作者

彭东林
pengdonglin137@163.com

平台

TQ2440
内核Linux4.9

概述

一直想抽时间学习一下DMA驱动，今天就以S3C2440为例，这款芯片的DMA控制器足够简单，也比较有代表性，弄懂一款，基本就对Linux的DMA子系统有一个认识了，再看其他的DMA控制器也就容易一些了。

参考博客

Linux DMA Engine framework(1)_概述
Linux DMA Engine framework(2)_功能介绍及解接口分析
Linux DMA Engine framework(3)_dma controller驱动

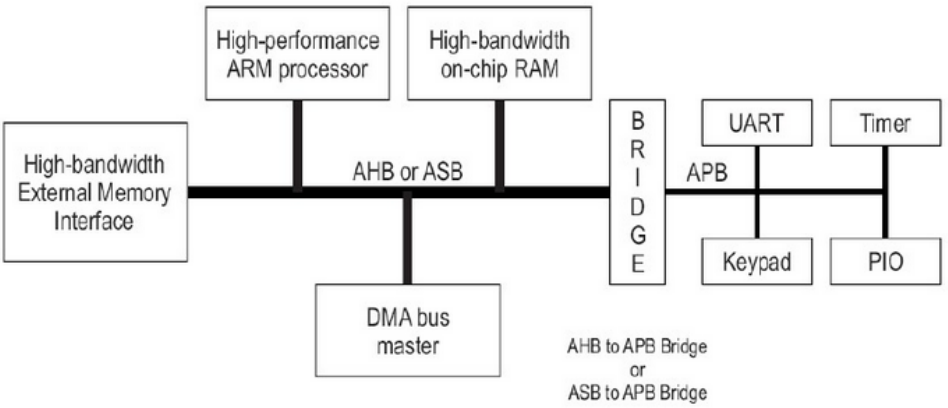
上面三篇博客来自**蜗窝科技**，对DMA子系统说的还是比较清楚的， 应该认真看看。
我打算分下面几个部分分析一下：
1、S3C2440的DMA控制器的寄存器手册
2、不使用Linux的DMA子系统提供的API的情况下的一个MEM2MEM的DMA设备驱动
3、DMA控制器驱动分析
4、利用DMA子系统提供的API写一个MEM2MEM的DMA设备驱动

下面开始第一部分， 学习一下2440跟DMA控制器相关的芯片手册。

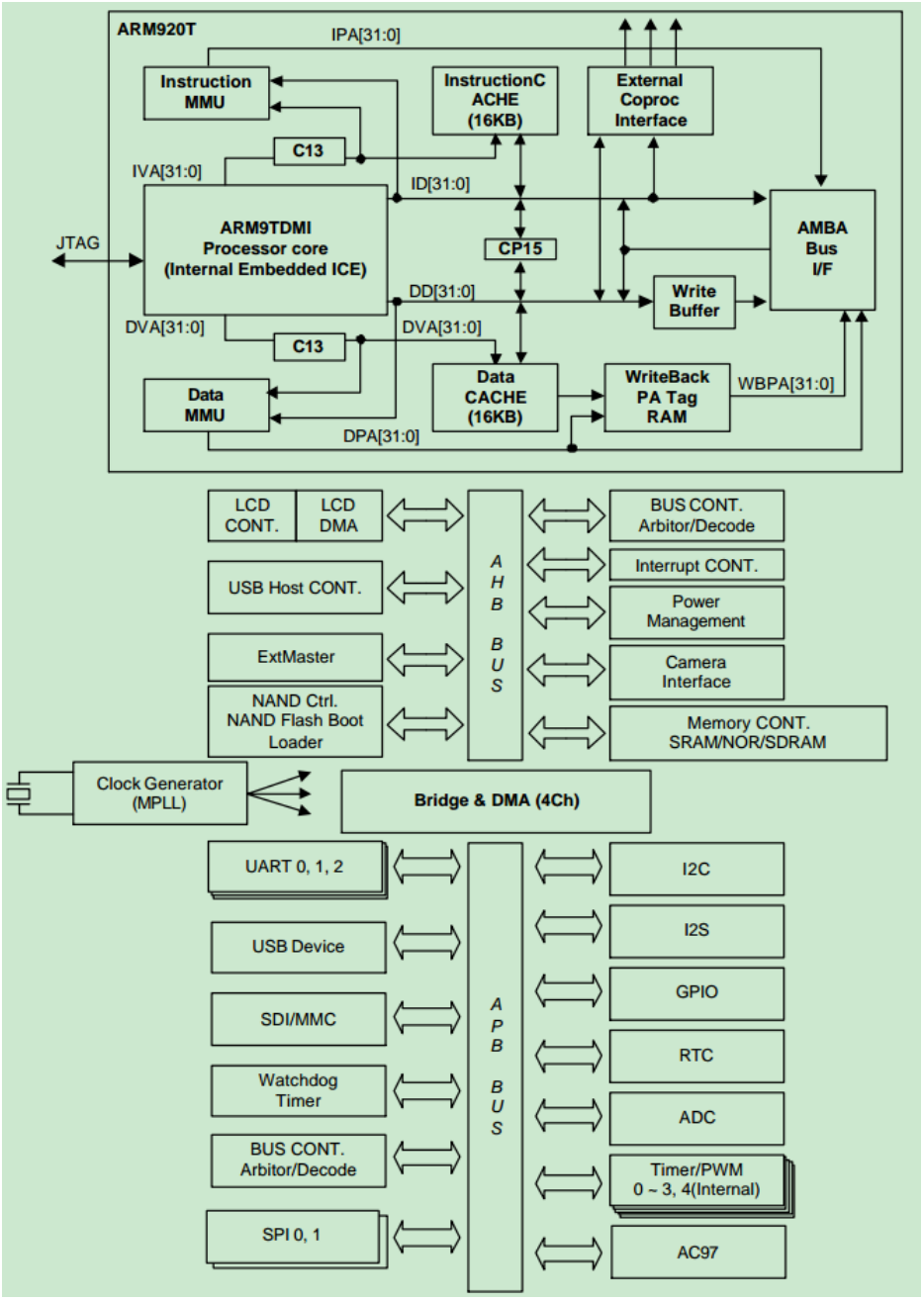
正文

一、 APB和AHB

在AHB上面挂的是一些高速的外设，使用的是HCLK，而在APB上面挂的是一些低速外设，使用的是PCLK。
二者都属于AMBA总线架构，AHB跟APB之间通过Bridge连接，一个典型的连接如下：



对于S3C2440,它的内部总线连接如下：



二、中断资源

阅读中断控制器一节，搞清楚DMA控制器占用的中断资源，这个在填写设备树的时候会用到：

INT_SDI	21	EINT8_23
INT_DMA3	20	EINT4_7
INT_DMA2	19	EINT3
INT_DMA1	18	EINT2
INT_DMA0	17	EINT1
INT_LCD	16	EINT0

S3C2440提供了四个硬件DMA，使用的是主中断。

三、DMA控制器

寄存器起始地址：0x4B000000， 长度：0x100
总共有四组DMA传输通道，每一组DMA都支持四种传输类型：

- MEM TO MEM
- MEM TO DEV
- DEV TO MEM

- DEV TO DEV

1、DMA请求源

DMA请求可以来自片内外设，也可以来自片外外设（SoC外面的设备）如下：

DMA REQUEST SOURCES

Each channel of the DMA controller can select one of the DMA request source among four DMA sources, if H/W DMA request mode is selected by DCON register. (Note that if S/W request mode is selected, this DMA request sources have no meaning at all.) Table 8-1 shows four DMA sources for each channel.

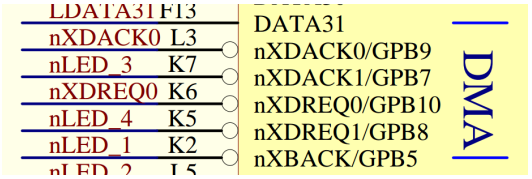
Table 8-1. DMA Request Sources for Each Channel

	Source0	Source1	Source2	Source3	Source4	Source5	Source6
Ch-0	nXDREQ0	UART0	SDI	Timer	USB device EP1	I2SSDO	PCMIN
Ch-1	nXDREQ1	UART1	I2SSDI	SPI0	USB device EP2	PCMOUT	SDI
Ch-2	I2SSDO	I2SSDI	SDI	Timer	USB device EP3	PCMIN	MICIN
Ch-3	UART2	SDI	SPI1	Timer	USB device EP4	MICIN	PCMOUT

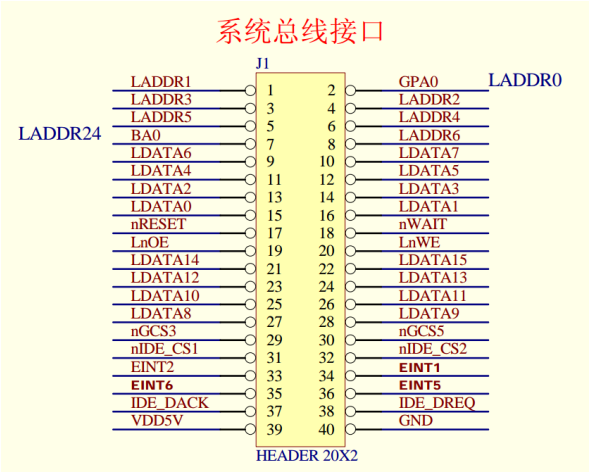
Here, nXDREQ0 and nXDREQ1 represent two external sources (External Devices), and I2SSDO and I2SSDI represent IIS transmitting and receiving, respectively.

从上面这张图可以知道：

- 表格中的是DMA请求信号，用于外设跟内存之间传输数据，对应的是触发方式必须是硬件触发
- 如果选择的是软件触发的话，那么表中的这些请求信号就没啥用了，比如后面我们要写一个MEM2MEM的demo，就需要软件触发，此时触发源字段随意填写
- 使用硬件触发的话，同一个触发源可能在多个DMA通道上都支持，但是传输只能占用一个DMA通道
- nXDREQ0/1在SoC上面有对应的GPIO引脚，意思是这两个触发源来自片外外设，而其他的都来自片内外设，下面是核心板的原理图，可以看到，tq2440上面只是将nXDREQ0跟nXDACK0引了出来，一直接到了底版上面，从说明看，应该是要跟外部IDE设备之间用DMA传输数据，比如外部IDE设备利用nXDREQ0产生DMA请求信号，然后DMA产生应答信号nXDACK0,然后就可以利用地址总线和数据总线互传数据了。



核心板



底板

- I2SD0和I2SDI分别表示I2S的playback和capture方向的DMA请求信号

2、DMA状态机

DMA uses three-state **FSM** (Finite State Machine) for its operation, which is described in the three following steps:

State-1. As an initial state, the DMA waits for a DMA request. Once the request is reached it goes to state-2. At this state, DMA ACK and INT REQ are 0.

State-2. In this state, DMA ACK becomes 1 and the counter (CURR_TC) is loaded from DCON[19:0] register. Note that the DMA ACK remains 1 until it is cleared later.

State-3. In this state, sub-FSM which handles the atomic operation of DMA is initiated. The sub-FSM reads the data from the source address and then writes it to destination address. In this operation, data size and transfer size (single or burst) are considered. This operation is repeated until the counter (CURR_TC) becomes 0 in Whole service mode, while performed only once in Single service mode. The main FSM (this FSM) counts down the CURR_TC when the sub-FSM finishes each of atomic operation. In addition, this main FSM asserts the INT REQ signal when CURR_TC becomes 0 and the interrupt setting of DCON[29] register is set to 1. In addition, it clears DMA ACK .if one of the following conditions is met.

1) CURR_TC becomes 0 in the Whole service mode
2) Atomic operation finishes in the Single service mode.

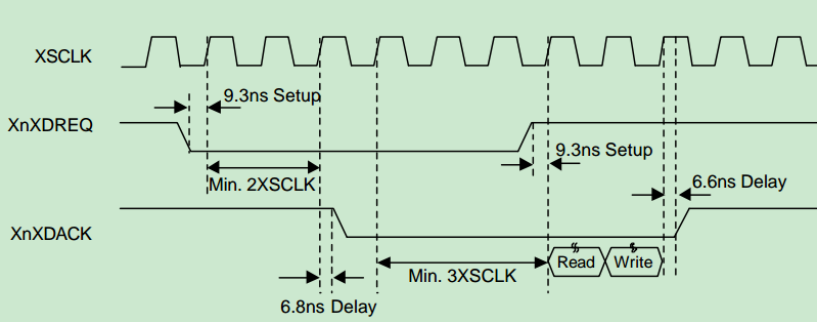
Note that in the Single service mode, these three states of main FSM are performed and then stops, and wait for another DMA REQ. And if DMA REQ comes in, all three states are repeated. Therefore, DMA ACK is asserted and then de-asserted for each atomic transfer. In contrast, in the Whole service mode, main FSM waits at state-3 until CURR_TC becomes 0. Therefore, DMA ACK is asserted during all the transfers and then de-asserted when TC reaches 0.

However, INT REQ is asserted only if CURR_TC becomes 0 regardless of the service mode (Single service mode or Whole service mode).

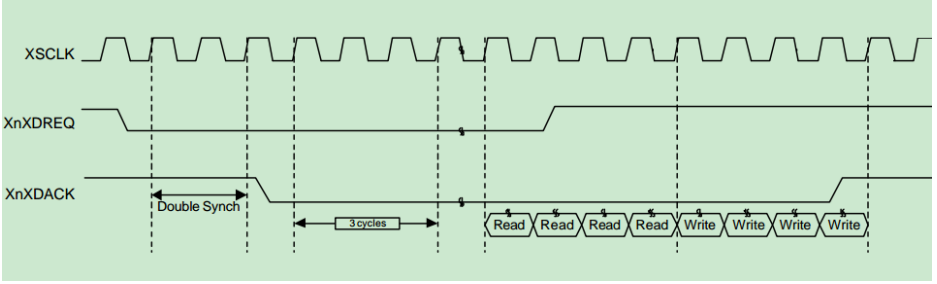
从上面的描述中可以知道：
每一个DMA通道都使用了一个三态有限状态机：
状态1： 这是初始状态，此时DMA等待外设的DMA请求。一旦收到DMA请求，就会进入状态2。在状态1的时候， DMA ACK（应答）和INT REQ（中断请求）都是0，也就是既没有DMA应答，也没有中断触发。
状态2： 此时，ACK变成1（注意区分跟nXDACK0的不同，nXDACK0为低电平表示应答）， 表示DMA应答(前面外设申请了DMA请求，此时DMA当然需要应答)， 然后，DCON[19:0]中的值被装载到CURR_TC寄存器中，表示需要进行DMA传输的次数（注意：是次数，不是需要传输的数据的总大小）
状态3： 在DMA获得系统总线后，进入该状态。负责**DMA原子传输（连续的若干个读写操作）**的sub-FSM被初始化，sub-FSM从源地址读取数据（read），然后把它写到目的地址（write）， 这个读写过程需要考虑两个因素： 每次DMA原子传输需要read/write的个数，以及每个read/wirte需要读/写多少byte数据。前者我们称之为transfer size（支持unit或burst4），后者我们称之为data size（或称为数据宽度），这两个参数都有对应的寄存器位。每一次DMA原子传输结束后，CURR_TC会减1.如果DMA通道被配置为Whole service模式，前面的DMA原子传输会一直进行，直到CURR_TC变成0，然后切回到状态1，而如果DMA被配置为Single Service模式的话，则每一次DMA原子传输后都会切到状态1，等待下一次DMA请求。如果DCON[29]被设置为1的话，当CURR_TC减到0的时候（不管是Whole还是Single Service模式），该DMA通道的中断请求就会触发。 对于DMA ACK信号的清除（置0），需要满下面的任意一个条件：第一个是在Whole Service模式下CURR_TC减到0，第二个是在Single Service模式下进行完一次DMA原子传输。

3、Transfer Size

每个DMA都支持两种不同的transfer size: unit 和 Burst 4， 也就是一次DMA原子传输中分别有1个或4个读写操作，如下：
unit:

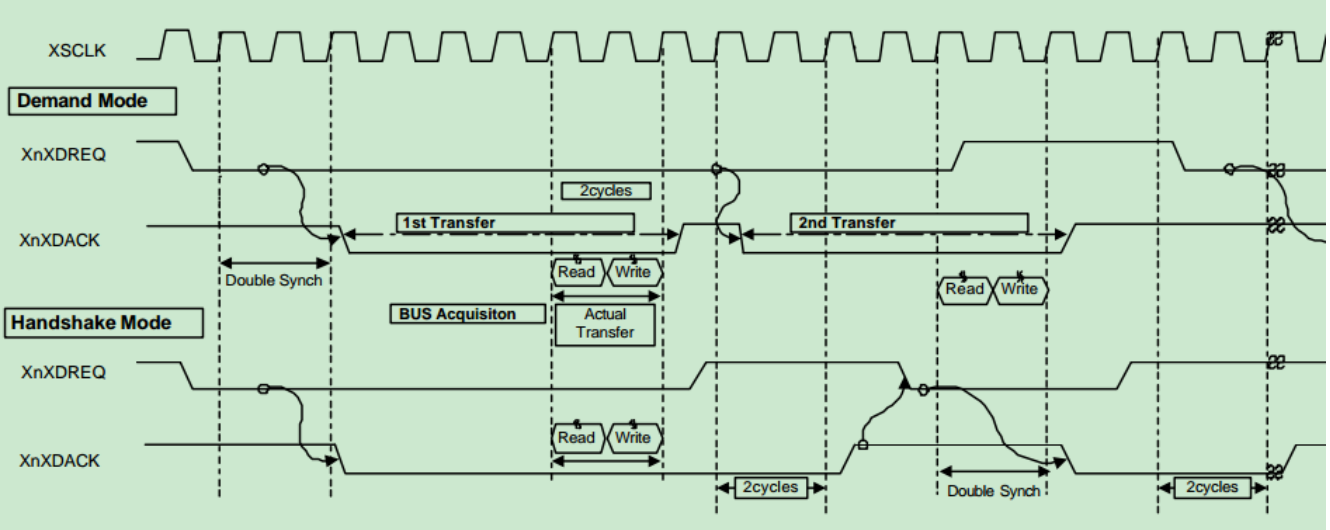


Burst4:



4、Demand & Handshake 模式

这两种模式描述的是XnXDREQ和XnXDACK这两个信号之间的协议
对比如下：



从图中知道：
这两种模式的相同点： DMA控制器在收到对应的DMA请求后（XnXDREQ拉低），都会发出DACK信号（XnXDACK拉低），并且开始传输数据。
不同之处：在完成一次DMA原子传输之后是否等待DREQ de-assert（拉高）。 对于Handshake模式，在完成一次DMA原子传输后，会等待DREQ de-assert，如果发现DREQ de-assert了，那么DACK也会de-assert，并且的等待DREQ的再次assert。而对于Demand模式，在完成一次DMA原子传输后不会等待DREQ的de-assert，而仅仅是DACK进行de-assert。在DACK de-assert后，会一直等待DREQ assert，如果发现DREQ assert的话，DACK也会立即assert，并开始下一次DMA原子传输。
SamSung建议对于片外设备的DMA传输的场景选用Handshake模式，以防止下一次传输不期望得发生。

5、具体分析寄存器

我们以DMAC0为例， 其他的完全相同。

寄存器	有效位	概述	Bits	功能
DISRC0	[30:0]	用于存放DMA数据源地址，也就是DMA原子传输开始后，从哪read数据。由于这个寄存器最高位不可见，所以，2440上面可以进行DMA的物理地址范围应该是0~256MB。		
DISRCC0	[1:0]	用于设置DMA read端的参数	[1]	如果数据源外设挂在AHB总线，该位置0， 如果挂在APB总线，该位置1
			[0]	用于控制源地址是否需要递增 置0表示需要递增，置1表示不需要 注： • 这里的递增的大小是跟data size和transfer size有关，应该是二者相乘，也就是一次DMA原子传输搬运的字节数 • 如果置1，在burst4模式下，地址也会递增，只不过做完一次DMA原子传输后，地址又会恢复成原来的 • 可以读取DCSRC0获得当前的源地址
DIDST0	[30:0]	DMA传输的目的地址		
DIDSTC0	[2:0]	用于设置DMA write端的参数	[2]	用于配置在auto reload使能的时，中断发生的时机 置0：当CURR_TC减到0的时候触发 置1：当auto-reload发生的时候触发
			[1]	如果目的地址外设挂在AHB总线，该位置0， 如果挂在APB总线，该位置1
			[0]	用于目的地址是否需要递增 置0表示需要递增，置1表示不需要 注：可以读取DCDST0获得当前目的地址
			[31]	用于选择Demand或Handshake模式 置0： Demand模式 置1： Handshake模式

DCON0	[31:0]	用于设置DMA通道的其他参数	[30]	设置DREQ/DACK的同步方式 对于有APB上面外设参与的DMA传输，应该置0，表示跟PCLK同步，如mem2dev或dev2mem 如果DMA传输只是AHB总线上的外设之间进行，则置1，表示跟AHB同步，如mem2mem
			[29]	用于控制在CURR_TC为零后是否触发中断 置0：不触发，此时就需要用户不断轮询STAT位了 置1：会触发
			[28]	用于设置前面我们讲到的transfer size 置0：表示每次DMA原子传输读写各1次 置1：表示每次DMA原子传输读写各4次
			[27]	用于选择Single或Whole sevice模式 置0：Single service模式，每做完一次DMA原子传输（在状态3），DMA会STOP（进入状态1），然后等待下一次DMA请求 置1：Whole service模式，会在状态3一直进行DMA传输，知道CURR_TC变成0，然后切到状态1 注： 即便在Whole Service模式，在完成一次DMA原子传输后，DMA控制器也会释放系统总线，然后重新再去获得，以防止其他bus masters由于长时间拿不到系统总线的控制权而导致“饥饿”
			[26:24]	用于设置DMA request source 参考前面的那张表 注：只有该DMA被设置为硬件触发时，这里的DMA request source才有意义，比如对于mem2mem，我们采用的是软件触发，所以这个字段忽略
			[23]	选择DMA是软件触发还是硬件触发 置0：软件触发，通过将SW_TRIG位置1来触发 置1：硬件触发，具体来自哪个外设由[26:24]来指出
			[22]	用于打开或在关闭auto reload 置0：开启auto reload，当CURR_TC减到0后，会auto reload（重新加载源和目的地址），并开始下一轮DMA传输 置1：关闭auto reload，当CURR_TC减到0后，停止DMA传输
			[21:20]	用于设置data size，也就是每个read/write操作读/写几个字节的数据，目前2440支持三种： 00： 一个字节 01： 两个字节 10： 四个字节
			[19:0]	总共需要进行DMA传输的次数 计算方法： 假如我需要通过DMA传输100字节的数据，那么需要的DMA传输的次数应该是： 100/ (transfer size) / (data size) transfer size: 每次DMA原子传输中有几个读/写操作 data size: 每个读/写操作会读/写几个字节 因此： (tranfer size) * (data size) 就可以理解为每次DMA原子传输会搬运多少字节的数据
DSTAT0	[21:0]	用于获得当前DMA通道的状态	[21:20]	用于查询当前DMA是否空闲 读00：当前DMA正在等待DMA请求，也就是状态1，空闲 读01：当前DMA正在传输数据，忙
			[19:0]	还需要进行的DMA原子传输的次数
DCSRC0	[30:0]	当前DMA原子传输的源地址，可能会递增		
DCDST0	[30:0]	当前DMA原子传输的目的地址，可能会递增		

DMASKTRIG0	[2:0]	用于开启或关闭DMA	[2]	置1：表示在完成当前正在进行的一次DMA原子传输后，DMA stop
			[1]	打开或关闭DMA 置0：关闭DMA，此时该DMA不会响应任何DMA请求 置1：打开DMA，此时DMA会响应DMA请求 注： 如果关闭了auto reload，在CURR_TC减到0后，该位会自动置0 如果设置了上面的STOP位[2]，那么在完成当前DMA原子传输后，该位也会自动置0
			[0]	软件触发的触发位，只有在前面设置为软件触发的时候才有意义 置1： 触发软件DMA请求 注：在触发的时候，当前DMA通道应该处于ON([1]), 并且STOP被置0([2])

注：
对DISRC、DIDST、DCON[19:0]的修改直到CURR_TC减到0，开始下一轮DMA传输的时才生效，而对其他寄存器位的修改则会立即生效。

第一部分对芯片手册的分析就到这里，下一片博客我们通过直接操作DMA控制器的寄存器完成一轮MEM2MEM的DMA传输。

完。