DSL-based modular IP core generators: Example FFT and related structures
Francois Serre and Markus Pueschel
Objective

DFT

© Jeferson Santiago da Silva, use with author’s permission
Cooley-Tukey FFT

\[ \text{DFT}_{pq} = (\text{DFT}_p \otimes \text{I}_q) \cdot T_{pq}^q \cdot (\text{I}_p \otimes \text{DFT}_q) \cdot L_{pq}^p \]

DFT_{16}

Linear algebra rules
Cooley-Tukey FFT

\[
\text{DFT}_{16} = L_8^{16} \cdot \left( \prod_{\ell=1}^{4} (I_8 \otimes \text{DFT}_2) \cdot E^{16}_\ell \cdot Q^{16}_\ell \right) \cdot R^{16}
\]
Cooley-Tukey FFT

Radix-2 Iterative Cooley-Tukey FFT on 16 elements

\[
\text{DFT}_{16} = L_{8}^{16} \cdot \left( \prod_{\ell=1}^{4} (I_{8} \otimes \text{DFT}_{2}) \cdot E_{\ell}^{16} \cdot Q_{\ell}^{16} \right) \cdot R^{16}
\]
Cooley-Tukey FFT

Radix-2 Iterative Cooley-Tukey FFT on 16 elements

\[
\text{DFT}_{16} = L_{8}^{16} \cdot \left( \prod_{\ell=1}^{4} (I_8 \otimes \text{DFT}_2) \cdot E_{\ell}^{16} \cdot Q_{\ell}^{16} \right) \cdot R^{16}
\]
Generator pipeline

Transform

Algorithm

Streaming algorithm

SPL
Streaming FFTs [Milder et al., TODAES12]
Streaming FFTs [Milder et al., TODAES12]

How do we obtain these?
Streaming permutations [S, Püschel, FPGA’16]

**Radix-2 Iterative Cooley-Tukey FFT on 16 elements**

**Radix-2 Iterative Cooley-Tukey FFT on 16 elements streamed on 4 ports**
Generator pipeline

- Transform
- Algorithm
- Streaming algorithm
- Verilog RTL

SPL

Streaming block DSL
Scala and LMS  [Rompf and Odersky, Com ACM 12]

Regular Scala function

```scala
def add(a: Double, b: Double) = {
  a + b
}
```

Staged Scala function

```scala
def add(a: Sig[Double], b: Sig[Double]) = {
  a + b
}
```

Idea: Type Sig[Double] mimics a Double, but keeps track of the operations

© Alen Stojanov, use with author’s permission
Each `Sig[_]` contains a hardware arithmetic representation

```scala
abstract class HW[T] (val size: Int) {
    def getBits(value: T): BigInt
    def Plus(lhs: Sig[T], rhs: Sig[T]): Sig[T]
    def Minus(lhs: Sig[T], rhs: Sig[T]): Sig[T]
    def Times(lhs: SigRep[T], rhs: Sig[T]): Sig[T]
    ...
}
```

Example: hardware representations of Double

```scala
case class FixPoint(integral: Int, fractional: Int) extends HW[Double](integral + fractional) {...}
case class IEEEFP(wE: Int, wF: Int) extends HW[Double](wE + wF + 1) {...}
case class FloPoCo(wE: Int, wF: Int) extends HW[Double](wE + wF + 3) {...}
```

Ad-hoc polymorphism: all existing code works with a new hardware arithmetic representation
Arithmetic representation [S. and Püschel, FPL’18]

**Example: twiddle factors**

```scala
case class Twiddles(n: Int, k: Int, j: Sig[Int])(implicit dt: HW[Complex[Double]])
  extends StreamingBlock[Complex[Double]](1 << n, 1 << k){
    override def implement(inputs: Vector[Sig[Complex[Double]]]){1 << n} = {
      val rootsOfUnity = Vector.tabulate(1 << n){i =>
        val angle = -2 * Math.PI * i / (1 << n)
        Complex(Math.cos(angle), Math.sin(angle))
      }
      val timer = Timer(1 << t)
      inputs.zipWithIndex.map{case (input, p) =>
        val i = timer ++ Unsigned(k)(p)
        val address = (i & 1) * ((i >>> (j + 1)) << j)
        val twiddle = rootsOfUnity(address)
        input * twiddle
      }
    } }
```

Works with all streaming scenarios and arithmetic representations
Generator pipeline

- Transform
  - SPL
- Algorithm
  - Streaming block DSL
- Streaming algorithm
  - RTL-level DSL
- Verilog RTL
Related work

■ Hardware DSLs in Scala:
  ▪ Bachrach et al., *Chisel: constructing hardware in a Scala embedded language*, DAC12
  ▪ Port and Etsion, *DFiant: A dataflow hardware description language*, FPL17
  ▪ Sujeeth et al., *OptiML: an implicitly parallel domain-specific language for machine learning*, ICML11
  ▪ George et al., *Making domain-specific hardware synthesis tools cost-efficient*, FPT13

■ FFT generators:
  ▪ Milder et al., *Computer generation of hardware for linear digital signal processing transforms*, TODAES12
  ▪ Mainland and Johnson, *A Haskell compiler for signal transforms*, GPCE17
Conclusion

- DSL-based generator for FFTs
- Uses three abstraction levels
- Implemented using modern language features and principled methods
- Principles applicable beyond FFT

WHT on 32 elements streaming on 8 ports