PRESENTATION SLIDES FOR PUN-LOSP

William Bricken

July 1997

Interface as Overview

- Input Logic
  - directory of files (edif, pun)
  - filename
  - logic specification string (symbolic logic notations)
    - table
    - propositional calculus
    - parens
  - state transition graph
  - *active-graph* (highlighted circuit or subgraph)
  - subgraph (schematic, dgraph)
  - library (basic, custom)

- Activate-circuit
  - test for equivalence, before-after
  - partial evaluation, what-if

- Run-circuit
  - evaluate provided bindings
  - regression test with random bindings
  - generate test vector for subgraph

- Transform-circuit
  - apply BM axioms and theorems individually
  - reduce active-graph with specified level of effort

- Connectivity
  - coalesce pattern
  - group active-graph

- Standardize form
  - canonical form
  - partition into independent sets-of-support
  - partition variables into groups of n

- Abstraction
    make-cell using pattern
    make-block using pattern
    find and apply current abstraction to active-graph
    abstract variable symmetry groups

- Retiming
    move register
    make timing blocks
    change critical path length
    memoize

- Layout
    map to graph of target technology
    rearrange graph display

- Set performance parameters
    packing (high level, # boards)
    timing (critical path, # of cycles)
    development time (degree of automation)
    area
    wires
    fan-in/out

- Statistics
    nodes, wires, path length, fan-in/out
    compare circuits

- Make active objects (cells, blocks)

Losp Project Research History

Pure Boundary Math:

- distinction/void
- multiple representation, form diversity
- new proof techniques (insertion, graph-coloring)
- spatial deduction, unification, and predicate calculus

Graph Modeling:

- wire-or, reversibility, information loss, equivalence
- graph representation, interaction languages
- hierarchy, abstraction

Other:
- MCNC benchmarking
- forms of representation (matrix, data structure, pun, array)
- imaginary logics

Technical Design Objectives

- out-of-nothing, inevitably (void-based BM)

- look for big win, potential of a paradigm shift
  as good as any published work
  better technique (traditional approaches are baroque)
  wider expressibility (size of problem is beyond comprehension)

- interactive symbolic tool suite
  seamless integration of design interactivity
  IA not AI, but high-level automation tools

- formal rigor is assumed

- set parallelism
  parallel model which gracefully degrades to software serialism

- graph-based
  homogeneous representation and transformation model

- capitalize on strength of BM
  logic synthesis (boolean optimization, retiming)

- hierarchically scaleable
  seamless traversal of abstraction hierarchy

- MCNC benchmarks

Technical Issues

High-level
- all interesting transforms are NP
- massive database
  (20K gates = 2MB edif file)
- many implementation levels
  (hardware, compiled software, emulation, simulation)
- unifying model
  (circuit, logic, graph)

Low-level

- management of dynamic subgraphs
- parens and graph traversal
- canonical forms
- subgraph abstraction
- deep extract

Hardware Architecture

Graphs are specified by sets of vertices.

Graph and set operations must be modeled for parallelism.

The circuit itself computes in parallel.

So the Losp implementation supports strong parallelism.

Grain size of hardware architectures:

- very-fine-grain CAM mask
- FPGA
- fine-grain atomic threads (instruction sequences)

...

- course-grain processor array
- distributed processors

What's a Graph

Graph:

A set of (labels for) vertices
and a set of vertex pairs (edges between vertices)

Structure Sharing:

Graphs with multiple outputs
(and subgraphs with multiple upper connections)
can be seen as a collection of single output graphs
that have shared components.

Graph algorithms:

- Traversal
  connectedness, reachability, timing analysis, retiming
- Covering
  graph equivalence, library mapping, partitioning, coalesce
- Coloring
  structure of connectivity, deep extract

What's Boundary Math

- a calculus based on void, distinction (mark), imaginaries (i, j)
- higher dimensional, spatial representation graph and container models
- rewrite rules based on void-substitution
  (absorb, clarify, extract, distribute)
- models for logic, numbers, lambda calculus, graphics,
  knot theory, imaginaries, others

What's Computational Logic

- table lookup (1940)
  brute force, memory >> time, exponential space
- natural deduction (1950)
  many rules, insightful proof paths, inefficient
- resolution (1970)
  single rule, automated, exponential clauses, dominant technique
- matrix logic (1970)
  similar to table lookup with function abstraction
- boundary logic (1990)
  automated, efficient, minimalist

Boundary Logic Transforms

Axioms (all are deep transforms)

Dominion: A ( ) = ( )

Involution: ((A)) = A

Pervasion: A (A B) = A (B)

Simple Theorems

Occlusion:	(() A) = void
Subsumption:	(A) (A B) = (A)
Cancellation:	((A B) (A (B))) = A
Resolution:	((A C)((A) B)(B C)) = ((A C)((A) B))
Distribution:	((A B)((A C)) = A ((B)(C))

Insertion is a new approach to computational logic:

```
(a b) (a b c)     ->    (a b) ((a b) a b c)
                        (a b) ((   ) a b c)
                        (a b)
```

Boolean Complexity

n	#fn = 2^2^n	abstracted	
0	2	1	{( )}
1	4	1	{(x)}
2	16	3	{(x), (:a :b), (((a b)((a)(b))))}
3	256	13	
4	65000	221	
5	2^32	~200000	
...			
400	2^2^400	<very large>	(function space of typical chip)

Abstraction operations:

- function negation
- combine literals
- variable symmetry and permutations

Pun Levels of Hierarchical Abstraction

- different levels of abstraction
- same transformation mechanism across levels
- separate purely functional design from physical finetuning (layout)

Levels:

```
circuit
      block
            cell
                  parens
                        ground
                              bound-variables
```

Compare cells	to blocks
strong intercell calculus	no interblock calculus
no functional reference	strong embedded functional reference
no binding mechanism	environmental binding is expected
no concept of i/o	i/o driven evaluation and expansion

General Pun Form

```
circuit =

  ((circuit-name)
   ((main) (main-bots) (main-tops) (main-body))
   ((library-name1)(local-bots1)(local-tops1)(local-body1))
     ...<as-many-other-blocks-as-desired> )

block =

  ((block-name1)(local-bots1)(local-tops1)(local-body1))

body =

  ((cell-id1  parens1)
   (cell-id2  parens2)
   (cell-id3  block-call1)
    ...)

Block call =

  (type-name <bot-bindings> <top-bindings>)
```

Graph Partitioning

- Logic (function) is independent of layout (functional invariance).

- Cell and library forms convey design semantics.

- Structure sharing is an implementation choice,
  driven by available layout and routing resources.

- Partitioning, coalescing, form abstraction,
  and expansion of blocks and cells
  all depend on the goals of
  design,
  implementation efficiency, and
  technology mapping.

Semantics of Pun Hierarchy

- Cell (dynamic changes)
  - logic optimization
  - variable abstraction
  - basis conversion
  - canonicalization
  - LUT partitioning
  - critical path optimization
  - temporary coalesce
  - fine-grained display

- Block (long-term storage)
  - timing blocks
  - pipelining and synchronized components
  - longterm coalesce
  - function and partial function evaluation
  - what-if modeling
  - layout partitioning/memoizing
  - block design display
  - components for reuse

- Circuit (problem, model)
  - chip and board boundaries
  - design parameters

Cell Reduction Model

Expand cells into parens form whenever possible
  (more efficient, less memory, more modular)

Reduction Strategies:

- grounds
```
(cell1  (( ) a))
(cell2  (1 3))          ==>  (cell2  (3))
```

- literals
```
(cell1  (a))
(cell2  (1 3))          ==>  (cell2  ((a) 3))
```

- single mention
```
(cell1  (2 3))
(cell2  (4 5))          ==>  (cell1  (3 (4 5)))
```

- canonical standardization
```
(cell1  ((b 3 a) (1 c)))  ==>  (cell1  ((c 1)(a b 3)))
```

- distribute out
```
(cell1  ((a b)(a c)))     ==>  (cell1  ((a ((b)(c)))))
```

- function abstraction
  <see handout>

- variable symmetry abstraction
  <see handout>

- subgraph restructuring
```
(cell1  ((a b)((a) c)))   ==>  (cell1  ((a b)((a) c)(b c)))
```

Function Abstraction

When a function is abstracted from the parens form, the behavior of the function is differentiated from the behavior of the distinction.

An abstracted function must include both the form and the form's interaction with boundaries and spaces. Otherwise, the function will have to be destructured into parens to participate in transformations.

Eg: the interaction between distinctions and equality:

```
(A)=B     <==>   A=(B)   <==>  (A=B)
 A  A=B   <==>   A (B)
(A) A=B   <==>  (A) B
```

Types of Abstraction

Structural

- coalesce (share identical circuit structure)

  A A = A

- grouping (isolate collections of subgraphs)

  (A B C) = (A ((B C)))

- distribute (factor, trade reference for depth)

  ((A B)(A C)) = A ((B)(C))

- pivot (standardize literal polarity)

  ((A B)((A) C)) = (A (B)) ((A) (C))

Functional <see handout>

- abstraction (make function into basis cell or block)
- partial evaluation (expand function by a bound variable)
- equivalence (test for tautology)
- symmetry groups (embed relational constraints)
- partitioning (specify subgraph based on set of support)

Defunfun

An essential premise of BM is that object and process
(form and function) are different views on the same thing.

Defunfun converts cell sets (and block sets) to active functions.

The *circuit* is a shared data structure for cell and block "active objects",
managed as a Linda object (get, put, copy)

The active-object model:

- programming language debugger becomes a circuit analysis tool
  (trace, step, etc.)
- can configure objects by adding message-handlers to function body
- function body becomes a site for hooking graphics actions
- memoization: functions with local store can retain previous results
- autonomous parallelism
- dynamic interactivity of dnodes eliminates search from evaluation
- only necessary activities are cued, can be handled locally

Implemented...mid-1997

- edif2pun
  - file management
  - modular inclusion of new gate types, library elements
  - make-library

- evaluate grounds
  - provided
  - random
  - propagated and trapped

- parens reduce

```
grounds
                                (( ) a)  ==>  <void>

shallow-literal-extract
                              (a (a b))  ==>  (a (b))

deep-literal-extract
                      (a (b (c (a d))))  ==>  (a (b (c (d))))

deep-recursive-literal-extract
                        (a (b (a b c)))  ==>  (a (b (c)))

shallow-bounds-extract
                       ((a b)(c (a b)))  ==>  ((a b) (c))

deep-bounds-extract
                   ((a b)(c (d (a b))))  ==>  ((a b) (c (d)))

deep-recursive-bounds-extract
      ((a b) ((c d (a b)) (e (c d))))  ==>  ((a b) ((c d)(e)))
```

Mostly Implemented...mid-1997

- functional equivalence (satisfiability checker)

- partition and form abstraction
  - coalesce (structure sharing)
  - set of support
  - n-lut-mapping

- parametric circuit generation
  - number of dnodes = area of circuit
  - number of variable mentions = number of wires
  - number of specific variable mentions = fan-in and fan-out
  - longest path through subgraph = critical path time

- display
  - circuit, bounds, dnet, parens

- statistical-compare-files
  - regression to provided test vectors
  - comparative changes in
    - tops, bots, cells, dnodes, vars, chars, max-depth, max-length

- meta-reduce
  - apply a list of transformations to a set of circuits

- generate-all-boolean-functions

- make-bdd