Overview of VDM-RT Constructs and Semantic Issues

Kenneth Lausdahl[1] Marcel Verhoef[2] Peter Gorm Larsen[1] Sune Wolff[3,1]

[1]Aarhus School of Engineering, Dalgas Avenue 2, DK-8000 Aarhus C, Denmark

[2]Chess, PO Box 5021, 2000 CA Haarlem, The Netherlands

[3]Terma A/S, Hovmarken 4, DK-8520 Lystrup, Denmark

13 September 2010 / 8th Overture Workshop

Outline

Outline

Language Evolution

1. VDM-SL: Aimed at specifying sequential programs.
2. VDM-PP: Added object-orientation and concurrency.
3. VICE: Added real-time constrains.
4. VDM-RT: Added distribution, communication and deployment.

Motivation
Why create a Real-Time Extension of VDM

To model distributed real-time systems and:

- do analysis of alternative deployment architectures
- express and validate time constraints
- Provide better tool support.

Case study with VICE failed

Case study with VDM In Constrain Environment (VICE) applied to a realistic real-time embedded system concluded that the VICE extension to VDM++ was insufficient to model a realistic real-time system. This lead to the need for VDM Real-Time extending VICE to support distribution.

Motivation
Why create a Real-Time Extension of VDM

To model distributed real-time systems and:

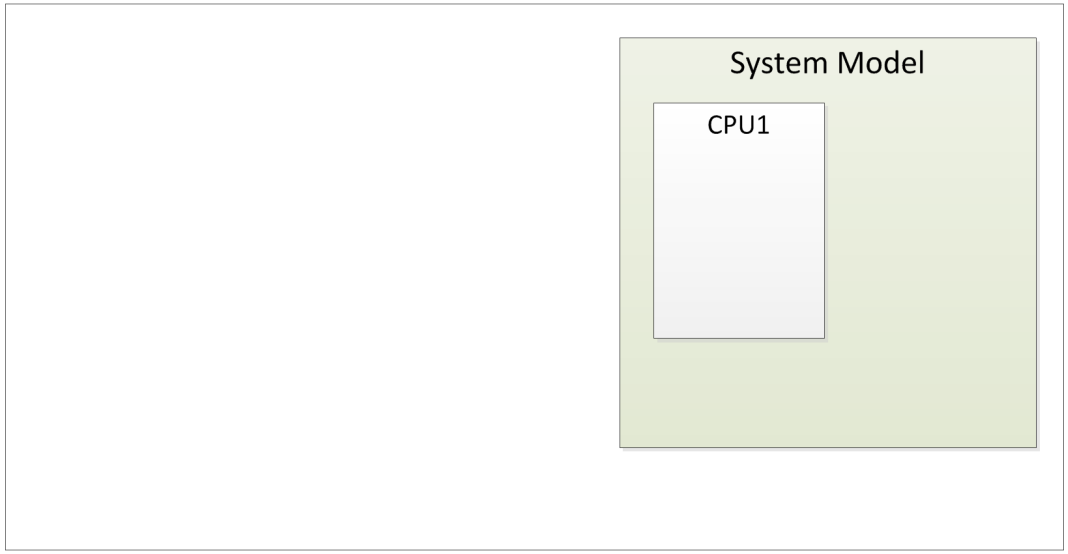
- do analysis of alternative deployment architectures
- express and validate time constraints
- Provide better tool support.

Case study with VICE failed

Case study with VDM In Constrain Environment (VICE) applied to a realistic real-time embedded system concluded that the VICE extension to VDM++ was insufficient to model a realistic real-time system. This lead to the need for VDM Real-Time extending VICE to support distribution.

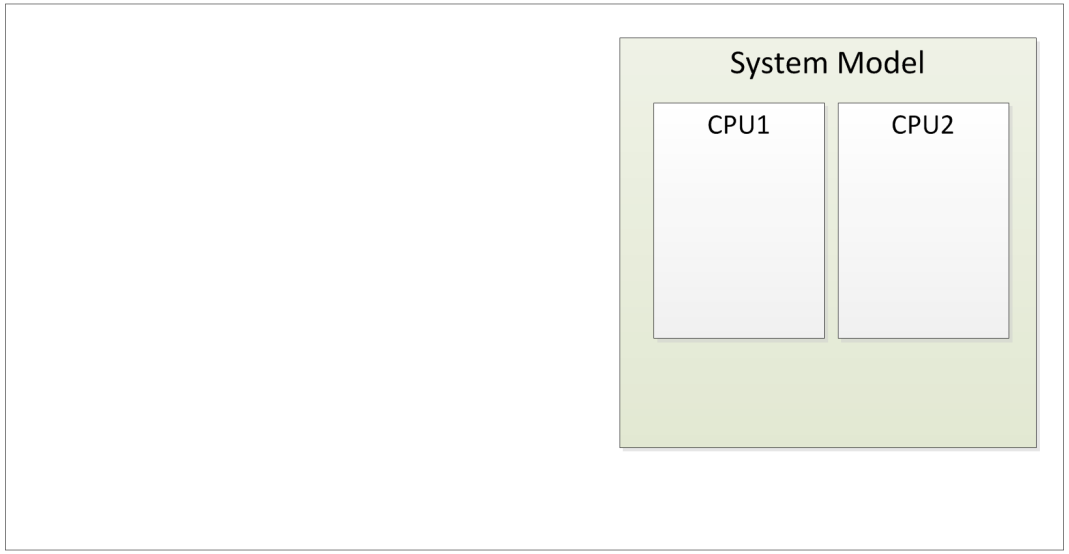
Motivation
VICE ⇒ VDM-RT

- Time
- Single System CPU
- Durations



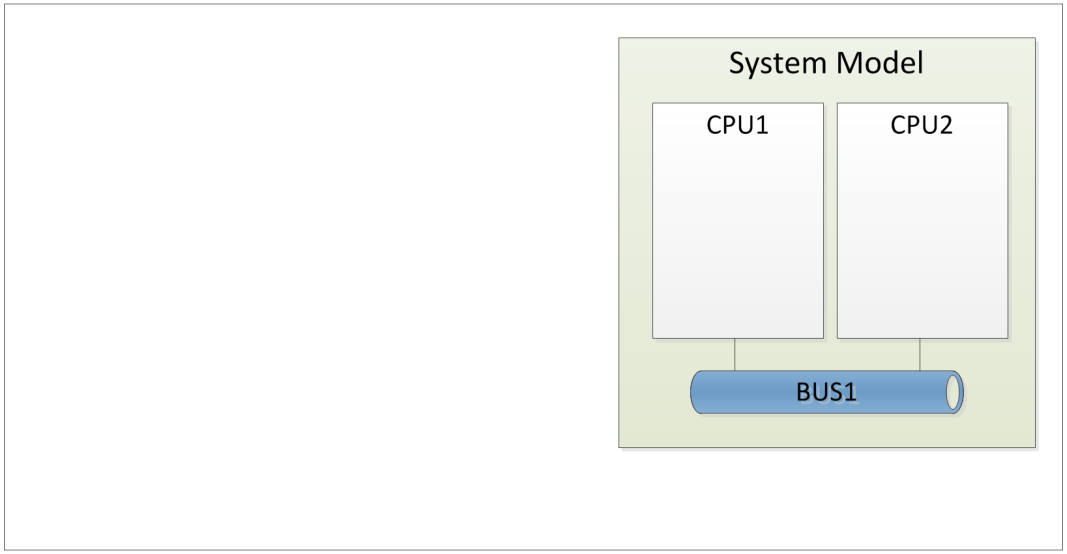
Motivation
VICE ⇒ VDM-RT

- Time
- Multiple System CPU
- Durations



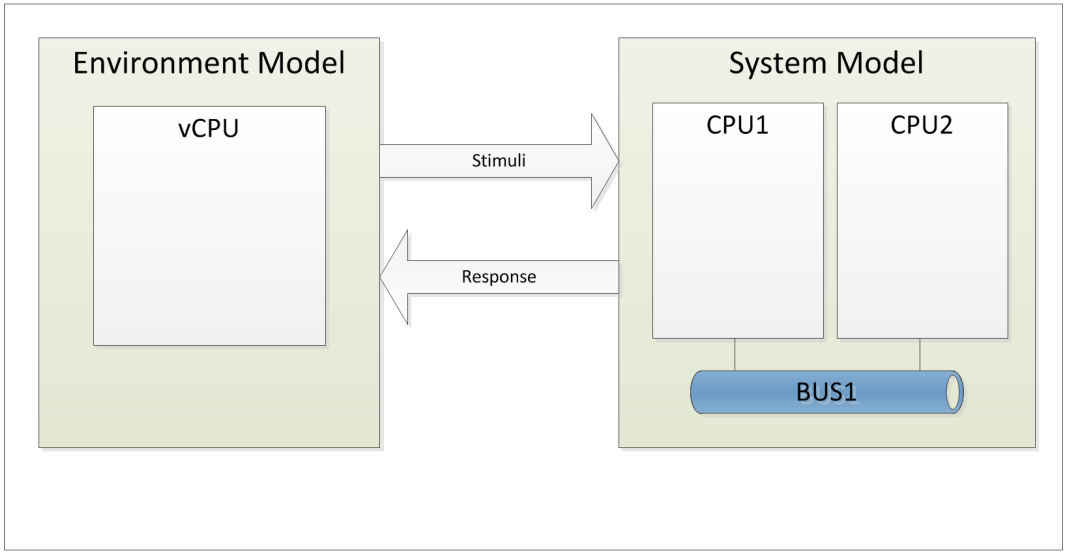
Motivation
VICE ⇒ VDM-RT

- Time
- Multiple System CPU
- Durations
- BUSes



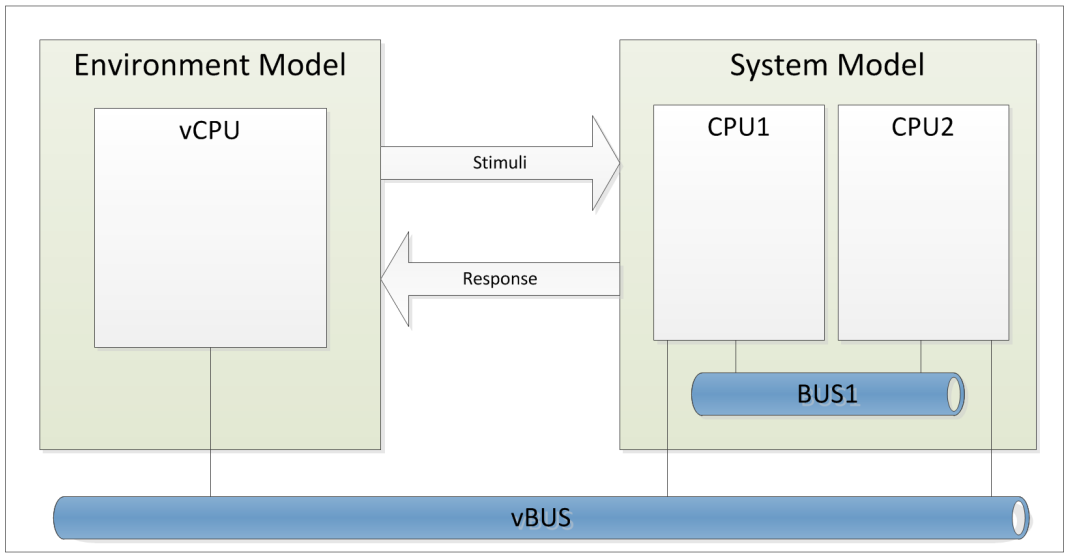
Motivation
VICE ⇒ VDM-RT

- Time
- Multiple System CPU
- Durations
- BUSes
- Virtual CPU



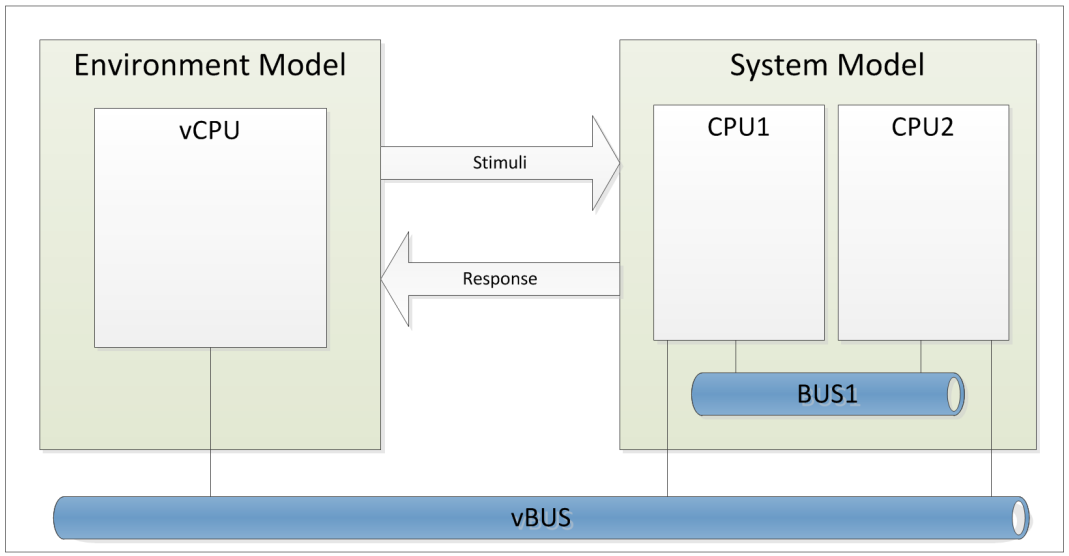
Motivation
VICE ⇒ VDM-RT

- Time
- Multiple System CPU
- Durations
- BUSes
- Virtual CPU
- Virtual BUS



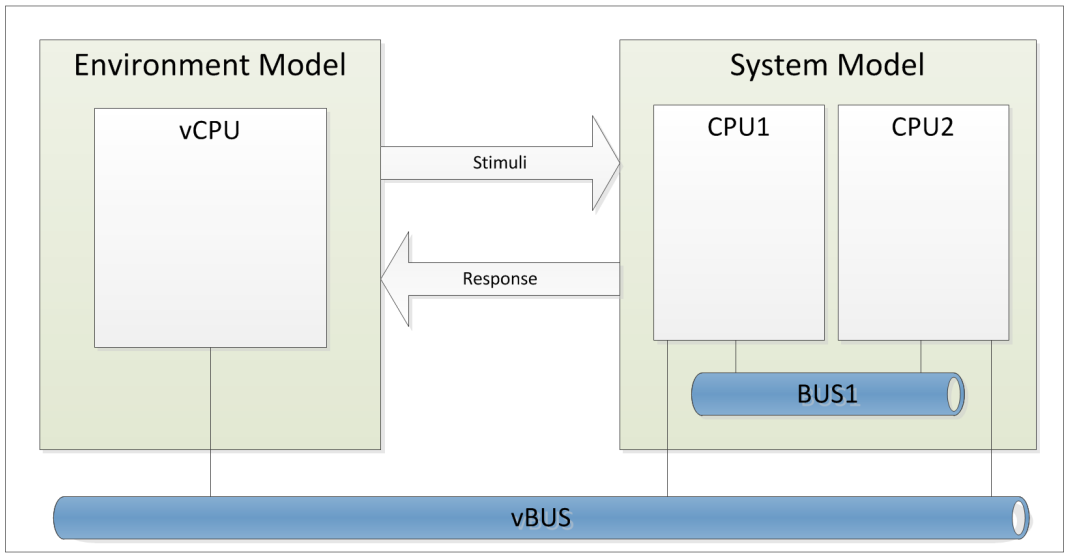
Motivation
VICE ⇒ VDM-RT

- Time
- Multiple System CPU
- Durations
- BUSes
- Virtual CPU
- Virtual BUS
- Cycles



Motivation
VICE ⇒ VDM-RT

- Time
- Multiple System CPU
- Durations
- BUSes
- Virtual CPU
- Virtual BUS
- Cycles
- Periodic Threads



Motivation
Semantics effort

Well founded Interpreter for VDM-RT

- Improve Tool support for VDM Real-Time
- Create semantic definitions to clarify known issues
- Extend the semantics of VDM-RT to support Co-simulation in the continuous time domain

Motivation
Extension with Co-Simulation

- Marcel Verhoef initial study of co-simulation $\Rightarrow$ DESTECS
- Bridge the gap between engineering disciplines.
- Enable system modeling and validation.
- Enable a more powerful and detailed controller model for continuous time models.
- Introduce a realistic model of an environment to VDM.

Outline

Existing Semantic Definitions

Much has already been done:

- VDM-SL ISO standard: Dynamic semantic (denotational style)
- VDMTools: executable subset, Mainly written in VDM-SL (proprietary CSK)
- VDM-PP: Afrodite papers, (Lano et al.)
- VDM-RT and Co-simulation: PhD by Marcel Verhoef
- Other informal descriptions

So much has been done but no consolidated full semantics definition exists for VDM-RT.

Existing Semantic Definitions

Much has already been done:

- VDM-SL ISO standard: Dynamic semantic (denotational style)
- VDMTools: executable subset, Mainly written in VDM-SL (proprietary CSK)
- VDM-PP: Afrodite papers, (Lano et al.)
- VDM-RT and Co-simulation: PhD by Marcel Verhoef
- Other informal descriptions

So much has been done but no consolidated full semantics definition exists for VDM-RT.

Outline

Overview of Issues

Overview of Issues

Topics:

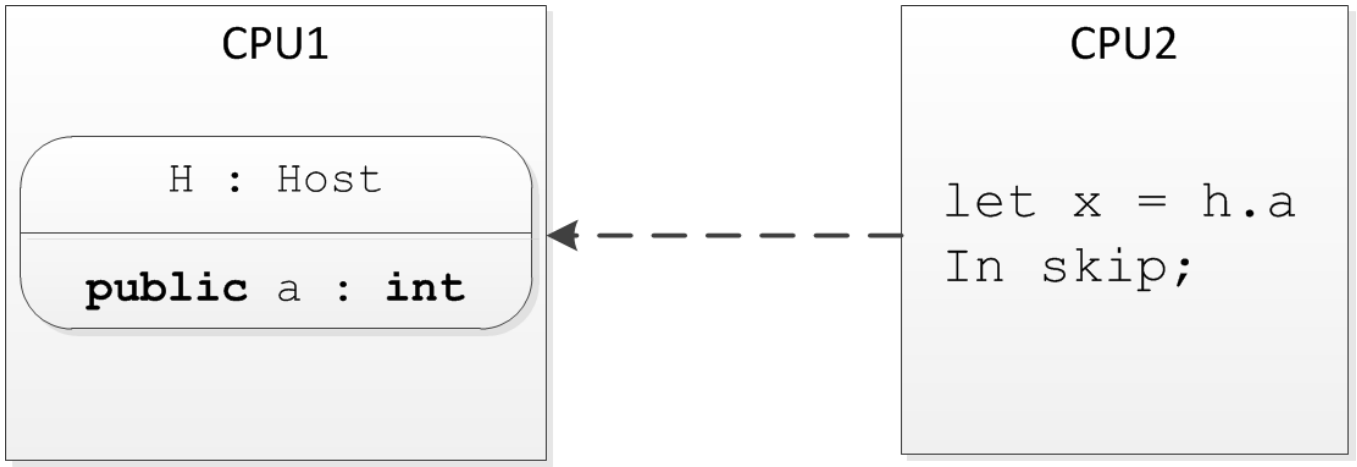
1. I1: Public Variables
2. I2: Static Variables
3. I3: Static Operations calls
4. I4: Time advance and periodic threads
5. I5: Interrupts
6. I6: Measurement of time in VDM-RT
7. I7: Limitation of Co-Simulation interface

Outline

I1: Public Variables

Issue

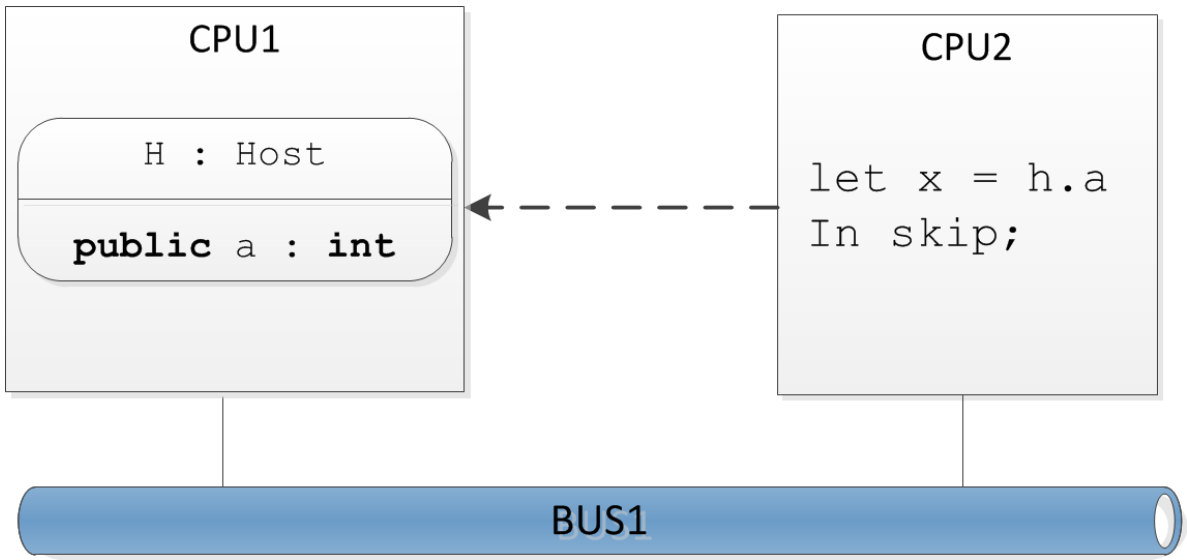
- The distributed architecture is not taken into account; No BUS communication
- Public variables are instantly available



I1: Public Variables
Suggestion

Suggestion

All access to public variables must be done through BUS communication if the caller is located on different CPU than the instance of the public variable.



I2: Static variables

Issue

In an ordinary programming language:
"A static variable is a variable that has been allocated statically"

What is a *static* variable in a distributed system?

VDM-RT static variable

Is it globally static and available from all CPUs?

I2: Static variables

Issue

In an ordinary programming language:
"A static variable is a variable that has been allocated statically"

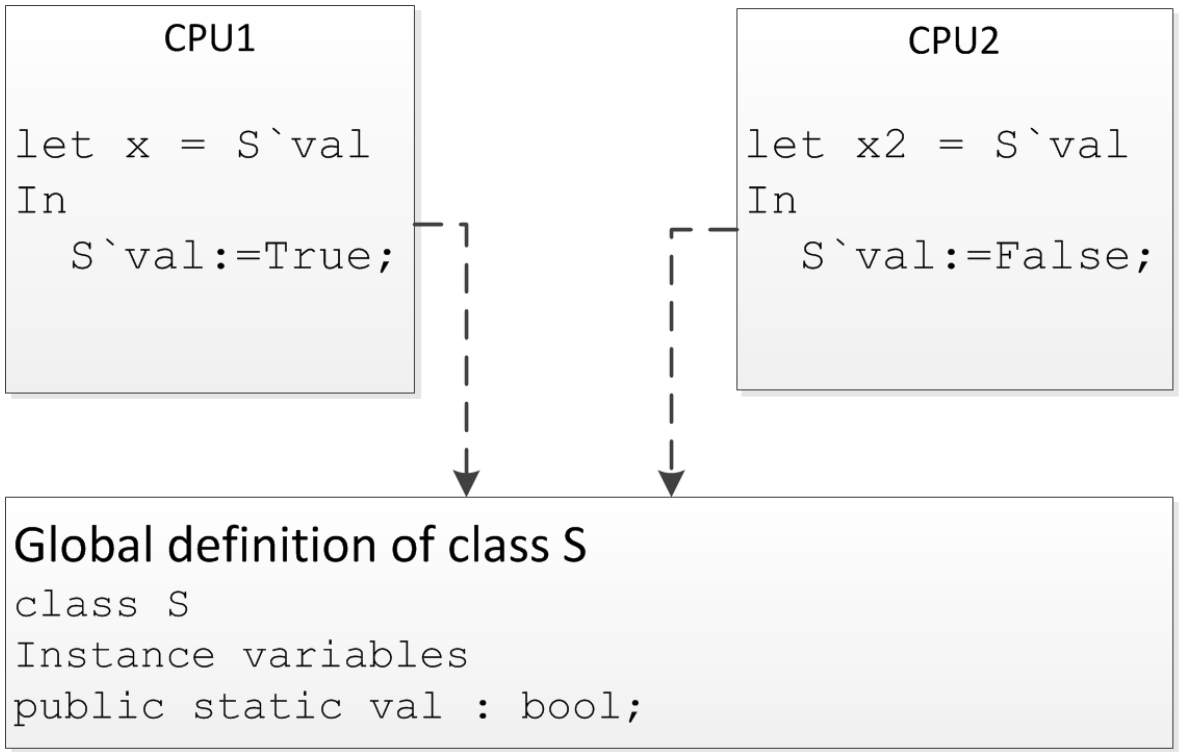
What is a *static* variable in a distributed system?

VDM-RT static variable

Is it globally static and available from all CPUs?

I2: Static variables

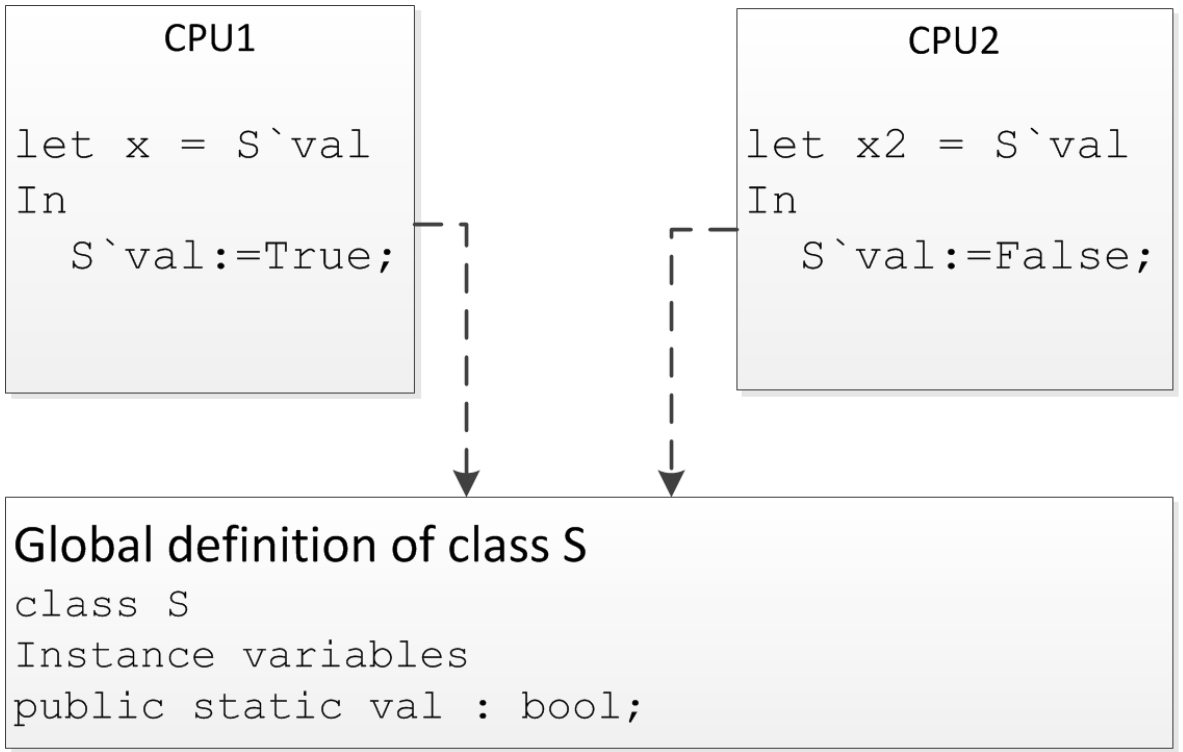
Issue 2/2



This is not possible to realize in a distributed implementation.

I2: Static variables

Issue 2/2

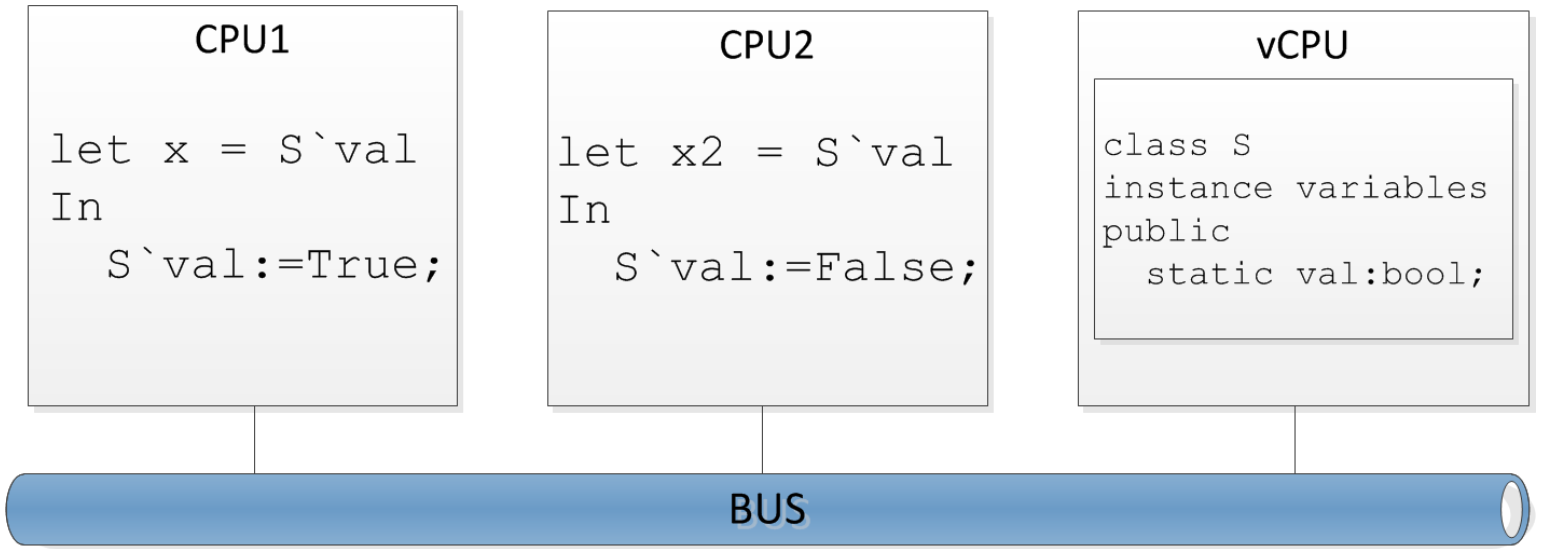


This is not possible to realize in a distributed implementation.

I2: Static variables

Suggestion

To solve the problem of distribution a static variable can either be distributed on change or made static within a single CPU only.



Distribution of static variables through a BUS.

We do not recommend this.

I2: Static variables

Suggestion

To solve the problem of distribution a static variable can either be distributed on change or made static within a single CPU only.

CPU1

```
class S
instance variables
public
  static val:bool;

let x = S`val
In
  S`val := True;
```

CPU2

```
class S
instance variables
public
  static val:bool;

let x2 = S`val
In
  S`val := False;
```

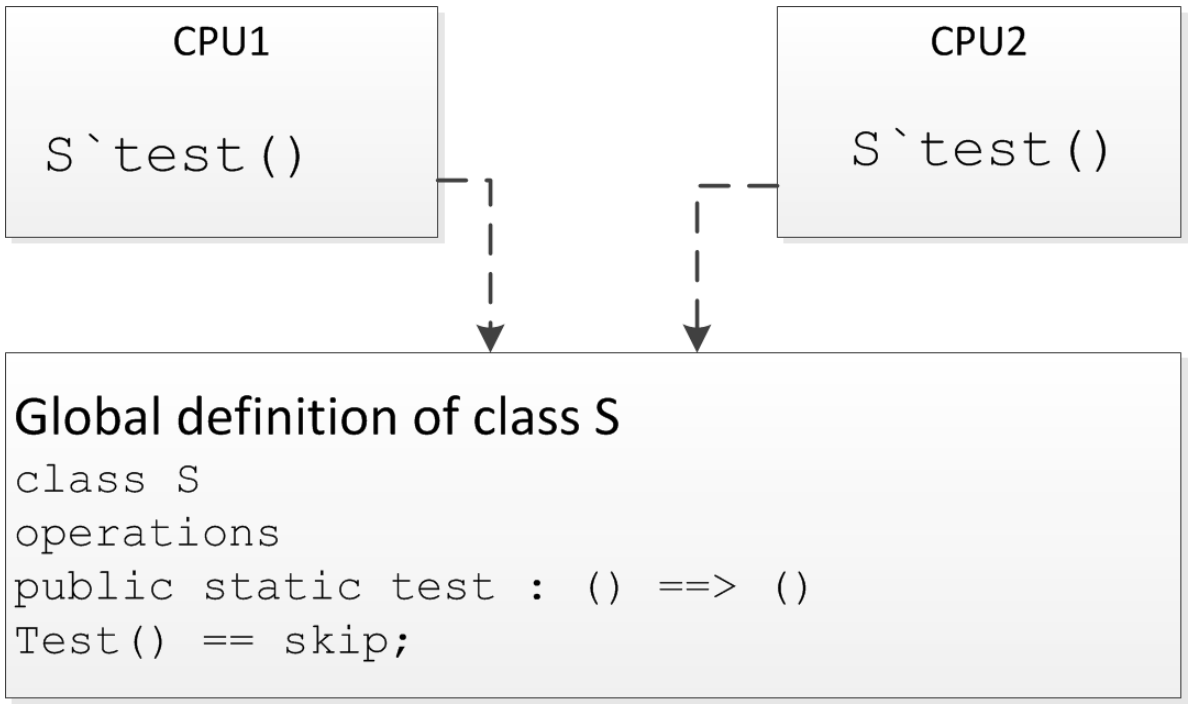
Statically allocated per CPU.

We recommend this.

I3: Static operations

Issue

Distribution is not taken into account.



I3: Static operations
Issue 2/2

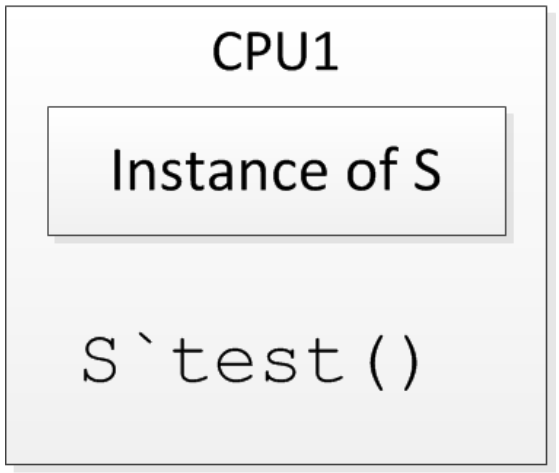
VDM-RT static operations

1. Static operations are always available independent of deployment
2. Static call do not use BUS communication
3. When used to access static variables the issue I2: Static variables applies

I3: Static operations

Suggestion: Definition Available on CPU

When definition is available locally: All execution is done locally.

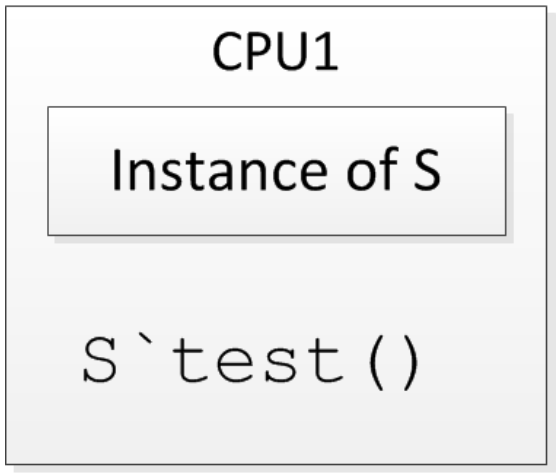


We recommend this.

I3: Static operations

Suggestion: Definition Available on CPU

When definition is available locally: All execution is done locally.

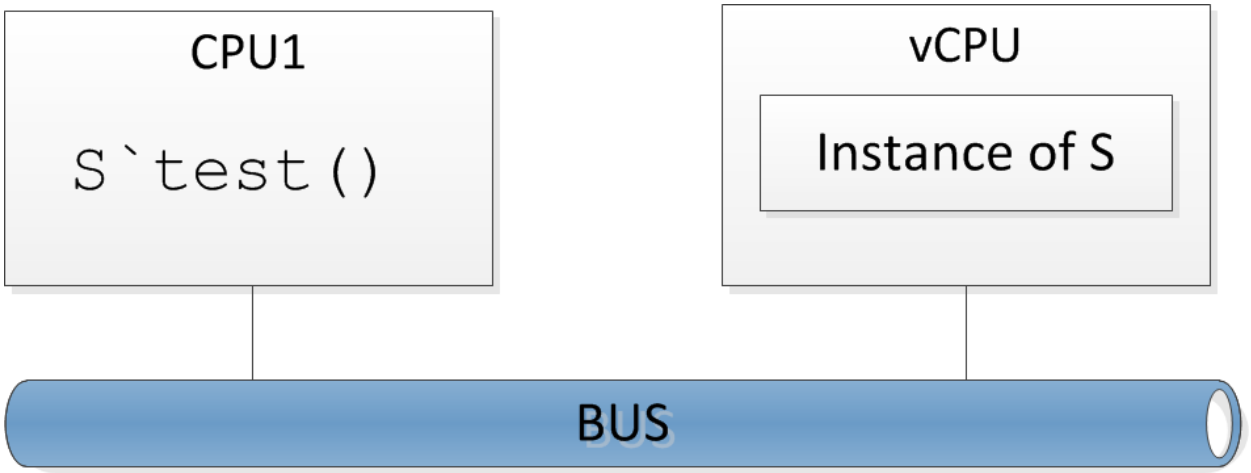


We recommend this.

I3: Static operations

Suggestion: Definition Not Available on CPU

When definition is not available locally: Execution is done through a BUS.

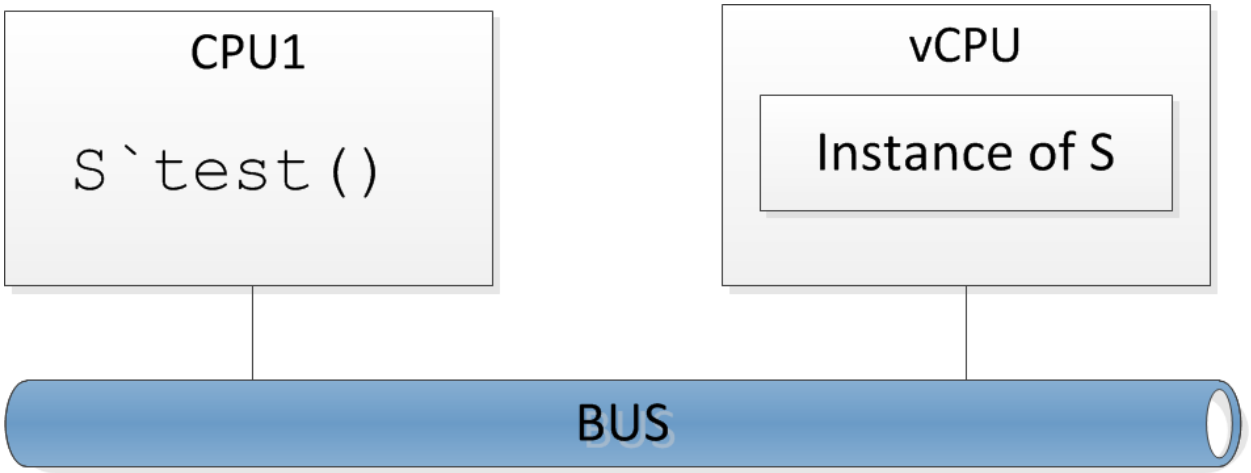


We recommend this.

I3: Static operations

Suggestion: Definition Not Available on CPU

When definition is not available locally: Execution is done through a BUS.



We recommend this.

I4: Time advance and periodic threads

Issue

Missing time advance cause models to wrongly deadlock.

Time advances when:

- Expressions and statements are executed on system CPUs.
- Duration or cycle statement are executed on any CPU.

Issue:

If a all execution in a model is blocked by permission predicates and a periodic thread exists then the model is deadlocked.

I4: Time advance and periodic threads

Issue

Missing time advance cause models to wrongly deadlock.

Time advances when:

- Expressions and statements are executed on system CPUs.
- Duration or cycle statement are executed on any CPU.

Issue:

If a all execution in a model is blocked by permission predicates and a periodic thread exists then the model is deadlocked.

I4: Time advance and periodic threads

Issue

Missing time advance cause models to wrongly deadlock.

Time advances when:

- Expressions and statements are executed on system CPUs.
- Duration or cycle statement are executed on any CPU.

Issue:

If a all execution in a model is blocked by permission predicates and a periodic thread exists then the model is deadlocked.

I4: Time advance and periodic threads

Issue Example

```
class A

public main : () ==> ()
main() ==
( start(self);
  block(); -- deadlock
)
public move : () ==> ()
move() == skip;

periodic(10,0,0,0)(move);

per block => false;
```

Time progress:

- 0 `start(self)`
- 2 `block()`
- 2 `move`
- 4 Model is deadlocked

I4: Time advance and periodic threads

Issue Example

```
class A

public main : () ==> ()
main() ==
( start(self);
  block(); -- deadlock
)
public move : () ==> ()
move() == skip;

periodic(10,0,0,0)(move);

per block => false;
```

Time progress:

- 0 `start(self)`
- 2 `block()`
- 2 `move`
- 4 Model is deadlocked

I4: Time advance and periodic threads

Issue Example

```
class A

public main : () ==> ()
main() ==
( start(self);
  block(); -- deadlock
)
public move : () ==> ()
move() == skip;

periodic(10,0,0,0)(move);

per block => false;
```

Time progress:

- 0 `start(self)`
- 2 `block()`
- 2 `move`
- 4 Model is deadlocked

I4: Time advance and periodic threads

Issue Example

```
class A

public main : () ==> ()
main() ==
( start(self);
  block(); -- deadlock
)
public move : () ==> ()
move() == skip;

periodic(10,0,0,0)(move);

per block => false;
```

Time progress:

- 0 `start(self)`
- 2 `block()`
- 2 `move`
- 4 Model is deadlocked

I4: Time advance and periodic threads

Issue Example

```
class A

public main : () ==> ()
main() ==
( start(self);
  block(); -- deadlock
)
public move : () ==> ()
move() == skip;

periodic(10,0,0,0)(move);

per block => false;
```

Time progress:

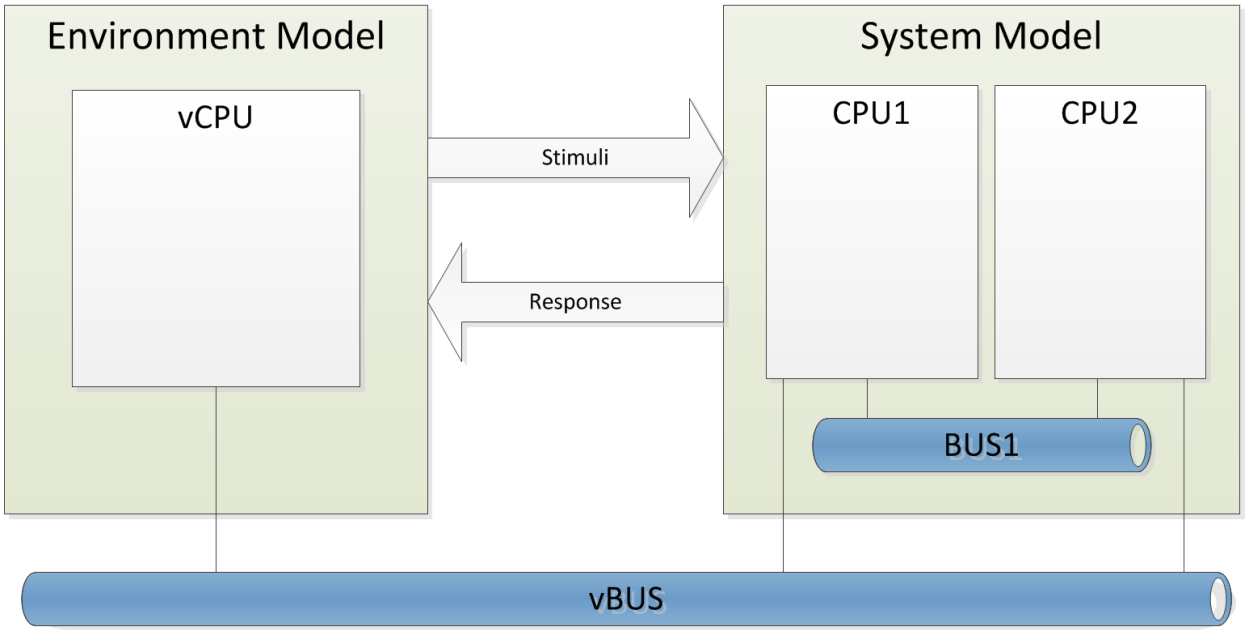
- 0 `start(self)`
- 2 `block()`
- 2 `move`
- 4 Model is deadlocked

Should be able to move to 10

Outline

I5: Interrupts
A normal VDM-RT model

A standard VDM-RT model will contain an environment class running on the vCPU with periodic threads which feeds the system model with events.

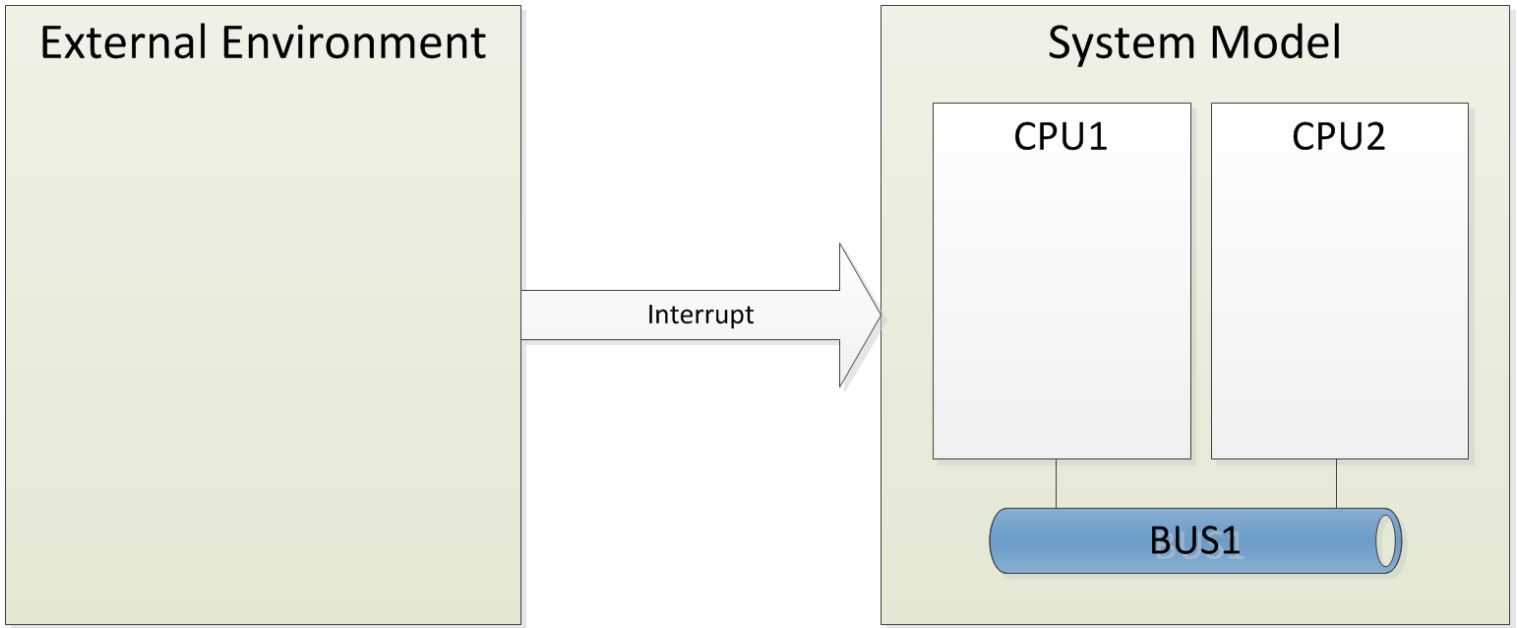


I5: Interrupts

Why do we want interrupts?

Interrupts are desired for co-simulation of interrupt driven controllers.

- Enables incoming events to be prioritized.
- Enables quick reaction to external environment changes.



I5: Interrupts

What needs to be added?

The VDM-RT semantics must be extended to include

- Interrupt handlers
- Association of interrupt handlers and target CPUs
- Invocation/Activation of interrupt handlers
- Prioritize interrupts and suspend normal execution

I5: Interrupts
Example Interrupt Handler

Definition of an generic interrupt handler:

```
class IntHandler
operations
public async notify : () ==> ()
notify == is subclass responsibility;
end IntHandler
```

A specific handler for a button press:

```
class ButtonPressIntHandler
  is subclass of IntHandler
public async notify : () ==> ()
notify == ...
end ButtonPressIntHandler
```

I5: Interrupts
Example Interrupt Handler

Definition of an generic interrupt handler:

```
class IntHandler
operations
public async notify : () ==> ()
notify == is subclass responsibility;
end IntHandler
```

A specific handler for a button press:

```
class ButtonPressIntHandler
  is subclass of IntHandler
public async notify : () ==> ()
notify == ...
end ButtonPressIntHandler
```

I5: Interrupts
Example Interrupt Handler

Definition of an generic interrupt handler: embedded in tool

```
class IntHandler
operations
public async notify : () ==> ()
notify == is subclass responsibility;
end IntHandler
```

A specific handler for a button press: user defined

```
class ButtonPressIntHandler
  is subclass of IntHandler
public async notify : () ==> ()
notify == ...
end ButtonPressIntHandler
```

I5: Interrupts
Example Registering Handlers

Associate a target CPU to a specific interrupt `handler` where the execution will take place when the interrupt occurs.

```
system S
...
handler : ButtonPressIntHandler;
...
cpu1.regIntHandler(handler,MAX_PRIORITY);
...
end S
```

I6: Measurement of time in VDM-RT

Issue

No semantics definition exists which defined the relation between `CPU`,`BUS`, `durations` and `cycles`.

Recommended architecture definition

Time units in VDM-RT are currently unspecified.

- **CPU** The capacity, **`nat`** *million instructions per second*.
- **BUS** The capacity, **`nat`**.
- **Duration** A duration, **`nat`**, *milliseconds*.
- **Cycles** The cycles, **`nat`** and converted to time through a CPU capacity.

All expressions and statements have default durations given making them independent of the architecture.

I6: Measurement of time in VDM-RT

Suggestion

Introduce time units with unique relationship in VDM-RT

Giving time units to time

CPU	The capacity, **`nat`** given in *[Hz]*. (calculations per second).
BUS	The capacity, **`nat`** given in *[bits/s]*.
Duration	A duration is given as **`nat`** in *[s]*.

All expressions and statements should have *default cycles* defined, depending them on the architecture.

I6: Measurement of time in VDM-RT

Calculating time for assignment

Creating default cycles for all expressions and statements to the architecture, enables timing to change based on the CPU capacity. As oppose to default durations.

Cycles of VDM assignment `x := y + z`	
Instruction	Description
mov	Move level to registry
mov	Move x to registry
cmp	Combine the two registrars
mov	Move result back to location of level
4	Total cycles

Table: Calculation of VDM equals by use of assembly instructions.

I6: Measurement of time in VDM-RT

Calculate time based on CPU speed

Using CPU speed to calculate the time which can be used for synchronization:

Example with a CPU of 2 KHz

- 2000 cycles can be calculated each 1 second
- Assignment `x := y + z` takes 4 cycles to complete

Thus assignment takes 0.002 seconds to complete. This way each time step can be converted to a time value in seconds.

I7: Limitation of Co-Sim interface

Limitation

- Only simple types can be transferred: **`int`**, **`bool`**, **`real`**/double.
- No complex types such as custom types or class hierarchies.
- No invariants. If enabled when should they hold?
- Decimal numbers must have a precision defined in the interface

Outline

Group Discussion

Discussion in groups of 4 persons.

Topics:

1. I1: Public Variables
2. I2: Static Variables
3. I3: Static Operations calls
4. I4: Time advance and periodic threads
5. I5: Interrupts
6. I6: Measurement of time in VDM-RT
7. I7: Limitation of Co-Simulation interface

Outline

Strategy

- Define core abstract syntax (CAS) for VDM
- Define mappings from VDM dialects to CAS
- Define CAS semantics

Discussion of Semantic Style

- Axiomatic
- Denotational
- Operational

Discussion of Proposed Plan