Plugin Development for Firrtl

Jiuyang Liu
SiFive China & Huazhong University of Science and Technology

Liu Jiuyang (Sequencer)
PhD candidates from HUST, Wuhan Hubei
SiFive China, Shanghai

GitHub: https://github.com/sequencer
Blog: https://jiuyang.me

Why firrtl?

Why firrtl?

- firrtl abstraction layer is the main difference between chisel2 and chisel3
- So just split chisel2 into chisel3 as front-end while firrtl as back-end?
  - SpinalHDL forked chisel2, adding the AST manipulation in the elaborating time
  - later chisel3 did it too
- AST transform framework is the reason

Why firrtl?

- AST Transform
  - firrtl provides the LLVM-like framework
  - Some elaboration part from chisel2
  - Target multi-backend like: ASIC/FPGA/Simulation
  - Circuit Optimization
  - Custom Transform with pattern match
- Phase/Stage Based Annotation Framework
  - Circuit analysis(diagrammer)
  - Place&Route plugin

Why firrtl?

- Is firrtl really important for the end user?
  - Or you want to achieve extreme PPA for your ASIC design
    - Special design follow like full custom design
  - Or chisel cannot achieve what you really want to describe
    - Generate specific design after some transform
  - Unless you wanna do some subtle change to your circuit
    - Find the source and sink of registers
    - Add print statement for debug to AST pattern

Why firrtl?

- There are 3 abstraction layer for common rocket-chip users:
  - Diplomacy
    - SoC
    - DAG
  - Chisel
    - chiselFrontend for circuit elaboration
  - Firrtl

- If you think chisel standard Verilog emitter is enough for your common usage, and you wanna try some advanced chisel, you may try diplomacy instead.

Glance to firrtl

Glance to firrtl

- Annotation
  - Store all metadata,
    - From Circuit to Logger
      - FirrtlCircuitAnnoation
      - LoggerAnnoation
    - Can directly transformed by Phase
    - Checks, Elaborate, AddImplicitOutputFile, AddImplicitOutputAnnotationFile, Emitter, Convert, MaybeFirrtlStage
    - AddDefaults, AddImplicitEmitter, Checks, AddCircuit, AddImplicitOutputFile, Compiler, WriteEmitted

Glance to firrtl

- Circuit
  - Circuit is stored at FirrtlCircuitAnnotaion
  - Extracted in Compiler for IR Transform
- IR
  - Info
  - PrimOp
  - Statement
  - Expression
  - DefModule
  - Circuit
  - Width, Orientation, Direction, Param, Type, Field

Glance to firrtl

- Target
  - Target is a pointer to a Statement
  - Transform may alter the name of Statement
  - CompleteTarget is common used by Transform developing
- TransformLike
  - Phase
  - Transform
- Visitor
- Emitter

The plug-in development

The plug-in development

- In general, firrtl contains two types of plugin
  - Transform-based Circuit manipulation
  - Phase-based metadata plugin
- Annotate clock edge information for register
  - THIS IS NOT AN MERGED MODIFICATION TO FIRRTL AND CHISEL
    - It may not be accepted, or merged in current version
    - This is a demo tutorial from Visitor to Emitter

The plug-in development

```
+ sealed abstract class Edge extends FirrtlNode
+ case object Posedge extends Edge {
+   def serialize: String = "posedge"
+ }
+ case object Negedge extends Edge {
+   def serialize: String = "negedge"
+ }
  case class DefRegister(info: Info,
      name: String,
      tpe: Type,
+     edge: Edge,
      clock: Expression,
      reset: Expression,
      init: Expression
) extends Statement with IsDeclaration
```

scala/firrtl/ir/IR.scala

proto/firrtl.proto

```
+ enum Edge {
+    REGISTER_EDGE_POSEDGE = 0;
+    REGISTER_EDGE_NEGEDGE = 1;
+  }
message Register {
      // Required.
      string id = 1;
      // Required.
      Type type = 2;
      // Required.
      Expression clock = 3;
      Expression reset = 4;
      Expression init = 5;
+     Edge edge = 6;
    }
```

antlr4/FIRRTL.g4

```
stmt
    : 'wire' id ':' type info?
-   | 'reg' id ':' type exp ('with' ':' reset_block)? info?
+   // use edge? for back compatibility
+   | 'reg' id ':' type edge? exp ('with' ':' reset_block)? info?
```

```
+ private def visitEdge[FirrtlNode](ctx: Option[EdgeContext]): Edge =
+     ctx match {
+       case Some(edge) => edge.getText match {
+         case "posedge" => Posedge
+         case "negedge" => Negedge
+       }
+       case None => Posedge
+     }
```

scala/firrtl/Visitor.scala

sequencer/firrtl/clockedge

The plug-in development

```
case class ClockEdgeAnnotation(register: ReferenceTarget, edge: Edge) extends
  SingleTargetAnnotation[ReferenceTarget] {
    override val target: ReferenceTarget = register

    override def duplicate(n: ReferenceTarget): Annotation = ClockEdgeAnnotation(register, edge: Edge)
}
```

```
object annoEdge {
  def apply(reg: Data, edge: Edge)(implicit compileOptions: CompileOptions): Unit = {
    // TODO: wait freechipsproject/chisel3#1120 to be merged
    annotate(new ChiselAnnotation {
      def toFirrtl = ClockEdgeAnnotation(reg.toNamed.toTarget, edge)
    })
  }
}
```

```
class DummyClockEdgeModule extends MultiIOModule {
  val in: Bool = IO(Input(Bool()))
  val out: Bool = IO(Output(Bool()))
  val dummyReg = RegNext(in)
  annoEdge(dummyReg, Negedge)
  out := dummyReg
}
```

The plug-in development

```
class CompleteClockEdge extends Transform {
  override def inputForm: CircuitForm = MidForm

  override def outputForm: CircuitForm = MidForm

  def addEdge(annoMap: Map[String, Edge])(s: Statement): Statement = s.mapStmt {
    case r: DefRegister => {
      annoMap.get(r.name) match {
        // need freechipsproject/firrtl#1125
        case Some(e) => r.copy(edge = e)
        case None => r
      }
    }
    case s: Statement => s.mapStmt(addEdge(annoMap))
  }

  override def execute(state: CircuitState): CircuitState = state.copy(circuit =
    state.circuit.copy(modules =
      state.circuit.modules.map(
        _.mapStmt(addEdge(state.annotations.collect {case ClockEdgeAnnotation(t, e) => t.name -> e}.toMap)
  ))))
}
```

Firrtl in the future

Firrtl in the future

- Processing
  - AsyncReset support (firrtl#1011)
  - Dependency API(firrtl#1123)
  - Analysis Transform (firrtl#937, firrtl#1132)
  - Attributes for Verilog(firrtl#1040)
  - Mill support for firrtl and chisel3(firrtl#1083)
- Feature Request
  - Full expression ability
  - Formal Support by Euclid or Kami
  - Verilog UDP support

SiFive

Thank you!