BCM2835 datasheet errata

From eLinux.org

Back to RPi_Hardware#Components

Contents

BCM 2835 datasheet errata

The (partial) datasheet was published here: at raspberrypi.org (http://www.raspberrypi.org/wp-content/uploads/2012/02/BCM2835-ARM-Peripherals.pdf) and a mirror (http://dmkenr5gtnd8f.cloudfront.net/wp-content/uploads/2012/02/BCM2835-ARM-Peripherals.pdf)

It has a couple of typos. Some more serious than others.

Let's gather those datasheet typos and errors here.

The quality of the datasheet is high. It looks like it contains the information that programmers need. Some of the tables from the datasheet have been reproduced here.

It also "does the right thing" with reserved bits. Many datasheets specify "write: don't care, read as zeroes". Broadcom specifies the reserved bits the other way around: "Write zeroes, read: don't care".

This is the correct way to do it. If you expand the hardware the hardware may be enhanced and do "different things" if you write ones to the previously "reserved" bits. If you follow the datasheet, and write zeroes as specified to the reserved bits, the hardware guys can make sure you're not going to run into surprises. And by specifying "read: don't care" you can allow future hardware to provide status bits there.

the BCM2835 on the RPI

Not really an erratum, but not worth it to make a whole page for this. The Raspberry Pi runs the BCM2835 with a core clock of 250MHz. This is relevant for the peripheral modules like I2C, SPI and Timer (ARM side) for calculating the desired clock rate. The I2C section on page 34 mentions 150MHz as a "nominal core clock".

PDF Generation

Switch on option for linking, so cross-references and table of contents can be jumped through. (RPi_BCM2835_GPIOs contains some useful cross-references)

p7 footnote typo

precuations should be precautions.

p8

The register names are AUX_SPI0_.... and AUX_SPI1_.... . The "description" is then SPI 1 ... and SPI 2 ....

These modules are in fact SPI1 and SPI2, and NOT SPI0 and SPI1. This is confusing as indeed there is a different module called SPI0 (documented on page 148 and onwards).

p10

It says "Al 16550 register bits" when it should say "All 16550 register bits"

p12

The title should be AUX_MU_IER_REG. The name in the Synopsis is correct

These are R/W bits not read only

Bits 3:2 are marked as don't care, but are actually required in order to receive interrupts.

Bits 1:0 are swaped. bit 0 is receive interrupt and bit 1 is transmit.

p13

The title should be AUX_MU_IIR_REG. The name in the Synopsis is correct

p14

LCR register, bit 1 must be set for 8 bit mode, like a 16550 write a 3 to get 8-bit mode

p17

The receiver enable and transmitter enable sections don't match the description in the AUX_MU_CNTL_REG table. They should both read "If this bit **cleared** no new symbols will be..."

p25 table

"AUX is IDLE:" should read "SPI is idle". (and for consistency below the line "Interrupts", SPI is Idle should be spelled "SPI is idle".

The table on page 25 has the bit numbers wrong. Some bits are mentioned twice, some not at all.

I'm guessing it should be something like:

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

The register reads as 0x280 after reset. Two bits high would be consistent with TX empty and RX empty. However, bits 7 and 9 does not match the original datasheet, nor my guess.....

p26

The Peek register is documented here as being at 0x7e21508c, whereas the table on page 8 shows 0x7e215094.

The IO register is documented as 0x7e2150a0 (with automatic deassert) and 0x7e2150b0, whereas the table on page 8 shows 0x7e215090.

p34 off-by-one?

I strongly suspect that the CDIV counter is only 14 bits wide. The bottom bit doesn't work as per specifications, and because the "0" results in 32768, the top bit doesn't either. An easy implementation would implement the 0 value as the maximum divisor. Not as "half the maximum".

Another option is that SCL = cor clock * 2 / CDIV and that the counter is 15 bits after all. (only the lowest bit missing).

p35 I2C clock stretching

There is a bug in the I2C master that it does not support clock stretching at arbitrary points. It does support clock stretching in I2C-Reads directly after the ACK phase, if the stretching-delay is >= 0.5 clock period (i.e. the low-period is >= 1 clock period).

See: http://www.raspberrypi.org/phpBB3/viewtopic.php?f=44&t=13771

A detailed analysis of this bug can be found at http://www.advamation.com/knowhow/raspberrypi/rpi-i2c-bug.html

p38 typo

harware instead of hardware (second paragraph)

p52 error

INTEN: a DMA interrupts only get triggered when the control block requesting the interrupt is the last in the chain (so next is NULL).

Otherwise no interrupt gets triggered.

Workaround is to use a second DMA chain just for triggering the interrupt.

To do this one needs to create a control-block (possibly length=0) on the second DMA channel. Link to it via two control blocks on the primary chain. The first control block sets the control-block_address of the "second DMA". The second starts the "unused" DMA by writing 1 to CS of the "second DMA", which then starts the "second DMA" and subsequently the interrupt gets triggered. After this the "primary" DMA can continue its work with other transfers without further interaction by the CPU.

p92 to 95 & 102 to 103

The GPIO Alternate function select Registers. This shows a bit pattern of 111 as alternative function 3. If you look at the values after boot up the Pi's SPI interface pins connected to the SD card have this value in them. This does not match the diagram on page 102 - 103 which shows this function is selected with alternative function 4. So either the bit pattern / function information is wrong or Table 6-31 is wrong.

p90

address 0x7e20 0000 is listed twice as gpfsel0.

p95

There is a spurious) in "..clear registers) are used..."

p96

Instead of "1 = Set GPIO pin n" for GPIO Output Clear Register 1, it should be "1 = Clear GPIO pin n"

The level registers have "0 = GPIO pin n is HIGH" which should be "1 = GPIO pin n is HIGH"

The level registers are listed as R/W but the table on page 90 states that they are only readable, which makes more sense.

p103 typo

In the "BSCSL SDA / MOSI" row of the "Special function legend" table, it says "salve" instead of "slave"

Where it says "BSC ISP slave" it should actually say "BSC/SPI slave"

p104 table

The table, legend for table 6-31, started on page 103 shows twice in red: TXD0, RXD0, CTS0, RTS0. The second block, with functions starting: UART 1 should be: TXD1, RXD1, CTS1, RTS1.

The table, legend for table 6-31, has no mention of any of the SD1_x functions included in the previous table

p105 table

In table 6-32 the values in columns "min output freq" and "max output freq" should be in each others. That is the values in column "min output freq" are the maximum output frequency values and the values in column "max output freq" are the minimum output frequency values [check: source/(DIVI) > source/(DIVI+1) as dividing by a larger value yields a smaller result!].

p111 spelling

top line: device should be devise.

p119 I2S clock

Allusions to the *APB clock domain* are made. However the exact speed of the APB clock is never explained.

http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0301h/Cbaifdib.html explains "APB" as the Advanced Peripheral Bus, but that doesn't help much either.

p126

There is amiguity on what register bits can be modified while the I2S system is active. p126 states that only the bottom three bits of CS_A can be modified. However p122 suggests to use the SYNC bit in CS_A for waiting. Does this mean, that the SYNC bit can also be changed at runtime as well?

p141

There is no base address listed for this register block. Should be: 0x7E20C000

p142/143 register table

The CTL register controls both PWM channels 1 and 2, but in the description column, only channel 1 is mentioned. Bits 8 to 15, PWEN2 to MSEN2 should be described as controlling PWM channel 2, not 1. (I think- not confirmed)

p153

Typo: wrirng -> writing.

Near the bottom of the page RXR. This bit would be useful if it signified more than half full. There is a space in "(full)" that would hint at that the word "half" was taken away. Another hint is that it says that the bit clears when "sufficient" data is read from the FIFO. The word sufficient is redundant when this is the "full and active" bit.

The way it is written now, this bit is just the same as bit RXF, except that the TA bit is anded into this one.

Two options. The hardware was changed (detecting "half full" was difficult?) and the documentation was changed accordingly. Or the hardware does what I expect: set this bit when more than half full.

p155

There is a CSPOL bit described here, wereas there are also CSPOL[0,1,2] described on page 153. How do these combine???

If CLEAR_TX_FIFO is set there is at least a slight chance (~4%) that a selected CS will get de-asserted for a short period of time (<63ns). This may happen every time this bit is set, but it is not measurable every time when sampling at 16MHz (higher sampling speeds would be needed to confirm that). Testing shows that CLEAR_RX_FIFO set alone does not exhibit such behavior.

Potential workaround is to assign CS as the "reserved" chip_select 3 and toggling the corresponding CSPOL pin for the cs that is really in use.

```
# assume the cs you want to set is in cs
cs^=(1<<(cs&0x03+21);
cs|=0x03|CLEAR_FIFO_RX|CLEAR_FIFO_TX;
writel(cs,0x7E204000);
```

It is not fully confirmed, if this does not introduce other glitches...

p156

The CDIV value is documented as "must be a power of 2". This is not true. I can perfectly set the register to 833 (not a power of 2) and get 250MHz/833 = 300kHz.

Maybe "must be a multiple of 2" was meant???

p158

SPI_FIFO is misspelled as SPIFIFO in the last line of 10.6.2.

10.6.3: The presented flow works in principle. But there are some limitations:

- ADCS is required to be set (not optional as per j)- otherwise no SPI-Clock shows on pins
- also the "described" first 4 byte of the transfer described in l-i) also impact the other bits of CS besides the 8 bit that can get set with this transfer - so the CSPolarity for all 3 chip selects get reset to 0.
- TX/RX fifos have to be reset if there was an earlier transfer of say 3 bytes in length via DMA.

In this example there is still 1 byte left which will get sent first.

The way around this is to configure CS, CDIV and DLAN via explicit DMA control blocks and only then do the RX/TX DMAs (always making use of the WAIT_RESPONSE flag in the DMA config register!). Note that using TDMODE for setting LEN and CS in one transfer may result in a reordering of the writes, which will start the SPI device before the lenght is set, so the original register value is used. Note also that it is NOT recommended resetting FIFO while starting SPI by setting TA Flag in SPI_CS. Under rare situations this may result in "lost" clocks while MOSI still shifts out the data! (so resetting FIFO needs another DMA control-block prior to setting DLEN and CS)

p160

The registers base addresses -> the registers' base address or The base address of the registers is 0x7e21_4000.

p175

top line. Two typos:

```
On mini UART and and PL011 UART
```

should be

```
One mini UART and one PL011 UART
```

p177

13.4 register view

First line:

```
The PL011 USRT is mapped on base adderss
```

should be:

```
The PL011 UART is mapped on base address
```

The base address is listed as 0x7e20100 . This should be 0x7e201000.

p196

SP804 not AP804

p197

bit 1, 32 bit counter not 23 bit counter

p198

neither are register 0x40C raw is 0x410, masked is 0x414

p202

The last entry of section 15.1.

The entries in the table should specify the choice that Broadcom made when instantiating the USB controller IP from Synopsys.

Possibly the "choice" hasn't been specified. Or maybe the "0=32, 1..5=512, 6,7=768" is the option that was chosen?

Retrieved from "http://elinux.org/index.php?title=BCM2835_datasheet_errata&oldid=299678"
Category: BCM2835
