Standardization Around Registers

What's needed?

Presentation Copyright Permission

– A non-exclusive, irrevocable, royalty-free copyright permission is granted by **GreenSocs, STMicroelectronics, Intel and Ericsson** to use this material in developing all future revisions and editions of the resulting draft and approved Accellera Systems Initiative **SystemC** standard, and in derivative works based on this standard.

On the panel

Mark Burton
GreenSocs

Jerome Cornet
ST

Ola Dahl
Ericsson

Philipp Hartmann
Intel

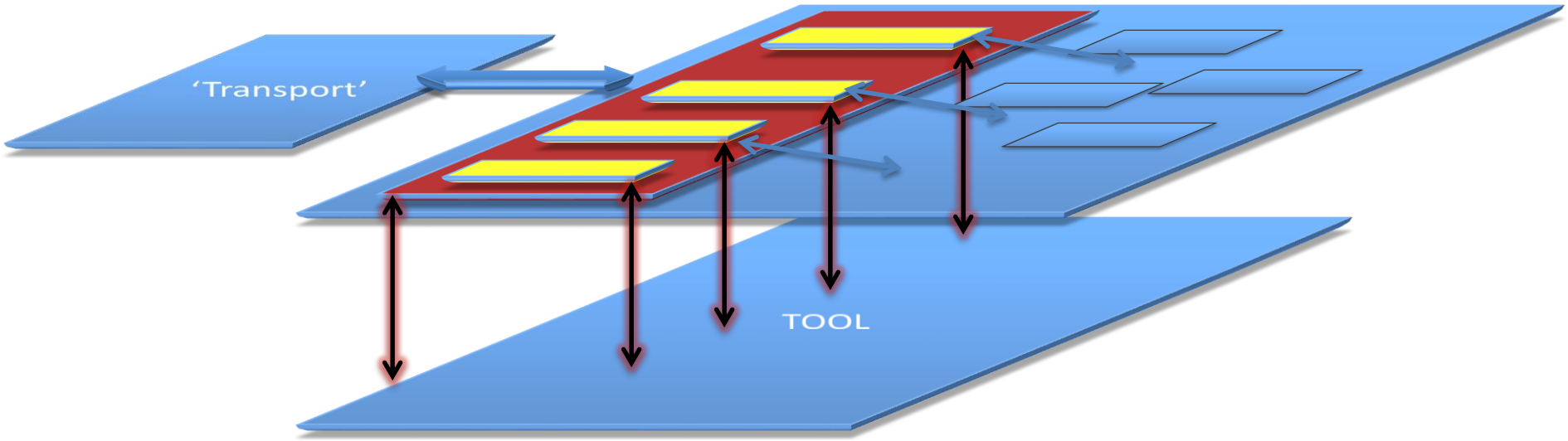
What do we mean: Registers?

- Accessible state ✓
  - Has an address/offset etc
- 'latches' in the design. ✗
  - Though – maybe …

Past work:

- Cadence/ST SCI Reg proposal
  - Possible starting point
- CCI / SystemC evolutions
  - Things to reuse or easier to do now…
- GreenSocs 'GreenReg'
  - Possible user code.

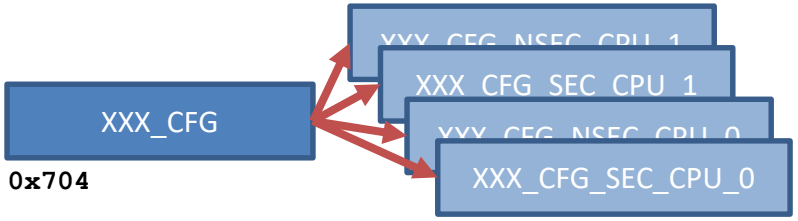
The problems



- Accessing Registers from a tool
  - (introspection and debug)
- Knowing the register map
  - Useful for the tool
  - Useful for debug
- ~~*Convenient ways to define registers*~~

Past experience

- Hardware engineers are very creative!
  - Split/single IO.
  - Register located at given offset
    - Bus accesses goes to different “locations” based on security bus attributes and CPU ID



  - Handling mixtures of endian-ness (even in the same model!)
  - Non processor address spaces – e.g. SPI, etc…
  - Our models and register implementations are full of special cases!
    - And they are at the VERY CORE of all (virtual prototyping) flows!

The ‘register’ object is complicated, and forever changing!!!
(impossible to standardize a ‘register model’ to meet everybody's needs)

Use cases

- Enable tool/validation use cases
  → without changes to the model itself

- Debug support
- Monitoring / Tracing / Logging
- Fault injection
- Coverage collection
- …

Use cases

- **Discovery**
  - Enumerate all available registers
  - Get the address map of all registers
    - » Not always directly connected to a processor! – e.g. SPI

- **Querying information** about a register/resource
  - "Basic" information
    - » Names, documentation
    - » Width
    - » Accessibility (read/write)
    - » Bit fields (names, width, offset, readable/writable, …)
    - » Reset values
  - Address, offset
    - » May be context dependent (i.e. who's asking?)
  - Has side effects? (whatever that means)

Backdoor/tool access to a register

- Read/write without going through a TLM-2.0 transaction
  - Access at bit field granularity needed
  - Without triggering side effects (i.e. debug access)
  - With triggering side effects (as if it accessed from SW)
- Forcing a register to a specific value
  - Ignoring other accesses until released again
  - Override access permissions
- Notification (callbacks) for accesses
  - “Software access” (e.g. coming from a bus)
  - “Hardware access” (modification from the model directly)
  - “Debug access” (backdoor access, e.g. for synching different tools)
- Could be extended to other resources
  - e.g. memories

Underlying difficult questions

- What do you mean with “basic properties”?
  - Where do we stop?
  - Address Maps? Types of registers? Definitions (IPXACT)?
  - Security features… etc
- Do we cover ‘memories’ too?
  - When is a register a memory?
  - Often memory areas embedded in ‘register map’,
    - helpful to load/store to/from file…
- What do you mean by “address map”?
  - The bit fields in a register?
  - The local offset of a register in a bank?
  - The (multiple) offsets of an register bank
  - The (multiple) offsets of IP base addresses
  - From who’s perspective… in what context…

Goals

- "Long lasting" API
  - At least try to design something still relevant in the future
- Vendor Neutral
  - Anyone can discover/access register information in the simulation (tools, model, internal utility blocks…)
- Keep it simple silly
  - Straightforward implementation and documentation

Path forward

- **Keep It Simple Silly!**
  - Narrow our scope
- ‘Introspection’
  - What needs to be ‘discoverable’.
  - What needs to be ‘introspect-able’.
  - Ex: What are the “basic properties”.
- ‘Interoperability’
  - Focus on the IP/Tool API
- Where do ‘address maps’ fit?