Chisel Recipes: An eDSL for Imperative Control Flow Machines

Vighnesh Iyer, Bryan Ngo, Bora Nikolic
UC Berkeley

Motivation

Conversion of imperative control flow to its FSM encoding is a common, mechanical, and error-prone process in RTL design

```
io.ready = true
waitUntil(io.valid)
data = io.bits
while(popcount(data) >= 3) {
    data = data >> 1
}
```

Why Not HLS?

High-level synthesis can automatically convert a subset of C to RTL, including imperative control flow. *But:*

- HLS-ed code is designed to work as a standalone circuit rather than being closely integrated with hand-written RTL
- HLS does not give fine control over cycle-level timing
- HLS frontends are often in a different language than the RTL HDL

Our Proposal

chisel-recipes is an embedded DSL for describing and compiling control flow machines at cycle granularity

```
val data = Reg( ... )
recipe (
  action { io.ready := true.B },
  waitUntil(io.valid),
  action { data := io.bits },
  tick(),
  whileLoop(popcount(data) >= 3.U) {
      action { data := data >> 1.U }
  }
).compile()
```

chisel-recipes Primitives

The control flow primitives are modeled after those in Blarney (an eDSL embedded in Haskell)

- `Action`: Combinational immediate assignments composed of arbitrary Chisel code
- `Tick`: Advance one cycle
- `Recipe`: Execute sub-recipes in sequential order
- `While`: Execute the body until the condition is false
- `When` / `ITE`: Conditionally execute a sub-recipe
- `Parallel*` / `Background*`: Concurrently execute sub-recipes

```
enum RecipePrimitive:
  case Tick()
  case Action(() => Bool)
  case Recipe(Seq[Recipe])
  case While(Bool, Recipe)
  case When(Bool, Recipe)
  case IfThenElse(Bool, Recipe, Recipe)
  case Parallel(Seq[Recipe])
  case Background(Seq[Recipe])
```

Recipe Combinators

Since recipe primitives are just Scala datatypes, regular Scala functions can be used to construct new recipes.

```
def waitUntil(cond: chisel3.Bool): Recipe =
    whileLoop(cond, tick())
def forever(r: Recipe): Recipe =
    whileLoop(true.B, r)
```

Recipes can be defined and composed just like Chisel circuits.

```
def fetchDecoupled[T](io: DecoupledIO[T], dataReg: T) =
  recipe (
    waitUntil(io.valid, active=io.ready),
    action { data := io.bits },
    tick()
  )
```

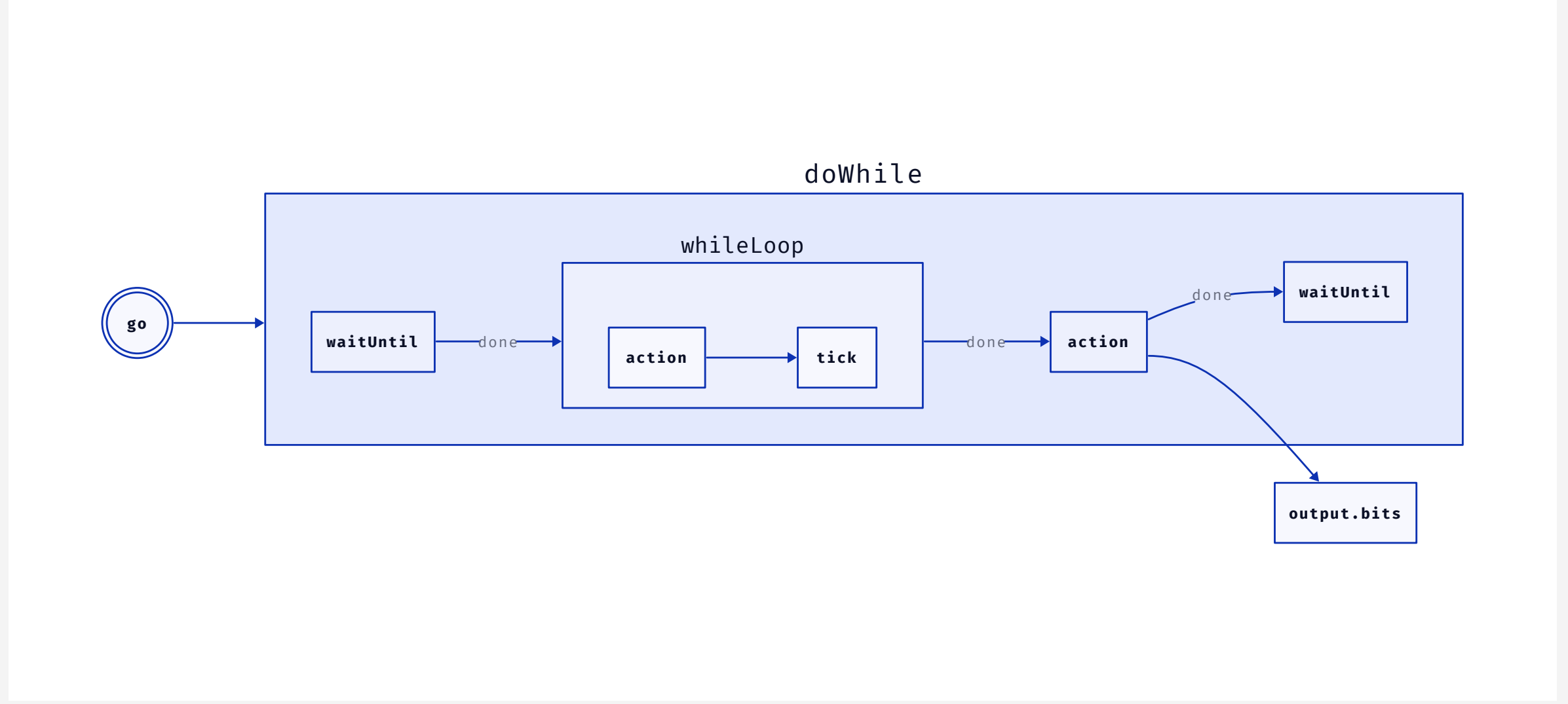
Recipes have an `active` signal, which can be used in the RTL.

Example - AXI4-Lite Slave

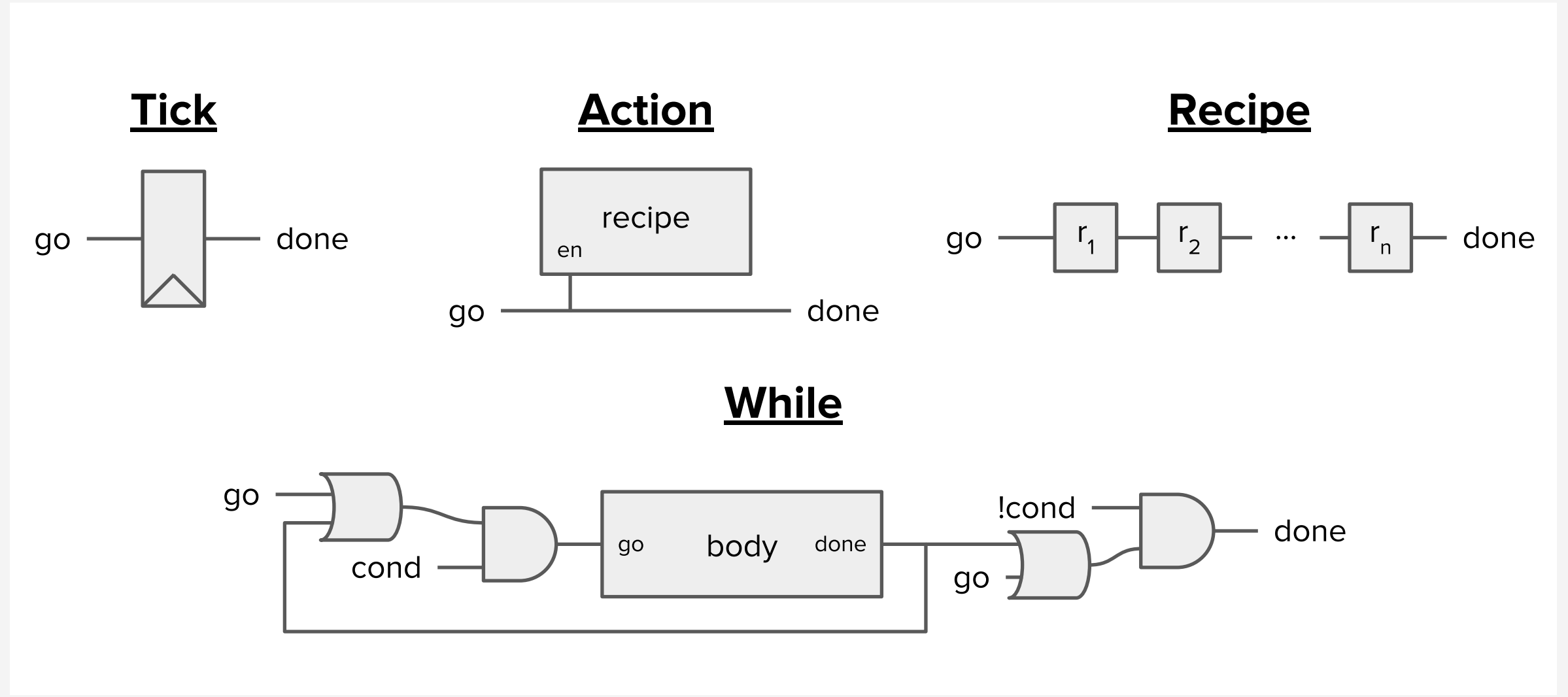
```
val readOnce = recipe (
  waitUntil(axi.ar_valid === 1.B, active=axi.ar_ready),
  action {
    axi.r_data := mem.read(axi.ar_addr)
    axi.r_valid := 1.B
  },
  waitUntil(axi.r_ready === 1.B, active=axi.r_valid),
  tick()
)
forever(readOnce).compile()
```

Example - Decoupled GCD Circuit

```
doWhile (
  waitUntil(input.valid, active=input.ready),
  action {
    x := input.bits.value1
    y := input.bits.value2
  },
  tick(),
  whileLoop(x > 0.U && y > 0.U)(
    action {
      when(x > y) {
        x := x - y
      }.otherwise {
        y := y - x
      }
    },
    tick()
  ),
  action {
    when(x === 0.U) {
      output.bits.gcd := y
    }.otherwise {
      output.bits.gcd := x
    }
  },
  waitUntil(output.ready, active=output.valid),
)(cond=true.B).compile()
```



Compiling Recipes



1. Combinators are expanded into primitives
2. Primitives become circuits with *go* and *done* signals
   - `go` and `done` are single-cycle wide pulses
   - The active signal is also elaborated if used
3. Chisel compiles the composed recipe circuit to RTL

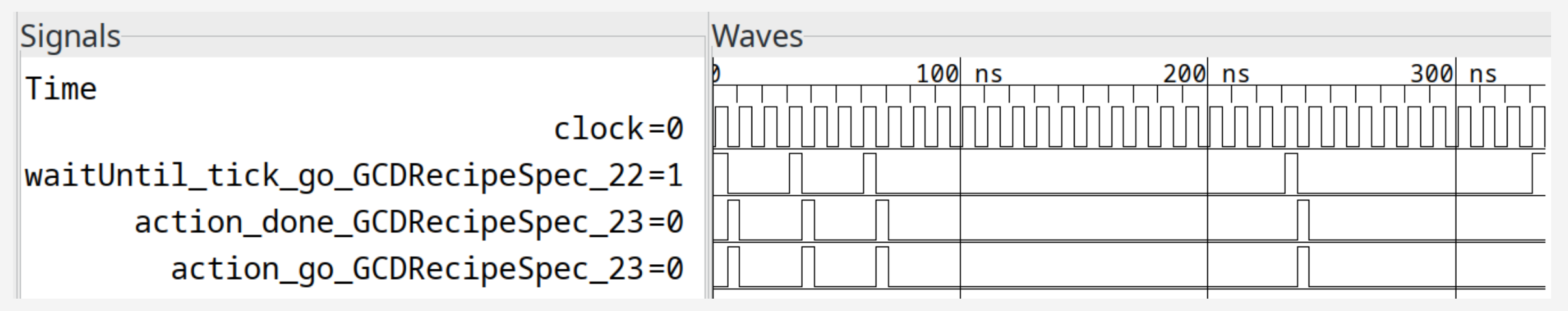
Debugging Recipes - Printf Instrumentation

When compiling a recipe, it can be instrumented with Chisel `printf`s with source locators.

```
time=[0] [doWhile] has started (GCDRecipe.scala:1)
time=[0] [waitUntil] has started (GCDRecipe.scala:2)
time=[0] [waitUntil] has finished (GCDRecipe.scala:2)
time=[0] [action] is active (GCDRecipe.scala:3)
time=[0] [tick] about to tick (GCDRecipe.scala:7)

time=[1] [whileLoop] has started (GCDRecipe.scala:8)
time=[1] [action] is active (GCDRecipe.scala:9)
time=[1] [tick] about to tick (GCDRecipe.scala:16)
```

Debugging Recipes - Waveform Instrumentation


Signals
Waves
Time
clock=0
waitUntil_tick_go_GCDRecipeSpec_22=1
action_done_GCDRecipeSpec_23=0
action_go_GCDRecipeSpec_23=0
100 ns
200 ns
300 ns

All recipe primitives and combinators have named go/active/done signals with source locators, injected in the RTL.

Conclusion and Future Work

- New eDSLs leveraging existing embedded HDLs (Chisel) are powerful and useful
- Chisel recipes is a new, debug-friendly, eDSL to help RTL designers describe control flow machines
- *Future work*: optimized recipes with static analysis, PPA tradeoff analysis, bounded fork/join functionality, demonstrate applications in accelerator/peripheral/cache design