ETH zürich - Department of Computer Science

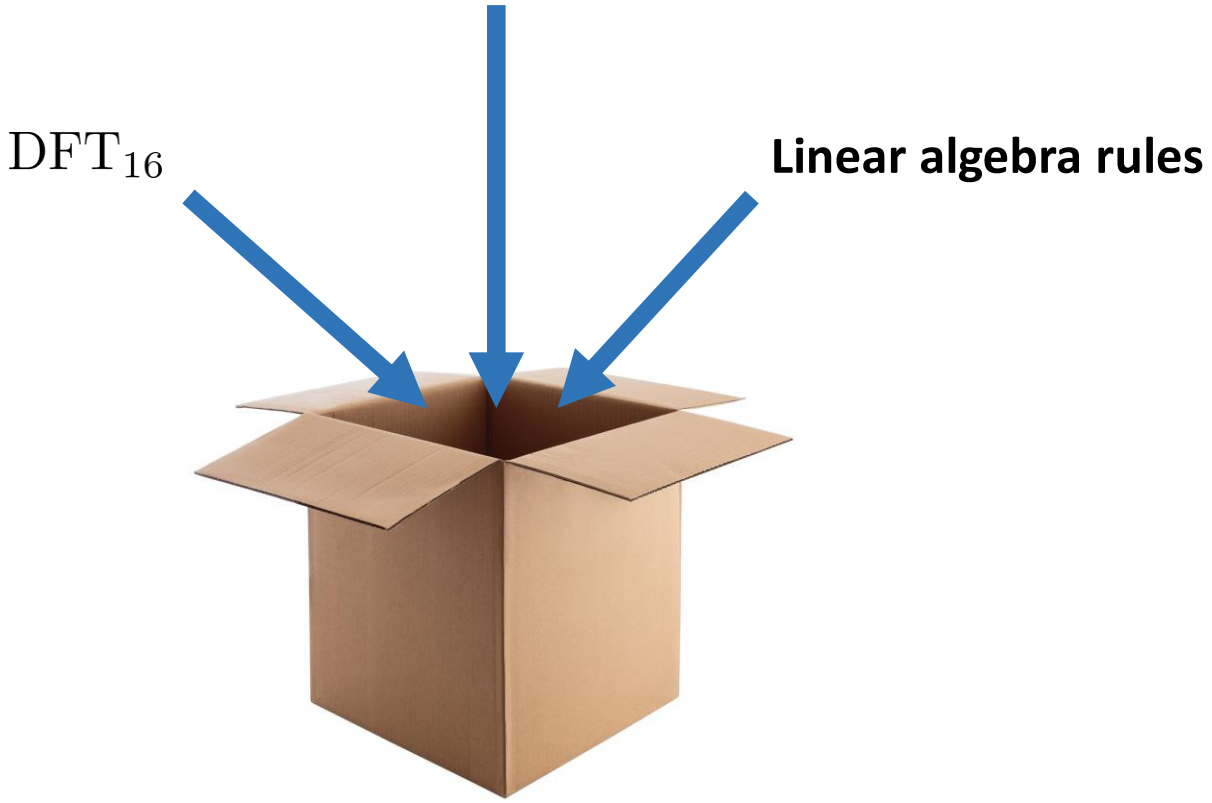
DSL-based modular IP core generators: Example FFT and related structures

François Serre and Markus Püschel

Objective

Cooley-Tukey FFT

$$\mathrm{DFT}_{pq} = (\mathrm{DFT}_p \otimes I_q) \cdot T^{pq}_q \cdot (I_p \otimes \mathrm{DFT}_q) \cdot L^{pq}_p$$



Cooley-Tukey FFT

$$\mathrm{DFT}_{16} = L_8^{16} \cdot \left(\prod_{\ell=1}^{4} (I_8 \otimes \mathrm{DFT}_2) \cdot E_\ell^{16} \cdot Q_\ell^{16} \right) \cdot R^{16}$$

Cooley-Tukey FFT

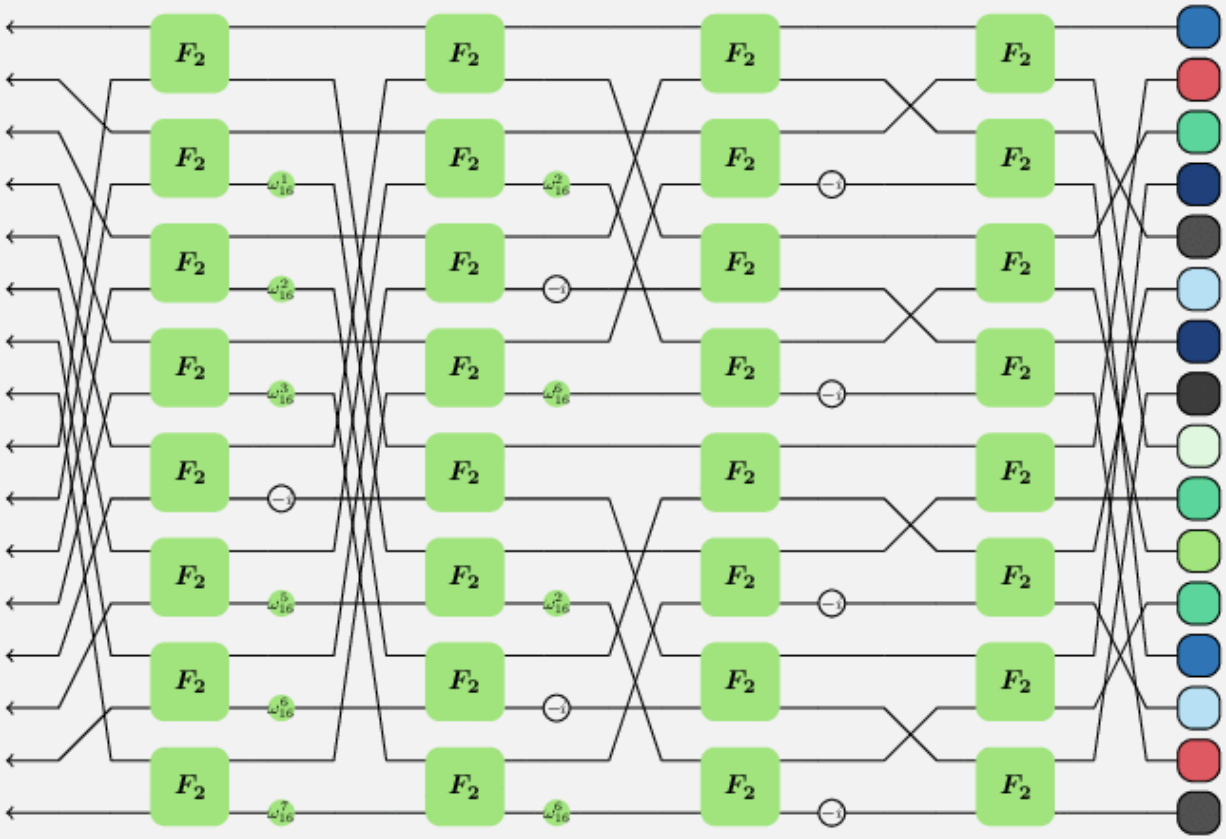
Radix-2 Iterative Cooley-Tukey FFT on 16 elements

$$\mathrm{DFT}_{16} = L_8^{16} \cdot \left(\prod_{\ell=1}^{4} (I_8 \otimes \mathrm{DFT}_2) \cdot E_\ell^{16} \cdot Q_\ell^{16} \right) \cdot R^{16}$$

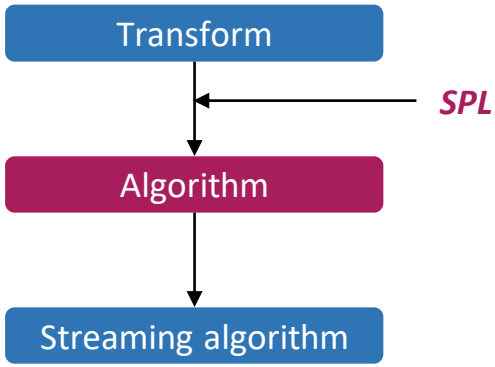
Cooley-Tukey FFT

Radix-2 Iterative Cooley-Tukey FFT on 16 elements

$$\mathrm{DFT}_{16} = L_8^{16} \cdot \left(\prod_{\ell=1}^{4} (I_8 \otimes \mathrm{DFT}_2) \cdot E_\ell^{16} \cdot Q_\ell^{16} \right) \cdot R^{16}$$

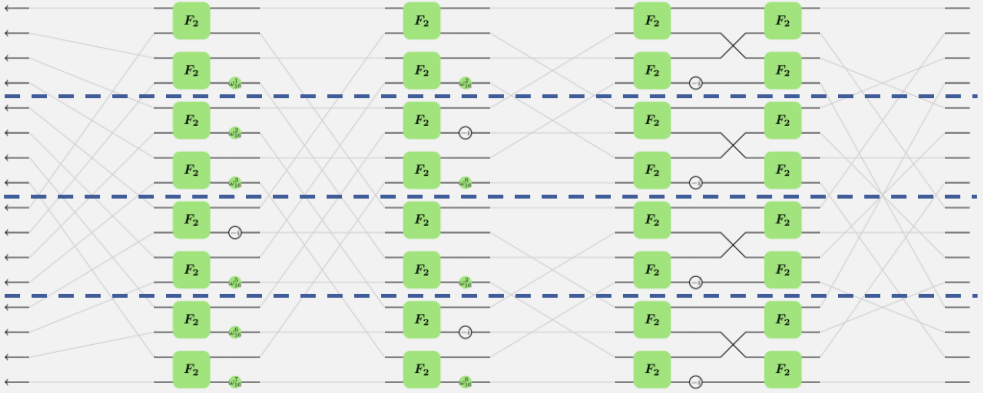


Generator pipeline



Streaming FFTs [Milder et al., TODAES12]

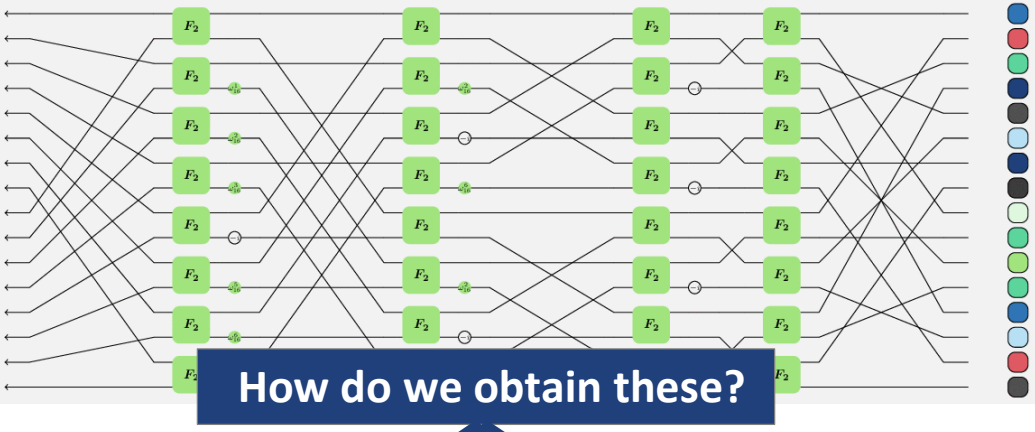
Radix-2 Iterative Cooley-Tukey FFT on 16 elements



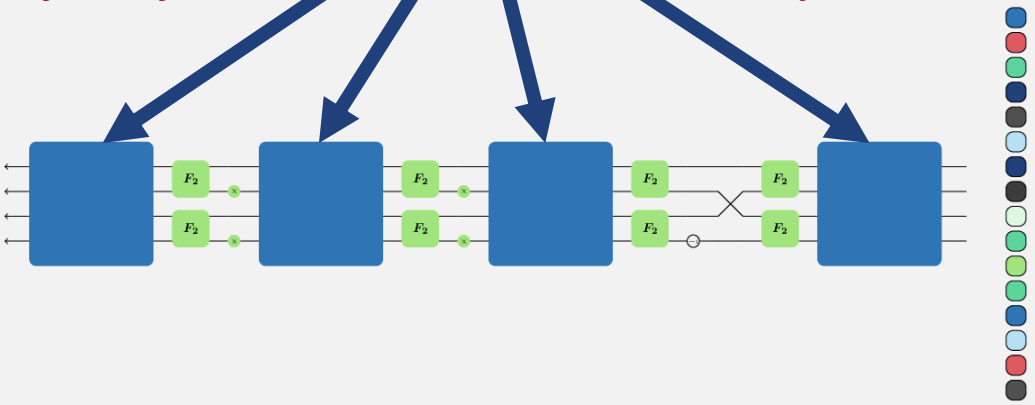
Radix-2 Iterative Cooley-Tukey FFT on 16 elements streamed on 4 ports

Streaming FFTs [Milder et al., TODAES12]

Radix-2 Iterative Cooley-Tukey FFT on 16 elements

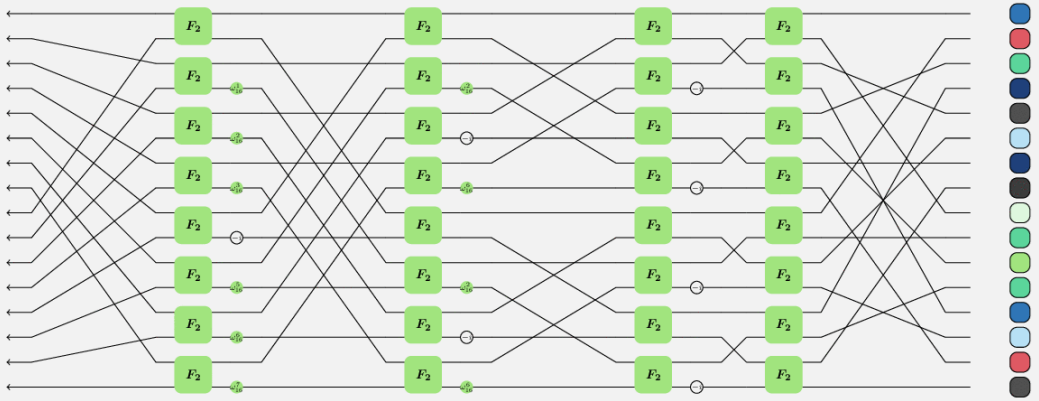


Radix-2 Iterative Cooley-Tukey FFT on 16 elements streamed on 4 ports

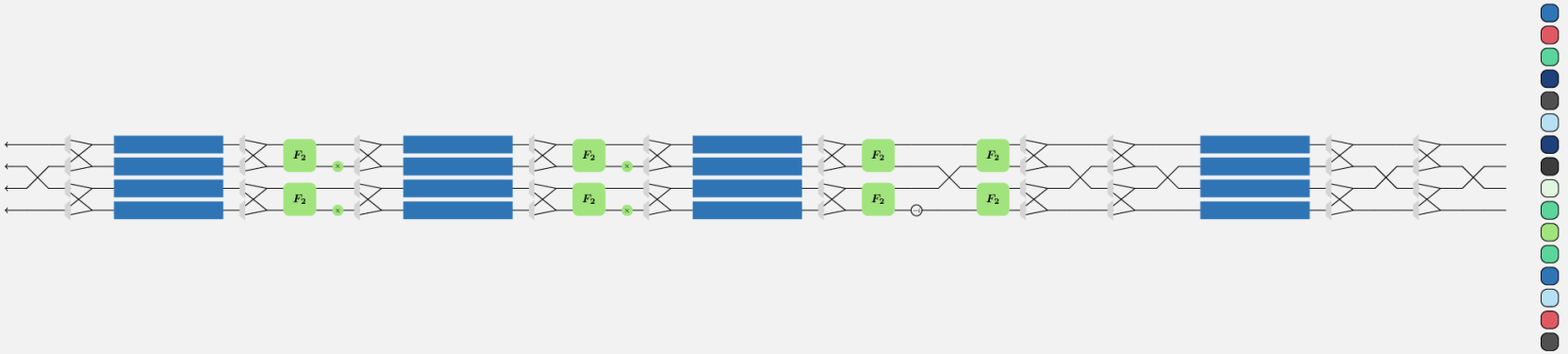


Streaming permutations [S, Püschel, FPGA'16]

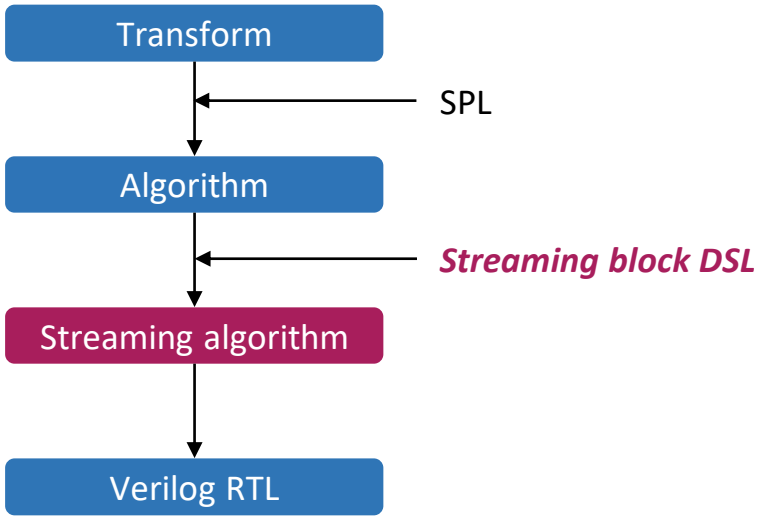
Radix-2 Iterative Cooley-Tukey FFT on 16 elements



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



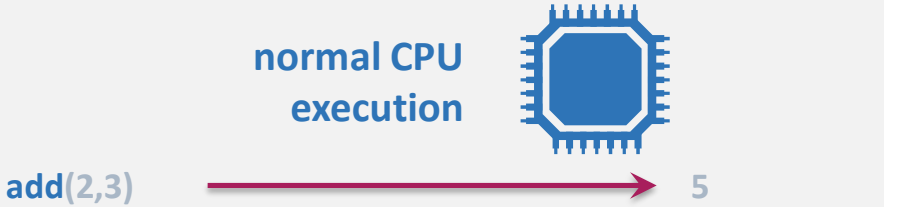
Generator pipeline



Scala and LMS [Rompf and Odersky, Com ACM 12]

Regular Scala function

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



Staged Scala function

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

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

Arithmetic representation [S. and Püschel, FPL'18]

Each Sig[_] ***contains a hardware arithmetic representation***

```
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

```
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**

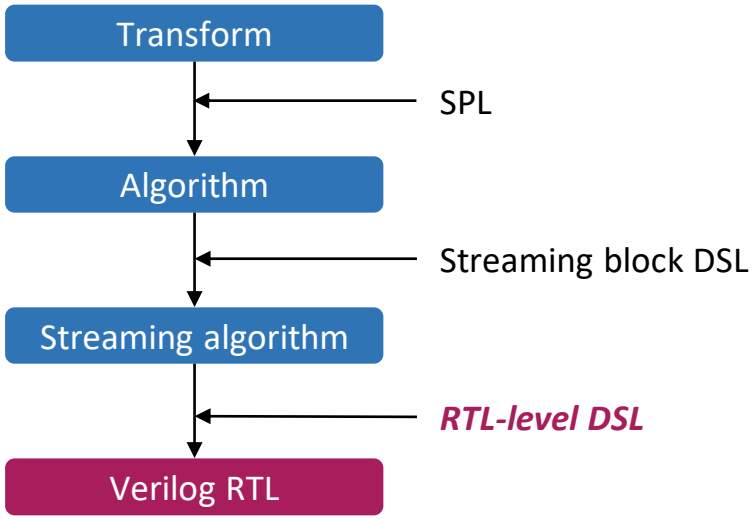
Arithmetic representation [S. and Püschel, FPL'18]

Example: twiddle factors

```
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]]]) = {
    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



Related work

- **Hardware DSLs in Scala:**
  - **Bachrach et al.,** *Chisel: constructing hardware in a Scala embeded language*, **DAC12**
  - **Port and Etsion,** *DFiant: A dataflow hardware descripion 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

serref@inf.ethz.ch