Hardware Specification with CλaSH

Jan Kuper

University of Twente

DSL2013, Cluj, Romania

July 13, 2013

No need to say again

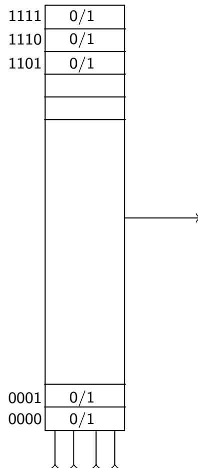
- Mainstream single-core developments have (more or less) stopped
- Heading for multi-core, heterogeneous, parallel, . . .
- A **lot** is going on in hardware land, there even may not be a new "mainstream"
- Reconfigurability
- Design for different platforms at the same time
- Hardware/software co-design
- We have to learn to *think parallel* – maybe: *think in terms of structure*

CλaSH

CλaSH: **C**AES **λa**nguage for **S**ynchronous **H**ardware

- Subset of Haskell
- Higher order functions, pattern matching, polymorphism, type derivation,
- Clear semantics
- Prototype, under development, developed at University of Twente
- Example implementations: reduction circuit, RISC architecture, (network of) dataflow processors, DES encryption algorithm, particle filter
- Lava, ForSyDe, BlueSpec
- People: Christiaan Baaij, Matthijs Kooijman, Rinse Wester, Marco Gerards, Arjan Boeijink, Kenneth Rovers, Anja Niedermeier

FPGA



Dot product

$$
\begin{aligned}
\vec{x} \bullet \vec{y} &= \sum_{i=0}^{n-1} x_i y_i \\
&= x_0 y_0 + x_1 y_1 + \cdots + x_{n-1} y_{n-1}
\end{aligned}
$$

Standard approach:

- sequentialization, for-loop
- parallelization afterwards

Our approach:

- mathematical, higher order functions
- exploit parallelism from the beginning

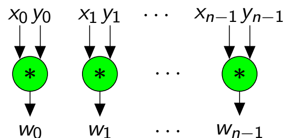
Dot product

$$\vec{x} \bullet \vec{y} \;=\; \sum_{i=0}^{n-1} x_i y_i \;=\; x_0 y_0 + x_1 y_1 + \cdots + x_{n-1} y_{n-1}$$

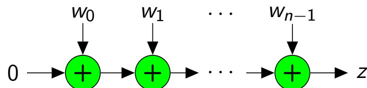
Dot product

$$\vec{x} \bullet \vec{y} \;=\; \sum_{i=0}^{n-1} x_i y_i \;=\; x_0 y_0 + x_1 y_1 + \cdots + x_{n-1} y_{n-1}$$

$$\vec{w} \;=\; \vec{x} \;\widehat{*}\; \vec{y}$$



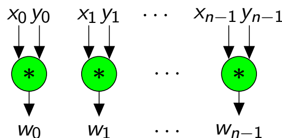
$$z \;=\; 0 \oplus \vec{w}$$



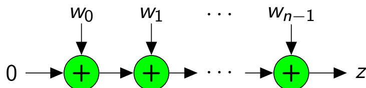
Dot product

$$\vec{x} \bullet \vec{y} = \sum_{i=0}^{n-1} x_i y_i = x_0 y_0 + x_1 y_1 + \cdots + x_{n-1} y_{n-1}$$

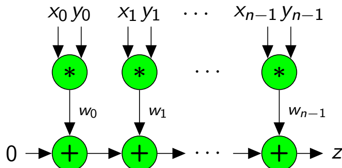
$$\vec{w} = \vec{x} \mathbin{\widehat{*}} \vec{y}$$



$$z = 0 \oplus \vec{w}$$



$$\vec{x} \bullet \vec{y} = 0 \oplus (\vec{x} \mathbin{\widehat{*}} \vec{y})$$

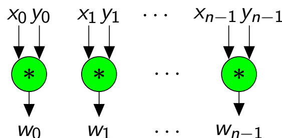


Dot product

$$\vec{x} \bullet \vec{y} = \sum_{i=0}^{n-1} x_i y_i = x_0 y_0 + x_1 y_1 + \cdots + x_{n-1} y_{n-1}$$

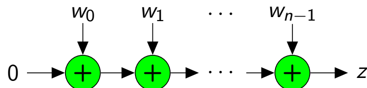
$$\vec{w} = \vec{x} \mathbin{\widehat{*}} \vec{y}$$

$$ws = zipWith\ (*)\ xs\ ys$$



$$z = 0 \bigoplus \vec{w}$$

$$z = foldl\ (+)\ 0\ ws$$

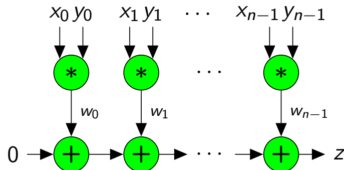


$$\vec{x} \bullet \vec{y} = 0 \bigoplus (\vec{x} \mathbin{\widehat{*}} \vec{y})$$

$$xs <\!\!*\!\!> ys = foldl\ (+)\ 0\ ws$$

where

$$ws = zipWith\ (*)\ xs\ ys$$



Functional notation

	Standard math	Functional language
No brackets:	$f(x)$	$f\ x$
Currying:	$f(x,y)$	$f(x,y);\quad f\ x\ y$

$\Rightarrow$ higher order from the very beginning

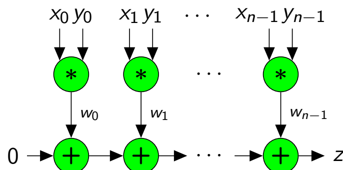
$\Rightarrow$ functions as first class citizens

Functional notation

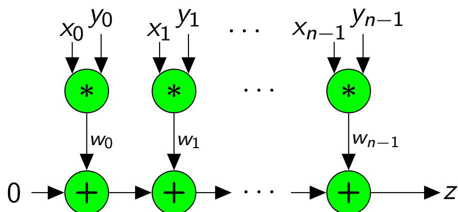
	Standard math	Functional language
No brackets:	$f(x)$	$f\ x$
Currying:	$f(x,y)$	$f(x,y);\quad f\ x\ y$

$\Rightarrow$ higher order from the very beginning
$\Rightarrow$ functions as first class citizens

$$
\begin{aligned}
xs <\!\!*\!\!> ys &= foldl\ (+)\ 0\ ws \\
&\quad \textbf{where} \\
&\quad ws = zipWith\ (*)\ xs\ ys
\end{aligned}
$$



Dot product



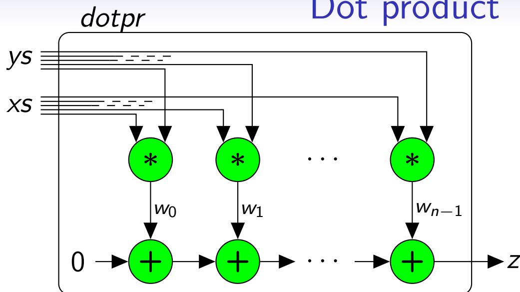
$$z = xs \circledast ys$$

$$= foldl\ (+)\ 0\ ws$$

where

$$ws = zipWith\ (*)\ xs\ ys$$

Dot product

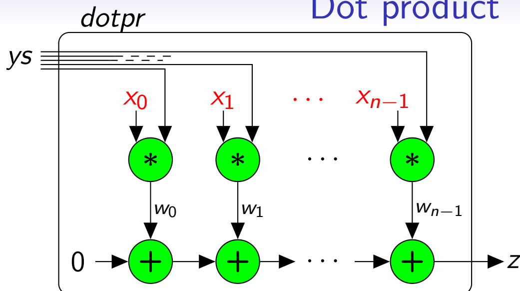


$$dotpr\ (xs, ys) = z$$

where

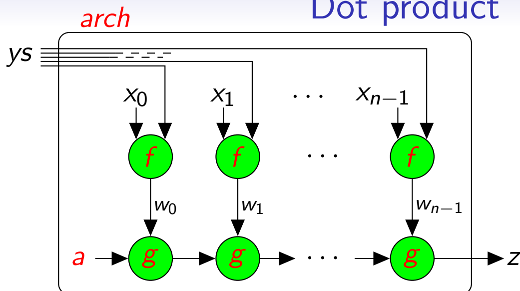
$$\begin{aligned} ws &= zipWith\ (*)\ xs\ ys \\ z &= foldl\ (+)\ 0\ ws \end{aligned}$$

Dot product



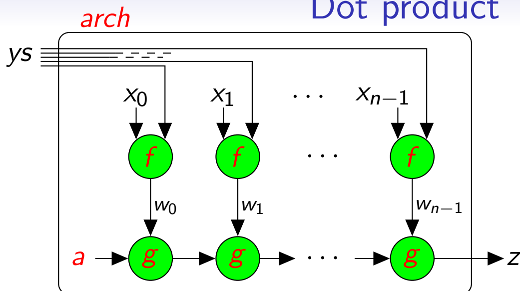
$$
\begin{aligned}
&dotpr\ xs\ ys = z \\
&\quad \textbf{where} \\
&\qquad ws = zipWith\ (*)\ xs\ ys \\
&\qquad z = foldl\ (+)\ 0\ ws
\end{aligned}
$$

Dot product



$$
\begin{array}{l}
arch\ (f, g, a, xs)\ ys\ = z \\
\quad\quad \textbf{where} \\
\quad\quad\quad ws = zipWith\ f\ xs\ ys \\
\quad\quad\quad z = foldl\ g\ a\ ws
\end{array}
$$

Dot product



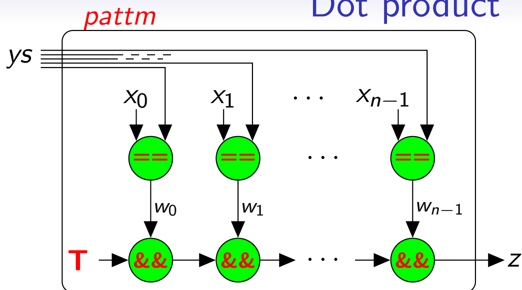
$$arch\ (f, g, a, xs)\ ys\ = z$$

where

$$ws = zipWith\ f\ xs\ ys$$
$$z = foldl\ g\ a\ ws$$

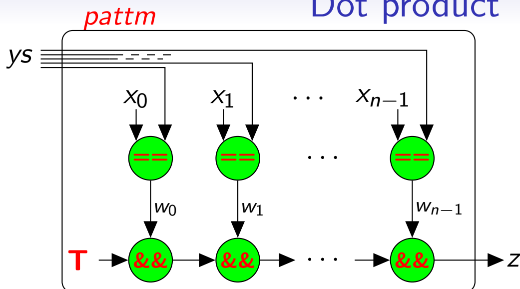
$$dotpr\ xs\ =\ arch\ ((*), (+), 0, xs)$$

Dot product



$$pattm\ xs \;=\; arch\ ((==), (\&\&), \mathbf{T}, xs)$$

Dot product



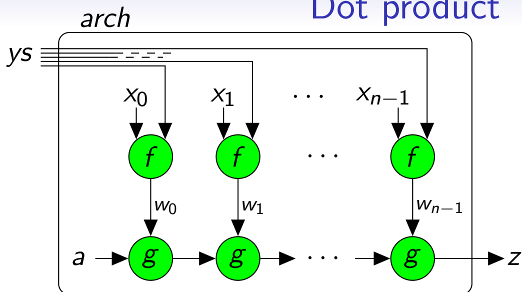
$$pattm\ xs \;=\; arch\ ((==), (\&\&), \mathbf{T}, xs)$$

$$pattm\ xs\ ys \;=\; z$$

where

$$ws \;=\; zipWith\ (==)\ xs\ ys$$
$$z \;=\; foldl\ (\&\&)\ \mathbf{T}\ ws$$

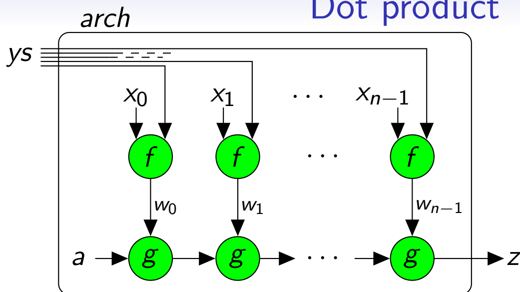
Dot product



$$arch\ (f, g, a, xs)\ ys\ = z \quad \textbf{where}\ \cdots$$

$$arch :: \cdots \ \Rightarrow$$
$$(\cdots, \cdots, \cdots, \cdots) \rightarrow \cdots \rightarrow \cdots$$

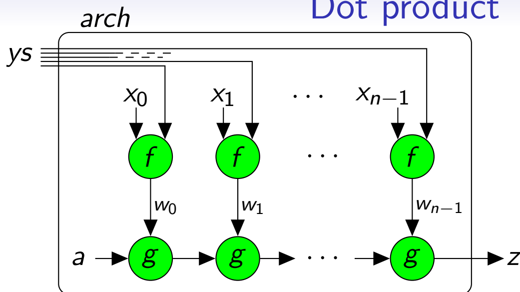
Dot product



$$arch\ (f, g, a, xs)\ ys\ = z \quad \textbf{where}\ \cdots$$

$$arch\ ::\ \cdots\ \Rightarrow$$
$$\big((t_0 \to t_1 \to t_2),\ (t_3 \to t_2 \to t_3),\ t_3,\ [t_0]\big)$$
$$\to\ [t_1]\ \to\ t_3$$

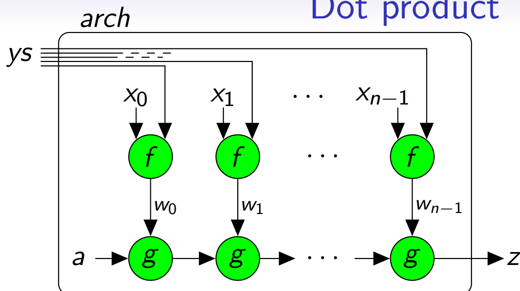
Dot product



$$arch\ (f, g, a, xs)\ ys\ = z \quad \textbf{where}\ \cdots$$

$$arch :: Repr\ t_0, t_1, t_2, t_3 \ \Rightarrow$$
$$\big((t_0 \to t_1 \to t_2),\ (t_3 \to t_2 \to t_3),\ t_3,\ [t_0]\big)$$
$$\to [t_1] \to t_3$$

Dot product



$$arch\ (f, g, a, xs)\ ys\ = z \quad \textbf{where}\ \cdots$$

$$arch :: Repr\ t_0, t_1, t_2, t_3 \Rightarrow \big((t_0 \to t_1 \to t_2), (t_3 \to t_2 \to t_3), t_3, [t_0]\big) \to [t_1] \to t_3$$

$$pattm :: Repr\ t \Rightarrow ((t \to t \to B), (B \to B \to B), B, [t]) \to [t] \to B$$

Some observations

- Polymorphism
- Type derivation
- Parallelism
- Data-oriented vs architecture oriented: "fold", "zip"
- Curried notation: $f(x,y) \Rightarrow f\ x\ y$; advantage: partial application
- Higher order vs RTL level
- Haskell types vs CλaSH types
- Close to mathematics: $\frown = zipWith, map$
- Functional character of hardware: input values $\Rightarrow$ function/operation $\Rightarrow$ output values

Dot product in CλaSH

Math 1: $\vec{x} \bullet \vec{y} \;=\; \sum_{i=0}^{n-1} x_i y_i$

Math 2: $\vec{x} \bullet \vec{y} = 0 \bigoplus (\vec{x} \;\widehat{*}\; \vec{y})$

Haskell: $(\lt\!\ast\!\gt) \;::\; Num\ a \;\Rightarrow\; [a] \rightarrow [a] \rightarrow a$

$$
\begin{aligned}
xs \lt\!\ast\!\gt ys \;&=\; foldl\ (+)\ 0\ ws \\
&\quad \textbf{where} \\
&\quad ws = \; zipWith\ (*)\ xs\ ys
\end{aligned}
$$

Dot product in CλaSH

Math 1: $\vec{x} \bullet \vec{y} = \sum_{i=0}^{n-1} x_i y_i$

Math 2: $\vec{x} \bullet \vec{y} = 0 \bigoplus (\vec{x} \; \widehat{*} \; \vec{y})$

Haskell: $(\lessdot\!\!*\!\!\gtrdot) \; :: \; Num \; a \; \Rightarrow \; [a] \rightarrow [a] \rightarrow a$

CλaSH:

$$\textbf{type} \; Int16 \; = \; Signed \; D16$$
$$\textbf{type} \; Int32 \; = \; Signed \; D32$$
$$\textbf{type} \; IntVec \; = \; Vector \; D4 \; Int16$$

$$(\lessdot\!\!*\!\!\gtrdot) \; :: \; IntVec \rightarrow IntVec \rightarrow Int32$$

$$xs \lessdot\!\!*\!\!\gtrdot ys \; = \textbf{v}foldl \; (+) \; 0 \; ws$$
$$\textbf{where}$$
$$ws = \textbf{v}zipWith \; (*) \; xs \; ys$$

Matrix-vector multiplication

$$(x_1, x_2, x_3) \bullet \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} \;=\; x_1 y_1 + x_2 y_2 + x_3 y_3$$

$$\vec{x} \bullet \vec{y} \;=\; 0 \bigoplus \left(\vec{x} \mathbin{\widehat{*}} \vec{y}\right)$$

$$xs \mathbin{<\!\!*\!\!>} ys \;=\; foldl\ (+)\ 0\ \big(zipWith\ (*)\ xs\ ys\big)$$

Matrix-vector multiplication

$$(x_1, x_2, x_3) \bullet \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} \quad = \quad x_1 y_1 + x_2 y_2 + x_3 y_3$$

$$\begin{pmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \\ 41 & 42 & 43 \end{pmatrix} \times \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 74 \\ 134 \\ 194 \\ 254 \end{pmatrix}$$

type *Row* = [*Int*]
type *Matrix* = [*Row*]

Matrix-vector multiplication

$$(x_1, x_2, x_3) \bullet \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} \quad = \quad x_1 y_1 + x_2 y_2 + x_3 y_3$$

$$\begin{pmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \\ 41 & 42 & 43 \end{pmatrix} \times \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 74 \\ 134 \\ 194 \\ 254 \end{pmatrix}$$

$$\begin{aligned} \textbf{type}\ Row &= Vector\ D3\ Int16 \\ \textbf{type}\ Matrix &= Vector\ D4\ Row \end{aligned}$$

Matrix-vector multiplication

$$(x_1, x_2, x_3) \bullet \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = x_1 y_1 + x_2 y_2 + x_3 y_3$$

$$\begin{pmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \\ 41 & 42 & 43 \end{pmatrix} \times \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 74 \\ 134 \\ 194 \\ 254 \end{pmatrix}$$

$$\begin{aligned} \textbf{type}\ \mathit{Row} &= \mathit{Vector}\ \mathit{D3}\ \mathit{Int16} \\ \textbf{type}\ \mathit{Matrix} &= \mathit{Vector}\ \mathit{D4}\ \mathit{Row} \end{aligned}$$

Matrix-vector multiplication

$$(x_1, x_2, x_3) \bullet \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = x_1 y_1 + x_2 y_2 + x_3 y_3$$

$$\begin{pmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \\ 41 & 42 & 43 \end{pmatrix} \times \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 74 \\ 134 \\ 194 \\ 254 \end{pmatrix}$$

$$\begin{aligned} \textbf{type } \mathit{Row} &= \mathit{Vector}\ \mathit{D3}\ \mathit{Int16} \\ \textbf{type } \mathit{Matrix} &= \mathit{Vector}\ \mathit{D4}\ \mathit{Row} \end{aligned}$$

Matrix-vector multiplication

$$(x_1, x_2, x_3) \bullet \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = x_1 y_1 + x_2 y_2 + x_3 y_3$$

$$\begin{pmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \\ 41 & 42 & 43 \end{pmatrix} \times \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 74 \\ 134 \\ 194 \\ 254 \end{pmatrix}$$

type *Row* = *Vector D3 Int*16
type *Matrix* = *Vector D4 Row*

Matrix-vector multiplication

$$(x_1, x_2, x_3) \bullet \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = x_1 y_1 + x_2 y_2 + x_3 y_3$$

$$\begin{pmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \\ 41 & 42 & 43 \end{pmatrix} \times \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 74 \\ 134 \\ 194 \\ 254 \end{pmatrix}$$

$$\begin{aligned} \textbf{type}\ Row &= Vector\ D3\ Int16 \\ \textbf{type}\ Matrix &= Vector\ D4\ Row \end{aligned}$$

Matrix-vector multiplication

$$(x_1, x_2, x_3) \bullet \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} \quad = \quad x_1 y_1 + x_2 y_2 + x_3 y_3$$

$$\begin{pmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \\ 41 & 42 & 43 \end{pmatrix} \times \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 74 \\ 134 \\ 194 \\ 254 \end{pmatrix}$$

$$\begin{array}{lcl} \textbf{type } \mathit{Row} & = & \mathit{Vector}\ \mathit{D3}\ \mathit{Int16} \\ \textbf{type } \mathit{Matrix} & = & \mathit{Vector}\ \mathit{D4}\ \mathit{Row} \end{array}$$

$$A \times \vec{y} \;=\; \widehat{\bullet\vec{y}}\;\; A \qquad\qquad (= [\vec{x} \bullet \vec{y} \mid \vec{x} \in A])$$

Matrix-vector multiplication

$$(x_1, x_2, x_3) \bullet \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} \quad = \quad x_1 y_1 + x_2 y_2 + x_3 y_3$$

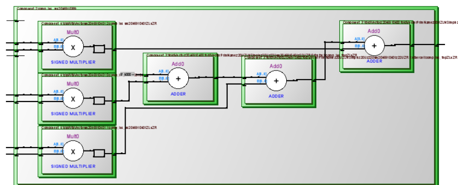
$$\begin{pmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \\ 41 & 42 & 43 \end{pmatrix} \times \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 74 \\ 134 \\ 194 \\ 254 \end{pmatrix}$$

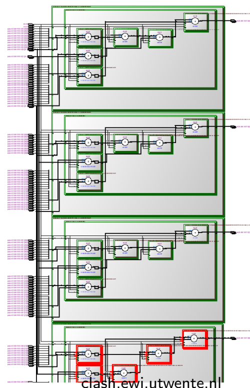
$$\begin{aligned} &\textbf{type } \mathit{Row} &&= \quad \mathit{Vector}\ \mathit{D3}\ \mathit{Int16} \\ &\textbf{type } \mathit{Matrix} &&= \quad \mathit{Vector}\ \mathit{D4}\ \mathit{Row} \end{aligned}$$

$$A \times \vec{y} \;=\; \widehat{\bullet \vec{y}}\;\; A \qquad\qquad (= [\vec{x} \bullet \vec{y} \mid \vec{x} \in A])$$

$$\mathit{xss} \lessdot\!\!**\!\!\gtrdot \mathit{ys} \;=\; \mathit{map}\ (\lessdot\!\!*\!\!\gtrdot \mathit{ys})\ \mathit{xss}$$

Matrix-vector multiplication





Matrix-matrix multiplication

$$\begin{pmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \\ 41 & 42 & 43 \end{pmatrix} \star \begin{pmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{pmatrix} = \begin{pmatrix} 74 & 182 & 290 & 398 \\ 134 & 332 & 530 & 728 \\ 194 & 482 & 770 & 1058 \\ 254 & 632 & 1010 & 1388 \end{pmatrix}$$

Matrix-matrix multiplication

$$\begin{pmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \\ 41 & 42 & 43 \end{pmatrix} \star \begin{pmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{pmatrix} = \begin{pmatrix} 74 & 182 & 290 & 398 \\ 134 & 332 & 530 & 728 \\ 194 & 482 & 770 & 1058 \\ 254 & 632 & 1010 & 1388 \end{pmatrix}$$

Matrix-matrix multiplication

$$\begin{pmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \\ 41 & 42 & 43 \end{pmatrix} \star \begin{pmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{pmatrix} = \begin{pmatrix} 74 & 182 & 290 & 398 \\ 134 & 332 & 530 & 728 \\ 194 & 482 & 770 & 1058 \\ 254 & 632 & 1010 & 1388 \end{pmatrix}$$

Matrix-matrix multiplication

$$\begin{pmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \\ 41 & 42 & 43 \end{pmatrix} \star \begin{pmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{pmatrix} = \begin{pmatrix} 74 & 182 & 290 & 398 \\ 134 & 332 & 530 & 728 \\ 194 & 482 & 770 & 1058 \\ 254 & 632 & 1010 & 1388 \end{pmatrix}$$

Matrix-matrix multiplication

$$\begin{pmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \\ 41 & 42 & 43 \end{pmatrix} \star \begin{pmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{pmatrix} = \begin{pmatrix} 74 & 182 & 290 & 398 \\ 134 & 332 & 530 & 728 \\ 194 & 482 & 770 & 1058 \\ 254 & 632 & 1010 & 1388 \end{pmatrix}$$

Matrix-matrix multiplication

$$\begin{pmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \\ 41 & 42 & 43 \end{pmatrix} \star \begin{pmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{pmatrix} = \begin{pmatrix} 74 & 182 & 290 & 398 \\ 134 & 332 & 530 & 728 \\ 194 & 482 & 770 & 1058 \\ 254 & 632 & 1010 & 1388 \end{pmatrix}$$

$$A \star B = (\widehat{A\times}\ B^T)^T$$

Matrix-matrix multiplication

$$\begin{pmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \\ 41 & 42 & 43 \end{pmatrix} \star \begin{pmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{pmatrix} = \begin{pmatrix} 74 & 182 & 290 & 398 \\ 134 & 332 & 530 & 728 \\ 194 & 482 & 770 & 1058 \\ 254 & 632 & 1010 & 1388 \end{pmatrix}$$

$$A \star B = (\widehat{A\times}\; B^T)^T$$

$$xss \;\lessdot\!\ast\ast\ast\!\gtrdot\; yss = transpose\ (map\ (xss \lessdot\!\ast\ast\!\gtrdot)\ (transpose\ yss))$$

Evaluation

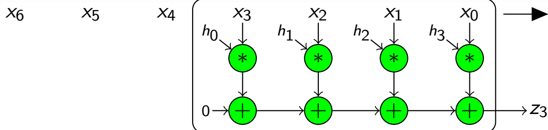
- some Haskell specifics: curried functions, where clause inherently parallel; polymorphism; higher order
- $\sum$ suggests sequentialized for-loop;
  alternative suggests structural view;

Convolution – FIR filter

x_6 x_5 x_4 x_3 x_2 x_1 x_0 $\longrightarrow$

xs: stream of samples

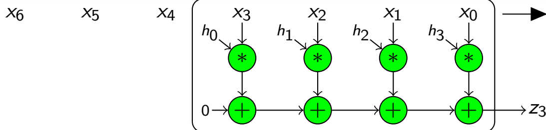
Convolution – FIR filter



xs: stream of samples

hs: filter coefficients

Convolution – FIR filter



xs: stream of samples

hs: filter coefficients

$$z3 = hs \lessdot\!\!\ast\!\!\gtrdot (reverse\ (take\ n\ xs))$$

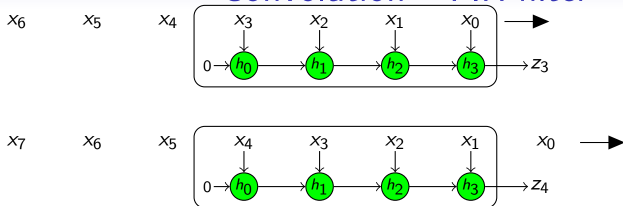
$$z3 = foldl\ (+)\ 0\ ws$$

where

$$\begin{aligned} n &= length\ hs \\ ws &= zipWith\ (*)\ hs\ (reverse\ (take\ n\ xs)) \end{aligned}$$

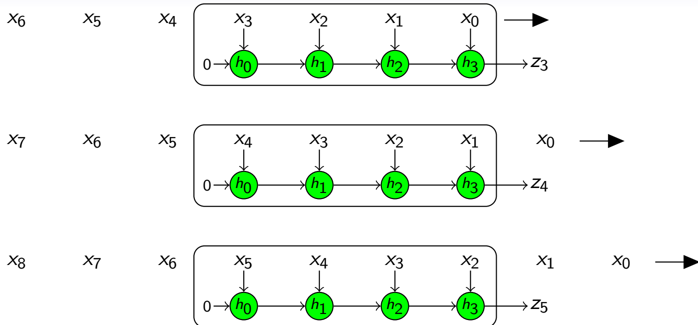
Convolution – FIR filter

Convolution – FIR filter



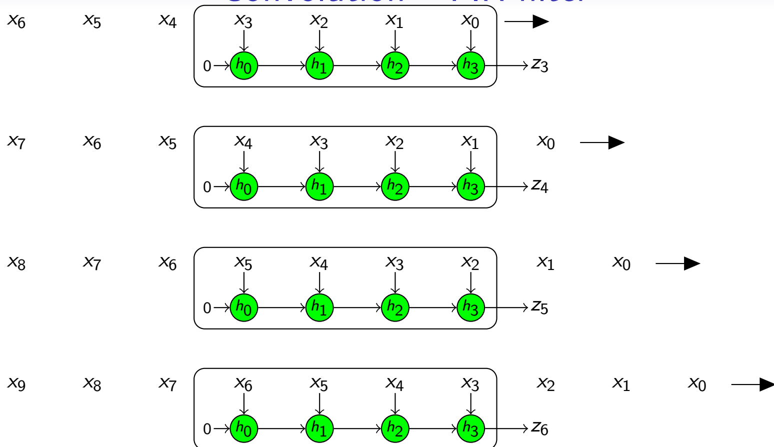
$$z4 = hs \circledast (reverse\ (take\ n\ (drop\ 1\ xs)))$$

Convolution – FIR filter



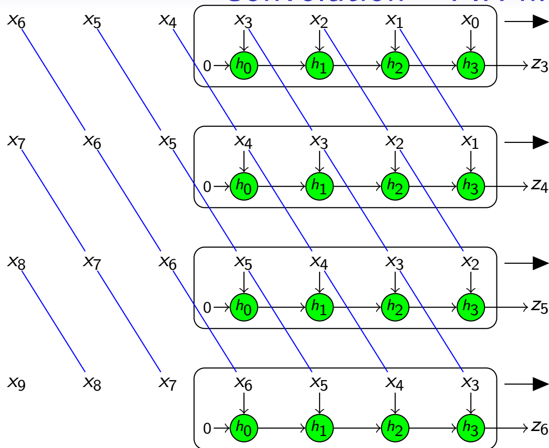
$$z5 = hs \lt\!\!\circledast\!\!\gt (reverse\ (take\ n\ (drop\ 2\ xs)))$$

Convolution – FIR filter



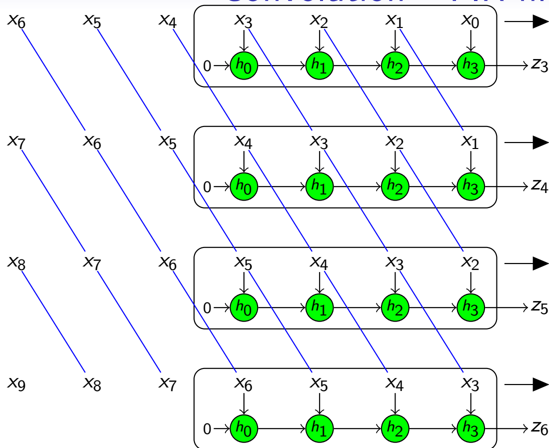
$$z6 = \cdots$$

Convolution – FIR filter



$$
\begin{aligned}
&\mathit{fir}\ \mathit{hs}\ \mathit{xs} = z : \mathit{fir}\ \mathit{hs}\ (\mathit{tail}\ \mathit{xs}) \\
&\quad \textbf{where} \\
&\quad\quad n = \mathit{length}\ \mathit{hs} \\
&\quad\quad z = \mathit{hs} \circledast (\mathit{reverse}\ (\mathit{take}\ n\ \mathit{xs}))
\end{aligned}
$$

Convolution – FIR filter

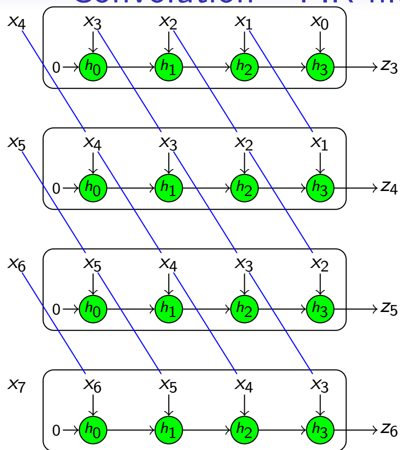


$$fir\ hs\ us\ (x{:}xs) = z : fir\ hs\ us'\ xs$$

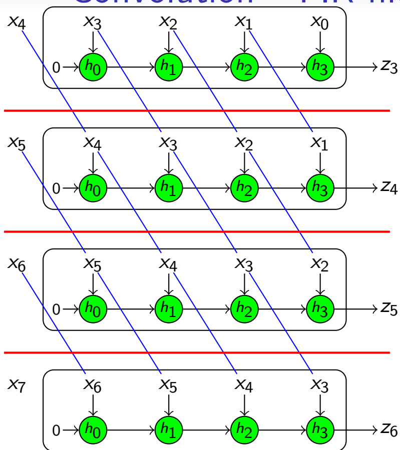
where

$$z = hs <\!\!*\!\!> us$$
$$us' = x : init\ us$$

Convolution – FIR filter

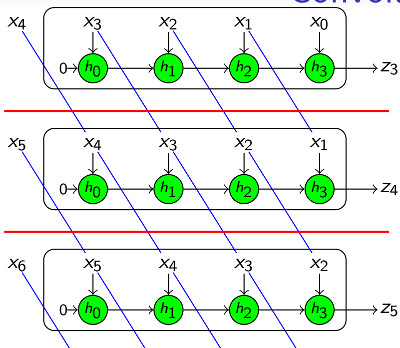


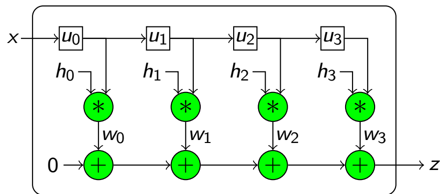
Convolution – FIR filter



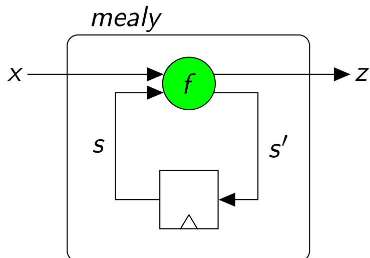
Convolution – FIR filter

Convolution – FIR filter

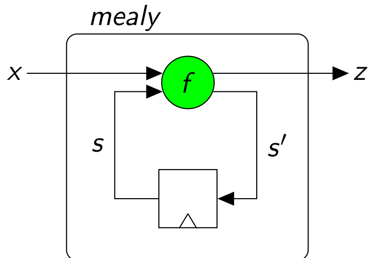




Mealy Machine



Mealy Machine



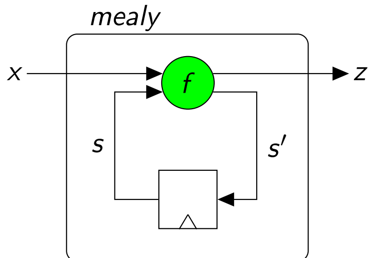
$$f\ (State\ s)\ x = (State\ s',\ z)$$

where

$$s' = \cdots$$

$$z = \cdots$$

Mealy Machine



$$f\ (State\ s)\ x\ =\ (State\ s',\ z)$$
where
$$s' = \cdots$$
$$z = \cdots$$

$$sim\ f\ s\ (x{:}xs)\ =\ z : sim\ f\ s'\ xs$$
where
$$(s', z) = f\ s\ x$$

Multiply-accumulate

Reference definition of *mac* over a sequence of (x, y)-pairs:

$$
\begin{aligned}
mac_0 \;\; s \; ((x,y) : xys) \;\; &= s' : mac_0 \; s' \; xys \\
&\quad \textbf{where} \\
&\quad s' = s + x * y
\end{aligned}
$$

Multiply-accumulate

Reference definition of *mac* over a sequence of (x, y)-pairs:

$$mac_0 \;\; s \; ((x,y) : xys) \;\; = s' : mac_0 \; s' \; xys$$
$$\textbf{where}$$
$$s' = s + x * y$$

$$test_0 \;\; = mac_0 \; 0 \; [(1,1),(2,2),(3,3),\cdots]$$

Multiply-accumulate

Reference definition of *mac* over a sequence of (x, y)-pairs:

$$mac_0 \;\; s \; ((x,y) : xys) \;\; = s' : mac_0 \; s' \; xys$$

$$\textbf{where}$$

$$s' = s + x * y$$

$$\begin{aligned} test_0 \;\; &= mac_0 \; 0 \; [(1,1),(2,2),(3,3),\cdots] \\ &= 1 : mac_0 \; 1 \; [(2,2),(3,3),\cdots] \end{aligned}$$

Multiply-accumulate

Reference definition of *mac* over a sequence of (x, y)-pairs:

$$mac_0 \;\; s \; ((x,y) : xys) \;\; = s' : mac_0 \; s' \; xys$$

where

$$s' = s + x * y$$

$$\begin{aligned} test_0 \;\; &= mac_0 \; 0 \; [(1,1),(2,2),(3,3),\cdots] \\ &= 1 : mac_0 \; 1 \; [(2,2),(3,3),\cdots] \\ &= 1 : 5 : mac_0 \; 5 \; [(3,3),\cdots] \end{aligned}$$

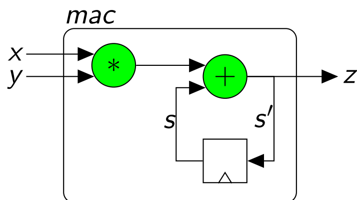
Multiply-accumulate

Reference definition of *mac* over a sequence of (x, y)-pairs:

$$
\begin{aligned}
mac_0\ \ s\ ((x,y) : xys)\ &= s' : mac_0\ s'\ xys \\
&\quad \textbf{where} \\
&\quad s' = s + x * y
\end{aligned}
$$

$$
\begin{aligned}
test_0\ &= mac_0\ 0\ [(1,1),(2,2),(3,3),\cdots] \\
&= 1 : mac_0\ 1\ [(2,2),(3,3),\cdots] \\
&= 1 : 5 : mac_0\ 5\ [(3,3),\cdots] \\
&= 1 : 5 : 14 : mac_0\ 14\ [\cdots]
\end{aligned}
$$

Multiply-accumulate

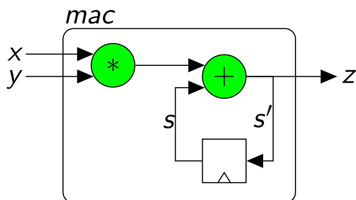


$$mac\ (State\ s)\ (x,y) \ = \ (State\ s', z)$$

where

$$s' = s + x * y$$
$$z = s'$$

Multiply-accumulate

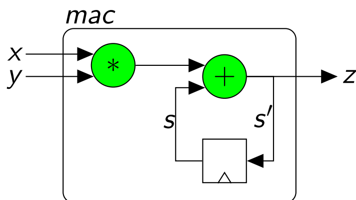


$$mac\ (State\ s)\ (x,y) = (State\ s', z)$$
where
$$s' = s + x * y$$
$$z = s'$$

Compare mac_0:

$$mac_0\ s\ ((x,y) : xys) = s' : mac_0\ s'\ xys$$
where
$$s' = s + x * y$$

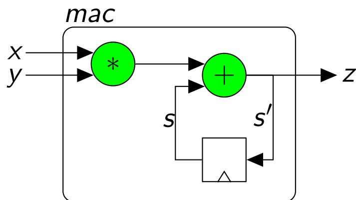
Multiply-accumulate



$$mac\ (State\ s)\ (x,y) = (State\ s',\ z)$$
where
$$s' = s + x * y$$
$$z = s'$$

$$test = sim\ mac\ (State\ 0)\ [(1,1),\ (2,2),\ (3,3),\ldots]$$

Multiply-accumulate



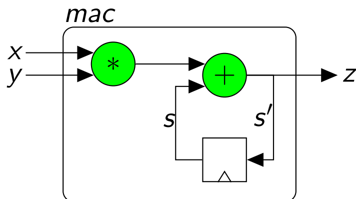
$$mac\ (State\ s)\ (x,y) = (State\ s',\ z)$$

where

$$s' = s + x * y$$
$$z = s'$$

$$test = sim\ mac\ (State\ 0)\ [(1,1),\ (2,2),\ (3,3),\ldots]$$
$$= 1 : sim\ mac\ (State\ 1)\ [(2,2),\ (3,3),\ldots]$$

Multiply-accumulate



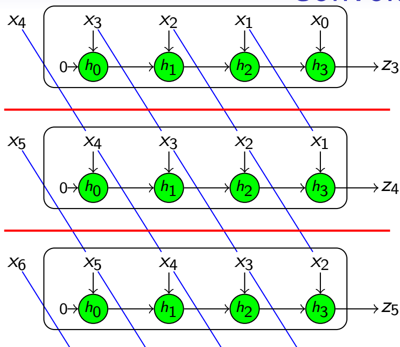
$$mac\ (State\ s)\ (x,y) \;=\; (State\ s',\ z)$$

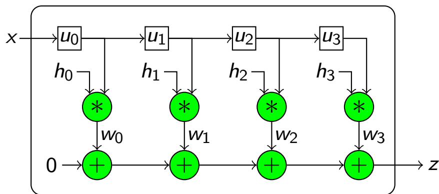
where

$$s' = s + x * y$$
$$z = s'$$

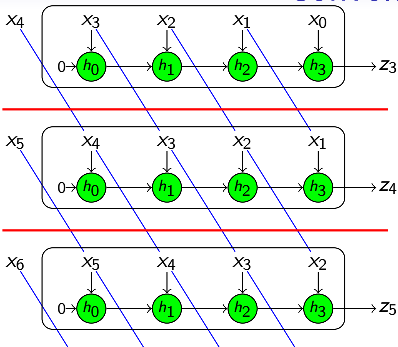
$$\begin{aligned} test &= sim\ mac\ (State\ 0)\ [(1,1),\ (2,2),\ (3,3),\ldots] \\ &= 1 : sim\ mac\ (State\ 1)\ [(2,2),\ (3,3),\ldots] \\ &= 1 : 5 : sim\ mac\ (State\ 5)\ [(3,3),\ldots] \end{aligned}$$

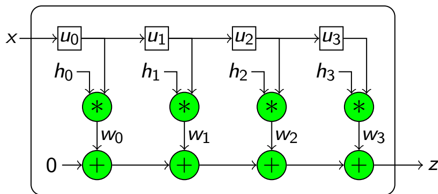
Convolution – FIR filter





Convolution – FIR filter





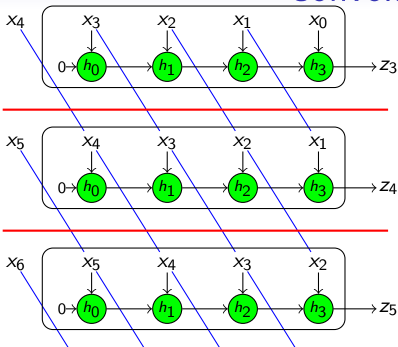
$$fir1\ hs\ (State\ us)\ x = (State\ us', z)$$

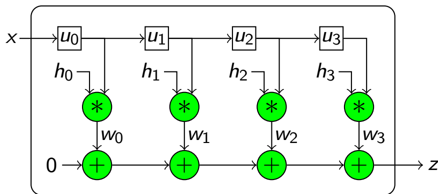
where

$$us' = x +\!\!\gg us$$
$$z = hs \lessdot\!\ast\!\gtrdot us$$

;

Convolution – FIR filter





$$fir1\ hs\ (State\ us)\ x = (State\ us', z)$$

where

$$\begin{aligned} us' &= x +\gg us \\ ws &= zipWith\ (*)\ hs\ us \\ z &= foldl\ (+)\ 0\ ws \end{aligned}$$

;

FIR-filter: VHDL-code

$$
\begin{aligned}
&fir :: IntVec \to (State\ IntVec) \to Int16 \\
&\qquad\qquad \to (State\ IntVec, Int32) \\
&fir\ hs\ (State\ us)\ x = (State\ us', z) \\
&\qquad\quad \textbf{where} \\
&\qquad\quad\ ws \;=\; zipWith\ (*)\ hs\ us \\
&\qquad\quad\ z \;=\; foldl\ (+)\ 0\ ws \\
&\qquad\quad\ us' \;=\; x +\!\!\gg us
\end{aligned}
$$

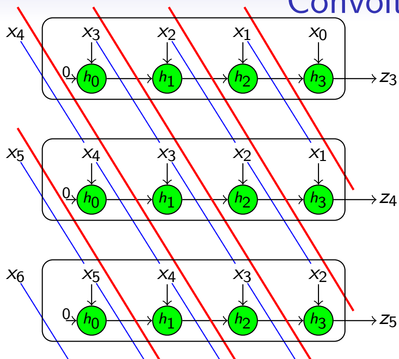
FIR-filter: VHDL-code

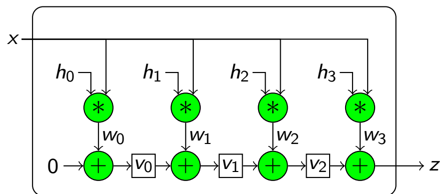
```
entity FIR is
  generic (hs : data_array := (3, 2, 5, 8));
  port (clk : in std_logic; x: in integer; z: out integer);
end FIR;

architecture reg_top of FIR is
  signal us: data_array(0 to hs'length − 1);
begin
  us ⇐ x & us(0 to hs'length − 2)   when rising_edge(clk);
  mult_add: process(us)
    variable tmp: integer;
   begin
    tmp := 0;
    for i in us'range loop
     tmp := tmp + hs(i) ∗ us(i);
    end loop;
    z ⇐ tmp;
   end process;
end reg_top;
```

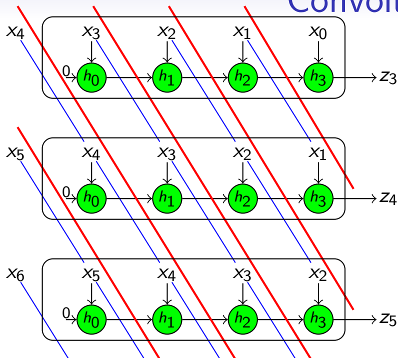
```
fir :: IntVec → (State IntVec) → Int16
            → (State IntVec, Int32)
fir hs (State us) x = (State us', z)
        where
          ws   =  zipWith (∗) hs us
          z    =  foldl (+) 0 ws
          us'  =  x +>> us
```

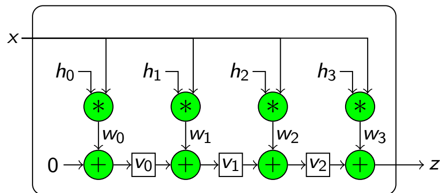
Convolution – FIR filter





Convolution – FIR filter





$$fir2\ hs\ (State\ vs)\ x = (State\ vs', z)$$

where

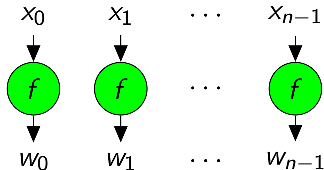
$$\begin{aligned} ws &= map\ (*x)\ hs \\ vs'' &= zipWith\ (+)\ (0 : vs)\ ws \\ vs' &= init\ vs'' \\ z &= last\ vs'' \end{aligned}$$

;

Map

$$\vec{w} = \widehat{f}\ (\vec{x})$$

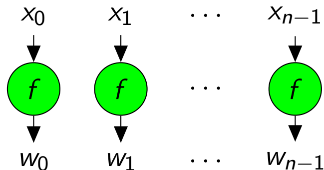
$$ws = map\ f\ xs$$



Map

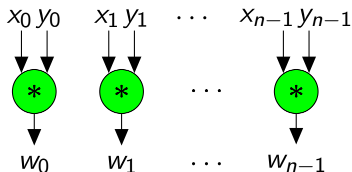
$$\vec{w} = \widehat{f}\ (\vec{x})$$

$$ws = map\ f\ xs$$

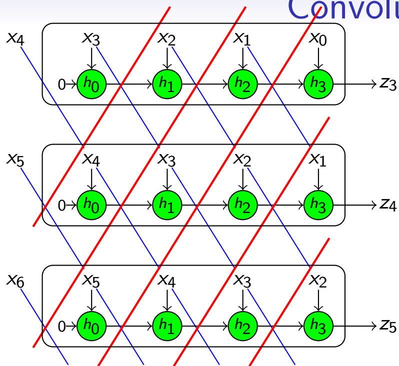


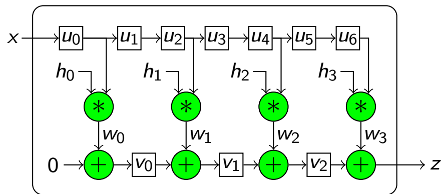
$$\vec{w} = \vec{x}\ \widehat{*}\ \vec{y}$$

$$ws = zipWith\ (*)\ xs\ ys$$

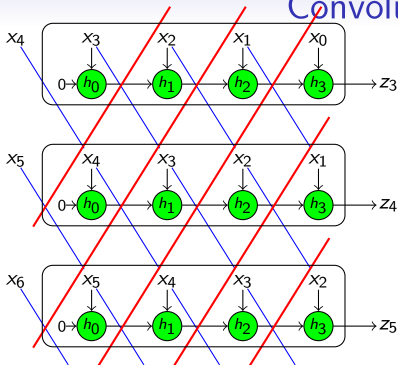


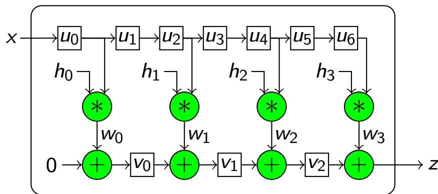
Convolution – FIR filter





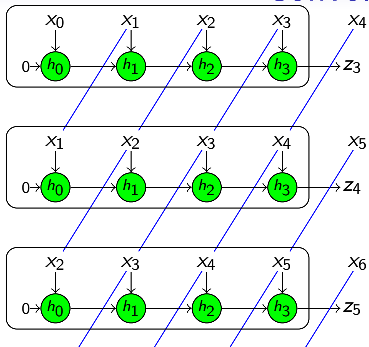
Convolution – FIR filter



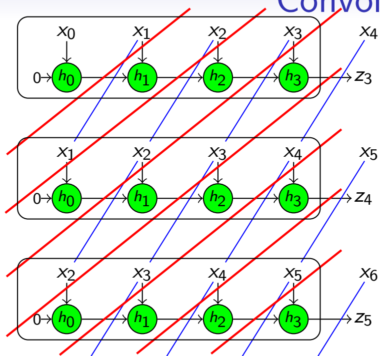


$$
\begin{aligned}
&\mathit{fir3}\ \mathit{hs}\ (\mathit{State}\ (\mathit{us}, \mathit{vs}))\ x \quad = \quad (\mathit{State}\ (\mathit{us}', \mathit{vs}'), z) \\
&\qquad \textbf{where} \\
&\qquad\quad \mathit{ws} \quad = \quad \mathit{zipWith}\ (*)\ \mathit{hs}\ (\mathit{us}!!![0, 2..]) \\
&\qquad\quad \mathit{vs}'' \quad = \quad \mathit{zipWith}\ (+)\ (0 : \mathit{vs})\ \mathit{ws} \\
&\qquad\quad (\mathit{us}', \mathit{vs}') \quad = \quad (x +\!\!\gg \mathit{us},\ \mathit{init}\ \mathit{vs}'') \\
&; \qquad\quad z \quad = \quad \mathit{last}\ \mathit{vs}''
\end{aligned}
$$

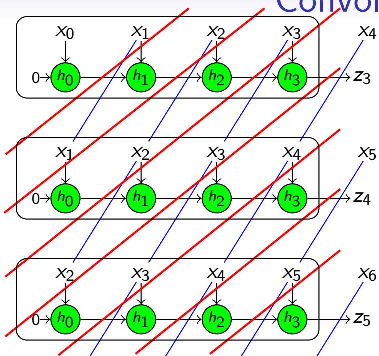
Convolution – FIR filter

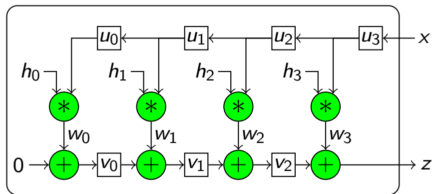


Convolution – FIR filter

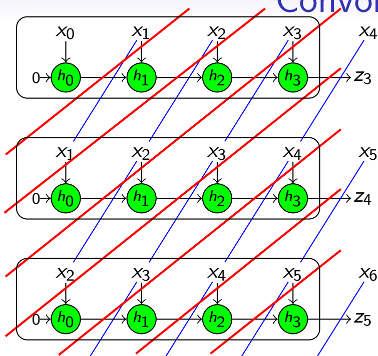


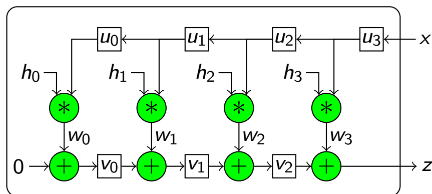
Convolution – FIR filter





Convolution – FIR filter





$$
\begin{array}{lcl}
\mathit{fir4}\ \mathit{hs}\ (\mathit{State}\ (\mathit{us},\mathit{vs}))\ x & = & (\mathit{State}\ (\mathit{us}',\mathit{vs}'), z)
\end{array}
$$

where

$$
\begin{array}{lcl}
\mathit{ws} & = & \mathit{zipWith}\ (*)\ \mathit{hs}\ \mathit{us} \\
\mathit{vs}'' & = & \mathit{zipWith}\ (+)\ (0 : \mathit{vs})\ \mathit{ws} \\
(\mathit{us}',\mathit{vs}') & = & (\mathit{us} \lll\!+ x,\ \mathit{init}\ \mathit{vs}'') \\
; \quad z & = & \mathit{last}\ \mathit{vs}''
\end{array}
$$

Arithmetic

Some basic arithmetical operations:

- Addition: half adder, full adder, ripple carry adder
- Multiplication: naive, Baugh-Wooley

Logic Gates

$\wedge$	0	1
0	0	0
1	0	1

$\vee$	0	1
0	0	1
1	1	1

$\otimes$	0	1
0	0	1
1	1	0

Logic Gates

$\wedge$	0	1
0	0	0
1	0	1

$$\begin{aligned} 0 \wedge 0 &= 0 \\ 0 \wedge 1 &= 0 \\ 1 \wedge 0 &= 0 \\ 1 \wedge 1 &= 1 \end{aligned}$$

$\vee$	0	1
0	0	1
1	1	1

$$\begin{aligned} 0 \vee 0 &= 0 \\ 0 \vee 1 &= 1 \\ 1 \vee 0 &= 1 \\ 1 \vee 1 &= 1 \end{aligned}$$

$\otimes$	0	1
0	0	1
1	1	0

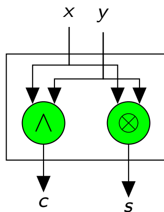
$$\begin{aligned} 0 \otimes 0 &= 0 \\ 0 \otimes 1 &= 1 \\ 1 \otimes 0 &= 1 \\ 1 \otimes 1 &= 0 \end{aligned}$$

Half Adder

x	y	(c', s)
0	0	$(0,0)$
0	1	$(0,1)$
1	0	$(0,1)$
1	1	$(1,0)$

Half Adder

x	y	(c', s)
0	0	$(0, 0)$
0	1	$(0, 1)$
1	0	$(0, 1)$
1	1	$(1, 0)$

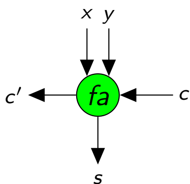


$$ha\ (x, y) \;=\; (c, s)$$

where

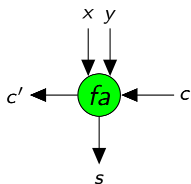
$$c = x \wedge y$$
$$s = x \otimes y$$

Full Adder

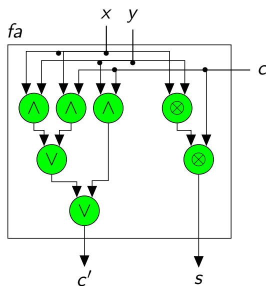


x	y	c	(c', s)
0	0	0	$(0, 0)$
0	0	1	$(0, 1)$
0	1	0	$(0, 1)$
0	1	1	$(1, 0)$
1	0	0	$(0, 1)$
1	0	1	$(1, 0)$
1	1	0	$(1, 0)$
1	1	1	$(1, 1)$

Full Adder



x	y	c	(c', s)
0	0	0	$(0, 0)$
0	0	1	$(0, 1)$
0	1	0	$(0, 1)$
0	1	1	$(1, 0)$
1	0	0	$(0, 1)$
1	0	1	$(1, 0)$
1	1	0	$(1, 0)$
1	1	1	$(1, 1)$

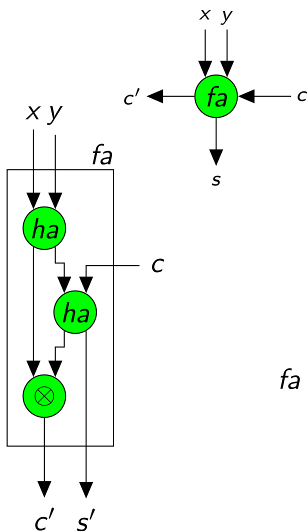


$$fa\ c\ (x, y) = (c', s)$$

where

$$c' = (x \wedge y) \vee (x \wedge c) \vee (y \wedge c)$$
$$s = x \otimes y \otimes c$$

Full adder



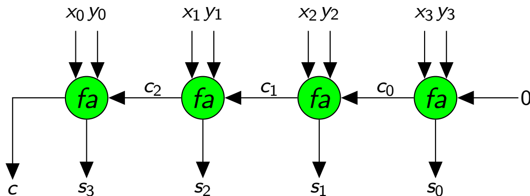
x	y	c	(c', s)
0	0	0	(0, 0)
0	0	1	(0, 1)
0	1	0	(0, 1)
0	1	1	(1, 0)
1	0	0	(0, 1)
1	0	1	(1, 0)
1	1	0	(1, 0)
1	1	1	(1, 1)

$$fa\ c\ (x, y) = (c', s')$$

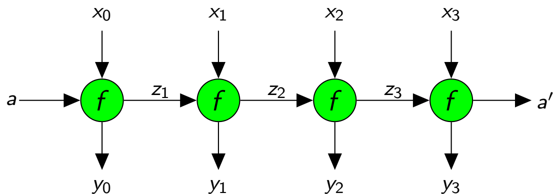
where

$$(c_0, s) = ha\ (x, y)$$
$$(c_1, s') = ha\ (s, c)$$
$$c' = c_0 \otimes c_1$$

Ripple Carry Adder



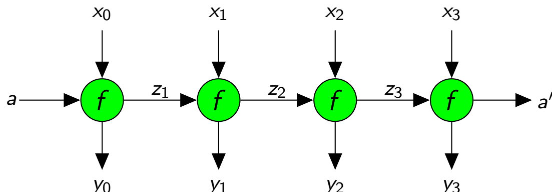
Chain



$$f :: a \rightarrow x \rightarrow (a, y)$$

$$f\ a\ x\ =\ (z, y)$$

Chain

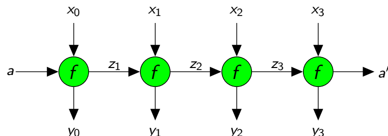


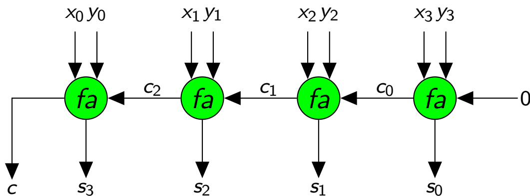
$$f :: a \to x \to (a, y)$$

$$f\ a\ x\ =\ (z, y)$$

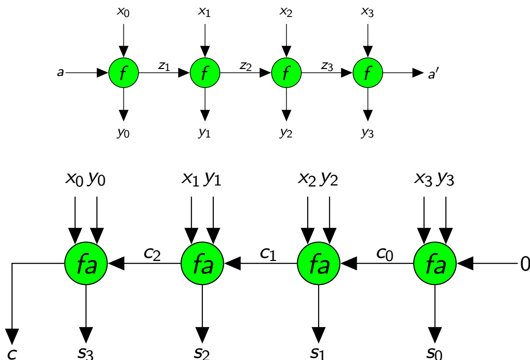
$$\begin{aligned} chain\ f\ a\ xs \ &= \ (a', ys) \\ &\quad \textbf{where} \\ &\quad\ (as, ys) \ = \ unzip\ \$\ zipWith\ f\ (a : as)\ xs \\ &\quad\ a' \qquad\ \ = \ last\ as \end{aligned}$$

Ripple Carry Adder



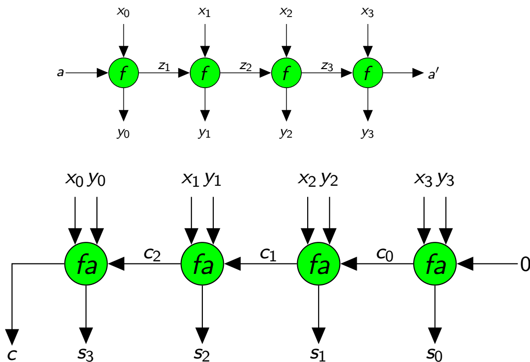


Ripple Carry Adder



$$
\begin{aligned}
rca\ xs\ ys &= reverse\ (ss ++ [c]) \\
&\quad \textbf{where} \\
&\quad xys = reverse\ \$\ zip\ xs\ ys \\
&\quad (c, ss) = chain\ fa\ 0\ xys
\end{aligned}
$$

Ripple Carry Adder in CλaSH



$$
\begin{aligned}
rca\ xs\ ys &= vreverse\ (ss <+ c) \\
&\textbf{where} \\
&\quad xys = vreverse\ \$\ vzip\ xs\ ys \\
&\quad (c, ss) = vchain\ fa\ 0\ xys
\end{aligned}
$$

Basic CλaSHdefinitions

- Types:

$$\begin{array}{ll} \textit{Booleans}: & \textbf{data } \textit{Bool} = \textit{True} \mid \textit{False} \\ \textit{Bits}: & \textbf{data } \textit{Bit} = \textit{High} \mid \textit{Low} \\ \textit{Vectors}: & \textit{Vector Dn a} \\ \textit{Integers}: & \textit{Signed Dn}, \textit{Unsigned Dn} \end{array}$$

- Predefined for vectors:

$$\textit{empty}, \textit{singleton } x$$

$$+>,\ <+,\ +>>,\ <<+,\ <++>,\ !$$

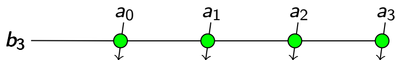
$$\begin{array}{l} \textit{vlength}, \textit{vnull}, \textit{vhead}, \textit{vtail}, \textit{vlast}, \textit{vinit}, \\ \textit{vtake}, \textit{vdrop}, \textit{vmap}, \textit{vzipWith}, \textit{vfoldl}, \textit{vfoldr}, \\ \textit{vzip}, \textit{vunzip}, \textit{vconcat}, \textit{vreverse}, \textit{viteraten}, \textit{vcopyn} \end{array}$$

Multiplication

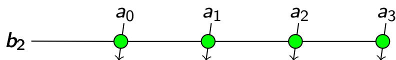
Multiplication

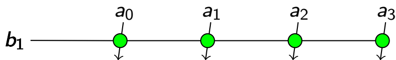
$$
\begin{array}{ccccccccccc}
 & & & & 1 & 1 & 0 & 1 & 0 & \\
 & & & & & 1 & 0 & 1 & 1 & \times \\
\hline
 & & & & 1 & 1 & 0 & 1 & 0 & \\
 & & & 1 & 1 & 0 & 1 & 0 & & \\
 & & 0 & 0 & 0 & 0 & 0 & & & \\
 & 1 & 1 & 0 & 1 & 0 & & & & + \\
\hline
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 &
\end{array}
$$

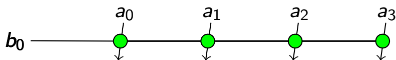
Multiplication



$$\begin{array}{ccccc} a_0 & a_1 & a_2 & a_3 & \\ b_0 & b_1 & b_2 & b_3 & \times \\ \hline \end{array}$$

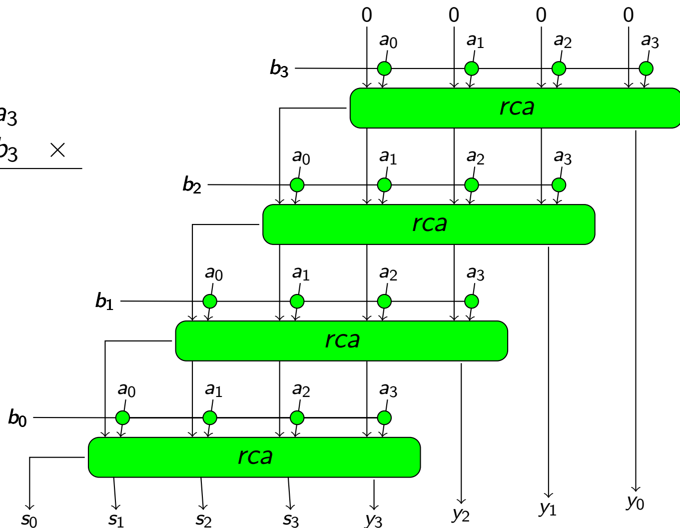






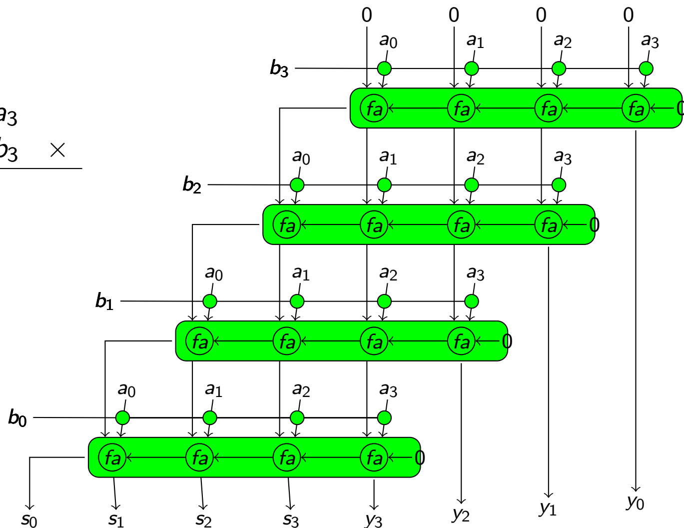
Multiplication

$$\begin{array}{ccccc} a_0 & a_1 & a_2 & a_3 & \\ b_0 & b_1 & b_2 & b_3 & \times \\ \hline \end{array}$$

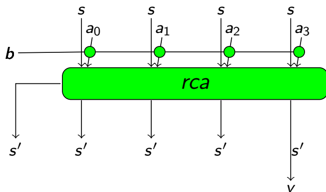


Multiplication

$$\begin{array}{ccccc} a_0 & a_1 & a_2 & a_3 & \\ b_0 & b_1 & b_2 & b_3 & \times \\ \hline \end{array}$$



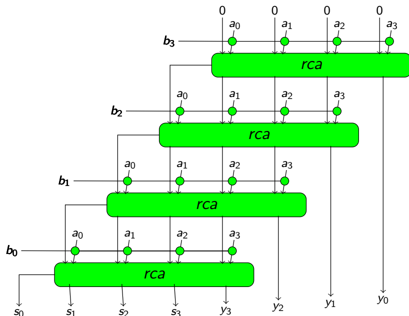
Multiplication



$$rca\ ss\ as\ = s's \quad \textbf{where} \ \cdots$$

$$\begin{aligned} addrow\ as\ ss\ b \ &= \ (s's, y) \\ &\textbf{where} \\ &\quad ss_{-} = rca\ ss\ (map\ (b\wedge)\ as) \\ &\quad (s's, y) = (init\ ss_{-}, last\ ss_{-}) \end{aligned}$$

Multiplication



$$addrow\ as\ ss\ b\ =\ (s's, y)\quad \textbf{where}\quad \cdots$$

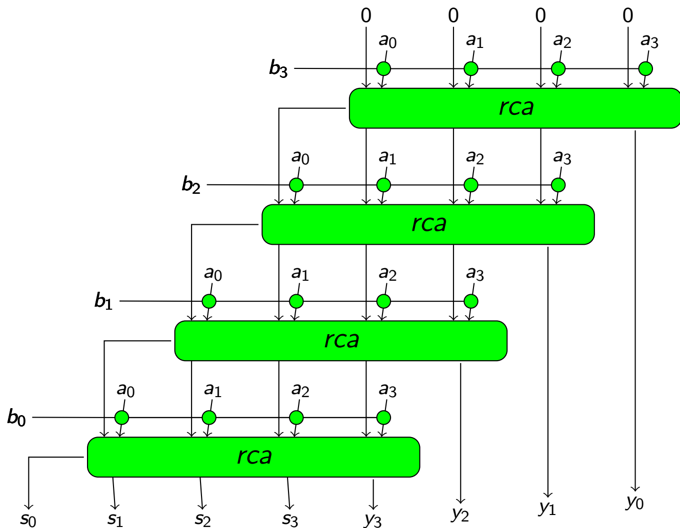
$$mul\ as\ bs\ =\ ss +\!\!+\ reverse\ ys$$

$\quad$ **where**

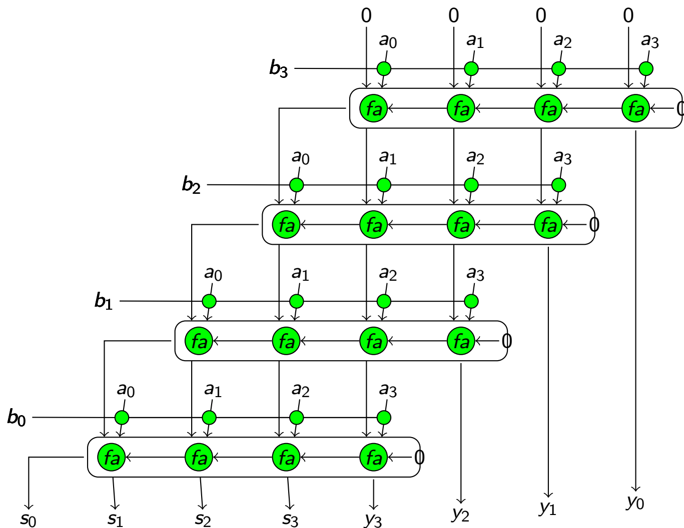
$$zeroes\ =\ replicate\ (length\ as)\ 0$$

$$(ss, ys)\ =\ chain\ (addrow\ as)\ zeroes\ (reverse\ bs)$$

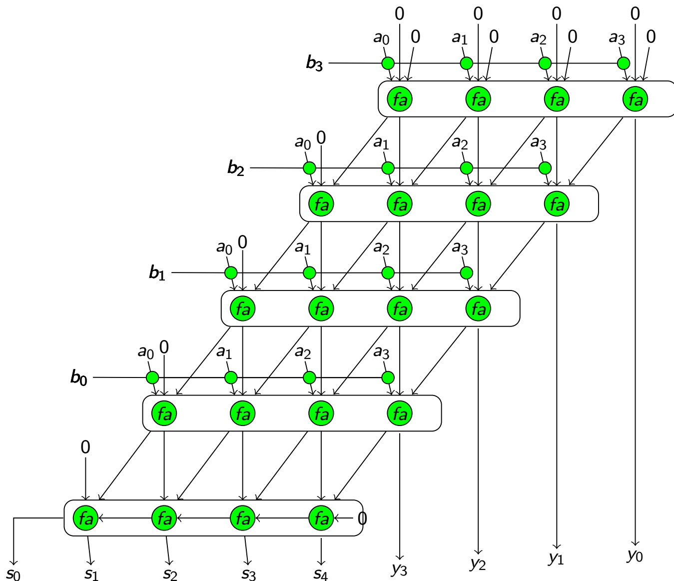
Multiplication



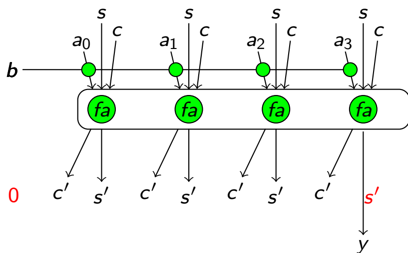
Multiplication



Baugh-Wooley Multiplication



Baugh-Wooley Multiplication

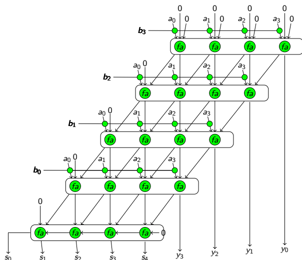


$$addrow' \; as \; (cs, ss) \; b = ((c's, s's), y)$$

where

$$
\begin{aligned}
a's &= map \; (b\wedge) \; as \\
sa's &= zip \; ss \; a's \\
(c's, s's_{-}) &= unzip \; \$ \; zipWith \; fa \; cs \; sa's \\
s's &= 0 : init \; s's_{-} \\
y &= last \; s's_{-}
\end{aligned}
$$

Baugh-Wooley Multiplication



$$addrow' \; as \; (cs, ss) \; b \; = \; ((c's, s's), y) \;\; \textbf{where} \;\; \cdots$$

$$
\begin{aligned}
mul' \; as \; bs \;\; &= \;\; res \mathbin{+\!\!+} reverse \; ys \\
&\textbf{where} \\
&\quad zs = replicate \; (length \; as) \; 0 \\
&\quad ((cs, ss), ys) = chain \; (addrow' \; as) \; (zs, zs) \; (reverse \; bs) \\
&\quad res = rca \; cs \; ss
\end{aligned}
$$

BW-Multiplication in CλaSH

$$addrow' :: [Int] \to ([Int] \to [Int]) \to Int \to (([Int],[Int]), Int)$$

$$\begin{array}{llcl} \textbf{data} & Bit & = & High \mid Low \\ \textbf{type} & V4B & = & Vector\ 4\ Bit \end{array}$$

$$addrow' :: V4B \to (V4B \to V4B) \to Bit \to ((V4B, V4B), Bit)$$

$$\begin{array}{l} addrow'\ as\ S\ (cs, ss)\ b \quad = \quad ((c's, s's), y) \\ \quad \textbf{where} \\ \quad\quad a's = vmap\ (hwand\ b)\ as \\ \quad\quad sa's = vzip\ ss\ a's \\ \quad\quad (c's, s's_{-}) = vunzip\ \$\ vzipWith\ fa\ cs\ sa's \\ \quad\quad s's = 0 \mathrel{+\!\!>\!\!>} s's_{-} \\ \quad\quad y = vlast\ s's_{-} \end{array}$$

BW-Multiplication in CλaSH

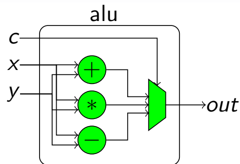
$$addrow'\ as\ (cs, ss)\ b \ = \ ((c's, s's), y) \ \ \textbf{where} \ \ \cdots$$

$$\begin{aligned}
mul'\ as\ bs \ &= \ res \ <{+\!\!+}> \ v reverse\ ys \\
&\textbf{where} \\
&\quad zs \quad = \ vreplicate\ (vlength\ as)\ 0 \\
&\quad ((cs, ss), ys) \quad = \ vchain\ (addrow'\ as)\ (zs, zs)\ (vreverse\ bs) \\
&\quad res \quad = \ rca\ cs\ ss
\end{aligned}$$

Rewrite system

Specification:

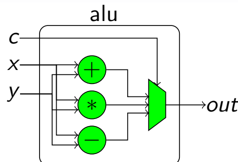
$$
\begin{aligned}
alu\ ADD &= (+) \\
alu\ MUL &= (*) \\
alu\ SUB &= (-)
\end{aligned}
$$



Rewrite system

Specification:

$$
\begin{aligned}
alu\ ADD &= (+)\\
alu\ MUL &= (*)\\
alu\ SUB &= (-)
\end{aligned}
$$



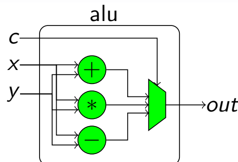
GHC ⇒ Core:

$$
\begin{aligned}
alu = \lambda c.\,\textbf{case}\ c\ \textbf{of}\\
ADD \rightarrow (+)\\
MUL \rightarrow (*)\\
SUB \rightarrow (-)
\end{aligned}
$$

Rewrite system

Specification:

$$\begin{aligned} alu\ ADD &= (+) \\ alu\ MUL &= (*) \\ alu\ SUB &= (-) \end{aligned}$$



GHC ⇒ Core:

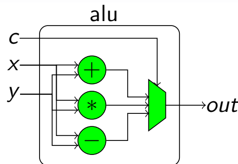
$$alu = \lambda c.\, \textbf{case}\ c\ \textbf{of} \quad \begin{aligned} ADD &\rightarrow (+) \\ MUL &\rightarrow (*) \\ SUB &\rightarrow (-) \end{aligned}$$

η−expansion:

$$alu = \lambda c.\, \lambda x.\, \lambda y. \left(\begin{aligned} &\textbf{case}\ c\ \textbf{of} \\ &\quad ADD \rightarrow (+) \\ &\quad MUL \rightarrow (*) \\ &\quad SUB \rightarrow (-) \end{aligned} \right) x\ y$$

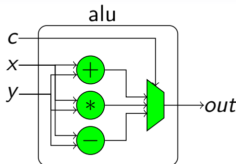
Rewrite system

$$alu = \lambda c.\,\lambda x.\,\lambda y.\left(\begin{array}{l}\textbf{case}\ c\ \textbf{of}\\ \quad ADD \rightarrow (+)\\ \quad MUL \rightarrow (*)\\ \quad SUB \rightarrow (-)\end{array}\right)\ x\ y$$



Rewrite system

$$alu = \lambda c.\, \lambda x.\, \lambda y. \left(\begin{array}{l} \textbf{case } c \textbf{ of} \\ \quad ADD \to (+) \\ \quad MUL \to (*) \\ \quad SUB \to (-) \end{array} \right) x\ y$$

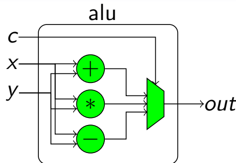


application propagation:

$$\begin{array}{rcl} alu & = & \lambda c\, x\, y.\ \textbf{case } c \textbf{ of} \\ & & \qquad ADD \to (+)\ x\ y \\ & & \qquad MUL \to (*)\ x\ y \\ & & \qquad SUB \to (-)\ x\ y \end{array}$$

Rewrite system

$$alu \;=\; \lambda c.\,\lambda x.\,\lambda y.\left(\begin{array}{l}\textbf{case}\ c\ \textbf{of}\\ \quad ADD \rightarrow (+)\\ \quad MUL \rightarrow (*)\\ \quad SUB \rightarrow (-)\end{array}\right)\ x\ y$$



application propagation:

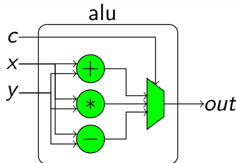
$$\begin{array}{l} alu \;=\; \lambda c\,x\,y.\ \textbf{case}\ c\ \textbf{of}\\ \qquad ADD \rightarrow (+)\ x\ y\\ \qquad MUL \rightarrow (*)\ x\ y\\ \qquad SUB \rightarrow (-)\ x\ y\end{array}$$

or, equivalently:

$$\begin{array}{l} alu \;=\; \lambda c\,x\,y.\ \textbf{case}\ c\ \textbf{of}\\ \qquad ADD \rightarrow x + y\\ \qquad MUL \rightarrow x * y\\ \qquad SUB \rightarrow x - y\end{array}$$

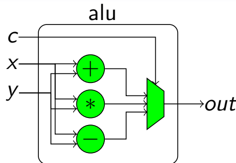
Rewrite system

$$
\begin{aligned}
alu \;=\; \lambda c\, x\, y.\ \textbf{case}\ c\ \textbf{of} \\
ADD \rightarrow x + y \\
MUL \rightarrow x * y \\
SUB \rightarrow x - y
\end{aligned}
$$



Rewrite system

$$alu = \lambda c\, x\, y.\ \textbf{case}\ c\ \textbf{of}$$
$$\quad ADD \rightarrow x + y$$
$$\quad MUL \rightarrow x * y$$
$$\quad SUB \rightarrow x - y$$

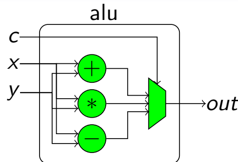


letification:

$$alu = \lambda c\, x\, y.\ \textbf{let}$$
$$\quad out = \textbf{case}\ c\ \textbf{of}$$
$$\qquad ADD \rightarrow x + y$$
$$\qquad MUL \rightarrow x * y$$
$$\qquad SUB \rightarrow x - y$$
$$\textbf{in}$$
$$\quad out$$

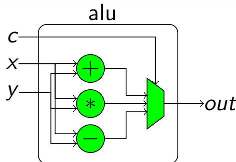
Rewrite system

$$alu \;=\; \lambda c\, x\, y.\ \textbf{let}\ out = \textbf{case}\ c\ \textbf{of}\ \begin{array}{l} ADD \rightarrow x + y \\ MUL \rightarrow x * y \\ SUB \rightarrow x - y \end{array}\ \textbf{in}\ out$$



Rewrite system

$$alu = \lambda c\, x\, y.\ \textbf{let}\ out = \textbf{case}\ c\ \textbf{of}\ \begin{array}{l} ADD \rightarrow x + y \\ MUL \rightarrow x * y \\ SUB \rightarrow x - y \end{array}\ \textbf{in}\ out$$



subexpression extraction:

$$alu = \lambda c\, x\, y.\ \textbf{let}\ \begin{array}{l} p = x + y \\ q = x * y \\ r = x - y \\ out = \textbf{case}\ c\ \textbf{of}\ \begin{array}{l} ADD \rightarrow p \\ MUL \rightarrow q \\ SUB \rightarrow r \end{array} \end{array}\ \textbf{in}\ out$$

Rewrite system

Rewrite system

$$\text{sub} - \text{expression extraction}: \quad alu = \lambda c.\lambda x\, y.\, \textbf{let}$$

$$p = x + y$$

$$q = x * y$$

$$r = x - y$$

$$out = \textbf{case}\ c$$

$$ADD \rightarrow p$$

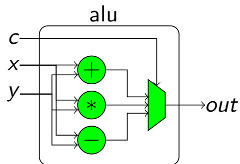
$$MUL \rightarrow q$$

$$SUB \rightarrow r$$

$$\textbf{in}$$

$$out$$

Rewrite system



Thnx

clash.ewi.utwente.nl

Exercises

http://goo.gl/WMqqd

$$
\begin{aligned}
&\mathit{mealyT}\ (\mathit{State}\ s)\ i\ =\ (\mathit{State}\ s', o)\\
&\qquad \textbf{where}\\
&\qquad\quad s' = \cdots \quad --\ \mathit{The\ value\ of\ the\ new/updated\ state}\\
&\qquad\quad o = \cdots \quad --\ \mathit{The\ value\ of\ the\ output}
\end{aligned}
$$

$$\mathit{machine} = \mathit{mealyT}\ \hat{}\hat{}\hat{}\ \mathit{initState}$$

Some exercises: *Shift register*, *FIFO buffer*, *pattern matcher*