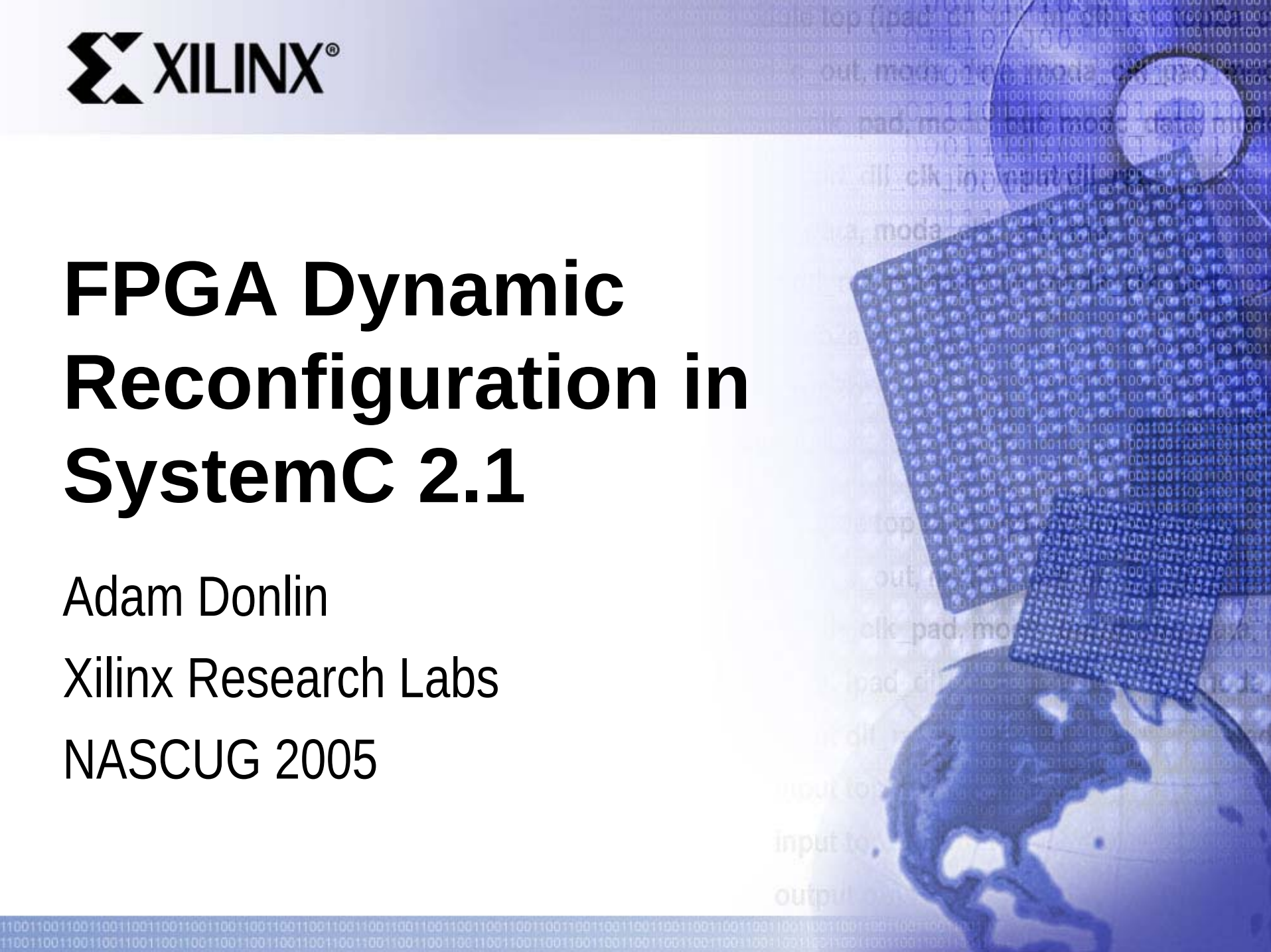


FPGA Dynamic Reconfiguration in SystemC 2.1

Adam Donlin

Xilinx Research Labs

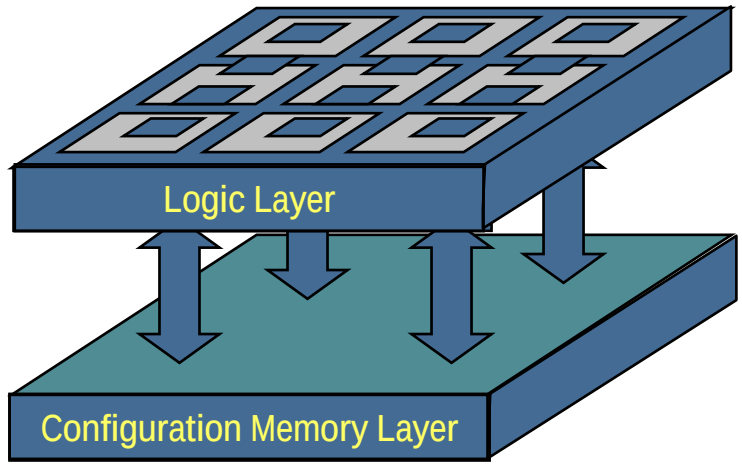
NASCUG 2005

Outline

- FPGA Reconfiguration Overview
- Modeling in SystemC 2.1
  - Interface and Factory Method
  - Dynamic SC_THREAD based
- Performance and Conclusions

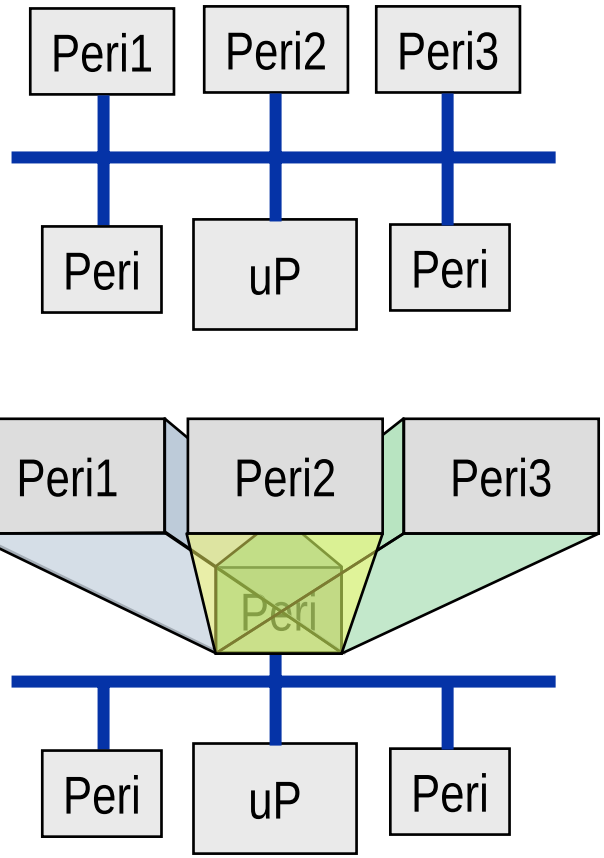
FPGA Dynamic Reconfiguration

- Think of an FPGA as Two Layers:
  - Configuration Memory
  - Logic Layer
- Configuration memory controls function computed on logic layer
- Non-reconfigurable operation:
  - Load configuration at power on…
  - Maintain configuration until power down…



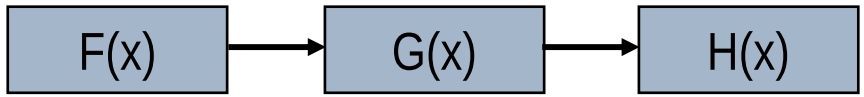
FPGA Dynamic Reconfiguration

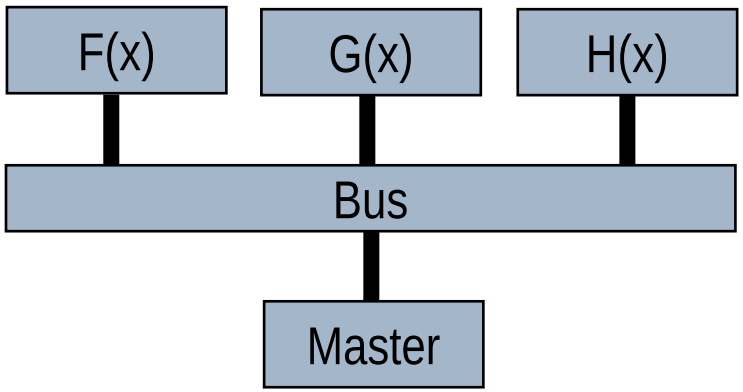
- Dynamic Reconfiguration:
  - Change the function computed by the logic layer
  - Re-write the contents of the configuration memory *at runtime*
- Why?
  - Time-multiplex logic resources
  - Load specialized circuits only when needed
  - Do more with less!



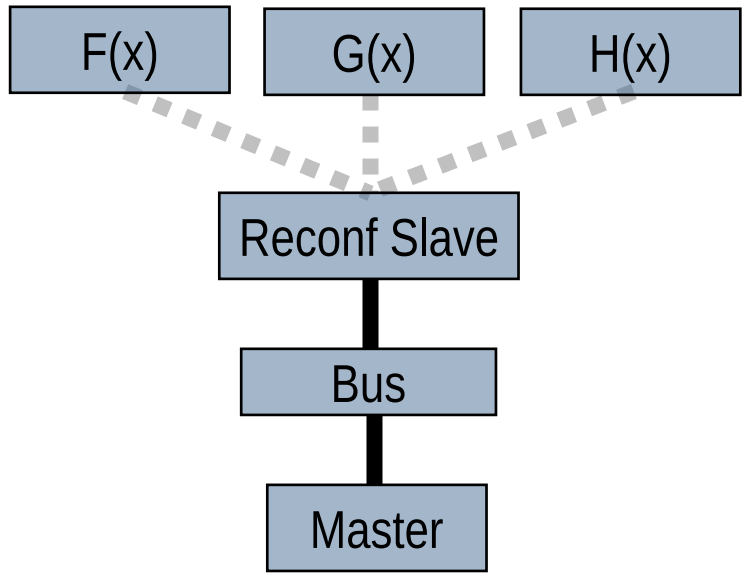
Example

Abstract Application:
A Simple Pipeline



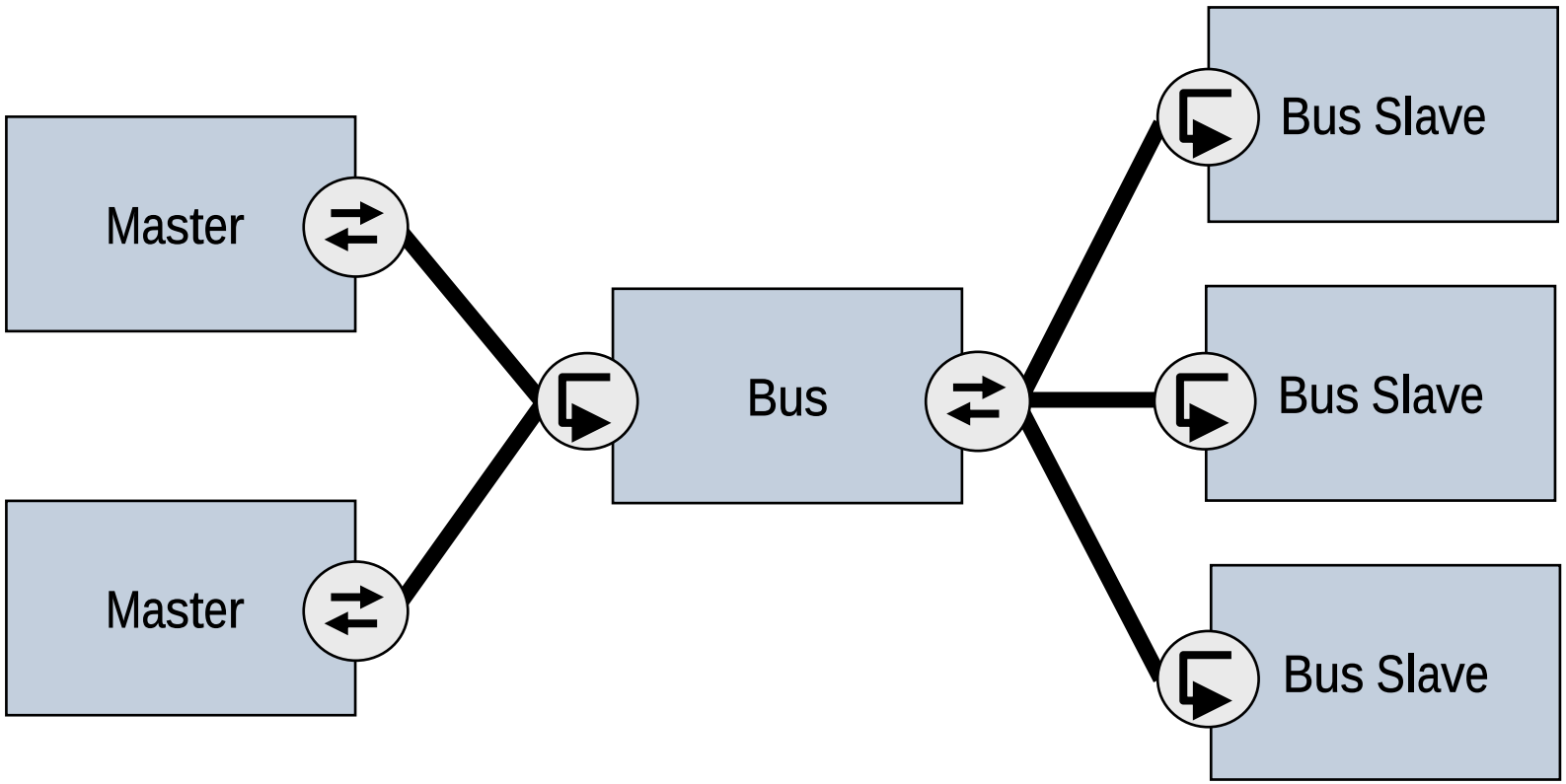


Implemented in a Static Bus System



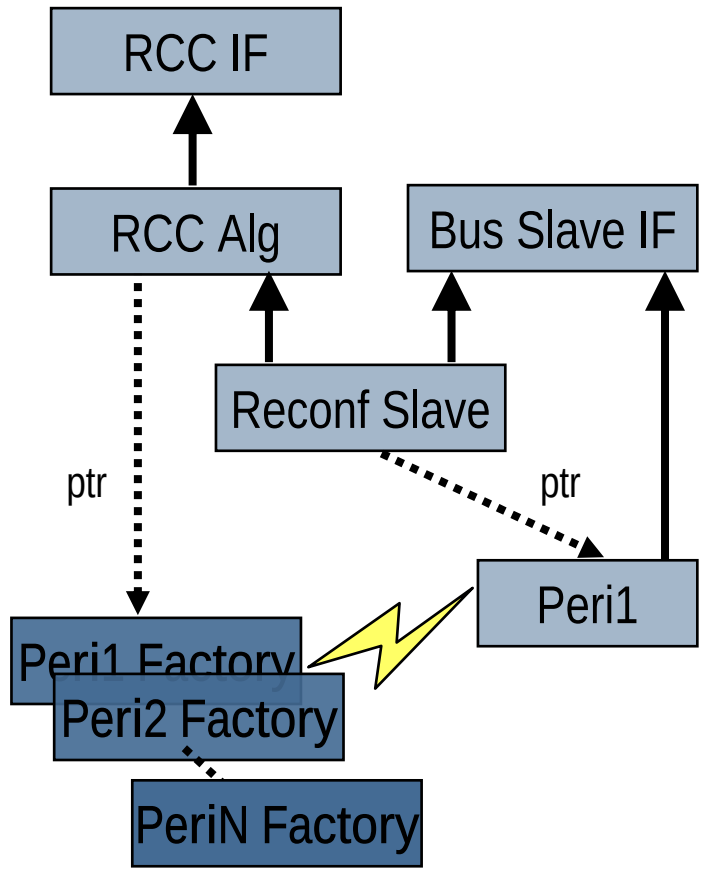
Implemented with Dynamically Reconfigurable Slave Module

Simple TLM Bus



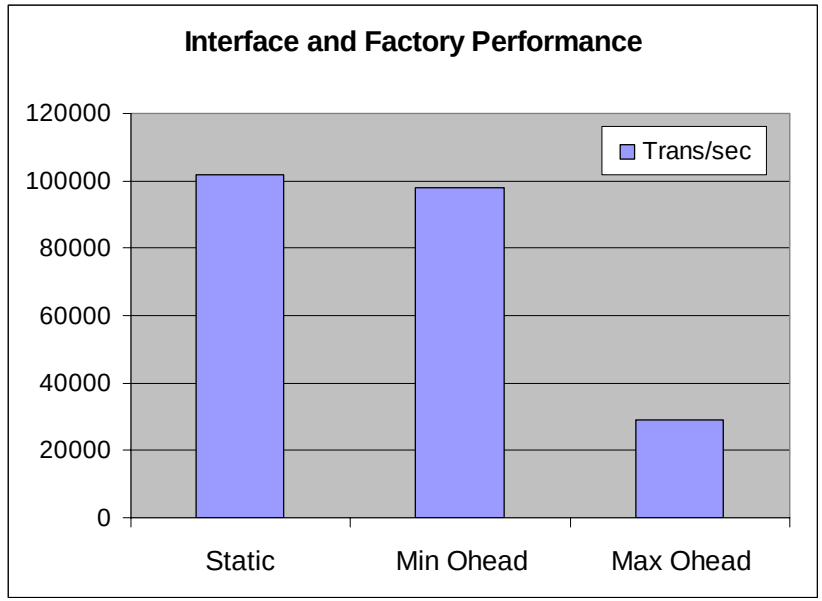
SystemC Models of Reconfiguration

- Interface and Factory Technique
  - `reconf_slave` encapsulates behavior of any object implementing `bus_slave_if`
    - Transactions forwarded to the encapsulated slave object…
  - Object factory generates `bus_slave` at runtime
    - with appropriate parameterization
  - `RCC_Alg` class implements
    - reconfiguration control interface
    - configuration schedule



Issues and Results

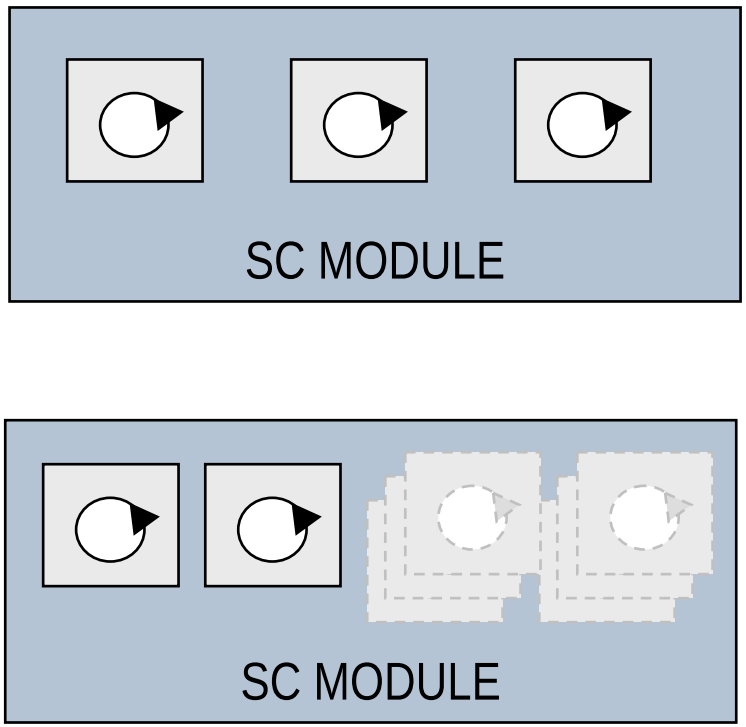
- Factory cannot create `sc_module` after simulation has started.
  - Multiply inherit `peripheral_base` and `sc_module` into static peripheral.
- Peripheral base must share threads with the reconfigurable slave…
  - Thread Pool will approximate a solution…
- Performance Measurements for:
  - Static System (no reconfiguration)
  - Reconfiguration Overhead Minimized
  - Maximized Configuration Overhead (one reconfiguration follows every bus transaction)



Experimental system:
SC v2.1, 3GHz Xeon,
No significant memory load.

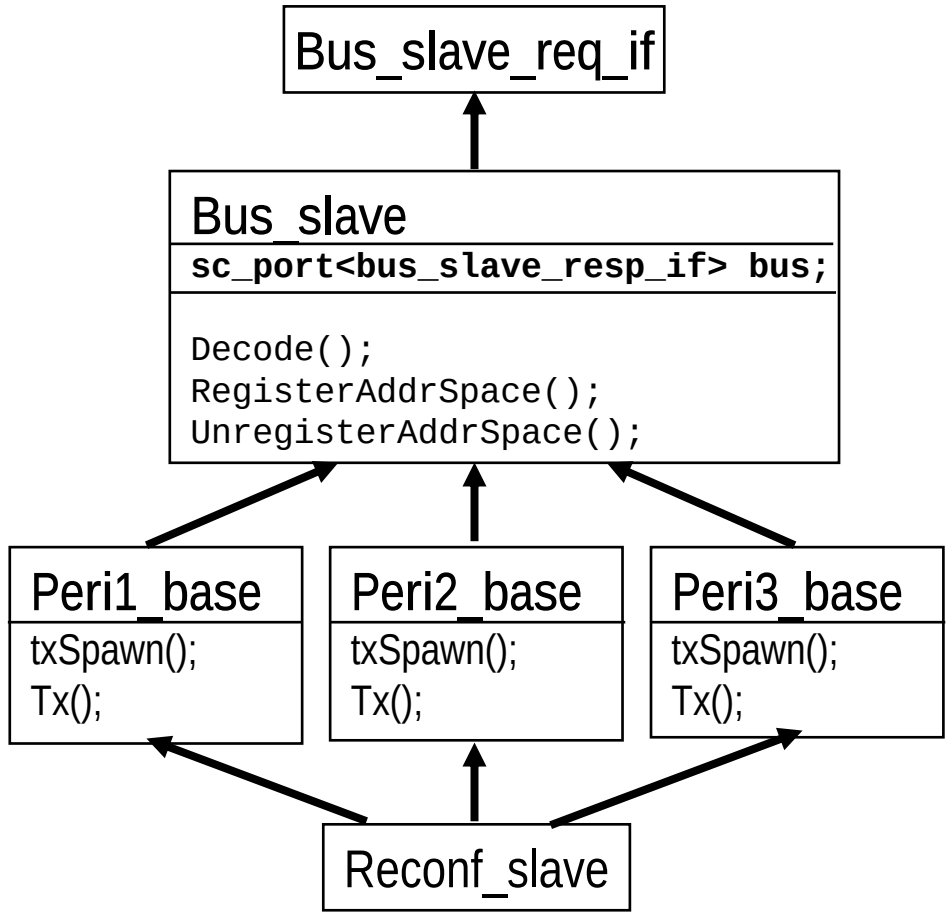
Dynamic Thread Method

- SystemC V2.0.1:
  - All threads must be declared before simulation begins
- SystemC V2.1
  - Class methods and other functions ‘spawned’ after simulation starts.
  - `sc_thread` or `sc_method` semantics
- Cannot create `sc_module` at runtime…
  - All dynamic threads share context within the same `sc_module`



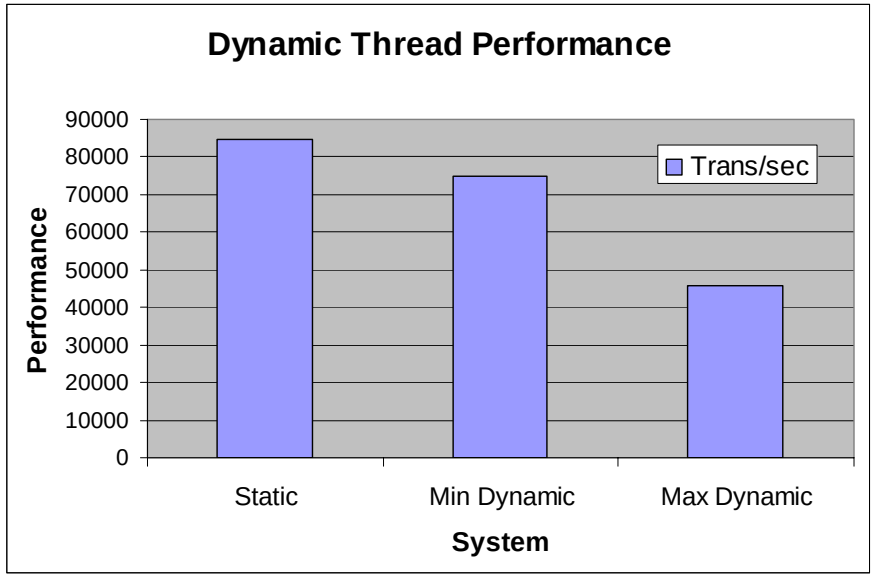
Reconfigurable Slaves with Dynamic Threads

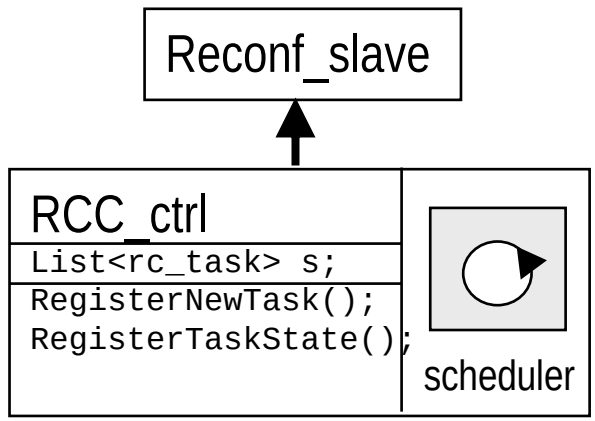
- No object factory:
  - Multiply inherit all possible behaviors for a given slave
  - More complex bus TLM
    - Decode interface
    - Dynamic threads register new address space with bus_slave base class
- Peripheral base class includes a `spawn` method
  - `reconf_slave` invokes `base::spawn` for each reconfiguration…
  - One base class spawned at any given time.
  - Base class
    - registers address space with the `bus_slave`
    - handles transactions that arrive through the shared bus port.



Issues and Example

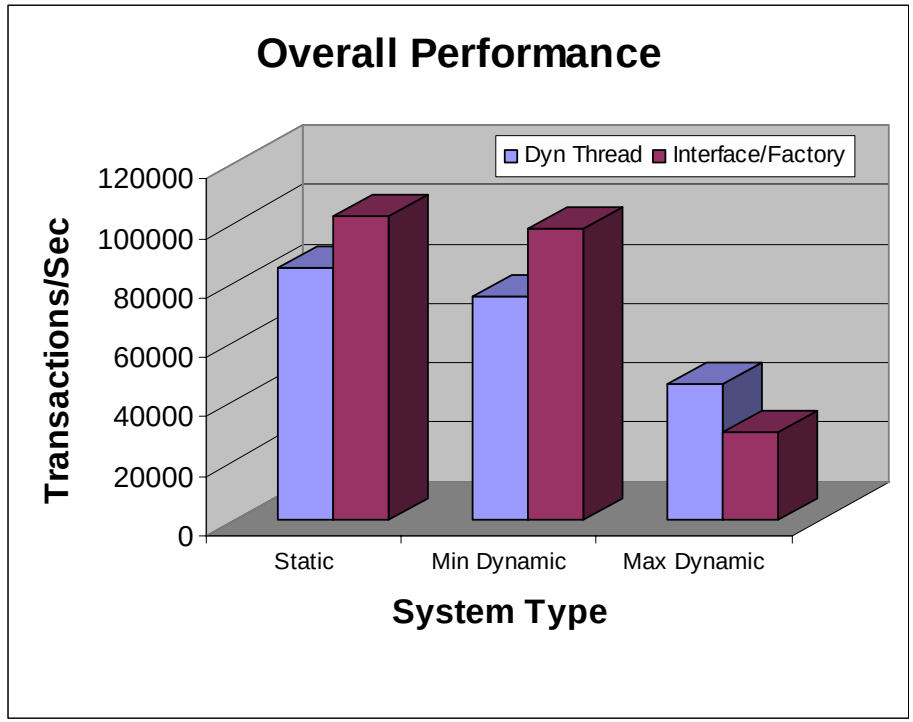
- Cannot kill a spawned thread from its sc_process_handle
  - Explicit 'suicide' events required.
  - Base class must be sensitive and perform explicit tests on these events.|
- Virtual base classes and multiple inheritance:
  - Effective but complicated.
- Single `sc_module` requires explicit state management for every spawned thread.





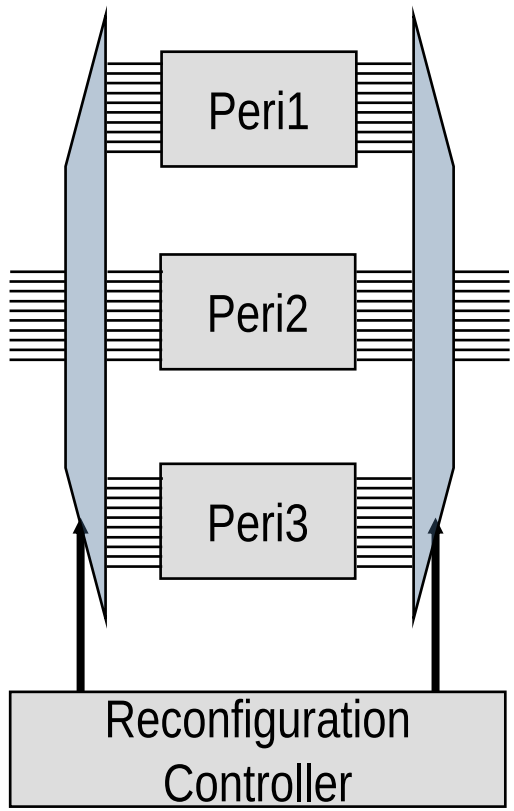
Conclusions

- Interface and Factory
  - Simple bus structure,
  - Thread pool can approximate multiple, dynamic threads
  - Better general performance but higher worst case performance
- Dynamic Threads
  - More complex bus structure
  - More complex C++ programming requirement
  - Type coupling of reconfigurable base classes to `reconf_slave` and reconfiguration controller.
- Both techniques are tractable
  - Dynamic `sc_module` would be even more helpful!



Simulating Dynamic Reconfiguration: RTL Models

- RTL Simulation
  - 'Virtual Multiplexor' approach
  - Instantiate all possible peripheral behaviors in the RTL simulator
  - De-Multiplex inputs to the currently configured module
  - Multiplex outputs to one peripheral output
- Reconfiguration Controller Model manipulates virtual multiplexors



Simple TLM Bus

- Master Interface

```
class bus_master_if : public virtual sc_interface {
public:
     virtual void read(ulong addr, ulong& data) = 0;
     virtual void write(ulong addr, ulong data) = 0;
     virtual bool arb(int& result) =0;
};
```

- Slave Interface

```
class bus_slave_if : public virtual sc_interface {
public:
     virtual void read  (ulong addr, ulong& data) = 0;
     virtual void write (ulong addr, ulong data) = 0;
     virtual bool decode(ulong addr) = 0;
};
```