Facilitating Cross-FPGA Platform Designs Using “Virtual” Platforms

Angshuman Parashar
Michael Adler
Joel Emer

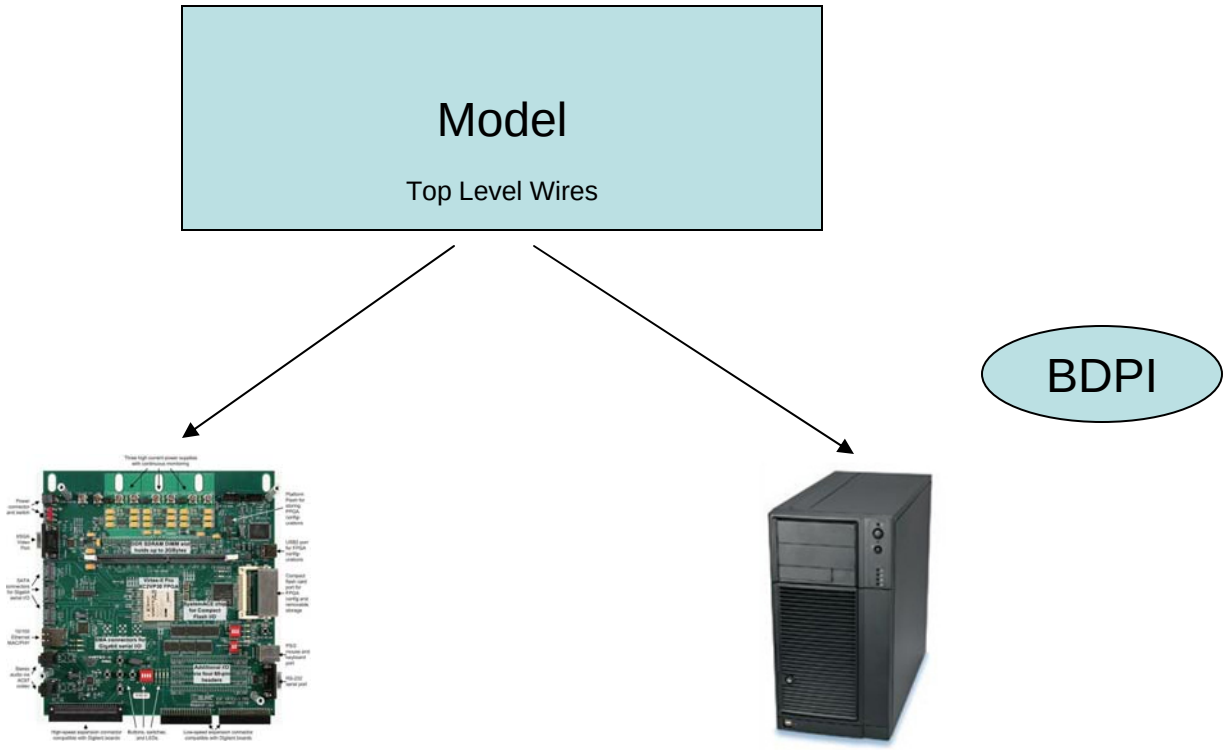
VSSAD, Intel

{angshuman.parashar, michael.adler, joel.emer}
@intel.com

Outline

- History and Motivation
- Virtual Platforms
- Hybrid Hardware/Software Modules
- Demo

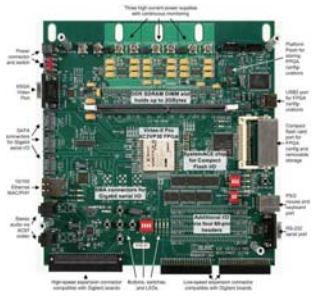
History and Motivation



History and Motivation

XUP Model

Top Level Wires



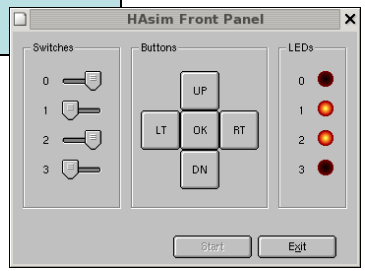
XUP Board

LEDs and Switches

(image source: xilinx.com)

"Sim" Model

BDPI

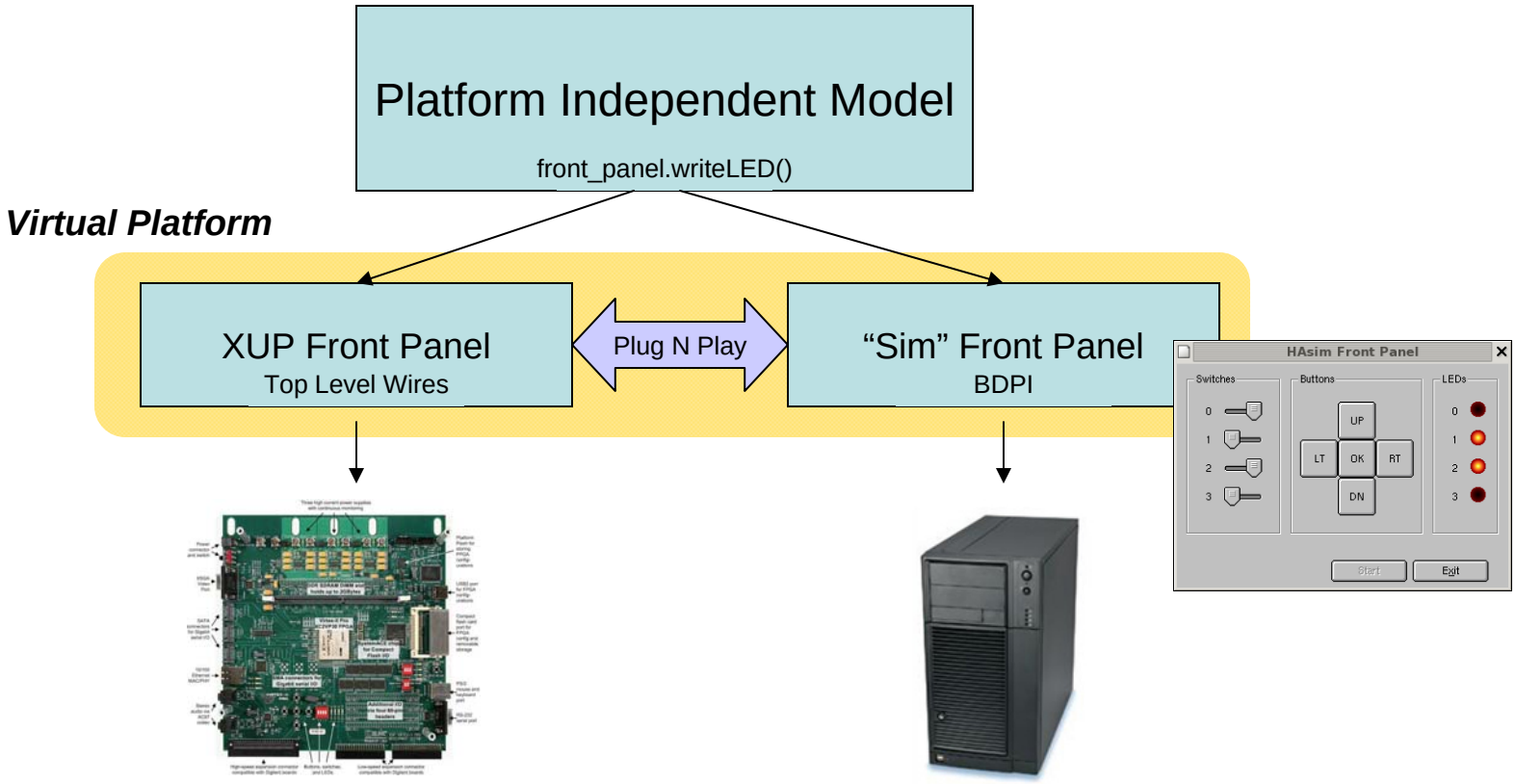


Software Simulator

Processes

(image source: intel.com)

History and Motivation



XUP Board

LEDs and Switches

(image source: xilinx.com)

Software Simulator

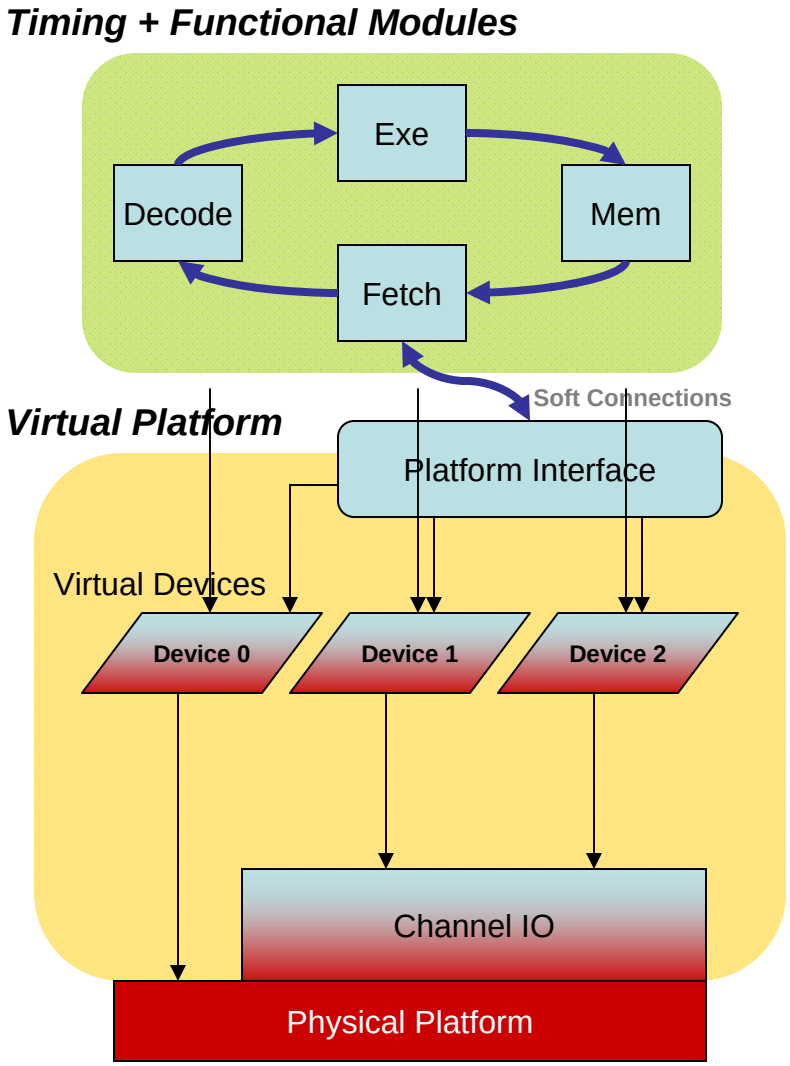
Processes

(image source: intel.com)

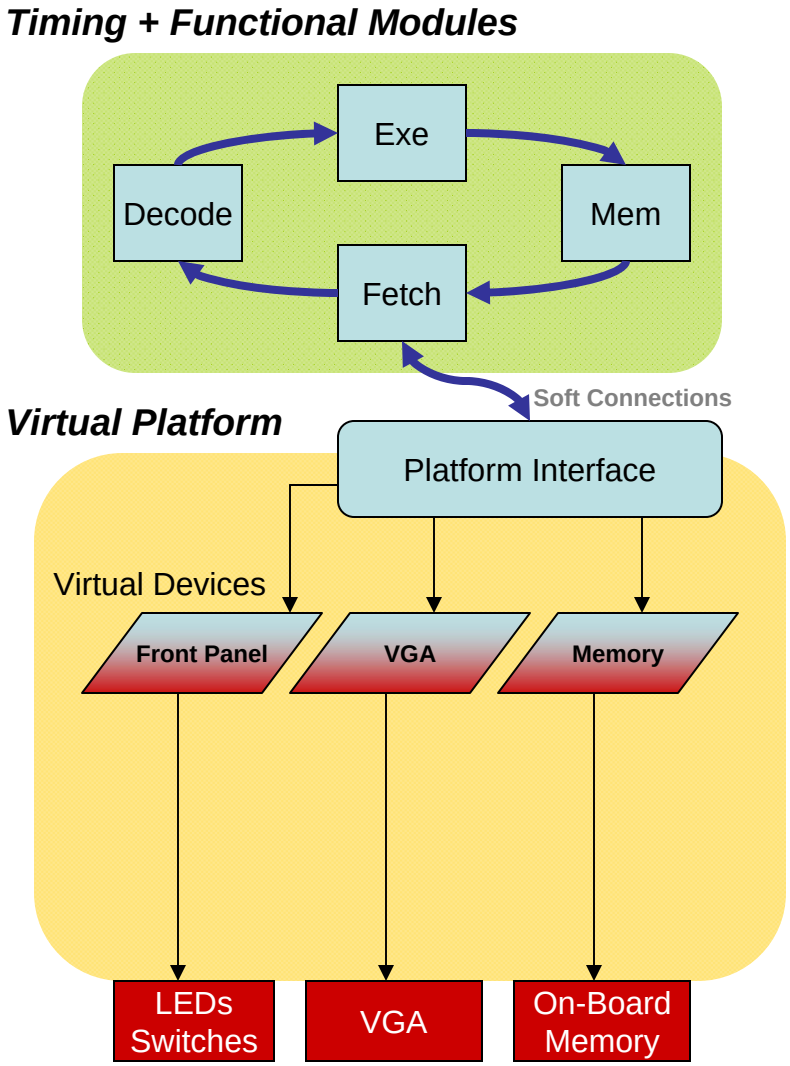
Virtual Platform

- Set of Abstractions
  - Provide common set of functionalities across multiple physical-platforms
    - Intel FSB
    - Bluesim/Vsim
    - PCI-express
    - XUP
  - Leverage Asim Plug N Play
    - Minimize module replacements/recoding while moving across platforms
  - Functionality + Efficiency

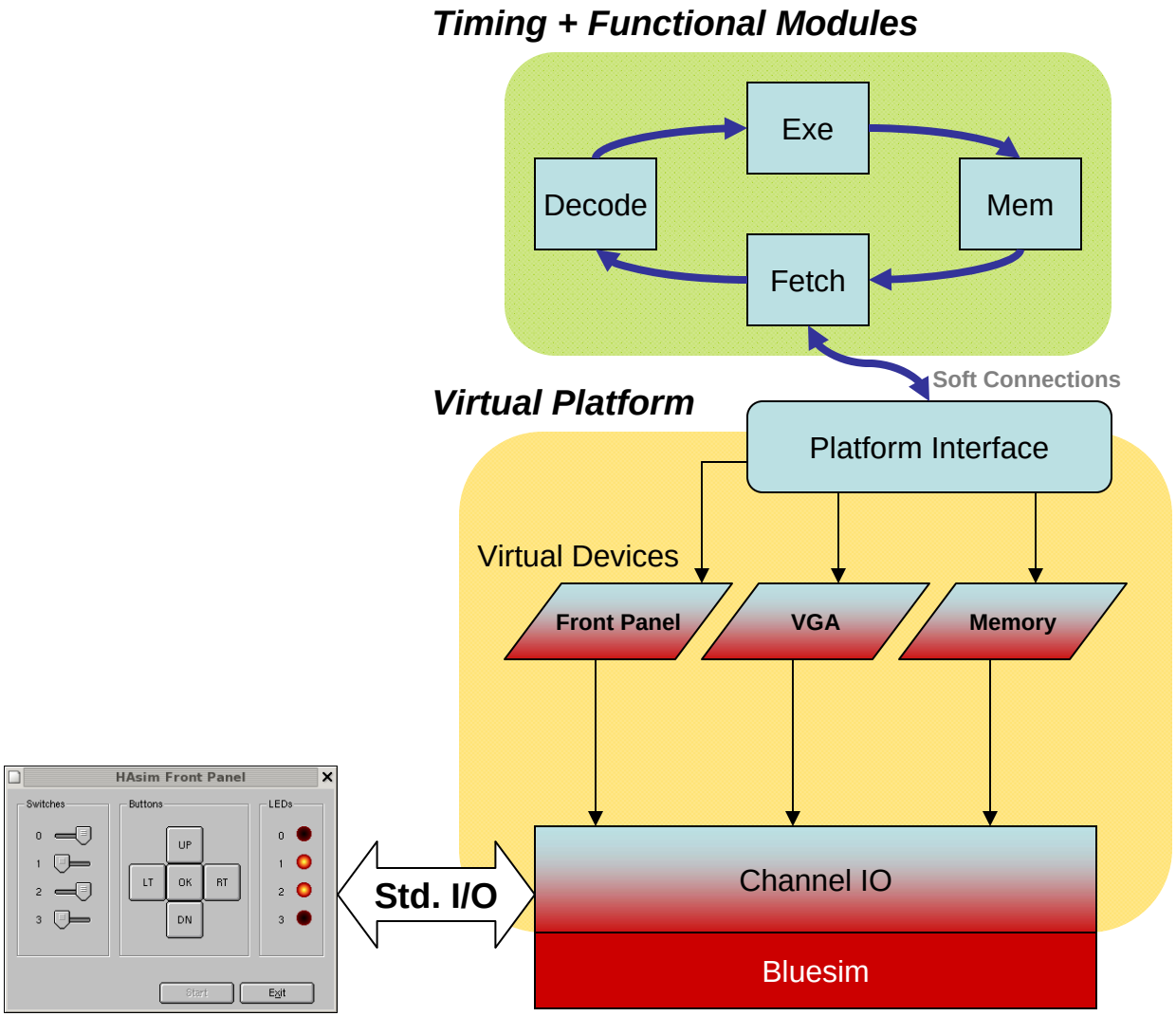
Virtual Platform



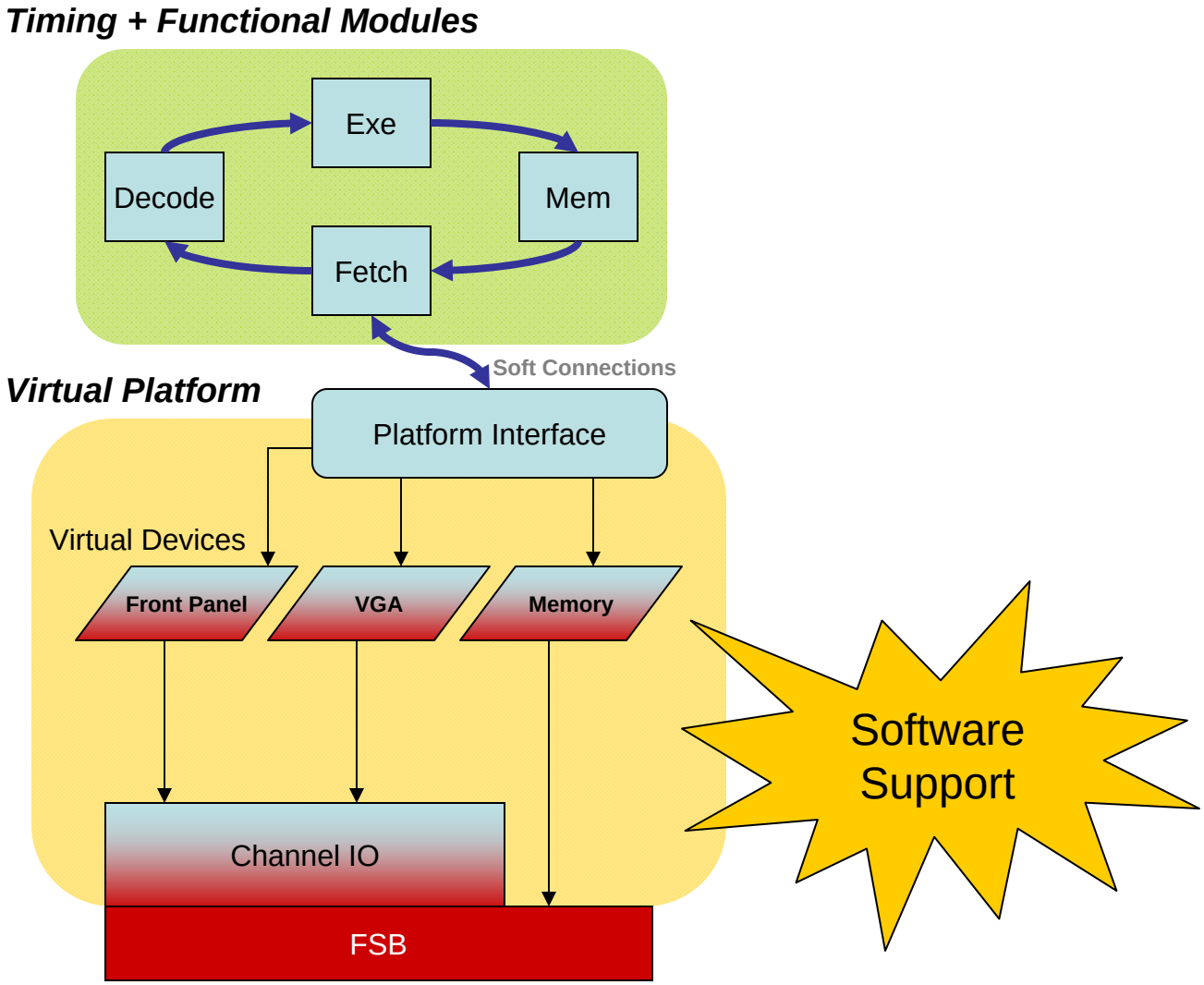
Virtual Platform on XUP



Virtual Platform on Simulator



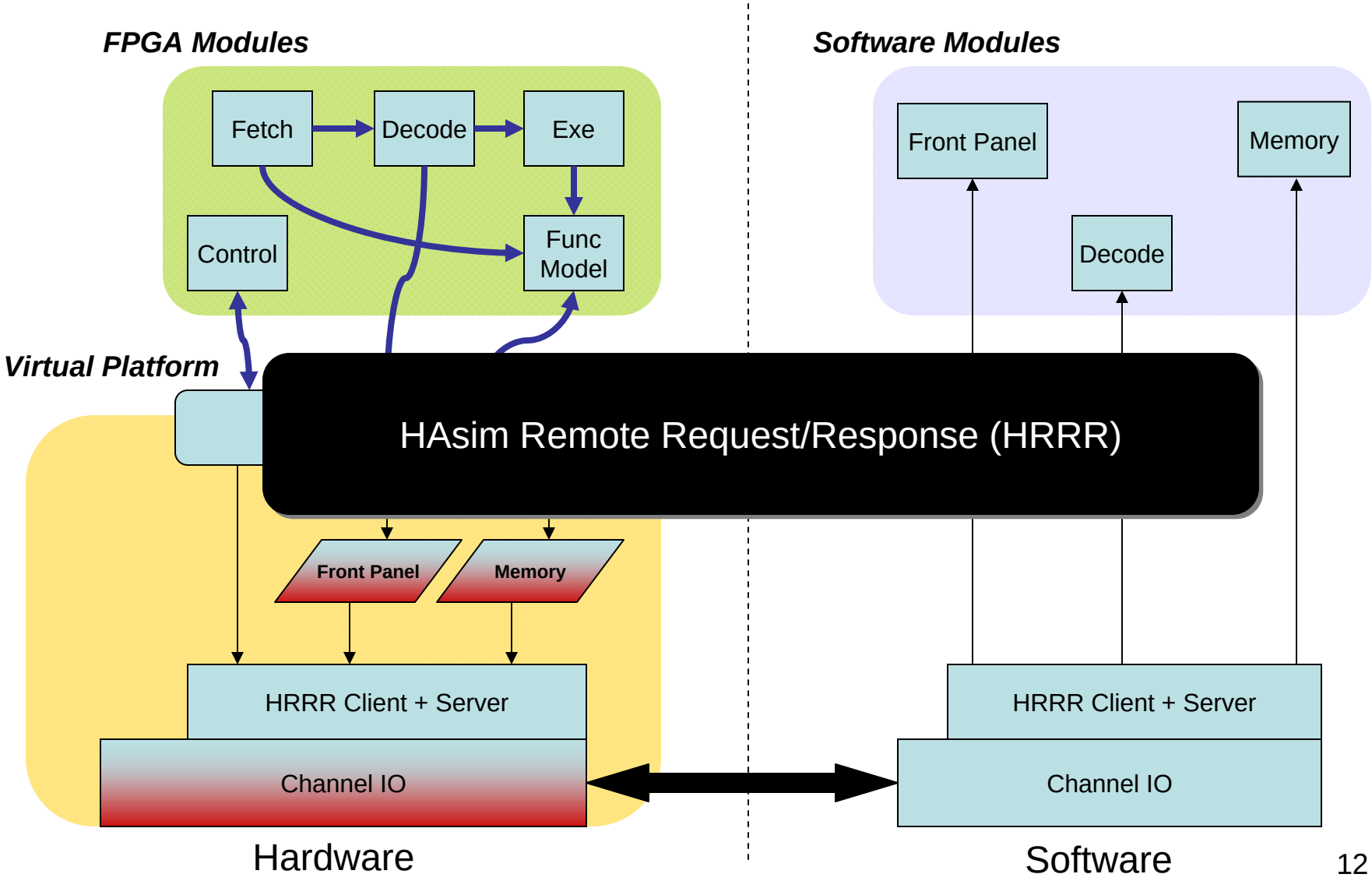
Virtual Platform on Intel FSB



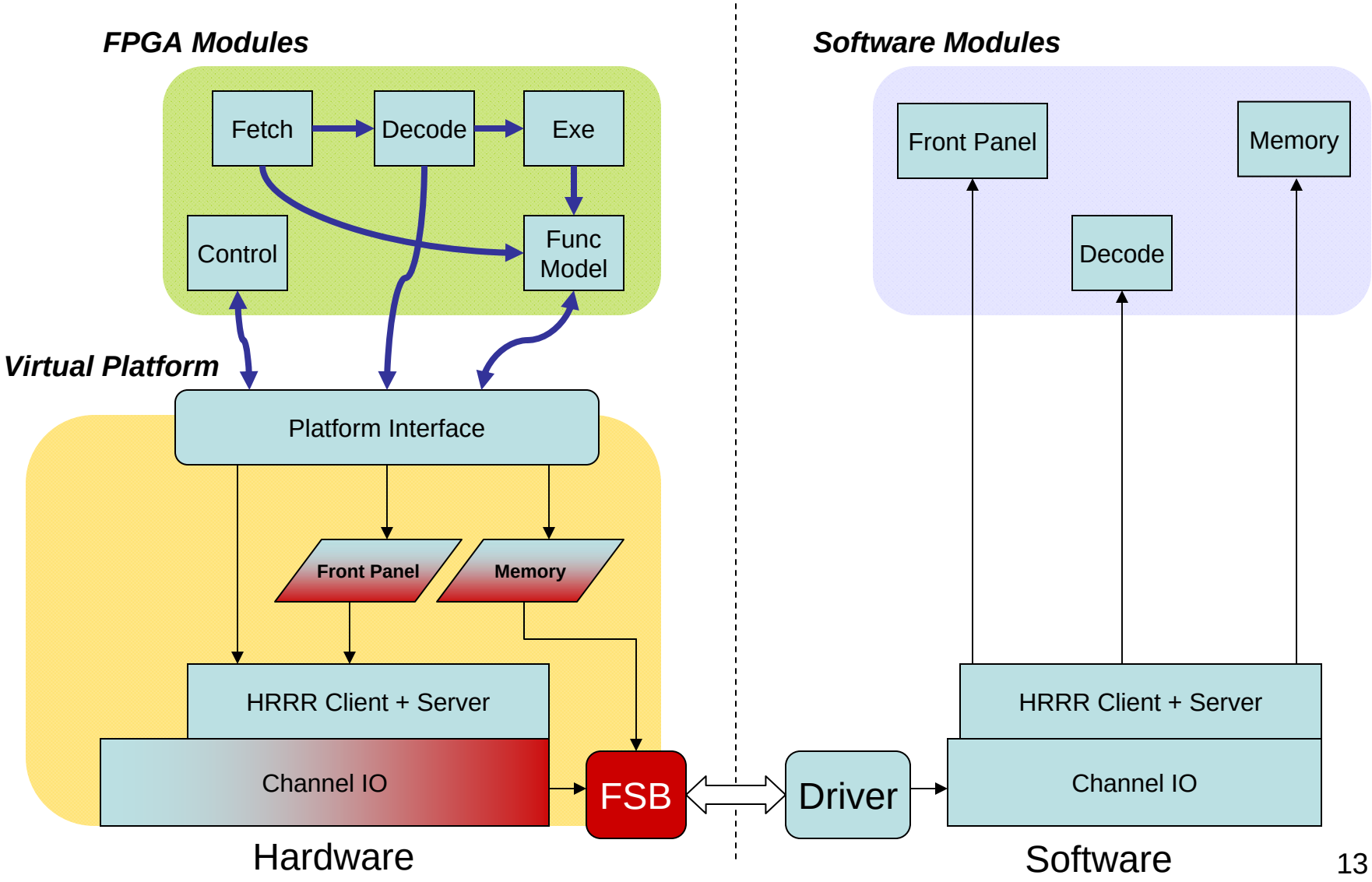
Hybrid Hardware/Software Modules

- Split module functionality between FPGA and software
- Leverage Virtual Platform Infrastructure

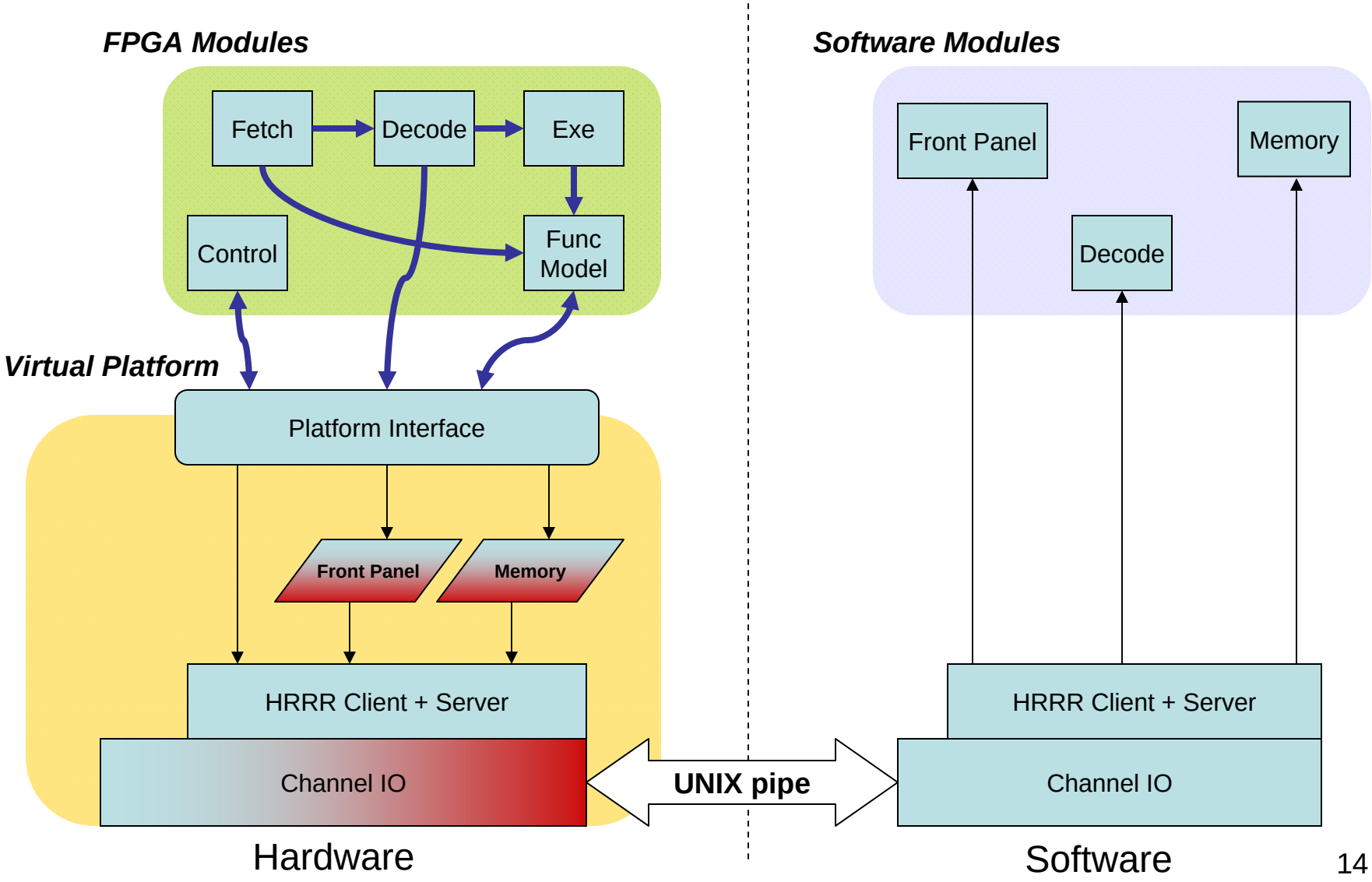
Hybrid Modules



HRRR: Intel FSB



HRRR: Simulator



Hybrid Modules

- Server module
  - Publishes the following:
    - `init()` method name (e.g. `fetch_server_init()`)
    - RRR service string (e.g. “FETCH”)
  - Build process collects these and generates global services table
  - Server main reads services table (during pre-processing) and registers all service modules by calling `init()`
- Client module
  - Reads global services table
  - `serviceID = serviceTable.search(“FETCH”);`
  - `reqID = RRRClient.sendReq(serviceID, params…)`
  - `result = RRRClient.getResp(reqID);`

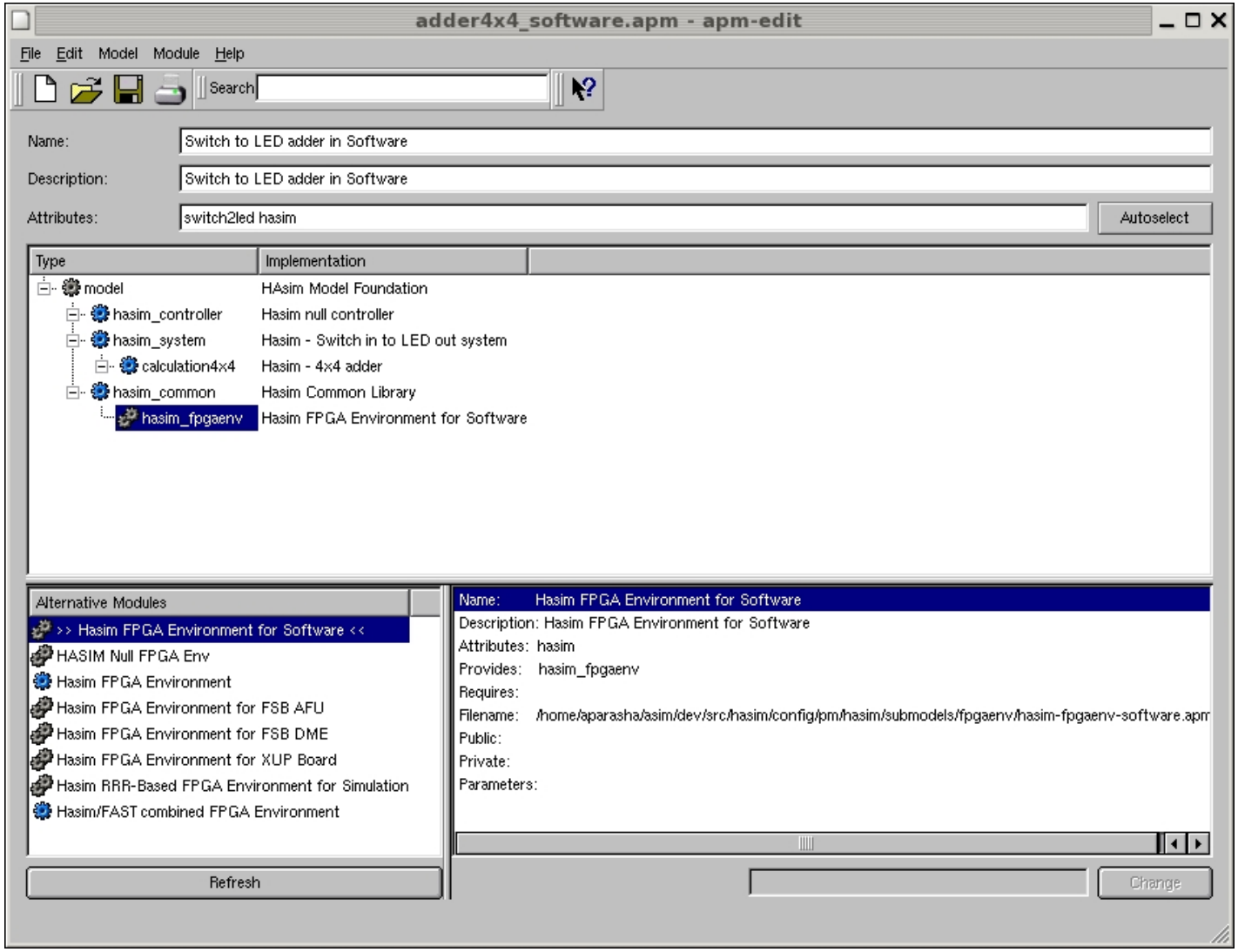
Hybrid Modules (cont.)

- Module definition
  - `hybrid_fetch_unit.awb`
    - `%sources –t BSV –v PUBLIC hardware_fetch_unit.bsv`
    - `%sources –t CPP –v PUBLIC software_fetch_unit.h`
    - `%sources –t CPP –v PRIVATE software_fetch_unit.cpp`
- Build process
  - Collects BSVs and generates “hardware” bitfile or simulation binary
  - Collects CPPs and generates “software” binary
- Runtime
  - Software binary loads bitfile onto FPGA, or forks off simulation “hardware” binary
  - Software sends “start” HRRR request to Hardware

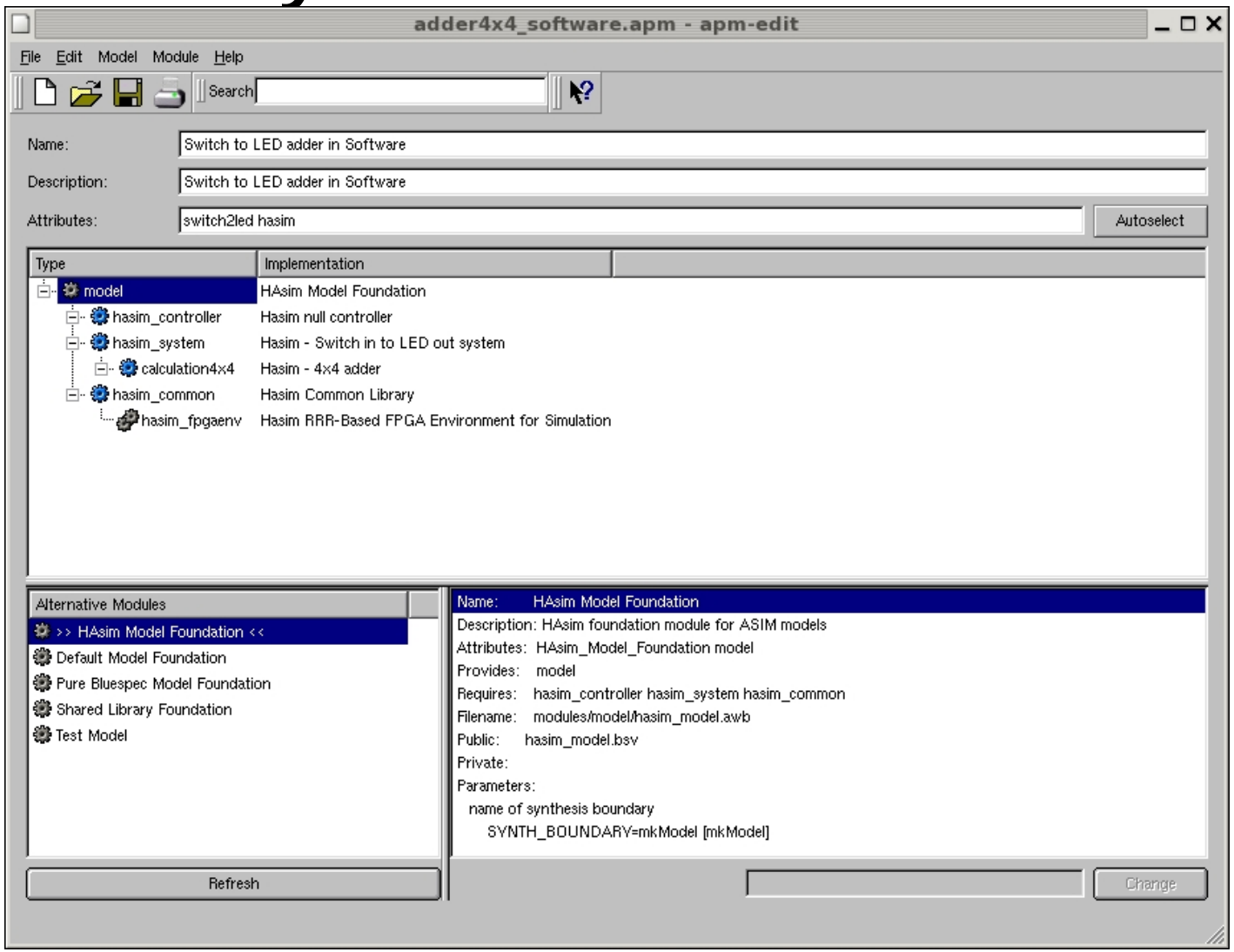
Demo

Backup Slides

“Sim” Front Panel



Hybrid Front Panel



Running on Bluesim…