Modeling an Asynchronous Circuit
Dedicated to the Protection Against Physical Attacks

Radu Mateescu
Wendelin Serwe
Aymane Bouzafour
Marc Renaudin
Secure microcontrollers

Asynchronous circuit on top of a circuit to protect against physical attacks (wire cuts, short-circuits, ...)

**Patented** by Tiempo (Renaudin, Folco, Boubkar)
FR 3 054 344 (July 25, 2016)

Series of sequencers

Idea: physical attacks yields different behaviour (e.g., deadlock)
Asynchronous Circuits

- No global clock
  - on-demand operation
  - Handshake communication (request/acknowledgement)

Advantages
- Low power consumption
- Harmonious electromagnetic emissions
- Better timing performance

Suitably modelled in process calculi

Here: LNT/CADP

https://cadp.inria.fr
Sequencer: Expected Behavior

process PROTOCOL [R\text{PRED}, A\text{PRED}, R\text{SUCC}, A\text{SUCC}: LINK] is
loop
\begin{align*}
    R\text{PRED} & (\text{UP}); \\
    A\text{SUCC} & (\text{UP}); \\
    A\text{SUCC} & (\text{DOWN}); \\
    R\text{PRED} & (\text{DOWN});
\end{align*}
end loop
end process

\begin{itemize}
    \item type VOLTAGE is DOWN, UP
    \item channel LINK is (VOLTAGE)
\end{itemize}
Plan

- Circuit-level modelling
  - Serial composition of sequencers
  - Attack analysis
- Gate-level modelling
  - Detailed analysis of a single sequencer
  - Exploration of various modelling styles
- Conclusion
Circuit-Level: Series of Sequencers

- Sequential composition: \( \text{pipe} \ (C_1, C_2) \)
  
  \[
  \text{hide } R, A \text{ in } \text{par } R, A \text{ in } \\
  \text{rename } R_{\text{SUCC}} \rightarrow R, A_{\text{SUCC}} \rightarrow A \text{ in } C_1 \text{ end rename } \\
  || \text{ rename } R_{\text{PRED}} \rightarrow R, A_{\text{PRED}} \rightarrow A \text{ in } C_2 \text{ end rename } \\
  \text{end par}
  \]

- \( \text{pipe} \ (\text{PROTOCOL, PROTOCOL}) \equiv \text{PROTOCOL} \)

- Extension (par induction)
  
  \[
  \begin{align*}
  \text{pipe}^0 (\text{PROTOCOL}) &= \text{PROTOCOL} \\
  \text{pipe}^{n+1} (\text{PROTOCOL}) &= \text{pipe} (\text{pipe}^n (\text{PROTOCOL}), \text{PROTOCOL}) \\
  \text{property: } \text{pipe}^n (\text{PROTOCOL}) &\equiv \text{PROTOCOL}
  \end{align*}
  \]
Modelling an Attack (2 Sequencers)

- Examples: wire-cut, stuck-at
- Add constraints with a multiway rendezvous (e.g., parallel composition with stop for wire-cut)
- Synchronize with (i.e., enforce constraints on)
  - both sequencers
  - only the receiving sequencer (left for A, right for R) (i.e., unconstrained outputs)
Modelling an Attack (>2 Sequencers)

- Example: short-circuit
- Dedicated additional gate
- Non-deterministic choice for disagreeing voltages
- Synchronization vectors (EXP)
  - No synchronization for outputs
  - Synchronization for unmodified wires
  - 3-party synchronization for modified wires
Modelling a Short-Circuit $A_1$-$A_2$

hide $R_1, A_2, R_2, A_2, R_1A_2$ in label par using

|-- synchronization vectors for unmodified wires
|-- synchronization vectors for the short-circuit

in

rename $R_{SUCC} \rightarrow R_1, A_{SUCC} \rightarrow A_1$ in $C_1$ end rename
|| rename $R_{PRED} \rightarrow R_1, A_{PRED} \rightarrow A_1, R_{SUCC} \rightarrow R_2, A_{SUCC} \rightarrow A_2$ in $C_2$
end rename
|| rename $R_{PRED} \rightarrow R_2, A_{PRED} \rightarrow A_2$ in $C_3$ end rename
end par
Attack Detection Results

- Check inclusion (equivalence/preorder) model < attack-model
- **Stuck-At**: All attacks detected
- **Wire-Cut**
  - Attack undetected for two unconstrained sides *(unrealistic)*
  - All other attacks detected
- **Short-Circuit** (3 sequencers)
  - Attacks detected for $R_1 R_2$, $R_1 A_2$, $A_1 A_2$, $A_1 R_2$
  - Attacks undetected for $R_1 A_1$, $R_2 A_2$ (scenario for 2 sequencers) *(shortened shield, impossible due to physical chip layout)*
- Results extended to sequences of arbitrary length
Gate-Level Analysis
Sequencer: Patented Implementation

(ignore RST)
(rename visible gates)
Modelling Wires

- No delay: (binary) rendezvous
  Wire as LNT gate

- With delay: dedicated process

```plaintext
process WIRE [INPUT, OUTPUT: LINK] is
  var X: VOLTAGE in
  loop
    INPUT (?X);
    OUTPUT (X)
  end loop
  end var
end process
```
Modelling Forks

- No delay: multiway rendezvous (*isochronic*)
- With delay: dedicated process
  - **WIRE**: isochronic (synchronised outputs)
  - **FORK**: parallel (unsynchronised outputs)

```plaintext
process FORK [INPUT, OUTPUT1, OUTPUT2: LINK] is
  var X: VOLTAGE in
  loop
    INPUT (?X);
    par OUTPUT1 (X) || OUTPUT2 (X) end par
  end loop
end var
end process
```
Modelling Variants for a Sequencer

- Depending on the models of wires and forks
- Depending on isochrony of forks
- Sequencer: 3 forks
- Code
  - I: isochronic
  - P: parallel
Modelling a Sequencer (Rendezvous)

process $SEQ_{RV}$ [$R_{PRED}$, $A_{PRED}$, $R_{SUCC}$, $A_{SUCC}$: LINK]
  ($X_1$, $X_2$, $INIT_C$: VOLTAGE) is

hide $G$, $H$: LINK in

par
  $R_{PRED}$, $A_{SUCC}$, $G$ ->
  MULLER [$R_{PRED}$, $A_{SUCC}$, $G$] ($X_1$, $X_2$, $INIT_C$)
  || $R_{PRED}$, $H$ -> AND [$R_{PRED}$, $H$, $R_{SUCC}$] ($X_1$, NOT ($INIT_C$))
  || $G$, $H$ -> INV [$G$, $H$] ($INIT_C$)
  || $A_{SUCC}$, $H$ -> NOR [$A_{SUCC}$, $H$, $A_{PRED}$] ($X_2$, NOT ($INIT_C$))
end par
end process

Isochronic forks only
Modelling a Sequencer (IIP)

process \( \text{SEQ}_{\text{IIP}} [R_{\text{PRED}}, A_{\text{PRED}}, R_{\text{SUCC}}, A_{\text{SUCC}} : \text{LINK}] \)
\[
(X_1, X_2, \text{INIT}_C : \text{VOLTAGE}) \text{ is }
\]
hide \( G, H, R_{\text{PRED2}}, A_{\text{SUCC2}}, G_2, H_1, H_2 : \text{LINK} \) in
par
\[
\begin{align*}
R_{\text{PRED}} &\rightarrow M\text{ULLER} [R_{\text{PRED2}}, A_{\text{SUCC2}}, G] (X_1, X_2, \text{INIT}_C) \\
R_{\text{PRED}} &\rightarrow \text{AND} [R_{\text{PRED}}, H_1, R_{\text{SUCC}}] (X_1, \text{NOT} (\text{INIT}_C)) \\
G_2 &\rightarrow \text{INV} [G_2, H] (\text{INIT}_C) \\
A_{\text{SUCC2}} &\rightarrow \text{NOR} [A_{\text{SUCC2}}, H_2, A_{\text{PRED}}] (X_2, \text{NOT} (\text{INIT}_C)) \\
R_{\text{PRED2}} &\rightarrow \text{WIRE} [R_{\text{PRED}}, R_{\text{PRED2}}] -- \text{isochronic fork X} \\
A_{\text{SUCC2}} &\rightarrow \text{WIRE} [A_{\text{SUCC}}, A_{\text{SUCC2}}] -- \text{isochronic fork Y} \\
H, H_1, H_2 &\rightarrow \text{FORK} [H, H_1, H_2] -- \text{parallel fork Z} \\
G, G_2 &\rightarrow \text{WIRE} [G, G_2]
\end{align*}
\]
end par end hide
end process
Modelling Gates

- State-based
  - Focus on voltage state
  - Seemingly intuitive
  - Represent subtle semantic differences
  - Several variants

- Transition-based
  - Focus on voltage change
  - Strong assumption: strict alternation
  - **Efficient**: smaller models, smaller state space
Transition-based AND

process AND [INPUT1, INPUT2, OUTPUT: LINK]
    (in var X1, X2: VOLTAGE) is

    var RESULT: VOLTAGE in
    RESULT := X1 AND X2;
    loop
        select INPUT1 (?X1) [] INPUT2 (?X2) end select;
        if RESULT != (X1 AND X2) then
            RESULT := X1 AND X2;
            OUTPUT (RESULT)
        end if
    end loop
end var
end process

Hypothesis: inputs alternate strictly ensure strict alternation on outputs
Intuitive State-based AND

process AND [INPUT1, INPUT2, OUTPUT: LINK]
  (in var X1, X2: VOLTAGE) is

  loop
    select -- accept some input
      INPUT1 (?X1)
      INPUT2 (?X2)
    end select;
    OUTPUT (X1 AND X2)
  end loop
end process

What about (quasi) simultaneous inputs?
State-based AND

process AND [INPUT1, INPUT2, OUTPUT: LINK]
  (in var X1, X2: VOLTAGE) is
  loop
    select -- accept one or two inputs in arbitrary order
      INPUT1 (?X1);
      select null [] INPUT2 (?X2) end select
      [] INPUT2 (?X2)
      select null [] INPUT1 (?X1) end select
    end select;
    OUTPUT (X1 and X2)
  end loop
end process
Parallel State-based AND

process AND [INPUT1, INPUT2, OUTPUT: LINK]
  (in var X1, X2: VOLTAGE) is

loop
  par -- accept zero, one, or two inputs in arbitrary order
    select null [] INPUT1 (?X1) end select
    | | select null [] INPUT2 (?X2) end select
  end par;
  OUTPUT (X1 and X2)
end loop
end process

might generate outputs (not triggered by an input)
# Gate-Level State Spaces

<table>
<thead>
<tr>
<th>model</th>
<th>forks</th>
<th>1 sequencer</th>
<th>series of 2 sequencers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>states</td>
<td>transitions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>states</td>
<td>transitions</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>intuitive</td>
<td>RV</td>
<td>90</td>
<td>222</td>
</tr>
<tr>
<td>IIP</td>
<td>6,124</td>
<td>1,307,889</td>
<td>5,968,266</td>
</tr>
<tr>
<td>PII</td>
<td>6,475</td>
<td>1,562,907</td>
<td>6,452,280</td>
</tr>
<tr>
<td>state</td>
<td>RV</td>
<td>766</td>
<td>2,406</td>
</tr>
<tr>
<td>IIP</td>
<td>86,846</td>
<td>3,002,896,049</td>
<td>18,494,246,894</td>
</tr>
<tr>
<td>PII</td>
<td>82,041</td>
<td>2,795,890,977</td>
<td>15,509,939,437</td>
</tr>
<tr>
<td>parallel</td>
<td>RV</td>
<td>916</td>
<td>3,404</td>
</tr>
<tr>
<td>IIP</td>
<td>768</td>
<td>589,440</td>
<td>6,741,584</td>
</tr>
<tr>
<td>PII</td>
<td>764</td>
<td>582,224</td>
<td>6,485,364</td>
</tr>
<tr>
<td>transition</td>
<td>RV</td>
<td>34</td>
<td>112</td>
</tr>
<tr>
<td>IIP</td>
<td>1,320</td>
<td>238,811</td>
<td>1,270,154</td>
</tr>
<tr>
<td>PII</td>
<td>952</td>
<td>135,814</td>
<td>666,185</td>
</tr>
</tbody>
</table>

|                |        | 790         | yes        |
|                |        | 5,968,266   | yes        |
|                |        | 6,452,280   | yes        |
|                |        | 906,342     | no         |
|                |        | 18,494,246,894| no       |
|                |        | 15,509,939,437| no      |
|                |        | 1,625,792   | no         |
|                |        | 6,741,584   | no         |
|                |        | 6,485,364   | no         |
|                |        | 1,101       | no         |
|                |        | 1,270,154   | no         |
|                |        | 666,185     | yes        |
Gate-Level Analysis Results

- Explicit modelling wires yields larger state spaces
- Transition-based models are smaller
- Deadlocks
  - Intuitive model: inappropriate
  - Transition-based model: pinpoint isochronic forks
- Series of two sequencers not equal to a sequencer
- With stubs: pinpoint isochronic forks by checking equivalence with the protocol (see the paper)
Conclusion

- Circuit- and gate-level modelling of asynchronous circuits
- Valuable feedback for designer (isochronic forks)
- Circuit-level attack analysis for series of arbitrary length (inductive reasoning)
- Interesting benchmark
  - MARS 2020 model repository [http://mars-workshop.org/repository/022-Shield.html](http://mars-workshop.org/repository/022-Shield.html)
- Challenges
  - Inductive reasoning for gate-level models
  - Modeling faulty gates / probabilistic analysis