Síntesis arquitectónica y de alto nivel

Módulo 4: Fragmentación y Predicción en SAN

Outline

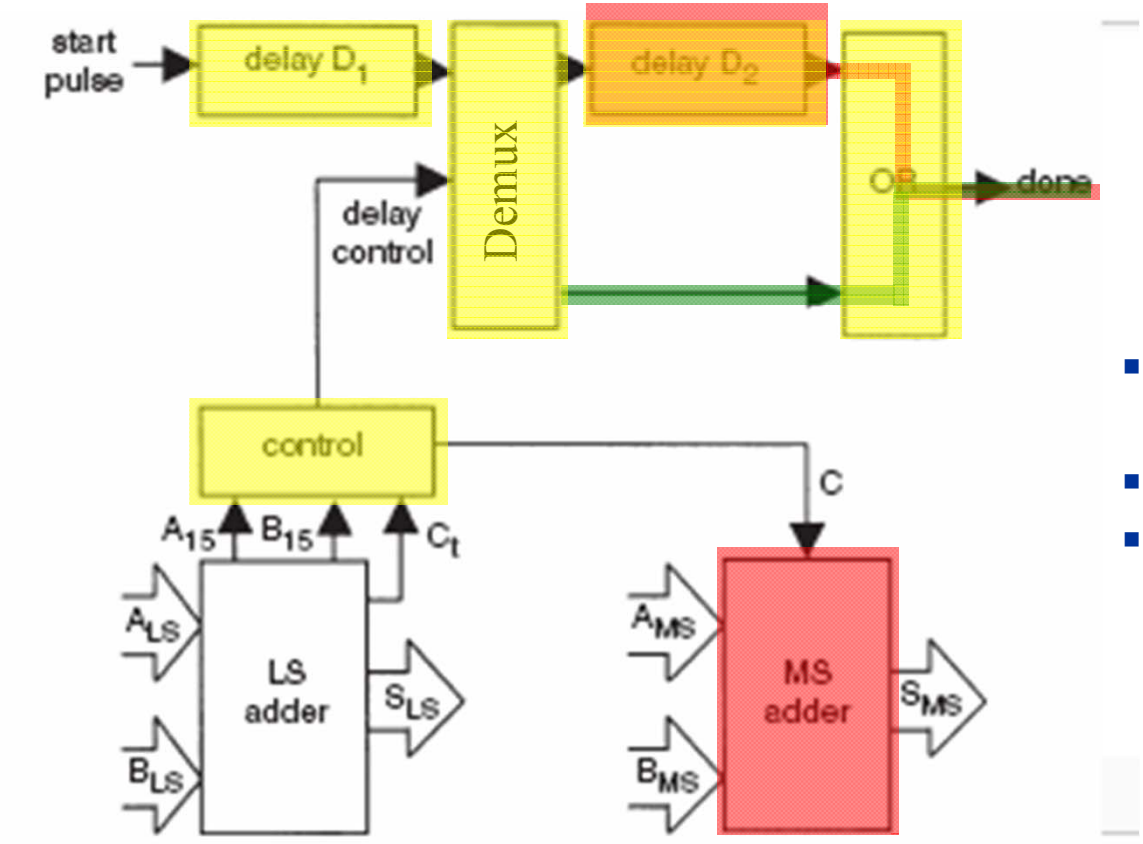
- HLS with predictive FUs
  - FUs design
  - Centralized management
  - Distributed management
- Fragmentation in HLS
  - Dealing with heterogeneous specs
  - Techniques for area optimization
  - Techniques for time optimization
  - Pre-synthesis optimizations

Prediction in HLS

- Main requirements
  - Low percentage of mispredictions
  - Rapid recover from mispredictions
  - Low hardware overhead
  - Ideally, mispredictions should not delay other computations
  - Integrating these ideas in the HLS flow

- Focus:
  - Designs with regularity are mainly based on interconnected cells
    - Problem: delays provoked by inter-cell communication
  - Approach:
    - Predict information from previous cells
  - Computation time depends of prediction accuracy: well suited for the asynchronous domain

A classical proposal[1]



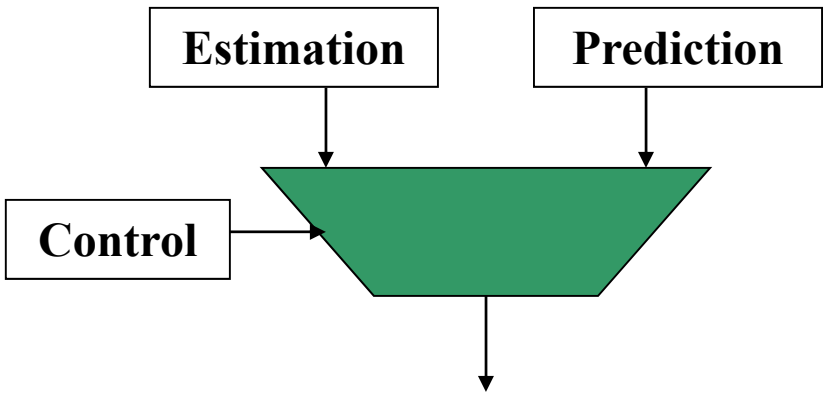
- Estimation based on internal signals
- No history
- Requires FF to work in synchronous domain

1. W.F. Wallace, S.S. Dlay and O.R. Hinton. "Probabilistic carry state estimate for improved asynchronous adder performance", IEE Proc. Comput. Digit. Tech., 2001, 148, (6), pp. 221-226.

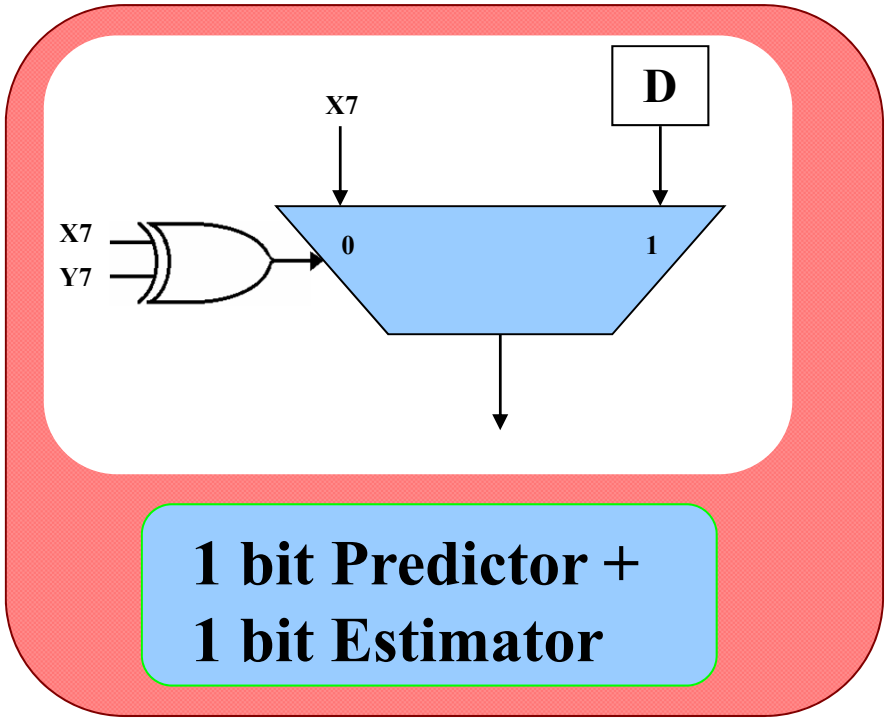
History based prediction

- Basic ideas:
  - Exploit data correlation to increase hit rates
  - Similar to branch prediction
  - Simple structures required: very small area penalty
- Pure prediction
  - Decide actual carry based on past information
- Hybrid prediction
  - In some cases, history is not necessary because the right carry can be easily obtained.

Hybrid Prediction

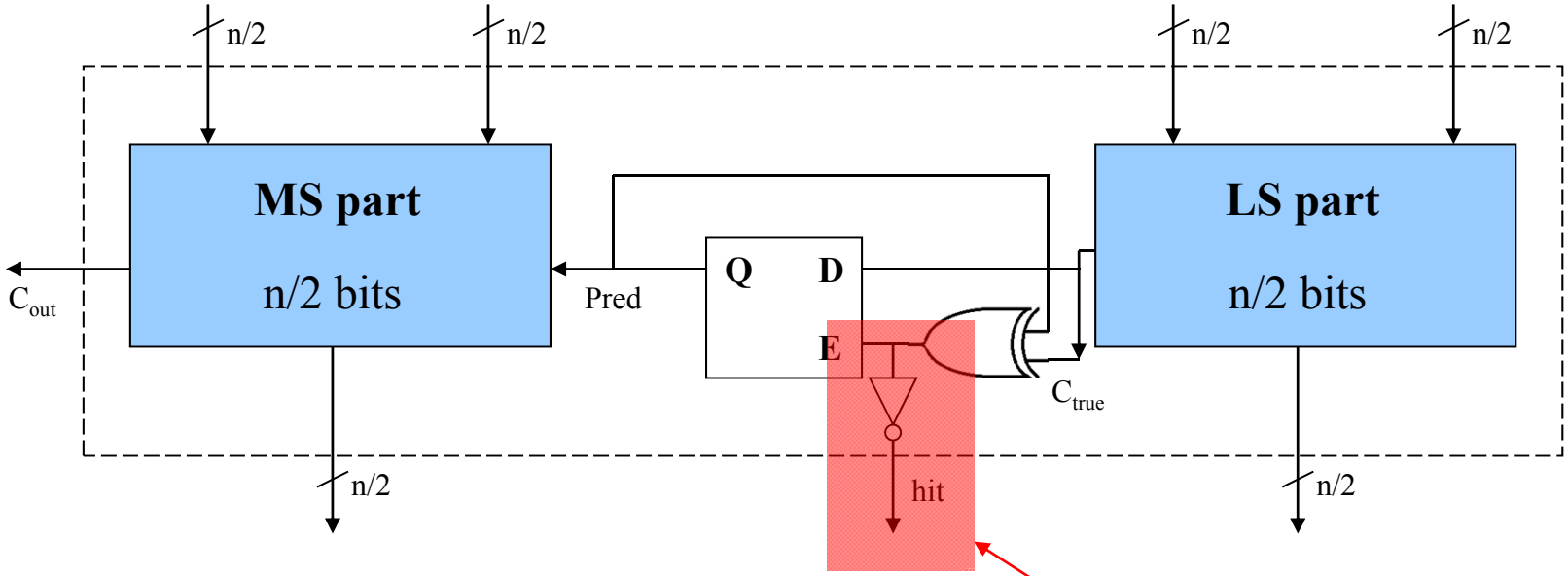


General scheme



1 bit Predictor +
1 bit Estimator

Simple adder design



Up to theoretical 50% gain in performance

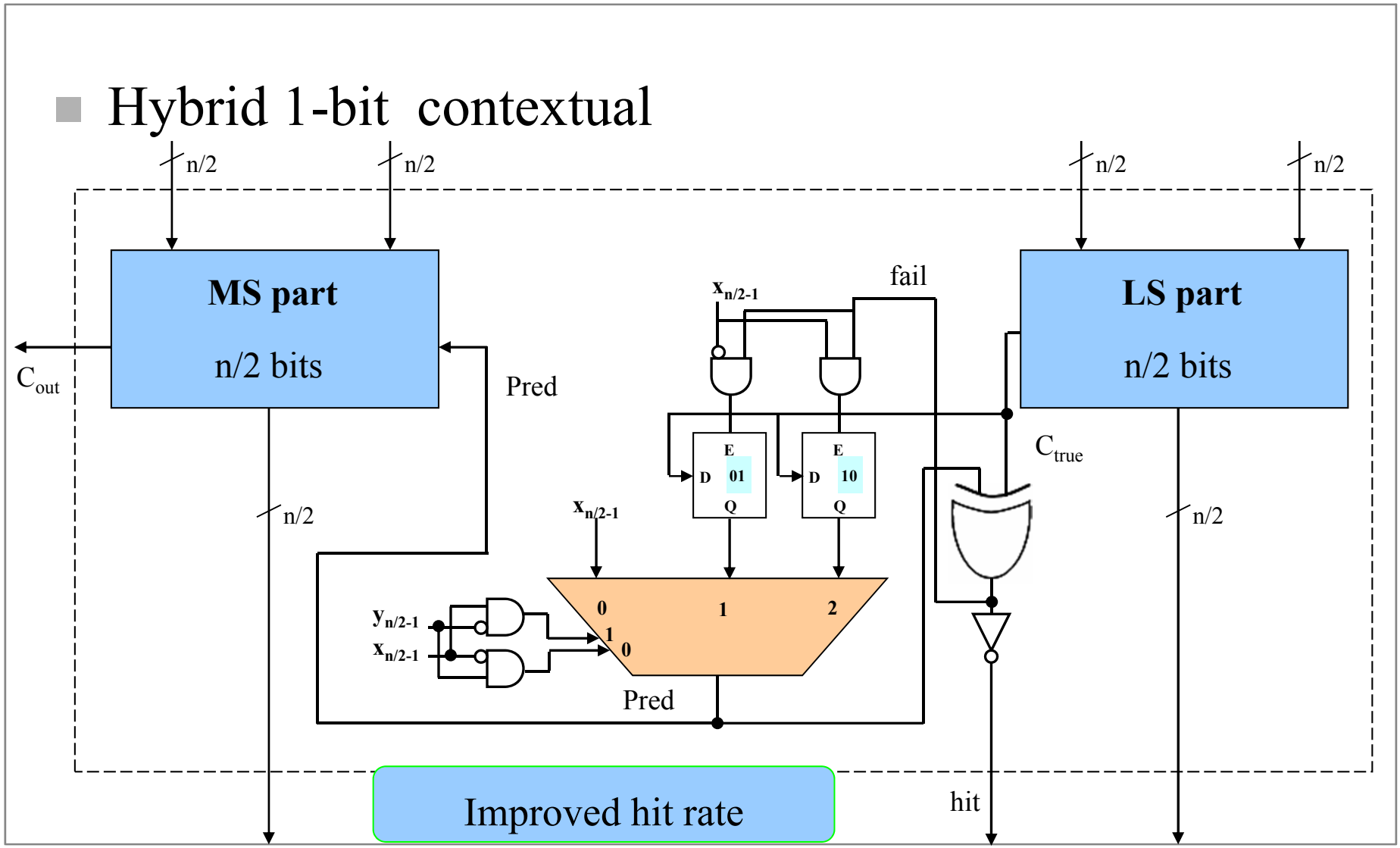
Hit: 1 cycle / Miss: 2 cycles

Hit rate essential

To the controller

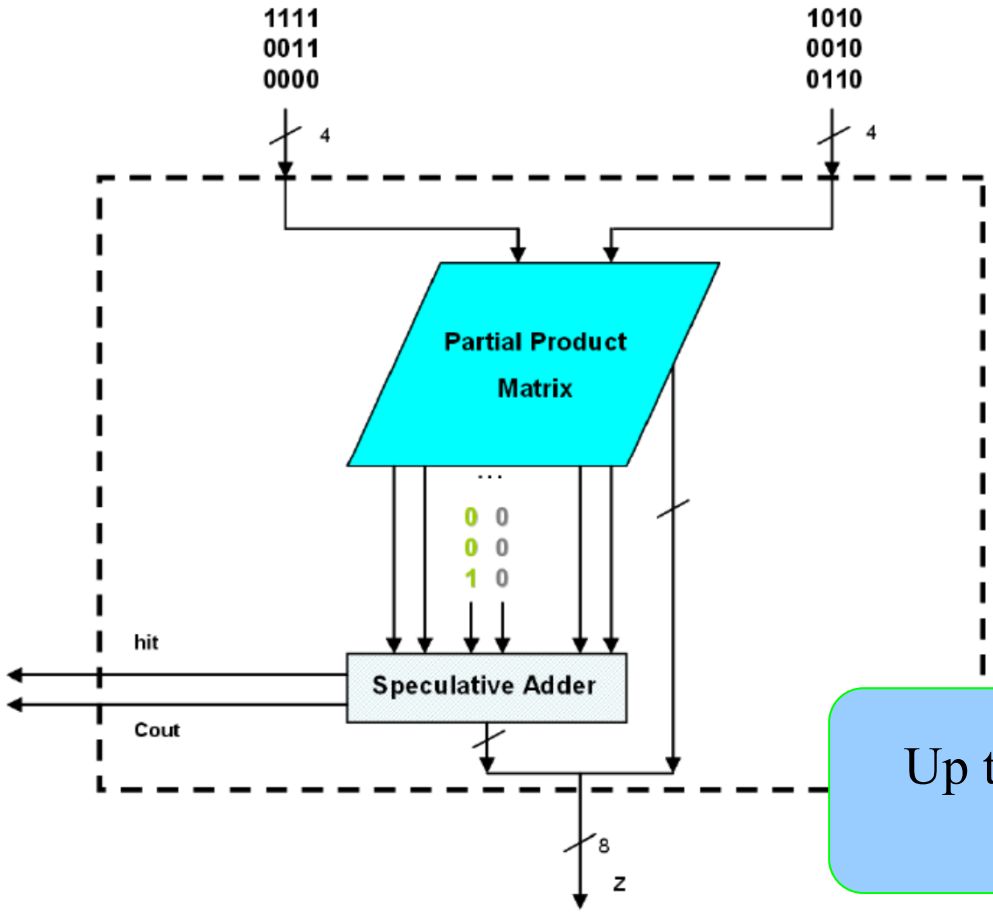
Alternative adder design

- Hybrid 1-bit contextual



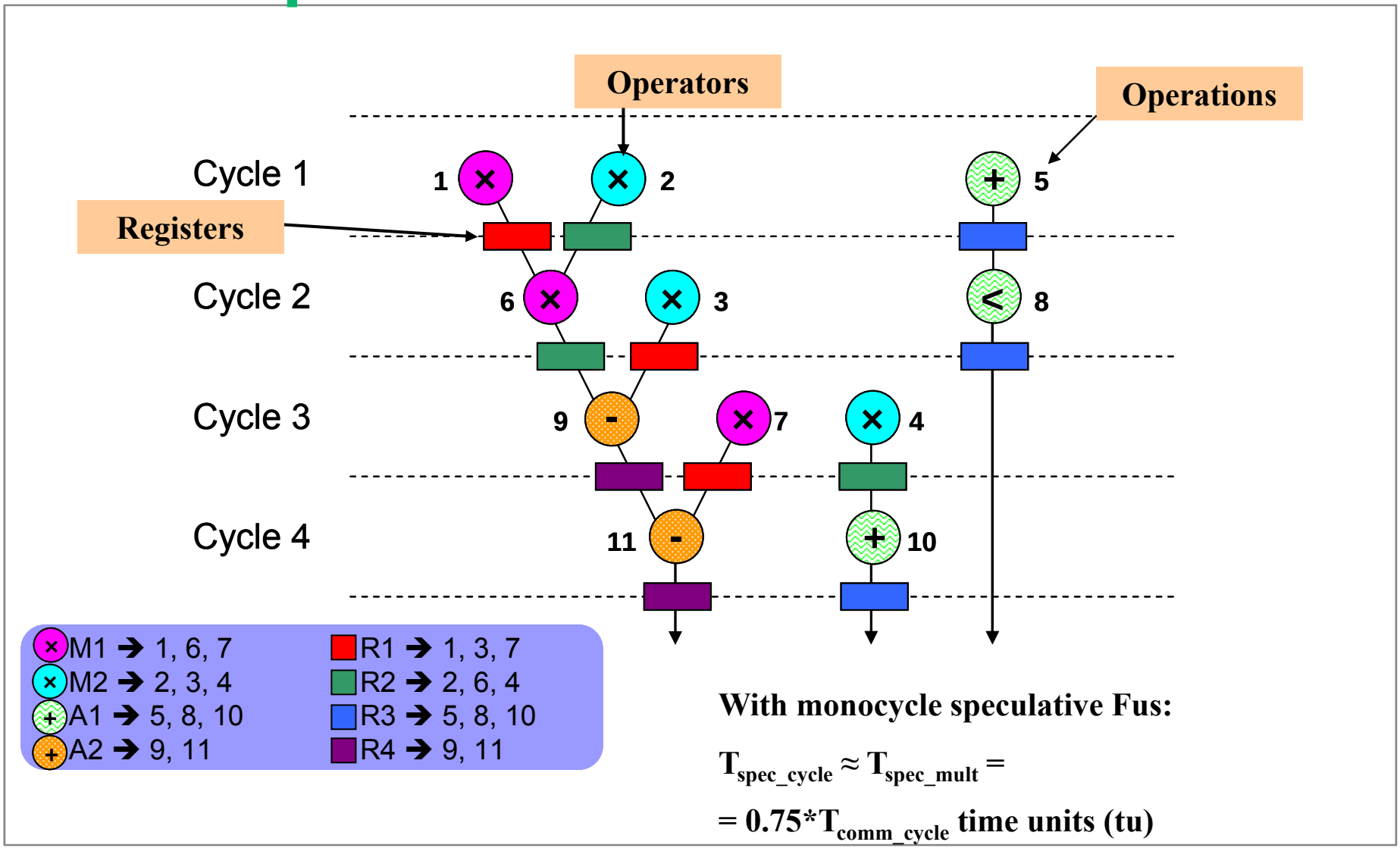
Speculative multiplier design

- Braun / Baugh-Wooley style



Up to theoretical 25% gain in performance

Using speculative FUs in HLS: example

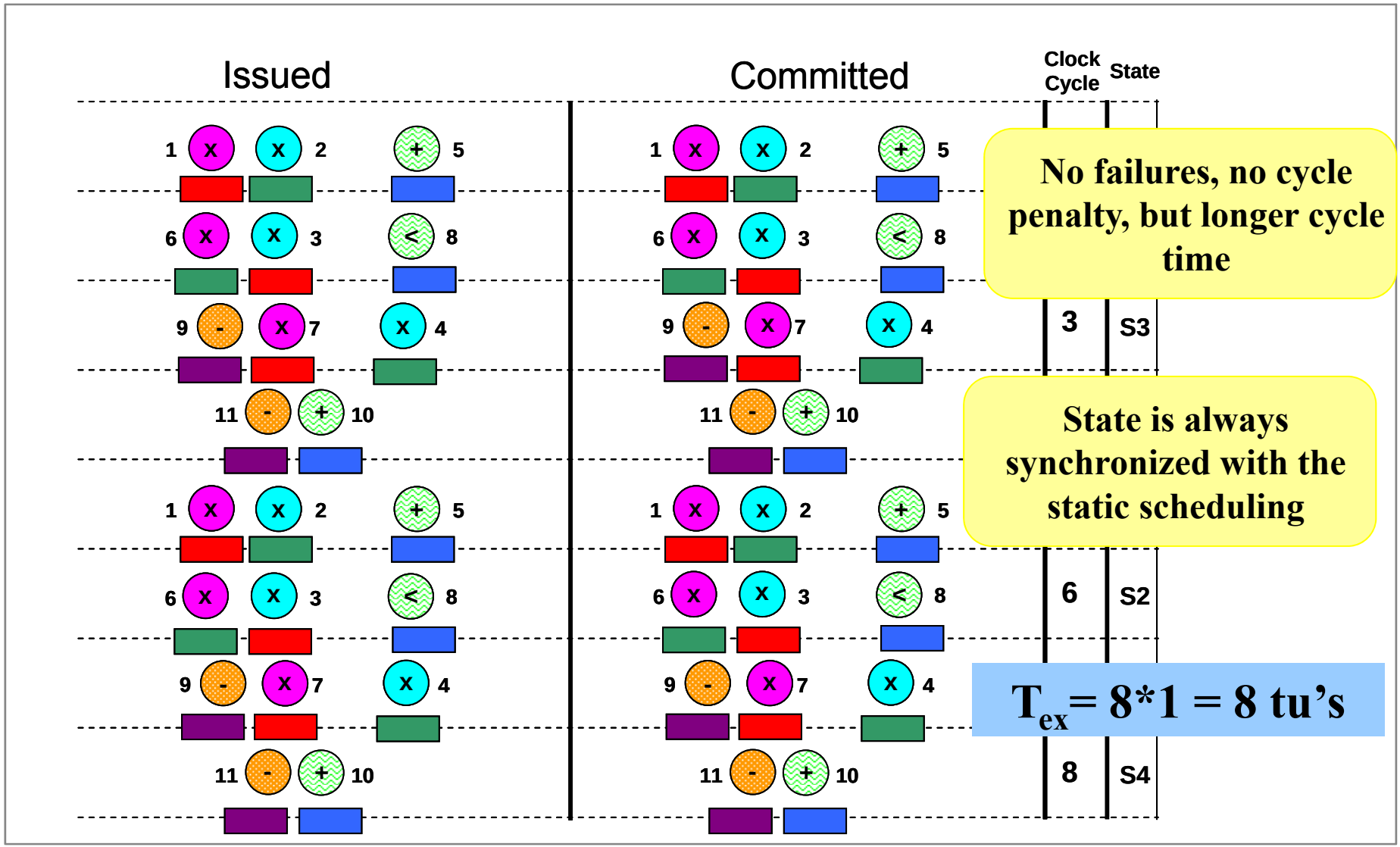


With monocycle speculative Fus:

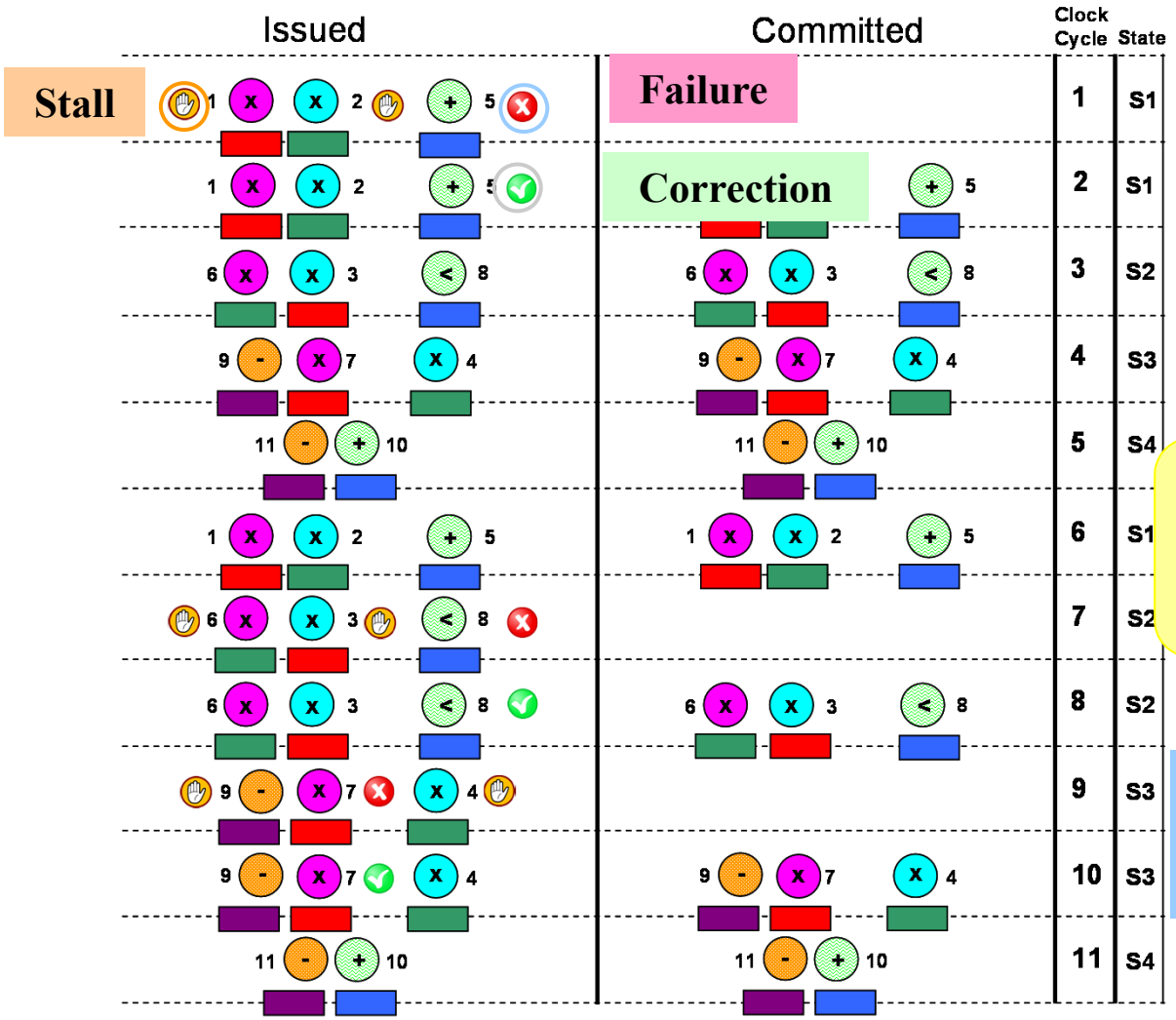
$T_{spec\_cycle} \approx T_{spec\_mult} =$

$= 0.75 * T_{comm\_cycle}$ **time units (tu)**

Common execution flow



SFUs + Centralized Management

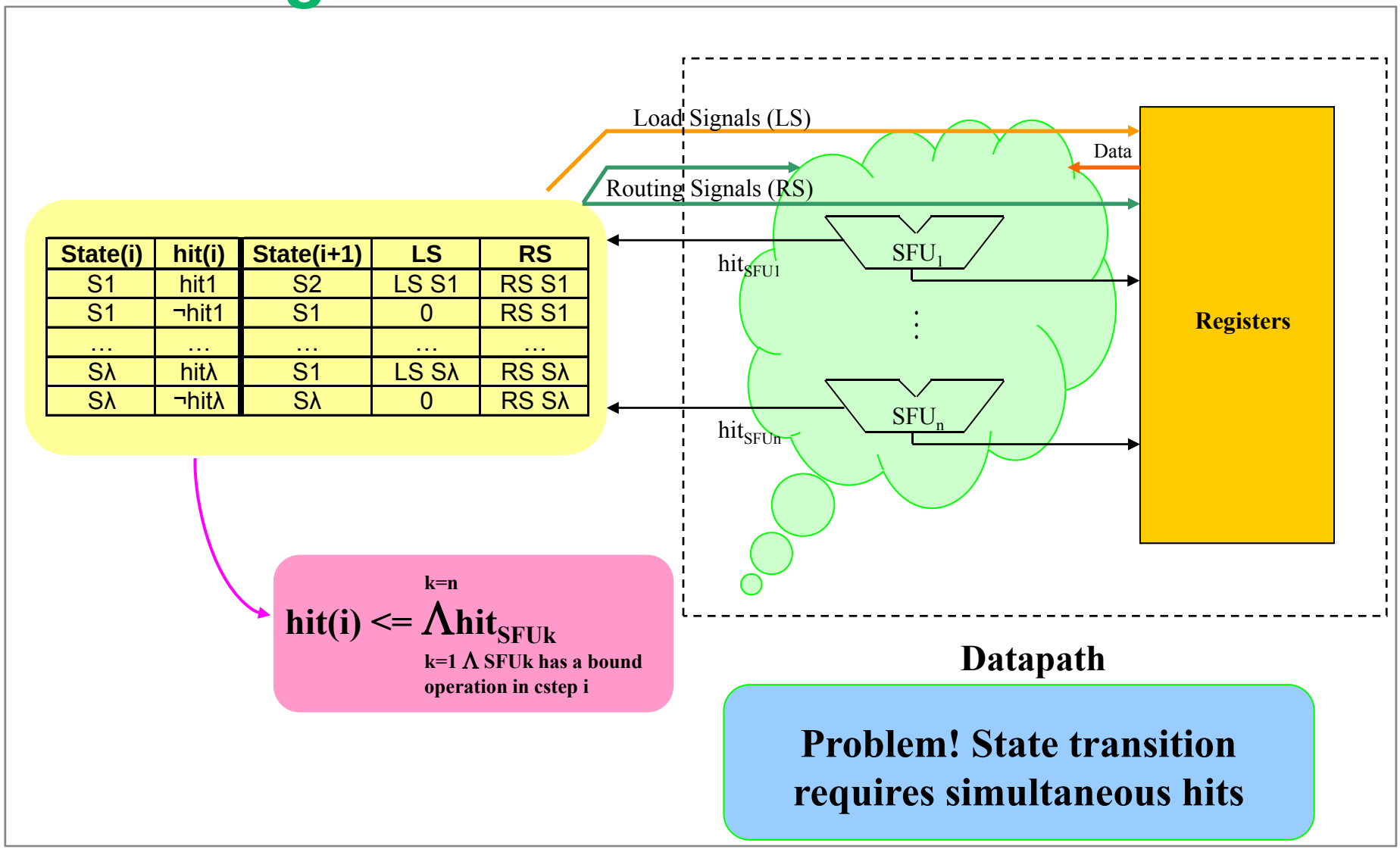


3 failures, 3 penalty cycles

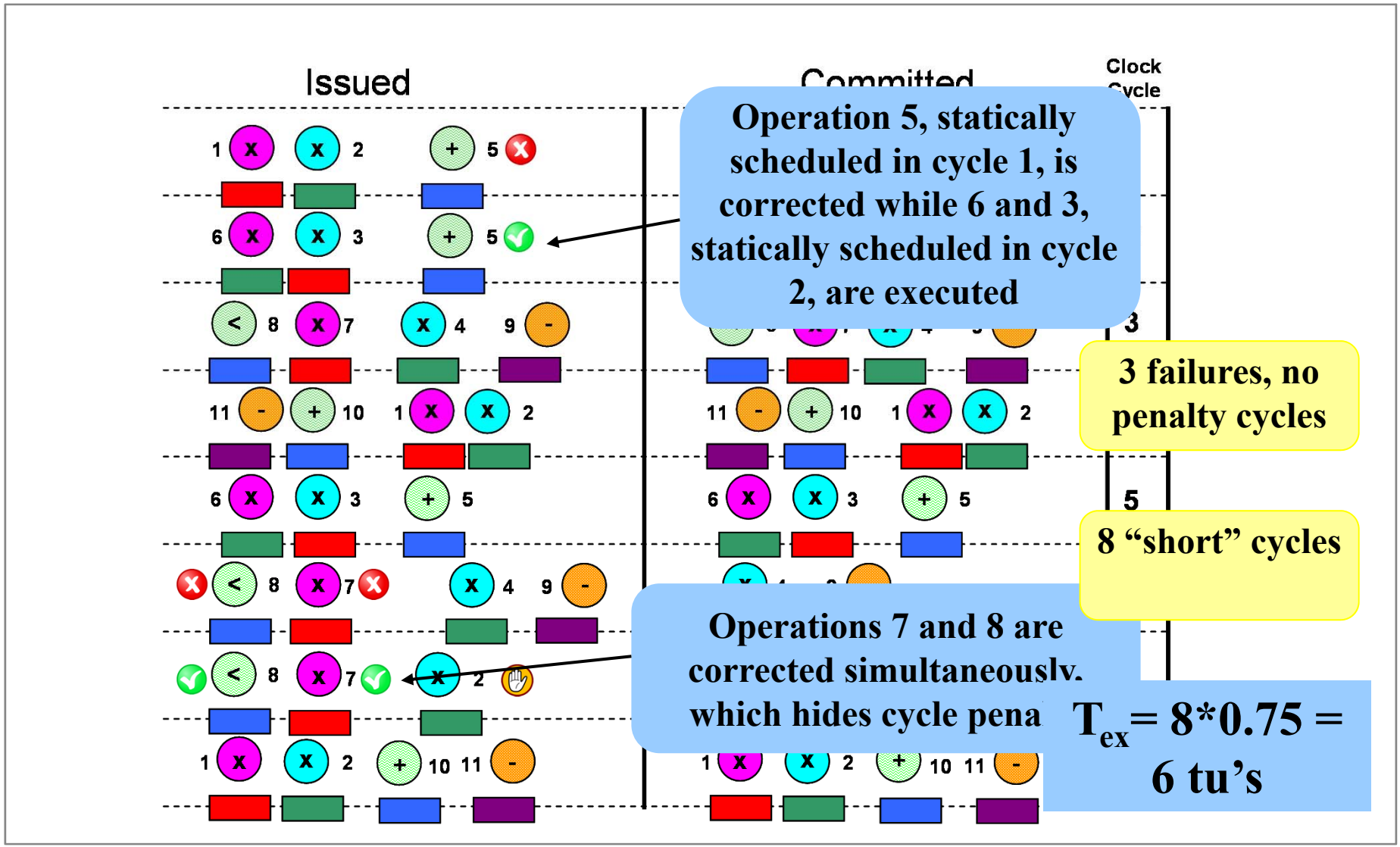
State is always synchronized with the static scheduling

T_{ex} = 11*0.75 = 8.25 tu's

Controller for Centralized Management



SFU + Distributed Management

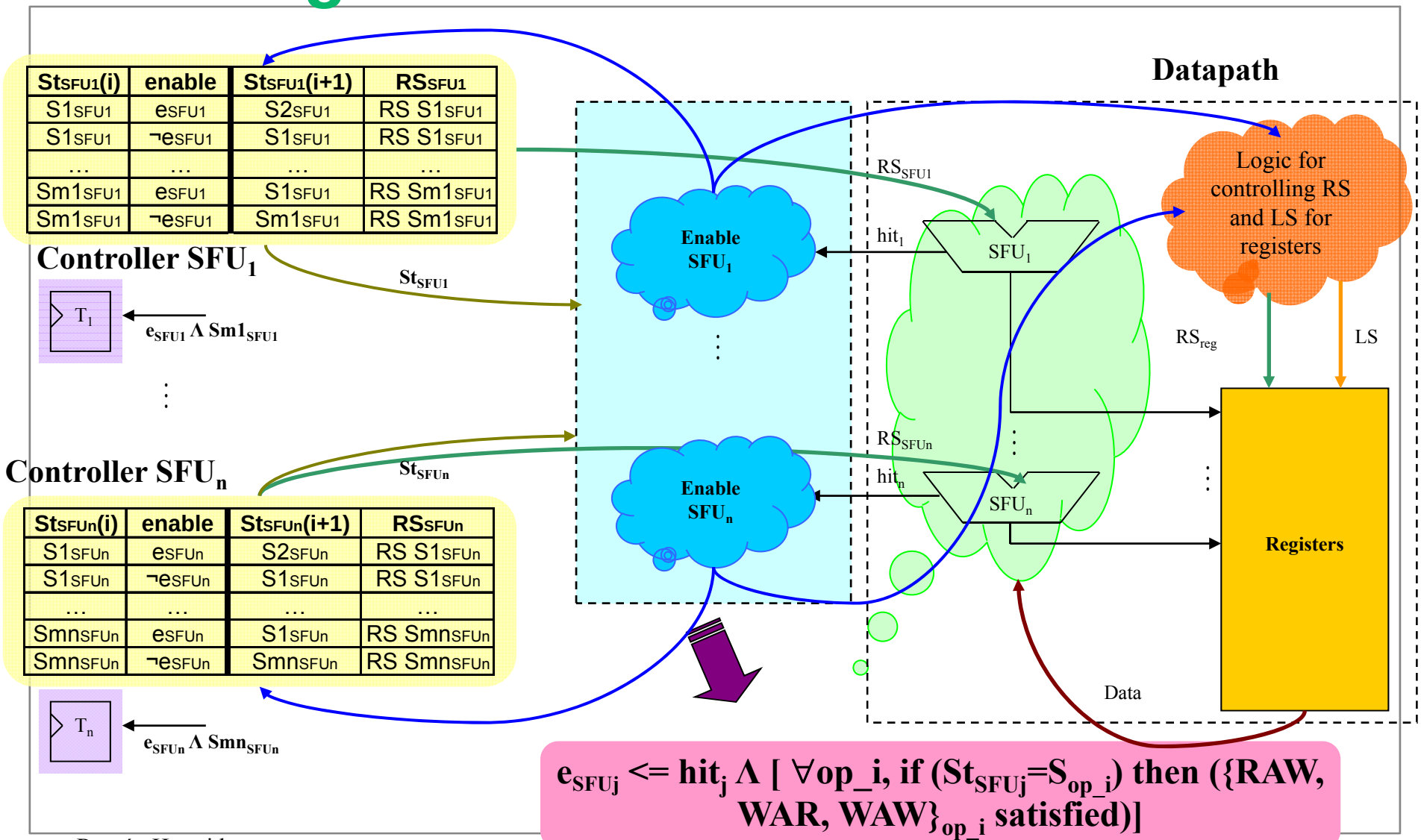


Distributed Management: theoretical foundation & requirements [1]

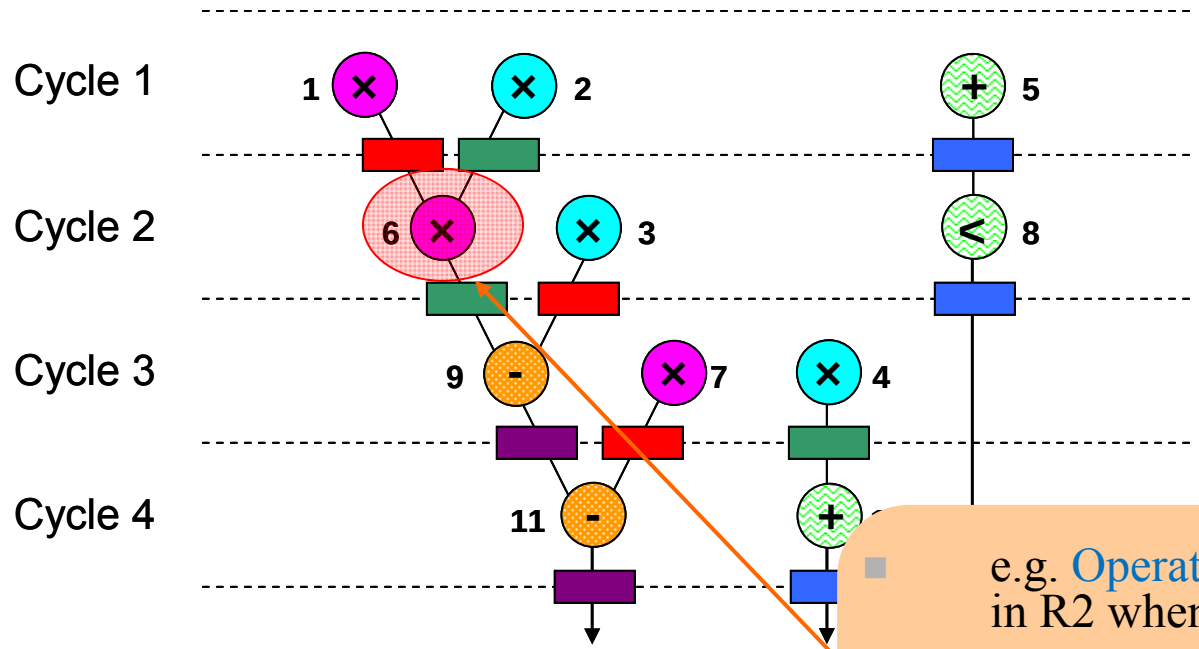
- State: 1 controller per SFU
  - 1 state value per operation bound to the SFU
- Commit Signals Logic Unit. Decides:
  - If an operation can be written (commit) in its corresponding register
  - If the SFU controller can advance to the next state
    - HW to generate these signals is deduced from DFG preprocessing
- Static schedule only defines a partial order
  - Strict order is dynamically calculated
  - Actual scheduling is influenced by mispredictions and hazards
- Multicycling and chaining must be supported
- Low area overhead required

1. A.A.D. Barrio, S. Ogrenci Memik, M.C. Molina, J.M. Mendías, and R. Hermida. A distributed controller for managing speculative functional units in high level synthesis. IEEE TCAD, 30:350-363, 2011.

Controller for Distributed Management



Distributed Management: commit conditions



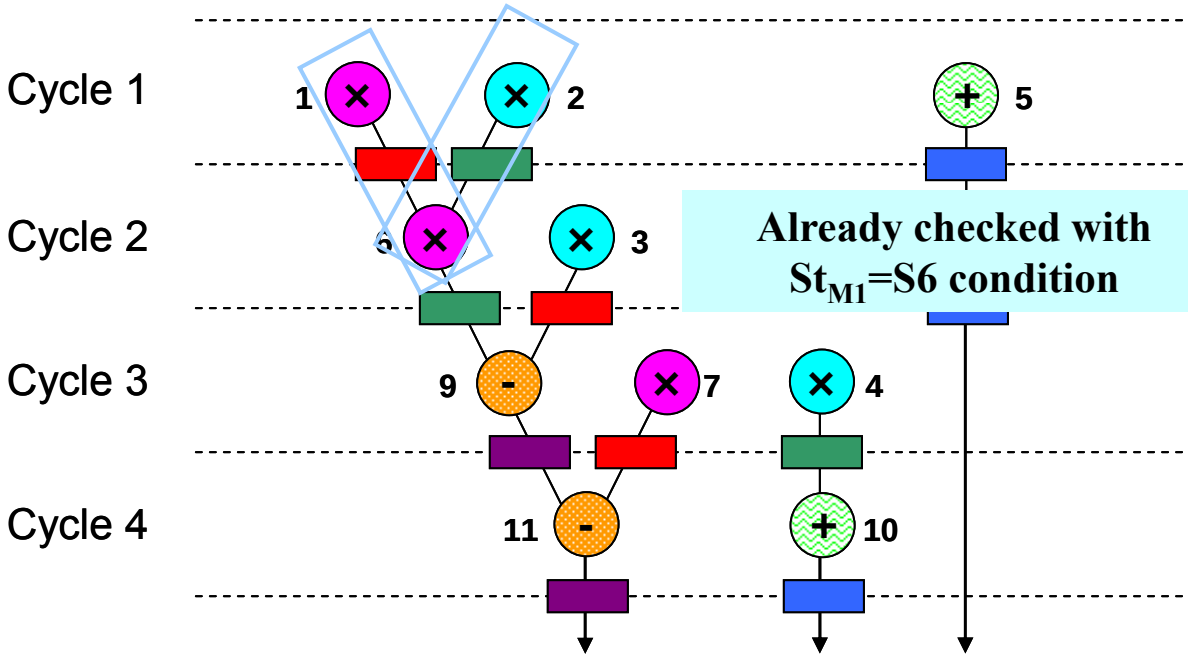
Speculative Execution Paradigm

- e.g. Operation 6 is written/committed in R2 when:
  - The M1 controller is in state S6
  - Operations 1 (RAW) and 2 (RAW, WAW) have been written

And M1 **hit** signal is **true**

e_{M1} <= hit_{M1} and [(St_{M1}=S6 and {hazards are solved}) or (St_{M1}=S1 and {h.s.}) or ((St_{M1}=S7 and {h.s.})]

Distributed Management: solving hazards



Already checked with St_{M1}=S6 condition

Operation 6 …

RAW hazard with operation 1 …

StM1 “>” S1

RAW and WAW hazards with operation 2 …

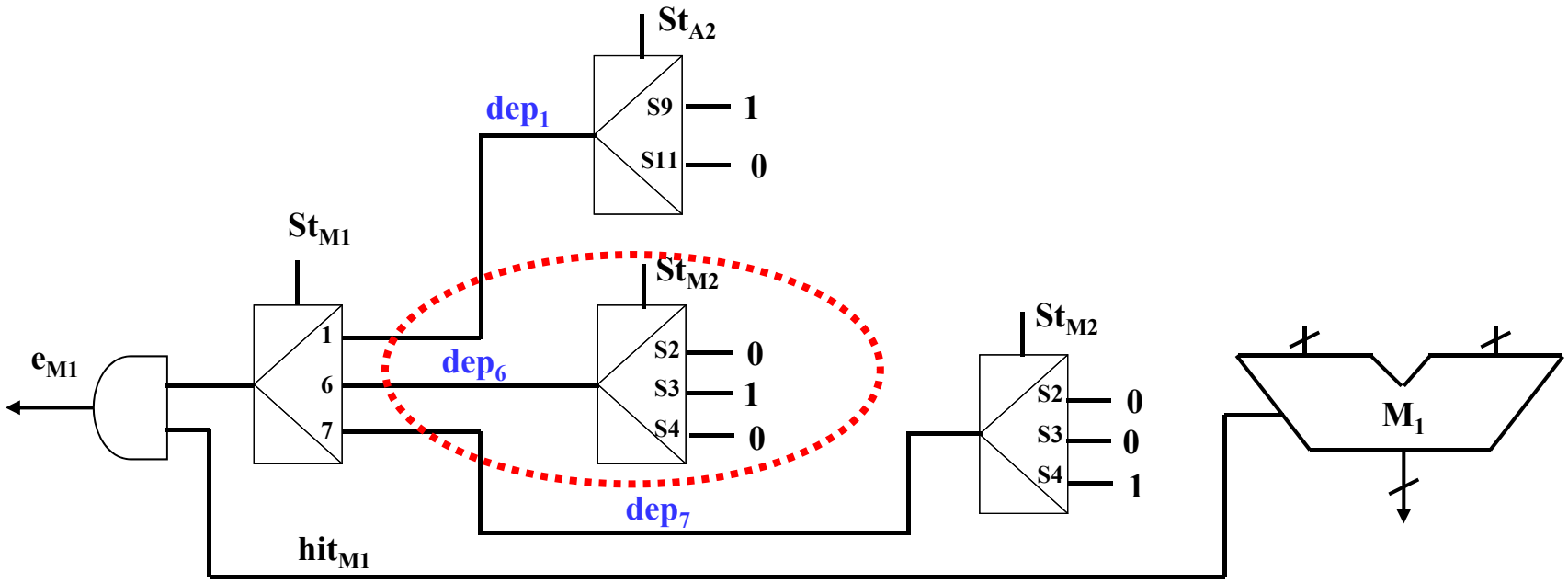
StM2 “>” S2

e_{M1} <= hit_{M1} and [(St_{M1}=S6 and {St_{M2}=S3 or St_{M2}=S4}) or (...) or (....)]

Enable, routing, and load signals must be synthesized

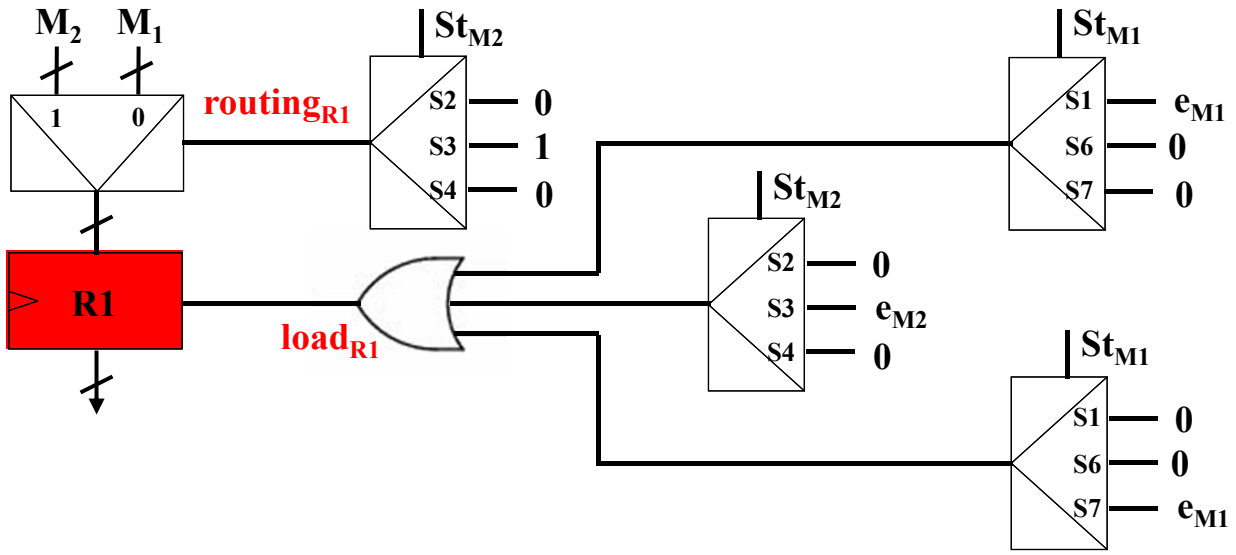
Enable signals: example

- Study of M1 controller
- Synthesis process must deduce the value for mux inputs (challenging problem)
- Remark: this is only a canonical representation. Actual implementation can be simplified.



Routing and load signals: example

- Study of register R1
- Loaded when
  - M2 controller in state 3 and state transition enabled, or
  - M1 controller in state 1 or 7, and state transition enabled
- Register input: if M2 controller in state 3, then M2; otherwise M1

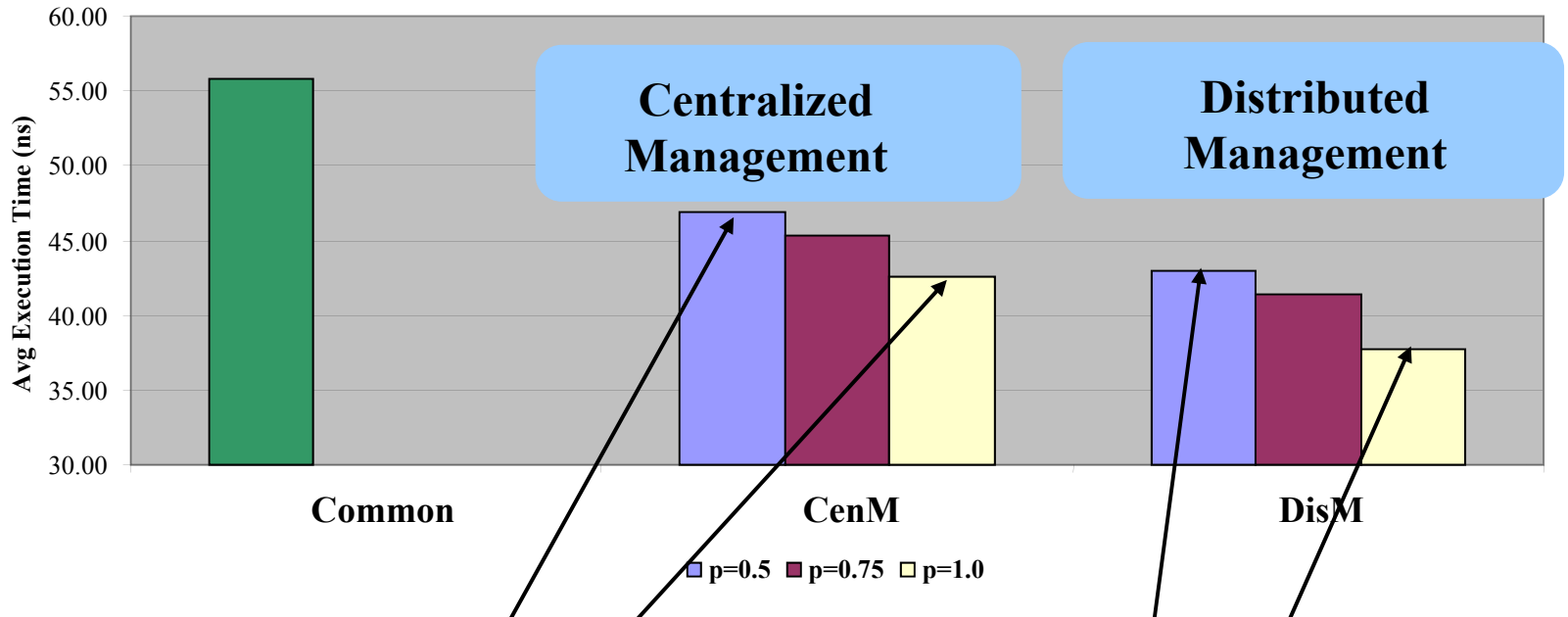


Support for multicycling and chaining

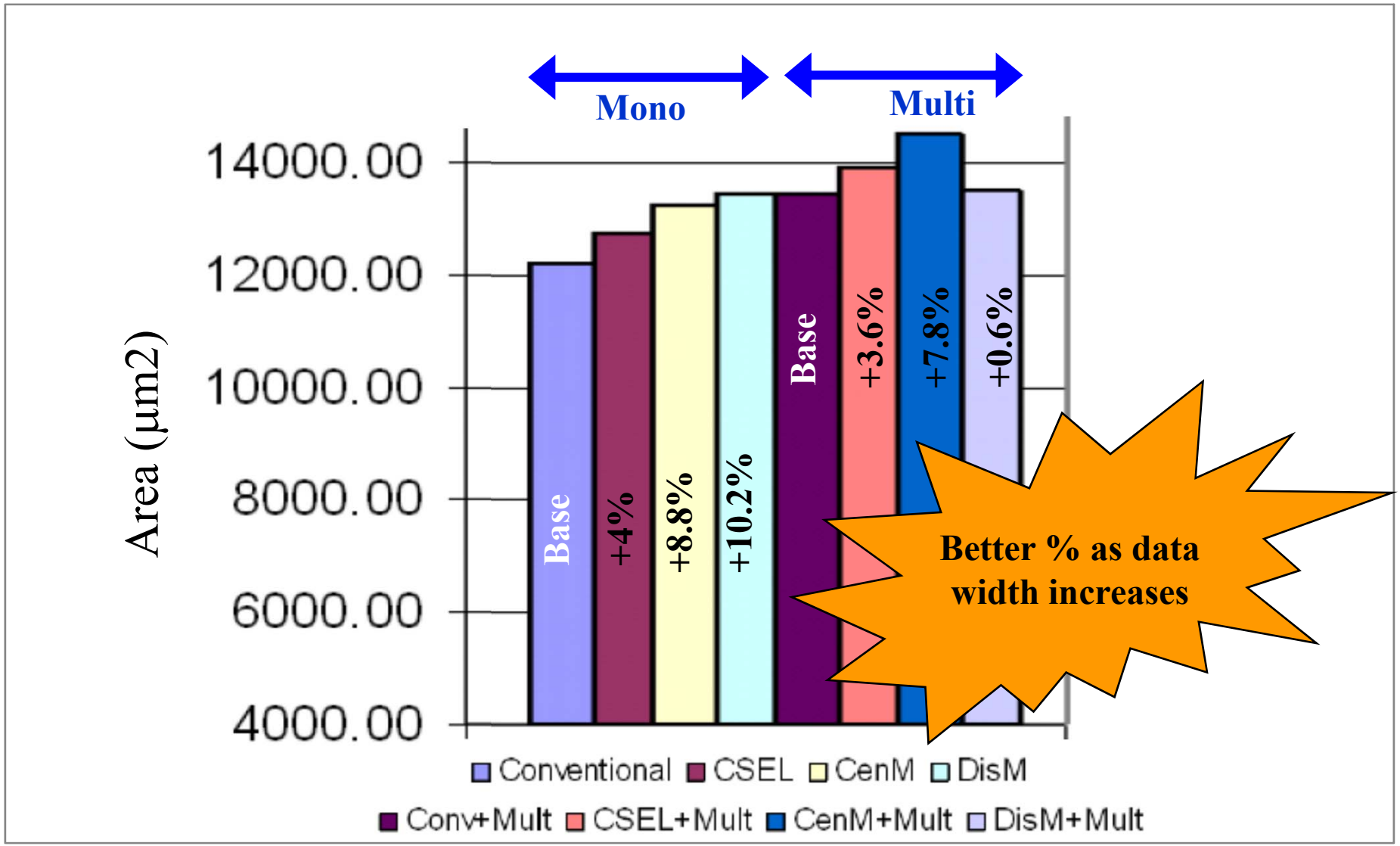
- Important requirement of any HLS strategy
- Multicycling is specially important with SFUs
  - In case of misprediction, only the last part of the operation must be done again
- HW implication of multicycling:
  - Small counter associated to each multicycled SFU
  - Loaded with: Number of computation cycles / number of recovery cycles.
- HW implications of chaining:
  - Chained operations must commit simultaneously

Computation time

- **Average Execution Time Per Iteration**
  - **Input specs: DiffEq, 2EWF, DCT, IDCT, Lattice Filter, LMS**
  - **Input data: from fully random to highly correlated**
  - **Multicicycling and chained allowed**



Area overhead

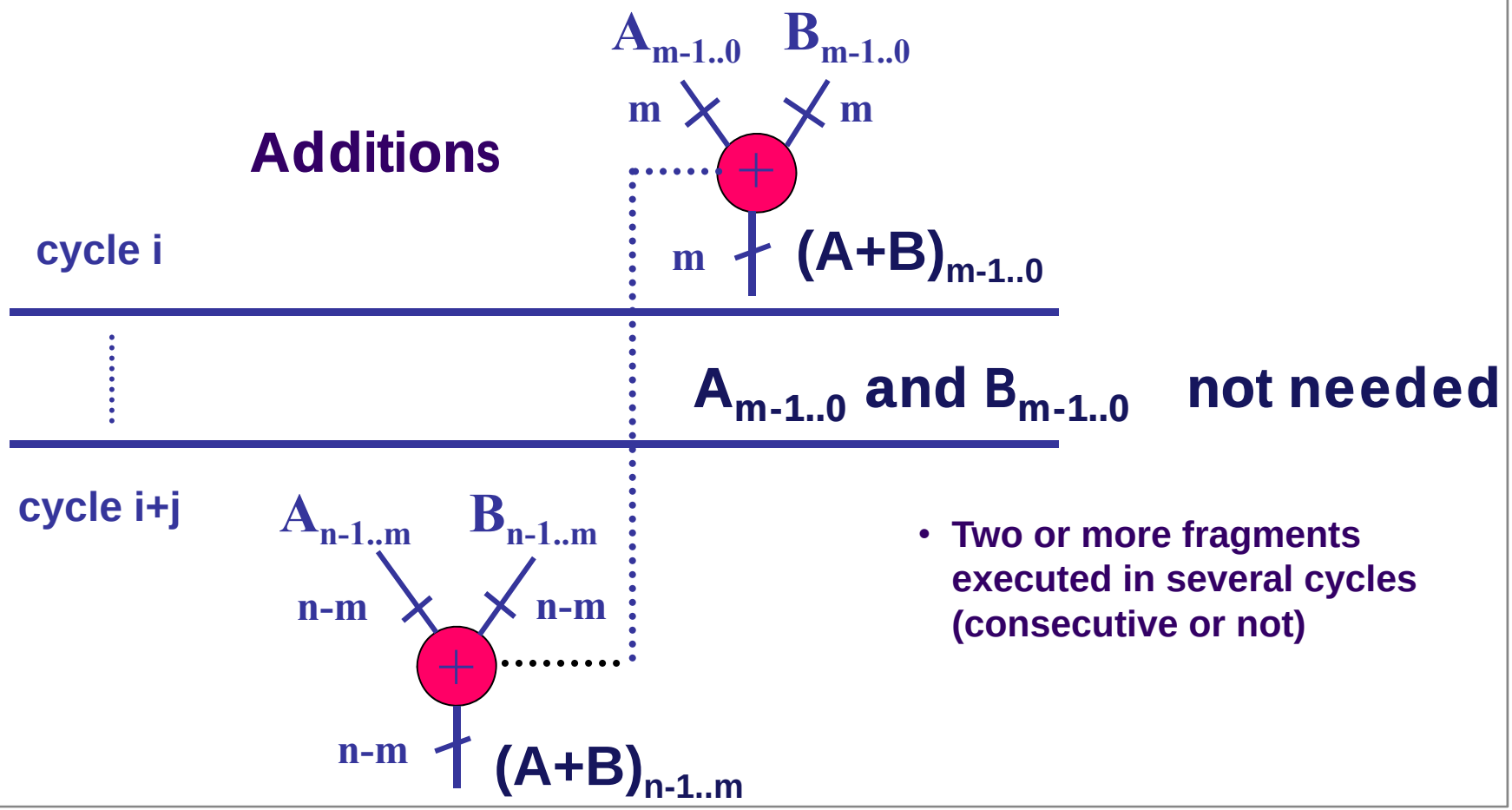


Fragmentation: opportunities for HLS

- Key idea:
  - HLS golden rule: Balanced designs at the clock cycle level
  - However… imbalance very common (problem-inherent causes)
- Area optimization
  - Balance of computational effort at the bit level among the different control steps: Minimizing idle FUs
- Execution time optimization
  - Balance the bit-level delays of operations in every cycle
- Approach
  - Splitting/merging operation slices than can be executed in several control steps (maybe non-consecutive)
  - Adopted in the context of heterogeneous specs … *but can also be applied to homogeneous specs*

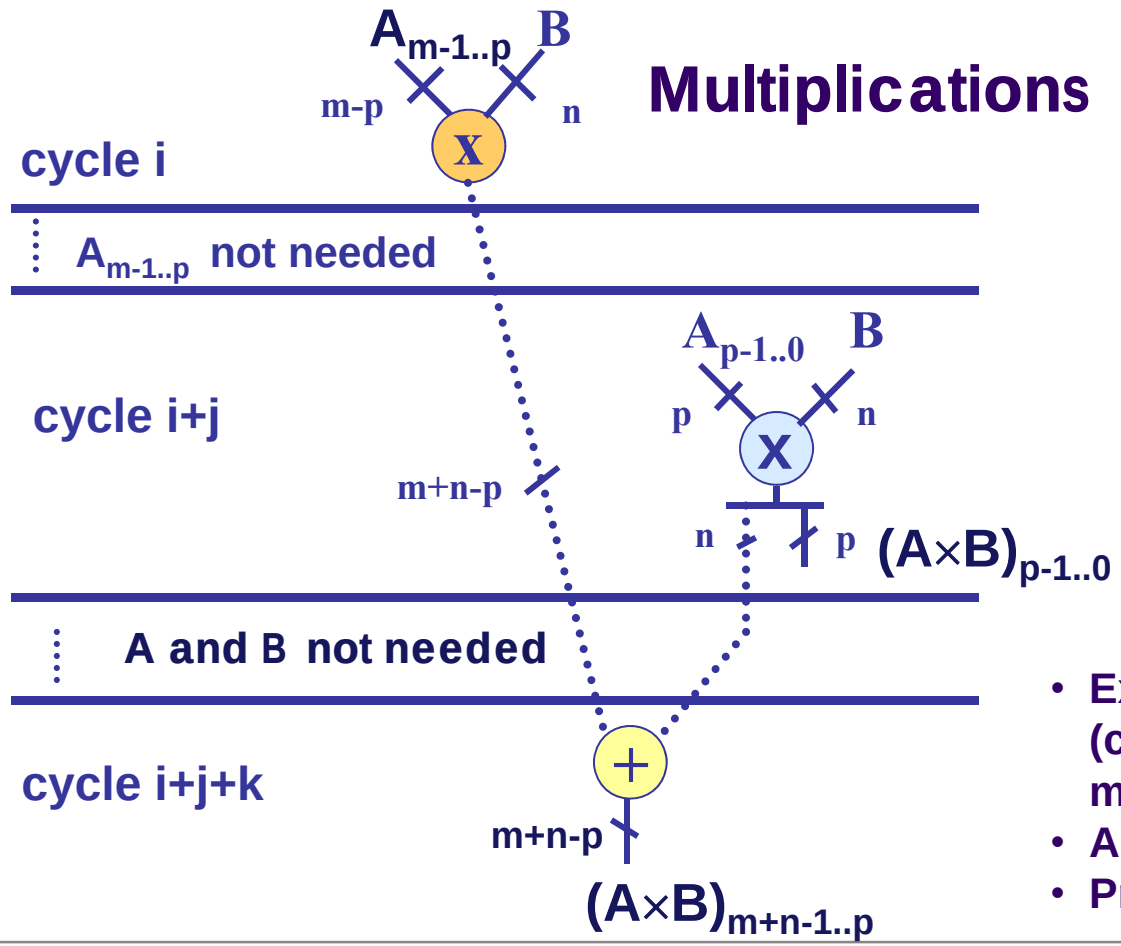
Area optimization: basic techniques

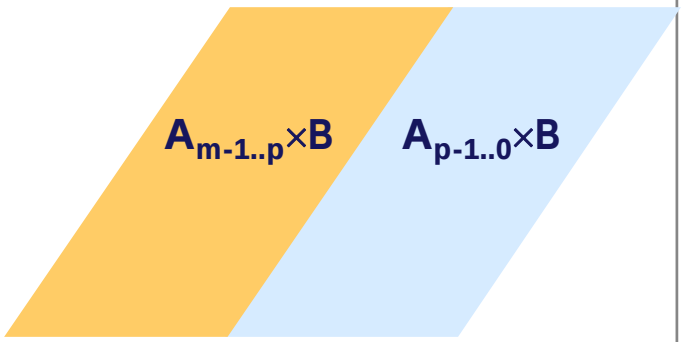
- Scheduling



Area optimization: basic techniques

- Scheduling





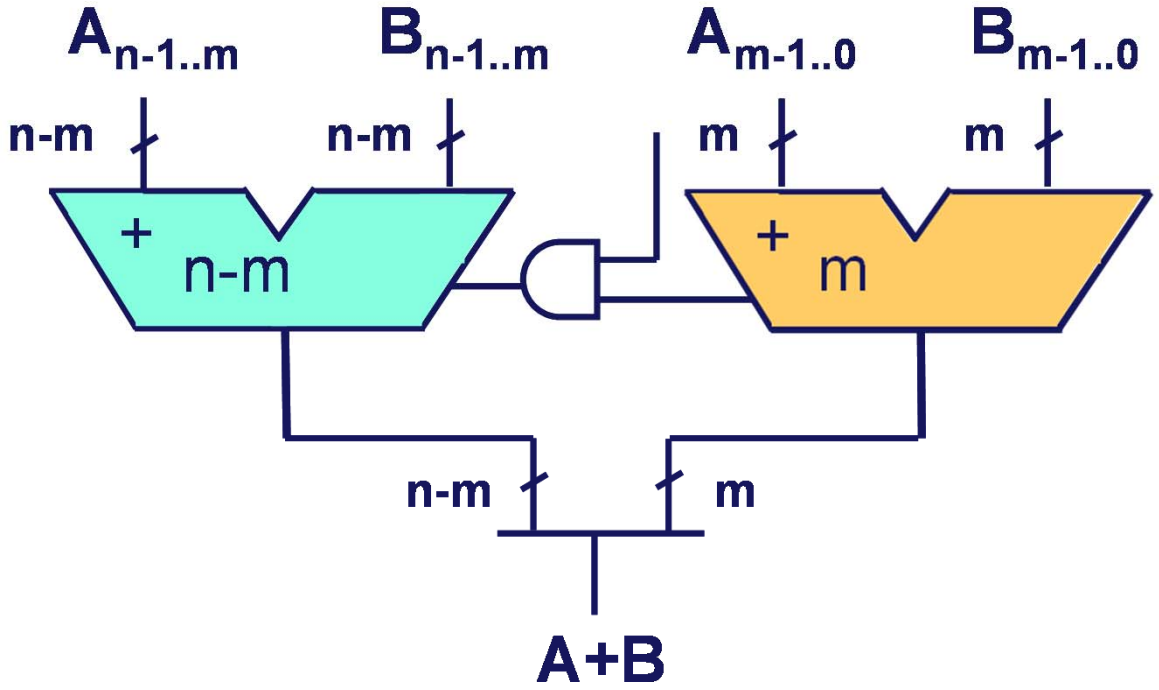
- Executed in several cycles (consecutive or not) using smaller multipliers.
- Adder to combine results
- Products computed in any order

Area optimization: basic techniques

- Operation scheduling with fragmentations is quite different from multicycling
  - One operation scheduled in several (consecutive or not) cycles: fragments
  - Each fragment could be implemented by a different FU
  - Every fragment narrower than original operation
  - Storage of operands not required in every cycle
  - Relaxing RAW dependences
- Also, quite different from pipelining
  - The same FU could be used to implement several fragments

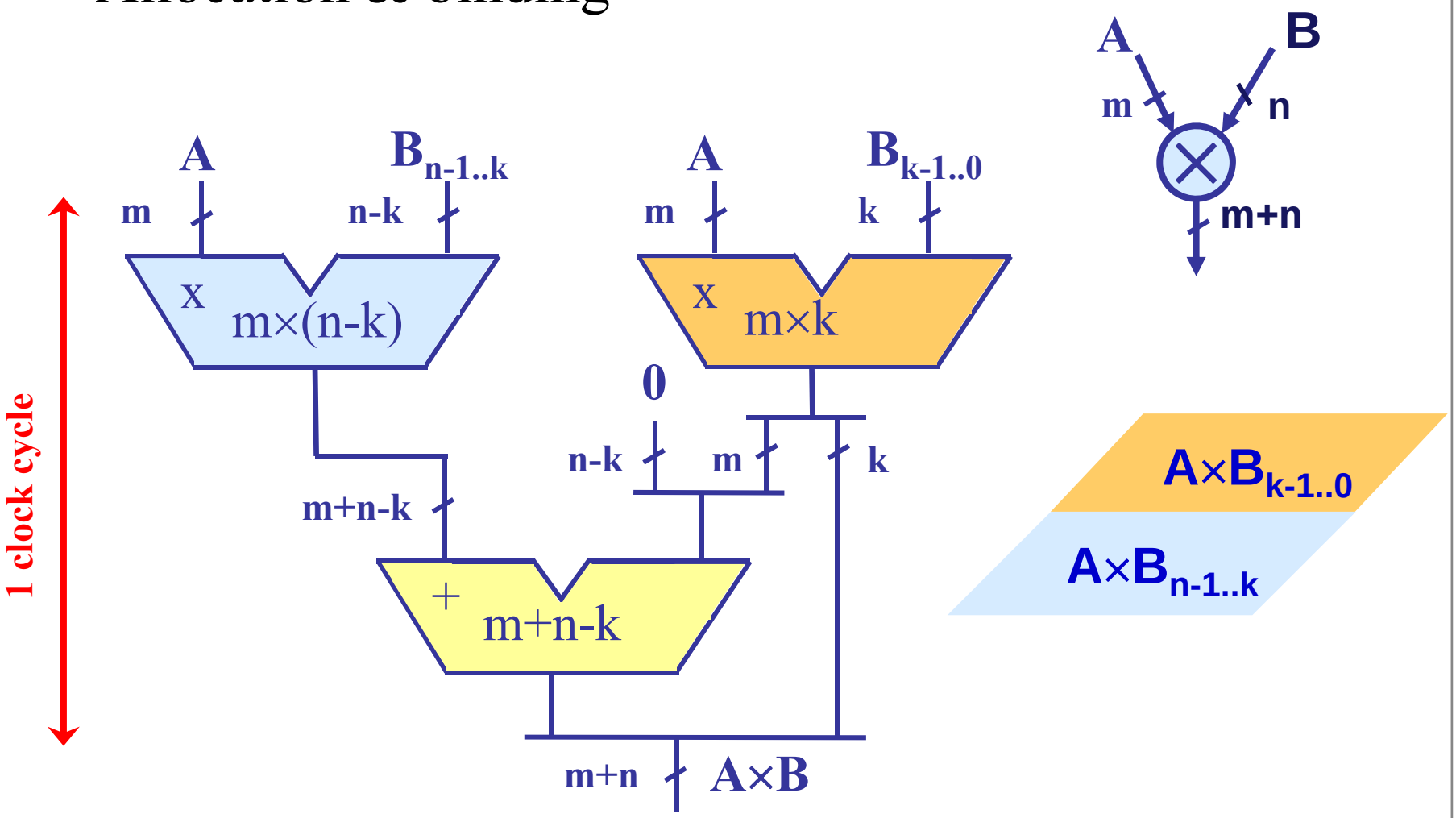
Area optimization: basic techniques

- Allocation & binding
  - Some operations implemented merging previously allocated HW modules



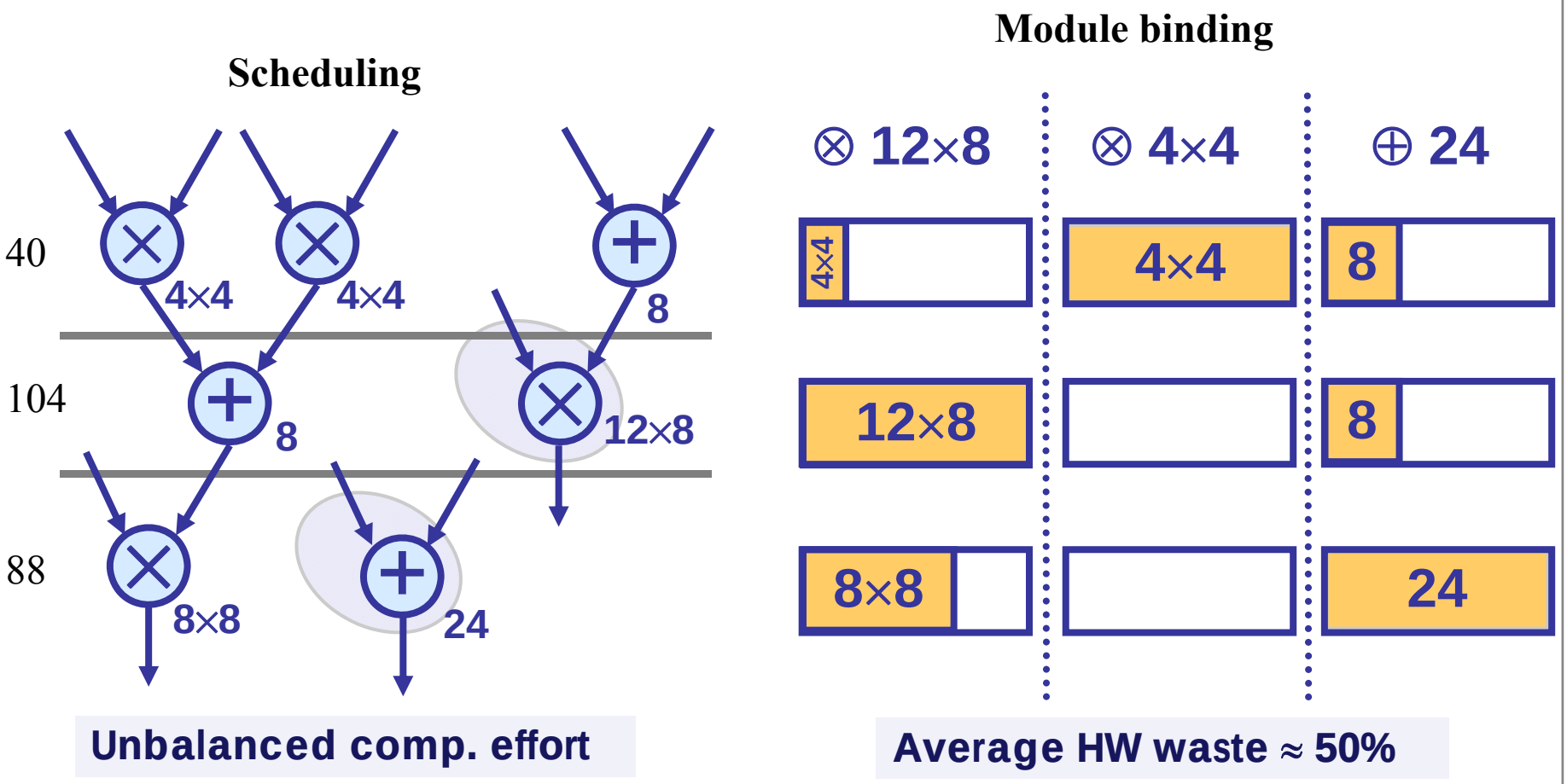
Area optimization: basic techniques

- Allocation & binding

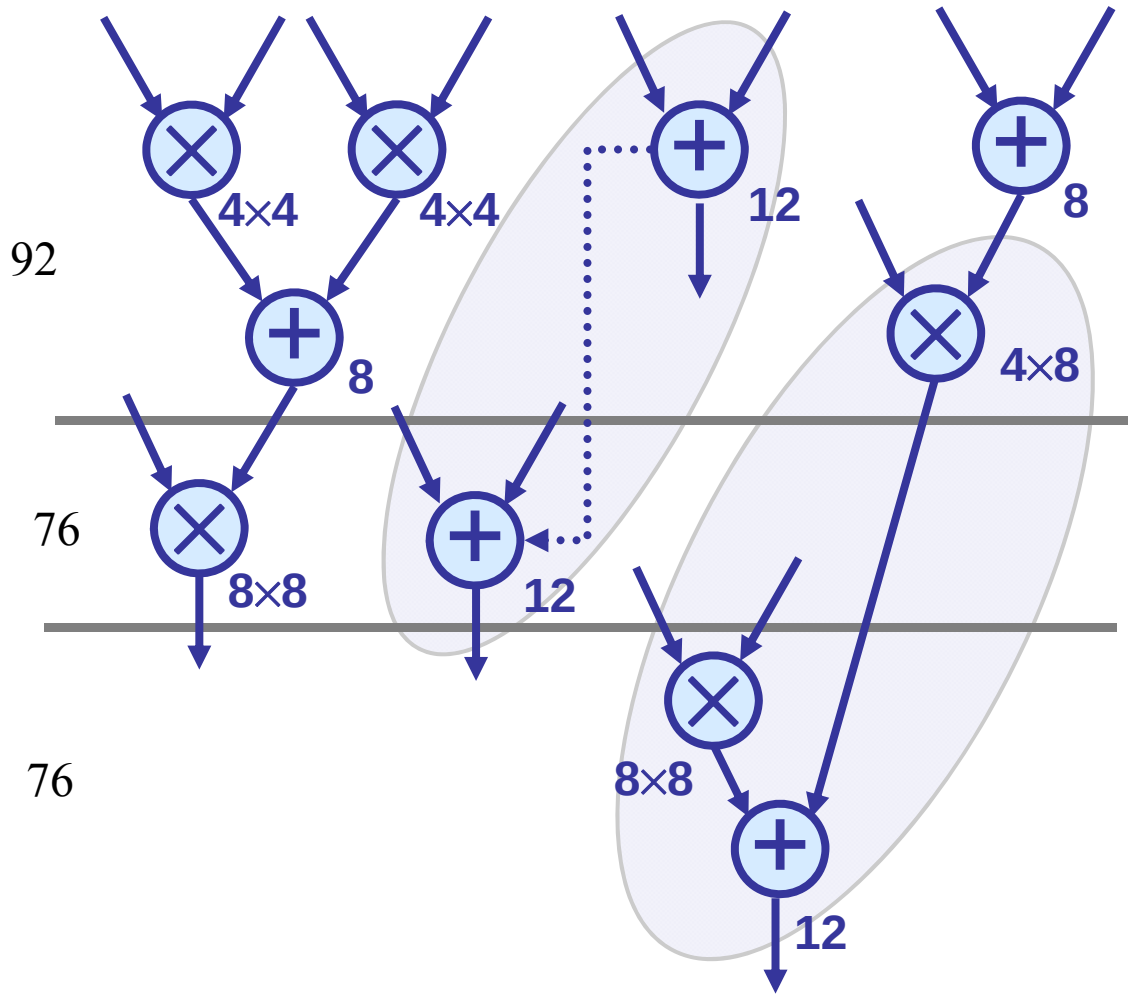


Example

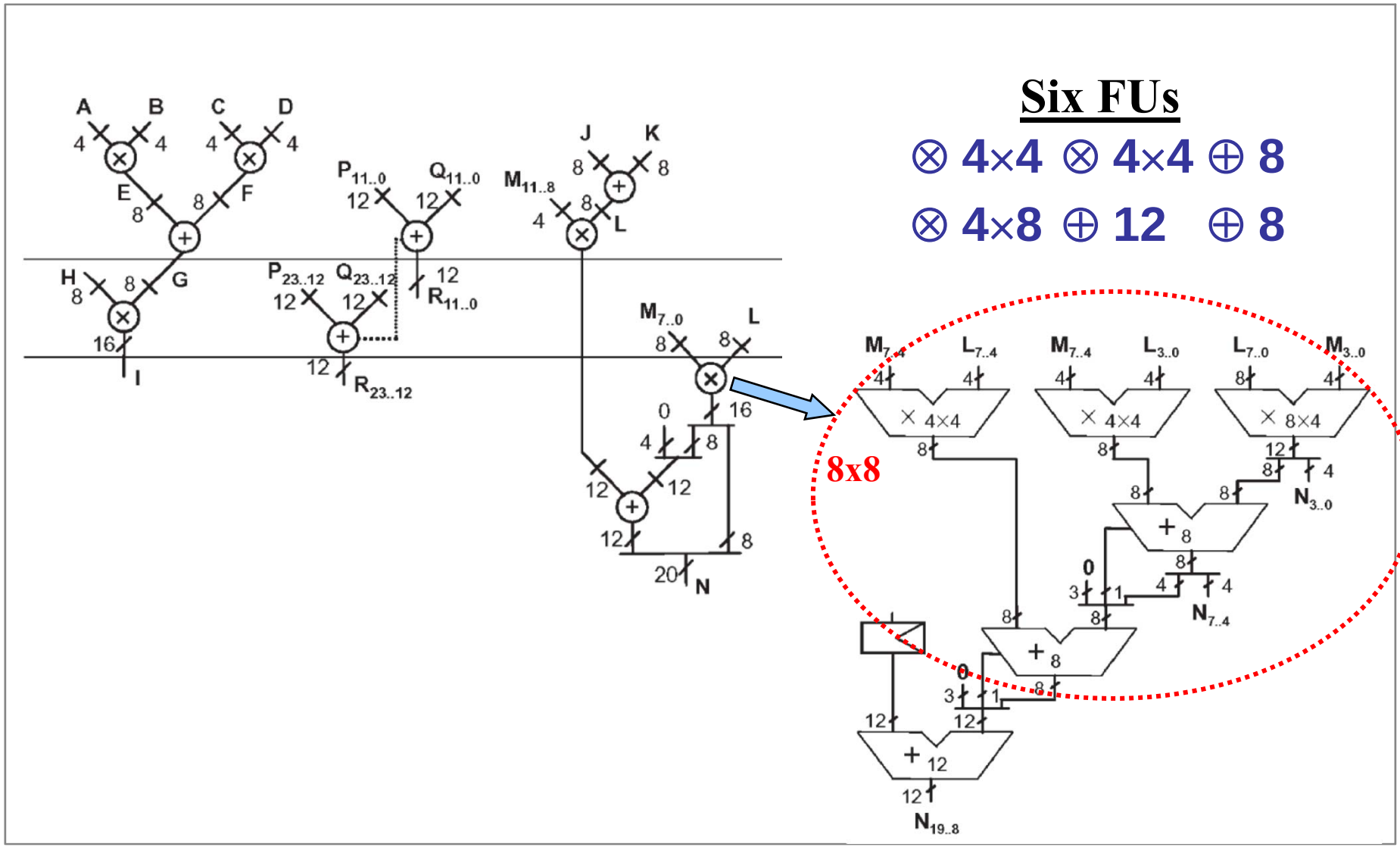
- Conventional S-A



Example: Scheduling



Example: Allocation



Example: Allocation

Operation	Fragmentation	Cycle 1	Cycle 2	Cycle 3
E = A × B		⊗4×4		
F = C × D		⊗4×4		
G = E + F		⊕ 8		
I = H × G	I1 = H × $G_{3..0}$		⊗8×4	
	I2 = $H_{3..0}$ × $G_{7..4}$		⊗4×4	
	I3 = $H_{7..4}$ × $G_{7..4}$		⊗4×4	
	I4 = $I1_{11..4}$ + I2		⊕ 8	
	I5 = ('000' & $I4_{8..4}$) + I3		⊕ 8	
	I = I5 & $I4_{3..0}$ & $I1_{3..0}$			
L = J + K		⊕ 8		
N = L × M	N1 = L × $M_{11..8}$	⊗8×4		
	N2 = L × $M_{3..0}$			⊗8×4
	N3 = $L_{3..0}$ × $M_{7..4}$			⊗4×4
	N4 = $L_{7..4}$ × $M_{7..4}$			⊗4×4
	N5 = N3 + $N2_{11..4}$			⊕ 8
	N6 = N4+ ('000' & $N5_{8..4}$)			⊕ 8
	N7 = N1+ ('000'& N6)			⊕ 12
	N = N7 & $N5_{3..0}$ & $N2_{3..0}$			
R = P + Q	R1 = $P_{11..0}$ + $Q_{11..0}$	⊕ 12		
	R2 = $P_{23..12}$ + $Q_{23..12}$		⊕ 12	
	R = R2 & R1			
		92 FA	**92 FA**	**92 FA**

No FUs waste

⊗ 4×4 ⊗ 4×4 ⊕ 8

⊗ 4×8 ⊕ 12 ⊕ 8

Example: Allocation

- FUs vs. full circuit

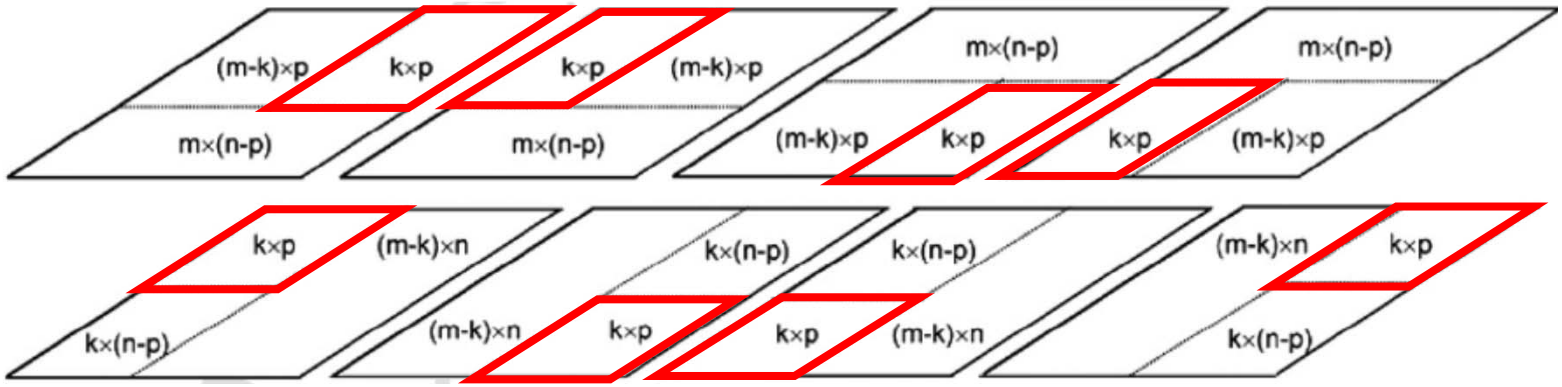
	Conventional	Proposed	Saved
Datapath FUs	⊗12×8, ⊗4×4, ⊕ 24	⊗8×4, 2 ⊗4×4, ⊕ 12, 2 ⊕ 8	—
FU's area	1401 equivalent gates	911 equivalent gates	35 %
Circuit area	3324 equivalent gates	2298 equivalent gates	30.86 %
Cycle length	23.1 ns	21.43 ns	7.22 %

Area optimization: implementation

- HLS algorithm including fragmentation during scheduling and allocation
  - Scheduling based on force-directed algorithm
    - Force considers the HW cost of every different operation or fragment
    - Priority: Favours ops with less mobility in cycles with small computational load
    - Bound: initially, the most uniform distribution. Relaxed as algorithm progresses.
  - Allocation & binding phase produces a datapath where all the FUs execute one operation of its same width in at least one cycle

Area optimization: implementation

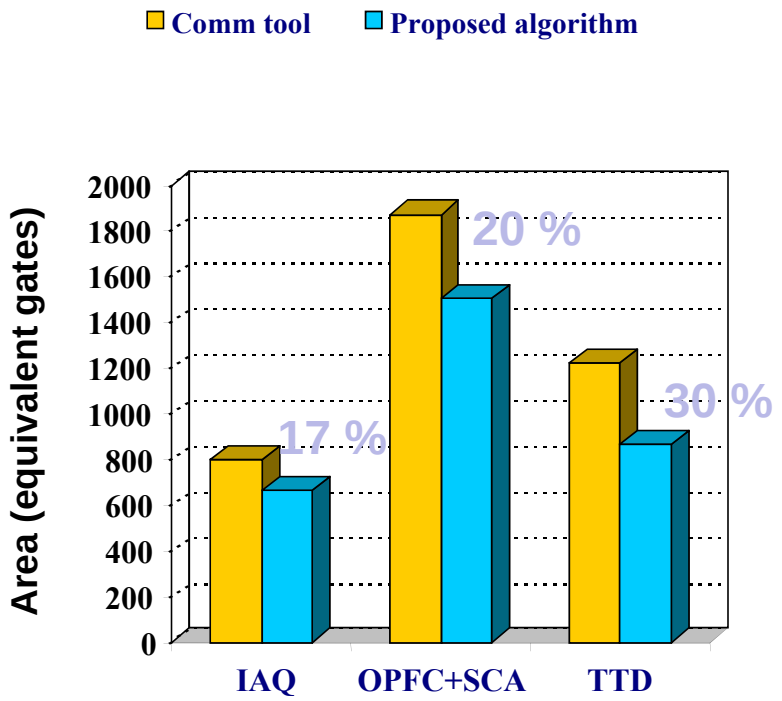
- Multiplications allocation (mxn). Iteratively:
  - 1. Allocation and binding w/o hw waste. When size mxn is repeated in every cycle. Bind 1 op per cycle.
  - 2. mxn size not exactly repeated every cycle, but waste below threshold. Bind 1 op per cycle.
  - 3. Fragmentation: Multiplication fragments explored:
    - If a smaller fragment appears in cycles where the bigger does not
    - The allocation produces two new fragments to be treated



Area optimization: experiments

- Synthesis of an ADPCM

- **IAQ**
  - 12 operations
  - 3 data representations
    - C2 (12 bits)
    - Magnitude & sign (13 bits)
    - Unsigned (4 bits)
- **OPFC+SCA**
  - 45 operations
  - 8 data representations
    - C2 (16, 15, 14, 12 bits)
    - Magnitude & sign(13 y 11 bits)
    - Unsigned (8 & 4 bits)
- **TTD**
  - 7 operations
  - 2 data representations
    - C2 (16 bits)
    - Unsigned (16 bits)



Area optimization: experiments

- Synthesis of an homogeneous specification
  - 5th order elliptic filter
    - **34 operations: 26 additions and 8 multiplications**
    - **Unsigned operations of 16 bits width**

More balanced computational load

		Comm tool	Proposed algorithm	% saved
λ=8	Area	7654	7398	3.34 %
	Clock cycle	58.63	37.27	36.43 %
λ=11	Area	7065	6438	8.87 %
	Clock cycle	51.59	41.81	18.95 %
λ=16	Area	6794	4953	27.09 %
	Clock cycle	32.27	31.13	3.53 %

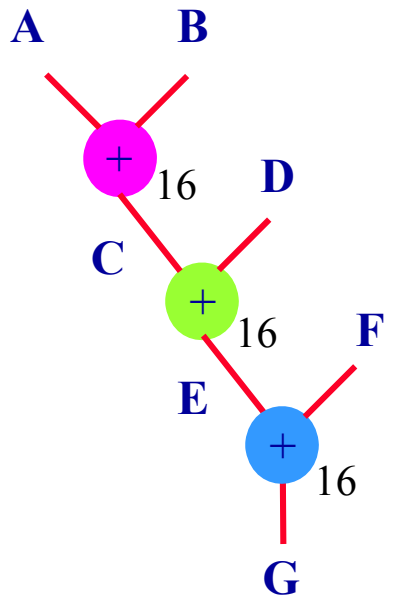
Reuse of smaller fragments

Execution time optimization

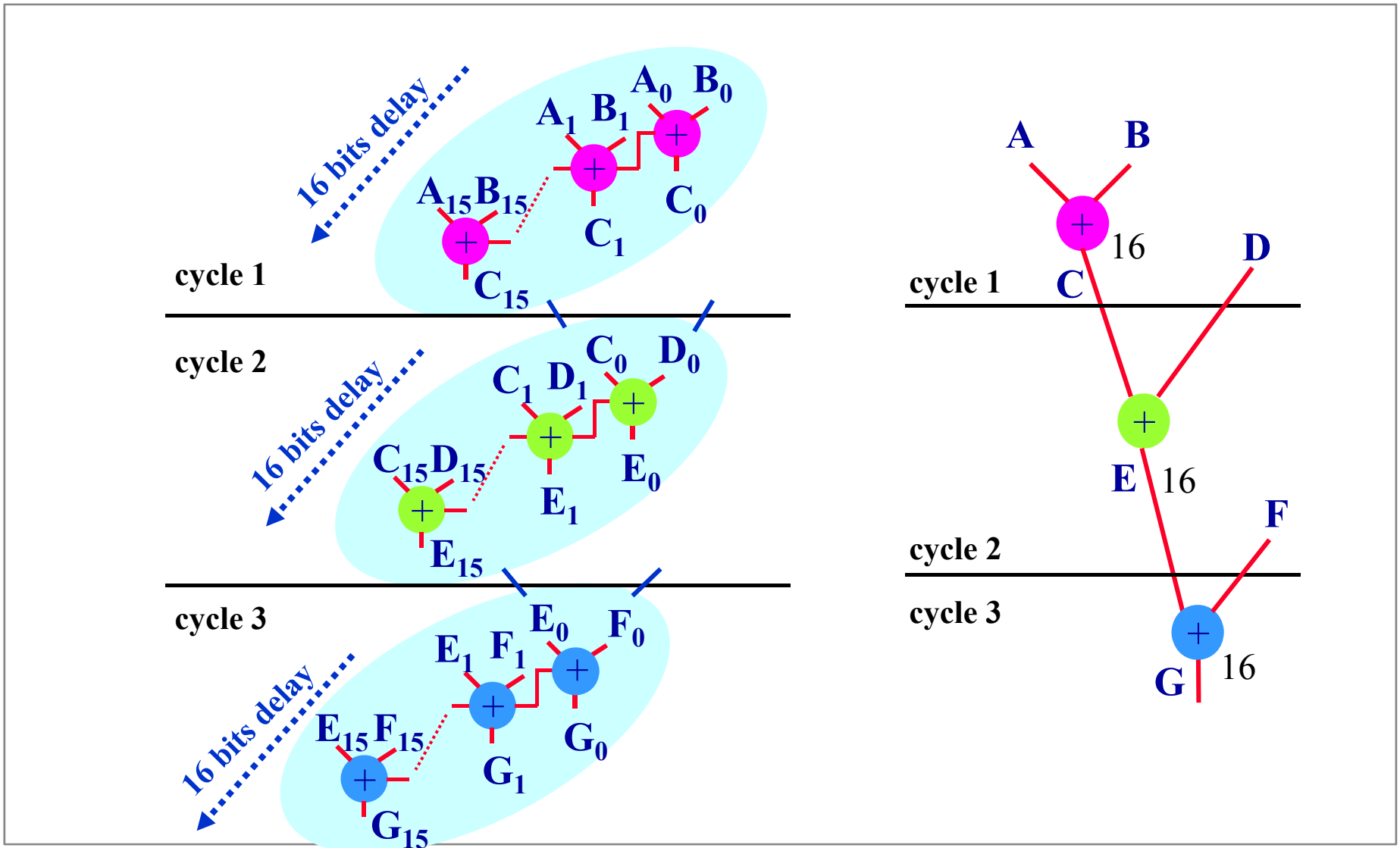
- Main ideas: similar to those applied for area reduction
- Goal:
  - Try to break long bit-level propagation chains in the different operations
- Approach: Pre-synthesis optimization
  - Transform the original spec into a new one where a HLS commercial tool can produce better results
  - Schedule fragments in different cycles
  - Preserve the benefit of using commercial tools

Example

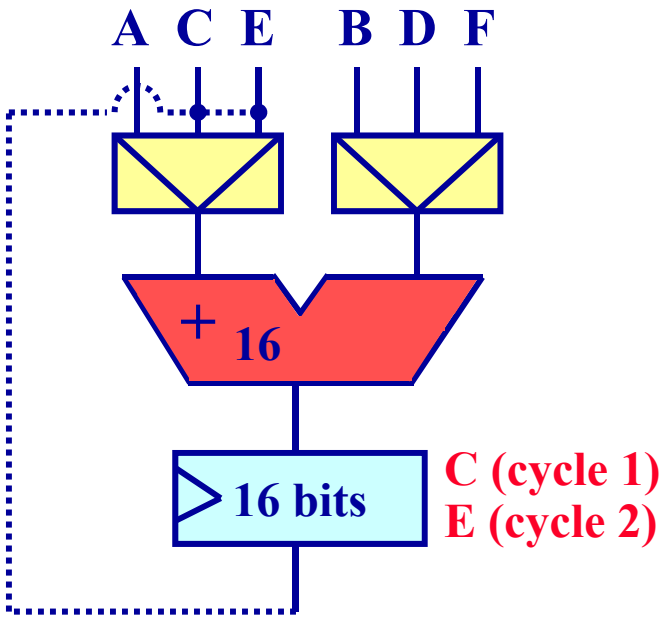
```
entity example is
port (clk: in std_logic;
    A, B, D, F: in std_logic_vector(15 downto 0);
    G: inout std_logic_vector(15 downto 0));
end example;

architecture beh1 of example is
begin
main: process
    variable C: std_logic_vector(15 downto 0);
    variable E: std_logic_vector(15 downto 0);
begin
    C := A + B;
    E := C + D;
    G <= E + F;
end process main;
end beh1;
```

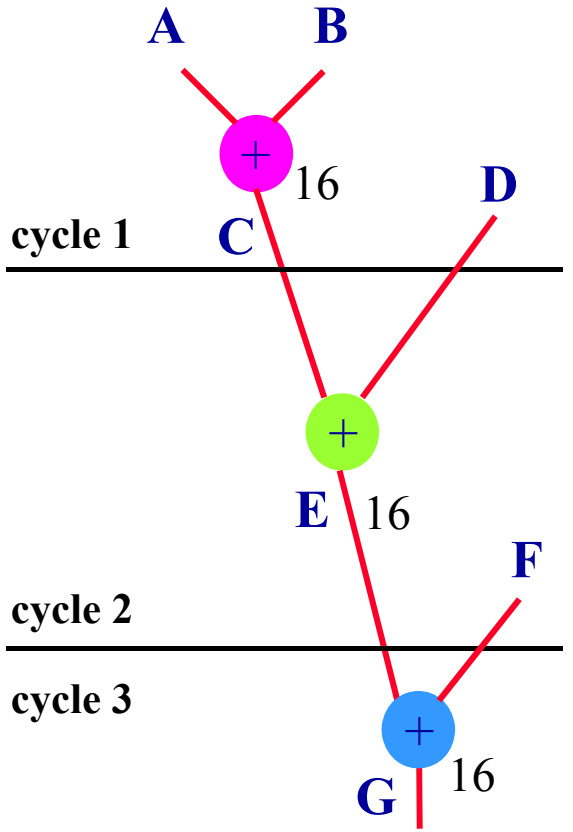


Scheduling: small area



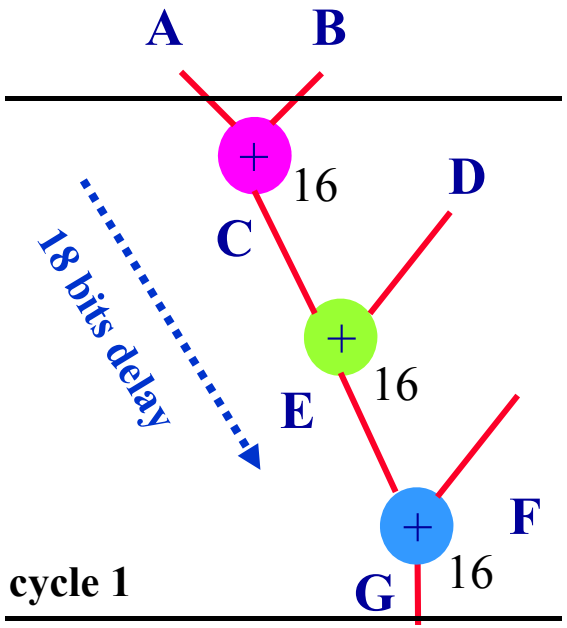
Allocation: small area

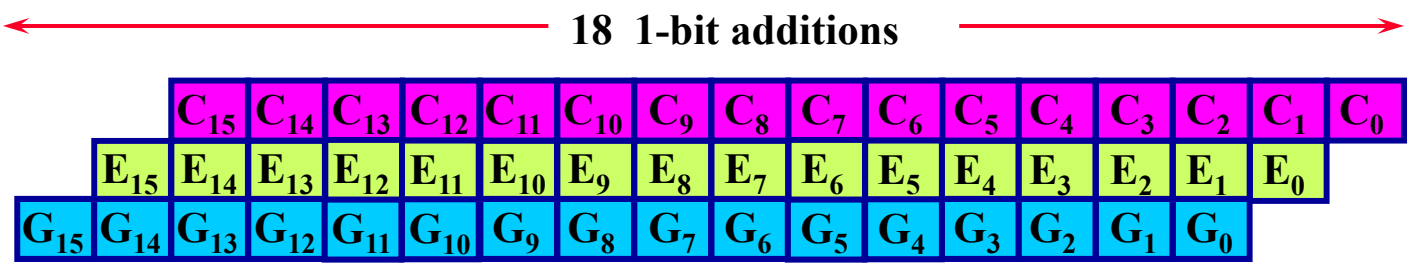




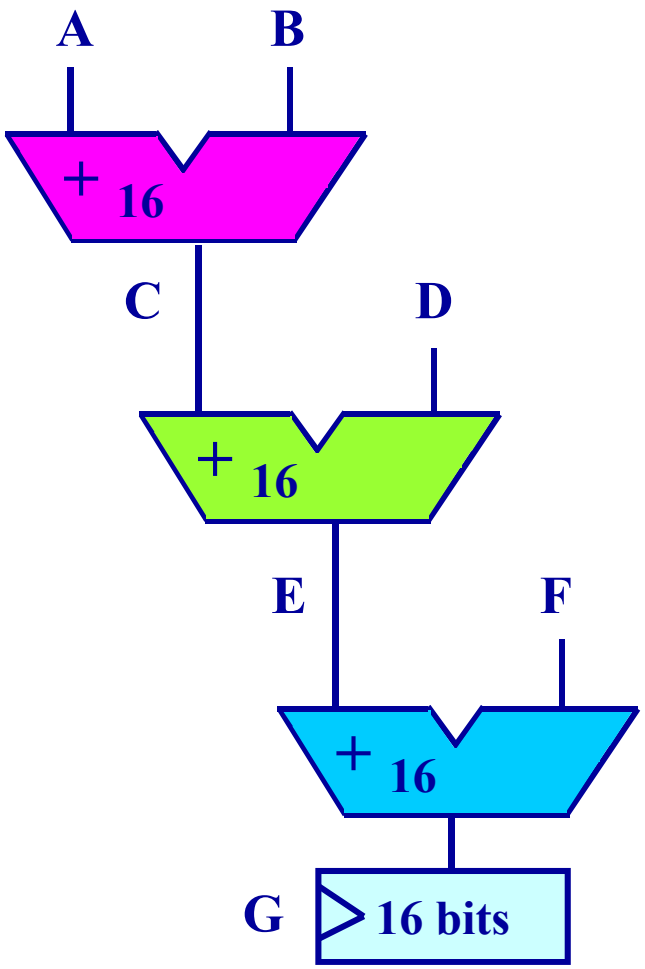
Scheduling: fast execution

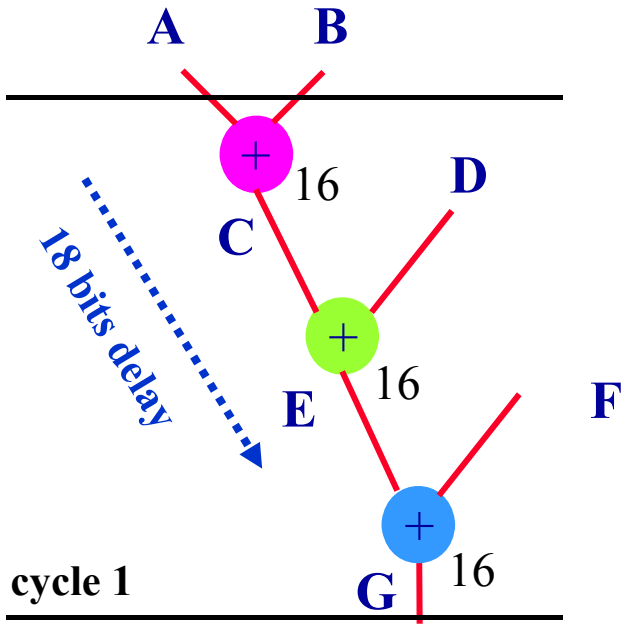
```
....
architecture beh1 of example is
begin
main: process
    variable C: std_logic_vector(15 downto 0);
    variable E: std_logic_vector(15 downto 0);
begin
    C := A + B;
    E := C + D;
    G <= E + F;
end process main;
end beh1;
```





Allocation: fast execution

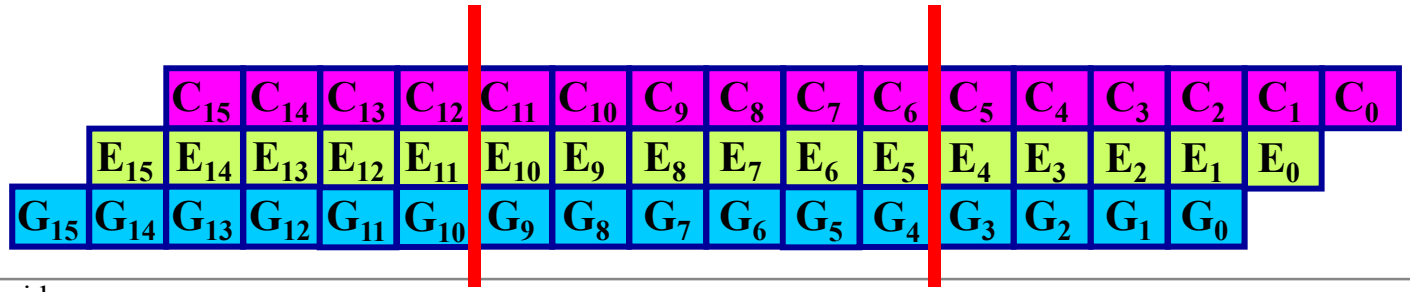




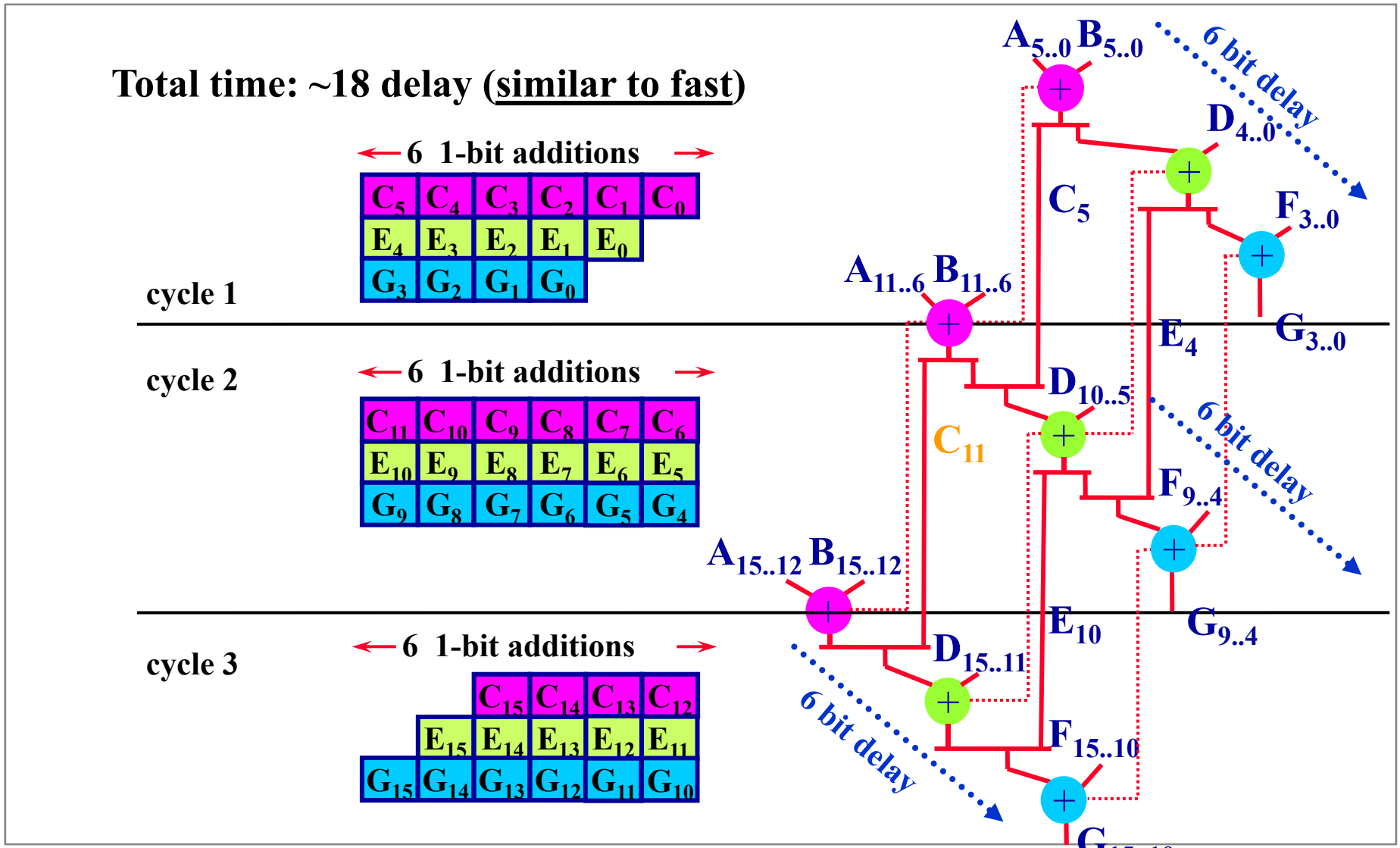
Pre-synthesis: vertical fragmentation

```
architecture beh2 of example is
.....

    C(6 downto 0) := ("0" & A(5 downto 0)) + ("0" & B(5 downto 0));
    C(12 downto 6) := ("0" & A(11 downto 6)) + ("0" & B(11 downto 6)) + C(6);
    C(15 downto 12) := A(15 downto 12)+ B(15 downto 12) + C(12);

    E(5 downto 0) := ("0" & C(4 downto 0)) + ("0" & D(4 downto 0));
    E(11 downto 5) := ("0" & C(10 downto 5)) + ("0" & D(10 downto 5)) + E(5);
    E(15 downto 11) := C(15 downto 11) + D(15 downto 11) + E(11);

    G(4 downto 0) <= ("0" & E(3 downto 0)) + ("0" & F(3 downto 0));
    G(10 downto 4) <= ("0" & E(9 downto 4)) + ( "0" & F(9 downto 4)) + G(4);
    G(15 downto 10) <= (E(15 downto 10)) + (F(15 downto 10)) + G(10);
.....
end beh2;
```

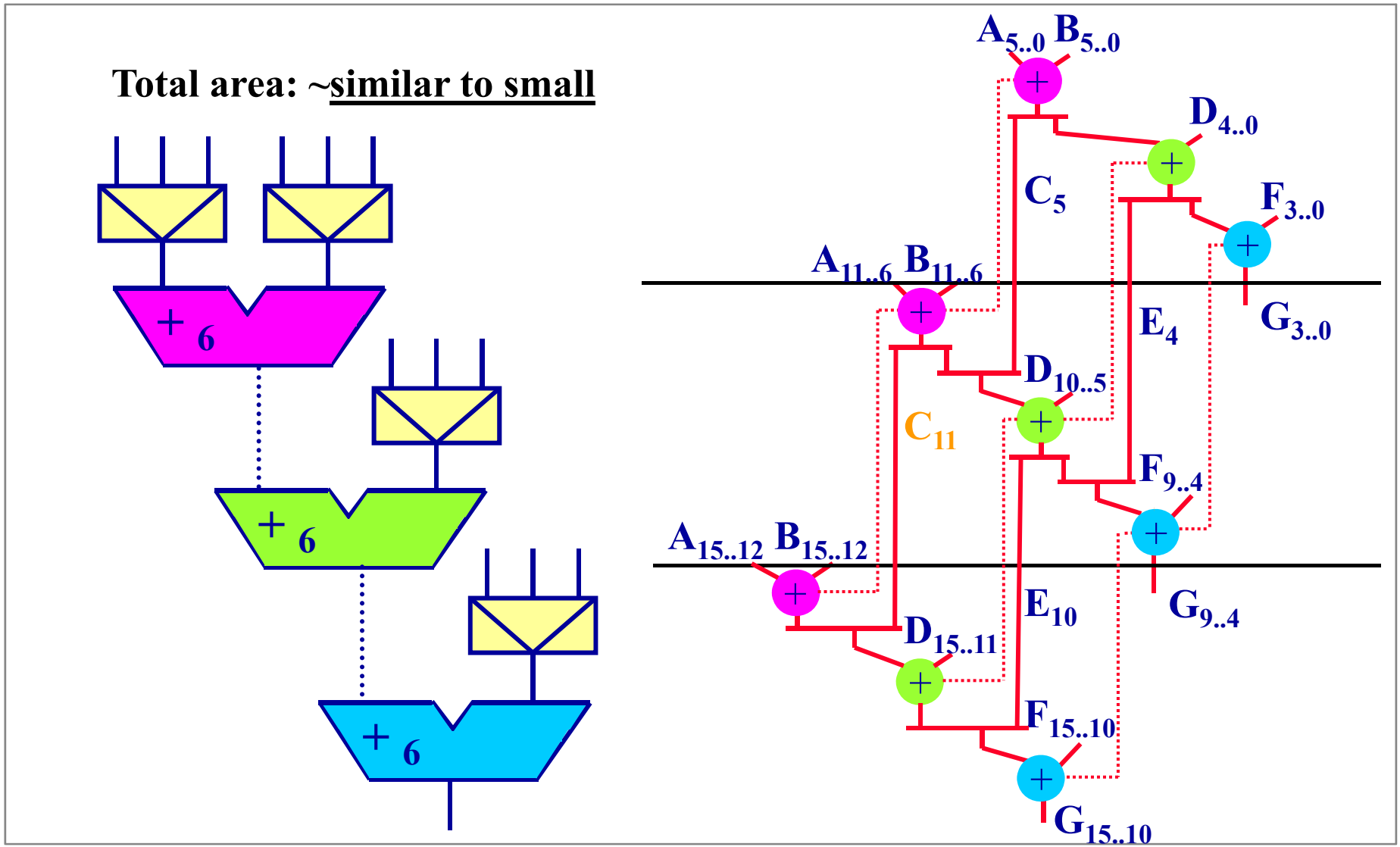
9 operations
Max width = 6 bits



Alternative scheduling

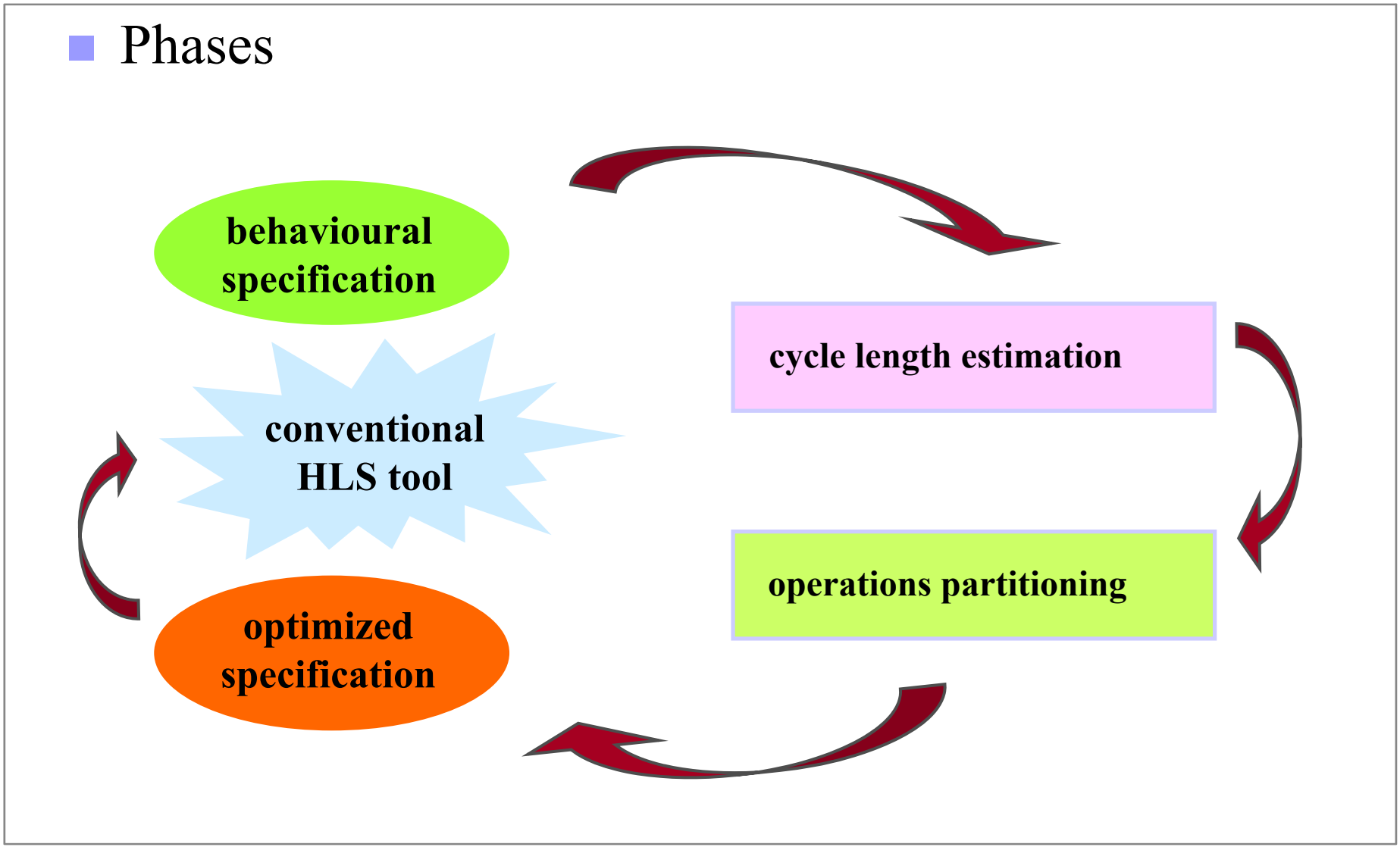


Alternative allocation



Method

- Phases



Cycle length estimation

- Identification of the critical path
  - Calculation of every path execution time
  - Operations are crossed from output to inputs
- Estimation of clock cycle length

$$\text{CycleLength} = \left\lceil \frac{\text{ExecTime(Critical Path)}}{\text{Latency}} \right\rceil$$

Partitioning

- Operations with execution times longer than CL
  - Minimum number of fragments
  - Fragments of similar size → HW reuse
  - Compute ASAP/ALAP of every fragment
    - Fragments with identical ASAP/ALAP already scheduled
    - Fragments with different ASAP/ALAP to be scheduled by the HLS tool
  - Multiplications
    - Can be treated as rows of additions (ICCAD 2005)
    - Specific HW modules for fragmentation of multiplications (DATE 2006, TCAD 2007)
      - Better in terms of area. Similar in terms of time.

Some results[1]

- Classical HLS benchmarks
  - Performance improvement 67 % on average
  - Area increment 7 % on average
- ADPCM decoder
  - Original: Initial description + HLS tool + RTL tool
  - Optimized: transformed description + HLS tool + RTL tool
  - Identical tool in both cases

ADPCM module	λ	Cycle duration (ns)			Area decrement
		Original	Optimized	Saved	
IAQ	3	6.96	2.4	66 %	2.4 %
TTD	5	9.28	3.66	61 %	6.2 %
OPFC + SCA	12	9.39	2.36	75 %	3.3 %

1. R. Ruiz-Sautua et al., Behavioural transformations to improve circuit performance in HLS, DATE 2005

To probe further …

- Fragmentation and bit-lvel HLS
  - Performance-driven Read-After-Write Dependencies Softening in High-Level Synthesis, **ICCAD** 2005
  - Bitwise Scheduling to Balance the Computational Cost of Behavioural Specifications, **IEEE Trans. on CAD**, 2006
  - Pre-synthesis Optimization of Multiplications to Improve Circuit Performance, **DATE** 2006
  - Exploiting Bit-Level Delay Calculations to Soften Read-After-Write Dependences in Behavioral Synthesis, **IEEE Trans. on CAD**, 2007
  - Area optimization of multi-cycle operators in high-level synthesis, **DATE** 2007
  - Frequent-pattern Guided Multi-level Decomposition of Behavioural Specifications, **IEEE Trans. on CAD**, 2009
  - Subword Switching Activity Minimization to Optimize Dynamic Power Consumption, **IEEE D&T of Computers**, 2009
  - Power Optimization in Heterogenous Datapaths, **DATE** 2011
- Exploiting speculative FUs in HLS
  - Applying branch prediction techniques to implement functional units, **ICCD** 2008
  - Using Speculative Functional Units in high level synthesis, **DATE** 2010
  - A Distributed Controller for Managing Speculative Functional Units in High Level Synthesis, **IEEE Trans. on CAD** , 2011