Variable	Type	Description
$input$	**file**	Input blif file
$targetRecoveryTime$	**float**	Per partition recovery time (in seconds)
$files$	**list of files**	circuit partitions, one per file
$file$	**file**	
$header$	**string**	string containing the first three lines of the input file
$output$	**file**	output file

varMain

Table 1: Variables for Main

Data Structures: == General structure: Blif represents a BLIF file, contains 1+ Models Model represents a BLIF model, or circuit, as a DFG. Each node is a BlifNode. Each BlifNode has the names of its inputs and outputs stored as strings. The model contains a map from string-¿Signal. Each Signal has a pointer to source and sinks.

signals[node-¿output]-¿sinks are the children of a node FOREACH(signal) in inputs signals[signal]-¿source provides the ancestors of each node. A Model also contains the primary input and output signals.

Storing signals as names, and providing a circuit specific map allows for nodes to be copied directly across with redirecting pointers at each step. Additionally, it means cutting loops is a simple case of string replacement, as the list of signals is dynamically generated from the set of nodes.

Model: Represents a BLIF Model i.e. a circuit or subcircuit within a BLIF file as a DFG. Fields: set(BlifNode) nodes - The set of all circuit elements map(string-¿Signal) - a map from signal name to signal string name - The name of this model list(Signal) inputs - The list of primary inputs for this circuit list(Signal) outputs - The list of primary outputs for this circuit

Methods: CutLoops() - described at ... AddNode(BlifNode) - adds a BlifNode to the current circuit. Doesn't create necessary signals. MakeSignalList() - Creates all appropriate signals MakeIOList() - Promotes appropriate signals to primary innputs or outputs

BlifNode: Represents a circuit element, or node within the DFG representing our circuit List(string) inputs - names of input signals string output - name of output signal string clock - name of clock signal (if applicable) string type - ".latch" or ".names" for latch or LUT string contents - body of node element. Type dependent text data. unsigned long id - Unique id unsigned cost - 0 for LUT, 1 for latch. Number of clock cycles added to critical path by this node. Methods: MakeNode(string type, list(string) params) - Creates a BlifNode from the text data in a BLIF file AddContents(string line) - Adds the provided line to the node body GetText() - Returns a textual representation for the node, suitable to be inserted into a BLIF file Clone() - Creates a clone of this node

Signal: Represents a signal between nodes, or a set of edges with common source from a node in the DFG. Fields: string name - name of the signal BlifNode source - node which drives this signal List(BlifNode) sinks - nodes which this signal drives

Blif: Represents a BLIF file, providing helper methods to read a file into a model. Fields: Map(string-¿Model) models - Map from model name to model List(string) masterInputs - Primary inputs for the master circuit List(string) masterOutputs - Primary inputs for the master circuit Model main - Master circuit model Methods: Blif(string path) - Constructor, create Blif object from path to a BLIF file Write(string path, Model model) - Writes the specified model to a BLIF file

Algorithm: == Syntax === variable/object - begins with lower case Function/Method/Procedure - begins with upper case Class.Method - Static method class-¿Method - instance method We're given a blif file as input. In line 11 we partition the input circuit into a number of sub circuits, each in a separate

main

Algorithm 1 Main Algorithm

1: **procedure** MAIN($input$, $targetRecoveryTime$)
2: $files \leftarrow$ Partition($input$)
3: **for all** $file \in files$ **do**
4: $file \leftarrow$ Triplicate($file$)
5: **end for**
6: $header \leftarrow input.lines[0 \rightarrow 3]$
7: $file \leftarrow$ Join($files, header$)
8: $output \leftarrow$ Flatten($output$)
9: **end procedure**

file. Then in lines 12-13 we iterate over all the partitions, and transform them into a triplicated partition with three copies and a voter circuit. Then in line 14 we extract the original header, which provides the name, inputs and outputs of the original circuit. We then, in line 15, join all the partitions together with the original name, inputs and outputs (in the same order), as the original circuit.

Lines 2-6 are setting up our variables with initial values. We read a blif file in to memory, where it is represented as a DFG with a number of properties as described in Reference. In line 3, circuit refers to the main circuit of a blif file. As we only support non-heirarchical blif files, this will always be the only circuit. In lines 7-8 we push our outputs onto the queue, to start traversing. Line 11 pops the node from the front of the queue. Next, in lines 12-15 we check if this node is already marked as visited. If so, we skip it as we only add each node to exactly one partition. Otherwise, we mark it as visited and proceed to partition it. In lines 16-17 we add the node to the current partition, and test if we're still within our recovery time. If not, then in lines 18-20 we remove the current node from the partition, cut cycles within the partition, and write the partition out to a file. One file per partition. Then in 21-22 we update our collection of output files and increment our counter for the number of partitions, and finally, in line 23-24 we create a new empty circuit for our next partition, and add the node to it. Then, we add the inputs to this node to our queue, and continue traversing and partitioning until we've reached every node. Lastly, in lines 31-35 we check if our current partition has anything in it. If so, cut loops and write it out.

Start recursing from outputs back to detect loops. Line 3 starts the recursive traversal for each output, with no parent.

Variable	Type	Description
$file$	file	input file
$targetRecoveryTime$	float	maximum per partition recovery time (in seconds)
$blif$	Blif*	In-memory representation of input blif file
$circuit$	BlifModel*	Main circuit from input file, represented as DFG
$partition$	BlifModel*	Circuit, which we are adding nodes to, to make our partition
$queue$	Queue	FIFO queue of nodes to visit
$visited$	Map(BlifNode*$\rightarrow$ bool)	Map of whether a BlifNode is visited
$signal$	Signal*	
$circuit.outputs$	List of Signal*	List of output Signal* of a circuit
$signal.source$	BlifNode*	Node which drives this Signal*
$queue.size$	integer	Number of nodes in queue
$node$	BlifNode*	
$file$	file	
$files$	List of file	
$numPartitions$	int	Counter of number of partitions
$signalName$	string	Name of a Signal*
$node.inputs$	List of string	List of names of signals which are inputs to this node
$model.signals$	Map(string $\rightarrow$ Signal*	Map from signal name to Signal* representing it in that BlifMo

varPart

Table 2: Variables for Partition

Variable	Type	Description
$latency$	float	Circuit latency (i.e. time for input to completely propagate to output) in seco
$clockFrequency$	Integer	Operating frequency of the circuit, in seconds
$criticalPath$	Integer	Maximum number of steps between an input and an output
$numFF$	Integer	Number of Latches in circuit
$numLUT$	Integer	Number of look up tables in circuit
$resynchronisationTime$	Float	Time, in seconds, that it takes to resynchronise circuit
$detectionTime$	Float	Time, in seconds, that it takes to detect an error
$ReconfigureTime$	Float	Time, in seconds, that it takes to reconfigure circuit
$communicationTime$	Float	Time, in seconds, that it takes to transmit reconfiguration request to controlle

varPart

Table 3: Variables for Partition

Variable	Type	Description
$partition$	BlifModel*	BlifModel* containing DFG representing partition to cut cycles in
$state$	Map(BlifNode* $\rightarrow$ int)	Map of whether a node is UNKNOWN, EXPLORING, or FINISHE
$signal$	Signal*	
$partition.outputs$	List of Signal*	List of Signal* representing primary outputs of circuit

varPart

Table 4: Variables for Partition

main

Algorithm 2 Main Algorithm

```
1: procedure PARTITION(file)
2:     blif ← new Blif(file)                                   ▷ Read in file
3:     circuit ← blif.main                                     ▷ The actual circuit within the blif file
4:     partition ← new BlifModel                               ▷ Empty Circuit
5:     queue ← new Queue                                       ▷ Empty Queue
6:     visited ← new Map(BlifNode → bool, DEFAULT: false)
7:     for all signal ∈ circuit.outputs do
8:         queue.Enqueue(signal.source)
9:     end for
10:    while queue.size > 0 do
11:        node ← queue.Dequeue()
12:        if visited[node] = true then
13:            continue                                        ▷ Handle each node once and only once
14:        end if
15:        visited[node] ← true
16:        partition.AddNode(node)
17:        if partition.RecoveryTime() > targetRecoveryTime then
18:            partition.RemoveNode(node)
19:            CutLoops(partition)
20:            file ← partition.WriteToFile()
21:            files ← files + file
22:            numPartitions ← numPartitions + 1
23:            partition ← new BlifModel                       ▷ Empty Circuit
24:        end if
25:        for all signalName ∈ node.inputs do
26:            signal ← model.signals[signalName]
27:            queue.Enqueue(signal)
28:        end for
29:    end while
30:    if partition.size > 0 then
31:        CutLoops(partition)
32:        file ← partition.WriteToFile()
33:        files ← files + file
34:    end if
35:    return files
36: end procedure
```

Variable	Type	Description
partition	BlifModel*	BlifModel* containing DFG representing partition to cut cycles in
state	Map(BlifNode* → int)	Map of whether a node is UNKNOWN, EXPLORING, or FINISHE
signal	Signal*	
partition.outputs	List of Signal*	List of Signal* representing primary outputs of circuit

varPart

Table 5: Variables for Partition

main

Algorithm 3 Main Algorithm

procedure RECOVERYTIME($partition$)
    $latency \leftarrow frequency \times criticalpath$
    $detectionTime \leftarrow latency$
    $resynchronisationTime \leftarrow latency$
    $reconfigurationTime \leftarrow max(numFF, numLUT)/10/15...morestuff$
    $communicationTime \leftarrow numPartitions \times latency \times morestuff$
    $recoveryTime \leftarrow detectionTime + resynchronisationTime + reconfigurationTime + communicationTime$
    return $recoveryTime$
end procedure

main

Algorithm 4 Main Algorithm

procedure CUTLOOPS($partition$)
    $state \leftarrow Map(BlifNode* \rightarrow int, DEFAULT : 0)$
    for all $signal \in partition.outputs$ **do**
        CutLoopsRecursive($state, NULL, signal$)
    end for
end procedure

main

Algorithm 5 Main Algorithm

procedure CUTLOOPSRECURSIVE($partition, state, parent, signal$)
    $node \leftarrow signal.source$
    if $state[node] = EXPLORING$ **then** ▷ Found a cycle
        ReplaceSignalName($parent.inputs, signal.name, "qqrin" + signal.name$)
    else if $state[node] = FINISHED$ **then** ▷ Already explored this path
        return
    else
        $state[node] = EXPLORING$
        for all $signalName \in node.inputs$ **do**
            CutLoopsRecursive($partition, state, node, partition.signals[signalName]$)
        end for
    end if
end procedure