TWL DMA Explained

Mark Jawad
Technical Leader

Software Development Support Group

Intro / Presentation Background

- Existing documentation on DMA makes a lot of assumptions about the reader's knowledge of how [our] hardware works
- Additional reference material is needed for: "Thoughts on Developing for Nintendo DSi", presented at the 2009 Technical Conference
- This presentation was developed to help meet those needs.

What is DMA?

- DMA is an acronym for: Direct Memory Access
  - In other words, it’s something that can touch memory and do so without relying on a proxy
- “DMA” can also be used as a verb
  - Meaning: to transfer data via DMA controller

What’s a DMA Controller?

- A DMA Controller is an independent processor who’s sole purpose is to move data from one location of memory to another
  - That’s all it does.
    - Read some memory, then write it somewhere else

How does it work?

- DMA controllers are usually programmed by other clients in the system (such as the CPU)
  - They idle until directed to do a transfer
- DMA controllers operate in parallel with other processing clients on the system
  - Allows those other clients to focus on getting more important work done while the DMA controller shuffles data around

Things to Note about DMA

- DMA controllers are not part of the CPU
- They can't see “inside” of the CPU
  - Cache, TCM, registers: all off-limits to DMA
- Shares the same bus with the CPU
- Only one of the clients (CPU, DMA, DSP) on a bus is allowed to access that bus at any given time. The others must wait for the one to relinquish access to the bus before they can access it

Why use DMA instead of the CPU?

- On Nitro/TWL, use it because it benefits from *burst access* to main RAM – it can move data **faster** than the CPU can when Main RAM is either the source or dest (but not both!)
- Use it to transfer data that you don't want to end up in the CPU's caches
- Use it to offload busy-work from the CPU
  - Allowing the CPU to do other operations

Applications of DMA

- Use it to move data 'in the foreground' while CPU does work 'in the background'
  - (True for Nitro and legacy DMA controller)
- Use it to move data 'in the background' while CPU does work 'in the foreground'
  - (True for TWL's New DMA controller)
- Useable on all (valid) addresses except TCM

Pros and Cons of DMA

Pros:

- Allows the CPU to work on computations instead of wasting time on 'simple' loads and stores
- Since CPU doesn't touch data, CPU cache pollution can't occur – good for performance

Cons:

- Does not have access to data in the CPU's caches. This can cause problems if not handled properly
- Locks the bus while active, potentially stalling the CPU until DMA is complete

DMA on Nitro

DMA on Nitro: Overview

- Each bus (ARM7, ARM9) has a CPU and DMA controller attached to it
- Each DMA controller is independently programmable
  - But slaved to the associated ARM CPU
- Each DMA controller has 4 programmable ‘channels’ that operate independently of the others

DMA on Nitro: Channels

- Each channel has:
  - Source Address
  - Destination Address
  - Control Register (Num bytes to transfer, Transfer mode, etc)
- One could think of each channel as a client of the bus
  - So, there are 4 DMA clients per bus
  - And 1 CPU client per bus

DMA on Nitro: Channel Scheduling

- After current client is done transferring, the highest priority client in the list (with an active data request) gets to run
  - DMA0 > DMA1 > DMA2 > DMA3 > CPU
- CPU has lower priority than DMA controller, so will get 'locked out' when any DMA channel is actively moving data
  - CPU won't get bus until all DMAs complete!

DMA on Nitro: Bus Scheduling

- The SDK configures the HW such that ARM7 bus requests are higher priority than ARM9 bus requests
  - Order is:
    ARM7 DMA > ARM7 CPU >
    ARM9 DMA > ARM9 CPU
- Thus ARM7 DMA/CPU could cause ARM9 DMA/CPU to stall when there's contention for a specific resource

DMA on Nitro: Operational Modes

- Manual Start
- Auto-start
- Auto-start (continuous)

DMA on Nitro: Manual Operation

- Each channel can be programmed to startup as soon as the scheduler allows
  - When this happens, the desired number of units (of u16 or u32 size) will get transferred, then the channel will deactivate by setting itself to ‘disabled’
- This is also known as Immediate Mode

DMA on Nitro: Auto-start Operation (continuous)

- Each channel may be programmed to automatically start based on an incoming signal from some other part of the system
  - Prime the channel with the number of units to transfer, the source and destination addresses, and source of start-up signal
  - When the appropriate signal arrives the transfer will run as soon as the scheduler allows. Afterwards the channel will relinquish the bus until the next signal arrives.

DMA on Nitro: Auto-start Operation (continuous)

- Continued from previous slide..
  - Transfers will continue as long as signals arrive
  - Some auto-start transfer types might automatically deactivate the channel after a certain amount of work
  - Other types continue indefinitely and require manual deactivation

DMA on Nitro: Auto-start Operation (one-shot)

- Nearly identical to Auto-start (continuous) mode, but will disable itself after the arrival of the first signal
- This allows for one single transfer to be kicked off by the hardware

DMA on Nitro: Incoming signals

- Auto-start signals are similar to interrupt signals
  - Raised by external devices when “something important” happens
- Auto-start signals come from a variety of onboard devices
  - Display hardware, graphics engines, card hardware, etc

DMA on Nitro: Outgoing signals

- Each DMA channel can send an interrupt to the CPU when it's done with it's work
  - This way the CPU won't have to poll for completion of a given DMA channel
  - It helps with asynchronous processing, which requires that the CPU stay off of the bus
    - Bus access is required to talk to anything in I/O space (including the DMA controller)
    - Polling the status of a DMA channel would therefore be disastrous, since we'd immediately stall until the DMA completed

DMA on Nitro: Block Transfers

- DMA transfers lock the bus for the entire duration of the transfer
  - Which is: WORD_COUNT worth of units
- Large WORD_COUNTs can cause system imbalance because other clients may not be able to get to the data they need in time
  - i.e. ARM9 can't get to RAM if ARM7 DMA has locked the external memory interface.
  - Can cause all sorts of problems

DMA on Nitro: Address Updates

- Source and destination address can independently:
  - Increment
  - Decrement
  - Stay at a fixed location
- Destination address can also:
  - Increment for the current transfer and then (when the current transfer is finished) reset to it’s starting address

DMA on TWL

DMA on TWL: Overview

- Nitro’s existing “legacy” DMA architecture is still fully available
  - And still functions exactly as it used to
  - Can toggle if Nitro auto-start bugs should persist as they were, or be corrected to conform to the original spec
- Additionally, we’ve added a “New DMA” (NDMA) controller

DMA on TWL: NDMA overview

- Adds 4 new channels + GCNT register
- All transfers are 32bit units now
- Provides more options, modes, auto-start triggers than the legacy DMA does
- Has multiple new scheduling / priority settings
  - Very important for maintaining system balance

DMA on TWL: NDMA channels

- Channels are more complex than on NTR
- Each contains:
  - Source Address
  - Destination Address
  - “Fill Data” value for cases when no source address is required (for memset, etc)
  - Multiple control registers
    - Complex transfer parameters, DMA scheduling

DMA on TWL:
Fill Data register

- New source for written data
  - Instead of continually reloading constant data from a fixed source address, you instead program a constant 32bit value in the channel's FDATA register
- FDATA value is repeatedly written to the destination address

DMA on TWL: Fill Data benefits

- When using Fill Data, the NDMA only has to do sequential writes to the bus (no data reading is necessary)
- This results in the fastest writes possible
  - Effectively a free performance boost by using this compared to the legacy DMA Fill method

DMA on TWL: Legacy vs New

- Legacy DMA transfers lock the bus for the duration of the transfer
  - As mentioned earlier, this can cause other clients in the system to stall
- NDMA introduces a time-slicing approach so that no one channel keeps ownership of the bus for too long
  - Thus allowing other clients regular bus access

DMA on TWL: NDMA Channel Time Slice Config

- Each NDMA channel has two settings to controls how the channel divides its time
- Block Transfer Word Count (BTWC)
  - This is the atomic unit (in number of u32s) moved without interruption / rescheduling
- Interval Timer (ICNT)
  - This is the amount of time (in terms of bus clocks) to sleep before the next schedule request

DMA on TWL: Legacy Mode Scheduling

- When NDMA's Arbitration Method flag is set to Legacy
  - DMA0 > DMA1 > DMA2 > DMA3 > NDMA0 > NDMA1 > NDMA2 > NDMA3 > DSP > CPU

DMA on TWL: Legacy Scheduling Notes

- All channels / clients are scheduled according to NITRO rules
  - Only clients requesting activity will be considered
  - Active NDMA channels that are sleeping due to ICNT timer won’t be considered for scheduling

DMA on TWL: Legacy Mode Time Slices

- Legacy DMA, CPU and DSP run according to NITRO rules
  - Active transfers hold the bus until WORDCNT units are moved
- NDMA channels run according to BTWC+ICNT settings per channel
  - Active channels hold the bus until BTWC u32's are moved

DMA on TWL: Round Robin Scheduling

- When NDMA's Arbitration Method flag is set to Round Robin
  - DMA0 > DMA1 > DMA2 > DMA3 > [next entry in the RR set]
- RR Set consists of: {NDMA0, NDMA1, NDMA2, NDMA3, DSP_or_CPU}
  - DSP_or_CPU: If DSP activity req'd, schedule the DSP. Otherwise, schedule the CPU

DMA on TWL: RR Scheduling Notes

- Legacy DMA channels schedule according to NITRO rules
- Active NDMA channels that are sleeping due to ICNT timer won't be considered for scheduling
  - Inactive NDMA channels won't be scheduled
- DSP and CPU must make requests to the bus to be considered for scheduling

DMA on TWL: Round Robin Time Slices

- Similar to legacy, with exception of CPU / DSP
  - These use the CPUCYCLE setting to determine the amount of bus cycles they have access to at any one time; it's their version of the BTWC

DMA on TWL: Address Updates

- Basically the same as on NITRO
- Source register adds a "use FDATA as the source" setting
- Source register also can now reset to the starting address after transfer is finished

DMA on TWL: Operational Modes

- Manual Start
- Auto-start
- Auto-start with data limit

DMA on TWL: Manual-Start Setup

- Program the channel:
  - Source Address (or fill data)
  - Destination Address
  - Word Count to transfer
  - Timeslice data (BTWC,ICNT)
  - Address incrementer settings
  - Set the Immediate Mode flag + Enable flag

DMA on TWL: Manual-Start Operation

- Transfer will begin as soon as the channel is scheduled to run
- Transfer is time-sliced as mentioned earlier
  - So, occasionally paused then resumed
- Transfer will be ongoing until WordCnt u32’s have been moved
  - After which, the channel will deactivate

DMA on TWL: Auto-Start Setup

- Program the Channel
  - Source Address (or fill data)
  - Destination Address
  - Word Count
  - Timeslice data
  - Address incrementer settings
  - Device to monitor for autostart signal
  - Set the Repeat flag + Enable flag

DMA on TWL: Auto-Start Operation

- After device signal is raised the transfer will be scheduled to start
- Transfer is time-sliced as mentioned earlier
- Transfer will be ongoing until WordCnt u32's have been moved
  - Afterwards the channel will reset for next time
  - Like on Nitro, the process may repeat forever until manually deactivated, or may auto-stop according to signal source device rules

DMA on TWL: Auto-Start with data limit Setup

- Program the Channel
  - Source Address (or fill data)
  - Destination Address
  - Word Count (for individual transfers)
  - Total Word Count (to determine when to stop)
  - Timeslice data
  - Address incrementer settings
  - Device to monitor for autostart signal
  - Set the Enable flag

DMA on TWL: Auto-Start w/ Auto-Stop Operation

- After device signal is raised the transfer will be scheduled to start
- Transfer is time-sliced as mentioned earlier
- Transfer will be ongoing until WordCnt u32's have been moved
  - Afterwards the channel will reset for next time
  - The process will repeat until TotalWordCnt u32's are moved. Then, it becomes disabled

DMA on TWL: Notes

- Channels are either deactivated or enabled
- Enabled channels may be idle
  - If waiting for Auto-Start signal
  - Are sleeping for ICNT duration
- SRL / DRL (address reload) doesn't happen after BTWC; it happens once Word Count u32's have been transferred

DMA on TWL: Notes

- BTWC can be smaller than WCNT value
- BTWC can be larger than WCNT value
  - Won’t hold on to the bus longer than necessary, so no need to worry about this
- WCNT can be smaller than TCNT value
- WCNT can be larger than TCNT value
  - “Auto-Start with data limit” transfers won’t move more than TCNT u32’s.
- NDMAs will not transfer “too much” data

Closing Thoughts

Summary

- All the stuff you know from Nitro is still there
- Powerful NDMA features can really make your life easier
- Use DMA to offload ‘trivial’ data copying work from the CPU
  - And experience a slight performance boost in the process

Things to consider

- Auto-start DMA is the way that background (asynchronous) loading happens from the DS Card ROM.
- NDMA can be used to stream data into and out of WRAM, which could prove useful

Things to consider

- Think carefully about how to avoid starving the CPU
  - Try using auto-start DMAs wherever possible
  - Keep legacy DMA transfers small
  - Watch out for ARM7 DMAs; they can still stall your ARM9 if both access main RAM
  - Experiment with BTWC + ICNT settings for NDMA channels

Things to consider

- Place interrupt callbacks + data in TCM
  - Otherwise CPU processing could stall during an interrupt if DMA has locked the bus
- Use the new Fill feature for fast memset
- Make sure to flush / invalidate the CPU cache as appropriate so that DMA moves correct data

Questions?

Email support@noa.com