Introduction to GSTE

Jin Yang, Ph.D.
Principal Research Scientist
Validation Research Lab
Microprocessor Technology Labs
Corporate Technology Group

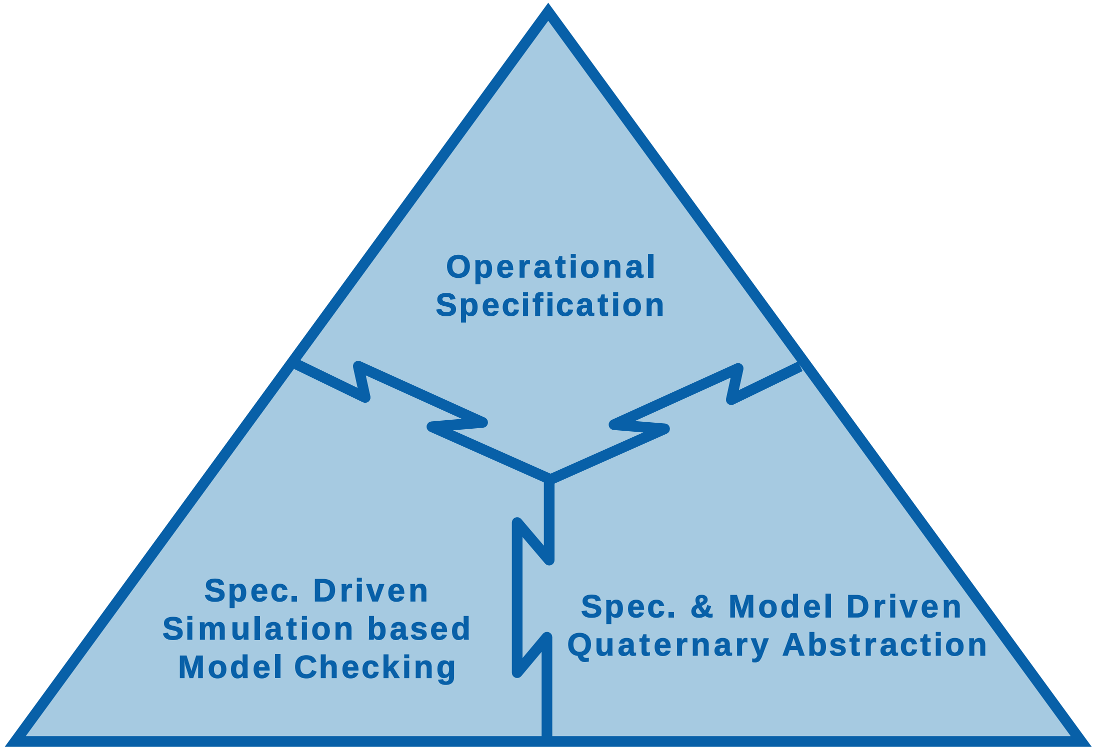
ATVA 2006, Beijing, China

Outline

- Background
- Circuit Model and Assertion Language
- STE
- GSTE
- GSTE for Concurrent Hardware
- Symbolic Simulation
- Quaternary Abstraction
- Conclusion

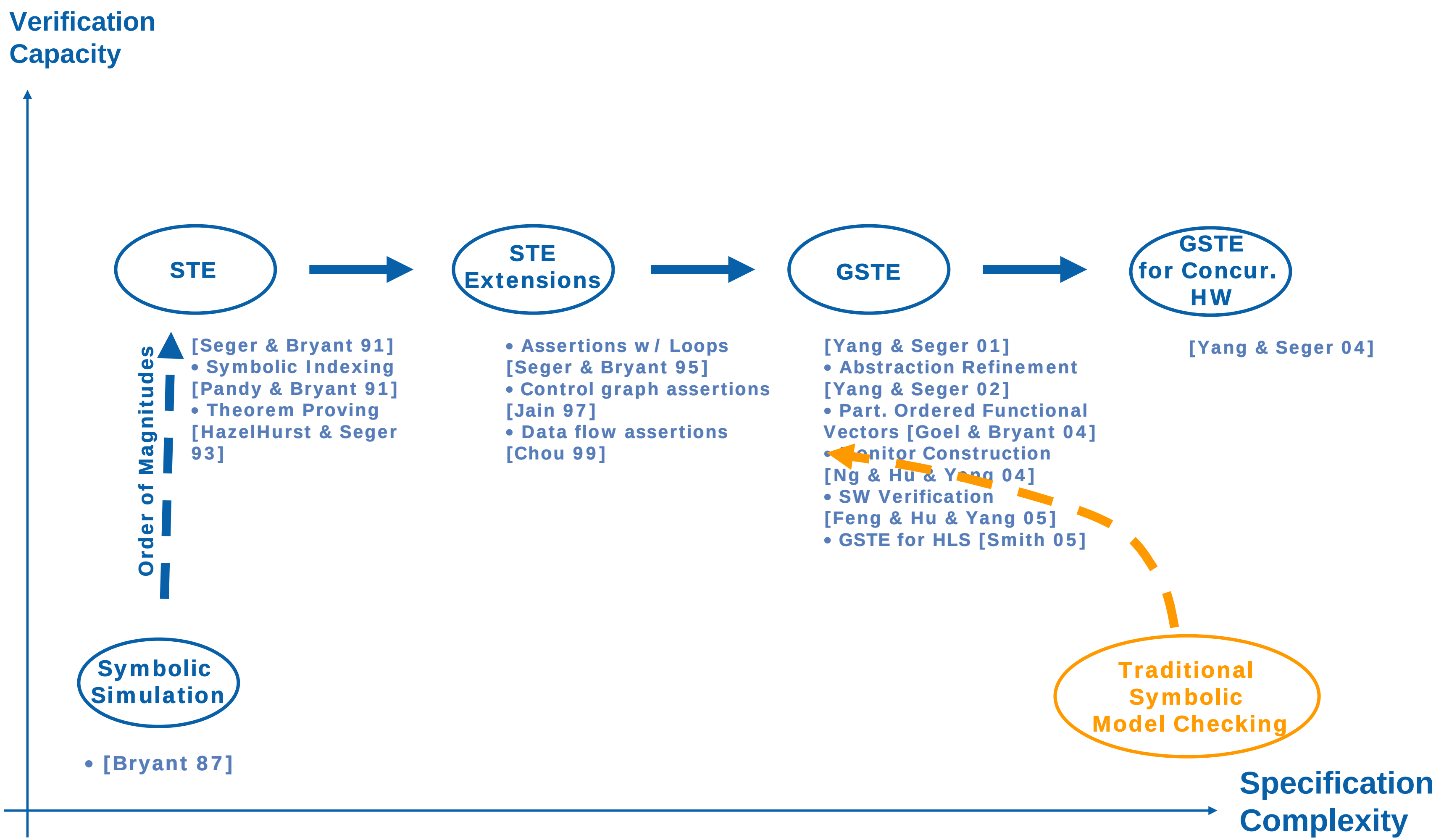
What Is GSTE?

Generalized Symbolic Trajectory Evaluation

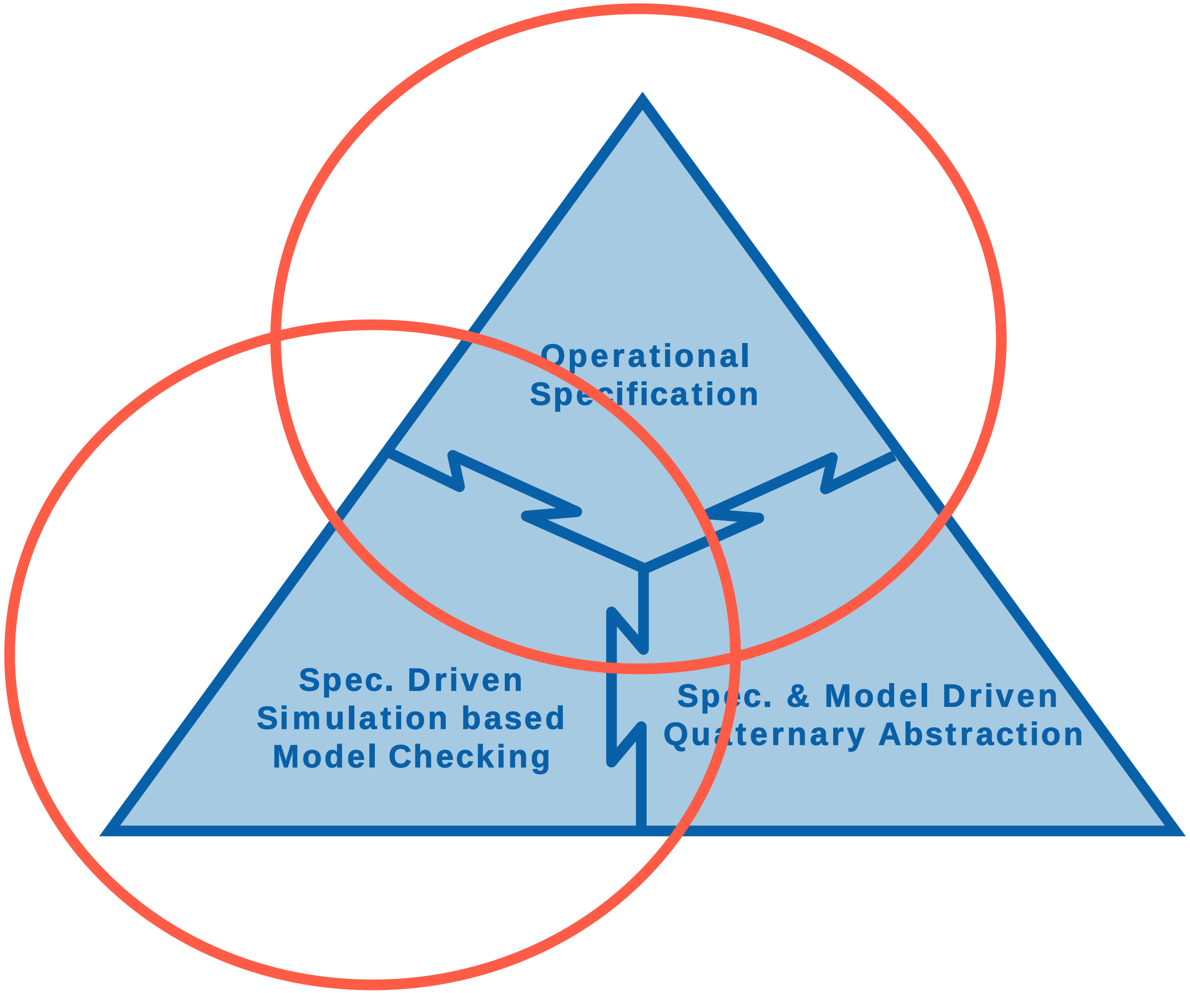


A system used by FVers since 2000 on verifying
Intel μ-processors with thousands of state elements

Historical Perspective



Assertion Languages and Model Checkings



Outline

- Background
- Circuit Model and Assertion Language
- STE
- GSTE
- GSTE for Concurrent Hardware
- Symbolic Simulation
- Quaternary Abstraction
- Conclusion

Circuit Model

$$M = (I, L, N; O, F)$$

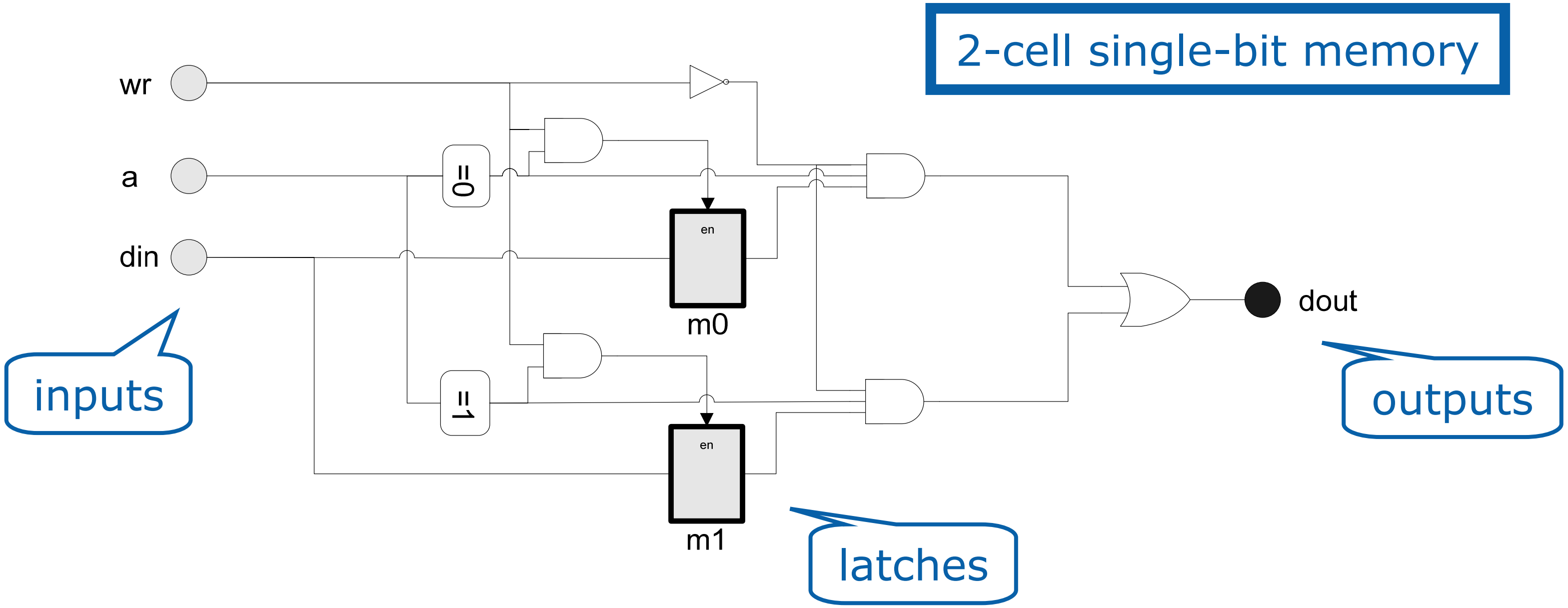
- I – vector of Boolean input nodes
- L – vector of Boolean latch nodes
- N – set of Boolean next state functions

$$n_l(I, L) \text{ for each latch node } l \in L$$

- O – vector of Boolean output nodes
- F – set of Boolean output functions

$$f_o(I, L) \text{ for each output node } o \in O$$

Circuit Model: Example



- Next state functions
  - m0: ((wr & !a) -> din) & ((!wr | a) -> m0)
  - m1: ((wr & a) -> din) & ((!wr | !a) -> m1)
- Output functions
  - dout = !wr & !a & m0 + !wr & a & m1

Circuit Model - Semantics

$$M = (I, L, N; O, F)$$

- State

$$s$$

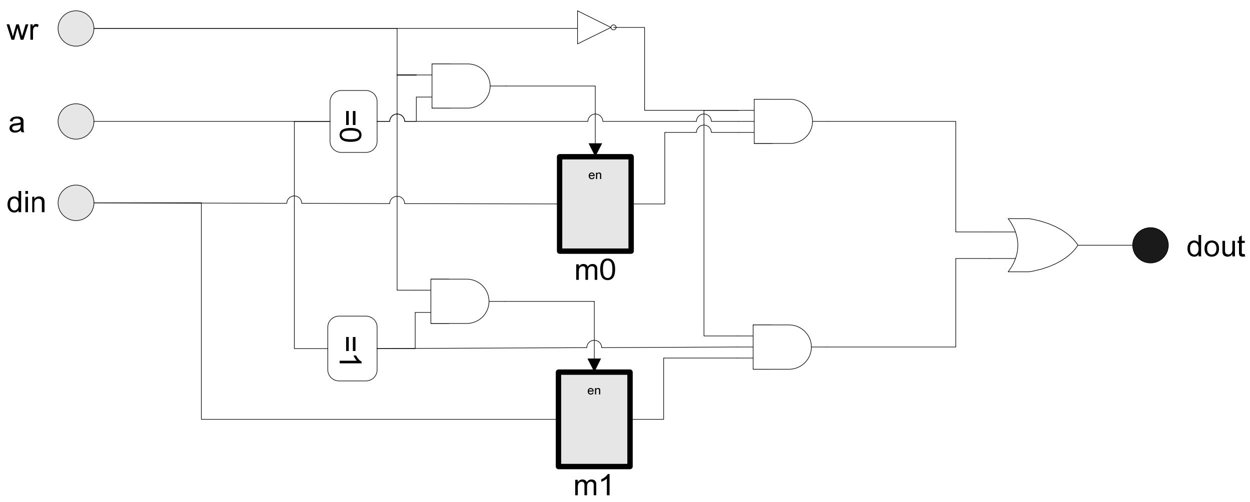
  - A Boolean assignment to elements in I and L
  - $s(o) = f_0(s(I), s(L))$ for each $o \in O$

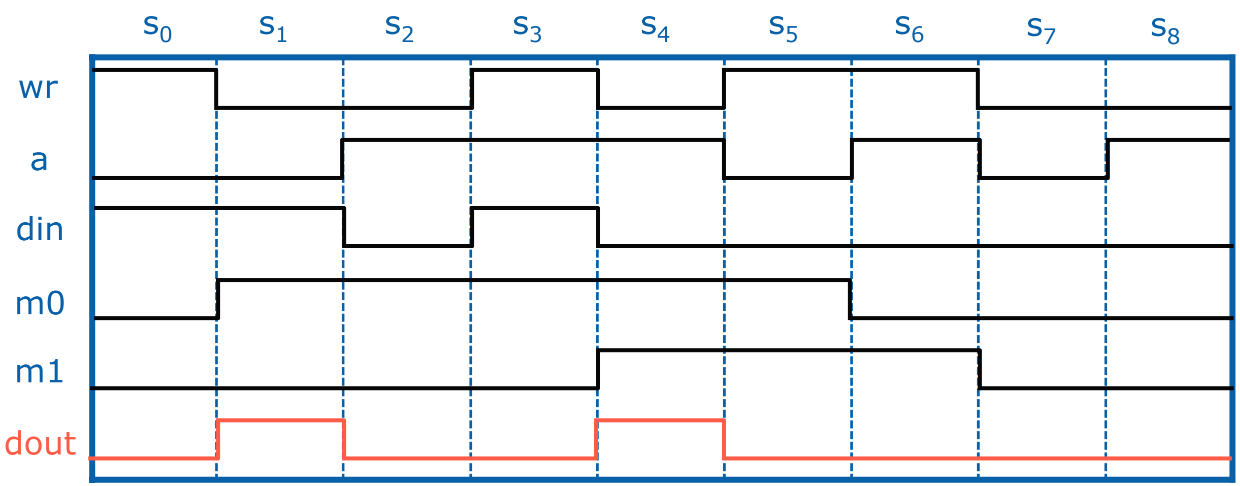
- Trace (waveform)

$$\tau = s_0\ s_1\ s_2\ \ldots$$

  - For all $i \geq 0$, $l \in L$, $s_{i+1}(l) = n_l(s_i(I), s_i(L))$
  - Infinite
  - No initial condition, and therefore any suffix of τ is also a trace
  - $\Gamma(M)$ – set of all traces in M

Circuit Model - Semantics: Example





Assertion Language

- Set of all Predicates over I, L, O and Z

$$P$$

  - Z – vector of rigid Boolean variables (symbolic constants)
    - BV_Z denotes a Boolean value assignment to Z
- Assertion Alphabet

$$\Sigma = \{ (a, c) \mid a, c \in P \}$$

  - a – antecedent
  - c – consequent
- Assertion Language
  - $A \subseteq \Sigma^*$ - (finite) assertion language
    - $w \in A$ – assertion word
  - $A^\omega \subseteq \Sigma^\omega$ - (infinite) ω-assertion language
    - $w \in A^\omega$ – ω-assertion word

Assertion Language - Semantics

- Trace Language
  - State sequence $\pi = s_0\ s_1\ s_2 \ldots$ satisfies word $w = (a_0,c_0)\ (a_1,c_1) \ldots (a_{k-1},c_{k-1})$

$$\pi \models w$$

if $\forall\ BV_Z,\ (\wedge_{0 \le i < k}\ a_i(s_i(I),\ s_i(L),\ s_i(O),\ BV_Z)) \Rightarrow (\wedge_{0 \le i < k}\ c_i(s_i(I),\ s_i(L),\ s_i(O),\ BV_Z))$.

"if all the antecedents are satisfied, then all the consequents must be satisfied"

  - Trace language of assertion word w:

$$\Gamma(w) = \{\pi \mid \pi \models w\}$$

  - Trace language of assertion language A:

$$\Gamma(A) = \cap_{w \in A}\ \Gamma(w)$$

  - Theorem: ("more words ⇒ more restricted behavior")

$$A_1 \subseteq A_2 \Rightarrow \Gamma(A_2) \subseteq \Gamma(A_1)$$

- Model satisfiablity

$$M \models A$$

if $\Gamma(M) \subseteq \Gamma(A)$

- ω-Semantics can be similarly defined

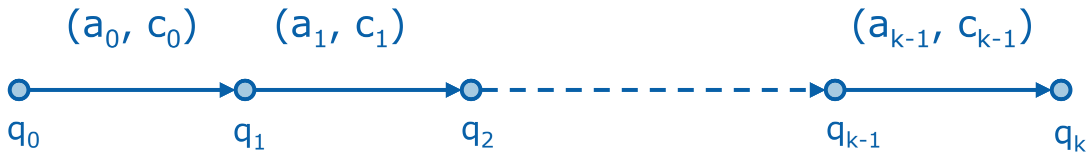
Outline

- Background
- Circuit Model and Assertion Language
- STE
- GSTE
- GSTE for Concurrent Hardware
- Symbolic Simulation
- Quaternary Abstraction
- Conclusion

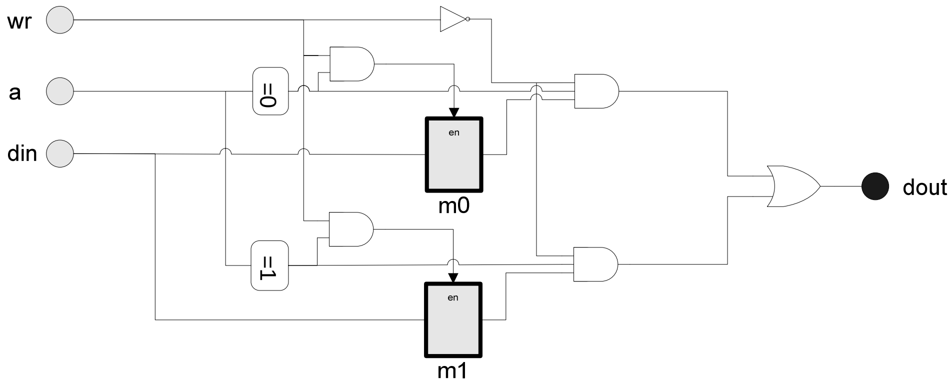
STE Assertion

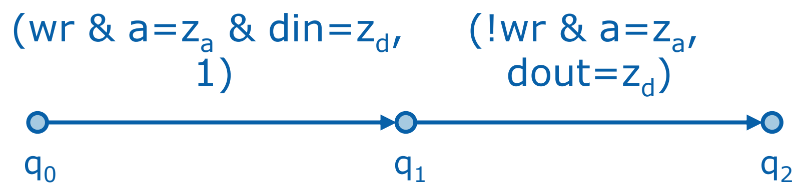
- STE Assertion – prefix closure of an assertion word

$$A = \{\varepsilon, (a_0,c_0), (a_0,c_0)\,(a_1,c_1), \ldots, (a_0,c_0)\,(a_1,c_1) \ldots (a_{k-1},c_{k-1})\}$$



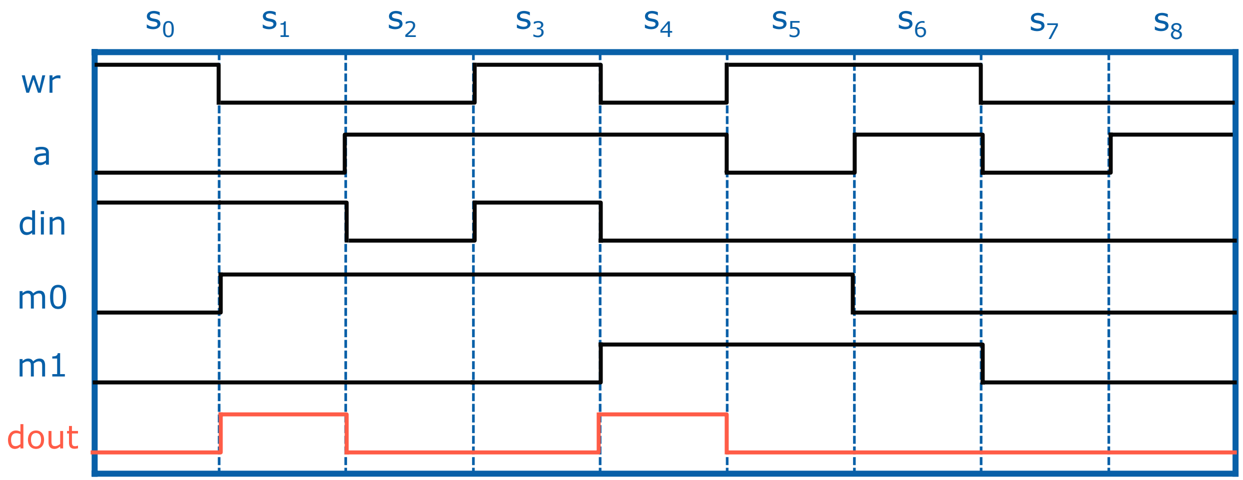
STE Assertion: Example





"If a value is written to a memory cell,
then the read from the cell immediately after will return the value."

STE Assertion Satisfiability: Example



$$\models$$

(wr & a=z_a & din=z_d, 1) (!wr & a=z_a, dout=z_d)

1) $z_a=0, z_d=0$: (wr & !a & !din, 1) (!wr & !a, !dout)

2) $z_a=0, z_d=1$: (wr & !a & din, 1) (!wr & !a, dout)

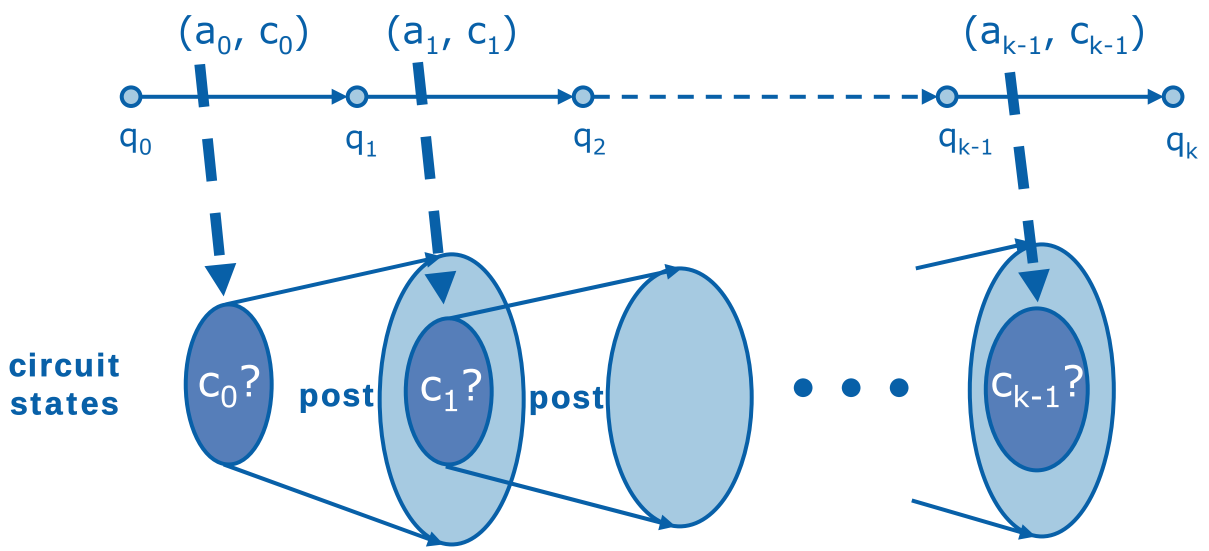
3) $z_a=1, z_d=0$: (wr & a & !din, 1) (!wr & a, !dout)

4) $z_a=1, z_d=1$: (wr & a & din, 1) (!wr & a, dout)

STE Model Checking

- Post-Image Function

$$\text{post}(p(I,L,Z)) =$$
$$(\exists\, I,L: p(I,L,Z) \wedge (\wedge_{l \in L}\, l' = n_l(I,L)))\,[L/L']$$



Constrained reachability analysis

STE Model Checking (cont)

STEMC(M, A)
begin
ckt_stt$((q_0, q_1))$:= stt_pred(a_0, M);
for i = 1 to k-1 do
  if (!(ckt_stt$((q_{i-1}, q_i)) \Rightarrow_M c_{i-1}$))
    return 0;
  ckt_stt$((q_i, q_{i+1}))$:= post(ckt_stt$((q_{i-1}, q_i))$) $\wedge$ stt_pred(a_i, M);
endfor;
return (ckt_stt$((q_{k-1}, q_k)) \Rightarrow_M c_{k-1}$);
end.

Notes:
- stt_pred(a, M) := $\exists O: a \wedge (\wedge_{o \in O} o = f_o(I, L))$
- $p \Rightarrow_M c$:= $p \wedge (\wedge_{o \in O} o = f_o(I,L)) \Rightarrow c$

STE Model Checking (cont)

- Lemma (Constrained Forward Reachability)

  For all BV_Z,

  $$s \in ckt\_stt((q_{j-1}, q_j))\ [BV_Z/Z] \text{ iff}$$

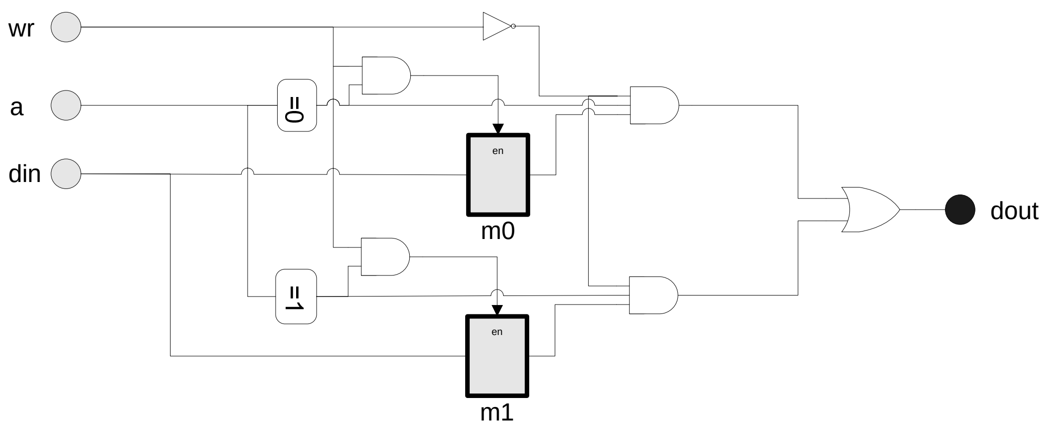
  $$\exists\ \tau = s_0\ s_1\ \ldots\ s_{j-1}=s\ \ldots,\ \text{s.t.},\ \forall 0 \le i < j,\ a_i(s_i(I), s_i(L), s_i(O), BV_Z)=1$$

  for all state s and all $0 \le j \le k$.

- Theorem

  $$STEMC(M, A)=1 \text{ iff } M \models A$$

STE Model Checking: Example



(wr & a=z_a & din=z_d, 1)

(!wr & a=z_a, dout=z_d)

q_0 → q_1 → q_2

post: (!z_a -> m0=z_d) & (z_a -> m1=z_d)

ckt_stt: wr & a=z_a & din=z_d → (!z_a -> m0=z_d) & (z_a -> m1=z_d) & !wr & a=z_a

Outline

- Background
- Circuit Model and Assertion Language
- STE
- GSTE
- GSTE for Concurrent Hardware
- Symbolic Simulation
- Quaternary Abstraction
- Conclusion

STE Limitation

- STE assertion only specifies a single pattern
  - E.g., (wr & $a=z_a$ & $din=z_d$, 1) (!wr & $a=z_a$, $dout=z_d$)
- In reality, (possibly infinitely) many patterns are needed for a complete specification
  - E.g., (wr & $a=z_a$ & $din=z_d$, 1) (!wr + $a!=z_a$, $1)^*$ (!wr & $a=z_a$, $dout=z_d$)

"After a value is written to a memory cell,
any read from the cell will return the value
as long as there is no other writes to the cell in-between."

GSTE Assertion

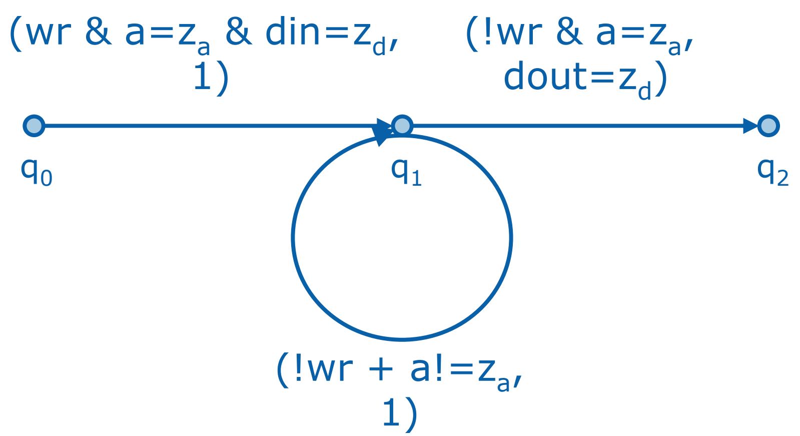
- Assertion Automaton

$$G = (\Sigma, Q, q_0, \Delta, T)$$

  - Σ - assertion alphabet
  - Q – a set of specification states
  - q_0 – the initial state
  - $\Delta \subseteq Q \times \Sigma \times Q$ – state transition relation
  - T – terminal (accept) states
- G is strong if T = Q. Otherwise, G is weak.
- Assertion Language – Language accepted by G

$$A(G) = \{ (a_0, c_0) \ldots (a_{k-1}, c_{k-1}) \mid \exists q_0,\ldots,q_k\text{: } q_k \in T, \forall 0 \leq i < k, (q_i, (a_i, c_i), q_{i+1}) \in \Delta \}$$

Strong GSTE Assertion: Example



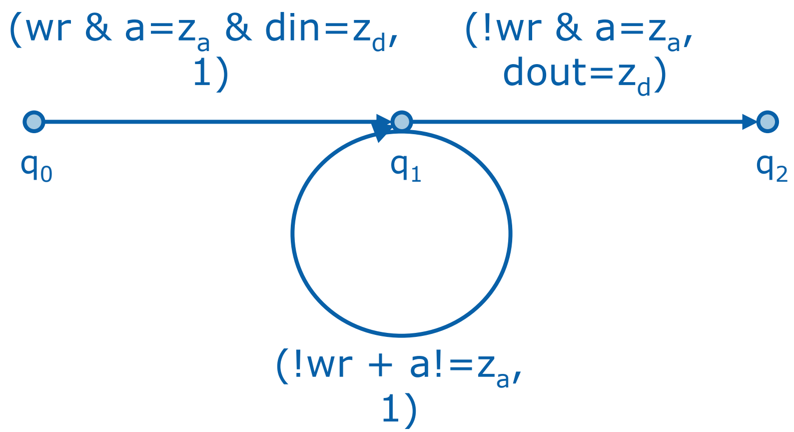
ε

(wr & a=z_a & din=z_d, 1) (!wr + a!=z_a, 1)*

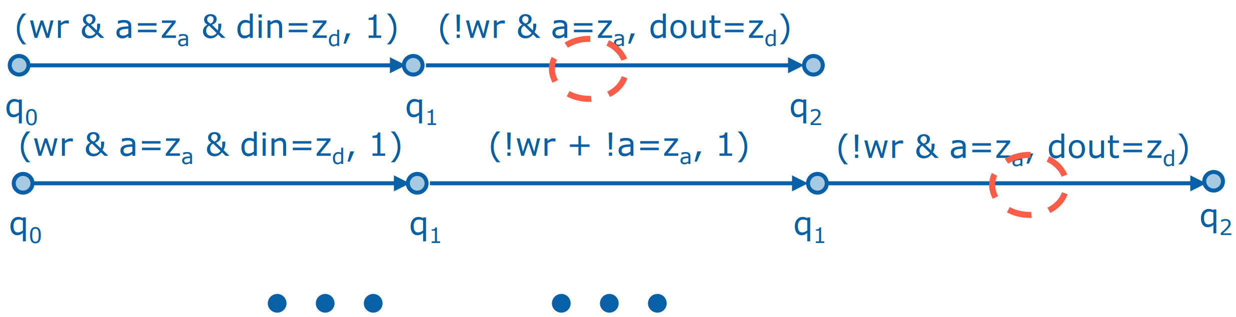
(wr & a=z_a & din=z_d, 1) (!wr + a!=z_a, 1)* (!wr & a=z_a, dout=z_d)

"After a value is written to a memory cell,
any read from the cell will return the value
as long as there is no other writes to the cell in-between."

Strong GSTE Assertion: Example (cont)



A collection of equivalent STE assertions:



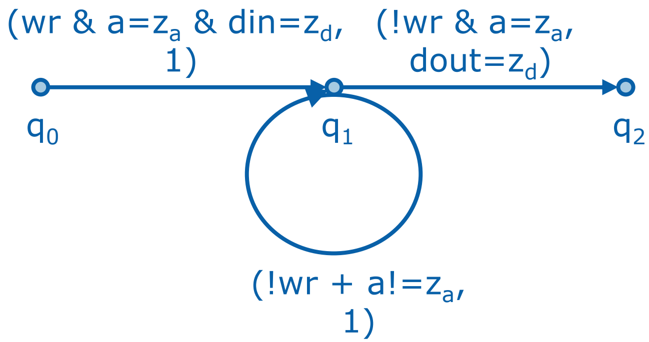
A state transition may appear in many assertions and/or many times in one assertion

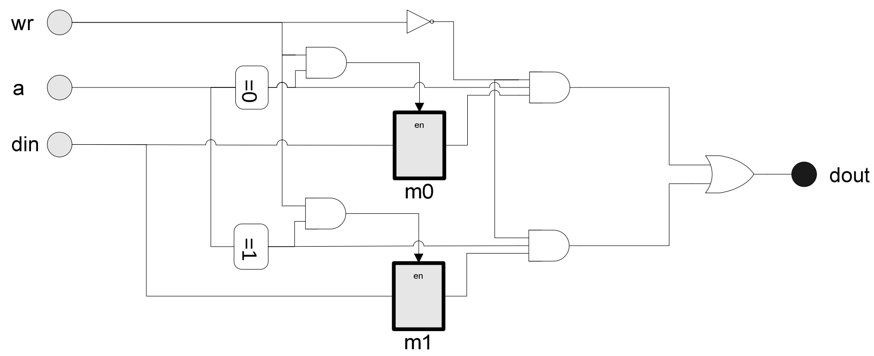
Strong GSTE Model Checking

Forward constrained reachability analysis

```
GSTEMC(M, G)
begin
for each δ ∈ Δ
   ckt_stt(δ) := 0;
add every δ' ∈ Δ from q_0 to queue;
while (queue is not empty)
   δ=(q, (a,c), q') := dequeue(queue);
   if (q = q_0)
     ckt_stt(δ) := stt_pred(a, M);
   else
     ckt_stt(δ) := post(∨_{δ' to q} ckt_stt(δ')) ∧ stt_pred(a, M);
   if ( !(ckt_stt(δ) ⇒_M c) )
     return 0;
   if there is a change in ckt_stt(δ)
     add every δ' ∈ Δ from q' to queue;
endwhile;
return 1;
end.
```

Strong GSTE Model Checking: Example





Iter. #	queue	ckt_stt(q_0, q_1)	ckt_stt(q_1, q_1)	ckt_stt(q_1, q_2)
1	{(q_0, q_1)}	wr & a=z_a & din=z_d	0	0
2, 3	{(q_1, q_1), (q_1, q_2)}	wr & a=z_a & din=z_d	(!z_a -> m0=z_d) & (z_a -> m1=z_d) & (!wr + a!=z_a)	0
4	{(q_1, q_2)}	wr & a=z_a & din=z_d	(!z_a -> m0=z_d) & (z_a -> m1=z_d) & (!wr + a!=z_a)	(!z_a -> m0=z_d) & (z_a -> m1=z_d) & !wr & a=z_a

Strong GSTE Model Checking (cont)

- Lemma (Constrained Forward Reachability)

  For all BV_Z,

  $$s \in ckt\_stt(\delta)[BV_Z/Z], \text{ iff}$$

  $$\exists\ (q_0, (a_0,c_0), q_1)\ (q_1, (a_1,c_1), q_2)\ \ldots\ (q_{j-1}, (a_{j-1},c_{j-1}), q_j)=\delta,$$

  $$\exists\ \tau = s_0\ s_1\ \ldots\ s_{j-1}=s\ \ldots, \text{ s.t., } \forall 0\leq i<j,\ a_i(s_i(I), s_i(L), s_i(O), BV_Z)=1$$

  for all state s and all $\delta \in \Delta$.

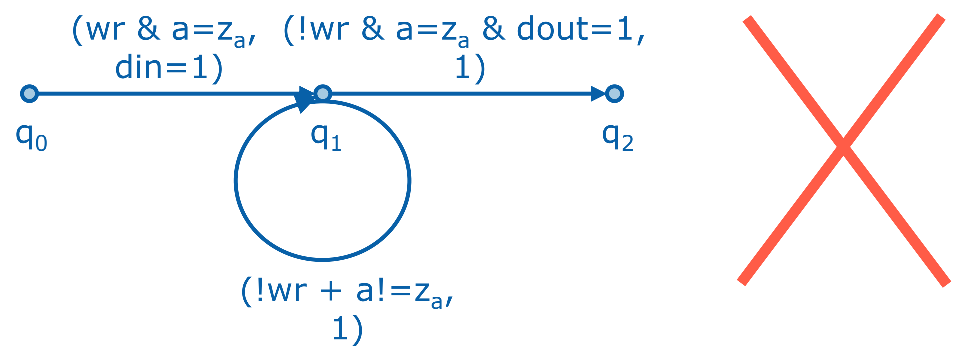
- Theorem

  $$GSTEMC(M, G)=1 \text{ iff } M \mid= A(G)$$

Strong GSTE limitation

Strong assertion automaton is typically enough for describing a large class of hardware correctness. However,

- It cannot describe properties where consequents also depend on antecedents in the future
- E.g., "if value 1 is read out from the cell, then the last write to the cell must be value 1."

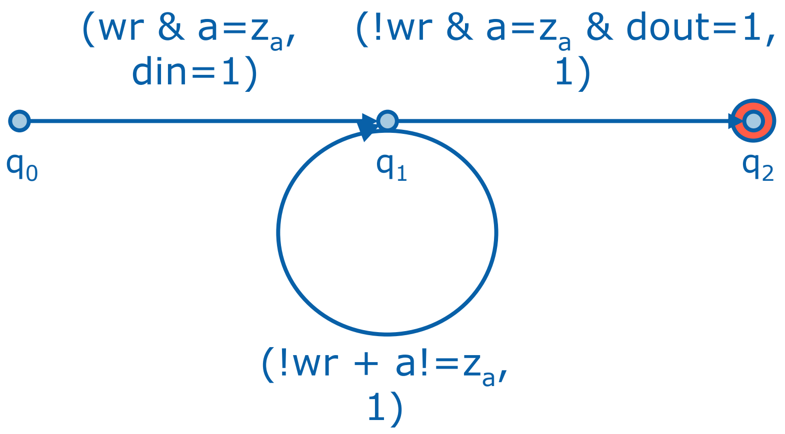


ε

$(wr \,\&\, a{=}z_a,\ din{=}1)\ (!wr + a!{=}z_a,\ 1)^*$ ← wont hold!

$(wr \,\&\, a{=}z_a,\ din{=}1)\ (!wr + a!{=}z_a,\ 1)^*\ (!wr \,\&\, a{=}z_a \,\&\, dout{=}1,\ 1)$

Weak GSTE Assertion: Example



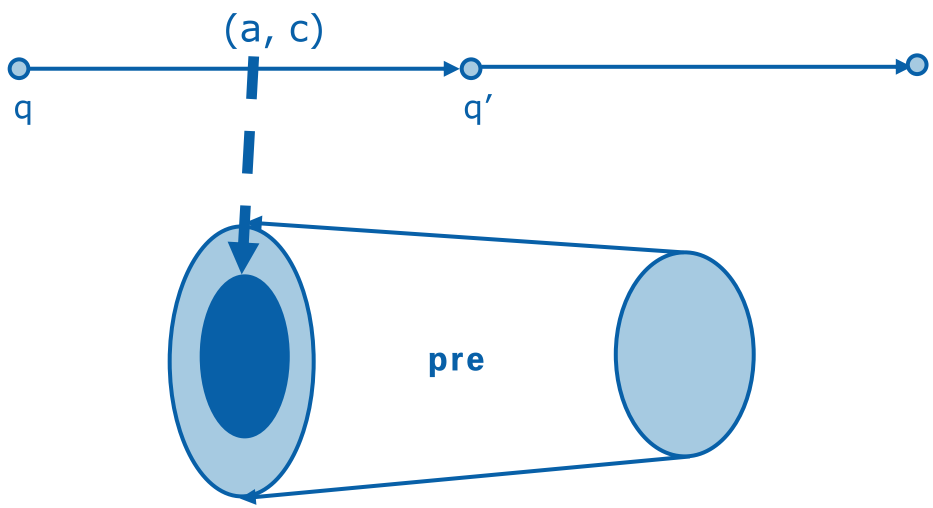
(wr & $a=z_a$, din=1) (!wr + $a!=z_a$, 1)* (!wr & $a=z_a$ & dout=1, 1)

Weak GSTE Model Checking

Step 1: constrained backward reachability analysis

- Pull back future constraints (reverse of the forward reachability analysis)
- Pre-Image Function

$$pre(p(I,L,Z)) = \exists L': (\exists I': p(I',L',Z)) \wedge (\wedge_{l \in L}\ l' = n_l(I,L))$$

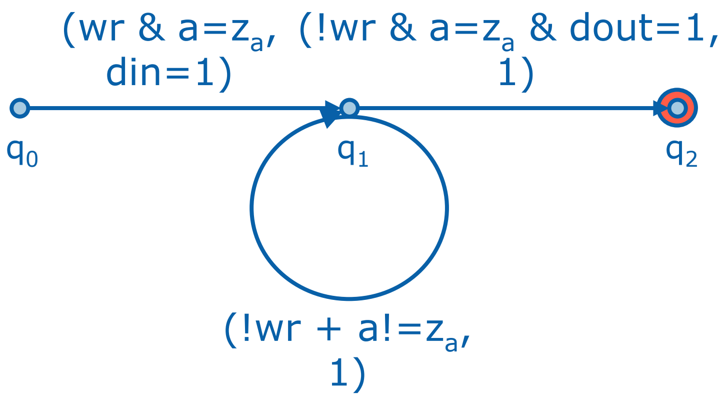


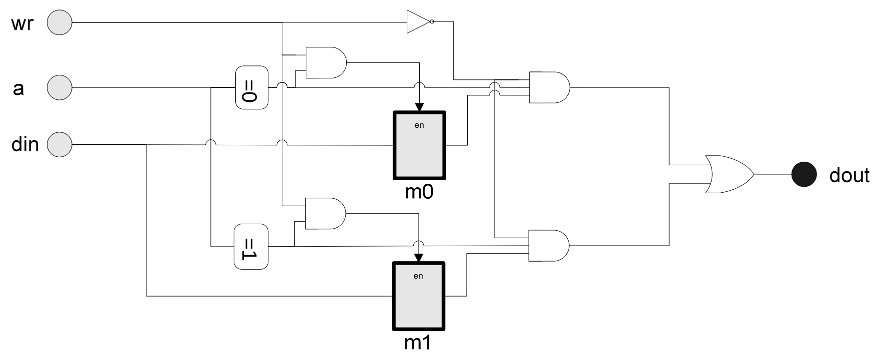
Weak GSTE Model Checking (cont)

Step 1: constrained backward reachability analysis

```
BackStrengthen(M, G)
begin
for each δ ∈ Δ
   br_stt(δ) := 0;
add every δ' ∈ Δ to T to queue;
while (queue is not empty)
   δ=(q, (a,c), q') := dequeue(queue);
   if (q' ∈ T)
     br_stt(δ) := stt_pred(a, M);
   else
     br_stt(δ) := pre(∨_{δ' from q'} br_stt(δ')) ∧ stt_pred(a, M);
   if there is a change in br_stt(δ)
     add every δ' ∈ Δ to q to queue;
endwhile;
end.
```

Backward Reachability Analysis: Example





Iter. #	queue	br_stt(q_0, q_1)	br_stt(q_1, q_1)	br_stt(q_1, q_2)
1	{(q_1, q_2)}	0	0	!wr & a=z_a & (!z_a -> m0=1) & (z_a -> m1=1)
2, 3	{(q_1, q_1), (q_0, q_1)}	0	(!wr + a!=z_a) & (!z_a -> m0=1) & (z_a -> m1=1)	!wr & a=z_a & (!z_a -> m0=1) & (z_a -> m1=1)
4	{(q_0, q_1)}	wr & a=z_a & din=1	(!wr + a!=z_a) & (!z_a -> m0=1) & (z_a -> m1=1)	!wr & a=z_a & (!z_a -> m0=1) & (z_a -> m1=1)

Weak GSTE Model Checking (cont)

- Lemma (Constrained Backward Reachability)

  For all BV_Z,

  $$s \in br\_stt(\delta)[BV_Z/Z], \text{ iff}$$

  $$\exists\ (q_1, (a_1,c_1), q_2)=\delta\ (q_2, (a_2,c_2), q_3) \ldots (q_{j-1}, (a_{j-1},c_{j-1}), q_j \in T),$$

  $$\exists\ \tau = s_1 = s \ldots s_{j-1} \ldots, \text{ s.t., } \forall 1 \le i < j,\ a_i(s_i(I), s_i(L), s_i(O), BV_Z)=1$$

  for all state s and all $\delta \in \Delta$.

- Lemma (Antecedent Strengthening)

  Construct $G' = (\Sigma, Q, q_0, \Delta', Q)$ from $G = (\Sigma, Q, q_0, \Delta, T)$ by strengthening a in each $\delta=(q, (a, c), q') \in \Delta$ with $br\_stt(\delta) \wedge a$. Then

  $$G' \mid= M \text{ iff } G \mid= M$$

- Theorem

  $$GSTEMC(M, G') \text{ iff } G \mid= M$$

ω-GSTE Assertion and Model Checking

- To achieve ω-regularity in expressiveness
- ω-Assertion Automaton

$$G^{\omega} = (\Sigma, Q, q_0, \Delta, \{T_1, T_2, \ldots, T_k\})$$

  - Σ - assertion alphabet
  - Q – a set of specification states
  - q0 – the initial state
  - $\Delta \subseteq Q \times \Sigma \times Q$ – state transition relation
  - $T_1, T_2, \ldots, T_k$ – fair sets of states
- Assertion Language – Language accepted by G^{ω}
  - A transition path is fair if it visits $T_1, \ldots, T_k$ infinitely often.
  - Only look at the ω-assertion word captured by a fair transition path.
- Model checking
  - A fix-point of backward reachability strengthening + GSTEMC

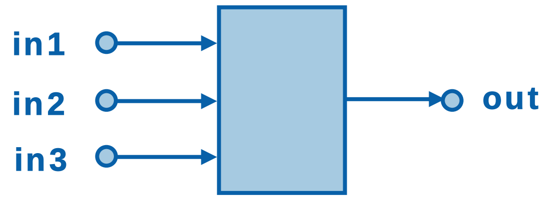
Outline

- Background
- Circuit Model and Assertion Language
- STE
- GSTE
- GSTE for Concurrent Hardware
- Symbolic Simulation
- Quaternary Abstraction
- Conclusion

GSTE Limitation

- GSTE assertions are sequential in nature, not suitable for describing concurrent behaviors
- In reality, there are possibly many current behaviors in a circuit

Ex. : out becomes 1 after each of the three inputs has been set to 1 at least once.



It will be very tedious to describe all possible orders of three inputs being set to 1!

The Meet Operator

- Meet of assertion letters:

$$(a_1, c_1) \sqcap (a_2, c_2) = (a_1 \wedge a_2, c_1 \wedge c_2)$$

- Meet of assertion words:

$$\sigma_1\sigma_2\ldots\sigma_k \sqcap \sigma'_1\sigma'_2\ldots\sigma'_k = (\sigma_1\sqcap\sigma'_1)(\sigma_2\sqcap\sigma'_2)\ldots(\sigma_k\sqcap\sigma'_k)$$

- Meet of assertion languages:

$$A_1 \sqcap A_2 = \{ w_1\sqcap w_2 \mid w_1\in A_1, w_2\in A_{2,} |w_1|=|w_2| \}$$

The Meet Operator (cont)

- Repeated application

$$\sqcap^0 A = A, \quad \sqcap^k A = (\sqcap^{k-1} A) \sqcap A \ (k>0)$$

- Lemma

$$\sqcap^k A \subseteq \sqcap^{k+1} A \text{ but } \Gamma(\sqcap^k A) = \Gamma(\sqcap^{k+1} A)$$

  - proof sketch
    - $(w_1 \sqcap w_2 \sqcap \ldots \sqcap w_k) \sqcap w_k = w_1 \sqcap w_2 \sqcap \ldots \sqcap w_k$
    - $w \sqcap w'$ may be new, but $\Pi(w) \cap \Pi(w') \subseteq \Pi(w \sqcap w')$

- Theorem (Self Consistency)

$$A \subseteq \cup_{k\geq 0} \sqcap^k A \text{ but } \Gamma(A) = \Gamma(\cup_{k\geq 0} \sqcap^k A)$$

Concurrent Specification in GSTE

$$CS = (\Sigma, Q_{\bullet} \cup Q_{+} \cup Q_{\times}, q_0, \Delta_{\bullet} \cup \Delta_{+} \cup \Delta_{\times})$$

- Concatenation (transition) $\Delta_{\bullet}$: for each $q_i \in Q_{\bullet}$,

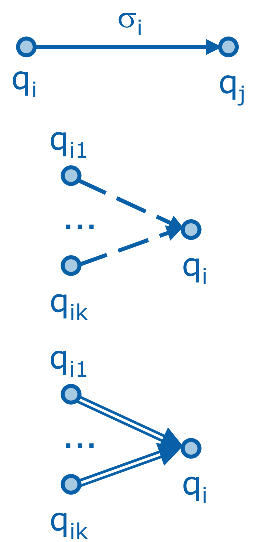
$$q_i = q_j \bullet \sigma_j \qquad (q_j \in Q_{\bullet} \cup Q_{+} \cup Q_{\times})$$

- Summation Δ_{+}: for each $q_i \in Q_{+}$,

$$q_i = q_{i_1} + \dots \dots + q_{i_k} \quad (q_{i_h} \in Q_{\bullet}, 0 \le h < k)$$

- Meet $\Delta_{\times}$: for each $q_i \in Q_{\times}$,

$$q_i = q_{i_1} \times \dots \dots \times q_{i_k} \quad (q_{i_h} \in Q_{\bullet} \cup Q_{+}, 0 \le h < k)$$



intel

Concurrent Specification: Semantics

- Initial state:

$$A(q_0) = (1, 1)^*$$

- Concatenation $q_i = q_j \bullet \sigma_j$:

$$A(q_i) = A(q_j) \bullet \sigma_j$$

- Summation $q_i = q_{i_1} + \dots \dots + q_{i_k}$:

$$A(q_i) = A(q_{i_1}) \cup \dots \dots \cup A(q_{i_k})$$

- Meet $q_i = q_{i_1} \times \dots \dots \times q_{i_k}$:

$$A(q_i) = A(q_{i_1}) \sqcap \dots \dots \sqcap A(q_{i_k})$$

- Assertion language of CS:

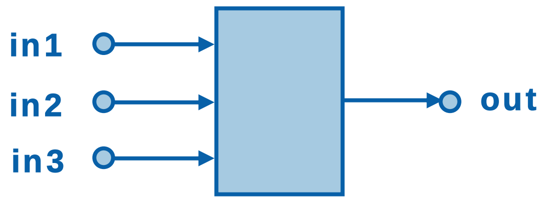
$$A(CS) = \cup_{q \in Q^{\bullet} \cup Q^{+} \cup Q^{\times}} A(q)$$

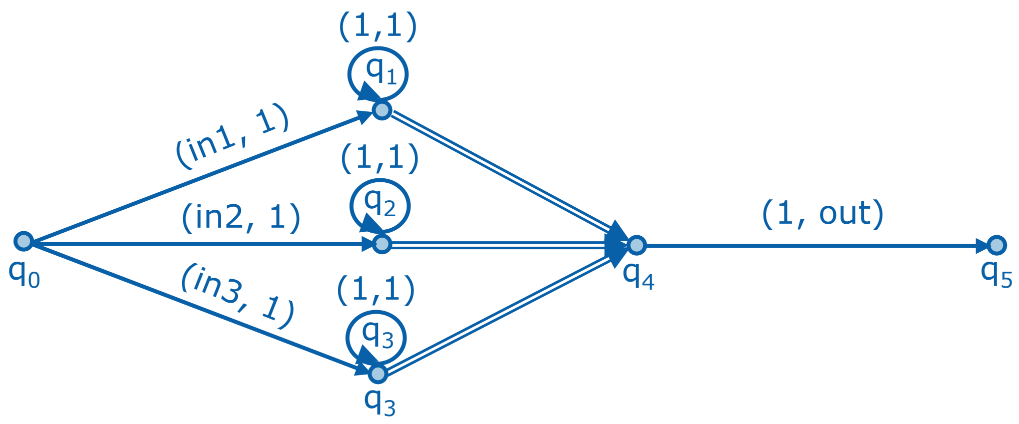
Theorem

There is a unique solution to the system

Concurrent Specification: Example

Out becomes 1 after each of the three inputs has been set to 1 at least once.





Concurrent Specification: Serialization

Theorem

$$\cup_{k\geq 0} \sqcap^{k} A(q) \text{ is regular}$$

for any q in CS.

- proof sketch
  - $\cup_{k\geq 0}\sqcap^{k} (A_j \bullet \sigma_j) = (\cup_{k\geq 0}\sqcap^{k} A_j) \bullet \sigma_j$
  - $\cup_{k\geq 0}\sqcap^{k} (A_1 \cup A_2) = (\cup_{k\geq 0}\sqcap^{k} A_1) \cup (\cup_{k\geq 0}\sqcap^{k} A_2) \cup (\cup_{k\geq 0}\sqcap^{k} A_1) \sqcap (\cup_{k\geq 0}\sqcap^{k} A_2)$
  - $\cup_{k\geq 0}\sqcap^{k} (A_1 \sqcap A_2) = (\cup_{k\geq 0}\sqcap^{k} A_1) \sqcap (\cup_{k\geq 0}\sqcap^{k} A_2)$
  - construct a strong GSTE automaton over $P(\{\cup_{k\geq 0}\sqcap^{k} A_1 , \ldots, \cup_{k\geq 0}\sqcap^{k} A_n\})$

- Since $\Gamma(A) = \Gamma(\cup_{k\geq 0} \sqcap^{k} A)$, this effectively provides a precise solution to model check a CS, however
- The sequential assertion automaton may be exponentially large

Model Checking Concurrency

```
cGSTEMC(M, CG)
begin
for each q ∈ Q
  ckt_stt(δ) := 0;
add q0 to queue;
while (queue is not empty)
  q := dequeue(queue);
  if (q = q0)
    ckt_stt(q) := 1;
  else if (q = q' • (a,c))
    ckt_stt(q) := post(ckt_stt(q')) ∧ stt_pred(a, M);
    if ( !(ckt_stt(q) ⇒M c) )
      return 0;
  else if (q = q1 + … … + qk)
    ckt_stt(q) := ∪1≤i≤k ckt_stt(qi);
  else
    ckt_stt(q) := ∩1≤i≤k ckt_stt(qi);
  endif;
  if there is a change in ckt_stt(q)
    add every q' that has q in its RHS to queue;
endwhile;
return 1;
end.
```

Model Checking Concurrency (cont)

- Lemma (Constrained Forward Reachability)

  For all BV_Z,

  $s \in ckt\_stt(q)[BV_Z/Z]$, if

  $\exists\ (a_0,c_0)\ (a_1,c_1)\ \ldots\ (a_{j-1},c_{j-1}) \in A(q)$,

  $\exists\ \tau = s_0\ s_1\ \ldots\ s_{j-1}=s\ \ldots$, s.t., $\forall 0 \leq i<j,\ a_i(s_i(I),\ s_i(L),\ s_i(O),\ BV_Z)=1$

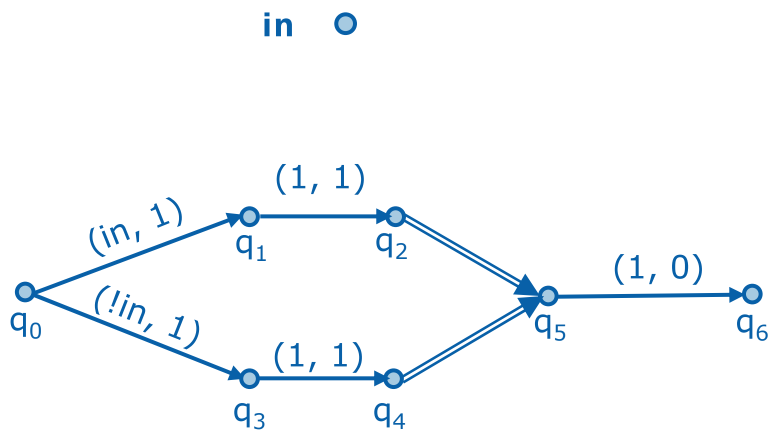
  for all state s and all $q \in Q$.

- Theorem

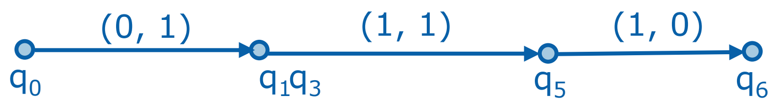
  cGSTEMC(M, CG)=1 implies M |= A(CG)

Model Checking Concurrency (cont)

- Why Completeness No Longer Holds?

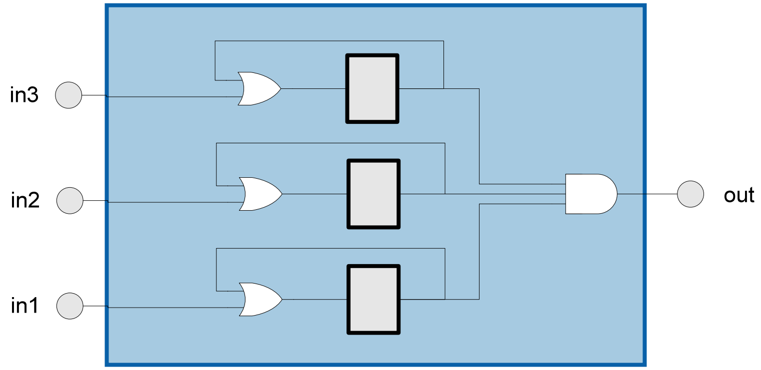


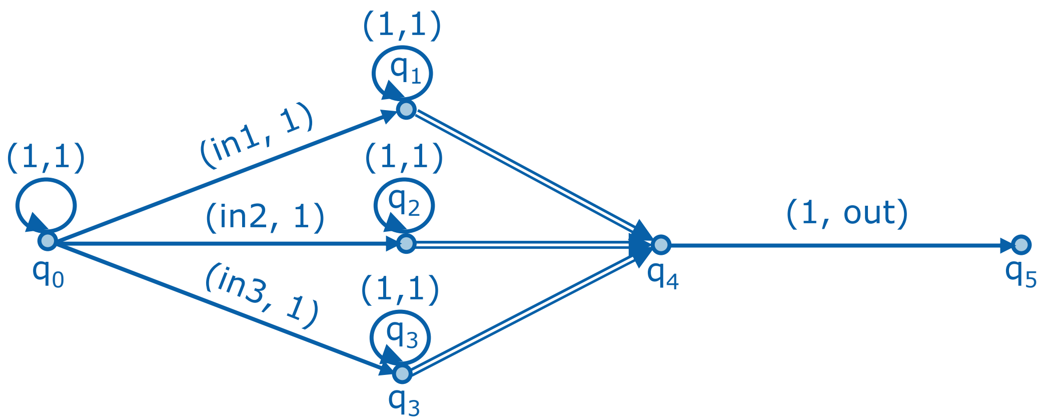
Solution: (Partial) Serialization



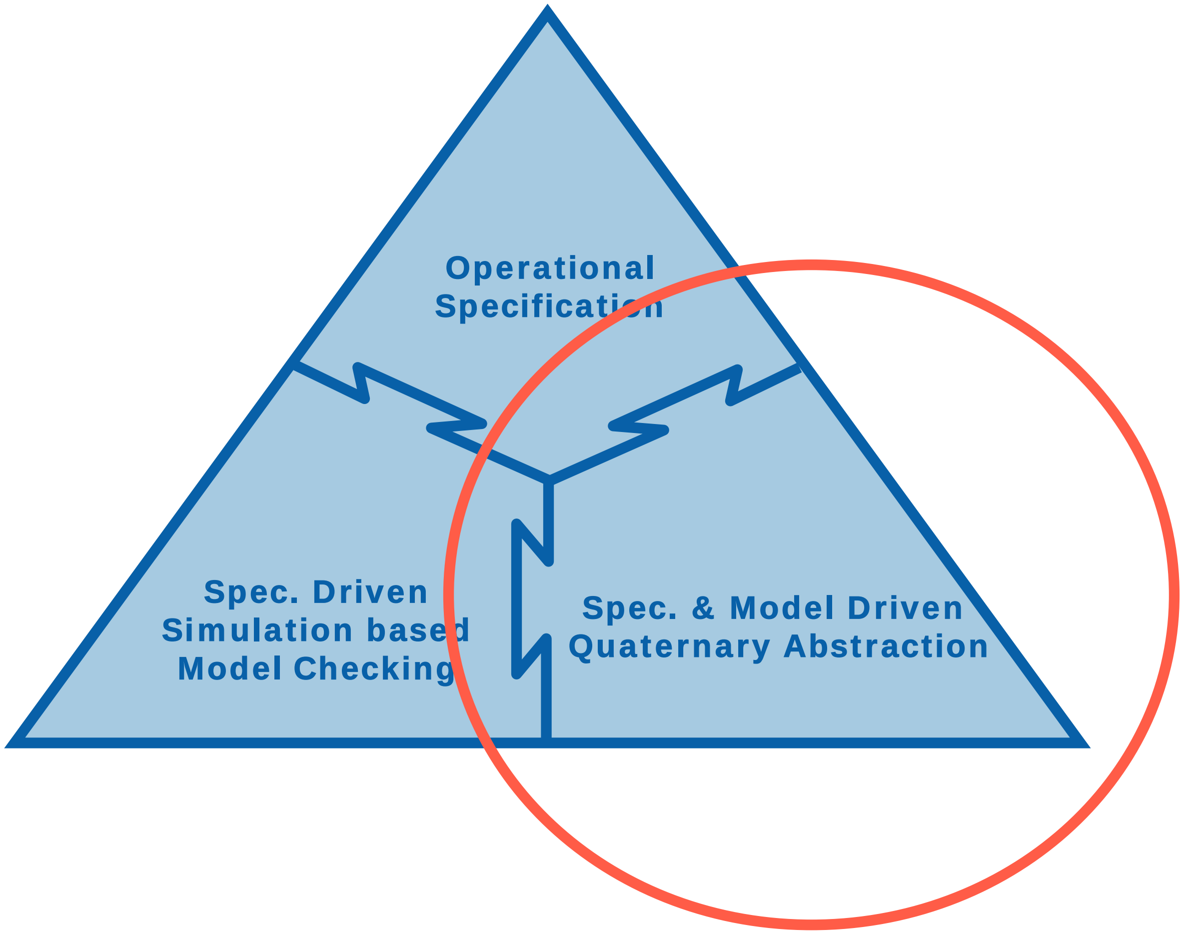
Concurrency GSTE: An Exercise

Out becomes 1 after each of the three inputs has been set to 1 at least once.





The Missing Piece

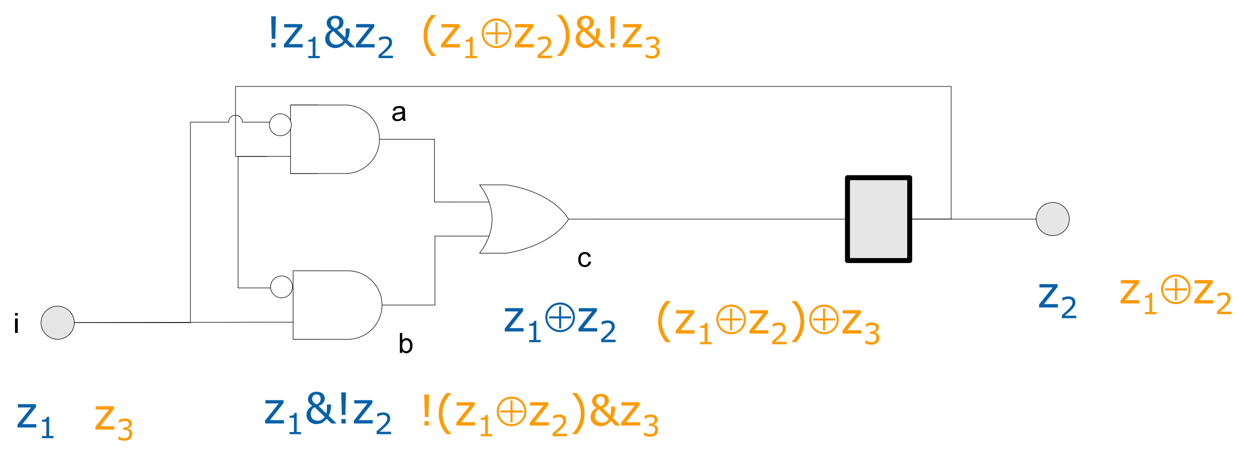


Outline

- Background
- Circuit Model and Assertion Language
- STE
- GSTE
- GSTE for Concurrent Hardware
- Symbolic Simulation
- Quaternary Abstraction
- Conclusion

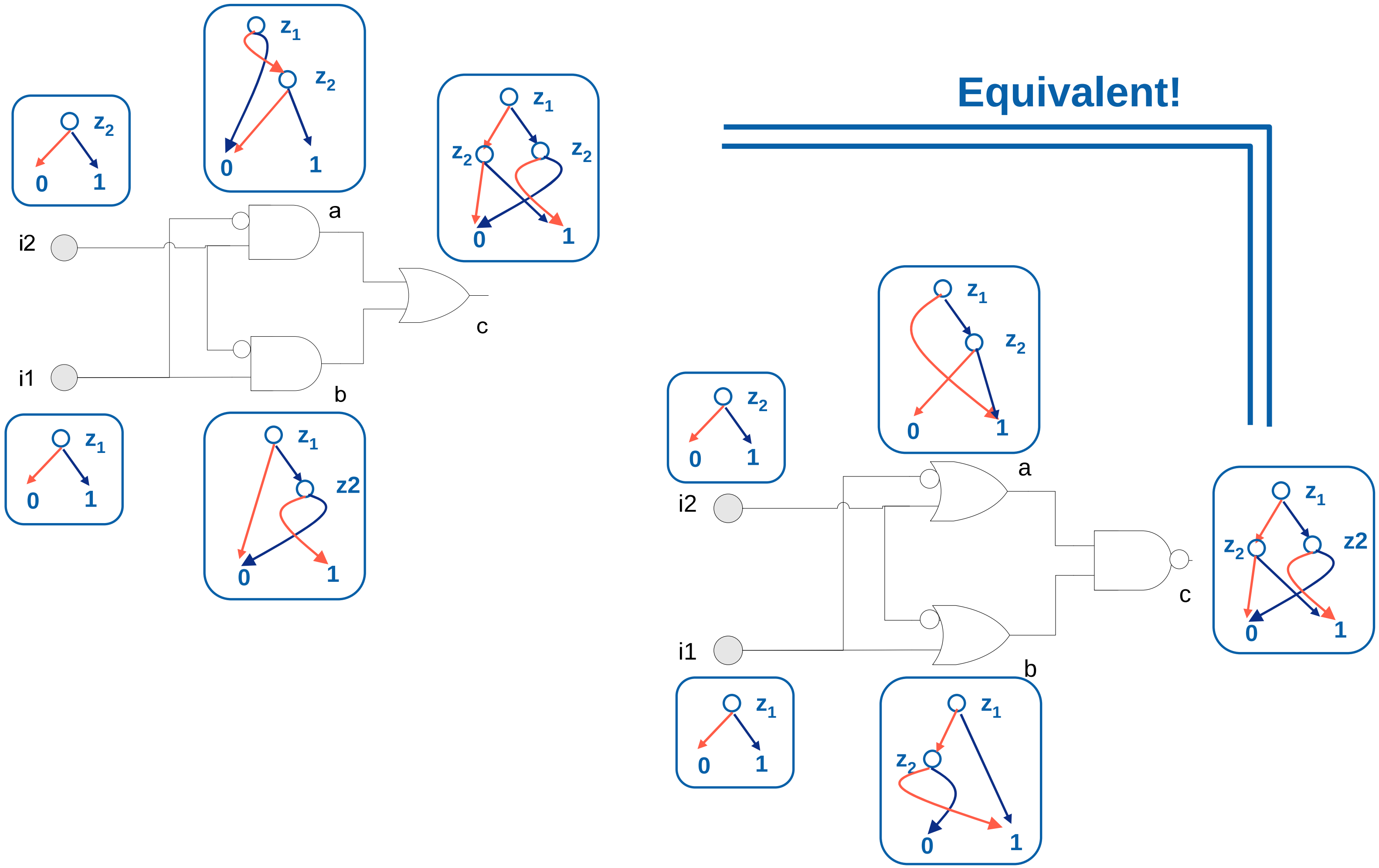
Symbolic Simulation

- Network of basic logic functions
  - Rather than a monolithic transition relation
- Simulation by successive evaluations of basic logic functions with Boolean expressions
  - Rather than relational product computation
  - Need Boolean function vectors

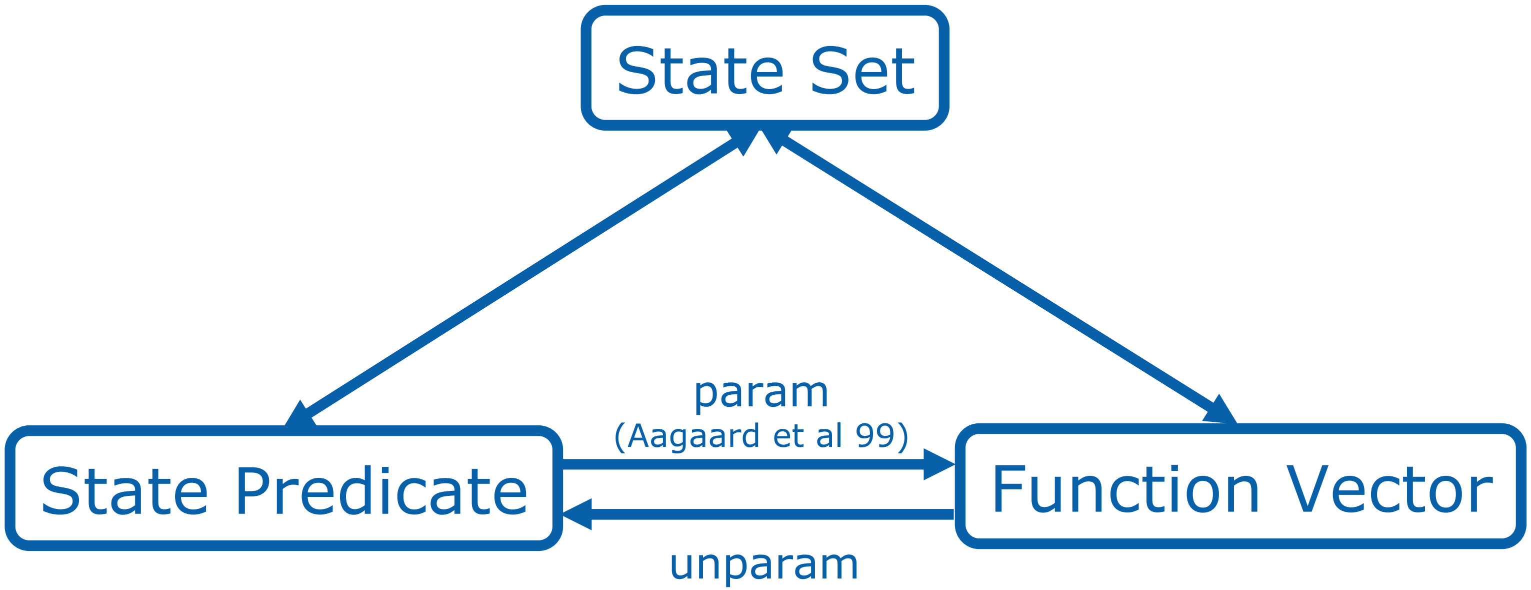


Ordered Binary Decision Diagrams

OBDD – provide a canonical form for any Boolean function/expression (Bryant 92)



Predicates vs Function Vectors



Predicates vs Function Vectors: Example

z_a	wr	a
0	0	0
0	0	1
0	1	1
1	0	0
1	0	1
1	1	0

z_a	P_1	P_2	wr	a
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	1	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	0
1	1	1	1	0

$!wr + a!=z_a$

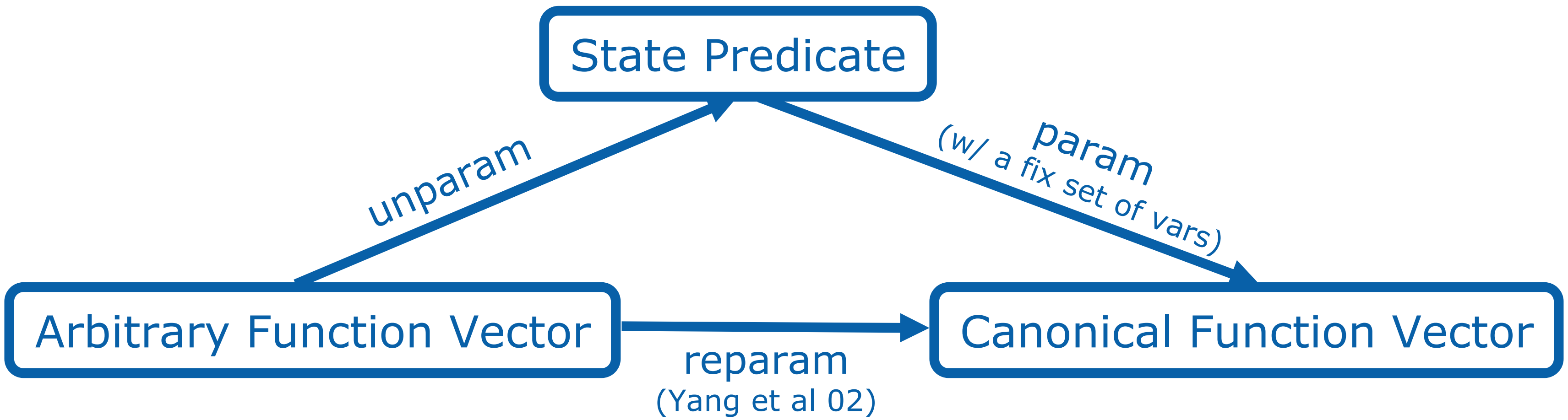
$(wr := p_1, a := p_1\&!z_a + !p_1\&p_2)$

Set Union and Canonization

- Set union is simple.
  - $(f_1, f_2, \ldots, f_k) \cup (g_1, g_2, \ldots, g_k) =$

    $p := newP()$ in

    $(!p\&f_1 + p\&g_1, !p\&f_2 + p\&g_2, \ldots, !p\&f_k + p\&g_k)$
- What about equivalence check?
  - $(f_1, f_2, \ldots, f_k) = (g_1, g_2, \ldots, g_k)$?
  - Need canonicity



Outline

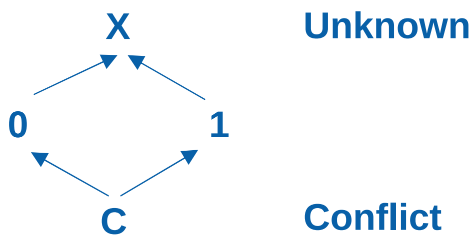
- Background
- Circuit Model and Assertion Language
- STE
- GSTE
- GSTE for Concurrent Hardware
- Symbolic Simulation
- Quaternary Abstraction
- Conclusion

Quaternary Abstraction: Movitation

```
GSTEMC(M, G)
begin
for each δ ∈ Δ
  ckt_stt_abs(δ) := 0;
add every δ' ∈ Δ from q_0 to queue;
while (queue is not empty)
  δ=(q, (a,c), q') := dequeue(queue);
  if (q = q_0)
    ckt_stt_abs(δ) := stt_pred_abs(a, M);
  else
    ckt_stt_abs(δ) := post_abs(∨_{δ' to q} ckt_stt_abs(δ')) ∧ stt_pred_abs(a, M);
  if ( !(ckt_stt_abs(δ) ⇒_M c) )
    return 0;
  if there is a change in ckt_stt_abs(δ)
    add every δ' ∈ Δ from q' to queue;
endwhile;
return 1;
end.
```

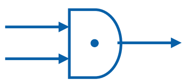
Would like to work on **dynamic** abstractions of circuit states to reduce complexity, not a **static abstraction** of M.

Quaternary Abstraction

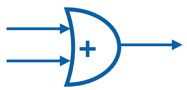


(reverse) information order

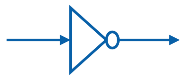
Basic gates:



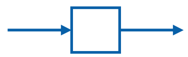
&	X	0	1	C
X	X	0	X	C
0	0	0	0	C
1	X	0	1	C
C	C	C	C	C



+	X	0	1	C
X	X	X	1	C
0	X	0	1	C
1	1	1	1	C
C	C	C	C	C



!	
X	X
0	1
1	0
C	C



I	
X	X
0	0
1	1
C	C

Quaternary Vector

- A quaternary vector is an abstraction of a set of Boolean vectors
  - A quaternary assignment to I and L is an abstraction of a set of states
- Point-wise abstraction of set union, intersection
- Complexity reduction

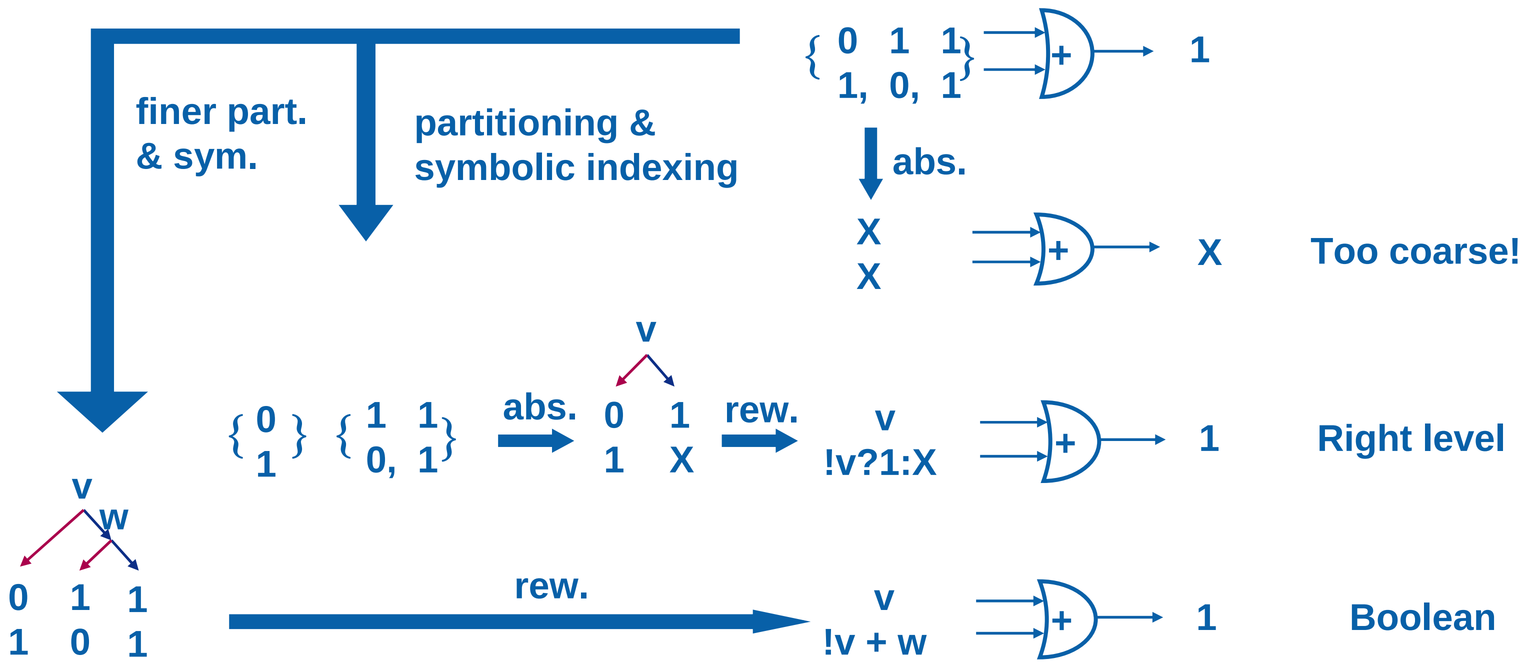
Boolean vector sets	quaternary vector
{ (1, 1) }	(1, 1)
{ (0, 1), (1, 1) }	(X, 1)
{ (1, 0), (1, 1) }	(1, X)
{ (0, 1), (1, 0), (1, 1) }	(X, X)
{ (0, 0), (0, 1), (1, 0), (1, 1) }	(X, X)

∩	X	0	1	C
X	X	0	1	C
0	0	0	C	C
1	1	C	1	C
C	C	C	C	C

∪	X	0	1	C
X	X	X	X	X
0	X	0	X	0
1	X	X	1	1
C	X	0	1	C

Quaternary Function Vector

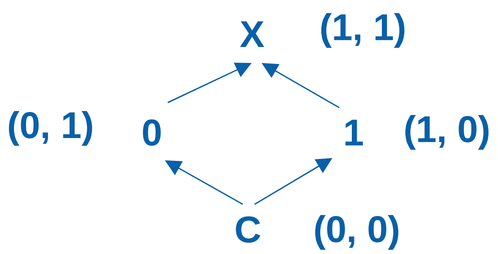
- It is often desired to work on a set of quaternary vectors
  - Capture different cases
  - Avoid too coarse abstraction
- A quaternary function vector encodes a set of quaternary vectors



Quaternary Abstraction

- Find the maximum level of quaternary abstraction without losing model checking precision
- Currently, it is up to the verifier to decide what is the right level, by
  - Encoding it in the specification, or
  - Specifying elements in L and L that must always have Boolean values
- Automation through X-driven abstraction refinement would help greatly

Dual Rail Encoding in Practice

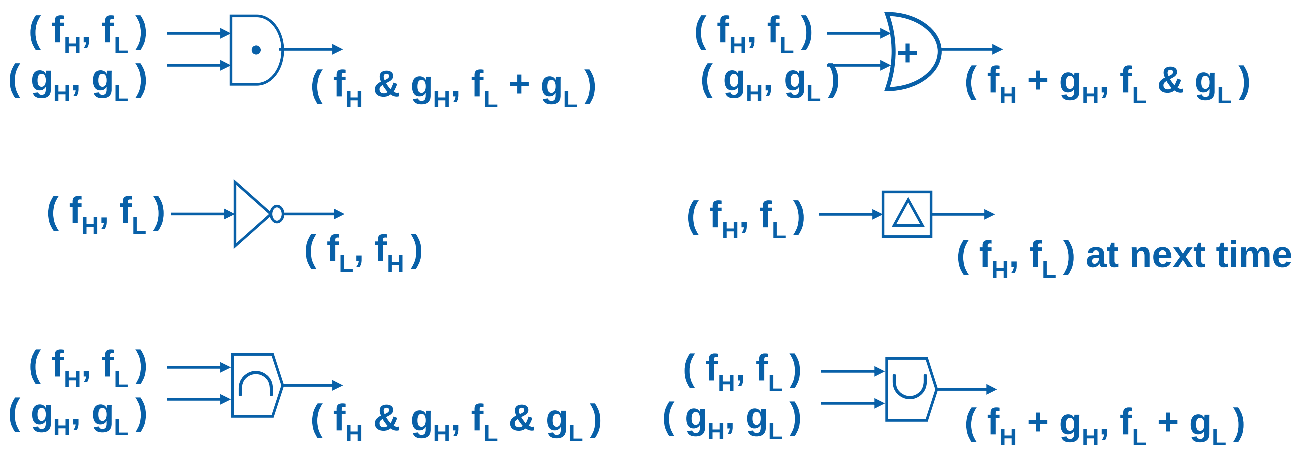


$$(f, g) = \begin{cases} X, \text{ when } f \,\&\, g \\ 1, \text{ when } f \,\&\, !g \\ 0, \text{ when } !f \,\&\, g \\ C, \text{ when } !f \,\&\, !g \end{cases}$$

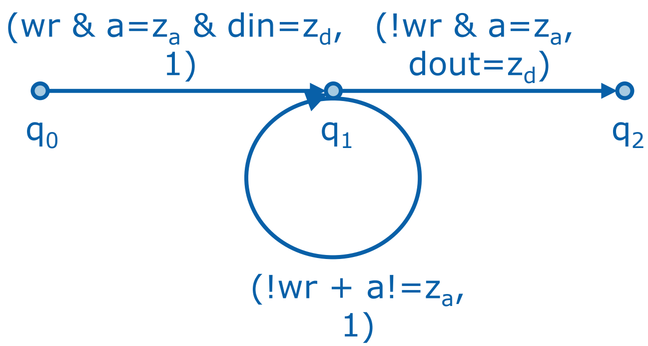
(v, !v?1:X)

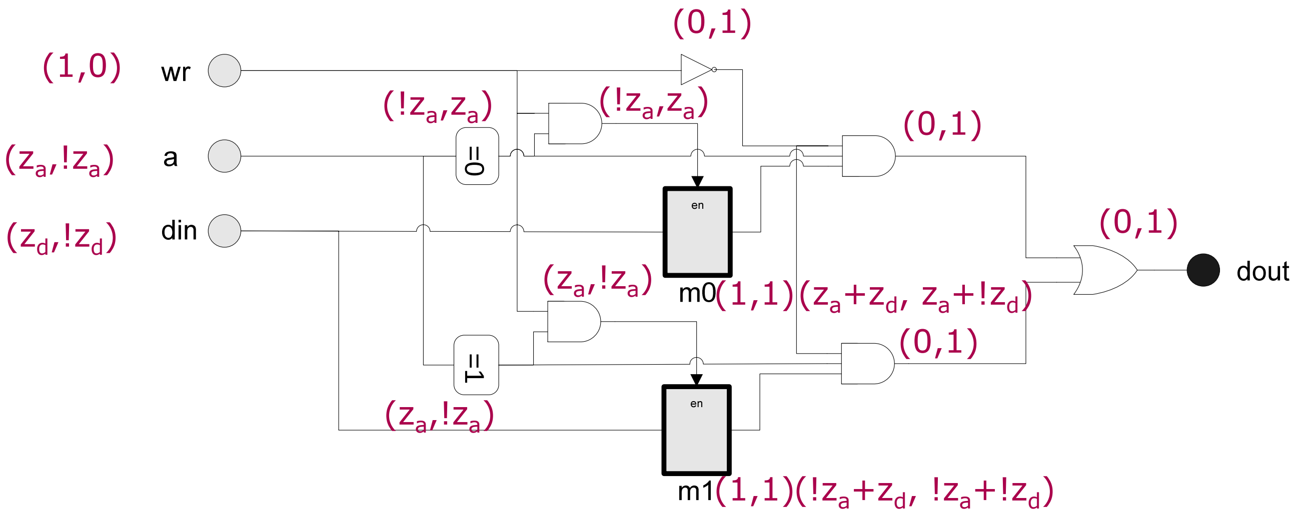
⬇

((v, !v) , (1, v))

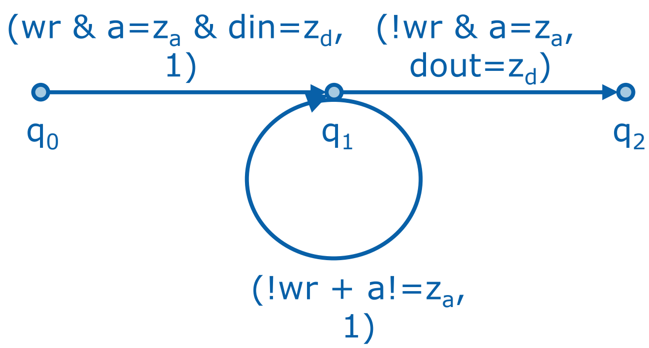


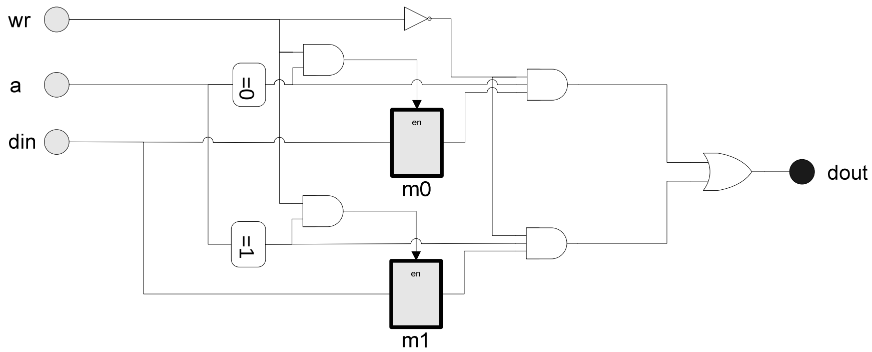
Memory Example Revisited





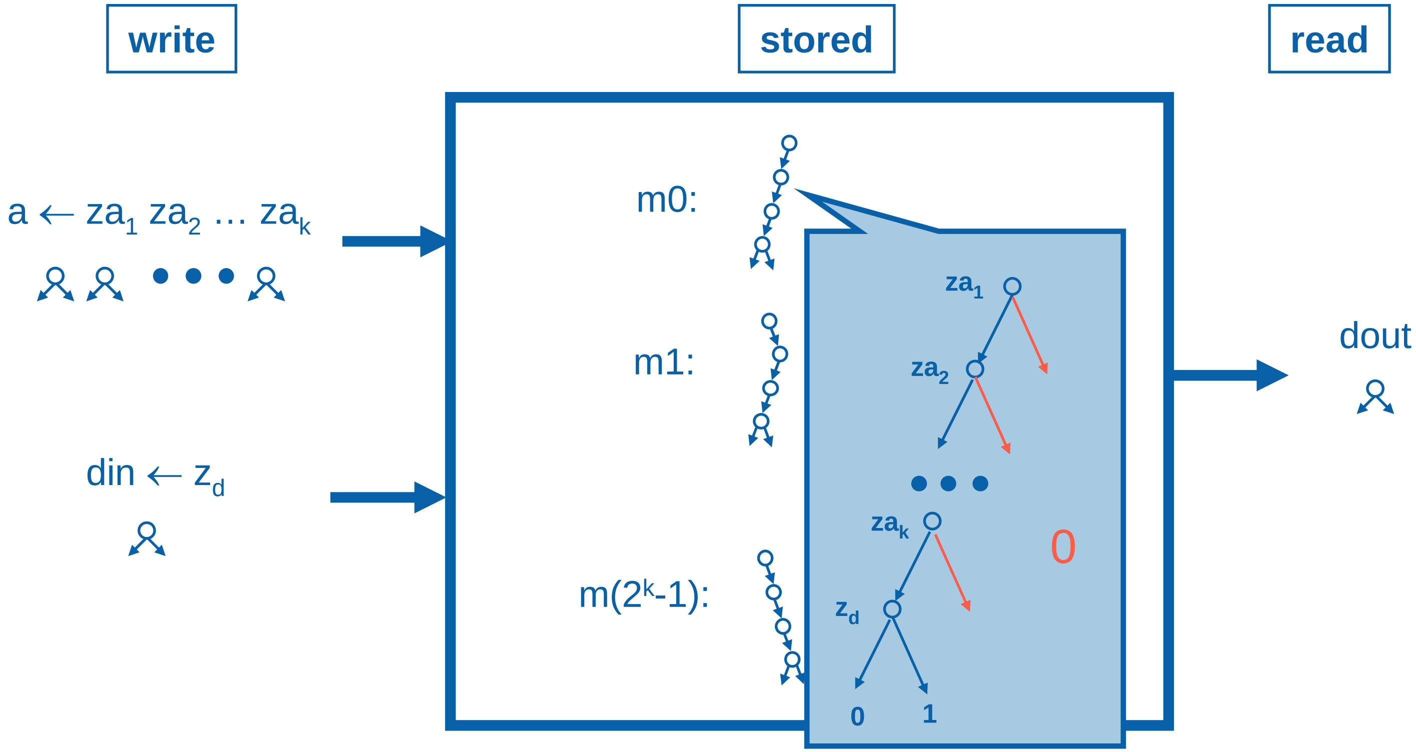
Memory Example Revisited





Iter. #	queue	ckt_stt(q_0, q_1)	ckt_stt(q_1, q_1)	ckt_stt(q_1, q_2)
1	$\{(q_0, q_1)\}$	wr ← (1, 0), a ← (z_a, $!z_a$), din ← (z_d, $!z_d$)	C	C
2,3	$\{(q_1, q_1),$ $(q_1, q_2)\}$	wr ← (1, 0), a ← (z_a, $!z_a$), din ← (z_d, $!z_d$)	wr ← (p, !p), a ← ($!p+!z_a$, $!p+z_a$), m0 ← (z_a+z_d, $z_a+!z_d$), m1 ← ($!z_a+z_d$, $!z_a+!z_d$)	C
4	$\{(q_1, q_2)\}$	wr ← (1, 0), a ← (z_a, $!z_a$), din ← (z_d, $!z_d$)	wr ← (p, !p), a ← ($!p+!z_a$, $!p+z_a$), m0 ← (z_a+z_d, $z_a+!z_d$), m1 ← ($!z_a+z_d$, $!z_a+!z_d$)	wr ← (1, 0), a ← (z_a, $!z_a$), m0 ← (z_a+z_d, $z_a+!z_d$), m1 ← ($!z_a+z_d$, $!z_a+!z_d$)

Complexity Analysis on A Single Rail



For an $n=2^k$ cell memory, the size of all the BDD nodes in a single rail is:

$$k + 1 + n * (k + 1) + 1$$

Outline

- Background
- Circuit Model and Assertion Language
- STE
- GSTE
- GSTE for Concurrent Hardware
- Symbolic Simulation
- Quaternary Abstraction
- Conclusion

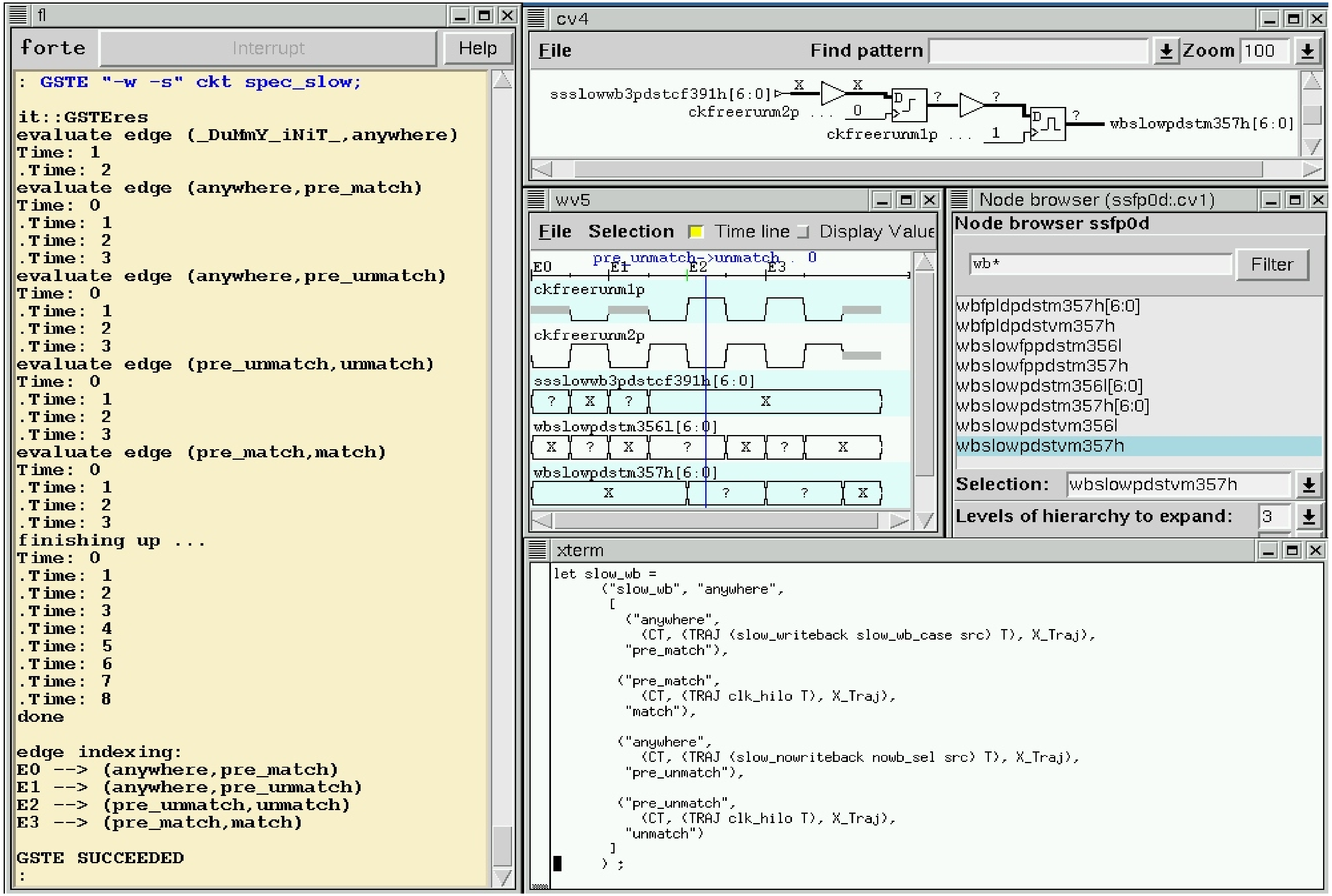
Real Life Results

<u>Intel® Pentium® 4 1.5G</u>

Ex.	#Latches	#Gates	# Spec. vars	#Prec. Nodes	Time (sec.)	Memory (MB)
1	718	17367	68	4	122	36
2	7506	62735	95	41	5220	260
3	22433	187928	401	0	117	509
4	22433	187928	103	0	500	240
5	34899	406630	24	0	451	361
6	46682	241854	282	12	132	295

- They are all over the places in a u-processor
- All cover non-trivial functionalities, a majority from inputs to outputs
- No prior model pruning/abstraction

The GSTE System



Conclusion

- An integration of high capacity of STE with expressive power of traditional model checking (Yang & Seger ICCD'00)
- Further extension to efficiently handle concurrency (Yang & Seger CAV'04)
- A multi-dimensional approach to achieve high capacity while maintaining accuracy (Yang & Seger FMCAD'02)
- A system used by FVers since 2000 on verifying Intel μ-processors with thousands of state elements (Bently HLDVT'02, Yang & Seger FMCAD'02, Schubert ICCAD'03)

Some future directions:

- Automated abstraction refinement
- High level abstraction schemes
- High level specification language with link to GSTE

Thank You Very Much!

Q/A