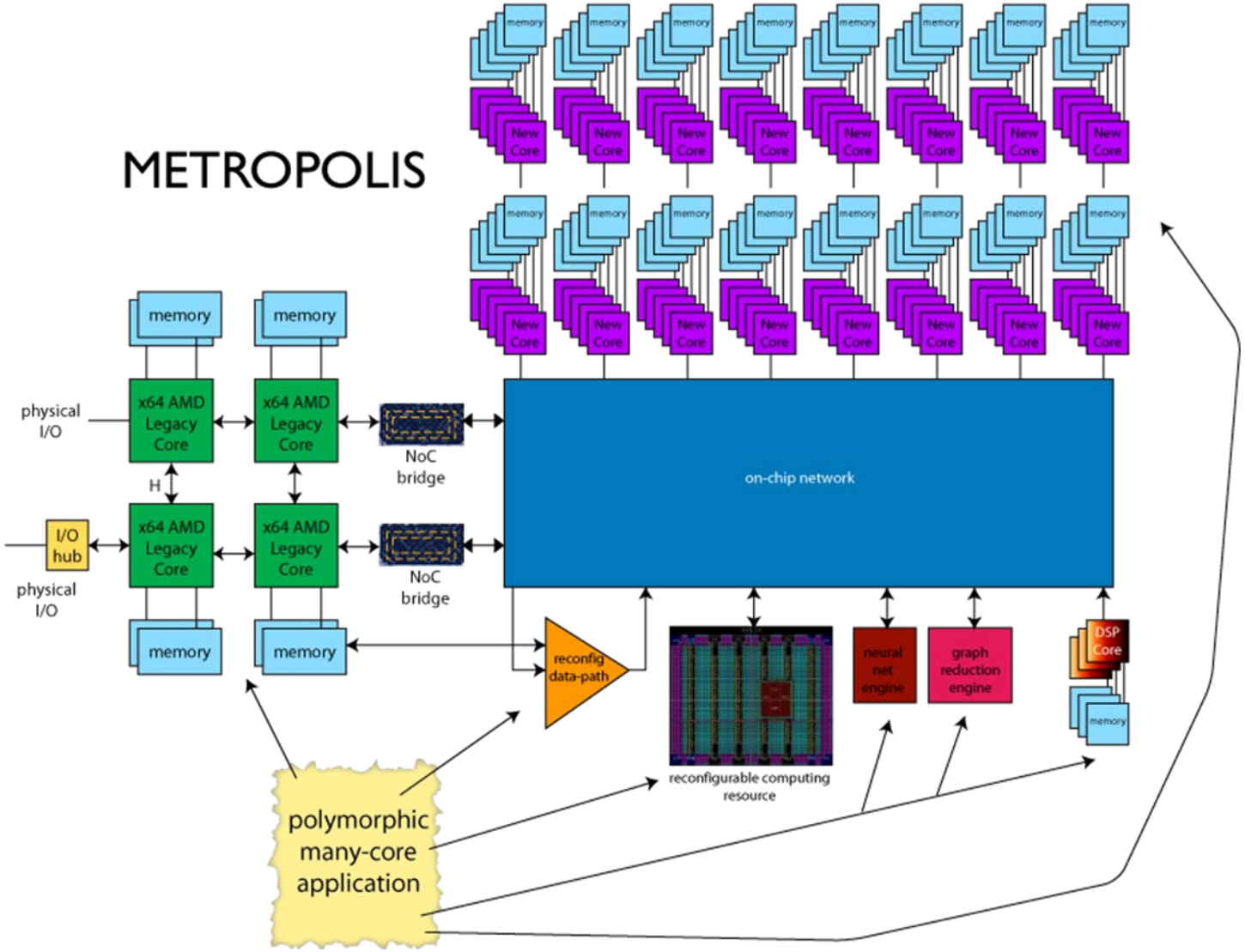
Programming Many-core Systems with Embedded Domain Specific Languages

Satnam Singh
satnams@microsoft.com

METROPOLIS


memory
New Core
x64 AMD Legacy Core
physical I/O
H
I/O hub
NoC bridge
on-chip network
reconfig data-path
reconfigurable computing resource
neural net engine
graph reduction engine
DSP Core
polymorphic many-core application

Heterogeneous => Multilingual?

- If future systems are heterogeneous then do we need to learn different languages and tools for different types of processing elements?
  - Concurrent and a parallel programming in C#, F#?
  - Parallel functional programming in Haskell?
  - VHDL and Verilog for hardware?
  - CUDA or Accelerator for GPUs?

Functional Domain Specific Languages

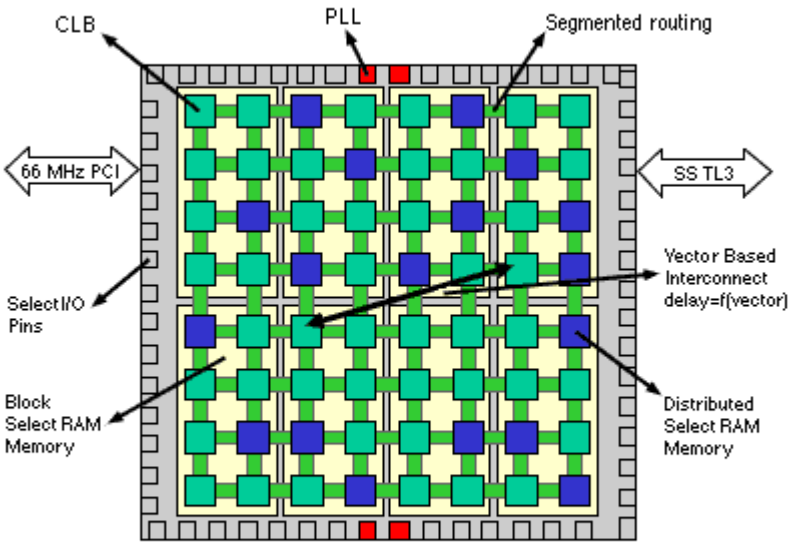
- A domain specific language captures the **essence** of a class of useful applications.
- Say what is important, leave out what is not.
- Greater freedom to compile to heterogeneous targets.
- **Embedded** Domain Specific Languages.

An Example of a DSL: Lava

- Haskell library
- Higher order functions for circuit behaviour and layout composition
- Library of basic Xilinx FPGA gates
- Library of useful circuits (arithmetic)
- Generates EDIF, VHDL and Verilog
- Interfaces to existing circuit design tools
- Chalmers: formal verification
- Other targets include multi-threaded software and GPUs

FPGAs

Functional Block Diagram



The New Xilinx Virtex architecture features Xilinx SelectRAM+memory of distributed, block, and high-speed access to external RAM via the SSTL3 standard, phase locked loops (PLL), Xilinx Select I/O pins, Xilinx segmented-routing and vector based interconnect

Look-up Table

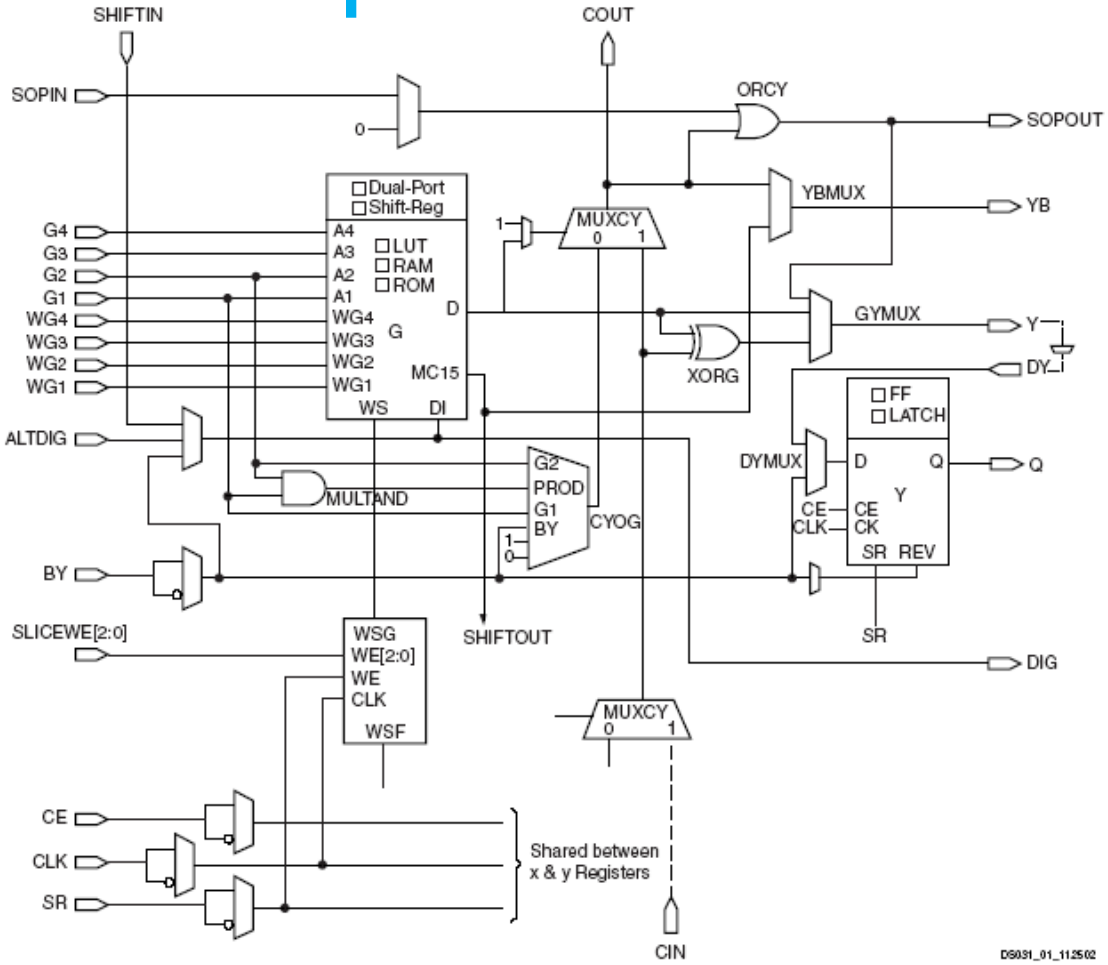
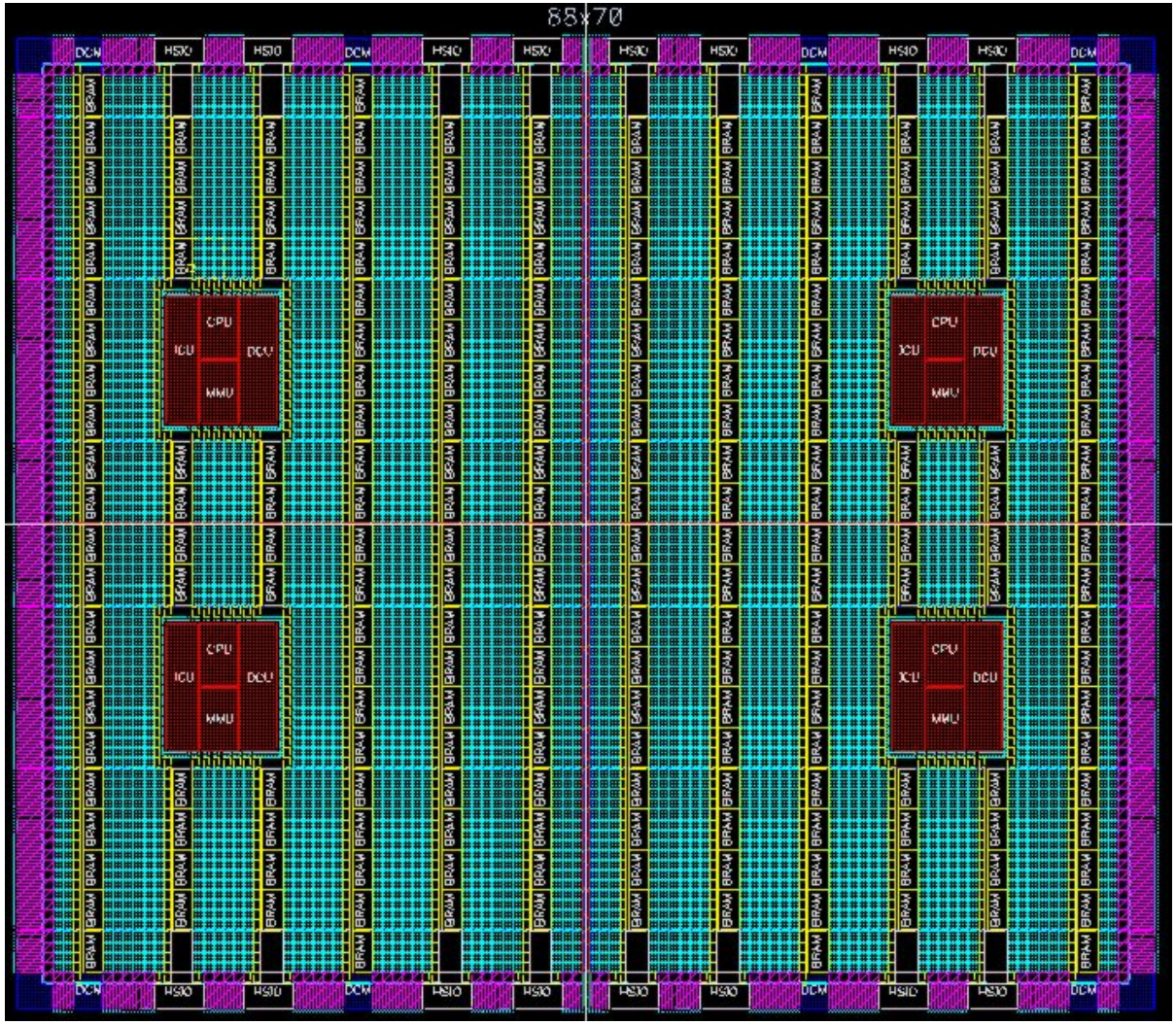
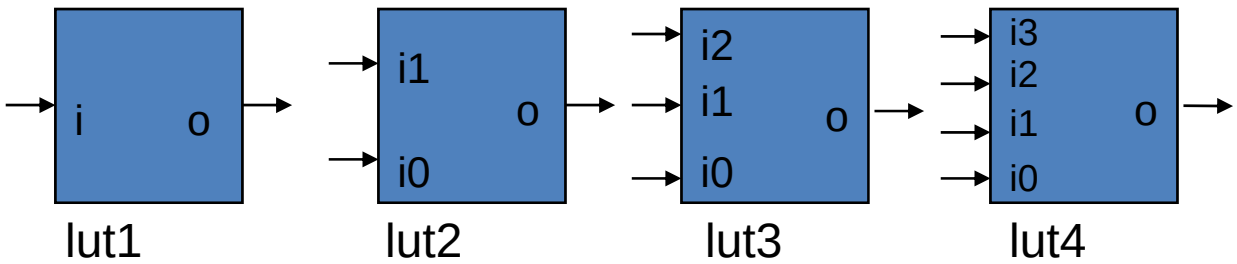


Figure 16: **Virtex-II Slice (Top Half)**


88x70
DCM
HSIO
CPU
ICU
DCU
MMU
BRAM

LUTs as higher order functions

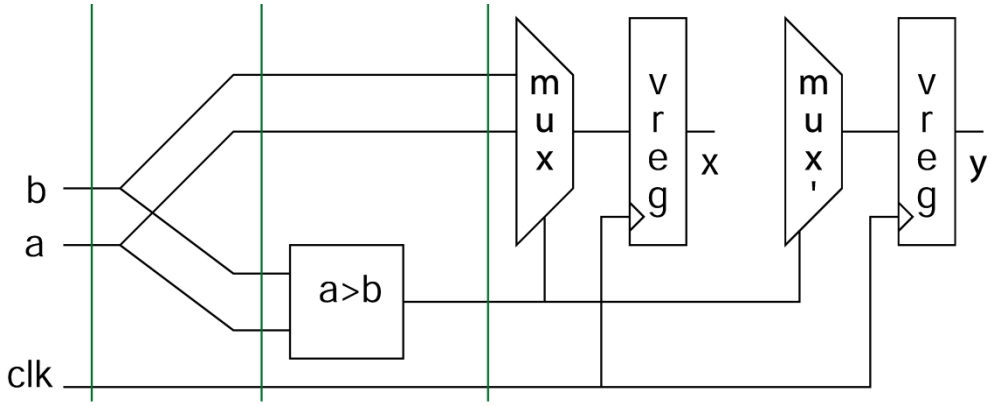


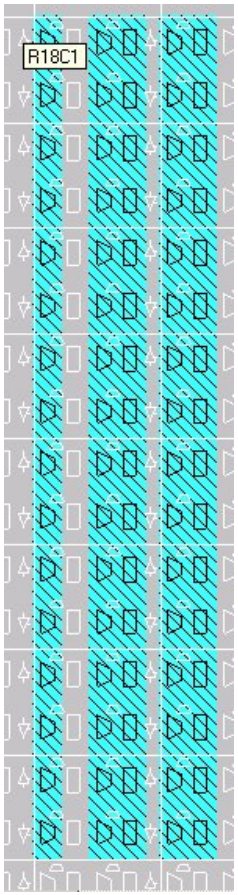
inv = **lut1** not

and2 = **lut2** (&&)

mux = **lut3** (λ s d0 d1 . **if** s **then** d1 **else** d0)

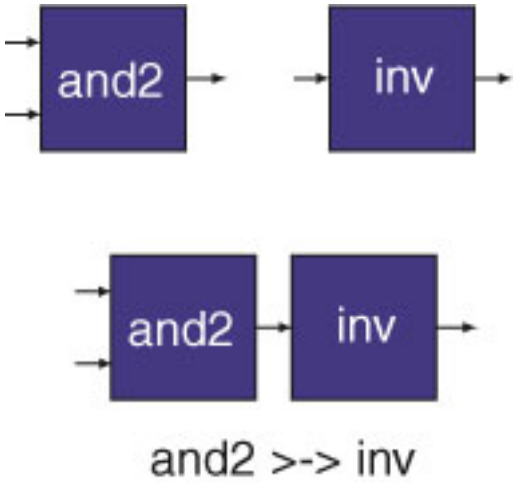
APL-style descriptions

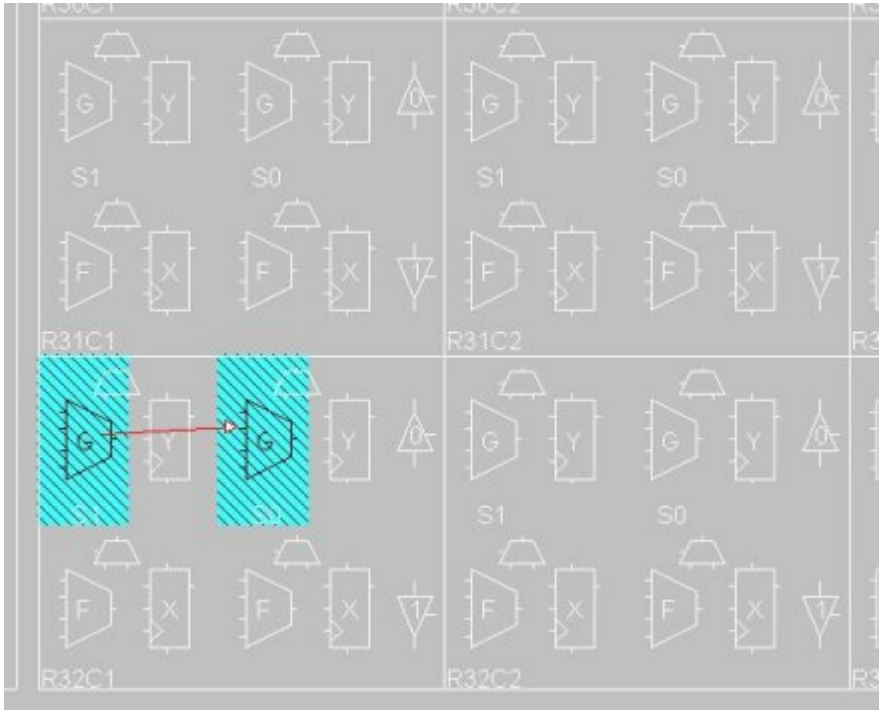




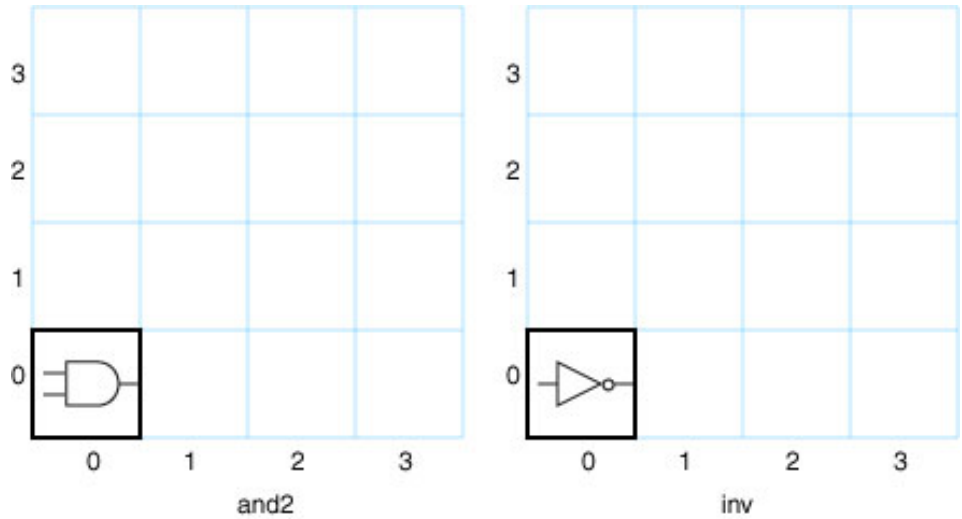
twoSorter clk = fork2 >=> fsT comparator >-> condSwap clk

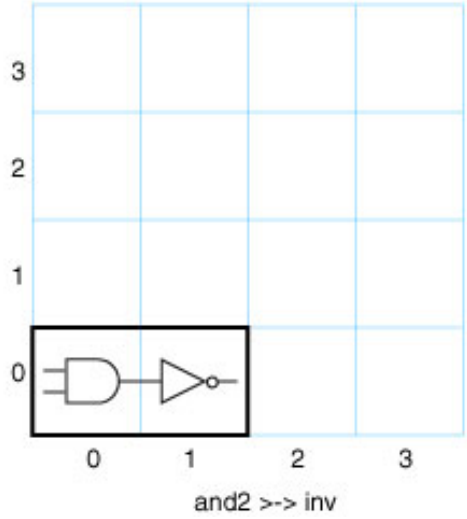
Serial Composition



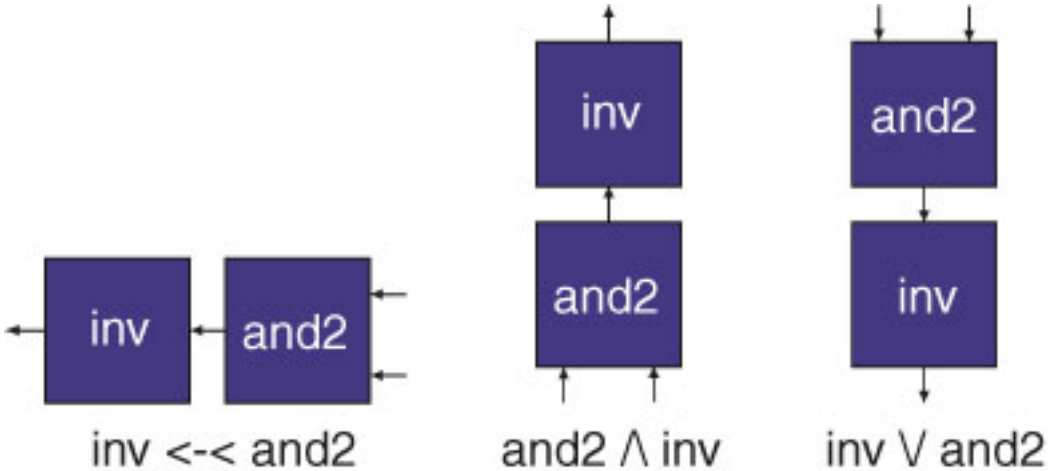


Functional Geometry

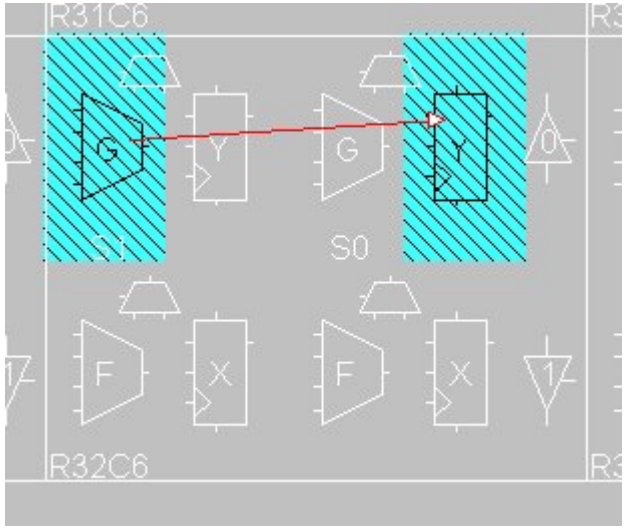




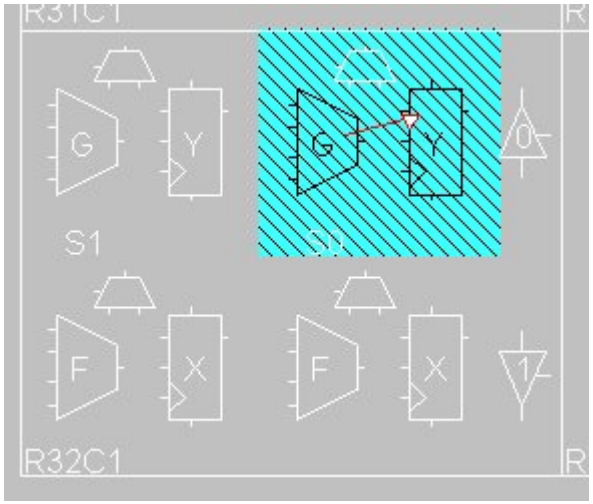
Behavior and Layout Composition



WYSIWYG

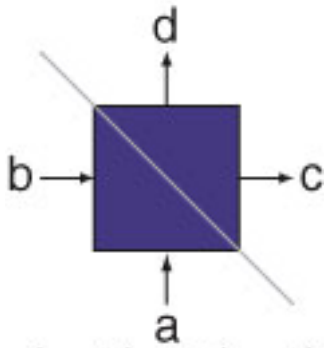


and2 >-> fd clk

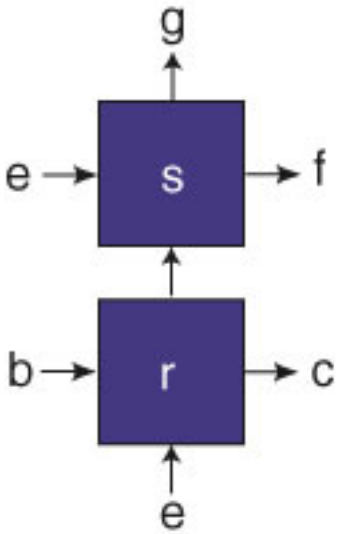


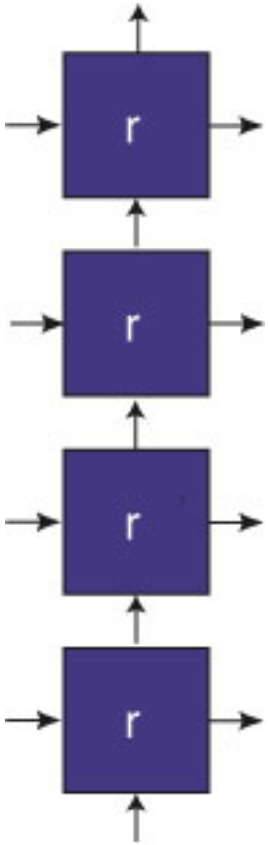
and2 >|> fd clk

Four sided tiles



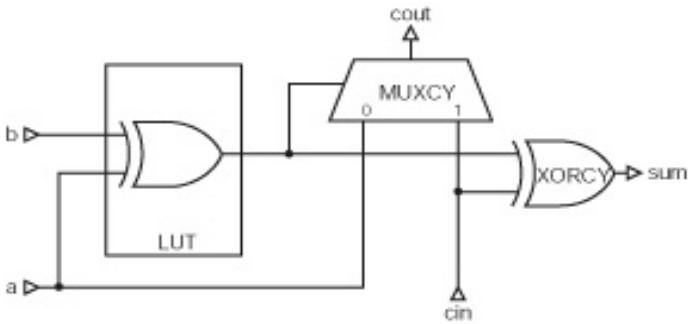
(a, b) -> (c, d)

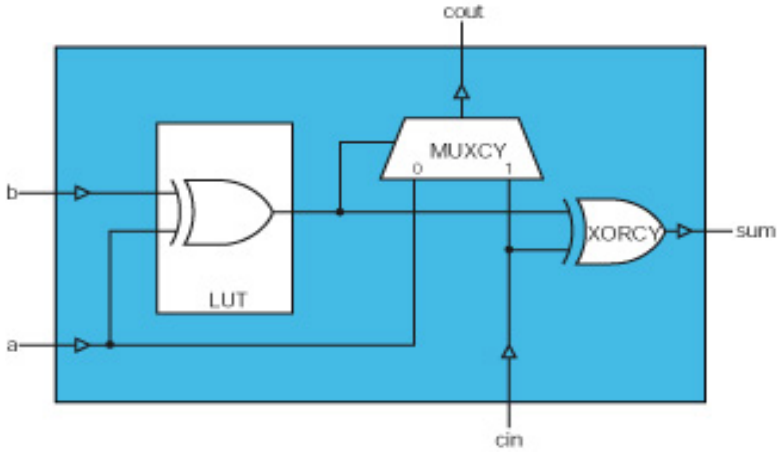


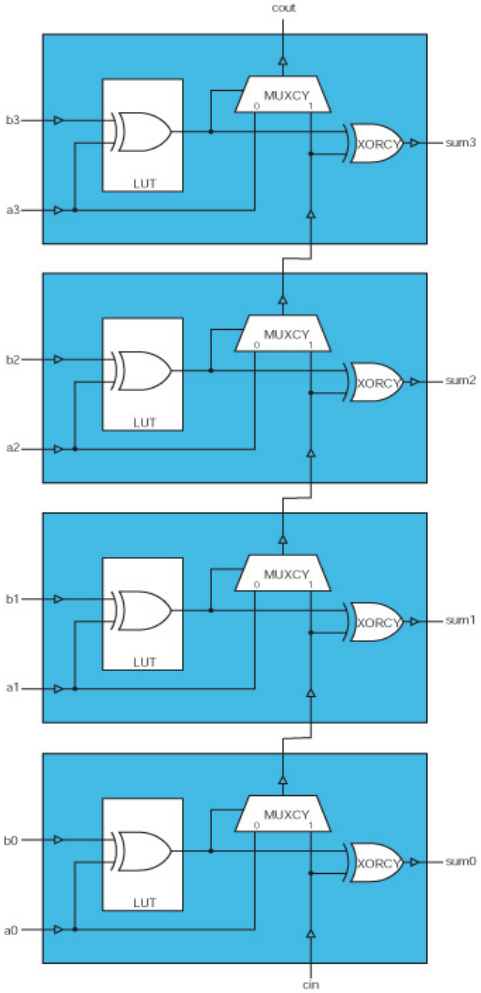


col 4 r

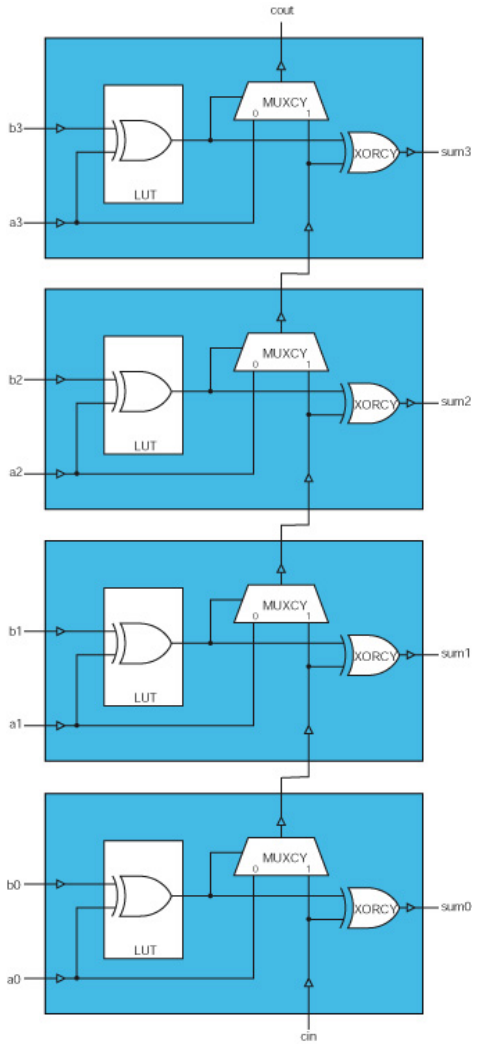
4-bit adder

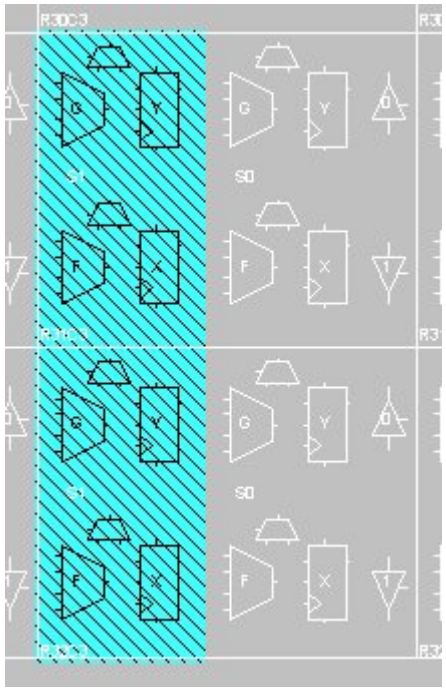

cout
MUXCY
0
1
b
LUT
XORCY
sum
a
cin


cout
MUXCY
0
1
b
LUT
XORCY
sum
a
cin

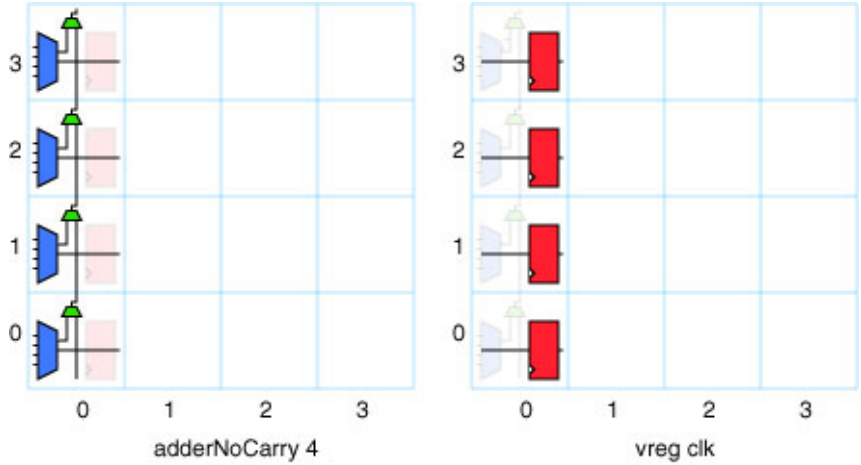

cout
b3
a3
LUT
MUXCY
XORCY
sum3
b2
a2
sum2
b1
a1
sum1
b0
a0
sum0
cin

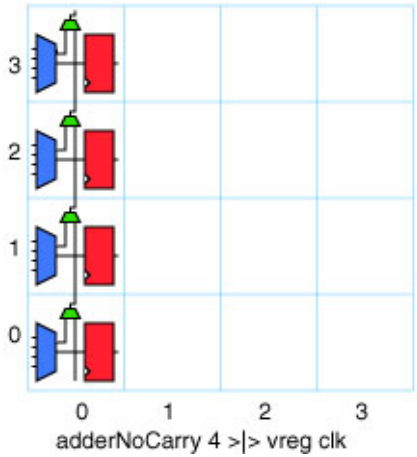
Adder Layout

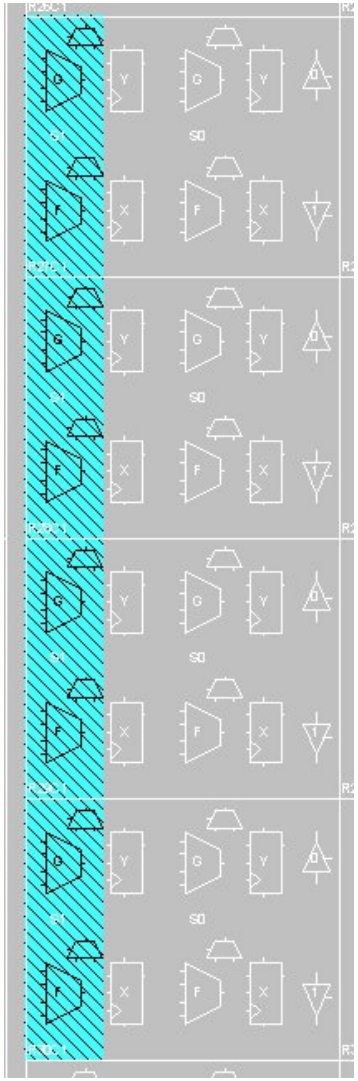




Registered Adder



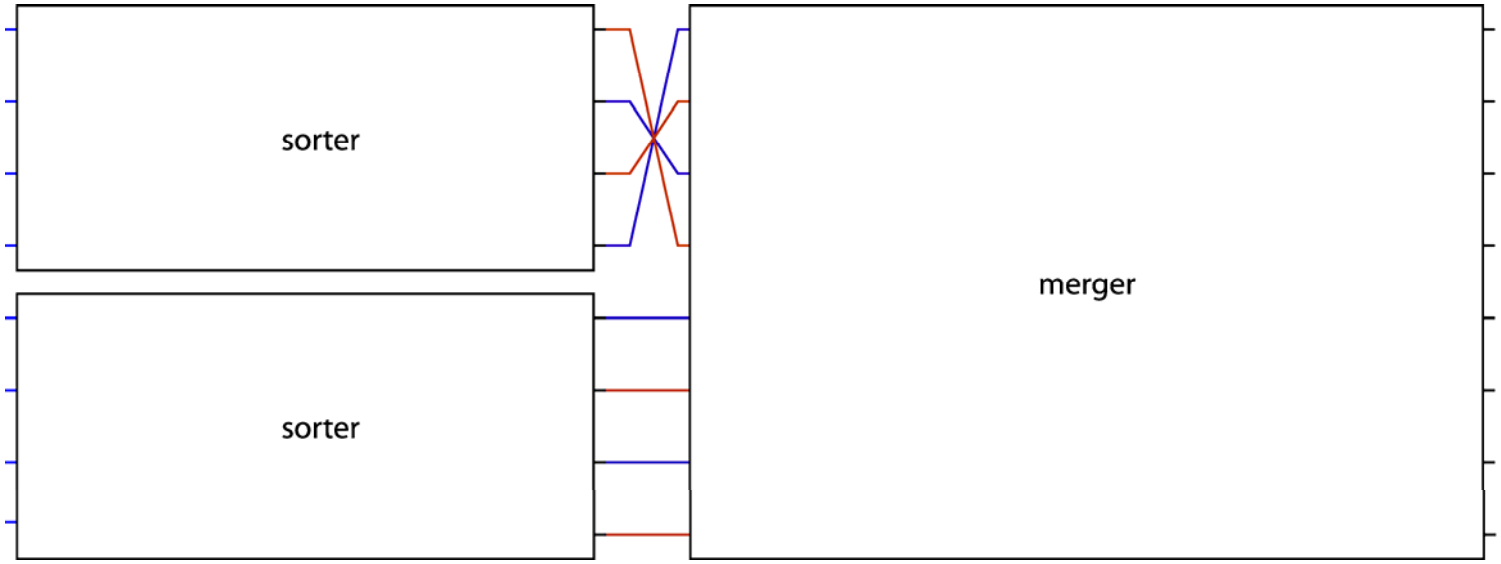




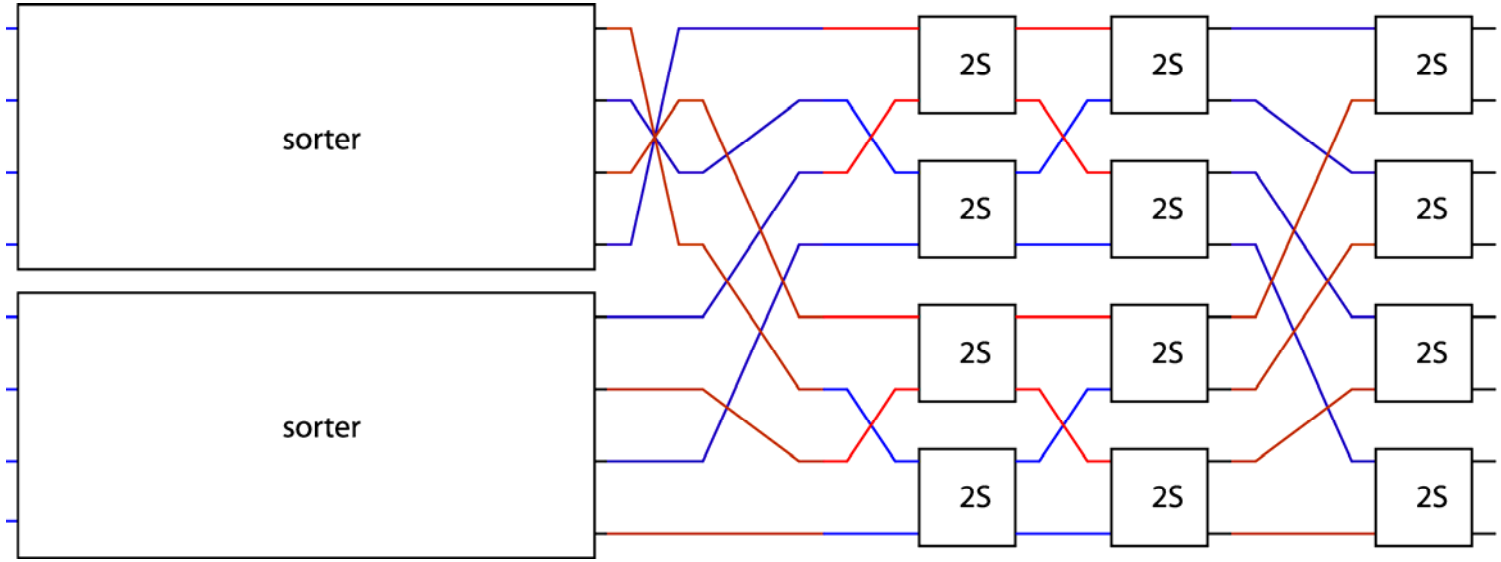
Example: A Parallel Sorter

- Batcher's parallel merger/sorter
- Carefully design and hand-layout one core element: two-sorter
- Use higher order combinantors to recursively combine this core element to produce a larger sorter
- Results in unprecedeneted layout compactness and excellent performance
- Beautiful description

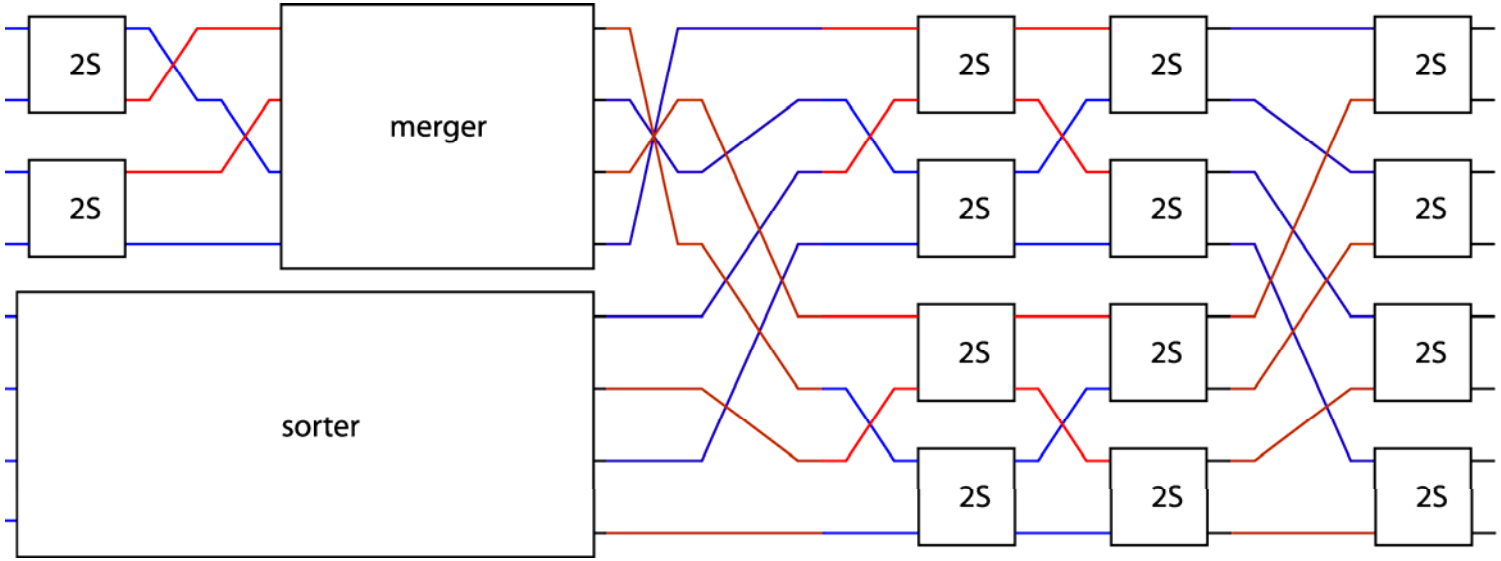
Batcher's Bitonic Sorter



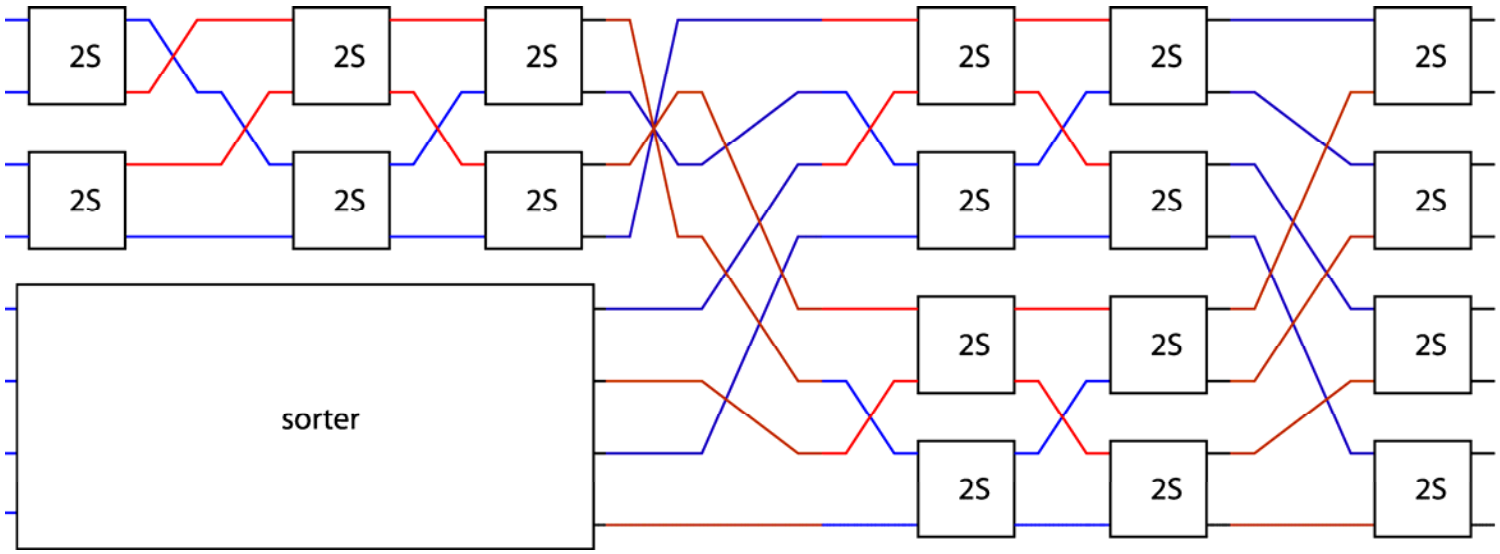
Batcher's Bitonic Sorter



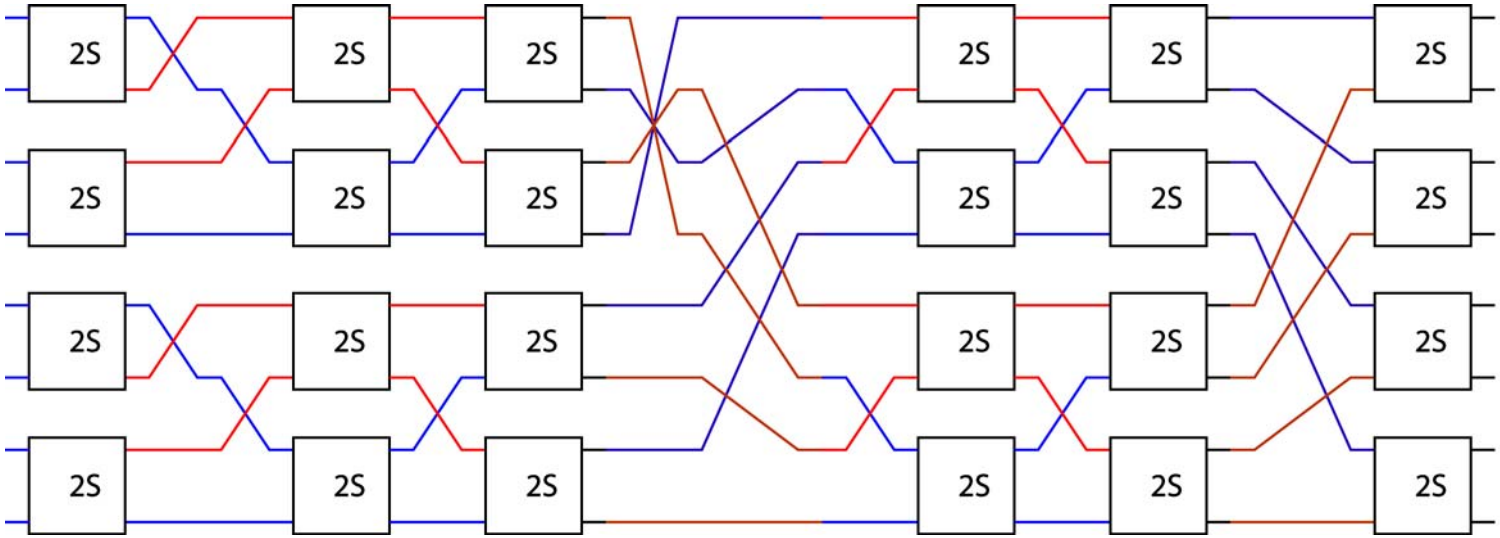
Batcher's Bitonic Sorter



Batcher's Bitonic Sorter



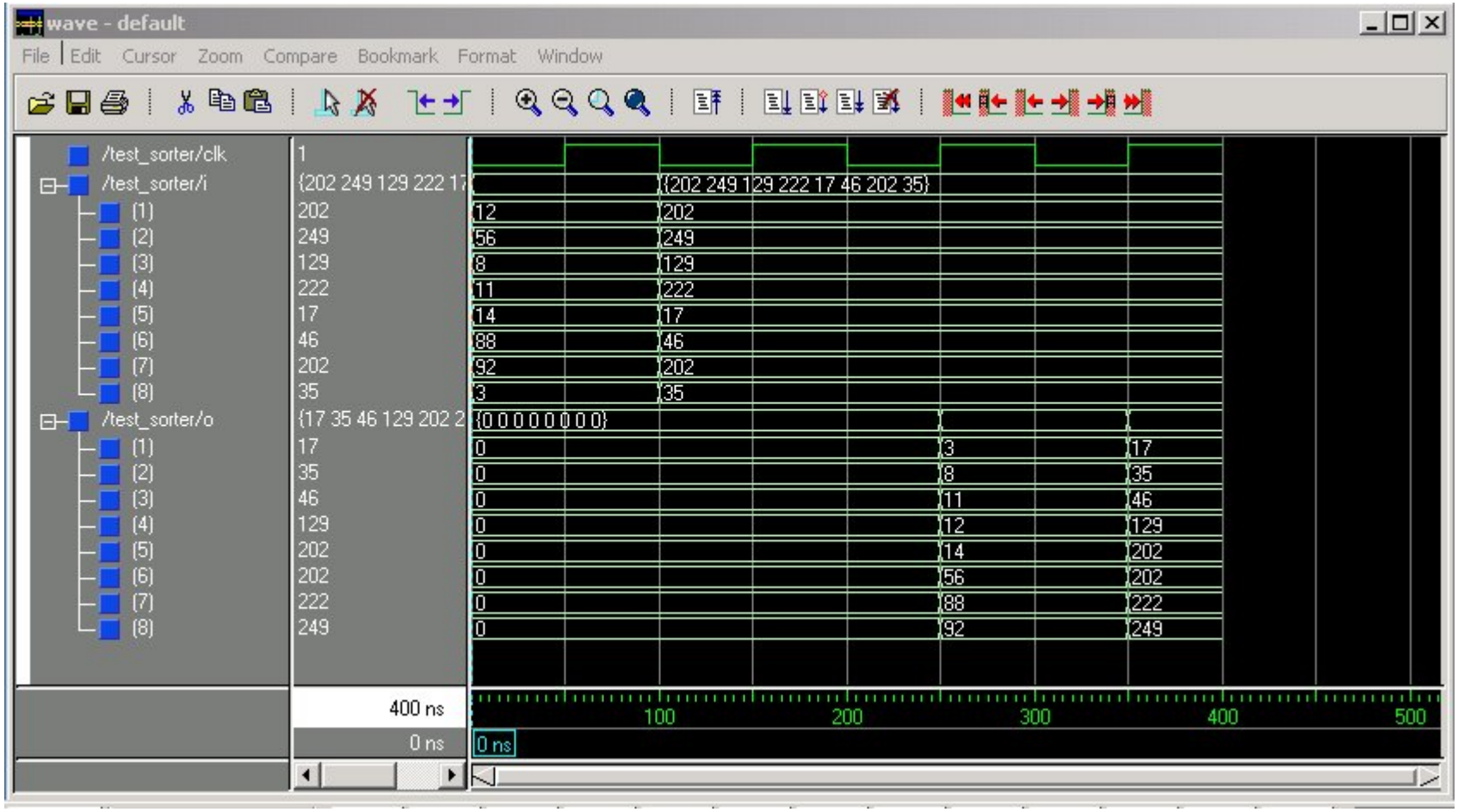
Batcher's Bitonic Sorter



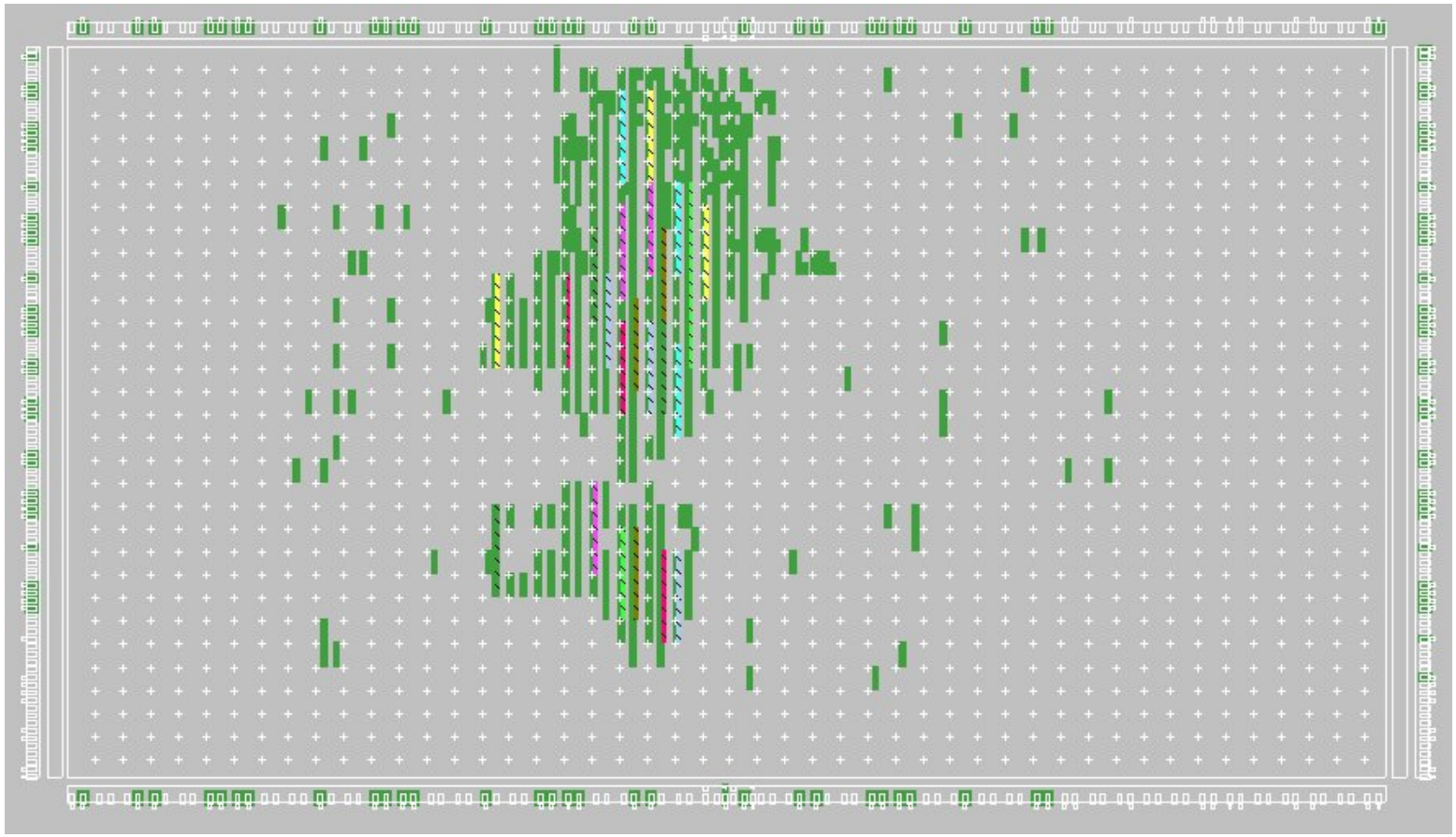
Batcher's Bitonic Sorter:VHDL

```
function sorter (i : in num_array) return num_array is
    constant halfi : positive := i'length / 2 ;
  begin
    if i'length = 2 then
      return two_sorter (i) ;
    else
      midsort(1 to halfi) := sorter (i(1 to halfi)) ;
      midsort(halfi+1 to i'length)
        := reverse (sorter (i(halfi+1 to i'length))) ;
      return butterfly (midsort) ;
    end if ;
  end function sorter ;
```

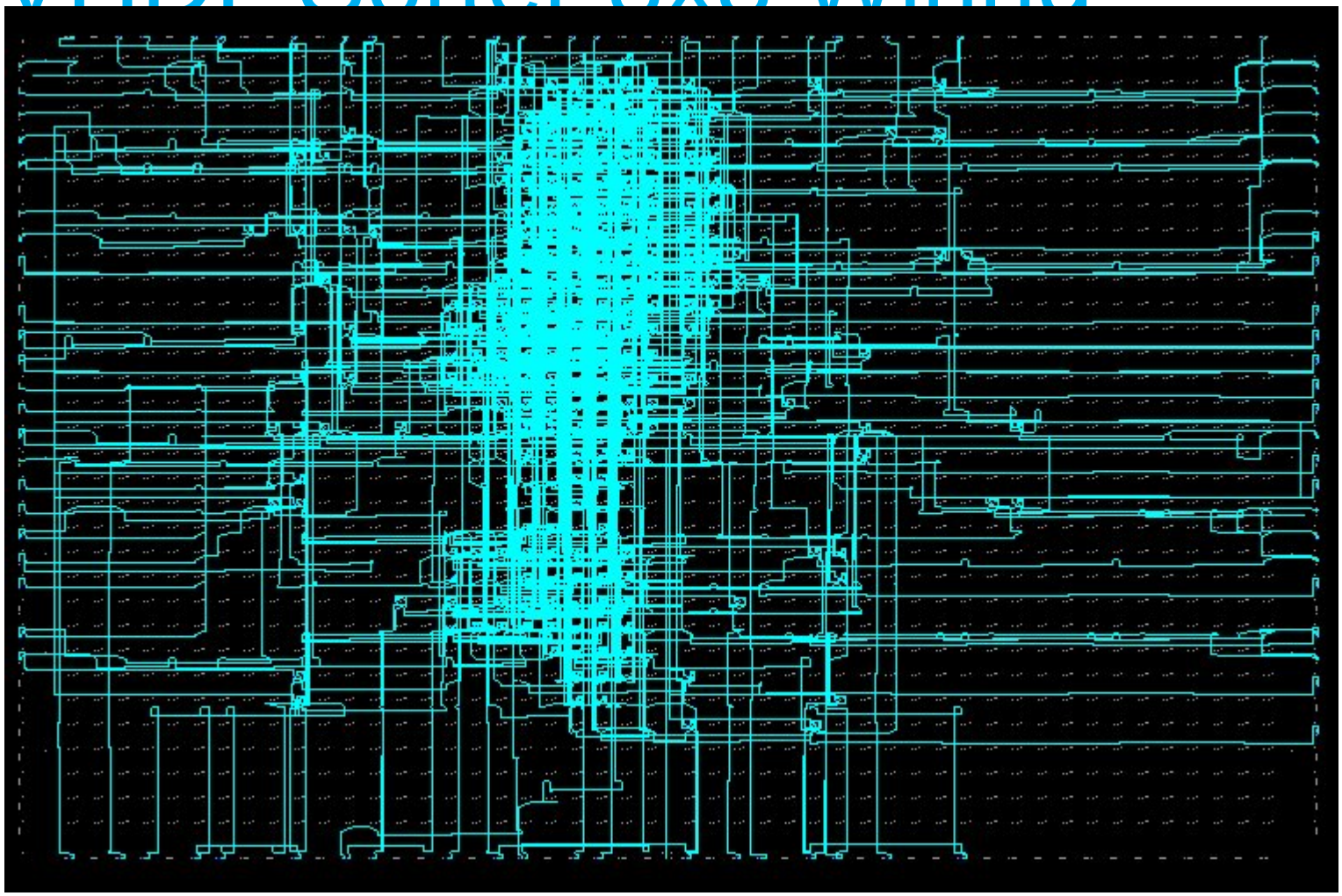
Simulation


wave - default
File Edit Cursor Zoom Compare Bookmark Format Window
/test_sorter/clk
/test_sorter/i
(1) (2) (3) (4) (5) (6) (7) (8)
/test_sorter/o
(1) (2) (3) (4) (5) (6) (7) (8)
1
{202 249 129 222 17
202 249 129 222 17 46 202 35
{17 35 46 129 202 2
{0 0 0 0 0 0 0 0}
400 ns
0 ns
100 200 300 400 500

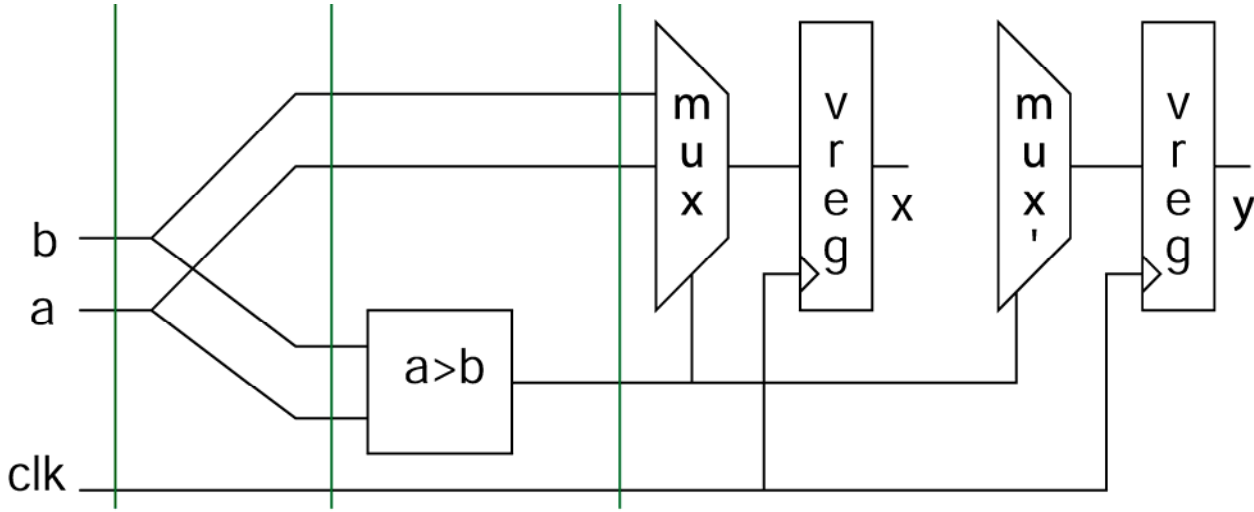
VHDL sorter 8x8 Floorplan



VHDL Sorter 8x8 Wiring

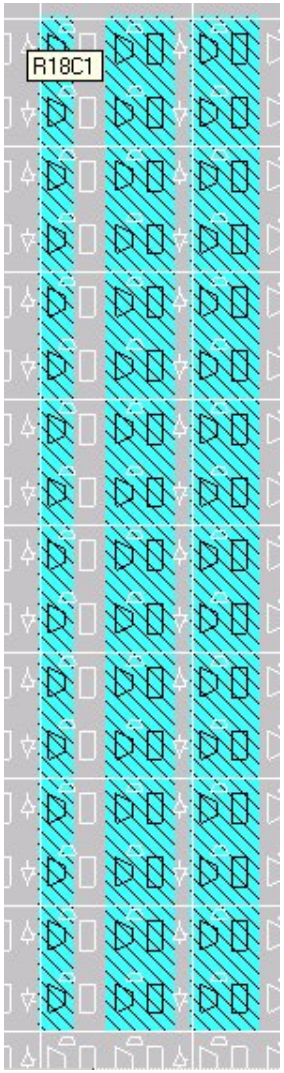


two sorter

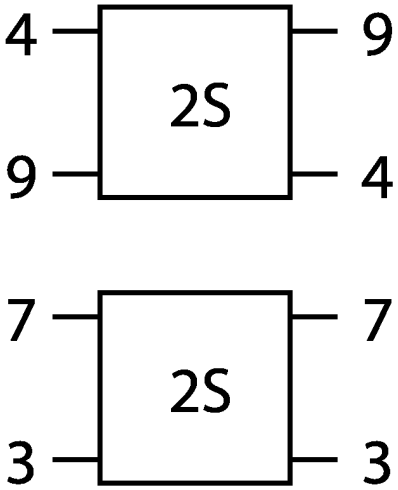


twoSorter clk = fork2 >=> fsT comparator >-> condSwap clk

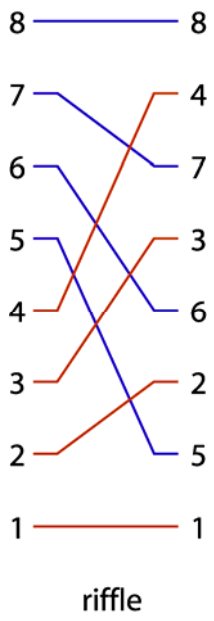
Two-sorter layout



Two Sorter

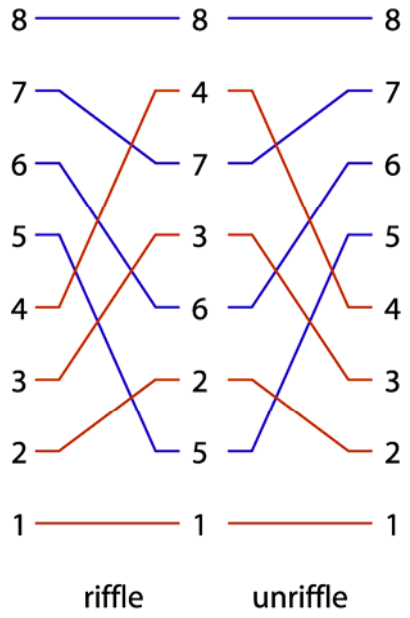


riffle



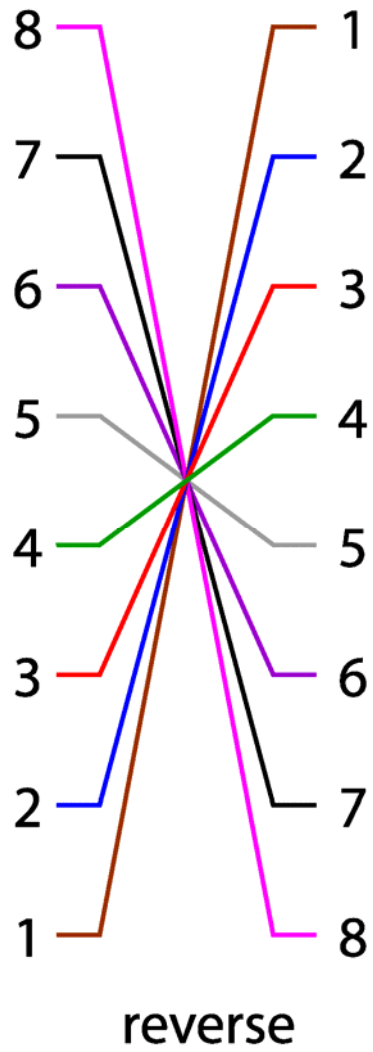
riffle = halve >-> ziP -> unpair

unriffle

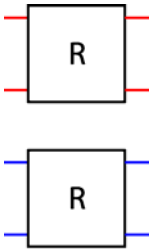


unriffle = pair >-> unzip >-> unhalve

reverse



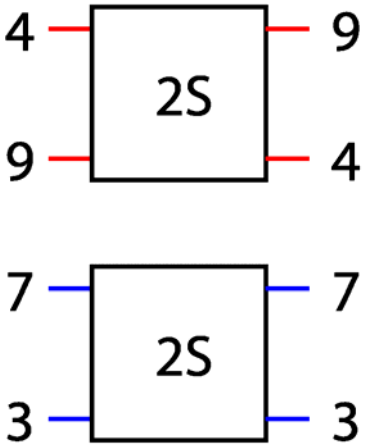
two R



two R

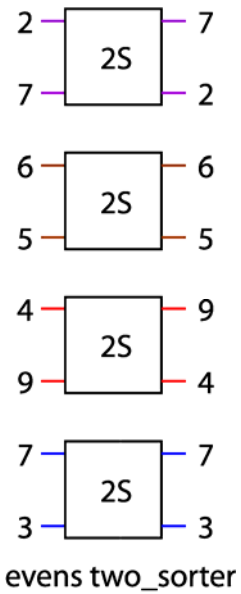
two r = halve >-> par [r,r] >-> unhalve

two two_sorter



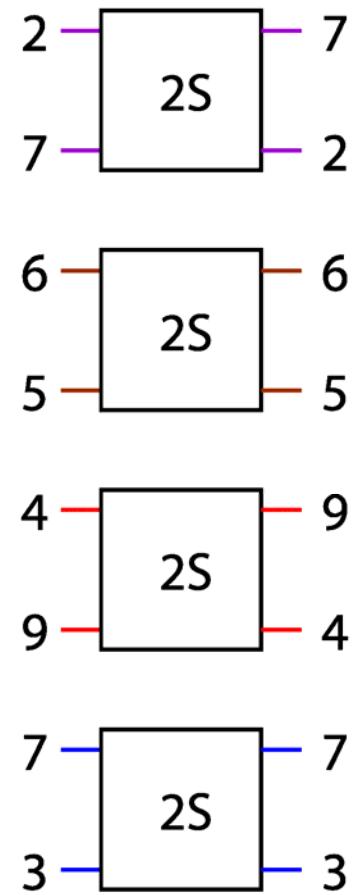
two two_sorter

evens R



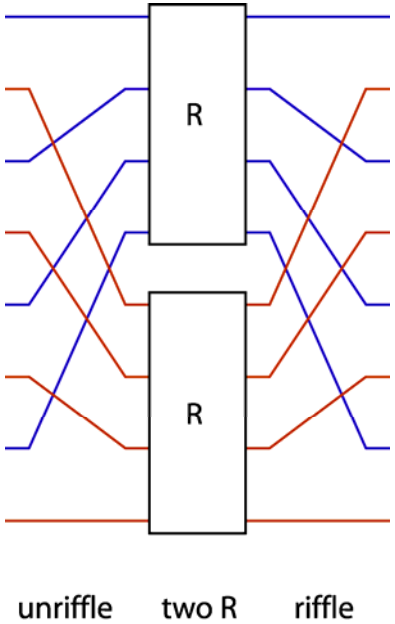
evens f = chop 2 >=> maP f >=> concat

evens two_sorter



evens two_sorter

ilv R



ilv r = unriffle >=> two r >=> riffle

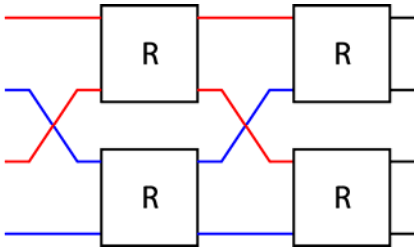
Butterflies

```
bfly R 1 = R
bfly R n = ilv (bfly R (n-1)) >-> evens R
```

bfly R 1

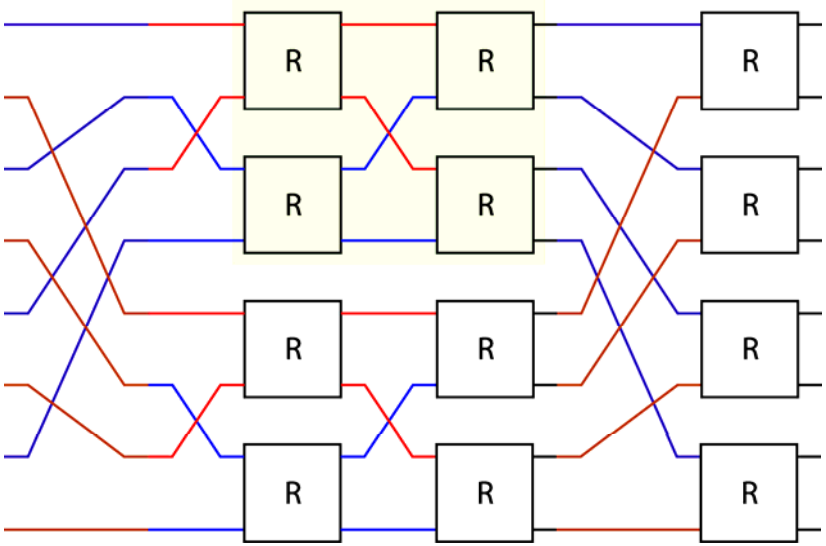
bfly R 1 = R

bfly R 2



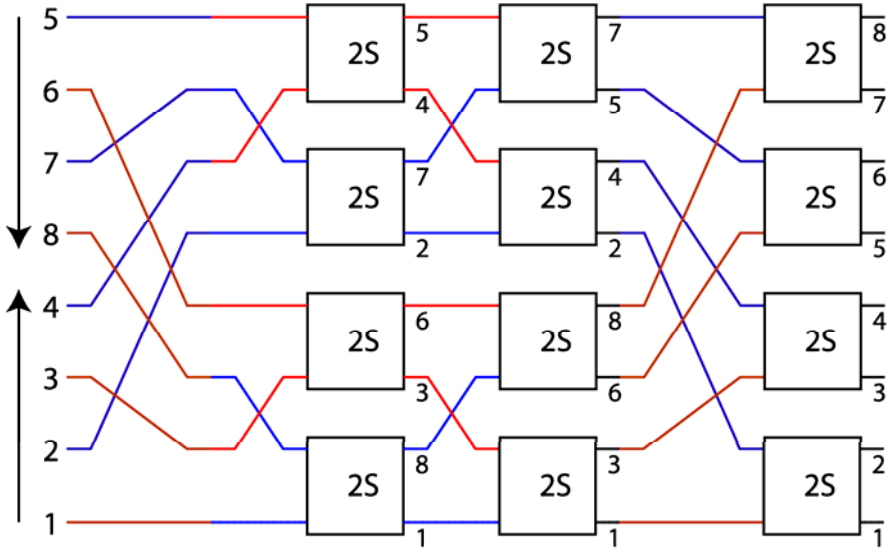
bfly R 2 = ilv (bfly R 1)) >-> evens R
= ilv R >-> evens R

bfly 3 R



bfly R 3 = ilv (bfly R 2)) >-> evens R
= ilv (ilv R >-> evens R) >-> evens R

bfly 3 two_sorter



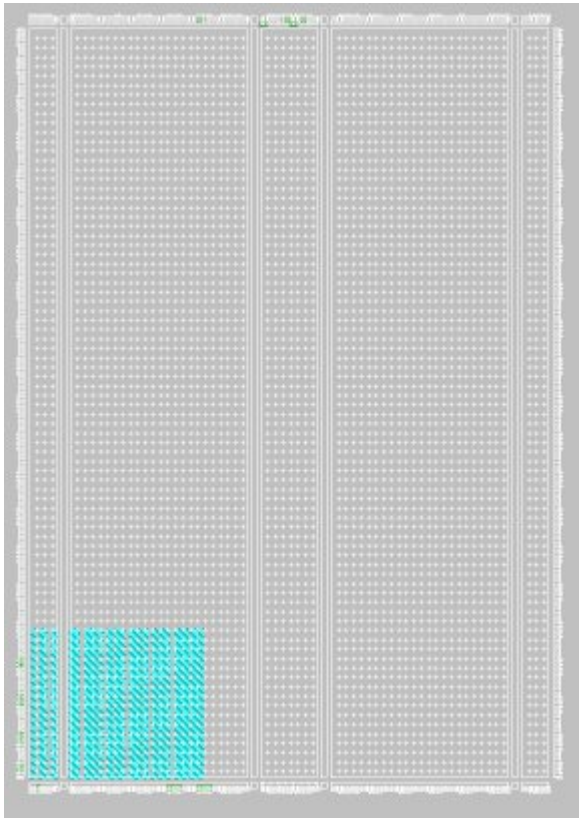
butterfly two_sorter

```
function butterfly (i : in num_array)
                        return num_array is
 begin
  if i'length = 2 then
    return two_sorter (i) ;
  else
    return (evens_two_sorter (ilv_butterfly (i))) ;
  end if ;
 end function butterfly ;
```

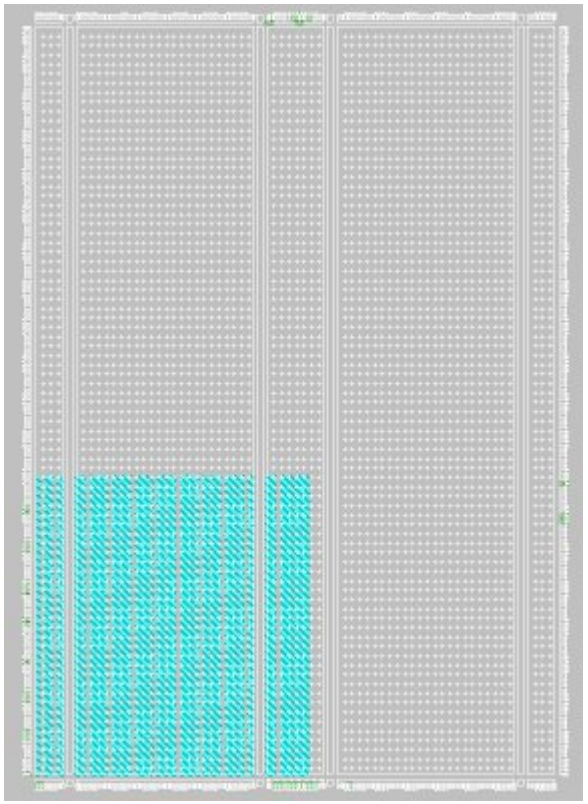
Batcher's Bitionic Sorter

```
sortB cmp 1 = cmp
sorB cmp n
  = two (sortB cmp (n-1)) >->
    pair >-> snD reverse >-> unpair >->
    butterfly cmp n
```

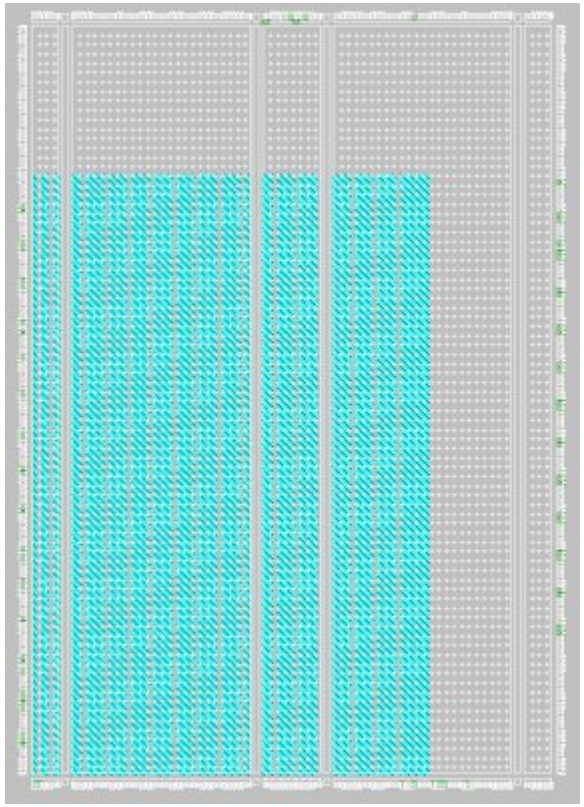
sorter twoSorter 3



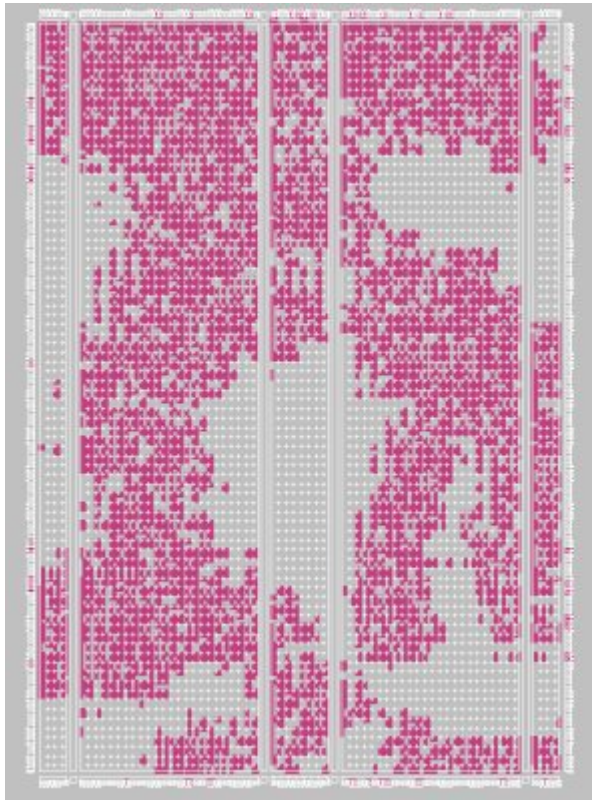
sorter twoSorter 4



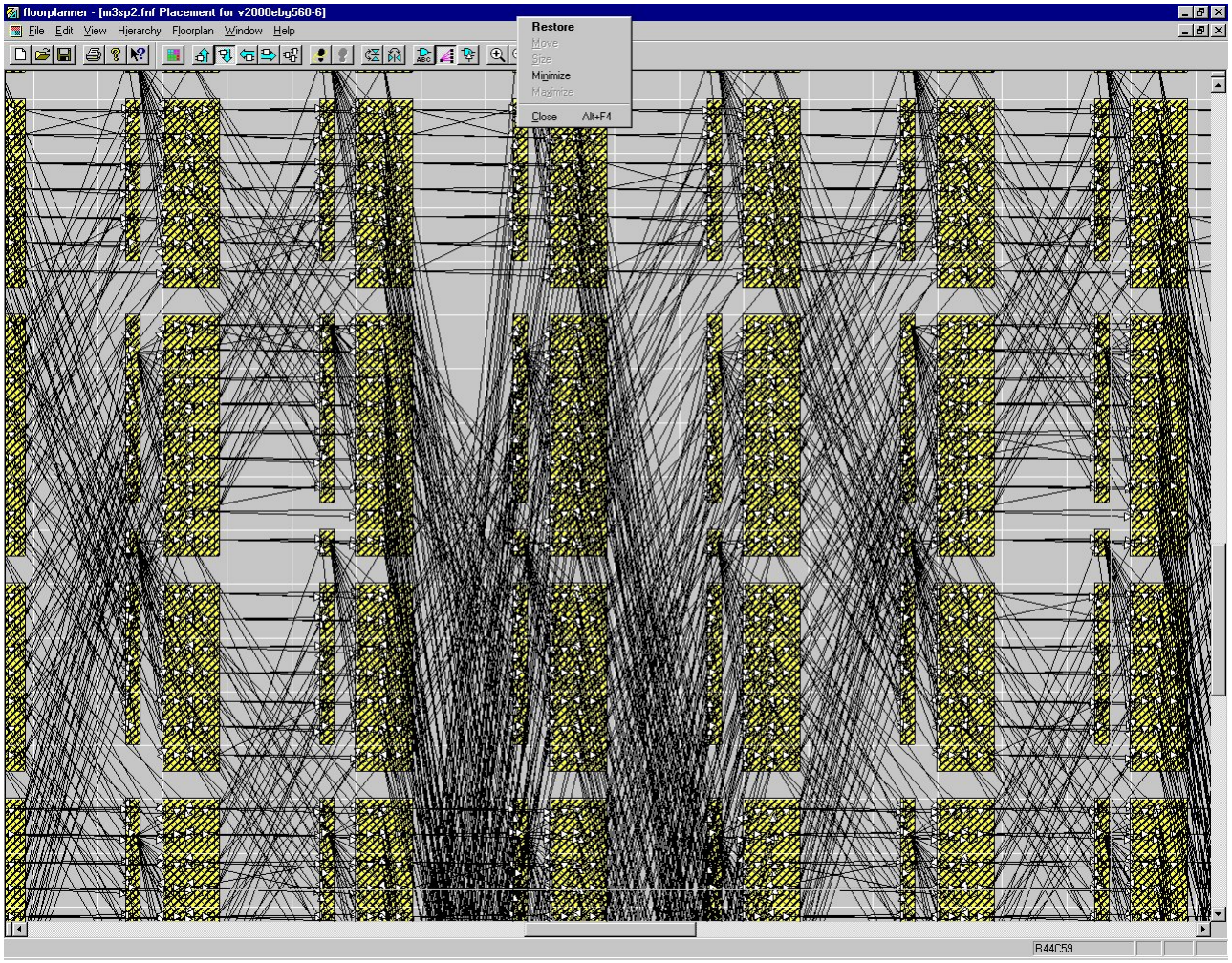
sorter twoSorter 5



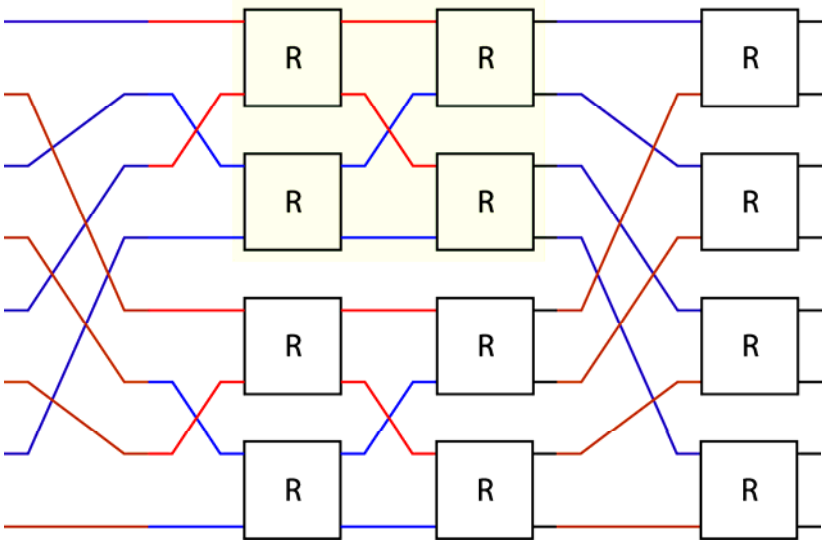
grep –v RLOC



FPGA Implementation

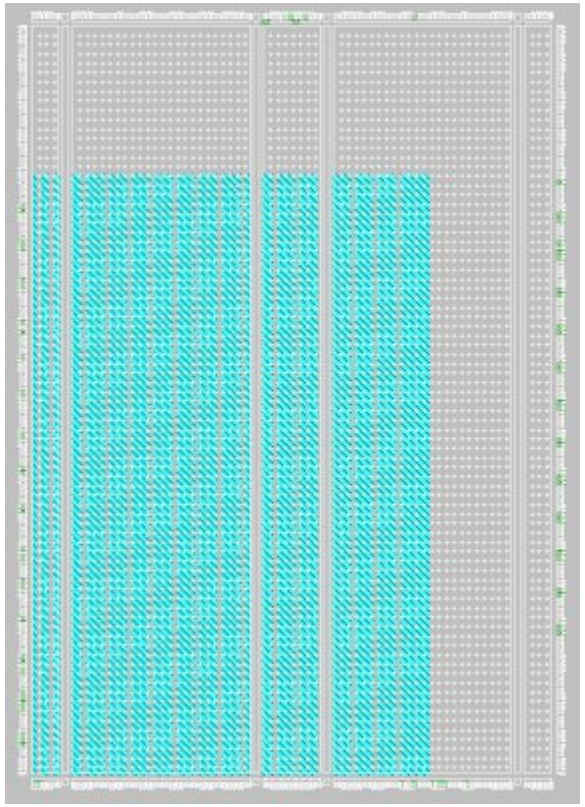


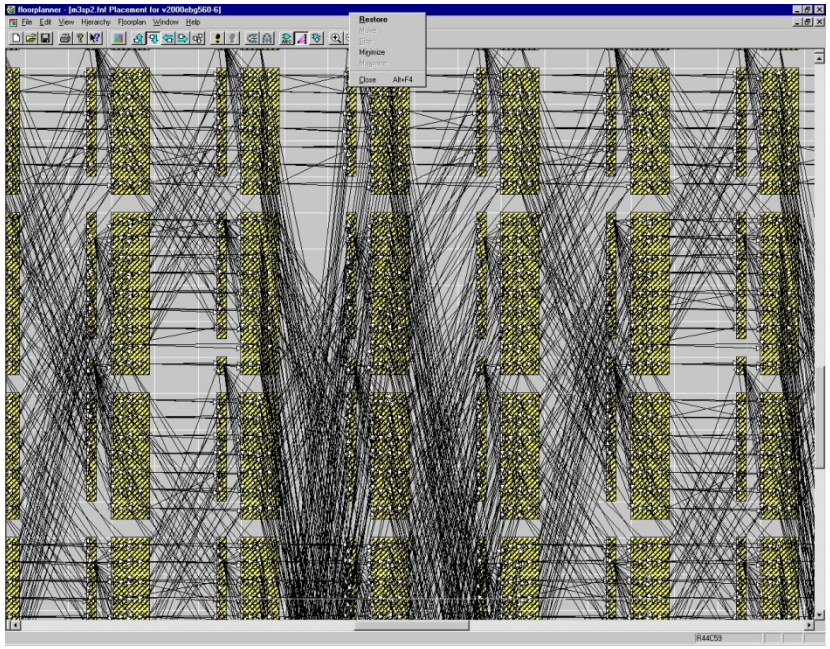
Lava



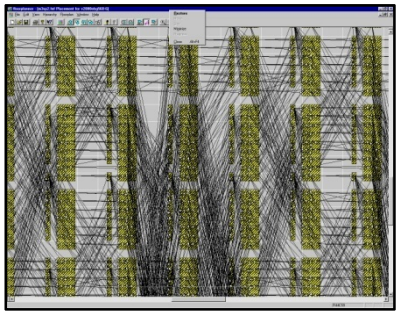
bfly R 3 = ilv (bfly R 2)) >-> evens R
= ilv (ilv R >-> evens R) >-> evens R

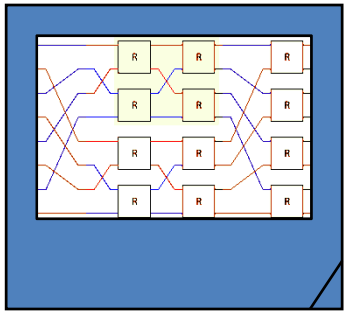
Lava Sorters





FPGA
hardware
(VHDL)



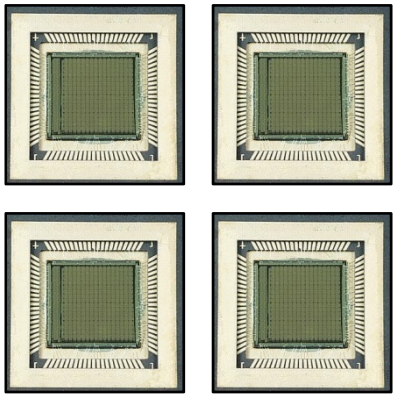


GPU code (Accelerator)

data parallel
description of
FFT-style operations
in an E-DSL

C#

CCR, Cw or
System.Threading



Multi-cores

Lava Embedded in F#

```
let rec bfly r n =
  match n with
  1 -> r
  | n -> ilv (bfly r (n-1)) >-> evens r
```

Lava Embedded in C#

```
public delegate List<T> ButterflyElement <T> (List<T> v);

public static List<T> bfly<T>(Func<List<T>,List<T>> r, List<T> l)
{  if (l.Count == 2)
      return r(l);
   else
      return evens(r, ilv<T,T>(i => bfly(r, i), l)) ;
}
```

Summary

- Future architectures will be heterogeneous.
- Domain specific languages will be essential tools for developing manycore software.
- Functional languages provide the right abstractions to build powerful embedded domain specific languages.