FPGA Design with CλaSH

Jan Kuper

QBayLogic, Enschede, The Netherlands

Electronics and Applications

Utrecht

June 1, 2017

Introduction

- Since 2005 CPUs hardly any development.
- FPGAs on the rise (HDLs, HLS tools).

Introduction

QBayLogic

- Since 2005 CPUs hardly any development.
- FPGAs on the rise (HDLs, HLS tools).

- CλaSH: Haskell ⇒ VHDL/Verilog.
- Developed at UTwente.

Introduction

QBayLogic

- Since 2005 CPUs hardly any development.
- FPGAs on the rise (HDLs, HLS tools).

- CλaSH: Haskell ⇒ VHDL/Verilog.
- Developed at UTwente.

- QBayLogic: Startup to use CλaSH in actual designs.
  Located in Enschede, in the Demcon building.
  Electronics & Applications: B.108.

Introduction

QBayLogic

- Since 2005 CPUs hardly any development.
- FPGAs on the rise (HDLs, HLS tools).

- CλaSH: Haskell ⇒ VHDL/Verilog.
- Developed at UTwente.

- QBayLogic: Startup to use CλaSH in actual designs.
  Located in Enschede, in the Demcon building.
  Electronics & Applications: B.108.

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

Introduction

QBayLogic

- Since 2005 CPUs hardly any development.
- FPGAs on the rise (HDLs, HLS tools).

- CλaSH: Haskell ⇒ VHDL/Verilog.
- Developed at UTwente.

- QBayLogic: Startup to use CλaSH in actual designs.
  Located in Enschede, in the Demcon building.
  Electronics & Applications: B.108.

- 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

QBayLogic

- Since 2005 CPUs hardly any development.
- FPGAs on the rise (HDLs, HLS tools).

- CλaSH: Haskell ⇒ VHDL/Verilog.
- Developed at UTwente.

- QBayLogic: Startup to use CλaSH in actual designs.
  Located in Enschede, in the Demcon building.
  Electronics & Applications: B.108.

- 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

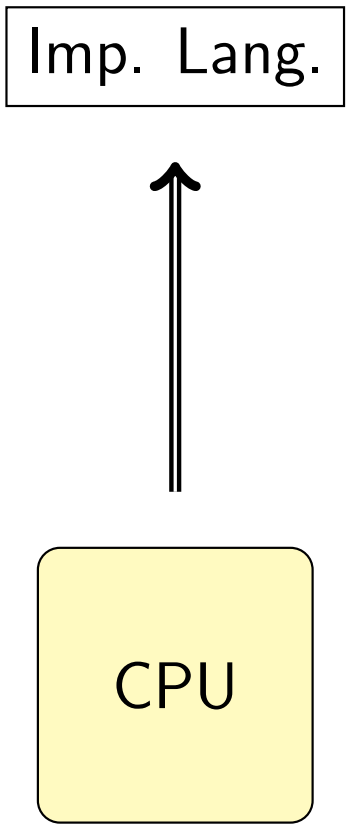
QBayLogic

- PID controller (inverted pendulum)
- Processor (derived from heat diffusion)
- Transformations

Examples:

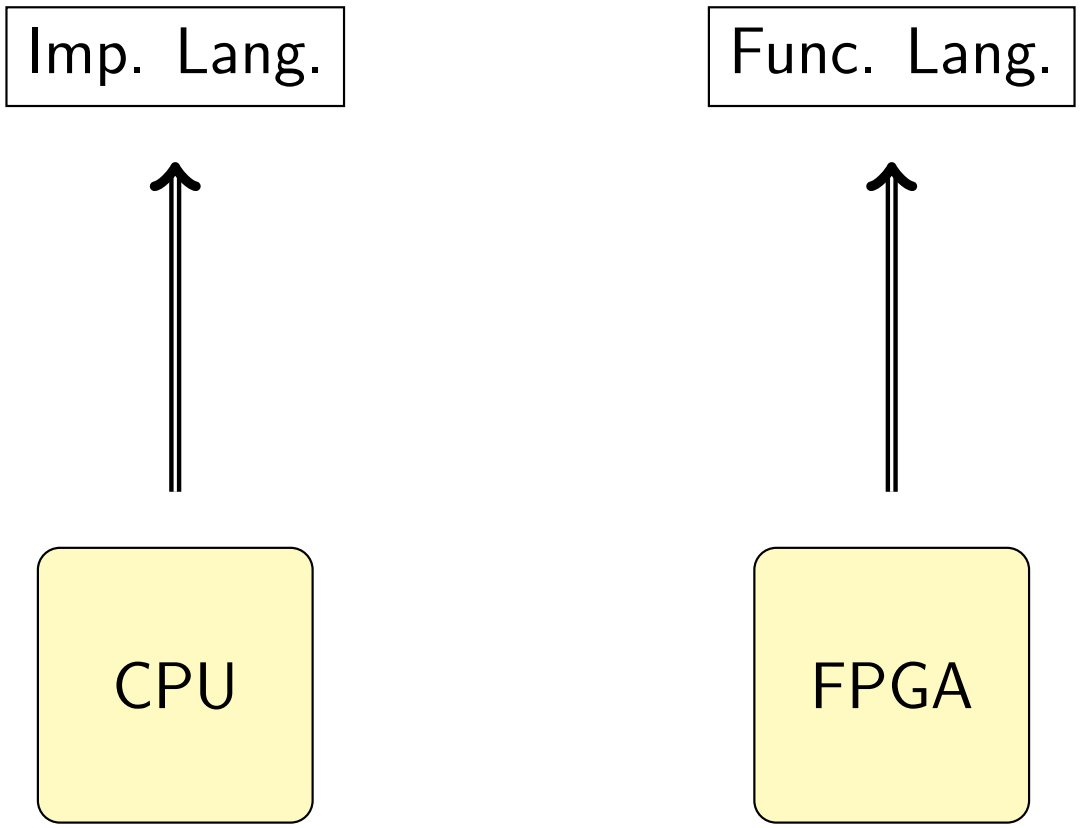
- Elementary architectures: adders, multipliers, FIR filters, FFT
- Cochlea Membrane, Adaptive Cruise Control, Slam algorithm, *N*-Queens, Processors, PID controllers, Tunnelling Ball Device

Languages and architectures



ALU
Assembly
Program counter

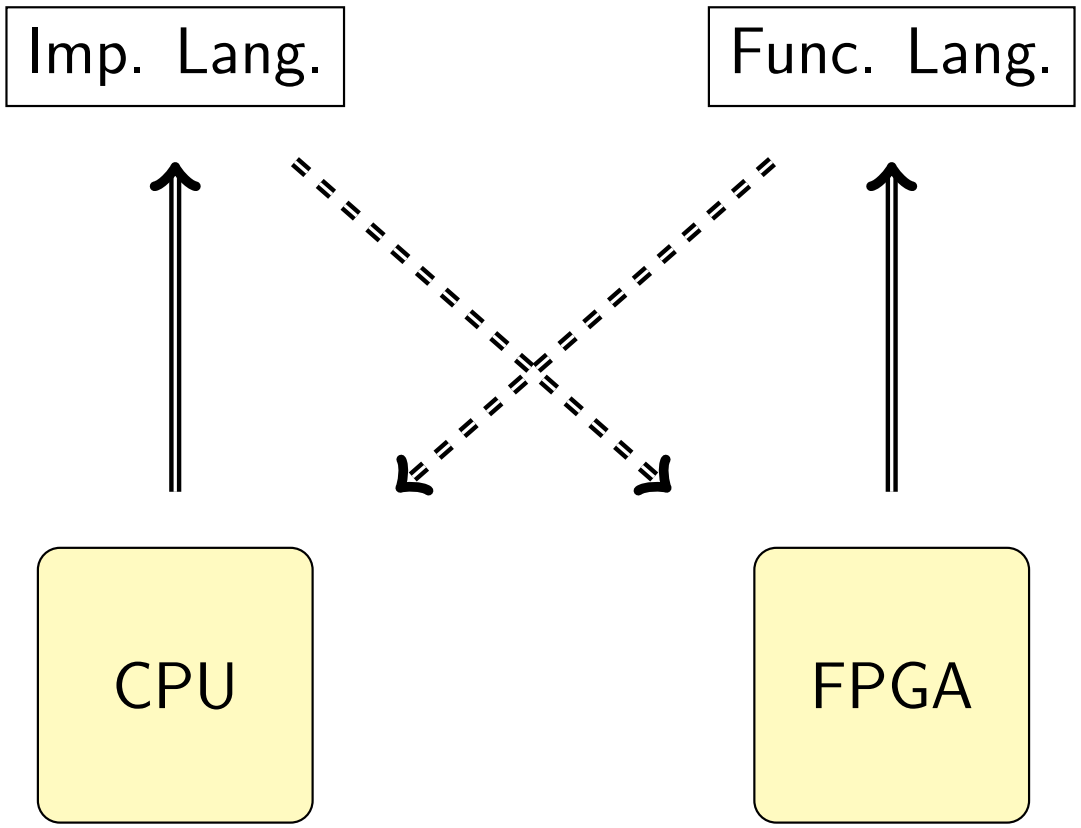
Languages and architectures



ALU
Assembly
Program counter

Logic gates
Circuits
Registers

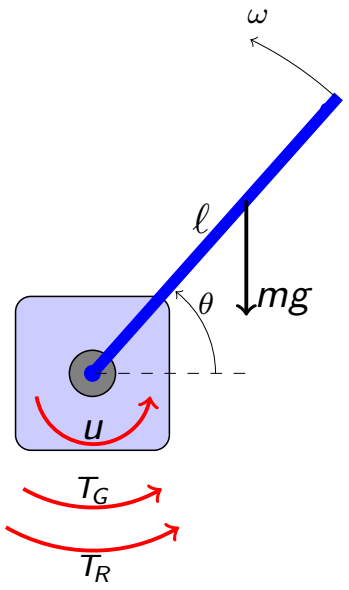
Languages and architectures



ALU
Assembly
Program counter

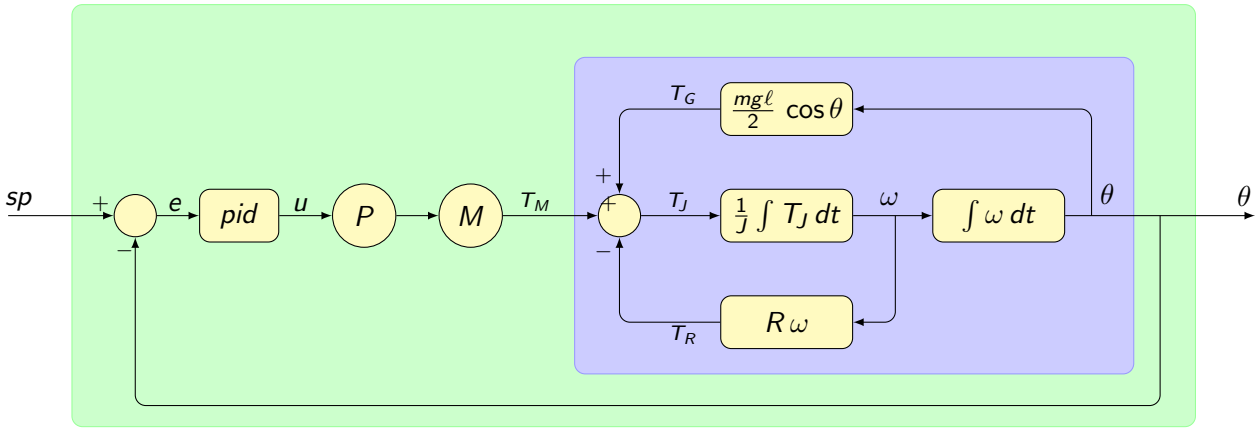
Logic gates
Circuits
Registers

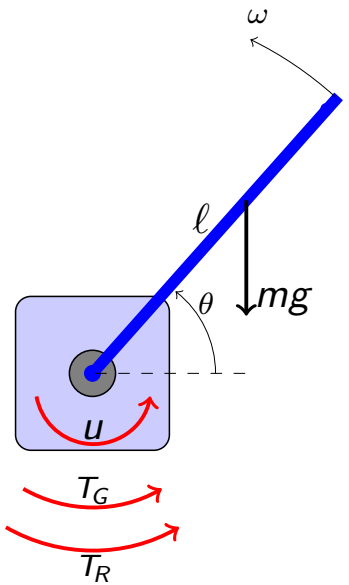
Pendulum



setpoints:
$45°, 135°$

Pendulum

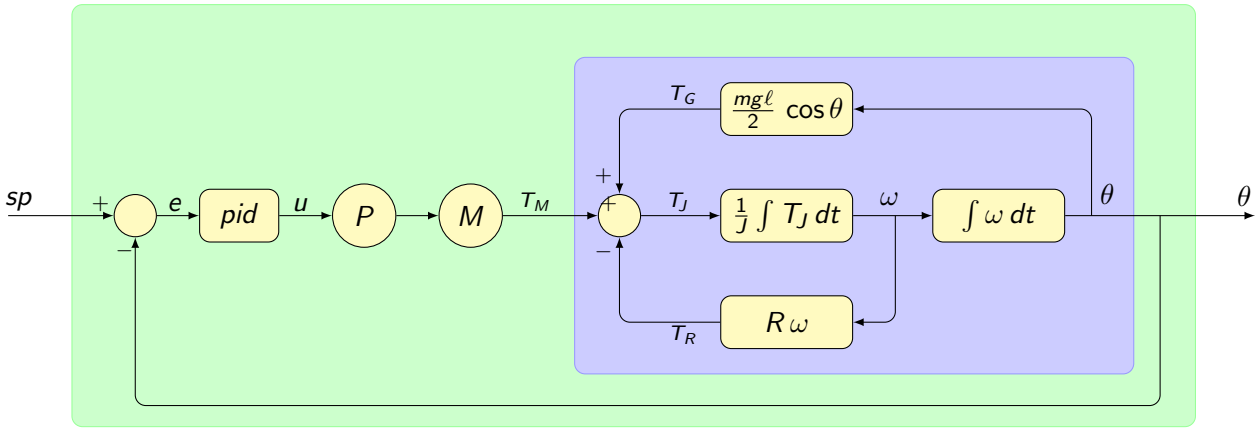


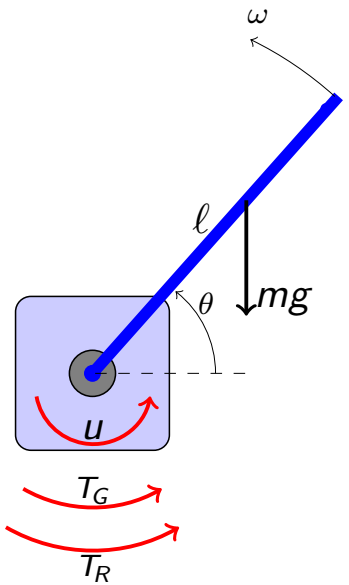


$$
\begin{aligned}
T_J(t) &= T_G + T_M - T_R \\
&\quad \textbf{where} \\
&\qquad T_G = \tfrac{mg\ell}{2}\cos(\theta(t)) \\
&\qquad T_M = M\,(P\,(pid(t))) \\
&\qquad T_R = R\,\omega(t) \\
\omega(t) &= \tfrac{1}{J}\int_0^t T_J(t)\,dt \\
\theta(t) &= \int_0^t \omega(t)dt
\end{aligned}
$$

setpoints:
$45°, 135°$

Pendulum

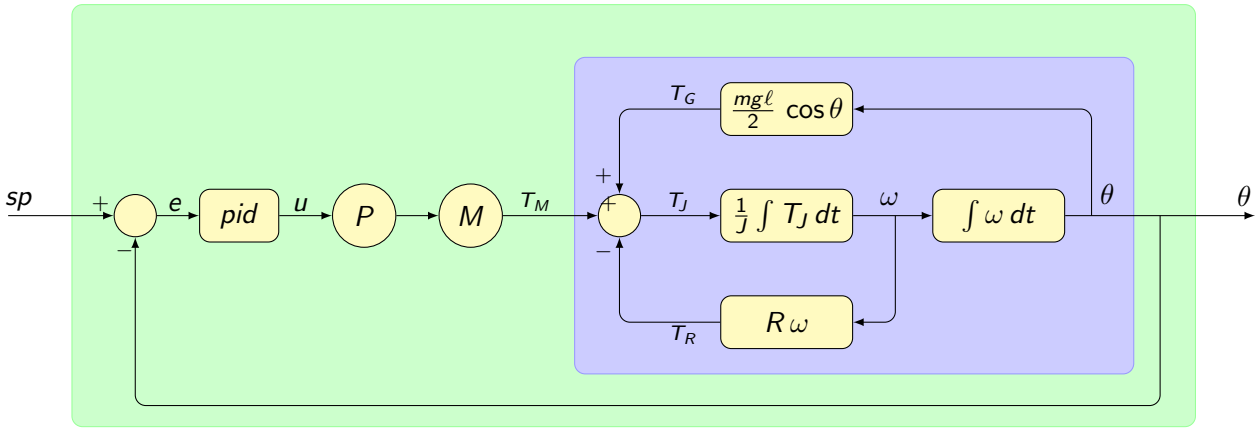


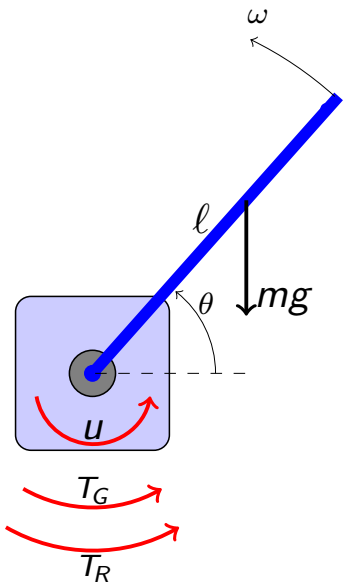


$$
\begin{aligned}
T_J(t) &= T_G + T_M - T_R \\
&\quad \textbf{where} \\
&\qquad T_G = \tfrac{mg\ell}{2}\cos(\theta(t)) \\
&\qquad T_M = M\,(P\,(pid(t))) \\
&\qquad T_R = R\,\omega(t) \\
\omega(t) &= \tfrac{1}{J}\int_0^t T_J(t)\,dt \\
\theta(t) &= \int_0^t \omega(t)dt
\end{aligned}
$$

```
trq_J t = trqG + trqM - trqR
        where
          trqG = m*g*l/2 * cos (theta t)
          trqM = motor $ power $ pid t
          trqR = res * omega t

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

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

Pendulum



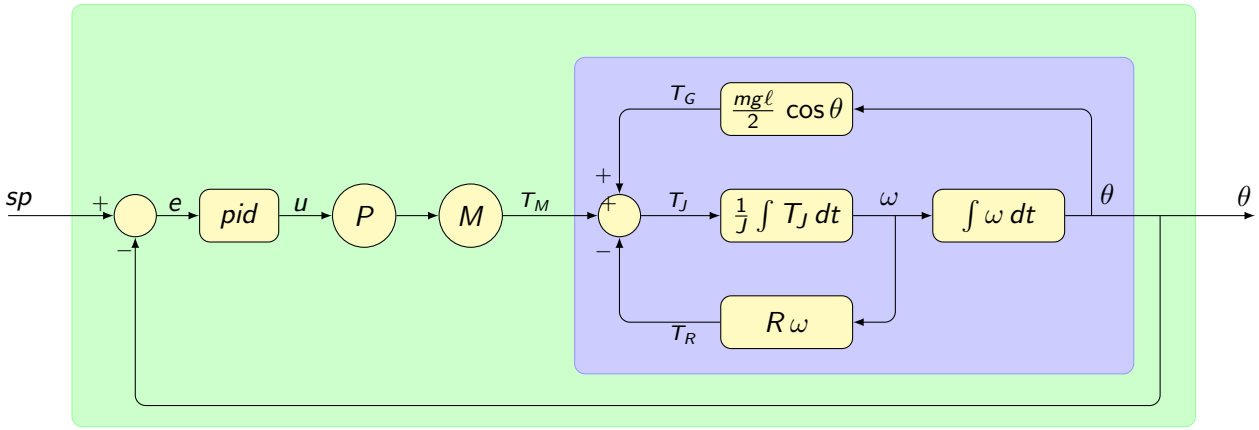


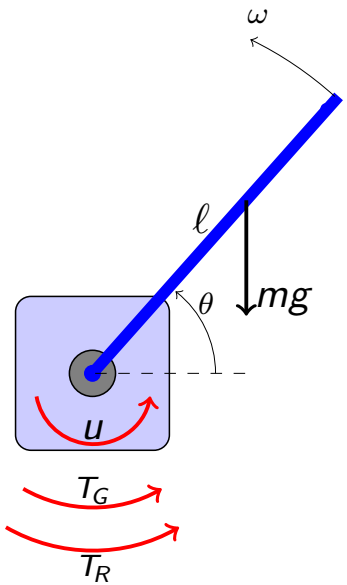
$$
\begin{aligned}
T_J(t) &= T_G + T_M - T_R \\
&\quad \textbf{where} \\
&\qquad T_G = \tfrac{mg\ell}{2}\cos(\theta(t)) \\
&\qquad T_M = M\,(P\,(pid(t))) \\
&\qquad T_R = R\,\omega(t) \\
\omega(t) &= \tfrac{1}{J}\int_0^t T_J(t)\,dt \\
\theta(t) &= \int_0^t \omega(t)dt
\end{aligned}
$$

```
trq_J t = trqG + trqM - trqR
        where
          trqG = m*g*l/2 * cos (theta t)
          trqM = motor $ power $ pid t
          trqR = 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 = sum [ f x * dx | x <- [a, a+dx .. b] ]
```

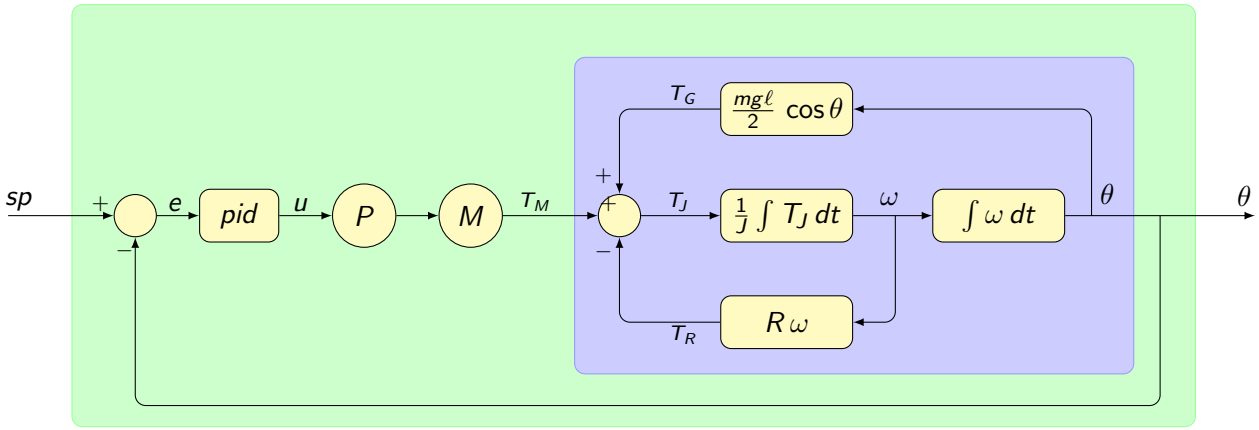
Pendulum

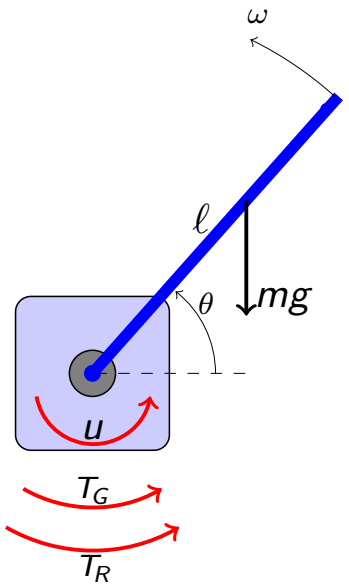





```
trq_J t = trqG + trqM - trqR
        where
          trqG = m*g*l/2 * cos (theta t)
          trqM = motor $ power $ pid t
          trqR = res * omega t

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

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

Pendulum

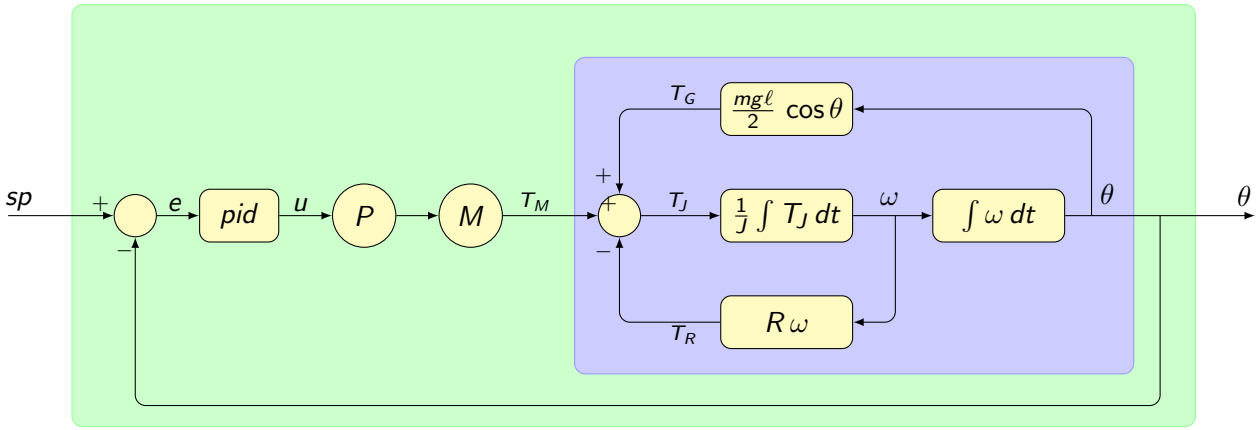


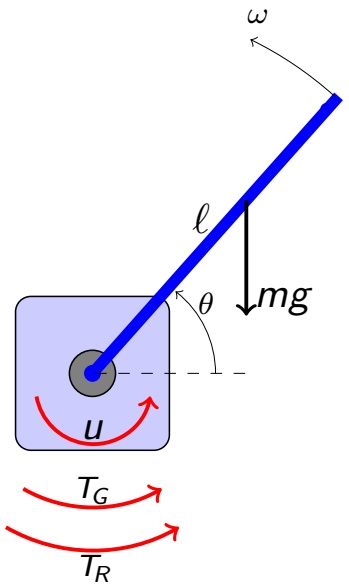



```
trq_J t = trqG + trqM - trqR
        where
          trqG = m*g*l/2 * cos (theta t)
          trqM = motor $ power $ pid t
          trqR = res * omega t

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

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

```
pendulum (om,th) trqM = ( (om',th') , thOut )
                where
                  trqJ = trqG + trqM - trqR

                  trqG = m*g*l/2 * cos th
                  trqR = res * omOut

                  (om',omOut) = omega om trqJ
                  (th',thOut) = theta th omOut

omega om trqJ = ( om' , om )
                where
                  om' = om + 1/j * trqJ*dt

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

Pendulum

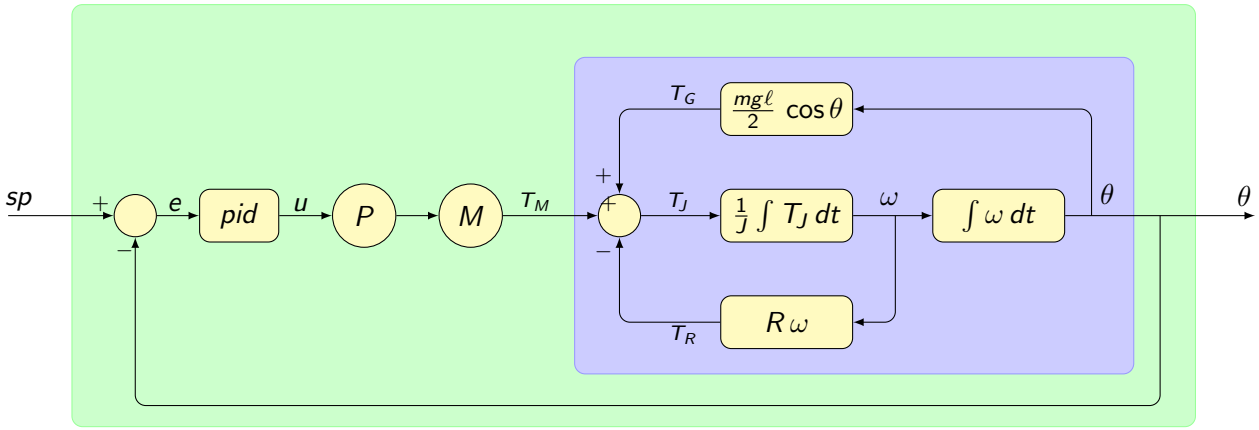


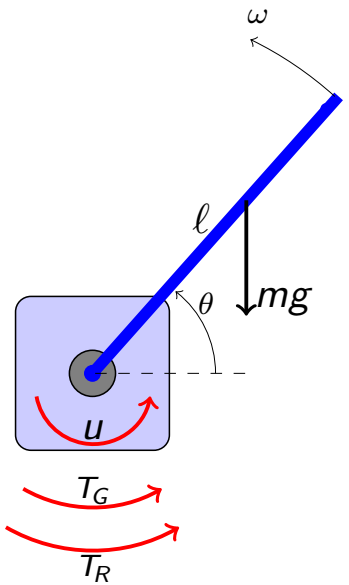



```
trq_J t = trqG + trqM - trqR
        where
          trqG = m*g*l/2 * cos (theta t)
          trqM = motor $ power $ pid t
          trqR = res * omega t

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

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

```
pendulum (om,th) trqM = ( (om',th') , thOut )
                where
                  trqJ = trqG + trqM - trqR

                  trqG = m*g*l/2 * cos th
                  trqR = res * omOut

                  (om',omOut) = omega om trqJ
                  (th',thOut) = theta th omOut

omega om trqJ = ( om' , om )
                where
                  om' = om + 1/j * trqJ*dt

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

Pendulum






```
trq_J t = trqG + trqM - trqR
        where
          trqG = m*g*l/2 * cos (theta t)
          trqM = motor $ power $ pid t
          trqR = res * omega t

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

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

```
pendulum (om,th) trqM = ( (om',th') , thOut )
              where
                trqJ = trqG + trqM - trqR

                trqG = m*g*l/2 * cos th
                trqR = res * omOut

                (om',omOut) = omega om trqJ
                (th',thOut) = theta th omOut

omega om trqJ = ( om' , om )
              where
                om' = om + 1/j * trqJ*dt

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

Architecture

Mealy Machine

Type transformations

- Number types

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

```
omega :: Float -> Float

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

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

Type transformations

QBayLogic

- Number types

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

```
omega :: Float -> Float

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

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

- Lists/Vectors

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

Type transformations

QBayLogic

- Number types

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

```
omega :: Float -> Float

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

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

- Lists/Vectors

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

- Top level

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

Type transformations

QBayLogic

- Number types

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

```
omega :: Float -> Float

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

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

- 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`

Systematic derivation

QBayLogic

- Mathematical specification
- Discretize
- Translate into Haskell
- Parameter accumulation, memoization
- Structural transformations
- Type transformations
- CλaSH rewrite system
- Synthesize

Systematic derivation

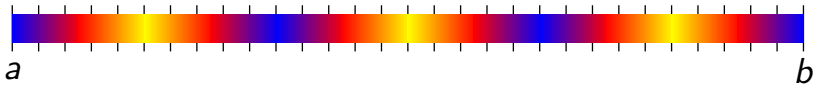
QBayLogic

- Mathematical specification
- Discretize
- Translate into Haskell
- Parameter accumulation, memoization
- Structural transformations
- Type transformations
- CλaSH rewrite system
- Synthesize

- Simulation, testability

Small processor

One dimensional heat diffusion:



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

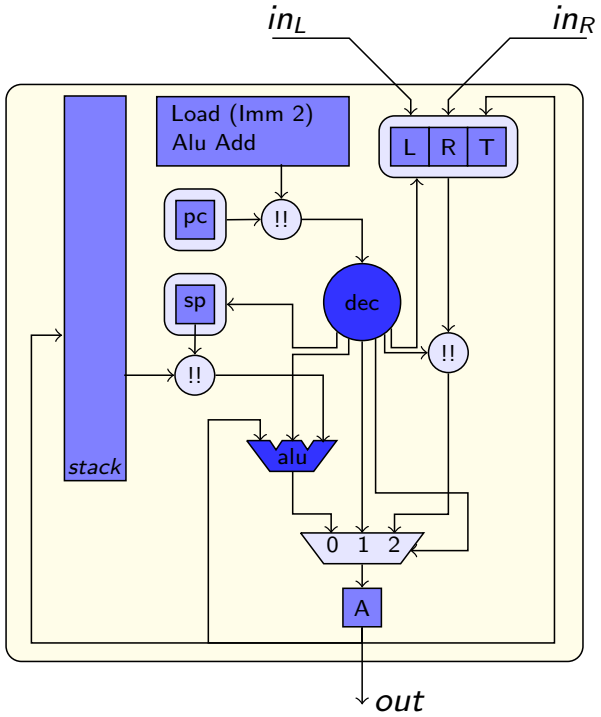
$\Longrightarrow$ derivation of (small) processor

Small processor

QBayLogic

```
data Value    = Imm Int | Reg Int
data Opcode   = Nop | Add | Mul | Sub
data Assembly = Read | Load Value | Alu Opcode
```

```
prog = [ Read
       , Load (Reg 2)
       , Load (Imm c)
       , Load (Reg 0)
       , Load (Imm 2)
       , Load (Reg 2)
       , Alu Mul
       , Alu Sub
       , Load (Reg 1)
       , Alu Add
       , Alu Mul
       , Alu Add
       ]
```

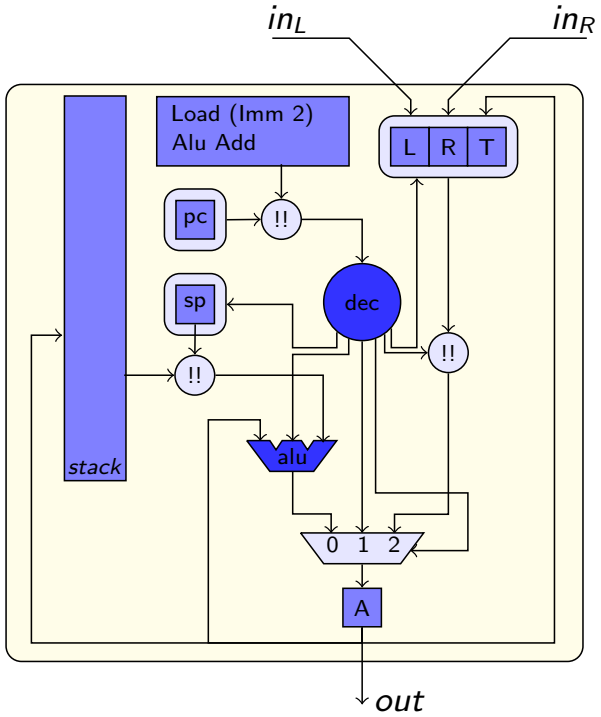


Small processor

QBayLogic

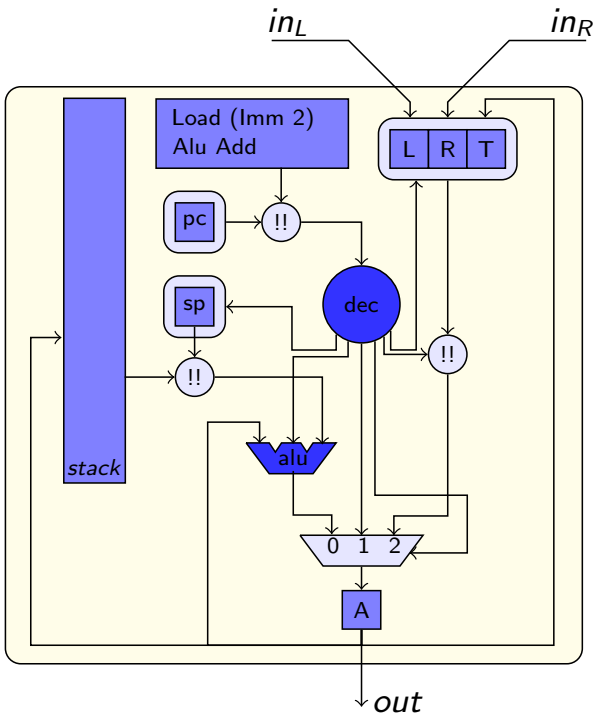
data *Value* = *Imm Int* | *Reg Int*
data *Opcode* = *Nop* | *Add* | *Mul* | *Sub*
data *Assembly* = *Read* | *Load Value* | *Alu Opcode*

```
prog = [ Read
       , Load (Reg 2)
       , Load (Imm c)
       , Load (Reg 0)
       , Load (Imm 2)
       , Load (Reg 2)
       , Alu Mul
       , Alu Sub
       , Load (Reg 1)
       , Alu Add
       , Alu Mul
       , Alu Add
       ]
```



Embedded Language = Data Type

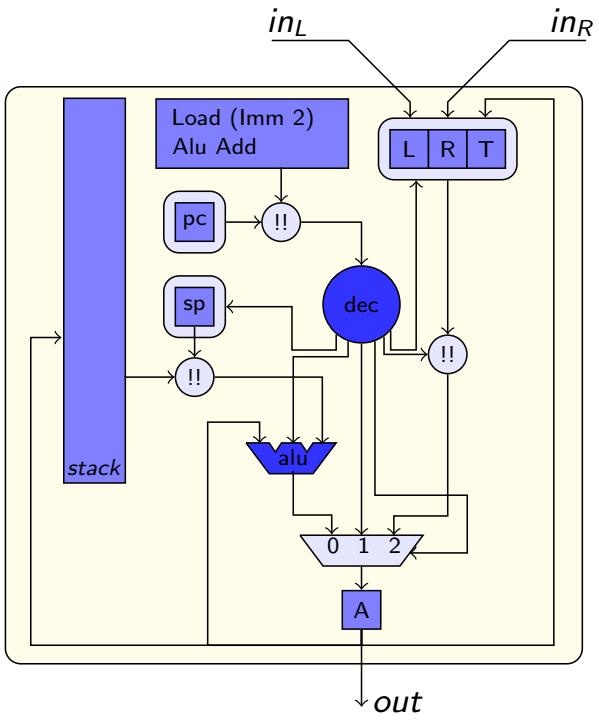
```
data Opcode = Nop | Add | Mul | Sub

alu opc x y = case opc of
                Nop  → y
                Add  → x + y
                Mul  → x * y
                Sub  → x − y
```



$mCode_0 = MCode\{rd{=}False,\ imm{=}0,\ addr{=}0,\ opc{=}Nop,\ toA{=}0,\ \ldots\}$

```
dec instr
  = case instr of
      Read          → mCode0 {rd=True,  toA=1, ... }
      Load (Reg r)  → mCode0 {addr=r,   toA=2, ... }
      Load (Imm c)  → mCode0 {imm=c,    toA=1, ... }
      Alu op        → mCode0 {opc=op,   toA=0, ... }
```

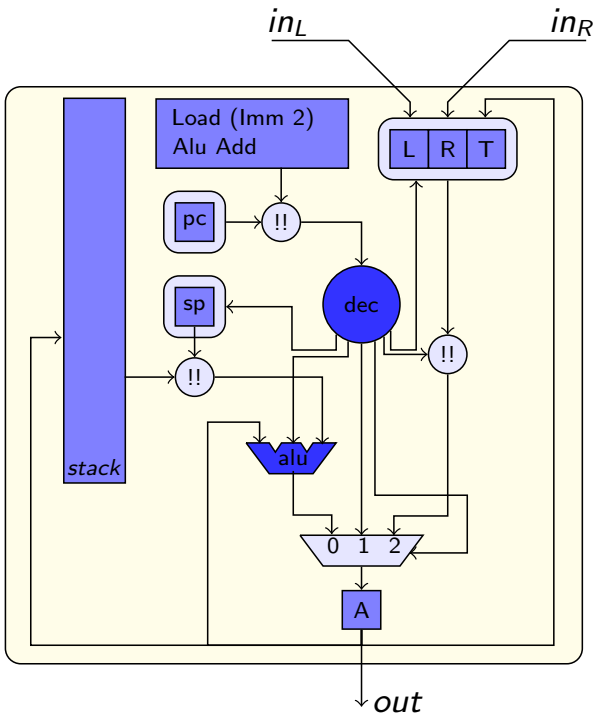
$$
\begin{array}{lll}
\textbf{data}\ \mathit{Opcode} & = & \mathit{Nop} \mid \mathit{Add} \mid \mathit{Mul} \mid \mathit{Sub} \\
\mathit{alu}\ \mathit{opc}\ x\ y & = & \textbf{case}\ \mathit{opc}\ \textbf{of} \\
 & & \quad \mathit{Nop} \rightarrow y \\
 & & \quad \mathit{Add} \rightarrow x + y \\
 & & \quad \mathit{Mul} \rightarrow x * y \\
 & & \quad \mathit{Sub} \rightarrow x - y
\end{array}
$$



$$mCode_0 = MCode\{rd{=}False,\ imm{=}0,\ addr{=}0,\ opc{=}Nop,\ toA{=}0,\ \ldots\}$$

$$
\begin{array}{lll}
\mathit{dec}\ \mathit{instr} & & \\
\quad = & \textbf{case}\ \mathit{instr}\ \textbf{of} & \\
 & \quad \mathit{Read} \rightarrow & mCode_0\ \{rd{=}True,\ \ toA{=}1,\ \ \ldots\ \} \\
 & \quad \mathit{Load}\ (\mathit{Reg}\ r) \rightarrow & mCode_0\ \{addr{=}r,\ \ toA{=}2,\ \ \ldots\ \} \\
 & \quad \mathit{Load}\ (\mathit{Imm}\ c) \rightarrow & mCode_0\ \{imm{=}c,\ \ toA{=}1,\ \ \ldots\ \} \\
 & \quad \mathit{Alu}\ op \rightarrow & mCode_0\ \{opc{=}op,\ \ toA{=}0,\ \ \ldots\ \}
\end{array}
$$

$$
\begin{array}{lcl}
\textbf{data}\ \mathit{Opcode} & = & \mathit{Nop} \mid \mathit{Add} \mid \mathit{Mul} \mid \mathit{Sub} \\
\mathit{alu}\ \mathit{opc}\ x\ y & = & \textbf{case}\ \mathit{opc}\ \textbf{of} \\
 & & \quad \begin{array}{lcl} \mathit{Nop} & \rightarrow & y \\ \mathit{Add} & \rightarrow & x + y \\ \mathit{Mul} & \rightarrow & x * y \\ \mathit{Sub} & \rightarrow & x - y \end{array}
\end{array}
$$



$$
\mathit{mCode}_0 = \mathit{MCode}\{\mathit{rd}{=}\mathit{False},\ \mathit{imm}{=}0,\ \mathit{addr}{=}0,\ \mathit{opc}{=}\mathit{Nop},\ \mathit{toA}{=}0,\ \ldots\}
$$

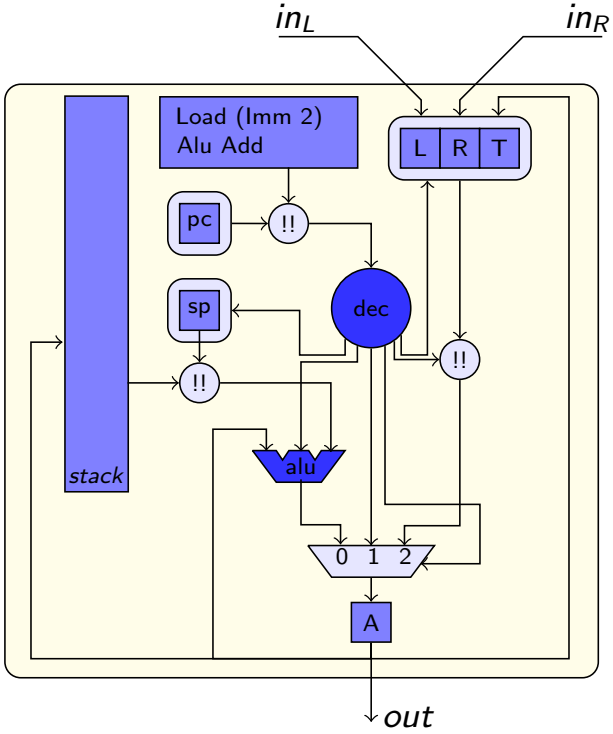
$$
\begin{array}{lcl}
\mathit{dec}\ \mathit{instr} & & \\
\quad = & & \textbf{case}\ \mathit{instr}\ \textbf{of} \\
 & & \quad \begin{array}{lcll}
\mathit{Read} & \rightarrow & \mathit{mCode}_0\,\{\mathit{rd}{=}\mathit{True}, & \mathit{toA}{=}1, \quad \ldots\ \} \\
\mathit{Load}\ (\mathit{Reg}\ r) & \rightarrow & \mathit{mCode}_0\,\{\mathit{addr}{=}r, & \mathit{toA}{=}2, \quad \ldots\ \} \\
\mathit{Load}\ (\mathit{Imm}\ c) & \rightarrow & \mathit{mCode}_0\,\{\mathit{imm}{=}c, & \mathit{toA}{=}1, \quad \ldots\ \} \\
\mathit{Alu}\ \mathit{op} & \rightarrow & \mathit{mCode}_0\,\{\mathit{opc}{=}\mathit{op}, & \mathit{toA}{=}0, \quad \ldots\ \}
\end{array}
\end{array}
$$

$$proc \;\; instrs \;\; (stack, reg_A, regs, \; pc, sp) \;\; (in_L, in_R)$$

$$= \; (\; (stack', reg'_A, regs', \; pc', sp') \; , \; out \;)$$

where

$$
\begin{array}{lcl}
MCode\{..\} & = & dec \; (instrs!!pc) \\
stack' & = & (sp{+}1, reg_A) \rightsquigarrow stack \\
reg'_A & = & \textbf{case} \; toA \; \textbf{of} \\
 & & \quad 0 \;\rightarrow\; alu \; opc \; (stack!!sp) \; reg_A \\
 & & \quad 1 \;\rightarrow\; imm \\
 & & \quad 2 \;\rightarrow\; regs!!addr \\
regs' & | & rd \quad = \; [in_L, in_R, reg_A] \\
 & | & otherwise \; = \; regs \\
pc' & | & pc < length \; instrs - 1 \; = \; pc + 1 \\
 & | & otherwise \quad = \; 0 \\
sp' & = & \textbf{case} \; spC \; \textbf{of} \; \cdots \\
out & = & reg_A
\end{array}
$$

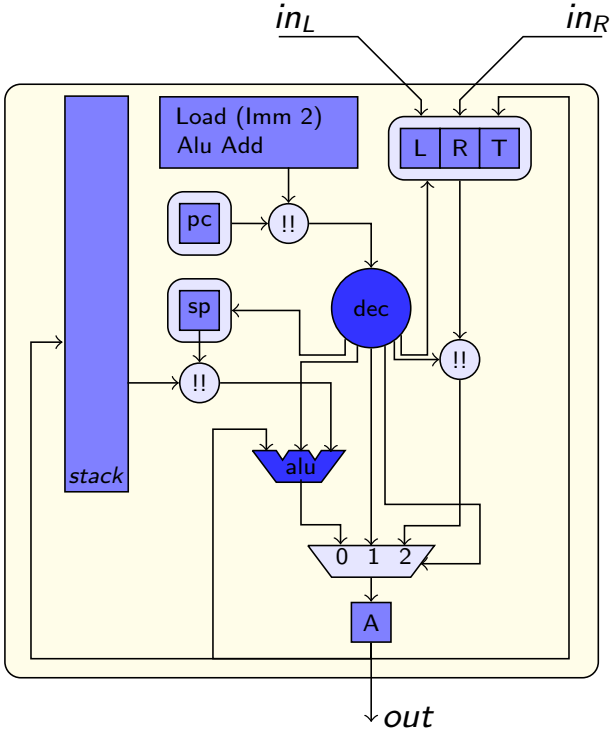


$$\mathit{proc}\ \mathit{instrs}\ (\mathit{stack}, \mathit{reg}_A, \mathit{regs},\ \mathit{pc}, \mathit{sp})\ \ (\mathit{in}_L, \mathit{in}_R)$$

$$= \ (\ (\mathit{stack}', \mathit{reg}'_A, \mathit{regs}',\ \mathit{pc}', \mathit{sp}')\ ,\ \mathit{out}\ )$$

where

$$\begin{array}{lcl}
\mathit{MCode}\{..\} & = & \mathit{dec}\ (\mathit{instrs}!!\mathit{pc}) \\[1ex]
\mathit{stack}' & = & (\mathit{sp}{+}1, \mathit{reg}_A) \rightsquigarrow \mathit{stack} \\[1ex]
\mathit{reg}'_A & = & \textbf{case}\ \mathit{toA}\ \textbf{of} \\
 & & \quad 0 \ \rightarrow\ \mathit{alu}\ \mathit{opc}\ (\mathit{stack}!!\mathit{sp})\ \mathit{reg}_A \\
 & & \quad 1 \ \rightarrow\ \mathit{imm} \\
 & & \quad 2 \ \rightarrow\ \mathit{regs}!!\mathit{addr} \\[1ex]
\mathit{regs}' & \mid & \mathit{rd} \quad = \ [\mathit{in}_L, \mathit{in}_R, \mathit{reg}_A] \\
 & \mid & \mathit{otherwise} \ = \ \mathit{regs} \\[1ex]
\mathit{pc}' & \mid & \mathit{pc} < \mathit{length}\ \mathit{instrs} - 1 \ = \ \mathit{pc} + 1 \\
 & \mid & \mathit{otherwise} \quad = \ 0 \\[1ex]
\mathit{sp}' & = & \textbf{case}\ \mathit{spC}\ \textbf{of}\ \cdots \\[1ex]
\mathit{out} & = & \mathit{reg}_A
\end{array}$$

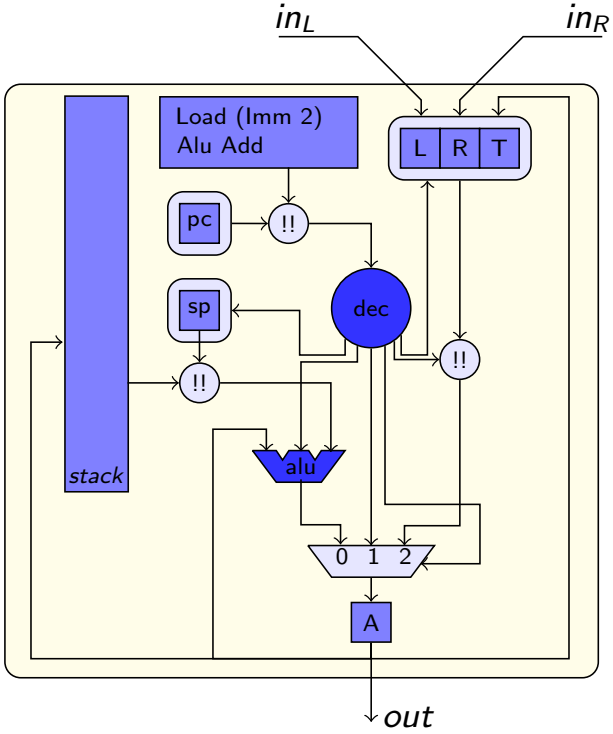


$$proc \;\; instrs \; (stack, reg_A, regs, \; pc, sp) \; (in_L, in_R)$$

$$= \; (\; (stack', reg'_A, regs', \; pc', sp') \; , \; out \;)$$

where

$$
\begin{array}{lcl}
MCode\{..\} & = & dec \; (instrs!!pc) \\
stack' & = & (sp{+}1, reg_A) \rightsquigarrow stack \\
reg'_A & = & \textbf{case} \; toA \; \textbf{of} \\
 & & \quad 0 \;\rightarrow\; alu \; opc \; (stack!!sp) \; reg_A \\
 & & \quad 1 \;\rightarrow\; imm \\
 & & \quad 2 \;\rightarrow\; regs!!addr \\
regs' & | & rd \quad = \; [in_L, in_R, reg_A] \\
 & | & otherwise \; = \; regs \\
pc' & | & pc < length \; instrs - 1 \; = \; pc + 1 \\
 & | & otherwise \quad = \; 0 \\
sp' & = & \textbf{case} \; spC \; \textbf{of} \; \cdots \\
out & = & reg_A
\end{array}
$$

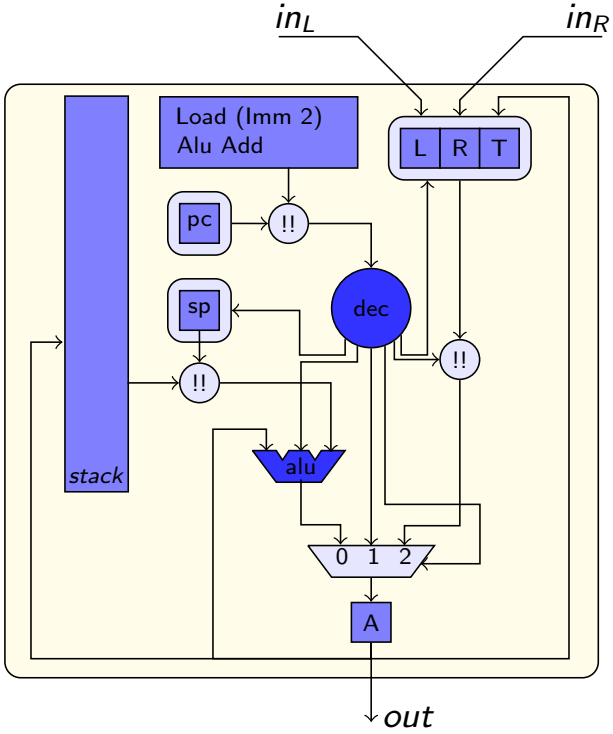


$$proc\ \ instrs\ \ (stack, reg_A, regs,\ \ pc, sp)\ \ (in_L, in_R)$$

$$= \ (\ (stack', reg'_A, regs',\ \ pc', sp')\ ,\ out\ )$$

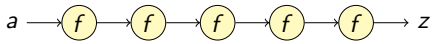
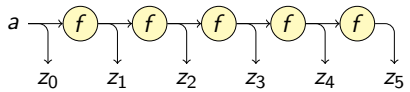
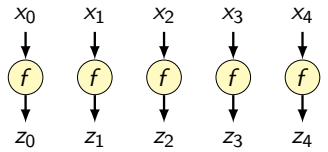
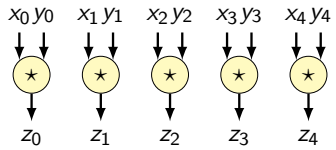
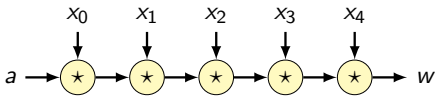
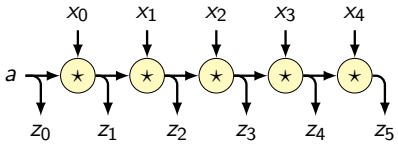
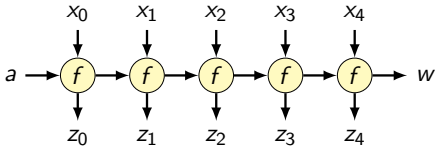
where

$$
\begin{array}{lcl}
MCode\{..\} & = & dec\ (instrs!!pc) \\
stack' & = & (sp{+}1, reg_A) \rightsquigarrow stack \\
reg'_A & = & \textbf{case}\ toA\ \textbf{of} \\
 & & \quad 0 \ \rightarrow\ alu\ opc\ (stack!!sp)\ reg_A \\
 & & \quad 1 \ \rightarrow\ imm \\
 & & \quad 2 \ \rightarrow\ regs!!addr \\
regs' & | & rd \quad = \ [in_L, in_R, reg_A] \\
 & | & otherwise \ = \ regs \\
pc' & | & pc < length\ instrs - 1 \ = \ pc + 1 \\
 & | & otherwise \ = \ 0 \\
sp' & = & \textbf{case}\ spC\ \textbf{of}\ \cdots \\
out & = & reg_A
\end{array}
$$



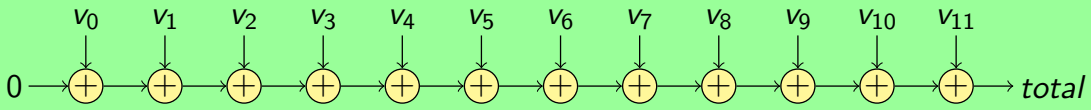
Higher Order Functions

Higher Order Functions

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		$f\ a\ x \Rightarrow (a', z)$	?	$(w, zs) = mapAccumL\ f\ a\ xs$

Transformation rules

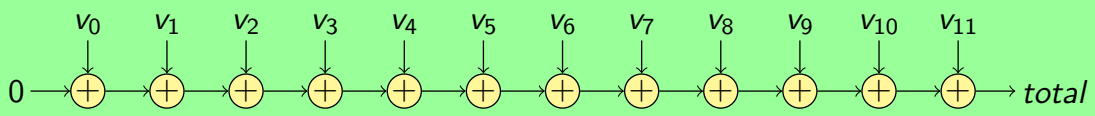
Transformation rules



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

$$total = 0 \oplus vs$$

Transformation rules

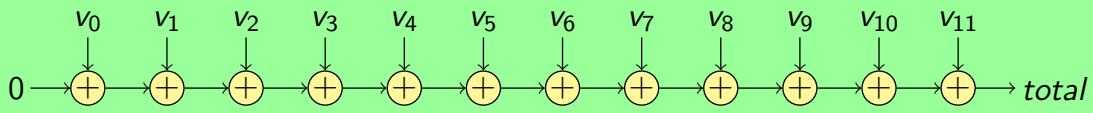


$$total = foldl\ (+)\ 0\ vs$$

$$total = 0 \oplus vs$$

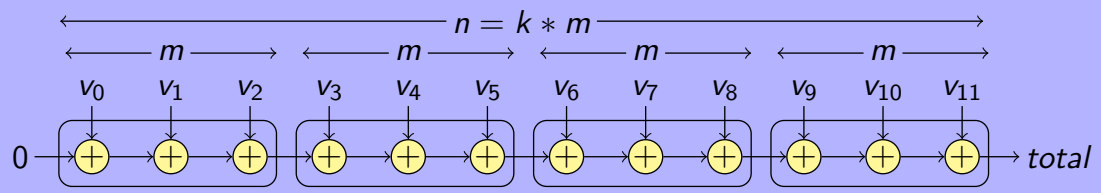
$$\begin{aligned} foldl\ f\ a\ [\,] &= a \\ foldl\ f\ a\ (x{:}xs) &= foldl\ f\ (f\ a\ x)\ xs \end{aligned}$$

Transformation rules

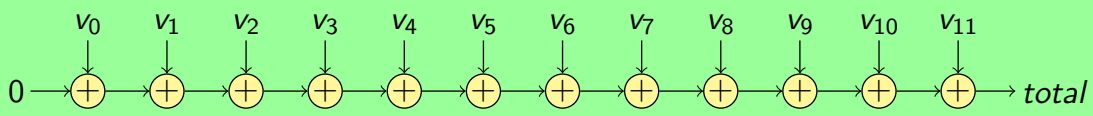


$$total = foldl\ (+)\ 0\ vs$$

$$total = 0 \oplus vs$$

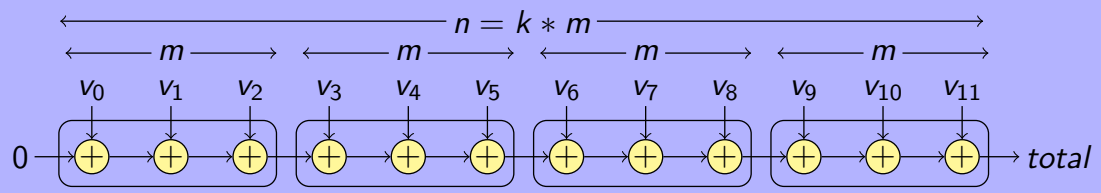


Transformation rules



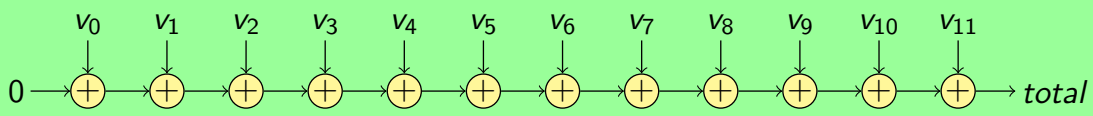
$$total = foldl\ (+)\ 0\ vs$$

$$total = 0 \oplus vs$$



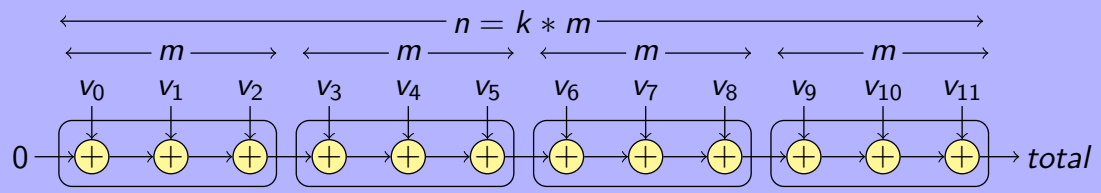
$$total = foldl\ (foldl\ (+))\ 0\ vss$$

Transformation rules



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

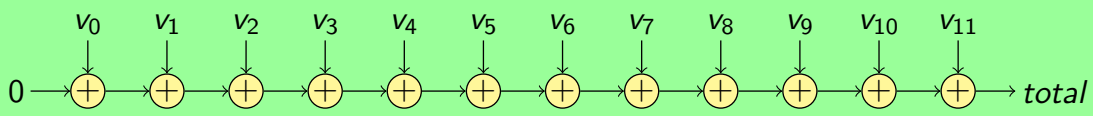
$$total = 0 \oplus vs$$



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

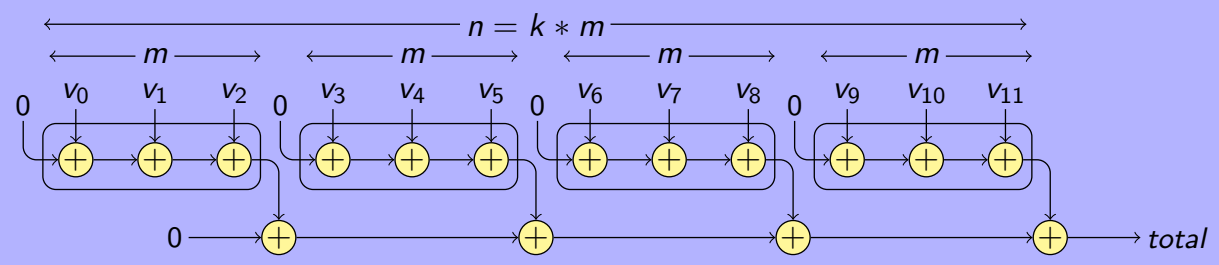
$$total = 0 \circledcirc vss$$

Transformation rules

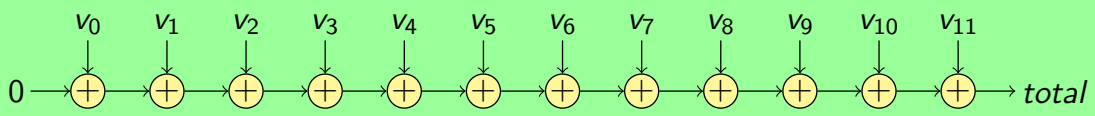


$$total = foldl\ (+)\ 0\ vs$$

$$total = 0 \oplus vs$$

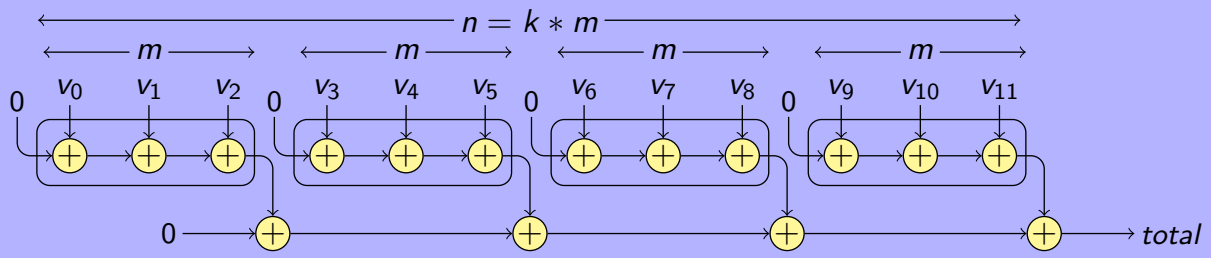


Transformation rules



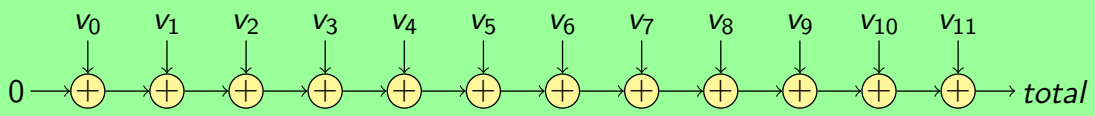
$$total = foldl\ (+)\ 0\ vs$$

$$total = 0 \oplus vs$$



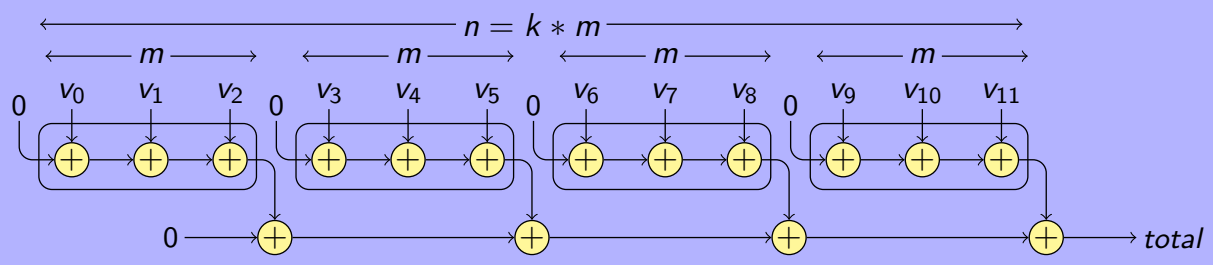
associative, neutral element

Transformation rules



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

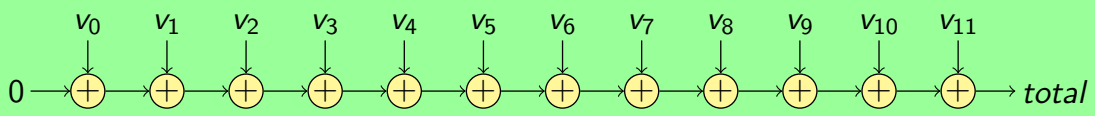
$$total = 0 \oplus vs$$



associative, neutral element

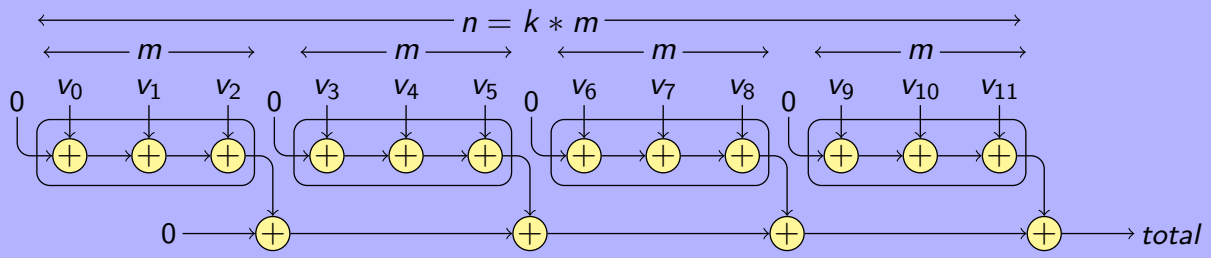
$$total = foldl\ (+)\ 0\ \$\ map\ (foldl\ (+)\ 0)\ vss$$

Transformation rules



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

$$total = 0 \oplus vs$$



associative, neutral element

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

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

Perspective

QBayLogic

- Start-up, started in July 2016
- Orders: deep learning (Cambridge), ¡confidential¿
- Co-operation in Demcon, a.o.: laser communication with satellites, 3D-printing

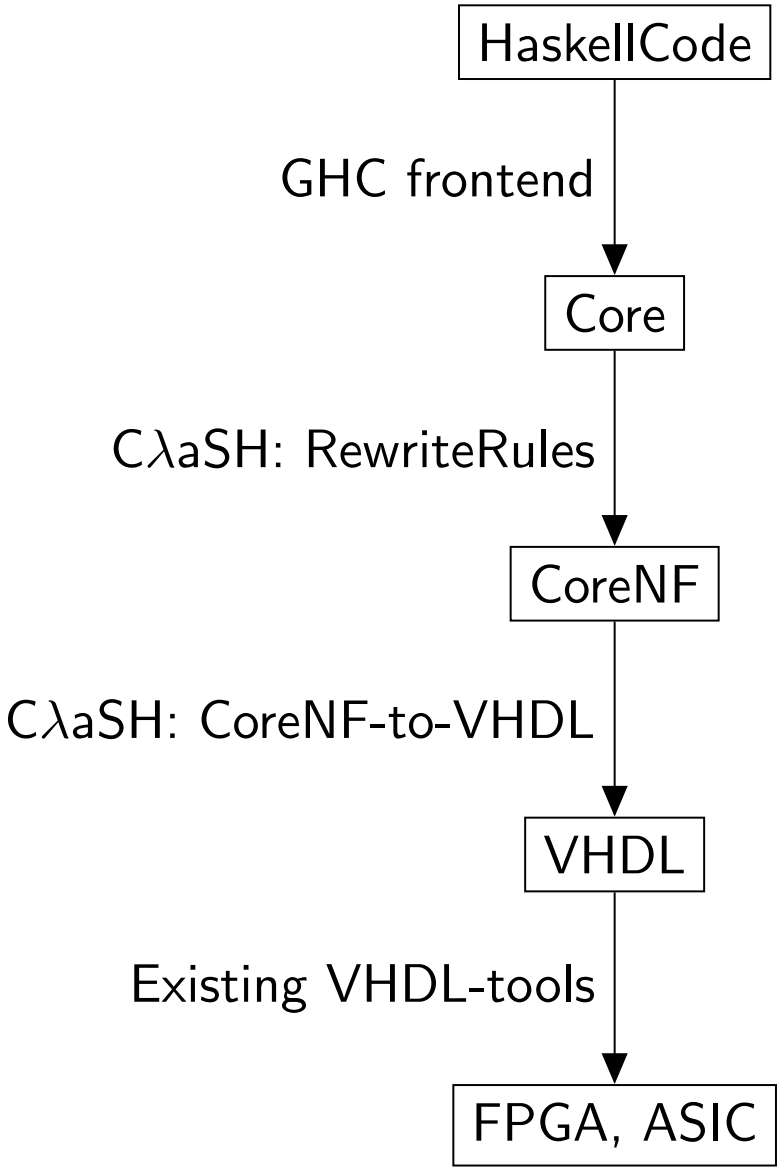
- E&A stand Demcon: B.108

Thank you

qbaylogic.com

The CλaSH Mechanism

CλaSH pipeline



Parsing
Desugaring
Type-checking

Remove polymorphism
Inlining
η-expansion
...

Translation into VHDL

Synthesis
Place-and-route

Rewrite system

Specification:

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

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

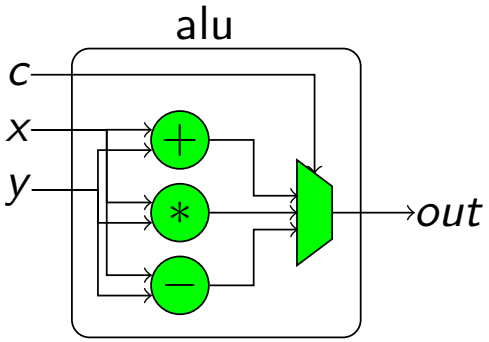
Note: $alu\ ADD\ ::\ Num\ a\ \Rightarrow a \rightarrow a \rightarrow a$

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

Rewrite system

Specification:

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



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

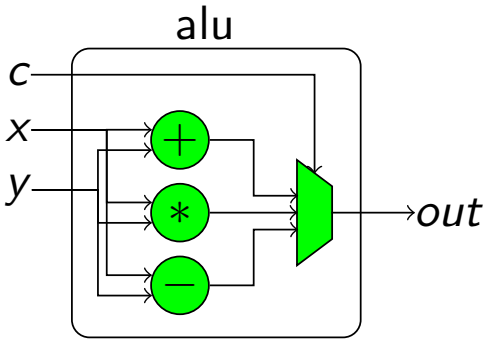
Note: $alu\ ADD\ ::\ Num\ a\ \Rightarrow a \rightarrow a \rightarrow a$

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

Rewrite system

Specification:

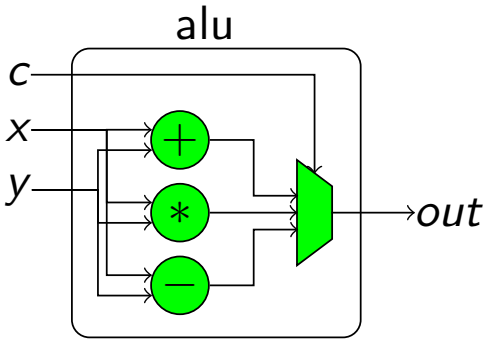
$$
\begin{aligned}
alu\ ADD &= (+) \\
alu\ MUL &= (*) \\
alu\ SUB &= (-)
\end{aligned}
$$



Rewrite system

Specification:

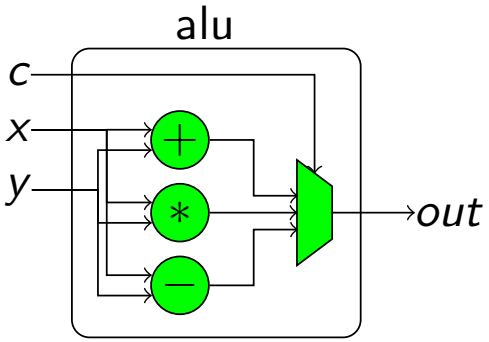
$$
\begin{aligned}
alu\ ADD &= (+) \\
alu\ MUL &= (*) \\
alu\ SUB &= (-)
\end{aligned}
$$



GHC ⇒ Core:

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

Rewrite system



Specification:

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

GHC ⇒ Core:

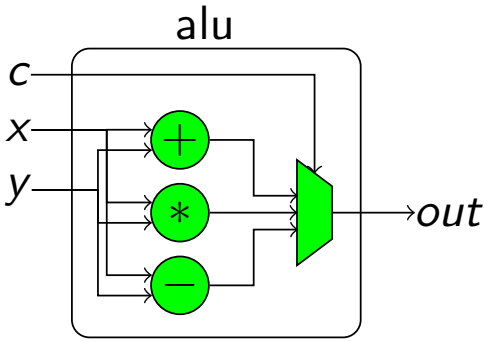
$$
\begin{aligned}
alu \quad = \quad & \lambda c.\, \textbf{case}\ c\ \textbf{of} \\
& \qquad ADD \rightarrow (+) \\
& \qquad MUL \rightarrow (*) \\
& \qquad SUB \rightarrow (-)
\end{aligned}
$$

$\eta-$expansion:

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

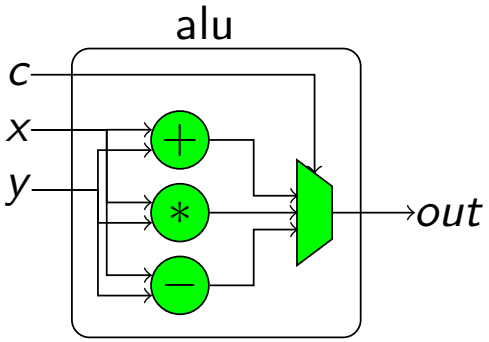
Rewrite system

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



Rewrite system

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

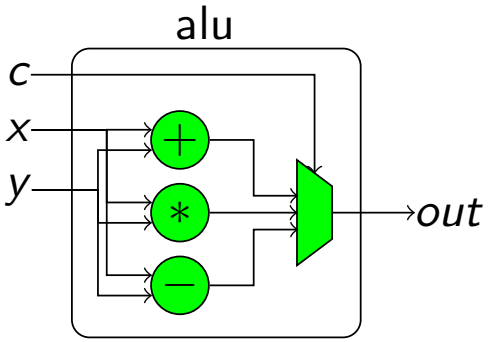


application propagation:

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

Rewrite system

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



application propagation:

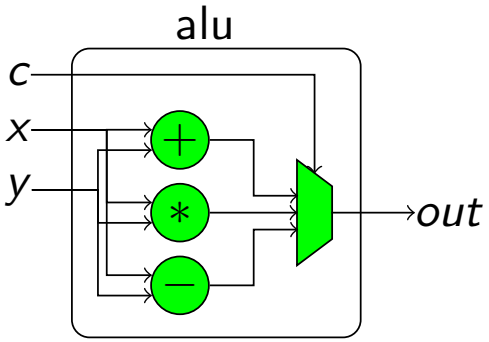
$$\begin{aligned} alu \;=\; \lambda c\,x\,y.\ \textbf{case}\ c\ \textbf{of}\\ & ADD \to (+)\ x\ y\\ & MUL \to (*)\ x\ y\\ & SUB \to (-)\ x\ y \end{aligned}$$

or, equivalently:

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

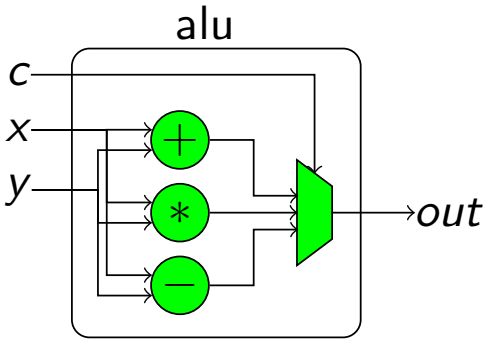
Rewrite system

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



Rewrite system

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

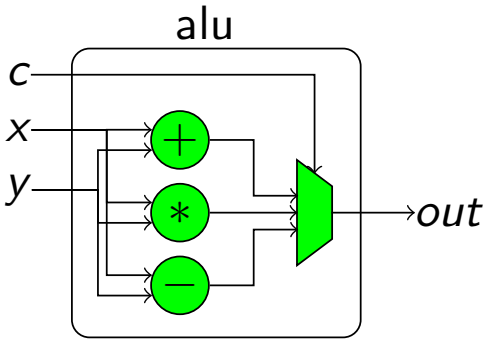


letification:

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

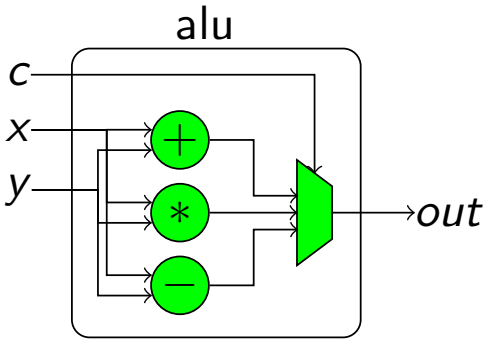
Rewrite system

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



Rewrite system

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



subexpression extraction:

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