From Math to Hardware

Jan Kuper

QBayLogic, Enschede, The Netherlands

ASCI Spring School 2017

Soesterberg

May 29, 2017

Introduction

- CPUs hardly any development
- FPGAs on the rise
- CλaSH: Haskell $\Rightarrow$ VHDL/Verilog

Introduction

- CPUs hardly any development
- FPGAs on the rise
- CλaSH: Haskell $\Rightarrow$ VHDL/Verilog

- Signs of development: by now there are users of CλaSH who use it for production ...

Introduction

- CPUs hardly any development
- FPGAs on the rise
- CλaSH: Haskell $\Rightarrow$ VHDL/Verilog

- Signs of development: by now there are users of CλaSH who use it for production ... but they all want to keep their names confidential.

Introduction

- CPUs hardly any development
- FPGAs on the rise
- CλaSH: Haskell $\Rightarrow$ VHDL/Verilog

- Signs of development: by now there are users of CλaSH who use it for production ... but they all want to keep their names confidential.

  They are big, world wide companies (search engines, networking, high frequency trading, car automation, ...)

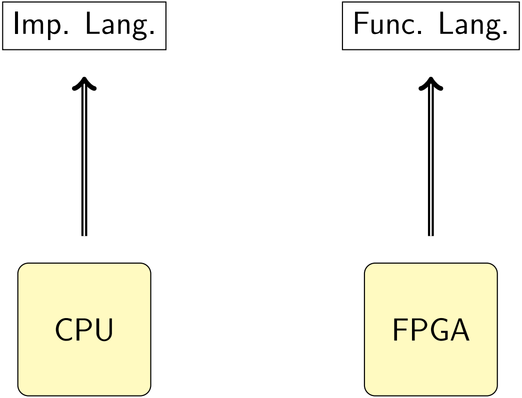
Introduction

QBAYLOGIC

- Ripple Carry adder
- Processor
- PID controller
- Transformations

- Examples: Cochlea Membrane, Adaptive Cruise Control, Slam algorithm, *N*-Queens, Processors, PID controllers, Tunnelling Ball Device
- Elementary architectures: adders, multipliers

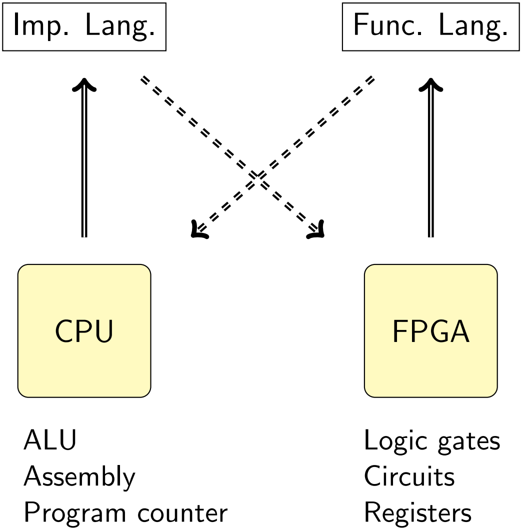
Languages and architectures



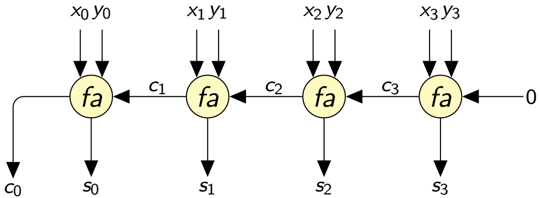
ALU
Assembly
Program counter

Logic gates
Circuits
Registers

Languages and architectures



Ripple Carry Adder



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

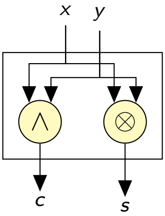
$$\begin{aligned} 0 \otimes 0 &= 0 \\ 0 \otimes 1 &= 1 \\ 1 \otimes 0 &= 1 \\ 1 \otimes 1 &= 0 \end{aligned}$$

Half Adder

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

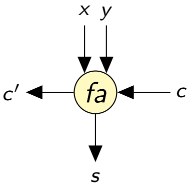
Half Adder

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



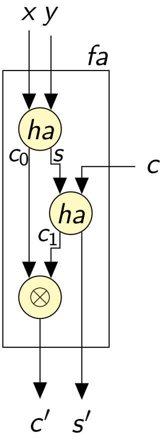
$$
\begin{aligned}
ha\ (x, y) \quad &= \quad (c, s) \\
&\textbf{where} \\
&\quad c \ = x \wedge y \\
&\quad s \ = x \otimes y
\end{aligned}
$$

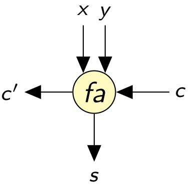
Full adder



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

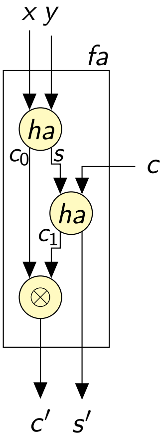




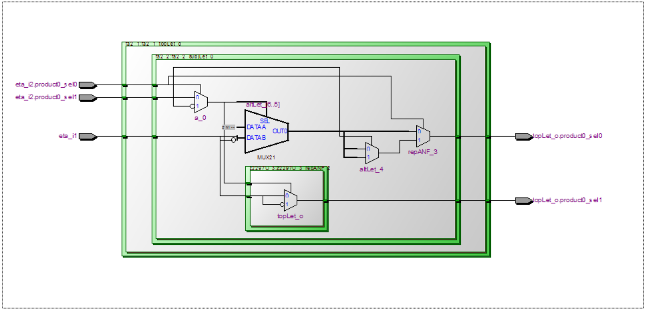
x	y	c	(c', s)
0	0	0	$(0,0)$
0	0	1	$(0,1)$
0	1	0	$(0,1)$
0	1	1	$(1,0)$
1	0	0	$(0,1)$
1	0	1	$(1,0)$
1	1	0	$(1,0)$
1	1	1	$(1,1)$

$$
\begin{aligned}
fa\ c\ (x, y) &= (c', s') \\
&\textbf{where} \\
&\quad (c_0, s) = ha\ (x, y) \\
&\quad (c_1, s') = ha\ (s, c) \\
&\quad c' = c_0 \otimes c_1
\end{aligned}
$$

Full adder

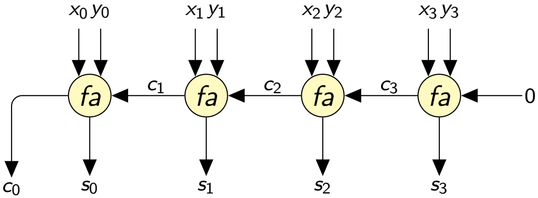


x y
fa
ha
c_0
s
c
ha
c_1
c′ s′



eta_i2.product0_sel0
eta_i2.product0_sel1
eta_i1
a_0
altLet_6
SEL
DATAA
DATAB
OUT0
MUX21
repANF_3
altLet_4
topLet_o
topLet_o.product0_sel0
topLet_o.product0_sel1

Ripple Carry Adder



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

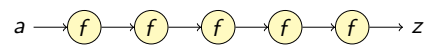
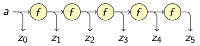
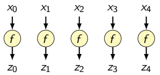
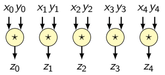
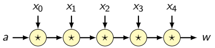
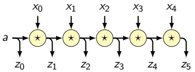
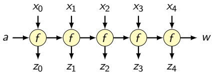
where

$\cdots$

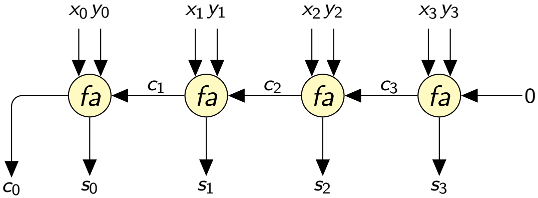
$$rca\ xs\ ys \quad = \quad \cdots$$

where

$\dots$

itn		$f\ x \Rightarrow z$	$z = f^n\ x$	$z = itn\ f\ x\ n$
itnscan		$f\ x \Rightarrow z$	$zs = \overline{f^n}\ x$	$zs = itnscan\ f\ x\ n$
map		$f\ x \Rightarrow z$	$zs = \widehat{f}\ xs$	$zs = map\ f\ xs$
zipWith		$x\ \star\ y \Rightarrow z$	$zs = xs\ \widehat{\star}\ ys$	$zs = zipWith\ (\star)\ xs\ ys$
foldl		$a\ \star\ x \Rightarrow a'$	$w = a \circledast xs$	$w = foldl\ (\star)\ a\ xs$
scanl		$a\ \star\ x \Rightarrow a'$ $z = a$	$w = a\ \overline{\circledast\ xs}$	$zs = scanl\ (\star)\ a\ xs$
mapAccumL		$f\ a\ x \Rightarrow (a', z)$	?	$(w, zs) = mapAccumL\ f\ a\ xs$

Ripple Carry Adder



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

where

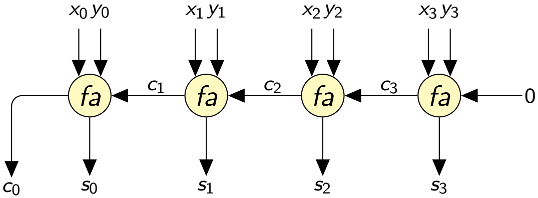
$\cdots$

$$rca\ xs\ ys \ = \ \cdots$$

where

$\cdots$

Ripple Carry Adder



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

where

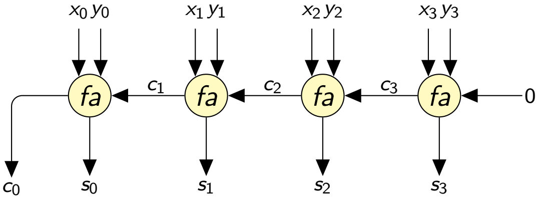
...

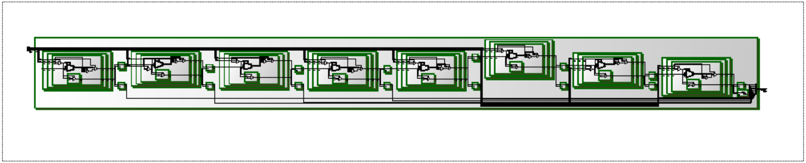
$$rca\ xs\ ys \;=\; c_0 : ss$$

where

$$xys \;=\; zip\ xs\ ys$$
$$(c_0, ss) \;=\; mapAccumR\ fa\ 0\ xys$$

Ripple Carry Adder



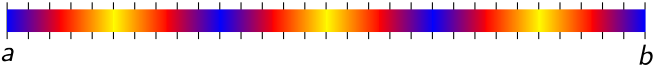


HPC Example: Heat Diffusion

HPC Example: Heat Diffusion

Problem:

Calculate temperature $T(t,x)$ for all time $t \geq 0$ and for each position $x \in [a,b]$.



Simplifications:

- One dimensional
- Temperatures of endpoints constant
- No external influences
- Endpoints constant temperature

Heatflow: Mathematics $\Rightarrow$ Discretization

QBAYLOGIC

Specification:

$$\frac{\partial T(t,x)}{\partial t} - \alpha \cdot \nabla^2 T(t,x) = 0$$

($\partial t, \nabla$: diff. operators w.r.t. time, space)

Definitions:

$$\frac{\partial T(t,x)}{\partial t} = \lim_{\Delta t \to 0} \frac{T(t+\Delta t,x) - T(t,x)}{\Delta t}$$

$$\nabla^2 T(t,x) = \lim_{\Delta x \to 0} \frac{T(t,x-\Delta x) - 2T(t,x) + T(t,x+\Delta x)}{(\Delta x)^2}$$

Take Δt, Δx small, $\Rightarrow$ $t_k = k \cdot \Delta t$, $x_i = a + i \cdot \Delta x$:

$$\frac{T(t_{k+1},x_i) - T(t_k,x_i)}{\Delta t} = \alpha \cdot \frac{T(t_k,x_{i-1}) - 2T(t_k,x_i) + T(t_k,x_{i+1})}{(\Delta x)^2}$$

Write τ_i, τ_i' for $T(t_k, x_i), T(t_{k+1}, x_i)$, define $c = \frac{\alpha \cdot \Delta t}{(\Delta x)^2}$:

$$\tau_i' = \tau_i + c \cdot (\tau_{i-1} - 2\tau_i + \tau_{i+1})$$

Aggregation

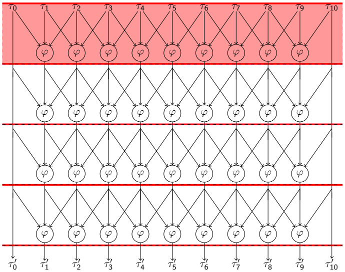
$$\tau_i' = \begin{cases} \tau_0 & (i = 0) \\ \varphi\left(\tau_{i-1}, \tau_i, \tau_{i+1}\right) & (1 \leq i \leq N{-}1) \\ \tau_N & (i = N) \end{cases}$$

$$\mathit{next}\ \mathit{ts}\ =\ [\mathit{ts}!!0]\ +\!+\ \quad \mathit{map}\ \varphi\ (\mathit{triples}\ \mathit{ts})\ \quad +\!+\ [\mathit{ts}!!n]$$

Aggregation

$$\tau_i' = \begin{cases} \tau_0 & (i = 0) \\ \varphi\left(\tau_{i-1}, \tau_i, \tau_{i+1}\right) & (1 \leq i \leq N{-}1) \\ \tau_N & (i = N) \end{cases}$$

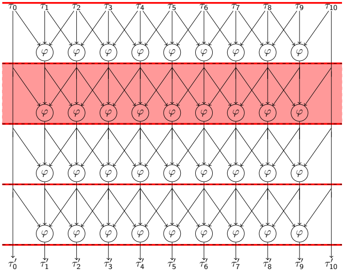
$$\mathit{next}\ \mathit{ts}\ =\ [\mathit{ts}!!0]\ {+}{+}\ \mathit{map}\ \varphi\ (\mathit{triples}\ \mathit{ts})\ {+}{+}\ [\mathit{ts}!!n]$$



Aggregation

$$\tau'_i = \begin{cases} \tau_0 & (i = 0) \\ \varphi\,(\tau_{i-1}, \tau_i, \tau_{i+1}) & (1 \leq i \leq N{-}1) \\ \tau_N & (i = N) \end{cases}$$

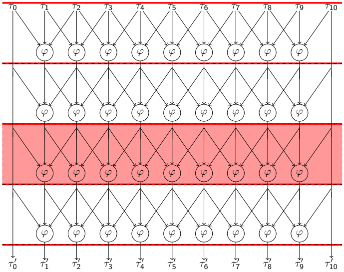
$$\mathit{next}\ \mathit{ts} \ = \ [\mathit{ts}!!0] \ {+}{+} \quad \mathit{map}\ \varphi\,(\mathit{triples}\ \mathit{ts}) \quad {+}{+} \ [\mathit{ts}!!n]$$



Aggregation

$$\tau'_i = \begin{cases} \tau_0 & (i = 0) \\ \varphi\,(\tau_{i-1}, \tau_i, \tau_{i+1}) & (1 \leq i \leq N{-}1) \\ \tau_N & (i = N) \end{cases}$$

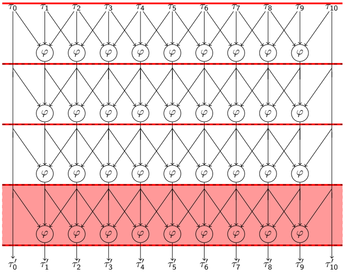
$$\mathit{next}\ \mathit{ts} \ = \ [\mathit{ts}!!0] \ {+}{+} \ \mathit{map}\ \varphi\ (\mathit{triples}\ \mathit{ts}) \ {+}{+} \ [\mathit{ts}!!n]$$



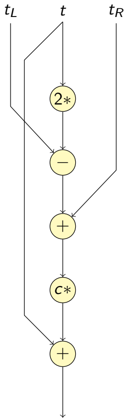
Aggregation

$$\tau'_i = \begin{cases} \tau_0 & (i = 0) \\ \varphi\,(\tau_{i-1}, \tau_i, \tau_{i+1}) & (1 \leq i \leq N{-}1) \\ \tau_N & (i = N) \end{cases}$$

$$\mathit{next}\ \mathit{ts}\ =\ [\mathit{ts}!!0]\ +\!+\ \mathit{map}\ \varphi\ (\mathit{triples}\ \mathit{ts})\ +\!+\ [\mathit{ts}!!n]$$

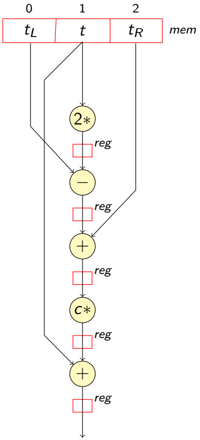


Processor derivation



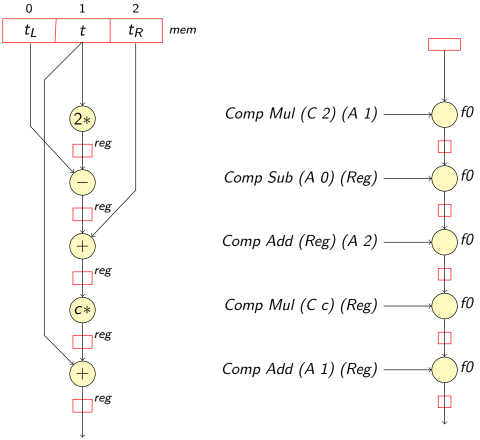

```
f t (tL,tR) = t + c * (tL -2*t + tR)
```

Processor derivation



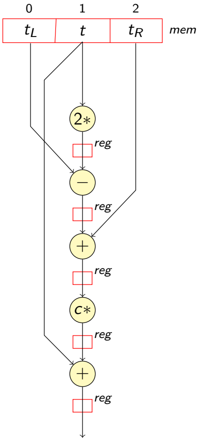

```
f t (tL,tR) = t + c * (tL -2*t + tR)
```

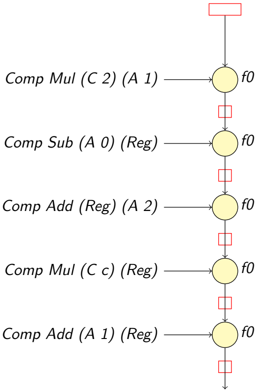
Processor derivation




```
f t (tL,tR) = t + c * (tL -2*t + tR)
```

Processor derivation

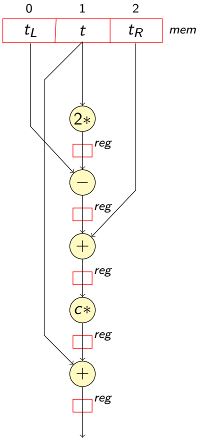


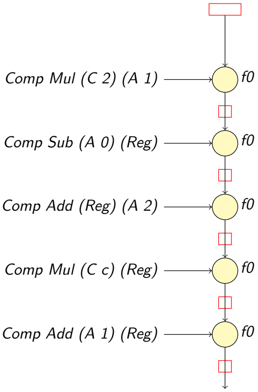



```haskell
data Data  = Addr Int
           | Const Int
           | Reg

data Opc   = Add
           | Mul
           | Sub
           | Nop

data Instr = Comp Opc Data Data
           | Read
```

```haskell
f t (tL,tR) = t + c * (tL -2*t + tR)
```

Processor derivation

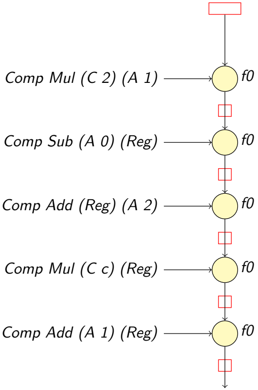





```haskell
data Data   = Addr Int
            | Const Int
            | Reg

data Opc    = Add
            | Mul
            | Sub
            | Nop

data Instr = Comp Opc Data Data
            | Read
```

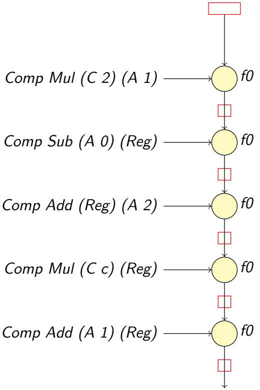
```haskell
prog0 = [ Comp Mul (Const 2) (Addr 1)
        , Comp Sub (Addr 0)   Reg
        , Comp Add  Reg      (Addr 2)
        , Comp Mul (Const c)  Reg
        , Comp Add (Addr 1)   Reg
        ]
```

```haskell
f t (tL,tR) = t + c * (tL -2*t + tR)
```



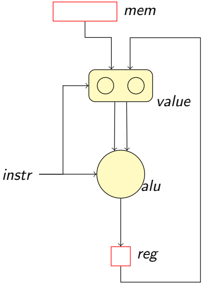

```
prog0 = [ Comp Mul (Const 2) (Addr 1)
        , Comp Sub (Addr 0)   Reg
        , Comp Add  Reg      (Addr 2)
        , Comp Mul (Const c)  Reg
        , Comp Add (Addr 1)   Reg
        ]
```

```
f t (tL,tR) = t + c * (tL -2*t + tR)
```

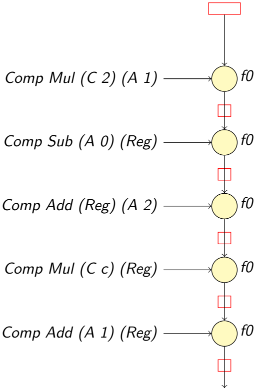



```haskell
prog0 = [ Comp Mul (Const 2) (Addr 1)
        , Comp Sub (Addr 0)   Reg
        , Comp Add  Reg      (Addr 2)
        , Comp Mul (Const c)  Reg
        , Comp Add (Addr 1)   Reg
        ]
```

```haskell
f0 mem reg (Comp opc d0 d1) = reg'
    where
      x = value (mem,reg) d0
      y = value (mem,reg) d1

      reg' = alu opc x y
```

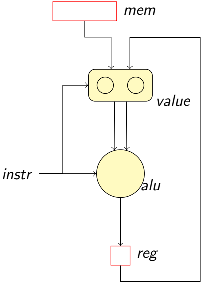



```haskell
f t (tL,tR) = t + c * (tL -2*t + tR)
```




```
prog0 = [ Comp Mul (Const 2) (Addr 1)
        , Comp Sub (Addr 0)   Reg
        , Comp Add  Reg       (Addr 2)
        , Comp Mul (Const c)  Reg
        , Comp Add (Addr 1)   Reg
        ]
```

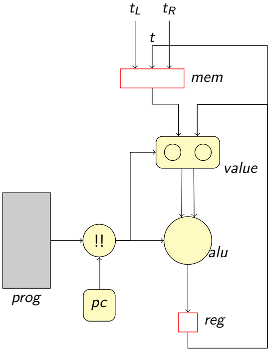
```
f0 mem reg (Comp opc d0 d1) = reg'
     where
        x = value (mem,reg) d0
        y = value (mem,reg) d1

        reg' = alu opc x y
```




```
value (mem,reg) d = case d of
        Addr  i -> mem!!i
        Const n -> n
        Reg     -> reg

alu opc x y = case opc of
        Add     -> x + y
        Mul     -> x * y
        Sub     -> x - y
        Nop     -> x
```

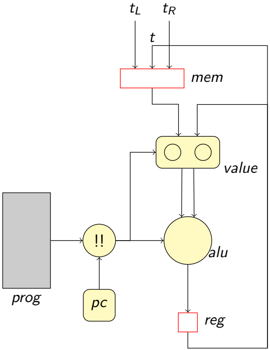
```
f t (tL,tR) = t + c * (tL -2*t + tR)
```

```
prog = [ Read
       , Comp Mul (Const 2) (Addr 1)
       , Comp Sub (Addr 0)   Reg
       , Comp Add  Reg      (Addr 2)
       , Comp Mul (Const c)  Reg
       , Comp Add (Addr 1)   Reg
       ]
```



```haskell
prog = [ Read
       , Comp Mul (Const 2) (Addr 1)
       , Comp Sub (Addr 0)   Reg
       , Comp Add  Reg      (Addr 2)
       , Comp Mul (Const c)  Reg
       , Comp Add (Addr 1)   Reg
       ]
```

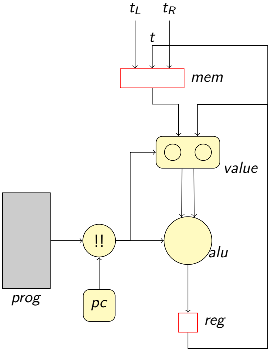
```haskell
f1 (mem,reg) (instr,(tL,tR)) = (mem',reg')
       where
         (mem',reg') = case instr of

                         Comp opc d0 d1  -> ( mem, alu opc x y )
                                            where
                                              x = value (mem,reg) d0
                                              y = value (mem,reg) d1

                         Read            -> ( [tL,reg,tR] , 0 )
```



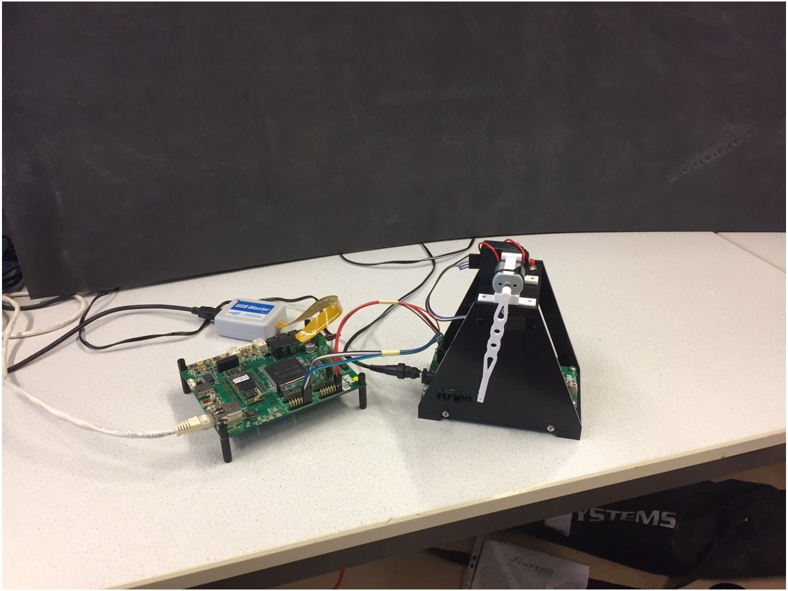
```
prog = [ Read
       , Comp Mul (Const 2) (Addr 1)
       , Comp Sub (Addr 0)   Reg
       , Comp Add  Reg      (Addr 2)
       , Comp Mul (Const c)  Reg
       , Comp Add (Addr 1)   Reg
       ]
```

```
f1 (mem,reg) (instr,(tL,tR)) = (mem',reg')
       where
         (mem',reg') = case instr of

                    Comp opc d0 d1  -> ( mem, alu opc x y )
                                       where
                                         x = value (mem,reg) d0
                                         y = value (mem,reg) d1

                    Read            -> ( [tL,reg,tR] , 0 )
```

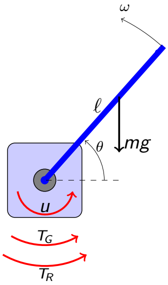
```
f_proc prog (mem,reg,pc) (tL,tR) = ( (mem',reg',pc') , outp )

       where

         pc'         | pc < length prog -1     = pc + 1
                     | otherwise               = 0

         instr       = prog!!pc

         (mem',reg') = f1 (mem,reg) (instr,(tL,tR))

         outp        = reg
```



Pendulum

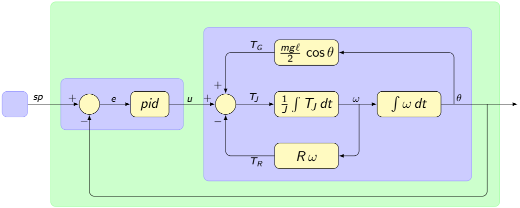


Pendulum

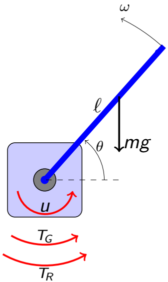


setpoints:
$45^\circ, 135^\circ$

Pendulum

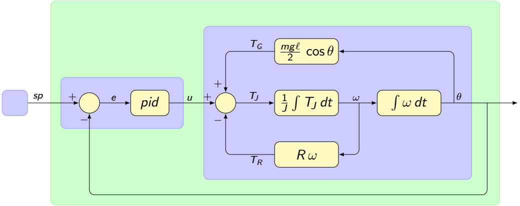


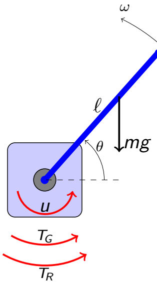
$$
\begin{aligned}
T_J(t) &= T_G + u - T_R \\
&\quad \textbf{where} \\
&\qquad T_G = \frac{mg\ell}{2}\cos(\theta(t)) \\
&\qquad u = pid(t) \\
&\qquad T_R = R\,\omega(t) \\
\\
\omega(t) &= \frac{1}{J}\int_0^t T_J(t)\,dt \\
\theta(t) &= \int_0^t \omega(t)dt
\end{aligned}
$$



setpoints:
$45^\circ, 135^\circ$

Pendulum



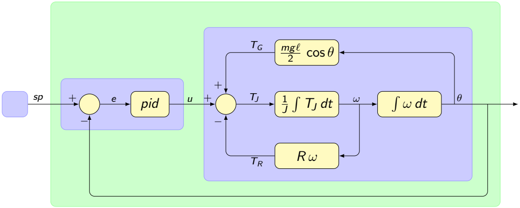


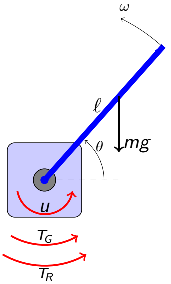
$$
\begin{aligned}
T_J(t) &= T_G + u - T_R \\
&\quad \textbf{where} \\
&\qquad T_G = \frac{mg\ell}{2}\cos(\theta(t)) \\
&\qquad u = pid(t) \\
&\qquad T_R = R\,\omega(t) \\
\omega(t) &= \frac{1}{J}\int_0^t T_J(t)\,dt \\
\theta(t) &= \int_0^t \omega(t)dt
\end{aligned}
$$

```
trq_J t = trq_G + u + trq_R
        where
          trq_G = m*g*l/2 * cos (theta t)
          u     = pid t
          trq_R = res * omega t

omega t = 1/j * integral trq_J (0,t-dt) dt

theta t = integral omega (0,t) dt
```

```
integral f (a,b) dx | b<=a      = 0
                    | otherwise = sum [ f x * dx | x <- [a, a+dx .. b-0.5*dx] ]
```

Pendulum





$$
\begin{aligned}
T_J(t) &= T_G + u - T_R \\
&\quad \textbf{where} \\
&\qquad T_G = \frac{mg\ell}{2}\cos(\theta(t)) \\
&\qquad u = pid(t) \\
&\qquad T_R = R\,\omega(t) \\
\omega(t) &= \frac{1}{J}\int_0^t T_J(t)\,dt \\
\theta(t) &= \int_0^t \omega(t)dt
\end{aligned}
$$

```
trq_J t = trq_G + u + trq_R
        where
          trq_G = m*g*l/2 * cos (theta t)
          u     = pid t
          trq_R = res * omega t

omega t = 1/j * integral trq_J (0,t-dt) dt

theta t = integral omega (0,t) dt
```

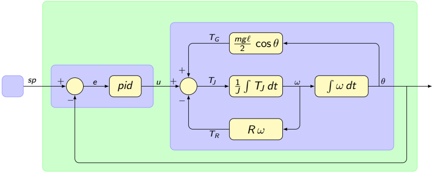
$$
\begin{aligned}
e(t) &= sp(t) - \theta(t) \\
pid(t) &= K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}
\end{aligned}
$$

```
err t = sp t - theta t

pid t =   kp * err t
        + ki * integral err (0,t) dt
        + kd * (err t - err (t-dt)) / dt
```

Pendulum

```
trq_J t = trq_G + u + trq_R
        where
          trq_G = m*g*l/2 * cos (theta t)
          u     = pid t
          trq_R = res * omega t

omega t = 1/j * integral trq_J (0,t-dt) dt

theta t = integral omega (0,t) dt
```

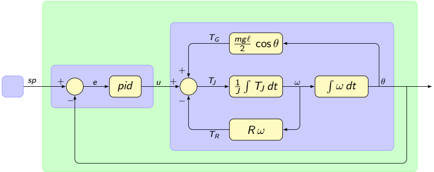
```
err t = sp t - theta t

pid t =   kp * err t
        + ki * integral err (0,t) dt
        + kd * (err t - err (t-dt)) / dt
```



Pendulum

```
trq_J t = trq_G + u + trq_R
      where
        trq_G = m*g*l/2 * cos (theta t)
        u     = pid t
        trq_R = res * omega t

omega t = 1/j * integral trq_J (0,t-dt) dt

theta t = integral omega (0,t) dt
```

```
err t = sp t - theta t

pid t =   kp * err t
        + ki * integral err (0,t) dt
        + kd * (err t - err (t-dt)) / dt
```




```
drivPend (sOmega,sTheta) u = ( (sOmega',sTheta') , th )
      where
        trq  = tG + u - tR
          where
            tG = m*g*l/2 * cos th
            tR = res * om

        (sOmega',om) = omega sOmega trq
        (sTheta',th) = theta sTheta om

omega int trq = ( int' , int )
      where
        int' = int + 1/j * trq*dt

theta int om  = ( int' , int' )
      where
        int' = int + om*dt
```

```
pid (int,prevE) (sp,mVal) = ((int',err), u)
      where
        err   = sp - mVal

        int'  = int + err*dt
        deriv = (err-prevE) / dt

        u     =   kp * err
                + ki * int'
                + kd * deriv
```

Pendulum

```
system (cyber,phys) (sCyber,sPhys) sp = ( (sCyber',sPhys') , mVal )
      where
        (sPhys' , mVal) = phys   sPhys   u
        (sCyber', u    ) = cyber sCyber (sp,mVal)
```

```
sim (system (pid,drivPend)) (sCyber_0,sPhys_0) setpoints
   where
     sCyber_0 = (0,0)
     sPhys_0  = (0,0)
     setpoints = ...
```

Towards CλaSH

- Number types:

  type *Int16* = *Signed 16*
  type *Nat32* = *Unsigned 32*
  type *Fix4_16* = *SFixed 4 16*

- Lists/Vectors:

  type *Prog* = *Vec 6 Instr*
  type *Mem* = *Vec 32 Int16*

- Top level:

  topEntity = f_proc prog <^> (mem0,0,0)

- To HDL:

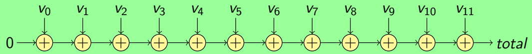
  `:vhdl`
  `:verilog`

```
omega :: Float -> Float

omega :: Float -> Float -> (Float,Float)

omega :: SFixed 18 18 -> SFixed 18 18 -> (SFixed 18 18, SFixed 18 18)
```

-

Transformation rules

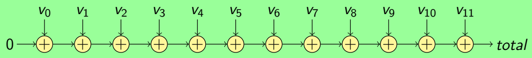
Transformation rules



$$total = foldl\ (+)\ 0\ vs$$

$$total = 0 \bigoplus vs$$

Transformation rules

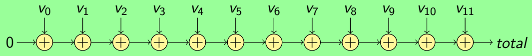


$$total = foldl\ (+)\ 0\ vs$$

$$total = 0 \oplus vs$$

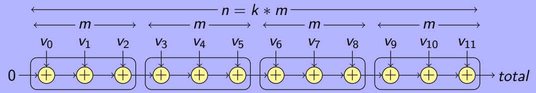
$$\begin{aligned} foldl\ f\ a\ [\,] &= a \\ foldl\ f\ a\ (x{:}xs) &= foldl\ f\ (f\ a\ x)\ xs \end{aligned}$$

Transformation rules

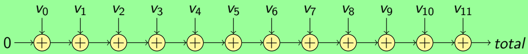


$$total = foldl\ (+)\ 0\ vs$$

$$total = 0 \oplus vs$$

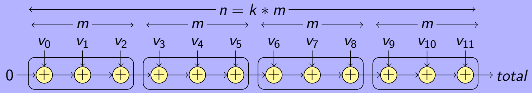


Transformation rules



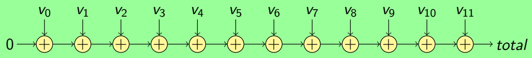
$$total = foldl\ (+)\ 0\ vs$$

$$total = 0 \oplus vs$$



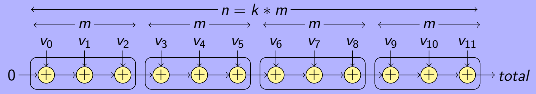
$$total = foldl\ (foldl\ (+))\ 0\ vss$$

Transformation rules



$$total = foldl\ (+)\ 0\ vs$$

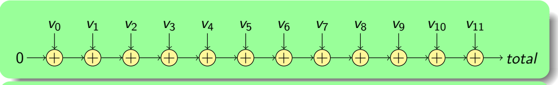
$$total = 0 \oplus vs$$



$$total = foldl\ (foldl\ (+))\ 0\ vss$$

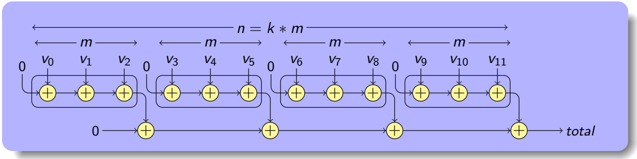
$$total = 0 \circledcirc vss$$

Transformation rules

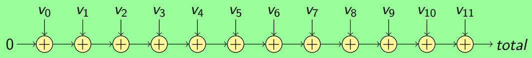


$$total = foldl\ (+)\ 0\ vs$$

$$total = 0 \oplus vs$$

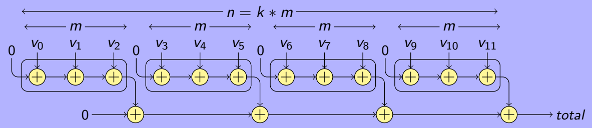


Transformation rules



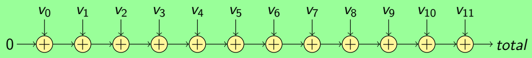
$$total = foldl\ (+)\ 0\ vs$$

$$total = 0 \oplus vs$$



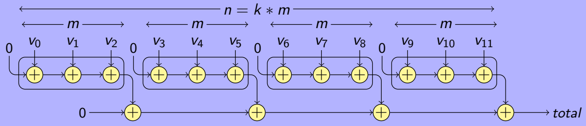
associative, neutral element

Transformation rules



$$total = foldl\ (+)\ 0\ vs$$

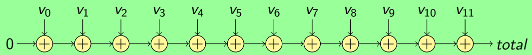
$$total = 0 \oplus vs$$



associative, neutral element

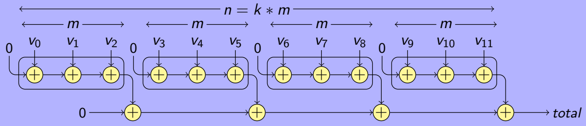
$$total = foldl\ (+)\ 0\ \$\ map\ (foldl\ (+)\ 0)\ vss$$

Transformation rules



$$total = foldl\ (+)\ 0\ vs$$

$$total = 0 \oplus vs$$



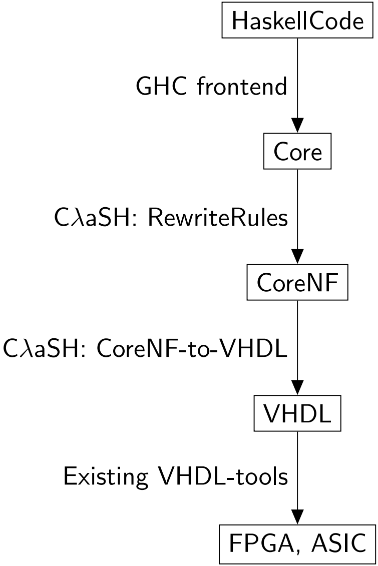
associative, neutral element

$$total = foldl\ (+)\ 0\ \$\ map\ (foldl\ (+)\ 0)\ vss$$

$$total = 0 \oplus (\widehat{0 \oplus vss})$$

The CλaSH Mechanism

CλaSH pipeline



Parsing
Desugaring
Type-checking

Remove polymorphism
Inlining
η-expansion
...

Translation into VHDL

Synthesis
Place-and-route

Rewrite system

Specification:

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

$$\textbf{data}\ \ OpCode = ADD \mid MUL \mid SUB$$

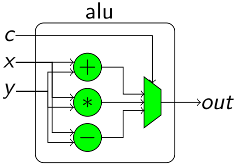
Note: $alu\ ADD :: Num\ a \Rightarrow a \to a \to a$

And: $alu\ ADD\ x\ y = x + y$

Rewrite system

Specification:

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

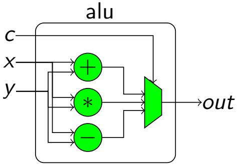


$$\textbf{data}\ \ OpCode = ADD \mid MUL \mid SUB$$

Note: $alu\ ADD :: Num\ a \Rightarrow a \rightarrow a \rightarrow a$

And: $alu\ ADD\ x\ y = x + y$

Rewrite system



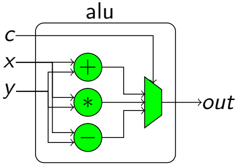
Specification:

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

Rewrite system

Specification:

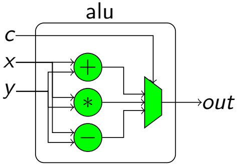
$$\begin{aligned} alu\ ADD &= (+) \\ alu\ MUL &= (*) \\ alu\ SUB &= (-) \end{aligned}$$



GHC ⇒ Core:

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

Rewrite system



Specification:

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

GHC $\Rightarrow$ Core:

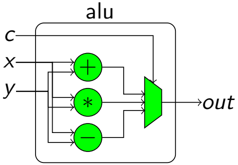
$$\mathit{alu} = \lambda c.\ \mathbf{case}\ c\ \mathbf{of} \begin{array}{l} \\ \mathit{ADD} \rightarrow (+) \\ \mathit{MUL} \rightarrow (*) \\ \mathit{SUB} \rightarrow (-) \end{array}$$

$\eta-$expansion:

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

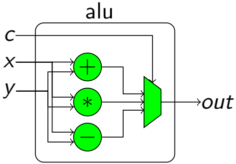
Rewrite system

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



Rewrite system

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

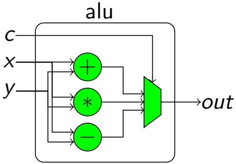


application propagation:

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

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:

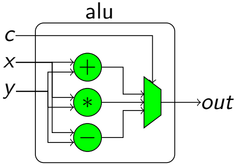
$$alu \;=\; \lambda c\,x\,y.\ \textbf{case}\ c\ \textbf{of}\quad \begin{array}{l} ADD \rightarrow (+)\ x\ y\\ MUL \rightarrow (*)\ x\ y\\ SUB \rightarrow (-)\ x\ y\end{array}$$

or, equivalently:

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

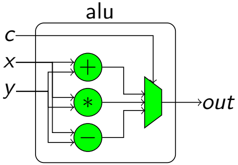
Rewrite system

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



Rewrite system

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

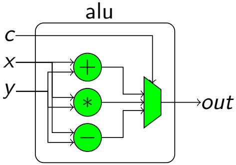


letification:

$$\begin{array}{lll} alu & = & \lambda c\, x\, y.\ \textbf{let} \\ & & \quad out = \textbf{case}\ c\ \textbf{of} \\ & & \qquad\qquad ADD \to x + y \\ & & \qquad\qquad MUL \to x * y \\ & & \qquad\qquad SUB \to x - y \\ & & \textbf{in} \\ & & \quad out \end{array}$$

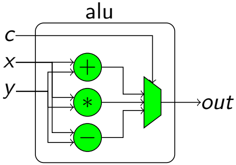
Rewrite system

$$
\begin{array}{lcl}
alu & = & \lambda c\,x\,y.\ \mathbf{let}\ out = \mathbf{case}\ c\ \mathbf{of} \\
 & & \qquad\qquad ADD \rightarrow x + y \\
 & & \qquad\qquad MUL \rightarrow x * y \\
 & & \qquad\qquad SUB \rightarrow x - y \\
 & & \quad\ \mathbf{in}\ out
\end{array}
$$



Rewrite system

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



subexpression extraction:

$$alu \;=\; \lambda c\, x\, y.\ \textbf{let}\ p = x + y \\ q = x * y \\ r = x - y \\ out = \textbf{case}\ c\ \textbf{of} \\ \qquad ADD \to p \\ \qquad MUL \to q \\ \qquad SUB \to r \\ \textbf{in}\ out$$

Thank you

qbaylogic.com