Specification of APERTIF Polyphase Filter Bank in CλaSH

Rinse Wester[a], Dimitrios Sarakiotis[a], Eric Kooistra[b], Jan Kuper[a]

University of Twente, Enschede
ASTRON, Dwingelo

August 27, 2012

Contents

- Introduction
- Background
- Describing the Filter bank using CλaSH
- Results
- Conclusions & Future Work

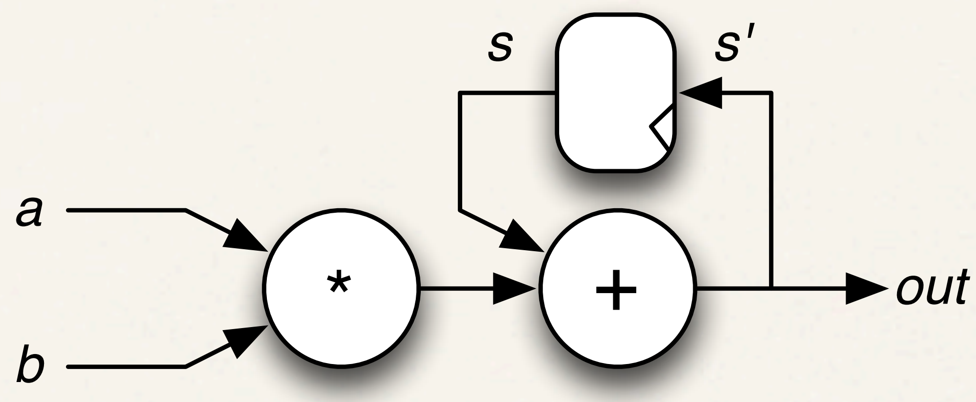
Introduction

- What is CλaSH?
  - Functional Language and Compiler for Concurrent Digital Hardware Design
- Motivation?
  - Testing CλaSH on real life complex application
- Why APERTIF Polyphase filter bank?
  - Strict specification on Throughput, Area and clock frequency

Background

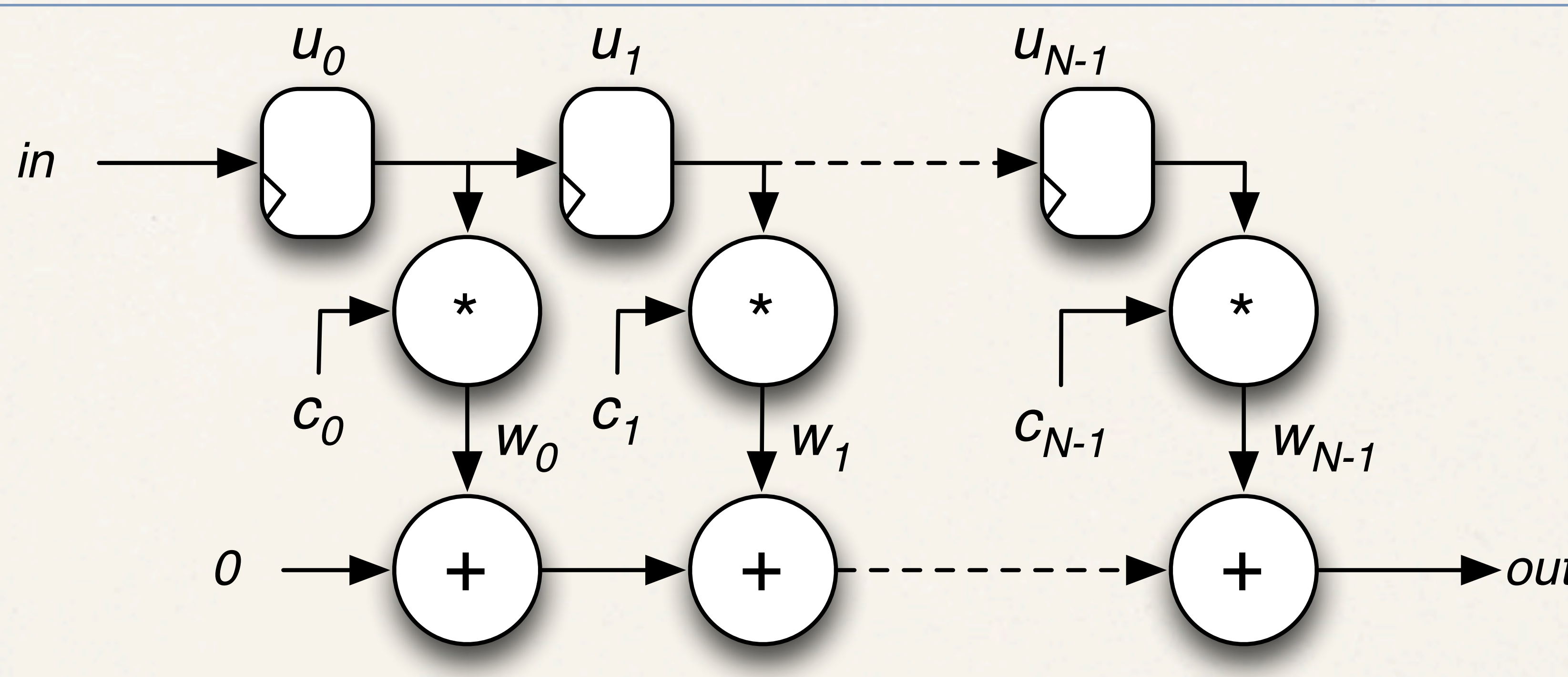
- CλaSH
  - A functional language and compiler for digital hardware design
  - On the lowest level, everything is a Mealy machine $f(s,i) = (s',o)$
  - A CλaSH description is purely structural i.e. all operations are performed in a single clock cycle
  - Simulation is cycle accurate

Background



$$
\begin{aligned}
&mac\ (State\ s)\ (a, b) = (State\ s', out) \\
&\quad \textbf{where} \\
&\qquad s' \quad = s + a * b \\
&\qquad out = s'
\end{aligned}
$$

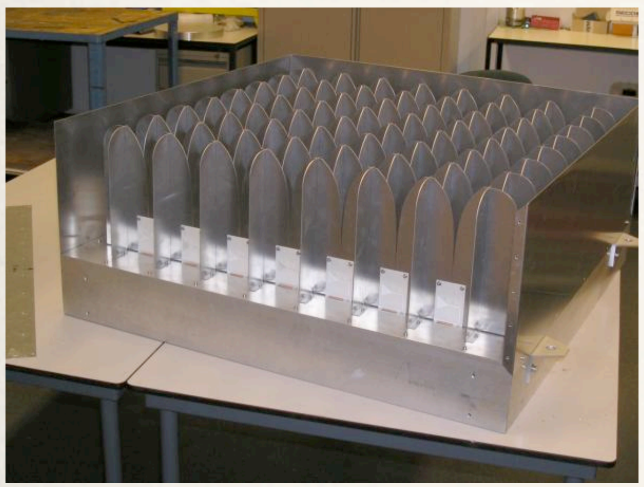
Background

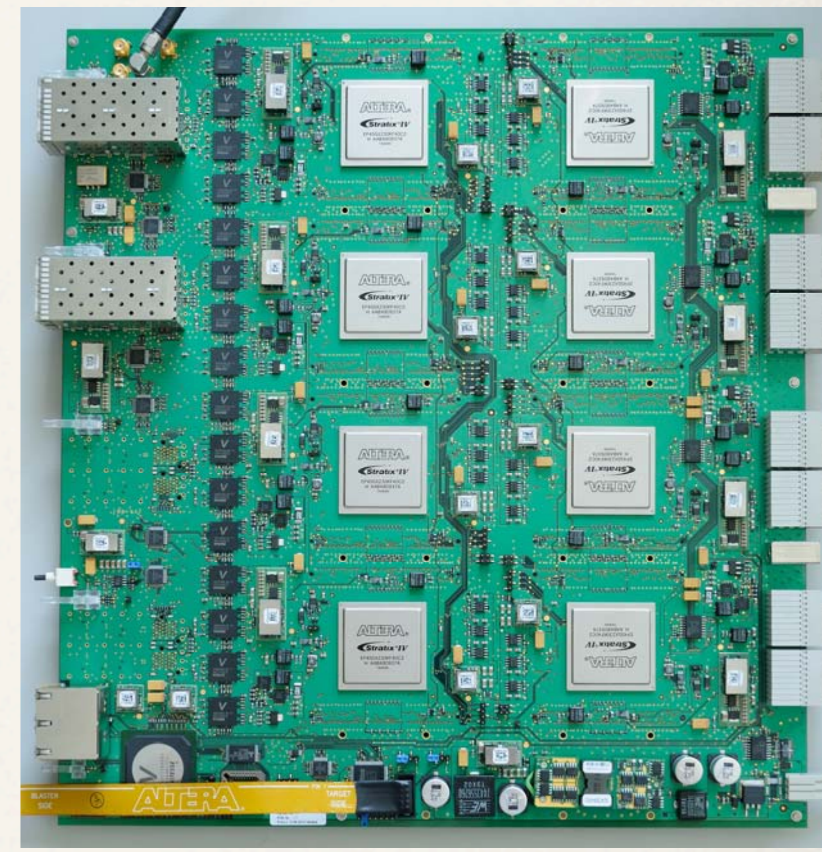


$$
\begin{aligned}
&\mathit{fir}\ cs\ (State\ us)\ inp = (State\ us', out)\\
&\quad \textbf{where}\\
&\qquad us' = inp \mathbin{+\!\!\gg} us\\
&\qquad ws\ \ = vzipWith\ (*)\ us\ cs\\
&\qquad out = vfoldl\ (+)\ 0\ ws
\end{aligned}
$$

Background

APERTIF project

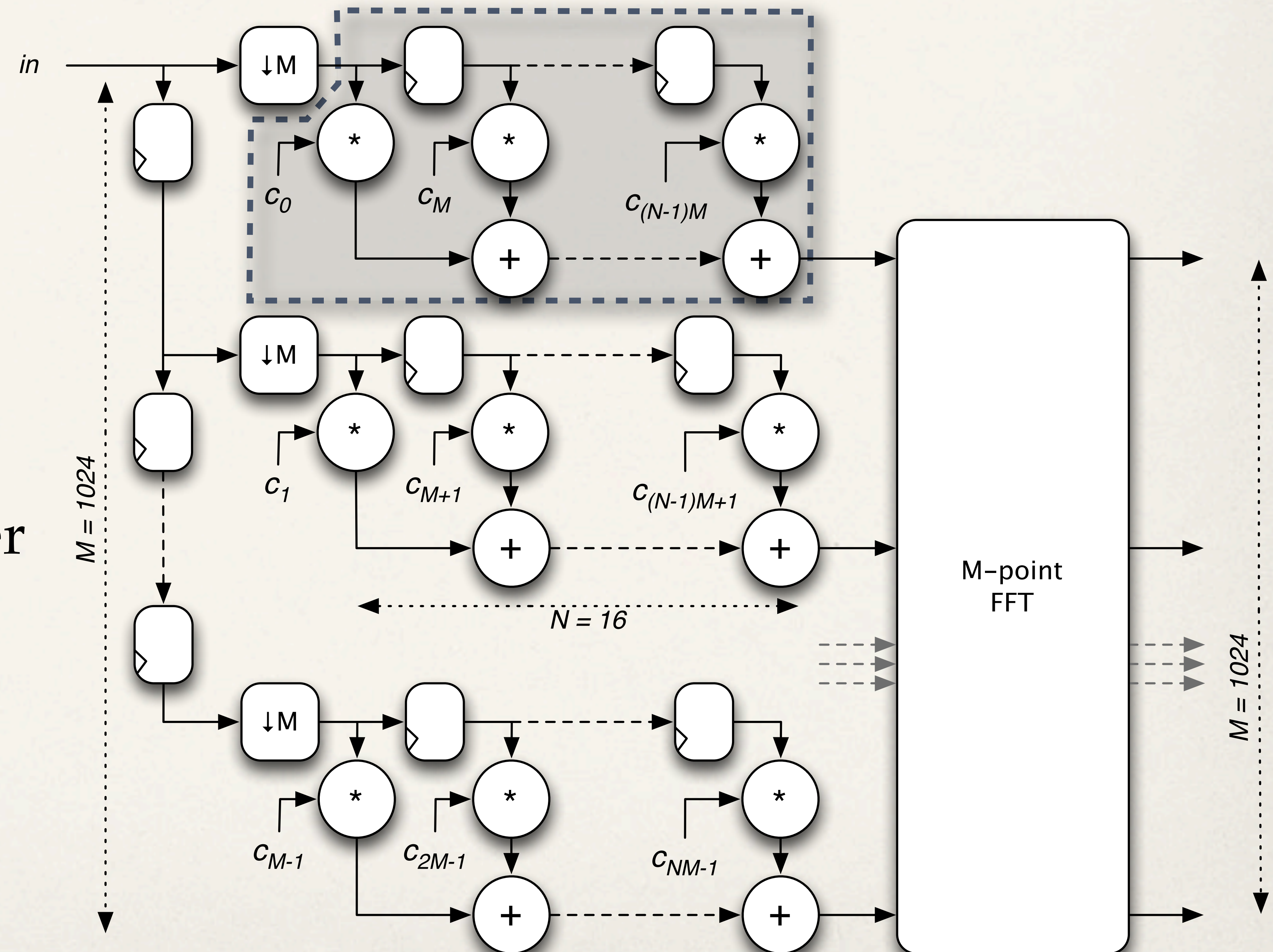




- APERTIF Polyphase Filter bank
  - Increasing field of view of Westerbork telescope using small array
  - Each antenna in the array requires a polyphase filter bank
  - Goals: $F_{clk} = 200\ MHz$ and throughput = $800\ MS/s$

Background

- APERTIF Polyphase Filter bank
  - Polyphase FIR filter
  - FFT

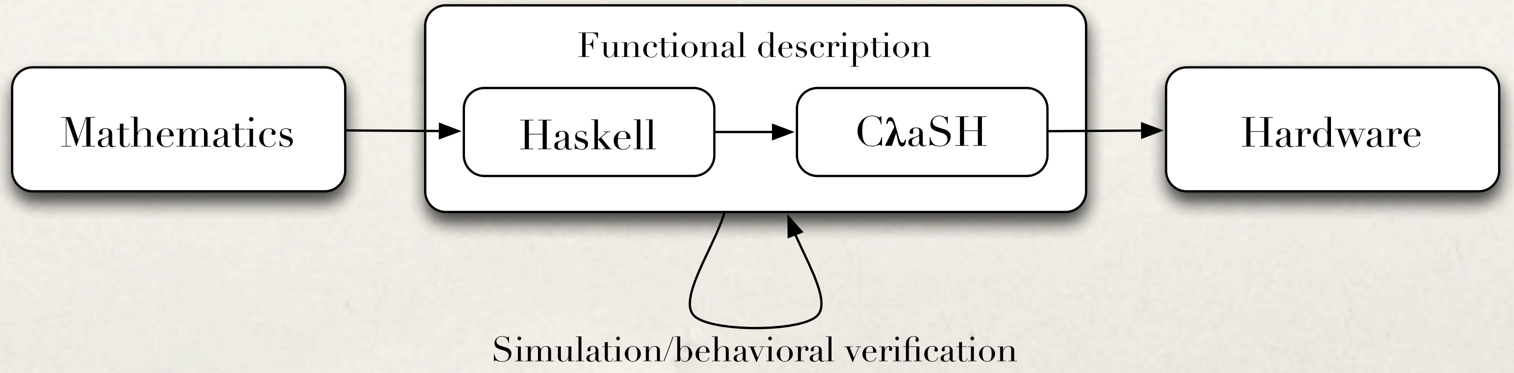


Describing the PFB

- Design method
- Polyphase filter
- FFT pipeline

Describing the PFB

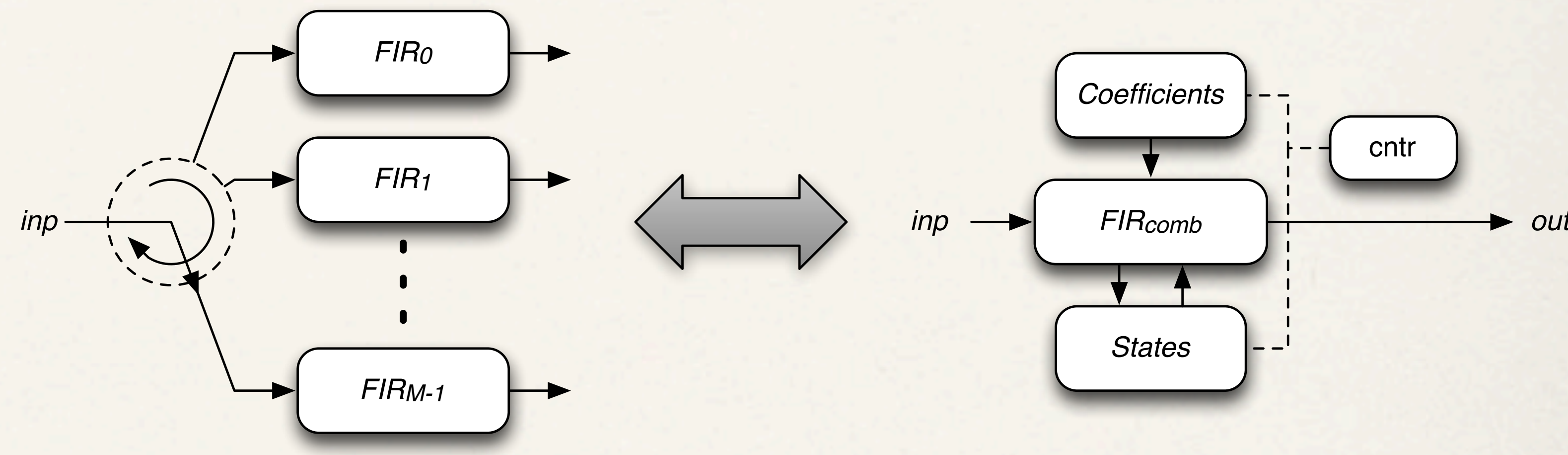
- Design whole Architecture first in plain Haskell
- Perform small modification such that the code is accepted by CλaSH
  - Lists are replaced by vectors: lists with fixed length
  - Fixed point representation for numbers
- A clear division between structure and low level hardware details



Describing the PFB

Polyphase Filter

- A set of FIR filters sequentially activated
- Parallelization of P=4 needed

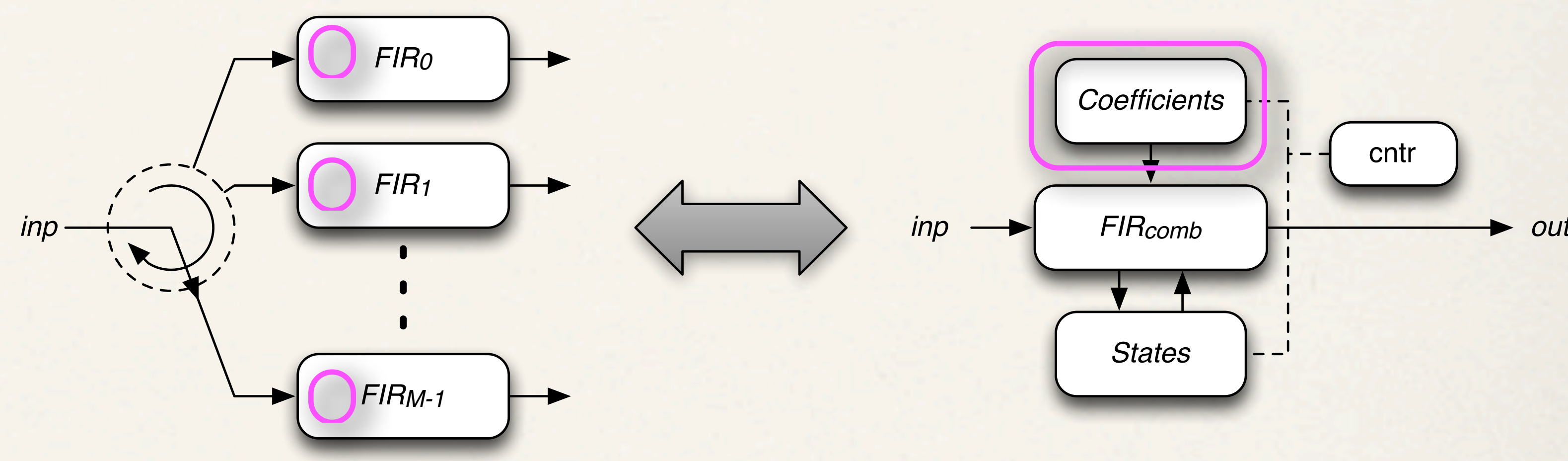


$$
\begin{aligned}
& \mathit{pfs}\ \mathit{css}\ (\mathit{uss}, \mathit{cntr})\ \mathit{inp} = ((\mathit{uss}', \mathit{cntr}'), \mathit{out}) \\
& \quad \mathbf{where} \\
& \qquad \mathit{cntr}' \qquad\quad = (\mathit{cntr} + 1)\ \text{`}\mathit{mod}\text{`}\ (\mathit{length}\ \mathit{css}) \\
& \qquad \mathit{us} \qquad\qquad = \mathit{uss}\ !\ \mathit{cntr} \\
& \qquad \mathit{cs} \qquad\qquad = \mathit{css}\ !\ \mathit{cntr} \\
& \qquad (\mathit{us}', \mathit{out}) = \mathit{fir}\ \mathit{cs}\ \mathit{us}\ \mathit{inp} \\
& \qquad \mathit{uss}' \qquad\quad = \mathit{replace}\ \mathit{cntr}\ \mathit{us}'\ \mathit{uss}
\end{aligned}
$$

Describing the PFB

Polyphase Filter

- A set of FIR filters sequentially activated
- Parallelization of P=4 needed

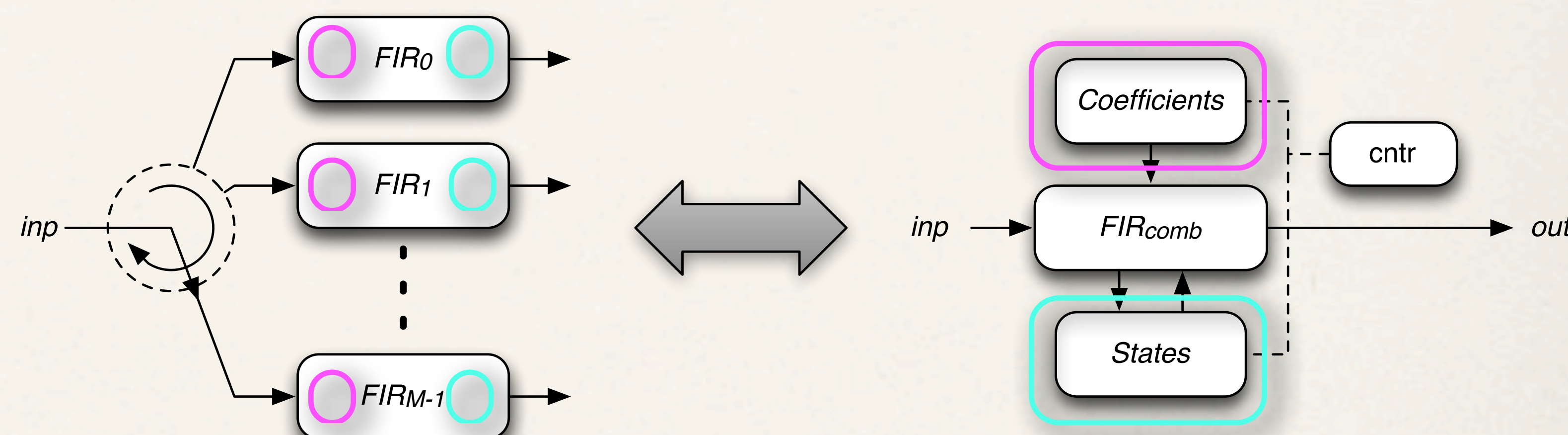


$$
\begin{aligned}
& pfs\ \boxed{css}\ (uss, cntr)\ inp = ((uss', cntr'), out) \\
& \quad \textbf{where} \\
& \qquad cntr' && = (cntr + 1)\ \text{`}mod\text{`}\ (length\ css) \\
& \qquad us && = uss\ !\ cntr \\
& \qquad \boxed{cs \quad = css\ !\ cntr} \\
& \qquad (us', out) && = fir\ cs\ us\ inp \\
& \qquad uss' && = replace\ cntr\ us'\ uss
\end{aligned}
$$

Describing the PFB

Polyphase Filter

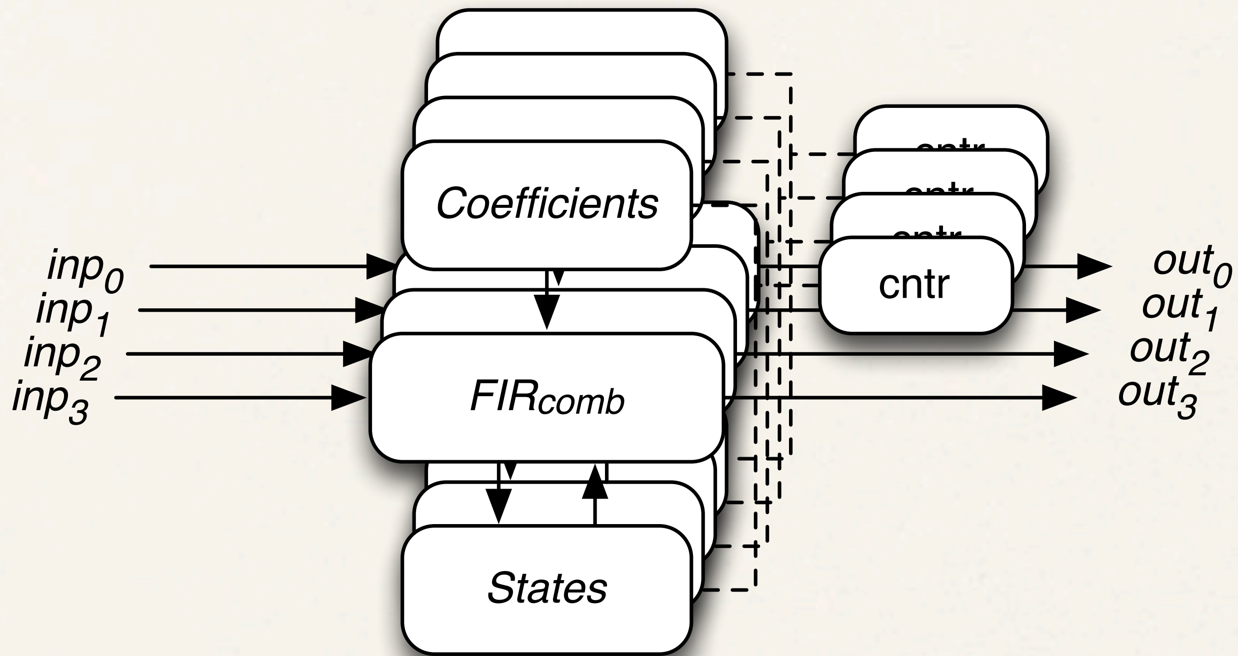
- A set of FIR filters sequentially activated
- Parallelization of P=4 needed



$$
\begin{aligned}
&pfs\ css\ (uss, cntr)\ inp = ((uss', cntr'), out) \\
&\quad \mathbf{where} \\
&\qquad cntr' \quad = (cntr + 1)\ \text{`}mod\text{`}\ (length\ css) \\
&\qquad us \quad = uss\ !\ cntr \\
&\qquad cs \quad = css\ !\ cntr \\
&\qquad (us', out) = fir\ cs\ us\ inp \\
&\qquad uss' \quad = replace\ cntr\ us'\ uss
\end{aligned}
$$

Describing the PFB

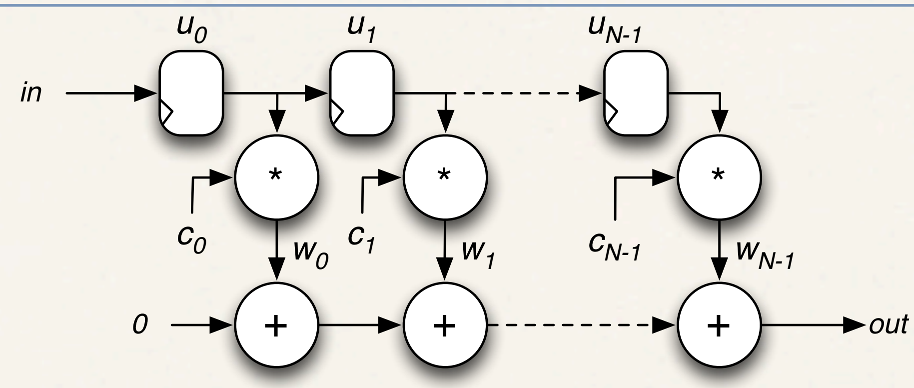
Parallel Polyphase Filter



$$
\begin{aligned}
&parpfs\ csss\ states\ inps = (states', outs)\\
&\quad \textbf{where}\\
&\qquad res \qquad\qquad\quad\; = zipWith3\ pfs\ csss\ states\ inps\\
&\qquad (states', outs) = unzip\ res
\end{aligned}
$$

Describing the PFB

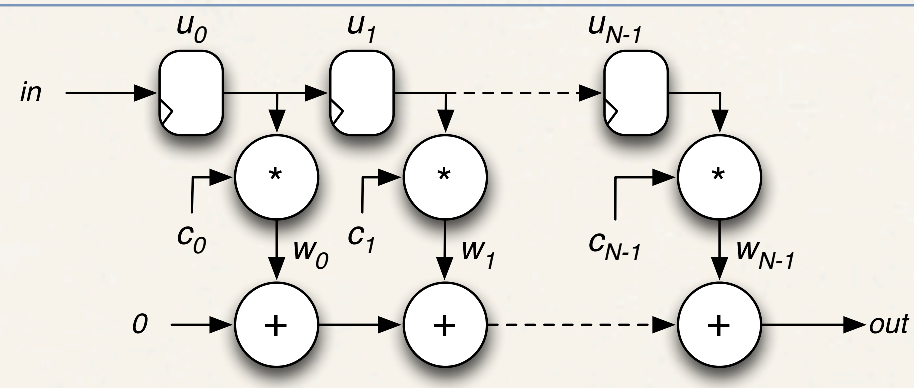
FIR filter: Haskell → CλaSH



$$
\begin{array}{l}
\mathit{fir} :: [\mathit{Double}] \rightarrow (\mathit{State}\ [\mathit{Double}]) \rightarrow \mathit{Double} \rightarrow (\mathit{State}\ [\mathit{Double}], \mathit{Double}) \\
\mathit{fir}\quad \mathit{cs}\qquad\qquad\quad (\mathit{State}\ \mathit{us})\qquad\qquad \mathit{inp}\qquad = (\mathit{State}\ \mathit{us}\ \ , \mathit{out}) \\
\quad \textbf{where} \\
\qquad \mathit{us}\ \ = \mathit{inp}\ \texttt{+>>}\ \mathit{us} \\
\qquad \mathit{ws}\ \ = \mathit{zipWith}\ (*)\ \mathit{uscs} \\
\qquad \mathit{out} = \mathit{foldl}\ (+)\ 0\ \mathit{ws}
\end{array}
$$

Describing the PFB

FIR filter: Haskell → CλaSH

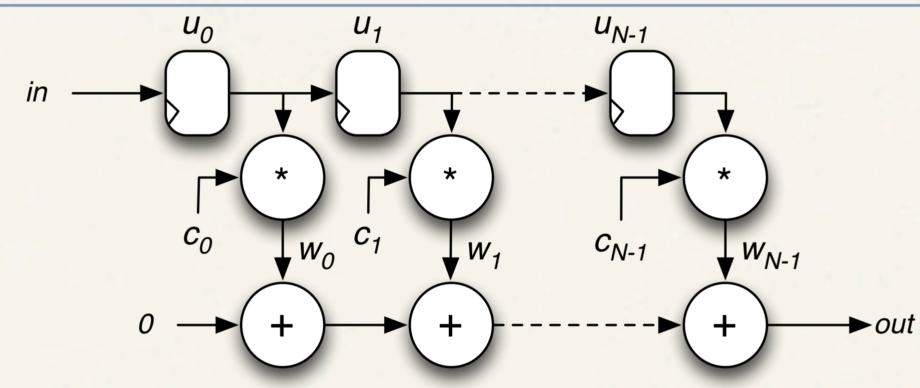


$$
\begin{array}{l}
\mathit{fir} :: [\mathit{Double}] \to (\mathit{State}\ [\mathit{Double}]) \to \mathit{Double} \to (\mathit{State}\ [\mathit{Double}], \mathit{Double}) \\
\mathit{fir}\quad \mathit{cs}\qquad (\mathit{State}\ \mathit{us})\qquad \mathit{inp}\qquad = (\mathit{State}\ \mathit{us}\ , \mathit{out}) \\
\quad \textbf{where} \\
\qquad \mathit{us}\ \ = \mathit{inp} \mathbin{+\!\!\gg} \mathit{us} \\
\qquad \mathit{ws}\ \ = \mathit{zipWith}\ (*)\ \mathit{uscs} \\
\qquad \mathit{out} = \mathit{foldl}\ (+)\ 0\ \mathit{ws}
\end{array}
$$

$$
\begin{array}{l}
\mathit{fir} :: (\mathit{Vector}\ \mathit{D16}\ S) \to (\mathit{State}\ (\mathit{Vector}\ \mathit{D16}\ S)) \to S \to (\mathit{State}\ (\mathit{Vector}\ \mathit{D16}\ S), S) \\
\mathit{fir}\quad \mathit{cs}\qquad (\mathit{State}\ \mathit{us})\qquad \mathit{inp} = (\mathit{State}\ \mathit{us},\qquad \mathit{out}) \\
\quad \textbf{where} \\
\qquad \mathit{us}\ \ = \mathit{inp} \mathbin{+\!\!\gg} \mathit{us} \\
\qquad \mathit{ws}\ \ = \mathit{vzipWith}\ \text{`}\mathit{fpmult}\text{`}\ \mathit{uscs} \\
\qquad \mathit{out} = \mathit{vfoldl}\ (+)\ 0\ \mathit{ws}
\end{array}
$$

Describing the PFB

FIR filter: Haskell → CλaSH

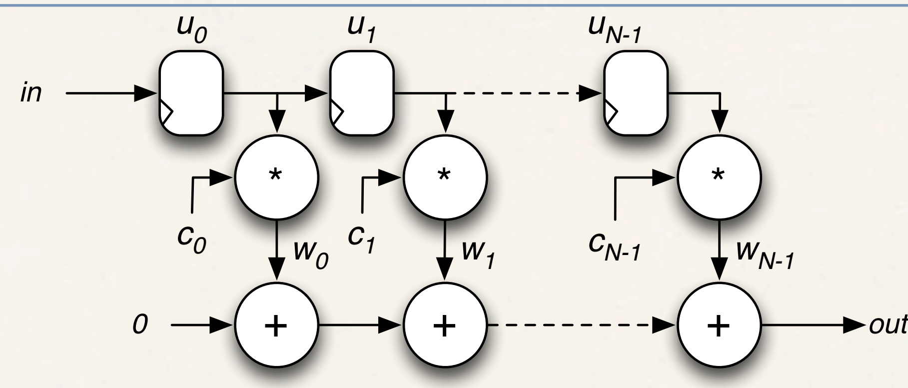



```
fir :: [Double] → (State [Double]) → Double → (State [Double], Double)
fir    cs          (State us)        inp    = (State us   , out)
  where
    us  = inp +>> us
    ws  = zipWith (*) uscs
    out = foldl (+) 0 ws
```

```
fir :: (Vector D16 S) → (State (Vector D16 S)) → S   → (State (Vector D16 S), S)
fir    cs               (State us)                inp = (State us,              out)
  where
    us  = inp +>> us
    ws  = vzipWith `fpmult` uscs
    out = vfoldl (+) 0 ws
```

Describing the PFB

FIR filter: Haskell → CλaSH

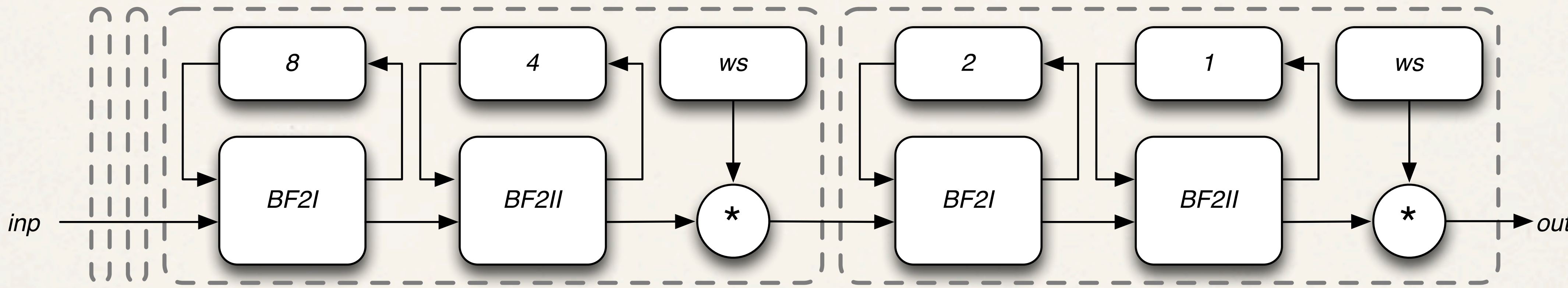



```
fir :: [Double] → (State [Double]) → Double → (State [Double], Double)
fir    cs          (State us)         inp    = (State us     , out)
  where
    us  = inp +>> us
    ws  = zipWith (*) us cs
    out = foldl (+) 0 ws
```

```
fir :: (Vector D16 S) → (State (Vector D16 S)) → S   → (State (Vector D16 S), S)
fir    cs                (State us)               inp = (State us,            out)
  where
    us  = inp +>> us
    ws  = vzipWith `fpmult` us cs
    out = vfoldl (+) 0 ws
```

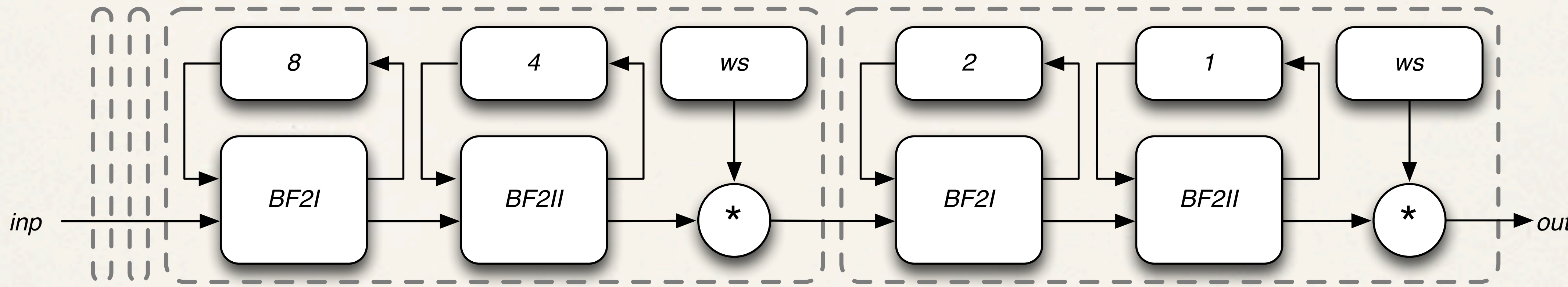
Describing the PFB

FFT pipeline



Describing the PFB

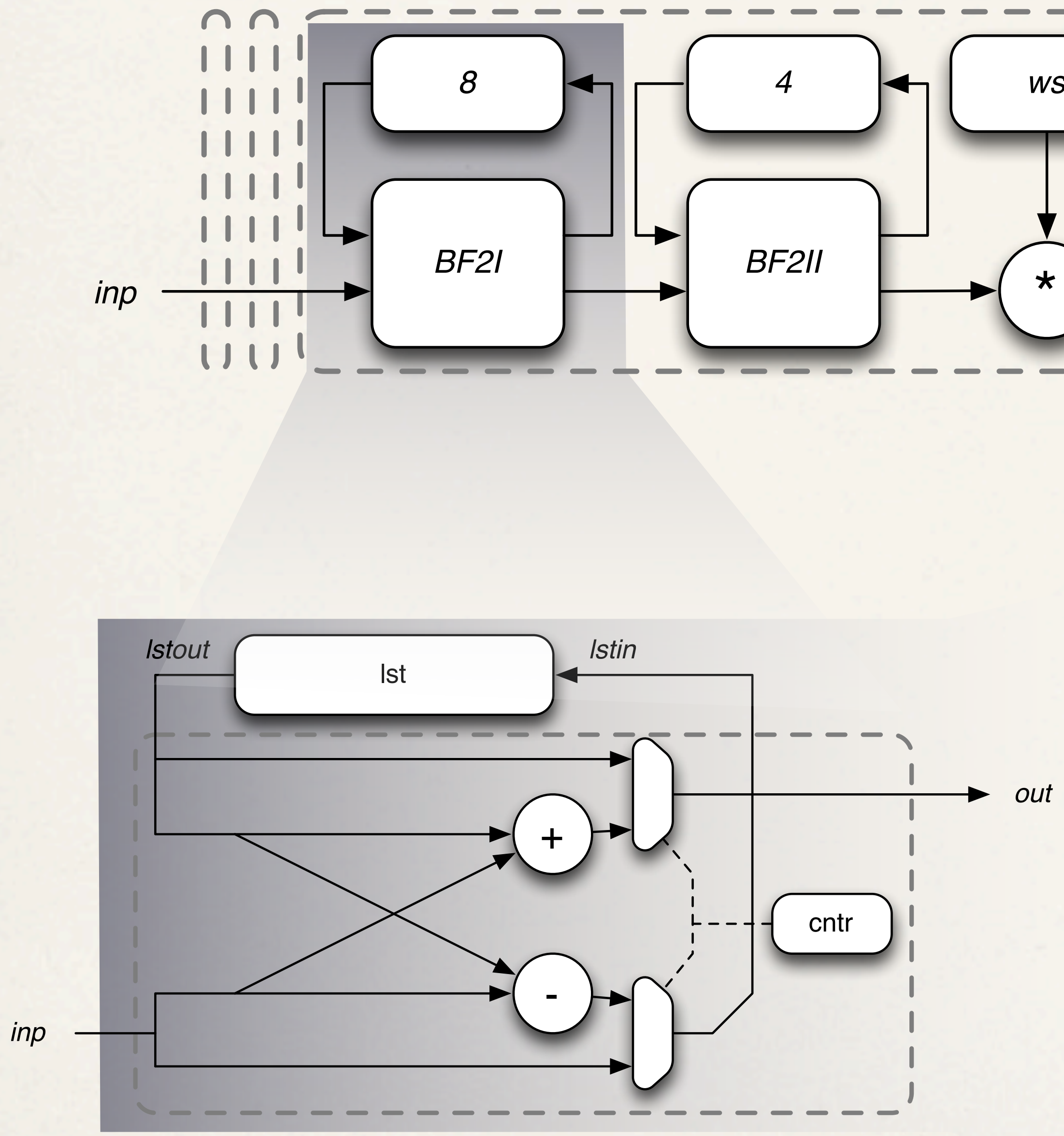
FFT pipeline



- FFT is implemented using pipeline
  - Two types of butterflies and Complex multiplier

Describing the PFB

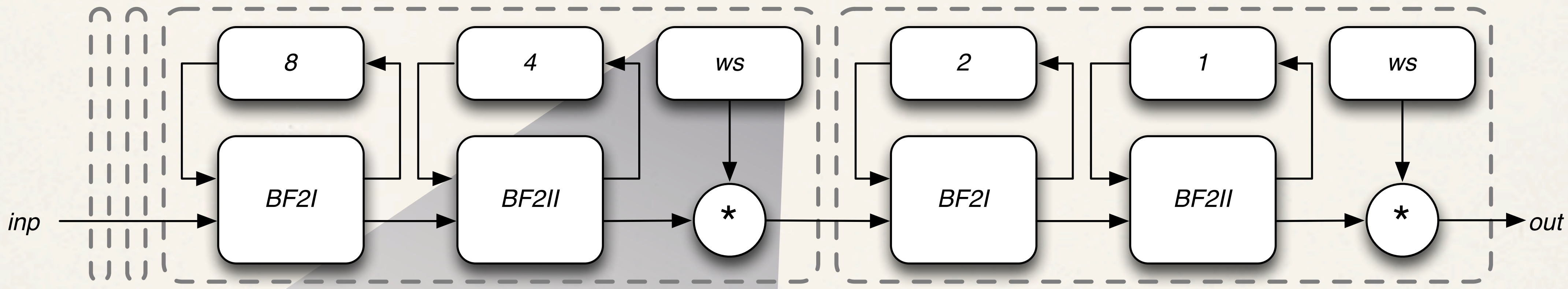
FFT pipeline




```
bf2i (cntr, lst) inp = ((cntr', lst'), out)
  where
    n             = length lst
    cntr'         = (cntr + 1) `mod` n
    lst'          = lstin +>> lst
    (out, lstin)  = if cntr >= n
                      then (lstout + inp, lstout - inp)
                      else (lstout      , inp          )
    lstout        = last lst
```

Describing the PFB

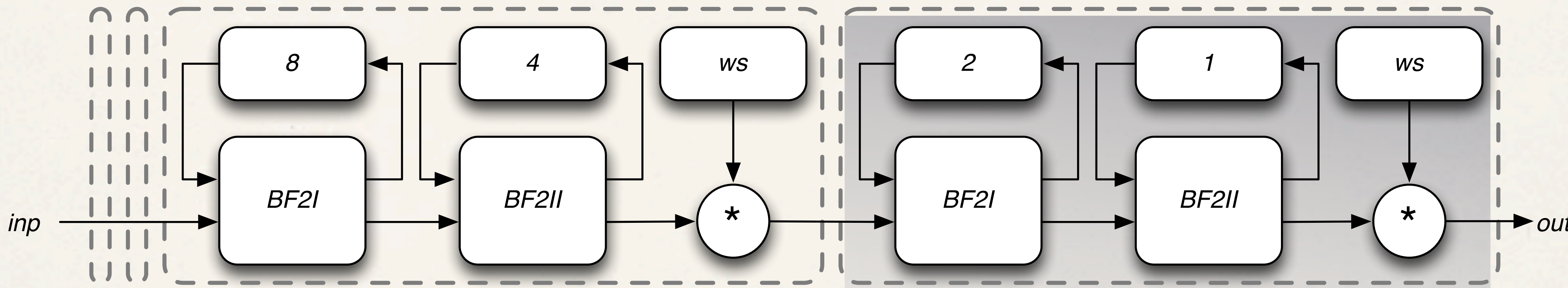
FFT pipeline



$$
\begin{array}{l}
\mathit{cmult}\ \mathit{ws}\ \mathit{cntr}\ \mathit{inp} = (\mathit{cntr}', \mathit{out}) \\
\quad \textbf{where} \\
\qquad \begin{array}{lcl}
n & = & \mathit{length}\ \mathit{ws} \\
\mathit{cntr}' & = & (\mathit{cntr} + 1)\ `\mathit{mod}`\ n \\
w & = & \mathit{ws}\ !\ \mathit{cntr} \\
\mathit{out} & = & \mathit{inp} * w
\end{array}
\end{array}
$$

Describing the PFB

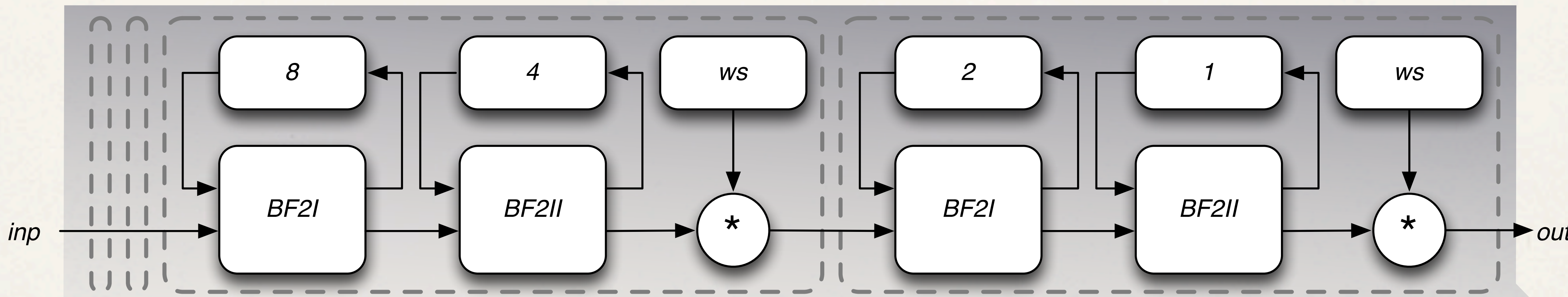
FFT pipeline



$$
\begin{aligned}
&\mathit{fftbb}\ ws\ (bf1state, bf2state, cmstate)\ inp = ((bf1state', bf2state', cmstate'), out)\\
&\quad \mathbf{where}\\
&\qquad (bf1state', a) \quad = bf2i\ bf1state\ inp\\
&\qquad (bf2state', b) \quad = bf2ii\ bf2state\ a\\
&\qquad (cmstate', out) = cmult\ ws\ cmstate\ b
\end{aligned}
$$

Describing the PFB

FFT pipeline



$$\begin{aligned}
&\mathit{fftchain}\ (ws1, ws2, \ldots)\ (bb1state, bb2state, \ldots)\ inp = ((bb1state', bb2state', \ldots), out)\\
&\quad \textbf{where}\\
&\qquad (bb1state', d1) \ \ = \mathit{fftbb}\ ws1\ \ bb1state\ inp\\
&\qquad (bb2state', d2) \ \ = \mathit{fftbb}\ ws2\ \ bb2state\ d1\\
&\qquad\quad \circ\\
&\qquad\quad \circ\\
&\qquad (bbNstate', out) = \mathit{fftbb}\ wsN\ \ bbNstate\ d9
\end{aligned}$$

Describing the PFB

FFT BF2I: Haskell → CλaSH

```
bf2i (cntr, lst) inp = ((cntr', lst'), out)
  where
    n            = length lst
    cntr'        = (cntr + 1) `mod` n
    lst'         = lstin +>> lst
    (out, lstin) = if cntr >= n
                   then (lstout + inp, lstout - inp)
                   else (lstout      , inp         )
    lstout       = last lst
```

```
bf2i_clash (cntr, lst) inp = ((cntr', lst'), out)
  where
    n            = vlength lst
    cntr'        = cntr + 1
    lst'         = lstin +>> lst
    (out, lstin) = if cntr >= n
                   then (lstout + inp, lstout - inp)
                   else (lstout,       inp)
    lstout       = vlast lst
```

Describing the PFB

FFT BF2I: Haskell → CλaSH

$$
\begin{array}{l}
\mathit{bf2i}\ (\mathit{cntr}, \mathit{lst})\ \mathit{inp} = ((\mathit{cntr}', \mathit{lst}'), \mathit{out})\\
\quad \textbf{where}\\
\qquad \boxed{n \qquad\qquad = \mathit{length}\ \mathit{lst}}\\
\qquad \mathit{cntr}' \qquad\quad = (\mathit{cntr} + 1)\ \text{`}\mathit{mod}\text{`}\ n\\
\qquad \mathit{lst}' \qquad\quad\ \ = \mathit{lstin} \ \mathbin{+\!\!\gg}\ \mathit{lst}\\
\qquad (\mathit{out}, \mathit{lstin}) = \textbf{if}\ \mathit{cntr} \geqslant n\\
\qquad\qquad\qquad\qquad \textbf{then}\ (\mathit{lstout} + \mathit{inp}, \mathit{lstout} - \mathit{inp})\\
\qquad\qquad\qquad\qquad \textbf{else}\ \ (\mathit{lstout}\qquad , \mathit{inp}\qquad\qquad)\\
\qquad \boxed{\mathit{lstout} \qquad = \mathit{last}\ \mathit{lst}}
\end{array}
$$

$$
\begin{array}{l}
\mathit{bf2i\_clash}\ (\mathit{cntr}, \mathit{lst})\ \mathit{inp} = ((\mathit{cntr}', \mathit{lst}'), \mathit{out})\\
\quad \textbf{where}\\
\qquad \boxed{n \qquad\qquad = \mathit{vlength}\ \mathit{lst}}\\
\qquad \mathit{cntr}' \qquad\quad = \mathit{cntr} + 1\\
\qquad \mathit{lst}' \qquad\quad\ \ = \mathit{lstin} \ \mathbin{+\!\!\gg}\ \mathit{lst}\\
\qquad (\mathit{out}, \mathit{lstin}) = \textbf{if}\ \mathit{cntr} \geqslant n\\
\qquad\qquad\qquad\qquad \textbf{then}\ (\mathit{lstout} + \mathit{inp}, \mathit{lstout} - \mathit{inp})\\
\qquad\qquad\qquad\qquad \textbf{else}\ \ (\mathit{lstout}, \qquad \mathit{inp})\\
\qquad \boxed{\mathit{lstout} \qquad = \mathit{vlast}\ \mathit{lst}}
\end{array}
$$

Describing the PFB

FFT BF2I: Haskell → CλaSH

$$
\begin{aligned}
&\mathit{bf2i}\ (\mathit{cntr}, \mathit{lst})\ \mathit{inp} = ((\mathit{cntr}', \mathit{lst}'), \mathit{out})\\
&\quad \textbf{where}\\
&\qquad n \qquad\qquad\ = \mathit{length}\ \mathit{lst}\\
&\qquad \mathit{cntr}' \qquad\quad = (\mathit{cntr} + 1)\ \text{`}\mathit{mod}\text{`}\ n\\
&\qquad \mathit{lst}' \qquad\quad\ \ = \mathit{lstin}\ +\!\!\gg \mathit{lst}\\
&\qquad (\mathit{out}, \mathit{lstin}) = \textbf{if}\ \mathit{cntr} \geqslant n\\
&\qquad\qquad\qquad\qquad \textbf{then}\ (\mathit{lstout} + \mathit{inp}, \mathit{lstout} - \mathit{inp})\\
&\qquad\qquad\qquad\qquad \textbf{else}\ \ (\mathit{lstout} \qquad\quad, \mathit{inp} \qquad\qquad\ )\\
&\qquad \mathit{lstout} \qquad\ \ = \mathit{last}\ \mathit{lst}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{bf2i\_clash}\ (\mathit{cntr}, \mathit{lst})\ \mathit{inp} = ((\mathit{cntr}', \mathit{lst}'), \mathit{out})\\
&\quad \textbf{where}\\
&\qquad n \qquad\qquad\ = \mathit{vlength}\ \mathit{lst}\\
&\qquad \mathit{cntr}' \qquad\quad = \mathit{cntr} + 1\\
&\qquad \mathit{lst}' \qquad\quad\ \ = \mathit{lstin}\ +\!\!\gg \mathit{lst}\\
&\qquad (\mathit{out}, \mathit{lstin}) = \textbf{if}\ \mathit{cntr} \geqslant n\\
&\qquad\qquad\qquad\qquad \textbf{then}\ (\mathit{lstout} + \mathit{inp}, \mathit{lstout} - \mathit{inp})\\
&\qquad\qquad\qquad\qquad \textbf{else}\ \ (\mathit{lstout}, \qquad\quad \mathit{inp})\\
&\qquad \mathit{lstout} \qquad\ \ = \mathit{vlast}\ \mathit{lst}
\end{aligned}
$$

Results

- Polyphase filter bank has been fully implemented using CλaSH
- Simulation shows that the PFB operates correctly
- Synthesis revealed some limitations of the current compiler

	Polyphase filter(256 elements)	1k-points FFT
Logic utilization	91%	6%
blockRAMS	0	0
DSP blocks	128	70
Max. F_{clk}	114 MHz	195 MHz

Conclusions

- The complete Polyphase Filter Bank has been implemented
- Haskell code needs only small modifications before it is accepted by the CλaSH compiler
- The description is purely parallel (structural) and cycle accurate
- Shortcomings of CλaSH compiler
  - Large coefficient vectors not supported
  - BlockRAM not supported, limiting F_{clk}

Future Work

- Develop area vs time time trade off based on functional description
- Improvements for the CλaSH compiler
  - Support for blockRAMs on FPGA
  - Support for memory initialization files for coefficient vectors

Questions ?