

# Bitonic Sort in ScalaHDL

Yao Li

# Outline

- New structure of ScalaHDL.
- Bitonic sort in ScalaHDL.
- Future plan.

# New Structure of ScalaHDL: FlipFlop Example

```
object Main extends ScalaHDL {  
  defMod.FlipFlop('d, 'q, 'clk) {  
    sync('clk is 1).logic {  
      'q := 'd  
    }  
  }  
  
  def main(args: Array[String]) {  
    val q = bool(0)  
    val d = bool(1)  
    val clk = bool(0)  
    println(convert('FlipFlop, d, q, clk))  
  }  
}
```

# New Structure of ScalaHDL: FlipFlop Example

```
object Main extends ScalaHDL {  
  defMod.FlipFlop('d, 'q, 'clk) {  
    sync('clk is 1).logic {  
      'q := 'd  
    }  
  }  
}
```

→

```
sync(...).logic {...}  
  instead of  
  sync(...)  
defMod.logic(...) {...}
```

```
def main(args: Array[String]) {  
  val q = bool(0)  
  val d = bool(1)  
  val clk = bool(0)  
  println(convert('FlipFlop, d, q, clk))  
}
```

# Bitonic Sort Program

```
object Main extends ScalaHDL {
  val ASC = 0
  val DES = 1

  defMod.sort8('a0, 'a1, 'a2, 'a3, 'a4, 'a5, 'a6, 'a7,
    'z0, 'z1, 'z2, 'z3, 'z4, 'z5, 'z6, 'z7) {
    val a = List('a0, 'a1, 'a2, 'a3, 'a4, 'a5, 'a6,
'a7).map(toHDLType)
    val z = List('z0, 'z1, 'z2, 'z3, 'z4, 'z5, 'z6,
'z7).map(toHDLType)

    def compare(a: HDLType, b: HDLType,
      x: HDLType, y: HDLType, dir: Int) {
      ...
    }

    def bitonicMerge(a: Seq[HDLType], z: Seq[HDLType],
      dir: Int) {
      ...
    }

    def bitonicSort(a: Seq[HDLType], z: Seq[HDLType],
      dir: Int) {
      ...
    }

    bitonicSort(a, z, ASC)
  }

  def main(args: Array[String]) {
```

```
    val a0 = bool(0)
    val a1 = bool(0)
    val a2 = bool(0)
    val a3 = bool(0)
    val a4 = bool(0)
    val a5 = bool(0)
    val a6 = bool(0)
    val a7 = bool(0)

    val z0 = bool(0)
    val z1 = bool(0)
    val z2 = bool(0)
    val z3 = bool(0)
    val z4 = bool(0)
    val z5 = bool(0)
    val z6 = bool(0)
    val z7 = bool(0)

    println(convert('sort8, a0, a1, a2, a3, a4, a5, a6,
a7,
      z0, z1, z2, z3, z4, z5, z6, z7))
  }
}
```

# Bitonic Sort Program

```
object Main extends ScalaHDL {  
  val ASC = 0  
  val DES = 1
```

defMod with '

```
  defMod.sort8('a0, 'a1, 'a2, 'a3, 'a4, 'a5, 'a6, 'a7,  
    'z0, 'z1, 'z2, 'z3, 'z4, 'z5, 'z6, 'z7) {  
    val a = List('a0, 'a1, 'a2, 'a3, 'a4, 'a5, 'a6,  
      'a7).map(toHDLType)  
    val z = List('z0, 'z1, 'z2, 'z3, 'z4, 'z5, 'z6,  
      'z7).map(toHDLType)
```

```
    val a0 = bool(0)  
    val a1 = bool(0)  
    val a2 = bool(0)  
    val a3 = bool(0)  
    val a4 = bool(0)  
    val a5 = bool(0)  
    val a6 = bool(0)
```

```
  def compare(a: HDLType, b: HDLType,  
    x: HDLType, y: HDLType, dir: Int) {  
    ...  
  }
```

normal Scala  
function definition

```
  def bitonicMerge(a: Seq[HDLType], z: Seq[HDLType],  
    dir: Int) {  
    ...  
  }
```

```
    val z3 = bool(0)  
    val z4 = bool(0)  
    val z5 = bool(0)  
    val z6 = bool(0)  
    val z7 = bool(0)
```

```
  def bitonicSort(a: Seq[HDLType], z: Seq[HDLType],  
    dir: Int) {  
    ...  
  }
```

```
    println(convert('sort8, a0, a1, a2, a3, a4, a5, a6,  
      a7,  
      z0, z1, z2, z3, z4, z5, z6, z7))
```

```
    bitonicSort(a, z, ASC)
```

normal Scala  
function call


```
  def main(args: Array[String]) {
```

# Bitonic Sort Program

```
def compare(a: HDLType, b: HDLType, x: HDLType, y: HDLType, dir: Int) {  
  if (dir == ASC) {  
    when (a > b) {  
      x := b  
      y := a  
    } .otherwise {  
      x := a  
      y := b  
    }  
  } else {  
    when (a > b) {  
      x := a  
      y := b  
    } .otherwise {  
      x := b  
      y := a  
    }  
  }  
}
```

# Bitonic Sort Program

```
def compare(a: HDLType, b: HDLType, x: HDLType, y: HDLType, dir: Int) {  
  if (dir == ASC) {  
    when (a > b) {  
      x := b  
      y := a  
    } .otherwise {  
      x := a  
      y := b  
    }  
  } else {  
    when (a > b) {  
      x := a  
      y := b  
    } .otherwise {  
      x := b  
      y := a  
    }  
  }  
}
```



**Good news:** no ' here!

**Bad news:** you have to specify type for arguments.

**Another good news:**  
HDLType instead of Bool,  
Signed, Unsigned, etc.



# Bitonic Sort Program

```
def compare(a: HDLType, b: HDLType, dir: Int) {  
  if (dir == ASC) {  
    when (a > b) {  
      x := b  
      y := a  
    } .otherwise {  
      x := a  
      y := b  
    }  
  } else {  
    when (a > b) {  
      x := a  
      y := b  
    } .otherwise {  
      x := b  
      y := a  
    }  
  }  
}
```

normal Scala conditional statement

ScalaHDL conditional statement (for now)

ScalaHDL assignment

# Bitonic Sort Program

```
def bitonicMerge(a: Seq[HDLType], z: Seq[HDLType], dir: Int) {  
  val n = a.size  
  val k = n / 2  
  if (n > 1) {  
    val t = (for (i <- 0 until n) yield bool(0)).map(toHDLType)  
    for (i <- 0 until k) {  
      compare(a(i), a(i + k), t(i), t(i + k), dir)  
    }  
    bitonicMerge(t.take(k), z.take(k), dir)  
    bitonicMerge(t.drop(k), z.drop(k), dir)  
  } else {  
    z.head := a.head  
  }  
}
```


# Bitonic Sort Program

```
def bitonicMerge(a: Seq[HDLType], z: Seq[HDLType], dir: Int) {  
  val n = a.size  
  val k = n / 2  
  if (n > 1) {  
    val t = (for (i <- 0 until n) yield bool(0)).map(toHDLType)  
    for (i <- 0 until k) {  
      compare(a(i), a(i + k), t(i), t(i + k), dir)  
    }  
    bitonicMerge(t.take(k), z.take(k), dir)  
    bitonicMerge(t.drop(k), z.drop(k), dir)  
  } else {  
    z.head := a.head  
  }  
}
```

convert `Signal` to `HDLType`,  
necessary for now

# Bitonic Sort Program

```
def bitonicSort(a: Seq[HDLType], z: Seq[HDLType], dir: Int) {  
  val n = a.size  
  val k = n / 2  
  if (n > 1) {  
    val t = (for (i <- 0 until n) yield bool(0)).map(toHDLType)  
    bitonicSort(a.take(k), t.take(k), ASC)  
    bitonicSort(a.drop(k), t.drop(k), DES)  
    bitonicMerge(t, z, dir)  
  } else {  
    z.head := a.head  
  }  
}
```



convert `Signal` to `HDLType`,  
necessary for now

# Bitonic Sort Program

```
defMod.sort8('a0', 'a1', 'a2', 'a3', 'a4', 'a5', 'a6', 'a7',  
  'z0', 'z1', 'z2', 'z3', 'z4', 'z5', 'z6', 'z7') {  
  val a = List('a0', 'a1', 'a2', 'a3', 'a4', 'a5', 'a6', 'a7').map(toHDLType)  
  val z = List('z0', 'z1', 'z2', 'z3', 'z4', 'z5', 'z6', 'z7').map(toHDLType)
```

```
  def compare(a: HDLType, b: HDLType,  
    x: HDLType, y: HDLType, dir: Int) {  
    ...  
  }
```

```
  def bitonicMerge(a: Seq[HDLType],  
    dir: Int) {  
    ...  
  }
```

```
  def bitonicSort(a: Seq[HDLType], z: Seq[HDLType],  
    dir: Int) {  
    ...  
  }
```

```
  bitonicSort(a, z, ASC) →  
}
```

convert HDLIdent to HDLType,  
necessary for now

HDLType is the reason why  
both HDLIdent and Signal  
can be applied here

# Distinguish between a and 'a

- a: Scala identifier, use val or var to declare it.
- 'a: ScalaHDL identifier, only used to hold Signals, and can only use := to update its value.
- You can do this in ScalaHDL:
  - `val a = 'a`, and then `a := bool(0)`
  - equivalent to `'a := bool(0)`

# Known Issues

- The logic to judge whether a signal is a register or wire, input or output is still broken.
- The generated code hasn't been tested.
- Simulation and unit test need to be adjusted.
- Features like generic if statements...

# Future Plan

- Fix the logic of signal type inference.
- Test the generated code.
- Simulation and unit test for the new structure.
- See to the generic if statement implementation.



Any Question?

Thanks!