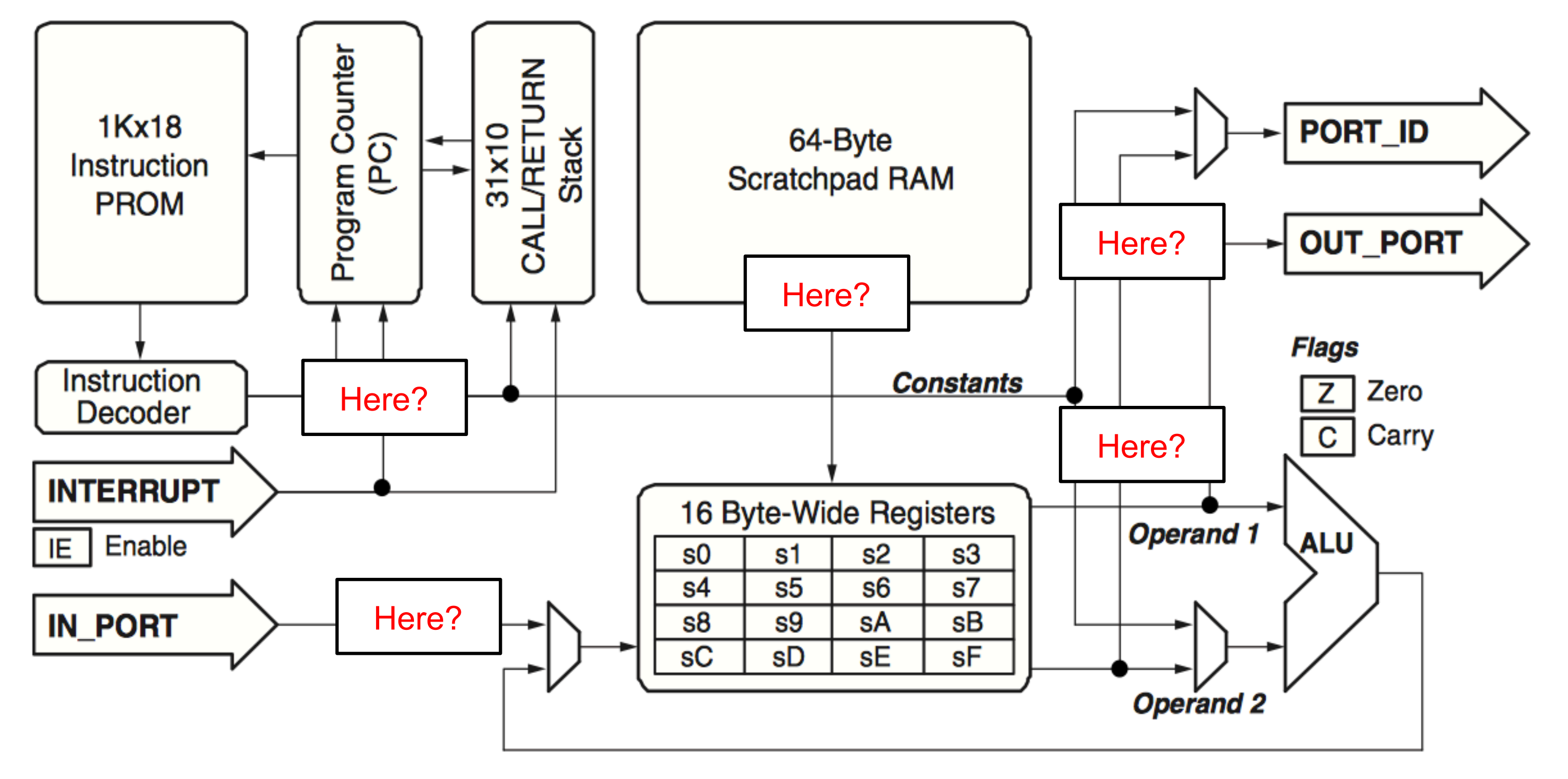
Semantics-directed Machine Architecture in ReWire

Adam Procter[1], William L. Harrison[1], Ian Graves[1], Michela Becchi[1], and Gerard Allwein[2]
(1) University of Missouri (2) U.S. Naval Research Laboratory

Semantic Modularity

Suppose we want to add hardware support for *separation* to a stock embedded processor design, allowing safe interleaving of processes handling classified and unclassified data.

Where does the new "separation module" go?



(Source: PicoBlaze 8-bit Embedded Microcontroller User Guide, Xilinx Inc.)

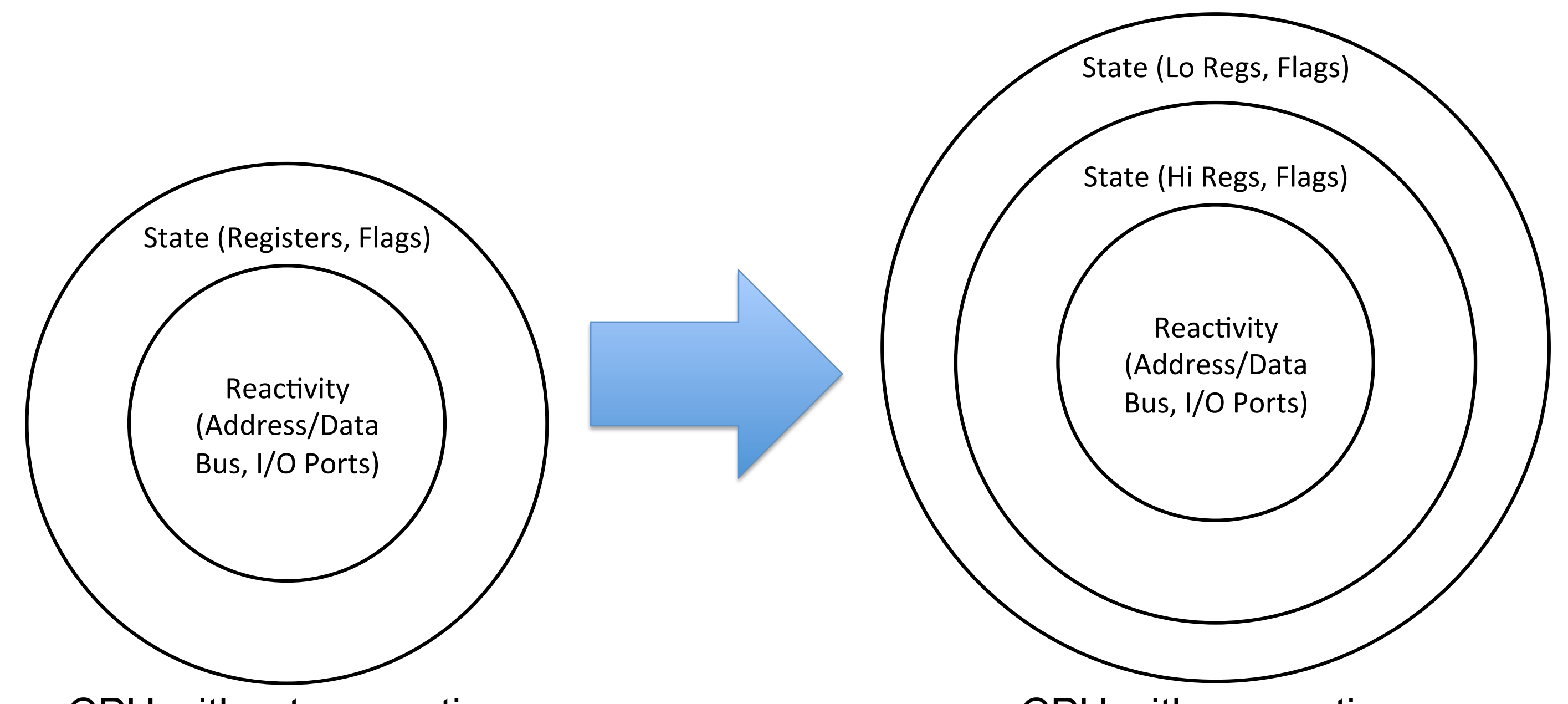
Problem: Original design is modular in a structural sense, but not in a **semantic** sense.

How to achieve semantic modularity in hardware design?

Idea from programming language theory: **modular monadic semantics (MMS)**.

With MMS, you construct custom domain-specific languages supporting *just* the kind of semantic effects you want, from building blocks called **monad transformers**.

In the figure, each "layer" of the "onion" corresponds to a monad transformer. Extending a semantically modular processor design with hardware-level separation is a simple matter of adding one more monad transformer.


State (Registers, Flags)
Reactivity (Address/Data Bus, I/O Ports)
State (Lo Regs, Flags)
State (Hi Regs, Flags)
Reactivity (Address/Data Bus, I/O Ports)

CPU without separation CPU with separation

Example at Scale: PicoBlaze from Xilinx

We have developed an MMS-style specification for the PicoBlaze soft microcontroller from Xilinx. The MMS semantics corresponds nicely to the informal documentation.

Instruction	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD sX,kk	0	1	1	0	0	0	x	x	x	x	k	k	k	k	k	k	k	k
ADD sX,sY	0	1	1	0	0	1	x	x	x	x	y	y	y	y	0	0	0	0
ADDCY sX,kk	0	1	1	0	1	0	x	x	x	x	k	k	k	k	k	k	k	k
ADDCY sX,sY	0	1	1	0	1	1	x	x	x	x	y	y	y	y	0	0	0	0

...

```
decode :: Instruction -> CPU ()
decode (W18  0  1  1  0  0  0 x0 x1 x2 x3 k0 k1 k2 k3 k4 k5 k6 k7) =
  addImm (W4 x0 x1 x2 x3) (W8 k0 k1 k2 k3 k4 k5 k6 k7)

decode (W18  0  1  1  0  0  1 x0 x1 x2 x3 y0 y1 y2 y3  0  0  0  0) =
  addReg (W4 x0 x1 x2 x3) (W4 y0 y1 y2 y3)

…
```

Left: PicoBlaze instruction code reference; Right: PicoBlaze instruction decoder in modular monadic semantics.

```
sX ← (sX + Operand) mod 256; always an 8-bit result

if ( (sX + Operand) > 255 ) then
   CARRY ← 1
else
   CARRY ← 0
endif

if ( ((sX + Operand) = 0) or ((sX + Operand) = 256) ) then
   ZERO ← 1
else
   ZERO ← 0
endif

PC ← PC + 1
```

```
binopImm :: Binop -> Register -> Byte -> CPU ()
binopImm oper sX kk = do v           <- getReg sX
                         c           <- getFlag FlagC
                         let (c',v') =  (v `oper` kk) c
                         putFlag FlagZ (toBit $ v' == 0)
                         putFlag FlagC c'
                         putReg sX v'
                         incrPC
                         tick
                         tick
```

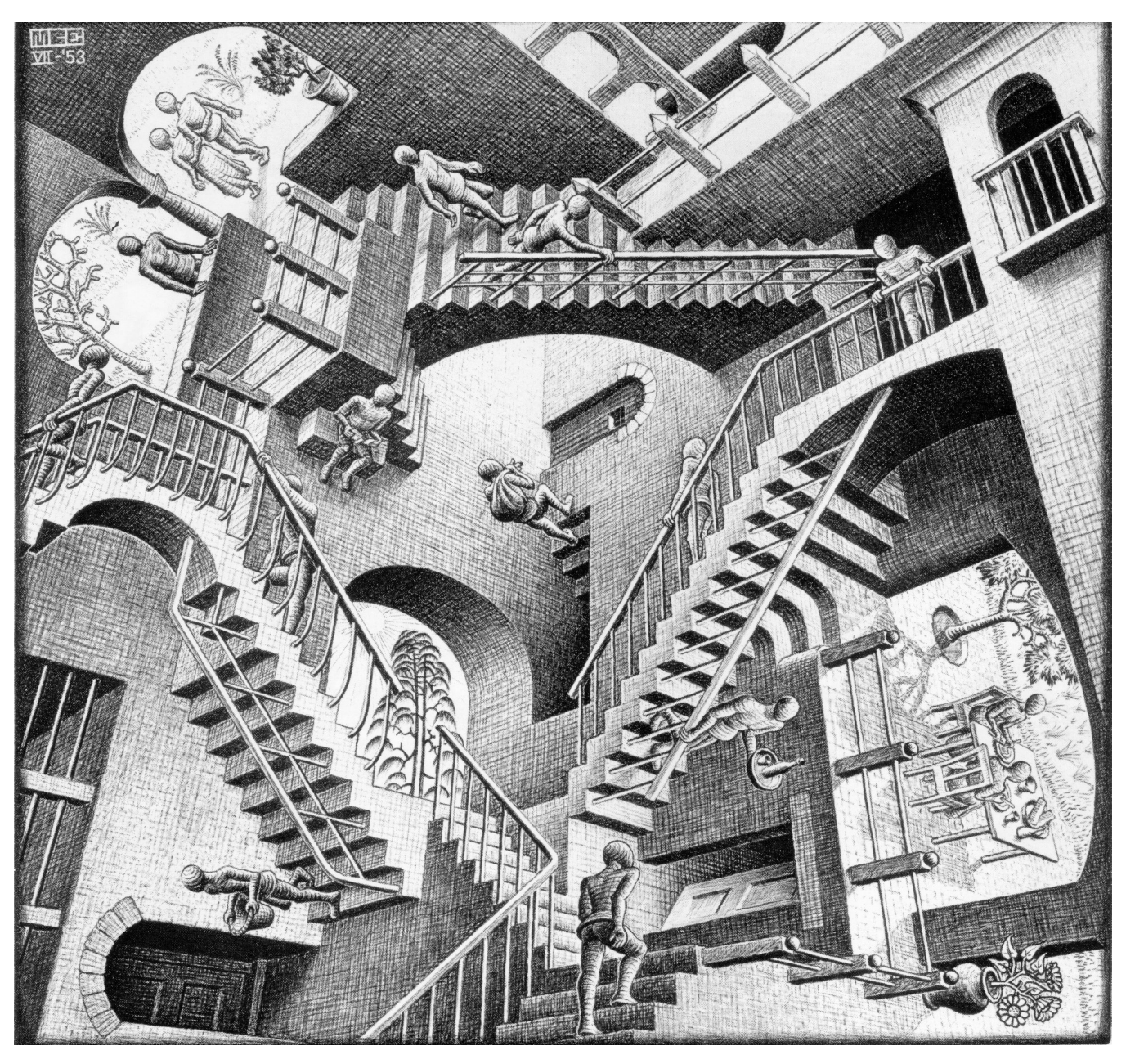
Left: Pseudocode for PicoBlaze ADD instruction from the reference manual; Right: ADD instruction in modular monadic semantics.

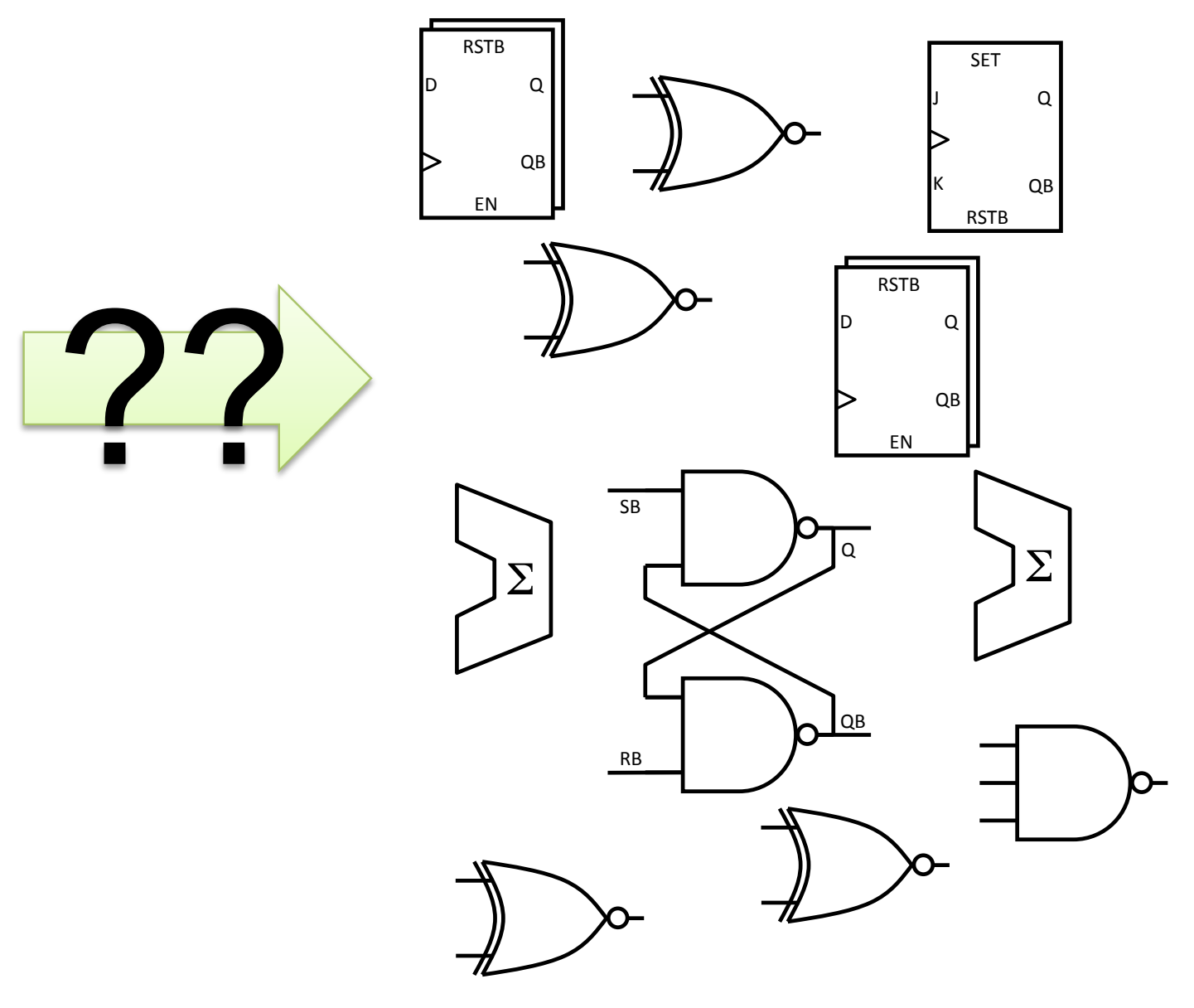
Challenge: Compilation

We need an expressive functional language to support MMS. We choose **Haskell**.

When it comes to synthesis, however, Haskell has many **features that are hard to translate** directly to gates.

- General recursion, recursive data types, higher-order functions…



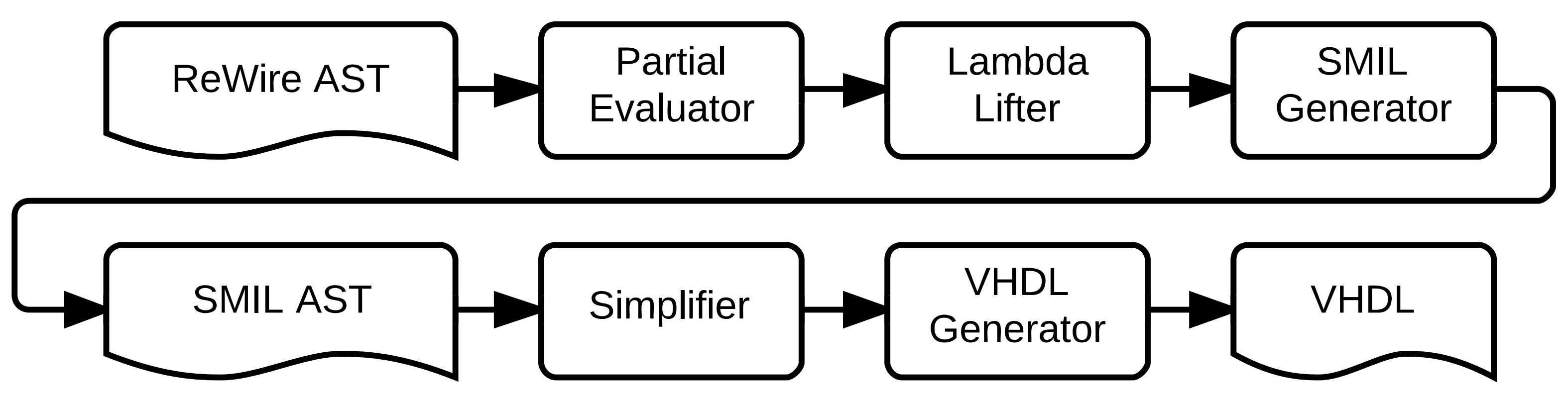


Solution: Partial Evaluation

We have implemented a prototype compiler called **ReWire** that translates MMS specifications written in Haskell into VHDL suitable for use on FPGAs.

The core technique here is **partial evaluation**, a program transformation technique that works by performing as much evaluation at compile time as possible.

Partial evaluation is effective at **eliminating language constructs** that cannot be directly translated to hardware, producing a normal-form program that can easily be translated into a finite state machine.



Case Study

Paper discusses the synthesis of a very simple processor design in MMS style, with a tiny instruction set (four instructions), two general-purpose registers, an external program ROM, and a single output line.

As compiled by ReWire, this processor design utilizes115 logic slices on a Spartan-3E series FPGA. Detailed usage statistics for a Spartan-3E XC3S500E FPGA, speed grade -4, are as follows. Maximum clock rate on this particular chip is around 133MHz.

	Used	Available	Utilization %
Slices	115	4656	2.47%
Slice Flip Flops	48	9312	0.52%
4-Input LUTs	213	9312	2.29%

Ongoing Work

Apart from the benefits in extensibility, monadic semantics offers a powerful basis for **formal verification**, namely **equational reasoning**.

Ongoing work involves adapting existing techniques by several of the authors, previously used to verify monadic security kernels implemented in software, to prove **separation properties** of hardware circuits.

Further reading:

W. L. Harrison, A. Procter and G. Allwein. The confinement problem in the presence of faults. ICFEM 2012.
W. L. Harrison and J. Hook. Achieving information flow security through monadic control of effects. J. Comput. Secur., October 2009.

This research was supported by the U.S. Department of Education under GAANN grant number P200A100053.