Software DMA Implementation

Introduction

A DMA (Direct Memory Access) Controller allows fast data transfers between memories without using the CPU. The transfer is activated by an external event. The control signals and the address and data buses are managed by the DMA controller.

The AT91M40400 does not feature a DMA controller as a peripheral. However, it can be implemented by software by using the load multiple (ldm) and store multiple (stm) instructions (refer to the “ARM Architectural Reference Manual”).

This application note proposes two different ways to implement a software DMA using the ARM Fast Interrupt (FIQ) as the external event. The code has been designed to reduce the number of instruction fetches as much as possible.

AT91 Series
ARM® Thumb® Microcontrollers

Application Note

Rev. 1169A–10/98

Theory of Operation

The principle of DMA is that an external event causes a fast memory copy. In this case the external event is the Fast Interrupt pin (FIQ) since the FIQ mode on the ARM7TDMI™ processor has banked registers (r8 to r14) which do not need to be saved before being used in FIQ mode. These registers are initialized before enabling the FIQ interrupt, then directly used when the external event occurs.

This application note describes two ways to implement a software DMA. Both implementations make the transfer by splitting the block to be transferred into packets of 4 words.

***Note**: In this document 1 packet = 4 words; 1 word = 4 bytes*

- **DMA Turbo**: Copies one packet (4 words) from a source buffer to a destination buffer at each FIQ interrupt.
- **DMA Channel**: Copies the number of packets (4 words) defined by the user from a source buffer to a destination buffer at each FIQ interrupt, thus freeing the core between packets.

As the FIQ interrupt is used, the DMA must be able to access the banked registers. This can only be done by switching to FIQ mode and only if the CPU is in privileged state (any mode except User Mode). This application note assumes that the user function calls are performed while the CPU is in privileged state. It does not describe how to switch from unprivileged to privileged state. This can be done on another level by using the “swi” instruction of the ARM core or the Advanced Interrupt Controller features allowing an interrupt entry to be forced. In the last case, FIQ or AIC’s software interrupt can be forced to set up software DMA.

Table 1. Source Files

Files	Contents
dma _turb.s	Code source for DMA Turbo
dma_chnl.s	Code source for DMA channel
dma.h	DMA prototypes and structures for application use
arm.inc	ARM core definition

DMA Turbo

The DMA Turbo allows 1 packet (4 words) to be transferred at each FIQ interrupt from a source buffer to a destination buffer. Source and destination buffer pointers are post-incremented after each transfer.

The main advantage to this implementation is that no registers need to be saved. It is also a faster way to transfer data than the DMA Channel implementation.

Table 2. Registers Used

Register	Parameter
r14	Link register - return address
r13	‘dest_pt’ = Destination buffer pointer
r12	‘src_pt’ = Source buffer pointer
r8 to r11	Transfer registers

User Functions

The functions used to manage the DMA Turbo are defined in *dma_turb.c*:

Table 3. DMA Turbo User Functions

void DmaTurboSet (int *source, int *destination)	Initialize the FIQ registers for the DMA transfer
void DmaTurboGet (int *source, int size)	Check the end of the transfer
void DmaTurboFiq (void)	FIQ treatment and DMA transfer

The **DmaTurboSet** function initializes the FIQ banked registers for the DMA Turbo transfer, then enables the DMA transfer by clearing bit F in the CPSR.

*void DmaTurboSet (int *source, int *destination)*

source the source buffer pointer

destination the destination buffer pointer

```
Begin
| Switch to FIQ mode and save entry mode
| Memorize source buffer pointer (r12_fiq = 'src_pt')
| Memorize destination buffer pointer (r13_fiq = 'dest_pt')
| Enable DMA (Clear bit F)
| Restore entry mode
End
```

The **DmaTurboGet** function checks that the end of the transfer occurred (“size” words received), and, if it did, disables the DMA transfer by setting bit F in the CPSR.

*int DmaTurboGet (int * source, int size)* returns the number of words already received

source the source buffer pointer at beginning

size the number of words to be copied

```
Begin
| Switch to FIQ mode and save entry mode
| Calculate number of transferred words
| Disable DMA (Clear bit F) if number of transferred words >= size
| Restore entry mode
End
```

The **DmaTurboFiq** function is activated by the FIQ (vector address 0x1C), and copies 4 words from source buffer to destination buffer.

void DmaTurboFiq (void)

```
Begin
| *dest_pt <- *src_pt (multiple load and store 4 words with post increment)
End
```

Alternate Implementation

The DMA Turbo can be modified to transfer more than 4 words per FIQ. For this, the pair of instructions “ldm” and “stm” (refer to the “ARM Architectural Reference Manual”) must be repeated as many times as necessary. Different types of source and destination buffers can be supported by using the options of post/pre increment/decrement of these instructions.

It can also be modified to make the number of words to transfer defined by the caller. This has the advantage of allowing the dynamic transfer size, but one register must be reserved for the count (e.g. r11). Therefore, only 3 registers are available for the transfer itself.

DMA Channel

The DMA Channel implementation copies the number of packets (4 words) defined by the user at each FIQ interrupt until the remaining number of bytes is less than 1 packet (4 words). These remaining bytes are transferred at the following FIQ interrupt, and the DMA is stopped by disabling the FIQ.

One inconvenience of this implementation is that non-banked registers are used. Therefore, it is mandatory to save these registers before the transfer, and to restore them afterwards.

The main advantage is that the size of the transfer is controlled, and there is no risk of overwriting the memory.

The block is copied from the source to the destination by boundaries of packets. When no more whole packets can be transferred, the remaining bytes (less than 16) are transferred in a way which optimizes the number of fetches:

- 2 words (8 to 15 remaining bytes)
- 1 word (4 to 7 remaining bytes)
- 1 half-word (2 or 3 remaining bytes)
- 1 byte (1 byte remaining)

The user can choose to increase, after each FIQ copy, the source pointer, the destination pointer, both pointers or neither.

Table 4. Registers Used

Register	Parameter
r14	Link register - return address
r13	Temporary buffer address
r12	Increment mask
r11	Number of packets to copy per FIQ
r10	Block size (byte number)
r9	Destination address
r8	Source address
r6 to r7	Working registers (must be saved and restored)
r0 to r3	Transfer registers (must be saved and restored)

Type Definition

The DMA channel parameters are set through a structure named “dma_descriptor”. This type is defined in dma.h:

```
typedef struct dma_descriptor
{
      u_char      *src;       // Source pointer
      u_char      *dest;      // Destination pointer
      u_int       nb_bytes;   // Total number of bytes to copy
      u_int       nb_packets; // Maximum number of packets to copy at each FIQ
      u_int       dma_mask;   // DMA channel mask (INC_SRC and/or INC_DEST)
} dma_descriptor;
```

The user can choose whether or not to increase the source/destination pointers using the “dma_mask” field, the value of which is an OR-combination of the constants INC_SRC and INC_DEST (These constants are defined in dma.h).

User Functions

The functions used to manage the DMA Channel are defined in *dma_chnl.c*:

Table 5. DMA Channel User Functions

void DmaChannelSet (dma_descriptor *dma_desc, int *buf)	Initialize the FIQ registers for the DMA transfer
void DmaChannelGet (void)	Get the count of remaining bytes to be transferred
void DmaChannelFiq (void)	FIQ treatment and DMA transfer

The **DmaChannelSet** function sets and enables the DMA Channel

*void DmaChannelSet (dma_descriptor *dma_desc, int *buf);*

dma_desc pointer to the DMA descriptor

buf temporary buffer (7*4 bytes needed) to store non-banked used registers

```
Begin
| Switch to FIQ mode and save entry mode
| r13_fiq <- temporary buffer pointer
| Load banked registers from the DMA descriptor
| Set default number of packets to transfer (1) if not defined by user
| Enable DMA (Clear bit F) if number of bytes to transfer > 0
| Restore entry mode
End
```

The **DmaChannelGet** function returns the remaining number of bytes to be transferred

int DmaChannelGet(void) returns remaining number of bytes

```
Begin
| Switch to FIQ mode and save entry mode
| Return number of bytes remaining to be transferred
| Restore entry mode
End
```

The **DmaChannelFiq** function is activated by the FIQ (vector address 0x1C), and copies r11 x 4 words from the <source> to the <destination> if possible.

void DmaChannelFiq (void)

```
Begin
| Save used registers in the temporary buffer
| Save source and destination
| While (nb_packets > 0) and (nb_bytes >= PACKET_SIZE)
| | Update nb_bytes by subtracting PACKET_SIZE from it
| | *dest <- *src (multiple load and store 4 words with post increment)
| | Decrement nb_packets
| EndWhile
| If (nb_packets == 0) (i.e. All blocks transferred)
| Then
| | === Copy last bytes to copy (< 16) ===
| | *dest <- *src
| | If needed, load and store multiple 2 words with post increment
| | If needed, load and store 1 word with post increment
| | If needed, load and store 1 half-word with post increment
| | If needed, load and store 1 byte with post increment
| | Clear nb_bytes
| | Disable DMA (SPSR.F <- 0)
| Endif
| Restore src (if not INC_SRC)
| Restore destination (if not INC_DEST)
| Restore used registers
End
```

Required Resources

Table 6. DMA Turbo

Parameter	Value
Code Size	20 words
Register Usage	R8 to R14 (banked FIQ registers)
Peripheral Usage	FIQ Interrupt

Table 7. DMA Channel

Parameter	Value
Code Size	51 words
Register Usage	R8 to R14 (banked FIQ registers)
Peripheral Usage	FIQ Interrupt

Tips and Warnings for Both Types of DMA Implementation

Pointer Alignment

Source and destination pointers must always be word-aligned because of the use of load and store multiple instructions. If this is not the case, a word-alignment must be performed, using an “AND” with the existing word in the memory to copy the first non-aligned bytes.

FIQ Rate

Since the DMA described in this application note performs the transfer of the packet(s) at the FIQ interrupt, it is necessary to pace the FIQ at a rate defined by the application.

This application note does not describe how to generate the rate, but it can easily be done by using a timer channel which is cyclically triggered to generate a tick. An FIQ is then generated on each tick. The FIQ can be generated by configuring source 0 to be edge triggered, and by setting bit 0 in the AIC_ISCR register to one (0x00000001). The port P12, multiplexed with FIQ, should be configured as a PIO in order to prevent the FIQ from being generated by an external event.

Another way is to configure the port P12 as a peripheral, and to make the external hardware generate the FIQ which can be configured to be edge or level triggered.

In any case, the source 0 (FIQ) of the AIC must be configured (register AIC_SMR0) and enabled (register AIC_IECR).

Tips and Warnings for DMA Channel Implementation

Temporary Buffer

This application does not implement a stack management to save the registers. For this reason, a temporary buffer has been chosen in order to save and restore the non-banked registers in the function **DmaChannelFiq**. Of course, this temporary buffer can also be performed by a stack management. In this case, the function **DmaChannelSet** needs only the DMA descriptor as a call parameter.

Pointer Incrementation

If the source and destination pointers are systematically incremented during the transfer, the command word with the pointer management mask *ptrMask* is useless. In this case, the saving and conditional restoring of the source and destination pointers is also useless and the registers *R6* and *R7* are unused in the function **DmaChannelFiq**. As the mask is not used, the register *R12* is also unused in the function **DmaChannelFiq** and can be used as the packet counter (in place of *R11* which must be restored).

This way only registers *R0* to *R3* in the function **DmaChannelFiq** must be saved, and the temporary buffer can be only 4-word-sized.

Large Sized Blocks

If the blocks to transfer are significant (more than 15 words), the use of the registers can be optimized by increasing the size of the temporary buffer. *R6* and *R7* are used only to save *R8* and *R9*. This can be done in the temporary buffer, and whole non-banked registers (*R0* and *R7*) can be used for the transfer in the function **DmaChannelFiq** (care must be taken however when restoring *R8* and *R9*).

The temporary buffer becomes 11 words (if optional increment feature is used), but the packets transferred are 8-words (PACKET_SIZE).

It is necessary to add a 4-word conditional transfer before the 2-word transfer:

```
| | | *dest <- *src (load and store multiple 4 words with post
| | |    increment) if needed
```

This is coded by:

```
ands                    r0, r10, #0x10
ldmneia                 r8!, {r0-r3}
stmneia                 r9!, {r0-r3}
```