Modeling an Asynchronous Circuit Dedicated to the Protection Against Physical Attacks

Radu Mateescu

Wendelin Serwe

Aymane Bouzafour

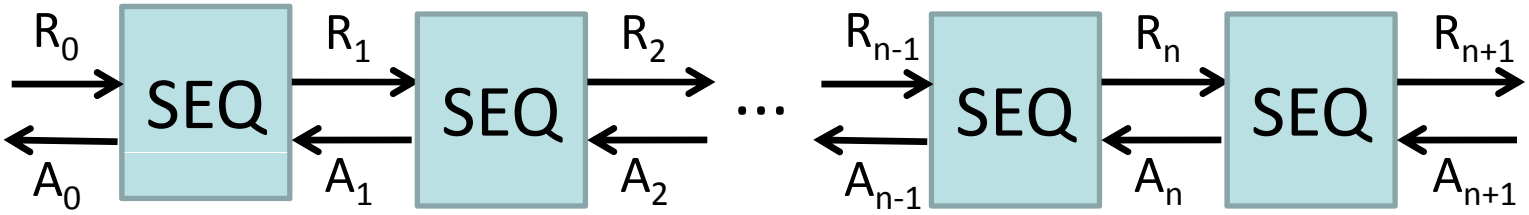
Marc Renaudin

Context

- Secure microcontrollers

SECURIOT
SECURITE POUR L INTERNET DES OBJETS

- 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_{PRED}, A_{PRED}, R_{SUCC}, A_{SUCC}: LINK] **is**

loop

R_{PRED} (UP); R_{SUCC} (UP);

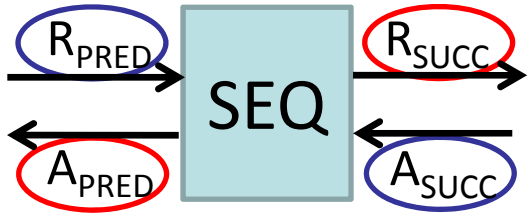
A_{SUCC} (UP); R_{SUCC} (DOWN);

A_{SUCC} (DOWN); A_{PRED} (UP);

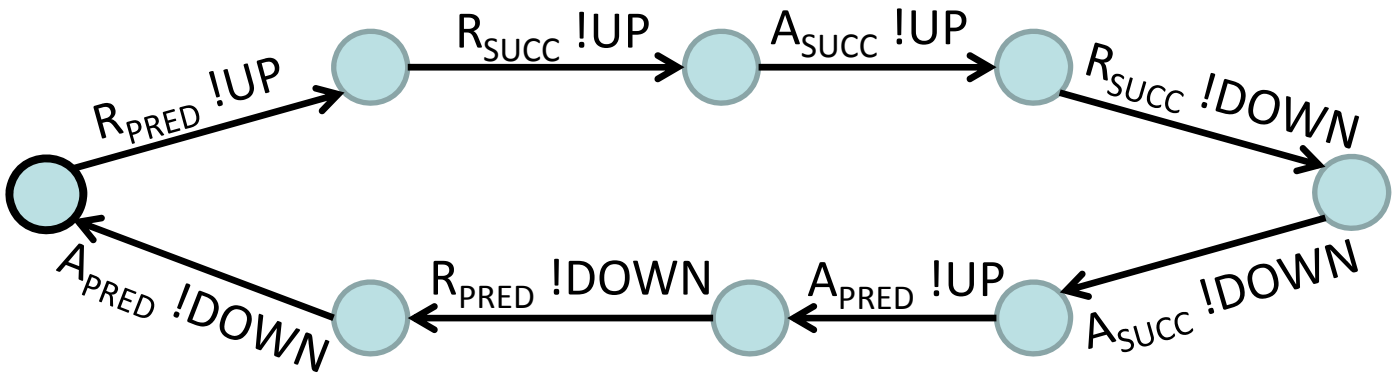
R_{PRED} (DOWN); A_{PRED} (DOWN)

end loop

end process



inputs: R_{PRED}, A_{SUCC}
outputs: A_{SUCC}, R_{PRED}



type VOLTAGE **is** DOWN, UP **end type**

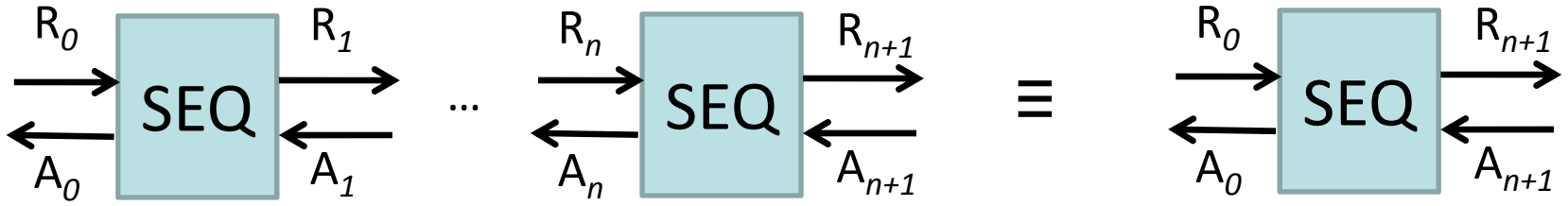
channel LINK **is** (VOLTAGE) **end channel**

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

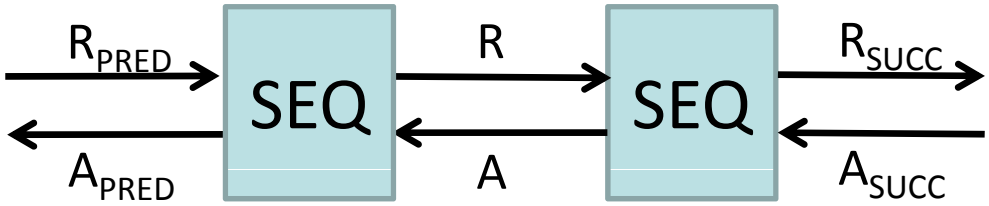
Circuit-Level: Series of Sequencers

- Sequential composition: *pipe* (C_1, C_2)
  hide R, A **in par** R, A **in**
  rename R_{SUCC} -> R, A_{SUCC} -> A **in** C_1 **end rename**
  || **rename** R_{PRED} -> R, A_{PRED} -> A **in** C_2 **end rename**
  end par
- *pipe* (PROTOCOL, PROTOCOL) ≡ PROTOCOL
- Extension (par induction)
  - $pipe^0$ (PROTOCOL) = PROTOCOL
  - $pipe^{n+1}$ (PROTOCOL) = *pipe* ($pipe^n$ (PROTOCOL), PROTOCOL)
  - property: $pipe^n$ (PROTOCOL) ≡ PROTOCOL



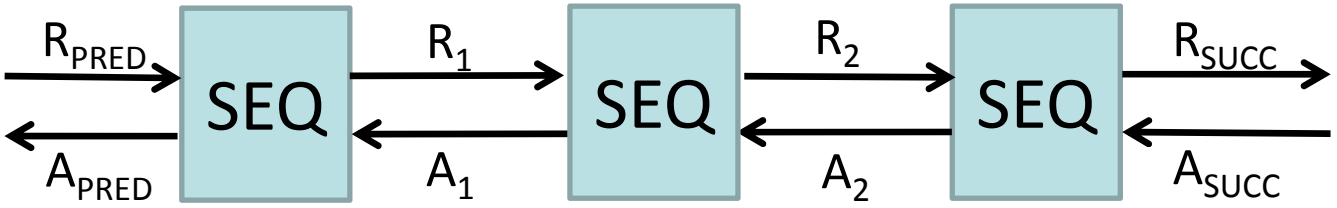
Modelling an Attack (2 Sequencers)

- 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} -> R_1, A_{SUCC} -> A_1 **in** C_1 **end rename**

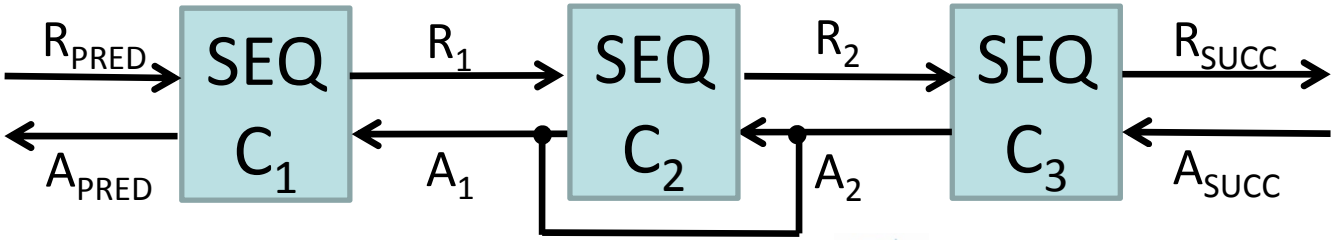
|| **rename** R_{PRED} -> R_1, A_{PRED} -> A_1, R_{SUCC} -> R_2, A_{SUCC} -> A_2 **in**

C_2

end rename

|| **rename** R_{PRED} -> R_2, A_{PRED} -> A_2 **in** C_3 **end rename**

end par

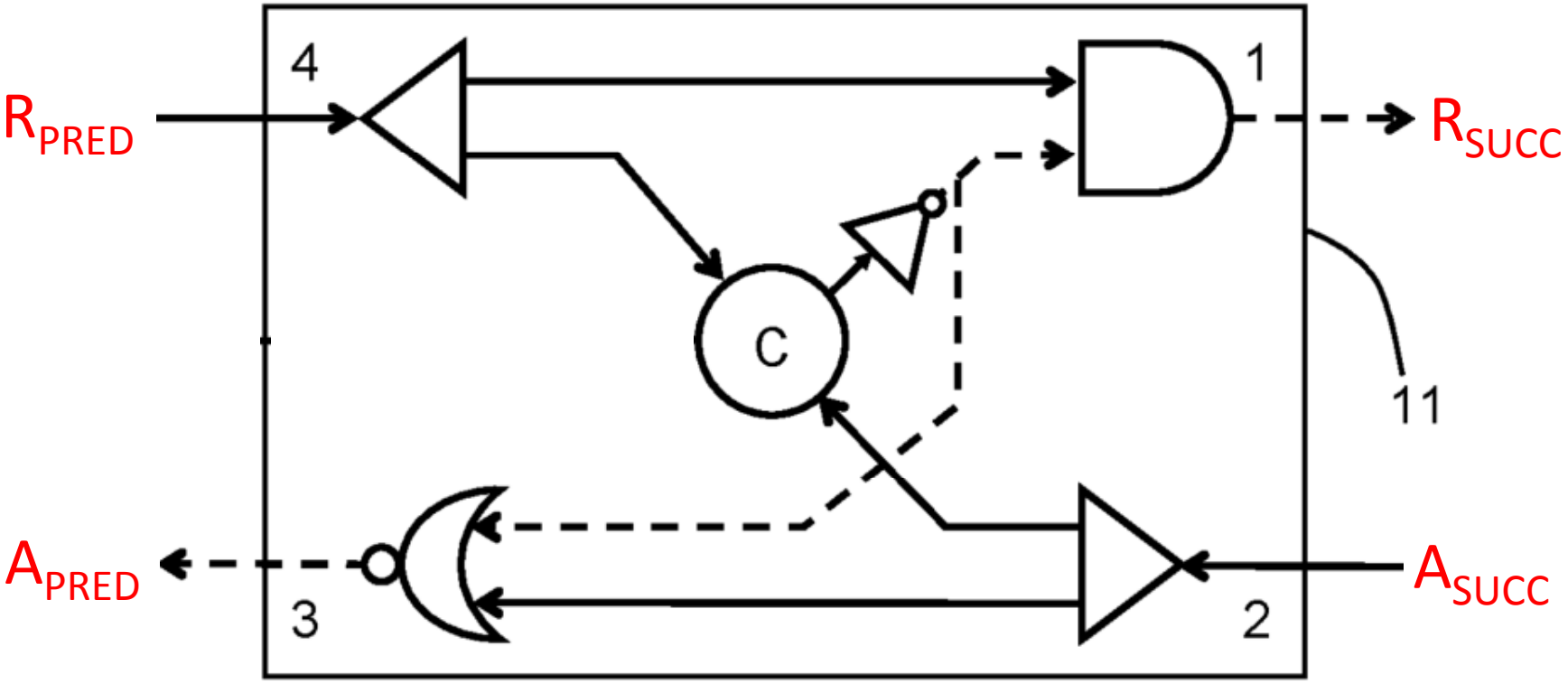


Attack Detection Results

- 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_1R_2, R_1A_2, A_1A_2, A_1R_2
  - Attacks undetected for R_1A_1, R_2A_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

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

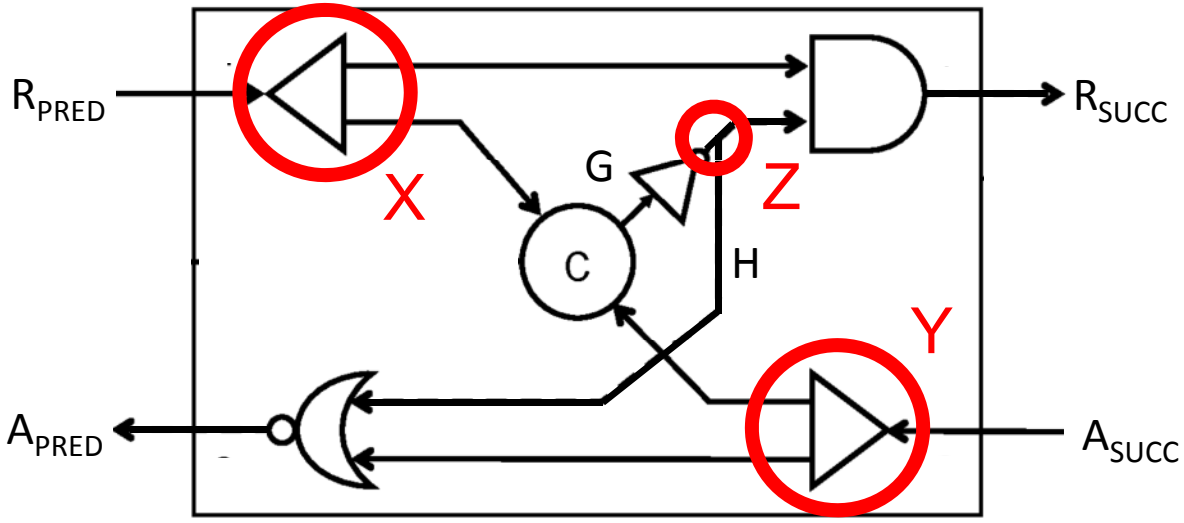
Modelling Forks

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

```
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

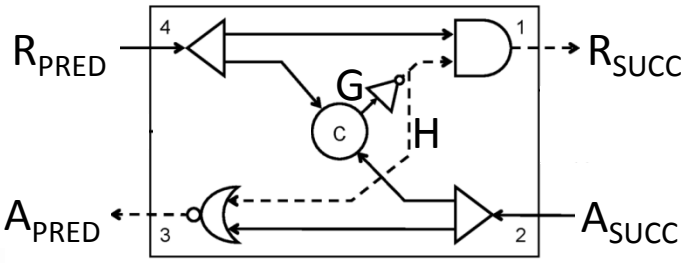
- Depending on the models of wires and forks
- Depending on isochrony of forks
- Sequencer: 3 forks
- Code
  - I: isochronic
  - P: parallel



Modelling a Sequencer (Rendezvous)

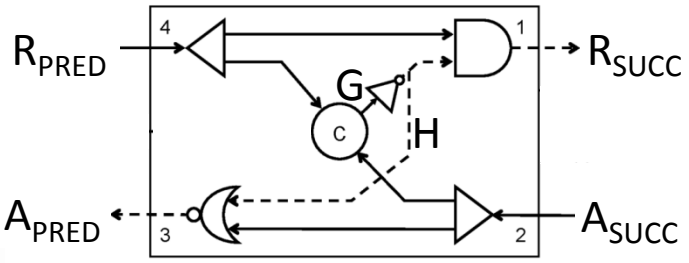
process SEQ_{RV} [R_{PRED}, A_{PRED}, R_{SUCC}, A_{SUCC}: LINK]
(X_1, X_2, $INIT_C$: VOLTAGE) **is**
hide G, H: LINK **in**
par
R_{PRED}, A_{SUCC}, G ->
MULLER [R_{PRED}, A_{SUCC}, G] (X_1, X_2, $INIT_C$)
|| R_{PRED}, H -> AND [R_{PRED}, H, R_{SUCC}] (X_1, NOT ($INIT_C$))
|| G, H -> INV [G, H] ($INIT_C$)
|| 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 SEQ_{IIP} [R_{PRED}, A_{PRED}, R_{SUCC}, A_{SUCC}: LINK]
(X_1, X_2, $INIT_C$: VOLTAGE) **is**
hide G, H, R_{PRED2}, A_{SUCC2}, G_2, H_1, H_2: LINK **in**
par
R_{PRED}, A_{SUCC}, G -> MULLER [R_{PRED2}, A_{SUCC2}, G] (X_1, X_2, $INIT_C$)
|| R_{PRED}, H_1 -> AND [R_{PRED}, H_1, R_{SUCC}] (X_1, NOT ($INIT_C$))
|| G_2, H -> INV [G_2, H] ($INIT_C$)
|| A_{SUCC2}, H_2 -> NOR [A_{SUCC2}, H_2, A_{PRED}] (X_2, NOT ($INIT_C$))
|| R_{PRED2} -> WIRE [R_{PRED}, R_{PRED2}] *-- isochronic fork X*
|| A_{SUCC2} -> WIRE [A_{SUCC}, A_{SUCC2}] *-- isochronic fork Y*
|| H, H_1, H_2 -> FORK [H, H_1, H_2] *-- parallel fork Z*
|| G, G_2 -> WIRE [G, G_2]
end par end hide
end process

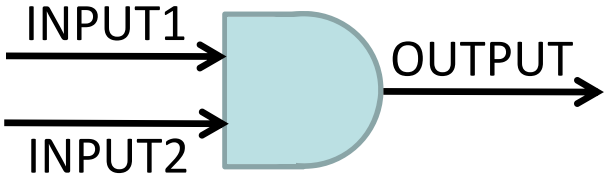


Modelling Gates

- 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
```

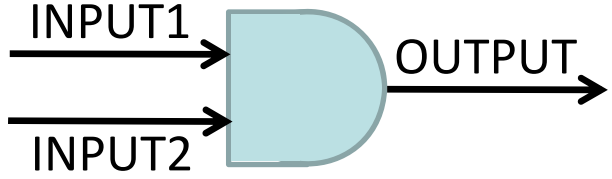


Hypothesis: inputs alternate strictly

ensure strict alternation on outputs

Intuitive State-based AND

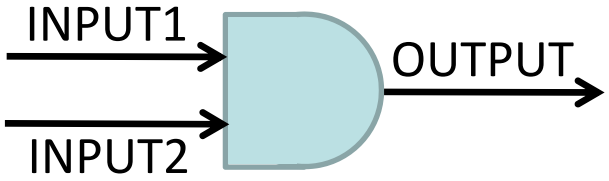
```
process AND [INPUT1, INPUT2, OUTPUT: LINK]
                    (in var X1, X2: VOLTAGE) is
   loop
      select -- accept some input
         INPUT1 (?X1)
      [] INPUT2 (?X2)
      end select;
      OUTPUT (X1 AND X2)
   end loop
end process
```



What about (quasi) simultaneous inputs?

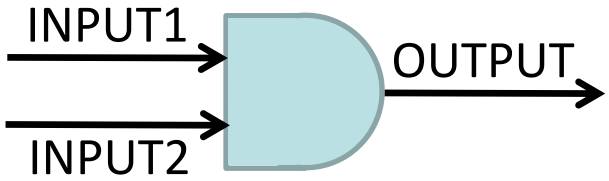
State-based AND

```
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

model	forks	1 sequencer		series of 2 sequencers		
		states	transitions	states	transitions	deadlock
intuitive	RV	90	222	308	790	yes
	IIP	6,124	21,454	1,307,889	5,968,266	yes
	PII	6,475	19,985	1,562,907	6,452,280	yes
state	RV	766	2,406	230,906	906,342	no
	IIP	86,846	374,292	3,002,896,049	18,494,246,894	no
	PII	82,041	315,312	2,795,890,977	15,509,939,437	no
parallel	RV	916	3,404	341,674	1,625,792	no
	IIP	768	5,544	589,440	6,741,584	no
	PII	764	5,306	582,224	6,485,364	no
transition	RV	34	112	279	1,101	no
	IIP	1,320	4,870	238,811	1,270,154	no
	PII	952	3,155	135,814	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
  - MCC 2020: 6 surprise models https://mcc.lip6.fr/models.php
- Challenges
  - Inductive reasoning for gate-level models
  - Modeling faulty gates / probabilistic analysis