Tools and Techniques for Debugging Embedded Linux Systems

Overview

- Debugging with prints
- Logging to circular buffers
- SW trace tools
- ETM
- Observability and GPIOs
- JTAG
- Register dumps and decoders

Printf debugging

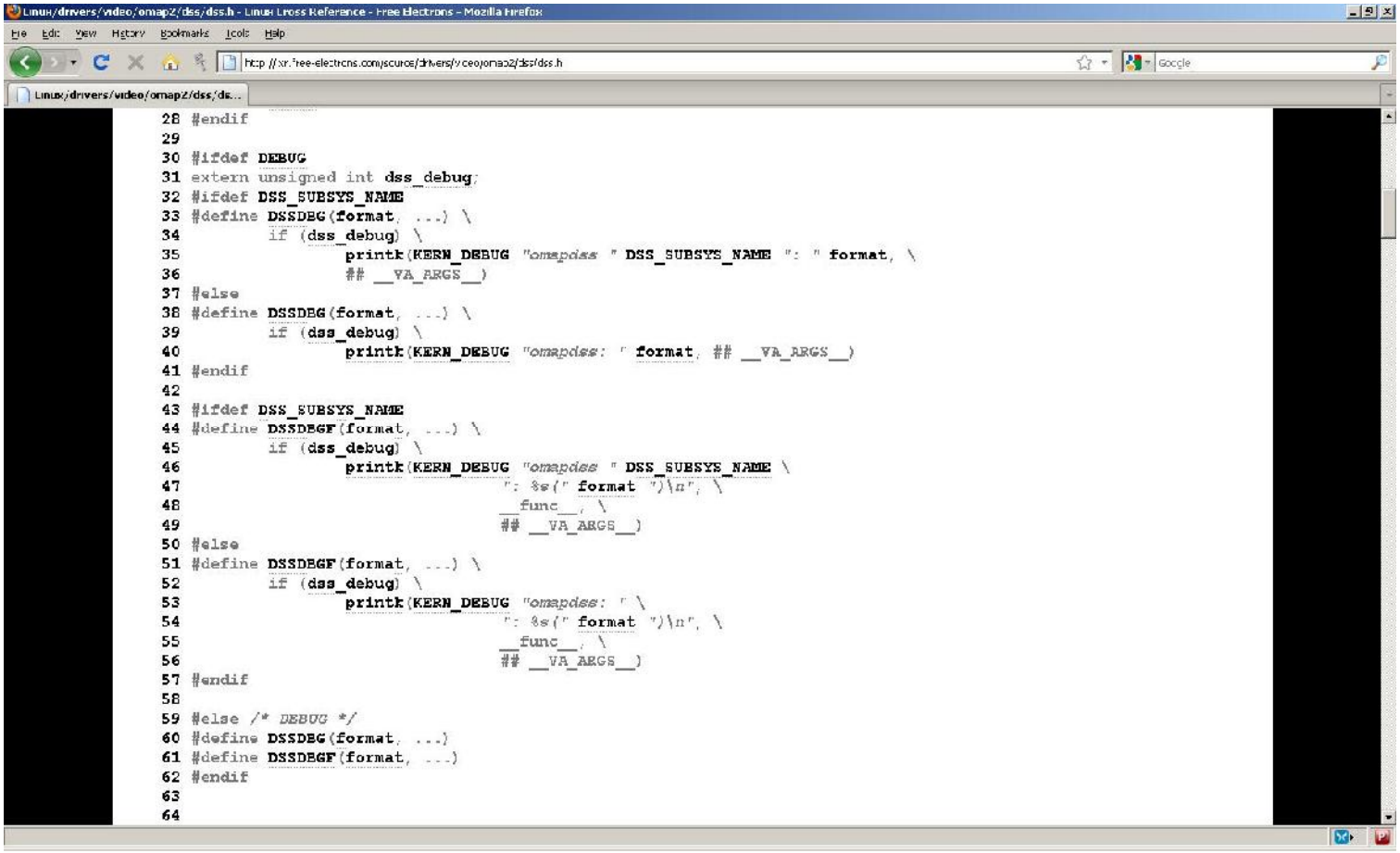
- Basic debugging technique
- Simple to use

printk loglevels

- From KERN_EMERG to KERN_DEBUG
  - pr_emerg to pr_debug
- Can change on the kernel command line
  - loglevel= parameter
- Can change after bootup
  - /proc/sys/kernel/printk
  - /proc/sysrq-trigger

Custom debug implementations

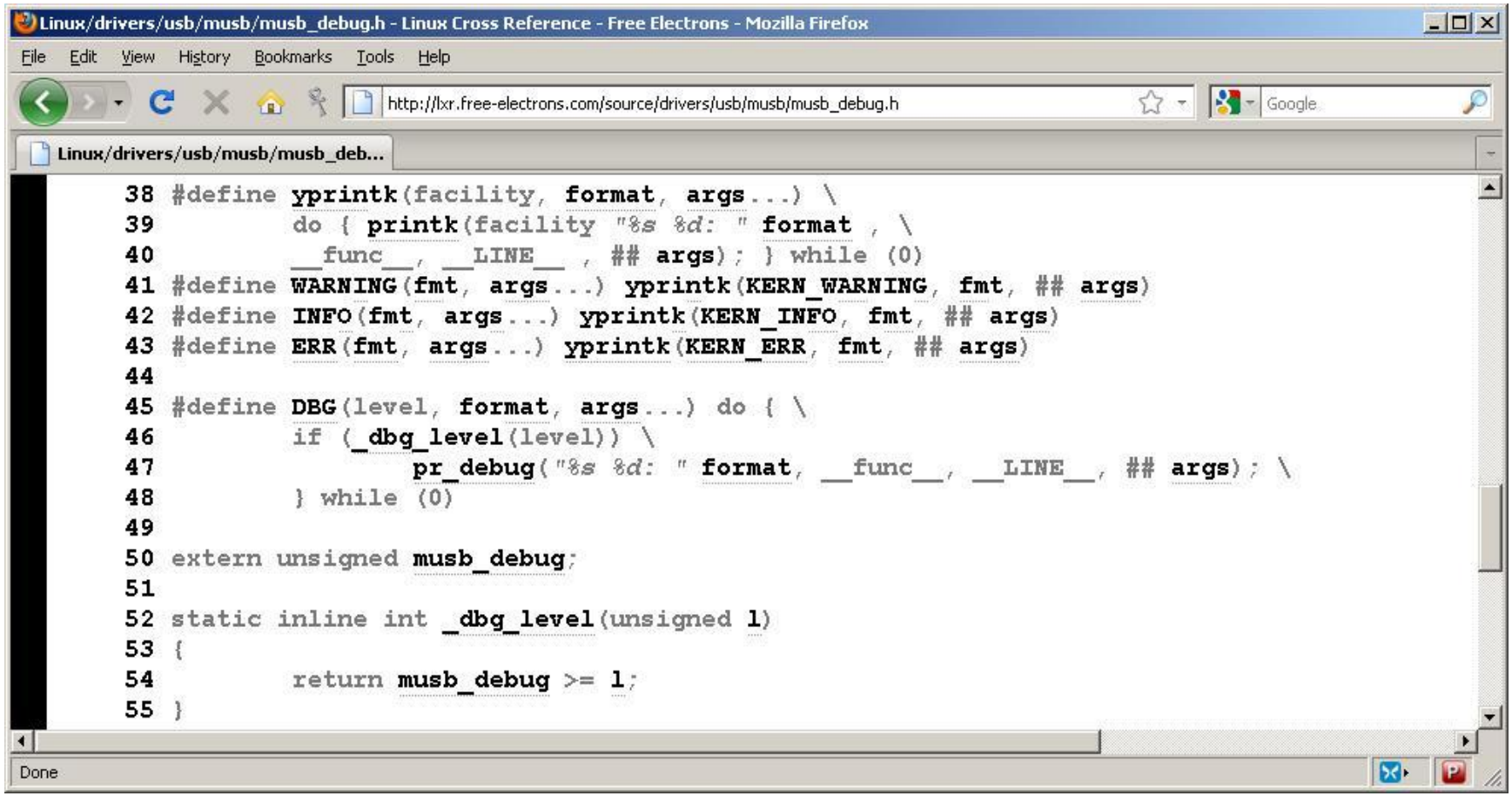
- Example: drivers/video/omap2/dss/dss.h



```
28 #endif
29
30 #ifdef DEBUG
31 extern unsigned int dss_debug;
32 #ifdef DSS_SUBSYS_NAME
33 #define DSSDBG(format, ...) \
34         if (dss_debug) \
35                 printk(KERN_DEBUG "omapdss " DSS_SUBSYS_NAME ": " format, \
36                 ## __VA_ARGS__)
37 #else
38 #define DSSDBG(format, ...) \
39         if (dss_debug) \
40                 printk(KERN_DEBUG "omapdss: " format, ## __VA_ARGS__)
41 #endif
42
43 #ifdef DSS_SUBSYS_NAME
44 #define DSSDBGF(format, ...) \
45         if (dss_debug) \
46                 printk(KERN_DEBUG "omapdss " DSS_SUBSYS_NAME \
47                                 ": %s(" format ")\n", \
48                                 __func__, \
49                                 ## __VA_ARGS__)
50 #else
51 #define DSSDBGF(format, ...) \
52         if (dss_debug) \
53                 printk(KERN_DEBUG "omapdss: " \
54                                 ": %s(" format ")\n", \
55                                 __func__, \
56                                 ## __VA_ARGS__)
57 #endif
58
59 #else /* DEBUG */
60 #define DSSDBG(format, ...)
61 #define DSSDBGF(format, ...)
62 #endif
63
64
```

Custom debug implementations

- Example: drivers/usb/musb/musb_debug.h



Printk tips and tricks

- CONFIG_PRINTK_TIME
- CONFIG_EARLY_PRINTK
  - CONFIG_DEBUG_LL and the printascii patch
- CONFIG_LOG_BUF_SHIFT

- Accessing the printk buffer with a JTAG debugger

- http://elinux.org/Kernel_Debugging_Tips

Use standard kernel debug interfaces

- pr_debug
- dev_dbg

- Why?

The problem with prints

- It can change the timing
  - sprintf call
    - How long does this take
  - serial port delays
    - How long does a UART transmission take?
      - Does this change with USB-UARTs?
        - » What about regular displays?
      - Can we use higher baud rate?

The problem with prints

Debug Level	Throughput (Mbps)			
	Prints to console disabled		Prints to console enabled	
	TX	RX	TX	RX
1	169	65		
3	161	32	1.25	0.16
5	113	18	0.49	0.07

Notes:
Debug level 3 adds 19 lines of print per transfer for TX and 40 for RX
Debug level 5 adds 37 and 92 respectively

Dynamic printks

- CONFIG_DYNAMIC_DEBUG
  - Introduced in 2.6.30
- Operates on pr_debug/dev_dbg

- More info
  - Documentation/dynamic-debug-howto.txt
  - http://lwn.net/Articles/434833/

Circular buffers

- Useful when you want to capture the last few things that were going on in the system
- In some cases, single character circular buffers are all that you can afford (DSP SW…)

Circular buffers (Case Study - MUSB)

- MUSB double buffering
  - Data transfers stop after a while when double-packet buffering enabled
  - Works for short amounts of data
    - Intermittent failure
  - With debug enabled, cannot reproduce failure
    - Even if not printing to console
  - No failures with single-packet buffering (existing code)

Circular buffers (Case Study - MUSB)

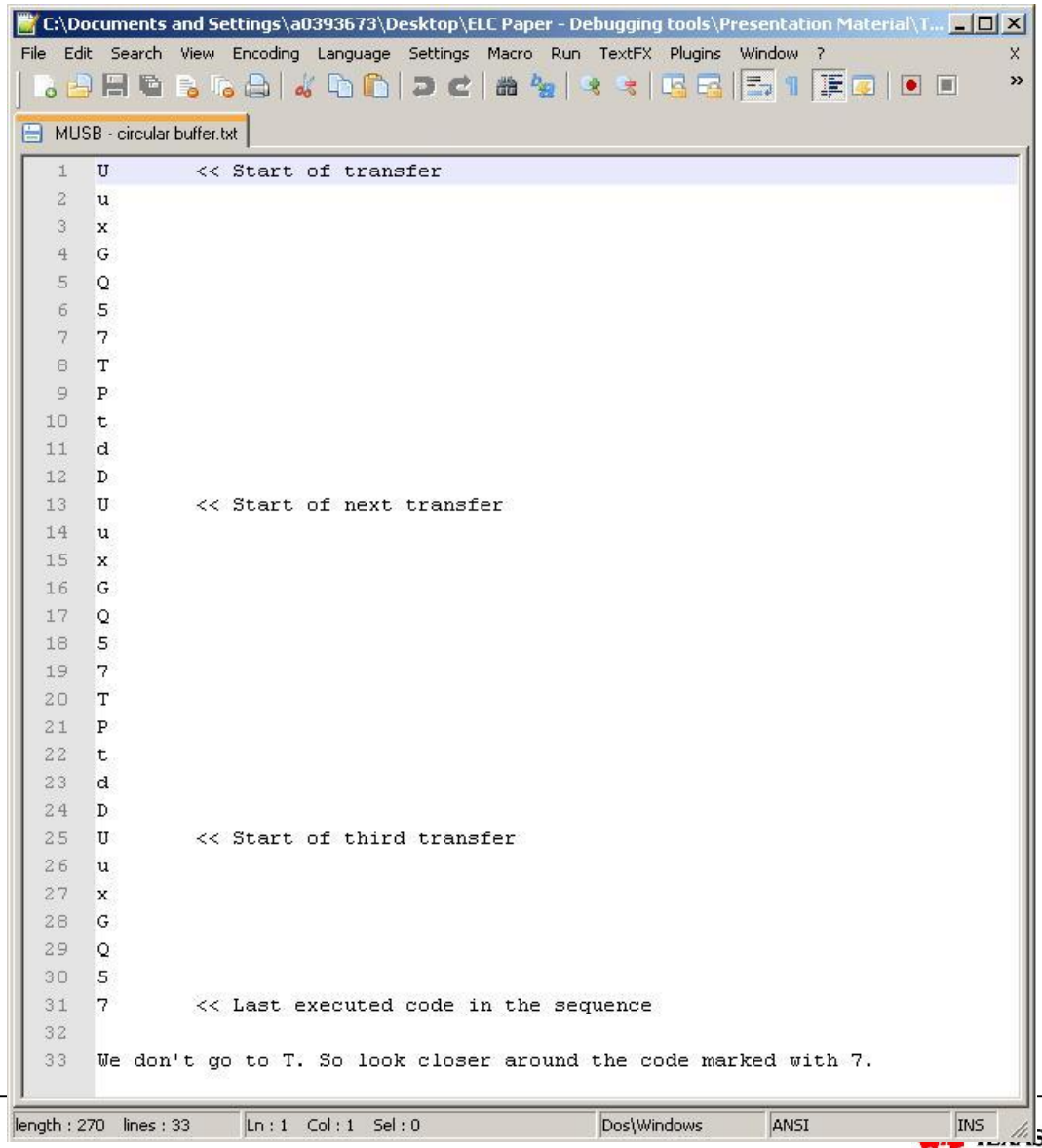
- Turned off prints, and selectively enabled key prints
  - No luck – still hard to reproduce
- Set up a circular buffer to which I sprintf interesting variables
  - read from debugfs when the issue is reproduced
  - No luck – failure disappears

Circular buffers (Case Study - MUSB)

- Set up a circular buffer to hold a single character
  - Instrument code to write a single character to this buffer at interesting points in the code
  - Dump this buffer when we hit the failure

- Bingo!
  - Hit the failure, and still have a good trace of the program flow
  - Now we know where to look

Circular buffers (Case Study - MUSB)



SW Trace Tools

- Tracepoints and markers
- Ftrace
- LTTng
- Perf

Protocol Analyzers

- USBMON
- Wireshark

- What about other protocols?
  - I2C, MMC, SPI, …?

HW trace – ETM/ETB

- What is ETM
  - Embedded Trace Macrocell
- The ETM can capture the program counter value upon certain events (waypoints)
  - A waypoint is a point where instruction execution may change the program flow
    - Branch instructions
    - Exceptions

HW trace – ETM/ETB

- ETB
  - ETB is on SoC buffer
  - ETB buffer is usually small – 2k to 8k
    - (about 10-30k lines of code)
- ETM
  - Streaming same trace content to an external trace port
  - Needs to be continuously read by an ‘external trace receiver’

HW trace – ETM/ETB

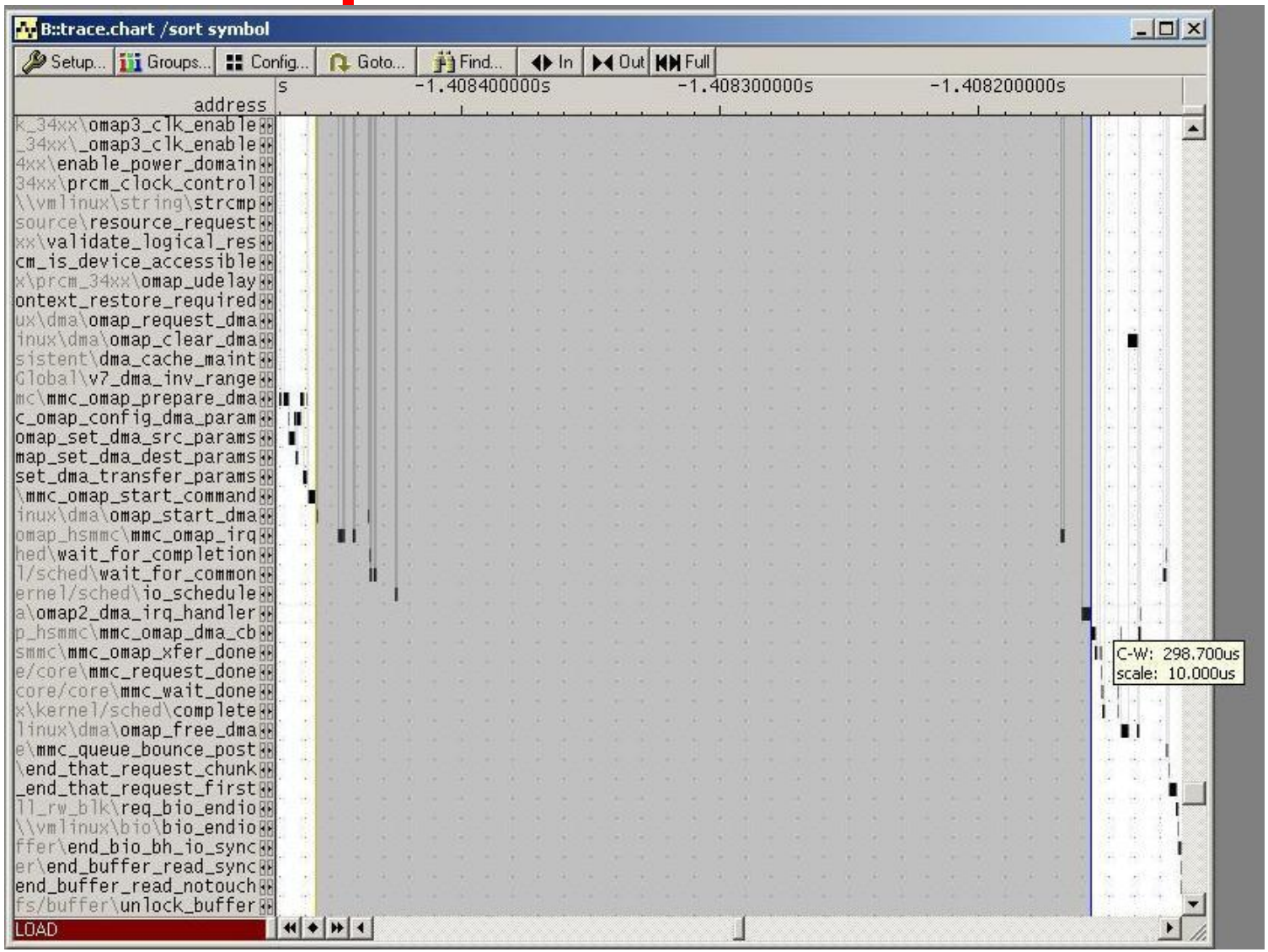
- ETM
  - Needs JTAG Debugger
  - Needs external trace receiver

- ETB
  - Can be dumped using just a JTAG debugger
  - Can be dumped using software
    - See kernel driver for ETB/ETM
      - arch/arm/kernel/etm.c
    - Analysis software:
      - https://github.com/virtuoso/etm2human

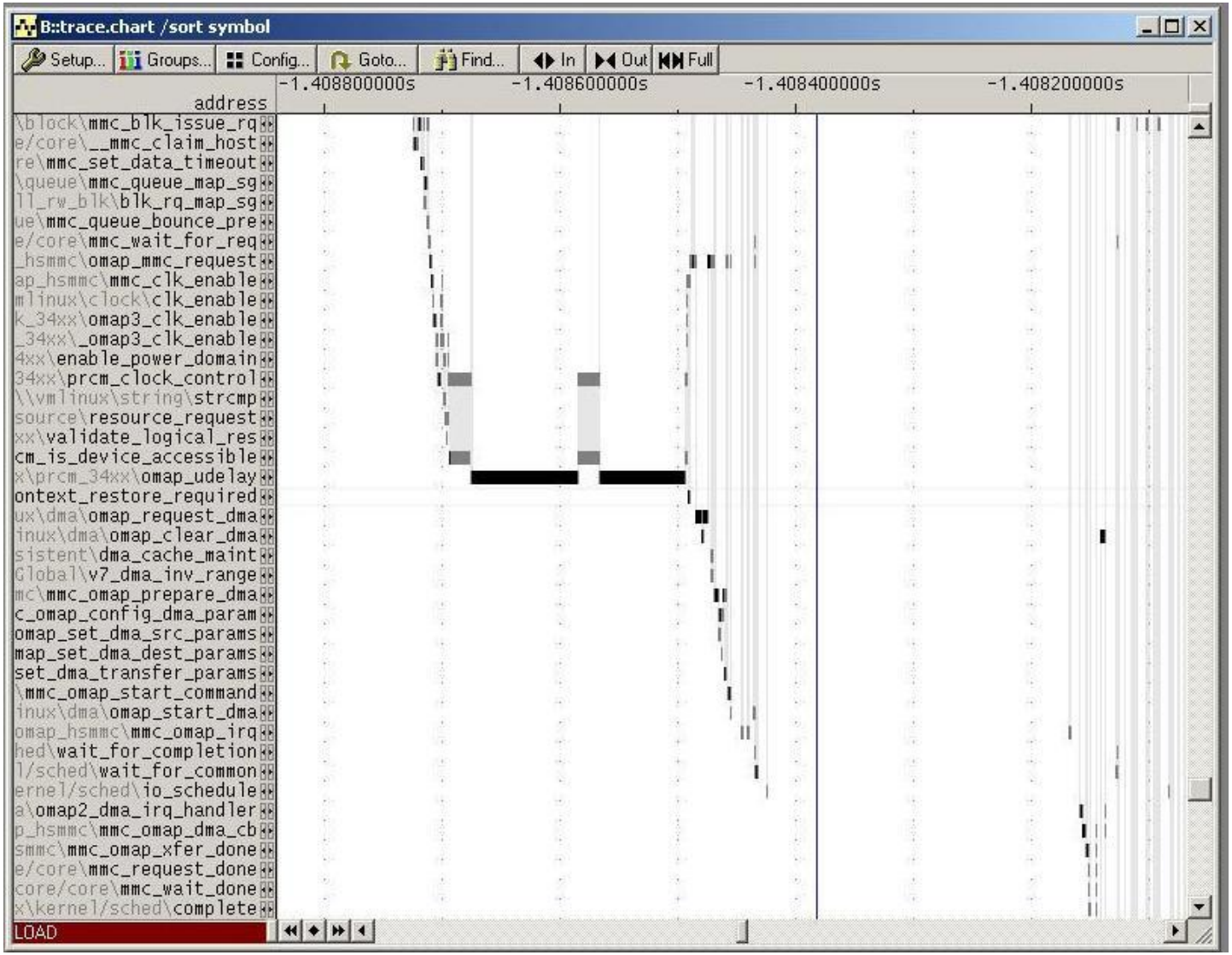
HW trace – ETM/ETB

- Why is it useful
  - Very accurate profiling
  - No need to instrument the code
  - Can be used to reconstruct program flow
  - Can step back in code
    - How?

ETM - Example


B::trace.chart /sort symbol
Setup...
Groups...
Config...
Goto...
Find...
In
Out
Full
address
s
-1.408400000s
-1.408300000s
-1.408200000s
k_34xx\omap3_clk_enable
_34xx_omap3_clk_enable
4xx\enable_power_domain
34xx\prcm_clock_control
\\vmlinux\string\strcmp
source\resource_request
xx\validate_logical_res
cm_is_device_accessible
x\prcm_34xx\omap_udelay
ontext_restore_required
ux\dma\omap_request_dma
inux\dma\omap_clear_dma
sistent\dma_cache_maint
Global\v7_dma_inv_range
mc\mmc_omap_prepare_dma
c_omap_config_dma_param
omap_set_dma_src_params
map_set_dma_dest_params
set_dma_transfer_params
\mmc_omap_start_command
inux\dma\omap_start_dma
omap_hsmmc\mmc_omap_irq
hed\wait_for_completion
l/sched\wait_for_common
ernel/sched\io_schedule
a\omap2_dma_irq_handler
p_hsmmc\mmc_omap_dma_cb
smmc\mmc_omap_xfer_done
e/core\mmc_request_done
core/core\mmc_wait_done
x\kernel/sched\complete
linux\dma\omap_free_dma
e\mmc_queue_bounce_post
\end_that_request_chunk
_end_that_request_first
ll_rw_blk\req_bio_endio
\\vmlinux\bio\bio_endio
ffer\end_bio_bh_io_sync
er\end_buffer_read_sync
end_buffer_read_notouch
fs/buffer\unlock_buffer
C-W: 298.700us
scale: 10.000us
LOAD

ETM - Example

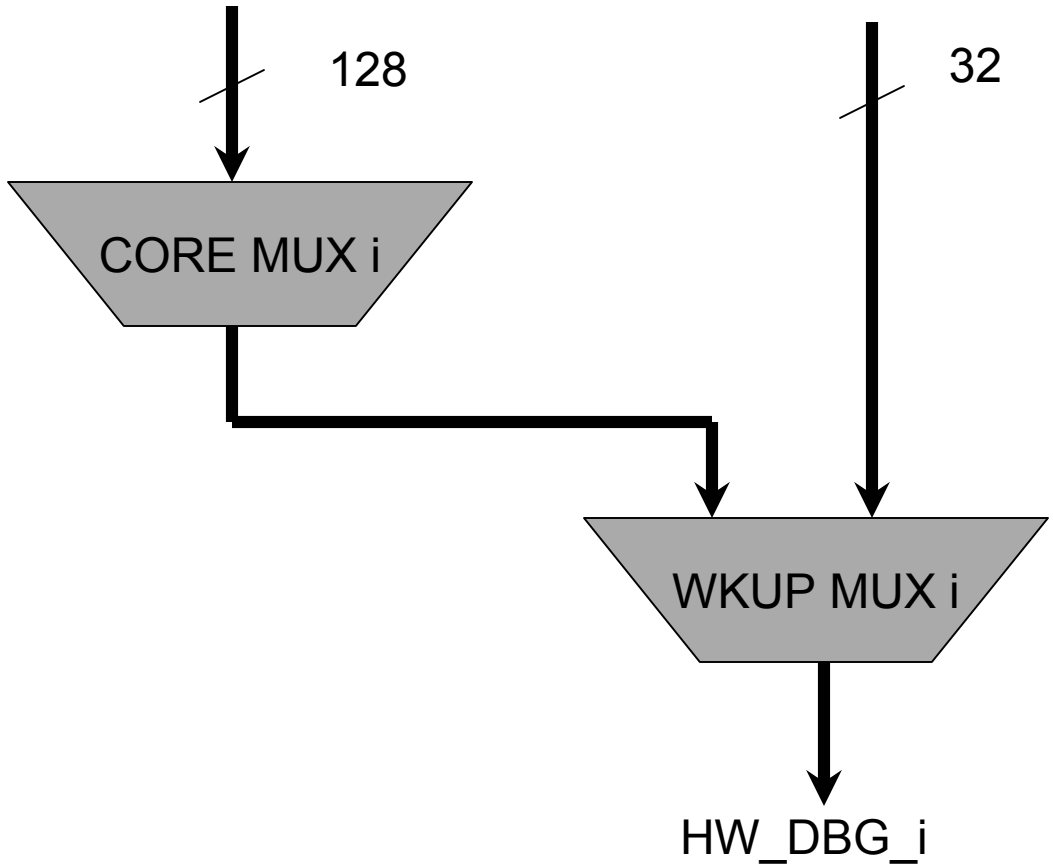


Texas Instruments

Observability of internal signals

- Some SoCs expose internal signals (DMA request lines, interrupt request lines, …) to the outside world
- Since there are a limited number of pads on an SoC, there is usually a way to configure which signal one wants to export out
  - Once configured, these signals can be observed on the corresponding pad

Example – Observability on OMAP3



i = 0..17

Example – Observability on OMAP3

- What is available
  - Internal clocks
  - IRQ lines (any IRQ - up to 4 at a time)
  - DMA request lines (up to 4 at a time)
  - Power domain status
  - Wakeup events

  - Tie high
  - Tie low
    - Useful to check if the pin muxing and other settings are configured correctly
    - and to check if you're actually observing the correct line

    - Also useful as general purpose GPIOs without going through the GPIO module ☺

GPIO markers

- Toggle GPIOs at interesting points in the code
  - Observe with a scope (or even better, a logic analyzer)

- Why is this technique needed?
  - No need to depend on time counters in the SoC
  - Time resolution offered by scope/LA is much better

  - Can trigger on bus events + software conditions
    - Can cross-trigger JTAG debugger to halt the CPU as well

Observability and GPIO markers

- Advantages
  - Good way to extract timing information (for debug and profiling both), without deeply affecting the system
  - Code instrumentation is simpler – may boil down to a simple register write
  - Good profiling tool
    - Especially when combined with ETM

- Disadvantages
  - Cannot get values of variables/parameters
  - No framework - easy for debugger to make mistakes?
  - May not have enough spare pins
    - Sometimes pads are not accessible on near-production boards
  - Scope/LA are expensive
    - especially the good ones

GPIO markers – Tips and Tricks

- Toggle each GPIO before starting to debug - to make sure the setup is right
- Beware: opposite drives and possible board damage

- Toggling GPIOs from userspace
  - Documentation/gpio.txt
    - See “Sysfs Interface for Userspace” section

Observability – Tips and Tricks

- Logic analyzer configuration
  - Use transitional storage mode
    - Don’t observe unnecessary clock signals if you want to capture for a long duration
- Test your setup before starting
  - Toggle all signals manually
    - Preferably one at a time, or in a pattern
    - Check both high and low

Using LEDs for debug

- Useful for initial board bringup
- Very useful to use these in bootloaders
  - in case of a crash before the UART comes up
- No need for scope/LA
  - Not useful for timing information
  - Very useful if all you need to know is state information

- Heartbeat LEDs
  - (don't enable in production – they drain power)

JTAG

- Examples:
  - Lauterbach Power Debug
  - ARM Realview ICE, ARM DS/5
  - XDS560

  - Flyswatter
  - OpenOCD

JTAG – tips and tricks

- Lauterbach PER files
  - Decode register dumps
- The while(1) loop
  - Sometimes you cannot connect with JTAG when the CPU is powered down in idle paths
    - Workaround: add a while(1) loop after CPU powers up. Connect with JTAG here, and then skip to the next instruction
- Read/write breakpoints on variables
  - Useful for debugging memory corruption

JTAG – tips and tricks

- Console over JTAG
  - CONFIG_HVC_DCC
  - CONFIG_DEBUG_ICEDCC

  - Introduced in kernel in which version?

- Extracting dmesg buffer over JTAG

Basic register access utilities

- Register access
  - omap_readl/writel
  - readmem, devmem2
  - i2c-utils

Register decoders

- Example
  - pxaregs
- Register dump scripts
  - Simple userspace scripts can be built around these utilities
  - Example
    - ehcidump.sh

Register decoders

- Exporting register info in debugfs
  - Example - MUSB in debugfs