit FIFO is enabled.  | RW   | 0x0   |
| 0      | RXDMAE                                    | Receive DMA enable. If this bit is set to 1, DMA for the receive FIFO is enabled.  | RW   | 0x0   |

Table 185. DMCCR Register

## ITCR Register

### Synopsis

This is the Test Control Register UART\_ITCR.

| Bit(s) | Field Name                                | Description   | Type | Reset |
|--------|---|---|------|-------|
| 31:2   | Reserved - Write as 0, read as don't care |   |      |       |
| 1      | ITCR1                                     | Test FIFO enable. When this bit is 1, a write to the Test Data Register, UART_TDR writes data into the receive FIFO, and reading from the UART_TDR register reads data out of the transmit FIFO.<br>When this bit is 0, data cannot be read directly from the transmit FIFO or written directly to the receive FIFO (normal operation). | RW   | 0x0   |
| 0      | ITCR0                                     | Integration test enable. When this bit is 1, the UART is placed in integration test mode, otherwise it is in normal operation.  | RW   | 0x0   |

Table 186. ITCR Register

## ITIP Register

### Synopsis

This is the Test Control Register UART\_ITIP.



| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 31:4   |            | Reserved - Write as 0, read as don't care             |      |       |
| 3      | ITIP3      | Reads return the value of the nUARTCTS primary input. | RW   | 0x0   |
| 2:1    |            | Reserved - Write as 0, read as don't care             |      |       |
| 0      | ITIP0      | Reads return the value of the UARTRXD primary input.  | RW   | 0x0   |

Table 187. ITIP Register

## ITOP Register

### Synopsis

This is the Test Control Register UART\_ITOP.

| Bit(s) | Field Name | Description  | Type | Reset |
|--------|------------|--|------|-------|
| 31:12  |            | Reserved - Write as 0, read as don't care  |      |       |
| 11     | ITOP11     | Intra-chip output. Writes specify the value to be driven on UARTMSINTR.<br>Reads return the value of UARTMSINTR at the output of the test multiplexor. | RW   | 0x0   |
| 10     | ITOP10     | Intra-chip output. Writes specify the value to be driven on UARTRXINTR.<br>Reads return the value of UARTRXINTR at the output of the test multiplexor. | RW   | 0x0   |
| 9      | ITOP9      | Intra-chip output. Writes specify the value to be driven on UARTTXINTR.<br>Reads return the value of UARTTXINTR at the output of the test multiplexor. | RW   | 0x0   |
| 8      | ITOP8      | Intra-chip output. Writes specify the value to be driven on UARTRTINTR.<br>Reads return the value of UARTRTINTR at the output of the test multiplexor. | RW   | 0x0   |
| 7      | ITOP7      | Intra-chip output. Writes specify the value to be driven on UARTEINTR.<br>Reads return the value of UARTEINTR at the output of the test multiplexor.   | RW   | 0x0   |
| 6      | ITOP6      | Intra-chip output. Writes specify the value to be driven on UARTINTR.<br>Reads return the value of UARTINTR at the output of the test multiplexor.     | RW   | 0x0   |
| 5:4    |            | Reserved - Write as 0, read as don't care  |      |       |



| Bit(s) | Field Name                                | Description  | Type | Reset |
|--------|---|--|------|-------|
| 3      | ITOP3                                     | Primary output. Writes specify the value to be driven on nUARTRTS. | RW   | 0x0   |
| 2:1    | Reserved - Write as 0, read as don't care |  |      |       |
| 0      | ITOP0                                     | Primary output. Writes specify the value to be driven on UARTTXD.  | RW   | 0x0   |

*Table 188. ITOP Register*

## TDR Register

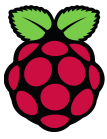
### Synopsis

UART\_TDR is the test data register. It enables data to be written into the receive FIFO and read out from the transmit FIFO for test purposes. This test function is enabled by the ITCR1 bit in the Test Control Register, UART\_ITCR.

| Bit(s) | Field Name                                | Description  | Type | Reset |
|--------|---|--|------|-------|
| 31:11  | Reserved - Write as 0, read as don't care |  |      |       |
| 10:0   | TDR10_0                                   | When the ITCR1 bit is set to 1, data is written into the receive FIFO and read out of the transmit FIFO. | RW   | 0x0   |

*Table 189. TDR Register*





## Chapter 12. Timer (ARM side)

### 12.1. Overview

The ARM Timer is *based* on a ARM SP804, but it has a number of differences with the standard SP804:

- There is only one timer
- It only runs in continuous mode
- It has a extra clock pre-divider register
- It has a extra stop-in-debug-mode control bit
- It also has a 32-bit free running counter

The clock from the ARM timer is derived from the system clock. This clock can change dynamically e.g. if the system goes into reduced power or in low power mode. Thus the clock speed adapts to the overall system performance capabilities. For accurate timing it is recommended to use the system timers.

### 12.2. Timer Registers

The base address for the ARM timer register is `0x7E00 B000`.

| Address Offset | Register Name | Description                             | Size |
|----------------|---------------|---|------|
| 0x400          | LOAD          | Load                                    | 32   |
| 0x404          | VALUE         | Value (Read-Only)                       | 32   |
| 0x408          | CONTROL       | Control                                 | 32   |
| 0x40C          | IRQCNTL       | IRQ Clear/Ack (Write-Only)              | 32   |
| 0x410          | RAWIRQ        | RAW IRQ (Read-Only)                     | 32   |
| 0x414          | MSKIRQ        | Masked IRQ (Read-Only)                  | 32   |
| 0x418          | RELOAD        | Reload                                  | 32   |
| 0x41C          | PREDIV        | Pre-divider (Not in real 804!)          | 32   |
| 0x420          | FREECNT       | Free running counter (Not in real 804!) | 32   |

Table 190. Timer Registers

#### LOAD Register

##### Synopsis

The timer load register sets the time for the timer to count down. This value is loaded into the timer value register after the load register has been written or if the timer-value register has counted down to 0.



| Bit(s) | Field Name | Description       | Type | Reset |
|--------|------------|-------------------|------|-------|
| 31:0   | LOAD       | Timer load value. | RW   | 0x0   |

Table 191. Timer Load register

## VALUE Register

### Synopsis

This register holds the current timer value and is counted down when the counter is running. It is counted down each timer clock until the value 0 is reached. Then the value register is re-loaded from the timer load register and the interrupt pending bit is set. The timer count down speed is set by the timer pre-divide register.

| Bit(s) | Field Name | Description          | Type | Reset |
|--------|------------|----------------------|------|-------|
| 31:0   | VALUE      | Current timer value. | RO   | 0x0   |

Table 192. Timer Value register

## CONTROL Register

### Synopsis

The standard SP804 timer control register consists of 8 bits but in the BCM2711 implementation there are more control bits for the extra features. Control bits 0-7 are identical to the SP804 bits, albeit some functionality of the SP804 is not implemented. All new control bits start from bit 8 upwards.

Differences between a real 804 and the BCM2711 implementation are shown in italics.

| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 31:24  |            | Reserved - Write as 0, read as don't care   |      |       |
| 23:16  | FREEDIV    | <i>Free running counter pre-scaler. Freq is sys_clk/(prescale+1)</i><br><b><i>These bits do not exist in a standard 804 timer!</i></b>  | RW   | 0x3E  |
| 15:10  |            | Reserved - Write as 0, read as don't care   |      |       |
| 9      | ENAFREE    | <i>0 : Free running counter Disabled</i><br><i>1 : Free running counter Enabled</i><br><b><i>This bit does not exist in a standard 804 timer!</i></b>                                       | RW   | 0x0   |
| 8      | DBGHALT    | <i>0 : Timers keeps running if ARM is in debug halted mode</i><br><i>1 : Timers halted if ARM is in debug halted mode</i><br><b><i>This bit does not exist in a standard 804 timer!</i></b> | RW   | 0x0   |
| 7      | ENABLE     | 0 : Timer disabled<br>1 : Timer enabled   | RW   | 0x0   |
| 6      |            | Reserved. <i>The timer is always in free running mode.</i><br><i>If this bit is set it enables periodic mode in a standard 804. That mode is not supported in the BCM2711.</i>              |      |       |



| Bit(s) | Field Name   | Description   | Type | Reset |
|--------|--|---|------|-------|
| 5      | IE   | 0 : Timer interrupt disabled<br>1 : Timer interrupt enabled   | RW   | 0x1   |
| 4      | Reserved - Write as 0, read as don't care  |   |      |       |
| 3:2    | DIV  | Pre-scale bits:<br>00 : pre-scale is clock / 1 (No pre-scale)<br>01 : pre-scale is clock / 16<br>10 : pre-scale is clock / 256<br>11 : pre-scale is clock / 1 ( <i>Undefined in 804</i> ) | RW   | 0x0   |
| 1      | 32BIT  | 0 : 16-bit counters<br>1 : 32-bit counter   | RW   | 0x0   |
| 0      | Reserved. <i>The timer is always in wrapping mode.<br/>If this bit is set it enables one-shot mode in real 804. That mode is not supported in the BCM2711.</i> |   |      |       |

Table 193. Timer control register

## IRQCNTL Register

### Synopsis

The timer IRQ clear register is write-only. When writing this register the interrupt-pending bit is cleared.

When reading this register it returns 0x544D5241 which is the ASCII reversed value for "ARMT".

| Bit(s) | Field Name                                | Description                      | Type | Reset |
|--------|---|----------------------------------|------|-------|
| 31:1   | Reserved - Write as 0, read as don't care |                                  |      |       |
| 0      | INT                                       | Write a 1 to clear the interrupt | WO   | 0x0   |

Table 194. Timer IRQ clear register

## RAWIRQ Register

### Synopsis

The raw IRQ register is a read-only register. It shows the status of the interrupt pending bit.

| Bit(s) | Field Name                                | Description   | Type | Reset |
|--------|---|---|------|-------|
| 31:1   | Reserved - Write as 0, read as don't care |   |      |       |
| 0      | INT                                       | 0 : The interrupt pending bit is clear<br>1 : The interrupt pending bit is set. | RO   | 0x0   |

Table 195. Timer Raw IRQ register

The interrupt pending bit is set each time the value register is counted down to zero. The interrupt pending bit can not by itself generate interrupts. Interrupts can only be generated if the interrupt enable



bit is set.

## MSKIRQ Register

### Synopsis

The masked IRQ register is a read-only register. It shows the status of the interrupt signal. It is simply a logical AND of the interrupt pending bit and the interrupt enable bit.

| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---|------|-------|
| 31:1   |            | Reserved - Write as 0, read as don't care   |      |       |
| 0      | INT        | 0 : Interrupt line not asserted.<br>1 : Interrupt line is asserted, (the interrupt pending and the interrupt enable bit are set.) | RO   | 0x0   |

Table 196. Timer Masked IRQ register

## RELOAD Register

### Synopsis

This register is a copy of the timer load register. The difference is that a write to this register does not trigger an immediate reload of the timer value register. Instead the timer load register value is only accessed if the value register has finished counting down to zero.

| Bit(s) | Field Name | Description       | Type | Reset |
|--------|------------|-------------------|------|-------|
| 31:0   | LOAD       | Timer load value. | RW   | 0x0   |

Table 197. Timer Reload register

## PREDIV Register

| Bit(s) | Field Name | Description                               | Type | Reset |
|--------|------------|---|------|-------|
| 31:10  |            | Reserved - Write as 0, read as don't care |      |       |
| 9:0    | PREDIV     | Pre-divider value.                        | RW   | 0x07D |

Table 198. Timer pre-divider register

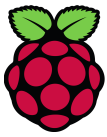
The Pre-divider register is not present in the SP804.

The pre-divider register is 10 bits wide and can be written or read from. This register has been added as the SP804 expects a 1MHz clock which we do not have. Instead the pre-divider takes the APB clock and divides it down according to:

$$timer\_clock = \frac{apb\_clock}{pre\_divider + 1}$$

The reset value of this register is 0x7D so gives a divide by 126.

## FREECNT Register



| Bit(s) | Field Name | Description   | Type | Reset |
|--------|------------|---------------|------|-------|
| 31:0   | FREECNT    | Counter value | RO   | 0x0   |

*Table 199. Free running counter*

The free running counter is not present in the SP804.

The free running counter is a 32-bit wide read-only register. The register is enabled by setting bit 9 of the Timer control register. The free running counter is incremented immediately after it is enabled. The timer can not be reset but when enabled, will always increment and roll-over. The free running counter is also running from the APB clock and has its own clock pre-divider controlled by bits 16-23 of the timer control register.

This register will be halted too if bit 8 of the control register is set and the ARM is in Debug Halt mode.



# Chapter 13. ARM Mailboxes

## 13.1. Overview

There are 16 ARM Mailboxes<sup>[12]</sup> which can be used to send messages or signals between the ARM cores. Each mailbox is a 32-bit wide value with separate write-set and write-clear registers (see [Section 6.5](#) for more information about write-set / write-clear registers), for a total of 32 registers.

There are no differences between any of the ARM mailboxes, so it is left to the programmer to decide how to use them. Mailbox bits can be set by writing to the appropriate MBOX\_SET register. Each mailbox generates an interrupt whenever any of its bits are non-zero - refer to [Chapter 6](#) for details on how these interrupts are routed. The mailbox's value can be read from the appropriate MBOX\_CLR register, and mailbox bits can be cleared by writing to the appropriate MBOX\_CLR register (these last two steps would typically be performed inside the relevant ARM core's interrupt handler).

## 13.2. Registers

The ARM\_LOCAL register base address is `0x4 C000 0000`. Note that, unlike other peripheral addresses in this document, this is an ARM-only address and not a legacy master address. If Low Peripheral mode is enabled this base address becomes `0x0 FF80 0000`.

The write-set registers (MBOX\_SET) are write-only, but the write-clear registers (MBOX\_CLR) are read-write.

| Address Offset | Register Name              | Description                 | Size |
|----------------|----------------------------|-----------------------------|------|
| 0x80           | <a href="#">MBOX_SET00</a> | Mailbox 00 Set Bit Register | 32   |
| 0x84           | <a href="#">MBOX_SET01</a> | Mailbox 01 Set Bit Register | 32   |
| 0x88           | <a href="#">MBOX_SET02</a> | Mailbox 02 Set Bit Register | 32   |
| 0x8C           | <a href="#">MBOX_SET03</a> | Mailbox 03 Set Bit Register | 32   |
| 0x90           | <a href="#">MBOX_SET04</a> | Mailbox 04 Set Bit Register | 32   |
| 0x94           | <a href="#">MBOX_SET05</a> | Mailbox 05 Set Bit Register | 32   |
| 0x98           | <a href="#">MBOX_SET06</a> | Mailbox 06 Set Bit Register | 32   |
| 0x9C           | <a href="#">MBOX_SET07</a> | Mailbox 07 Set Bit Register | 32   |
| 0xA0           | <a href="#">MBOX_SET08</a> | Mailbox 08 Set Bit Register | 32   |
| 0xA4           | <a href="#">MBOX_SET09</a> | Mailbox 09 Set Bit Register | 32   |
| 0xA8           | <a href="#">MBOX_SET10</a> | Mailbox 10 Set Bit Register | 32   |
| 0xAC           | <a href="#">MBOX_SET11</a> | Mailbox 11 Set Bit Register | 32   |
| 0xB0           | <a href="#">MBOX_SET12</a> | Mailbox 12 Set Bit Register | 32   |



| Address Offset | Register Name              | Description                   | Size |
|----------------|----------------------------|-------------------------------|------|
| 0xB4           | <a href="#">MBOX_SET13</a> | Mailbox 13 Set Bit Register   | 32   |
| 0xB8           | <a href="#">MBOX_SET14</a> | Mailbox 14 Set Bit Register   | 32   |
| 0xBC           | <a href="#">MBOX_SET15</a> | Mailbox 15 Set Bit Register   | 32   |
| 0xC0           | <a href="#">MBOX_CLR00</a> | Mailbox 00 Clear Bit Register | 32   |
| 0xC4           | <a href="#">MBOX_CLR01</a> | Mailbox 01 Clear Bit Register | 32   |
| 0xC8           | <a href="#">MBOX_CLR02</a> | Mailbox 02 Clear Bit Register | 32   |
| 0xCC           | <a href="#">MBOX_CLR03</a> | Mailbox 03 Clear Bit Register | 32   |
| 0xD0           | <a href="#">MBOX_CLR04</a> | Mailbox 04 Clear Bit Register | 32   |
| 0xD4           | <a href="#">MBOX_CLR05</a> | Mailbox 05 Clear Bit Register | 32   |
| 0xD8           | <a href="#">MBOX_CLR06</a> | Mailbox 06 Clear Bit Register | 32   |
| 0xDC           | <a href="#">MBOX_CLR07</a> | Mailbox 07 Clear Bit Register | 32   |
| 0xE0           | <a href="#">MBOX_CLR08</a> | Mailbox 08 Clear Bit Register | 32   |
| 0xE4           | <a href="#">MBOX_CLR09</a> | Mailbox 09 Clear Bit Register | 32   |
| 0xE8           | <a href="#">MBOX_CLR10</a> | Mailbox 10 Clear Bit Register | 32   |
| 0xEC           | <a href="#">MBOX_CLR11</a> | Mailbox 11 Clear Bit Register | 32   |
| 0xF0           | <a href="#">MBOX_CLR12</a> | Mailbox 12 Clear Bit Register | 32   |
| 0xF4           | <a href="#">MBOX_CLR13</a> | Mailbox 13 Clear Bit Register | 32   |
| 0xF8           | <a href="#">MBOX_CLR14</a> | Mailbox 14 Clear Bit Register | 32   |
| 0xFC           | <a href="#">MBOX_CLR15</a> | Mailbox 15 Clear Bit Register | 32   |

Table 200. ARM Mailbox registers

## MBOX\_SET00, MBOX\_SET01, MBOX\_SET02, MBOX\_SET03, MBOX\_SET04, MBOX\_SET05, MBOX\_SET06, MBOX\_SET07, MBOX\_SET08, MBOX\_SET09, MBOX\_SET10, MBOX\_SET11, MBOX\_SET12, MBOX\_SET13, MBOX\_SET14 & MBOX\_SET15 Registers

### Synopsis

Writing a '1' to a bit position in this register causes the corresponding bit in the mailbox word to be set to 1.

There are 16 mailboxes in total, four per ARM core. Mailboxes 4C to 4C+3 'belong' to core number C. Each mailbox may raise an interrupt to its core when any bits in the 32-bit word are set to '1'.

| Bit(s) | Field Name | Description         | Type | Reset |
|--------|------------|---------------------|------|-------|
| 31:00  | MBOX_DATA  | 32-bit mailbox word | WO   | 0x0   |

Table 201. MBOX\_SET00, MBOX\_SET01, MBOX\_SET02, MBOX\_SET03, MBOX\_SET04, MBOX\_SET05,



*MBOX\_SET06, MBOX\_SET07, MBOX\_SET08, MBOX\_SET09, MBOX\_SET10, MBOX\_SET11, MBOX\_SET12,  
MBOX\_SET13, MBOX\_SET14 & MBOX\_SET15 Registers*

## **MBOX\_CLR00, MBOX\_CLR01, MBOX\_CLR02, MBOX\_CLR03, MBOX\_CLR04, MBOX\_CLR05, MBOX\_CLR06, MBOX\_CLR07, MBOX\_CLR08, MBOX\_CLR09, MBOX\_CLR10, MBOX\_CLR11, MBOX\_CLR12, MBOX\_CLR13, MBOX\_CLR14 & MBOX\_CLR15 Registers**

### **Synopsis**

Writing a '1' to a bit position in this register causes the corresponding bit in the mailbox word to be cleared to 0. A read returns the current state of the mailbox word.

There are 16 mailboxes in total, four per ARM core. Mailboxes  $4C$  to  $4C+3$  'belong' to core number  $C$ . Each mailbox may raise an interrupt to its core when any bits in the 32-bit word are set to '1'.

| Bit(s) | Field Name | Description         | Type | Reset |
|--------|------------|---------------------|------|-------|
| 31:00  | MBOX_DATA  | 32-bit mailbox word | W1C  | 0x0   |

*Table 202. MBOX\_CLR00, MBOX\_CLR01, MBOX\_CLR02, MBOX\_CLR03, MBOX\_CLR04, MBOX\_CLR05,  
MBOX\_CLR06, MBOX\_CLR07, MBOX\_CLR08, MBOX\_CLR09, MBOX\_CLR10, MBOX\_CLR11, MBOX\_CLR12,  
MBOX\_CLR13, MBOX\_CLR14 & MBOX\_CLR15 Registers*

[12] Note that these are distinct from the VPU Mailboxes in the ARMC block