Improving Non-Functional Properties of Operating Systems with Reconfigurable Hardware (EuroDW’11)

Michael Gernoth

System Software Group

Friedrich-Alexander University Erlangen-Nuremberg

April 10, 2011
Achieving system properties purely in software can be hard:

- **Security**: Running software must not alter security policies
- **Safety**: Entering a safe state, even when the system crashes
- **Determinism**: Enforcing hard limits to reach real-time goals
- **Performance**: Increasing system throughput

Reconfigurable hardware is becoming a commodity
- But only used for user space applications
Motivation

Achieving system properties purely in software can be hard:

- **Security**: Running software must not alter security policies

- **Safety**: Entering a safe state, even when the system crashes

- **Determinism**: Enforcing hard limits to reach real-time goals

- **Performance**: Increasing system throughput

Reconfigurable hardware is becoming a commodity

- But only used for user space applications
Problem

- Custom hardware needs to know about the OS
  - No stable kernel API
  - Transforming system state to a hardware-defined format is expensive
  - Manually adapting hardware to changing software is fragile

Solution

Generate hardware interfaces automatically

(see linux-2.6/Documentation/stable_api_nonsense.txt)
Compile-Time

- Idea: Use debug symbols
- Memory layout as created by the compiler
- Used for HDL interface generation
- Stable interface for hardware implementation
- Incompatible OS changes are detected
- Integrateable in kernel build process

entity struct_simple is port (
  signal flags_i: in std_logic;
  signal info_i: in std_logic;
  signal data_i: in std_logic;
  signal base_addr_i: in std_logic_vector(31 downto 0);
  signal addr_o: out std_logic_vector(31 downto 0);
  signal clk_i: in std_logic);
end entity struct_simple;

architecture struct_simple__arch of struct_simple is
begin
  process (clk_i)
  begin
    if (clk_i’event and clk_i = ‘1’) then
      if (flags_i = ‘1’) then
        addr_o <= base_addr_i + 0 + 0;
      elsif (info_i = ‘1’) then
        addr_o <= base_addr_i + 0 + 2;
      elsif (data_i = ‘1’) then
        addr_o <= base_addr_i + 0 + 4;
      else
    end process (clk_i);
end architecture struct_simple__arch;
Run-Time

- Access to system data via DMA
- Generated hardware can be activated on-the-fly
- Passive and active functionality possible
Status and Future Work

What is working so far?

- Interface generation and hardware build process
- Integration into Linux
  - Detecting and terminating prohibited processes on the system

What am I working on?

- Implementation of more hardware functions
  - Security: Hardware-based policy enforcement
  - Determinism: Real-time scheduling
  - Performance: Network routing
- Semi-automatic transformation of kernel code