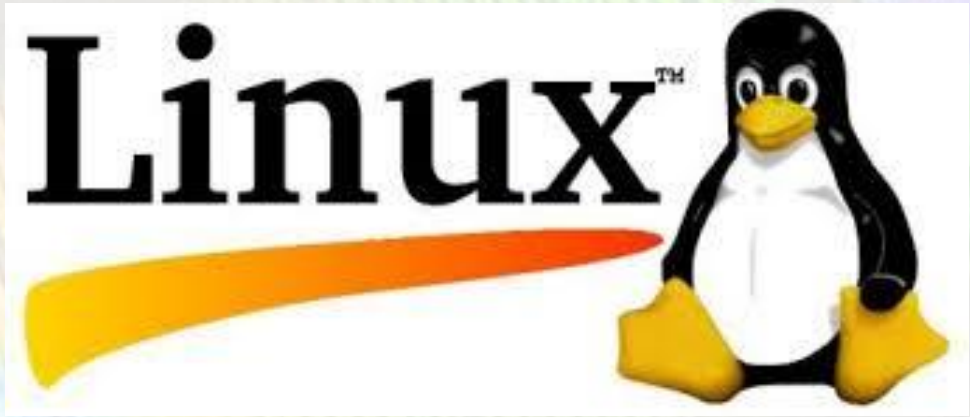
ARM Linux CPU PL Data Transfer



John Linn
Based on Linux kernel 3.14
Petalinux 2014.2

Introduction

- The goal of this presentation is to clarify the process of communicating between the Cortex A9 CPU and the PL of Zynq in Linux without DMA
- Depending on the amount of data to be transferred it may be quicker and easier to use the CPU rather than DMA
- The CPU, together with the L1 and L2 caches, incorporate a lot features that make it good at transferring data when set up correctly
- Setting up Linux for performance is not always clear

ARM Kernel Memory Mapping Functions

- **There are a number of functions in the kernel that are designed to map a memory region into the virtual address space of the CPU**
- **These functions cannot be used to map memory that is already part of the kernel**
- **These functions are used to map I/O device registers or device memory**
- **ioremap() is used often in device drivers**
  - It maps a memory region, typically device registers, into the virtual address space of the CPU using the device memory attribute for the MMU
  - There is no caching for this memory attribute
  - Device memory is defined in the Zynq TRM

ARM Kernel Memory Mapping Functions

- **ioremap_wc() is used some in device drivers**
  - It maps a memory region into the virtual address space of the CPU using normal memory attributes with no caching for the MMU
  - Writes are combined (bufferable) such that they can be more efficient
  - Usage tends to be video/gpu drivers (but not exclusively) in the kernel
  - This memory has the same memory attributes as memory allocated with dma_alloc_coherent()
  - The kernel page table dump shows MEM/BUFFERABLE/WC
  - Memory barriers may be required to ensure writes are completed before other memory operations
  - reference: https://lkml.org/lkml/2013/4/19/116
- **ioremap_cache() is not used much in the kernel**
  - It maps a memory region into the virtual address space of the CPU using normal memory attributes with caching for the MMU
  - Software must manage the caches for this memory region
  - Can be used to access BRAM faster

ARM Cache Terminology

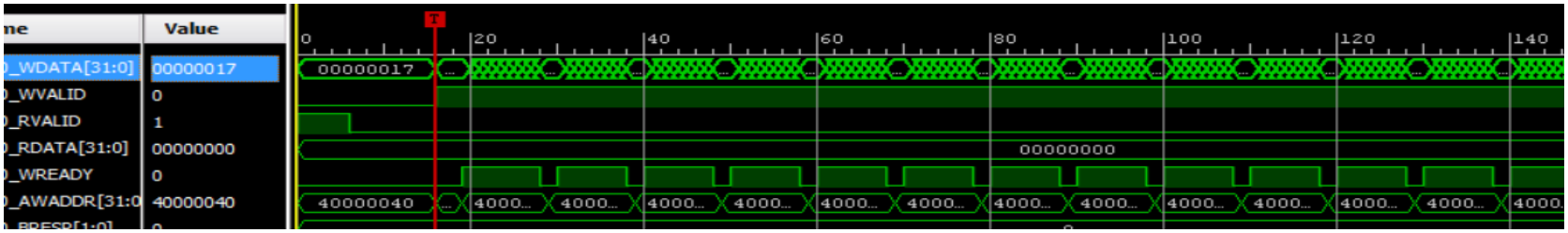
- **Caching terminology seems to vary such that clarity is needed**
- **For ARM processors the following definitions are used and these carry over into Linux**
- **Flushing the cache causes entries in the cache to be invalid, sometimes called invalidating the cache entry**
  - The next time the CPU accesses the cache entry it will cause a cache miss and memory will be accessed
- **Cleaning the cache causes entries which are dirty (changed) to be written to the memory (or next level cache)**

Linux (ARM) Cache Control Functions

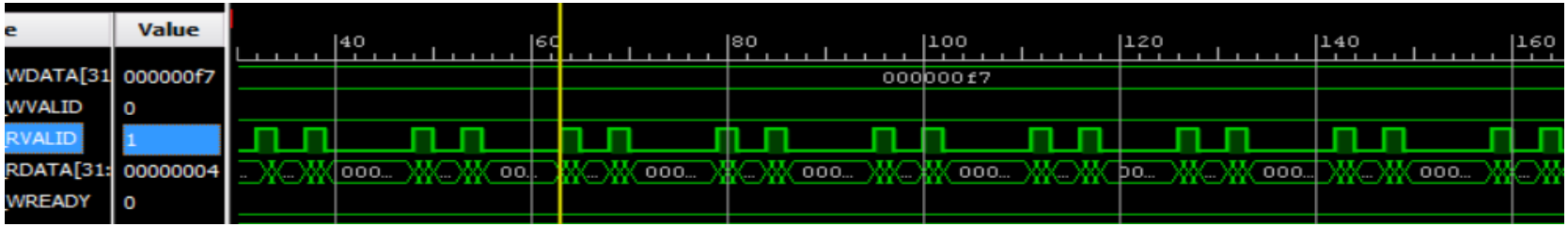
- **The L1/L2 functions must be called in the correct order depending on the direction of data flow, similar to DMA.**
  - The cache closest to the memory source is the 1st always
  - For moving data from DDR to the PL, the L1 should be first then the L2
  - For moving data from the PL to DDR, the L2 should be first then the L1
- **L1**
  - __cpuc_flush_dcache_area(virtual address, length)
  - This function does a clean and invalidate and it's the only function to do an invalidate
  - A clean is only done when lines are dirty such that the function acts like an invalidate only when the lines are clean
- **L2**
  - outer_inv_range(start physical address, end physical address)
  - outer_clean_range(start physical address, end physical address)

ILA AXI Read Transactions

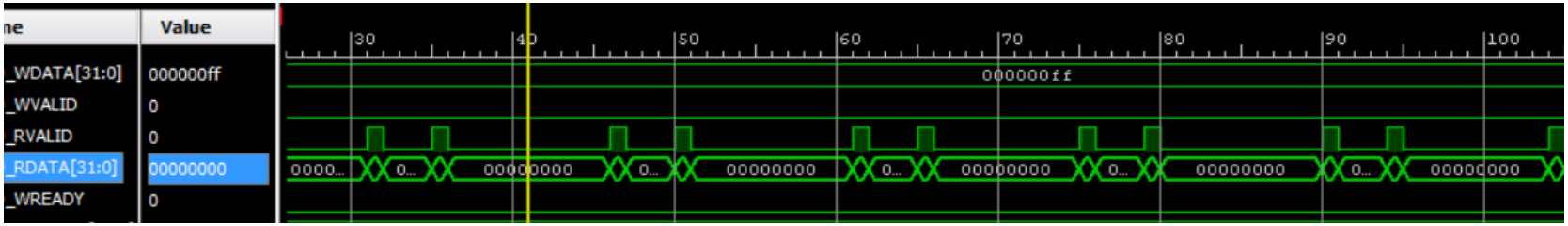
- ioremap_cache(), cached memory



- ioremap_wc(), uncached write combined memory

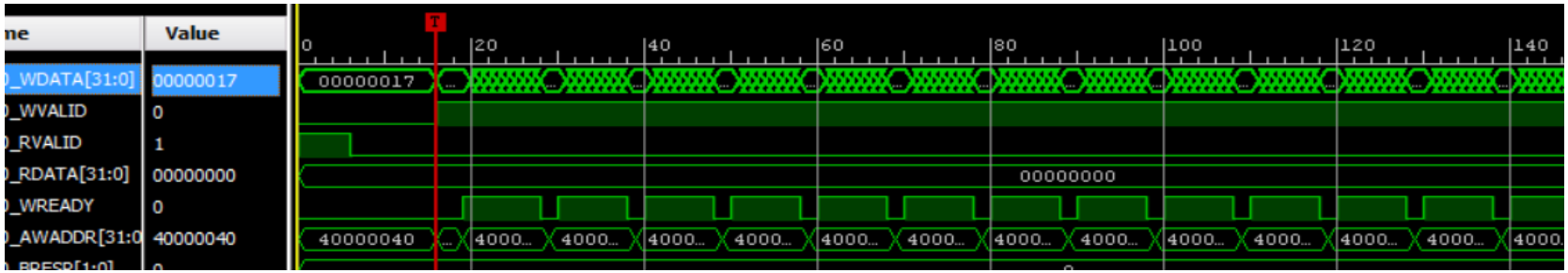


- ioremap(), uncached memory

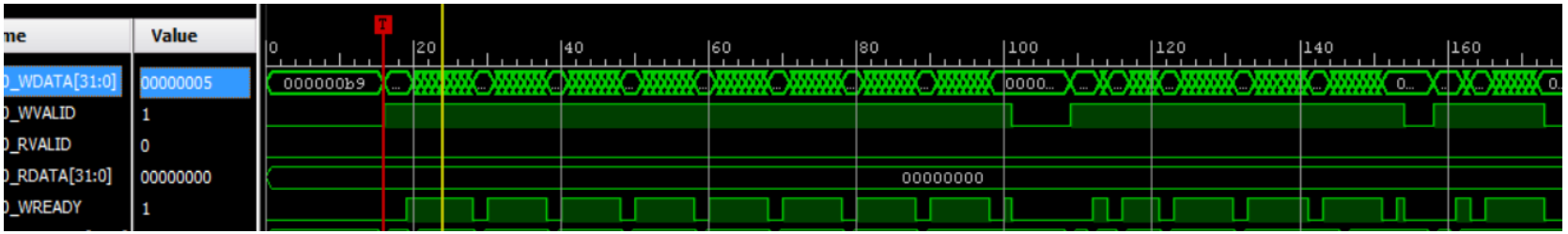


ILA AXI Write Transactions

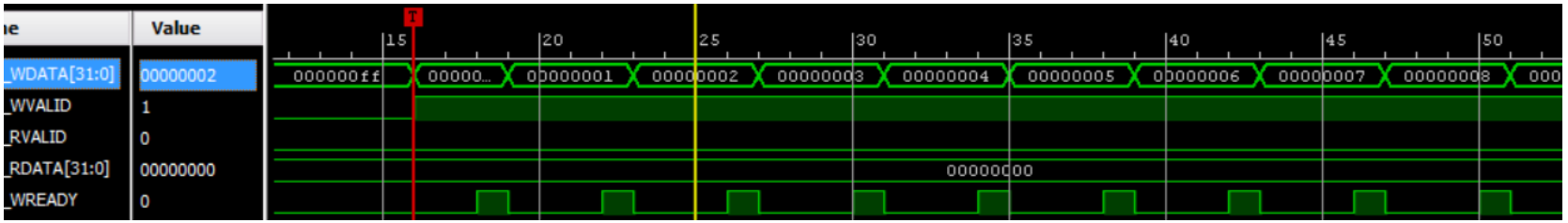
- ioremap_cache(), cached memory



- ioremap_wc(), uncached, write combined (bufferable) memory



- ioremap(), uncached memory



ARM Kernel Memory Access Functions

- **Memory access functions are provided in the kernel for mapped memory regions such as device registers or device memory**
  - arch/arm/include/asm/io.h, you can spend hours looking at this file
- **These functions are recommended rather than a raw pointer**
- **These functions are typically macros and inline such that there no more overhead to use them than a raw pointer**
- **There tend to be a lot of these functions and it's somewhat challenging to determine the right choice**
  - Drivers using each of them can be found in the kernel
- **The volatile keyword is handled in the memory access functions such that volatile in the kernel is not needed generally**
  - https://www.kernel.org/doc/Documentation/volatile-considered-harmful.txt
- **A special data type, iomem void *, is returned from an ioremap*() function and represents the virtual address**

ARM Kernel Memory Access Functions

- Functions are named based on the size of the data
  - with “l” for long which is 32 bits on Zynq, typically used by Linux device drivers
- Memory barriers and endian conversion are typically the differing features
- **__raw_read*(), __raw_write*()**
  - These function do not incorporate memory barriers
- **read*_relaxed(), write*_relaxed()**
  - These functions are equivalent to __raw*() functions on ARM little endian
- **read*(), write*()**
  - These functions incorporate memory barriers
- **ioread*(), iowrite*()**
  - These functions use the *relaxed() functions and also incorporate memory barriers

CPU<->PL Kernel Module

- A kernel module, cpu-pl-test.c, is provided to help with prototyping in this area
- It is not intended to be a complete benchmark, but a tool for learning
- The module allows all three memory mapping functions, ioremap(), ioremap_wc() and ioremap_cache() to be tested and better understood
- It was used to gather the ILA AXI Transactions in this presentation
- It is designed to run a single test when inserted, then be removed, and re-inserted to run another test
- Module parameters are used to control the module and allow other tests to be performed
- The source code documents the input parameters and their defaults
- It does measure the execution time of key areas in the module for determining rough performance

Conclusions

- A clear understanding of the data to be transferred between the CPU and PL is important for getting the best performance
- Cached memory requires cached operations to be performed by the CPU and those operations take time which must be considered
- Measuring the execution time of software allows the transfers to be analyzed and then optimized
- An ILA on the AXI bus allows the transactions to be clearly observed to verify the performance and see more details that execution time of software cannot reveal
- The kernel nomenclature does not always easily map to the ARM hardware (write combined -> buffereable)
- Dumping the kernel page tables can help verify memory is mapped as expected