LegendOfBluespec

kr469

October 2021

1 Core loop of Bluespec

In Bluespec all computation is done in form of rules. Each cycle we will take a subset of all rules that we are going to execute in this cycle, rule is fired (executed) in cycle only if it's ready (or will be ready) and it's not conflicting with other rules (If this happens compiler must issue a warning, and picks arbitrary rule to fire from subset of conflicting rules). Each rule can fire at most one time per cycle. For rule to be ready to fire it needs it's implicit and explicit conditions to be true. Rule can be fired in a cycle even if it's not ready at the start of cycle, for example if you add item on an empty queue and then pop can happen in same cycle.

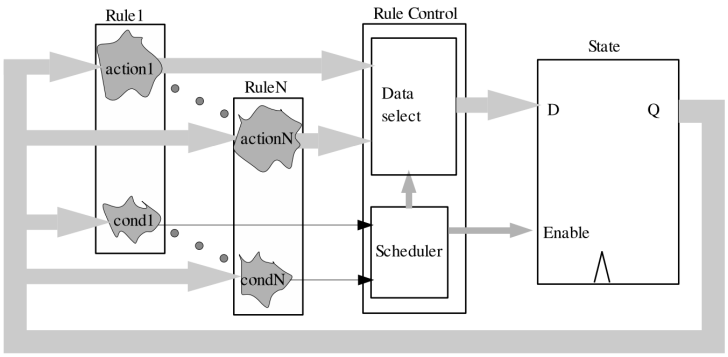

Figure 1: A general scheme for mapping an N-rule system into clocked synchronous hardware.

```
TODO: piece of Bluespec with module using fifo and other rule
 explaining types of conditions.
```

2 Types

In Bluespec we can enounter types in 4 categories:

2.1 Bit Types

This is most common category and contains evey type that is ment to represent data like Int, Bit , Tuples. They are synthesiazable and can store data, during a cycle.

2.2 Non-Bit types

Those are things like Integer, Real , String that are ment to be used for debug or polymorphism. Most important difference is that you can't syntheside those types so it needs to be possible to know theris values at elaboration.

2.3 Interface types

This set of types are ment to represent implementation independed sets of functions exposed by a module, or subinterfaces. Here we will find stuff like FIFO intersce with Actions push and pull, or something like Reg (which is interface of an register).

2.4 Compiler types

Those are things like Action, ActionValue,Rules which are more of a keywords rather than types, and they are used to distinguish functions that are Rules(they can't be called), from Actions which can be called and ActionValues which are functions that return Values.

3 Interfaces

In Bluespec, all modules need to have interfaces. (If it's a top-level module, it can have interface *Empty*). Those interfaces provide a framework for communication between modules. They are like structs that contain only methods or sub-structs. Interfaces can be polymorphic, and we will use this fact to create interfaces that allow for translation between interfaces. A common example would be polymorphic FIFO that can store any type. What's important is that this polymorphism makes it unsyntesiable, and its exact type needs to be resolved before a module with such type can be synthesized.
TODO : I might add grammar for defining an interface from reference guide.

4 Typeclasses

Typesclasses in Bluespec are used to group types that contain specific set of functions, like all Bit types are convertable to bit vectors or Arith on which arithmetics are defined.

5 Provisios

In Bluespec there is notion of Provisios they are used to restrict what types can be used in given place. list places where provisos can appear in Example use or proviso would be to ensure given type is of certain typeclass so arthmetic operations can be used with it or that it's length in bits is within certain range.

6 Bluetcl

Here I will explain what can be extracted from Bluetcl. For me most important will be extracting interfaces and functions for module creation. To do this I have written down grammar (it's currently in type.lark but I will paste it at some point here).

7 GetPut and other libraries

GetPut is library used to make it easier to connect modules, but because it is written using normal type system it therore it should be possibe to generalize and handle them via main algorithm without special exception.

8 System Verilog

Using respective compiler flags, one can compile Bluespec to Verilog, and one can add Verilog inserts into Bluespec. Because of this, I will focus only on support for Bluespec as there is available tooling to make my tools work in Verilog projects with Bluespec inserts.

9 Toolchain

1. User either starts with .bsc files that needs to be complied to .bo pacages using Bsc.
2. User feeds .bo/.ba pacages to python libary.
3. Library uses commands in Bluetcl to extract functions for creating modules, and interfaces of those modules. This information is then cashed.
4. If GUI is available, then GUI is fed data about known modules and exposes possible modules to the user. Then user creates JSON configuration using this GUI, what might happen underneath is following with steps bellow to immidetly give user feedback about stuff like correctness of values according to the provisos which will be checked only during compliation as they are not exposed in pacages.
5. User then feeds library with JSON topmodule configuration file.

6. Using this file .bsv file is produced containing code of this toplevel module.
7. This then can be compled using bsv complier.

Write section about how GUI will also contain tools for debuging and interactivity

Tool chain / Pipeline of the library.

1. Pacages are found using pythons os library.
2. For each pacage we find its dependencies
3. check if loading packages to bluetcl needs to be done in order
4. For each pacage library performs exploration procedure to find all types and functions.
5. All outputs of bluetcl are red using lark parser.
6. Then my algorithm will take care of parsing data from those parse trees.
7. For each interface , and module I will try to create coresponding class in python.
8. If this is done correctly then I will be able to ensure typing correctness via pythons typing system.
9. JSON will be read using some python library.
10. After all information is accquired I will just instnatiate python object representing toplevel module.
11. This object will then have function that converts it into Bluespec code via some tree exploration algorithm.
12. Such code will need fluff that added at this stage.
13. Code is then save to file.
14. Library then calls complier to make sure produced code complies.
15. In case of errors, due to violating provisos library reports them to user.
16. I'm considering currently reporting some of those errors also directly in JSON so it's easy for users to find places that need modifcation.