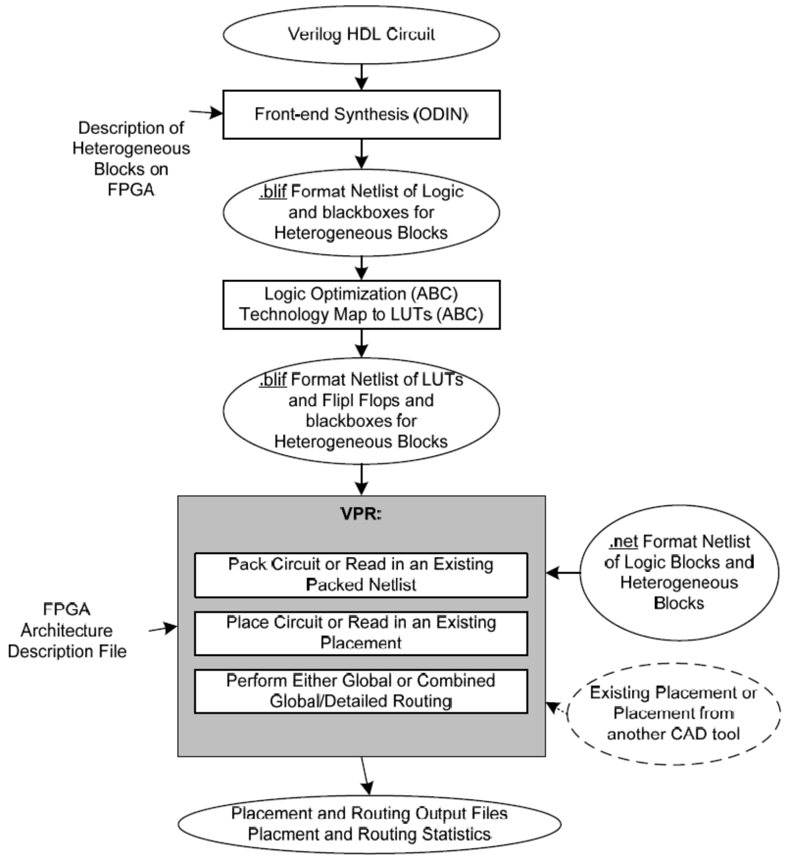
Partitioning Algorithm and Initial Results

Workflow

1. Synthesis
2. Optimisation
3. Partitioning
   1. Split
   2. Triplicate
   3. Join
   4. Flatten
4. Packing
5. Placing
6. Routing



Verilog HDL Circuit
Front-end Synthesis (ODIN)
Description of Heterogeneous Blocks on FPGA
.blif Format Netlist of Logic and blackboxes for Heterogeneous Blocks
Logic Optimization (ABC) Technology Map to LUTs (ABC)
.blif Format Netlist of LUTs and Flipl Flops and blackboxes for Heterogeneous Blocks
VPR:
Pack Circuit or Read in an Existing Packed Netlist
Place Circuit or Read in an Existing Placement
Perform Either Global or Combined Global/Detailed Routing
FPGA Architecture Description File
.net Format Netlist of Logic Blocks and Heterogeneous Blocks
Existing Placement or Placement from another CAD tool
Placement and Routing Output Files Placment and Routing Statistics

CAD Design Flow
(VPR Manual)

Blif Format

- Text format.
- List of elements/nodes. Latches and Combinational Logic.
- Each node has text describing inputs, outputs, element.
- E.g (And Gate):

```
.names in1 in2 out
11 1
```

Algorithm

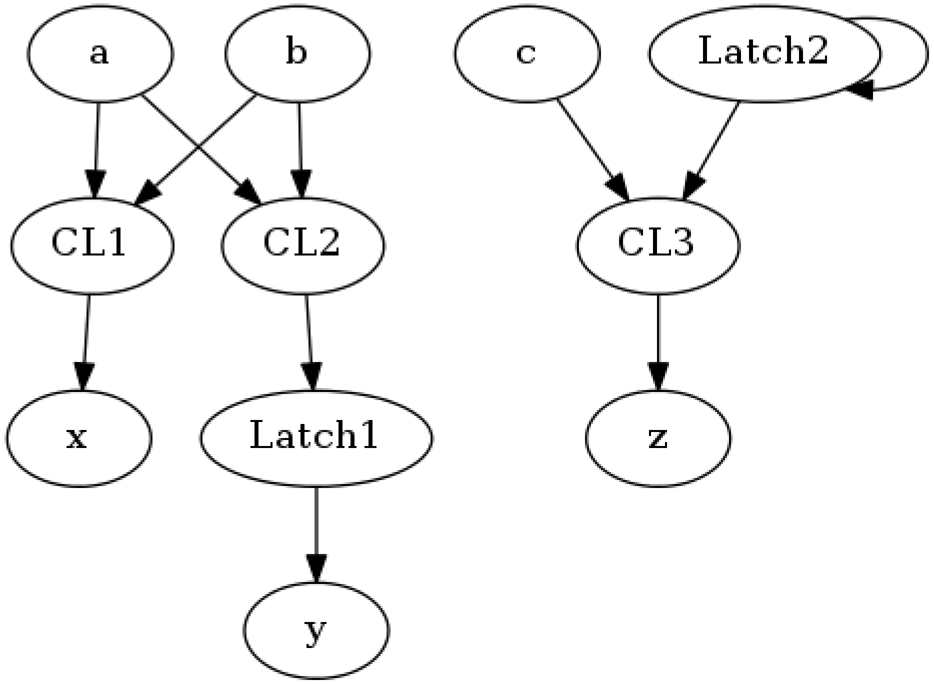
1. Read network into memory
2. Traverse breadth first from outputs
3. Add node to new partition until partition full
4. Write partition to file
5. Continue traversing each node once
6. Treat each partition as black box and triplicate it
7. Treat each TMR partition as black box and join them

Algorithm – Adding Node

1. Insert node into network and create signals
2. Recursively traverse network keeping track of visited nodes
3. If an already visited node is reached, cut link between node and parent

Example

```
.model main
.inputs a b c
.outputs x y z
.clocks pclk
.names a b x
11 1
.names a b [1]
10 1
01 1
.latch [1] y re pclk 2
.latch [2] [2] re pclk 2
.names c [2] z11 1
.end
```



Example – End Result

```
# Benchmark "main" written by ABC on Wed Mar 13 02:18:50 2013
.model main
.inputs a b c
.outputs x y z

.latch      n31 main|p1output(1)|[qq40]  2
.latch      n34 main|p1output(1)|[qq10]  2
.latch      n40 main|p1output(1)|[qq41]  2
.latch      n43 main|p1output(1)|[qq11]  2
.latch      n49 main|p1output(1)|[qq42]  2
.latch      n52 main|p1output(1)|[qq12]  2

.names a b n25
11 1
.names c main|p1output(1)|[qq10] n26
11 1
.names a b n31
10 1
01 1
.names a b n28
11 1
.names c main|p1output(1)|[qq11] n29
11 1
.names a b n40
10 1
01 1
.names a b n31_1
11 1
.names c main|p1output(1)|[qq12] n32_1
11 1
.names a b n49
10 1
01 1
.names n26 n29 n32_1 z
11- 1
1-1 1
--1 1
.names n25 n28 n31_1 x
11- 1
1-1 1
--1 1
.names main|p1output(1)|[qq40] main|p1output(1)|[qq41] \
 main|p1output(1)|[qq42] y
11- 1
1-1 1
--1 1
.names main|p1output(1)|[qq10] n34
1 1
.names main|p1output(1)|[qq11] n43
1 1
.names main|p1output(1)|[qq12] n52
1 1
.end
```

Algorithm

- 1. Read entire file into memory, and represent it as a graph.
- Each Model has a list of nodes and map of signalName->Signal
- Each signal points to its source and sinks
- Each node has a list (string) of inputs and outputs (+type, etc).
- Why do nodes have strings, which are looked up in a map to the signal, which points to the node? Why not have each node just point to the other node? Because of the way models are made and manipulated.

Algorithm 2

- 2. Traverse
- Start at one end, adding connected nodes to partition. Once partition reaches limit, write it out, remove those nodes from the network, and repeat.
- Specifically, start at outputs not inputs due to e.g. [IMAGE]
- 3. Adding to Partition
- Add to node collection
- Update Signals
- Recalculate critical path (max cost without cycles)

Algorithm 3

```
model = network->MainModel
FOREACH output in model
  q.Add(output->source)
partition = EmptyModel
WHILE node = q.pop
  IF visited(node)
    CONTINUE
  IF partition+node > limits
    WriteModelToNewFile(partition)
    partition = EmptyModel
  partition.Add(node)
  FOREACH input in node->inputs
    q.Add(input->source)
 WriteModelToNewFile(partition)

Model::Add(node)
  nodeCollection.Add(node)
  UpdateSignals(node)
  inCost = MaxInputCost
  UpdateCosts(node, inCost)

Model::UpdateCosts(node, inCost)
  nodeCost = inCost+InnateCost
  MarkVisited(node)
  FOREACH sink in node->sinks
    IF visited(sink)
      Cut(node, sink)
    ELSE
      UpdateCosts(sink, nodeCost)

//Rename the signals, then after TMR we can rejoin them outside the partition
Model::Cut(source, sink)
  source->output = "SpecialOut"+source->output
  sink->inputs[source->output] = "SpecialIn"source->output
  Model.AddInput("SpecialIn"source->output)
  Model.AddOutput("SpecialOut"+source->output)
```

Results

- Use very small partitions, to magnify effect.
- Routing is still the dominant contributor to time spent in workflow.
- Latency from 10%-80% increase.
- Area, number of elements, etc, all more or less triple.
- Number of nets on the critical path usually increases, sometimes decreases.