ForSyDe's embedded compiler

First development stage results.

Alfonso Acosta
alfonso.acosta@gmail.com

ICT/ECS
Royal Institute of Technology, Stockholm

January 14th, 2008

Outline

Outline

General Goal Review

- During this first stage it was agreed to obtain a robust implementation attaining to this particular goals:
  - Finish the implementation of components (previously named Blocks and Ports).
  - Add a simulation backend with support for **any** signal type.
  - Support all the synchronous process constructors in ForSyDe.
  - Improve the error handling and reporting of the compiler.
  - Optionally. Document the code with haddock and cabalize the project.
  - Create a project webpage
  - Improve the VHDL backend.
- Additional work done
  - The project was rewritten from scratch (slightly based on the old code).
    - New module hierarchy
    - More general identifiers.
    - Access to external scope of `ProcFuns`.
    - Redesigned component API.

General Goal Review

- During this first stage it was agreed to obtain a robust implementation attaining to this particular goals:
  - Finish the implementation of components (previously named Blocks and Ports).
  - Add a simulation backend with support for **any** signal type.
  - Support all the synchronous process constructors in ForSyDe.
  - Improve the error handling and reporting of the compiler.
  - Optionally. Document the code with haddock and cabalize the project.
  - Create a project webpage
  - Improve the VHDL backend.
- Additional work done
  - The project was rewritten from scratch (slightly based on the old code).
    - New module hierarchy
    - More general identifiers.
    - Access to external scope of `ProcFuns`.
    - Redesigned component API.

General Goal Review

- During this first stage it was agreed to obtain a robust implementation attaining to this particular goals:
  - Finish the implementation of components (previously named Blocks and Ports). figures/tick
  - Add a simulation backend with support for **any** signal type.
  - Support all the synchronous process constructors in ForSyDe.
  - Improve the error handling and reporting of the compiler.
  - Optionally. Document the code with haddock and cabalize the project.
  - Create a project webpage
  - Improve the VHDL backend.
- Additional work done
  - The project was rewritten from scratch (slightly based on the old code).
    - New module hierarchy
    - More general identifiers.
    - Access to external scope of `ProcFuns`.
    - Redesigned component API.

General Goal Review

- During this first stage it was agreed to obtain a robust implementation attaining to this particular goals:
  - Finish the implementation of components (previously named Blocks and Ports). figures/tick
  - Add a simulation backend with support for **any** signal type.
  - Support all the synchronous process constructors in ForSyDe.
  - Improve the error handling and reporting of the compiler.
  - Optionally. Document the code with haddock and cabalize the project.
  - Create a project webpage
  - Improve the VHDL backend.
- Additional work done
  - The project was rewritten from scratch (slightly based on the old code).
    - New module hierarchy
    - More general identifiers.
    - Access to external scope of `ProcFuns`.
    - Redesigned component API.

General Goal Review

- During this first stage it was agreed to obtain a robust implementation attaining to this particular goals:
  - Finish the implementation of components (previously named Blocks and Ports). figures/tick
  - Add a simulation backend with support for **any** signal type. figures/tick
  - Support all the synchronous process constructors in ForSyDe.
  - Improve the error handling and reporting of the compiler.
  - Optionally. Document the code with haddock and cabalize the project.
  - Create a project webpage
  - Improve the VHDL backend.
- Additional work done
  - The project was rewritten from scratch (slightly based on the old code).
    - New module hierarchy
    - More general identifiers.
    - Access to external scope of `ProcFuns`.
    - Redesigned component API.

General Goal Review

- During this first stage it was agreed to obtain a robust implementation attaining to this particular goals:
  - Finish the implementation of components (previously named Blocks and Ports). figures/tick
  - Add a simulation backend with support for **any** signal type. figures/tick
  - Support all the synchronous process constructors in ForSyDe.
  - Improve the error handling and reporting of the compiler.
  - Optionally. Document the code with haddock and cabalize the project.
  - Create a project webpage
  - Improve the VHDL backend.
- Additional work done
  - The project was rewritten from scratch (slightly based on the old code).
    - New module hierarchy
    - More general identifiers.
    - Access to external scope of `ProcFuns`.
    - Redesigned component API.

General Goal Review

- During this first stage it was agreed to obtain a robust implementation attaining to this particular goals:
  - Finish the implementation of components (previously named Blocks and Ports). figures/tick
  - Add a simulation backend with support for **any** signal type. figures/tick
  - Support all the synchronous process constructors in ForSyDe. figures/tick
  - Improve the error handling and reporting of the compiler.
  - Optionally. Document the code with haddock and cabalize the project.
  - Create a project webpage
  - Improve the VHDL backend.
- Additional work done
  - The project was rewritten from scratch (slightly based on the old code).
    - New module hierarchy
    - More general identifiers.
    - Access to external scope of `ProcFuns`.
    - Redesigned component API.

General Goal Review

- During this first stage it was agreed to obtain a robust implementation attaining to this particular goals:
  - Finish the implementation of components (previously named Blocks and Ports). figures/tick
  - Add a simulation backend with support for **any** signal type. figures/tick
  - Support all the synchronous process constructors in ForSyDe. figures/tick
  - Improve the error handling and reporting of the compiler.
  - Optionally. Document the code with haddock and cabalize the project.
  - Create a project webpage
  - Improve the VHDL backend.
- Additional work done
  - The project was rewritten from scratch (slightly based on the old code).
    - New module hierarchy
    - More general identifiers.
    - Access to external scope of `ProcFuns`.
    - Redesigned component API.

General Goal Review

- During this first stage it was agreed to obtain a robust implementation attaining to this particular goals:
  - Finish the implementation of components (previously named Blocks and Ports). figures/tick
  - Add a simulation backend with support for **any** signal type. figures/tick
  - Support all the synchronous process constructors in ForSyDe. figures/tick
  - Improve the error handling and reporting of the compiler. figures/tick
  - Optionally. Document the code with haddock and cabalize the project.
  - Create a project webpage
  - Improve the VHDL backend.
- Additional work done
  - The project was rewritten from scratch (slightly based on the old code).
    - New module hierarchy
    - More general identifiers.
    - Access to external scope of `ProcFuns`.
    - Redesigned component API.

General Goal Review

- During this first stage it was agreed to obtain a robust implementation attaining to this particular goals:
    - Finish the implementation of components (previously named Blocks and Ports). figures/tick
    - Add a simulation backend with support for **any** signal type. figures/tick
    - Support all the synchronous process constructors in ForSyDe. figures/tick
    - Improve the error handling and reporting of the compiler. figures/tick
    - Optionally. Document the code with haddock and cabalize the project.
    - Create a project webpage
    - Improve the VHDL backend.
- Additional work done
    - The project was rewritten from scratch (slightly based on the old code).
        - New module hierarchy
        - More general identifiers.
        - Access to external scope of `ProcFuns`.
        - Redesigned component API.

General Goal Review

- During this first stage it was agreed to obtain a robust implementation attaining to this particular goals:
    - Finish the implementation of components (previously named Blocks and Ports). figures/tick
    - Add a simulation backend with support for **any** signal type. figures/tick
    - Support all the synchronous process constructors in ForSyDe. figures/tick
    - Improve the error handling and reporting of the compiler. figures/tick
    - Optionally. Document the code with haddock and cabalize the project. figures/tick
    - Create a project webpage
    - Improve the VHDL backend.
- Additional work done
    - The project was rewritten from scratch (slightly based on the old code).
        - New module hierarchy
        - More general identifiers.
        - Access to external scope of `ProcFuns`.
        - Redesigned component API.

General Goal Review

- During this first stage it was agreed to obtain a robust implementation attaining to this particular goals:
  - Finish the implementation of components (previously named Blocks and Ports). figures/tick
  - Add a simulation backend with support for **any** signal type. figures/tick
  - Support all the synchronous process constructors in ForSyDe. figures/tick
  - Improve the error handling and reporting of the compiler. figures/tick
  - Optionally. Document the code with haddock and cabalize the project. figures/tick
  - Create a project webpage
  - Improve the VHDL backend.
- Additional work done
  - The project was rewritten from scratch (slightly based on the old code).
    - New module hierarchy
    - More general identifiers.
    - Access to external scope of `ProcFuns`.
    - Redesigned component API.

General Goal Review

- During this first stage it was agreed to obtain a robust implementation attaining to this particular goals:
  - Finish the implementation of components (previously named Blocks and Ports). figures/tick
  - Add a simulation backend with support for **any** signal type. figures/tick
  - Support all the synchronous process constructors in ForSyDe. figures/tick
  - Improve the error handling and reporting of the compiler. figures/tick
  - Optionally. Document the code with haddock and cabalize the project. figures/tick
  - Create a project webpage figures/cross Not until there's a release
  - Improve the VHDL backend.
- Additional work done
  - The project was rewritten from scratch (slightly based on the old code).
    - New module hierarchy
    - More general identifiers.
    - Access to external scope of `ProcFuns`.
    - Redesigned component API.

General Goal Review

- During this first stage it was agreed to obtain a robust implementation attaining to this particular goals:
  - Finish the implementation of components (previously named Blocks and Ports). figures/tick
  - Add a simulation backend with support for **any** signal type.figures/tick
  - Support all the synchronous process constructors in ForSyDe.figures/tick
  - Improve the error handling and reporting of the compiler.figures/tick
  - Optionally. Document the code with haddock and cabalize the project.figures/tick
  - Create a project webpagefigures/crossNot until there's a release
  - Improve the VHDL backend.
- Additional work done
  - The project was rewritten from scratch (slightly based on the old code).
    - New module hierarchy
    - More general identifiers.
    - Access to external scope of `ProcFuns`.
    - Redesigned component API.

General Goal Review

- During this first stage it was agreed to obtain a robust implementation attaining to this particular goals:
  - Finish the implementation of components (previously named Blocks and Ports). figures/tick
  - Add a simulation backend with support for **any** signal type. figures/tick
  - Support all the synchronous process constructors in ForSyDe. figures/tick
  - Improve the error handling and reporting of the compiler. figures/tick
  - Optionally. Document the code with haddock and cabalize the project. figures/tick
  - Create a project webpage figures/cross Not until there's a release
  - Improve the VHDL backend. figures/cross Still not advanced support
- Additional work done
  - The project was rewritten from scratch (slightly based on the old code).
    - New module hierarchy
    - More general identifiers.
    - Access to external scope of `ProcFuns`.
    - Redesigned component API.

General Goal Review

- During this first stage it was agreed to obtain a robust implementation attaining to this particular goals:
  - Finish the implementation of components (previously named Blocks and Ports). figures/tick
  - Add a simulation backend with support for **any** signal type.figures/tick
  - Support all the synchronous process constructors in ForSyDe.figures/tick
  - Improve the error handling and reporting of the compiler.figures/tick
  - Optionally. Document the code with haddock and cabalize the project.figures/tick
  - Create a project webpagefigures/crossNot until there's a release
  - Improve the VHDL backend.figures/crossStill not advanced support
- Additional work done
  - The project was rewritten from scratch (slightly based on the old code).
    - New module hierarchy
    - More general identifiers.
    - Access to external scope of `ProcFun`s.
    - Redesigned component API.

General Goal Review

- During this first stage it was agreed to obtain a robust implementation attaining to this particular goals:
  - Finish the implementation of components (previously named Blocks and Ports). figures/tick
  - Add a simulation backend with support for **any** signal type.figures/tick
  - Support all the synchronous process constructors in ForSyDe.figures/tick
  - Improve the error handling and reporting of the compiler.figures/tick
  - Optionally. Document the code with haddock and cabalize the project.figures/tick
  - Create a project webpagefigures/crossNot until there's a release
  - Improve the VHDL backend.figures/crossStill not advanced support
- Additional work done
  - The project was rewritten from scratch (slightly based on the old code).
    - New module hierarchyfigures/tick
    - More general identifiers.
    - Access to external scope of `ProcFun`s.
    - Redesigned component API.

General Goal Review

- During this first stage it was agreed to obtain a robust implementation attaining to this particular goals:
  - Finish the implementation of components (previously named Blocks and Ports). figures/tick
  - Add a simulation backend with support for **any** signal type.figures/tick
  - Support all the synchronous process constructors in ForSyDe.figures/tick
  - Improve the error handling and reporting of the compiler.figures/tick
  - Optionally. Document the code with haddock and cabalize the project.figures/tick
  - Create a project webpagefigures/crossNot until there's a release
  - Improve the VHDL backend.figures/crossStill not advanced support
- Additional work done
  - The project was rewritten from scratch (slightly based on the old code).
    - New module hierarchyfigures/tick
    - More general identifiers.figures/tick
    - Access to external scope of `ProcFun`s.
    - Redesigned component API.

General Goal Review

- During this first stage it was agreed to obtain a robust implementation attaining to this particular goals:
    - Finish the implementation of components (previously named Blocks and Ports). figures/tick
    - Add a simulation backend with support for **any** signal type.figures/tick
    - Support all the synchronous process constructors in ForSyDe.figures/tick
    - Improve the error handling and reporting of the compiler.figures/tick
    - Optionally. Document the code with haddock and cabalize the project.figures/tick
    - Create a project webpagefigures/crossNot until there's a release
    - Improve the VHDL backend.figures/crossStill not advanced support
- Additional work done
    - The project was rewritten from scratch (slightly based on the old code).
        - New module hierarchyfigures/tick
        - More general identifiers.figures/tick
        - Access to external scope of `ProcFun`s.figures/tick
        - Redesigned component API.

General Goal Review

- During this first stage it was agreed to obtain a robust implementation attaining to this particular goals:
    - Finish the implementation of components (previously named Blocks and Ports). figures/tick
    - Add a simulation backend with support for **any** signal type.figures/tick
    - Support all the synchronous process constructors in ForSyDe.figures/tick
    - Improve the error handling and reporting of the compiler.figures/tick
    - Optionally. Document the code with haddock and cabalize the project.figures/tick
    - Create a project webpagefigures/crossNot until there's a release
    - Improve the VHDL backend.figures/crossStill not advanced support
- Additional work done
    - The project was rewritten from scratch (slightly based on the old code).
        - New module hierarchyfigures/tick
        - More general identifiers.figures/tick
        - Access to external scope of `ProcFun`s.figures/tick
        - Redesigned component API.figures/tick

Outline

Outline

Components

- Components were reimplemented from scratch, creating a more functional and intuitive API.
- Let's see a simple example. Design a serial adder using components in 5 simple steps.

Components

- Components were reimplemented from scratch, creating a more functional and intuitive API.
- Let's see a simple example. Design a serial adder using components in 5 simple steps.

figures/SeqAddFour

Components

- Components were reimplemented from scratch, creating a more functional and intuitive API.
- Let's see a simple example. Design a serial adder using components in 5 simple steps.

figures/SeqAddFour

1) Create a process function which adds one to its input

```
addOnef :: ProcFun (Int -> Int)
addOnef = $(newProcFun [d| addOnef :: Int -> Int
                           addOnef n = n + 1 |])
```

Components

- Components were reimplemented from scratch, creating a more functional and intuitive API.
- Let's see a simple example. Design a serial adder using components in 5 simple steps.

figures/SeqAddFour

2) Create a system function corresponding to the unit adder

```
addOneProc :: Signal Int -> Signal Int
addOneProc = mapSY "addOne" addOnef
```

Components

- Components were reimplemented from scratch, creating a more functional and intuitive API.
- Let’s see a simple example. Design a serial adder using components in 5 simple steps.

figures/SeqAddFour

3) Subsystem definition associated to the unit adder

```
addOneSysDef :: SysDef (Signal Int -> Signal Int)
addOneSysDef = $(newSysDef 'addOneProc ["in1"] ["out1"])
```

Components

- Components were reimplemented from scratch, creating a more functional and intuitive API.
- Let's see a simple example. Design a serial adder using components in 5 simple steps.

figures/SeqAddFour

4) Create the main system function

```
addFour :: Signal Int -> Signal Int
addFour = $(instantiate "addOne3" 'addOneSysDef) .
          $(instantiate "addOne2" 'addOneSysDef) .
          $(instantiate "addOne1" 'addOneSysDef) .
          $(instantiate "addOne0" 'addOneSysDef)
```

Components

- Components were reimplemented from scratch, creating a more functional and intuitive API.
- Let's see a simple example. Design a serial adder using components in 5 simple steps.

figures/SeqAddFour

5) Finally, build the main system definition

```
addFourSys :: SysDef (Signal Int -> Signal Int)
addFourSys = $(newSysDef 'addFour ["in1"] ["out1"])
```

Design Flow Using Components

Outline

Supporting any `Signal` type

- Challenge: support simulating signals of any type.
- How is it possible? `Typeable` and `Lift` constraints.

```
class Typeable a where
  typeOf :: a -> TypeRep

toDyn :: Typeable a => a -> Dynamic

class Lift t where
  lift :: t -> Q Exp
```

```
delaySY :: (Typeable a, Lift a) =>
             ProcId -> a -> Signal a ->  Signal a
```

- What about the instantiation boilerplate code?
  - GHC supports automatic derivation of `Typeable`
  - I Improved Igloo's Lift library (GHC 6.10 won't need it, i.e. `instance Data a => Lift a` will probably be included).

```
data LogicVal = High | Low deriving (Eq, Typeable)
$(deriveLift1 ''LogicVal)
```

Supporting any Signal type

- Challenge: support simulating signals of any type.
- How is it possible? Typeable and Lift constraints.

```
class Typeable a where
  typeOf :: a -> TypeRep

toDyn :: Typeable a => a -> Dynamic

class Lift t where
  lift :: t -> Q Exp
```

```
delaySY :: (Typeable a, Lift a) =>
            ProcId -> a -> Signal a ->  Signal a
```

- What about the instantiation boilerplate code?
  - GHC supports automatic derivation of Typeable
  - I Improved Igloo's Lift library (GHC 6.10 won't need it, i.e. instance Data a => Lift a will probably be included).

```
data LogicVal = High | Low deriving (Eq, Typeable)
$(deriveLift1 ''LogicVal)
```

Supporting any `Signal` type

- Challenge: support simulating signals of any type.
- How is it possible? `Typeable` and `Lift` constraints.

```
class Typeable a where
  typeOf :: a -> TypeRep

toDyn :: Typeable a => a -> Dynamic

class Lift t where
  lift :: t -> Q Exp
```

```
delaySY :: (Typeable a, Lift a) =>
             ProcId -> a -> Signal a ->  Signal a
```

- What about the instantiation boilerplate code?
  - GHC supports automatic derivation of `Typeable`
  - I Improved Igloo's Lift library (GHC 6.10 won't need it, i.e. `instance Data a => Lift a` will probably be included).

```
data LogicVal = High | Low deriving (Eq, Typeable)
$(deriveLift1 ''LogicVal)
```

Supporting any `Signal` type

- Challenge: support simulating signals of any type.
- How is it possible? `Typeable` and `Lift` constraints.

```
class Typeable a where
  typeOf :: a -> TypeRep

toDyn :: Typeable a => a -> Dynamic

class Lift t where
  lift :: t -> Q Exp
```

```
delaySY :: (Typeable a, Lift a) =>
            ProcId -> a -> Signal a ->  Signal a
```

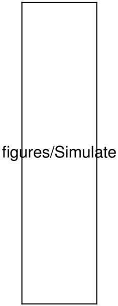
- What about the instantiation boilerplate code?
  - GHC supports automatic derivation of `Typeable`
  - I Improved Igloo's Lift library (GHC 6.10 won't need it, i.e. `instance Data a => Lift a` will probably be included).

```
data LogicVal = High | Low deriving (Eq, Typeable)
$(deriveLift1 ''LogicVal)
```

Outline

The simulation backend

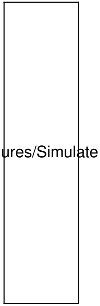
- A new sequential simulation backend has been implemented



- It detects combinational loops (i.e. loops not including a `delaySY` process).
  - The detection was impossible with the previous stream-based signal implementation.
- Completely usable but still not optimal:
  - Due to some implementation problems, simulation is strict and its efficiency could be improved.

The simulation backend

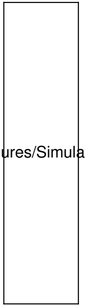
- A new sequential simulation backend has been implemented



- It detects combinational loops (i.e. loops not including a `delaySY` process).
  - The detection was impossible with the previous stream-based signal implementation.
- Completely usable but still not optimal:
  - Due to some implementation problems, simulation is strict and its efficiency could be improved.

The simulation backend

- A new sequential simulation backend has been implemented



- It detects combinational loops (i.e. loops not including a `delaySY` process).
  - The detection was impossible with the previous stream-based signal implementation.
- Completely usable but still not optimal:
  - Due to some implementation problems, simulation is strict and its efficiency could be improved.

Outline

Full support of the Synchronous Process Library

- All ForSyDe synchronous process and process constructors are supported in the frontend and the simulation backend.
  - Even polymorphic processes work (big thanks to Oleg Kisleyov for his help at haskell-cafe)
- The Equalizer was ported to the new compiler API and is correctly simulated
- The frontend already supports supplying identifiers for each process (detection of duplicates is not implemented yet)
- Error reporting was improved
  - GHC 6.10 will provide line information to Template Haskell, allowing to make error reports more useful.

Full support of the Synchronous Process Library

- All ForSyDe synchronous process and process constructors are supported in the frontend and the simulation backend.
  - Even polymorphic processes work (big thanks to Oleg Kisleyov for his help at haskell-cafe)
- The Equalizer was ported to the new compiler API and is correctly simulated
- The frontend already supports supplying identifiers for each process (detection of duplicates is not implemented yet)
- Error reporting was improved
  - GHC 6.10 will provide line information to Template Haskell, allowing to make error reports more useful.

Full support of the Synchronous Process Library

- All ForSyDe synchronous process and process constructors are supported in the frontend and the simulation backend.
  - Even polymorphic processes work (big thanks to Oleg Kisleyov for his help at haskell-cafe)
- The Equalizer was ported to the new compiler API and is correctly simulated
- The frontend already supports supplying identifiers for each process (detection of duplicates is not implemented yet)
- Error reporting was improved
  - GHC 6.10 will provide line information to Template Haskell, allowing to make error reports more useful.

Full support of the Synchronous Process Library

- All ForSyDe synchronous process and process constructors are supported in the frontend and the simulation backend.
  - Even polymorphic processes work (big thanks to Oleg Kisleyov for his help at haskell-cafe)
- The Equalizer was ported to the new compiler API and is correctly simulated
- The frontend already supports supplying identifiers for each process (detection of duplicates is not implemented yet)
- Error reporting was improved
  - GHC 6.10 will provide line information to Template Haskell, allowing to make error reports more useful.

Outline

Cabal-ready and Haddock-tagged

- By the time the library was rewritten, all modules were haddock-tagged.
- Thanks to the recent release of Haddock 2.0 (which is embedded in GHC), using Template Haskell is not a problem anymore.
- The package was cabalized. Configuring the package, building it, generating the documentation and installing it is as easy as typing:

```
$ cd ForSyDe; ./Setup.hs configure; ./Setup.hs build; \
              ./Setup.hs haddock; ./Setup.hs install
```

Cabal-ready and Haddock-tagged

- By the time the library was rewritten, all modules were haddock-tagged.
- Thanks to the recent release of Haddock 2.0 (which is embedded in GHC), using Template Haskell is not a problem anymore.
- The package was cabalized. Configuring the package, building it, generating the documentation and installing it is as easy as typing:

```
$ cd ForSyDe; ./Setup.hs configure; ./Setup.hs build; \
              ./Setup.hs haddock; ./Setup.hs install
```

Cabal-ready and Haddock-tagged

- By the time the library was rewritten, all modules were haddock-tagged.
- Thanks to the recent release of Haddock 2.0 (which is embedded in GHC), using Template Haskell is not a problem anymore.
- The package was cabalized. Configuring the package, building it, generating the documentation and installing it is as easy as typing:

```
$ cd ForSyDe; ./Setup.hs configure; ./Setup.hs build; \
              ./Setup.hs haddock; ./Setup.hs install
```

Outline

Improved VHDL backend

- Not ready yet and currently broken.
  - Got the point of supporting components, `Bool`, `Int` signals, and only basic process constructors.
  - Currently working on a common API for translation backends.
- Reasons
  - Underestimated the task cost.
  - Time: stuck in various Template Haskell bugs and working in the frontend and the simulation backend.
  - Design problems still unsolved:
    - What types to support? How? i.e. Vectors, Floating point types.
    - How to support user-defined types? (i.e. typeclass constraints in process constructors could be an option).
    - What Haskell subset should be allowed inside `ProcFuns`?

Improved VHDL backend

- Not ready yet and currently broken.
  - Got the point of supporting components, `Bool`, `Int` signals, and only basic process constructors.
  - Currently working on a common API for translation backends.
- Reasons
  - Underestimated the task cost.
  - Time: stuck in various Template Haskell bugs and working in the frontend and the simulation backend.
  - Design problems still unsolved:
    - What types to support? How? i.e. Vectors, Floating point types.
    - How to support user-defined types? (`Data` typeclass constraints in process constructors could be an option).
    - What Haskell subset should be allowed inside `ProcFuns`?

Improved VHDL backend

- Not ready yet and currently broken.
  - Got the point of supporting components, `Bool`, `Int` signals, and only basic process constructors.
  - Currently working on a common API for translation backends.
- Reasons
  - Underestimated the task cost.
  - Time: stuck in various Template Haskell bugs and working in the frontend and the simulation backend.
  - Design problems still unsolved:
    - What types to support? How? i.e. Vectors, Floating point types.
    - How to support user-defined types? (`Data` typeclass constraints in process constructors could be an option).
    - What Haskell subset should be allowed inside `ProcFuns`?

Improved VHDL backend

- Not ready yet and currently broken.
  - Got the point of supporting components, `Bool`, `Int` signals, and only basic process constructors.
  - Currently working on a common API for translation backends.
- Reasons
  - Underestimated the task cost.
  - Time: stuck in various Template Haskell bugs and working in the frontend and the simulation backend.
  - Design problems still unsolved:
    - What types to support? How? i.e. Vectors, Floating point types.
    - How to support user-defined types? (`Data` typeclass constraints in process constructors could be an option).
    - What Haskell subset should be allowed inside `ProcFun`s?

Improved VHDL backend

- Not ready yet and currently broken.
    - Got the point of supporting components, `Bool`, `Int` signals, and only basic process constructors.
    - Currently working on a common API for translation backends.
- Reasons
    - Underestimated the task cost.
    - Time: stuck in various Template Haskell bugs and working in the frontend and the simulation backend.
    - Design problems still unsolved:
        - What types to support? How? i.e. Vectors, Floating point types.
        - How to support user-defined types? (`Data` typeclass constraints in process constructors could be an option).
        - What Haskell subset should be allowed inside `ProcFun`s?

Improved VHDL backend

- Not ready yet and currently broken.
    - Got the point of supporting components, `Bool`, `Int` signals, and only basic process constructors.
    - Currently working on a common API for translation backends.
- Reasons
    - Underestimated the task cost.
    - Time: stuck in various Template Haskell bugs and working in the frontend and the simulation backend.
    - Design problems still unsolved:
        - What types to support? How? i.e. Vectors, Floating point types.
        - How to support user-defined types? (`Data` typeclass constraints in process constructors could be an option).
        - What Haskell subset should be allowed inside `ProcFuns`?

Improved VHDL backend

- Not ready yet and currently broken.
  - Got the point of supporting components, `Bool`, `Int` signals, and only basic process constructors.
  - Currently working on a common API for translation backends.
- Reasons
  - Underestimated the task cost.
  - Time: stuck in various Template Haskell bugs and working in the frontend and the simulation backend.
  - Design problems still unsolved:
    - What types to support? How? i.e. Vectors, Floating point types.
    - How to support user-defined types? (`Data` typeclass constraints in process constructors could be an option).
    - What Haskell subset should be allowed inside `ProcFun`s?

Improved VHDL backend

- Not ready yet and currently broken.
  - Got the point of supporting components, `Bool`, `Int` signals, and only basic process constructors.
  - Currently working on a common API for translation backends.
- Reasons
  - Underestimated the task cost.
  - Time: stuck in various Template Haskell bugs and working in the frontend and the simulation backend.
  - Design problems still unsolved:
    - What types to support? How? i.e. Vectors, Floating point types.
    - How to support user-defined types? (`Data` typeclass constraints in process constructors could be an option).
    - What Haskell subset should be allowed inside `ProcFun`s?

Improved VHDL backend

- Not ready yet and currently broken.
  - Got the point of supporting components, `Bool`, `Int` signals, and only basic process constructors.
  - Currently working on a common API for translation backends.
- Reasons
  - Underestimated the task cost.
  - Time: stuck in various Template Haskell bugs and working in the frontend and the simulation backend.
  - Design problems still unsolved:
    - What types to support? How? i.e. Vectors, Floating point types.
    - How to support user-defined types? (`Data` typeclass constraints in process constructors could be an option).
    - What Haskell subset should be allowed inside `ProcFun`s?

Improved VHDL backend

- Not ready yet and currently broken.
    - Got the point of supporting components, `Bool`, `Int` signals, and only basic process constructors.
    - Currently working on a common API for translation backends.
- Reasons
    - Underestimated the task cost.
    - Time: stuck in various Template Haskell bugs and working in the frontend and the simulation backend.
    - Design problems still unsolved:
        - What types to support? How? i.e. Vectors, Floating point types.
        - How to support user-defined types? (`Data` typeclass constraints in process constructors could be an option).
        - What Haskell subset should be allowed inside `ProcFun`s?

Outline

Outline

Accessing the external scope within ProcFuns

- Previously, ProcFuns had to be selfcontained.
  - i.e. The translation backends didn't have a way of accesing the external scope of a ProcFun, for instance:

```
$(newProcFun [d| filterer :: (a -> Bool) -> a -> AbstExt a
                 filterer pred val =
                   if pred val then Prst val else Abst  |])
```

- New functions allow to pass parameters without loosing encapsulation:

```
defArgVal :: (Lift a, Typeable a) => ProcFun (a -> b) -> a
          -> ProcFun b
defArgPF :: ProcFun (a -> b) -> ProcFun a -> ProcFun b
```

- Implementation of filterSY

```
filterSY id pred = mapSY id (filterer `defArgPF` pred)
```

  - Not a primitive anymore

Accessing the external scope within ProcFuns

- Previously, ProcFuns had to be selfcontained.
  - i.e. The translation backends didn't have a way of accesing the external scope of a ProcFun, for instance:

```
$(newProcFun [d| filterer :: (a -> Bool) -> a -> AbstExt a
                 filterer pred val =
                   if pred val then Prst val else Abst  |])
```

- New functions allow to pass parameters without loosing encapsulation:

```
defArgVal :: (Lift a, Typeable a) => ProcFun (a -> b) -> a
          -> ProcFun b
defArgPF :: ProcFun (a -> b) -> ProcFun a -> ProcFun b
```

- Implementation of filterSY

```
filterSY id pred = mapSY id (filterer `defArgPF` pred)
```

  - Not a primitive anymore

Accessing the external scope within ProcFuns

- Previously, ProcFuns had to be selfcontained.
  - i.e. The translation backends didn't have a way of accesing the external scope of a ProcFun, for instance:

```
$(newProcFun [d| filterer :: (a -> Bool) -> a -> AbstExt a
                 filterer pred val =
                   if pred val then Prst val else Abst  |])
```

- New functions allow to pass parameters without loosing encapsulation:

```
defArgVal :: (Lift a, Typeable a) => ProcFun (a -> b) -> a
          -> ProcFun b
defArgPF :: ProcFun (a -> b) -> ProcFun a -> ProcFun b
```

- Implementation of filterSY

```
filterSY id pred = mapSY id (filterer `defArgPF` pred)
```

  - Not a primitive anymore

Outline

Lessons learned

1. Template Haskell is in a less mature state than I expected
   - Reported ≈ a dozen bugs/feature requests during this stage.
   - I got stuck finding workarounds for many of them.
   - Fortunately the GHC team is more open and understanding than I can even wish.
   - All the reported problems are likely to be fixed for GHC's next major release (GHC 6.10)
2. E-mail communication should be more fluent when possible
   - Quick feedback is extremely important!

Lessons learned

1. Template Haskell is in a less mature state than I expected
    - Reported $\approx$ a dozen bugs/feature requests during this stage.
    - I got stuck finding workarounds for many of them.
    - Fortunately the GHC team is more open and understanding than I can even wish.
    - All the reported problems are likely to be fixed for GHC's next major release (GHC 6.10)
2. E-mail communication should be more fluent when possible
    - Quick feedback is extremely important!

Lessons learned

1. Template Haskell is in a less mature state than I expected
   - Reported $\approx$ a dozen bugs/feature requests during this stage.
   - I got stuck finding workarounds for many of them.
   - Fortunately the GHC team is more open and understanding than I can even wish.
   - All the reported problems are likely to be fixed for GHC's next major release (GHC 6.10)
2. E-mail communication should be more fluent when possible
   - Quick feedback is extremely important!

Lessons learned

1. Template Haskell is in a less mature state than I expected
   - Reported $\approx$ a dozen bugs/feature requests during this stage.
   - I got stuck finding workarounds for many of them.
   - Fortunately the GHC team is more open and understanding than I can even wish.
   - All the reported problems are likely to be fixed for GHC's next major release (GHC 6.10)
2. E-mail communication should be more fluent when possible
   - Quick feedback is extremely important!

Lessons learned

1. Template Haskell is in a less mature state than I expected
    - Reported ≈ a dozen bugs/feature requests during this stage.
    - I got stuck finding workarounds for many of them.
    - Fortunately the GHC team is more open and understanding than I can even wish.
    - All the reported problems are likely to be fixed for GHC's next major release (GHC 6.10)
2. E-mail communication should be more fluent when possible
    - Quick feedback is extremely important!

Lessons learned

1. Template Haskell is in a less mature state than I expected
   - Reported $\approx$ a dozen bugs/feature requests during this stage.
   - I got stuck finding workarounds for many of them.
   - Fortunately the GHC team is more open and understanding than I can even wish.
   - All the reported problems are likely to be fixed for GHC's next major release (GHC 6.10)
2. E-mail communication should be more fluent when possible
   - Quick feedback is extremely important!

Lessons learned

1. Template Haskell is in a less mature state than I expected
    - Reported ≈ a dozen bugs/feature requests during this stage.
    - I got stuck finding workarounds for many of them.
    - Fortunately the GHC team is more open and understanding than I can even wish.
    - All the reported problems are likely to be fixed for GHC's next major release (GHC 6.10)
2. E-mail communication should be more fluent when possible
    - Quick feedback is extremely important!

Outline

Specific issues to discuss

- Technical issues
  - VHDL backend
    - What primitives types to accept?
    - Custom types?
    - What Haskell subset in `ProcFun`s?
    - Process identifiers. Continue with current approach?
- Bureaucratic issues
  - Release
    - When should the package be released in Hackage?
    - What version number should it carry?
    - Name? ForSyDe vs ForSyDeStdLib
    - License, copyright holder, maintainer e-mail
  - Resources
    - Darcs repository
    - Mailing list archives
    - Maybe use external hosting? (`code.haskell.org`, Sourceforge ...)

Specific issues to discuss

- Technical issues
  - VHDL backend
    - What primitives types to accept?
    - Custom types?
    - What Haskell subset in `ProcFun`s?
    - Process identifiers. Continue with current approach?
- Bureaucratic issues
  - Release
    - When should the package be released in Hackage?
    - What version number should it carry?
    - Name? ForSyDe vs ForSyDeStdLib
    - License, copyright holder, maintainer e-mail
  - Resources
    - Darcs repository
    - Mailing list archives
    - Maybe use external hosting? (`code.haskell.org`, Sourceforge ...)

Specific issues to discuss

- Technical issues
  - VHDL backend
    - What primitives types to accept?
    - Custom types?
    - What Haskell subset in `ProcFun`s?
    - Process identifiers. Continue with current approach?
- Bureaucratic issues
  - Release
    - When should the package be released in Hackage?
    - What version number should it carry?
    - Name? ForSyDe vs ForSyDeStdLib
    - License, copyright holder, maintainer e-mail
  - Resources
    - Darcs repository
    - Mailing list archives
    - Maybe use external hosting? (`code.haskell.org`, Sourceforge ...)

Specific issues to discuss

- Technical issues
  - VHDL backend
    - What primitives types to accept?
    - Custom types?
    - What Haskell subset in `ProcFun`s?
    - Process identifiers. Continue with current approach?
- Bureaucratic issues
  - Release
    - When should the package be released in Hackage?
    - What version number should it carry?
    - Name? ForSyDe vs ForSyDeStdLib
    - License, copyright holder, maintainer e-mail
  - Resources
    - Darcs repository
    - Mailing list archives
    - Maybe use external hosting? (code.haskell.org, Sourceforge ...)

Specific issues to discuss

- Technical issues
  - VHDL backend
    - What primitives types to accept?
    - Custom types?
    - What Haskell subset in `ProcFun`s?
    - Process identifiers. Continue with current approach?
- Bureaucratic issues
  - Release
    - When should the package be released in Hackage?
    - What version number should it carry?
    - Name? ForSyDe vs ForSyDeStdLib
    - License, copyright holder, maintainer e-mail
  - Resources
    - Darcs repository
    - Mailing list archives
    - Maybe use external hosting? (`code.haskell.org`, Sourceforge ...)

Specific issues to discuss

- Technical issues
  - VHDL backend
    - What primitives types to accept?
    - Custom types?
    - What Haskell subset in `ProcFun`s?
    - Process identifiers. Continue with current approach?
- Bureaucratic issues
  - Release
    - When should the package be released in Hackage?
    - What version number should it carry?
    - Name? ForSyDe vs ForSyDeStdLib
    - License, copyright holder, maintainer e-mail
  - Resources
    - Darcs repository
    - Mailing list archives
    - Maybe use external hosting? (code.haskell.org, Sourceforge ...)

Specific issues to discuss

- Technical issues
  - VHDL backend
    - What primitives types to accept?
    - Custom types?
    - What Haskell subset in `ProcFun`s?
    - Process identifiers. Continue with current approach?
- Bureaucratic issues
  - Release
    - When should the package be released in Hackage?
    - What version number should it carry?
    - Name? ForSyDe vs ForSyDeStdLib
    - License, copyright holder, maintainer e-mail ...
  - Resources
    - Darcs repository
    - Mailing list archives
    - Maybe use external hosting? (code.haskell.org, Sourceforge ...)

Specific issues to discuss

- Technical issues
  - VHDL backend
    - What primitives types to accept?
    - Custom types?
    - What Haskell subset in `ProcFun`s?
    - Process identifiers. Continue with current approach?
- Bureaucratic issues
  - Release
    - When should the package be released in Hackage?
    - What version number should it carry?
    - Name? ForSyDe vs ForSyDeStdLib
    - License, copyright holder, maintainer e-mail.
  - Resources
    - Darcs repository
    - Mailing list archives
    - Maybe use external hosting? (`code.haskell.org`, Sourceforge ...)

Specific issues to discuss

- Technical issues
  - VHDL backend
    - What primitives types to accept?
    - Custom types?
    - What Haskell subset in `ProcFun`s?
    - Process identifiers. Continue with current approach?
- Bureaucratic issues
  - Release
    - When should the package be released in Hackage?
    - What version number should it carry?
    - Name? ForSyDe vs ForSyDeStdLib
    - License, copyright holder, maintainer e-mail.
  - Resources
    - Darcs repository
    - Mailing list archives
    - Maybe use external hosting? (`code.haskell.org`, Sourceforge ...)

Specific issues to discuss

- Technical issues
  - VHDL backend
    - What primitives types to accept?
    - Custom types?
    - What Haskell subset in `ProcFun`s?
    - Process identifiers. Continue with current approach?
- Bureaucratic issues
  - Release
    - When should the package be released in Hackage?
    - What version number should it carry?
    - Name? ForSyDe vs ForSyDeStdLib
    - License, copyright holder, maintainer e-mail.
  - Resources
    - Darcs repository
    - Mailing list archives
    - Maybe use external hosting? (`code.haskell.org`, Sourceforge ...)

Specific issues to discuss

- Technical issues
    - VHDL backend
        - What primitives types to accept?
        - Custom types?
        - What Haskell subset in `ProcFun`s?
        - Process identifiers. Continue with current approach?
- Bureaucratic issues
    - Release
        - When should the package be released in Hackage?
        - What version number should it carry?
        - Name? ForSyDe vs ForSyDeStdLib
        - License, copyright holder, maintainer e-mail.
    - Resources
        - Darcs repository
        - Mailing list archives
        - Maybe use external hosting? (`code.haskell.org`, Sourceforge ...)

Specific issues to discuss

- Technical issues
  - VHDL backend
    - What primitives types to accept?
    - Custom types?
    - What Haskell subset in `ProcFun`s?
    - Process identifiers. Continue with current approach?
- Bureaucratic issues
  - Release
    - When should the package be released in Hackage?
    - What version number should it carry?
    - Name? ForSyDe vs ForSyDeStdLib
    - License, copyright holder, maintainer e-mail.
  - Resources
    - Darcs repository
    - Mailing list archives
    - Maybe use external hosting? (`code.haskell.org`, Sourceforge ...)

Specific issues to discuss

- Technical issues
  - VHDL backend
    - What primitives types to accept?
    - Custom types?
    - What Haskell subset in `ProcFun`s?
    - Process identifiers. Continue with current approach?
- Bureaucratic issues
  - Release
    - When should the package be released in Hackage?
    - What version number should it carry?
    - Name? ForSyDe vs ForSyDeStdLib
    - License, copyright holder, maintainer e-mail.
  - Resources
    - Darcs repository
    - Mailing list archives
    - Maybe use external hosting? (`code.haskell.org`, Sourceforge ...)

Specific issues to discuss

- Technical issues
  - VHDL backend
    - What primitives types to accept?
    - Custom types?
    - What Haskell subset in `ProcFun`s?
    - Process identifiers. Continue with current approach?
- Bureaucratic issues
  - Release
    - When should the package be released in Hackage?
    - What version number should it carry?
    - Name? ForSyDe vs ForSyDeStdLib
    - License, copyright holder, maintainer e-mail.
  - Resources
    - Darcs repository
    - Mailing list archives
    - Maybe use external hosting? (`code.haskell.org`, Sourceforge ...)

Specific issues to discuss

- Technical issues
  - VHDL backend
    - What primitives types to accept?
    - Custom types?
    - What Haskell subset in `ProcFun`s?
    - Process identifiers. Continue with current approach?
- Bureaucratic issues
  - Release
    - When should the package be released in Hackage?
    - What version number should it carry?
    - Name? ForSyDe vs ForSyDeStdLib
    - License, copyright holder, maintainer e-mail.
  - Resources
    - Darcs repository
    - Mailing list archives
    - Maybe use external hosting? (`code.haskell.org`, Sourceforge ...)

Specific issues to discuss

- Technical issues
    - VHDL backend
        - What primitives types to accept?
        - Custom types?
        - What Haskell subset in `ProcFun`s?
        - Process identifiers. Continue with current approach?
- Bureaucratic issues
    - Release
        - When should the package be released in Hackage?
        - What version number should it carry?
        - Name? ForSyDe vs ForSyDeStdLib
        - License, copyright holder, maintainer e-mail.
    - Resources
        - Darcs repository
        - Mailing list archives
        - Maybe use external hosting? (`code.haskell.org`, Sourceforge ...)

Specific issues to discuss

- Technical issues
  - VHDL backend
    - What primitives types to accept?
    - Custom types?
    - What Haskell subset in `ProcFun`s?
    - Process identifiers. Continue with current approach?
- Bureaucratic issues
  - Release
    - When should the package be released in Hackage?
    - What version number should it carry?
    - Name? ForSyDe vs ForSyDeStdLib
    - License, copyright holder, maintainer e-mail.
  - Resources
    - Darcs repository
    - Mailing list archives
    - Maybe use external hosting? (`code.haskell.org`, Sourceforge ...)

Outline

What's next?

- What should be done now?
- Personal proposal
  - Finish the VHDL backend
  - Move on to the new library
  - Continue adding features
- What features? With what precedence? Options:
  - Graphical backend.
  - SystemC backend.
  - Verification backend (SMV).
  - Graphical frontend.
  - Transformational refinement.

What's next?

- What should be done now?
- Personal proposal
  - Finish the VHDL backend
  - Move on to the new library
  - Continue adding features
- What features? With what precedence? Options:
  - Graphical backend.
  - SystemC backend.
  - Verification backend (SMV).
  - Graphical frontend.
  - Transformational refinement.

What’s next?

- What should be done now?
- Personal proposal
  - Finish the VHDL backend
  - Move on to the new library
  - Continue adding features
- What features? With what precedence? Options:
  - Graphical backend.
  - SystemC backend.
  - Verification backend (SMV).
  - Graphical frontend.
  - Transformational refinement.

What's next?

- What should be done now?
- Personal proposal
  - Finish the VHDL backend
  - Move on to the new library
  - Continue adding features
- What features? With what precedence? Options:
  - Graphical backend.
  - SystemC backend.
  - Verification backend (SMV).
  - Graphical frontend.
  - Transformational refinement.

What's next?

- What should be done now?
- Personal proposal
  - Finish the VHDL backend
  - Move on to the new library
  - Continue adding features
- What features? With what precedence? Options:
  - Graphical backend.
  - SystemC backend.
  - Verification backend (SMV).
  - Graphical frontend.
  - Transformational refinement.

What's next?

- What should be done now?
- Personal proposal
  - Finish the VHDL backend
  - Move on to the new library
  - Continue adding features
- What features? With what precedence? Options:
  - Graphical backend.
  - SystemC backend.
  - Verification backend (SMV).
  - Graphical frontend.
  - Transformational refinement.

What’s next?

- What should be done now?
- Personal proposal
  - Finish the VHDL backend
  - Move on to the new library
  - Continue adding features
- What features? With what precedence? Options:
  - Graphical backend.
  - SystemC backend.
  - Verification backend (SMV).
  - Graphical frontend.
  - Transformational refinement.

What's next?

- What should be done now?
- Personal proposal
  - Finish the VHDL backend
  - Move on to the new library
  - Continue adding features
- What features? With what precedence? Options:
  - Graphical backend.
  - SystemC backend.
  - Verification backend (SMV).
  - Graphical frontend.
  - Transformational refinement.

What’s next?

- What should be done now?
- Personal proposal
  - Finish the VHDL backend
  - Move on to the new library
  - Continue adding features
- What features? With what precedence? Options:
  - Graphical backend.
  - SystemC backend.
  - Verification backend (SMV).
  - Graphical frontend.
  - Transformational refinement.

What’s next?

- What should be done now?
- Personal proposal
  - Finish the VHDL backend
  - Move on to the new library
  - Continue adding features
- What features? With what precedence? Options:
  - Graphical backend.
  - SystemC backend.
  - Verification backend (SMV).
  - Graphical frontend.
  - Transformational refinement.

What’s next?

- What should be done now?
- Personal proposal
  - Finish the VHDL backend
  - Move on to the new library
  - Continue adding features
- What features? With what precedence? Options:
  - Graphical backend.
  - SystemC backend.
  - Verification backend (SMV).
  - Graphical frontend.
  - Transformational refinement.