How to synthesize DDR multiplexers

Michael Krieger, ZITI, Uni Heidelberg

28th CBM Collaboration Meeting
Tübingen, 09/2016
Introduction

- SPADIC – readout ASIC for TRD
- Largely consists of synthesized digital logic
- Uses DDR serial data output
- Versions: 1.0 (Nov. 2011), 1.1 (Nov. 2015), 2.0 (June 2016)
- There were some problems implementing DDR correctly
- What went wrong? How to do it right?
Scope

Results apply in particular to

- Synthesized (‘semi-custom’) ASICs
- Using a standard cell library
- Using Cadence RTL Compiler (Versions 12–14)

Fundamentals also applicable to

- FPGAs – understanding built-in DDR macros
- Full-custom ASICs – building your own DDR serializer
Serial data transmission

- Output the individual bits one after the other
- Synchronous to the clock signal that is used inside the chip
- ‘Single data rate’ – one bit per clock cycle (→ trivial)
- ‘Double data rate’ – two bits per clock cycle
- Serialized data is *sampled* at the receiving side – must be stable as long as possible (SDR: 1, DDR: ½ clock cycle)

![Diagram showing clock, SDR, and DDR signals with good and bad states]
Implementing DDR

Problem

- Flip-flops are sensitive to one of the clock edges
- How to update a signal (exactly) *twice* per clock cycle?

Basic idea

- Use two flip-flops and a multiplexer
- Alternate between the inputs *using the clock*

![Diagram](image)

- Must ensure that *first* is output before *second*!
Choosing clock polarities

- Model all possible configurations \((2^3 = 8)\) by inserting inverters
- Naming convention: \((n|p)(n|p)(-|+)\)
  - first sensitive edge, second sensitive edge, select polarity
- Only 4 out of 8 combinations give correct results:\(^1\)
  - \(pp-, np+, np-, nn+\) (see Appendix)

\(^1\)Considering the ideal case where gate/wire delays \(<\) clock period
Choosing clock polarities: Examples

(clk connections implied, inverters denoted by O)

- **PN+**
  - First
  - Second

- **PP-**
  - First
  - Second

Another naming convention: (1|2)(1|2)
- Selected half of first, selected half of second

- Correct! (First before second)
- Wrong! (Second before first)
Implementing the multiplexer

Synthesis tool translates HDL code to logic function:

<table>
<thead>
<tr>
<th>sel</th>
<th>a</th>
<th>b</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ y = a \cdot \neg sel + b \cdot sel \]

Problem

\( \text{sel and } \neg \text{sel are independent inputs } \rightarrow \text{glitch when relative delay } \neq 0 \)
- True CMOS implementation of the multiplexer logic function – PMOS and NMOS networks are complementary (or dual)
- Glitch occurs when \( a = b = 1 \) and \( \text{sel} = 1 \rightarrow 0 \)
- \( y \) should be 1 (\(!y = 0\)) at all times
Glitch in SPADIC 1.0

- For each output bit (½ clock period), for the duration equal to the relative delay between sel and !sel, the signal is potentially wrong.
- Safe ‘sampling window length’ is reduced by this amount.
- Observed: $\approx 2$ ns (distance of red line to black line).

- Simple inverter delay only $\approx 0.1$ ns.
- **Reason:** sel and !sel taken from different depths of the clock tree!
- Limits SPADIC 1.0 practically usable clock speed to $< 250$ MHz (which was the target).
Solution (Part 1)

- A dedicated multiplexer cell using *gated inverters* – PMOS and NMOS networks are *symmetric* rather than *complementary*
- \(!sel\) derived internally from \(sel\) (not shown here)
- More importantly: no glitch possible (see Appendix for all cases)
- Again considering \(a = b = 1, \; sel = 1 \rightarrow 0:\)

![Circuit Diagram]

Michael Krieger, ZITI, Uni Heidelberg

How to synthesize DDR multiplexers 11
Enforcing the dedicated multiplexer

Option 1: Compiler directive

```verilog
assign y = sel ? // pragma map_to_mux
    b : a;

y <= b when sel -- pragma map_to_mux
    else a;
```

Must be ‘applied’ to the respective selection operator – code layout matters!

Option 2: Create an instance manually

Must set the `dont_touch` attribute for the instance, otherwise it will be replaced by AOI22 again during optimization!

Tradeoff: Keeping the HDL code independent from the target technology vs. knowing the instance name (important later)
Problem solved in SPADIC 1.1?

Indeed the sampling window is good – but only up to $\approx 150$ MHz.

It diminishes at higher speeds depending on the clock frequency (red points).

Still limited to $< 250$ MHz.

There’s more that can go wrong!
Multiplexer input timing

- Arbitrary timing relationship between each of the data inputs and the select input of the multiplexer (wire delays, inserted buffers)
- Each input must be stable while it is selected
- Synthesis tools *could* enforce this if they *knew* what the desired situation is
Intermediate solution for SPADIC 2.0

- How to teach the correct multiplexer input requirements to the synthesis tools was not known at the time when SPADIC 2.0 was submitted.
- After synthesis, it turned out that the inputs were off by about ½ clock period using the chosen configuration ($pp$–).
- Inputs were shifted by changing the configuration to $np$– and running synthesis again, hoping that the results would otherwise stay similar enough (this was the case).
- Both variants were submitted to confirm the simulation results.
Solution (Part 2)

Setting the right constraints

- `set_data_check` should work, but ignored by *RTL Compiler* (bug)

- ‘Workaround’ provided by Marios Karatzias from Cadence support (actually more elegant – normally used in a different context, but expresses precisely what is needed here)

- Important: must be applied to the instance of a library cell (‘leaf pins’, as opposed to ‘hierarchical pins’) → must know its name

```plaintext
set_clock_gating_check -low my_mux_instance/a
set_clock_gating_check -high my_mux_instance/b
```
Remark about setting constraints

My personal recommendation

- Use the corresponding commands (dc::prefix) in the Tcl synthesis script instead of reading an SDC file!
- Reason: Misspelled pin names, etc. will stop the script instead of silently(!) being ignored.
Choosing clock polarities revisited

- Of the 4 possible logically correct configurations, which one is ‘best’?
- They differ in which halves of the inputs are selected:
  \[ pp-, \ n n+ \rightarrow “12”, \ n p+ \rightarrow “11”, \ n p– \rightarrow “22” \]
- Synthesis likes to correct timing violations by inserting delays into \textit{data} (first, second), not \textit{clock} (sel) signals
- ‘too late’ data signal (w.r.t. sel) will be shifted into the next clock cycle → wrong output order

\begin{itemize}
  \item \textit{np–} (“22”) is ‘best’, however there is no guarantee: if a data signal is more than \( \frac{1}{2} \) cycle too late, a different configuration may lead to better results.
\end{itemize}
Summary

1. Choose one of the logically correct DDR mux configurations
2. Enforce the use of a proper mux library cell (no glitch possible)
3. Apply the proper timing constraints (`set_clock_gating_check`) to the *leaf pins*
4. Verify the result, adjust configuration if necessary
## DDR mux configurations table

<table>
<thead>
<tr>
<th>Configuration</th>
<th>First</th>
<th>Second</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pp+</strong></td>
<td>0</td>
<td>1</td>
<td>&quot;21&quot;</td>
</tr>
<tr>
<td><strong>pn+</strong></td>
<td>0</td>
<td>1</td>
<td>&quot;22&quot;</td>
</tr>
<tr>
<td><strong>np+</strong></td>
<td>0</td>
<td>1</td>
<td>&quot;11&quot;</td>
</tr>
<tr>
<td><strong>nn+</strong></td>
<td>0</td>
<td>1</td>
<td>&quot;12&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Configuration</th>
<th>First</th>
<th>Second</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pp−</strong></td>
<td>0</td>
<td>1</td>
<td>&quot;12&quot;</td>
</tr>
<tr>
<td><strong>pn−</strong></td>
<td>0</td>
<td>1</td>
<td>&quot;11&quot;</td>
</tr>
<tr>
<td><strong>np−</strong></td>
<td>0</td>
<td>1</td>
<td>&quot;22&quot;</td>
</tr>
<tr>
<td><strong>nn−</strong></td>
<td>0</td>
<td>1</td>
<td>&quot;21&quot;</td>
</tr>
</tbody>
</table>
## Multiplexer transitions table

### Gated inverters

<table>
<thead>
<tr>
<th>sel = 0</th>
<th>sel = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>!y</td>
<td>!y</td>
</tr>
<tr>
<td>a → 1</td>
<td>b → 1</td>
</tr>
</tbody>
</table>

### Same in both implementations

- $a = b = 0$
  - sel = 0 → 1
  - sel = 1 → 0
- $a = b = 1$
  - sel = 0 → 1
  - sel = 1 → 0

### CMOS logic

- $a = 0, b = 1$
  - !y
  - !y
  - b = 0 → 1
- $a = 1, b = 0$
  - !y
  - !y
  - a = 0 → 1

### Different in each implementation

- Glitch!