# Structural FSM Traversal - Theory and a Practical Algorithm*

*Dominik Stoffel* *Wolfgang Kunz*

Institute of Computer Science
University of Potsdam

* Parts of the reported research have been conducted while both authors visited Mentor Graphics Inc., North Billerica, MA, USA

# 1 Introduction

**Problem formulation:** *check equivalence* of two sequential circuits with reset signal or initializing sequence

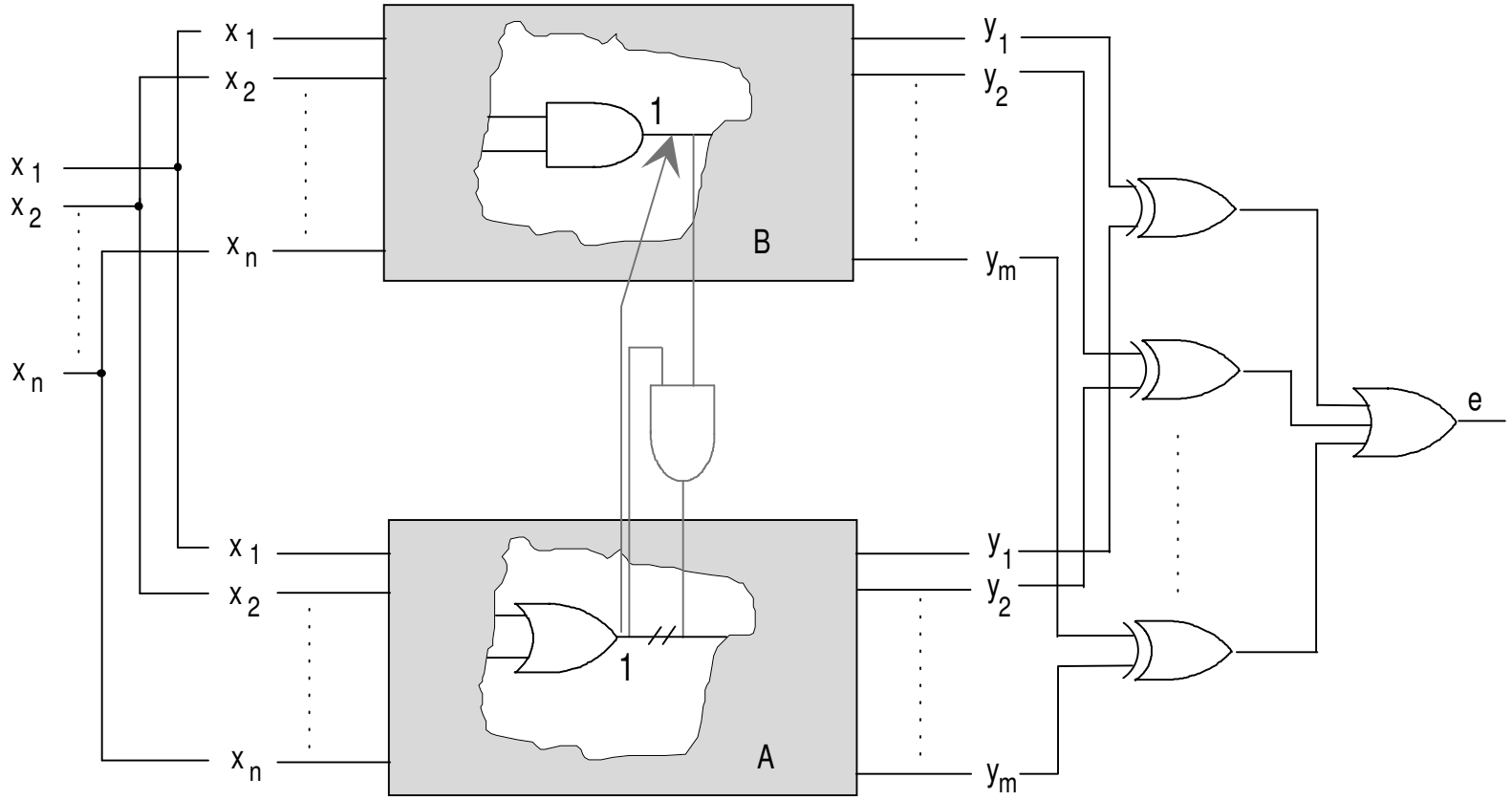
("sequential hardware equivalence", Pixley et al.)

**Conventional solution:** symbolic FSM traversal
(Coudert et al., 1990, McMillan 1990)

**Research goal:** "structural" FSM traversal,
formulate a fixed point iteration based on structural techniques (synthesis, ATPG, implications)

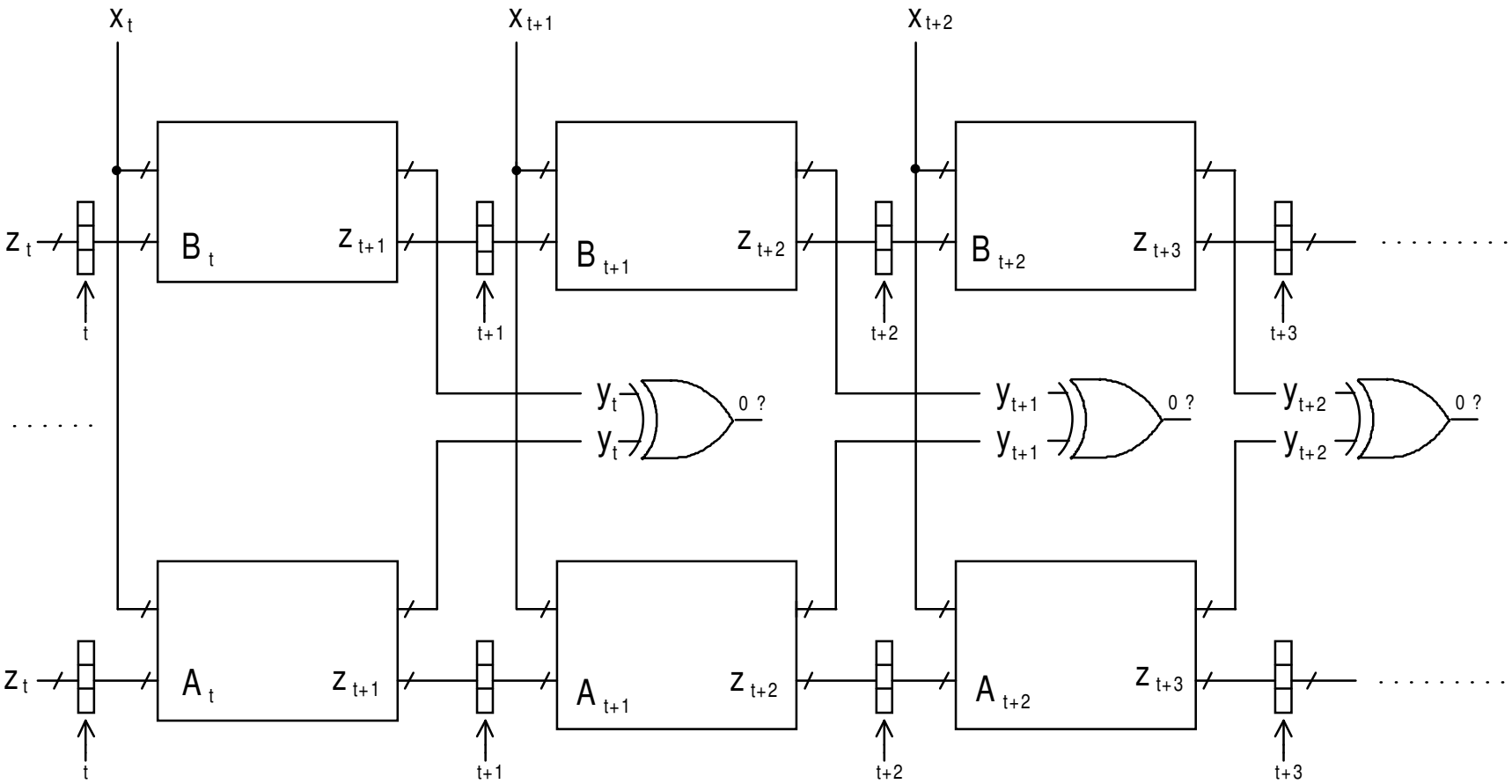
# Combinational Equivalence Checking

use similarity between designs (ATPG, implications, transformations)




"optimize" the miter

# Sequential Equivalence Checking

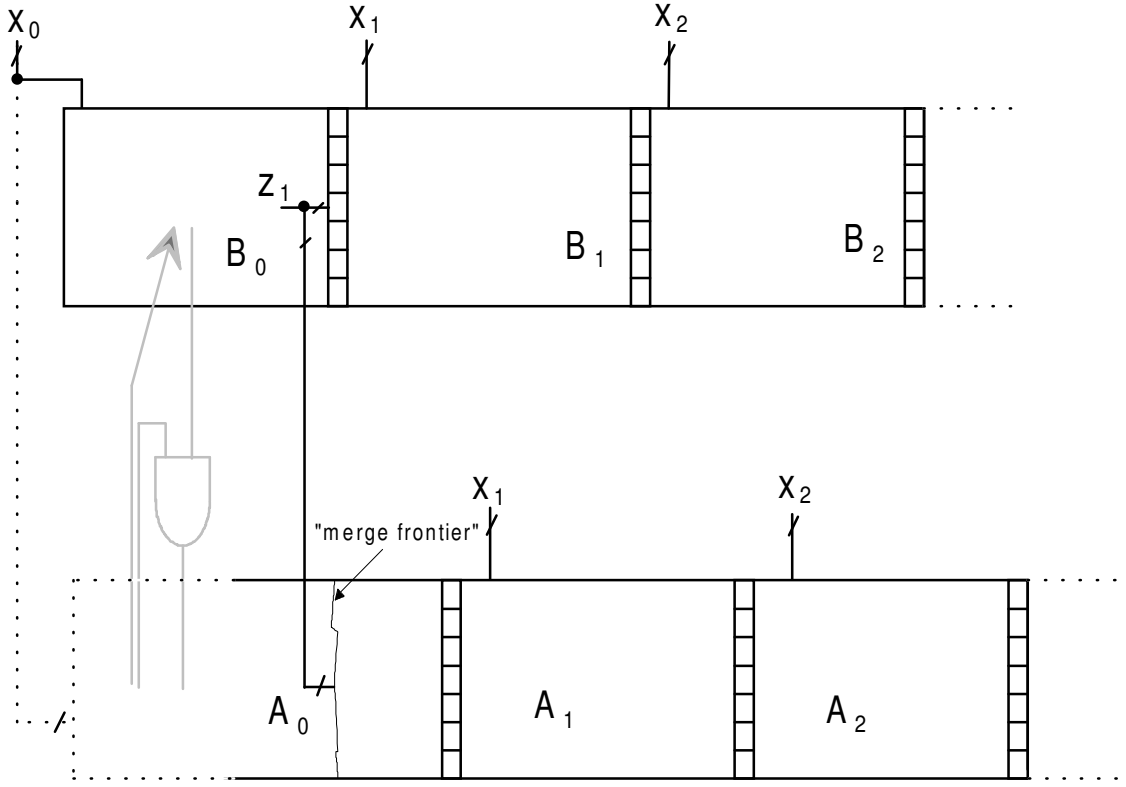



iterative miter array

# Combinational Transformations in Miter Array



- start at initial state
- "merge" circuitry of two circuits moving from time frame to time frame



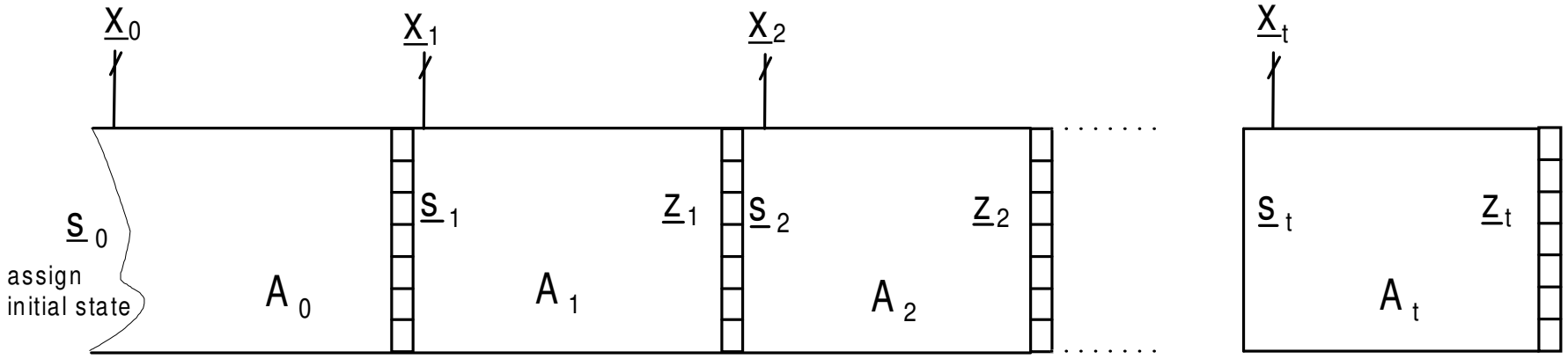

fixed point ?

# 2 Structural FSM Traversal

Completely specified, deterministic finite state machine $M = (I, S, \delta, S_0, O, \lambda)$, where:

$I$ is the input alphabet, $S$ is the set of states, $\delta : S \times I \rightarrow S$ is the next-state function, $S_0 \subseteq S$ is the (set of) initial state(s), $O$ is the output aphabet, $\lambda : S \times I \rightarrow O$ is the output function

Consider the iterative circuit array of the finite state transition structure (FST) $H = (I, S, \delta, S_0)$




$R(t)$: set of reachable states at the state variables $\underline{s}_t$ at time $t$

# "Stub Circuit"

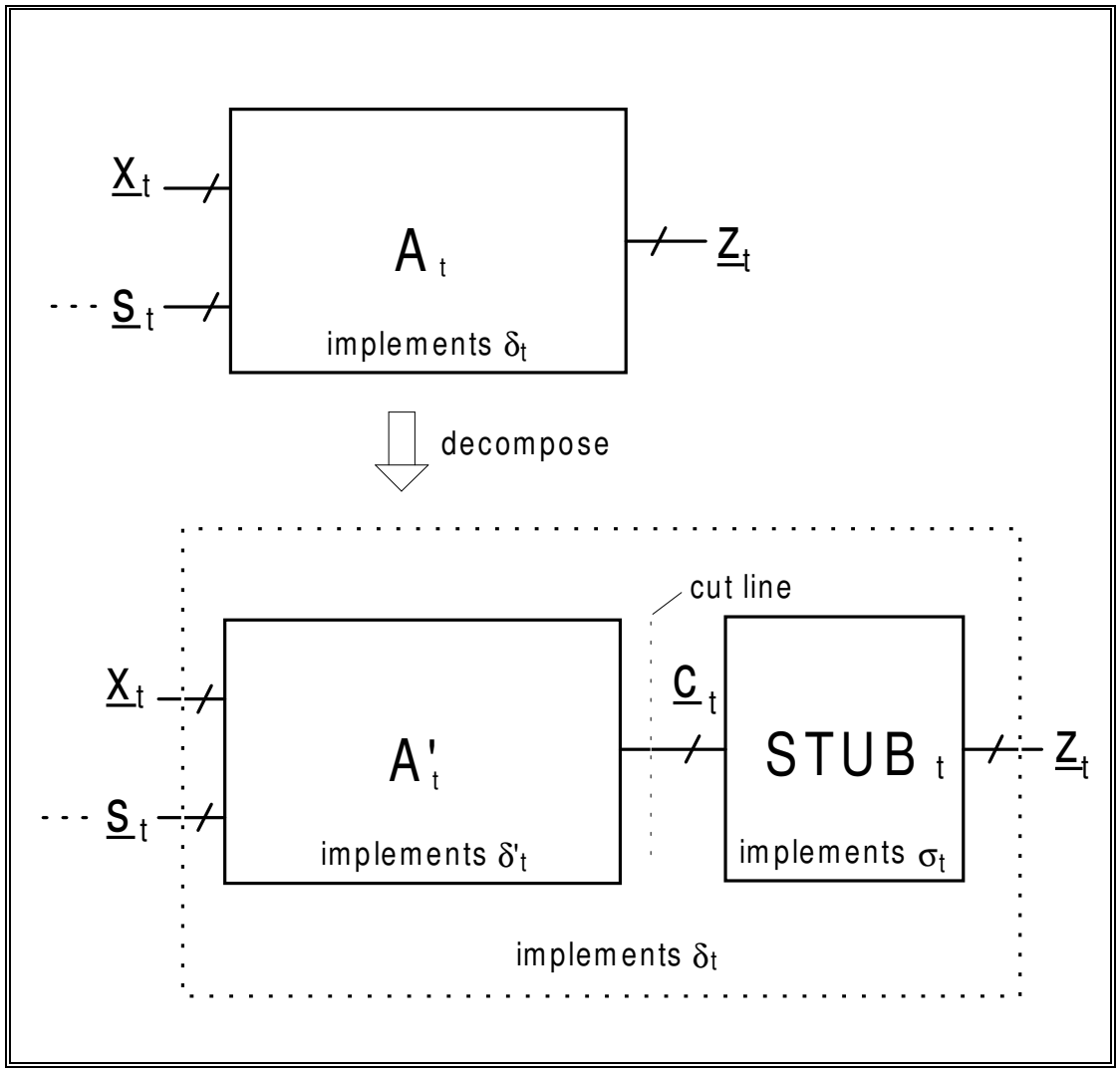
$R(t)$: set of reachable states at the state variables $\underline{s}_t$ at time $t$

**Definition:**

$\sigma_t(\underline{c}_t)$: "*stub function*" with

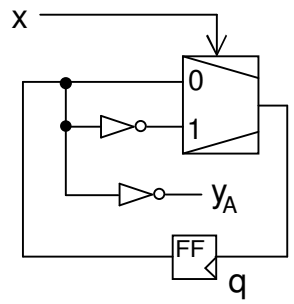
1) $\sigma_t$: $\{0, 1\}^n \rightarrow R(t+1)$

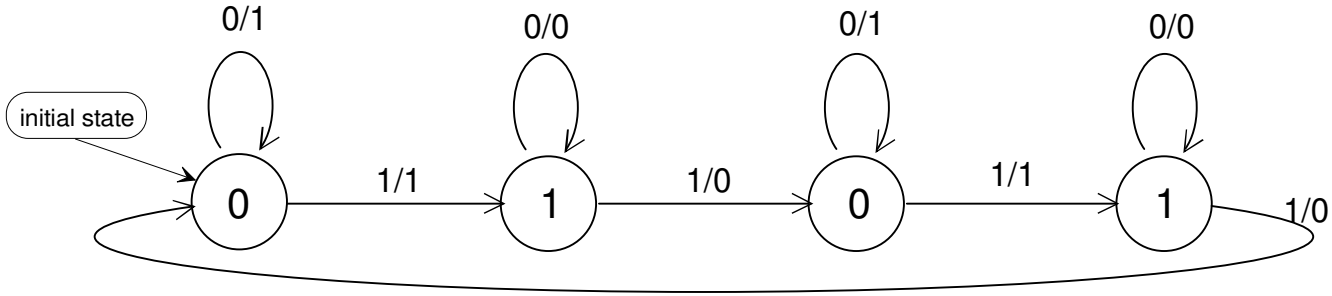
2) it is $\sigma_t(\delta_t'(\underline{s}_t, \underline{x}_t)) = \delta(\underline{s}_t, \underline{x}_t)$

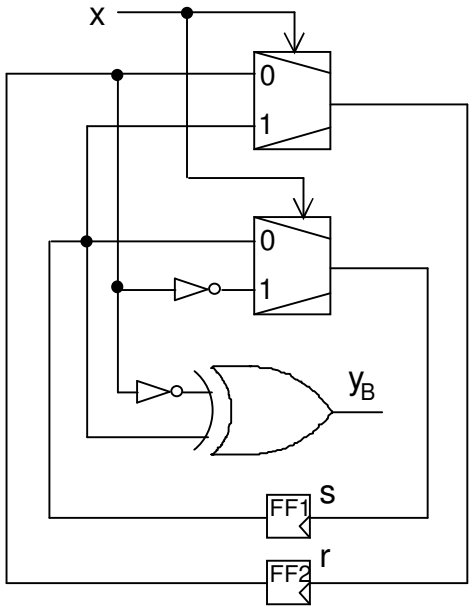


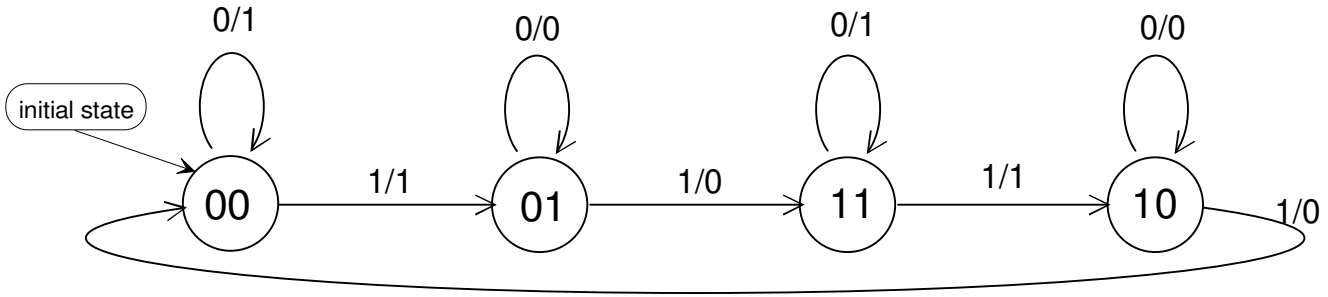

Existential abstraction - in a structural way

# Example

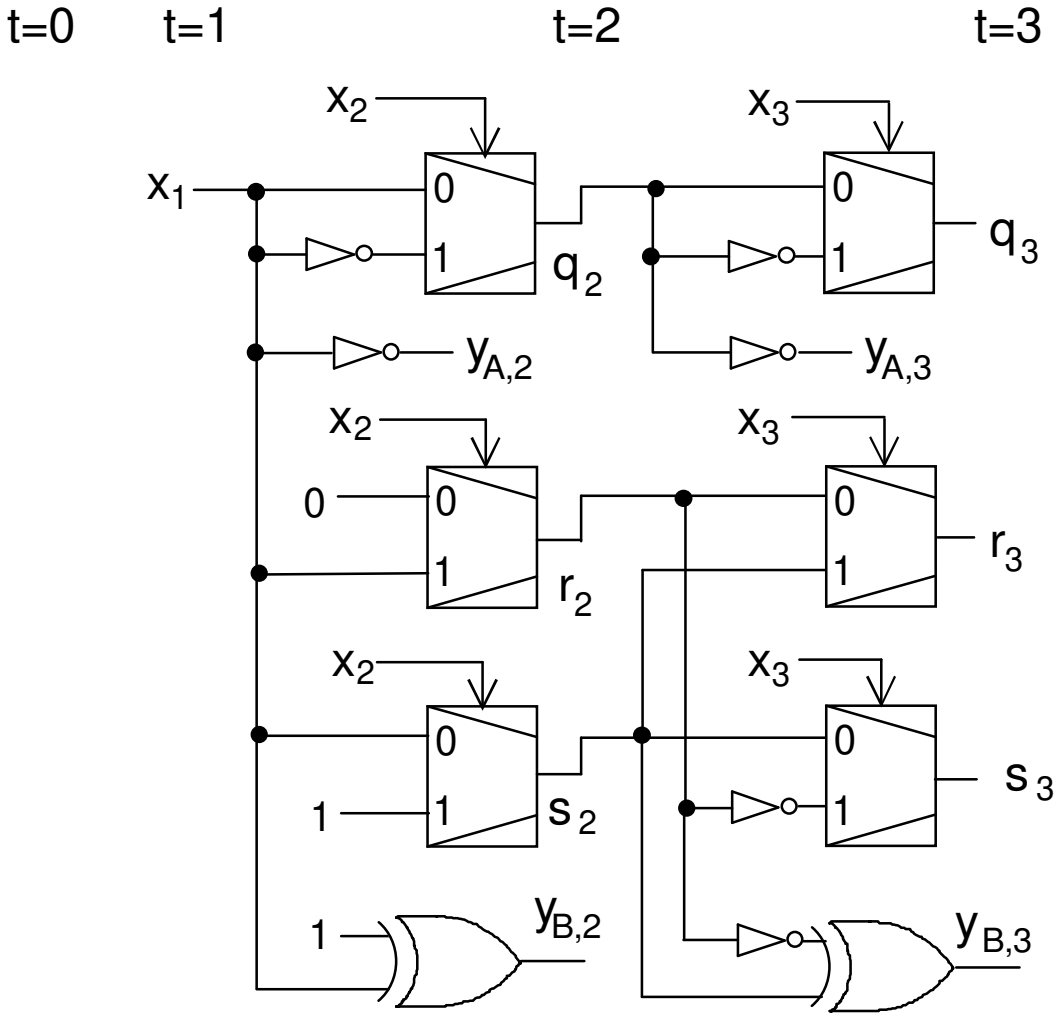

x
0
1
$y_A$
FF
q


0/1
0/0
0/1
0/0
initial state
0
1/1
1
1/0
0
1/1
1
1/0


x
0
1
0
1
$y_B$
FF1
s
FF2
r


0/1
0/0
0/1
0/0
initial state
00
1/1
01
1/0
11
1/1
10
1/0

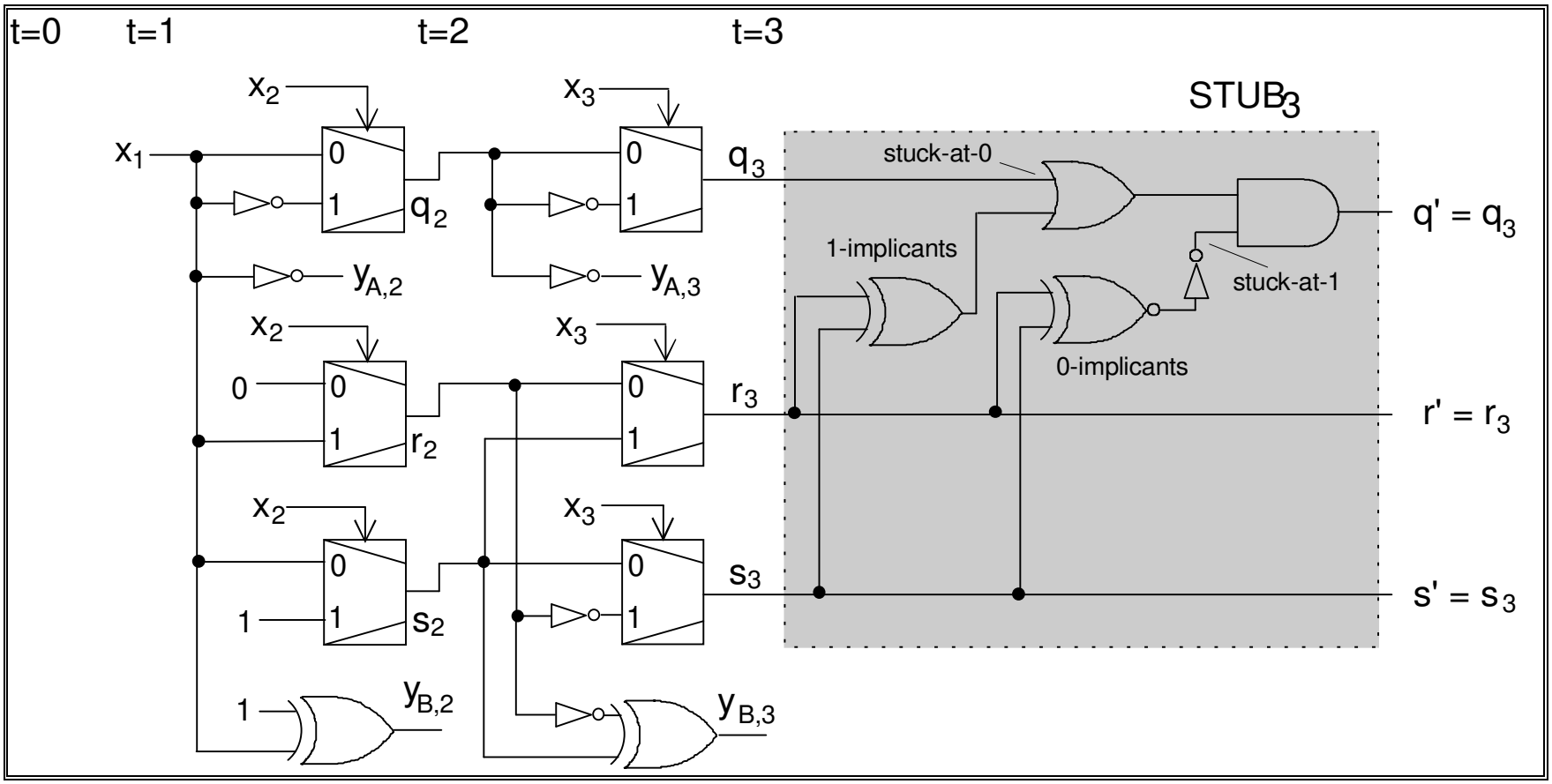
# Example (contd.)



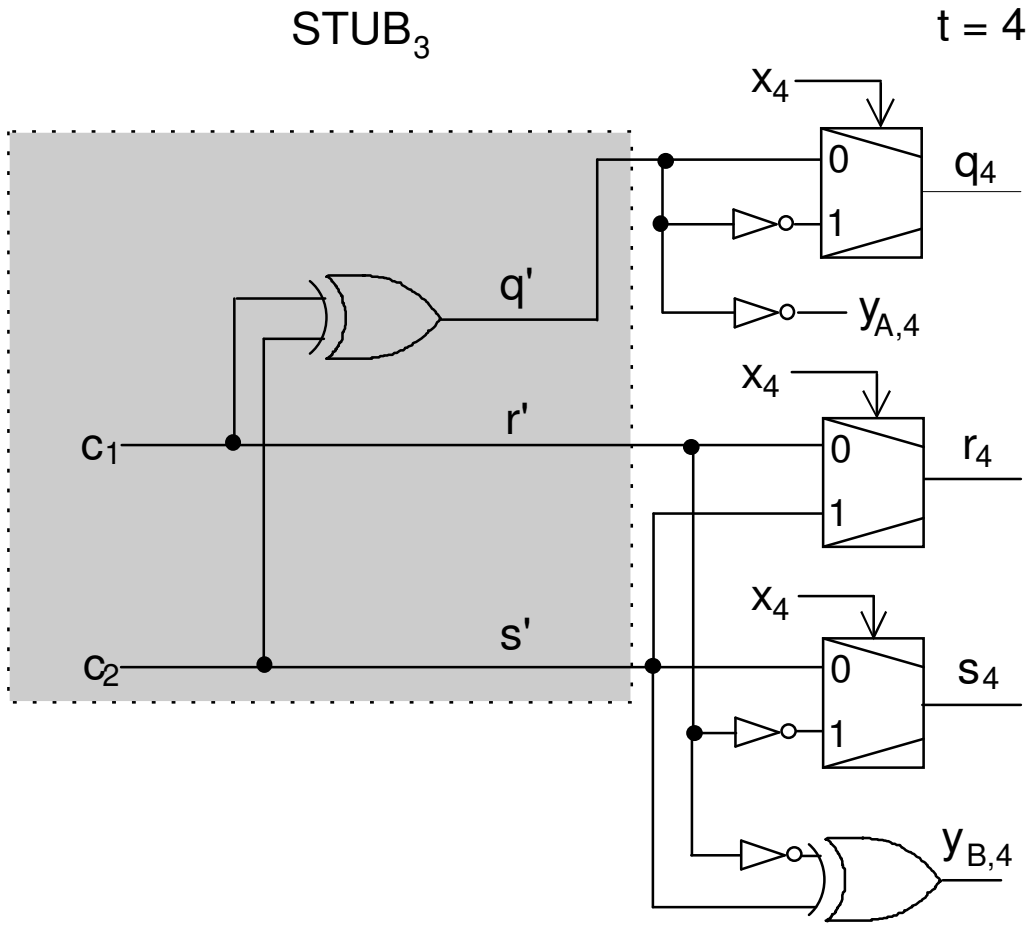

miter array until $t = 3$

# Example (contd.)

| states ($q$, $r$, $s$) reachable at $t = 0$ | states ($q$, $r$, $s$) reachable at $t = 1$ | states ($q$, $r$, $s$) reachable at $t = 2$ | states ($q$, $r$, $s$) reachable at $t = 3$ | implicants |
|---|---|---|---|---|
| 000 | 000<br>101 | 000<br>101<br>011 | 000<br>101<br>011<br>110 | 1-implicants for $q_3$: $\overline{r_3} \cdot s_3$, $r_3 \cdot \overline{s_3}$<br>0-implicants for $q_3$: $r_3 \cdot s_3$, $\overline{r_3} \cdot \overline{s_3}$ |



# Example (contd.)

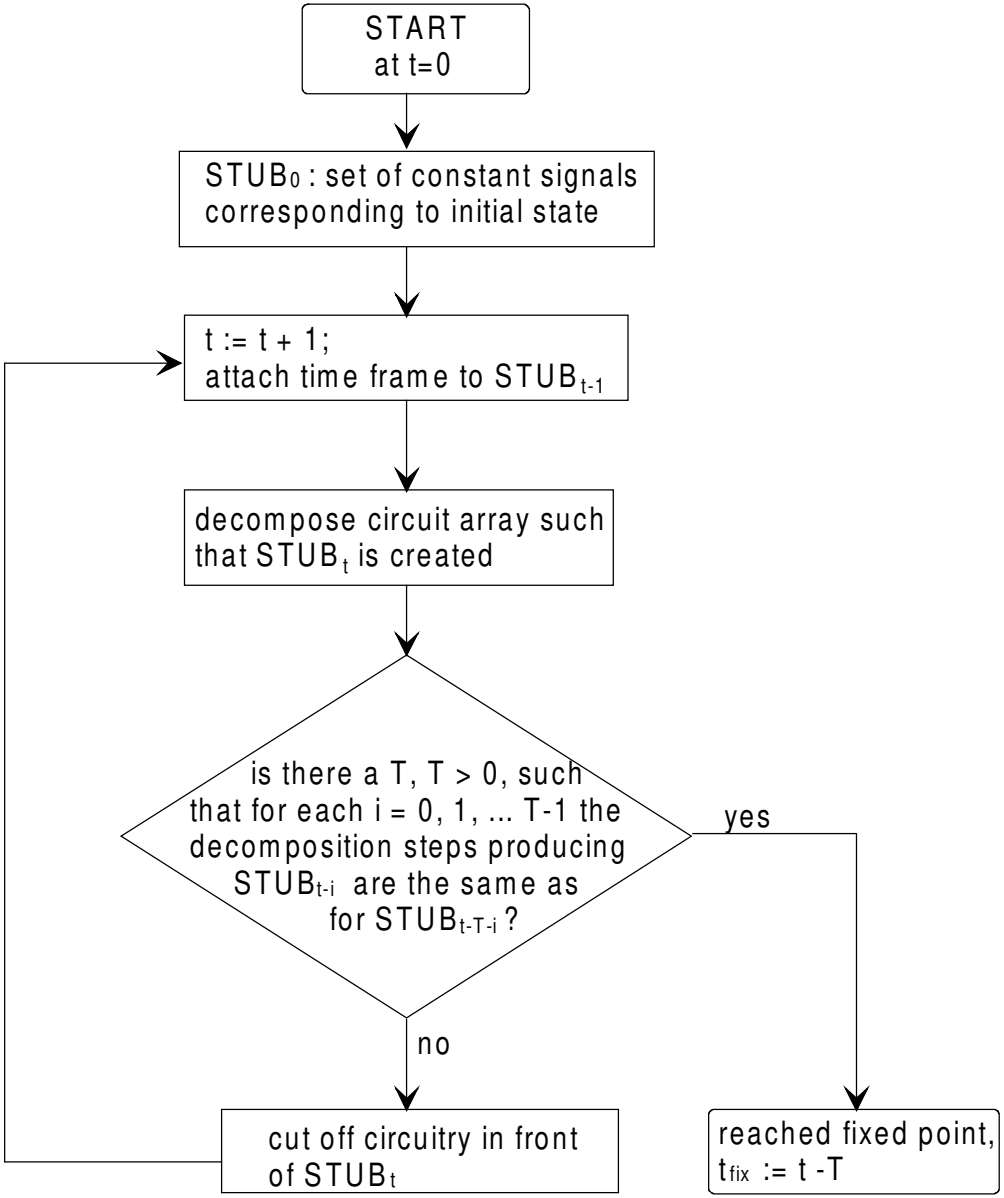



circuit array after cutting in front of stub circuit (existential abstraction)

# *determine_stub_circuit()*

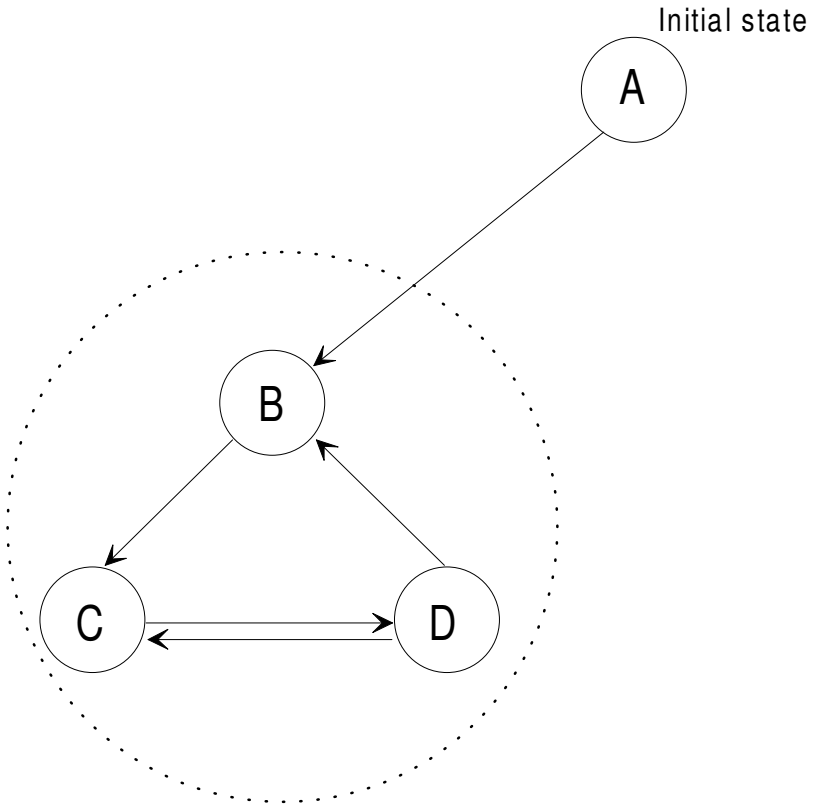
```
/* routine takes as input an iterative circuit array H_t */
determine_stub_circuit(H_t)
{
   L := {z_0,t, z_1,t, z_2,t, ... z_m,t}  /* list of state variables in last time frame */
   for (each state variable z_i,t ∈ L)
   {
         L : = L \ { z_i,t};
         determine all 1-implicants for z_i,t such that all literals
         belong to state variables of L;

         modify H_t by adding 1-implicants to cover at z_i,t;

         determine all 0-implicants for z_i,t such that all literals
         belong to state variables of L;

         modify H_t by adding 0-implicants to cover at z_i,t;
   }

   mark added implicants and fanout systems which feed them;
   cut off all unmarked logic in H_t;
   redundancy removal in H_t;

   return (STUB_t : = H_t);
}
```

# Fixed Point Iteration



# Example

| $t$ | $R(t)$ |
|---|---|
| 0 | $\{A\}$ |
| 1 | $\{B\}$ |
| 2 | $\{C\}$ |
| 3 | $\{D\}$ |
| 4 | $\{B, C\}$ |
| 5 | $\{C, D\}$ |
| 6 | $\{B, C, D\}$ |
| 7 | $\{B, C, D\}$ |
| … | … |



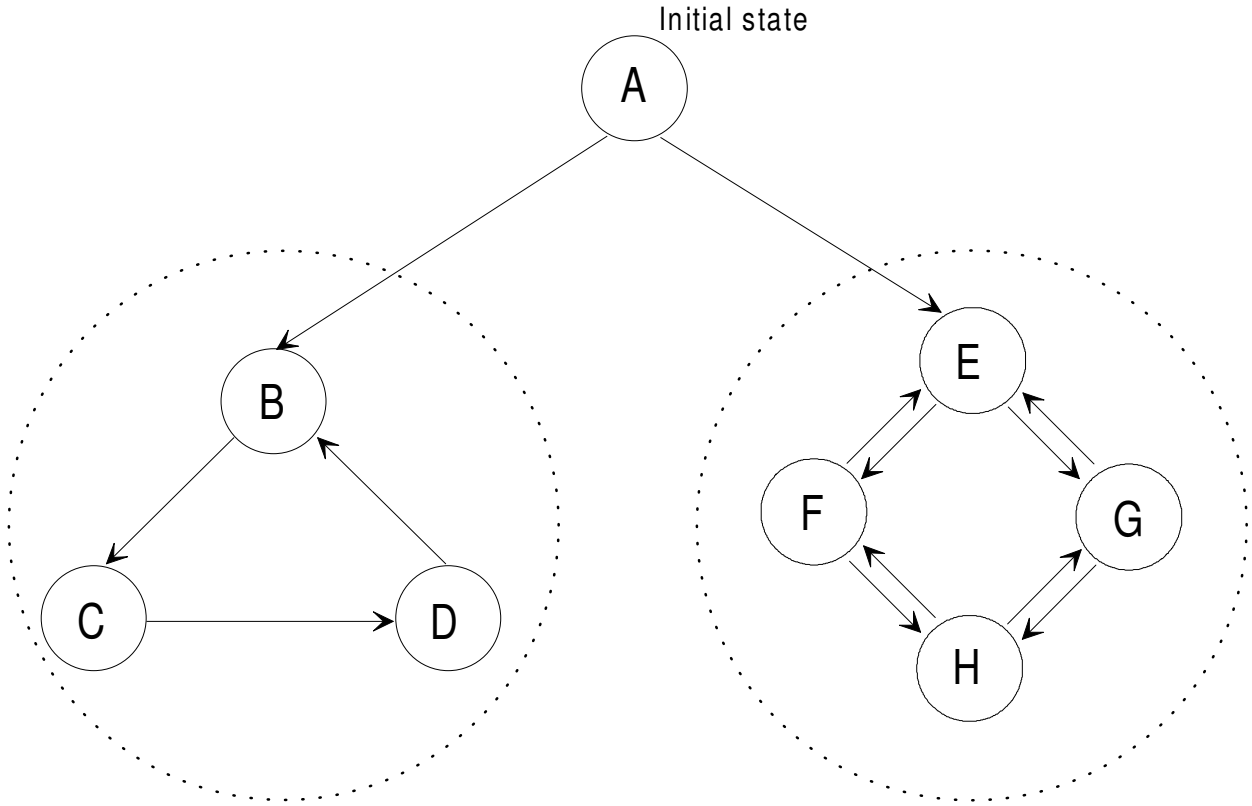
# Theoretical Results

**Definition 2:** If a state $S$ occurs in the reachable state set $R(t)$ repeatedly with a period of $p$ starting at a certain time $t'$, i.e., $S \in R(t' + k \cdot p)$, for all $k \in \{1, 2, ...\}$, we call $p$ the *recurrence period of S* and $t'$ its *birth time*.

**Lemma 1:** After a finite transition time $t'_{SCC}$, the smallest recurrence period $p_{SCC}$ is the same for all states in an SCC. This period $p_{SCC}$ is given by the greatest common divisor (GCD) of all cycle lengths in the SCC and of all recurrence periods for states in the SCC that have been "inherited" from predecessor states outside the SCC.

# Example (two entry-SCCs)

| $t$ | $S(t)$ | $C_j$ |
|---|---|---|
| 0 | $\{A\}$ | - |
| 1 | $\{B, E\}$ | - |
| 2 | $\{C, F, G\}$ | $C_0$ |
| 3 | $\{D, E, H\}$ | $C_1$ |
| 4 | $\{B, F, G\}$ | $C_2$ |
| 5 | $\{C, E, H\}$ | $C_3$ |
| 6 | $\{D, F, G\}$ | $C_4$ |
| 7 | $\{B, E, H\}$ | $C_5$ |
| 8 | $\{C, F, G\}$ | $C_0$ |
| 9 | $\{D, E, H\}$ | $C_1$ |
| … | … | … |



# Theoretical Results

**Definition 3:** An SCC which is entered by initialization or reached exclusively through transient states after initialization is called *entry-SCC*.

**Theorem 1:** Let $P$ be the set of all periodic states of an FSM. There always exists a cover $\{C_0, C_1, \ldots C_{T-1}\}$ of $P$ with $C_k \in 2^P$ and there is a time $0 \leq t_{fix} < \infty$, such that

$$R(t_{fix} + m \cdot T + k) = C_k,\ m \in \{0, 1, 2, \ldots\},\ 0 \leq k \leq T - 1$$

and it is $T = \text{LCM}(p_1, p_2, \ldots, p_n)$, where $p_i$ are the periods of the entry-SCCs of the FSM. The period $T$ is called *fixed point oscillation period*.

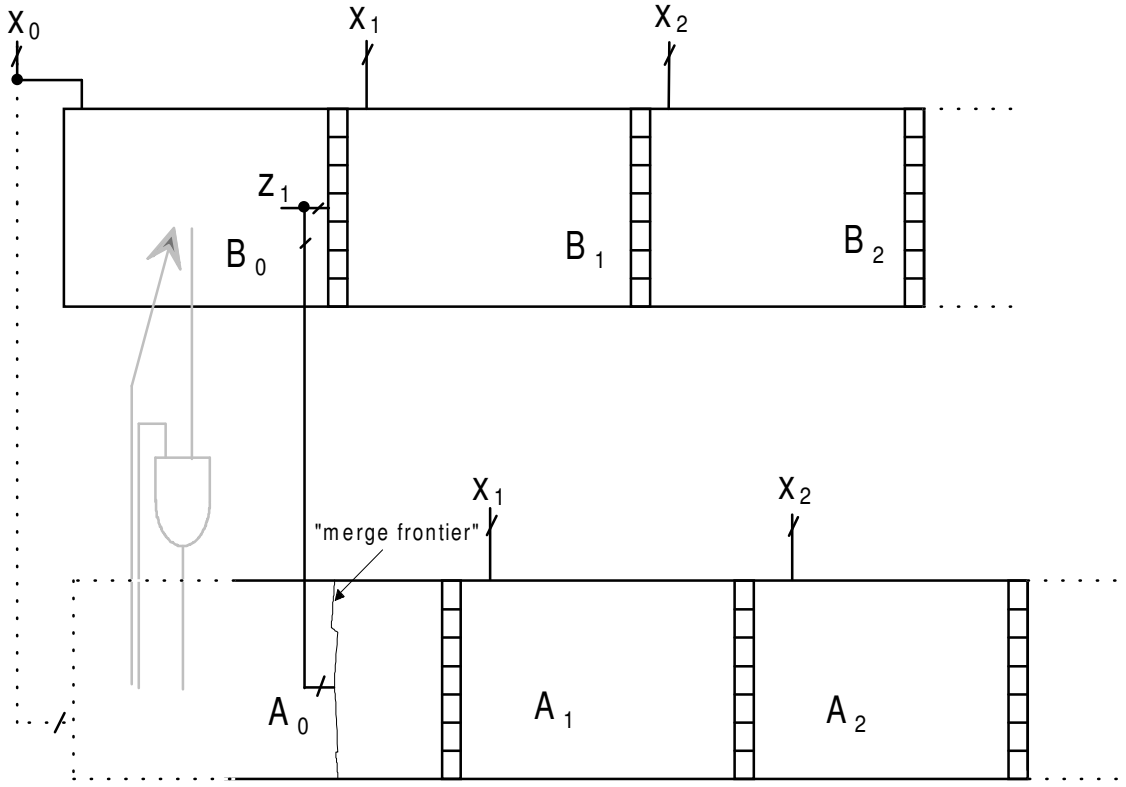
# Theoretical Results

**Corollary 1:** If a circuit has a synchronizing sequence of length *l* the fixed point oscillation period *T* of the machine is upper-bounded by *l* .

**Corollary 2:** If a sequential circuit has an input to synchronously reset all state variables, the fixed point oscillation period *T* of the machine is exactly 1. The time needed to reach the fixed point is given by the *sequential depth* of the machine.

# 3 Record & Play

- start at initial state
- "merge" circuitry of two circuits moving from time frame to time frame




fixed point ?

# Instruction Queue (FIFO)

**Instruction queue (Q):** ordered set of instructions for circuit transformations; Q is processed in a first-in-first-out (FIFO) manner.

**record:** store all circuit transformations of current time frame in Q

**play:** check validity of each instruction in Q for current time frame and perform transformation if valid

# Finding a Fixed Point

**Problem 1:** find an instruction queue that works for all time frames (basis for induction)

**Problem 2:** if such an instruction queue is given, prove that it works for all time frames (induction)

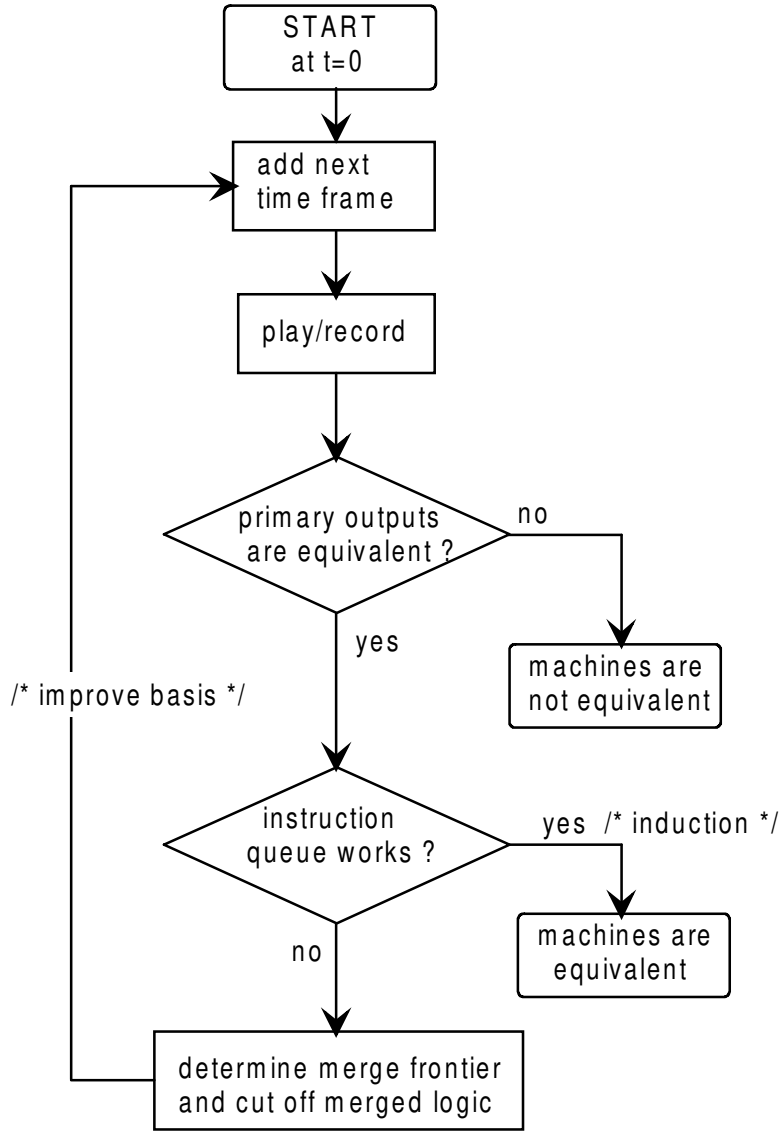
**solution for 1:** *record* and *play*

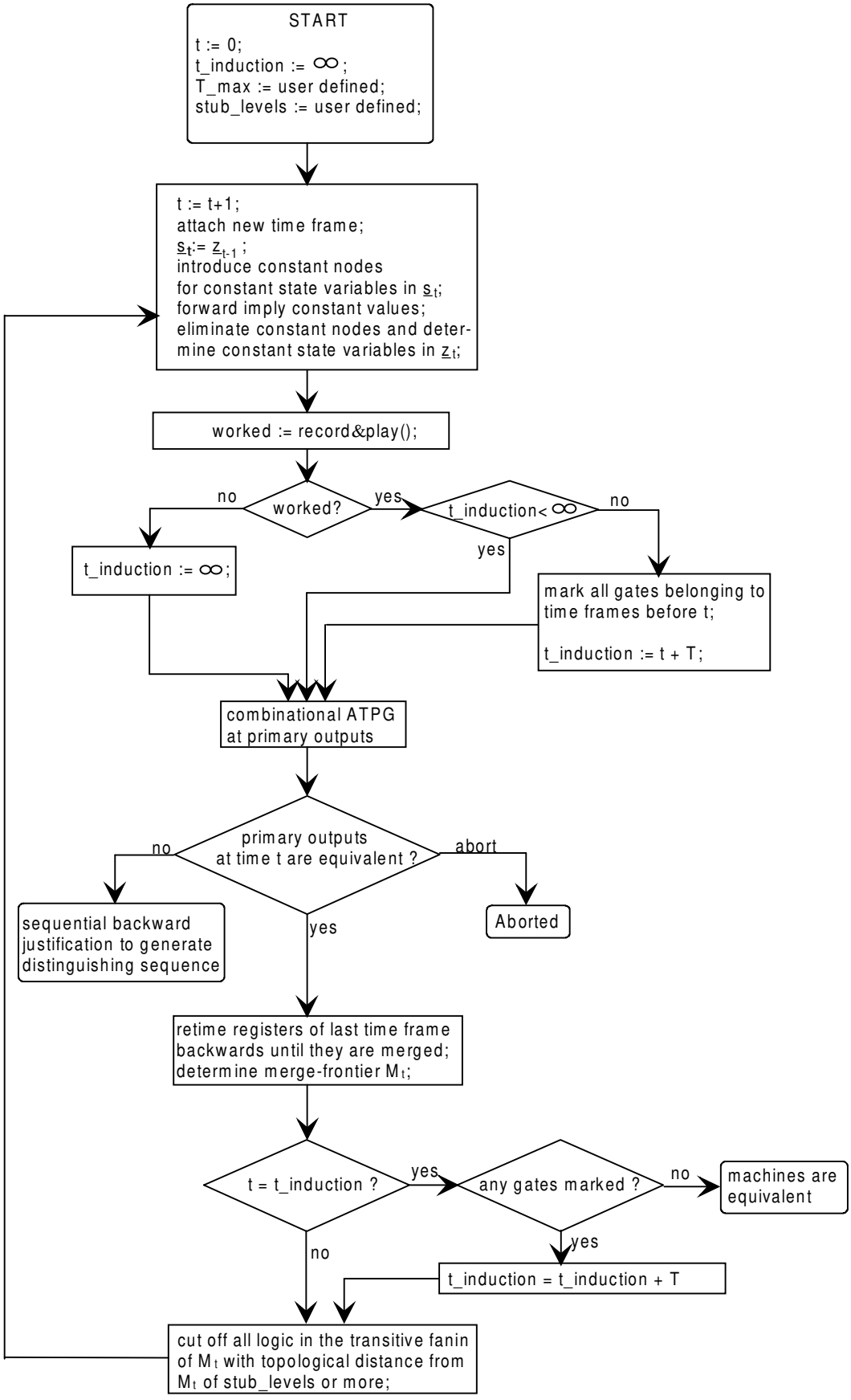
store circuit transformations that minimize miter in instruction queue (*record*), try to use these transformations in subsequent time frames (*play*), remove bad instructions and add new instructions.

**solution for 2:** *find merge frontier* and *cut*

identify a merge-frontier by retiming the registers of the last time frame such that registers of different circuits are "merged". Cut off circuitry behind merge frontier and prove that instruction queue is still working *(play*).

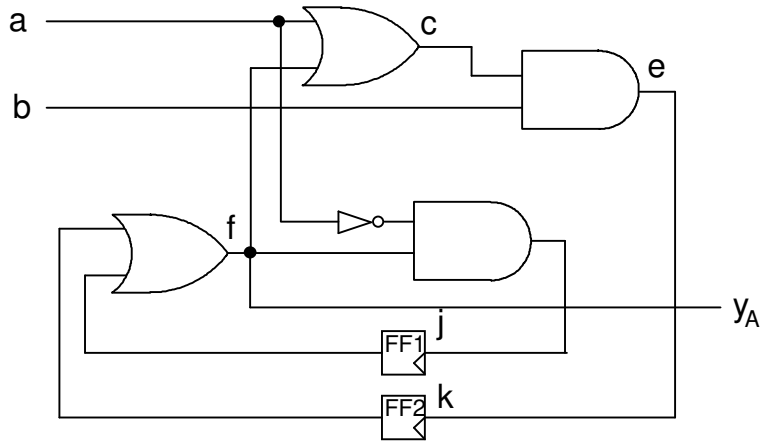
# Program Flow



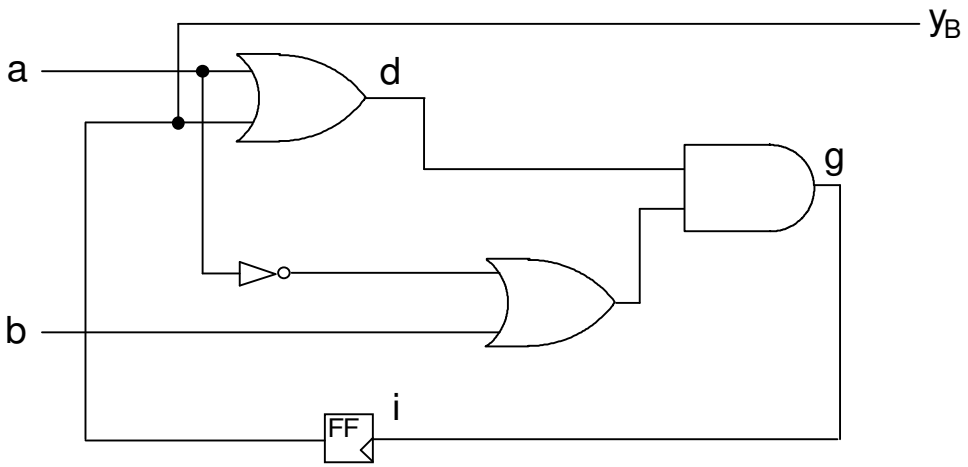

START
t := 0;
t_induction := ∞;
T_max := user defined;
stub_levels := user defined;
t := t+1;
attach new time frame;
s_t := z_t-1 ;
introduce constant nodes
for constant state variables in s_t;
forward imply constant values;
eliminate constant nodes and deter-
mine constant state variables in z_t;
worked := record&play();
worked?
no
yes
t_induction< ∞
no
yes
t_induction := ∞;
mark all gates belonging to
time frames before t;
t_induction := t + T;
combinational ATPG
at primary outputs
primary outputs
at time t are equivalent ?
no
abort
yes
sequential backward
justification to generate
distinguishing sequence
Aborted
retime registers of last time frame
backwards until they are merged;
determine merge-frontier M_t;
t = t_induction ?
yes
any gates marked ?
no
machines are
equivalent
yes
no
t_induction = t_induction + T
cut off all logic in the transitive fanin
of M_t with topological distance from
M_t of stub_levels or more;

# 4 Example

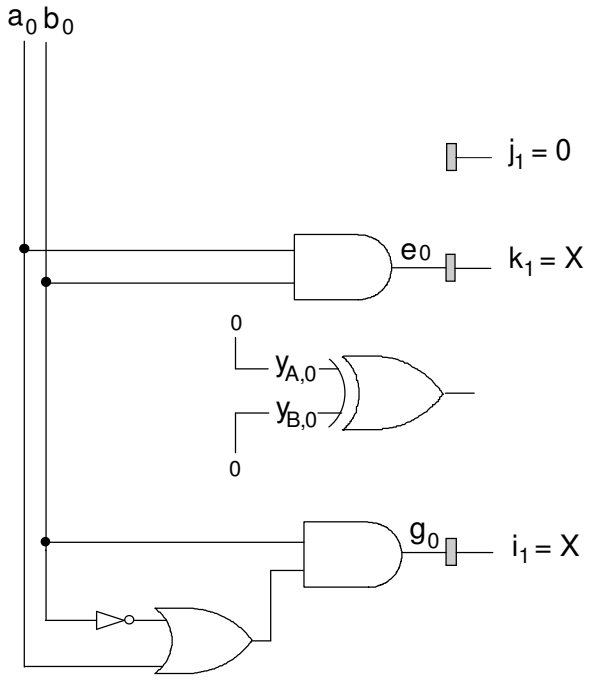
Machine *A*, $\underline{I}_A = 0$




Machine *B*, $\underline{I}_B = 0$

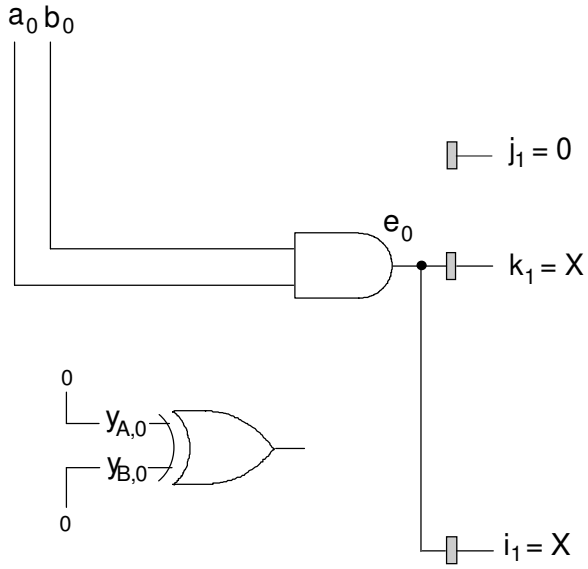


first time frame,
assign initial state



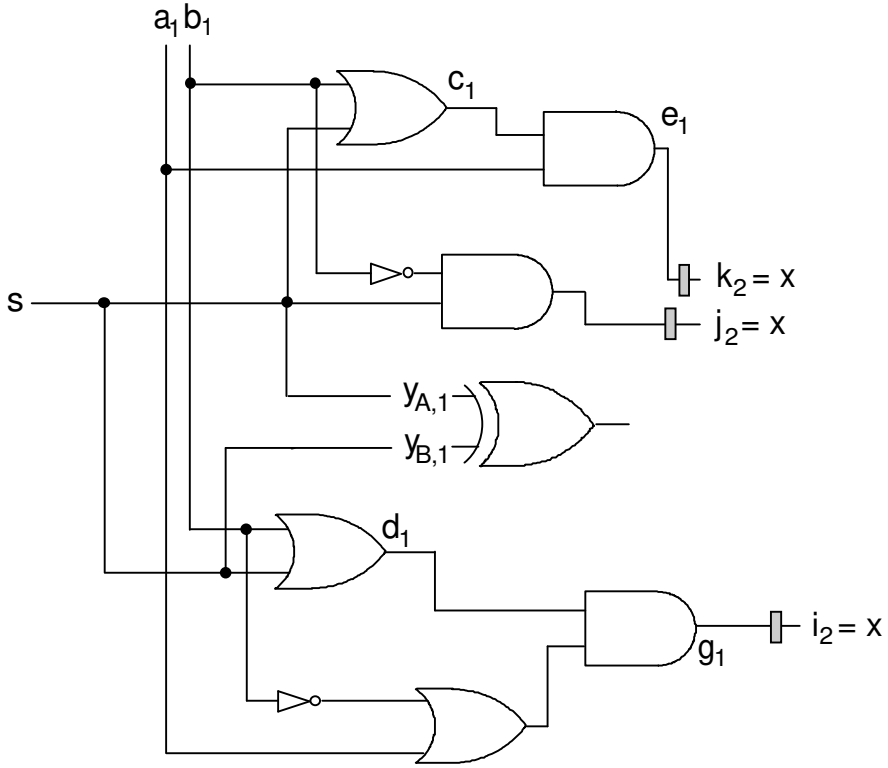

$\underline{z}_{A,0} = \underline{I}_A = (j_0 = 0, k_0 = 0)$
$\underline{z}_{B,0} = \underline{I}_B = (i_0 = 0)$

merge logic,
determine merge frontier



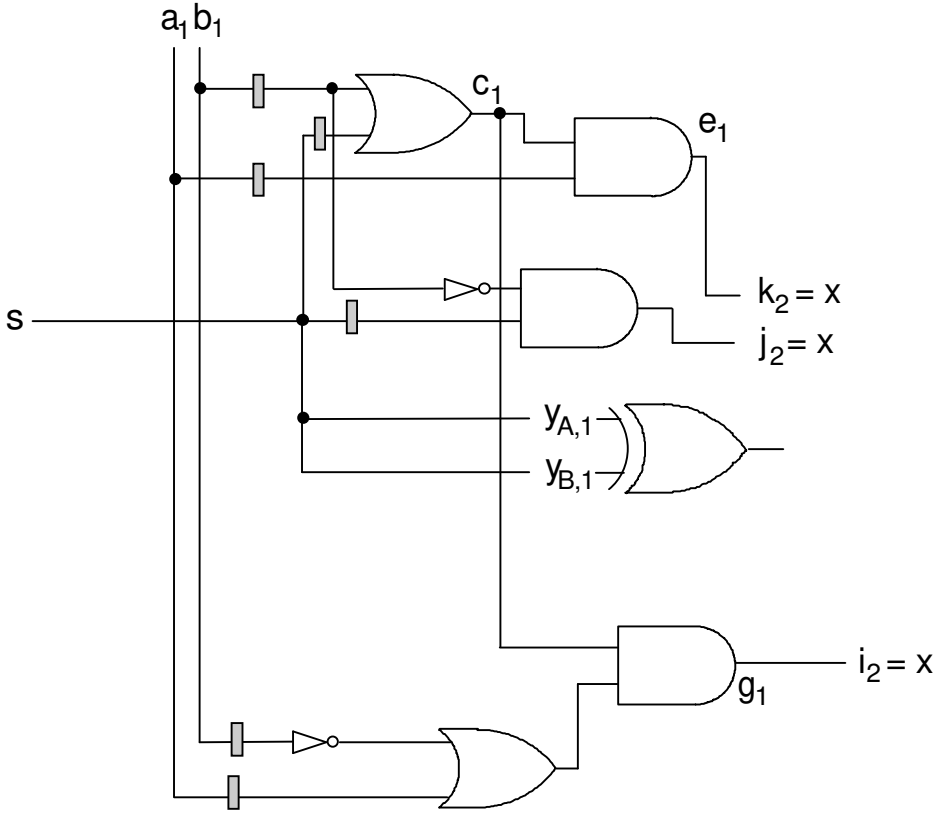

$Q_0 = \{$(substitute $g_t$ by $e_t$)$\}$

cut off merged logic,
attach new time frame



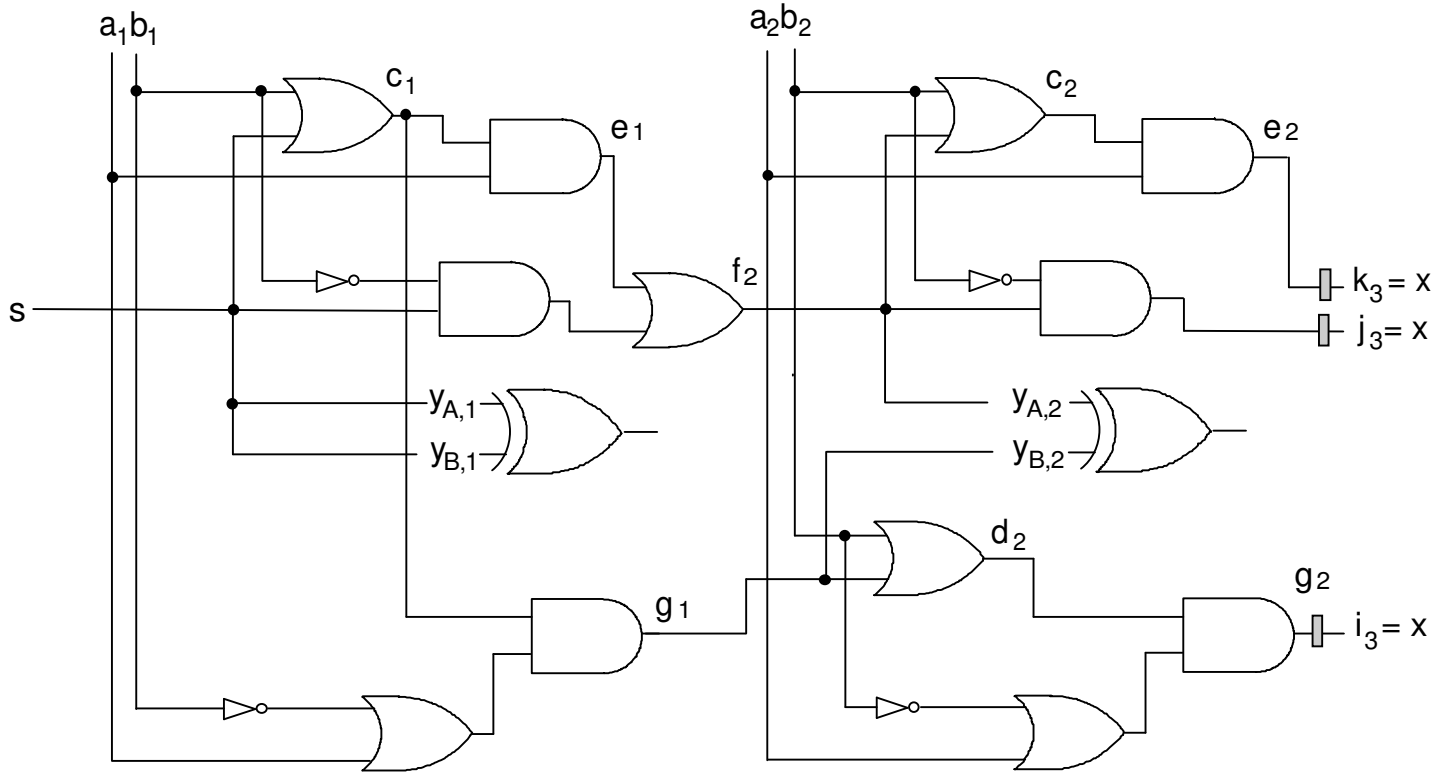

$\underline{z}_{A,1} = (j_1 = 0,\ k_1 = x)$
$\underline{z}_{B,1} = (i_1 = x)$

merge logic,
determine merge frontier




$Q_1 = \{(\text{substitute } d_t \text{ by } c_t)\}$

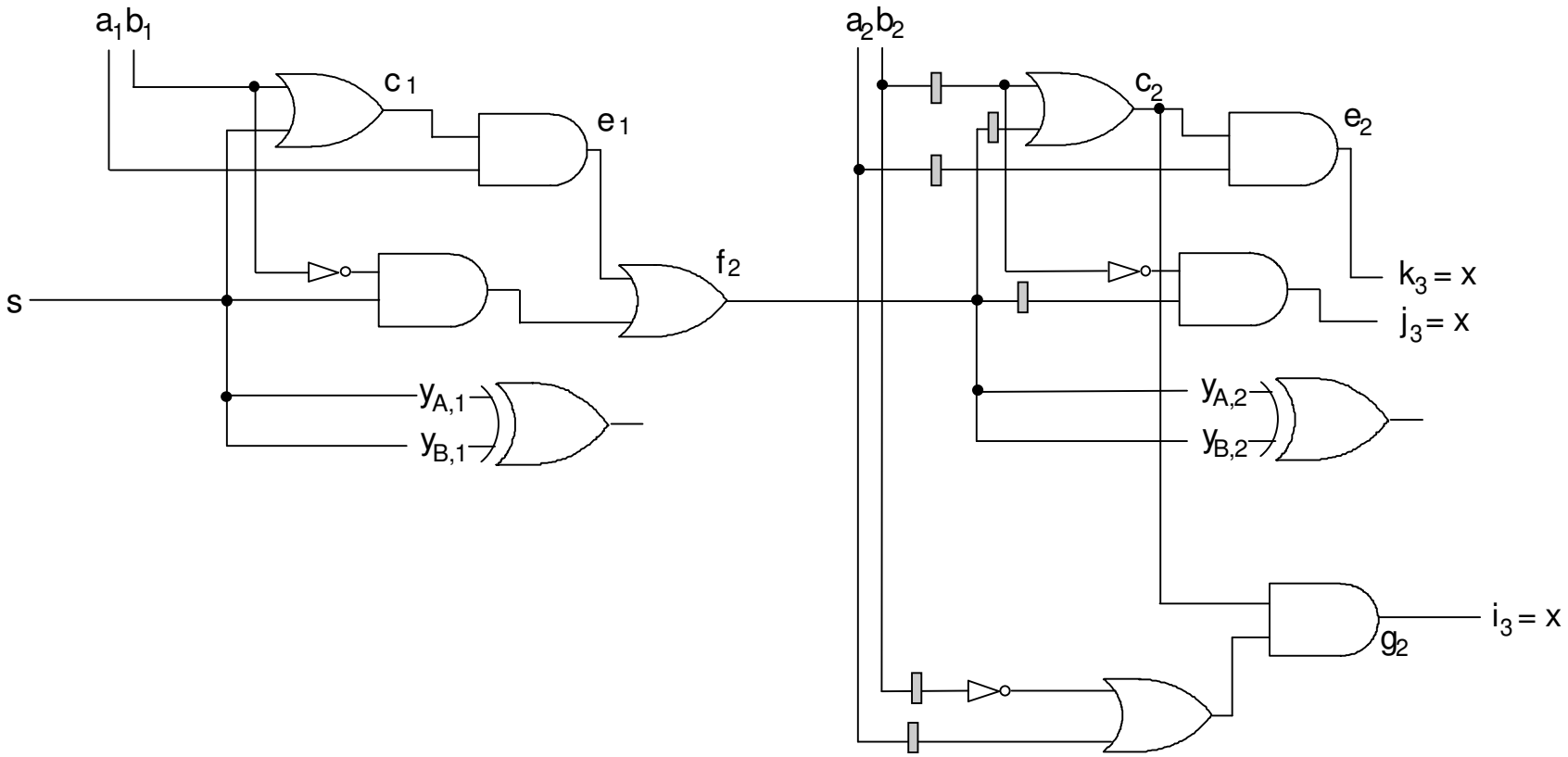
cut off merged logic, attach new time frame




$\underline{Z}_{A,2} = (j_2 = x,\ k_2 = x)$

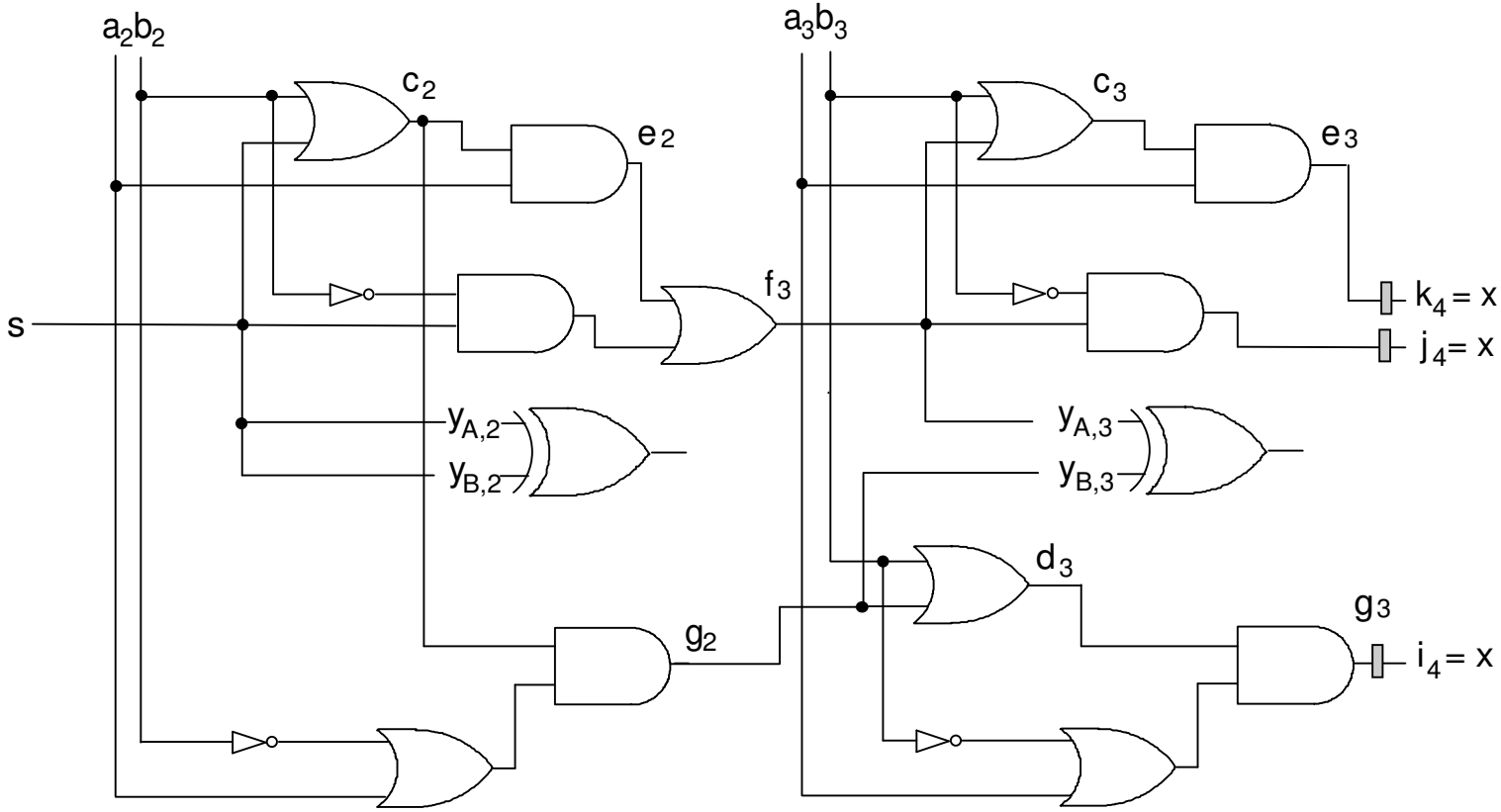
$\underline{Z}_{B,2} = (i_2 = x)$

merge logic, determine merge frontier



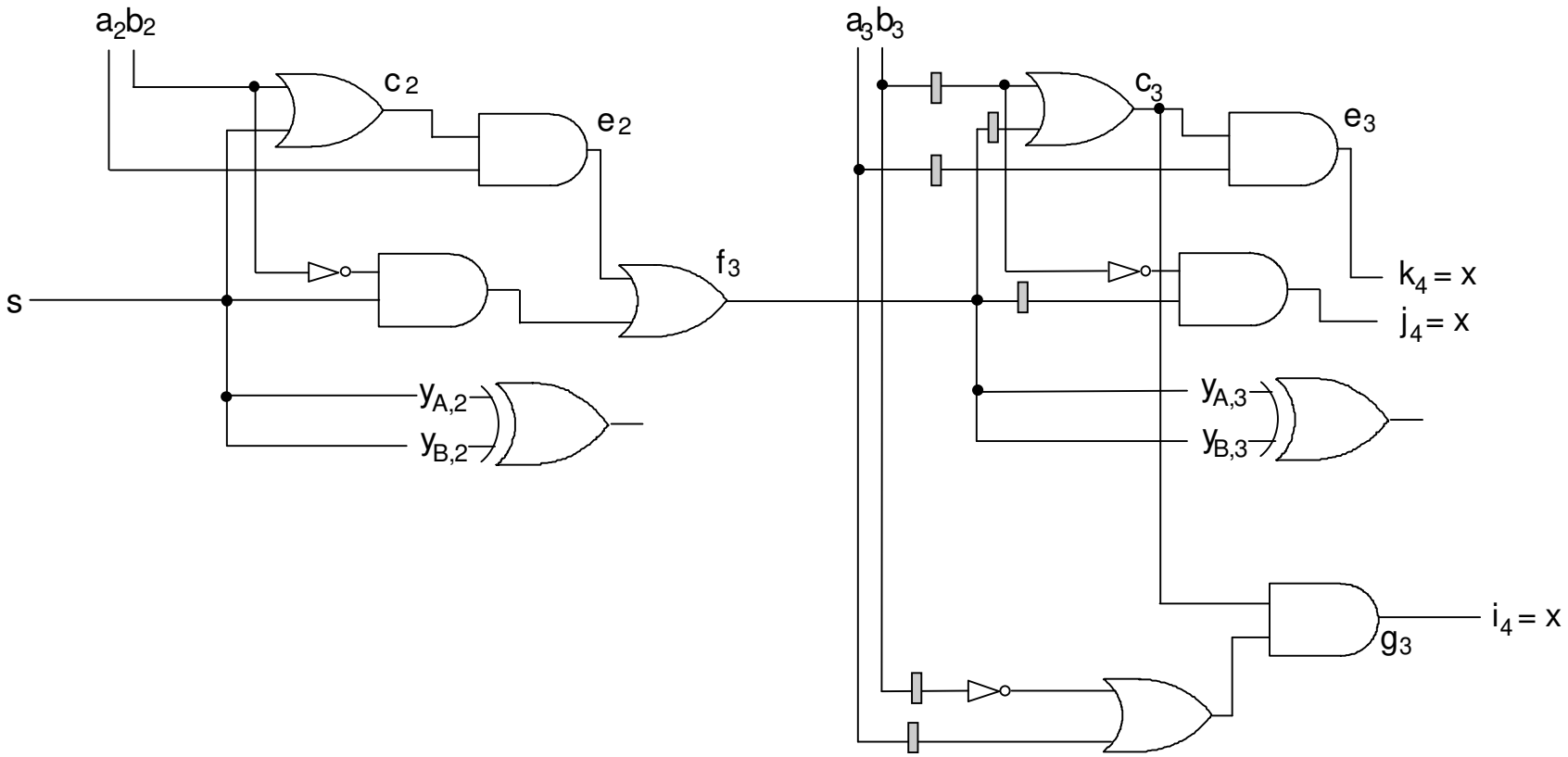

$Q_2$ = {(substitute $g_{t\text{-}1}$ by $f_t$), (substitute $d_t$ by $c_t$)}

cut off merged logic, attach new time frame




$\underline{Z}_{A,3} = (j_3 = \mathrm{x},\ k_3 = \mathrm{x})$
$\underline{Z}_{B,3} = (i_3 = \mathrm{x})$

merge logic, determine merge frontier




playing $Q_2$ worked, $Q_3 = Q_2$

## 5 Experimental Results

| Circuit Name | # registers | ***record&play*** (HANNIBAL) | | ***verify_fsm*** (SIS) |
|---|---|---|---|---|
| | | # iterations til fixed point | CPU-time (h:min:sec) | CPU-time (h:min:sec) |
| s208 | 8 | 15 | 0: 00: 08 | 0: 00: 03 |
| s298 | 14 | 10 | 0: 00: 09 | 0: 00: 03 |
| s344 | 15 | 9 | 0: 00: 11 | 0: 00 06 |
| s349 | 15 | 9 | 0: 00: 11 | 0: 00: 06 |
| s382 | 21 | 16 | 0: 00: 17 | 0: 00: 38 |
| s386 | 6 | 9 | 0: 00: 48 | 0: 00: 01 |
| s420 | 16 | 27 | 0: 00: 43 | 0: 27: 19 |
| s444 | 21 | 16 | 0: 00: 18 | 0: 00: 28 |
| s510 | 6 | 12 | 0: 00: 35 | 0: 00: 02 |
| s526 | 21 | 21 | 0: 00: 35 | 0: 00: 12 |
| s635 | 32 | 37 | 0: 01: 32 | unable |
| s641 | 19 | 9 | 0: 00: 12 | 0: 00: 08 |
| s713 | 19 | 9 | 0: 00: 12 | 0: 00: 09 |
| s820 | 5 | 17 | 0: 36: 50 | 0: 00: 04 |
| s832 | 5 | 16 | 0: 26: 37 | 0: 00: 04 |
| s838 | 32 | 51 | 0: 08: 13 | unable |
| s953 | 29 | 11 | 0: 01: 09 | 0: 00: 12 |
| s1196 | 18 | 6 | 0: 00: 40 | 0: 00: 08 |
| s1238 | 18 | 6 | 0: 00: 46 | 0: 00: 08 |
| s1423 | 74 | 14 | 0: 03: 31 | unable |
| s1512 | 57 | 16 | 0: 04: 09 | > 30 h |
| s3271 | 116 | 19 | 0: 21: 17 | unable |
| s3330 | 132 | 9 | 0: 11: 33 | unable |
| s3384 | 183 | 17 | 0: 31: 24 | unable |
| s4863 | 104 | 8 | 0: 36: 52 | unable |
| s5378 | 179 | 36 | 0: 55: 23 | unable |
| s6669 | 239 | 11 | 0: 47: 15 | unable |

ISCAS 89 circuits after kernel extraction and retiming versus original

# CONCLUSION

Structural FSM traversal: fixed point iteration based on structural concepts and techniques

Record & Play: a practical implementation of structural FSM traversal

- promising approach for verifying circuits after synthesis with retiming
- applications in model checking ?

ERROR: undefined
OFFENDING COMMAND: 9çÝ6ß

STACK: