Spring/Summer 2018 Quarterly Meeting

University of Glasgow

27 June 2018

Section 1

Overview

Past Activities

- **Q1** — Reading + Talking
- **Q2** — Reading + Talking + Visiting + Squiggling
- **Q3** — Reading + Talking + Visiting + Squiggling + Bodging

Q4 Goals

- **Heriot Watt**: Verifying Protocol Implementations
- **Imperial**: Extending theory of Session Types for Hardware.
- **Glasgow**: Developing SoC Architecture Description Language.

Summary

- **Q4** — Reading + Talking + Visiting + Squiggling + Bodging

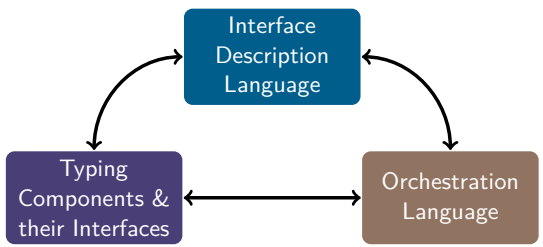
Section 2

Glasgow

Activities

- **Squiggling & Bodging**
  - Developing SoC Architecture Description Language.
  - Examples using AXI, APB in progress
- **Talks**—Self Invited
  - **A Type-System for describing the Structural Topology of System-on-a-Chip Architectures.** *MSP 101 University of Strathclyde*
  - **A Type-System for describing System-on-a-Chip Architectures.** *FP-Group University of St Andrews*
  - **A Type-System for describing System-on-a-Chip Architectures.** *PL-Interest University of Edinburgh* date tbc
- **Papers**
  - **Fake it until they make it: Implementing Substructural Type-Systems for EDSLs using Dependent Types** *Jan de Muijnck-Hughes, Edwin Brady, Wim Vanderbauwhede* submitted to TyDe 2018.

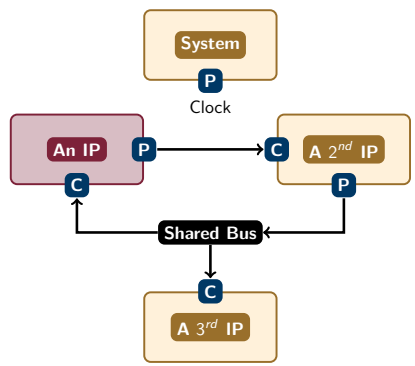
Cordial: A Language to Describe SoC Architectures



Note In progress

- Stratified System of Systems.
- Take inspiration from existing work.
- Feed into behavioural Design.
- 'Correct-by-Construction'.
- 'Cool Types'

Illustrative Example



UoG Protocol

Usage

- RW to 'Memory'

Communication Style

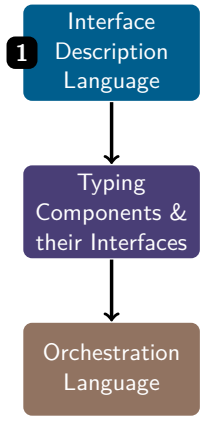
- Unicast & Broadcast

Signals

- Clock
- Control Write
- Control Read
- Address (8/16)
- Data (64/32)
- Optional Consumer Error

Interface Description Language

- Language to specify agnostic structure of interface.
- Ensure role uniqueness.
- Generic & parameterised interface descriptions.



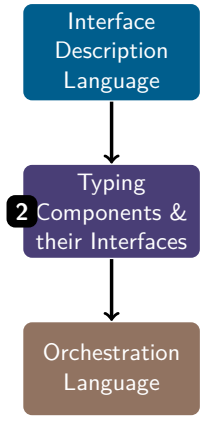
An Interface Description Language

```
data RTy = CLOCK | WRITE | READ | ADDRESS | DATA | ERROR

UoG : (dWidth, aWidth : Nat) -> IDL RTy
UoG dWidth aWidth = do
  c <- role CLOCK
  w <- role WRITE
  r <- role READ
  e <- role ERROR
  a <- role ADDRESS
  d <- role DATA

  signal c Clock       ConstC 1      IsRequired       SYS
  signal w Control     PC     1      IsRequired       IP
  signal r Control     PC     1      IsRequired       IP
  signal e DataWire    CP     2      ConsumerOptional IP
  signal a AddressWire PC     aWidth IsRequired       IP
  signal d DataWire    PCCP   dWidth IsRequired       IP
  end

UoG : (cStyle : CommStyle) -> (a,b:Nat) -> InterfaceTy cStyle RTy
UoG c a b = build c 1 2 (UoG_Gen a b)
```

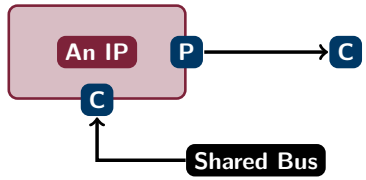
Type Checking Components & Interfaces

Typing Components & their Interfaces

- State trustworthiness of components.
- Type-check 'local' interfaces against 'global' description.
- Source for HDL generation.



A IP Core and it's Interfaces



A Consumer

```
MyIP : Component UnTrusted
                 [Producer, Consumer]
                 [UoG Unicast 64 8, UoG Broadcast 32 16]
MyIP = [uog_64_8_producer, uog_32_16_consumer]
```

Type Checking Interfaces

A Producing Interface

```
uog_64_8_producer : Interface cStyle RTy PRODUCER Producer (UoG cStyle 64 8)
uog_64_8_producer =
  MkInterface $  (IPort (N DATA) DataWire INOUT 64 ...)
           <::> (IPort (N ADDRESS) AddressWire OUT 8 ...)
           <::> (IPort (N ERROR) DataWire IN 2 ...)
           <::>  (IPort (N READ) Control OUT 1 ...)
           <::>  (IPort (N WRITE) Control OUT 1 ...)
           <::>  (IPort (N CLOCK) Clock IN 1 ...)
           <::>  Empty
```

A Consuming Interface

```
uog_32_16_consumer : Interface cStyle  RTy CONSUMER Consumer (UoG cStyle 32 16)
uog_32_16_consumer =
  MkInterface $ (IPort (N DATA) DataWire INOUT 32 ...)
           <::> (IPort (N ADDRESS) AddressWire IN 16 ...)
           <::~> (IPort (N READ) Control IN 1 ...)
           <::> (IPort (N WRITE) Control IN 1 ...)
           <::> (IPort (N CLOCK) Clock IN 1 ...)
           <::> Empty
```

Projecting Interfaces

Global

```
signal c Clock       ConstC 1      IsRequired       SYS
signal w Control     PC     1      IsRequired       IP
signal r Control     PC     1      IsRequired       IP
signal e DataWire    CP     2      ConsumerOptional IP
signal a AddressWire PC     aWidth IsRequired       IP
signal d DataWire    PCCP   dWidth IsRequired       IP
```

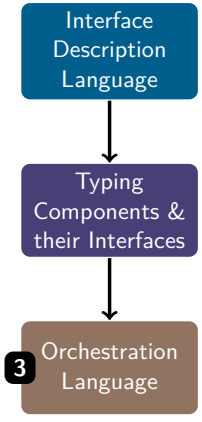
Local

```
MkInterface $ (IPort (N DATA) DataWire INOUT 32 ...)
         <::> (IPort (N ADDRESS) AddressWire IN 16 ...)
         <::~> (IPort (N READ) Control IN 1 ...)
         <::> (IPort (N WRITE) Control IN 1 ...)
         <::> (IPort (N CLOCK) Clock IN 1 ...)
         <::> Empty
```

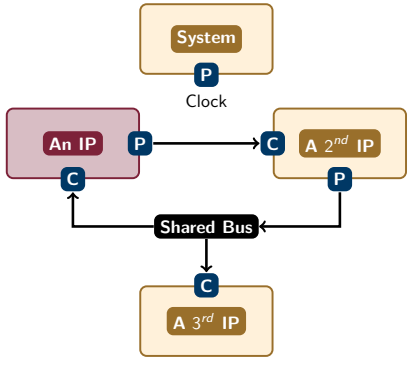
An Orchestration Language

Orchestration Language

- Connect the boxes
- Ensure signal/wire compatibility.
- Ensure interfaces are only connected once.
- Ensure shared channels respect max masters & slaves.
- Note: In progress...



Illustrative Example



UoG Protocol Usage

- RW to 'Memory'

Communication Style

- Unicast & Broadcast

Signals

- Clock
- Control Write
- Control Read
- Address (8/16)
- Data (64/32)
- Optional Consumer Error

Declaring Components & Interfaces

```
myFirstSoC : SoC
myFirstSoC = do
  myFirstIP  <- newComponent UnTrusted
  mySecondIP <- newComponent Trusted
  myThirdIP  <- newComponent Trusted

  mySharedBus <- newBus (UoG Broadcast 32 16)

  a1p <- newInterface (UoG Unicast 64 8) Producer $ (uog_64_8_producer)
  a1c <- newInterface (UoG Broadcast 32 16) Consumer $ (uog_32_16_consumer)

  b2c <- newInterface (UoG Unicast 64 8) Consumer $ (uog_64_8_consumer)
  b2p <- newInterface (UoG Broadcast 32 16) Consumer $ (uog_32_16_consumer)

  c2c <- newInterface (UoG Broadcast 32 16) Consumer $ (uog_32_16_consumer)

  addInterface myFirstIP  a1p
  addInterface myFirstIP  a1c
  addInterface mySecondIP b2p
  addInterface mySecondIP b2c
  addInterface myThirdIP  c2c
```

Connecting Boxes

```
connect myFirstIP a1p mySecondIP b2c

addConsumer myFirstIP a1c mySharedBus
addConsumer myThirdIP c2c mySharedBus
addProducer mySecondIP b2p mySharedBus

end
```

Limitations

- Ignoring System connections for now.
- Whole interface connections only.

Ensuring Valid Direct Connections

A Producing Interface

```
uog_64_8_producer : Interface cStyle RTy PRODUCER Producer (UoG cStyle 64 8)
uog_64_8_producer =
  MkInterface $  (IPort (N DATA) DataWire INOUT 64 ...)
           <::> (IPort (N ADDRESS) AddressWire OUT 8 ...)
           <::> (IPort (N ERROR) DataWire IN 2 ...)
           <::>  (IPort (N READ) Control OUT 1 ...)
           <::>  (IPort (N WRITE) Control OUT 1 ...)
           <::>  (IPort (N CLOCK) Clock IN 1 ...)
           <::>  Empty
```

A Consuming Interface

```
uog_32_16_consumer : Interface cStyle  RTy CONSUMER Consumer (UoG cStyle 32 16)
uog_32_16_consumer =
  MkInterface $ (IPort (N DATA) DataWire INOUT 32 ...)
           <::> (IPort (N ADDRESS) AddressWire IN 16 ...)
           <::~> (IPort (N READ) Control IN 1 ...)
           <::> (IPort (N WRITE) Control IN 1 ...)
           <::> (IPort (N CLOCK) Clock IN 1 ...)
           <::> Empty
```

Current & Future Challenges

1. **Interplay of Descriptions & Interfaces**
   - What goes in the 'type' and how does it **influence** the 'value'.
2. **Reasoning about Component Connections?**
   - Wire/Interface Compatibility
   - Connecting wires to interfaces
   - Payloads
3. **Reasoning about Components**
   - Clock speeds
   - Parameters
4. **Generating Output**
   - IP-XACT
   - HTML Documentation
   - HDL Stubbs
   - Block Diagrams
5. **User Experience**
   - Prof. Resilient EDSL Usage
   - Prof. Resilient DSL Usage?

Section 3

Imperial

Recap: Transaction handshaking with session types

Representing transaction handshaking in session types

- Signal assert/unassert events as messages
- Messages are either sent (asserted by) Master or Slave
- The sequence of messages form a protocol
- Master/Slave protocols valid if the protocols
  - are instances of basic handshake process
  - conform to transaction dependencies

Given a valid handshaking protocol

- Generate (partial) **endpoint** IP implementation
- Generate **monitor** for signals in Verilog

AXI protocol checker

Categories of checks

1. Bound checks, e.g. memory address, size of data
2. Initialisations, e.g. signals after resets
3. Event dependencies, e.g.
   X must remain stable when Y is asserted and Z { high, low }
4. Other specific checks

Category 3 is especially suitable for session type-based approach, but

- No notion of **time**: next FSM transition when signal *changes*
- Uniformly combining the dependencies?
  - Globally: explicitly specify all signal changes
  - Hierarchical: separate binary sessions, duration dependent on parent

Timed Multiparty Session Types

Multiparty Session Types + Communicating **timed** Automata

- Modular verification in distributed system
- Constraints on global types, concrete values from system
- Progress of timed processes if
  - **Feasibility**: global time constraints are satisfiable
  - **Wait-freedom**: message must be ready when receiving

Validating the specification

- Can (global) timing constraints be satisfied?
- Global types: No overlapping of time constraints within (multiparty) session
- Clocks and clock resets: Statically known concrete time constraints

Timed Multiparty Session Types - Bocchi, Yang, Yoshida, CONCUR'14

Temporal modalities

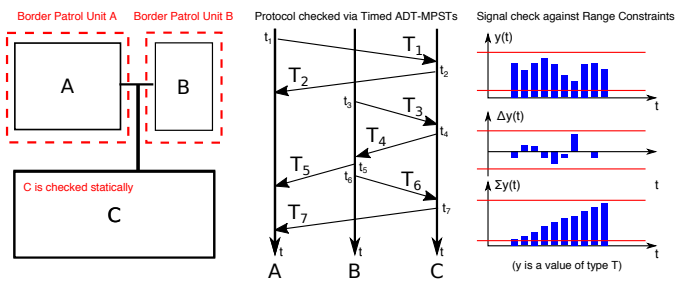
Enrich binary session types with temporal modalities

- Adds 3 temporal modalities *next* ($\circ A$), *always* ($\Box A$), and *eventually* ($\diamond A$)
- Uses **when?** and **now!** messages to express eventually
- Time reconstruction - *temporal refinement*

Validating the specification

- Previously (tMPST) difficult to express nondeterminate abstract time passage
- Overlap disjoint binary session as delay
- No global type, but can we stack/overlap the sessions for a global view?
- We want to know if global view is satisfiable:
  - suitable for model checking in this form (hint: barbs)?
  - raising model checker specification to types
  - if so proceed with endpoint/monitor generation

Parallel Complexity Analysis with Temporal Session Types - Das, Hoffmann, Pfenning, ICFP'18 also *ForSpec Temporal Logic, TACAS'02*



Section 4

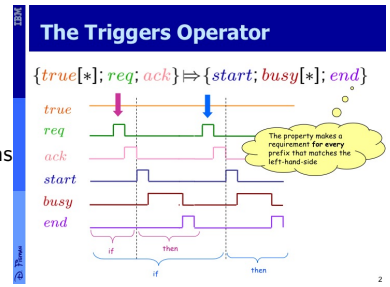
Heriot Watt

Assertion based Hardware Bus Protocol Verification

- **Checker Design for On-line Testing of Xilinx FPGA Communication Protocols**
  Straka et al, IEEE Defect and Fault Tolerance in VLSI Systems, 2007.
  - formal FSM language; monitor IP generated, LocalLink use case
- **Integration of Hardware Assertions in System-on-Chip**
  Geuzebroek et al, IEEE Int. Test Conf., 2008.
  - 1% additional area cost for verifying hardware
- **A Synthesisable AXI Protocol Checker for SoC Integration**
  Chen et al, Int. SoC Design Conf,, 2010.
  - 44 rules for AXI protocol, 242MHz, 71k gate counts
- **A Distributed AXI-based Platform for Post-Silicon Validation**
  Neishaburi et al, IEEE VLSI, 2011.
- **Verification of Memory Transactions in AXI Protocol using System Verilog Approach**
  Mahesh et al, IEEE ICCSP, 2015.
  - use case: memory transactions, pseudo random coverage

Temporal assertions

- IEEE 1850 PSL Standard
- IEEE 1800 SystemVerilog Standard
- Trigger operators
  - `s |=> t!`
  - `s |-> t!`
  - `s |=> t`
  - `s |-> t`
- Concatenation, fusion, union, intersection etc..
  - `s ; t`
  - `s :  t`
  - `s | t`
  - `s && t`
  - `s & t`
  - `s within t`
- Consecutive repetitions
  - `s[*i]`
  - `s[*i..j]`
  - `s[*i..]`
  - `s[*]`
  - `s[+]`



MBAC

- Hardware verification tool developed at McGill University
- **Input**: PSL/SVA assertions, hardware module under test
- **Output**: Hardware protocol checker for each assertion
- Generates **assertion-checking hardware** from temporal assertions
- Generated hardware is **efficient** and **synthesisable**
- Both for simulation and runtime checking
- Academic license agreement between McGill University and Heriot-Watt
- Imperial and Glasgow sign MBAC license agreement too?

MBAC: assertions

```
vunit vu1(test){
  default clock = (posedge clk);

  assert always a-> {b;d} until c;
  assert next_event_a(a)[1:2]({{b;c}|{d;e}});
  assert next_event_a(a)[2:5]({b;c;d});
  assert never {a;b;c};
  assert {[+]:{a;b;c}} |-> false;
  assert never {a;b;c};
}
```

Use:

```
mbac source_design.v assertions.psl
```

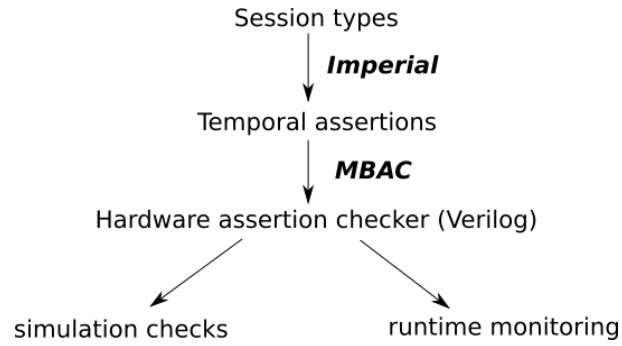
MBAC: generated checker

```
//----------------
//ASR_1 : assert always a -> {b ; d} until c;
//----------------
always @(posedge clk) if (`AKRPS reset) s1sq<=4'h2;
                      else s1sq<=s1s;
assign s1s={(a && b && !(c))
            || (s1sq[2] && (b && !(c))),
  (a && !(c)) || (s1sq[2] && !(c)),
  1'b1,
  (a && !(b) && !(c))
  || (s1sq[2] && (!(b) && !(c)))
  || (s1sq[3] && !(d))};
always @(posedge clk) if (`AKRPS reset) ASR_1<=0;
                      else ASR_1 <= (s1s[0]);
```

MBAC: generated checker

```
//----------------
//ASR_5 : assert {[+] : {a ; b ; c}} |-> 1'b0;
//----------------
always @(posedge clk) if (`AKRPS reset) s5sq<=4'h4;
                      else s5sq<=s5s;
assign s5s={a,
  1'b1,
  (s5sq[3] && b),
  (s5sq[1] && c)};
always @(posedge clk) if (`AKRPS reset) ASR_5<=0;
                      else ASR_5 <= (s5s[0]);
```

Session Types to Assertion Checking Hardware?



Border Patrol Publications

- Accepted
  - **Recursive Array Comprehensions in a Call by Value Language**, Artem Šinkarovs, *Sven-Bodo Scholz*, *Robert Stewart*, Hans Vießmann, IFL, 2017.
  - **Replicable Parallel Branch and Bound Search**, Blair Archibald, Patrick Maier, Ciaren McCreesh, *Robert Stewart*, Phil Trinder, JPDC, 2017.
  - **Parallel Mean Shift Accuracy and Performance Trade-Offs**, Kirsty Duncan, *Robert Stewart*, Greg Michaelson, IEEE ICIP, 2018.
  - **RIPL: A Parallel Image Processing Language for FPGAs**, *Robert Stewart*, Kirsty Duncan, Greg Michaelson, Paulo Garcia, Deepayan Bhowmik, Andy Wallace, ACM TRETS, 2018.
- Submitted
  - **Parallel Dataflow Transformations with Model Checking for FPGAs** *Robert Stewart*, Bernard Berthomieu, Paulo Garcia, Idris Ibrahim, Greg Michaelson, Andrew Wallace, ATVA, 2018.
  - **Shared-variables synchronization approaches on dynamic dataflow programs**, Apostolos Modas, Simone Casale Brunet, *Robert Stewart*, Junaid Jameel Ahmad, Endri Bezati, Marco Mattavelli, IEEE SiPS, 2018.
  - **Graph Reduction Hardware Revisited** *Robert Stewart*, Evgenij Belikov, Hans-Wolfgang Loidl, Paulo Garcia, Springer LNCS, TFP, 2018.

Section 5

Goals

Q5 — Goals

- **Heriot Watt**:
- **Imperial**:
- **Glasgow**:
  - Further develop Cordial and generate HDL & IP-XACT
  - Talks at Edinburgh
  - Papers on theory being Cordial, and Cordial itself.
- **Project**
  - Inter-Group Collaboration

- **Q5** — Reading + Talking + Visiting + Squiggling + Bodging + **?**

Section 6

AOCB

Section 7

Next Meeting