SPFDs - A new method for specifying flexibility

Sets of Pairs of Functions to be Distinguished

From the paper:
“A new Method to Express Functional Permissibilities for LUT Based FPGAs and Its Applications”

by S. Yamashita, H. Sawada, and A. Nagoya, NTT Communications Science Lab., Kyoto, Japan ICCAD'96

BUT: this paper is more about a new kind of “don’t cares”, rather than about application to FPGAs
Methods for Expressing Sets of Functions

• Don’t Cares (=> incompletely specified functions (ISFs))
  - Satisfiability DCs
  - Observability DCs
  - XDCs
• ATPG Redundancy Removal
• Boolean Relations
• Sets of Boolean Relations
• SPFDs. (Actually, they are sets of ISFs).
ODCs and SPFDS

ODCs:
- Different ways to compute them
  • Full method (difficult)
  • Subsets
    - Compatible Sets of Permissible functions (CSPFs)
      » (Easier to compute)
    - Compatible observability don’t cares (CODCs)
      » see Savoj thesis - generalization of CSPFs
- CSPFs and CODCs
  • have advantage that they can be computed fast - from outputs to inputs
  • are independent
ODCs and SPFDS

SPFDs:
- like CSPFs/CODCs
  - can be computed fast from outputs to inputs
  - are independent
- but more powerful and
- not a subset of DCs (they are a sets of ISFs)
**ISFs**

Usually we deal with binary ISFs.

- They can be represented as complete bipartite graphs on minterms.
- Vertices are minterms. Set of vertices may be subset of all minterms
- An edge means that we need to build a function that distinguishes (i.e. has different values) the two minterms on the edge.

A connected bipartite graph has only two colorings, one and the other the offset and vice versa (on set, offset)
Building $F$

We start with a complete bipartite graph (ISF) for $F$. This gives a set of minterms that must be distinguished.

$Y$ space is $(y_1, y_2, y_3)$. Each $y^i$ is a minterm in $Y$.
Building F

Suppose that the inputs to F are 
\[ y = (g_1(x), g_2(x), g_3(x)). \]
As long as \( \{y^1, y^2, y^3\} \) are encoded differently from 
\( \{y^4, y^5\} \) by \( g(x) \), there exists a function \( F \) which implements the ISF.

<table>
<thead>
<tr>
<th>old</th>
<th>new</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y^1 )</td>
<td>001 101</td>
</tr>
<tr>
<td>( y^2 )</td>
<td>010 110</td>
</tr>
<tr>
<td>( y^3 )</td>
<td>011 010</td>
</tr>
<tr>
<td>( y^4 )</td>
<td>100 001</td>
</tr>
<tr>
<td>( y^5 )</td>
<td>101 111</td>
</tr>
</tbody>
</table>

**New F:**
\[
\begin{align*}
  f(101) &= 1 \\
  f(001) &= 0 \\
  f(110) &= 1 \\
  f(111) &= 0 \\
  f(010) &= 1 
\end{align*}
\]
Distributing the job

We can distribute the “encoding” job separately to the input wires.

Note: as long as each edge is distinguished by one of the inputs, this is sufficient for a new encoding.
Input Graphs

Each of the edges on the input graphs means that the input signal $g_i(x)$ has to have a different value for the two nodes.

Note: the graphs are bipartite, but not complete and may be disconnected. In the above case, each graph has 4 different legal colorings.
Assigning Values

Now we can assign values to the graphs (color the graphs).

Thus, for example

\[ y^1 = 1^1, \ y^2 = 00^*, \ y^3 = *10, \ y^4 = 011, \ y^5 = 100 \]
Assigning Values

Note that $y^1, y^2, y^3, y^4, y^5$ are all distinctly encoded in this case. But, in general, that is not necessary,

\[ \begin{align*}
\text{input 1: } & y^1 \\
\text{input 2: } & y^2 \\
\text{input 3: } & y^3
\end{align*} \]

e.g. could have $y^1$, $y^2$, $y^3$ all encoded with the same value.

The original ISF can be implemented with either encoding $F(y^1) = F(y^2) = F(y^3) = 1$ and $F(y^4) = F(y^5) = 0$.

Now just call ESPRESSO to get a minimized $F$. 

**SPFDs**

Now we generalize this because

- we can
- it is more powerful
  (can be applied to multiple outputs and multiple valued variables)
- it improves understanding

Definition 1: A SPFD $F(y)$ on domain $Y$ is an undirected graph $(V,E)$ where each $v \in V$ is associated with a minterm $m = (y_1, \ldots, y_k) \in Y$
**Implements**

Definition 2: A function \( f(y) \) implements \( F(y) = (V,E) \) iff \( f(y) \) is a valid coloring of \( F \), i.e.

\[(y^1, y^2) \in E \text{ iff } f(y^1) \neq f(y^2),\]

Three Colorings

Must have at least 3 colors since the chromatic number of graph is 3
Compatible

Definition 3: A mapping
\[ y(x) = (y_1(x), \ldots, y_k(x)) \]
is \( F(z) \)-compatible if
\[ \forall ((z^i, z^j) \in F(z)) \]
y(\( X(z^i) \)) \( \cap \) y(\( X(z^j) \)) = 0
where \( X(z^i) = \{ x \mid z(x) = z^i \} \).
Re-implementing a Function

Theorem 1: If $y(x)$ is $F(z)$-compatible, then there exists a function $g(y)$ which implements $F(z)$.

Proof.
Translate SPFD $F(z)$ into $F(y)$. Thus for each $(z^i, z^j) \in F(z)$, put an edge for every pair $(y^k, y^l)$ where $y^k \in y(X(z^i))$ and $y^l \in y(X(z^j))$. Since $y(X(z^i)) \cap y(X(z^j)) = 0$, there is no self edge in $F(y)$. Hence a coloring exists.
Re-implementing a Function

Color $F(y)$ and implement the corresponding ISF by some function $g(y)$.

Each $y(X(z^i))$ creates an equivalence class which can all be colored with the same color, (but different colors can be used if preferred.)
Re-implementing the Inputs to a Function

We want to build new functions $y'(x)$ at the inputs to $G$. 

Boolean Network

Y-space

G

Z-space

x-space (primary inputs)
Re-implementing the Inputs to a Function

For new functions \( y'(x) \), just need for each edge in \( F(y) \), at least one input that has different values for the two minterms of the vertices of the edge.

Theorem 2: \( y'(x) \) is \( F(z) \)-compatible if
\[
\forall (z^i, z^j) \in F, \exists k, \forall ((x^i, x^j) \in (X(z^i), X(z^j))), \ y'_k(x^i) \neq y'_k(x^j)
\]

Proof. Since \( y'_k(x^i) \neq y'_k(x^j) \), then \( y'(x^i) \neq y'(x^j) \). Hence \( y'(X(z^i)) \cap y'(X(z^j)) = 0 \).
Re-using Old Topology

Note: up to now we could have assigned any edge of the node SPFD to any of the inputs.
An efficient way of building a new mapping $y'(x)$ is to use part of the old implementation.

Boolean Network

x-space (primary inputs)

Y-space

Z-space

G

$w^1$ $w^2$ $w^3$
Re-using Old Topology

Construct SPFDs \( \{F_1(y),...,F_r(y)\} \) (for each local input) such that

- \( F_k(y) = \{ (y^i,y^j) \in F(y) \text{ only if } y^i_k \neq y^j_k \} \)
  - i.e. the \( k^{th} \) input currently distinguishes \((y^i,y^j)\) for all those \( x \) which map to \( y^i \), or to \( y^j \)

- \( \forall (y^i,y^j) \in F(y), \exists k, (y^i,y^j) \in F_k(y) \)
  - i.e. each edge is assigned to one of the inputs
Efficient Re-implementation

Theorem 3: Suppose the current implementation of the global functions at $y, y(x)$, are implemented with sets of intermediate variables $w = (w^1, ..., w^r)$,

i.e. $y(x) = (f_1(w^1(x)), ..., f_r(w^r(x)))$.

Then for each $k$, any coloring of $F_k(y)$ can be implemented with some function $f'_k(w^k(x))$.

(i.e. the new fanins of $G$ can be implemented with the same inputs $w^1, ..., w^r$ as the old fanins)

Thus a new set of input functions $y'(x)$ can be obtained as

$y' = (f'_1(w^1(x)), ..., f'_r(w^r(x)))$

i.e. a set of functions of the current intermediate inputs to $f_1, ..., f_r$.
End of lecture 15
Efficient Re-implementation

Proof. Consider $F_k(y)$ for some input $k$. The current $f_k$ implements $F_k(y)$ since for $(y^i, y^j) \in F_k$, $f_k(y^i) \neq f_k(y^j)$, i.e.

$f_k(X(y^i)) \cap f_k(X(y^j)) = 0$

Thus for all $x^i \in X(y^i)$, $x^j \in X(y^j)$, we must have $w^k(x^i) \neq w^k(x^j)$. Otherwise, $f_k(w^k(x^i)) = f_k(w^k(x^j))$ contradicting that $f_k(y^i) \neq f_k(y^j)$. Since $w^k$ distinguishes $x^i$ from $x^j$ (i.e. the information is available at the inputs of node $k$), functions $f'_k(w^k)$ exist which distinguish $x^i$ and $x^j$. 
How Do We Use This

Boolean Network

$y^+ = (y(x), y_{k+1}(x))$. 

**Theorem 4**: Wire $y_1$ can be removed if and only if $(y_2, ..., y_{k+1})$ is $F(z)$-compatible.
Conclusions and future

- SPFDs offer greater choice in implementing a circuit
  - Can change both a node and its inputs
  - An SPFD can be colored in many ways
- New way to obtain Boolean divisors
- Implementation using MDDs in progress
- Implementation using SAT in progress
- Useful for DSM for rewiring for noise avoidance or better layout
- Can be used to define “don’t care wires”
\[ y = (x)^2 \text{ and } z = (x)^2 \]

If there exists \( x \) such that \( y \rightarrow z \), then the image computation is almost done. Have developed a method using \( \text{SAT} \) for medium sized circuits. Currently done using \( \text{BBDD} \) computing.
MV-SPFDs and Wire Removal

An MV-SPFD represents the information content of a wire in a network.

- the MV-SPFD of each fanin to a node, gives the information that it must provide.

The sum total of the information content of a node's fanin wires must cover the information required of the output of the node.

We determine a minimal subset of fanins at each node which cover the required information by solving a covering problem.
Wire Removal and Wire Substitution

Wire removal: suppose
\[ SPFD_{out} \subseteq SPFD_{in3} \land SPFD_{in4} \land SPFD_{in5} \]
Then, can remove in1 and in2

Wire substitution: suppose
\[ SPFD_{in4} \subseteq SPFD_{in6} \]
Then, can replace in4 by in6

Important:
Because wires are changed, the internal logic of a block may change.
Don’t Care Wires

Each input pin $\sigma$ will have a set of sources, $S_\sigma$ (among which we can choose any one).

Each source $\omega$ (driver of a wire) distinguishes a set of pairs of minterms (SPFD) in the primary input space.

Wire $\omega$ is assigned as a candidate source for input pin $\sigma$ if:

$$\text{SPFD}_\sigma \subseteq \text{SPFD}_\omega$$
Compatible sets of don't care wires

Want to be able to select one source from each set $S_{\sigma}$ independent of the selection made from any other set $S_{\sigma}'$ (compatible set of candidate sets of sources)

**Theorem:** A set of sets $\{S_{\sigma}\}$ of candidate wires is compatible if
- it can never produce a cyclic network
- for each wire $\omega \in S_{\sigma}$, $\text{SPFD}_{\sigma} \subseteq \text{SPFD}_{\omega}$
Choosing best subset of wires

AWC: alternate wire choice problem

Given: a point placement of pins and a set of candidate wires for each input pin
Find: a selection of wires for each input pin that minimizes the total half-perimeters of the nets’ bounding boxes.

(The problem is NP-complete)
Placement using don’t care wires

Current placement by Mincut/Annealing or by a Force Directed Method:
  - wire choices considered in move to determine cost of move
    - use semi-greedy AWC algorithm during each placement move
Experiments - total wire length (placement, no routing)

Max Alternates (15%)

Alternate wires (13.5%)

Initial PLA netlist

Wire removal and Re-synthesis

Place 0%

Place 8.5%

Place 6.6%

Wire choices

Re-synthesis

Place 10.2%

Wire removal

Place 14.4%

Wire removal currently does not use placement information

Notes: 13.5% → 15% in alternate wires results in 6.6% → 10.2% improvement in wire length.
Wireplanning before decomposition

Primary outputs: information required (in terms of PIs)

Function inside block only needs to provide the information required of its fanouts

A block logic function exists iff the information at the inputs covers the information required at the output
Information Flow Theorem

Theorem: A set of logic functions exists for the blocks if and only if there exists a path from input $I_k$ to output $O_j$ whenever $O_j$ depends on $I_k$.

Open question: How do you make the assignment of SPFD edges so that the resulting logic in the blocks is as simple as possible?