Simulation Driver Driver Develpment

Michael Davidsaver Osprey DCS



How, When, ... Why?

- Test driven development is great!
- How to do this with hardware drivers?
- Cost/benefit?



Case studies

- HPI 6012/6016 radiation monitor
 - Tx only serial/ethernet
- **FEED** (FPGA Embedded Ethernet Driver)
 - UDP request/response
- CaenELS PICO8
 - uTCA
- MRF EVR/EVG

- VME



HPI 6012/6016

- Counter for use with various detector chambers
- Sends one line of text at 1Hz
- Simple right?



HPI 6012/6016 (2)

- Safety "related" device
- Closely scrutinized
 - NSLS2 beam mis-steering incident
- Develop simulator and driver concurrently
- Cross-test driver with simulator and real HW
 - Can test "impossible" faults
 - Part of (re)validation after driver changes



HPI 6012/6016 (3)

- Simulator
 - TCP server
 - GUI interface
- Simulate
 - Dose
 - Device alarms
 - Comm. faults

Dose Rate 0.00 \$ EEPROM Content Dose 0.00 \$ 0x00 255 Level 1 5.00 \$ 0x01 10 Level 2 1.00 \$ 0x02 10 Level 3 0.00 \$ 0x03 10 HVP Test Fail 0x05 3 Pause Tx 0x06 0 Inject byte 0x08 1 Inject fault 0x09 244 Ox0a 0 0x0b 0 Original FW 0x06 0 0x06	🐹 🗶	HPI60	16 Simula	ation	~ ^	
Dose 0.00 \$ Value Level 1 5.00 \$ 0x00 255 Level 2 1.00 \$ 0x02 10 Level 3 0.00 \$ 0x03 10 Level 3 0.00 \$ 0x04 3 HVP Test Fail 0x05 3 Pause Tx 0x06 0 Inject byte 0x08 1 Inject fault 0x09 244 Ox04 0 0 Original FW 0x06 0	Dose Rate		b.oo 🗘	EEPR	OM Conte	nts
Level 1 5.00 0x00 255 0x01 10 Level 2 1.00 0x02 10 Level 3 0.00 0x03 10 0x04 3 0x05 3 0x05 3 0x06 0 HVP Test Fail 0x06 0 NUP Test 0x06 0 Inject byte 0x08 1 Corrupt byte 0x0a 0 Original FW 0x0c 100	Dose		0.00 🗘 Value			e
Level 1 5.00 \$ 0x01 10 Level 2 1.00 \$ 0x02 10 Level 3 0.00 \$ 0x03 10 HVP Test Fail 0x05 3 Pause Tx 0x06 0 HVP Test 0x06 0 Inject byte 0x08 1 Corrupt byte 0x0a 0 Original FW 0x0c 100	Lough	5.00 (0x00	255	- [
Level 2 1.00 \$ 0x02 10 Level 3 0.00 \$ 0x03 10 HVP Test Fail 0x05 3 Pause Tx 0x06 0 HVP Test 0x07 0 Inject byte 0x08 1 Corrupt byte 0x0a 0 Original FW 0x0c 100	Level 1	5.00		0x01	10	
Level 3 0.00 \$ 0x03 10 HVP Test Fail 0x04 3 Pause Tx 0x05 3 HVP Test 0x06 0 Inject byte 0x08 1 Inject fault 0x09 244 Corrupt byte 0x0a 0 Original FW 0x0c 100	Level 2	1.00 🔇	2	0x02	10	
HVP Test Fail 0x04 3 Pause Tx 0x05 3 HVP Test 0x06 0 Inject byte 0x08 1 Inject fault 0x09 244 Ox0a 0 0 Original FW 0x06 0	Level 3	0.00 🔇	2	0x03	10	
Pause Tx 0x05 3 HVP Test 0x06 0 Inject byte 0x07 0 Inject fault 0x09 244 Corrupt byte 0x0a 0 Original FW 0x0b 0	HVP Te	st Fail		0x04	3	
HVP Test 0x06 0 Inject byte 0x07 0 Inject fault 0x08 1 Corrupt byte 0x0a 0 Original FW 0x0b 0	Pause T	x		0x05	3	
Invertext0x070Inject byte0x081Inject fault0x09244Corrupt byte0x0a0Original FW0x0b0				0x06	0	
Inject byte0x081Inject fault0x09244Corrupt byte0x0a0Original FW0x0b00x0c100		IIVI Test		0x07	0	
Inject fault0x09244Corrupt byte0x0a0Original FW0x0b00x0c100	I	nject byte		0x08	1	
Corrupt byte0x0a0Original FW0x0c100	Inject fault			0x09	244	
Original FW 0x0b 0 0x0c 100	Corrupt byte			0x0a	0	
Original FW V 0x0c 100	contrape byte			0x0b	0	
	Original FV	V	~	0x0c	100	~
### 000000 01f4 0064 0000 3c 00	###		000000	01f4 006	54 0000 3c	00
· · · · · · · · · · · · · · · · · · ·				4		



Output text

HPI 6012/6016 (4)

Testing process

Initial testing uses the hpisim3.py software device simulation.

Repeat for both Original FW and Integrated Dose modes

Return the dose rate to 0## Verify that alarms clear

Reduce simulation dose rate to -0.05

Verify that the fail holdoff counter begins counting down ## Verify that \$(P)Alrm:Fail-Sts becomes active when the counter reaches zero.

🐹 🗶	HPI6016 Simulation 🛛 🗸 🗙					
Dose Rate	þ.oo 🗘		EEPROM Contents			
Dose	0.00 ^		Value 🏠			
0050	0.00 V		0x00	255		
Level 1	5.00 🗘		0x01	10		
Level 2	1.00 🗘		0x02	10		
Level 3	0.00 🗘		0x03	10		
🗌 HVP Te	HVP Test Fail			3		
Pause Tx			0x05	3		
HVP Test		٦	0x06	0		
			0x07	0		
Inject byte			0x08	1		
Inject fault		1	0x09	244		
Corrupt byte		ī	0x0a	0		
contape syste			0x0b	0		
Original F	N ~		0x0c	100 ~		
###	000000 01f4 0064 0000 3c 00					



...

FEED (FPGA Embedded Ethernet Driver)

- UDP/IP request → response
- Register based
- Introspect device to get register name \leftrightarrow address mapping.
 - Compressed JSON encoded in ROM
- Develop simulator and driver concurrently
- Cross-test driver with simulator and real HW
 - Help identify FW (doc) issues
 - Test (re)connection and timeout behavior
 - Limited access to shared test stand



FEED Packet Format

- 8 byte header echoed
- 26 bit address space
- Only 4 byte access
- Read and Write
 formats identical

	0x0	0x1	0x2	0x3	
0x00	Header/Tag				
0x04					
0x08	Bits	Address 1			
0x0C	Data 1				
0x10	Bits Address 2				
0x14	Data 2				



FEED simulator

./bin/linux-x86_64/feedsim -h Usage: ./bin/linux-x86_64/feedsim [-hd] [-H <iface>[:<port>]] [-L none|rfs] [-S <sec>] <json_file> [initials_file]

- Load register description from file
- Optionally
 - Provide initial register values
 - Firmware specific logic

eg. waveform arm/wait/readout sequence



CaenELS PICO8

• uTCA fast digitizer (pico ammeter)

- PCIe w/ interrupts and DMA

- Develop simulator and driver(s) concurrently
 - Update Linux kernel module
 - Create EPICS driver
- FW bug in DMA handling
 - Developed driver (mostly) in advance of working HW



QEMU Device Models

- QEMU emulator provides callbacks on MMIO
 - static uint64_t sis_read(void *opaque, hwaddr addr, unsigned size) { ...
 - static void sis_write(void *opaque, hwaddr addr, uint64_t val, unsigned size) { ...
 - Lots of helpers for PCI device mechanics
- "Ultimate" bus analyzer

Everything except (most) timing!

\$ qemu-system-x86_64 -device amc-pico-8,?

amc-pico-8.addr=int32 (Slot and optional function number, example: 06.0 or 06) amc-pico-8.frib=bool

```
$ qemu-system-x86_64 -device amc-pico-8 ...
```

Osprey DCS

Madala interrupta and	Language	files	blank	comment	code
Scatter DMA	С	1	108	71	549

https://github.com/mdavidsaver/qemu/blob/vme/hw/misc/caen_pico8.c

MRF EVR/EVG

- Event timing cards
 - Various PCI/PCIe and VME
- Develop simulator with existing driver(s)
- Validate driver changes w/o access to HW



MVME3100 with QEMU

- Modeling MVME3100
 - e500v2 CPU already supported
 - Start from existing mpc8544ds
 - Add I²C controller, eeprom, and RTC
 - Bootloader w/ RTEMS
- Modeling TSI148 PCI ↔ VME bridge
 - Really complicated device!
 - VME bus infrastructure



How, When, Why

- How?
 - Sockets (Serial) is easy
 - MMIO w/ emulator
- When?
 - Need to exercise handling of uncommon errors
 - Lack access to (working) hardware
 - Low-level tracing
 - More predictable, but longer, development time
 - \$\$\$
- Why?



How, When, Why

- How?
 - Sockets (Serial) is easy
 - MMIO w/ emulator
- When?
 - Need to exercise handling of uncommon errors
 - Lack access to (working) hardware
 - Low-level tracing
 - More predictable, but longer, development time
 - \$\$\$
- Why?
 - Simulator is my bubble!



