TU WIEN Informatics

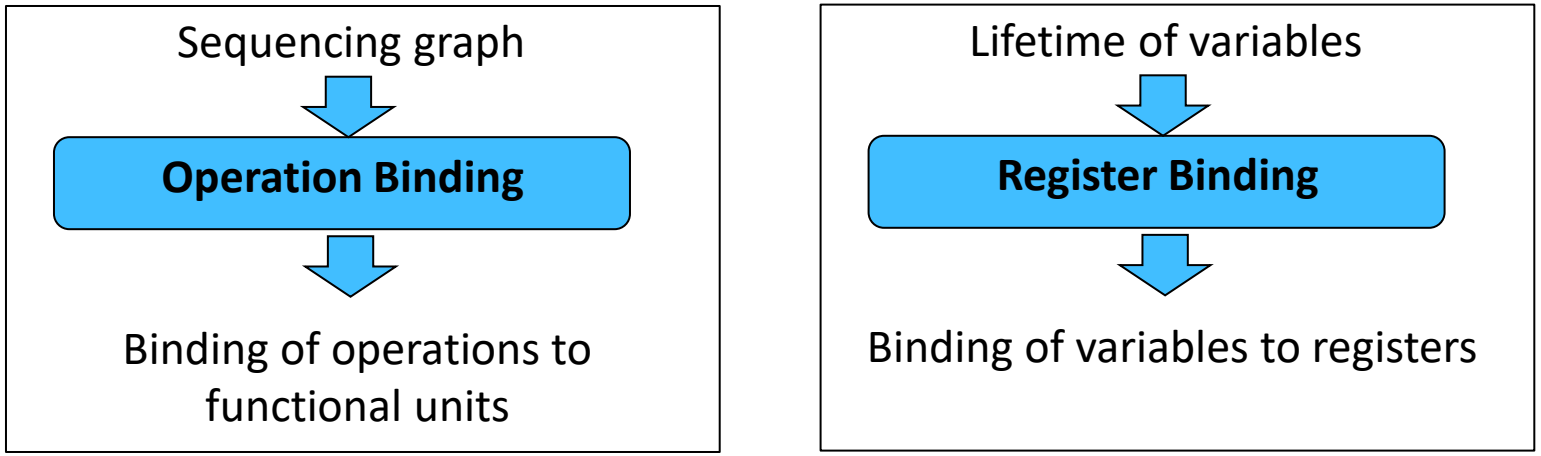
Advanced Computer Architecture

D3 –High Level Synthesis (HLS) – Binding and HW Generation

Daniel Mueller-Gritschneder

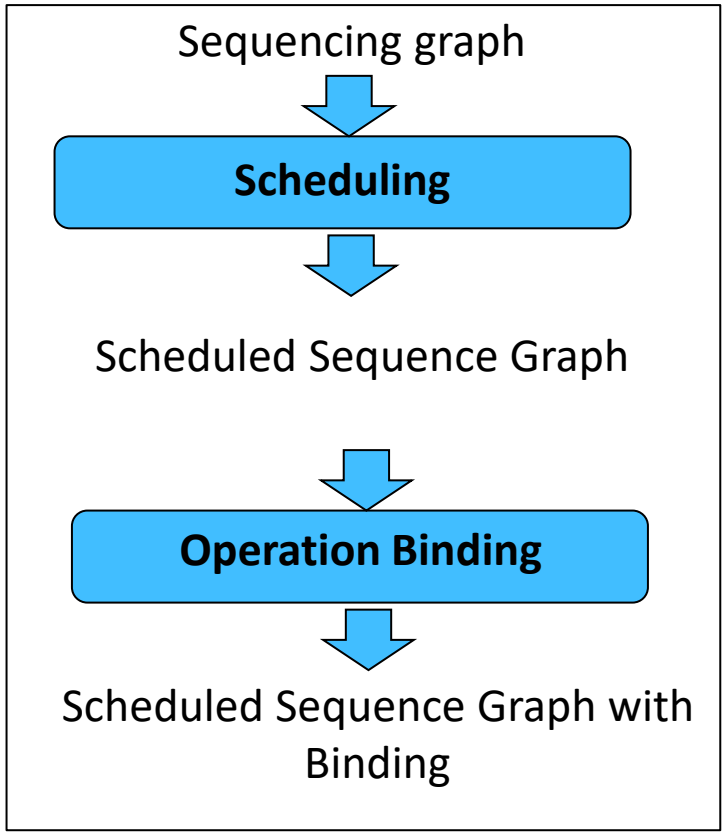
D3-1 The Binding Task

Binding Tasks



- **Goal**: Save resources by sharing of functional units and registers.

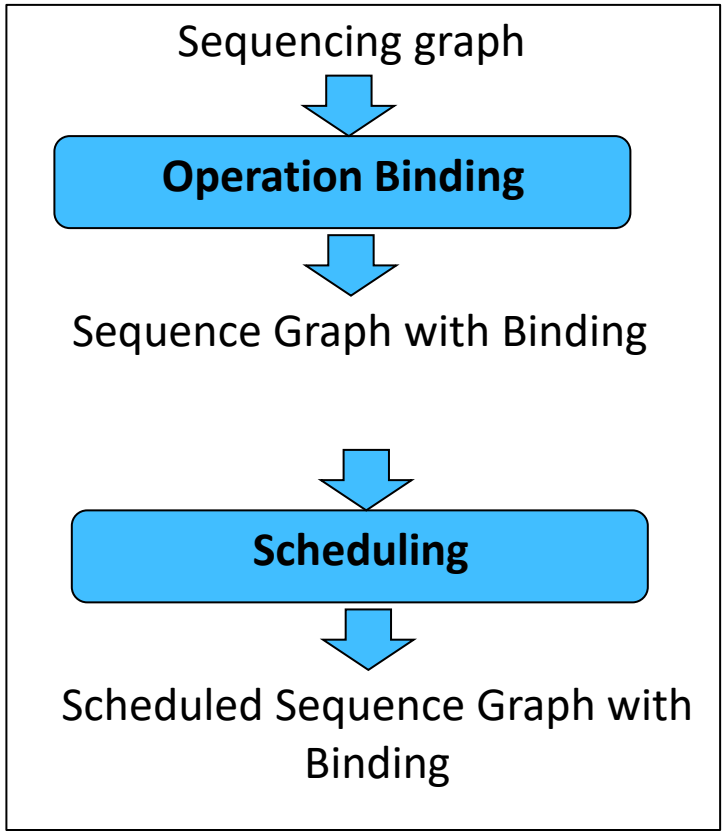
Binding after Scheduling



Concurrent operations can be scheduled to be executed in parallel.

Scheduling restricts binding:
Concurrent operations scheduled to be executed in parallel can not be bound to the same functional unit.

Binding before Scheduling



Concurrent operations can be bound to the same functional unit.

Binding restricts scheduling:
Concurrent operations bound to the same functional unit can not be executed in parallel (in same clock cycle).

D3-2 Graph Coloring

Graph Theory – Cover and Partition

- The **cover** of a set S, is a set of subsets of S such that their union is equal to S.
- **The partition** of S is a cover of S, such that all subsets in the cover are disjoint.

Graph Theory - Cliques

- A **Clique** of an undirected graph $G(V, E)$ is a subset of the nodes V, in which all nodes are fully connected between each other.

- A **Clique-Cover** of an undirected graph $G(V, E)$s a set of cliques of the graph, such that their union is equal to V.

- A **Clique-Partition** is a disjoint clique-cover**.**

Graph Theory – Graph Coloring

- A coloring of an undirected graph $G(V, E)$ is a division of the nodes V into subsets such that:
  - Each node is assigned a color.
  - The two end nodes of each edge must have different colors.

- The minimal number of colors that allows a coloring of $G(V, E)$ is called the chromatic number $\chi(G)$.

Graph Theory – Complementary Graph

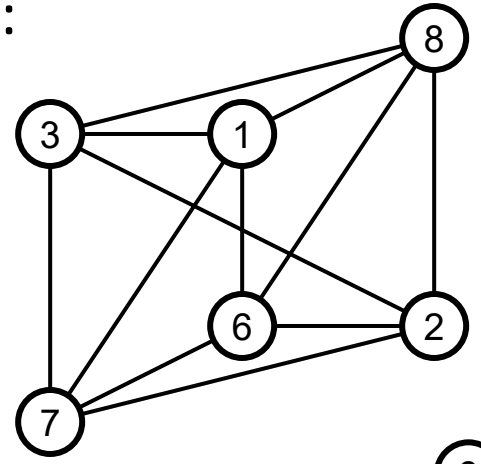
- In the complementary graph $G(V, V \times V \backslash E)$, the nodes
  - that are connected with an edge in the original graph $G(V, E)$, are not connected.
  - that are not connected with an edge in the original graph $G(V, E)$, are connected.

- A clique-partitioning of the undirected graph $G(V, E)$ can be found by a coloring of its complementary graph $G(V, V \times V \backslash E)$

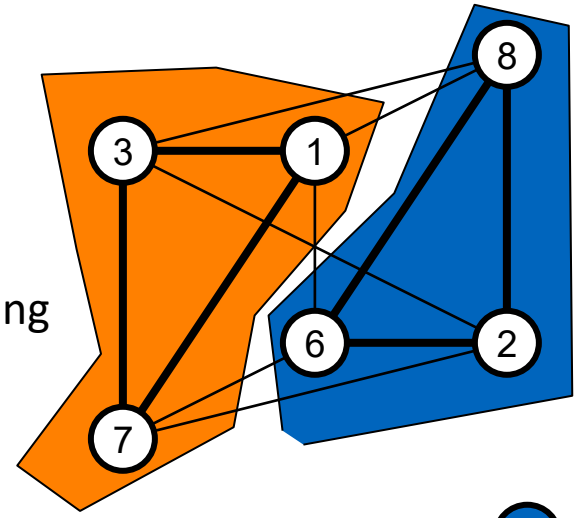
Graph Theory – Example Graph Coloring

- Example:

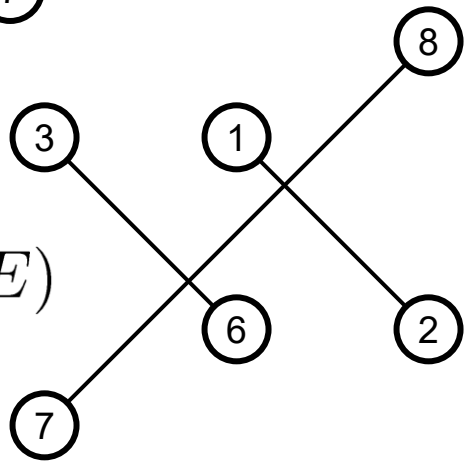
$G(V, E)$



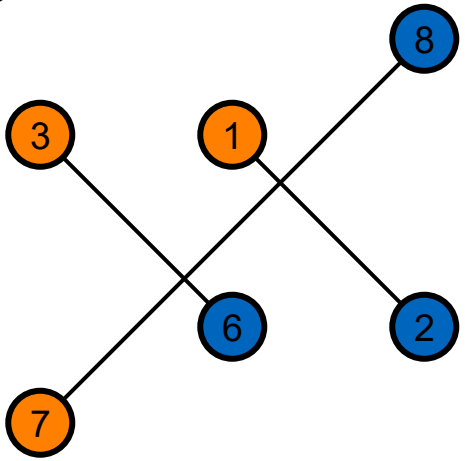
Clique-Partitioning



$G(V, V \times V \backslash E)$



Coloring



Graph Theory – Interval Graphs

- **Interval Graph**: $G_I(V_I, E_I)$
  - Set of nodes describes intervals in space or time:

$$V_I = \{I_1, I_2, ..., I_S\} \quad \text{with } I_s = [l_s \; r_s]$$

  - Edges between nodes exist, if their intervals overlap.

D3-3 Operation Binding

Func. Unit Types and Operation Types

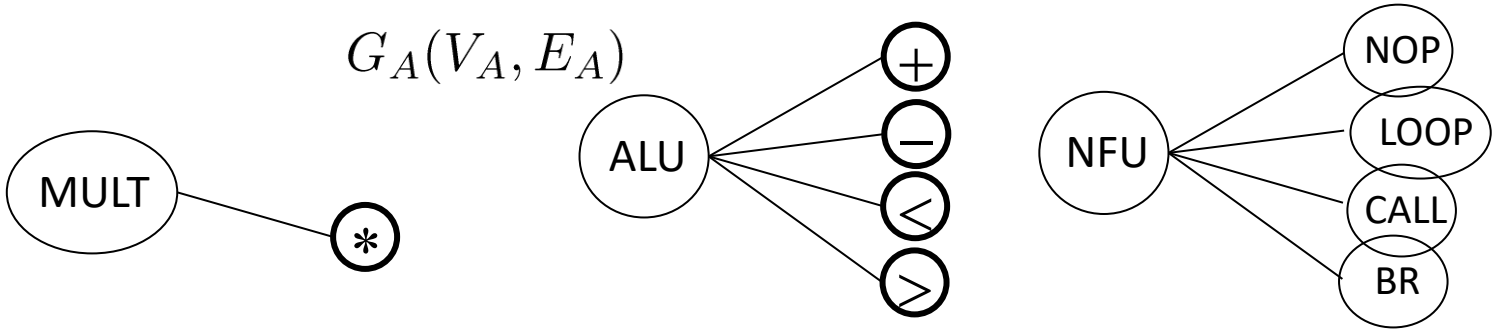
- Type of functional units: $k_r \in K$
- Set of func. unit types: $K = \{\mathrm{ALU}, \mathrm{MULT}, ...\}$
- Index of functional unit: $z_r = 1, 2, ...$
- A functional unit is defined by its type and index, e.g. (*ALU,1*)

- Operation: $O = \{\mathrm{NOP}, +, -, >, <, *, ...\}$
- Set of operation types: v_i
- Operation type: $o(v_i) \in O$

Executability

- Operation is executable on functional units, if the operation is supported by the functional unit.
- Cover: (NFU for auxiliary nodes in sequence graph with no HW)
- Bipartite graph:

k	+	-	>	<	*	NOP	LOOP	BR	CALL
ALU	x	x	x	x					
MULT					x				
NFU						x	x	x	x



Operation-Compatibility-Graph

- Operations are compatible,
  - if they can be executed on the same functional unit and
  - if either one operation always starts after the other has finished
  - or if they are on alternative paths in the control flow, such that only one of them is executed.

- Operation-compatibility-graph: $G_K^+(V, E^+)$
  - Set of nodes are the operations. $V = \{v_i : i = 1, 2, ...n\}$
  - Edges connect compatible operations

$$E^+ = \{(v_i, v_j) : i, j = 1, 2, ..., n\}$$

  - Cliques in the compatibility-graph are sets of operations that can be bound to the same functional unit.

Operation-Conflict-Graph

- Operations are incompatible,
  - if they must be executed on different functional units.
  - if they are executed in parallel due to given schedule.

- Operation-conflict-graph: $G_K^-(V, E^-)$
  - Nodes are the operations: $V = \{v_i : i = 1, 2, ...n\}$
  - Edges connect nodes of incompatible operations:
    $$E^- = \{(v_i, v_j) : i, j = 1, 2, ..., n\}$$
  - Interval graph can be seen as a special case of a conflict graph, in which overlaps of intervals produce conflicts.
  - Conflict-graph and compatibility-graph are complementary.

Operation Binding

- Problem formulation:

Mapping:

$$\beta : V \rightarrow K \times 1, 2, 3, ...$$

such that

$$\beta(v_i) = (k_r, z_r) \quad \text{mit } i = 1, ..., n$$

and

$$(k_r, o(v_i)) \in E_A$$

$$\beta(v_i) \neq \beta(v_j) \quad \forall_{i,j} : (v_i, v_j) \in E^-$$

Operation Binding

- Each operation is bound to one functional unit, which is identified by resource type index and instance index k_r:

- The operation must be executable on the functional unit: z_r

$$\beta(v_i) = (k_r, z_r) \quad \text{mit } i = 1, ...n$$

- Two operations that are bound to same functional unit must be compatible:

$$(k_r, o(v_i)) \in E_A$$

$$\beta(v_i) \neq \beta(v_j) \quad \forall_{i,j} : (v_i, v_j) \in E^-$$

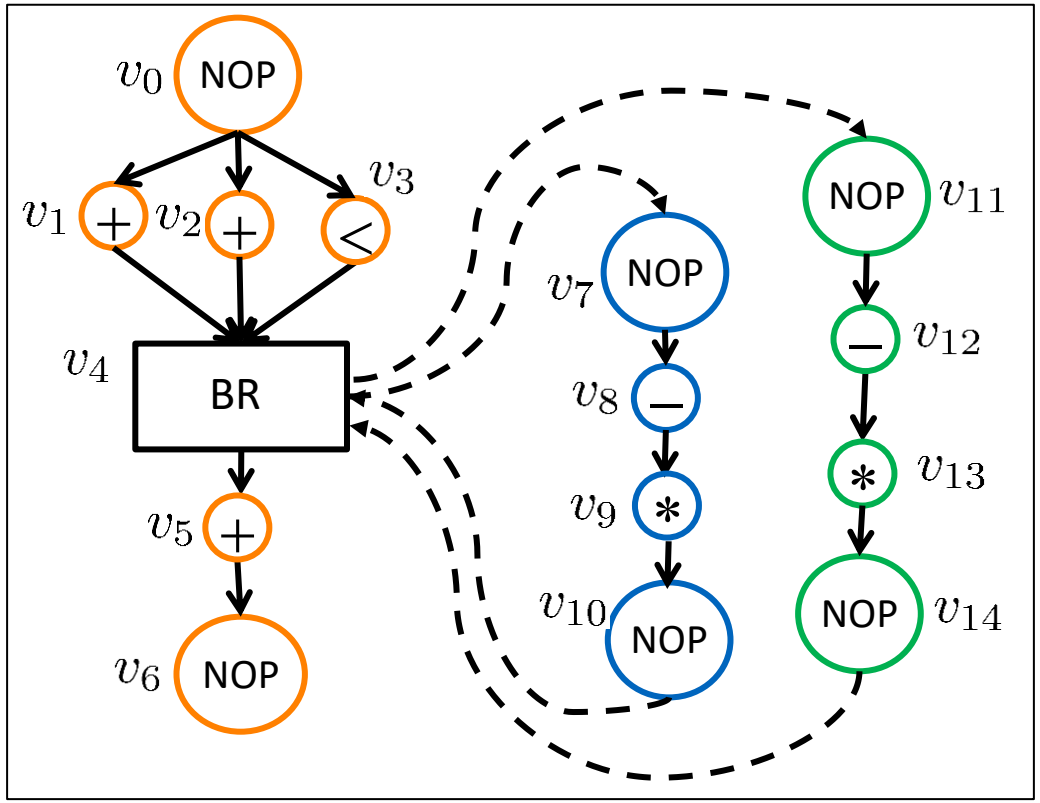
Operation Binding

- Example: Condition

Program part:

```
B1: t1=a+b
    t2=c+d
    if e>f got B2
    t3=t1-e
    g=t2*t3
    got B3
B2: t4=t1-f
    g=t2*t4
B3: z=z+g
```

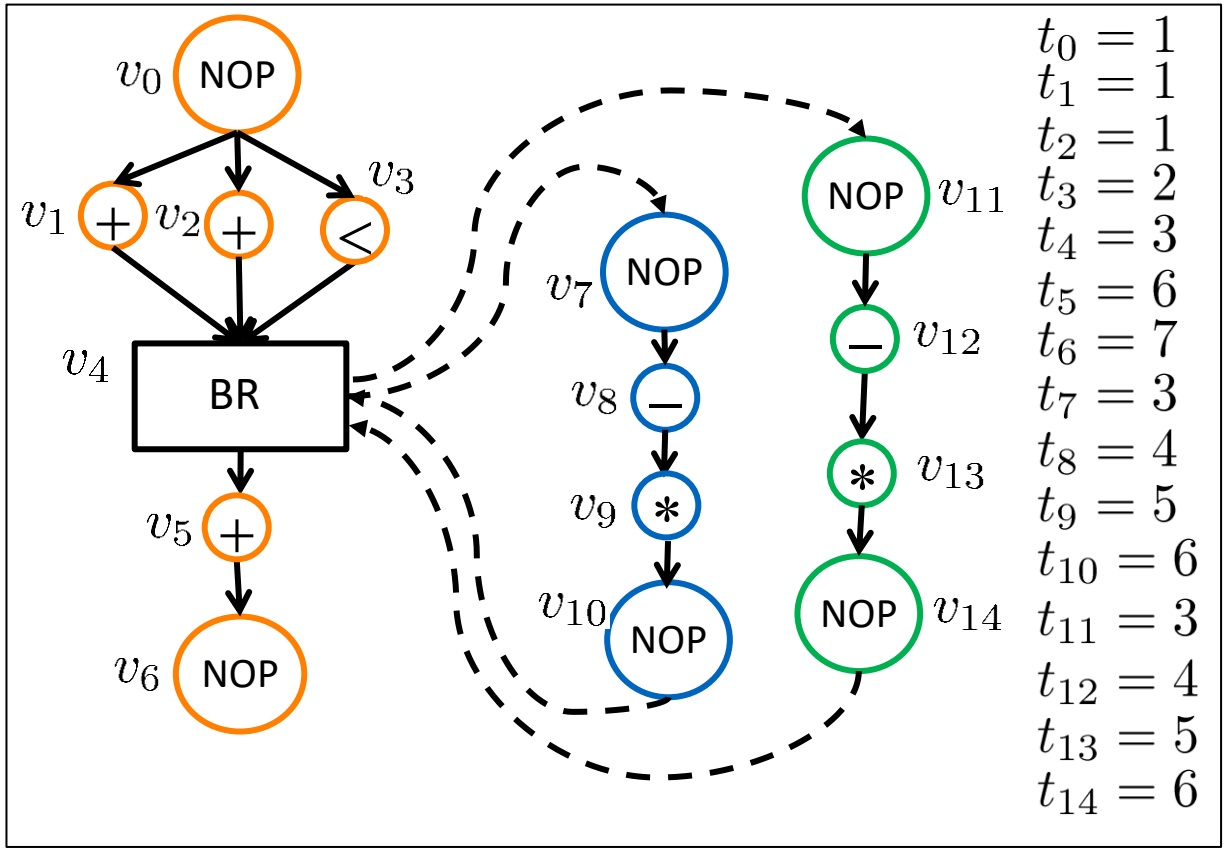
Sequence graph:



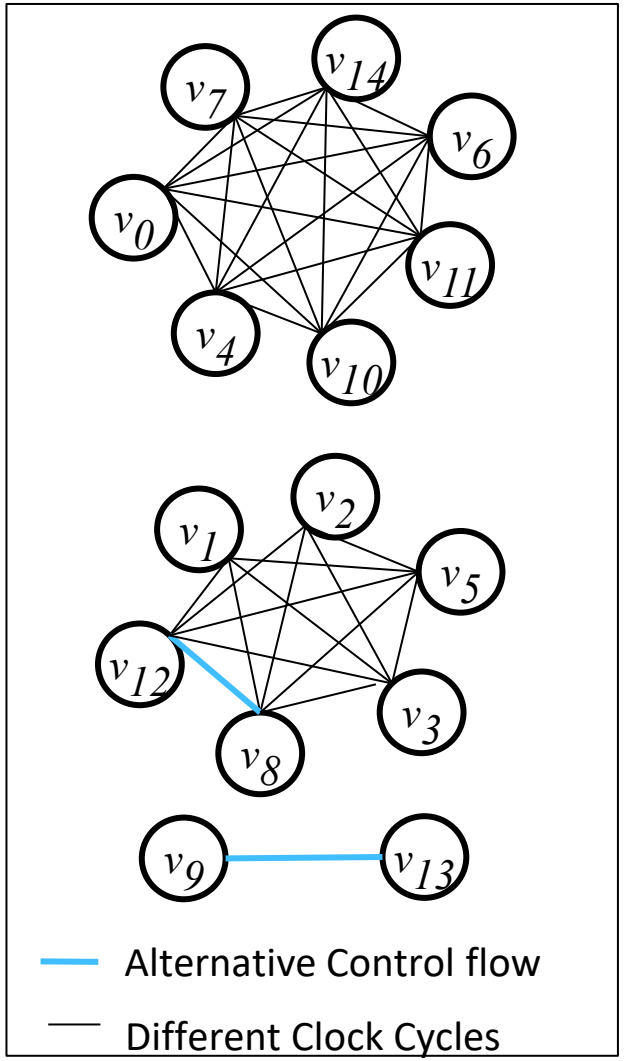
Operation Binding

- Example: Condition

Sequencing graph with schedule:



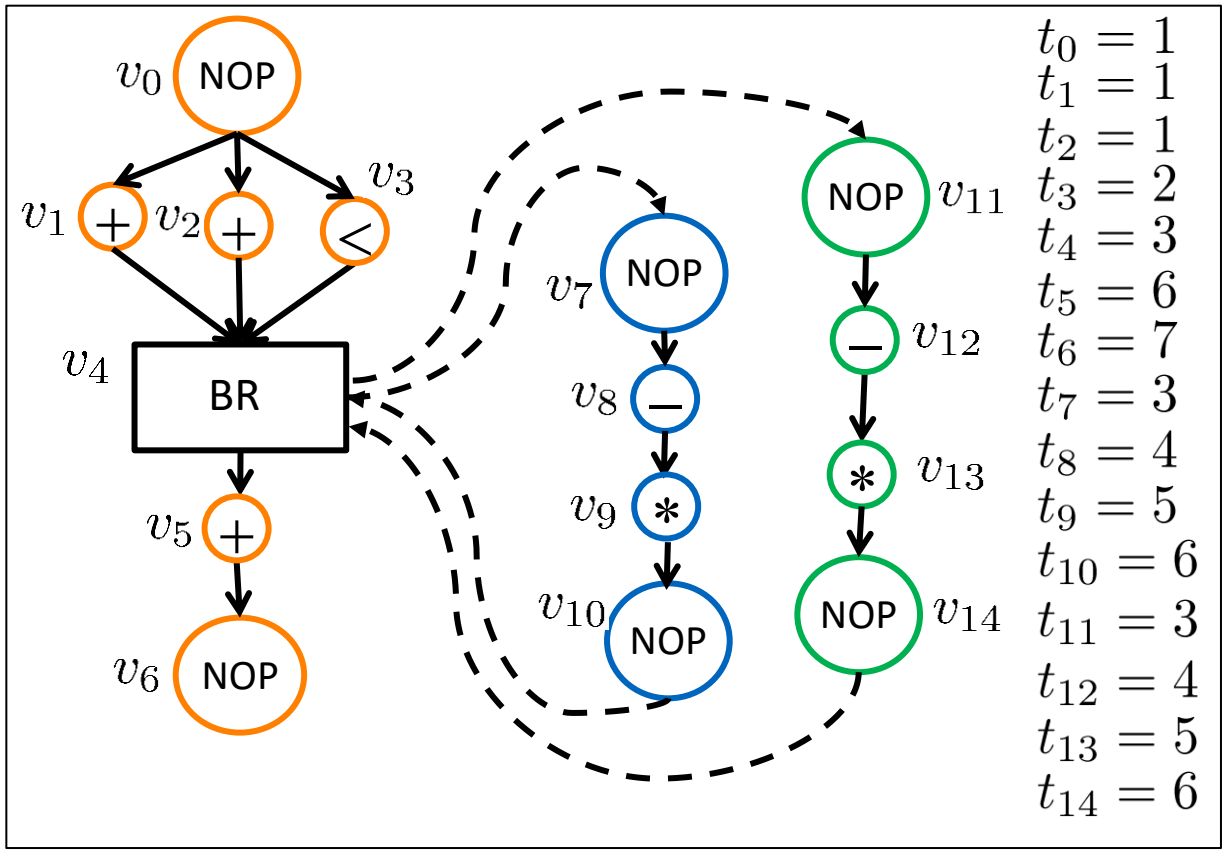
Compatibility-Graph:



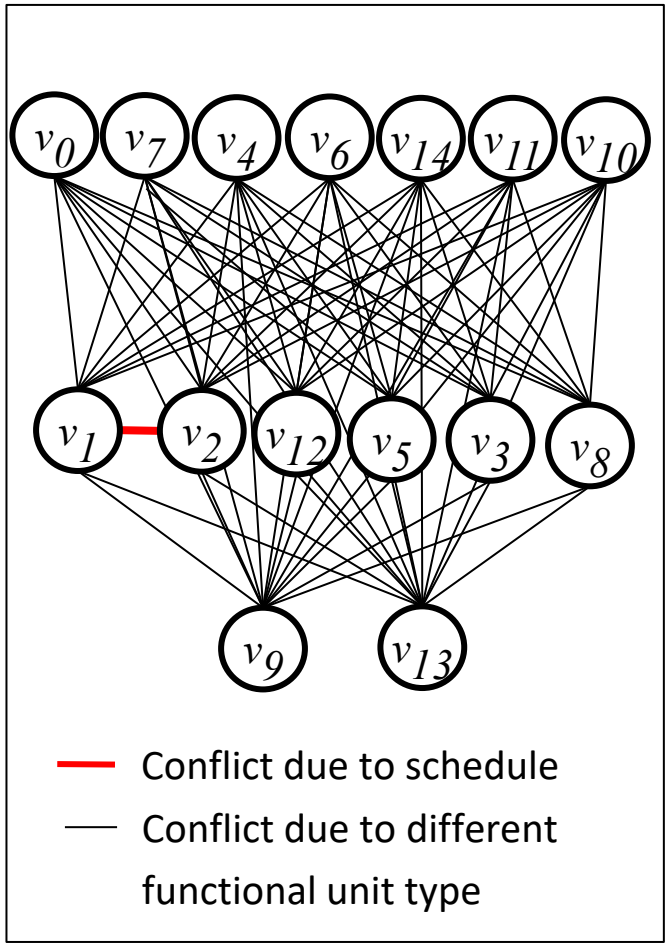
Operation Binding

- Example: Condition

Sequencing graph with schedule:

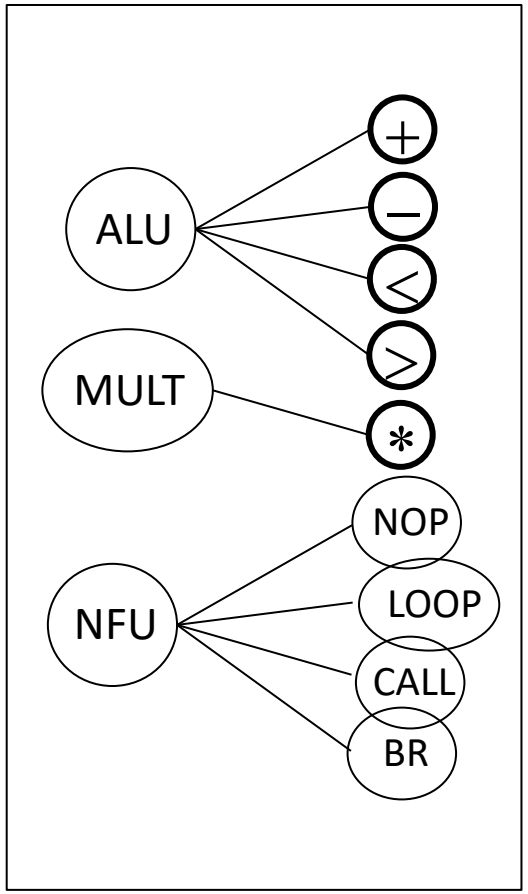


Conflict-Graph:

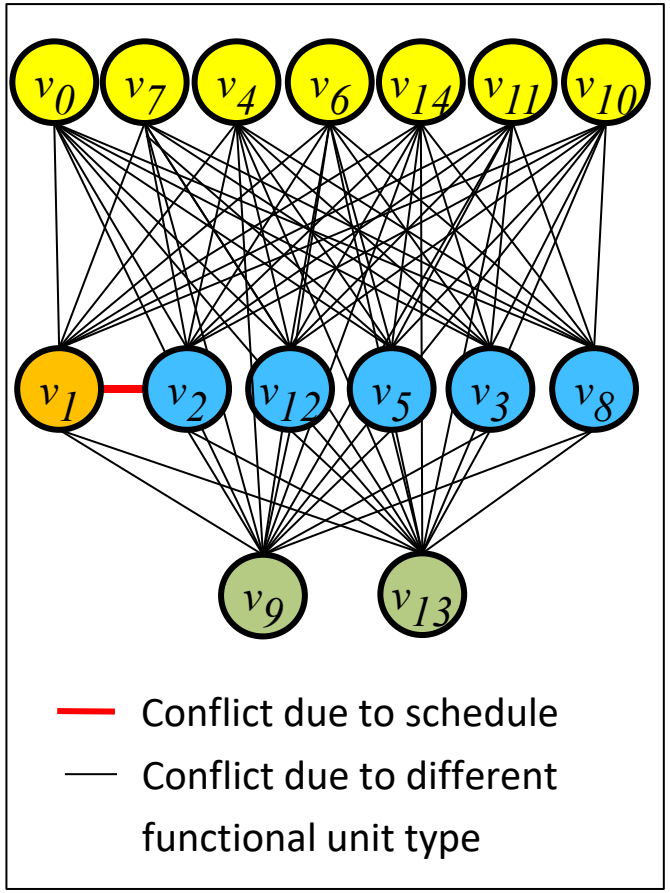


Operation Binding

- Example: Condition



Conflict-Graph with Coloring:



Binding :

$(k_1, z_1) = (NFU, 1)$

$(k_2, z_2) = (ALU, 1)$

$(k_3, z_3) = (ALU, 2)$

$(k_4, z_4) = (MULT, 1)$

Left-Edge-Algorithm

- Algorithm to color an interval graph with minimum number of colors
- Worst-case complexity: $O(|V| \cdot \log(|V|))$
- Published by Hashimoto & Stevens, DAC 1971

Left-Edge-Algorithm

Algorithm:

```
LeftEdge(G_I(V_I,E_I)) {
     Sort intervals I[i]=[l[i] r[i]] in list L by increasing l[i]
     Set color number c=0;
     repeat {
       Goto start of List L
       Set S={}
       Set r_act=0;
       repeat {
              Select next interval I[s]=[l[s] r[s]] from List L
              if (l[s] >= r_act) {
                       Insert I[s] in set S
                       Set r_act=r[s]
                       Delete I[s] from List L
              }
        until (End of List L reached)
        Assign color c to all intervals in set S
        Select next color number: c=c+1;
     }
     until (List L is empty)
}
```

Operation Binding with Left-Edge Algorithm

- Input:
  - Given Schedule
  - Interval graph of the execution times for the operations.
  - Operation-conflict graph for each functional unit type.

- Left-Edge Algorithm is applied for each functional unit type with a disjoint set of colors.

- Output: Binding of operations to the functional units.

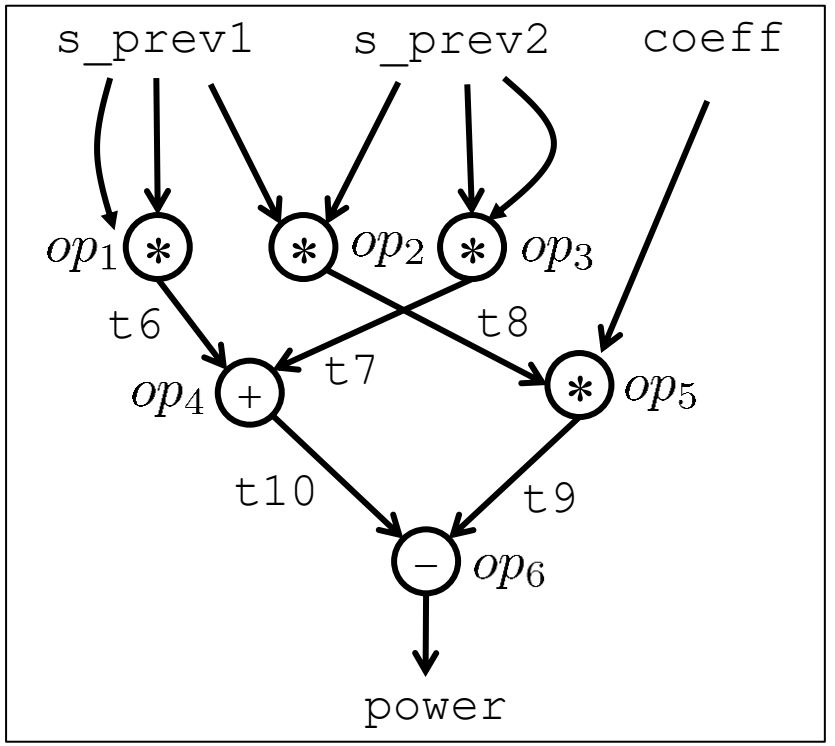
Operation Binding with Left-Edge Algorithm

- Example 1: Basic block B3 of the Goertzel algorithm:

```
return (s_prev1 * s_prev1) + (s_prev2 * s_prev2) - (s_prev1 * s_prev2 * coeff);
```

Three address code:

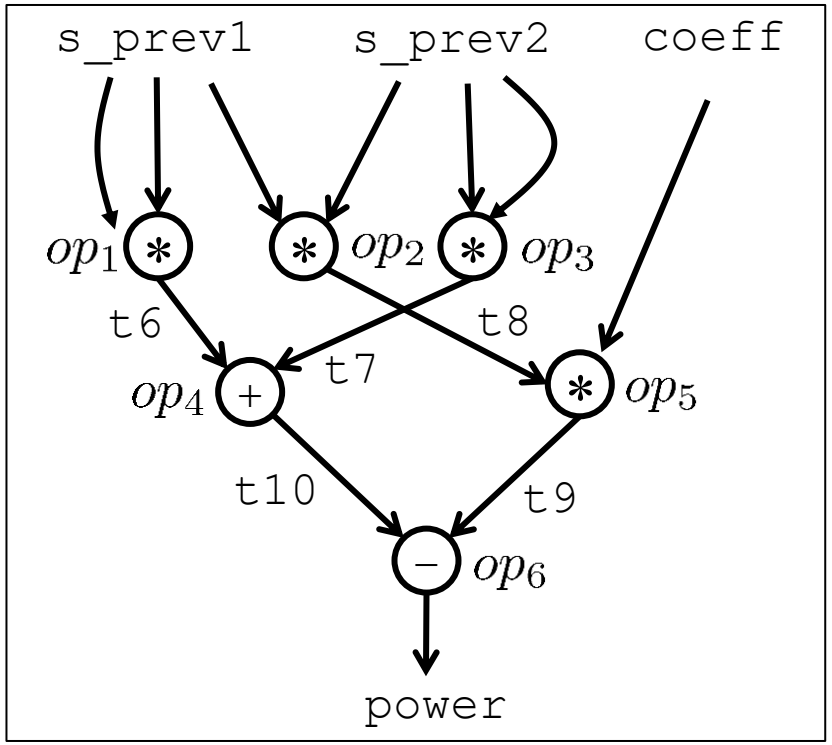
```
B3: t6= s_prev1 * s_prev1
    t7= s_prev2 * s_prev2
    t8= s_prev1 * s_prev2
    t9= t8 * coeff
    t10= t6+t7
    power= t10 – t9
```



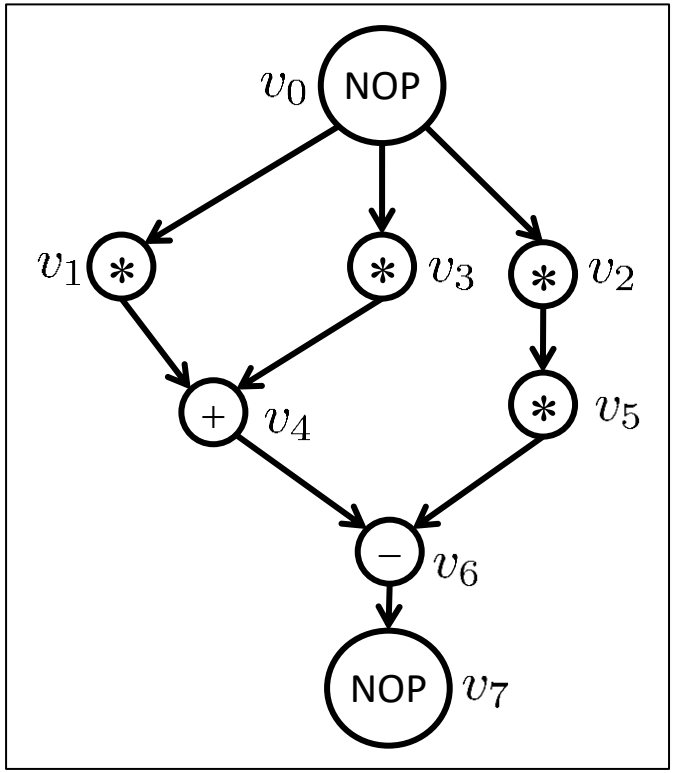
Operation Binding with Left-Edge Algorithm

- Example: SGU for basic block B3 of Goertzel algorithm

Data flow graph

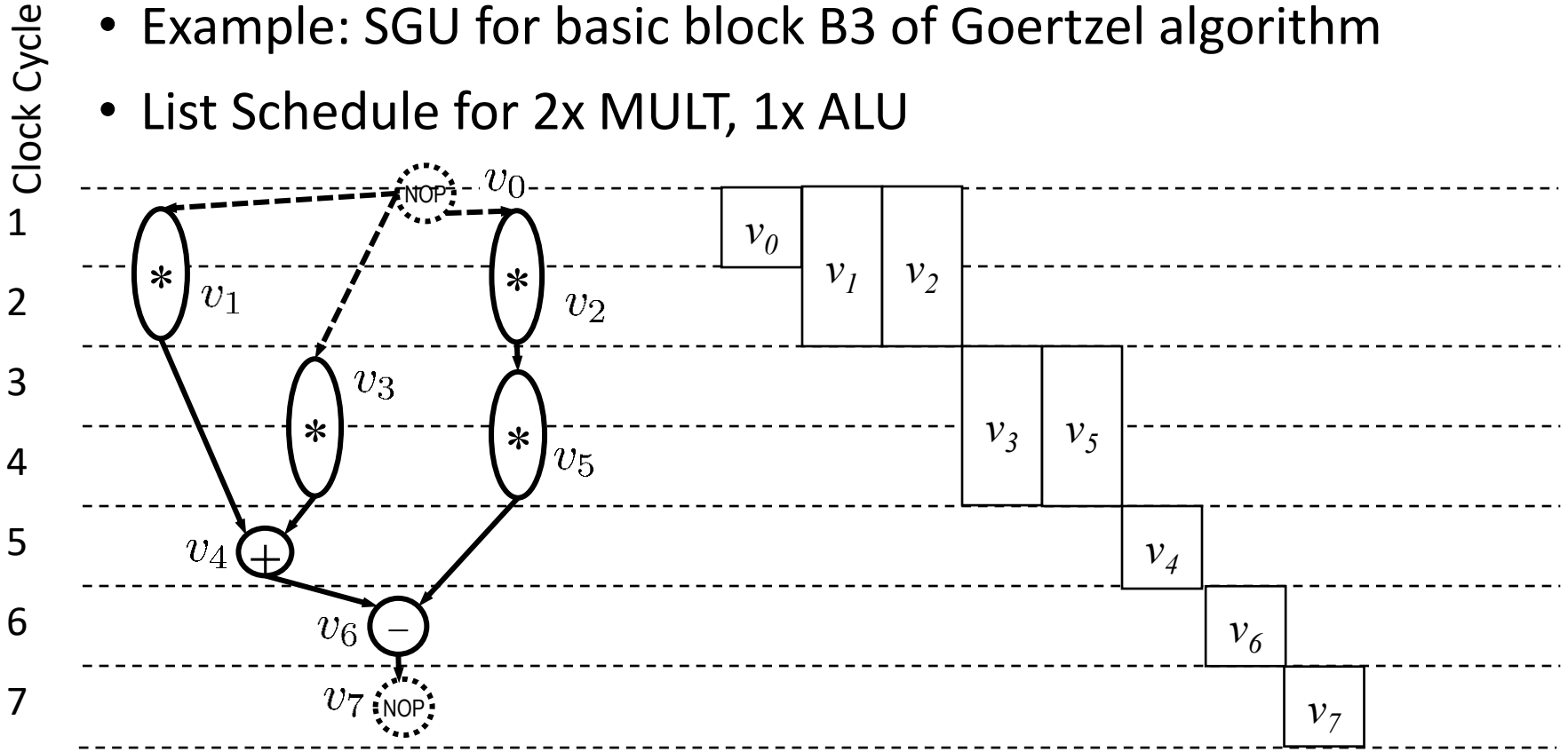


Sequencing graph unit



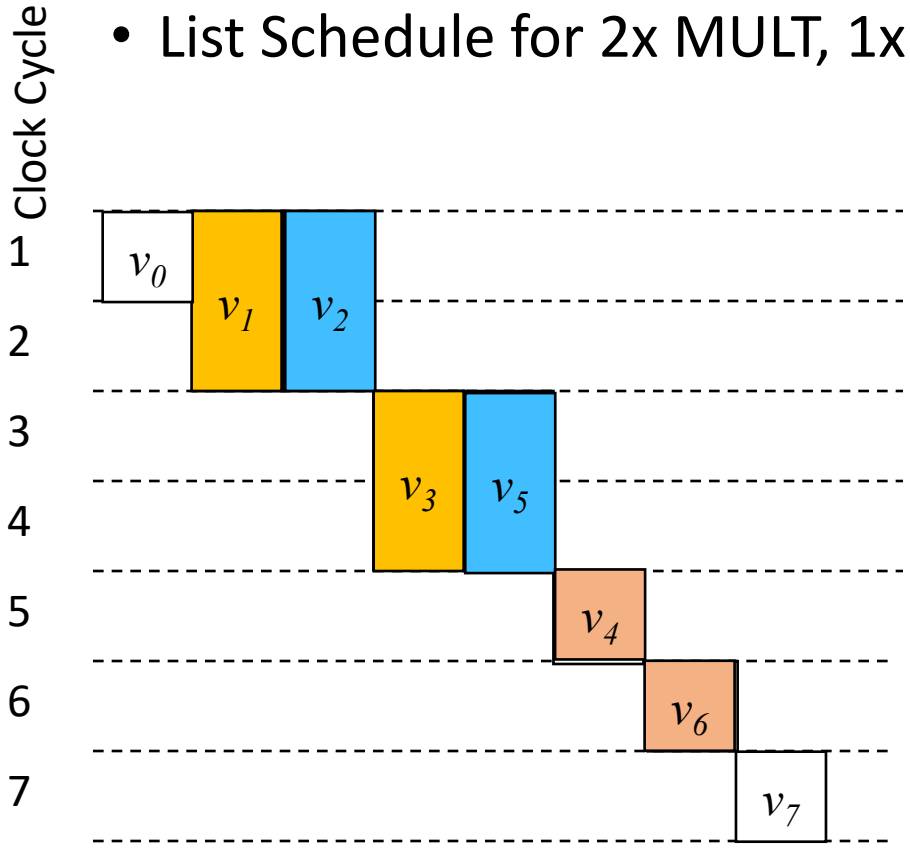
Operation Binding with Left-Edge Algorithm

- Example: SGU for basic block B3 of Goertzel algorithm
- List Schedule for 2x MULT, 1x ALU

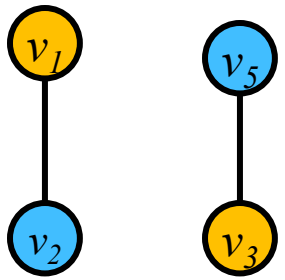


Operation Binding with Left-Edge Algorithm

- Example: SGU for basic block B3 of Goertzel algorithm
- List Schedule for 2x MULT, 1x ALU



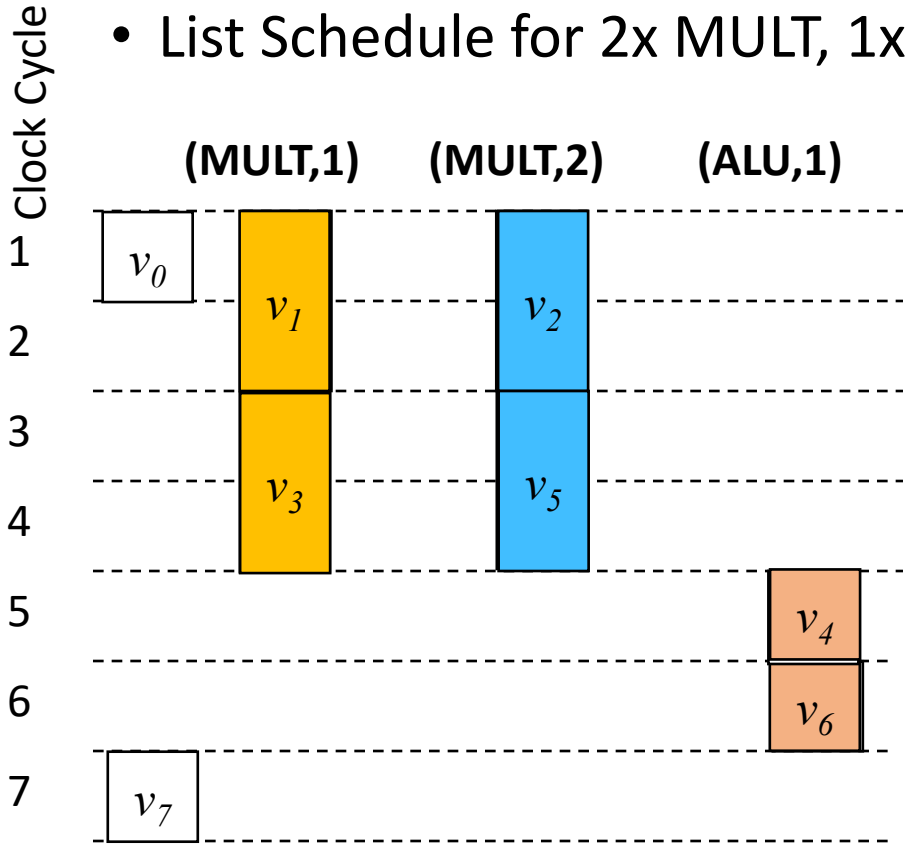
Operations-Conflict-Graph (MULT)



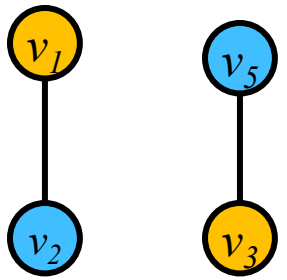
Operations-Conflict-Graph (ALU)

Operation Binding with Left-Edge Algorithm

- Example: SGU for basic block B3 of Goertzel algorithm
- List Schedule for 2x MULT, 1x ALU



Operations-Conflict-Graph (MULT)



Operations-Conflict-Graph (ALU)

D3-4 Register Binding

Register Binding

- Registers store values of variables in between operations in the data path.
- **Live variables:** Variables are live from their generation until their last use (lifetime). After their last use they are called dead.
- **Register Sharing:** Number of registers is decreased by storing variables with non-overlapping lifetimes in the same register..

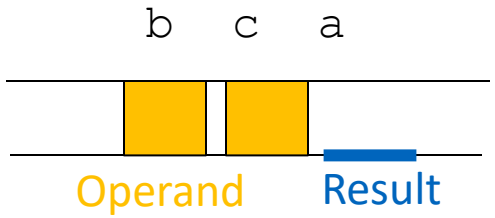
Register Binding

- Variables are incompatible, if their lifetimes overlap.
- **Register-conflict graph**:
  - Variables are the nodes
  - Incompatible variables are connected by an edge.
  - Interval graph
- Left-Edge Algorithm can compute minimal coloring.
- Coloring defines cliques of variables that can share one register.

Local Live Variable Analysis

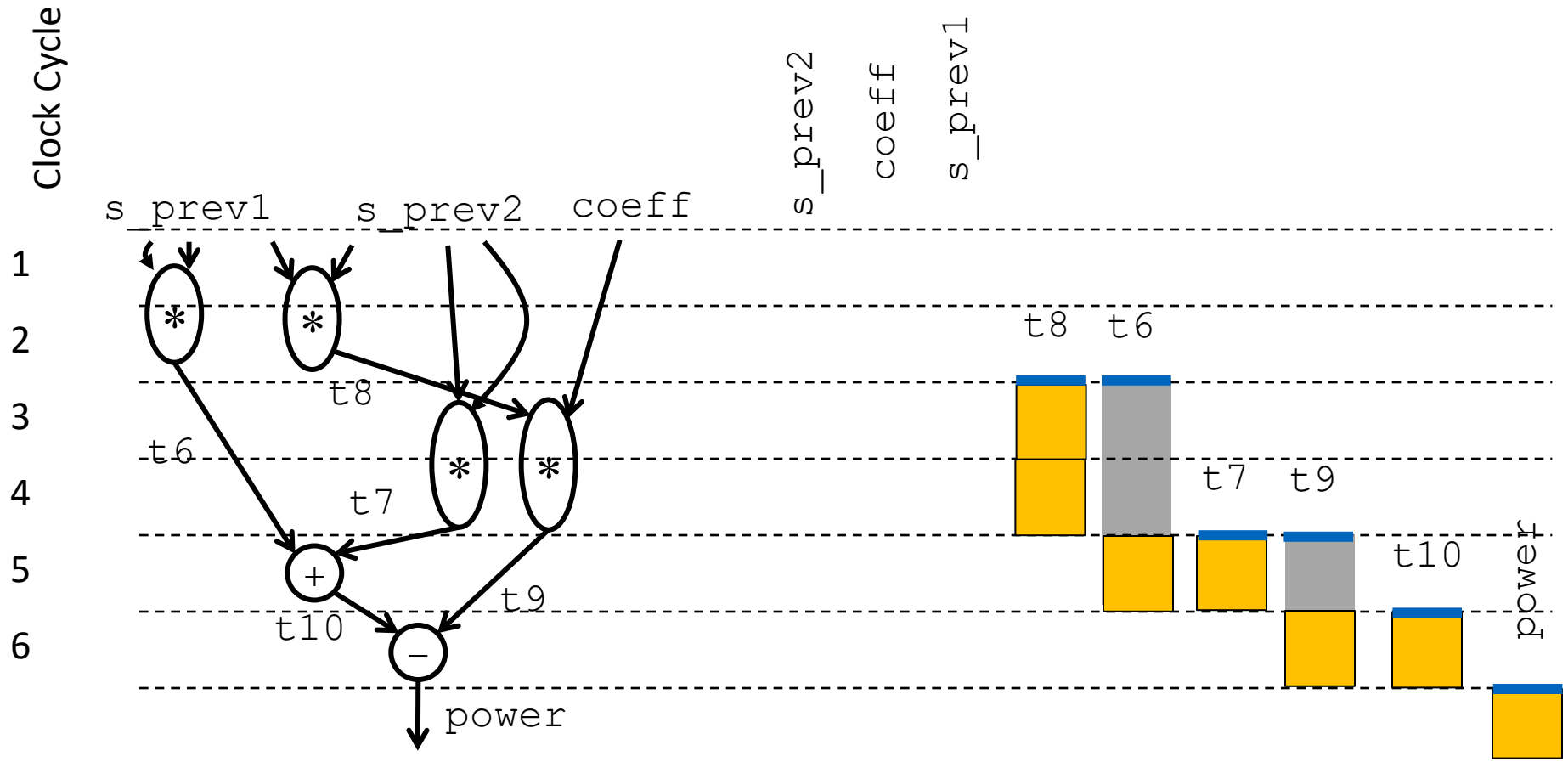
- We first need to run a Global Live Variable Analysis
- For each basic block Bx mark variables live
  - IN[Bx]: at entry as result of clock cycle 0: Input variables or variables computed in previous block that are used in this basic block.
  - Out [Bx]: at exit as operand of clock cycle (latency+1): Output variables, that are used in successor basic blocks.
- For all other clock cycles mark variables as operand or results of operation.
- Lifetime: Interval between variable being for the first time the result of an operation until being the last time operand of an operation

Clock cycle X: `a=b+c`



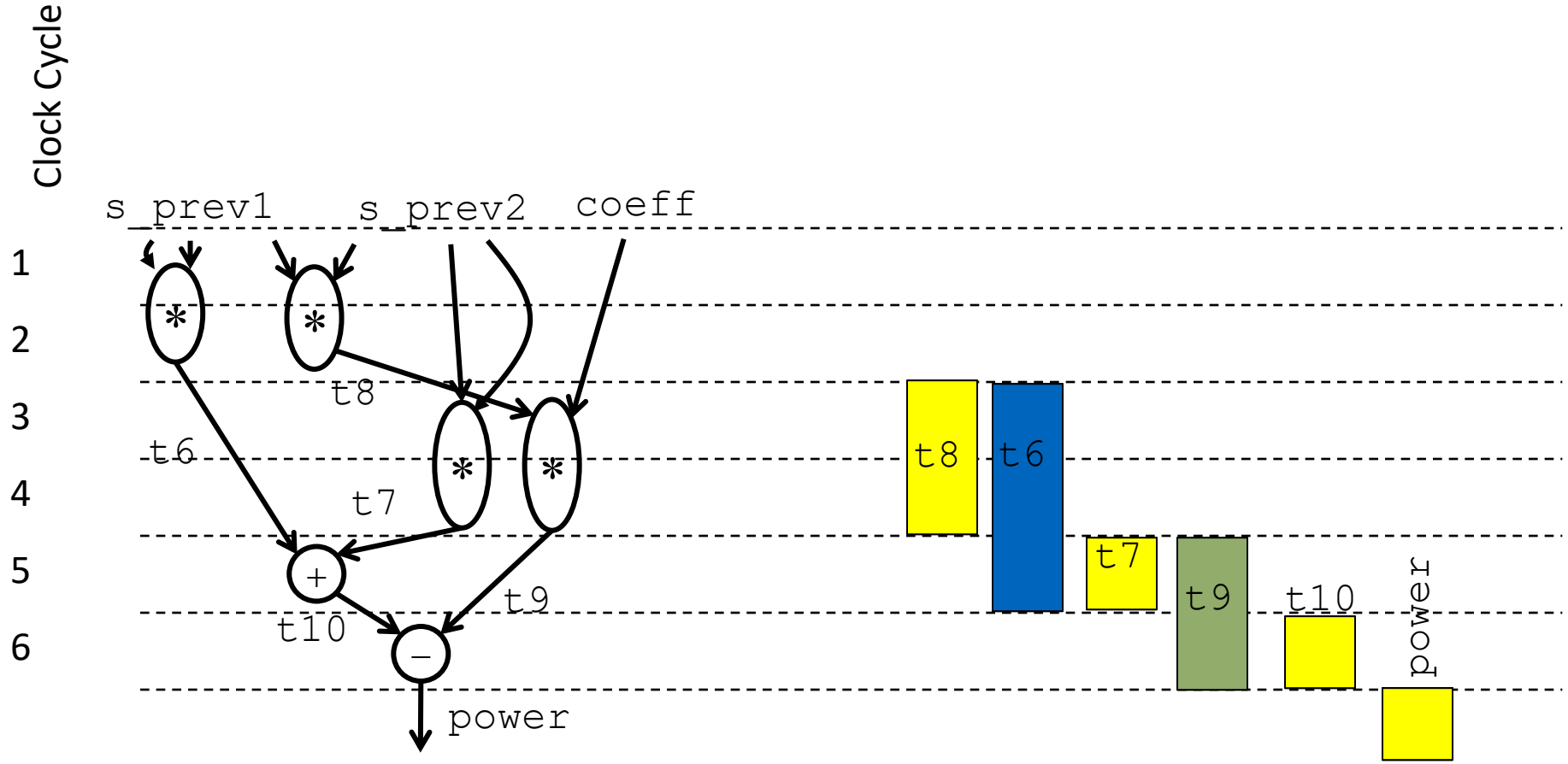
Local Live Variable Analysis

- Example: SGU for basic block B3 of Goertzel algorithm
- List Schedule for 2x MULT, 1x ALU



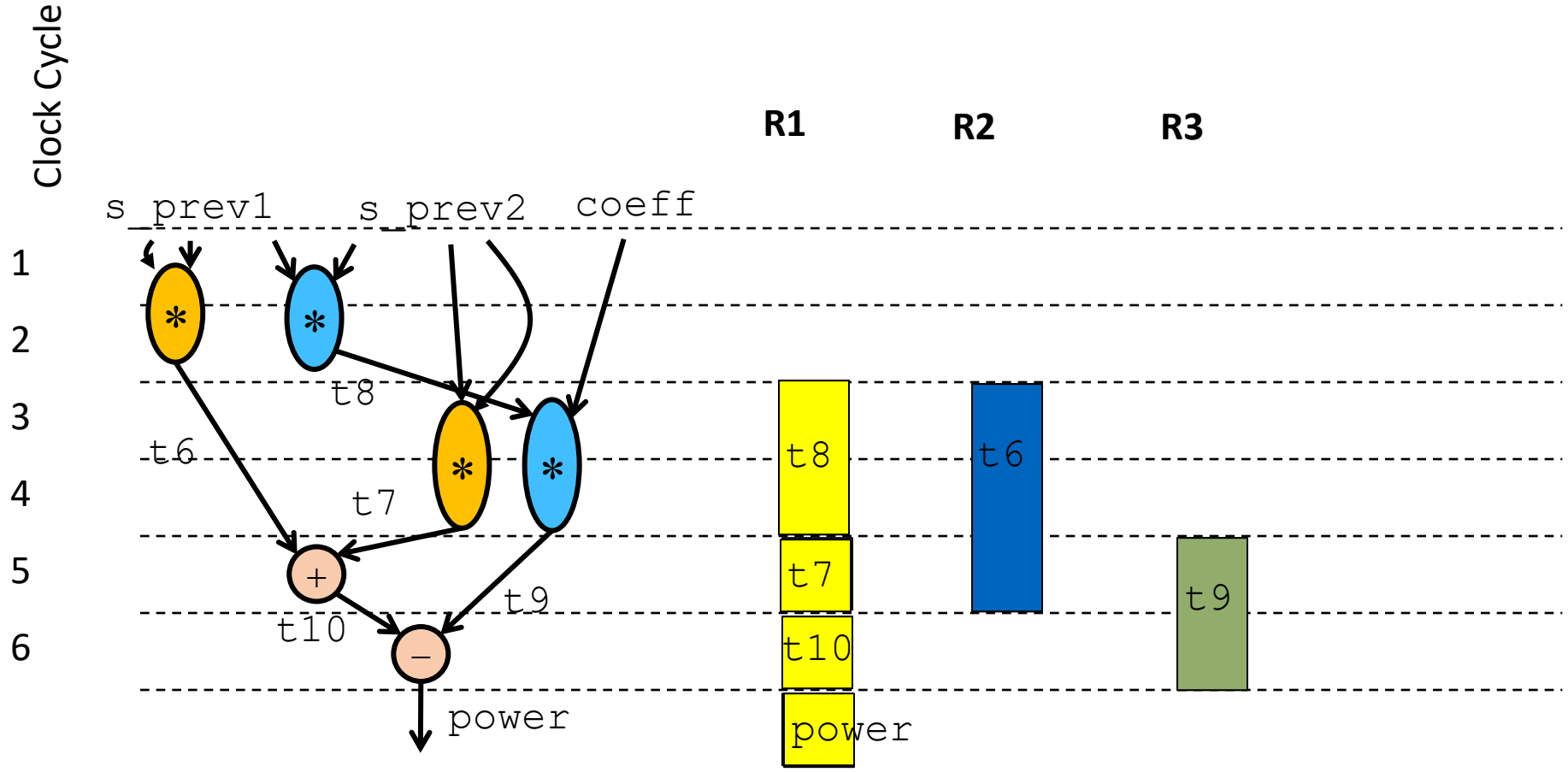
Register Binding with Left-Edge Algorithm

- Example: SGU for basic block B3 of Goertzel algorithm
- List Schedule for 2x MULT, 1x ALU



Register Binding with Left-Edge Algorithm

- Example: SGU for basic block B3 of Goertzel algorithm
- List Schedule for 2x MULT, 1x ALU



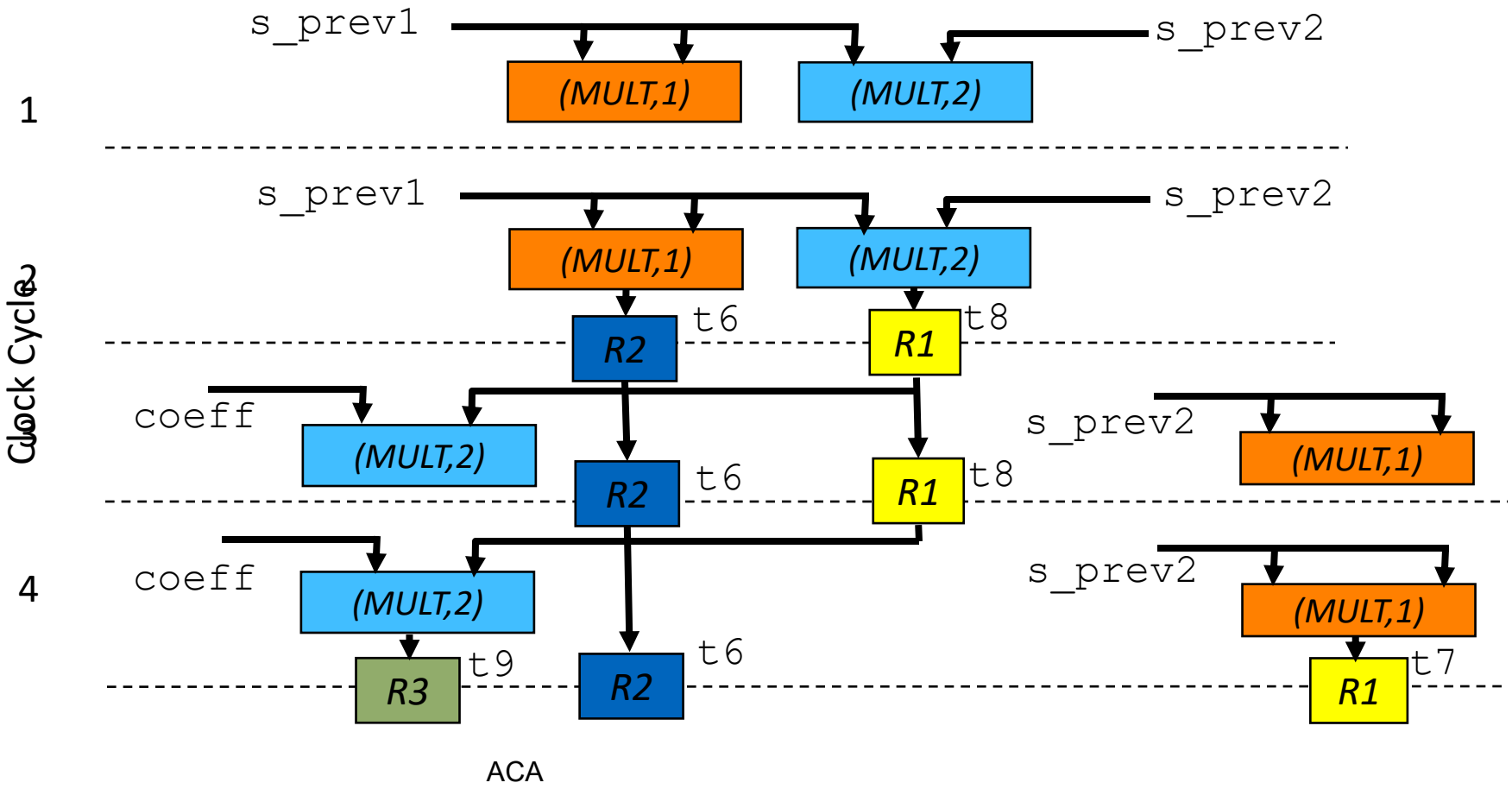
D3-5 Datapath Generation

Data Flow Graph with Schedule and Binding

- Shows data flow between functional units and registers.
- Number of multiplexers can be determined.
- Required control signals (activation signals) can be determined.
- Allows to generate RTL description of data path.
- Degree of freedom to minimize multiplexers: Choice of equivalent functional unit inputs.

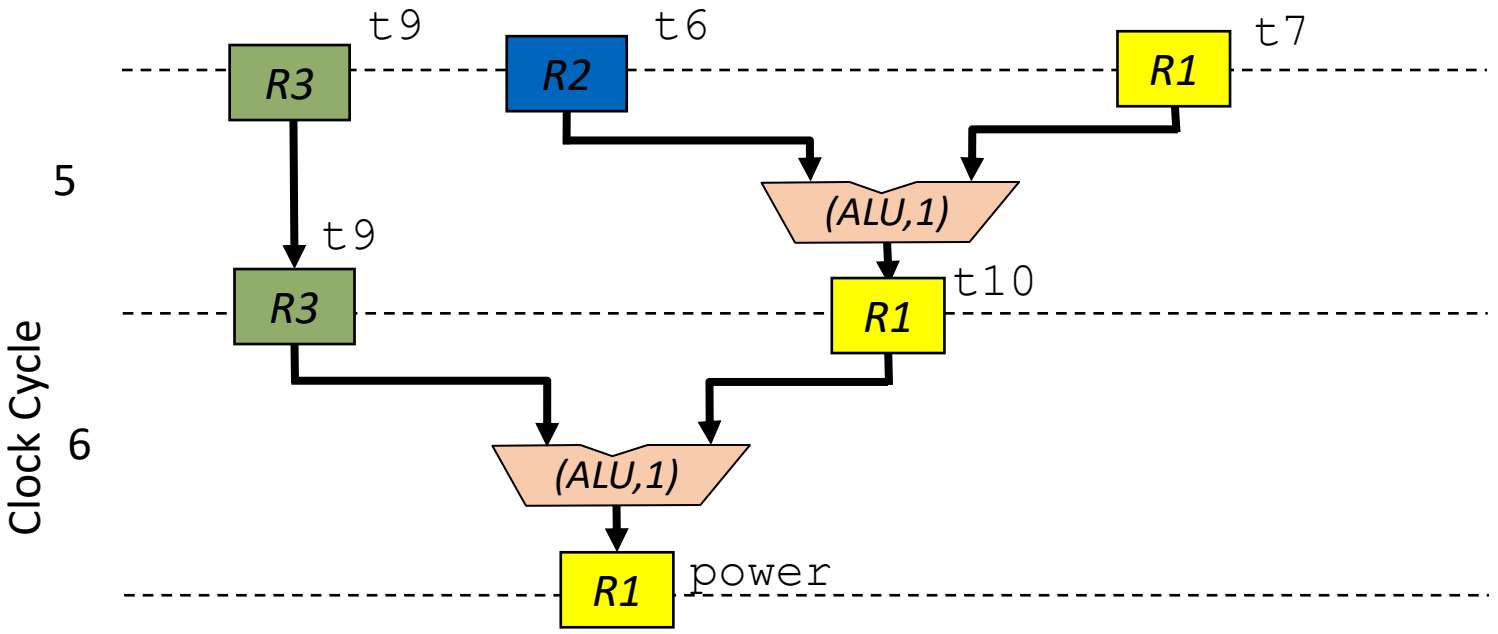
Data Flow Graph with Schedule and Binding

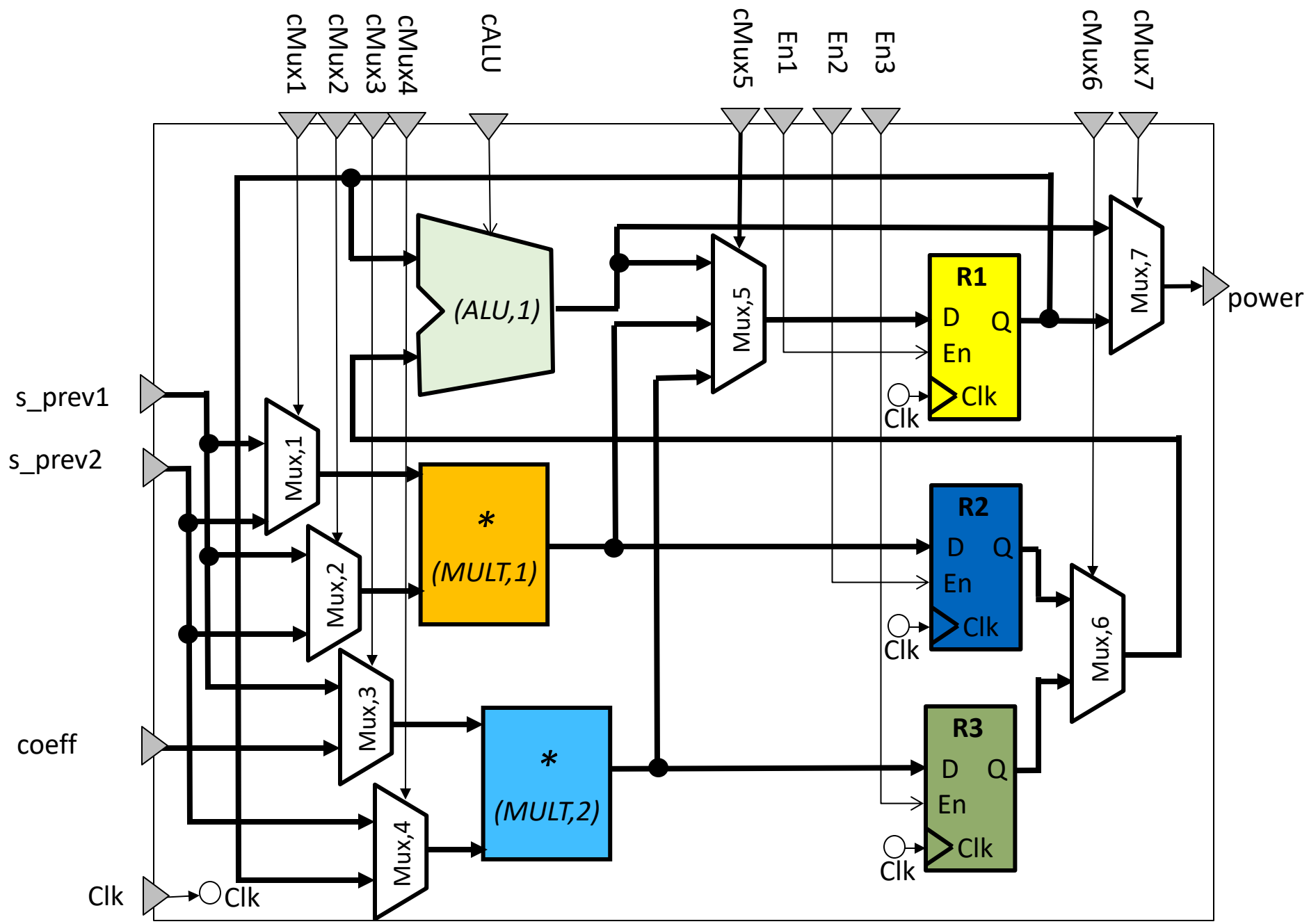
- Example: SGU for basic block B3 of Goertzel algorithm



Data Flow Graph with Schedule and Binding

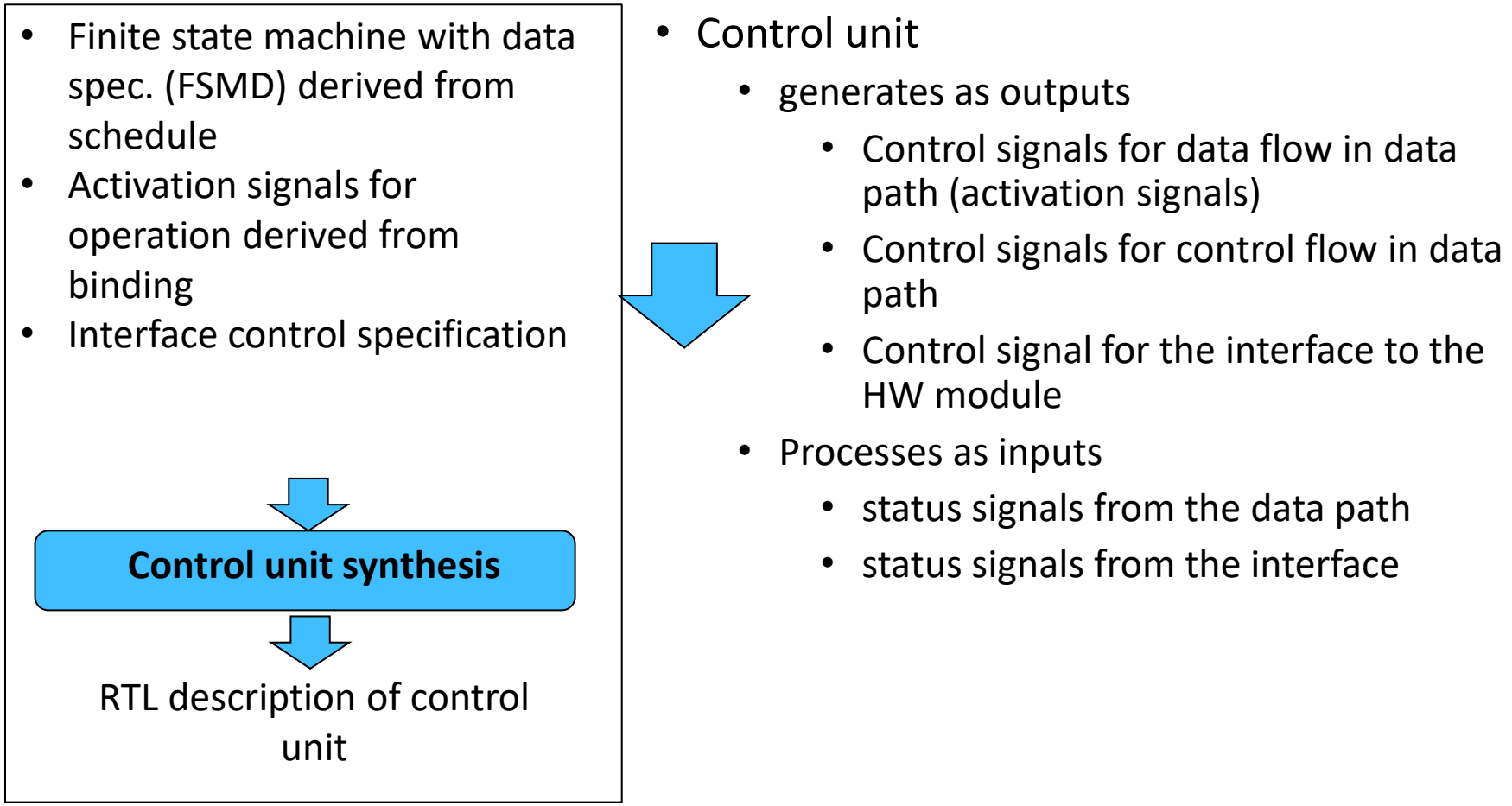
- Example: SGU for basic block B3 of Goertzel algorithm




cMux1
cMux2
cMux3
cMux4
cALU
cMux5
En1
En2
En3
cMux6
cMux7
(ALU,1)
Mux,5
R1
D
Q
En
Clk
Clk
Mux,7
power
s_prev1
s_prev2
Mux,1
Mux,2
*
(MULT,1)
R2
D
Q
En
Clk
Clk
Mux,6
coeff
Mux,3
*
(MULT,2)
R3
D
Q
En
Clk
Clk
Mux,4
Clk
Clk

D3-5 Control Unit Generation

Overview


Finite state machine with data spec. (FSMD) derived from schedule
Activation signals for operation derived from binding
Interface control specification
Control unit synthesis
RTL description of control unit
Control unit
generates as outputs
Control signals for data flow in data path (activation signals)
Control signals for control flow in data path
Control signal for the interface to the HW module
Processes as inputs
status signals from the data path
status signals from the interface

Finite State Machines(FSM)

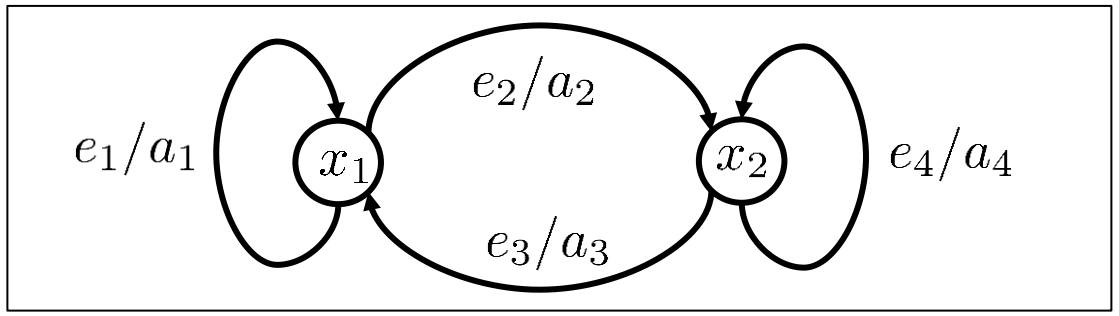
- Finite State Machine (FSM)
- FSM is 6-tuple:
  - Input alphabet: I
  - Output alphabet: O
  - Set of states: X
  - Set of starting states: $R \subseteq X$
  - State transition relation: $f \subseteq (X \times I \times X)$
  - Output relation: $g \subseteq (X \times O \times X)$

- FSM is deterministic, if there is a single starting state: $\mid R \mid = 1$
  and state transition and output relation are functions:
  $$f: (X \times I) \rightarrow X; g: (X \times I) \rightarrow O$$
- FSM is completely specified, if the state transition and output relation are completely defined.

State Diagram

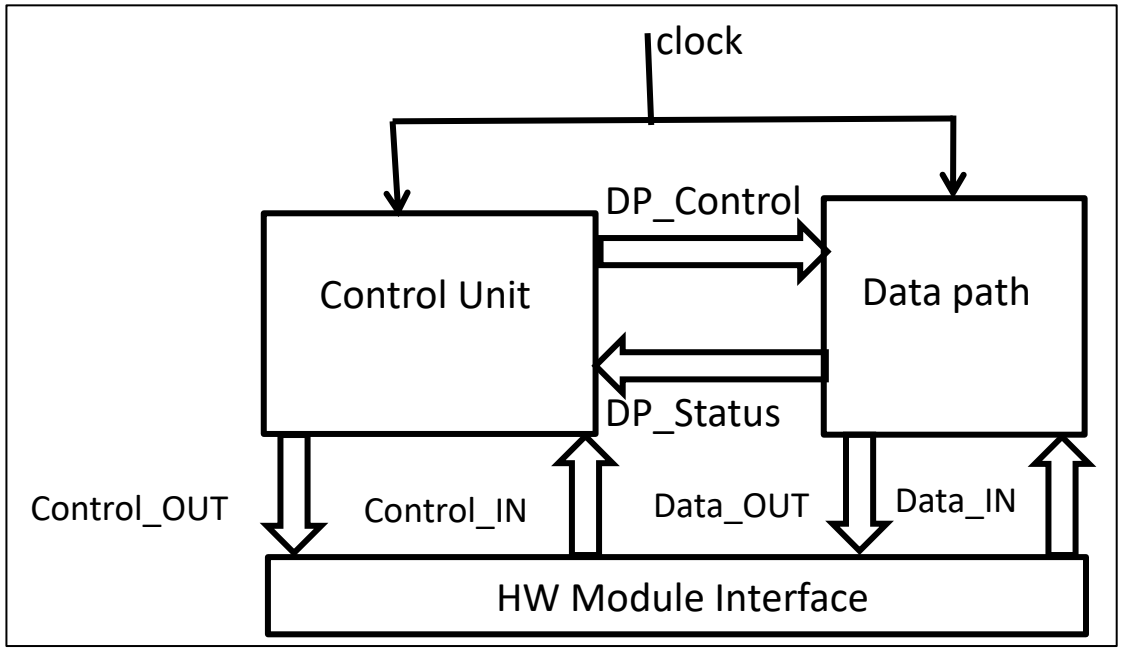
- Graphical representation of a FSM
- Directed graph $G_Z(V, E)$
- Nodes represent the states: $x_i \in X$
- Edges represent state transitions with labels e/a ,

  whereby $(x_i, e, x_j) \in f$ and $(x_i, a, x_j) \in g$.
  - State transition, if transition event e occurs.
  - Output event a happens, when state transition occurs.



Synchronous Implementation

- Control unit described as FSM
- Synchronous:
  - Common clock signal for data path and control unit
  - State transition of FSM of control unit at predefined clock event (e.g. rising edge)
- Description:
  - State coding
  - Next State logic
  - Output logic

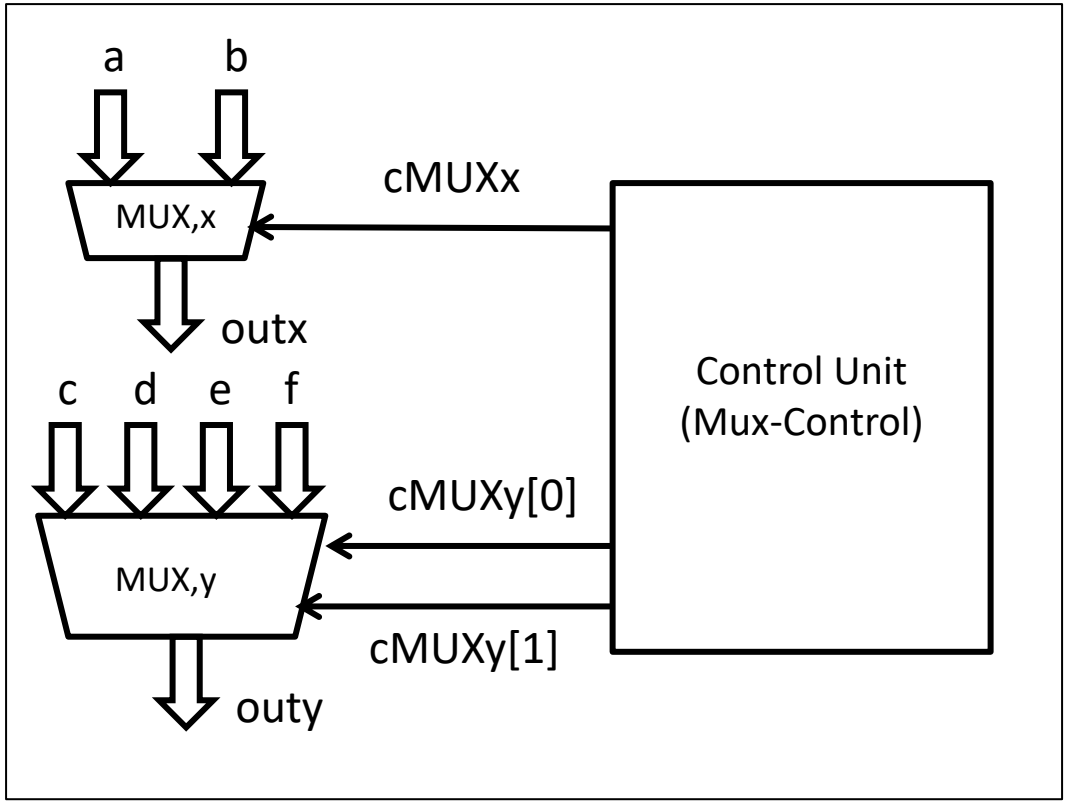


Multiplexer Control

- Example: 2 and 4 outputs

cMUXx	out1
0	a
1	b

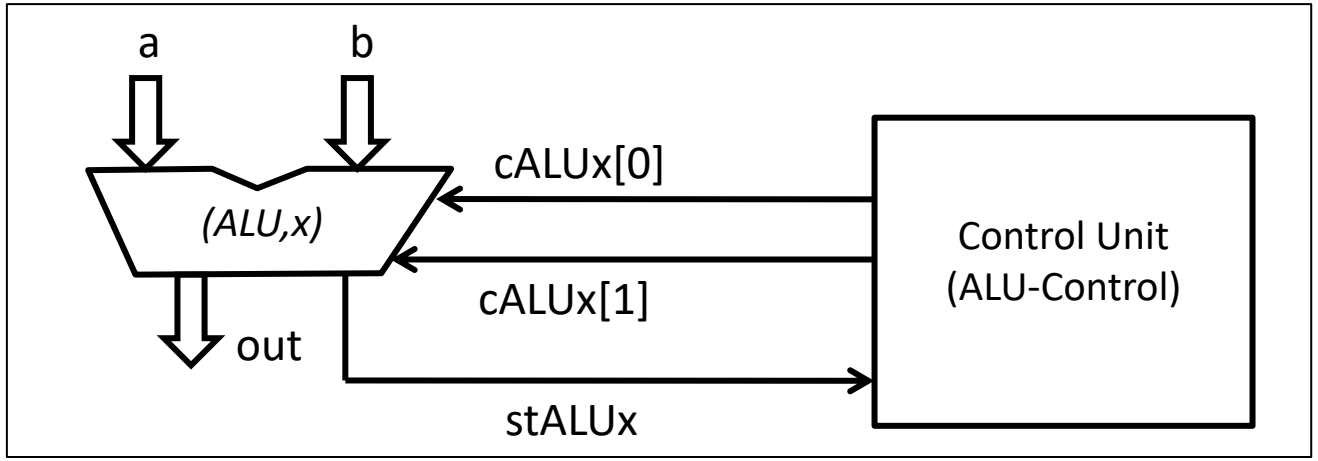
cMUXy[1]	cMUXy[0]	out2
0	0	c
0	1	d
1	0	e
1	1	f



ALU-Control

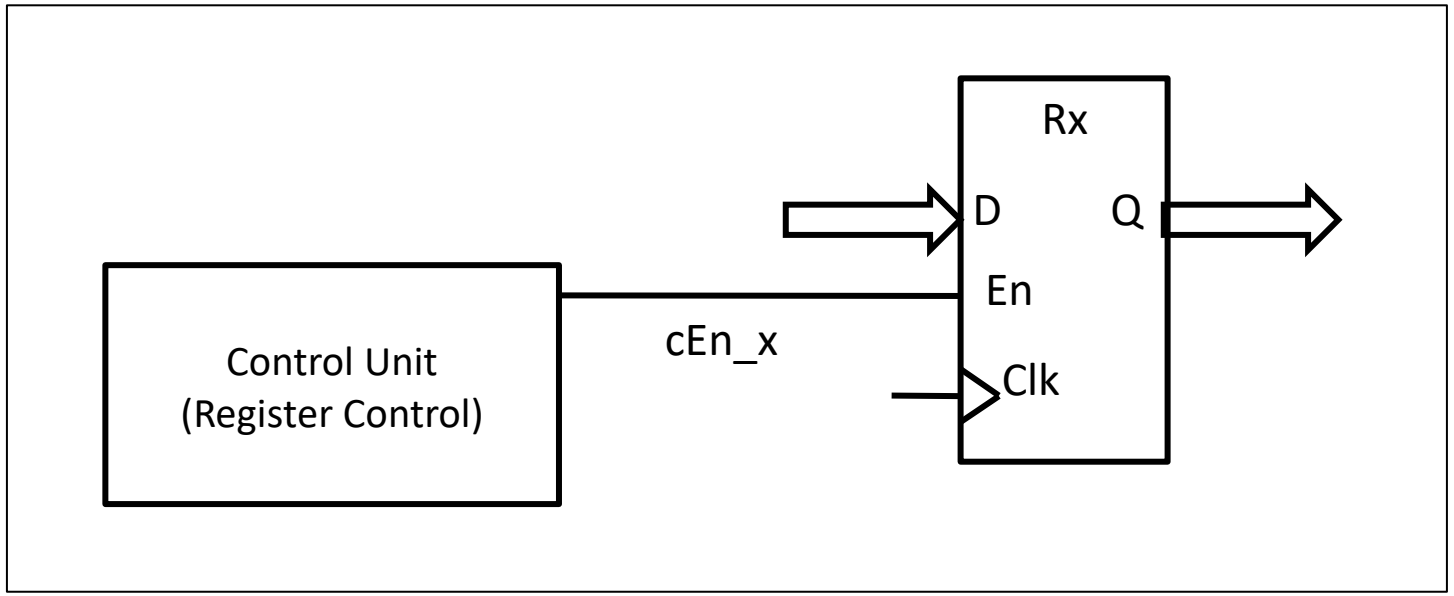
- Example: ALU with 4 Operations (+,-,<,<=)

Op	cALUx[1]	cALUx[0]	a,b	stALUx	out
+	0	0			a+b
-	0	1			a-b
<	1	0	a<b	0	
<	1	0	a>=b	1	
<=	1	1	a<=0	0	
<=	1	1	A>b	1	



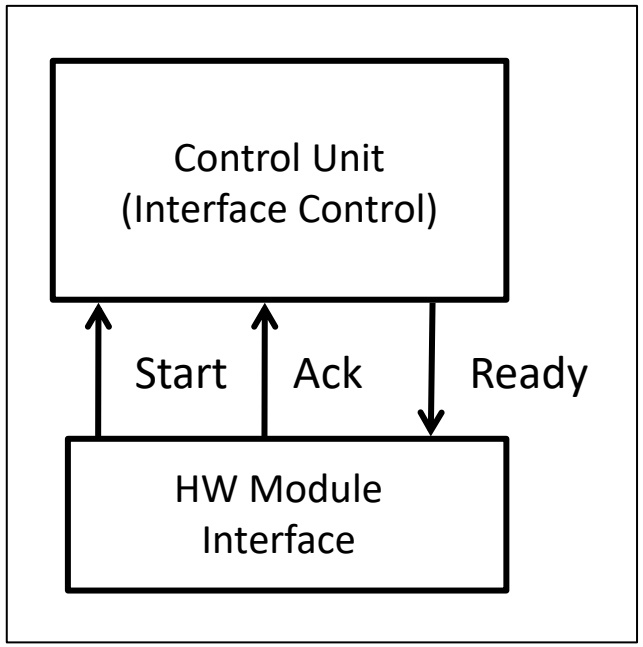
Register Control

- Example: Register with Enable

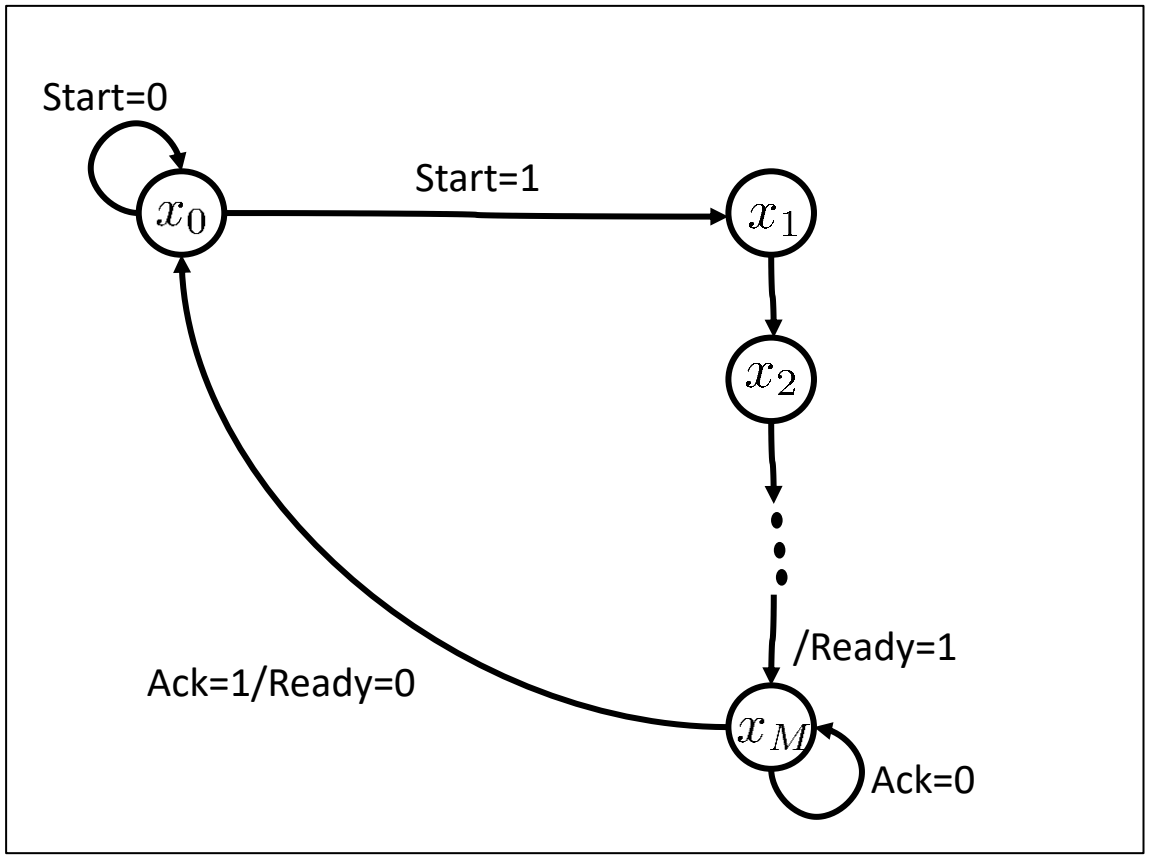


Interface Control

- Example 1: Start, Ready, Acknowledge

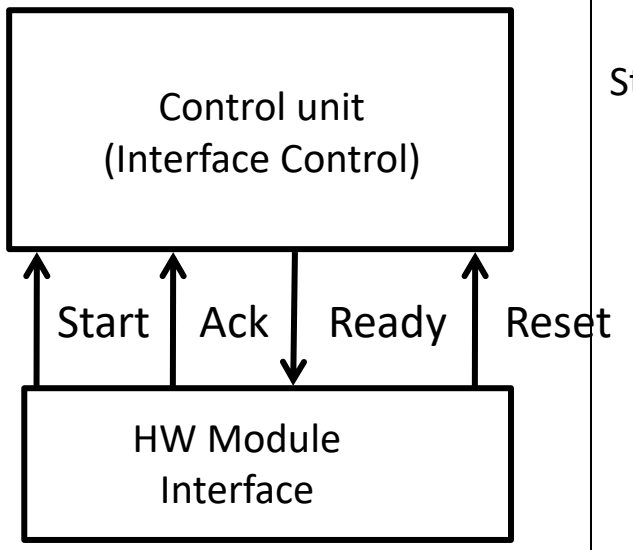


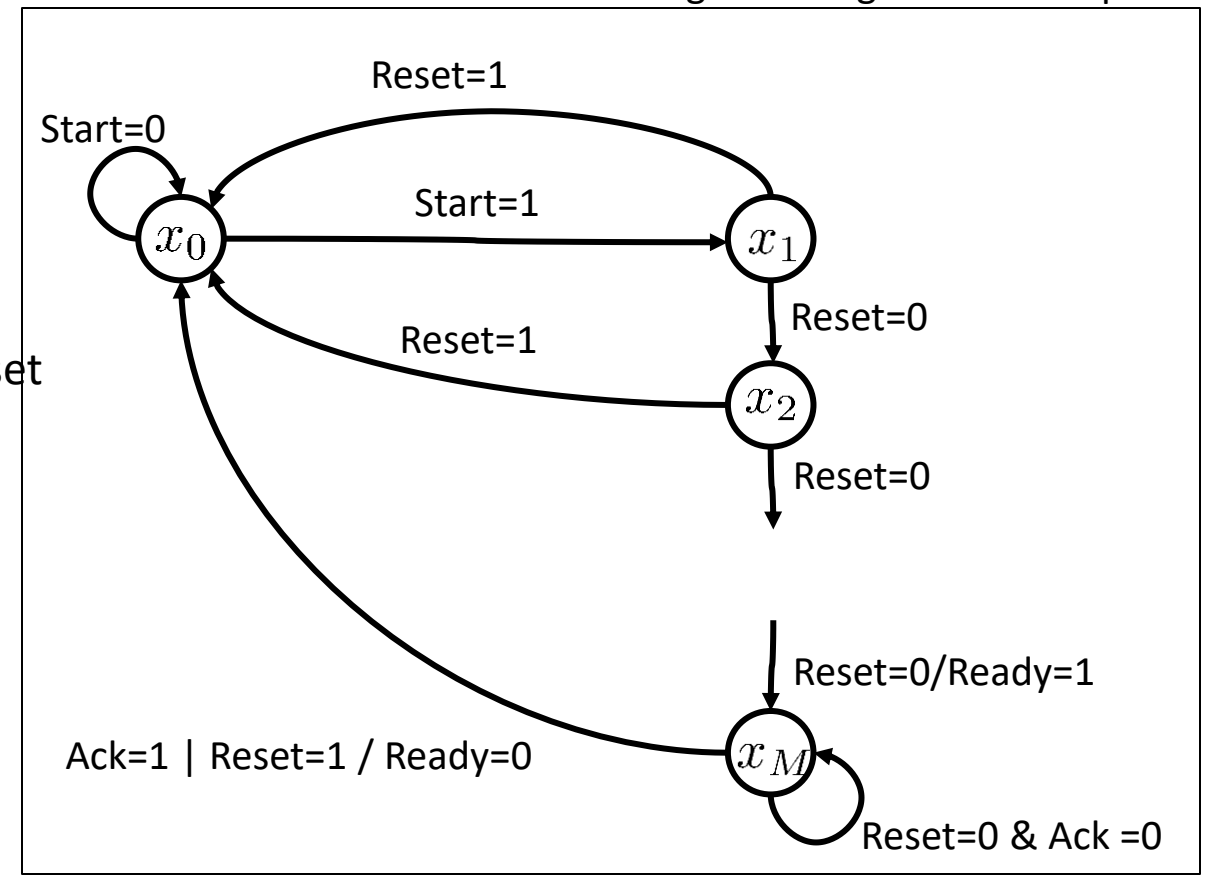
Rising clock edge ⌈ clock is impizit



Interface Control

- Example 2: Start, Ready, Acknowledge, Reset

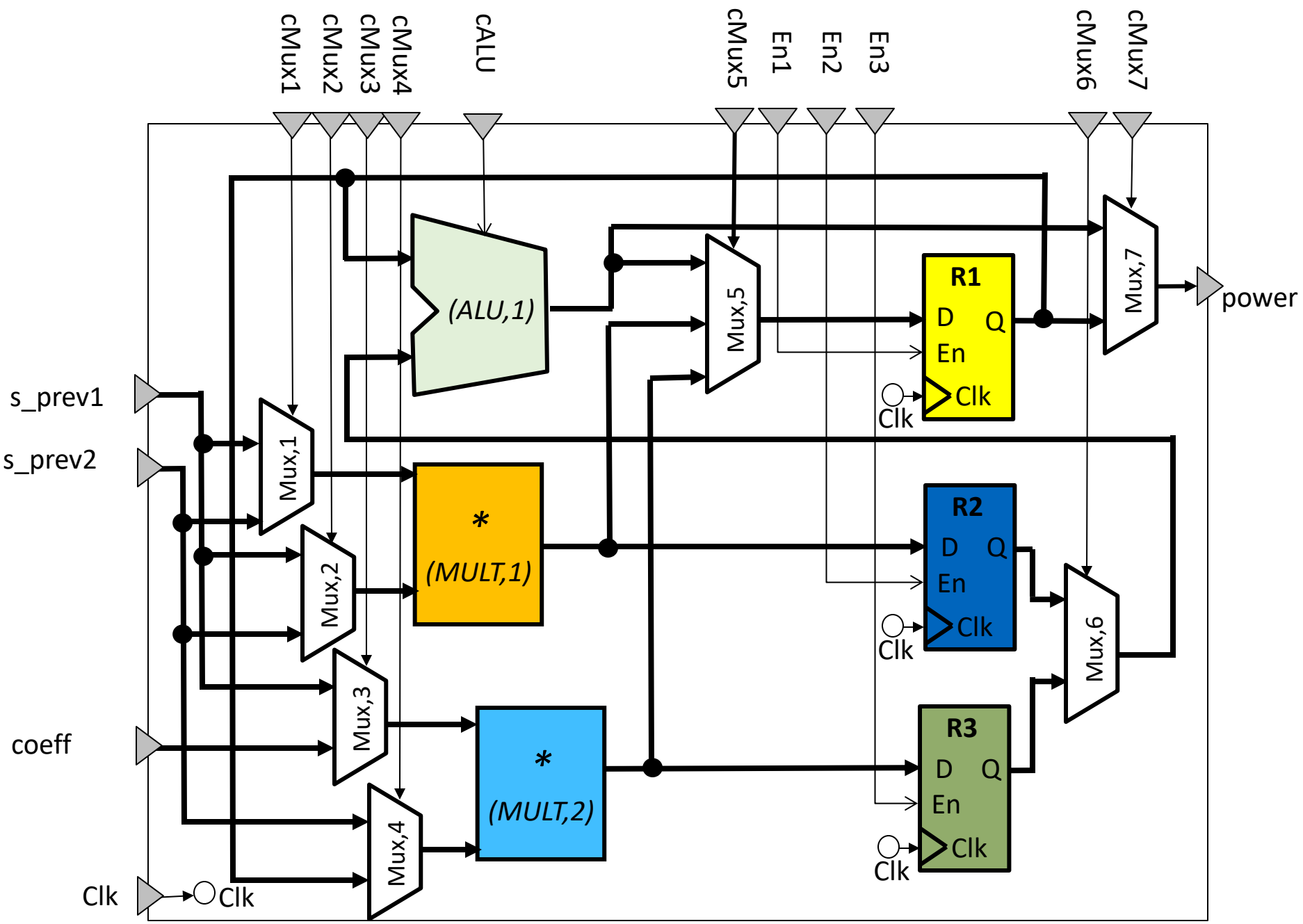




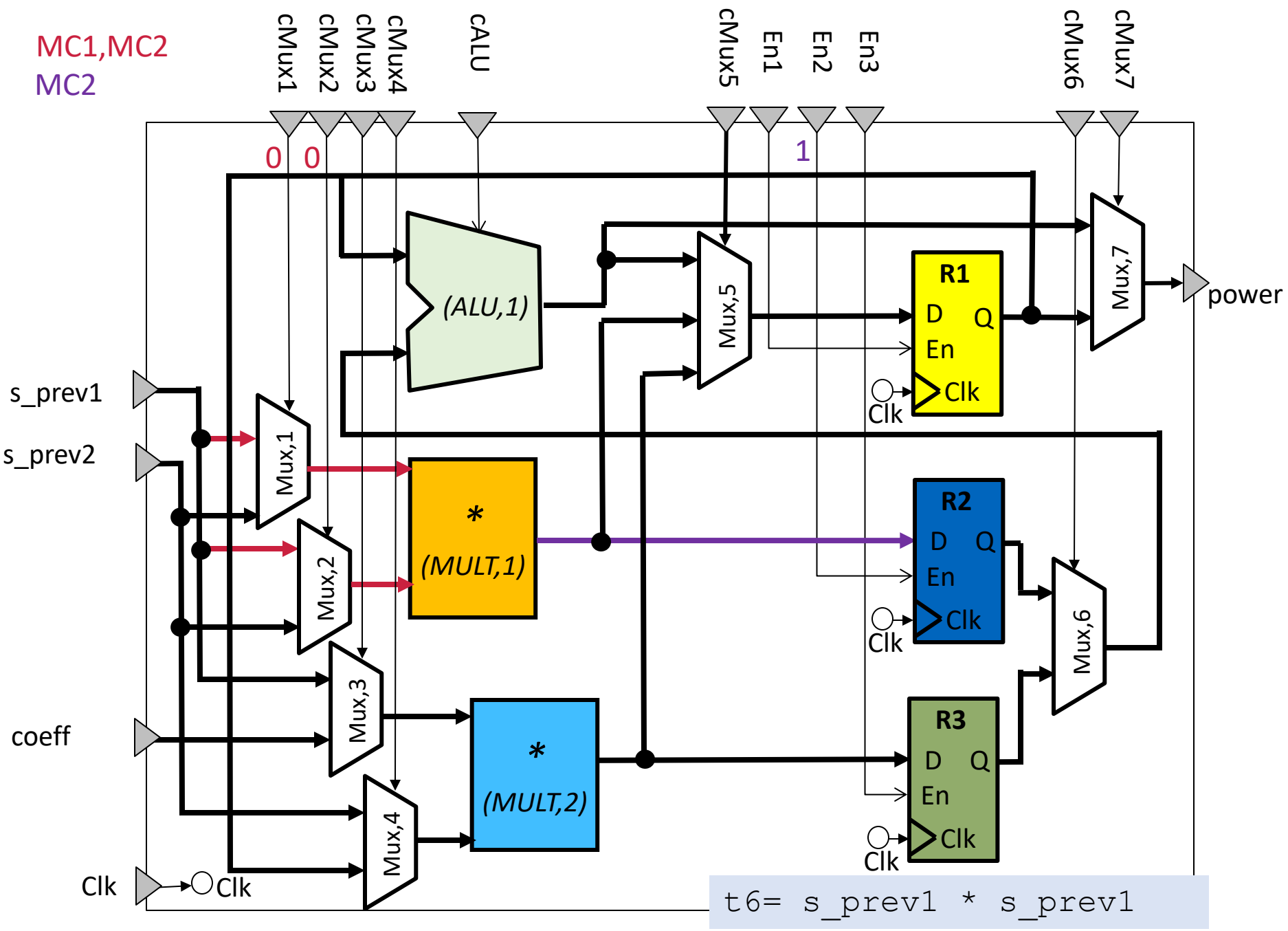
Activation Signals For Operations

- The activation signals of an operation are all control signals required for its execution.

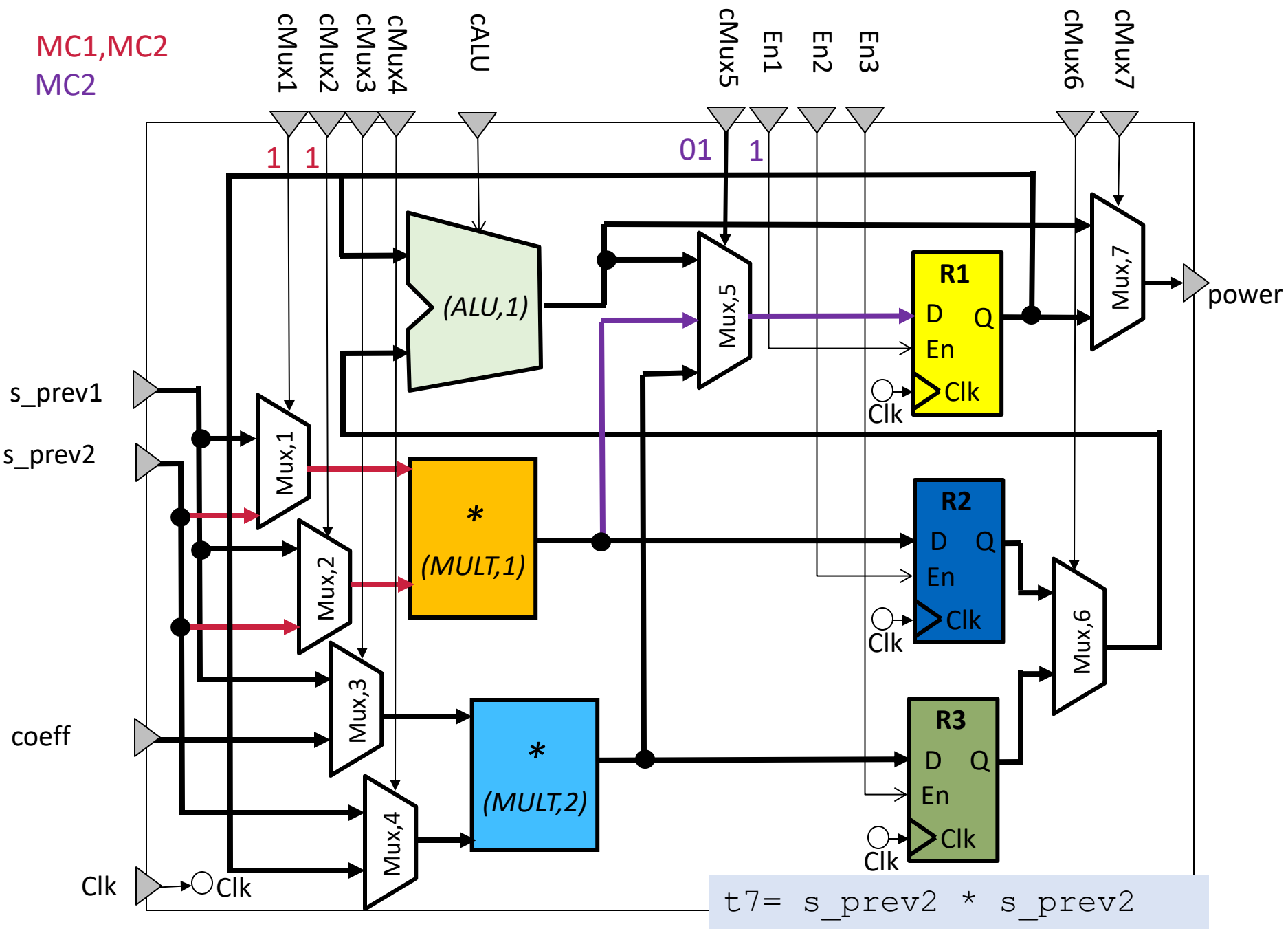
- Activation signals may include:
  - The multiplexer control signals to establish connection between input registers, functional unit and output register
  - Register enable signal to write result to output register
  - ALU control signals to select correct operation


cMux1
cMux2
cMux3
cMux4
cALU
cMux5
En1
En2
En3
cMux6
cMux7
(ALU,1)
Mux,5
R1
D
Q
En
Clk
Clk
Mux,7
power
s_prev1
s_prev2
Mux,1
Mux,2
*
(MULT,1)
R2
D
Q
En
Clk
Clk
Mux,6
coeff
Mux,3
*
(MULT,2)
R3
D
Q
En
Clk
Clk
Mux,4
Clk
Clk

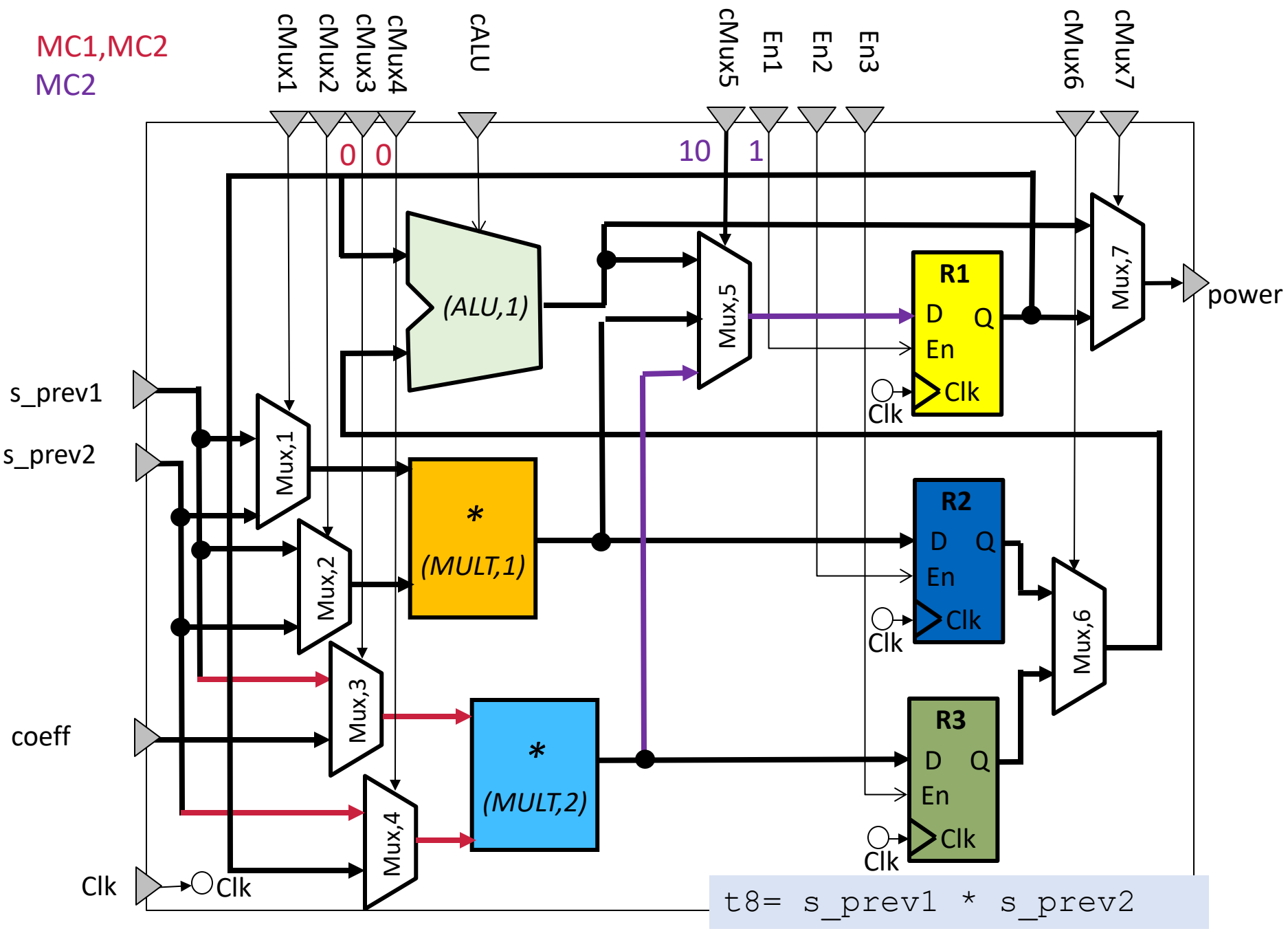
ACA


MC1,MC2
MC2
cMux1
cMux2
cMux3
cMux4
cALU
cMux5
En1
En2
En3
cMux6
cMux7
0
0
1
(ALU,1)
Mux,5
R1
D
Q
En
Clk
Clk
Mux,7
power
s_prev1
s_prev2
Mux,1
*
(MULT,1)
Mux,2
R2
D
Q
En
Clk
Clk
Mux,6
Mux,3
coeff
*
(MULT,2)
R3
D
Q
En
Clk
Clk
Mux,4
Clk
Clk
t6= s_prev1 * s_prev1

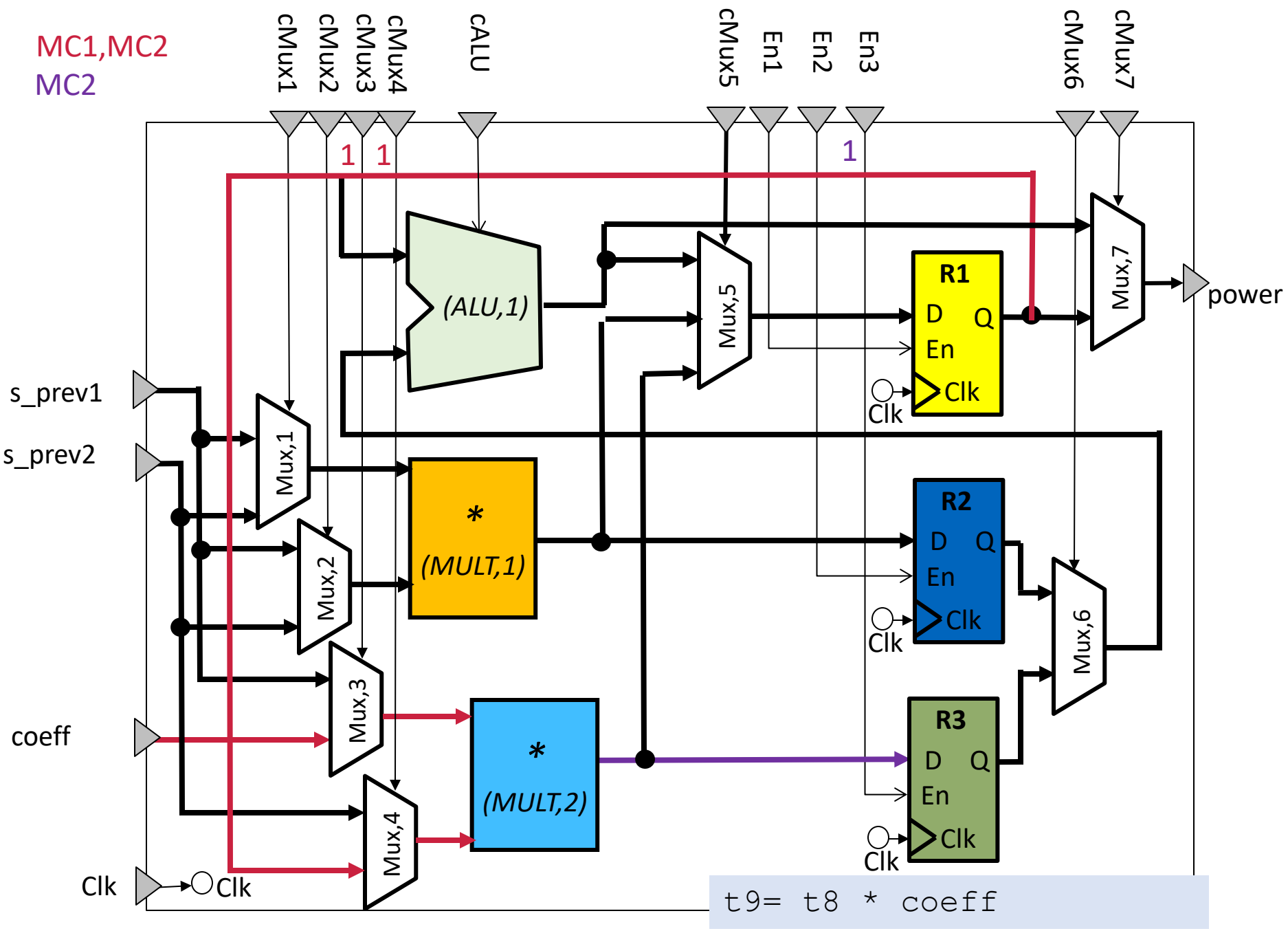
ACA


MC1,MC2
MC2
cMux1
cMux2
cMux3
cMux4
cALU
cMux5
En1
En2
En3
cMux6
cMux7
1
1
01
1
(ALU,1)
Mux,5
R1
D
Q
En
Clk
Clk
Mux,7
power
s_prev1
s_prev2
Mux,1
Mux,2
*
(MULT,1)
R2
D
Q
En
Clk
Clk
Mux,6
coeff
Mux,3
*
(MULT,2)
R3
D
Q
En
Clk
Clk
Mux,4
Clk
Clk
t7= s_prev2 * s_prev2

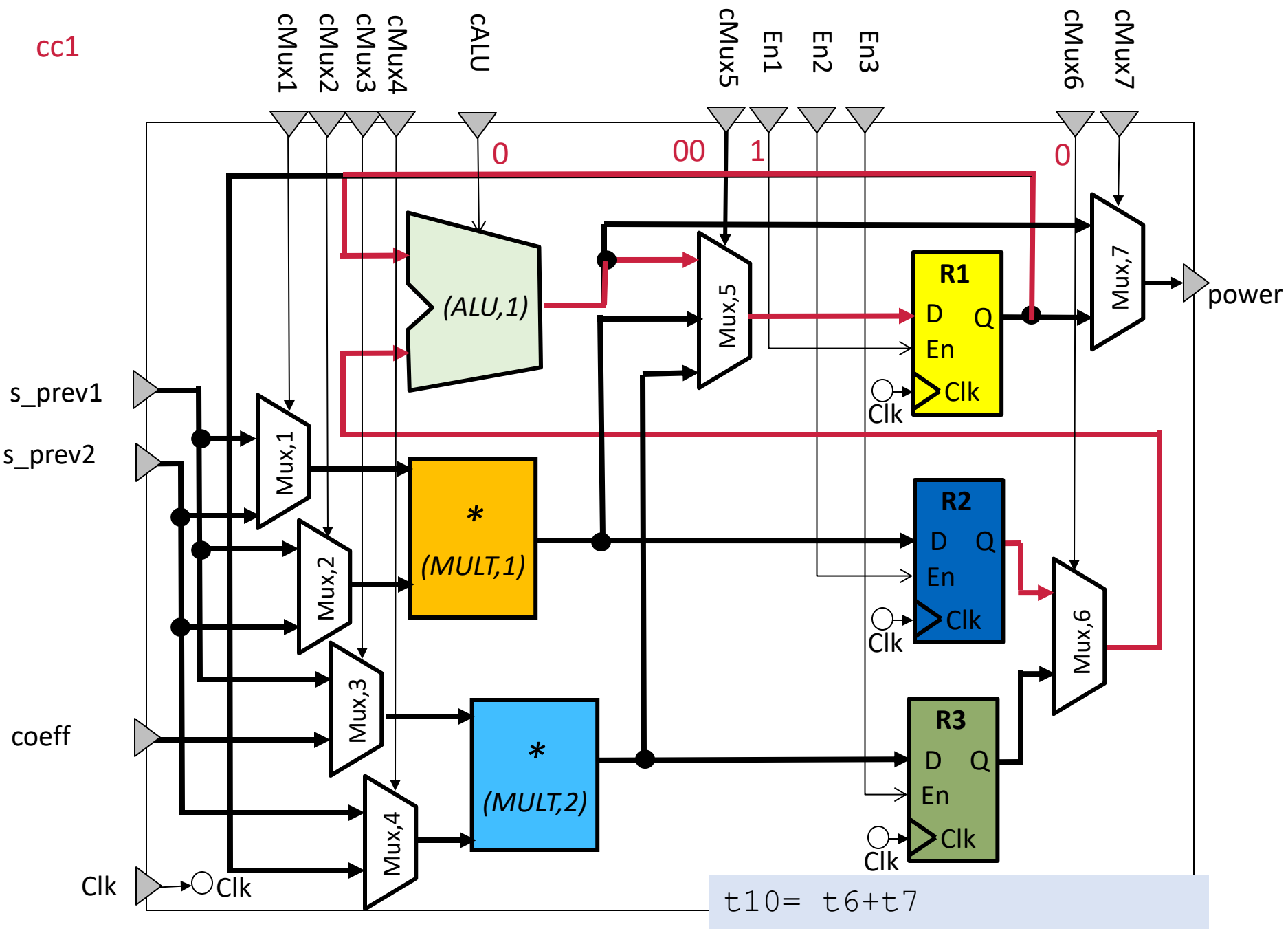
ACA


MC1,MC2
MC2
cMux1
cMux2
cMux3
cMux4
cALU
cMux5
En1
En2
En3
cMux6
cMux7
0
0
10
1
(ALU,1)
Mux,5
R1
D
Q
En
Clk
Clk
Mux,7
power
s_prev1
s_prev2
Mux,1
*
(MULT,1)
Mux,2
R2
D
Q
En
Clk
Clk
Mux,6
coeff
Mux,3
*
(MULT,2)
R3
D
Q
En
Clk
Clk
Mux,4
Clk
Clk
t8= s_prev1 * s_prev2

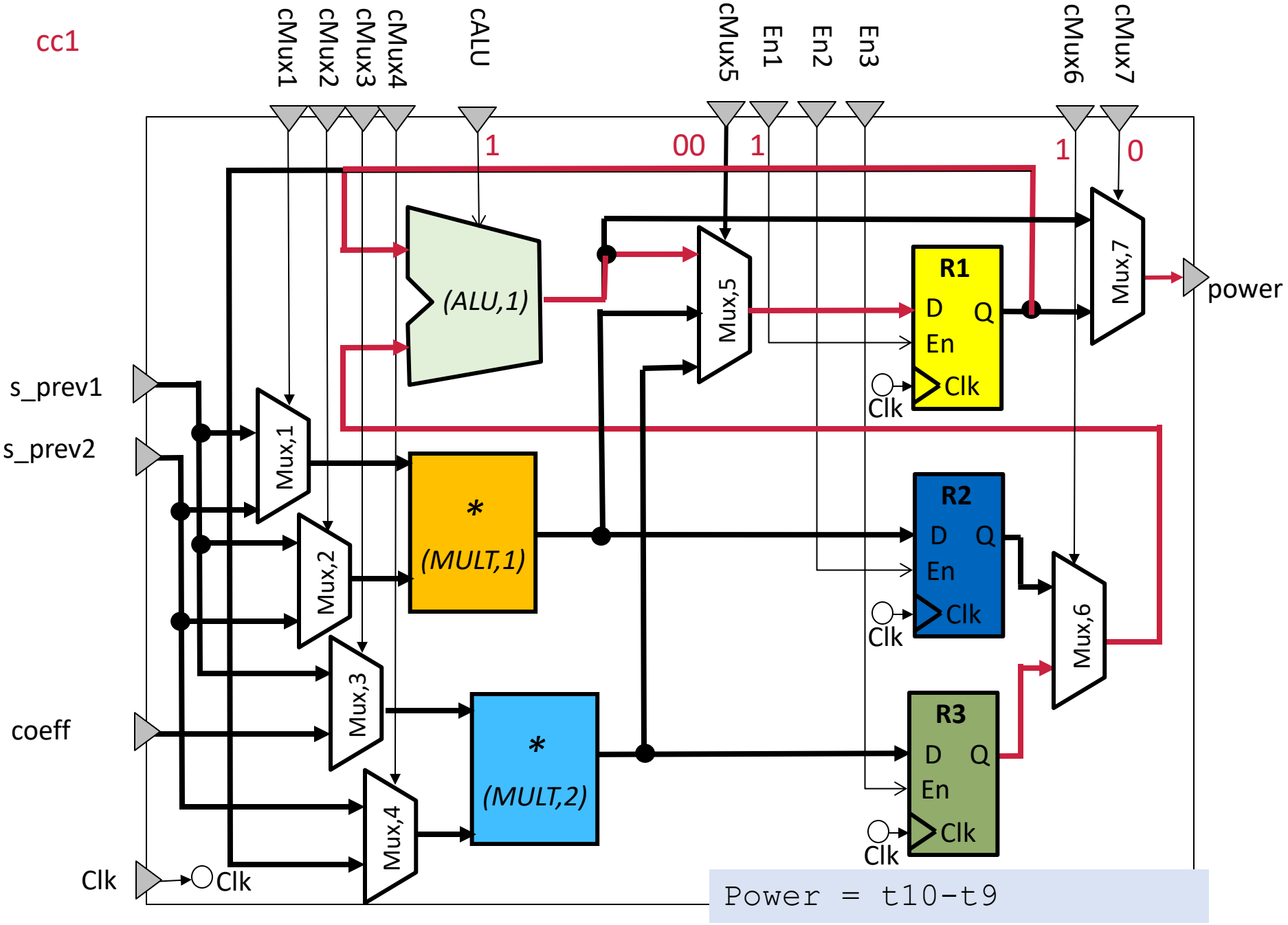
ACA


MC1,MC2
MC2
cMux1
cMux2
cMux3
cMux4
cALU
cMux5
En1
En2
En3
cMux6
cMux7
1
1
1
(ALU,1)
Mux,5
R1
D
Q
En
Clk
Clk
Mux,7
power
s_prev1
s_prev2
Mux,1
*
(MULT,1)
Mux,2
R2
D
Q
En
Clk
Clk
Mux,6
Mux,3
coeff
*
(MULT,2)
R3
D
Q
En
Clk
Clk
Mux,4
Clk
Clk
t9= t8 * coeff

ACA


cc1
cMux1
cMux2
cMux3
cMux4
cALU
cMux5
En1
En2
En3
cMux6
cMux7
0
00
1
0
(ALU,1)
Mux,5
R1
D
Q
En
Clk
Clk
Mux,7
power
s_prev1
s_prev2
Mux,1
*
(MULT,1)
Mux,2
R2
D
Q
En
Clk
Clk
Mux,6
Mux,3
coeff
*
(MULT,2)
R3
D
Q
En
Clk
Clk
Mux,4
Clk
Clk
t10= t6+t7

ACA


cc1
cMux1
cMux2
cMux3
cMux4
cALU
cMux5
En1
En2
En3
cMux6
cMux7
1
00
1
1
0
(ALU,1)
Mux,5
R1
D
Q
En
Clk
Clk
Mux,7
power
s_prev1
s_prev2
Mux,1
*
(MULT,1)
Mux,2
R2
D
Q
En
Clk
Clk
Mux,6
Mux,3
coeff
*
(MULT,2)
R3
D
Q
En
Clk
Clk
Mux,4
Clk
Clk
Power = t10-t9

ACA

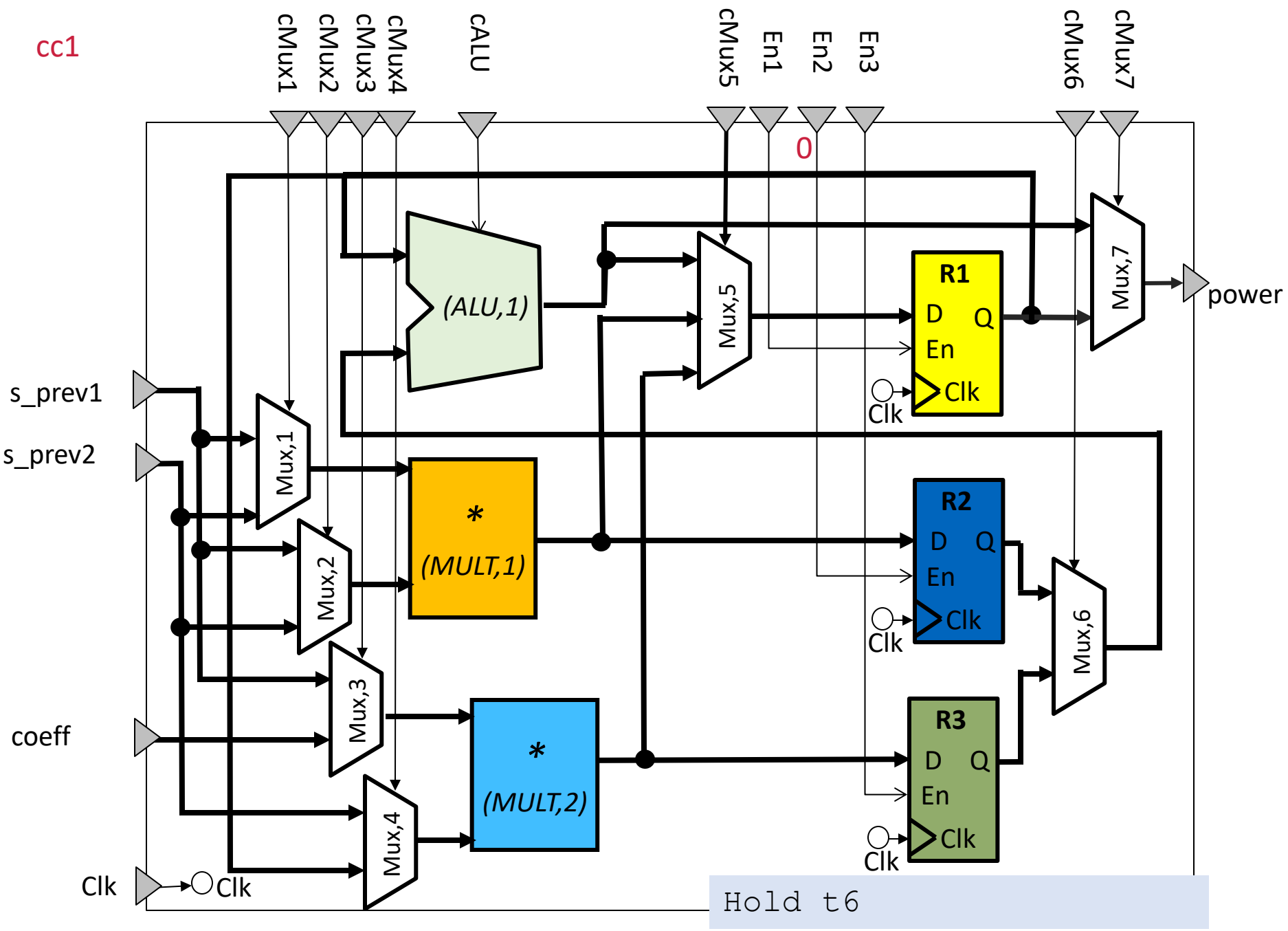
Activation Signals For Operations

- Example: SGU for basic block B3 of Goertzel algorithm

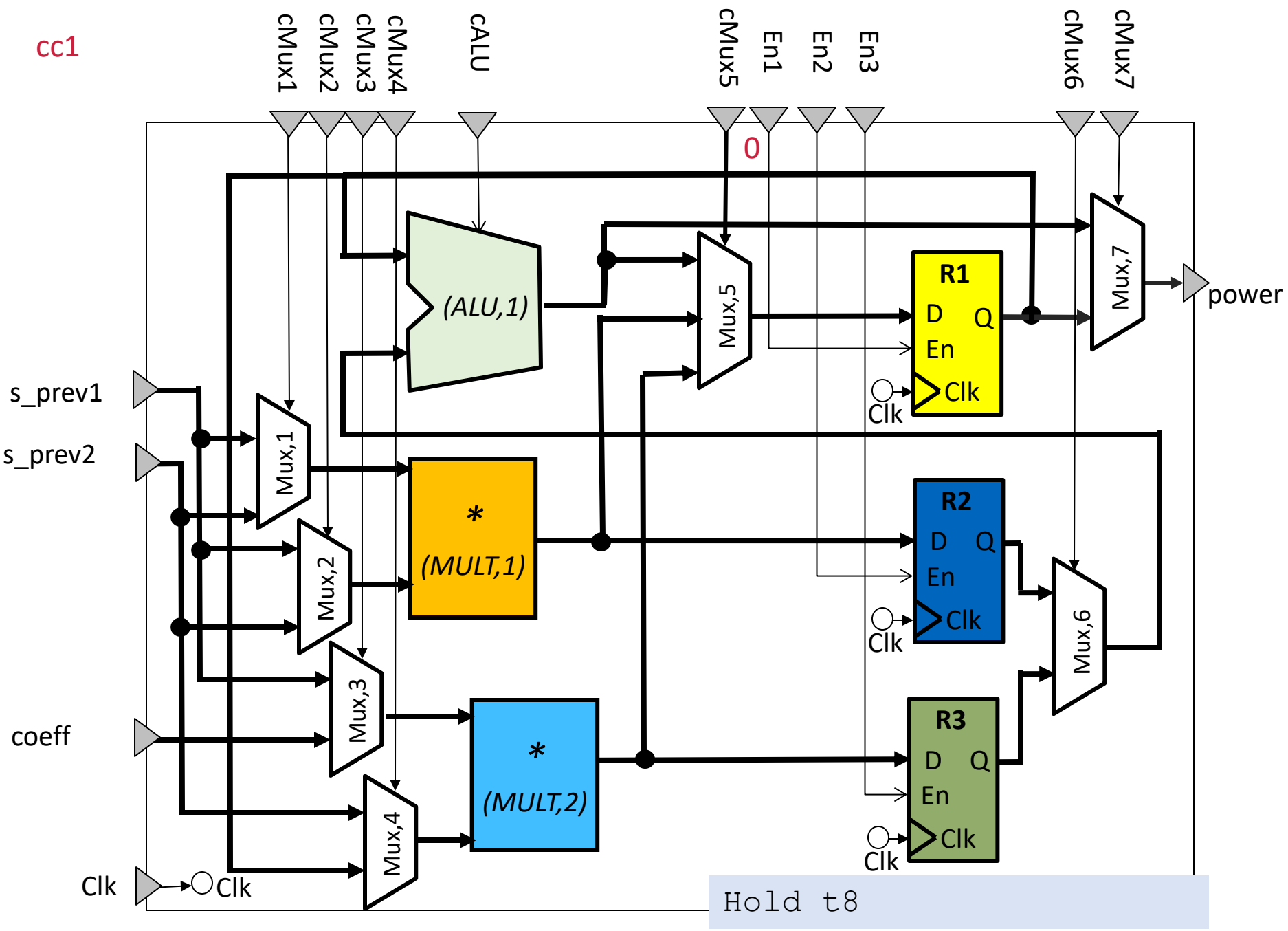
SG node	Multi-Cycle	Operation	En1	En2	En3	cMux1	cMux2	cMux3	cMux4	cMux5[1]	cMux5[0]	cMux6	cMux7	cALU
$v1$	MC1 MC2	`t6=s_prev1*s_prev1`		1		0 0	0 0							
v_2	MC1 MC2	`t8=s_prev1*s_prev2`	1					0 0	0 0	1	0			
v_3	MC1 MC2	`t7=s_prev2*s_prev2`	1			1 1	1 1			0	1			
v_4		`t10=t6+t7`	1							0	0	0		0
v_5	MC1 MC2	`t9=coeff*t7`			1			1 1	1 1					
v_6		`power=t10-t9`	1							0	0	1	0	1

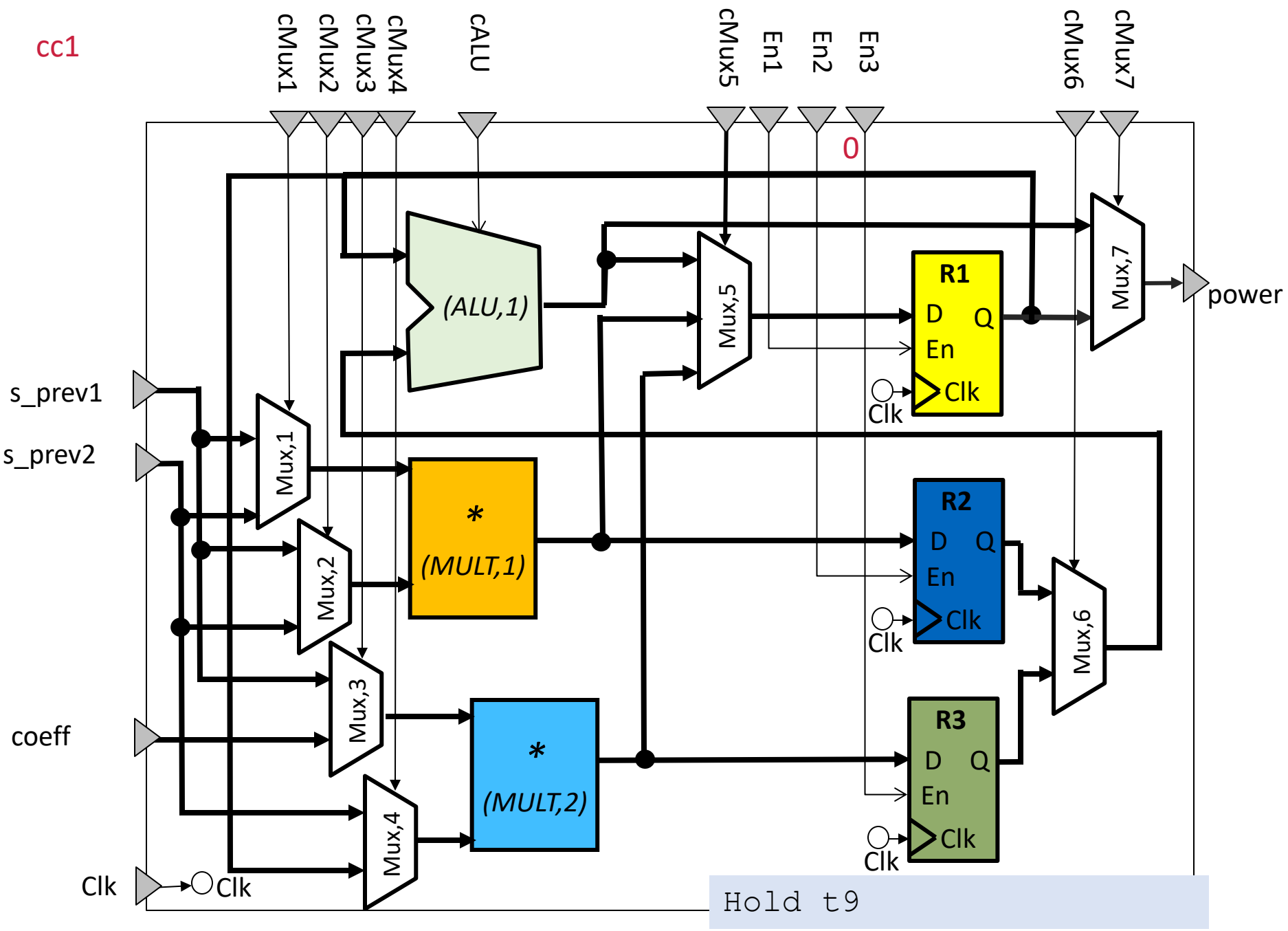
Activation Signals for Hold and Read

- Read activation signal is used to enter input values into the register.
- Hold signal is used to keep value of variable in register.

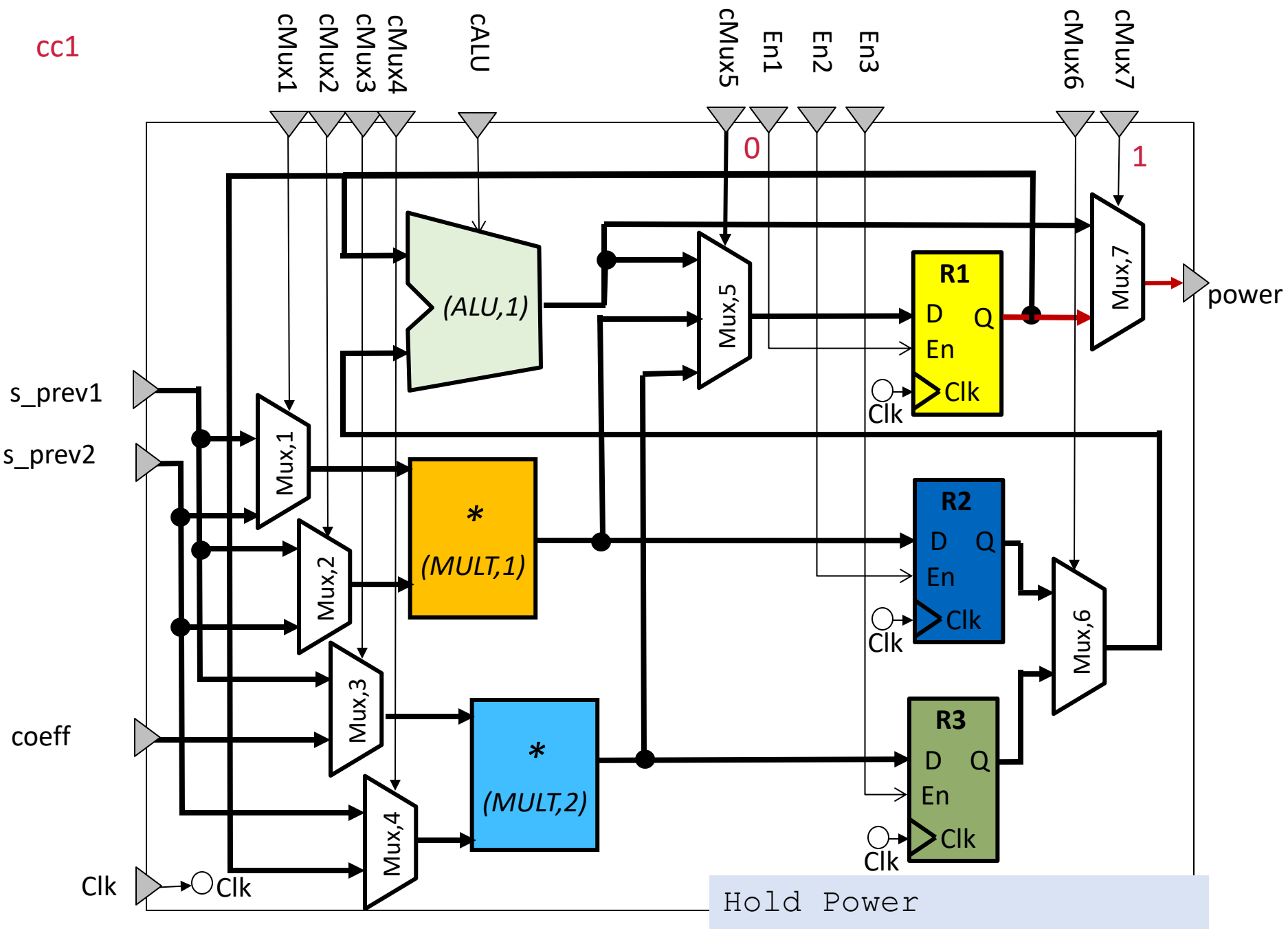

cc1
cMux1
cMux2
cMux3
cMux4
cALU
cMux5
En1
En2
En3
cMux6
cMux7
0
(ALU,1)
Mux,5
R1
D
Q
En
Clk
Clk
Mux,7
power
s_prev1
s_prev2
Mux,1
*
(MULT,1)
Mux,2
R2
D
Q
En
Clk
Clk
Mux,6
Mux,3
coeff
*
(MULT,2)
R3
D
Q
En
Clk
Clk
Mux,4
Clk
Clk
Hold t6

ACA


cc1
cMux1
cMux2
cMux3
cMux4
cALU
cMux5
En1
En2
En3
cMux6
cMux7
0
(ALU,1)
Mux,5
R1
D
Q
En
Clk
Clk
Mux,7
power
s_prev1
s_prev2
Mux,1
*
(MULT,1)
Mux,2
R2
D
Q
En
Clk
Clk
Mux,6
Mux,3
coeff
*
(MULT,2)
R3
D
Q
En
Clk
Clk
Mux,4
Clk
Clk
Hold t8


cc1
cMux1
cMux2
cMux3
cMux4
cALU
cMux5
En1
En2
En3
cMux6
cMux7
0
(ALU,1)
Mux,5
R1
D
Q
En
Clk
Clk
Mux,7
power
s_prev1
s_prev2
Mux,1
*
(MULT,1)
Mux,2
R2
D
Q
En
Clk
Clk
Mux,6
Mux,3
coeff
*
(MULT,2)
R3
D
Q
En
Clk
Clk
Mux,4
Clk
Clk
Hold t9

ACA


cc1
cMux1
cMux2
cMux3
cMux4
cALU
cMux5
En1
En2
En3
cMux6
cMux7
0
1
(ALU,1)
Mux,5
R1
D
Q
En
Clk
Mux,7
power
s_prev1
s_prev2
Mux,1
Mux,2
*
(MULT,1)
R2
Mux,6
coeff
Mux,3
*
(MULT,2)
R3
Mux,4
Clk
Hold Power

ACA

Activation Signals for Hold and Read

- Example: SGU for basic block B3 of Goertzel algorithm

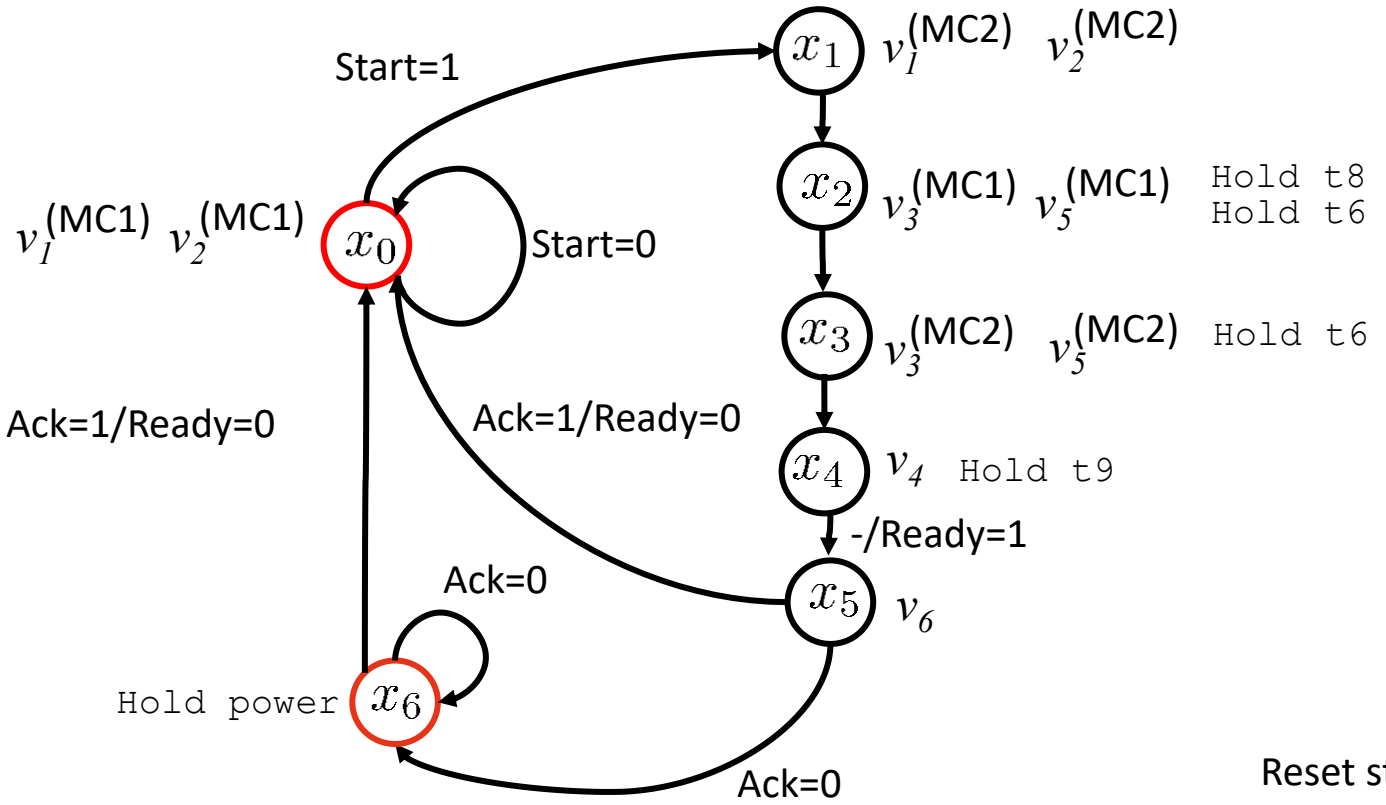
Operation	En1	En2	En3	cMUX7
Hold t6		0		
Hold t8	0			
Hold t9			0	
Hold power	0			1

FSM with Data Specification

- The FSM with data specification describes the schedule and the control flow of the data path and the HW module interface.
- The operations of the data path are assigned to the states of the FSMD.
- The FSMD has one state transition for each clock cycle.
- Transitions may depend on status signals from the data path or interface.

FSM with Data Specification (FSMD)

- Example: SGU for basic block B3 of Goertzel algorithm



Reset state transitions omitted.

State Assignment

- Number of state variables: n_{bit}
- Number of possible states: $|X| = 2^{n_{bit}}$
- Minimal number of state variables: $n_{bit} = \lceil \log_2 |X| \rceil$
- State coding:

State	Binary	One-hot	Almost one-hot
x_0	000	00001	0000
x_1	001	00010	0001
x_2	010	00100	0010
x_3	011	01000	0100
x_4	100	10000	1000

State Assignment

- Example: SGU for basic block B3 of Goertzel algorithm

State name	State variables
x_0	000
x_1	001
x_2	010
x_3	011
x_4	100
x_5	101
x_6	110

Next-State and Output Logic

- Output logic:
  - Inputs: State variables
  - Output: Activation signals for the operations

- Next state logic:
  - Inputs: State variables and status variables
  - Outputs: State variables in next state

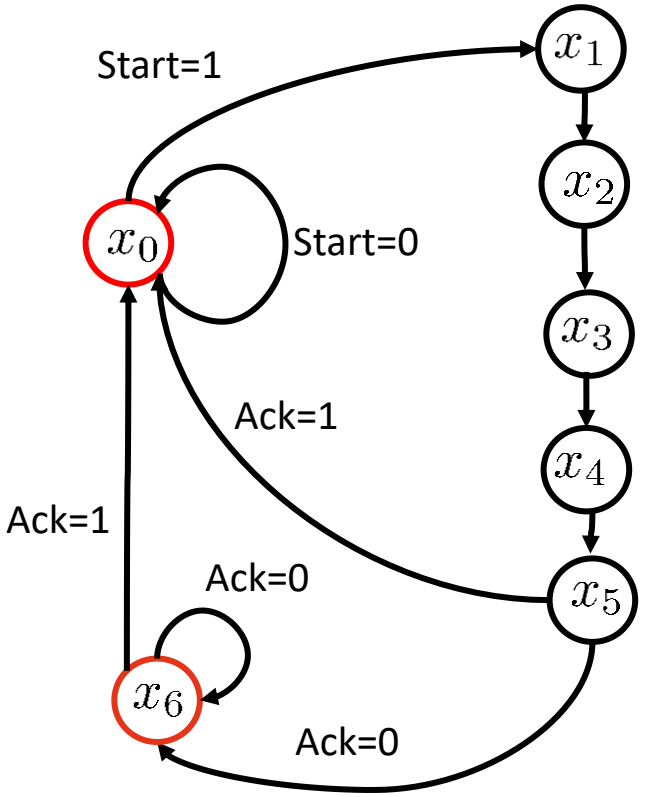
Next-State and Output Logic

- Example: SGU for basic block B3 of Goertzel algorithm

State variables (SV)	Ready	En1	En2	En3	cMux1	cMux2	cMux3	cMux4	cMux5[1]	cMux5[0]	cMux6	cMux7	cALU
000	0	DC	DC	DC	0	0	0	0	DC	DC	DC	DC	DC
001	0	1	1	DC	0	0	0	0	1	0	DC	DC	DC
010	0	0	0	DC	1	1	1	1	DC	DC	DC	DC	DC
011	0	1	0	1	1	1	1	1	0	1	DC	DC	DC
100	0	1	DC	0	DC	DC	DC	DC	0	0	0	DC	0
101	1	1	DC	DC	DC	DC	DC	DC	0	0	1	0	1
110	1	0	DC	DC	DC	DC	DC	DC	DC	DC	DC	1	DC

Next-State and Output Logic

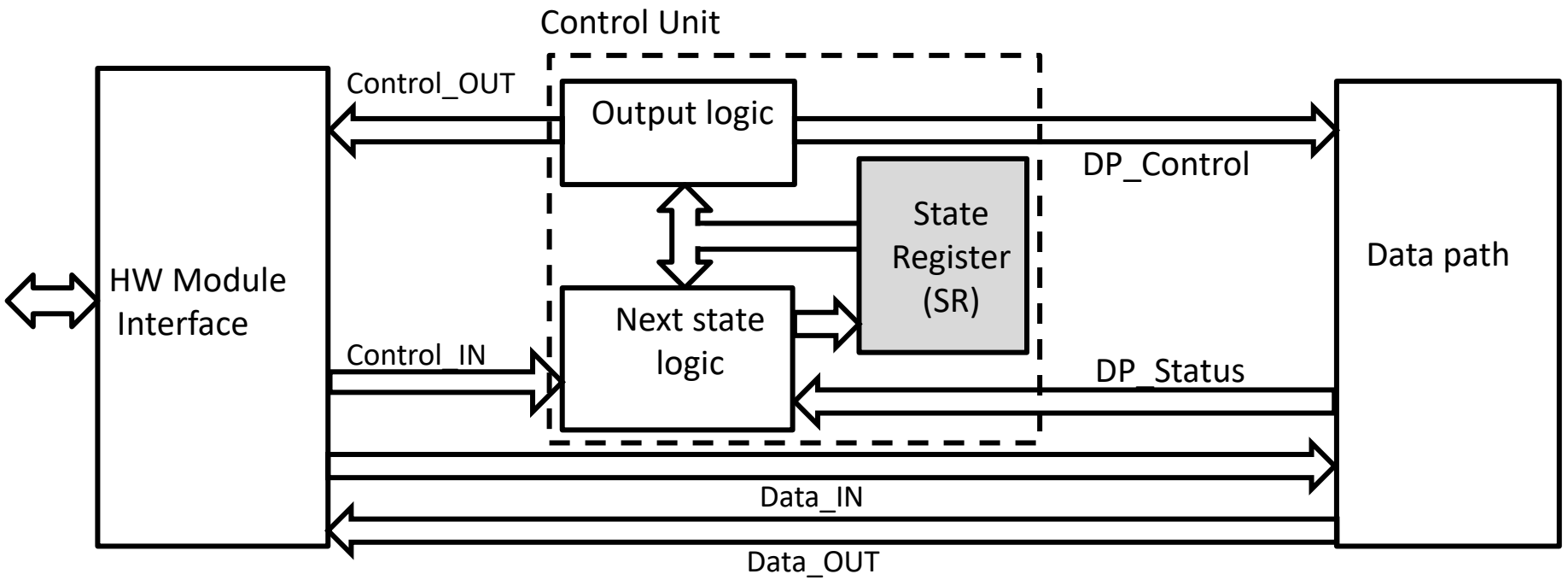
- Example: SGU for basic block B3 of Goertzel algorithm (Reset transitions not shown in state transition diagram are includes in next state logic)



Start	Reset	Ack	SV	SV next
0	0	X	000	000
1	0	X	000	001
X	0	X	001	010
X	0	X	010	011
X	0	X	011	000
X	0	X	100	101
X	0	0	101	110
X	0	1	101	000
X	0	0	110	110
X	0	1	110	000
X	1	X	XXXX	000

FSM Implementation

- Output/next state logic implemented as combinatorial circuit



FSM Implementation

- Example: SGU for basic block B3 of Goertzel algorithm