

partially annotated copy for cs140e, win19, engler.

when reading, always check the bcm2835 errata!



BCM2835 ARM Peripherals

© 2012 Broadcom Corporation.
All rights reserved

Broadcom Europe Ltd. 406 Science Park Milton Road Cambridge CB4 0WW



Table of Contents

1	Introduction	4
1.1	Overview	4
1.2	Address map	4
1.2.1	Diagrammatic overview	4
1.2.2	ARM virtual addresses (standard Linux kernel only)	6
1.2.3	ARM physical addresses	6
1.2.4	Bus addresses	6
1.3	Peripheral access precautions for correct memory ordering	7
2	Auxiliaries: UART1 & SPI1, SPI2	8
2.1	Overview	8
2.1.1	AUX registers	9
2.2	Mini UART	10
2.2.1	Mini UART implementation details.	11
2.2.2	Mini UART register details.	11
2.3	Universal SPI Master (2x)	20
2.3.1	SPI implementation details	20
2.3.2	Interrupts	21
2.3.3	Long bit streams	21
2.3.4	SPI register details.	22
3	BSC	28
3.1	Introduction	28
3.2	Register View	28
3.3	10 Bit Addressing	36
4	DMA Controller	38
4.1	Overview	38
4.2	DMA Controller Registers	39
4.2.1	DMA Channel Register Address Map	40
4.3	AXI Bursts	63
4.4	Error Handling	63
4.5	DMA LITE Engines	63
5	External Mass Media Controller	65
○	Introduction	65
○	Registers	66
6	General Purpose I/O (GPIO)	89
6.1	Register View	90
6.2	Alternative Function Assignments	102
6.3	General Purpose GPIO Clocks	105
7	Interrupts	109
7.1	Introduction	109
7.2	Interrupt pending.	110
7.3	Fast Interrupt (FIQ).	110
7.4	Interrupt priority.	110
7.5	Registers	112
8	PCM / I2S Audio	119
8.1	Block Diagram	120
8.2	Typical Timing	120
8.3	Operation	121
8.4	Software Operation	122
8.4.1	Operating in Polled mode	122
8.4.2	Operating in Interrupt mode	123



BCM2835 ARM Peripherals

8.4.3	DMA	123
8.5	Error Handling.	123
8.6	PDM Input Mode Operation	124
8.7	GRAY Code Input Mode Operation	124
8.8	PCM Register Map	125
9	Pulse Width Modulator	138
9.1	Overview	138
9.2	Block Diagram	138
9.3	PWM Implementation	139
9.4	Modes of Operation	139
9.5	Quick Reference	140
9.6	Control and Status Registers	141
10	SPI	148
10.1	Introduction	148
10.2	SPI Master Mode	148
10.2.1	Standard mode	148
10.2.2	Bidirectional mode	149
10.3	LoSSI mode	150
10.3.1	Command write	150
10.3.2	Parameter write	150
10.3.3	Byte read commands	151
10.3.4	24bit read command	151
10.3.5	32bit read command	151
10.4	Block Diagram	152
10.5	SPI Register Map	152
10.6	Software Operation	158
10.6.1	Polled	158
10.6.2	Interrupt	158
10.6.3	DMA	158
10.6.4	Notes	159
11	SPI/BSC SLAVE	160
11.1	Introduction	160
11.2	Registers	160
12	System Timer	172
12.1	System Timer Registers	172
13	UART	175
13.1	Variations from the 16C650 UART	175
13.2	Primary UART Inputs and Outputs	176
13.3	UART Interrupts	176
13.4	Register View	177
14	Timer (ARM side)	196
14.1	Introduction	196
14.2	Timer Registers:	196
15	USB	200
15.1	Configuration	200
15.2	Extra / Adapted registers.	202



1 Introduction

1.1 Overview

BCM2835 contains the following peripherals which may safely be accessed by the ARM:

- Timers
- Interrupt controller
- GPIO
- USB
- PCM / I2S
- DMA controller
- I2C master
- I2C / SPI slave
- SPI0, SPI1, SPI2
- PWM
- UART0, UART1

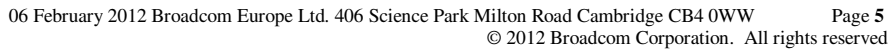
The purpose of this datasheet is to provide documentation for these peripherals in sufficient detail to allow a developer to port an operating system to BCM2835.

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

In addition to the ARM's MMU, BCM2835 includes a second coarse-grained MMU for mapping ARM physical addresses onto system bus addresses. This diagram shows the main address spaces of interest:





BCM2835 ARM Peripherals

Addresses in ARM Linux are:

- issued as virtual addresses by the ARM core, then
- mapped into a physical address by the ARM MMU, then
- mapped into a bus address by the ARM mapping MMU, and finally
- used to select the appropriate peripheral or location in RAM.

1.2.2 ARM virtual addresses (standard Linux kernel only)

As is standard practice, the standard BCM2835 Linux kernel provides a contiguous mapping over the whole of available RAM at the top of memory. The kernel is configured for a 1GB/3GB split between kernel and user-space memory.

The split between ARM and GPU memory is selected by installing one of the supplied start*.elf files as start.elf in the FAT32 boot partition of the SD card. The minimum amount of memory which can be given to the GPU is 32MB, but that will restrict the multimedia performance; for example, 32MB does not provide enough buffering for the GPU to do 1080p30 video decoding.

Virtual addresses in kernel mode will range between 0xC0000000 and 0xEFFFFFFF.

Virtual addresses in user mode (i.e. seen by processes running in ARM Linux) will range between 0x00000000 and 0xBFFFFFFF.

this is why
addresses
have a
0x2000
as upper
half.

Peripherals (at physical address 0x20000000 on) are mapped into the kernel virtual address space starting at address 0xF2000000. Thus a peripheral advertised here at bus address 0x7Ennnnnn is available in the ARM kernel at virtual address 0xF2nnnnnn.

1.2.3 ARM physical addresses

Physical addresses start at 0x00000000 for RAM.

- The ARM section of the RAM starts at 0x00000000.
- The VideoCore section of the RAM is mapped in only if the system is configured to support a memory mapped display (this is the common case).

The VideoCore MMU maps the ARM physical address space to the bus address space seen by VideoCore (and VideoCore peripherals). The bus addresses for RAM are set up to map onto the uncached¹ bus address range on the VideoCore starting at 0xC0000000.

Physical addresses range from 0x20000000 to 0x20FFFFFF for peripherals. The bus addresses for peripherals are set up to map onto the peripheral bus address range starting at 0x7E000000. Thus a peripheral advertised here at bus address 0x7Ennnnnn is available at physical address 0x20nnnnnn.

1.2.4 Bus addresses

The peripheral addresses specified in this document are bus addresses. Software directly accessing peripherals must translate these addresses into physical or virtual addresses, as described above. Software accessing peripherals using the DMA engines must use bus addresses.

¹ BCM2835 provides a 128KB system L2 cache, which is used primarily by the GPU. Accesses to memory are routed either via or around the L2 cache depending on senior two bits of the bus address.



BCM2835 ARM Peripherals

i.e., null derefs won't crash!

Software accessing RAM directly must use physical addresses (based at 0x00000000). Software accessing RAM using the DMA engines must use bus addresses (based at 0xC0000000).

1.3 Peripheral access precautions for correct memory ordering

The BCM2835 system uses an AMBA AXI-compatible interface structure. In order to keep the system complexity low and data throughput high, the BCM2835 AXI system does not always return read data in-order². 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 so you should 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.

ignoring this advice will almost always work. until it does not. i once wasted three days b/c there was not a mem barrier when setting up the miniUART.

however, the "bare metal" forums don't have a lot of clarity on this point: why *exactly* it is needed and when. safest is to dmb/dsb.

²Normally 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 theoretical 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.



BCM2835 ARM Peripherals

2 Auxiliaries: UART1 & SPI1, SPI2

2.1 Overview

The Device has three Auxiliary peripherals: One mini UART and two SPI masters. These three peripheral 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.

Auxiliary peripherals Register Map (offset = 0x7E21 5000)			
Address	Register Name ³	Description	Size
0x7E21 5000	AUX_IRQ	Auxiliary Interrupt status	3
0x7E21 5004	AUX_ENABLES	Auxiliary enables	3
0x7E21 5040	AUX_MU_IO_REG	Mini Uart I/O Data	8
0x7E21 5044	AUX_MU_IER_REG	Mini Uart Interrupt Enable	8
0x7E21 5048	AUX_MU_IIR_REG	Mini Uart Interrupt Identify	8
0x7E21 504C	AUX_MU_LCR_REG	Mini Uart Line Control	8
0x7E21 5050	AUX_MU_MCR_REG	Mini Uart Modem Control	8
0x7E21 5054	AUX_MU_LSR_REG	Mini Uart Line Status	8
0x7E21 5058	AUX_MU_MSR_REG	Mini Uart Modem Status	8
0x7E21 505C	AUX_MU_SCRATCH	Mini Uart Scratch	8
0x7E21 5060	AUX_MU_CNTL_REG	Mini Uart Extra Control	8
0x7E21 5064	AUX_MU_STAT_REG	Mini Uart Extra Status	32
0x7E21 5068	AUX_MU_BAUD_REG	Mini Uart Baudrate	16
0x7E21 5080	AUX_SPI0_CNTL0_REG	SPI 1 Control register 0	32
0x7E21 5084	AUX_SPI0_CNTL1_REG	SPI 1 Control register 1	8
0x7E21 5088	AUX_SPI0_STAT_REG	SPI 1 Status	32
0x7E21 5090	AUX_SPI0_IO_REG	SPI 1 Data	32
0x7E21 5094	AUX_SPI0_PEEK_REG	SPI 1 Peek	16
0x7E21 50C0	AUX_SPI1_CNTL0_REG	SPI 2 Control register 0	32
0x7E21 50C4	AUX_SPI1_CNTL1_REG	SPI 2 Control register 1	8

³ These register names are identical to the defines in the AUX_IO header file. For programming purposes these names should be used wherever possible.



BCM2835 ARM Peripherals

0x7E21 50C8	AUX_SPI1_STAT_REG	SPI 2 Status	32
0x7E21 50D0	AUX_SPI1_IO_REG	SPI 2 Data	32
0x7E21 50D4	AUX_SPI1_PEEK_REG	SPI 2 Peek	16

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.

AUXIRQ Register (0x7E21 5000)

SYNOPSIS The AUXIRQ 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.	R	0
1	SPI 1 IRQ	If set the SPI1 module has an interrupt pending.	R	0
0	Mini UART IRQ	If set the mini UART has an interrupt pending.	R	0

AUXENB Register (0x7E21 5004)

SYNOPSIS The AUXENB register is used to enable the three modules; UART, 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. If clear the SPI 2 module is disabled. That also disables any SPI 2 module register access	R/W	0
1	SPI 1 enable	If set the SPI 1 module is enabled. If clear the SPI 1 module is disabled. That also disables any SPI 1 module register access	R/W	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 low. If clear the mini UART is disabled. That also disables any mini UART register access	R/W	0



BCM2835 ARM Peripherals

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 the before enabling the UART. The UART core is build 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 stop bit and is only used so there is clean timing start for the next bit.

Looking after a reset: the baudrate will be zero and the system clock will be 250 MHz. So only 2.5 μ seconds suffice to fill the receive FIFO. The result will be that the FIFO is full and overflowing in no time flat.

2.2 Mini UART

The mini UART is a secondary low throughput⁴ 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.

The mini Uart has the following features:

- 7 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.

dwelch67 does not.

fixed sized buffer, small.
better be prompt on rx.
and check before xmit.

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.

⁴ 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.

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 (0x7E21 5040)

SYNOPSIS The AUX_MU_IO_REG register is primary 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)

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

same: read to recv (must check if data there first), write to xmit (must check if space first).



errata. is MU_IER_REG

BCM2835 ARM Peripherals

AUX_MU_IER_REG Register (0x7E21 5044)

SYNOPSIS The AUX_MU_IER_REG register is primary 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)

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



BCM2835 ARM Peripherals

AUX_MU_IIR_REG Register (0x7E21 5048)

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	R	11
5:4	-	Always read as zero	R	00
3	-	Always read as zero as the mini UART has no timeout function	R	0
2:1	READ: Interrupt ID bits WRITE: FIFO clear bits	On read this register shows the interrupt ID bit 00 : No interrupts 01 : Transmit holding register empty 10 : Receiver holds valid byte 11 : <Not possible> On write: Writing with bit 1 set will clear the receive FIFO Writing with bit 2 set will clear the transmit FIFO	R/W	00
0	Interrupt pending	This bit is clear whenever an interrupt is pending	R	1



BCM2835 ARM Peripherals

AUX_MU_LCR_REG Register (0x7E21 504C)

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 to Mini UART register give access the Baudrate register. During operation this bit must be cleared.	R/W	0
6	Break	If set high the UART1_TX line is pulled low continuously. If held for at least 12 bits times that will indicate a break condition.	R/W	0
5:1		Reserved, write zero, read as don't care <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 If set the UART works in 8-bit mode	R/W	0

don't use.
use baud
reg.

huge errata: <data size> is a two bit field! write 0b11 for 8bit. dwelch67 figured out.

AUX_MU_MCR_REG Register (0x7E21 5050)

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 <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 If set the UART1_RTS line is low This bit is ignored if the RTS is used for auto-flow control. See the Mini Uart Extra Control register description)	R/W	0
0		Reserved, write zero, read as don't care <i>This bit has a function in a 16550 compatible UART but is ignored here</i>		0

we don't
need this.



BCM2835 ARM Peripherals

AUX_MU_LSR_REG Register (0x7E21 5054)

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 <i>This bit has a function in a 16550 compatible UART but is ignored here</i>		0
6	Transmitter idle	This bit is set if the transmit FIFO is empty and the transmitter is idle. (Finished shifting out the last bit).	R	1
5	Transmitter empty	This bit is set if the transmit FIFO can accept at least one byte.	R	0
4:2		Reserved, write zero, read as don't care <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.	R/C	0
0	Data ready	This bit is set if the receive FIFO holds at least 1 symbol.	R	0

AUX_MU_MSR_REG Register (0x7E21 5058)

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		
7:6		Reserved, write zero, read as don't care <i>Some of these bits have functions in a 16550 compatible UART but are ignored here</i>		0
5	CTS status	This bit is the inverse of the UART1_CTS input Thus : If set the UART1_CTS pin is low If clear the UART1_CTS pin is high	R	1
3:0		Reserved, write zero, read as don't care <i>Some of these bits have functions in a 16550 compatible UART but are ignored here</i>		0



BCM2835 ARM Peripherals

AUX_MU_SCRATCH Register (0x7E21 505C)

SYNOPSIS The AUX_MU_SCRATCH is a single byte storage.

Bit(s)	Field Name	Description	Type	Reset
31:8		Reserved, write zero, read as don't care		
7:0	Scratch	One whole byte extra on top of the 134217728 provided by the SDC	R/W	0

AUX_MU_CNTL_REG Register (0x7E21 5060)

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. If set the CTS auto flow assert level is low* If clear the CTS auto flow assert level is high*	R/W	0
6	RTS assert level	This bit allows one to invert the RTS auto flow operation polarity. If set the RTS auto flow assert level is low* If clear the RTS auto flow assert level is high*	R/W	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. 00 : De-assert RTS when the receive FIFO has 3 empty spaces left. 01 : De-assert RTS when the receive FIFO has 2 empty spaces left. 10 : De-assert RTS when the receive FIFO has 1 empty space left. 11 : De-assert RTS when the receive FIFO has 4 empty spaces left.	R/W	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. If this bit is clear the transmitter will ignore the status of the CTS line	R/W	0



BCM2835 ARM Peripherals

2	Enable receive Auto flow-control using RTS	If this bit is set the RTS line will de-assert if the receive FIFO reaches it 'auto flow' level. In fact the RTS line will behave as an RTR (<i>Ready To Receive</i>) line. If this bit is clear the RTS line is controlled by the AUX_MU_MCR_REG register bit 1.	R/W	0
1	Transmitter enable	If this bit is set the mini UART transmitter is enabled. If this bit is clear the mini UART transmitter is disabled	R/W	1
0	Receiver enable	If this bit is set the mini UART receiver is enabled. If this bit is clear the mini UART receiver is disabled	R/W	1

do last. for hygiene, also set to 00 first thing.

Receiver enable

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

Transmitter enable

If this bit is set no new symbols will be send 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-asserted 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.



BCM2835 ARM Peripherals


AUX_MU_STAT_REG Register (0x7E21 5064)

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 The value is in the range 0-8	R	0
23:20		Reserved, write zero, read as don't care		
19:16	Receive FIFO fill level	These bits shows how many symbols are stored in the receive FIFO The value is in the range 0-8	R	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. It is a logic AND of bits 2 and 8	R	1
8	Transmit FIFO is empty	If this bit is set the transmitter FIFO is empty. Thus it can accept 8 symbols.	R	1
7	CTS line	This bit shows the status of the UART1_CTS line.	R	0
6	RTS status	This bit shows the status of the UART1_RTS line.	R	0
5	Transmit FIFO is full	This is the inverse of bit 1	R	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.	R	0
3	Transmitter is idle	If this bit is set the transmitter is idle. If this bit is clear the transmitter is idle.	R	1
2	Receiver is idle	If this bit is set the receiver is idle. If this bit is clear the receiver is busy. This bit can change unless the receiver is disabled	R	1
1	Space available	If this bit is set the mini UART transmitter FIFO can accept at least one more symbol. If this bit is clear the mini UART transmitter FIFO is full	R	0




BCM2835 ARM Peripherals



0	Symbol available	If this bit is set the mini UART receive FIFO contains at least 1 symbol If this bit is clear the mini UART receiver FIFO is empty	R	0
---	------------------	---	---	---

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 Register (0x7E21 5068)

SYNOPSIS The AUX_MU_BAUD 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	R/W	0

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



BCM2835 ARM Peripherals

6.2 Alternative Function Assignments

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

	Pull	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5
GPIO0	High	SDA0	SA5	<reserved>			
GPIO1	High	SCL0	SA4	<reserved>			
GPIO2	High	SDA1	SA3	<reserved>			
GPIO3	High	SCL1	SA2	<reserved>			
GPIO4	High	GPCLK0	SA1	<reserved>			ARM_TDI
GPIO5	High	GPCLK1	SA0	<reserved>			ARM_TDO
GPIO6	High	GPCLK2	SOE_N / SE	<reserved>			ARM_RTCK
GPIO7	High	SPI0_CE1_N	SWE_N / SBW_N	<reserved>			
GPIO8	High	SPI0_CE0_N	SD0	<reserved>			
GPIO9	Low	SPI0_MISO	SD1	<reserved>			
GPIO10	Low	SPI0_MOSI	SD2	<reserved>			
GPIO11	Low	SPI0_SCLK	SD3	<reserved>			
GPIO12	Low	PWM0	SD4	<reserved>			ARM_TMS
GPIO13	Low	PWM1	SD5	<reserved>			ARM_TCK
GPIO14	Low	TXD0	SD6	<reserved>			TXD1
GPIO15	Low	RXD0	SD7	<reserved>			RXD1
GPIO16	Low	<reserved>	SD8	<reserved>	CTS0	SPI1_CE2_N	CTS1
GPIO17	Low	<reserved>	SD9	<reserved>	RTS0	SPI1_CE1_N	RTS1
GPIO18	Low	PCM_CLK	SD10	<reserved>	BSCSL SDA / MOSI	SPI1_CE0_N	PWM0
GPIO19	Low	PCM_FS	SD11	<reserved>	BSCSL SCL / SCLK	SPI1_MISO	PWM1
GPIO20	Low	PCM_DIN	SD12	<reserved>	BSCSL / MISO	SPI1_MOSI	GPCLK0
GPIO21	Low	PCM_DOUT	SD13	<reserved>	BSCSL / CE_N	SPI1_SCLK	GPCLK1
GPIO22	Low	<reserved>	SD14	<reserved>	SD1_CLK	ARM_TRST	
GPIO23	Low	<reserved>	SD15	<reserved>	SD1_CMD	ARM_RTCK	
GPIO24	Low	<reserved>	SD16	<reserved>	SD1_DAT0	ARM_TDO	
GPIO25	Low	<reserved>	SD17	<reserved>	SD1_DAT1	ARM_TCK	
GPIO26	Low	<reserved>	<reserved>	<reserved>	SD1_DAT2	ARM_TDI	
GPIO27	Low	<reserved>	<reserved>	<reserved>	SD1_DAT3	ARM_TMS	
GPIO28	-	SDA0	SA5	PCM_CLK	<reserved>		
GPIO29	-	SCL0	SA4	PCM_FS	<reserved>		
GPIO30	Low	<reserved>	SA3	PCM_DIN	CTS0		CTS1
GPIO31	Low	<reserved>	SA2	PCM_DOUT	RTS0		RTS1
GPIO32	Low	GPCLK0	SA1	<reserved>	TXD0		TXD1
GPIO33	Low	<reserved>	SA0	<reserved>	RXD0		RXD1
GPIO34	High	GPCLK0	SOE_N / SE	<reserved>	<reserved>		
GPIO35	High	SPI0_CE1_N	SWE_N / SBW_N		<reserved>		
GPIO36	High	SPI0_CE0_N	SD0	TXD0	<reserved>		
GPIO37	Low	SPI0_MISO	SD1	RXD0	<reserved>		
GPIO38	Low	SPI0_MOSI	SD2	RTS0	<reserved>		
GPIO39	Low	SPI0_SCLK	SD3	CTS0	<reserved>		
GPIO40	Low	PWM0	SD4		<reserved>	SPI2_MISO	TXD1
	Pull	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5