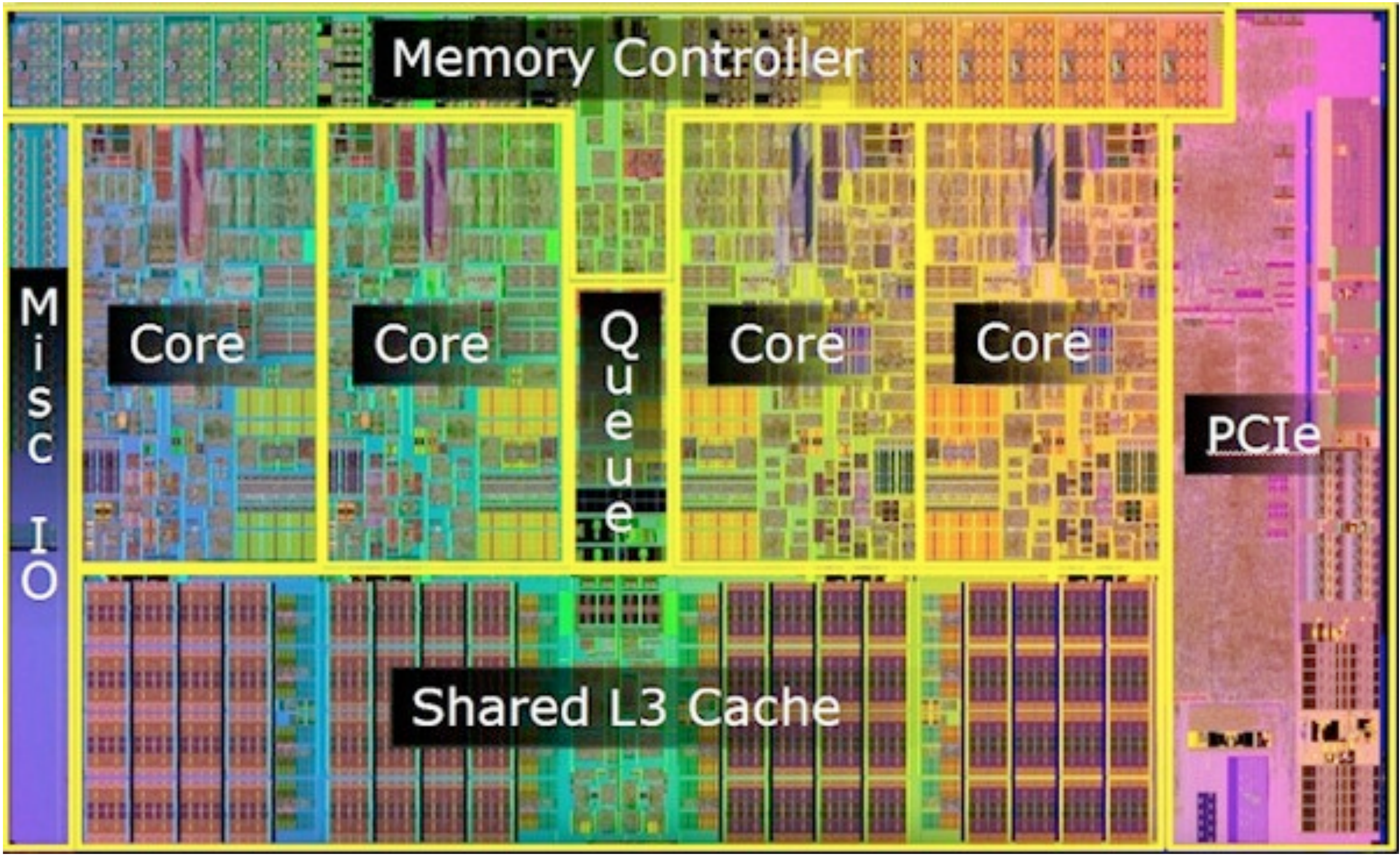
High-level development and debugging of FPGA-based network programs

nik.sultana@cl.cam.ac.uk

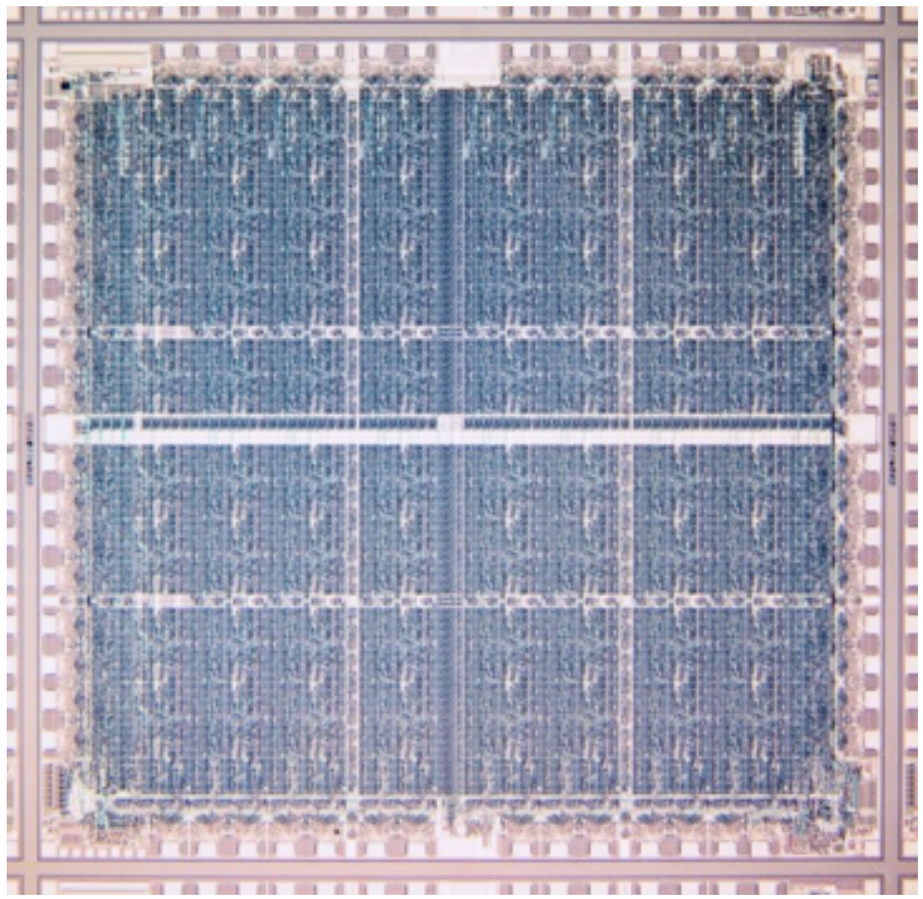
Pietro Bressana, Richard Clegg, Paolo Costa, Jon Crowcroft,
Salvator Galea, David Greaves, Luo Mai, Andrew W Moore,
Richard Mortier, Peter Pietzuch, Jonny Shipton, Robert Soule,
Nik Sultana, Marcin Wojcik, Noa Zilberman



Intel Lynnfield

(c) Intel

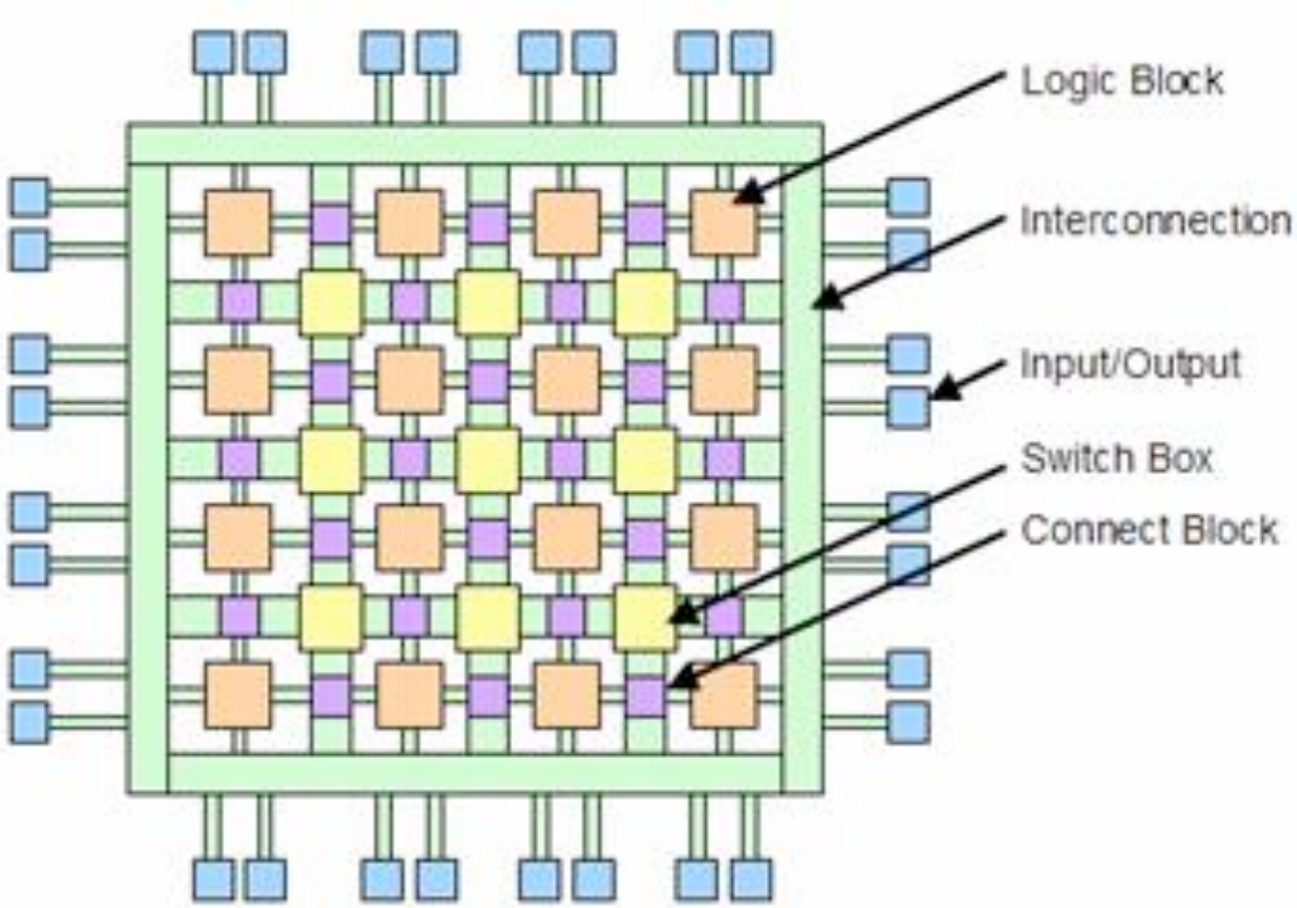
Field-programmable gate arrays



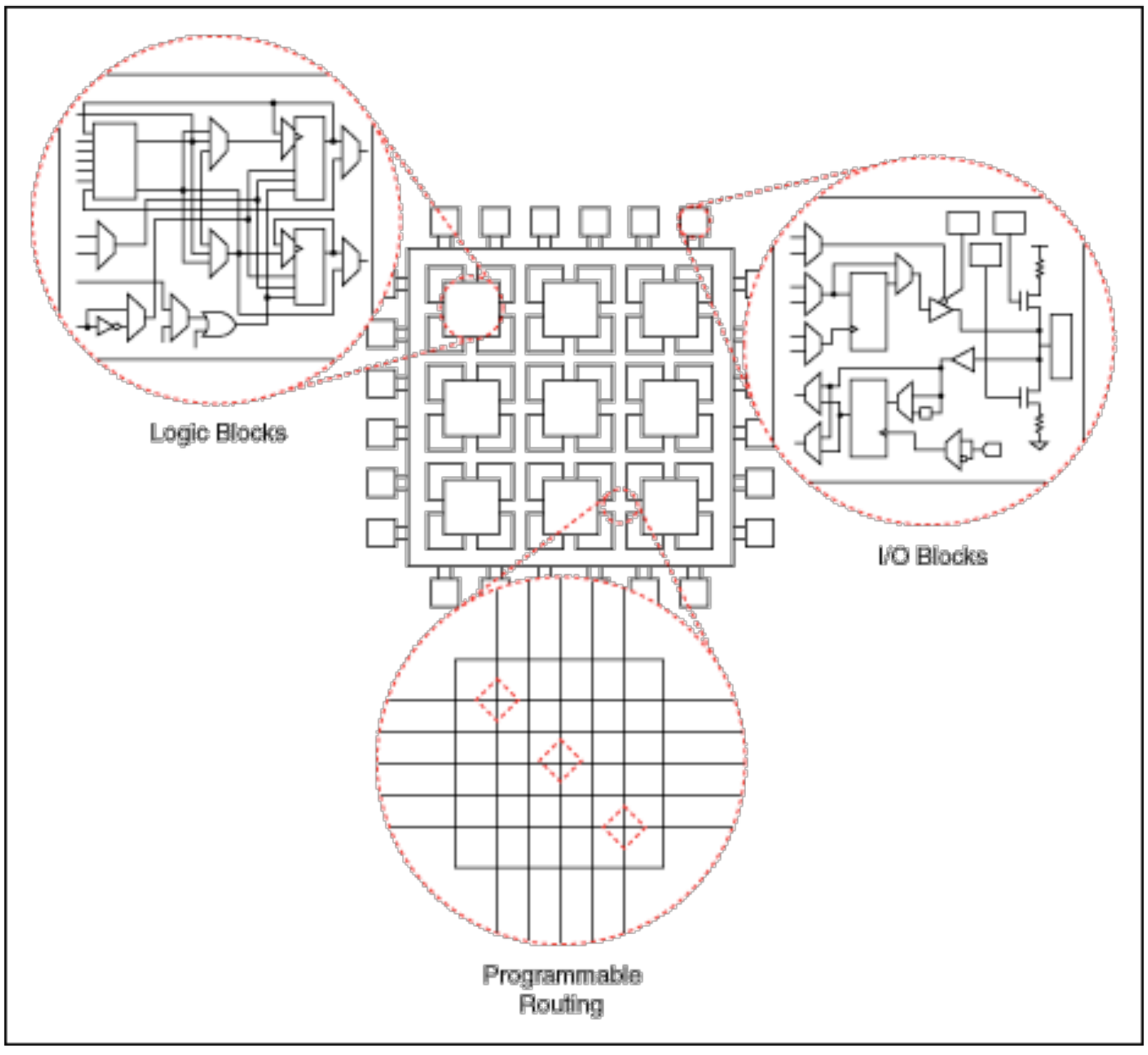
Xilinx XC2064 FPGA

64 CLB x (2 3-LUTs + FF)

(c) Xilinx



http://ca.olin.edu/2005/fpga_dsp/fpga.html



http://zone.ni.com/reference/en-XX/help/371599J-01/lvfpgaconcepts/fpga_basic_chip_terms/

- FPGAs becoming more **powerful** and **prevalent.**

  Most recently in datacentres.

  - e.g., Azure, Baidu, Amazon.

- FPGAs becoming more **powerful** and **prevalent.**

  Most recently in datacentres.

- But still **difficult** to program and debug!

  To both **hardware** and **non-hardware** people.

  Both **technical** and **non-technical** difficulties.

  - Hardware programming unlike software programming.
  - Generating a bitstream is a lengthy process.

  - Differing interpretations for standard HDLs.
  - Quality of the tools is less polished than for software.
  - Strong vendor bias, closed formats, hold things back.

- FPGAs becoming more **powerful** and **prevalent.**

  Most recently in datacentres.

- But still **difficult** to program and debug!

  To both **hardware** and **non-hardware** people.

  Both **technical** and **non-technical** difficulties.

- 30+ years of research into using **high-level languages** for circuit description. 20+ years commercial tooling (Synopsys Behavioural Compiler in 1994)

- FPGAs becoming more **powerful** and **prevalent.**

  Most recently in datacentres.

- But still **difficult** to program and debug!

  To both **hardware** and **non-hardware** people.

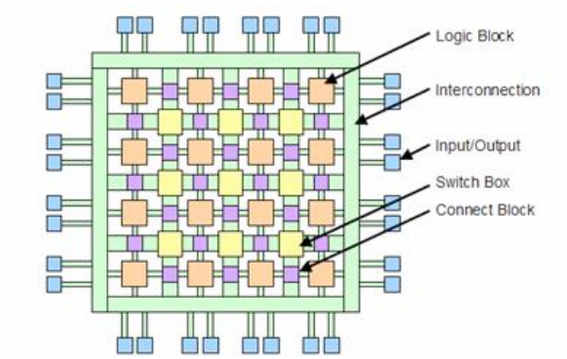
  Both **technical** and **non-technical** difficulties.

- 30+ years of research into using **high-level languages** for circuit description. 20+ years commercial tooling (Synopsys Behavioural Compiler in 1994)
- Experience has been mixed — you can't be everything to everybody! But lots of progress.

High-level development and debugging of **FPGA-based** network programs

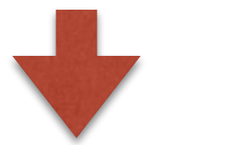
High-level
development and debugging
of
FPGA-based network programs

Gates (+ Interconnections)

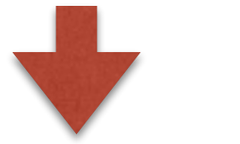


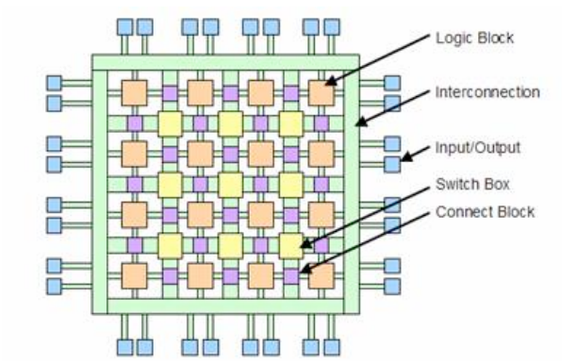
Hardware Description Language

(e.g., Verilog)



Gates (+ Interconnections)

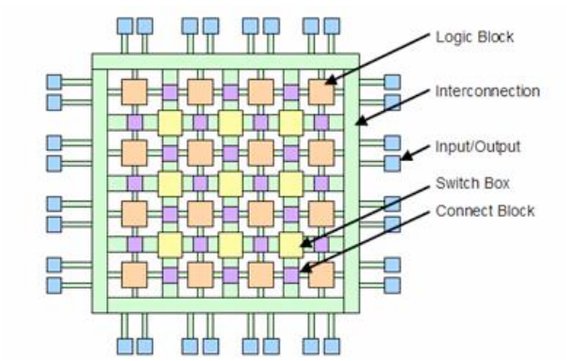




General Purpose Language
(High-Level Synthesis)
(e.g., C)

Hardware Description Language
(e.g., Verilog)

Gates (+ Interconnections)

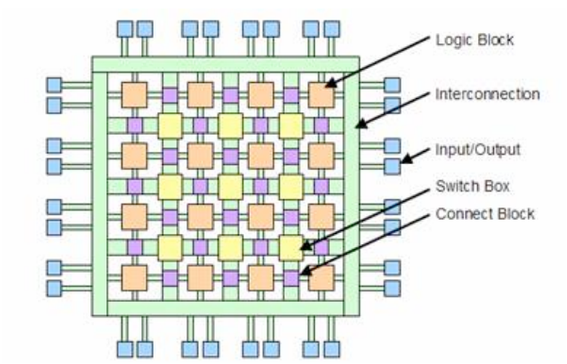


Domain Specific Language
(e.g., PP, PX, P4)

General Purpose Language
(High-Level Synthesis)
(e.g., C)

Hardware Description Language
(e.g., Verilog)

Gates (+ Interconnections)

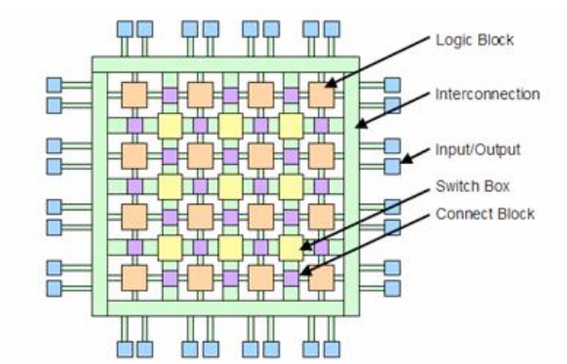


Domain Specific Language
(e.g., PP, PX, P4)

General Purpose Language
(High-Level Synthesis)
(e.g., C)

Hardware Description Language
(e.g., Verilog)

Gates (+ Interconnections)



Hardware Description Language

- Lots of **flexibility**.

- How you **think** code will behave
  vs how it's **translated**
  vs how it **executes/implements**.

- "reg" ~ register

- "inference"

- "technology mapping"

```
struct node {
  unsigned int prev_node : IDX_WIDTH;
  unsigned int next_node : IDX_WIDTH;
  unsigned int data      : DATA_WIDTH;
}

struct node memory [MAX_DEPTH_IDX+1];
```

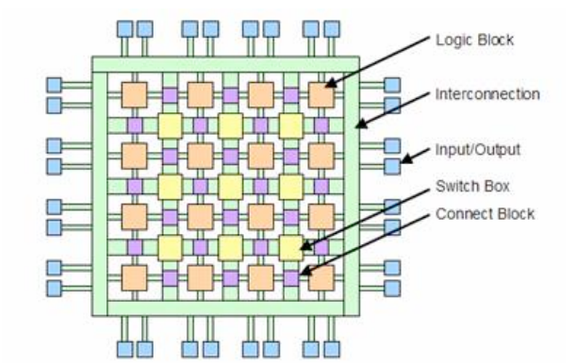
```
`define DATA_F_LSB 0
`define DATA_F_MSB (`DATA_F_LSB + DATA_WIDTH - 1)
`define DATA_F_WORD `DATA_F_MSB:`DATA_F_LSB

`define NEXT_NODE_F_LSB (`DATA_F_MSB + 1)
`define NEXT_NODE_F_MSB (`NEXT_NODE_F_LSB + IDX_WIDTH - 1)
`define NEXT_NODE_F_WORD `NEXT_NODE_F_MSB:`NEXT_NODE_F_LSB

`define PREV_NODE_F_LSB (`NEXT_NODE_F_MSB + 1)
`define PREV_NODE_F_MSB (`PREV_NODE_F_LSB + IDX_WIDTH - 1)
`define PREV_NODE_F_WORD `PREV_NODE_F_MSB:`PREV_NODE_F_LSB

reg [`PREV_NODE_F_MSB:`DATA_F_LSB] memory [MAX_DEPTH_IDX:0];
```

Domain Specific Language
(e.g., PP, PX, P4)

General Purpose Language
(High-Level Synthesis)
(e.g., C)

Hardware Description Language
(e.g., Verilog)

Gates (+ Interconnections)



Domain-Specific Language

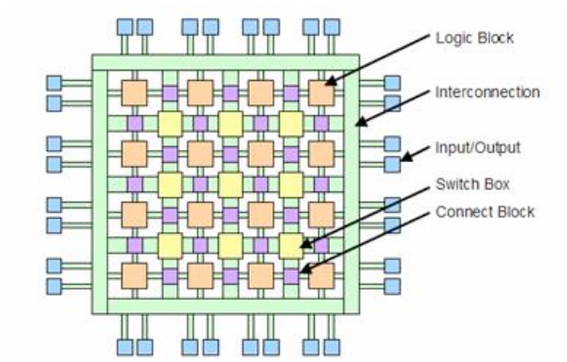
- Much **less flexibility**. Must stay within “domain”.
- Can achieve **good performance** and more **development support** (e.g., richer types), and **shorter cycles** of development.
- **Tuning** can be tricky — e.g., breakout to HDL.

Domain Specific Language
(e.g., PP, PX, P4)

General Purpose Language
(High-Level Synthesis)
(e.g., C)

Hardware Description Language
(e.g., Verilog)

Gates (+ Interconnections)

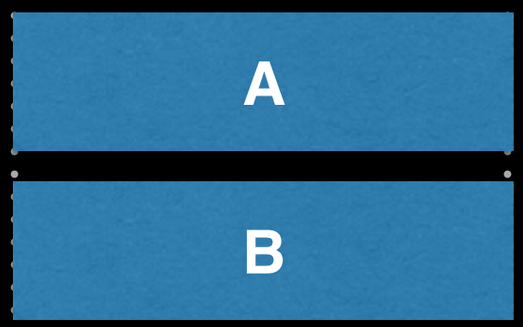


High-Level Synthesis

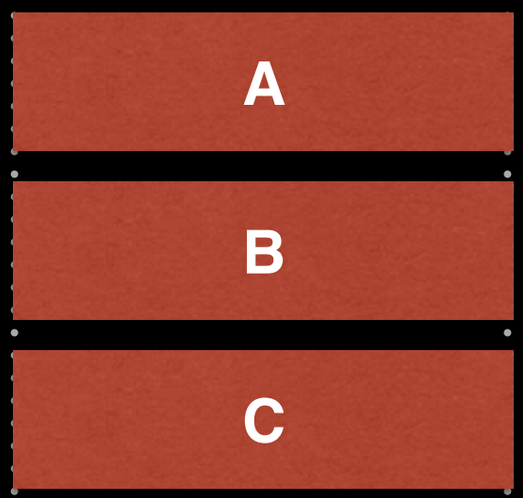
- Use **“familiar” language**.
- Usually **not the full language**. e.g., dynamic allocation only partly supported.
- Often involves library support and language extensions.
- **Tuning** can be tricky — e.g., breakout to HDL.

```
…
A: x = (some expression)
B: y = (some expression)
…
```

```
…
A: x = (some expression)
B: y = (some expression)
…
```

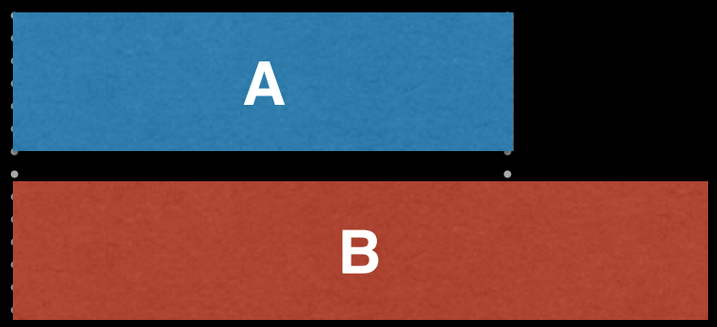


```
…
A: x = (some expression)
C: f(x)
B: y = (some expression)
…
```

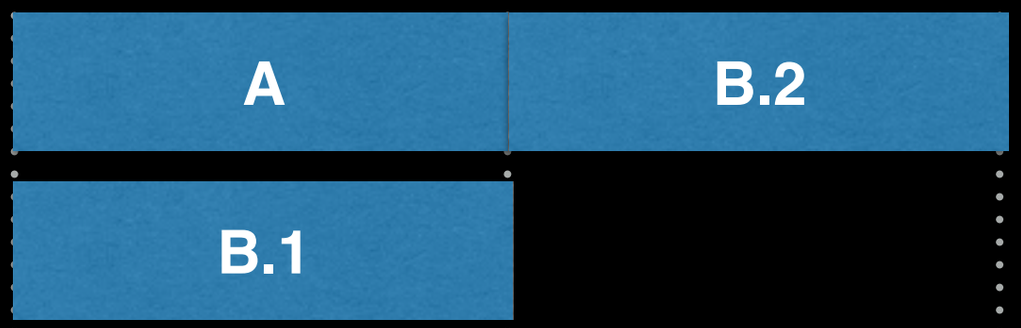


```
…
A: x = (some expression)
C: f(x)
B: y = (some expression)
…
```

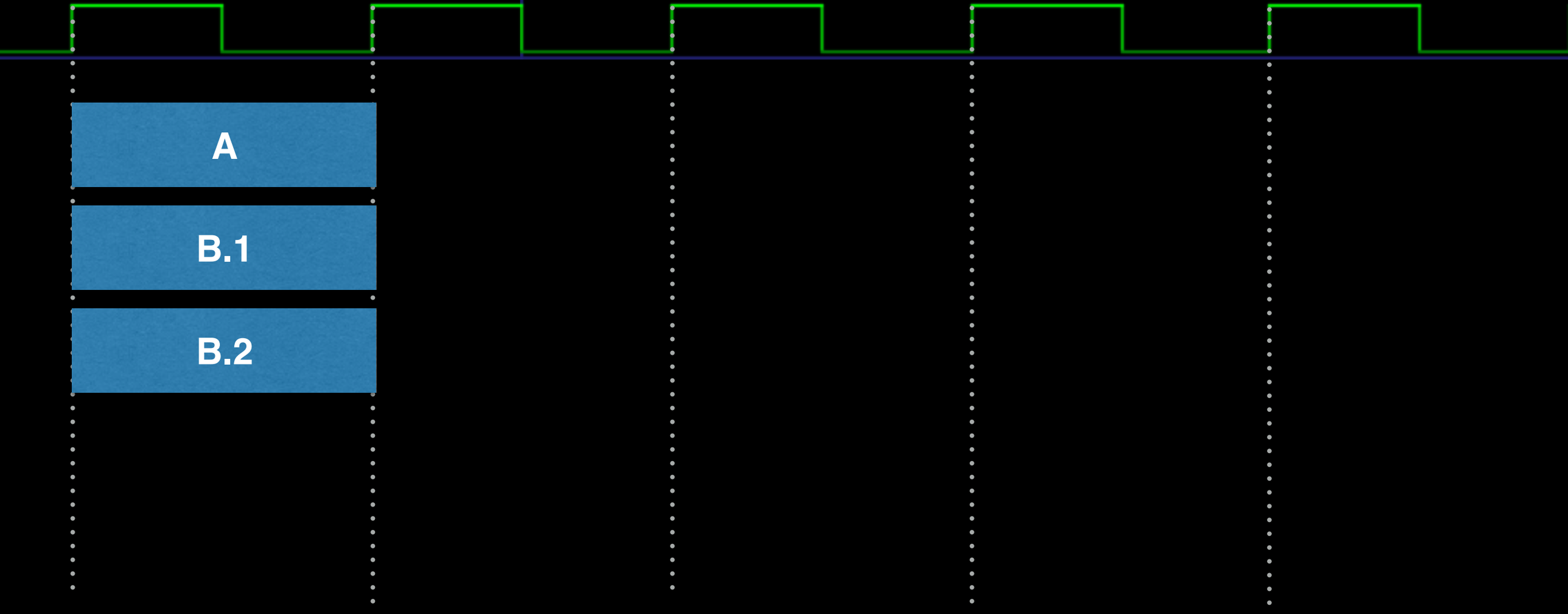
```
…
A: x = (some expression)
B: y = (v.cplx expression)
…
```



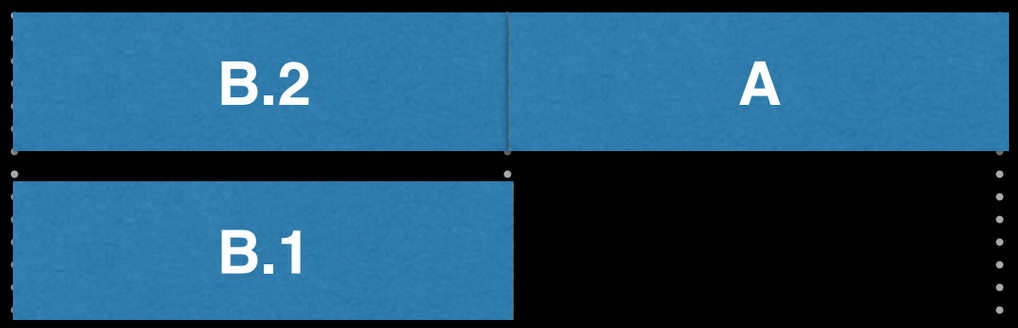
```
…
A: x = (some expression)
B: y = (some expression)
…
```



```
…
A: x = (some expression)
B: y = (some expression)
…
```



```
…
A: x = (some expression)
B: y = (some expression)
…
```



“The user can control how aggressively Stratus HLS packs these operations into each clock period. Creating designs with Stratus HLS can save months of backend effort by preventing timing closure problems.”

cadence®

https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/tools/digital-design-signoff/stratus-ds.pdf

Goal

High-level
development and debugging
of
FPGA-based network programs

…using HLS

It won’t work…

It won’t work…

- **Performance will suck!**
  Use HDL modules from HLL for resource-related IP.

It won’t work…

- **Performance will suck!**
  Use HDL modules from HLL for resource-related IP.

- **HLS tools are expensive and closed-source.**
  Various academic tools exist.

It won't work…

- **Performance will suck!**
  Use HDL modules from HLL for resource-related IP.

- **HLS tools are expensive and closed-source.**
  Various academic tools exist.

- **Not sufficiently expressive.**
  Can be fixed. (With ingenuity.)

It won't work…

- **Performance will suck!**
  Use HDL modules from HLL for resource-related IP.

- **HLS tools are expensive and closed-source.**
  Various academic tools exist.

- **Not sufficiently expressive.**
  Can be fixed. (With ingenuity.)

- **How to run this in software?**
  Create emulation environment + shadow library.

Concerns

- Providing benefits for **hardware people**: improved time-to-market, prototyping, development support, debugging, can breakout to HDL.

Concerns

- Providing benefits for **hardware people**: improved time-to-market, prototyping, development support, debugging, can breakout to HDL.
- Providing benefits for **non-hardware people**: through less steep learning curve, software-like development mindset.

Concerns

- Providing benefits for **hardware people**: improved time-to-market, prototyping, development support, debugging, can breakout to HDL.

- Providing benefits for **non-hardware people**: through less steep learning curve, software-like development mindset.

- Comparable or better **performance** (latency +throughput) or resource **utilisation** to hand-written HDL.

Our system: **Emu**

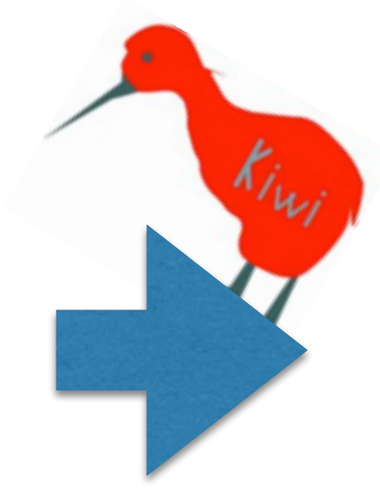
(High-level)
Software
description
of network
program

Hardware
description
of network
program

(1) HLS

(High-level) Software description of network program

C#



Hardware description of network program

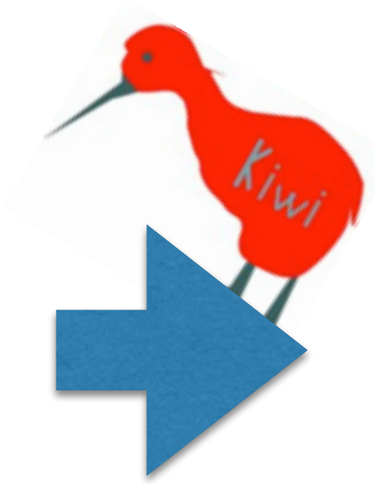
Verilog

http://www.cl.cam.ac.uk/~djg11/kiwi/

```
public class Data {
  public bool matched = false;
  public ulong result = 0;
}

public class LRU
{
    public static Data Lookup(ulong key_in)
    {
      Data res = new Data();
      ulong idx = HashCAM.Read(key_in);
      if (HashCAM.matched) {
        res.matched = HashCAM.matched;
        res.result = NaughtyQ.Read(idx);
        NaughtyQ.BackOfQ(idx);
      }
      return res;
    }

    public static void Cache(ulong key_in, ulong value_in)
    {
      ulong idx = NaughtyQ.Enlist(value_in);
      HashCAM.Write(key_in, idx);
    }
}
```

(2) Library support

(High-level) Software description of network program



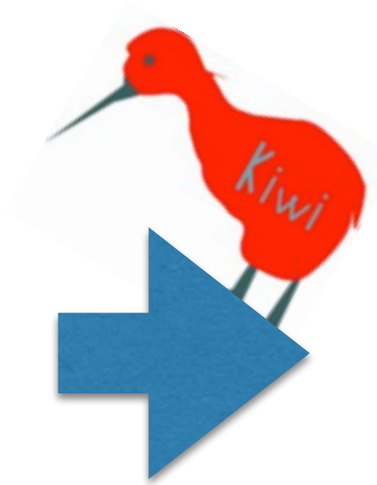
Hardware description of network program

C#
+ libraries

Verilog
+ libraries

(3) Host environment

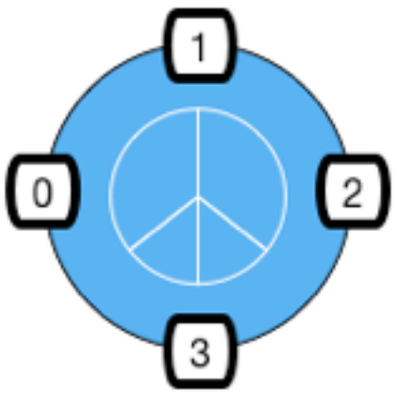
(High-level)
Software
description
of network
program



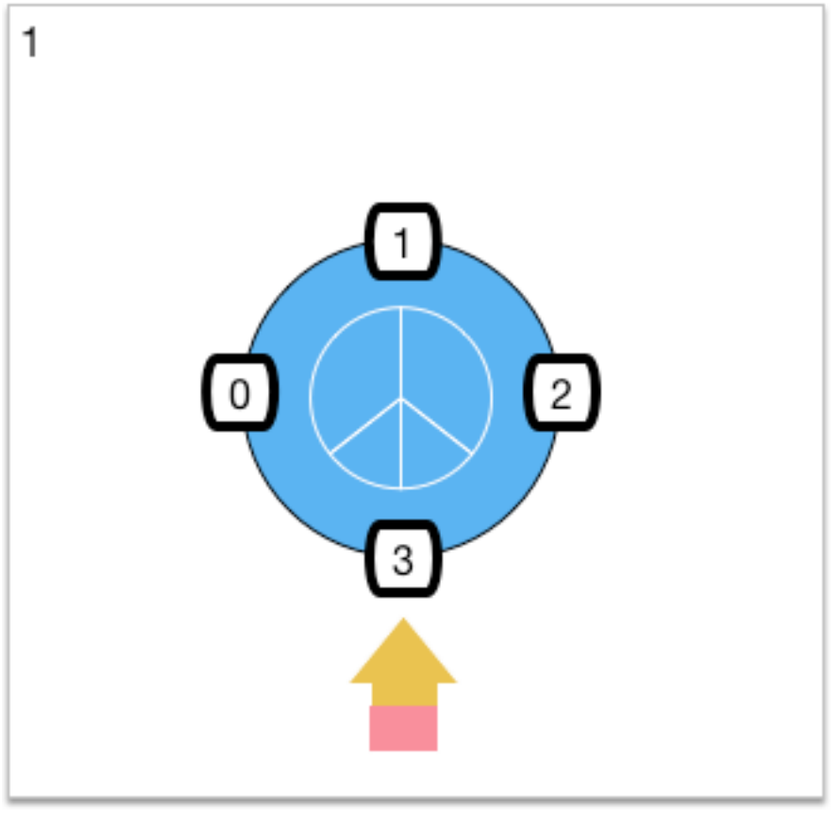
Hardware
description
of network
program

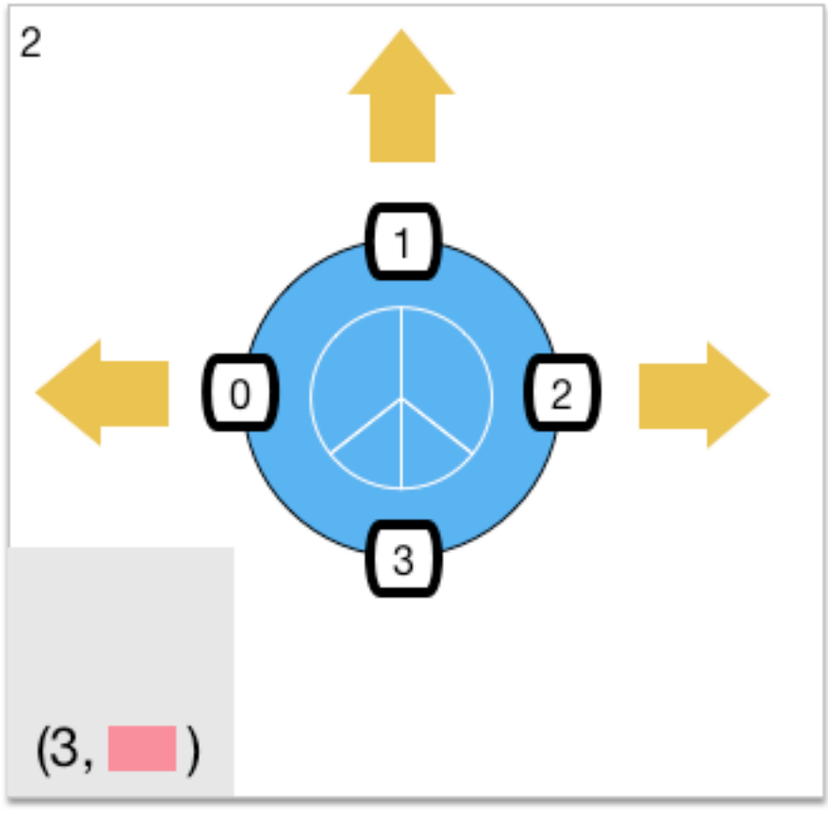
C#
+ libraries

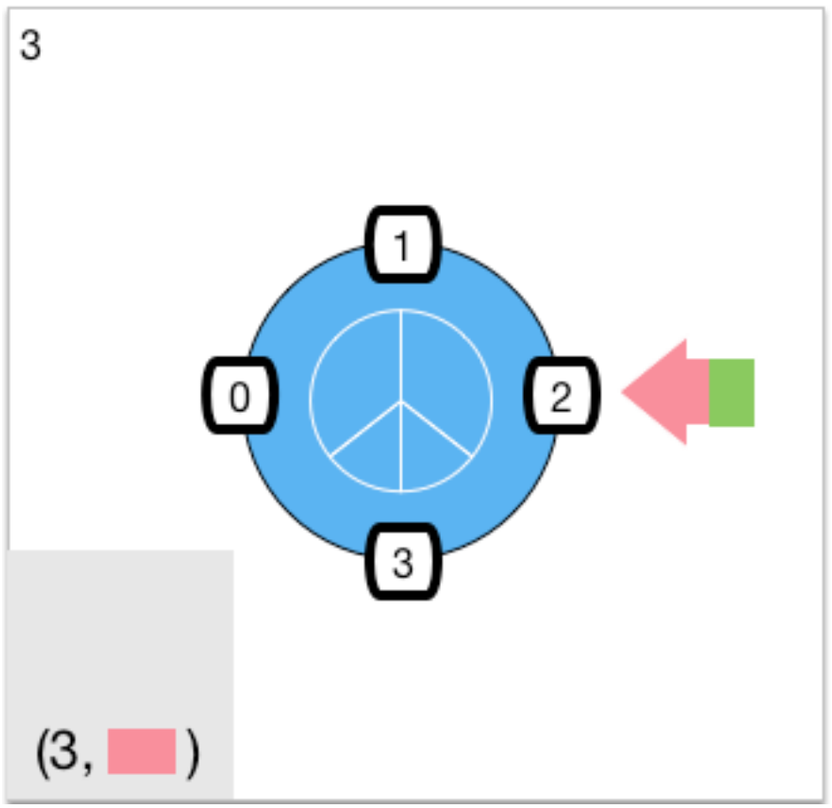
Verilog
+ libraries

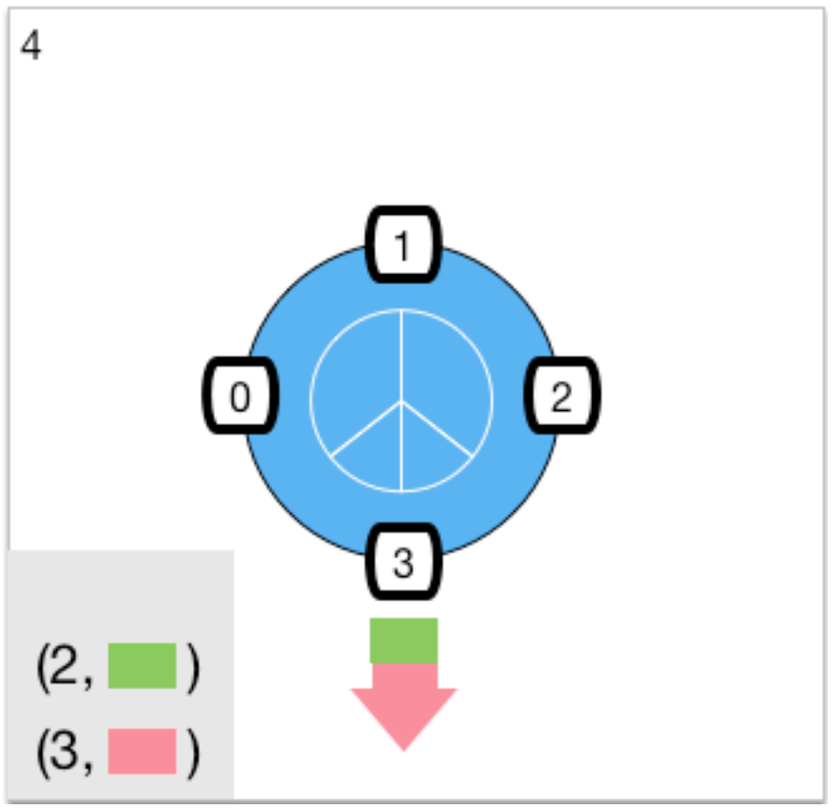


http://github.com/niksu/Pax


1
1
0
2
3


2
1
0
2
3
(3,)


3
1
0
2
3
(3,)

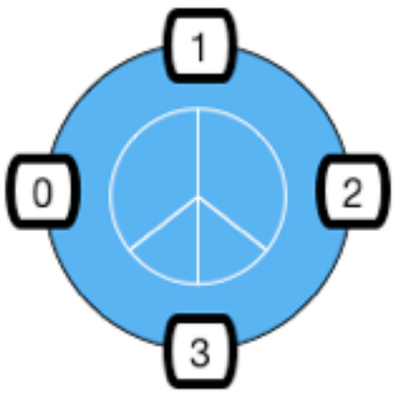

4
1
0
2
3
(2,)
(3,)

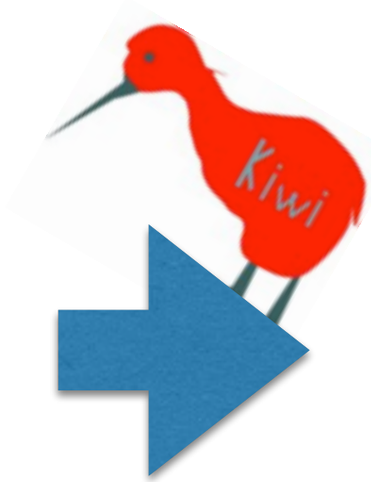
```
private ForwardingDecision OutsideToInside(TEncapsulation packet)
{
  // Retrieve the mapping. If a mapping doesn't exist, then it means that we're not
  // aware of a session to which the packet belongs: so drop the packet.
  var key = new ConnectionKey(packet.GetSourceNode(), packet.GetDestinationNode());
  NatConnection<TPacket,TNode> connection;
  if (NAT_MapToInside.TryGetValue(key, out connection))
  {
    var destination = connection.InsideNode;

    // Update any connection state, including resetting the inactivity timer
    connection.ReceivedPacket(packet, packetFromInside: false);

    // Rewrite the packet destination
    packet.SetDestination(destination);

    // Update checksums
    packet.UpdateChecksums();

    // Forward on the mapped network port
    return new ForwardingDecision.SinglePortForward(destination.InterfaceNumber);
  }
  else
  {
    return Drop;
  }
}
```

(3) Hardware envir.

(High-level) Software description of network program

C#
+ libraries

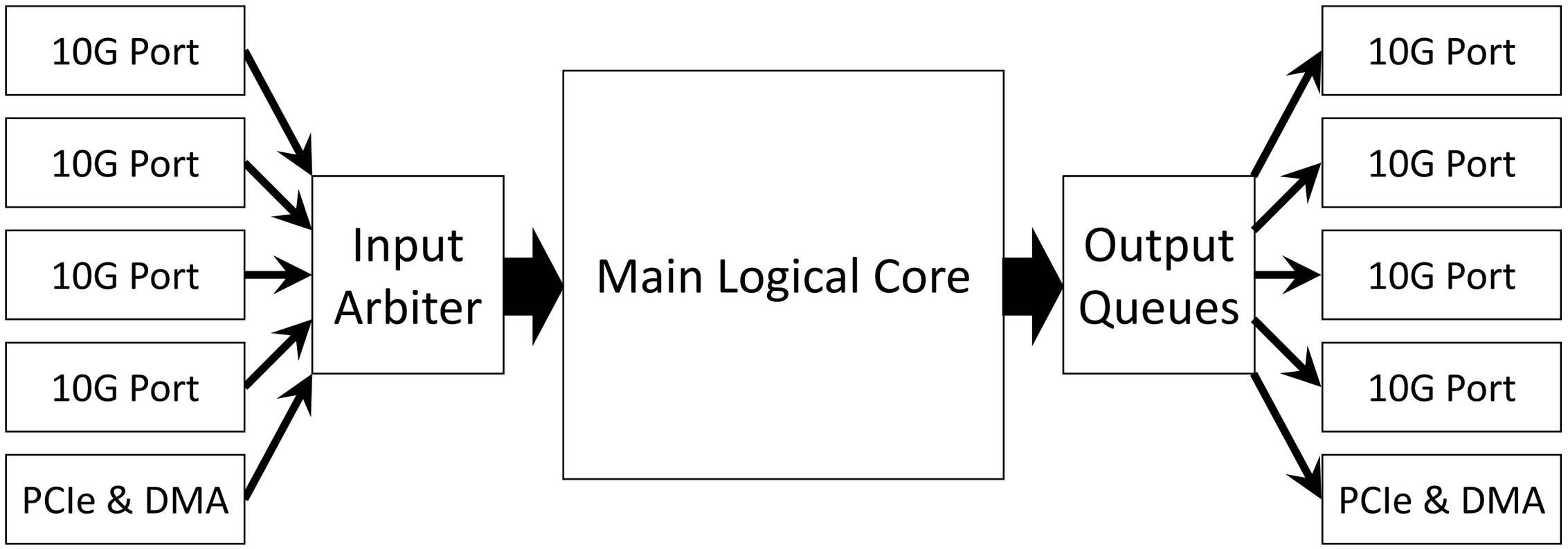




Hardware description of network program

Verilog
+ libraries

http://netfpga.org/


10G Port
10G Port
10G Port
10G Port
PCIe & DMA
Input Arbiter
Main Logical Core
Output Queues
10G Port
10G Port
10G Port
10G Port
PCIe & DMA

Lifting & shadowing

(High-level)
Software
description
of network
program

Hardware
description
of network
program

C#
+ libraries

Verilog
+ libraries

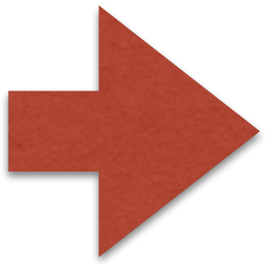
```
[Kiwi.OutputBitPort("NQ_enable")]
protected static bool enable;
[Kiwi.InputBitPort("NQ_ready")]
protected static bool ready;
[Kiwi.InputBitPort("NQ_crashed")]
protected static bool crashed;

[Kiwi.OutputWordPort(3, 0, "NQ_command")]
protected static byte command;

[Kiwi.InputWordPort(3, 0, "NQ_idx_out")]
protected static ulong idx_out;
[Kiwi.InputWordPort(7, 0, "NQ_data_out")]
protected static ulong data_out;

[Kiwi.OutputWordPort(3, 0, "NQ_idx_in")]
protected static ulong idx_in;
[Kiwi.OutputWordPort(7, 0, "NQ_data_in")]
protected static ulong data_in;
```

```
// Nonvolatile copies of outputs.
public static ulong idx_out_nv;
public static ulong data_out_nv;

public static ulong Enlist(ulong data_in)
{
  while (ready) { Kiwi.Pause(); }
  command = (byte)op_code.Enlist;
  NaughtyQ.data_in = data_in;
  enable = true;
  Kiwi.Pause();
  while (!ready) { Kiwi.Pause(); }
  Kiwi.Pause();
  idx_out_nv = idx_out;
  data_out_nv = data_out;
  enable = false;
  Kiwi.Pause();
  return idx_out_nv;
}
```

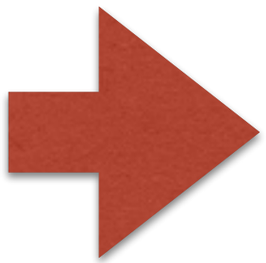
```
[Kiwi.OutputBitPort("NQ_enable")]
protected static bool enable;
[Kiwi.InputBitPort("NQ_ready")]
protected static bool ready;
[Kiwi.InputBitPort("NQ_crashed")]
protected static bool crashed;

[Kiwi.OutputWordPort(3, 0, "NQ_command")]
protected static byte command;

[Kiwi.InputWordPort(3, 0, "NQ_idx_out")]
protected static ulong idx_out;
[Kiwi.InputWordPort(7, 0, "NQ_data_out")]
protected static ulong data_out;

[Kiwi.OutputWordPort(3, 0, "NQ_idx_in")]
protected static ulong idx_in;
[Kiwi.OutputWordPort(7, 0, "NQ_data_in")]
protected static ulong data_in;

// Nonvolatile copies of outputs.
public static ulong idx_out_nv;
public static ulong data_out_nv;
```

```
public static ulong Enlist(ulong data_in)
{
  while (ready) { Kiwi.Pause(); }
  command = (byte)op_code.Enlist;
  NaughtyQ.data_in = data_in;
  enable = true;
  Kiwi.Pause();
  while (!ready) { Kiwi.Pause(); }
  Kiwi.Pause();
  idx_out_nv = idx_out;
  data_out_nv = data_out;
  enable = false;
  Kiwi.Pause();
  return idx_out_nv;
}
```

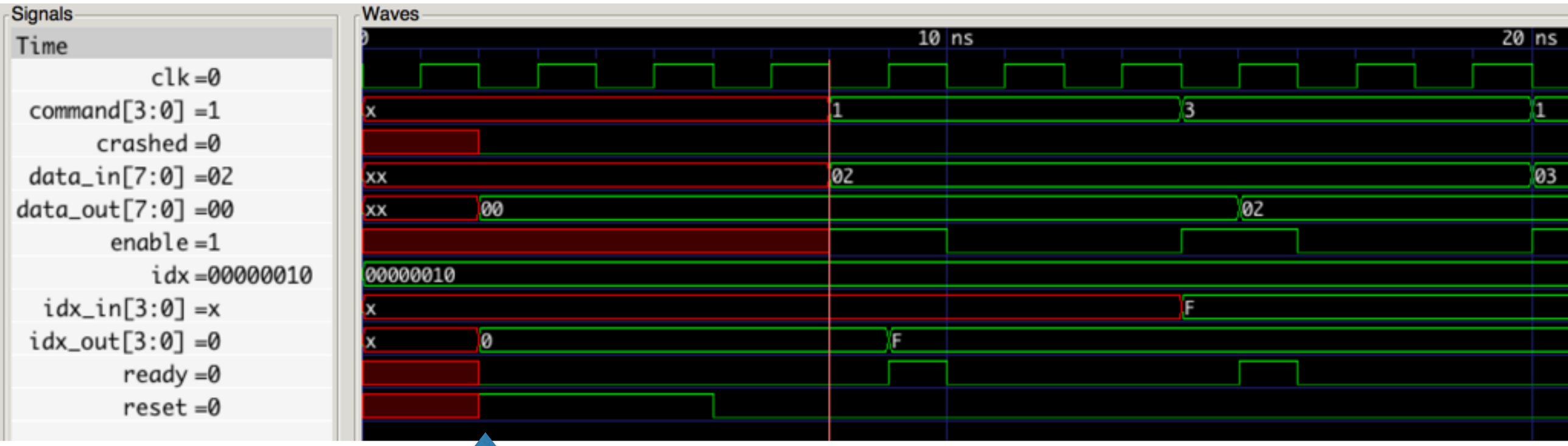
```
public static ulong Enlist(ulong data_in)
{
  while (ready) { Kiwi.Pause(); }
  command = (byte)op_code.Enlist;
  NaughtyQ.data_in = data_in;
  enable = true;
  Kiwi.Pause();
  while (!ready) { Kiwi.Pause(); }
  Kiwi.Pause();
  idx_out_nv = idx_out;
  data_out_nv = data_out;
  enable = false;
  Kiwi.Pause();
  return idx_out_nv;
}
```

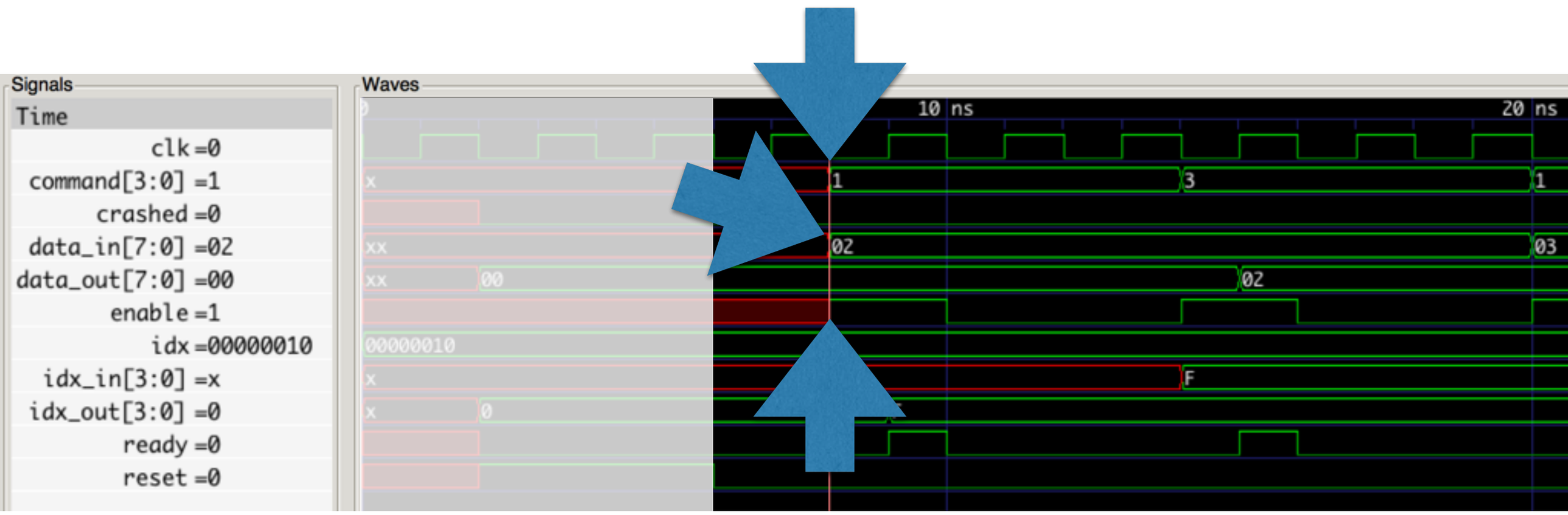


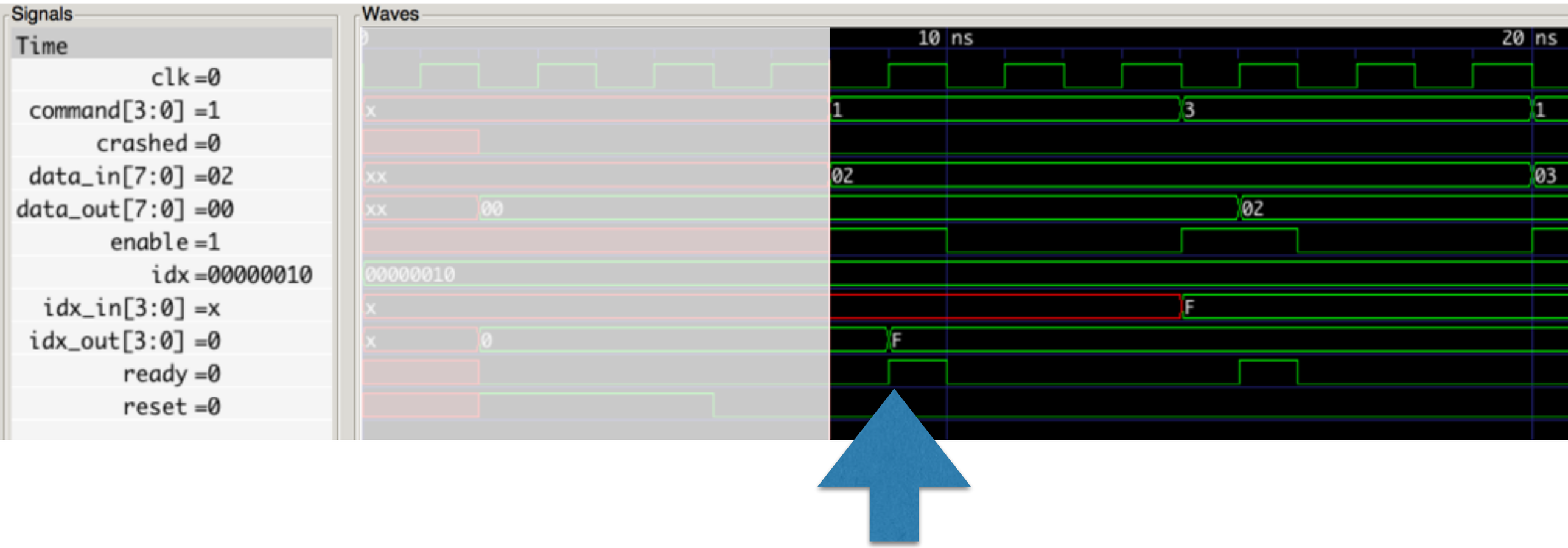
```
public static ulong Enlist(ulong data_in)
{
  while (ready) { Kiwi.Pause(); }
  command = (byte)op_code.Enlist;
  NaughtyQ.data_in = data_in;
  enable = true;
  Kiwi.Pause();
  while (!ready) { Kiwi.Pause(); }
  Kiwi.Pause();
  idx_out_nv = idx_out;
  data_out_nv = data_out;
  enable = false;
  Kiwi.Pause();
  return idx_out_nv;
}
```

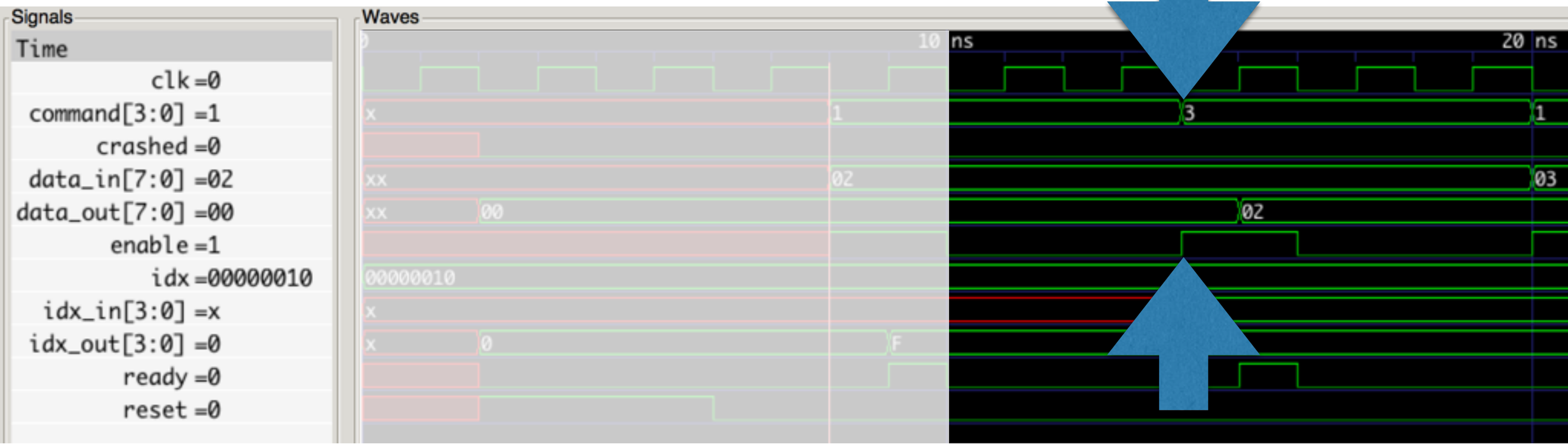


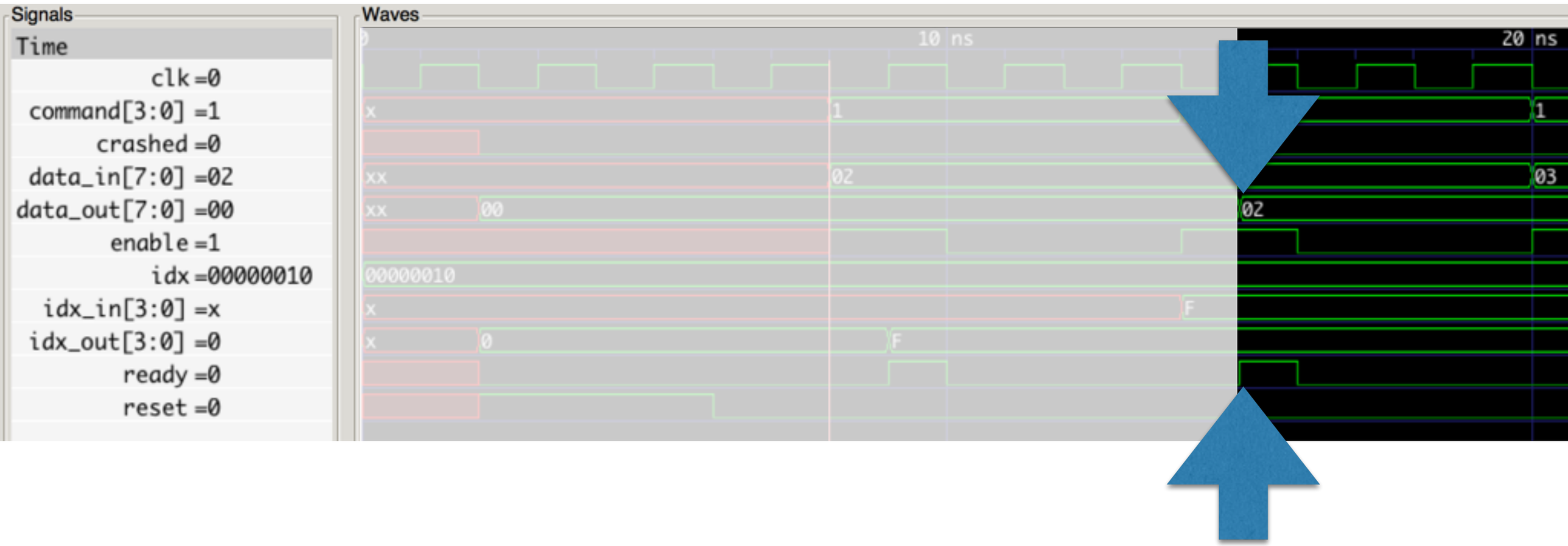
```
public static ulong Enlist(ulong data_in)
{
  while (ready) { Kiwi.Pause(); }
  command = (byte)op_code.Enlist;
  NaughtyQ.data_in = data_in;
  enable = true;
  Kiwi.Pause();
  while (!ready) { Kiwi.Pause(); }
  Kiwi.Pause();
  idx_out_nv = idx_out;
  data_out_nv = data_out;
  enable = false;
  Kiwi.Pause();
  return idx_out_nv;
}
```

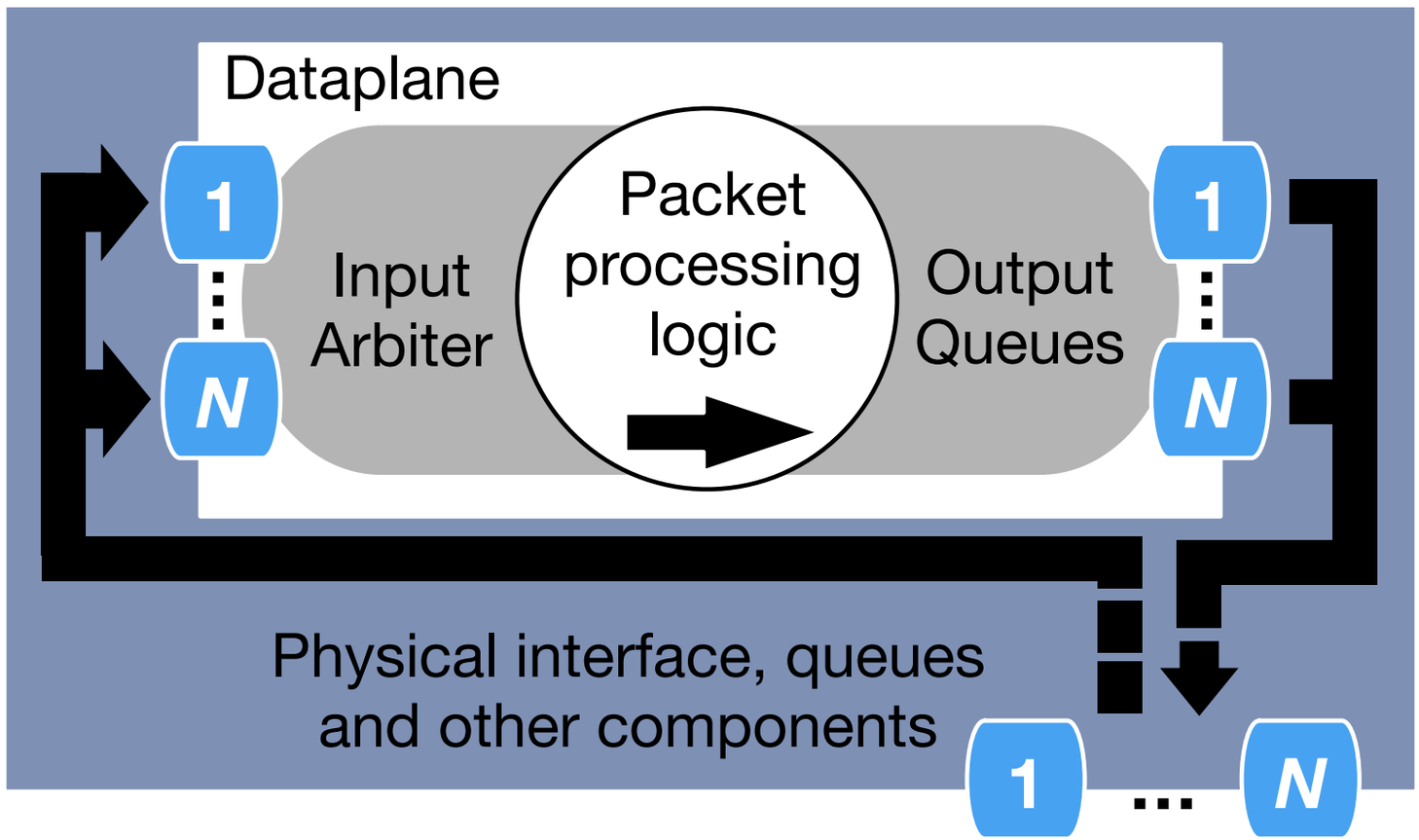

Signals
Waves
Time
clk =0
command[3:0] =1
crashed =0
data_in[7:0] =02
data_out[7:0] =00
enable =1
idx =00000010
idx_in[3:0] =x
idx_out[3:0] =0
ready =0
reset =0
10 ns
20 ns
x
1
3
1
xx
02
03
xx
00
0Z
00000010
x
F
x
0
F


Signals
Waves
Time
clk =0
command[3:0] =1
crashed =0
data_in[7:0] =02
data_out[7:0] =00
enable =1
idx =00000010
idx_in[3:0] =x
idx_out[3:0] =0
ready =0
reset =0
10 ns
20 ns
1
3
1
02
03
00
02
F
0
0Z


Signals
Waves
Time
clk =0
command[3:0] =1
crashed =0
data_in[7:0] =02
data_out[7:0] =00
enable =1
idx =00000010
idx_in[3:0] =x
idx_out[3:0] =0
ready =0
reset =0
10 ns
20 ns
1
3
1
02
03
02
00000010
F
F


Signals
Waves
Time
clk =0
command[3:0] =1
crashed =0
data_in[7:0] =02
data_out[7:0] =00
enable =1
idx =00000010
idx_in[3:0] =x
idx_out[3:0] =0
ready =0
reset =0
10 ns
20 ns
x
1
3
1
xx
02
03
xx
00
02
00000010
x
x
0
F

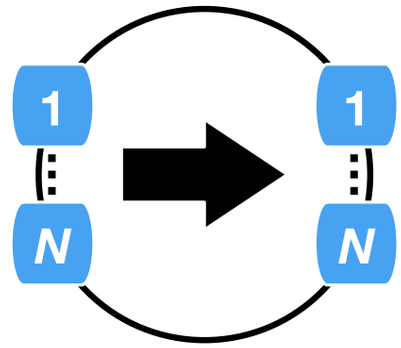

Signals
Waves
Time
clk =0
command[3:0] =1
crashed =0
data_in[7:0] =02
data_out[7:0] =00
enable =1
idx =00000010
idx_in[3:0] =x
idx_out[3:0] =0
ready =0
reset =0
10 ns
20 ns
x
1
xx
02
xx
00
00000010
x
F
x
0
F
1
03
02


Dataplane
1
N
Input
Arbiter
Packet
processing
logic
Output
Queues
1
N
Physical interface, queues
and other components
1
...
N

Workflow

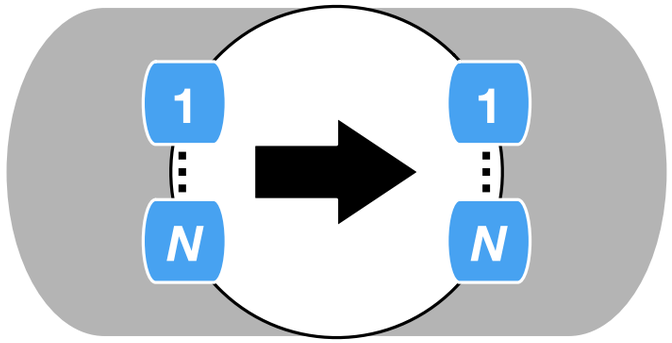
HDL-based development and integration stages

Packet processor modelled in Verilog



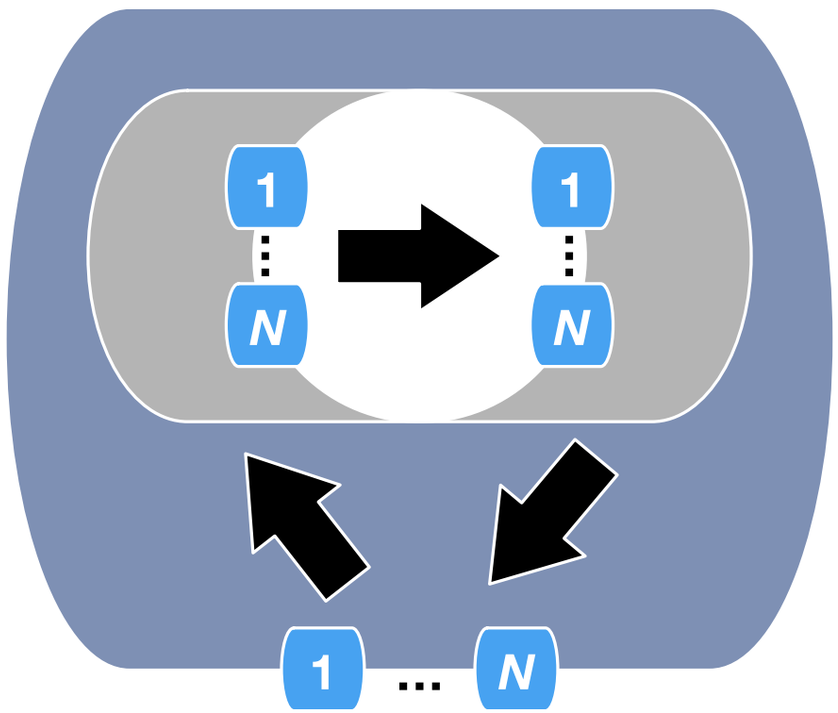
Embedding in **dataplane**

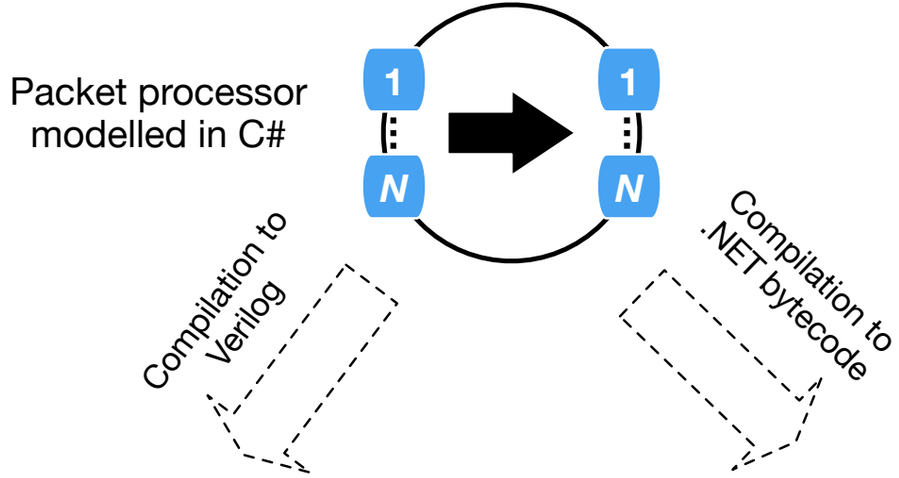
Design is tested using a **testbench** that simulates the hardware being fed a set of packets, and checking the packets that result.



Embedding in rest of **pipeline**

The design is ultimately processed by FPGA vendor-specific tools to generate an image that programs the FPGA.
Packets received on physical network ports are processed using our logic.





HDL-based development and integration stages

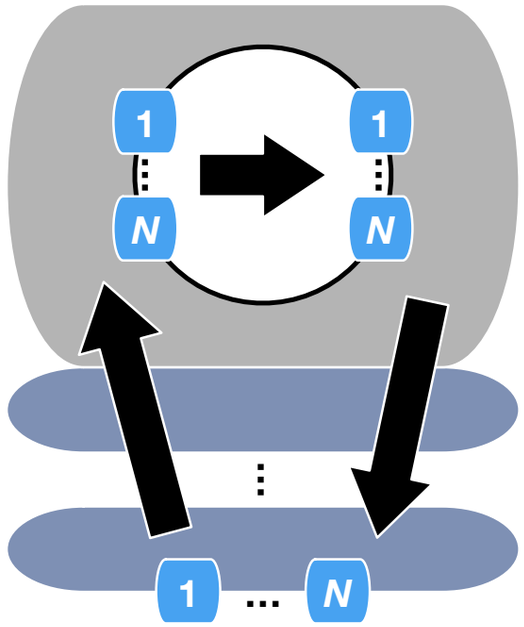
Packet processor modelled in Verilog

Embedding in **dataplane**

Design is tested using a **testbench** that simulates the hardware being fed a set of packets, and checking the packets that result.

Embedding in rest of **pipeline**

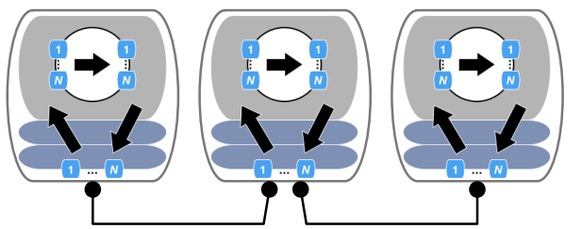
The design is ultimately processed by FPGA vendor-specific tools to generate an image that programs the FPGA. Packets received on physical network ports are processed using our logic.



.NET bytecode can be executed in .NET VM on various OSs, and debugged using existing tools.

Layers of abstraction between the .NET VM and the OS-provided **virtual** or **physical** network interfaces.

Virtualisation of interfaces enables us to use the packet processor inside a **network simulator**.



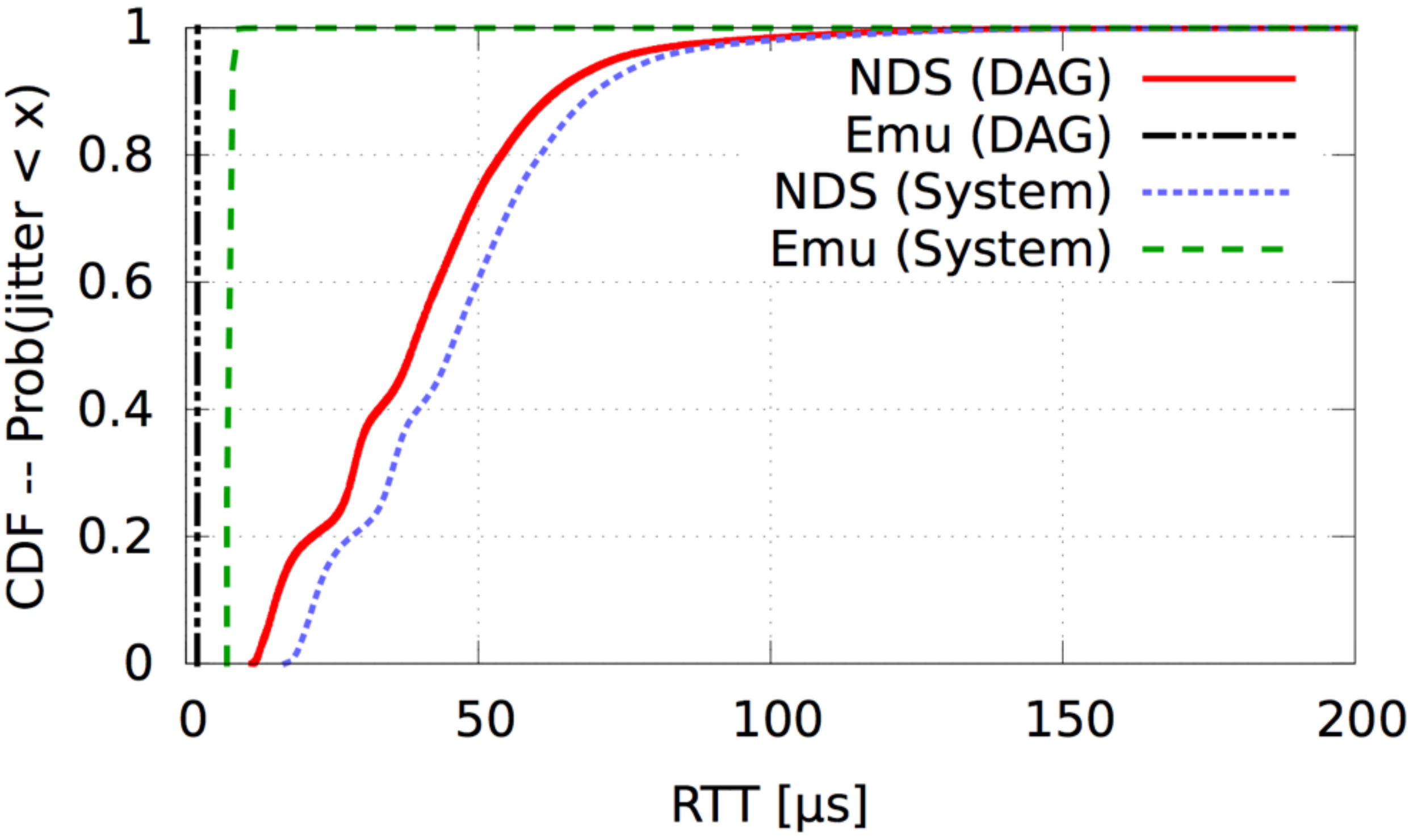
Network-scale testing can be done cheaply and easily within a single machine.

Some examples

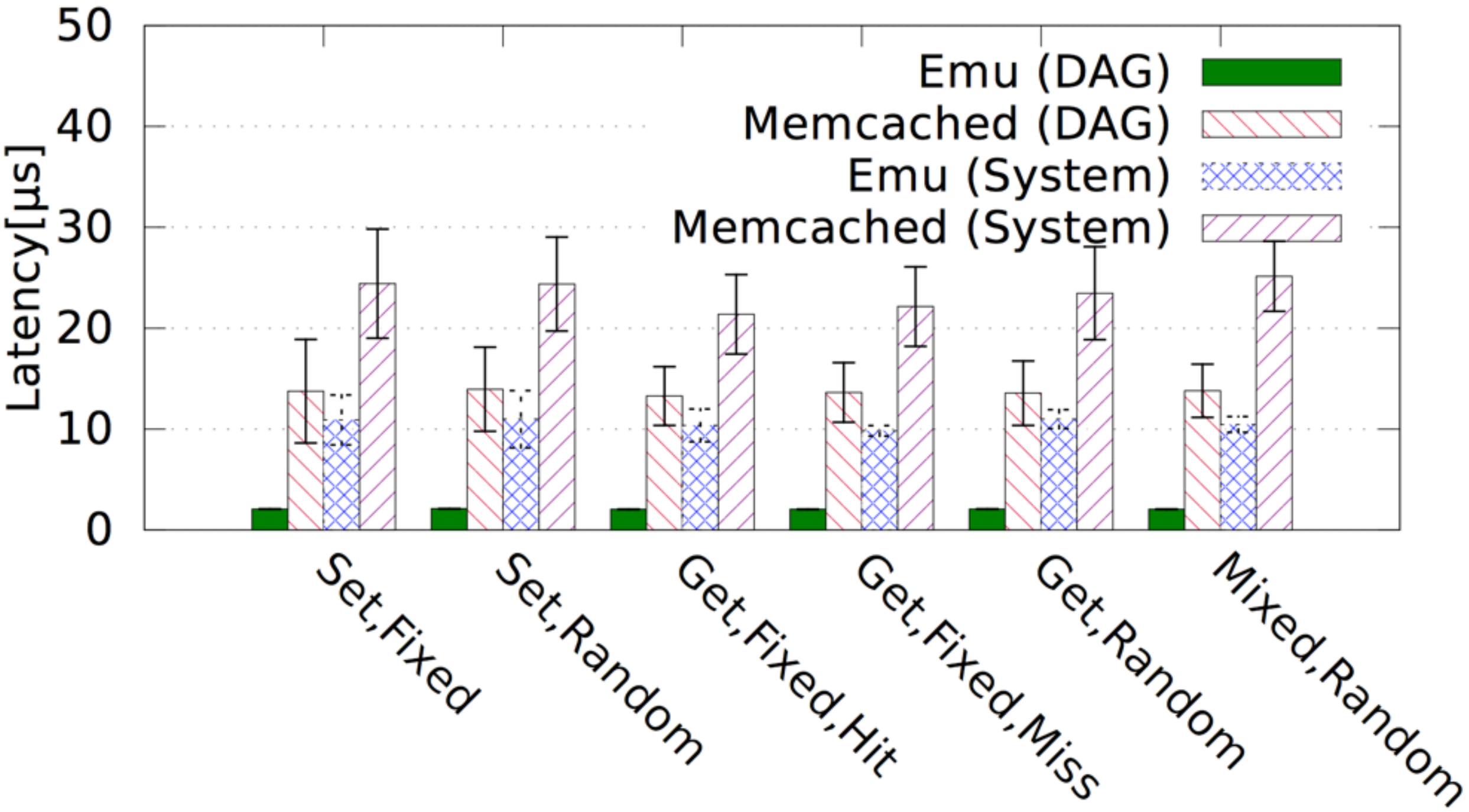
- Learning switch
- ICMP echo and TCP ping
- DNS
- Memcached
- NAT

Some results

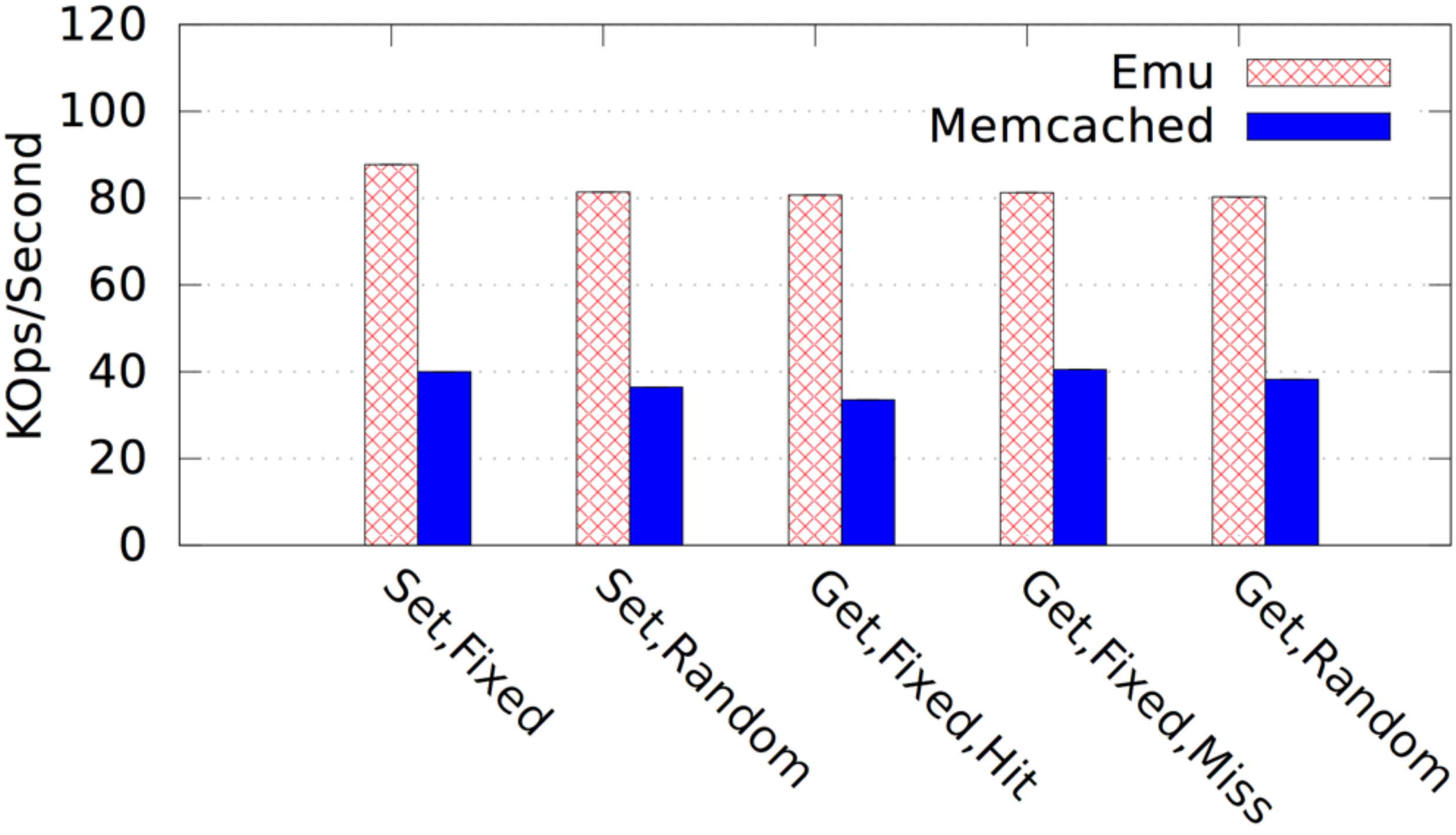
Platform	Reference Switch	Emu Switch
Logic Utilisation	11.42%	12.9 %
Memory Utilisation	13.23%	13.5%
Bandwidth	32Gbps	32.7Gbps
Port-to-port Latency	823ns	825ns



(a) DNS Hit - CDF of Query Latency



(a) Memcached - Latency

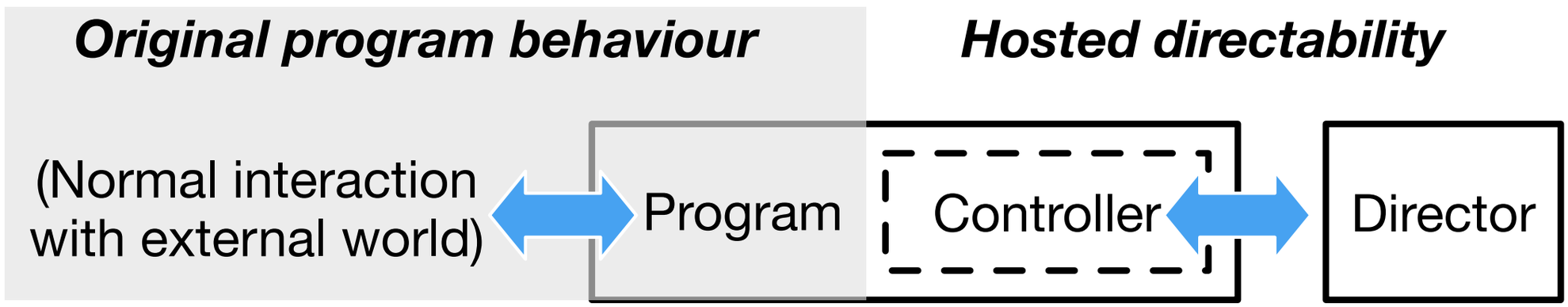


(b) Memcached - Throughput

High-level
development and **debugging**
of
FPGA-based network programs

“Program-hosted Directability” (PhD)

- Program direction
- PhD: transforming programs to **host their own directability features**.
- “direction feature” becomes a program in **constrained language**.
- Can **invoke/reconfigure** these features **at runtime**.

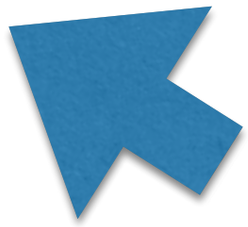

Original program behaviour
(Normal interaction with external world)
Program
Hosted directability
Controller
Director

trace V max_trace_idx

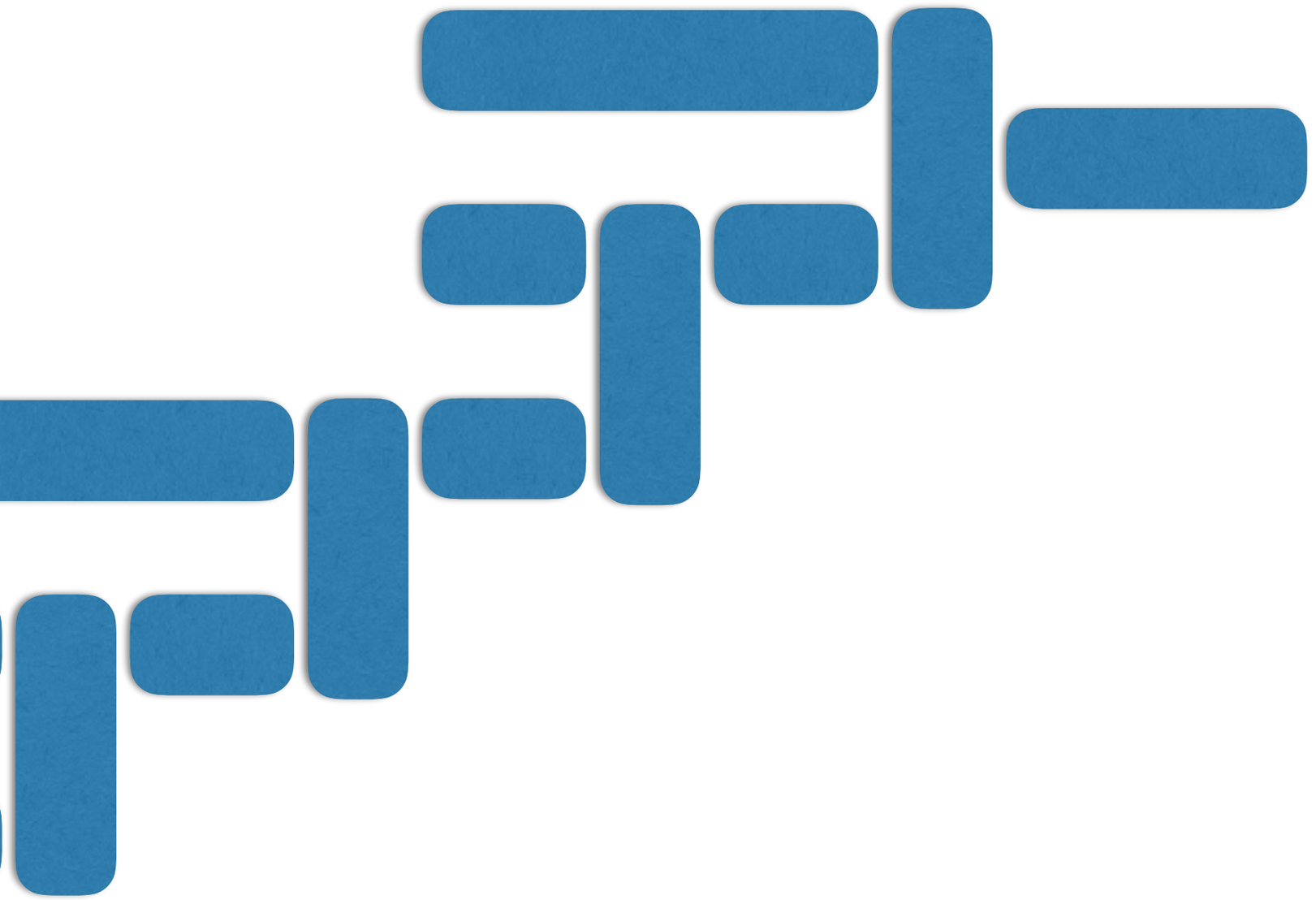
```
g();
V = f(X, Y);
N++;
```

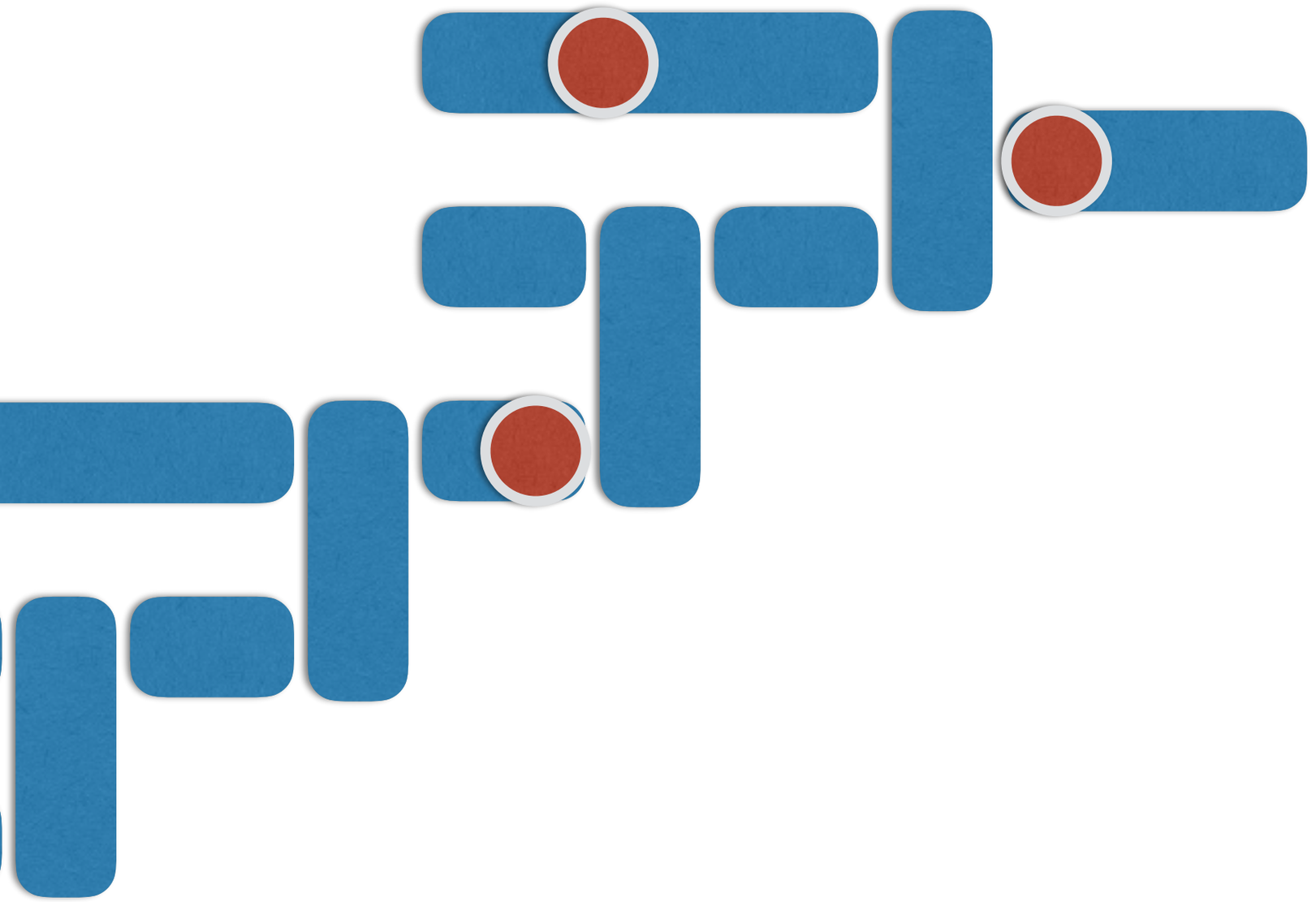
trace V max_trace_idx

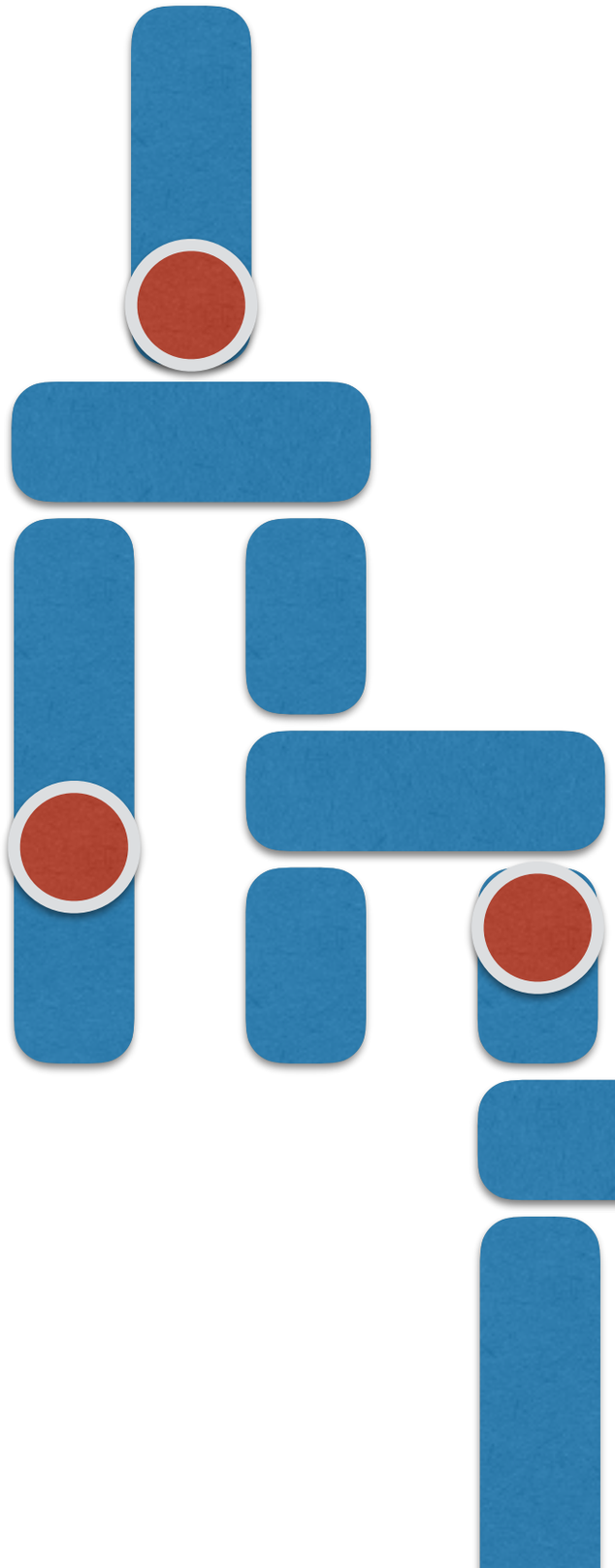
```
g();
V = f(X, Y);
N++;
```



```
if V_trace_idx < max_trace_idx then
    V_trace_buf[V_trace_idx] := V;
    inc V_trace_idx;
    continue
else
    inc V_trace_overflow;
    break
```



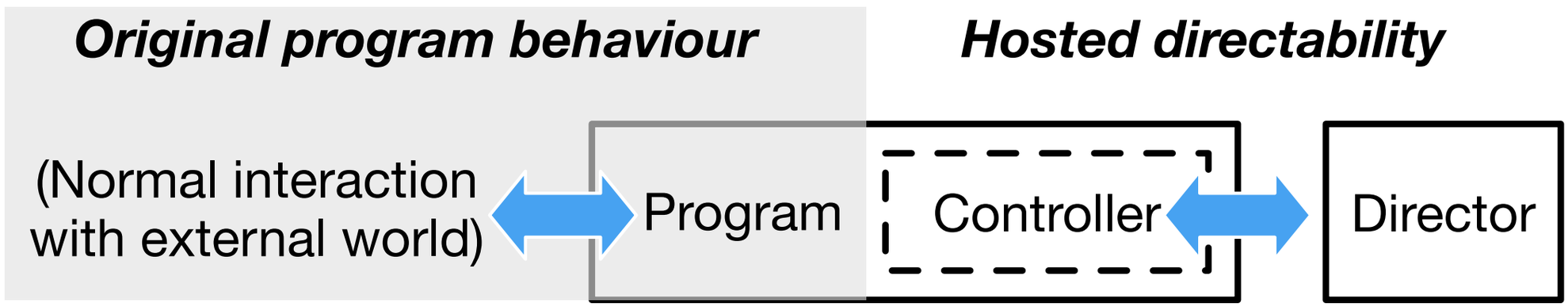


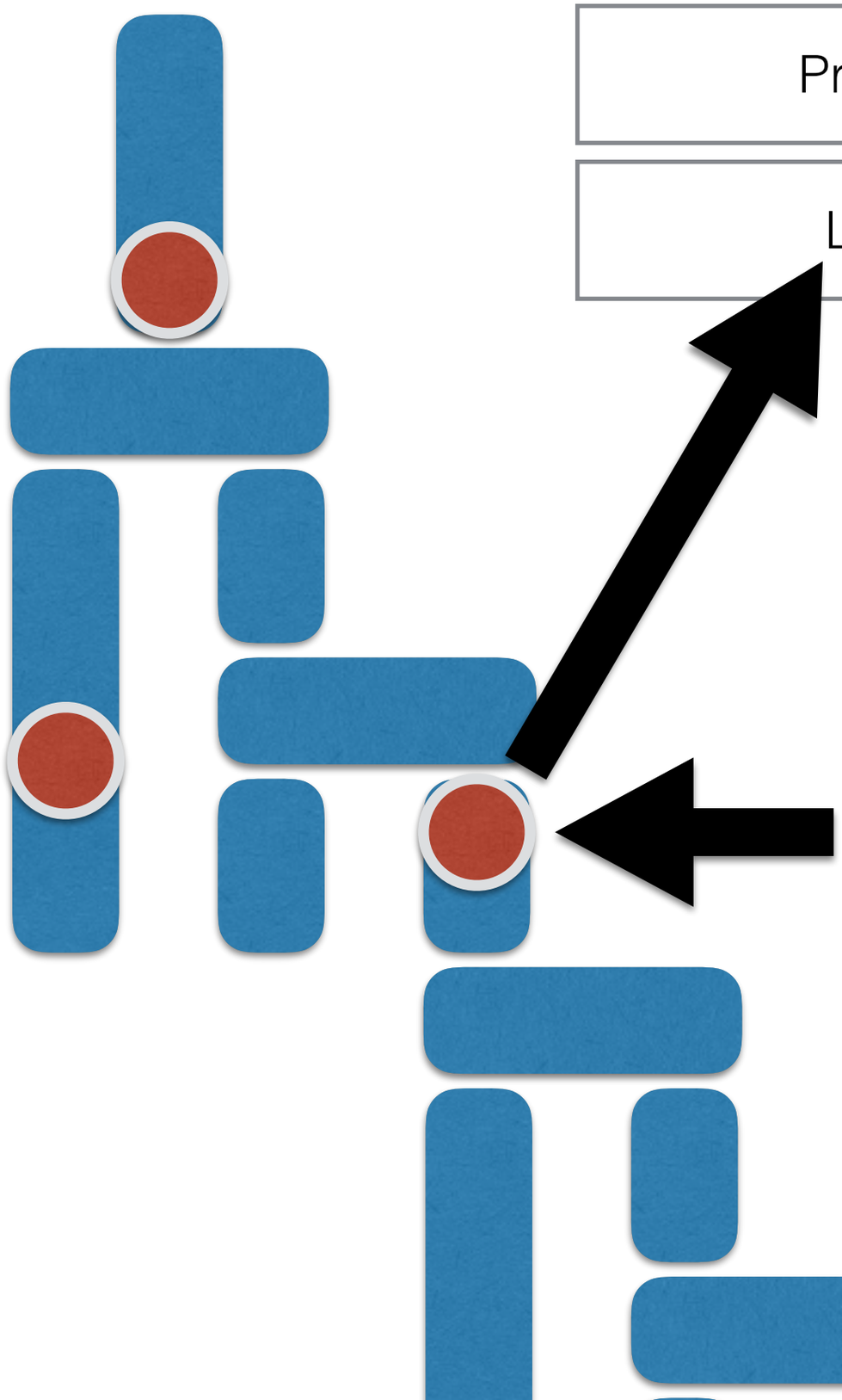


Program memory

Location code

Controller's memory


Original program behaviour
(Normal interaction with external world)
Program
Hosted directability
Controller
Director



Program memory

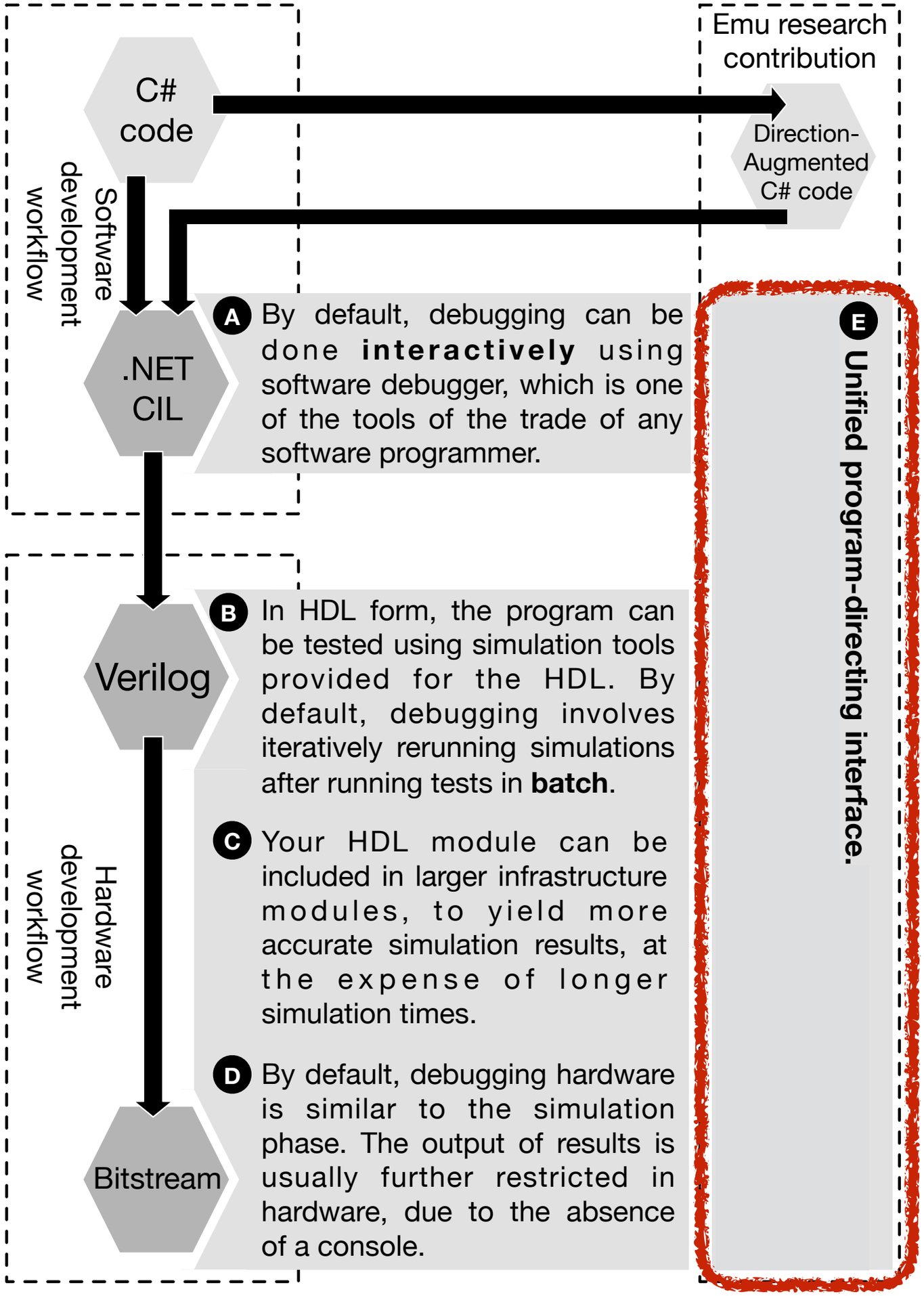
Location code

Controller's memory

```
if V_trace_idx < max_trace_idx then
     V_trace_buf[V_trace_idx] := V;
     inc V_trace_idx;
     continue
else
     inc V_trace_overflow;
     break
```

Command	Behaviour
print X	Print the value of variable X from the source program.
break L $\langle B \rangle$	Activate a (conditional) breakpoint at the position of label L.
unbreak L	Deactivate a breakpoint.
backtrace $\langle \$ \rangle$	Print the "function call stack".
watch X $\langle B \rangle$	Break when X is updated and satisfies a given condition.
unwatch X	Cancel the effect of the "watch" command.
count { reads X $\langle B \rangle$ $\langle \$ \rangle$ / writes X $\langle B \rangle$ $\langle \$ \rangle$ / calls *fname* $\langle B \rangle$ $\langle \$ \rangle$ }	Count the reads or writes to a variable X, or the calls to a function *fname*.
trace { start X $\langle B \rangle$ $\langle \$ \rangle$	Trace a variable, subject to a condition being satisfied, and up to trace some length.
trace { stop X	Stop tracing a variable.
trace { clear X	Clear a variable's trace buffer.
trace { print X	Print the contents of a variable's trace buffer.
trace { full X	Check if a variable's trace buffer is full.

Table 2. Directing commands making up language $\mathfrak{D}$. Note that count has similar subcommands to those of trace, to clear the counters, get their current value, and find out if a maximum value has been reached.


Emu research contribution
C# code
Direction-Augmented C# code
Software development workflow
.NET CIL
A By default, debugging can be done **interactively** using software debugger, which is one of the tools of the trade of any software programmer.
Verilog
B In HDL form, the program can be tested using simulation tools provided for the HDL. By default, debugging involves iteratively rerunning simulations after running tests in **batch**.
C Your HDL module can be included in larger infrastructure modules, to yield more accurate simulation results, at the expense of longer simulation times.
Hardware development workflow
Bitstream
D By default, debugging hardware is similar to the simulation phase. The output of results is usually further restricted in hardware, due to the absence of a console.
E **Unified program-directing interface.**

Artefact	**Utilisation (%)**		**Performance**		
	Logic	Flip-flops	Duration (#cycles)	Latency (μs)	Queries-per-sec (KQPS)
DNS+ELA	99.74	100.40	57	1.83	1176
DNS+2e	234.61	151.06	57	1.86	1176
(Count)	234.46	151.81	62	1.94	1064
(Trace)	218.30	151.84	70	1.99	1010

Table 4. Utilisation and performance profile of the DNS+ELA against the DNS having one extension point, where the extension point is NOP, packet counting, or variable tracing. Latency is indicated at the 99^{th} percentile.

System \ Features	state inspection	trace recording	state updating	extension points	break points	stepping	interruption	fine granularity	assertion checking	hang detection	timing checks	software instance	runtime reconfigurable	HLS (vs HDL)	Network/Control	use leftover resource	embed at Source/HDL
(Sosič 1992) **Dynascope**	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓	✓				
(Goeders and Wilton 2014) **HLS-Scope**	✓	✓			✓	✓		✓						✓	C		S
(Calagar et al. 2014) **Inspect**	✓				✓	✓		✓				✓		✓	C		H
(Panjkov et al. 2015)	✓													✓	C		S
(Hung and Wilton 2014) **QuickTrace**	✓							✓				✓	✓	✓	N	✓	H
(Koch et al. 1998) **SLE/CADDY**	✓				✓	✓		✓				✓		✓	C		H
(Monson and Hutchings 2015)	✓	✓												✓	C		S
(Curreri et al. 2011)									✓	✓	✓			✓	C		H
(Camera et al. 2005) **BORPH**	✓	✓	✓		✓	✓	✓	✓							C		H
(see §2.5) **PhD**	✓	✓	✓	✓	✓		✓	✓	✓			✓	✓		C		

Table 1. Survey of features provided by debugging systems. Blacked-out boxes mean "not applicable".

Conclusion

- **Goal**: HLS-based development of network programs

Conclusion

- **Goal**: HLS-based development of network programs
- **Currently**: done using HDL or DSL. We use HLS.

Conclusion

- **Goal**: HLS-based development of network programs
- **Currently**: done using HDL or DSL. We use HLS.
- **What's new**: lifting+shadowing, host support environment, improved debugging support.

Conclusion

- **Goal**: HLS-based development of network programs
- **Currently**: done using HDL or DSL. We use HLS.
- **What's new**: lifting+shadowing, host support environment, improved debugging support.
- **Relevance**: will help experienced and novice users of FPGAs, at a time when FPGAs becoming more prevalent.

Thank you

naas-project.org

Extra slides

Three examples:

- print
- break
- unbreak

$$(\mathsf{break}\ L\ \langle B \rangle) \in \mathfrak{C} \qquad X \in \mathrm{Var}_p$$

$$\frac{(\mathsf{break}\ L\ \langle B \rangle) \in \mathfrak{C} \qquad X \in \mathrm{Var}_p}{p \ \overset{\mathsf{print}\ X}{\sqsubset_{\mathfrak{C}}}\ p}$$

(need differentiating criteria)

$$\frac{(\mathsf{break}\ L\ \langle B \rangle) \in \mathfrak{C} \qquad X \in \mathrm{Var}_p}{p \quad \begin{array}{c}\mathsf{print}\ X \\ \sqsubset_{\mathfrak{C}}\end{array} \quad p \left\{ \begin{array}{l} \breve{\mathcal{D}} = \{\} \\ \breve{\mathcal{C}} = \{\} \\ D_c = \lambda \mathcal{D}'.\ \ X \rightsquigarrow N; \\ \qquad\qquad\qquad \mathrm{print}(N) \end{array} \right.}$$

$$\text{conditional } \langle B \rangle\ t\ = \begin{cases} t & \text{if } \langle B \rangle = \text{true} \\ \texttt{if } I_1 == I_2 \texttt{ then } t & \text{if } \langle B \rangle = (I_1 = I_2) \\ \quad \texttt{else continue} & \end{cases}$$

$$[\![\text{break } L\ \langle B \rangle]\!]_{SP} = \text{conditional } \langle B \rangle\ \texttt{break}$$

$$\text{conditional } \langle B \rangle\ t\ = \begin{cases} t & \text{if } \langle B \rangle = \text{true} \\ \texttt{if } I_1 == I_2 \texttt{ then } t & \text{if } \langle B \rangle = (I_1 = I_2) \\ \quad \texttt{else continue} & \end{cases}$$

$$[\![\text{break } L\ \langle B\rangle]\!]_{SP} = \text{conditional } \langle B\rangle\ \texttt{break}$$

$$[\![\text{break } L\ \langle B\rangle]\!] = @L : \{[\![\text{break } L\ \langle B\rangle]\!]_{SP}\}$$

$$\text{conditional } \langle B\rangle\ t\ =$$

$$\begin{cases} t & \text{if } \langle B\rangle = \text{true} \\ \texttt{if } I_1 == I_2 \texttt{ then } t & \text{if } \langle B\rangle = (I_1 = I_2) \\ \quad \texttt{else continue} & \end{cases}$$

$$\frac{L \notin p \qquad p <^{1}_{L} p'}{p \overset{\text{break } L \langle B \rangle}{\sqsubset_{\mathfrak{C}}} p'}$$

$$[\![\text{break } L \langle B \rangle]\!]_{SP} = \text{conditional } \langle B \rangle \texttt{ break}$$
$$[\![\text{break } L \langle B \rangle]\!] = @L : \{[\![\text{break } L \langle B \rangle]\!]_{SP}\}$$
$$\text{conditional } \langle B \rangle \; t \; =$$
$$\begin{cases} t & \text{if } \langle B \rangle = \text{true} \\ \texttt{if } I_1 == I_2 \texttt{ then } t & \text{if } \langle B \rangle = (I_1 = I_2) \\ \quad \texttt{else continue} & \end{cases}$$

$$\frac{L \notin p \qquad p <^{1}_{L} p'}{p \quad \overset{\text{break } L \ \langle B \rangle}{\sqsubset_{\mathfrak{C}}} \quad p' \ \begin{cases} \breve{\mathcal{D}} = \{(\text{«bp»}, L, 1)\} \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{cases}}$$

$$[\![\text{break } L \ \langle B \rangle]\!]_{SP} = \text{conditional } \langle B \rangle \ \texttt{break}$$

$$[\![\text{break } L \ \langle B \rangle]\!] = @L : \{[\![\text{break } L \ \langle B \rangle]\!]_{SP}\}$$

$$\text{conditional } \langle B \rangle \ t \ =$$

$$\begin{cases} t & \text{if } \langle B \rangle = \text{true} \\ \texttt{if } I_1 == I_2 \texttt{ then } t & \text{if } \langle B \rangle = (I_1 = I_2) \\ \quad \texttt{else continue} & \end{cases}$$

$$\frac{L \notin p \qquad p <^1_L p'}{p \quad \overset{\text{break } L \ \langle B \rangle}{\sqsubset_{\mathfrak{C}}} \quad p' \ \begin{cases} \breve{\mathcal{D}} = \{(\text{«bp»}, L, 1)\} \\ \breve{\mathcal{C}} = \{SP[L \mapsto [\![\text{break } L \ \langle B \rangle]\!]_{SP}]\} \end{cases}}$$

$$[\![\text{break } L \ \langle B \rangle]\!]_{SP} = \text{conditional } \langle B \rangle \ \texttt{break}$$

$$[\![\text{break } L \ \langle B \rangle]\!] = @L : \{[\![\text{break } L \ \langle B \rangle]\!]_{SP}\}$$

$$\text{conditional } \langle B \rangle \ t \ = \begin{cases} t & \text{if } \langle B \rangle = \text{true} \\ \texttt{if } I_1 == I_2 \texttt{ then } t & \text{if } \langle B \rangle = (I_1 = I_2) \\ \quad \texttt{else continue} & \end{cases}$$

$$\frac{L \notin p \qquad p <^1_L p'}{p \overset{\text{break } L \langle B \rangle}{\sqsubset_{\mathfrak{C}}} p' \begin{cases} \breve{\mathcal{D}} = \{(\text{«bp»}, L, 1)\} \\ \breve{\mathcal{C}} = \{SP[L \mapsto [\![\text{break } L \langle B \rangle]\!]_{SP}]\} \\ D_c = \lambda \mathcal{D}'.\ \text{if } (\text{«bp»}, L, 1) \in \mathcal{D}' \text{ then } \mathcal{D}' \\ \qquad\qquad \text{else} \\ \qquad\qquad\quad [\![\text{break } L \langle B \rangle]\!] \rightsquigarrow \ulcorner L \urcorner; \\ \qquad\qquad\quad (\text{«bp»}, L, 0 \mapsto 1) :\in \mathcal{D}' \end{cases}}$$

where

$$[\![\text{break } L \langle B \rangle]\!]_{SP} = \text{conditional } \langle B \rangle \texttt{ break}$$

$$[\![\text{break } L \langle B \rangle]\!] = @L : \{[\![\text{break } L \langle B \rangle]\!]_{SP}\}$$

$$\text{conditional } \langle B \rangle\ t\ = \begin{cases} t & \text{if } \langle B \rangle = \text{true} \\ \texttt{if } I_1 == I_2 \texttt{ then } t & \text{if } \langle B \rangle = (I_1 = I_2) \\ \quad \texttt{else continue} & \end{cases}$$

$$(\mathsf{break}\ L\ \langle B \rangle) \in \mathfrak{C}$$

$$p \quad \overset{\mathsf{unbreak}\ L}{\sqsubset_{\mathfrak{C}}} \quad p$$

$$\frac{(\textsf{break}\ L\ \langle B \rangle) \in \mathfrak{C}}{p \xrightarrow[\sqsubset \mathfrak{C}]{\textsf{unbreak}\ L} p}$$

$$[\![\textsf{unbreak}\ L]\!] = @L : \{\texttt{continue}\}$$

$$(\textsf{break}\ L\ \langle B \rangle) \in \mathfrak{C}$$

$$p \quad \overset{\textsf{unbreak}\ L}{\sqsubset_{\mathfrak{C}}} \quad p \left\{ \begin{array}{l} \breve{\mathcal{D}} = \{\} \\ \breve{\mathcal{C}} = \{\} \\ D_c = \lambda \mathcal{D}'.\ \text{if } (\text{«bp»}, L, 0) \in \mathcal{D}' \text{ then } \mathcal{D}' \\ \qquad\qquad \text{else } [\![\textsf{unbreak}\ L]\!] \rightsquigarrow \ulcorner L \urcorner; \\ \qquad\qquad (\text{«bp»}, L, 1 \mapsto 0) :\in \mathcal{D}' \end{array} \right.$$

where

$$[\![\textsf{unbreak}\ L]\!] = @L : \{\texttt{continue}\}$$