Circuit Folding: From Combinational to Sequential Circuits

Presenter: 錢柏均

Advisor: Jie-Hong Roland Jiang

ALCom Lab

Graduate Institute of Electronics Engineering
Department of Electrical Engineering
National Taiwan University

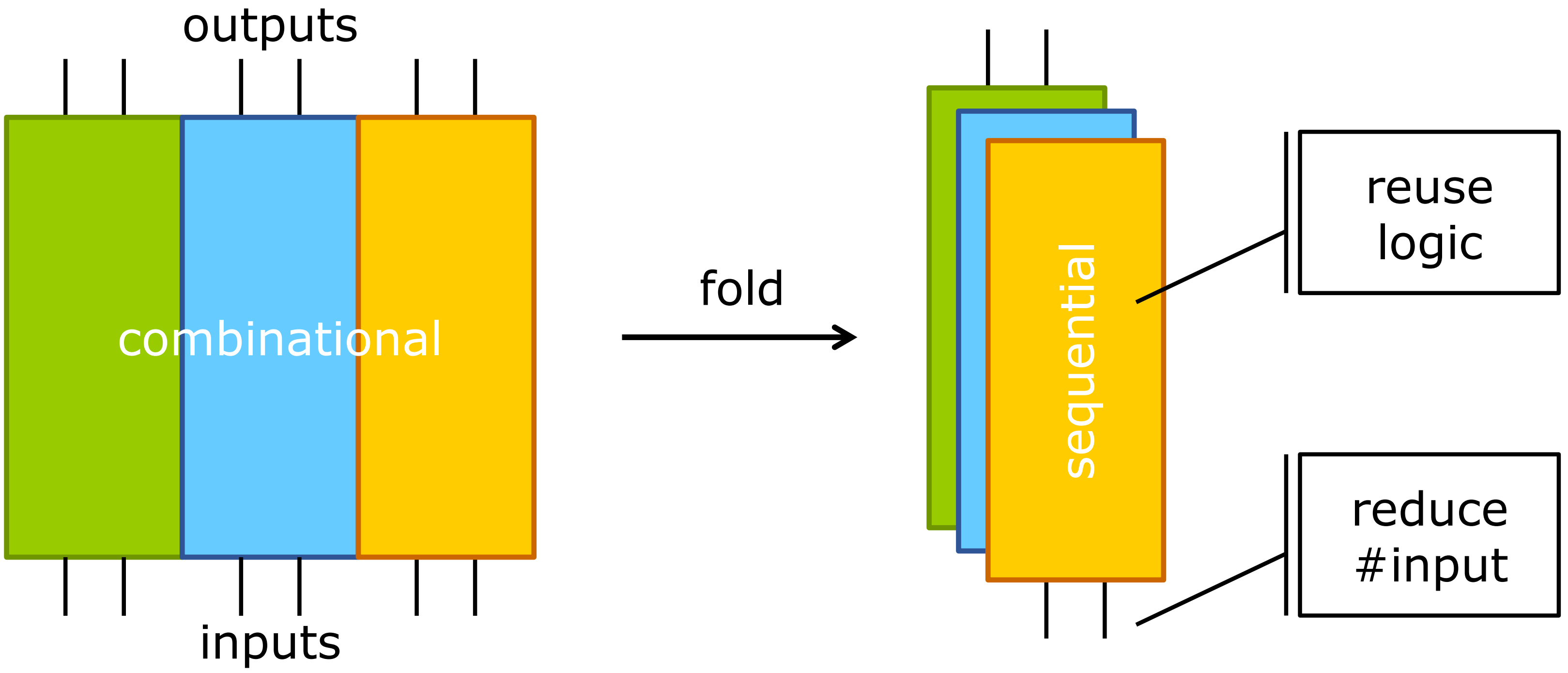
Outline

- Introduction
- Time-frame Folding (TFF)
  - state identification & transition reconstruction
- Time Multiplexing via Circuit Folding
  - structural method
  - functional method based on TFF
- Experiments
  - TFF for circuit compaction
  - structural vs. functional method
- Conclusions & Future Work

INTRODUCTION

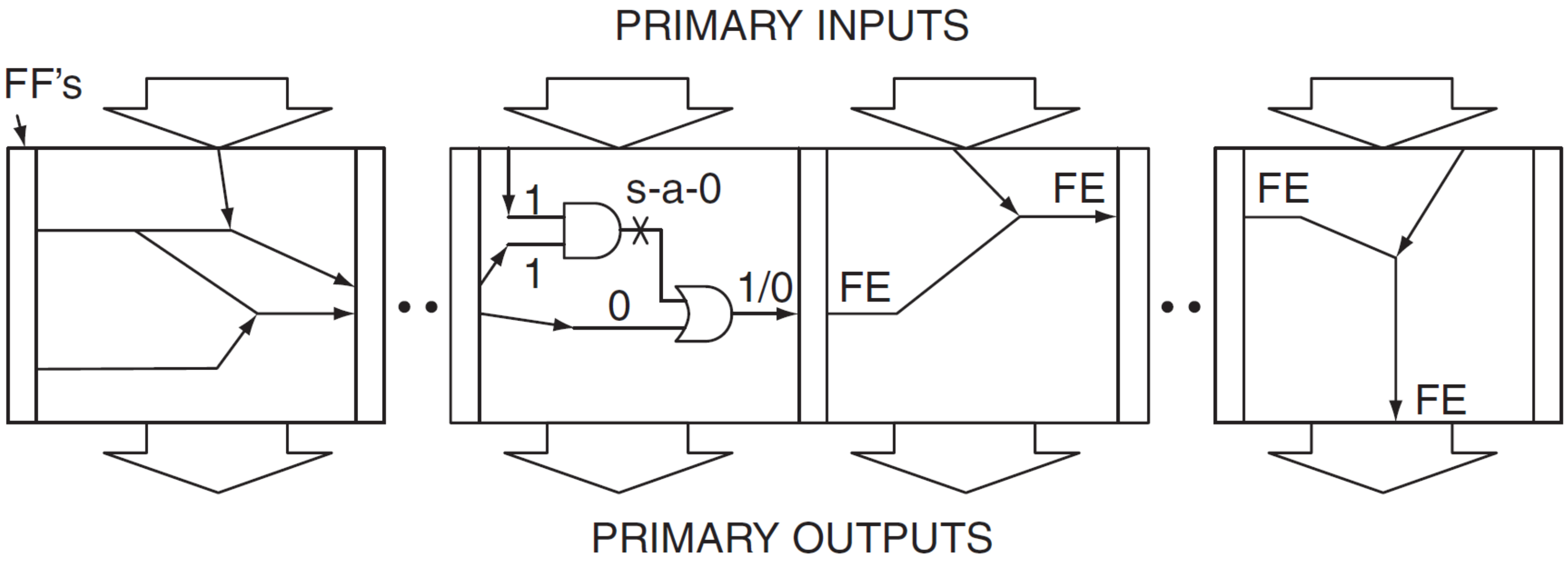
Circuit Folding Illustration

- Circuit folding is a process of transforming a combinational circuit C_c into a sequential circuit C_s, which after time-frame expansion, is functionally equivalent to C_c.



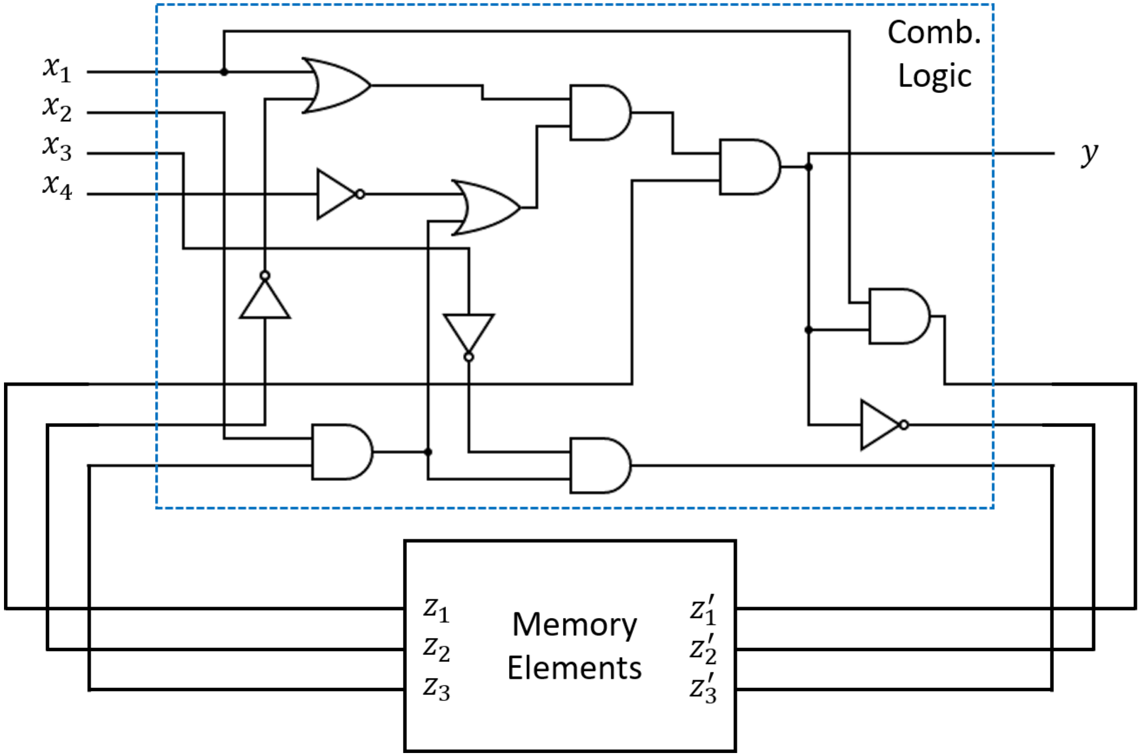
Time-Frame Unfolding

- TFU, or time-frame expansion
- A technique often used in ATPG, BMC



Time-Frame Unfolding

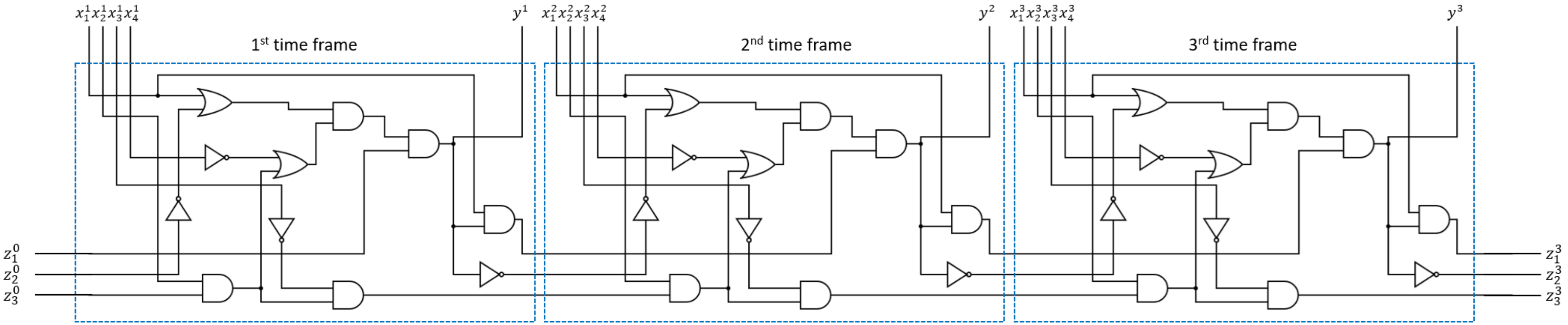
- An example sequential circuit



Sequential circuit *s*27

Time-Frame Unfolding

- Expand 3 time-frames

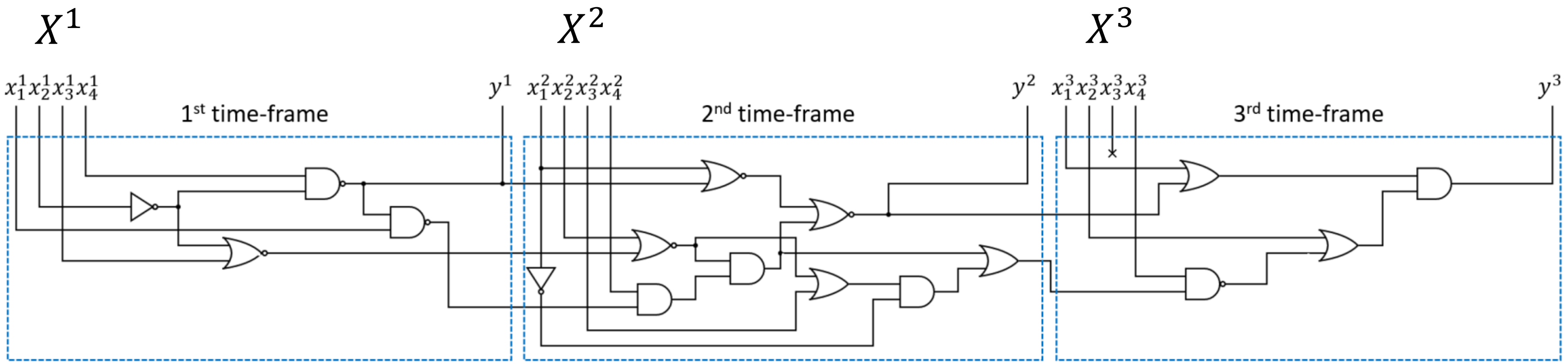


Regular duplication

with flip-flops from consecutive time-frames connected

Time-Frame Unfolding

- Expand 3 time-frames


X^1
X^2
X^3
$x_1^1 x_2^1 x_3^1 x_4^1$
y^1
$x_1^2 x_2^2 x_3^2 x_4^2$
y^2
$x_1^3 x_2^3 x_3^3 x_4^3$
y^3
1st time-frame
2nd time-frame
3rd time-frame

with initial state propagation and simplification

$$y^1 = f(\underline{X^1}) \qquad y^2 = g(\underline{X^1, X^2}) \qquad y^3 = h(\underline{X^1, X^2, X^3})$$

Can we reverse it?

Iterative form

Motivation of TFF

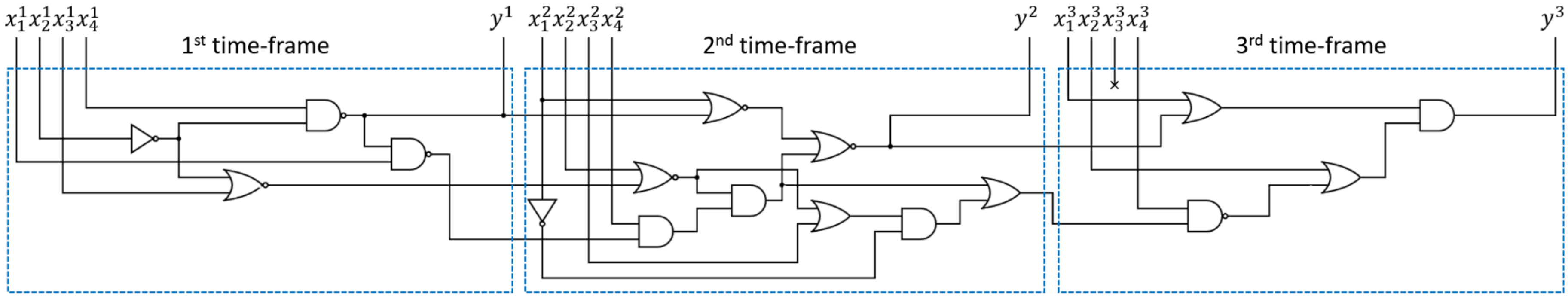
- In model-based testing of software systems [1, 2], one may be asked to compute synchronizing, distinguishing, or homing sequences.
- These problems can be formulated as quantified Boolean formula (QBF) [3, 4] solving of strategy derivation.
- The derived strategy corresponds to **a large (iterative) combinational circuit.** However, it can be alternatively represented **more compactly by a sequential circuit**.
- How can one reconstruct a sequential circuit from an iterative combinational circuit?

[1] Sandberg et al., 2005. [2] Kushiket al., 2016. [3] Bieree t al., 2009. [4] Wang et al., 2017.

TFF Formulation

- TFF is a reverse operation of TFU

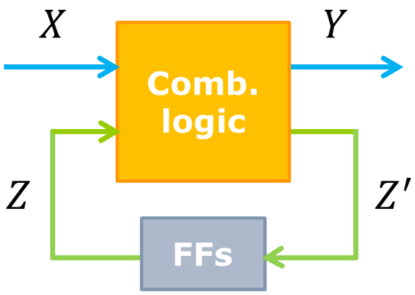
Given: a k-iterative combinational circuit



Goal: obtain a sequential circuit that

- is equivalent within bounded k time-frames
- has minimized finite state machine (FSM)

(no assumption is made on the circuit structure except for the iterative form)



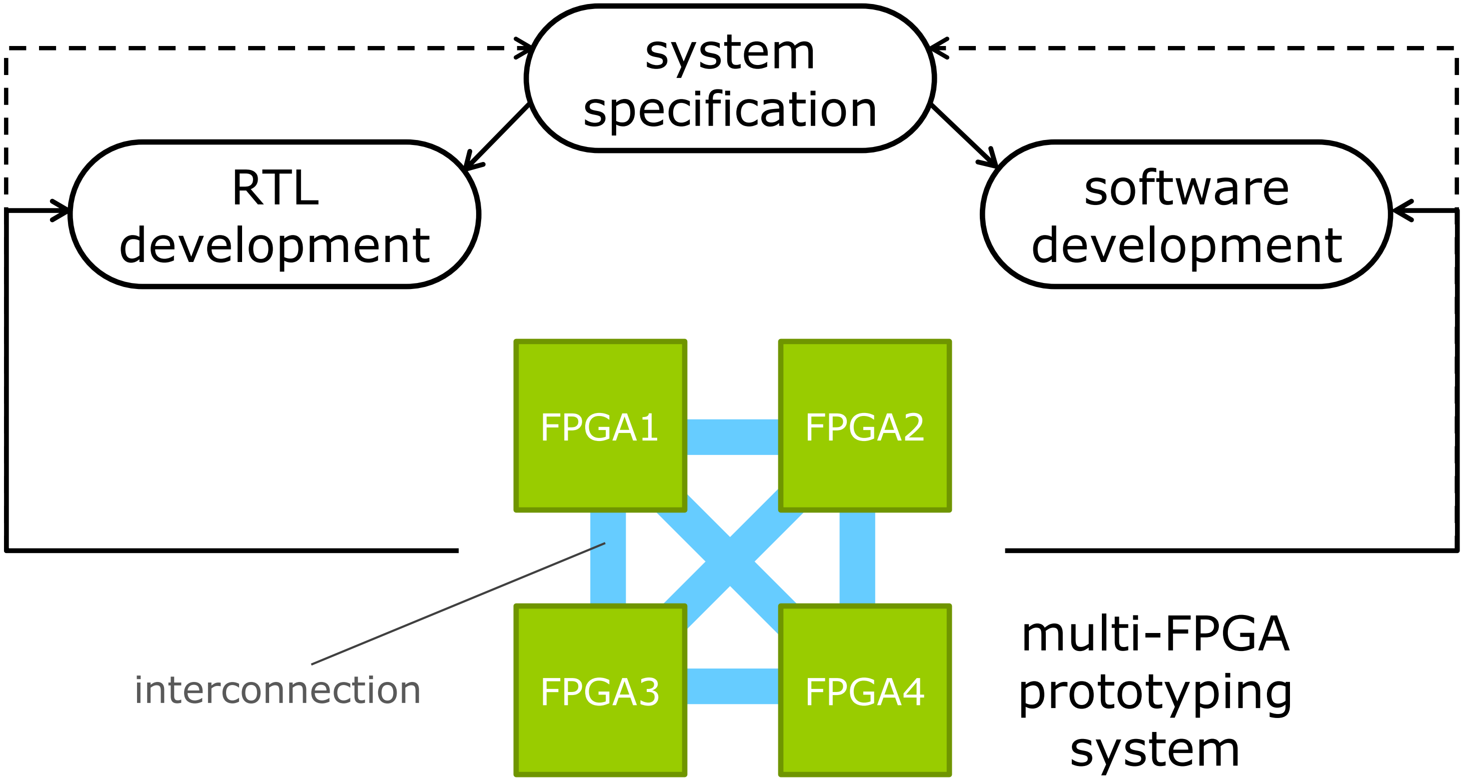
Extension for General Circuits

- Time-frame folding is a special case of circuit folding, as it only works for iterative circuits.

- Therefore, we extend the concept of “folding” for general combinational circuits to achieve **time multiplexing** in FPGAs.

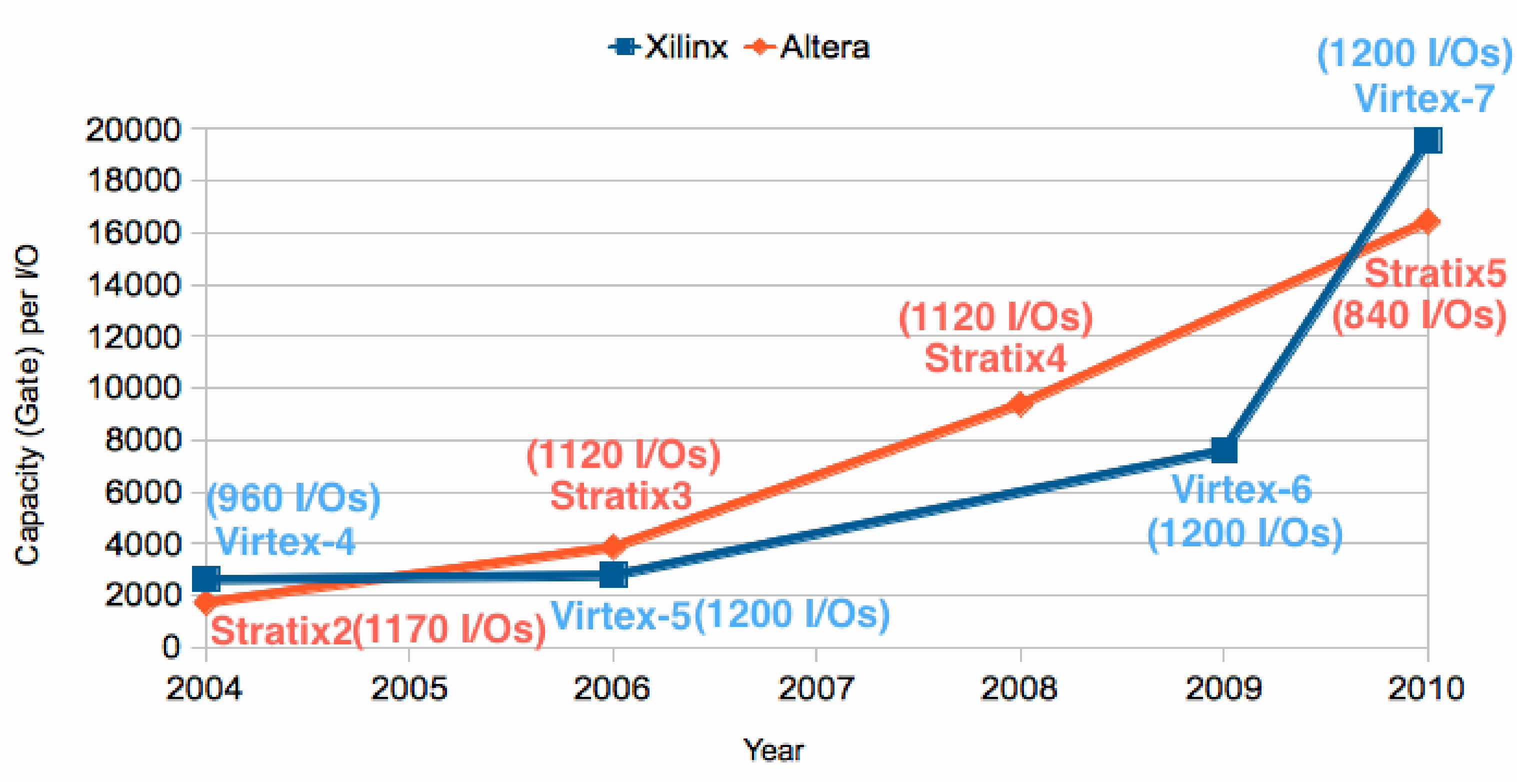
Multi-FPGA System

- Multi-FPGA boards are commonly used for system emulation [5] and prototyping.



[5] Myaing et al., 2011.

FPGA I/O bottleneck



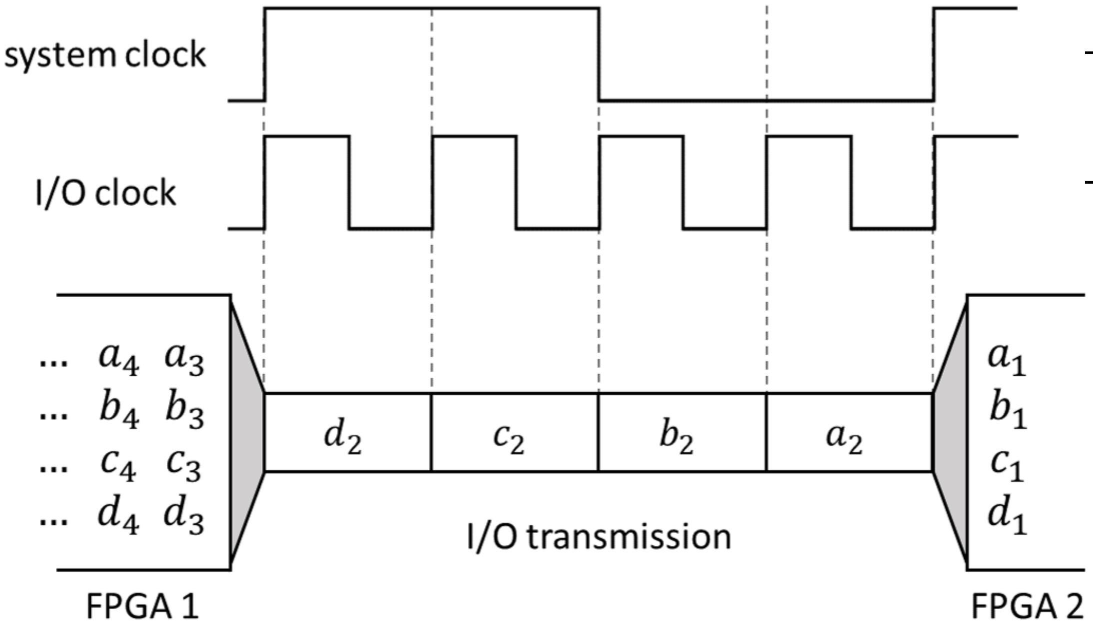
ratio of FPGA logic capacity over I/Os [6]

→ I/Os become scarce resources

[6] Hung et al., 2018.

Time Division Multiplexing (TDM)

- The system requires 2 separate clocks.



→ system operation

→ I/O data transmission

↓

4x #I/O can be transmitted in 1 system clock

↓

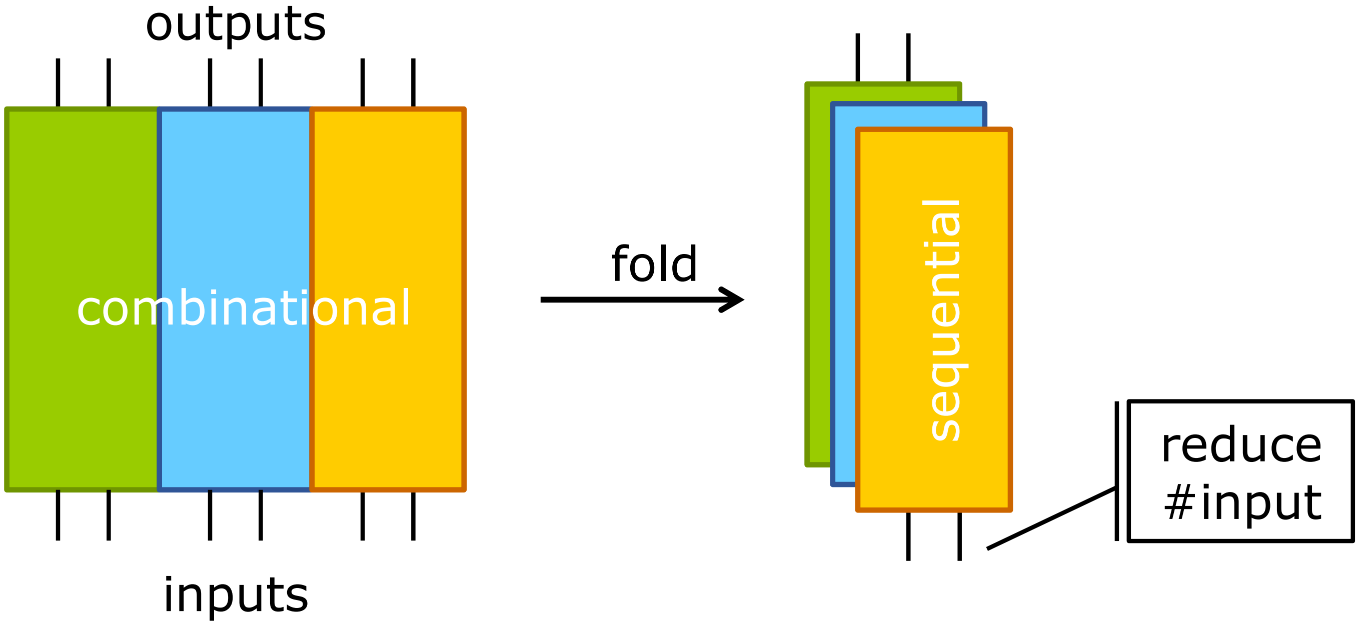
increase the effective I/O pins of FPGAs

TDM [7] I/O transmission with ratio 4

[7] Babb et al., 1993.

Time Multiplexing Motivation

- Instead of **increasing the effective I/O pins**, we try to **decrease the required input pin** by folding the circuit.



Time Multiplexing Formulation

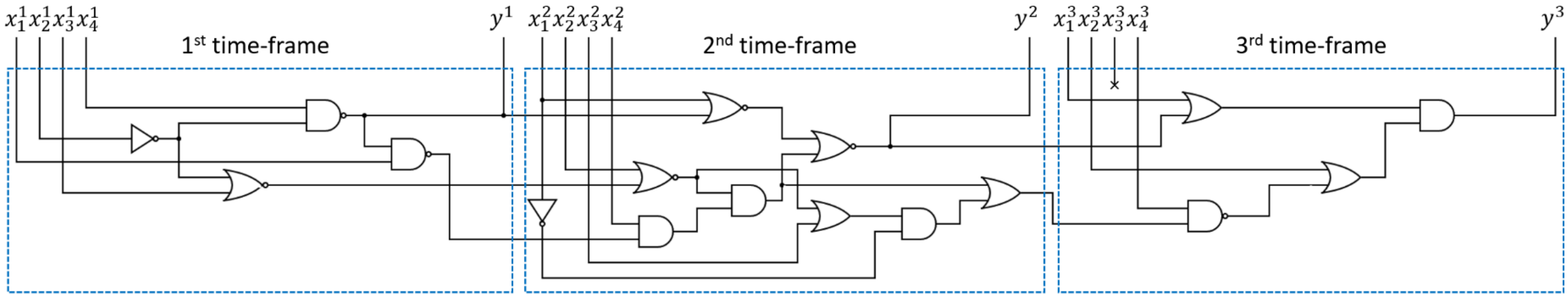
- Given a combinational circuit C_c with n inputs and m outputs, and a folding number T, we are asked to fold C_c into a sequential circuit C_S, which
  - has $\boldsymbol{n/T}$ **inputs** and less than m outputs
  - after expanding for T time-frames, becomes functionally equivalent to C_c under proper association of their inputs and outputs.

TIME-FRAME FOLDING

Recall:
TFF Formulation

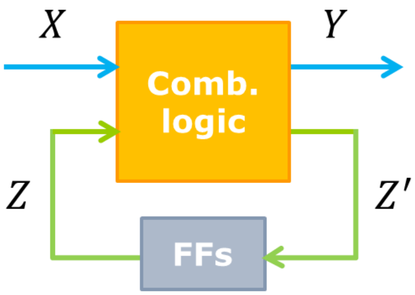
- TFF is a reverse operation of TFU

Given: a k-iterative combinational circuit

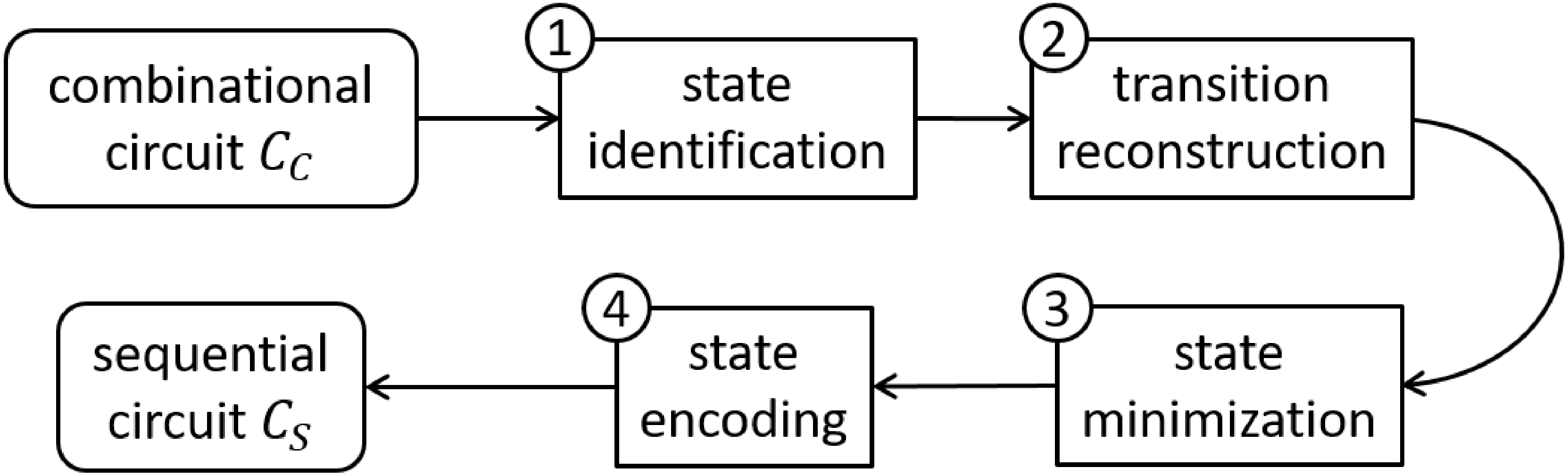


Goal: obtain a sequential circuit that
- is equivalent within bounded k time-frames
- has minimized finite state machine (FSM)

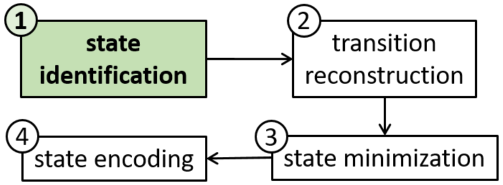
(no assumption is made on the circuit structure except for the iterative form)



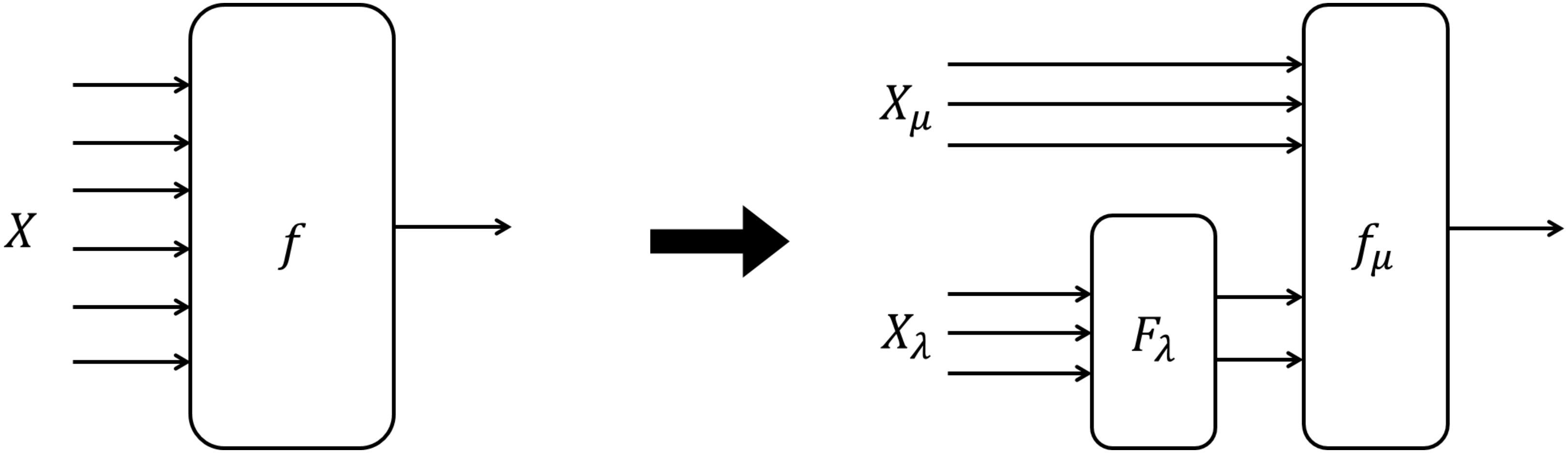
Computation Flow



State Identification



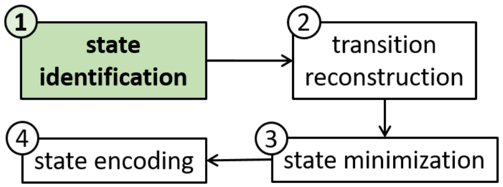
- Functional decomposition [8, 9]



X_λ: bound set, X_μ: free set

[8] Lai et al., 1993. [9] Chang et al., 1996.

State Identification

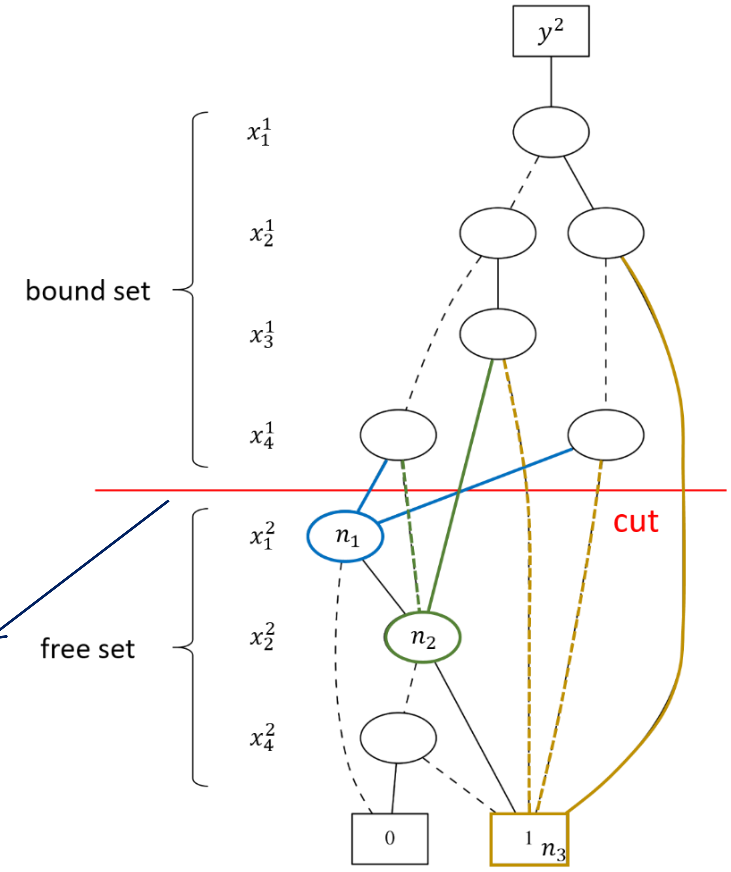


□ BDD-based decomposition

Decomposition chart

free $x_1^2x_2^2x_3^2x_4^2$ / bound $x_1^1x_2^1x_3^1x_4^1$	-0-1	011-	00-0	11--	10-0	010-
0000	0	1	1	1	1	1
0001	0	0	0	1	1	1
0010	0	1	1	1	1	1
0011	0	0	0	1	1	1
⋮	⋮	⋮			⋮	

column patterns

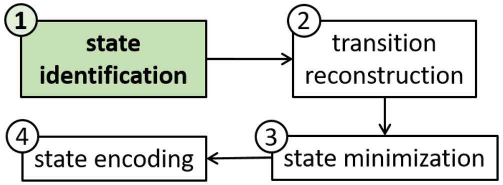


3 equivalence classes

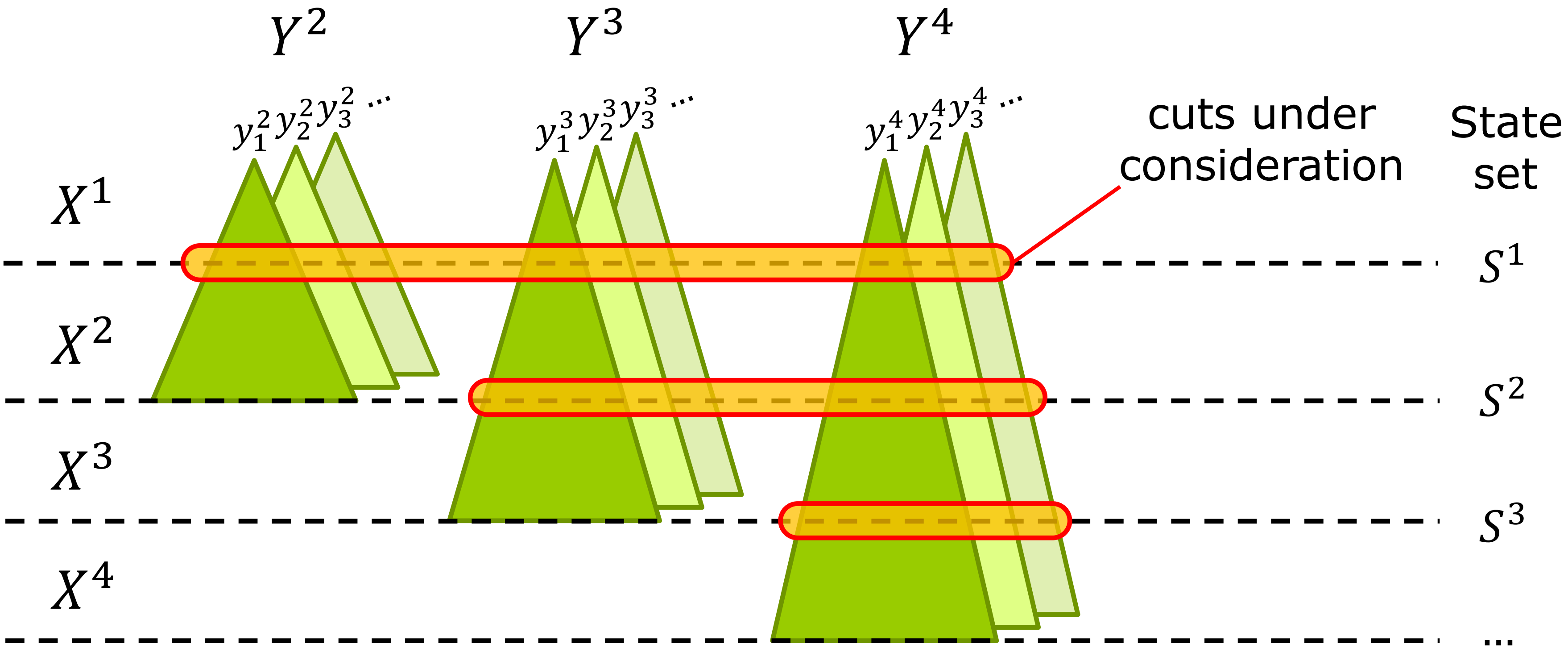
$$\left\{ \underline{\overline{x_2^1}x_4^1}, \underline{\overline{x_1^1}\left(x_2^1x_3^1 + \overline{x_2^1}\,\overline{x_4^1}\right)}, \underline{x_1^1\left(x_2^1 + \overline{x_4^1}\right) + \overline{x_1^1}x_2^1\overline{x_3^1}} \right\}$$

forming a partition on $\mathbb{B}^{|X^1|}$

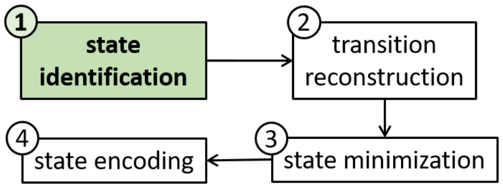
State Identification



- State set S^t reached at t^{th} time-frame is determined by $Y^{t+1}, Y^{t+2}, \dots, Y^T$


Y^2
Y^3
Y^4
$y_1^2\, y_2^2\, y_3^2 \cdots$
$y_1^3\, y_2^3\, y_3^3 \cdots$
$y_1^4\, y_2^4\, y_3^4 \cdots$
cuts under consideration
State set
X^1
X^2
X^3
X^4
S^1
S^2
S^3
…

State Identification



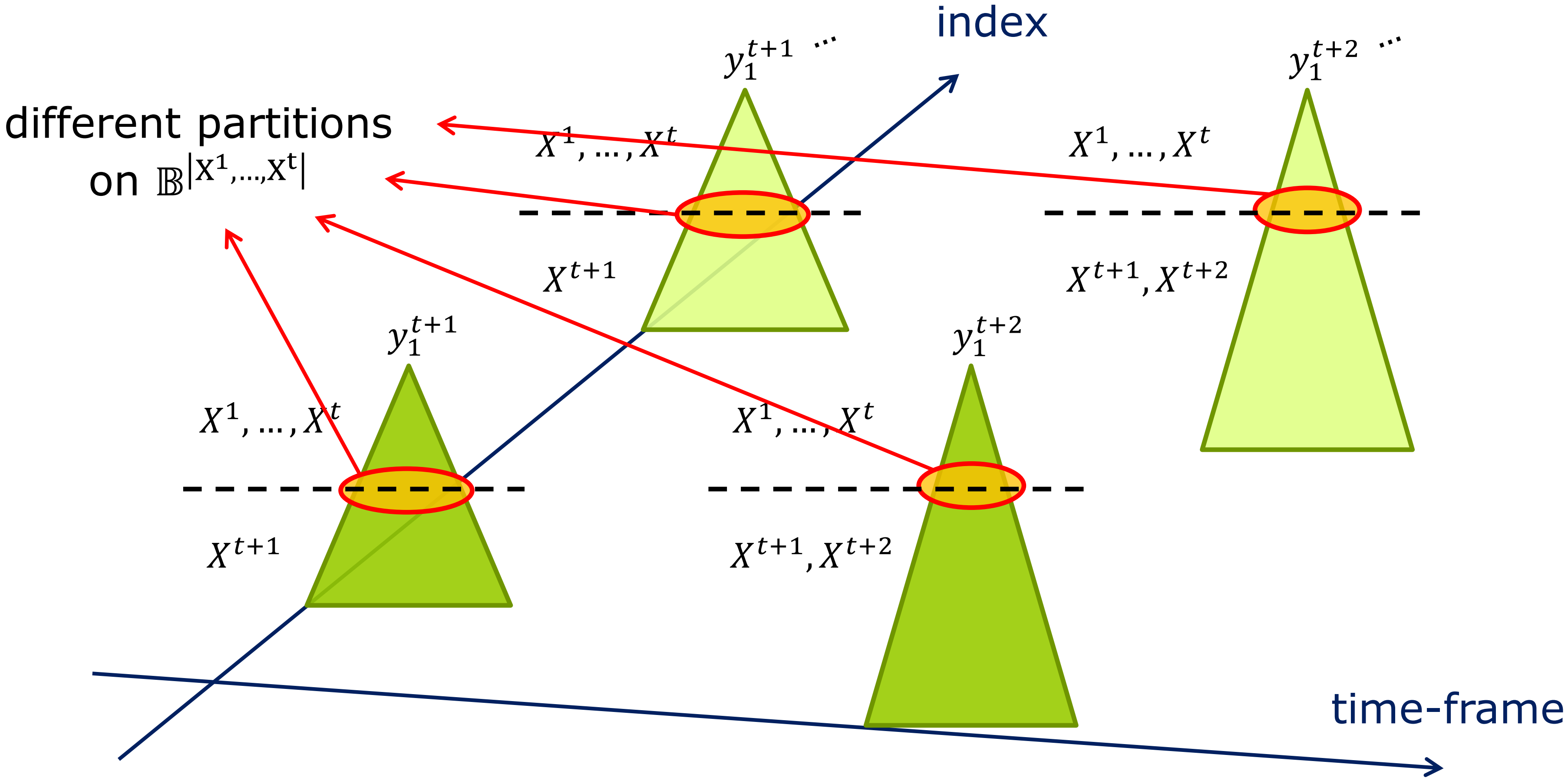
1 state identification

2 transition reconstruction

3 state minimization

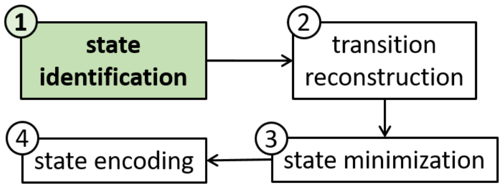
4 state encoding

- s^t derivation

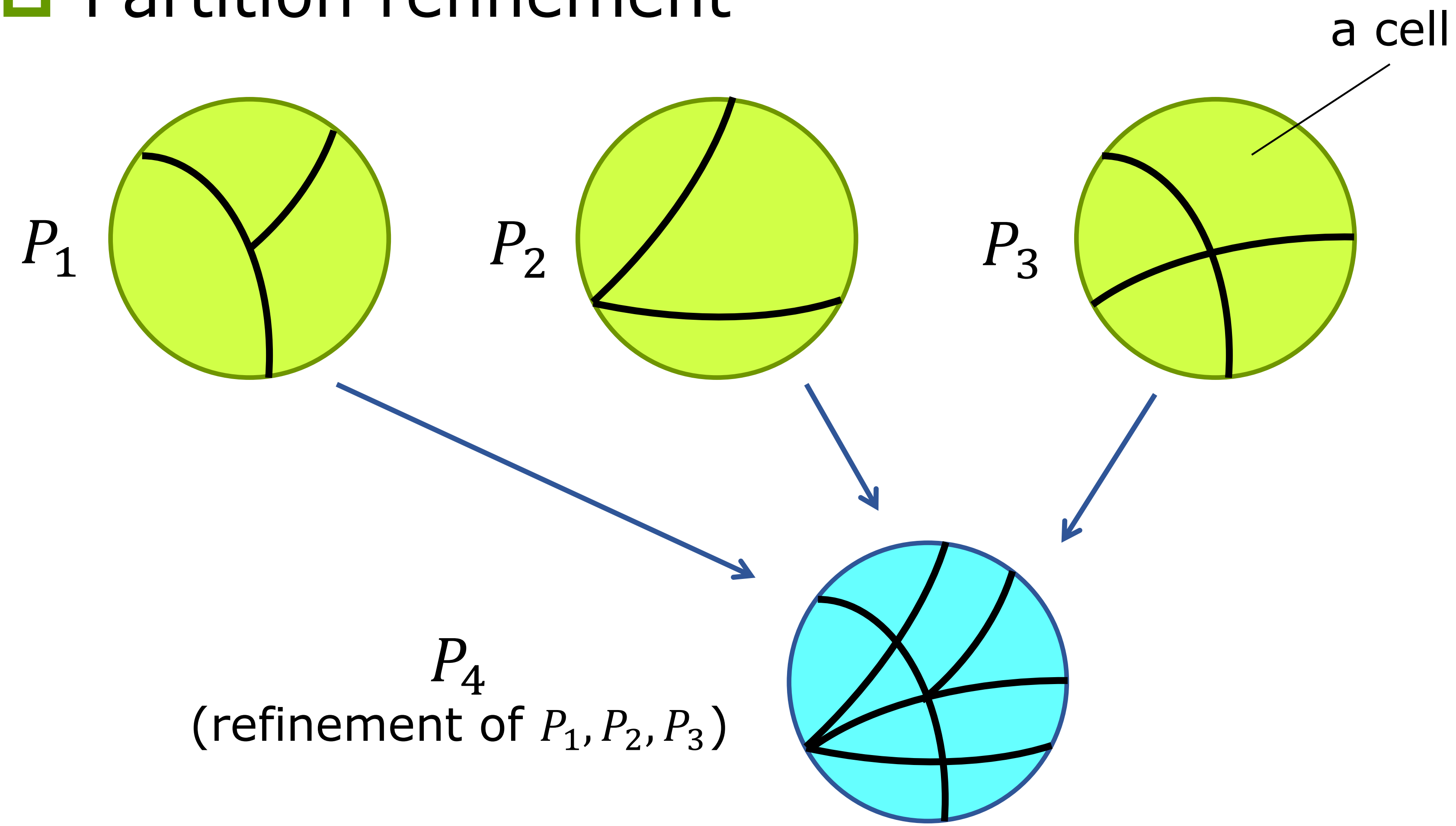


different partitions on $\mathbb{B}^{|x^1,\ldots,x^t|}$

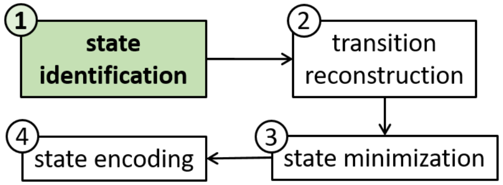
State Identification



- Partition refinement



State Identification



1 state identification

2 transition reconstruction

3 state minimization

4 state encoding

- Hyper-function encoding [10]:
  E.g. for a multi-output function

$$F(X) = \{f_1(X), f_2(X), f_3(X), f_4(X)\}$$

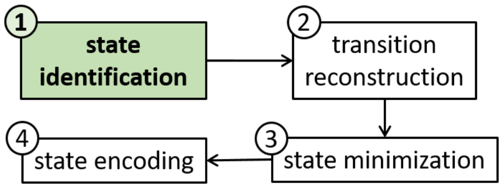
introduce $A = \{\alpha_1, \alpha_2\}$ to encode F into

$$h(X, A) = \overline{\alpha_1}\,\overline{\alpha_2} f_1 + \overline{\alpha_1}\alpha_2 f_2 + \alpha_1\overline{\alpha_2} f_3 + \alpha_1\overline{\alpha_2} f_4$$

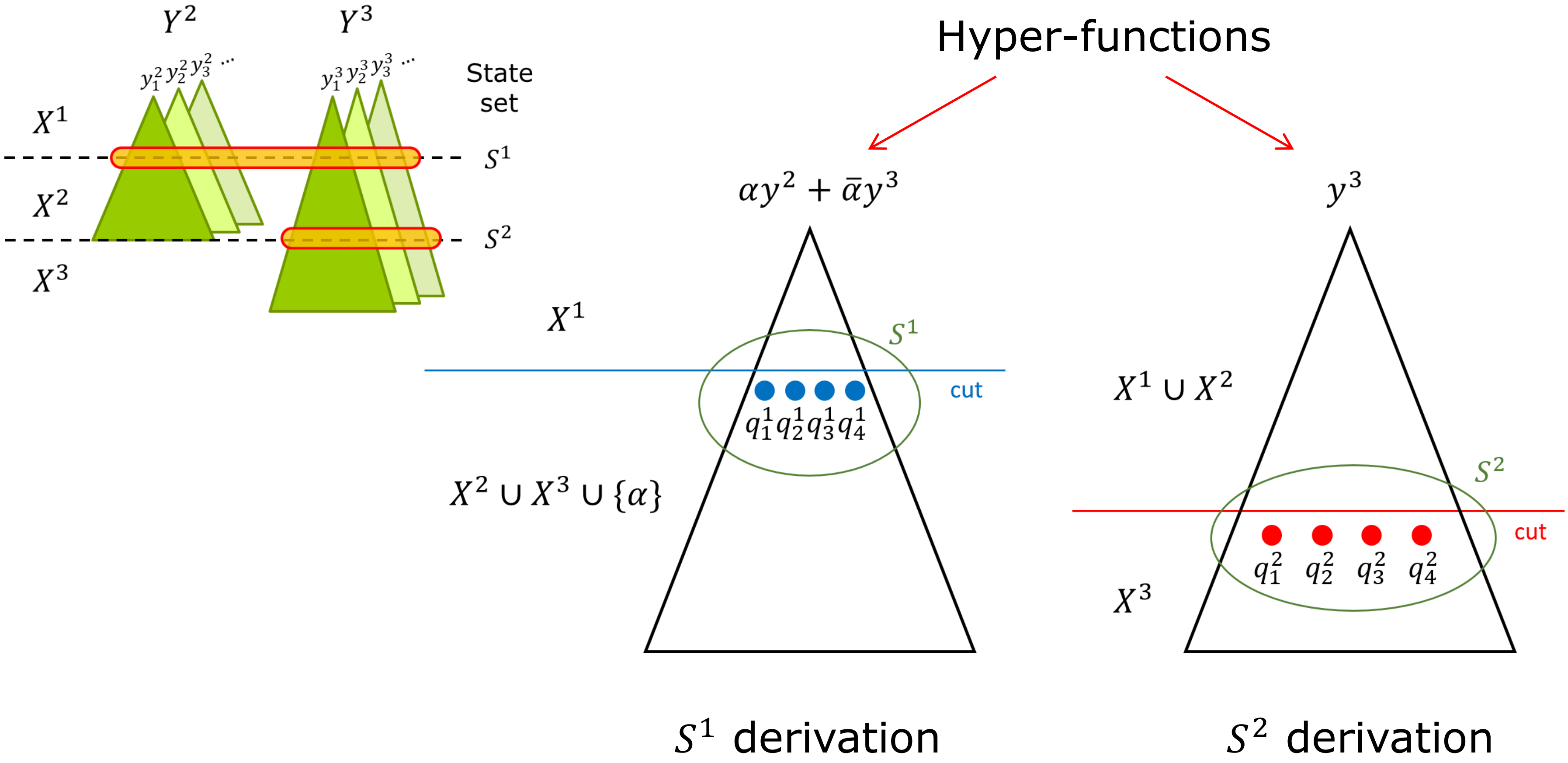
single-output functional decomposition algorithm can then be applied.

[10] Jiang et al., 1998.

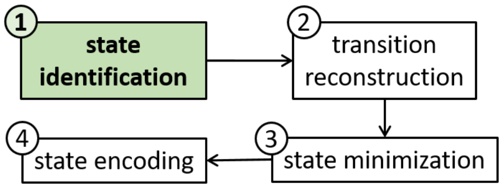
State Identification



- *s*27 example revisited


Y^2
Y^3
$y_1^2\, y_2^2\, y_3^2\, \cdots$
$y_1^3\, y_2^3\, y_3^3\, \cdots$
State set
X^1
X^2
X^3
S^1
S^2
Hyper-functions
$\alpha y^2 + \bar{\alpha} y^3$
y^3
X^1
S^1
cut
$q_1^1 q_2^1 q_3^1 q_4^1$
$X^2 \cup X^3 \cup \{\alpha\}$
$X^1 \cup X^2$
S^2
cut
$q_1^2\ q_2^2\ q_3^2\ q_4^2$
X^3
S^1 derivation
S^2 derivation

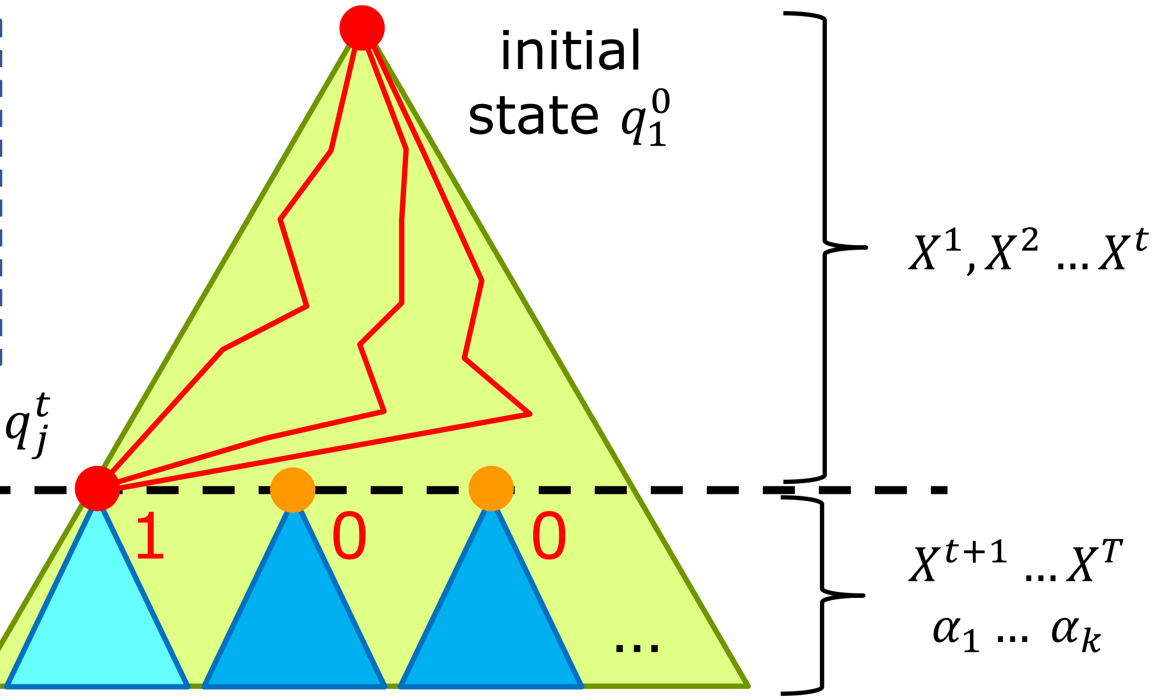
State Identification



- Transition condition $\tau_{q_j^t}$ of state q_j^t

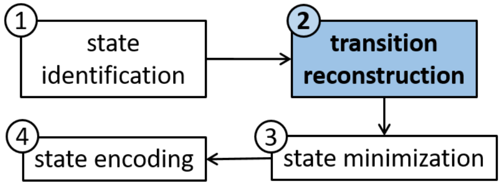
$hyper\text{-}function\ of\ \{Y^{t+1}, \ldots, Y^T\}$

$\tau_{q_j^t}(X^1, X^2 \ldots X^t)$:
collection of paths starting from q_1^0 leading to q_j^t

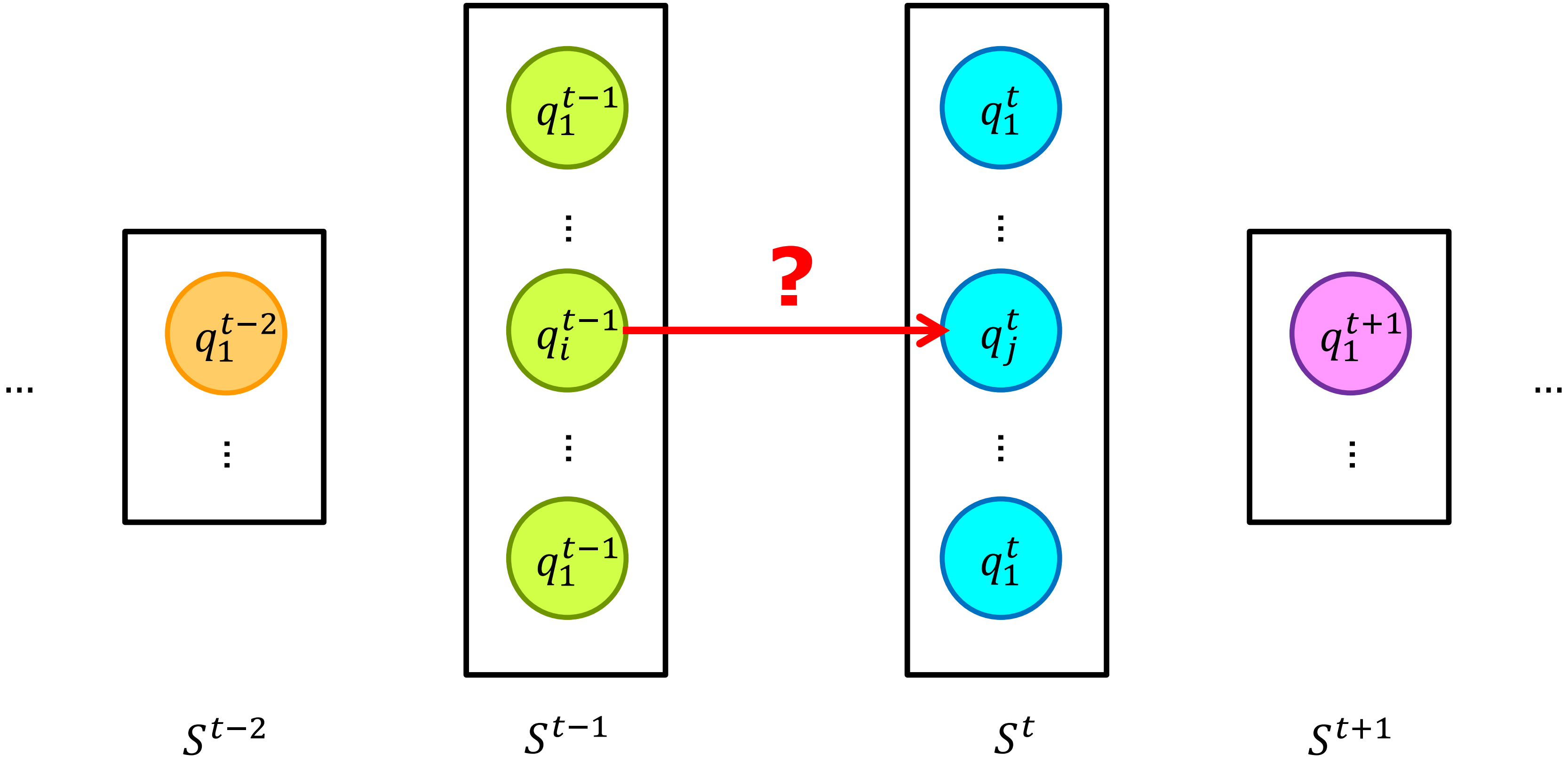


state set S^t

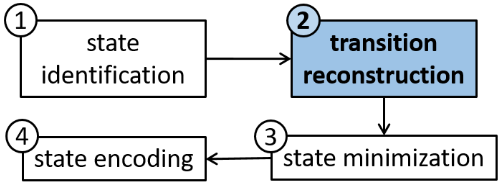
Transition Reconstruction



- Find the transition between state pairs



Transition Reconstruction



1 state identification

2 transition reconstruction

3 state minimization

4 state encoding

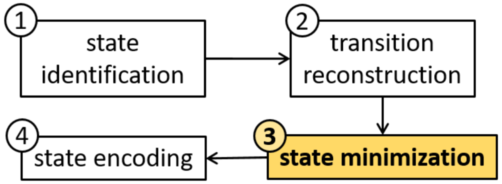
- For each pair of state (q_i^{t-1}, q_j^t) in adjacent 2 time-frames:
  - Input transition condition:

$$\varphi^t_{i,j} = \underbrace{\exists X^1, \dots, X^{t-1}}_{\text{global} \rightarrow \text{local info.}} . \underbrace{\tau_{q_i^{t-1}} \wedge \tau_{q_j^t}}_{\text{paths to } q_j^t \text{ through } q_i^{t-1}}$$

  - Output transition response

$$\psi^t_{i,k} = \exists X^1, \dots, X^{t-1} . \tau_{q_i^t} \wedge y_k^t$$

State Minimization



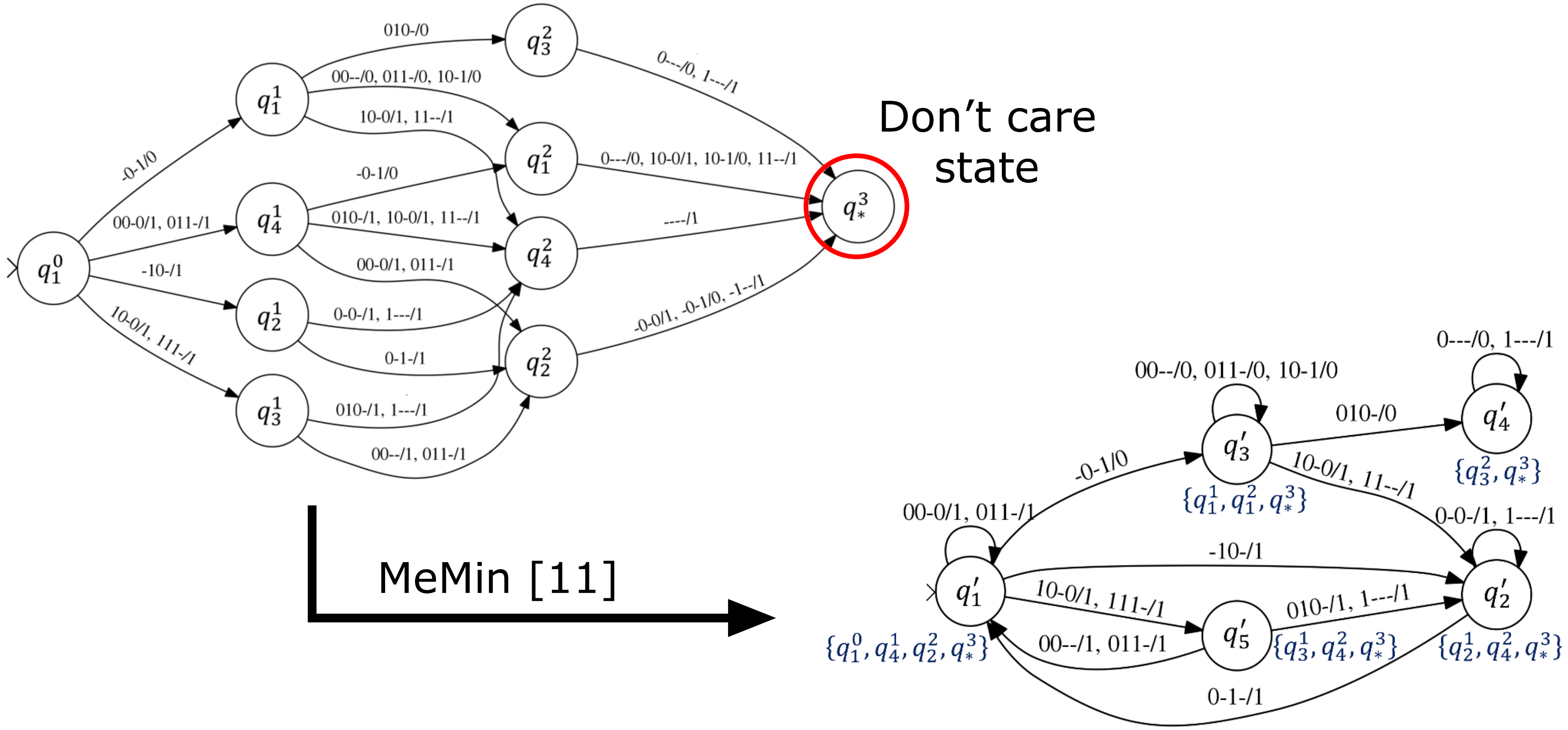
① state identification

② transition reconstruction

③ **state minimization**

④ state encoding

- *s*27 example revisited

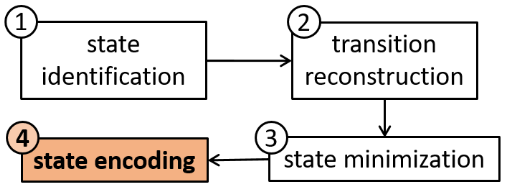


Don’t care state

MeMin [11]

[11] Abel et al., 2015.

State Encoding



1 state identification

2 transition reconstruction

3 state minimization

4 state encoding

- Encode each state in the state set Q with actual bits, 2 schemes are applied:
    - Natural Encoding with $\lceil \log(|Q|) \rceil$ bits
    - One-hot encoding with $|Q|$ bits, each of which represents a state in Q.

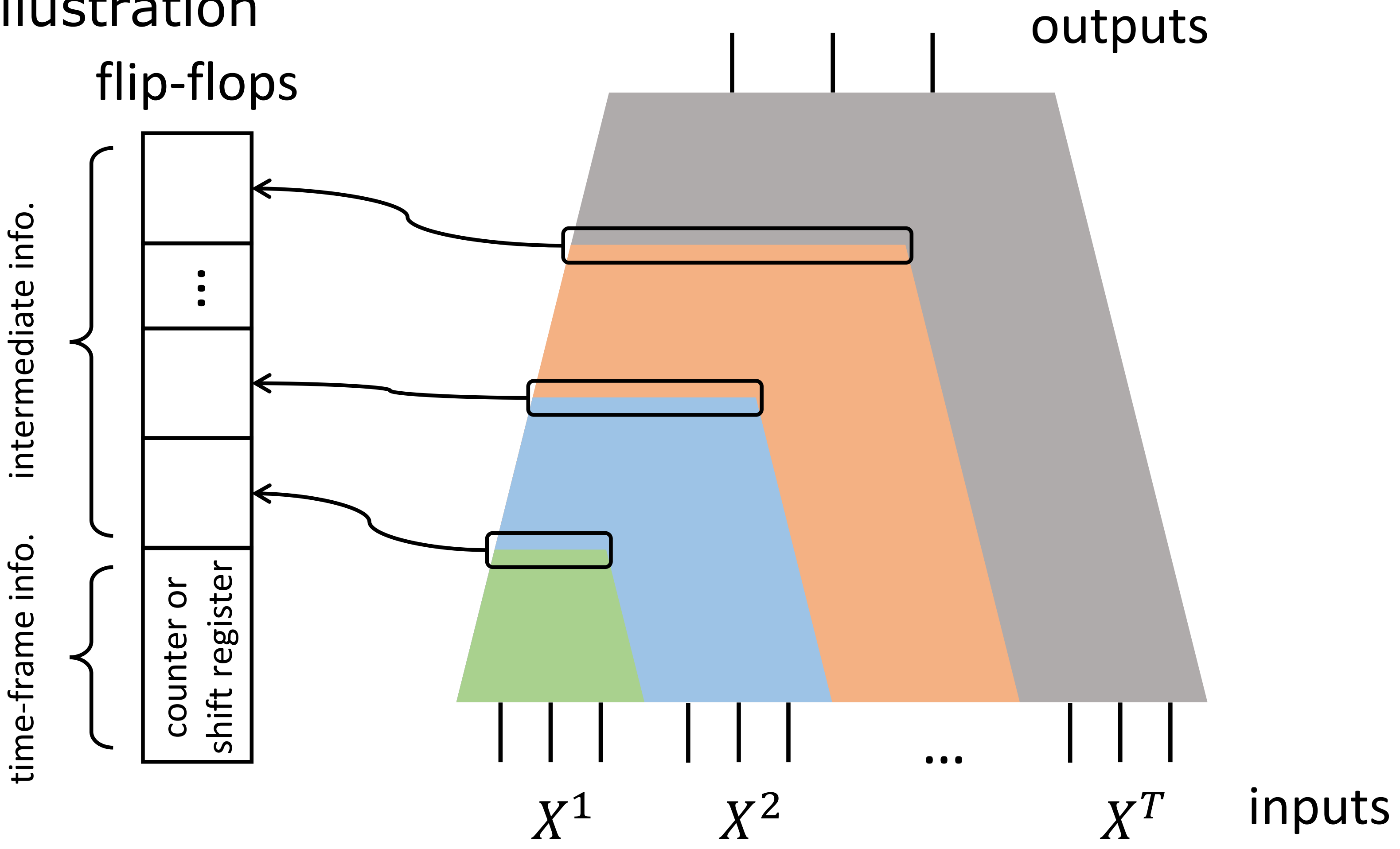
CIRCUIT FOLDING FOR TIME MULTIPLEXING

Recall:
Time Multiplexing Formulation

- Given a combinational circuit C_c with n inputs and m outputs, and a folding number T, we are asked to fold C_c into a sequential circuit C_S, which
  - has $\boldsymbol{n/T}$ **inputs** and less than m outputs
  - after expanding for T time-frames, becomes functionally equivalent to C_c under proper association of their inputs and outputs.

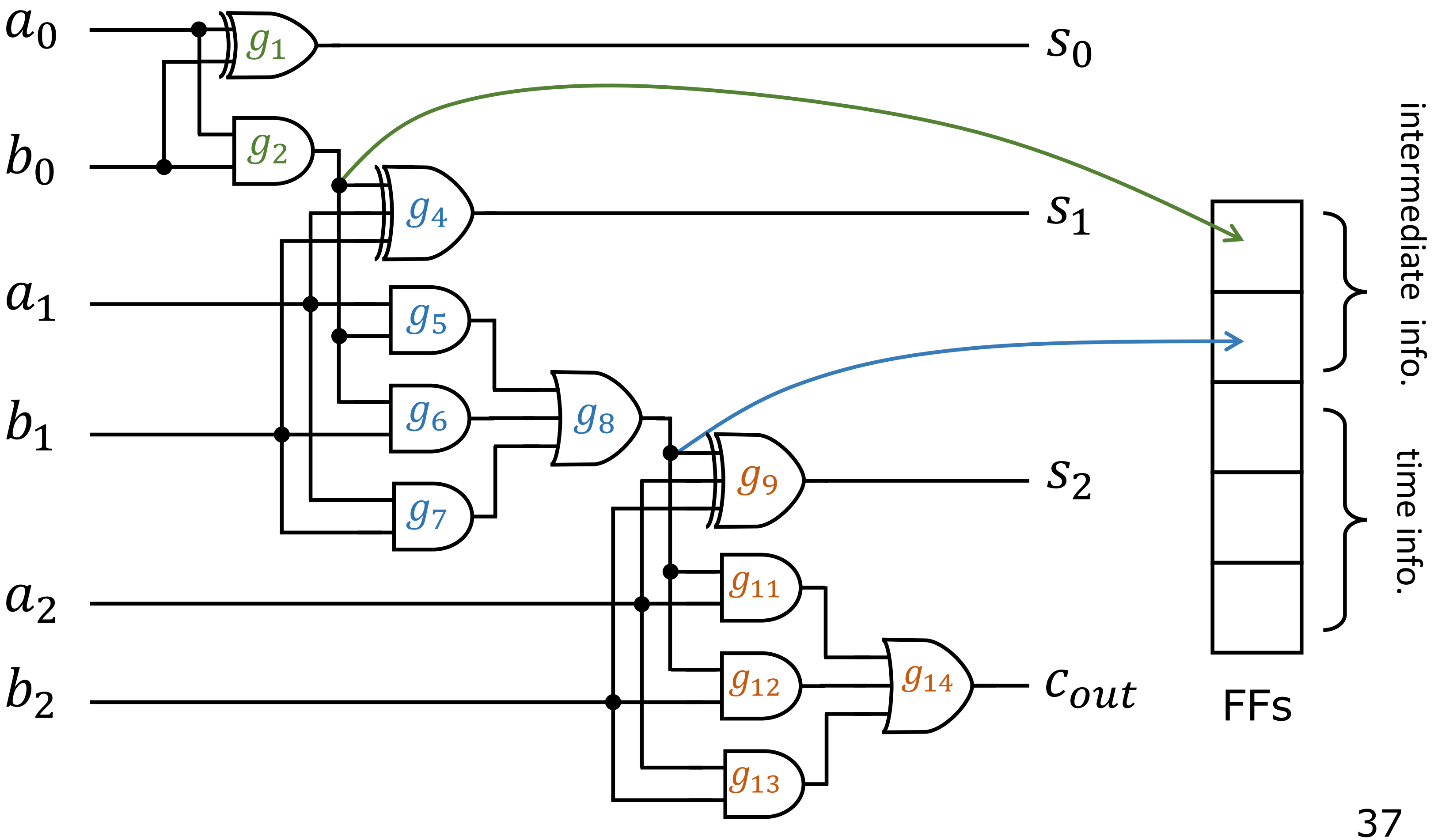
Structural Method

- Illustration



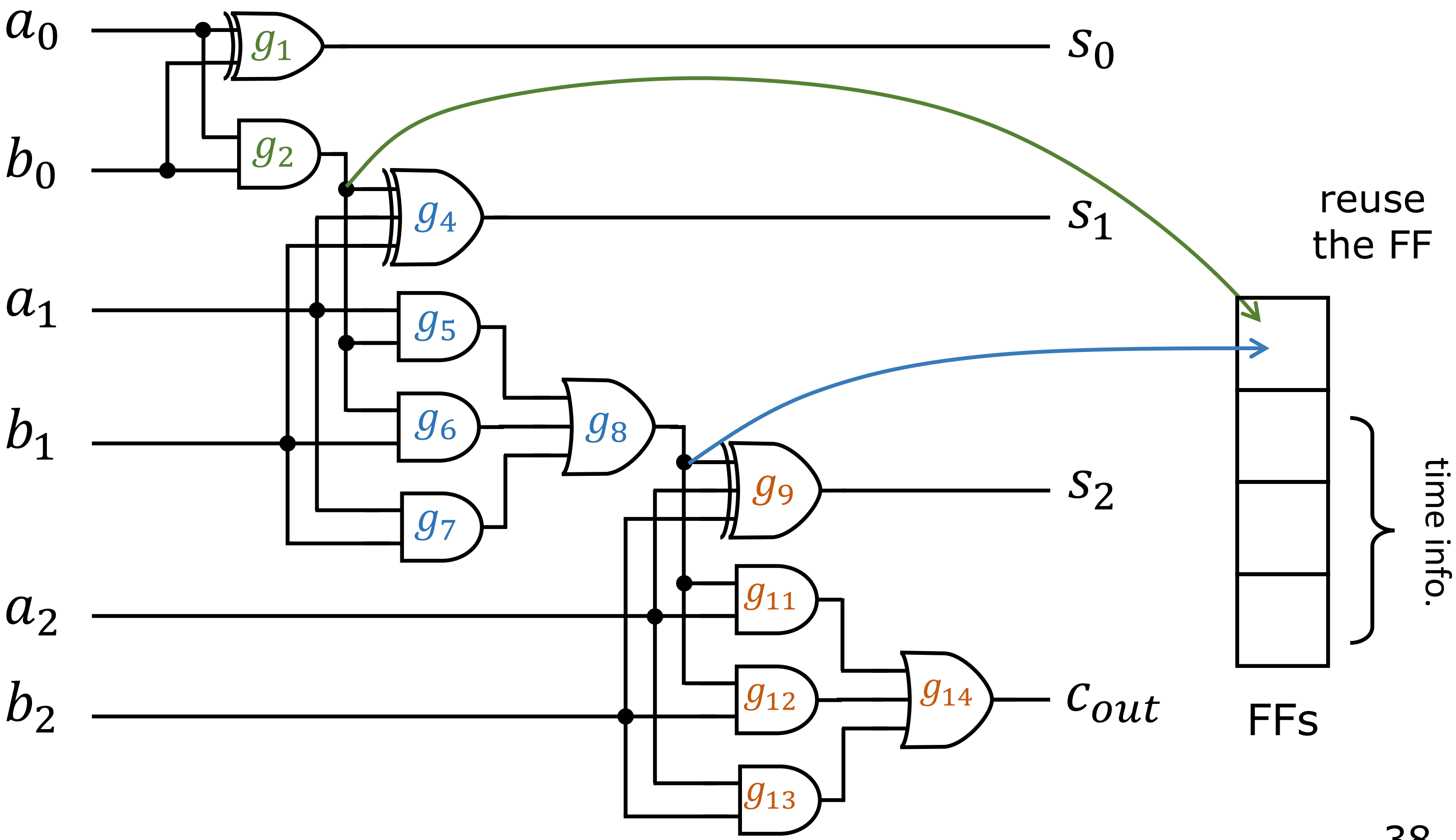
Structural Method

- 3-adder example ($T = 3$)



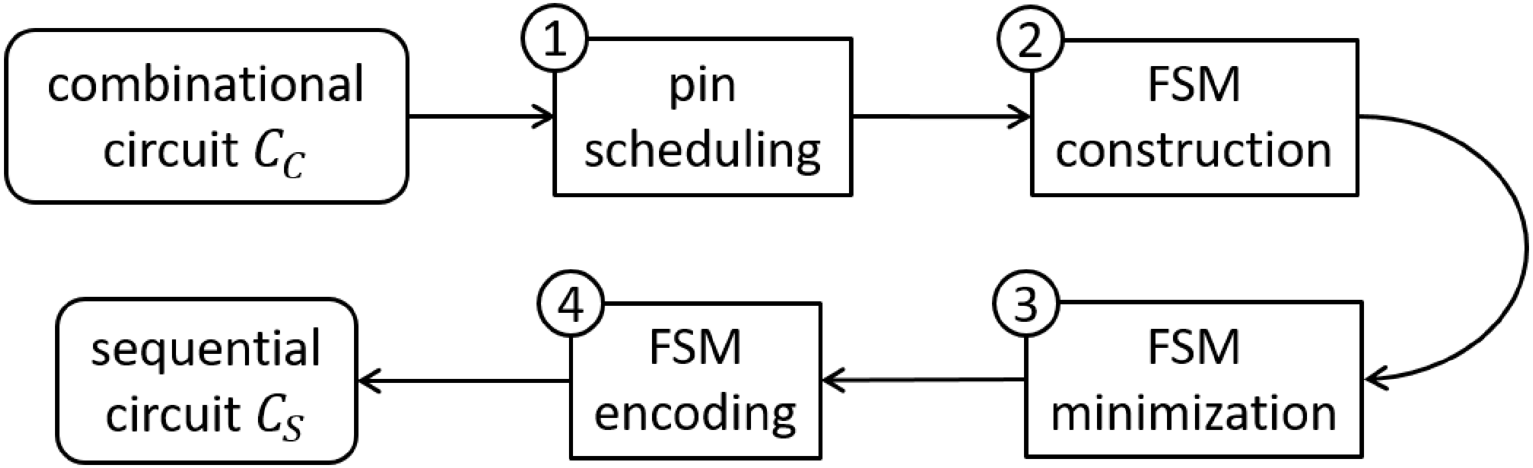
Structural Method

- A little improvement



Functional Method

- Computation flow



Functional Method

- Pin scheduling heuristic:

  Convert the given combinational circuit into pseudo-iterative form.

  - output pin scheduling
  - input pin scheduling

Functional Method

- Output pin scheduling
  1. sort the outputs according to their support sizes in an ascending order.

output	s_0	s_1	s_2	c_{out}
support	a_0, b_0	a_0, b_0, a_1, b_1	$a_0, b_0, a_1, b_1, a_2, b_2$	$a_0, b_0, a_1, b_1, a_2, b_2$
|support|	2	4	6	6

ascending order

Functional Method

- Output pin scheduling
  2. determine the iteration of each output to be scheduled at.

#input of the folded circuit = 2

output	s_0	s_1	s_2	c_{out}
support	a_0, b_0	a_0, b_0, a_1, b_1	$a_0, b_0, a_1, b_1, a_2, b_2$	$a_0, b_0, a_1, b_1, a_2, b_2$
|support|	$2 / 2$	$4 / 2$	$6 / 2$	$6 / 2$
iteration	1	2	3	3

Functional Method

- Output pin scheduling
  3. null outputs insertion.

iteration	1	2	3
scheduled outputs	$s_0, null$	$s_1, null$	s_2, c_{out}

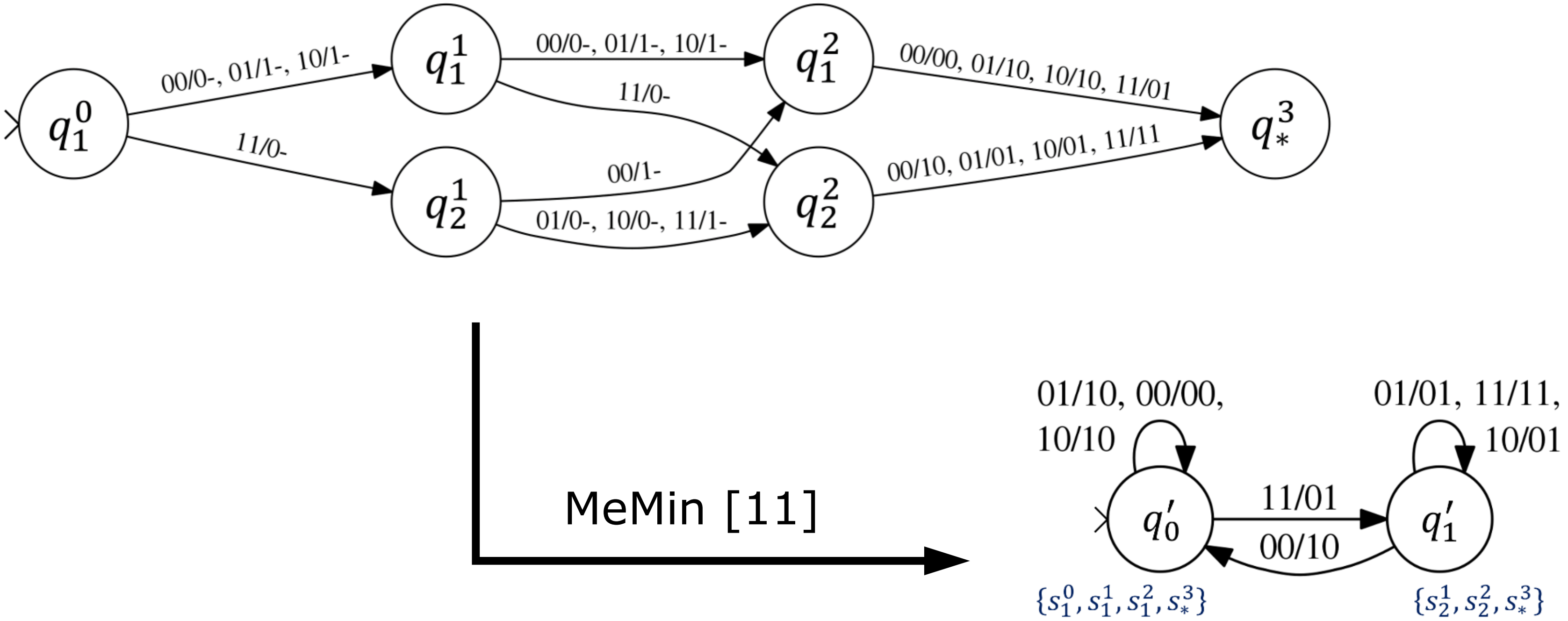
Functional Method

- Input pin scheduling
  - schedule the inputs according to the outputs.

iteration	1	2	3
scheduled outputs	$s_0, null$	$s_1, null$	s_2, c_{out}
scheduled inputs	a_0, b_0	a_1, b_1	a_2, b_2

Functional Method

- FSM construction & minimization



Carry-save adder

[11] Abel et al., 2015.

EXPERIMENTS

Setup

- Implemented in C++ within ABC [12] and used CUDD [13] as the underlying BDD package.
- Environment: Intel(R) Core(TM) i7-8700 3.20GHz CPU and 32GB RAM
- Benchmark circuits: ISCAS, ITC, MCNC(LGSynth), LEKO/LEKU, Adder, and EPFL

[12] Brayton et al., 2010. [13] Somenzi et al., 2005.

TFF - Setup

- Benchmark circuits
  - Unfolded ISCAS/ITC circuits
  - QBF solving of homing sequence [4]
- 300s timeout limit on FSM construction and minimization, individually

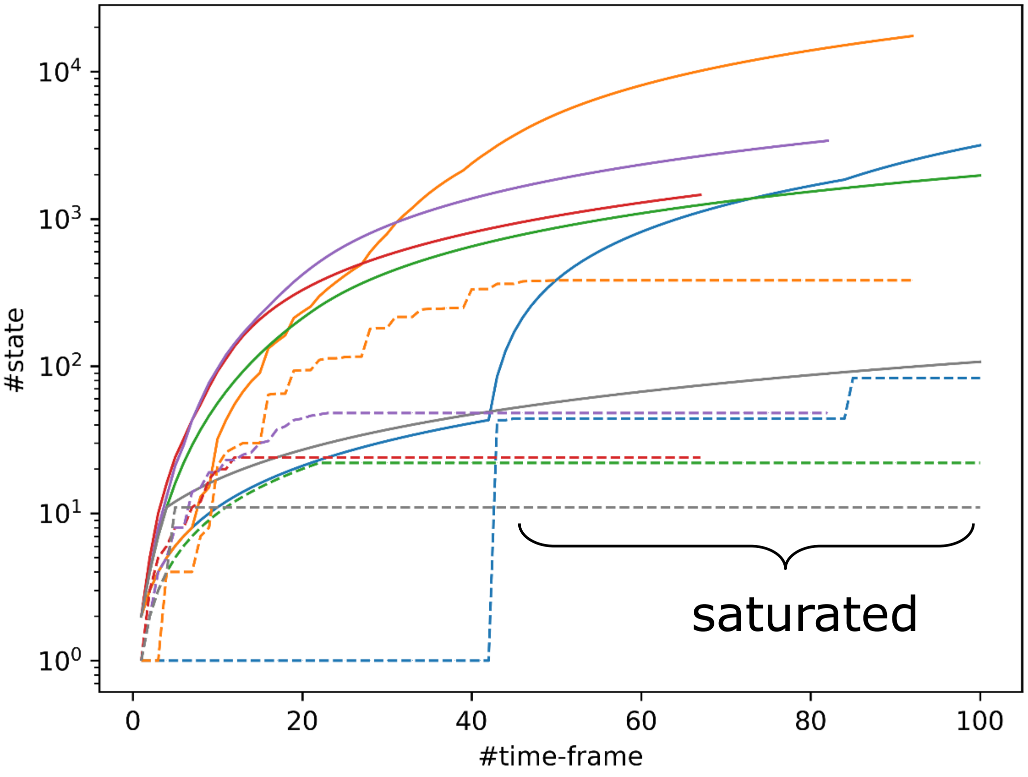
[4] Wang et al., 2017.

TFF - Results

□ Number of states

🟦 b07
🟧 b18
🟩 s499
🟥 s832
🟪 s1494
⬛ s15850

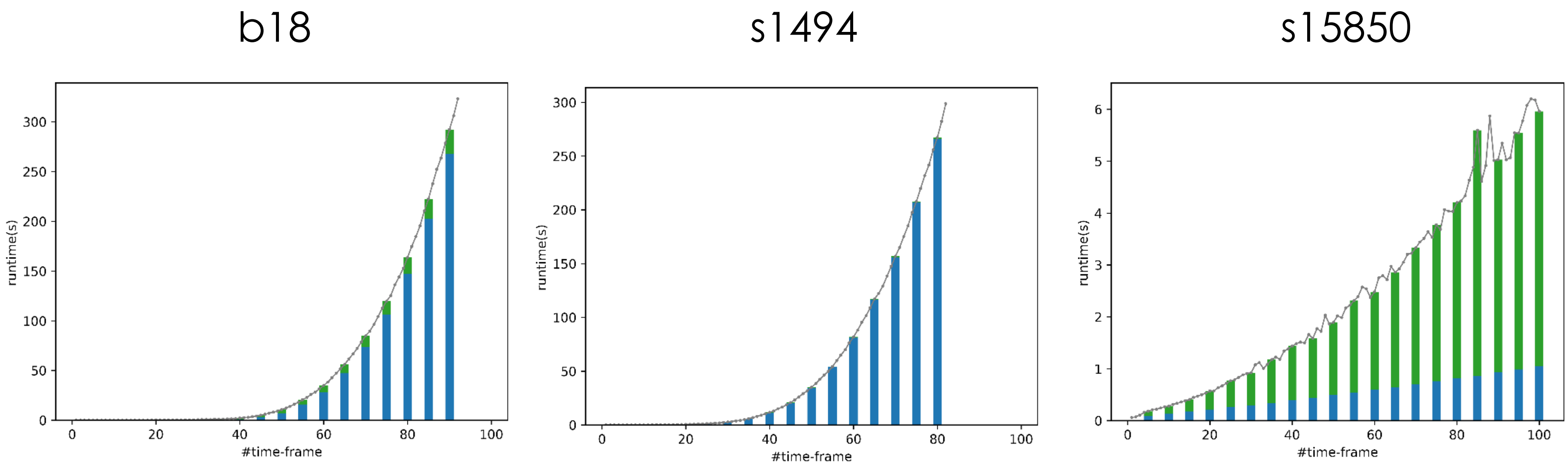
—— : #state **before** minimization
---- : #state **after** minimization



#state vs. #time-frame.

TFF - Results

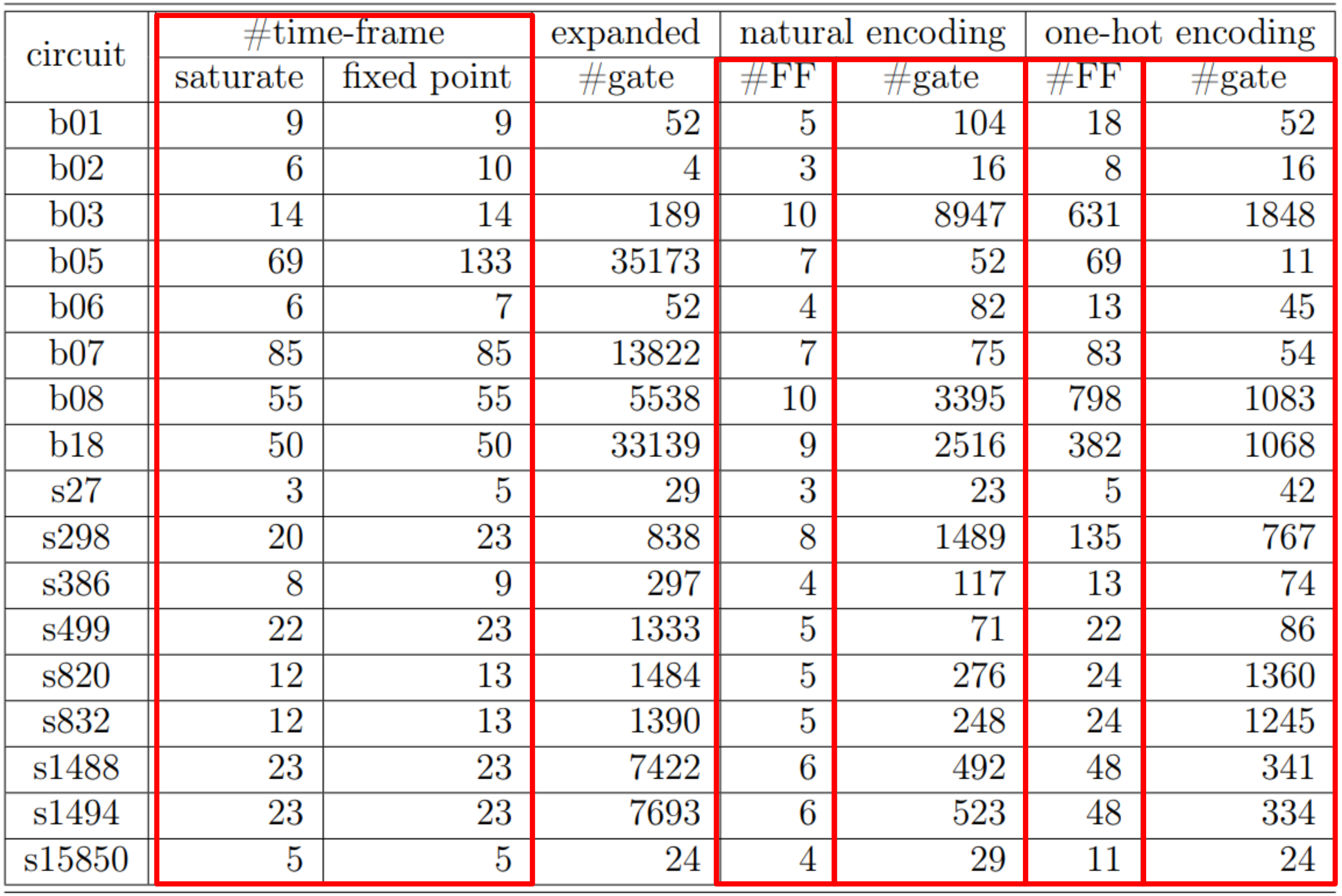
- Total runtime



runtime vs. #time-frame.

🟦 Time-Fold
🟩 MeMin

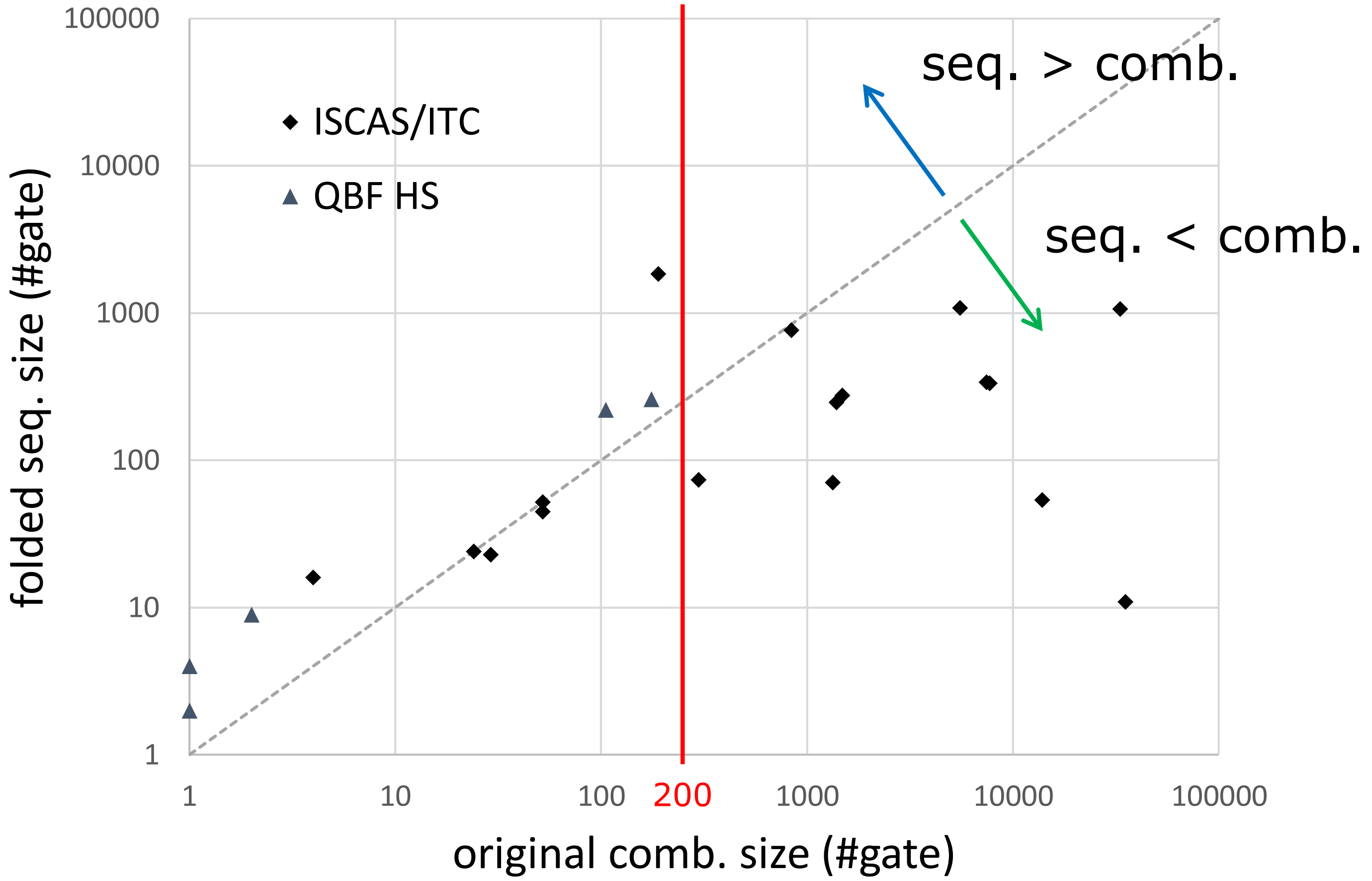
TFF - Results



circuit	#time-frame		expanded	natural encoding		one-hot encoding	
	saturate	fixed point	#gate	#FF	#gate	#FF	#gate
b01	9	9	52	5	104	18	52
b02	6	10	4	3	16	8	16
b03	14	14	189	10	8947	631	1848
b05	69	133	35173	7	52	69	11
b06	6	7	52	4	82	13	45
b07	85	85	13822	7	75	83	54
b08	55	55	5538	10	3395	798	1083
b18	50	50	33139	9	2516	382	1068
s27	3	5	29	3	23	5	42
s298	20	23	838	8	1489	135	767
s386	8	9	297	4	117	13	74
s499	22	23	1333	5	71	22	86
s820	12	13	1484	5	276	24	1360
s832	12	13	1390	5	248	24	1245
s1488	23	23	7422	6	492	48	341
s1494	23	23	7693	6	523	48	334
s15850	5	5	24	4	29	11	24

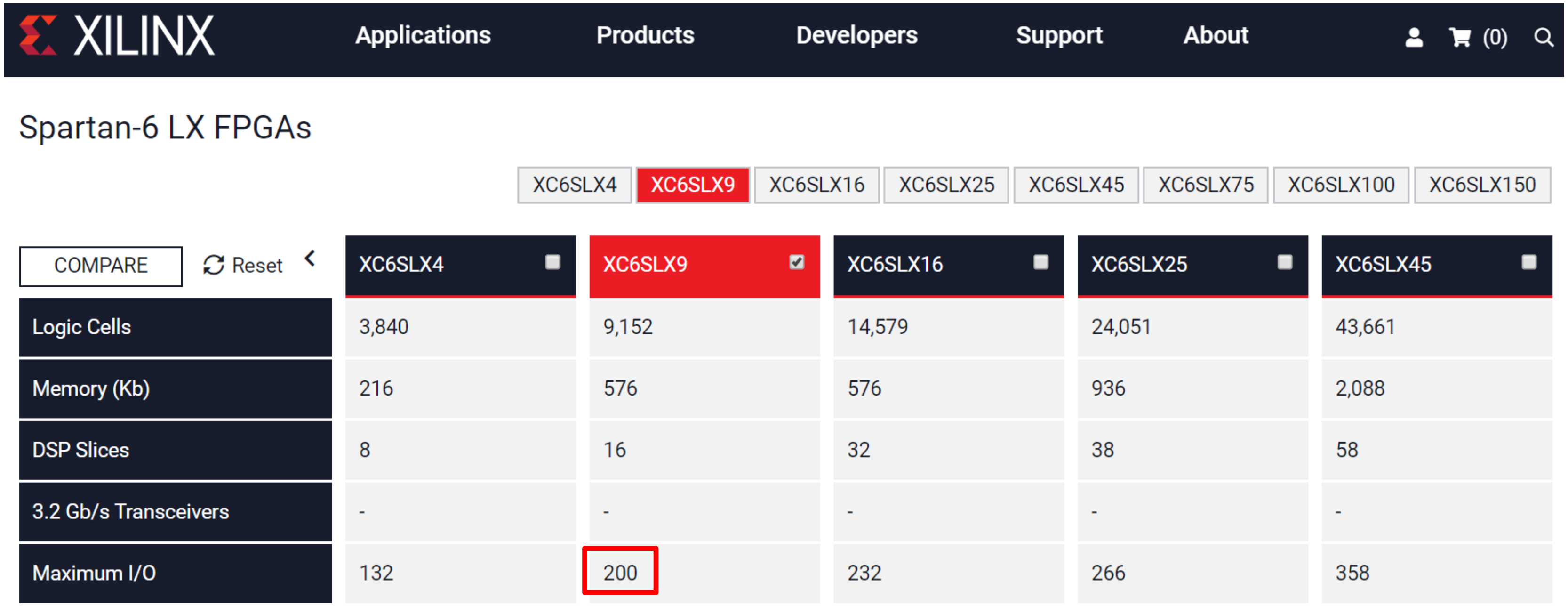
Results on folding with "fixed points" reached.

TFF - Results



Circuit size comparison.

Structural Method - Setup



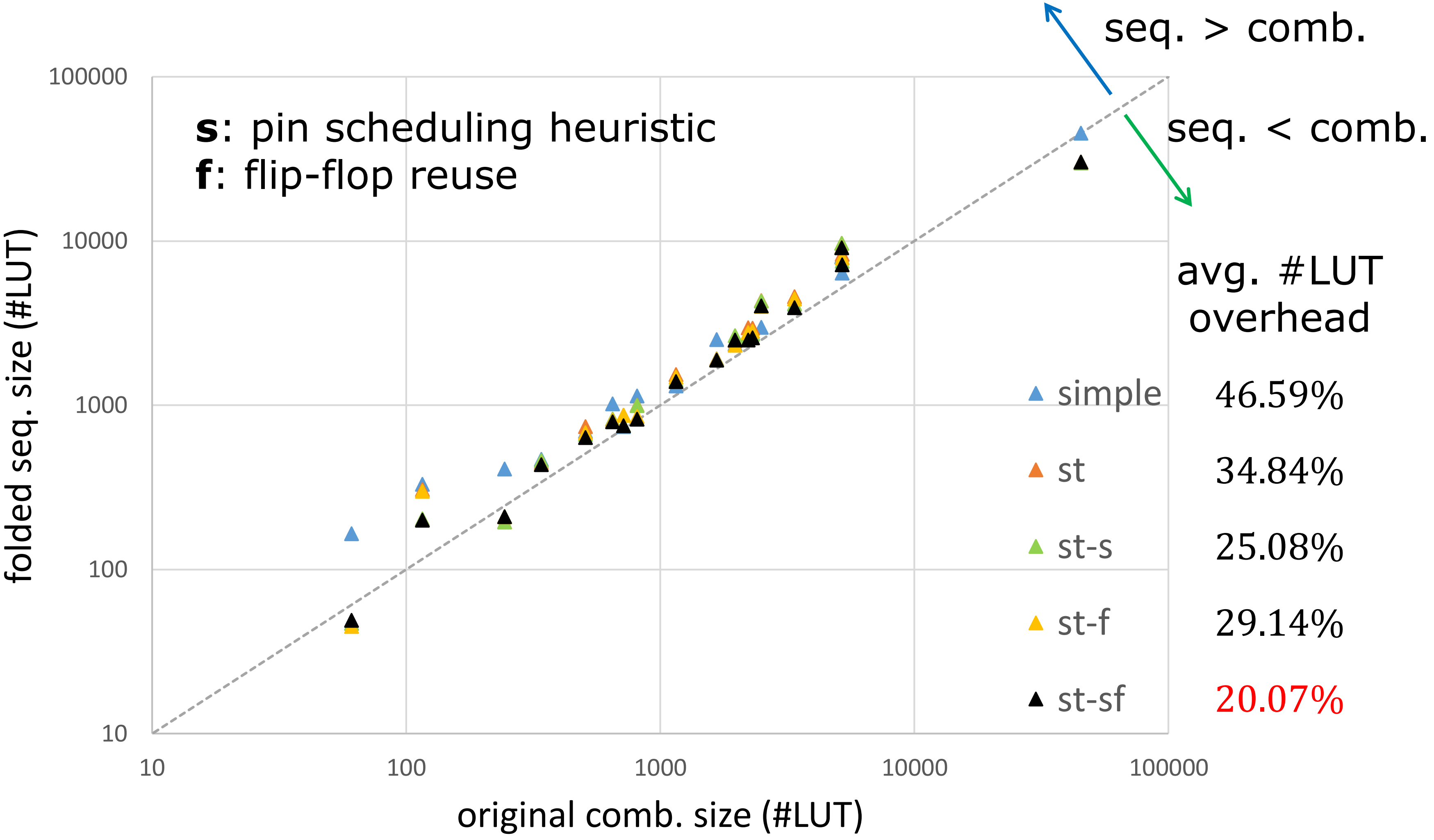
Spartan-6 LX FPGAs

	XC6SLX4 □	XC6SLX9 ☑	XC6SLX16 □	XC6SLX25 □	XC6SLX45 □
Logic Cells	3,840	9,152	14,579	24,051	43,661
Memory (Kb)	216	576	576	936	2,088
DSP Slices	8	16	32	38	58
3.2 Gb/s Transceivers	-	-	-	-	-
Maximum I/O	132	200	232	266	358

impose the input pin count limitation to 200 [14]

[14] https://www.xilinx.com/products/silicon-devices/fpga/spartan-6.html#productTable

Structural Method - Results



seq. > comb.

seq. < comb.

s: pin scheduling heuristic
f: flip-flop reuse

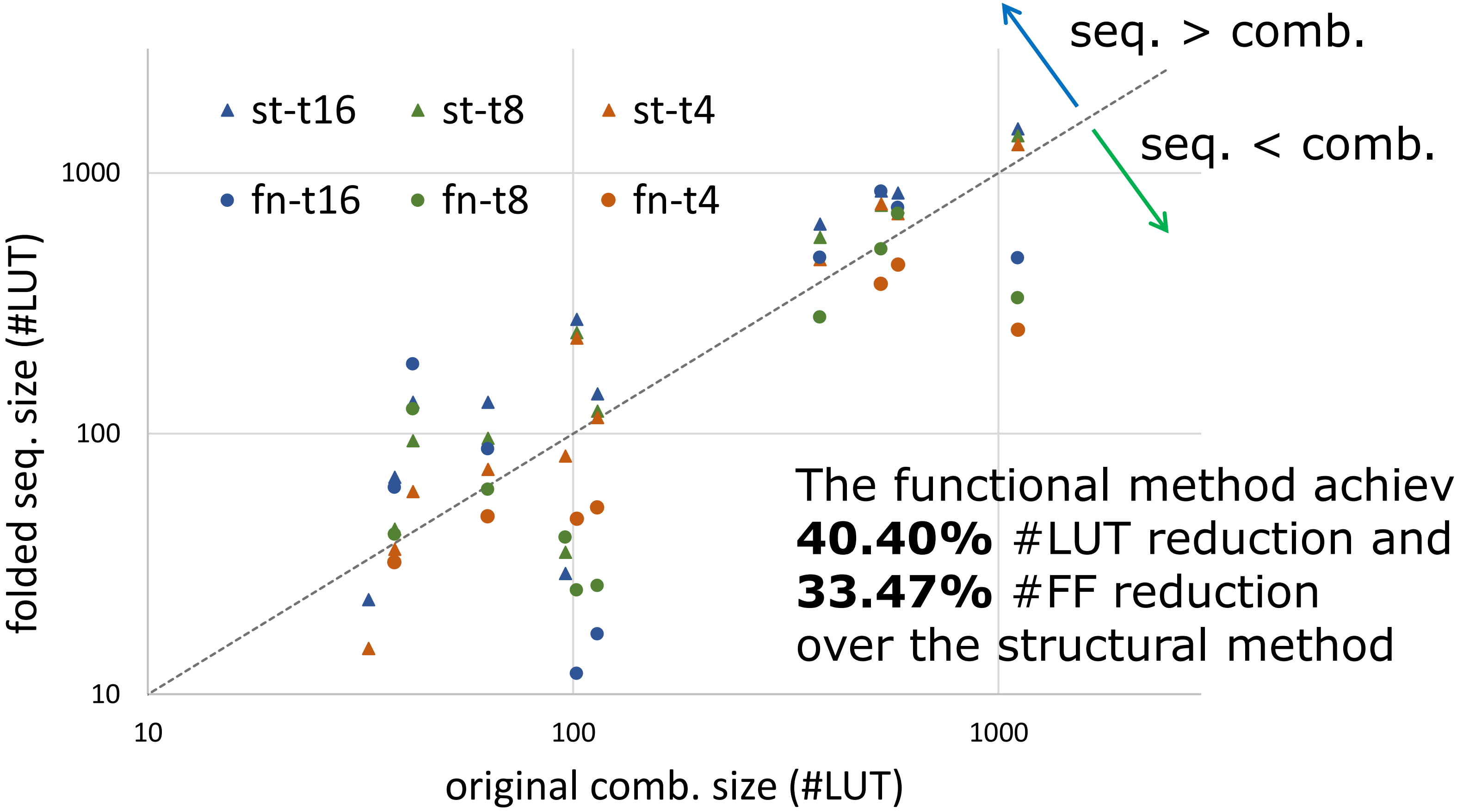
	avg. #LUT overhead
▲ simple	46.59%
▲ st	34.84%
▲ st-s	25.08%
▲ st-f	29.14%
▲ st-sf	20.07%

Functional Method - Setup

- 11 benchmark circuits, each folded by 4, 8 and 16 time-frames
- 300s timeout limit on FSM construction and minimization, individually

Functional Method - Results

- 29 out of 33 cases done within time limit



The functional method achieved **40.40%** #LUT reduction and **33.47%** #FF reduction over the structural method

Comparison of the 2 Methods

Structural

- fast and efficient
- higher circuit complexity

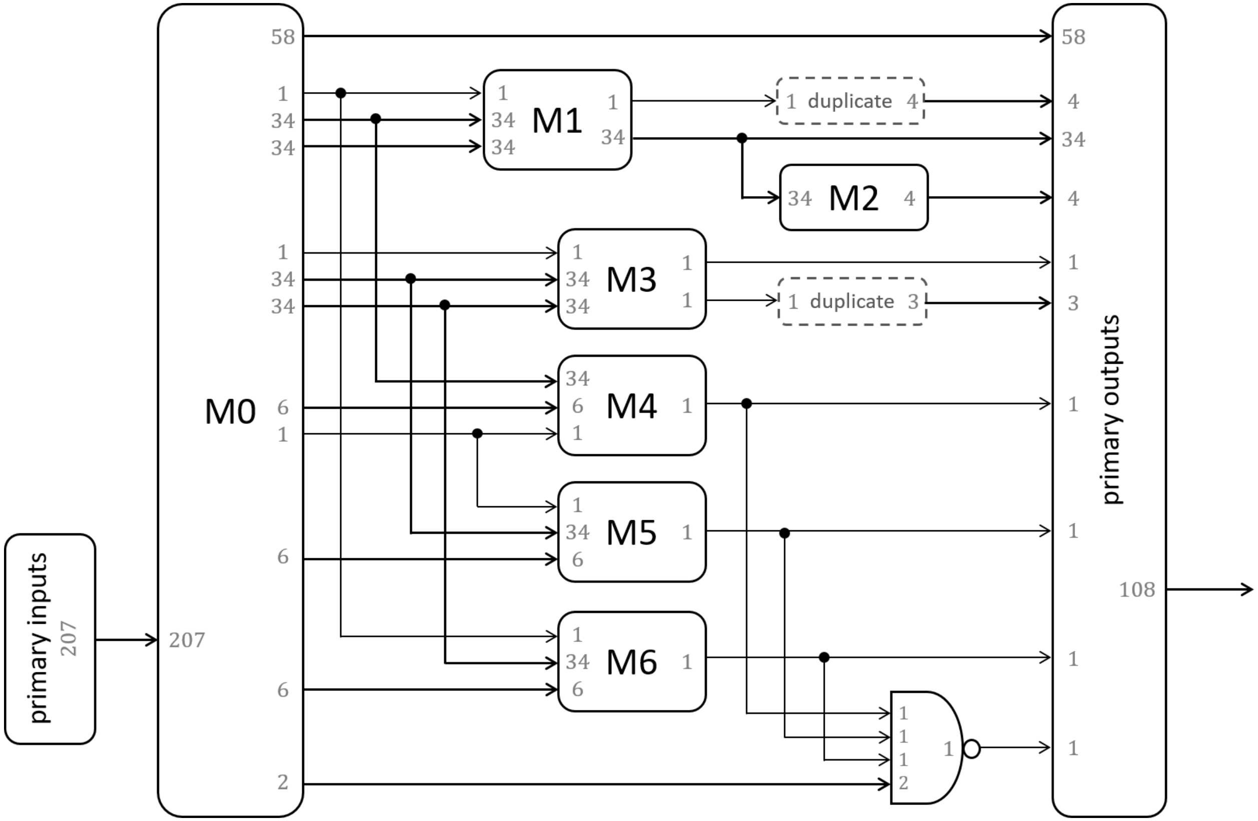
Functional

- high computational cost
- less FF and LUT usage

Is it possible to combine the advantages of the 2 methods?

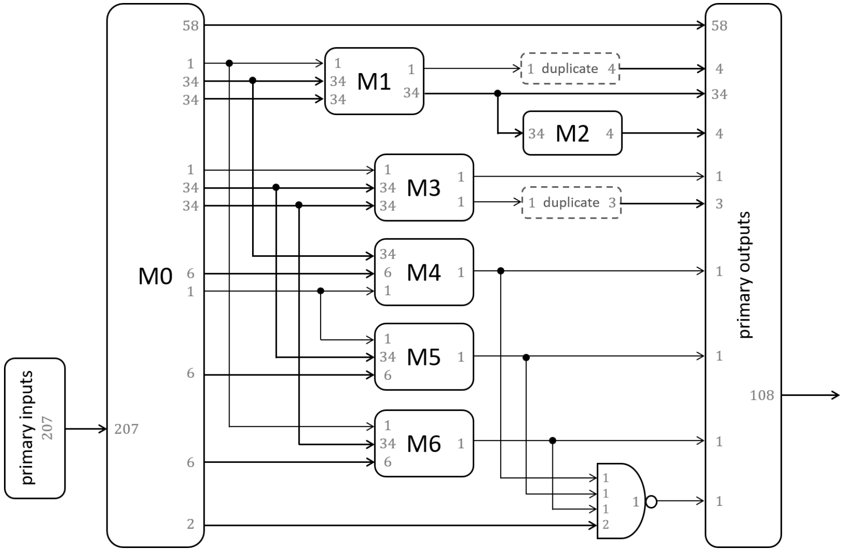
Hybrid Method

- A case study on C7552



hierarchical structure of C7552

Hybrid Method



- C7552 could not be folded by the functional method within timeout limit.
- The hybrid method achieved **55.26%** and **28.81%** reduction in flip-flop and LUT usage over the structural method.

circuit	original					folded						
	#PI	#PO	#gate	#LUT	description	#in	#out	#FF	#gate	#LUT	overhead	method
M0	207	217	330	217	bus signal controller	104	111	5	506	158	-27.19%	structural
M1	69	35	298	78	34-bit adder	35	18	2	128	33	-57.69%	functional
M2	34	4	168	12	sum parity checker	17	3	2	62	9	-25.00%	functional
M3	69	2	144	40	34-bit comparator	35	2	2	72	26	-35.00%	functional
M4	42	1	195	15	sanity checker	21	1	3	87	13	-13.33%	functional
M5	42	1	195	15	sanity checker	21	1	3	87	13	-13.33%	functional
M6	42	1	195	15	sanity checker	21	1	3	92	14	-6.67%	functional
C7552	207	108	1485	340	overall circuit	104	64	17	946	257	-24.41	combined
						104	64	38	1658	361	6.18	structural

CONCLUSIONS
& FUTURE WORK

Conclusions

- We have formulated and provided algorithmic solutions to
  - time-frame folding (TFF)
  - time multiplexing in FPGAs
- The experimental results demonstrated
  - the circuit compaction ability of TFF
  - the scalability of the structural method, the optimization power of the functional method, and the potential of the hybrid method that can achieve the advantages of the 2
- The proposed methods can be applied to
  - sequential synthesis of bounded strategies
  - alleviate the I/O-pin bottleneck of FPGAs
  - various tasks in logic synthesis

Future Work

- We would like to fully automate the hybrid folding method. In the case study we conducted, we relied on the given high-level hierarchical design and manually partitioned the circuit into smaller modules. Therefore, it would be more desirable if the partitioning could be done automatically from a flattened gate-level logic netlist.
- In addition, we would like to investigate other functional decomposition techniques to help mitigate the high computational cost of BDD-based operations.

THE END