FUN with Lego Mindstorms

Jan Tobias Mühlberg & Matthew Naylor
<muehlber|mfn>@cs.york.ac.uk

University of York, UK

PLASMA Seminar, York, 01st March 2007

Reactive Systems Design

- Gerald, Matthew and Tobias
- Contents:
  - Fixed-Point Theory
  - Lustre, Esterel, Statecharts
  - Compilation and Verification
  - Interesting bit: Practicals :-)

RSD Practicals

- Several tutorials on using SCADE
- Constructing a Lego Bricksorter
- Programming in SCADE using data-flow diagrams and Safe State Machines
- Design Verification

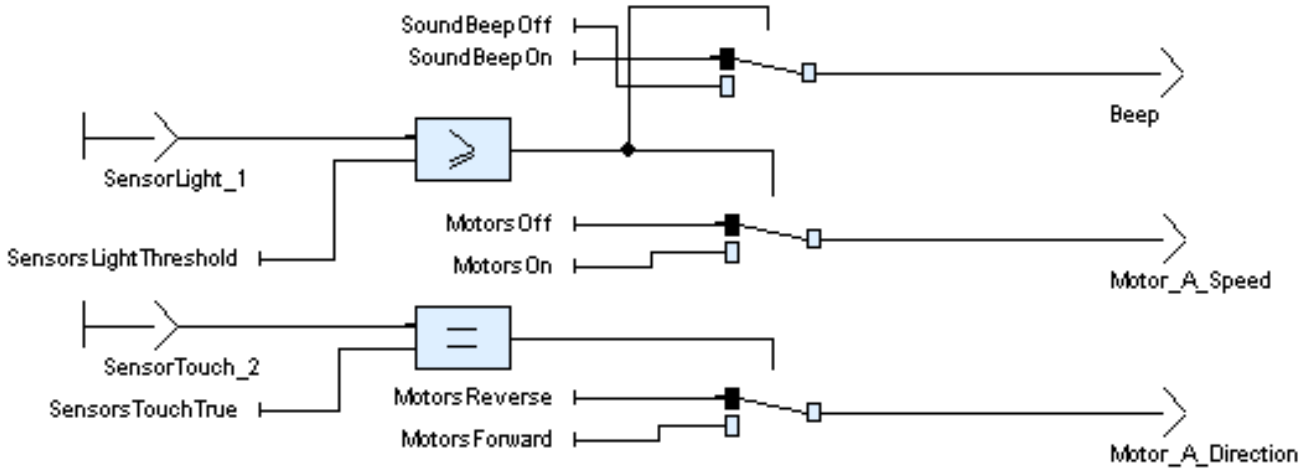
SCADE

- "The Standard for the Development of Safety-Critical Embedded Software in the Avionics Industry"
- Programming in data-flow diagrams (graphical Lustre) and Safe State Machines (Statecharts)
- Facilitate simulation and verification of safety properties
- Code generation (C)

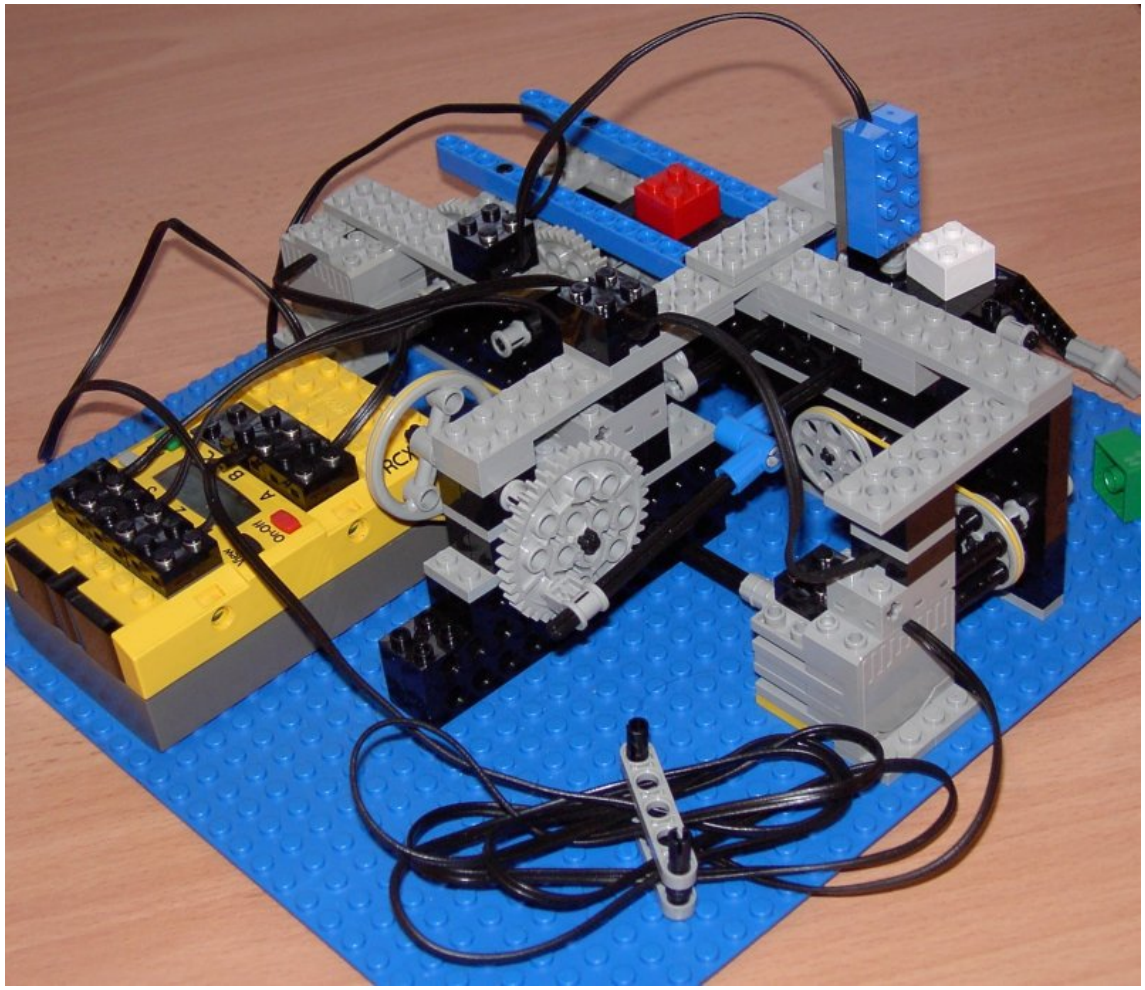
SCADE to Lego

- A simple example:

Title : Example for scade2brick		
Description: A Mindstorms robot with a light sensor, a touch sensor and a motor.		
Created by : Jan Tobias Muehlberg	20 Nov 2006	0.9



The Task



The Task

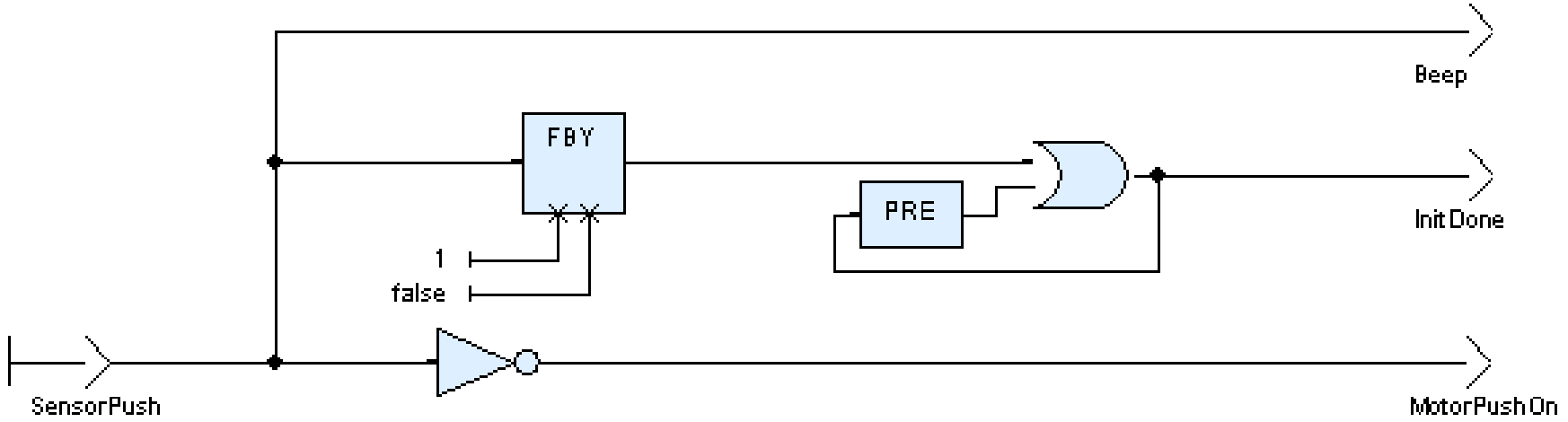
- Two phases:
  - Initialisation: Move the pusher into a position in which it does not block the belt.
  - Normal operation: White bricks (threshold >= 40) shall be pushed out, black bricks shall stay on the conveyor belt. The RCX shall beep on detection of a white brick.

Additional Tasks

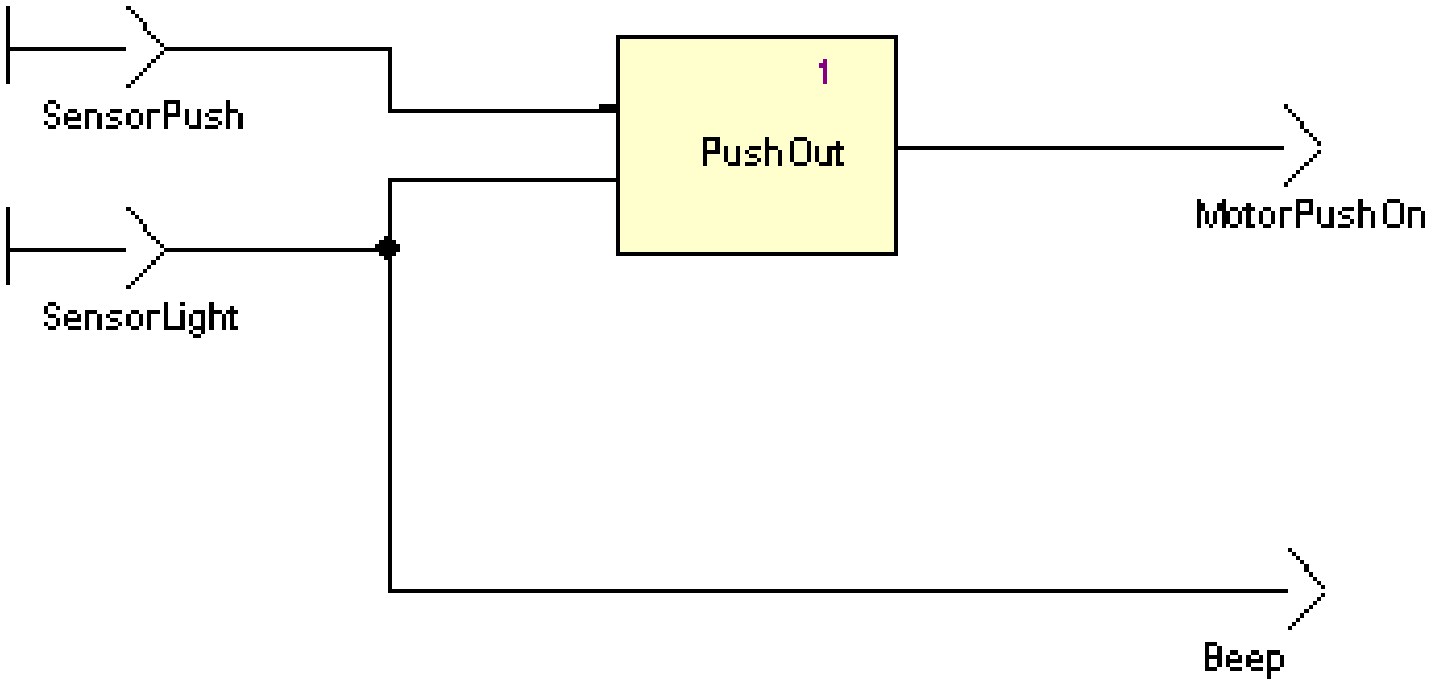
- Dealing with machines such as our Bricksorter requires some health and safety measures to be put in place. Modify your program so that all motors are turned off as long as the `Program` button is pressed.

- The Bricksorter does not work very well for bricks larger than 2x2. In fact it tends to be self-destructive. Think about a way to deal with long bricks.

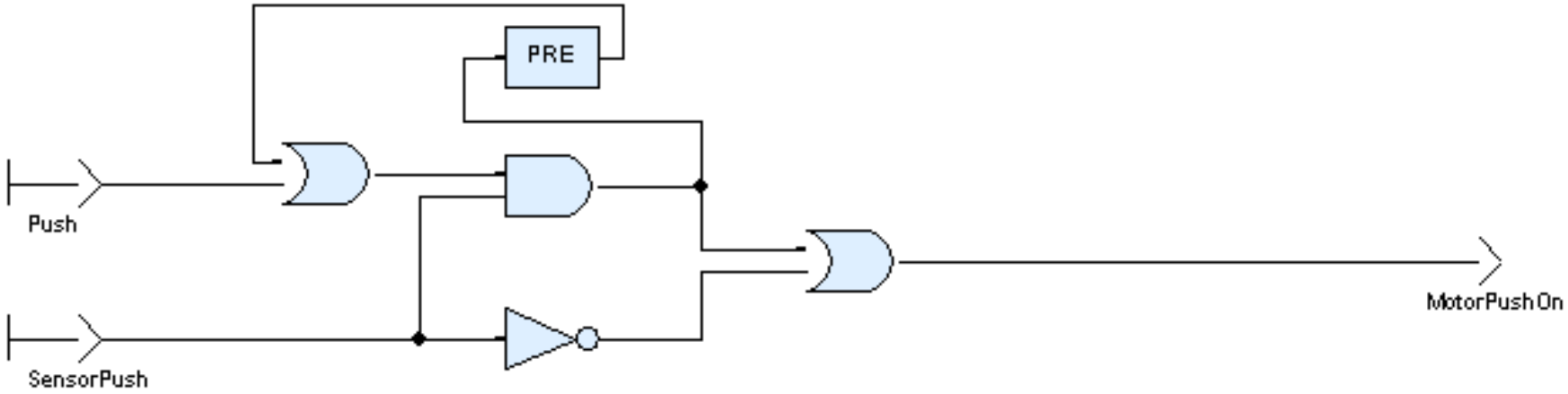
Node Initialise



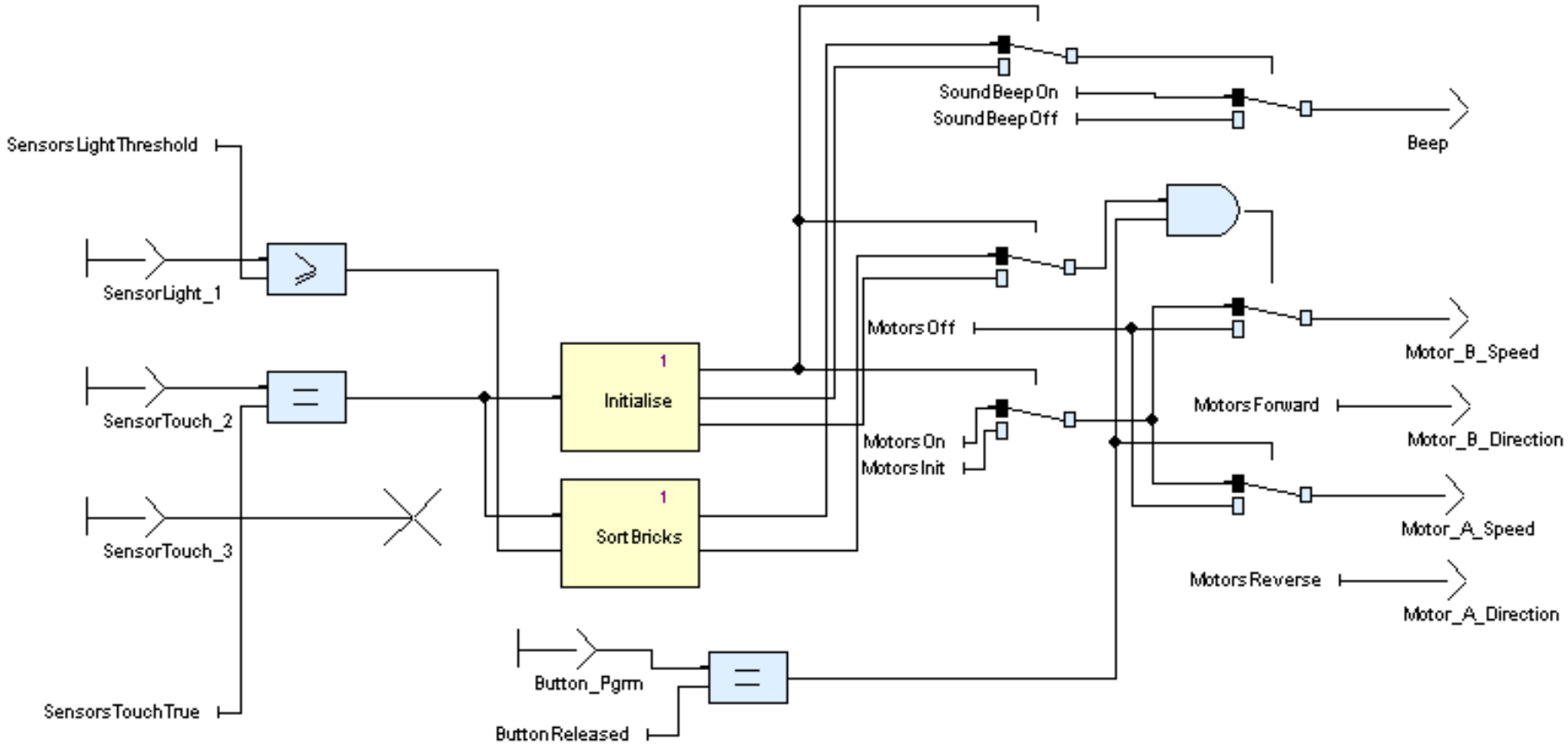
Node SortBricks



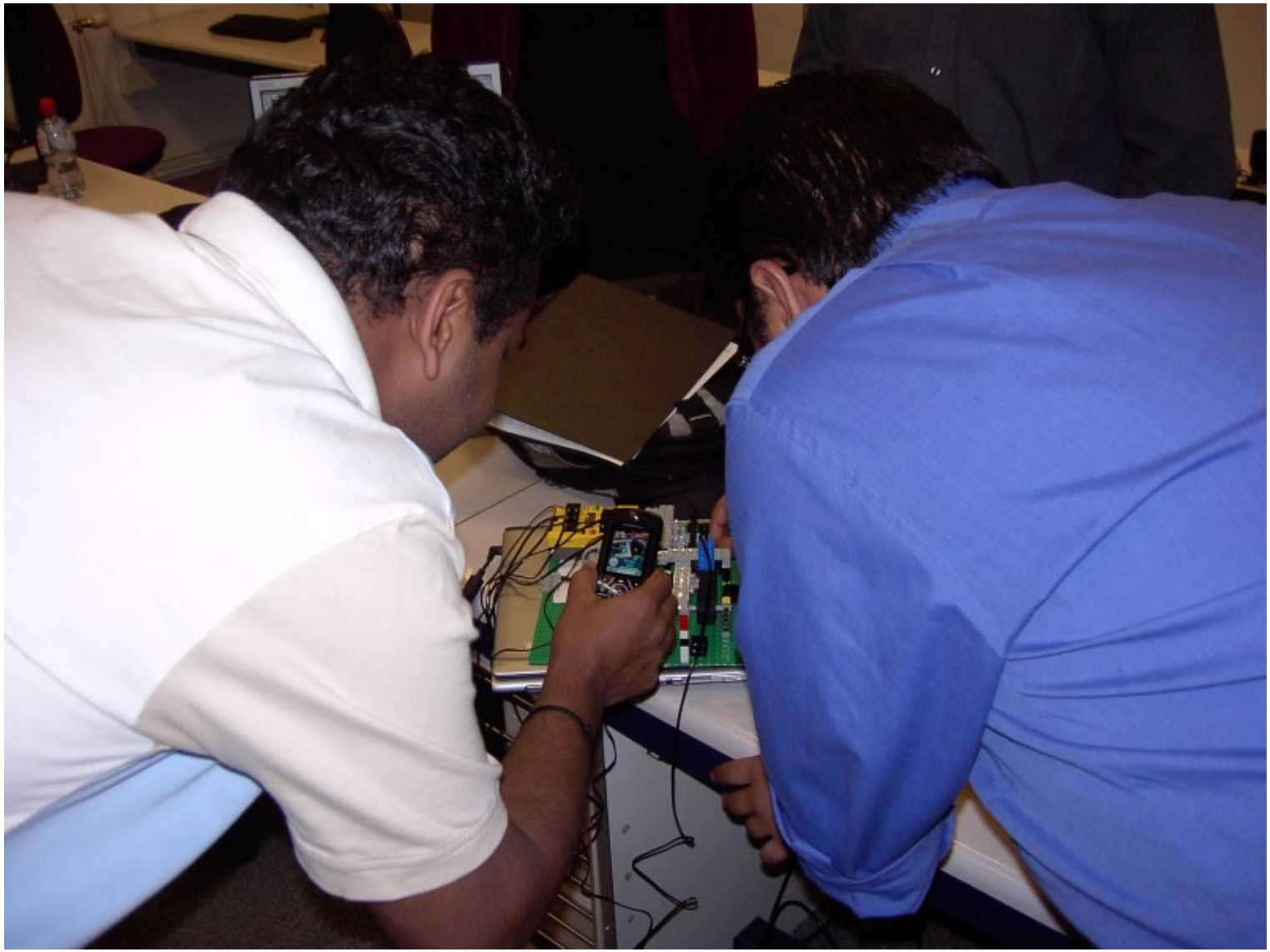
Node PushOut



Node Bricksorter

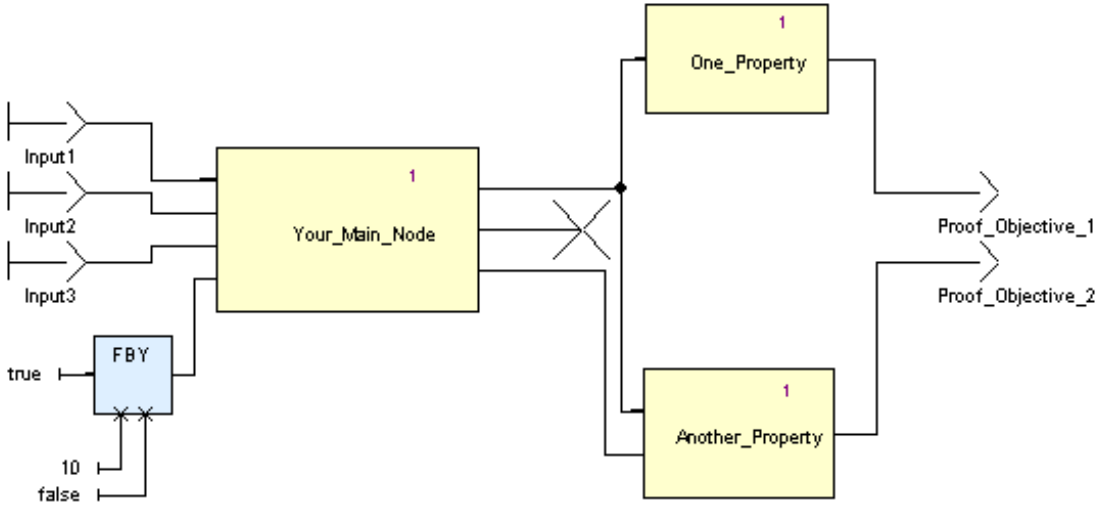


We had a lot of fun.



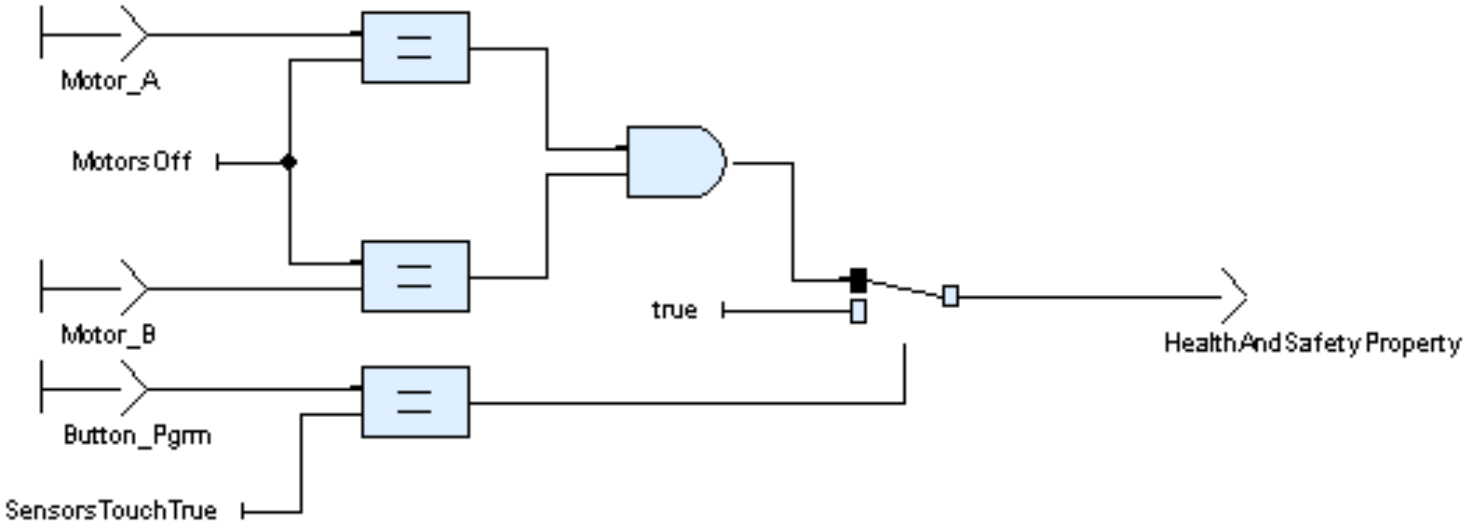
Design Verification

- Basic idea: Write an `Observer` in terms of a data flow equation
- The Design Verifier will prove that its `Proof Objectives` hold true for all possible executions



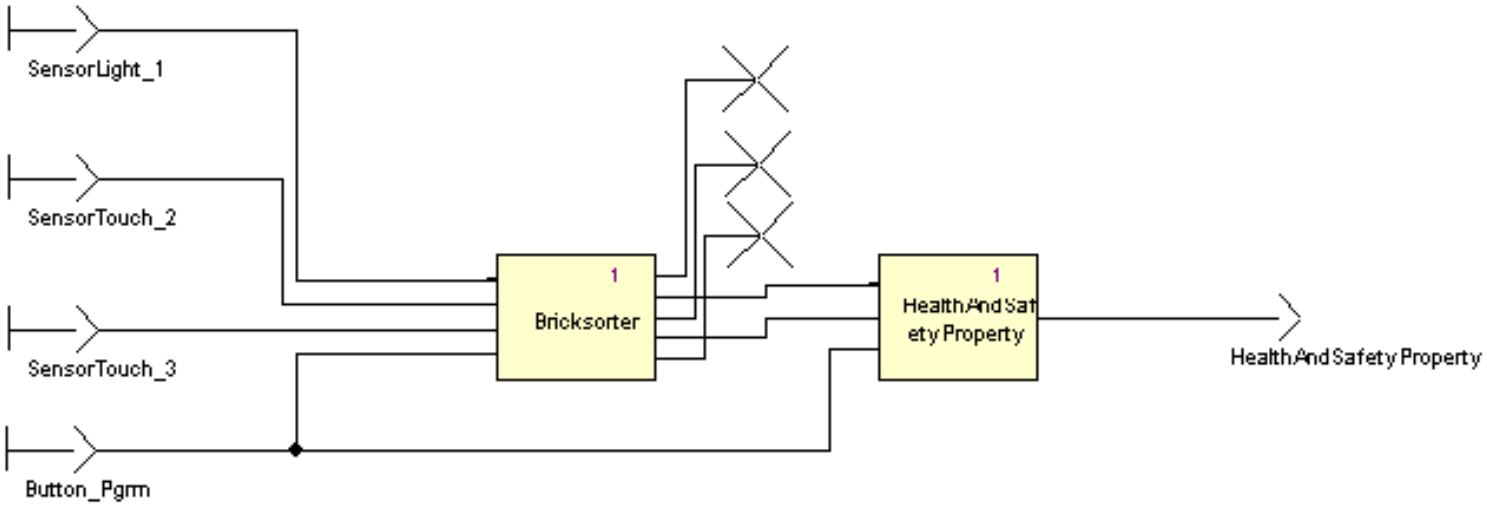
Design Verification (1)

- `HealthAndSafetyProperty`: All motors are turned off as long as the `Program` button is pressed



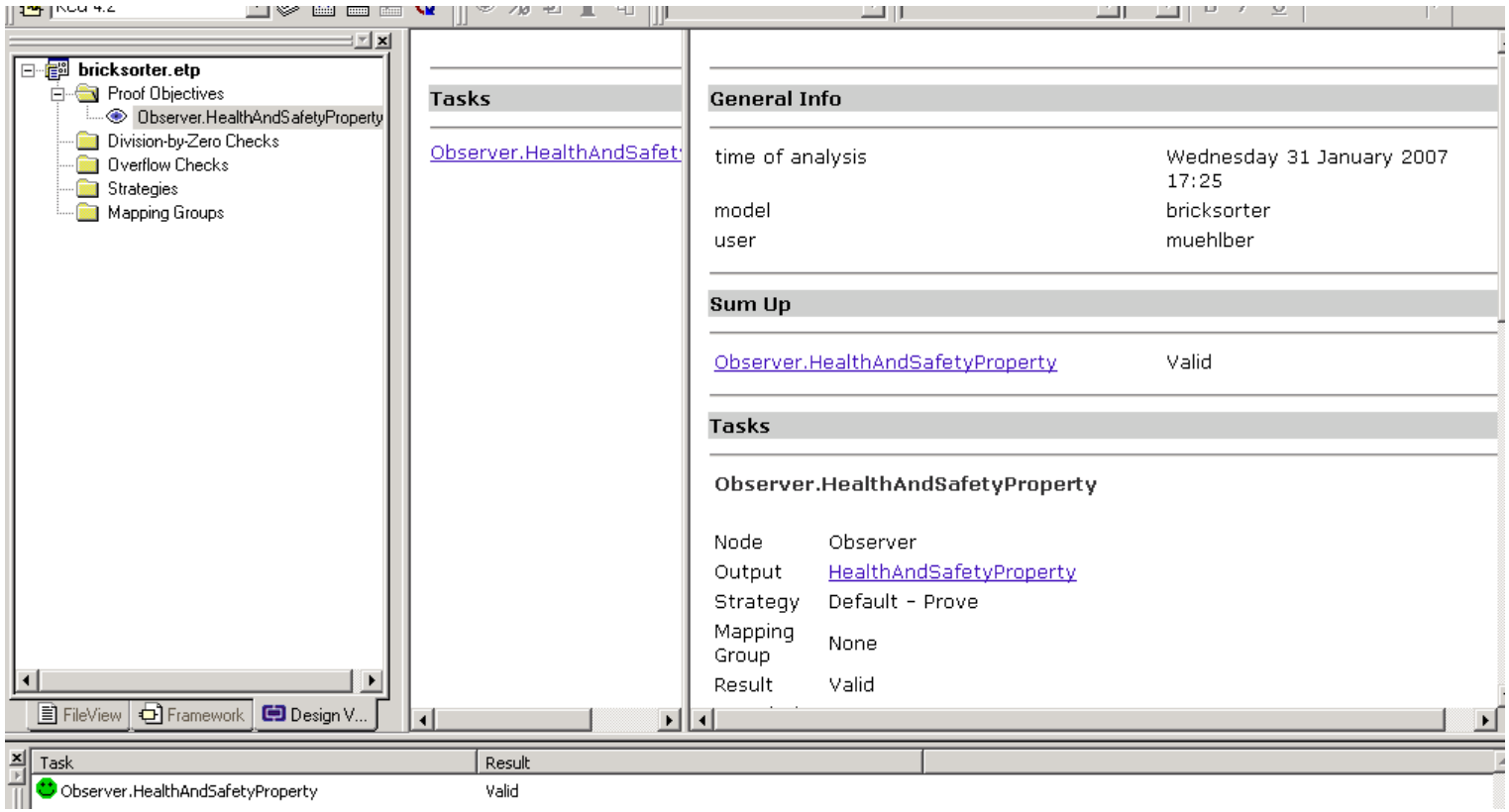
Design Verification (2)

- Embed the "Property Node" into an `Observer` Node



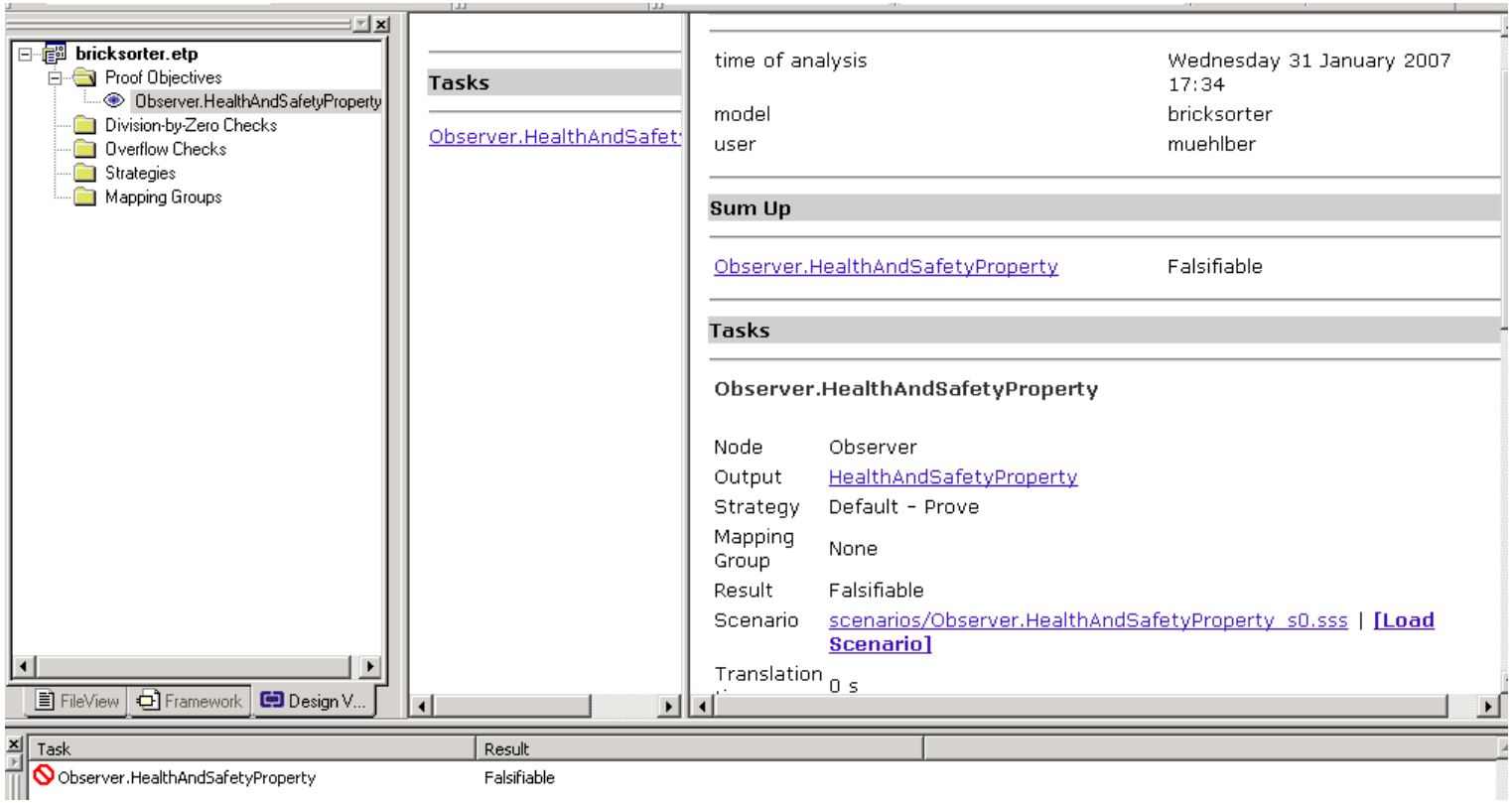
Design Verification (3)

- In this case everything is fine:



Design Verification (4)

- If the proof fails, the Design Verifier generates a scenario which can be loaded into the SCADE Simulator



Issues with SCADE

- Graphical languages are *urgs*
- Slow and difficult program development
- Generates slow programs
- Undocumented specialities (cycle 0)
- Tends to crash
- Matthew likes to do it in Haskell. . .

Lava

- Lava is a Haskell library for circuit design
- It provides two *abstract data types*: `Bit` and `Word`
- Here are some example operators of the ADTs:

```
(==>)  :: Bit -> Bit -> Bit
(+)    :: Word -> Word -> Word
```

(Note how `Word` is an instance of Haskell's `Num` class)

Lava 2

- Any Haskell function over tuples/lists of `Bit`s can be:
  - simulated (of course!)
  - verified for all inputs of a given size (if it returns a `Bit`, i.e. is a proposition)
  - turned into a circuit! (e.g. for FPGA)

Example 1 - Simple Circuits

```
import Lava

-- An ordering relation on bits
a `leq` b           =  a ==> b

-- A nice operator for multiplexing
a ? (b, c)          =  ifThenElse a (b, c)

bitSort             :: (Bit, Bit) -> (Bit, Bit)
bitSort (a, b)      =  (a `leq` b) ? ((a, b), (b, a))

Lava> simulate bitSort (high, low)
(low,high)

propBitSort (a, b) =  c `leq` d
  where
    (c, d)          =  sort (a, b)

Lava> smv propBitSort
Valid.
```

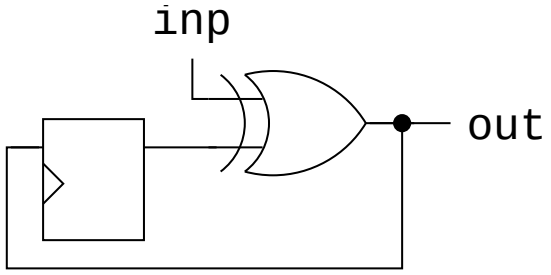
Example 2 - Listy Circuits

```
-- A linear reduction array
linear           :: ((a, a) -> a) -> [a] -> a
linear f [a]     =  a
linear f (a:as)  =  f (a, linear f as)

Lava> simulate (linear and2) [high, low, high]
low

-- Tree-shaped reduction: much better!
tree             :: ((a, a) -> a) -> [a] -> a
tree f [a]       =  a
tree f (a:b:bs)  =  tree f (bs ++ [f (a, b)])

propAndTree      =  forAll (list 8) $ λas ->
                      linear and2 as <==> tree and2 as

Lava> smv propAndTree
Valid

-- NOTE: We can't parameterise the property over "and2".
--       But, we could with SmallCheck!
--       SmallCheck properties are more expressive.
```

How does Lava work?

- It expands out recursion to give a graph-shaped processing network, AKA a circuit.
  - Nodes represent operators of the ADTs
  - Edges represent data flow
- Can we express loops in the graph?
  - Yes, using “circular programming”

```
parity          :: Bit -> Bit
parity inp      =  out
  where
    out'        =  delay low out
    out         =  xor2 (inp, out')
```



Turning Circuits into C

1. Break loops in graph by extracting flipflops. Call the resulting acyclic graph G.

2. Generate a C program as follows:

    a. Initialise flipflops

    b. Create an infinite while loop which

        i. Reads inputs from sensors

        ii. Executes a sequence of assignment statements that satisfies the data dependencies of G

        iii. Writes outputs to actuators

        iv. Performs flipflop updates

C Code for Parity Example

```
int main(void) {
int w1; ... ; int w15;
w1 = 0;  w2 = 0;  w3 = 0;
w4 = 0;  w5 = 0;  w6 = 0; w11 = 1; w13 = 0; w14 = 0;
w15 = 1; w12 = w13;
ds_active(&SENSOR_1);
ds_active(&SENSOR_2); ds_active(&SENSOR_3);
while (1) {
w1 = 0; w2 = 0; w3 = 0; w4 = 0;
w5 = 0; w6 = 0; w10 = TOUCH_1; w11 = 1; w9 = (w10==w11)&&1;
w13 = 0; w8 = w9!=w12; w14 = 0; w15 = 1; w7 = w8?w14:w15;

motor_a_dir(w1); motor_b_dir(w2); motor_c_dir(w3);
motor_a_speed(w4); motor_b_speed(w5); motor_c_speed(w6);

w12 = w8;
}}
```

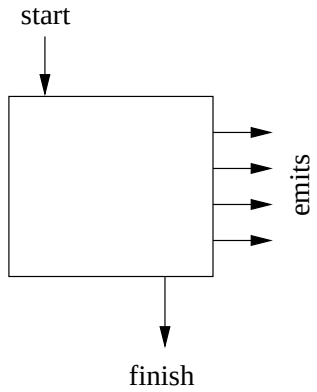
Now for the Brick Sorter!

- Problem:
  - Lava is great for data parallelism...
  - But horrible for control systems!

- Solution:
  - Haskell is great for writing interpreters...
  - Let's define our own little language for control systems in Lava
  - Actually, Koen Claessen and Gordon Pace have already done it for us :-)

Flash Gordon to the Rescue!

```
data Flash out
  = Skip
  | Emit out
  | Wait
  | IfThenElse Bit (Flash out, Flash out)
  | While Bit (Flash out)
  | Flash out :>> Flash out
  | Flash out :|| Flash out

compile :: Eq out => Flash out -> [out] -> [Bit]
compile =  ... about 50 lines of code :-) ...
```



A tidier interface

```
skip           =  Skip
emit a         =  Emit a
wait           =  Wait
ifte b (p, q)  =  IfThenElse b (p, q)
while b p      =  While b p
p ||| q        =  p :|| q
p >>> q        =  p :>> q

forever p      =  while high p
shout a        =  emit a >>> wait
waitUntil a    =  while (inv a) wait
```

Finally, our Brick Sorter

```
data Emitters             =  Push | Belt
                               deriving Eq

sorter (touch, light)     =  initialise >>> (moveBelt ||| pushWhenLight)
  where
    initialise            =  while (inv touch) (shout Push)
    moveBelt              =  forever (shout Belt)
    pushWhenLight         =  forever (waitUntil light
                                  >>> while touch (shout Push)
                                  >>> while (inv touch) (shout Push)
                                  >>> wait)
```

Rest of code (We have nothing to hide!)

```
main                        =  writeLego "sorter" f
  where
    f inp                   =  Output { dirMotorA   = motorReverse
                                      , dirMotorB   = motorForward
                                      , dirMotorC   = motorNeutral
                                      , speedMotorA = belt
                                      , speedMotorB = push
                                      , speedMotorC = motorStop
                                      , beep        = noBeep }
      where
        (push, belt)        =  interface (touch2 inp, light1 inp)

interface (touch, light)    =  (when push 100, when belt 80)
  where
    reflection              =  light />=/ lightThreshold
    touching                =  touch /=/ sensorTouched
    [push, belt]            =  compile (sorter (touching, reflection))
                                       [Push, Belt]
    when sig speed          =  sig ? (speed, motorStop)
```

Conclusions and Future work

- Mixing Lava with custom languages is nice. Many useful languages can be nicely expressed in Lava, e.g. a while ago I embedded a version of Occam in Lava – like Flash, it was only about 50 lines of code, and very useful!
- Could BlueSpec ideas be nicely embedded in Lava?

Conclusions and Future work 2

- Lava could do with subroutine support. Replication in software isn't very helpful!
- Sized vectors in Haskell's type system would be useful in Lava. With the current enthusiasm in GADTs and type-level programming, maybe this will happen?
- Mary Sheeran and Koen Claessen are interested in generating C from Lava for an ATI 64 processor GPU

Thank you!

References

Lüttgen, G. and Mühlberg, J. T.: 2006, *Reactive Systems Design Course Page*, http://www-course.cs.york.ac.uk/rsd/.

Mühlberg, J. T.: 2006, *scade2brick – Prepare C code generated by SCADE for brickOS.*, http://zeus.fh-brandenburg.de/~muehlber/content/software/scade2brick/.

Random other things

- Publications list: Still no response from users with UIDs 1041 and 1788. . . Please!
- The term is over in two weeks. We need talks for next term!
- Next talk. . .