System/Software Verification Using Specman ISX and Open Virtual Platforms

Henry Von Bank,
Posedge Software
29 July 2009

Software Verification

- Software complexity is increasing exponentially
- Particularly an issue in embedded applications (cell phones, GPS, MP3 players,...)
- Solution?
  - Borrow methodology and tools from HW world, along with virtual platforms
  - Specman+ISX+OVP

Software Verification Using Specman ISX and Imperas M*SIM

- Using Cadence's Specman Elite and Incisive Software Extensions (ISX) along with Open Virtual Platform (OVP) models and Imperas M*SIM tools
- This demo focuses on a System-Level verification task, but similar approach could be used for purely software verification
- Verify software with as few modifications as possible to the target

Why Use Specman for Software Verification?

- Constrained Random Generation
- Reusable components
- Testing interactions with HW
- Many of the same reasons as using Specman for HW verification!

Why Use a Virtual Platform (in particular OVP)?

- Adds FAST software execution to verification environment, instead of slow RTL models
- Run complete OS and SW stack during system-level verification
- Verifying of SW in a controlled environment
- Interception features of OVP allow introspection of running processor/processes with minimal performance impact

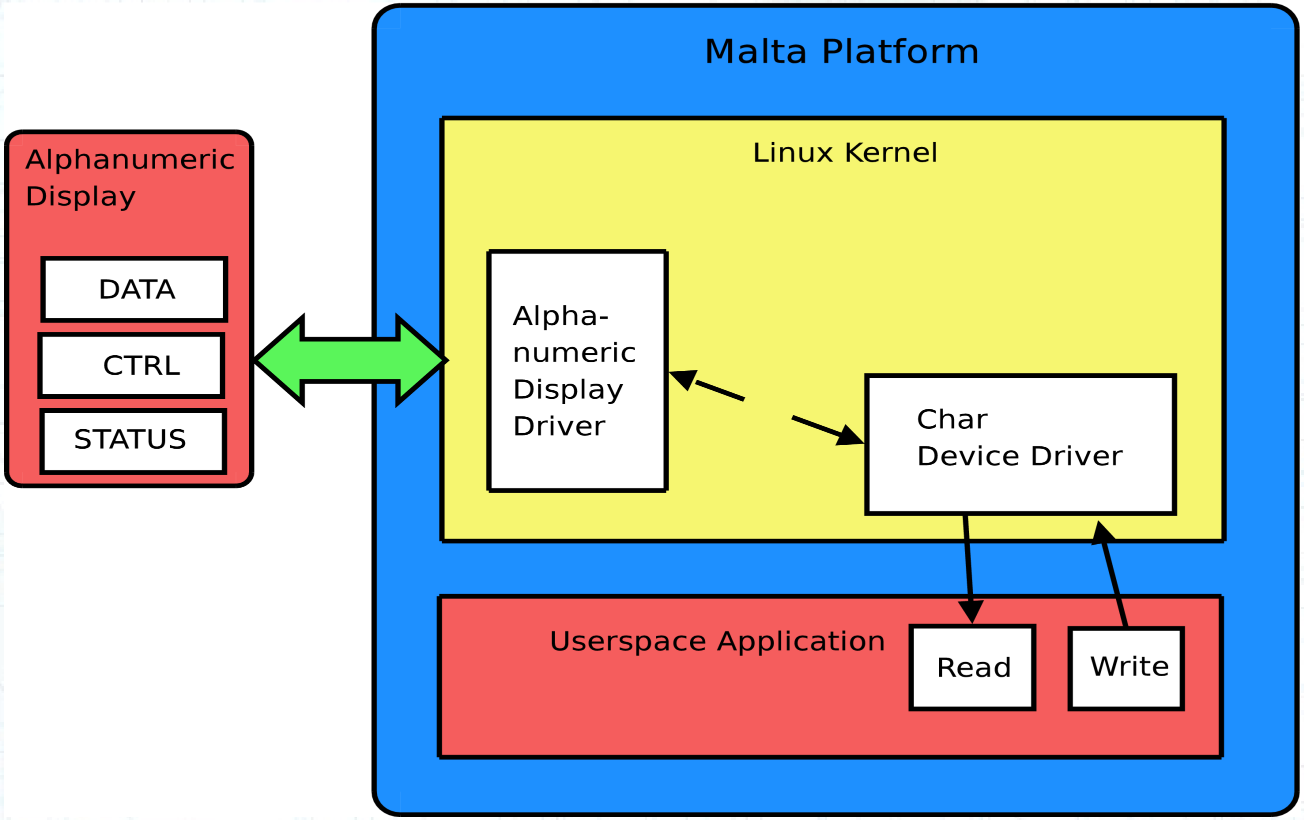
Malta Demo

- MIPS Malta OVP models
- M*SIM Built as shared library loaded into Specman
- Uses Specman C interface for communicating with M*SIM
- Generic Software Adapter (GSA) Mailbox resides in simulated processor's memory
- Shows testing of Linux kernel driver for a “fake” alphanumeric display (16x2 chars)

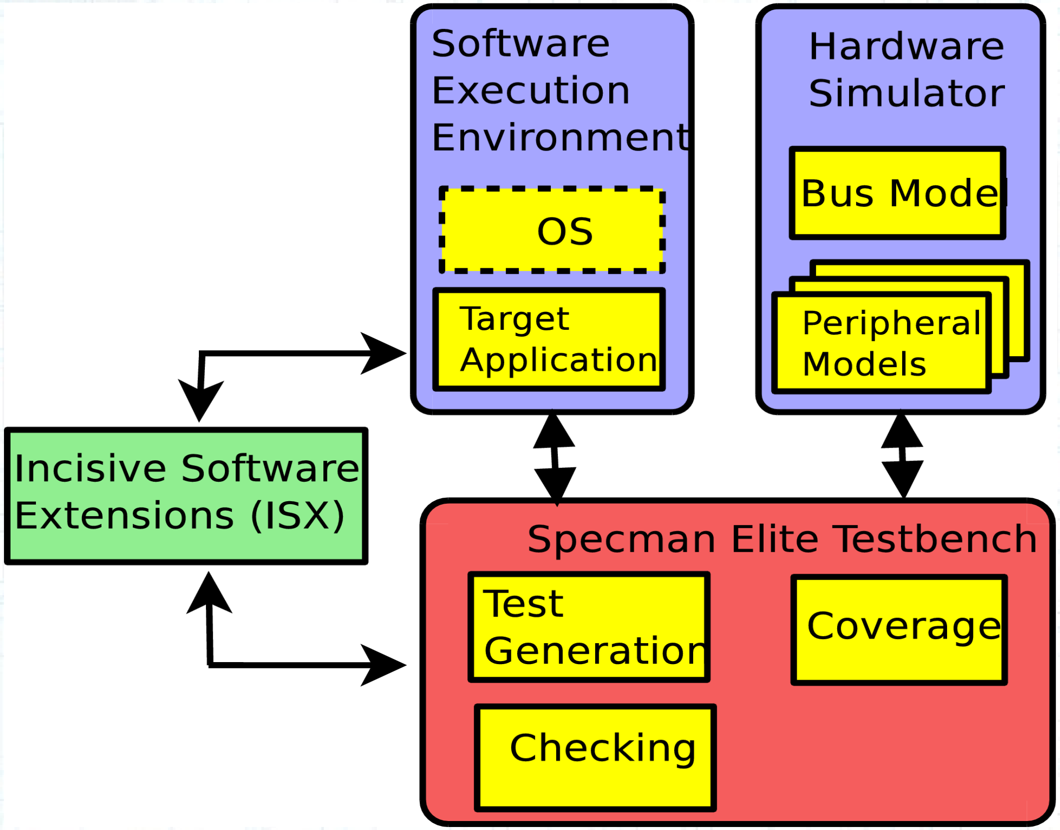
Malta Platform

- Evaluation board from MIPS
- Contains VGA, IDE, Keyboard, Ethernet, and other peripherals
- Supports Linux 2.4 and 2.6
- Full system emulation of Malta supported by a variety of tools including OVP, M* tool suite

Alpha-numeric Display Example

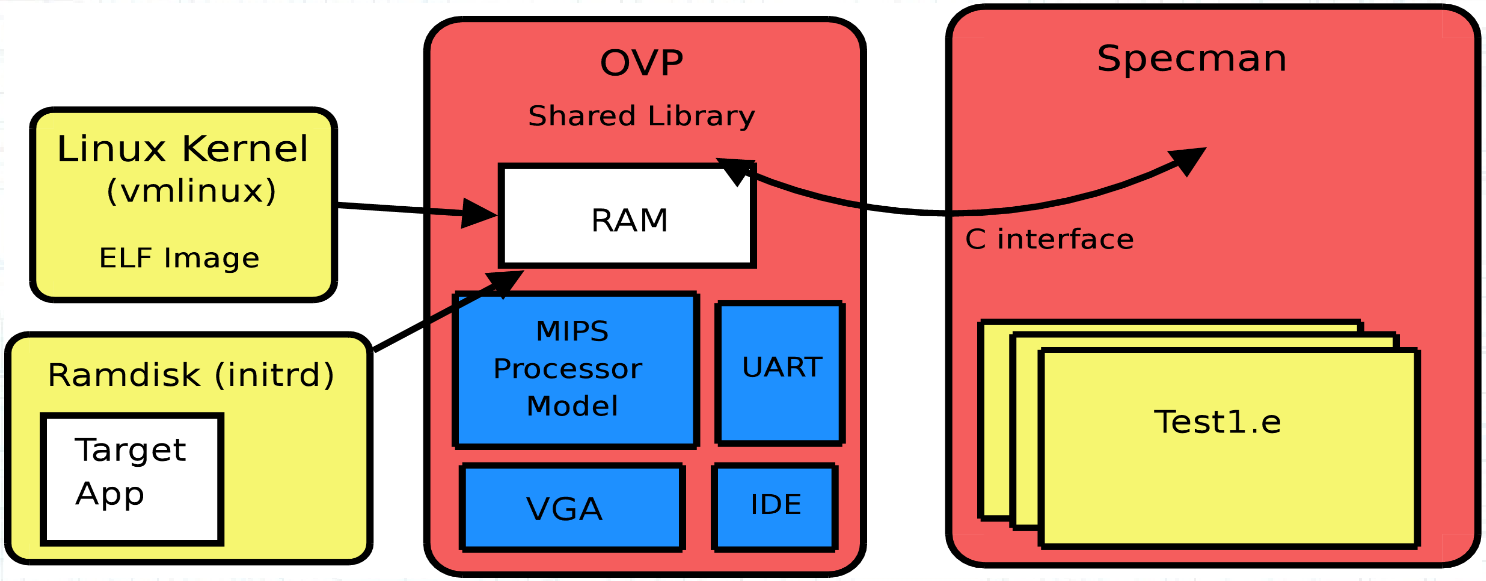


Basic System-Level VE

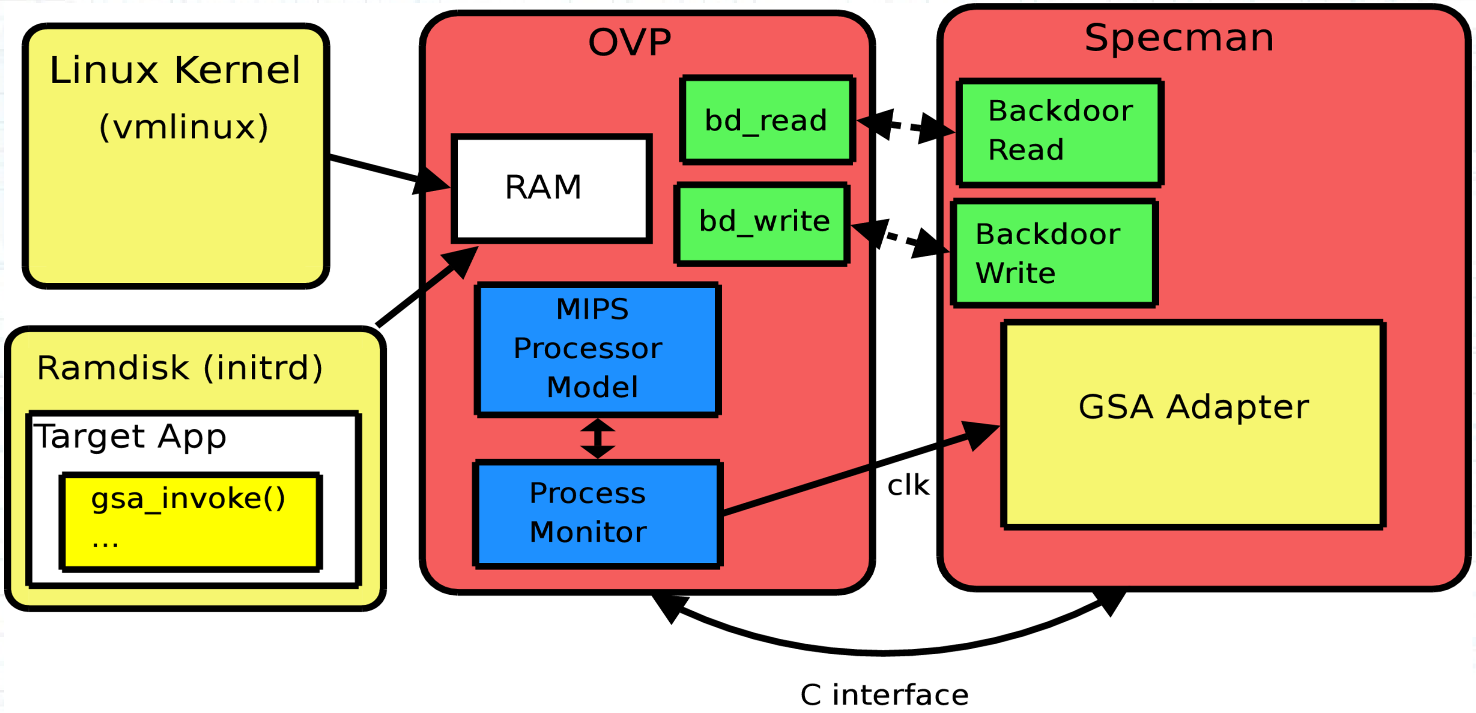


Posedge Software

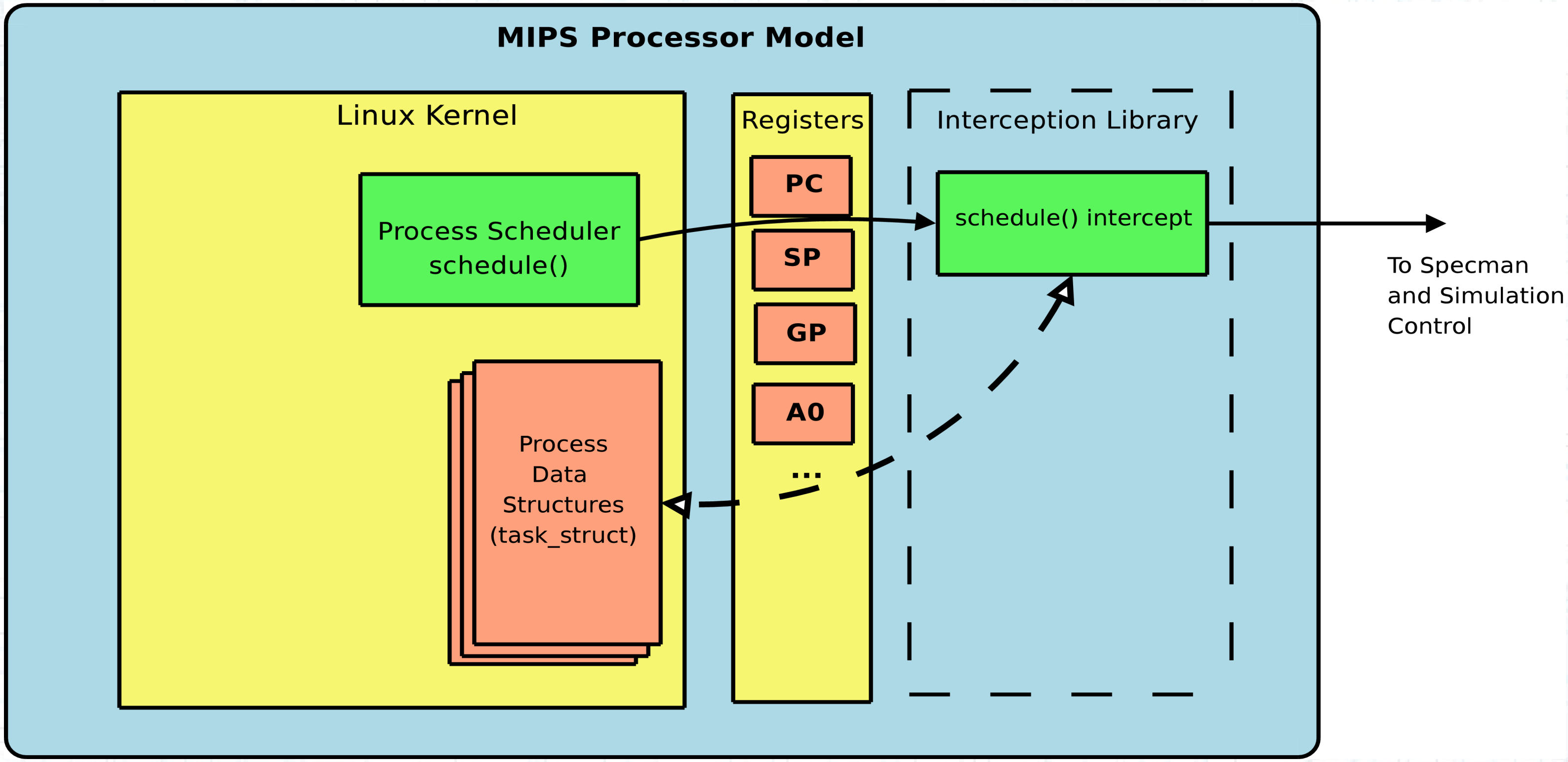
Host/Target Software



GSA Interfacing



Linux Kernel Intercepts



example_test.e

```
do load_module keeping { .filename == "/alphaExample/alpha_drv.ko" };
do open_device keeping { .deviceName == "/tdev/alpha" };
fh = open_device.return_val;

// Send enable command
gen current_packet keeping {.kind == COMMAND;.command == ENABLE;.value == 1};
send(fh,current_packet);


for i from 1 to 10 do {
    //Send random command
    gen current_packet;
    send(fh,current_packet);
    // Send command to set cursor position to random location
    gen current_packet keeping {.kind == COMMAND;.command == ADDRESS};
    send(fh,current_packet);
};

do wait keeping {.delay == 500000};

do close_device keeping { .fh == fh };
do unload_module keeping { .path == "/alphaExample/alpha_drv.ko" };
```

Observations

- Specman and the *e* language provide a robust platform for verification
- ISX works very well for driving stimulus, but not as ideal for monitoring/coverage of SW
- Using M*SIM allows for running/monitoring SW more transparently and with fewer modifications
- SW verification could benefit from having this precise control over the entire platform

Future Work

- Integrating SW coverage info into Specman
- Using multiple GSA adapters, or ISX interface to peripheral models
- Use M*SIM SystemC TLM2.0 interface
- Verifying user-space applications, especially multi-threaded or in a multi-processor system