ACID’2002: Design Flow and CAD Tools for asynchronous cells

by
Cédric Cuche
Christian Piguet
Vojin G. Oklobdzija

<table>
<thead>
<tr>
<th>1. Application domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Design flow of asynchronous cells</td>
</tr>
<tr>
<td>3. Software Tools and examples:</td>
</tr>
<tr>
<td>3.1. A D-Flip-Flop (design flow)</td>
</tr>
<tr>
<td>3.2. A D-Flip-Flop with critical races</td>
</tr>
<tr>
<td>4. Performances</td>
</tr>
<tr>
<td>5. Conclusions</td>
</tr>
</tbody>
</table>
- What class of asynchronous cells?
  Speed-Independent CMOS standard cells (low complexity)
  (CMOS are built up with NOR’s, NAND’s => only Negative CMOS gates)

- What kind of help and analysis tools?
  STG tools:
  - STG -> Boolean Equations generator (Petrify)
  - Negative STG properties verification
  Alcyon:
  - Only negative equations verification
  - Flow Tables with races and stable states
  - Races analysis

- How tools are validated?
  Tools applied on already known SI and non SI cells (D-Flip-Flop, Latch, …)
ACID’2002: CAD tools for asynchronous cells design

2. Design flow of asynchronous cells

Our Asynchronous Cell design workflow

- Timing diagrams
- STG
- Bounded-PN
- State graph
- Karnaugh Map
- Petri

STG properties for St: Bounded + Complete State Assignment + Complete State Coding + Persistency + Commutative

- Modified STG for negative gates
- STG Properties for Negative Gates: Alternate transitions × no inputs conflicts × cycle rule × ...

Tool: hands and head!

Initial D-Flip-Flop STG specification

Tool: hands and head!

Initial D-Flip-Flop STG specification

Tool: hands and head!

Initial D-Flip-Flop STG specification
Tool: Petrify

STG properties to check for SI implementation:

Petrify is able to check if the STG is SI

=> “The STG has CSC” and gives the cell equations

1. Alternate transitions: a+->b-
   -> insertion of a new signal
2. Inputs conflicts: i--->a- et i--->b+
   -> insertion of a new signal
3. Cycle rule: in a STG cycle (a+……>a+)
   -> try to use memory elements
4. Memory of successive input transitions

Tool: hands and head!

Hints to obtain a NEGATIVE STG
3.1. A D-Flip-Flop example: Synthesis STG

Result: Negative Speed-Independent STG specification for D-Flip-Flop

Tool: Verify STG

STG properties to check for implementation in negative gates SI:

VerifySTG is able to say if the STG is NEGATIVE:

---check for alternate transition rule---
rule 3.4 ok

---check for input conflicts rule---
rule 3.5 ok

---check for cycle rule---
rule 3.6 ok

If a rule is not satisfied, VerifySTG print the faulty STG fragment
3.1. A D-Flip-Flop example: STG synthesis's

**Tool:** Petrify or manual

**STG -> equations transformation:**

- **Manual:** STG -> manual transformation -> Flow Table -> Simplification -> Equations
- **Petrify:** STG -> Petrify -> Equations

```
Petrify output

INORDER = ck d q nq mm;
OUTORDER = [s] [mq] [mm];
[s] = mq';
q = mm';
[mq] = nq (s' + ck') + a';
[mm] = d' (mm + ck') + a';
a = mm' + ck';
```

---

```
Alcyon input

#D-Flip-flop
A = NM' + CK';
NM = (D' + (M' + CK')) + A';
M = NM';
NQ = (Q' + (M' + CK')) + A';
Q = NQ;
```

---

ACID'2002: CAD tools for asyn cells design

**Tool:** Petrify or manual

**STG -> equations transformation:**

- **Manual:** STG -> manual transformation -> Flow Table -> Simplification -> Equations
- **Petrify:** STG -> Petrify -> Equations

```
Petrify output format

INORDER = ck d q nq mm;
OUTORDER = [s] [mq] [mm];
[s] = mq';
q = mm';
[mq] = nq (s' + ck') + a';
[mm] = d' (mm + ck') + a';
a = mm' + ck';
```

---

```
Alcyon input format

#D-Flip-flop
A = NM' + CK';
NM = (D' + (M' + CK')) + A';
M = NM';
NQ = (Q' + (M' + CK')) + A';
Q = NQ;
```
3.1. A D-Flip-Flop example: equations transformation

Petrify outputted equations are not minimal

D-Flip-Flop from the petrified STG (equations without races) with 26 MOS

\[
\begin{align*}
A &= NM'CK' \\
NM &= (D'^*(M'^*CK'))+A' \\
H &= NM' \\
NQ &= (Q'^*(M'^*CK'))+A' \\
Q &= NQ' \\
\end{align*}
\]

Another D-Flip-Flop (equations without races) with 24 MOS

\[
\begin{align*}
A &= NM^*CK' \\
NM &= ((C^*K^*M)+(D^*A))' \\
H &= NM' \\
NQ &= ((C^*K^*M)+(A^*Q))' \\
Q &= NQ' \\
\end{align*}
\]

Logic equations have MANY CMOS transistor mapping

The most simple mapping:

P network equations = the equations
N network equations = complement of the equations

A N&P equations mapping for this D-Flip-Flop:

\[
\begin{align*}
A &= NM'^*CK' \\
H &= NM' \\
Q &= NQ' \\
NM &= (D'^*(M'^*CK'))+A' \\
NQ &= (Q'^*(M'^*CK'))+A' \\
\end{align*}
\]
Tool: Alcyon

Input: equations (implementation spec.)

Output:

- All total stable state (=permanent state without input changes)
  \( \{ \text{present State} \mid \text{excitation state} = \text{equations (inputs state, present state)} \} \)

- All tristates or shortcircuits
  \( \{ \text{states with X or H or L signal} \} \)
  
  Alcyon implement a partial part of IEEE1164 Bit logic for equations resolution

- All reachable states from a given stable state
  -> construct a graph of all states from the stable states
  -> mark states as: stable, unstable OR with a race

Equations from Petrify:
Gate implementation

N&P equations from optional step:
Transistor implementation

No race -> analysis is finished

Gate implementation is equivalent to transistor implementation
Another D-Flip-Flop, manually generated, searched for transistor count reduction, in order to reduce power and improve speed. Only 4 negative gates.

```plaintext
# D-Flip-flop
#
# US Patent 5.748.522
#
A = (B*CK)';
B = ((D*A) + (CK*A))';
NQ = ((Q'*A) + (CK*A))';
Q = NQ';
```


- Is there any race?
- Are races critical or not?
- Show a tree of race executions
Tool: Alcyon

- Is there any race? YES

C K D
A  B  N  Q  01  01  11  10
---------------------------------------
1 0 0 1  1101 (1001) (1001) (1001)
1 1 0 1  1 (1101) 1001 - [0001]
1 1 1 0  1 (1110) 1010 - [0000]
1 0 1 0  1110 (1010) 1000 -
1 0 0 1  -  1001 -
0 1 1 0  1110 1110 (0110) (0110)

1101 differs from 0001 by 2 bits => A and B are in race
1110 differs from 0000 by 3 bits => A, B, NQ are in race

Tool: Alcyon

- Are races critical or not?

Show a tree of races executions

No critical races ⇔ all paths do end up to the same stable state
ACI02002: CAD tools for asyn cells design  3.2. A D-Flip-Flop with critical races: 2nd race

Tool: Alcyon

- Are races critical or not?
  - Show a tree of races executions

Conclusion: This D-Flip-Flop is not SI (without delay assumptions)

Our tools:

- All tools run on Window2k at least and must be simple enough for design engineer’s

- All tools are written in Java2. Also you need Petrify properly running on our platform for using VerifySTG and generating equations

- All tools are designed for ease of modification

- Alcyon logic uses parts of VHDL IEEEl164 bit library as internal bit representation

- Alcyon supports up to 65’000 variables per equations (also algorithms are not yet designed for efficiency)
ACID'2002: CAD tools for asyn cells design

4. Performances

University California Davis performance comparison framework

D-Flip-Flop compared in the same submicron technologies

CSEM non SI D-Flip-Flop (patented) compared with single-ended static structures (colored ones) is good!

Comparison provided by UC DAVIS CA, Vojin G. Oklobdzija

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CFFF</td>
<td>246.8</td>
<td>127.3</td>
<td>18.5</td>
<td>7.7</td>
<td>153.4</td>
<td>37.9</td>
</tr>
<tr>
<td>CPFF</td>
<td>202.3</td>
<td>117.0</td>
<td>24.4</td>
<td>4.9</td>
<td>146.3</td>
<td>29.6</td>
</tr>
<tr>
<td>invCCFF</td>
<td>257.4</td>
<td>110.8</td>
<td>10.2</td>
<td>0.7</td>
<td>121.7</td>
<td>31.3</td>
</tr>
<tr>
<td>HFF</td>
<td>187.9</td>
<td>161.3</td>
<td>18.0</td>
<td>4.4</td>
<td>183.8</td>
<td>34.5</td>
</tr>
<tr>
<td>SFF</td>
<td>164</td>
<td>256.0</td>
<td>50.4</td>
<td>4.4</td>
<td>311.4</td>
<td>51.4</td>
</tr>
<tr>
<td>CMOS</td>
<td>354</td>
<td>110.8</td>
<td>27.5</td>
<td>2.8</td>
<td>141.1</td>
<td>49.9</td>
</tr>
<tr>
<td>PosPC</td>
<td>300</td>
<td>80.0</td>
<td>32.1</td>
<td>11.1</td>
<td>123.2</td>
<td>36.9</td>
</tr>
<tr>
<td>This work</td>
<td>280</td>
<td>87.7</td>
<td>28.4</td>
<td>4.5</td>
<td>120.7</td>
<td>31.5</td>
</tr>
<tr>
<td>TSPC</td>
<td>254</td>
<td>97.8</td>
<td>30.1</td>
<td>2.8</td>
<td>130.7</td>
<td>31.7</td>
</tr>
<tr>
<td>2TCFF</td>
<td>292</td>
<td>110.5</td>
<td>8.7</td>
<td>9.3</td>
<td>128.5</td>
<td>37.5</td>
</tr>
<tr>
<td>STSC</td>
<td>501</td>
<td>79.8</td>
<td>5.7</td>
<td>1.2</td>
<td>86.7</td>
<td>43.4</td>
</tr>
<tr>
<td>DSTD</td>
<td>508</td>
<td>82.5</td>
<td>5.7</td>
<td>1.2</td>
<td>89.4</td>
<td>45.4</td>
</tr>
<tr>
<td>SAFF</td>
<td>274</td>
<td>87.0</td>
<td>5.2</td>
<td>1.3</td>
<td>93.5</td>
<td>25.7</td>
</tr>
<tr>
<td>SABFF</td>
<td>169</td>
<td>100.8</td>
<td>5.8</td>
<td>1.3</td>
<td>107.9</td>
<td>18.2</td>
</tr>
<tr>
<td>SLFF</td>
<td>155</td>
<td>95.9</td>
<td>6.6</td>
<td>1.3</td>
<td>104.0</td>
<td>16.1</td>
</tr>
<tr>
<td>DE-CFF</td>
<td>169</td>
<td>112.5</td>
<td>17.0</td>
<td>2.6</td>
<td>132.1</td>
<td>22.3</td>
</tr>
</tbody>
</table>

VerifySTG helps in specifying new correct SI cells for CMOS libraries

Alycon helps:
- to discover implementation incorrect behaviors
- trace complex races executions
- design robust SI and non-SI cells

The proposed methodology works well with all tested cases

Non-SI circuits may be derived from robust SI design’s for Speed and Power optimizations