Süsteemide diagnostika

2. Teoreetilised alused

2.1. Boole’i differentsiaalalgebra

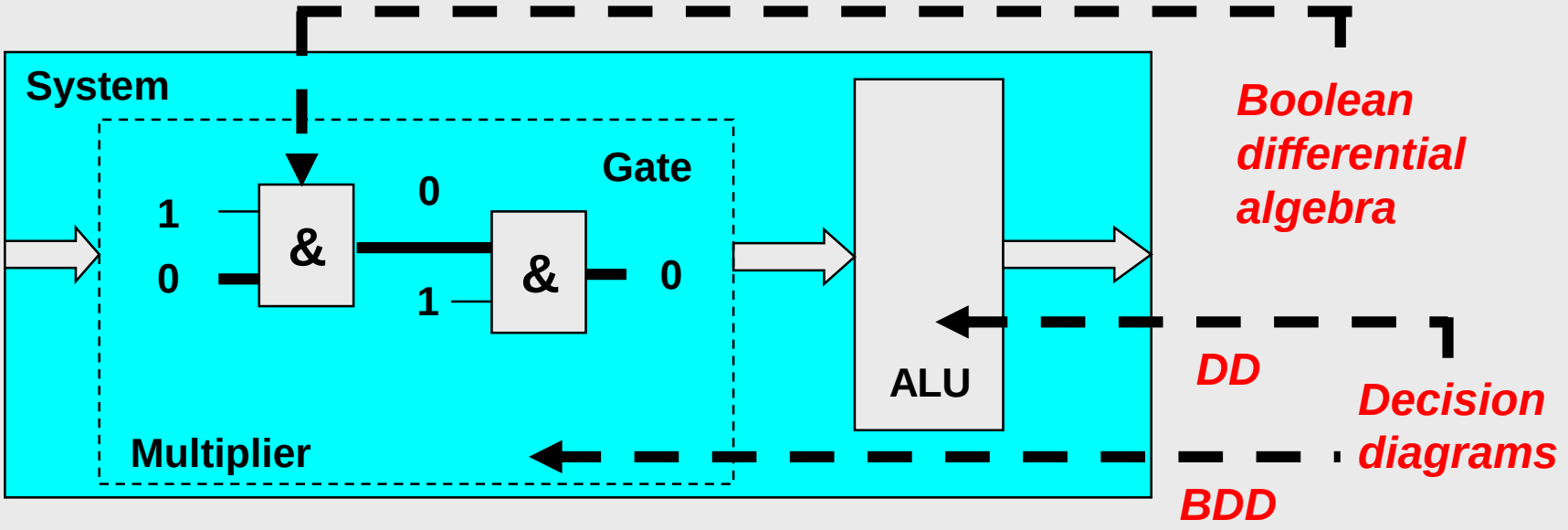
2.2. Binaarsed otsustusdiagrammid (BDD)

2.3. Kõrgtasandi otsustusdiagrammid

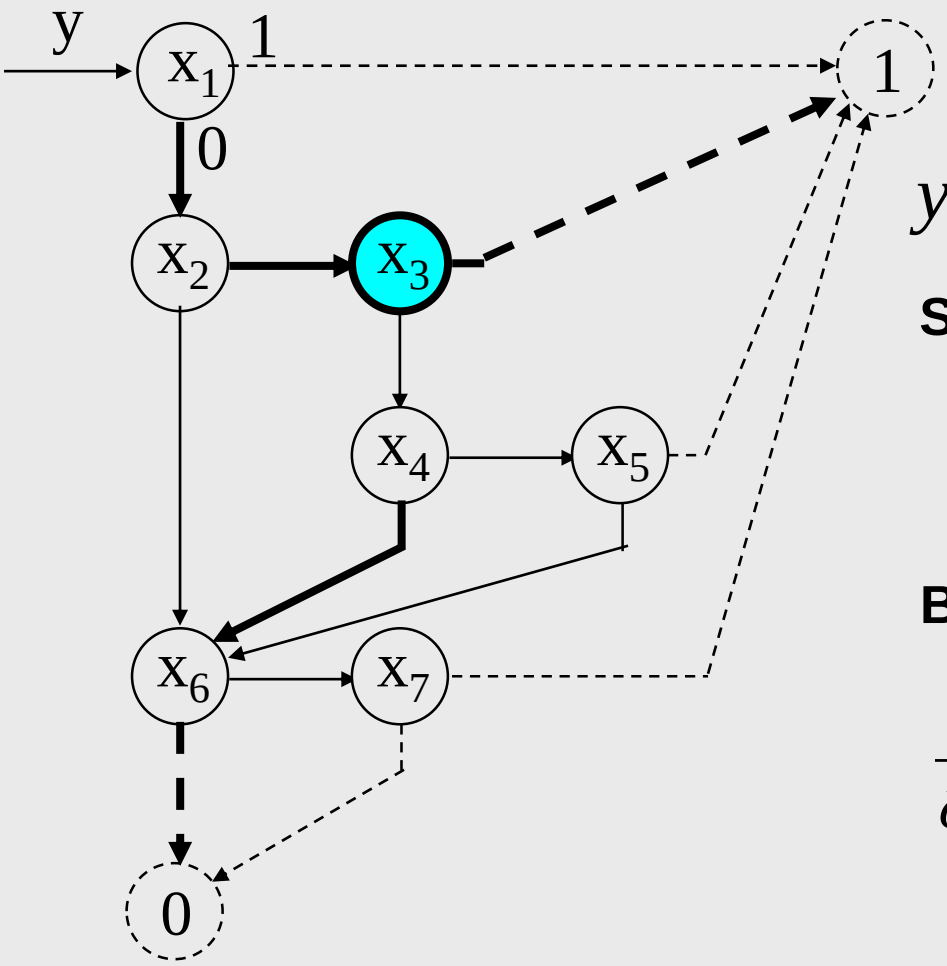
How to Go Beyond the Boolean World?

Two basic tasks:

1. **Which test patterns are needed to detect a fault (or all faults)**
2. **Which faults are detected by a given test (or by all tests)**



Logic Level BDDs



Functional BDD

$$y = x_1 \vee x_2(x_3 \vee x_4 x_5) \vee x_6 x_7$$

Simulation:

x_1	x_2	x_3	x_4	x_5	x_6	x_7		
0	1	1	0	1	0	0	$\longrightarrow$	$y = 1$

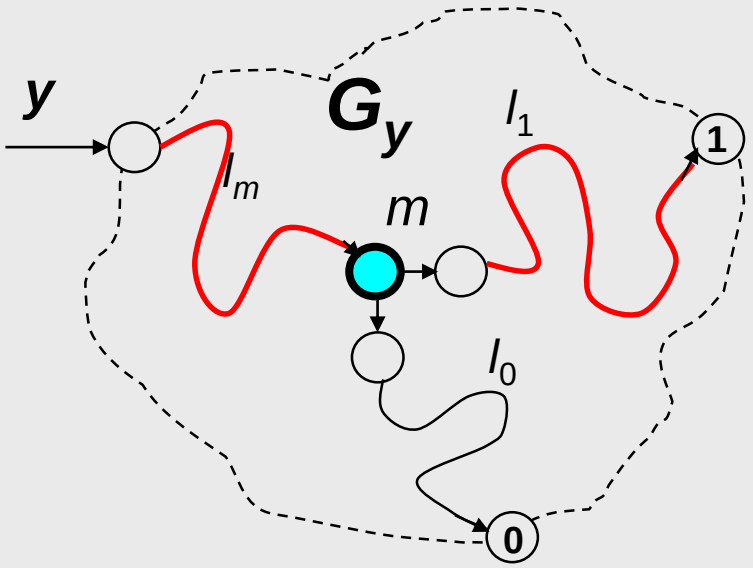
Boolean derivative:

$$\frac{\partial y}{\partial x_3} = \overline{x_1 \vee x_6 x_7 x_2 \, \overline{x_4 x_5}} = 1$$

Generalization of BDDs

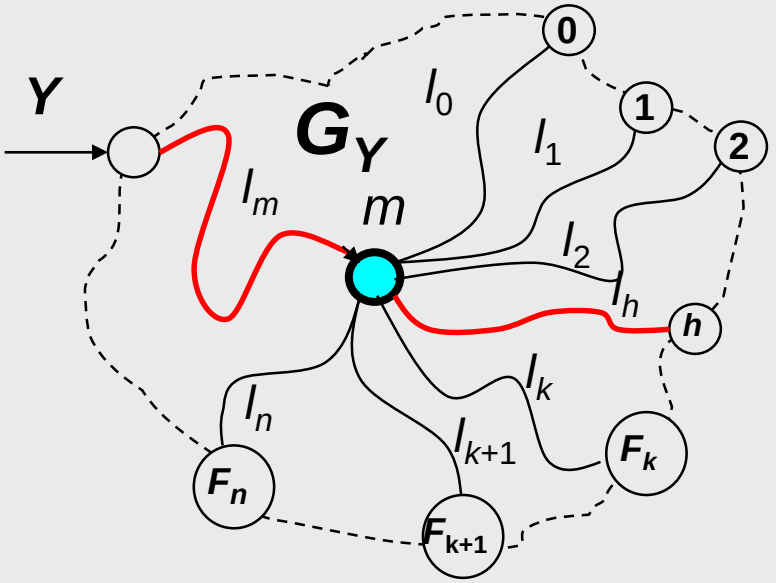
Binary DD

2 terminal nodes and
2 edges from each node



General case of DD

n ≥ 2 terminal nodes and
n ≥ 2 edges from each node



Novelty: **Boolean methods can be generalized in a straightforward way to higher functional levels**

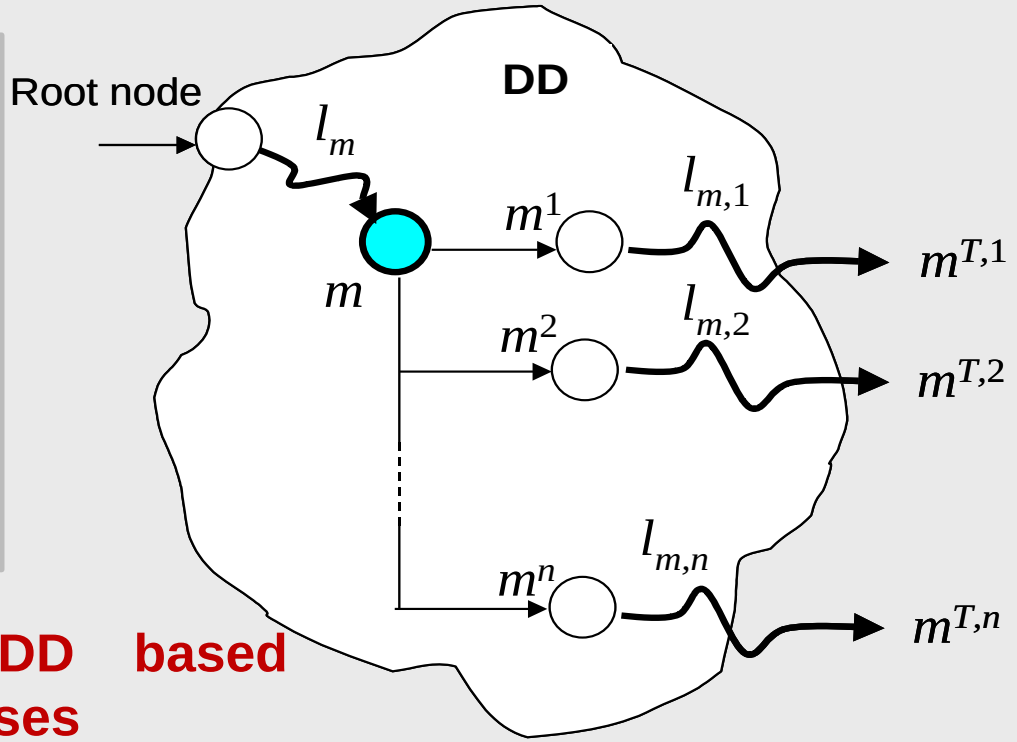
HLDDs and Faults

Each path in a DD represents a working mode of the system
The faults having effect on the behaviour can be associated with nodes along the path

D1: node output is **always activated (SAF-1)**

D2: node output is **broken (SAF-0)**

D3: instead of the given output **another output set** is **activated**



Instead of the 14 fault classes,the HLDD based fault model includes only 3 fault classes

HLDDs and Faults

RTL-statement:

K: (*If* T,C) $R_D \leftarrow F(R_{S1},R_{S2},\ldots R_{Sm}), \rightarrow N$

Nonterminal nodes

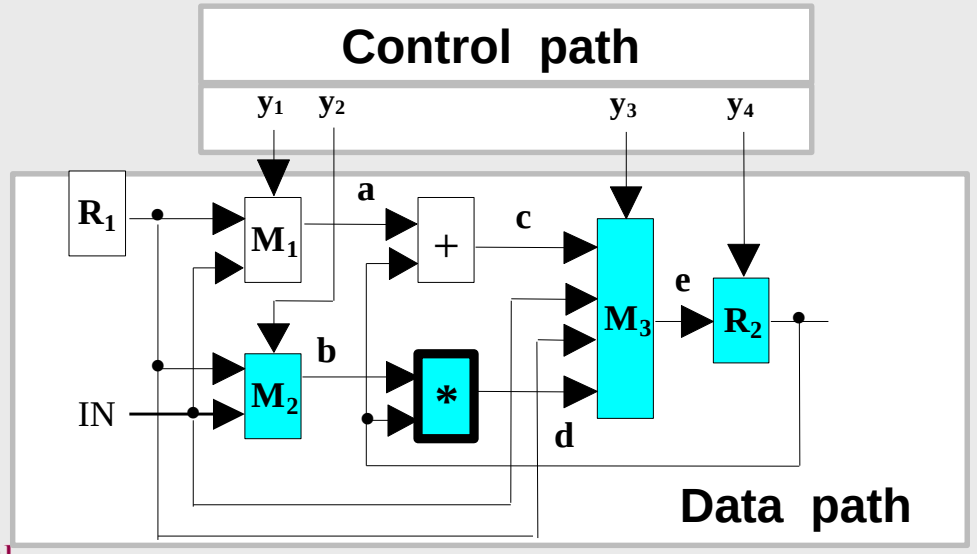
RTL-statement faults:

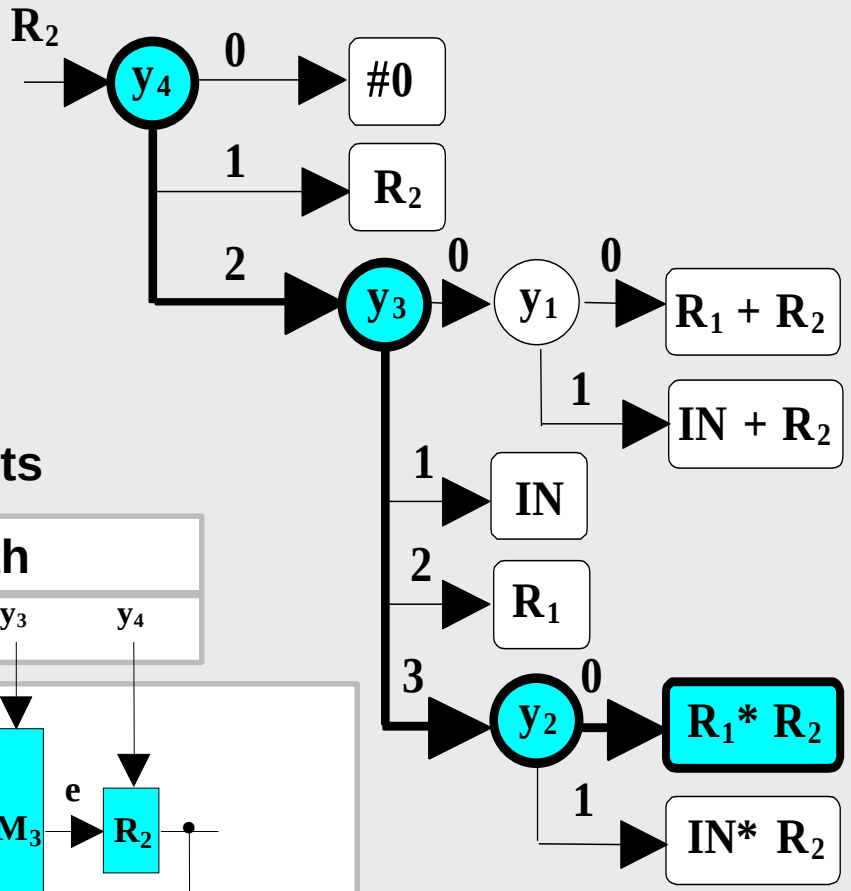
label,
timing condition,
logical condition,
register decoding,
operation decoding,
control faults

Terminal nodes

RTL-statement faults:

data storage,
data transfer,
data manipulation faults



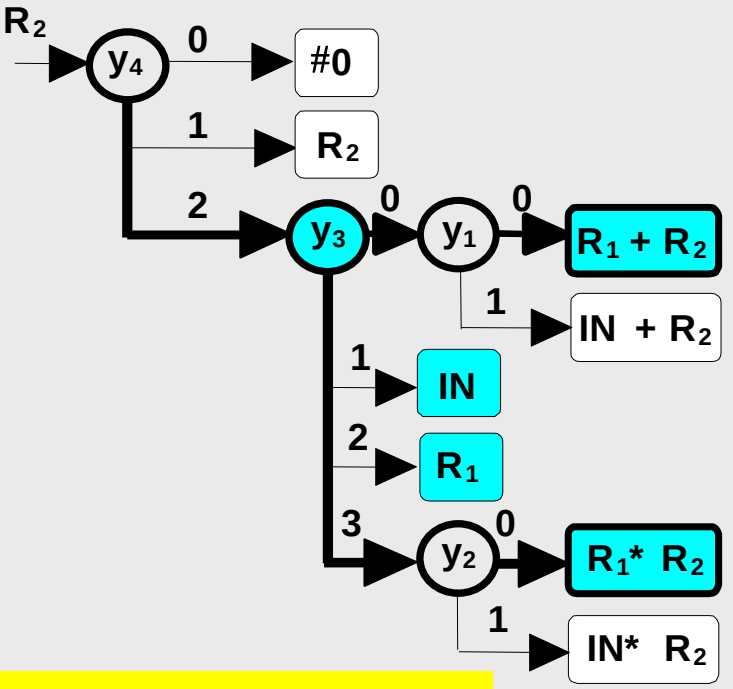


Test Generation in Digital Systems

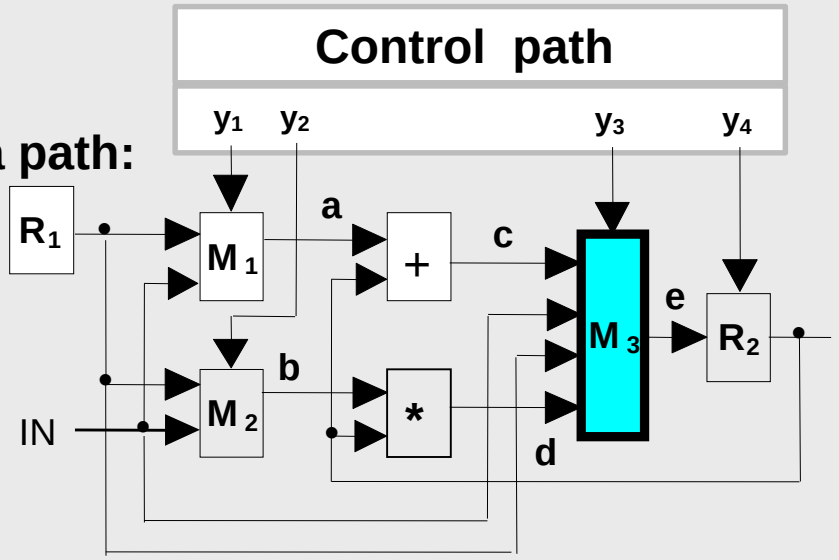
Multiple paths activation in a DD

Control function y_3 is tested

Decision Diagram



Data path:



Test cycle:

Control: *For* **D = 0,1,2,3:** $y_1\, y_2\, y_3\, y_4 = 00D2$

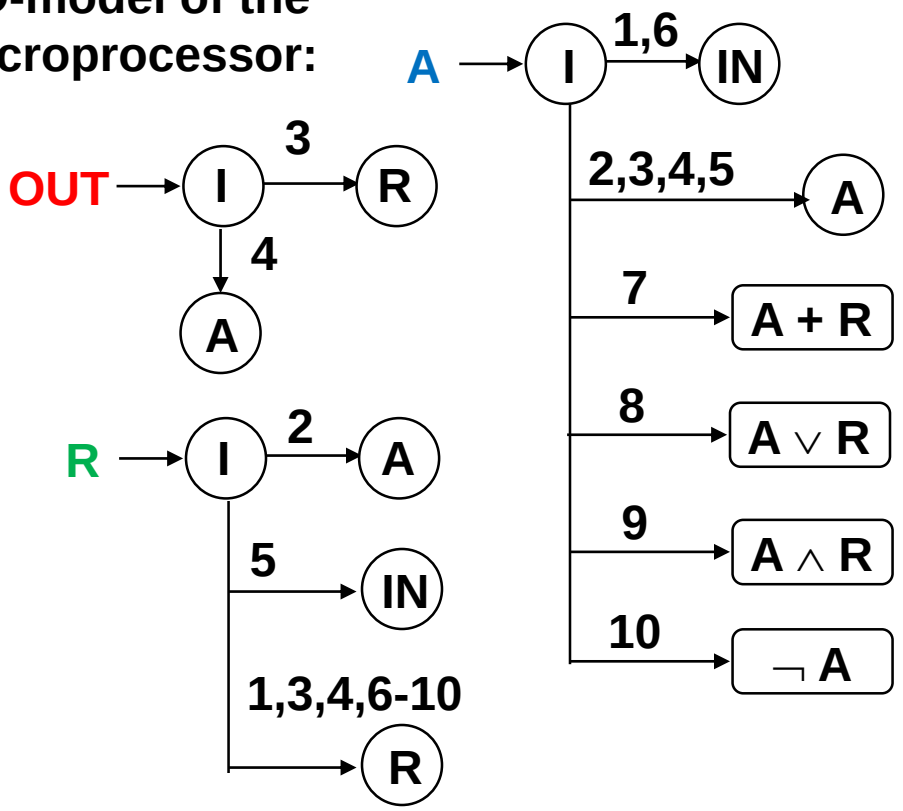
Data: *Solution of* $R_1 + R_2 \neq IN \neq R_1 \neq R_1^* R_2$

Decision Diagrams for Microprocessors

Instruction set:

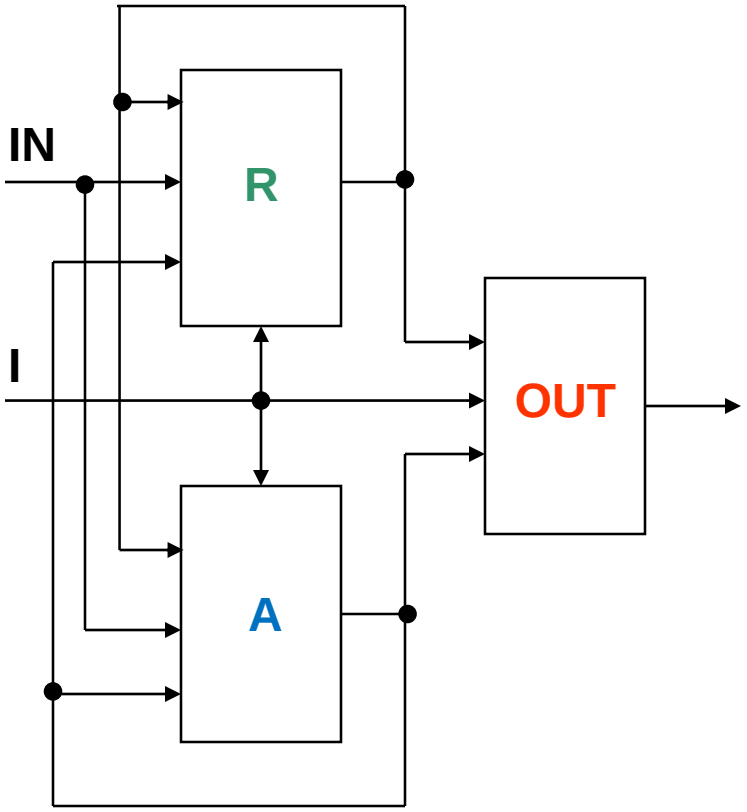
I_1:	MVI A,D	A ← IN
I_2:	MOV R,A	R ← A
I_3:	MOV M,R	OUT ← R
I_4:	MOV M,A	OUT ← A
I_5:	MOV R,M	R ← IN
I_6:	MOV A,M	A ← IN
I_7:	ADD R	A ← A + R
I_8:	ORA R	A ← A ∨ R
I_9:	ANA R	A ← A ∧ R
I_{10}:	CMA A,D	A ← ¬ A

DD-model of the microprocessor:

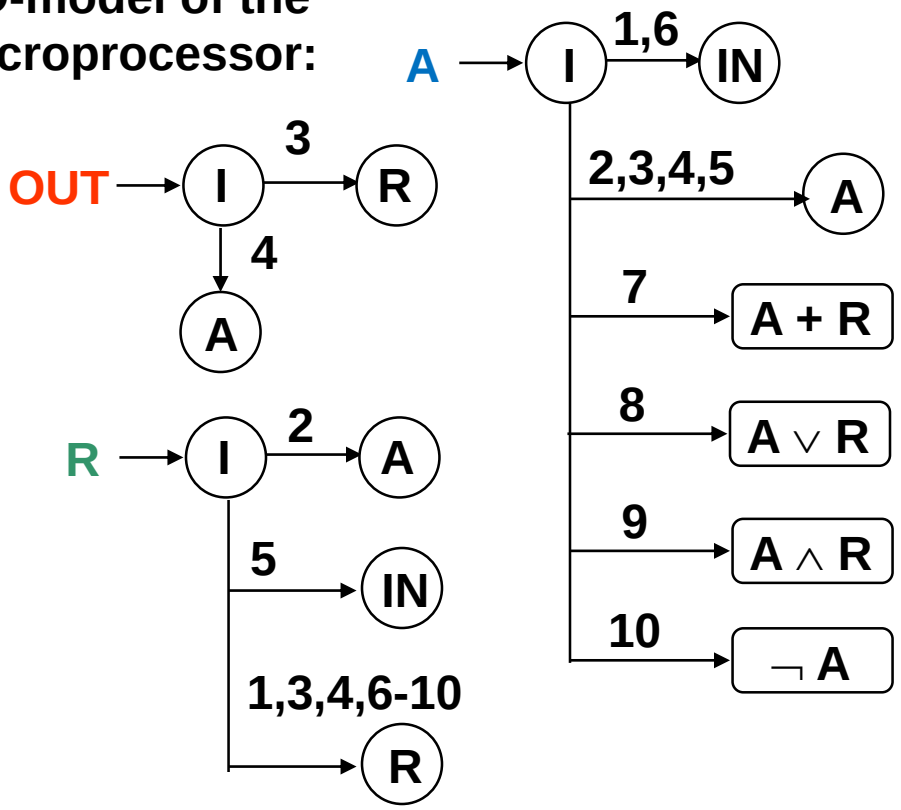


Decision Diagrams for Microprocessors

High-Level DD-based structure of the microprocessor (example):



DD-model of the microprocessor:



From MP Instruction Set to HLDDs

OP	B	Semantic	RT level operations	
0	0	READ memory	$R(A1) = M(A)$	$PC = PC + 2$
	1	WRITE memory	$M(A) = R(A2)$	$PC = PC + 2$
1	0	Transfer	$R(A1) = R(A2)$	$PC = PC + 1$
	1	Complement	$R(A1) = \neg R(A2)$	$PC = PC + 1$
2	0	Addition	$R(A1) = R(A1) + R(A2)$	$PC = PC + 1$
	1	Subtraction	$R(A1) = R(A1) - R(A2)$	$PC = PC + 1$
3	0	Jump	$PC = A$	
	1	Conditional jump	IF C=1, THEN PC = A,ELSE PC = PC + 2	

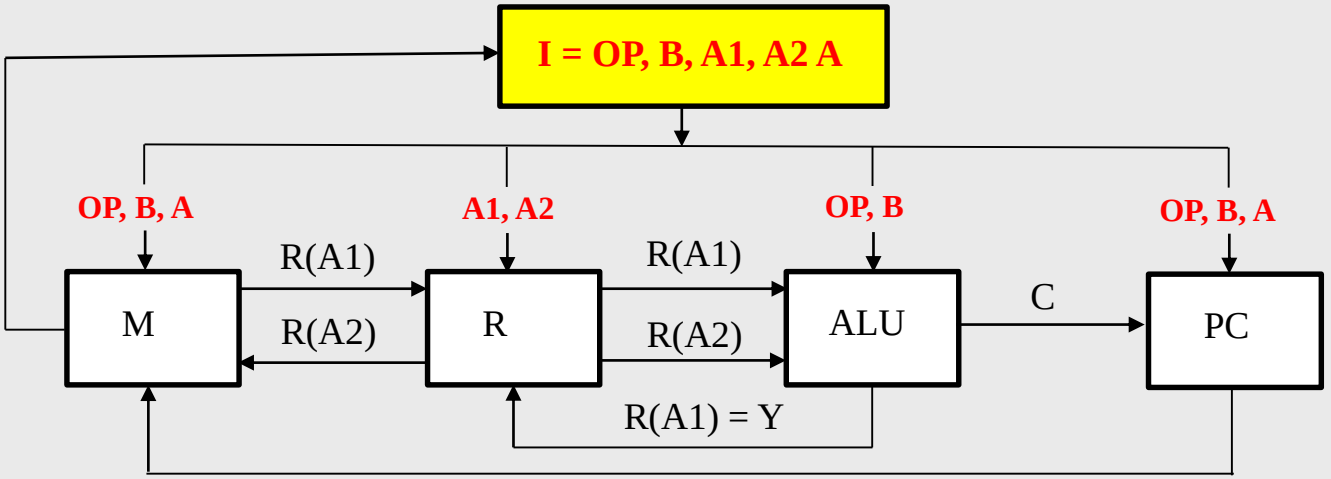
Instruction code:

ADD A1 A2

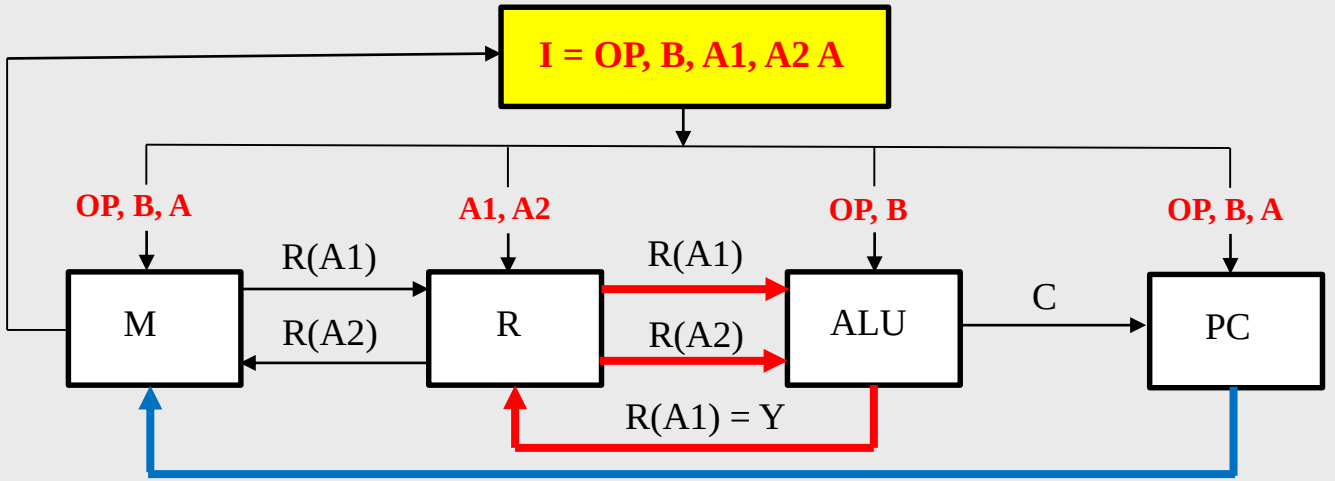
OP=2. B=0. A1=3. A2=2

$R_3 = R_3 + R_2$

$PC = PC+1$



From MP Instruction Set to HLDDs



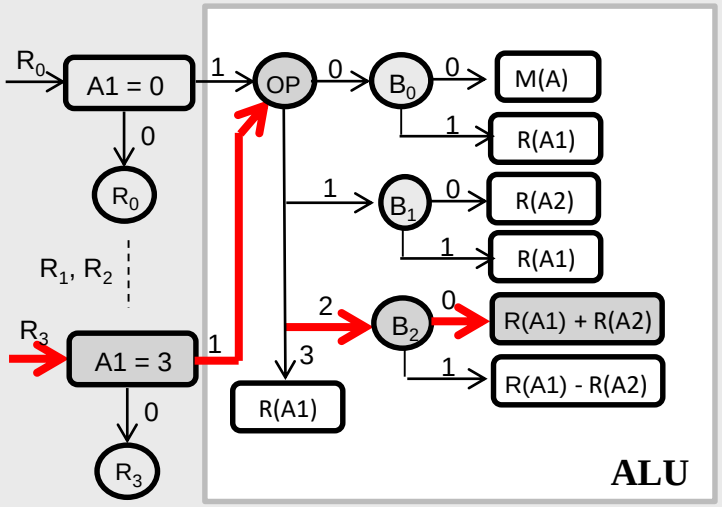
Instruction code:

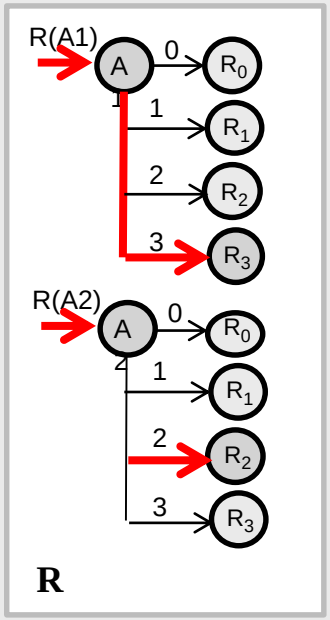
ADD A1 A2

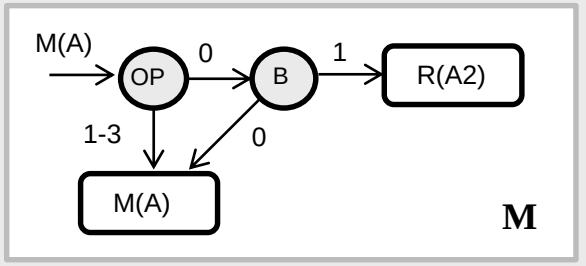
OP=2. B=0. A1=3. A2=2

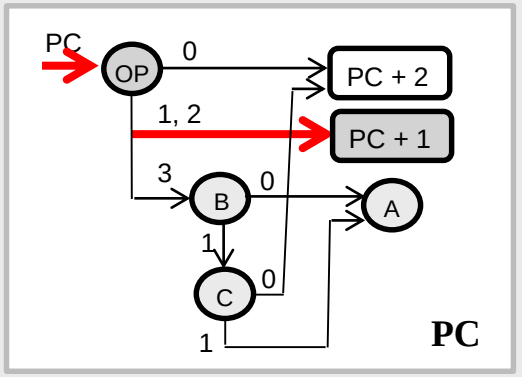
$R_3 = R_3 + R_2$

PC = PC+1









From MP Instruction Set to HLDDs

OP	B	Semantic	RT level operations	
0	0	READ memory	$R(A1) = M(A)$	$PC = PC + 2$
	1	WRITE memory	$M(A) = R(A2)$	$PC = PC + 2$
1	0	Transfer	$R(A1) = R(A2)$	$PC = PC + 1$
	1	Complement	$R(A1) = \neg R(A2)$	$PC = PC + 1$
2	0	Addition	$R(A1) = R(A1) + R(A2)$	$PC = PC + 1$
	1	Subtraction	$R(A1) = R(A1) - R(A2)$	$PC = PC + 1$
3	0	Jump	$PC = A$	
	1	Conditional jump	IF C=1, THEN PC = A,ELSE PC = PC + 2	

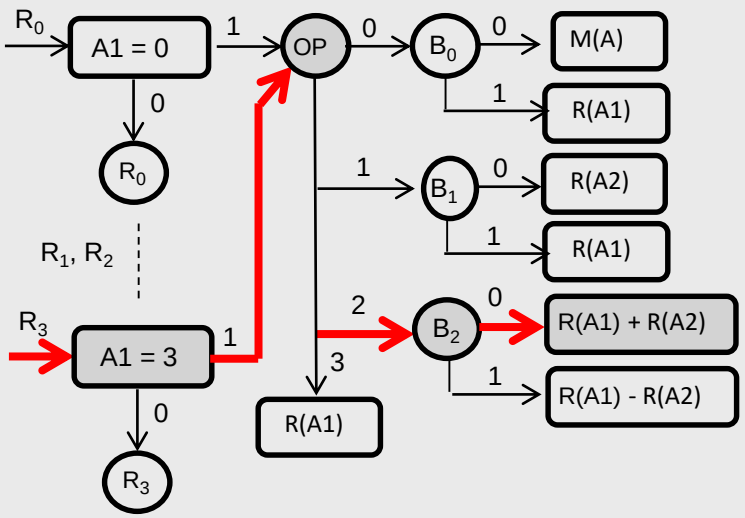
Instruction code:

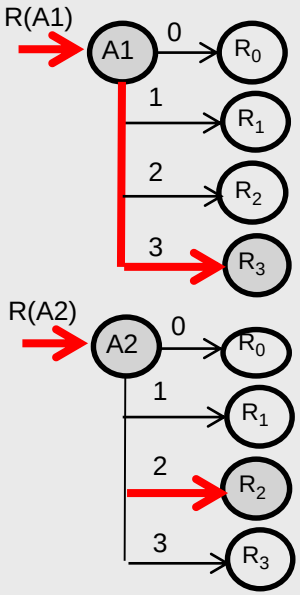
ADD A1 A2

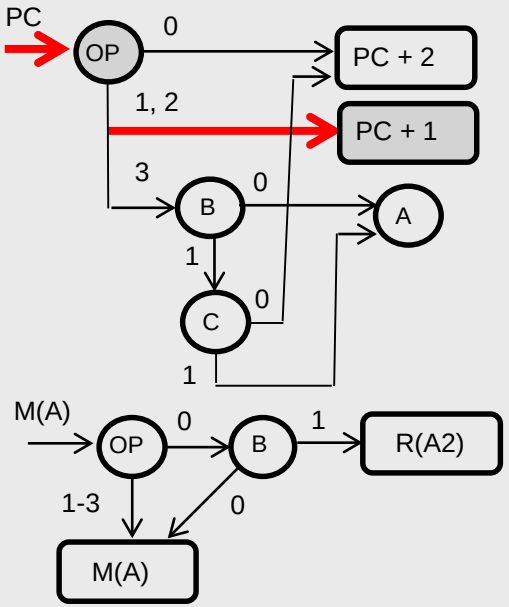
OP=2. B=0. A1=3. A2=2

$R_3 = R_3 + R_2$

PC = PC+1

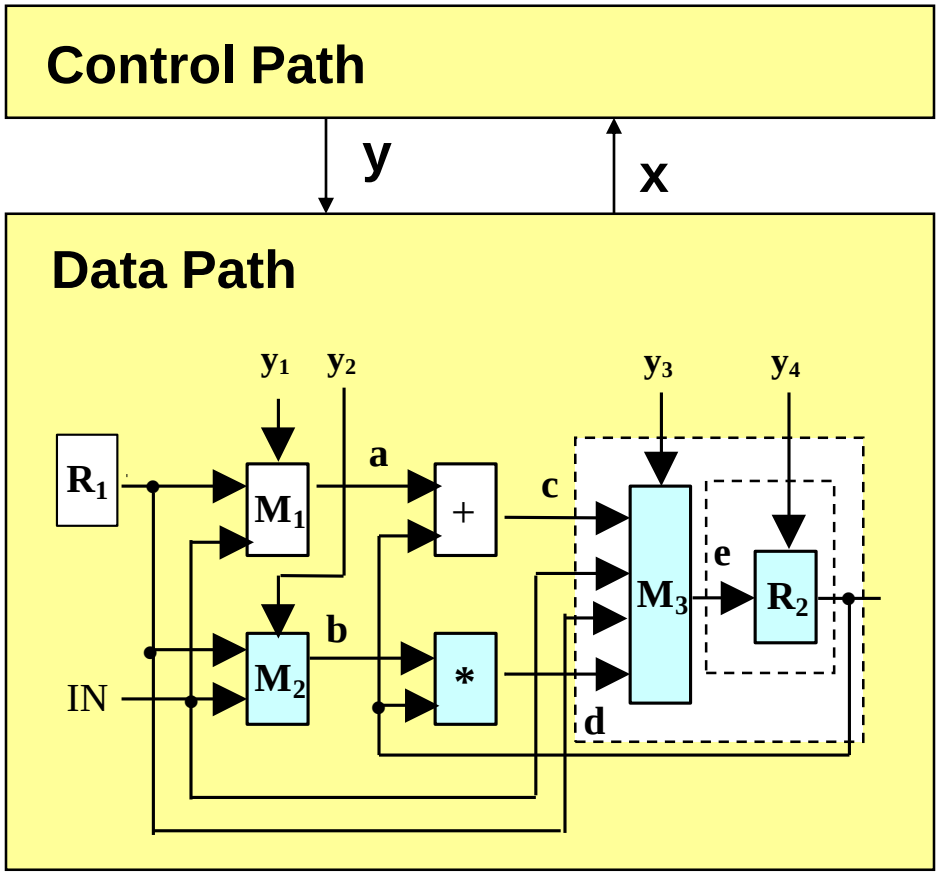






Data Path: Structural Synthesis of HLDDs

Structural description of the Data Path



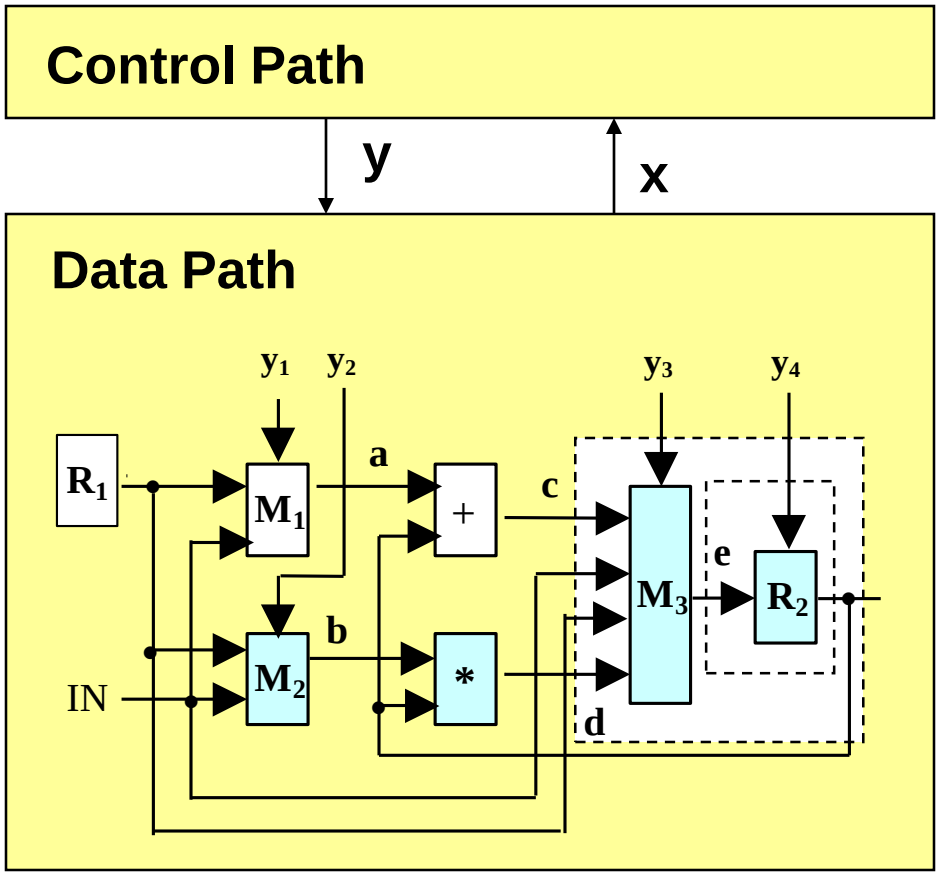
c (M1)	
y1	Function
0	M1 = R
1	M1 = IN

d (M2)	
y2	Function
0	M2 = R
1	M2 = IN

e (M3)	
y3	Function
0	M3 = M1 + R2
1	M3 = IN
2	M3 = R2
3	M3 = M2 * R2

R2		
y4	Operation	Function
0	Reset	R2 = 0
1	Hold	R2 = R2
2	Load	R2 = M3

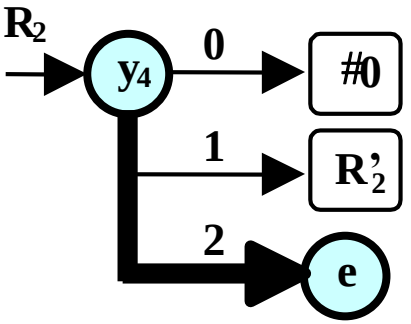
Data Path: Structural Synthesis of HLDDs



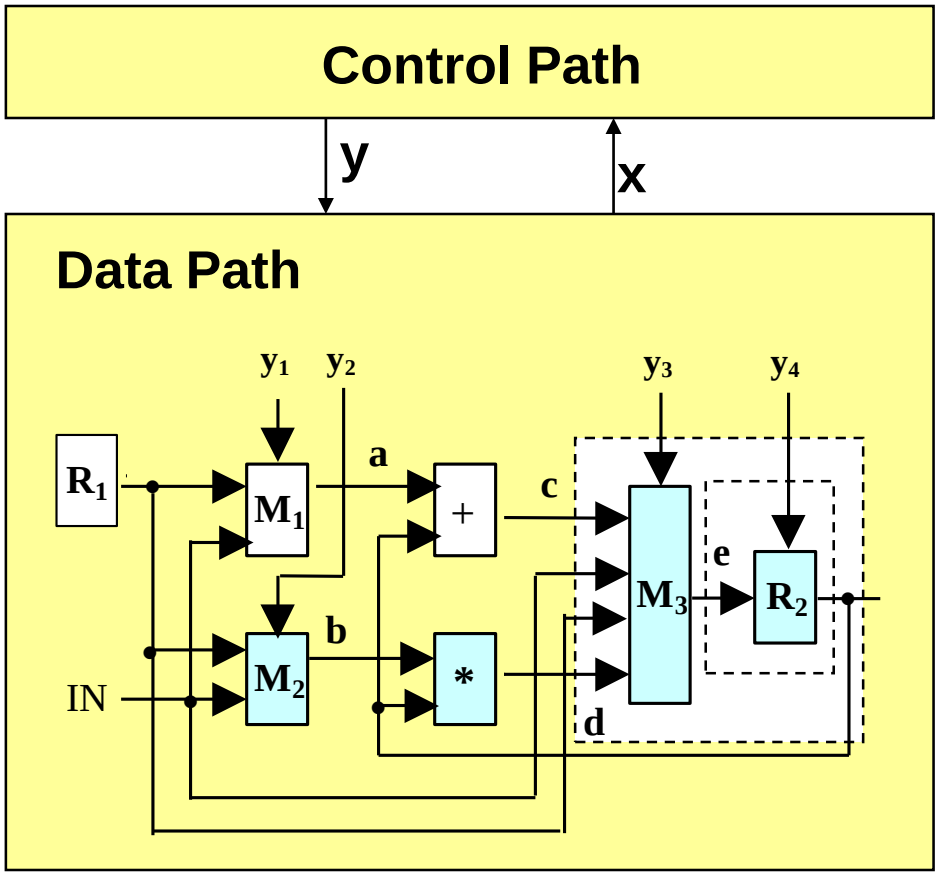
R_2		
y_4	Operatior	Functior
0	Reset	$R_2 = 0$
1	Hold	$R_2 = R'_2$
2	Load	$R_2 = M_3$

Start of the synthesis:

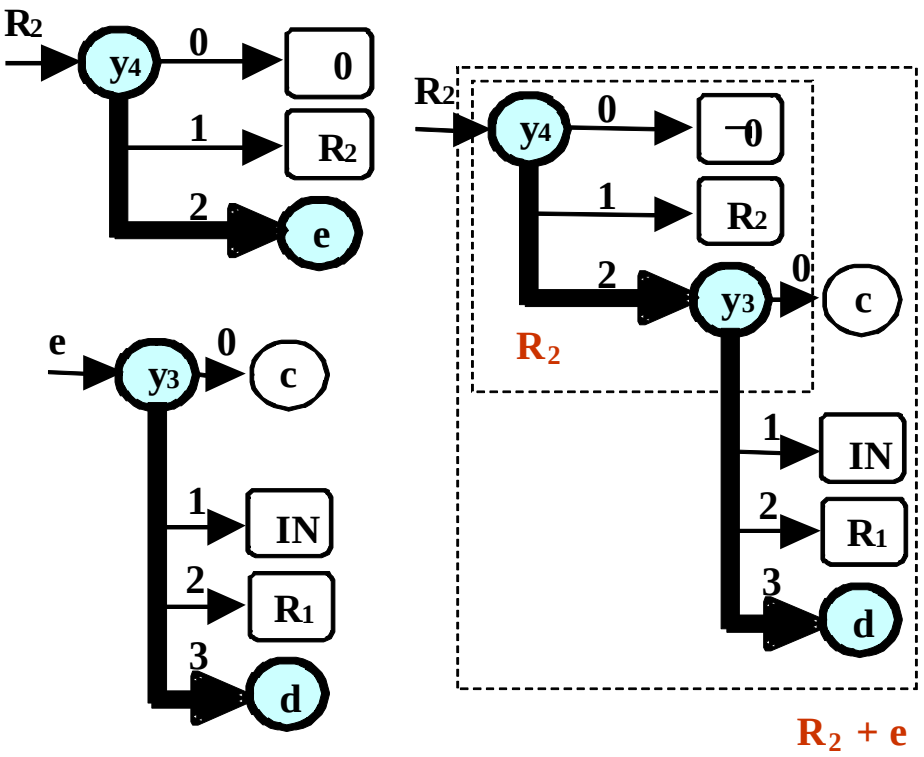
Creation of the HLDD for the output subcircuit



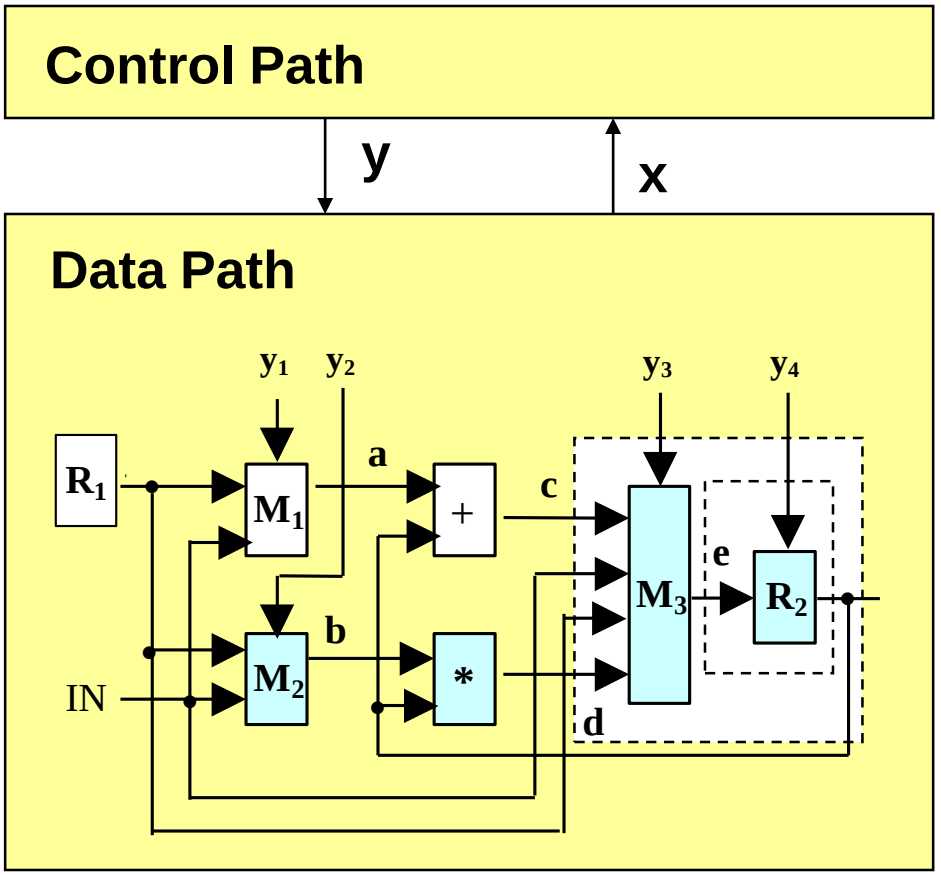
Data Path: Structural Synthesis of HLDDs



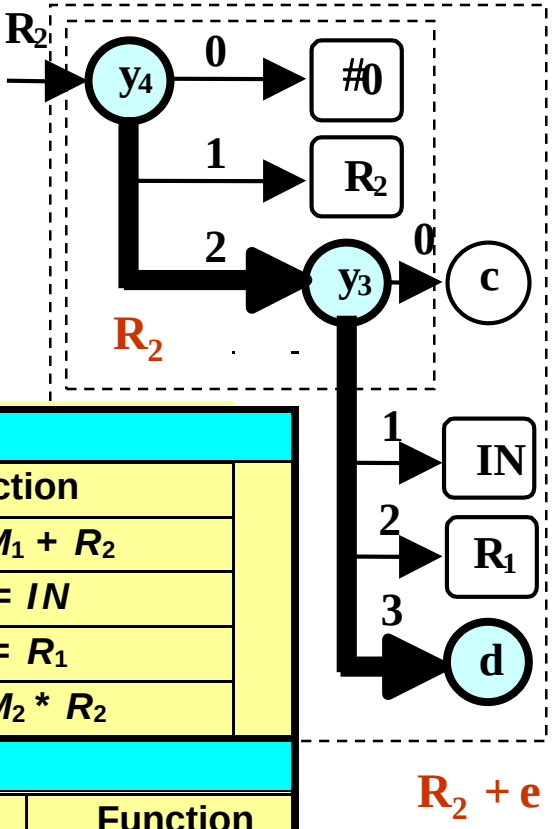
Superposition of the HLDDs



Data Path: Structural Synthesis of HLDDs



Superposition of the HLDDs



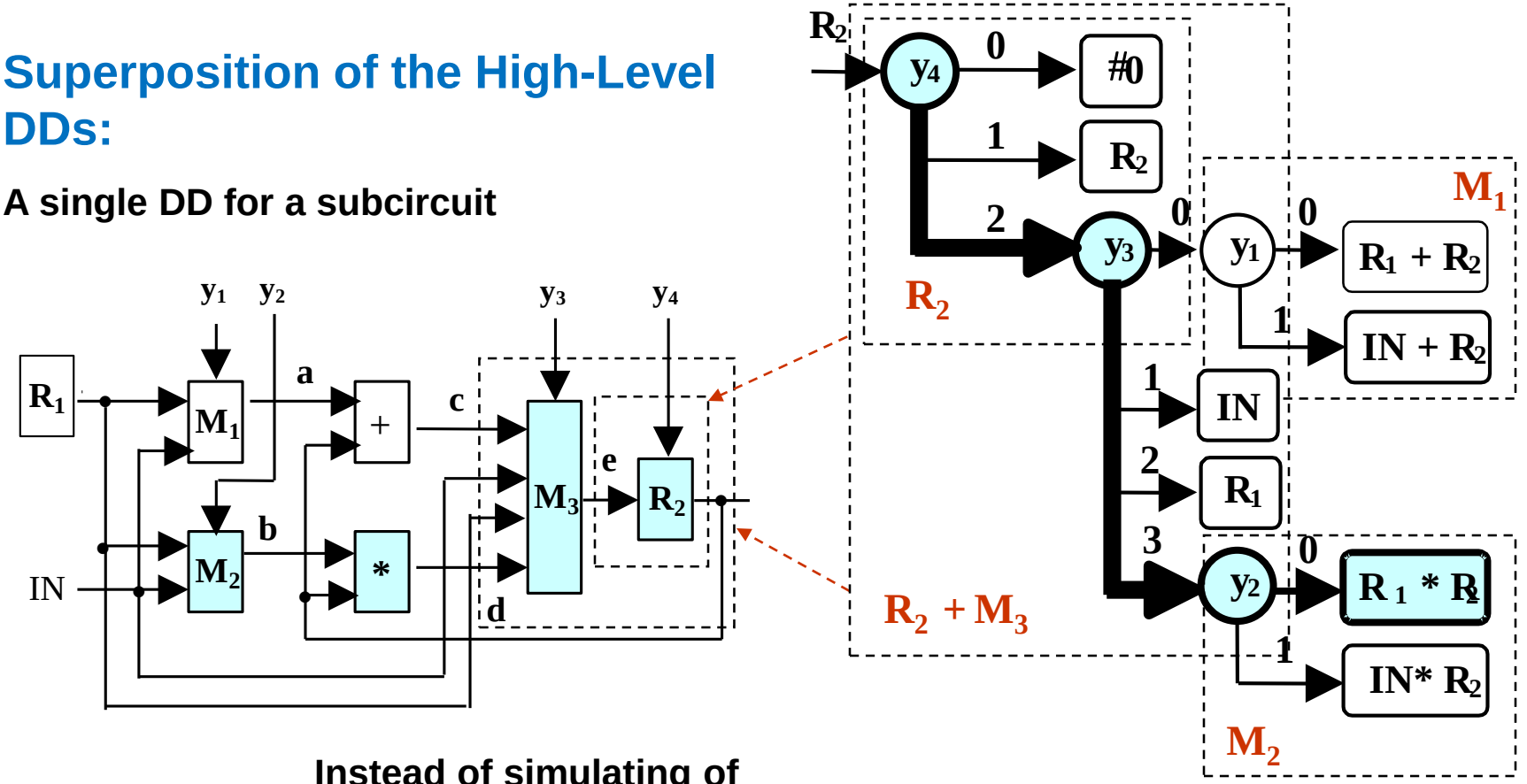
M_3	
y_3	Function
0	$M_3 = M_1 + R_2$
1	$M_3 = IN$
2	$M_3 = R_1$
3	$M_3 = M_2 * R_2$

R_2		
y_4	Operation	Function
0	Reset	$R_2 = 0$
1	Hold	$R_2 = R'_2$
2	Load	$R_2 = M_3$

Mapping Between a Circuit and HLDDs

Superposition of the High-Level DDs:

A single DD for a subcircuit



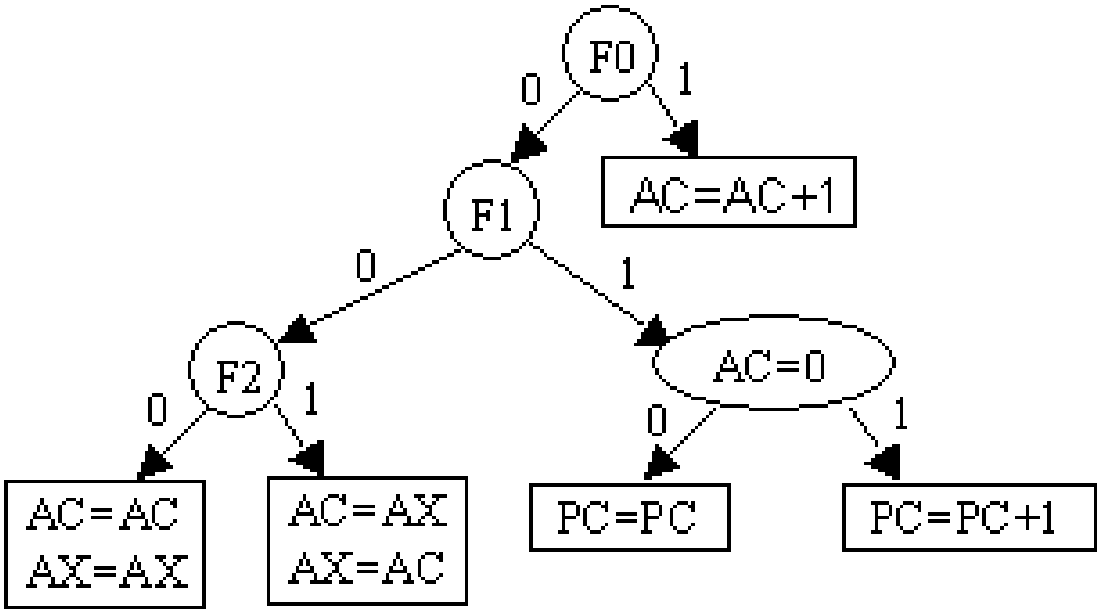
Instead of simulating of all the components in the circuit, only a single path in the DD should be traced

Functional Synthesis of High-Level DDs

Data-Flow Diagram

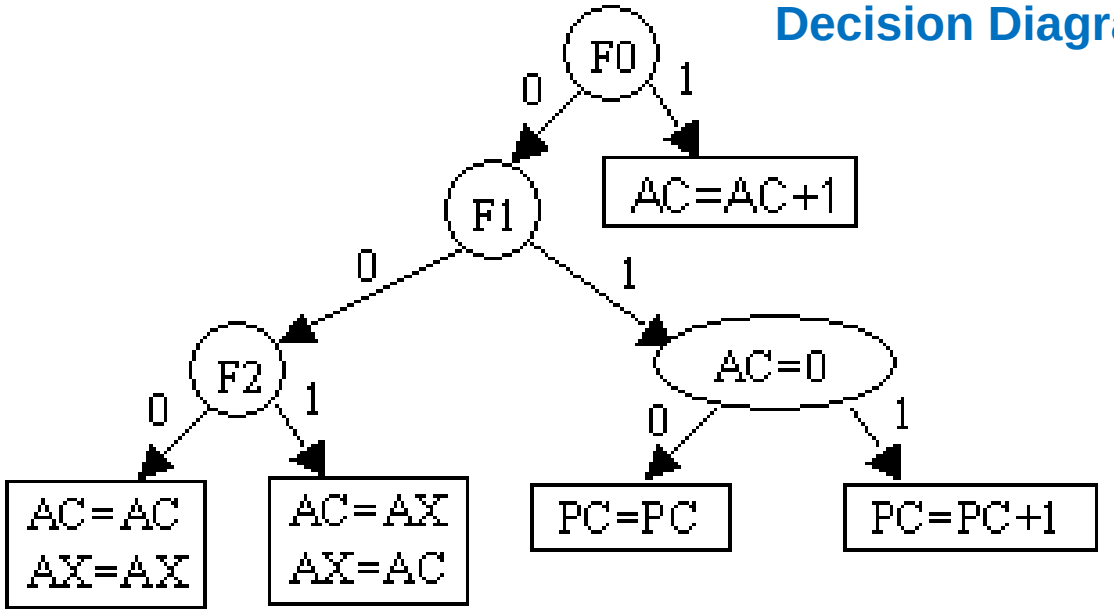
IF **F0** THEN **AC = AC + 1**
ELSE
IF **F1** THEN
IF AC = 0
THEN **PC = PC + 1**
ELSE
IF **F2** THEN
AC = AX, AX = AC

Symbolic execution of the Data-Flow Diagram

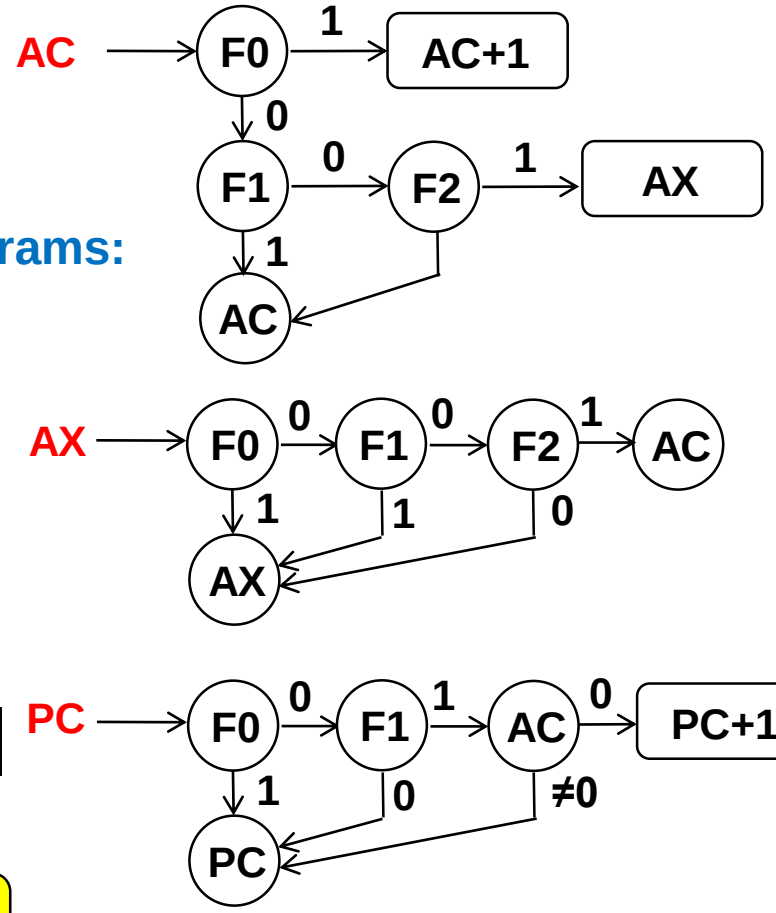


Functional Synthesis of High-Level DDs

High-Level DDs can be synthesized by symbolic execution of the Data-Flow Diagram:



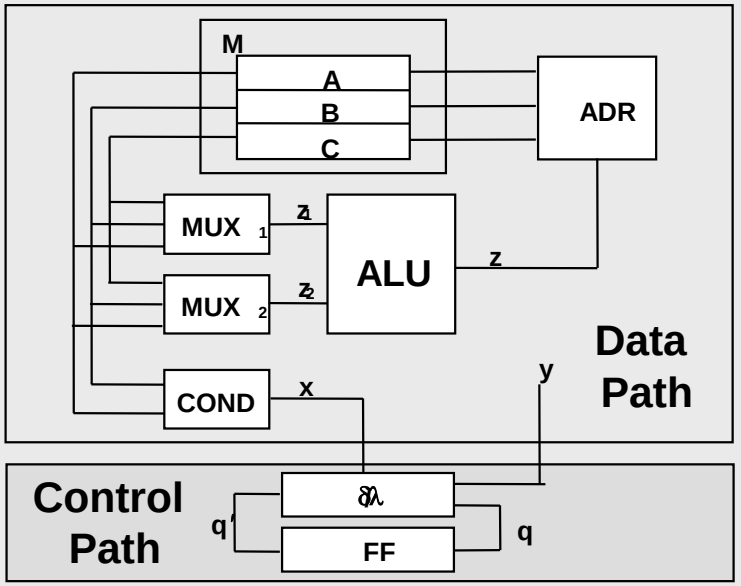
Decision Diagrams:



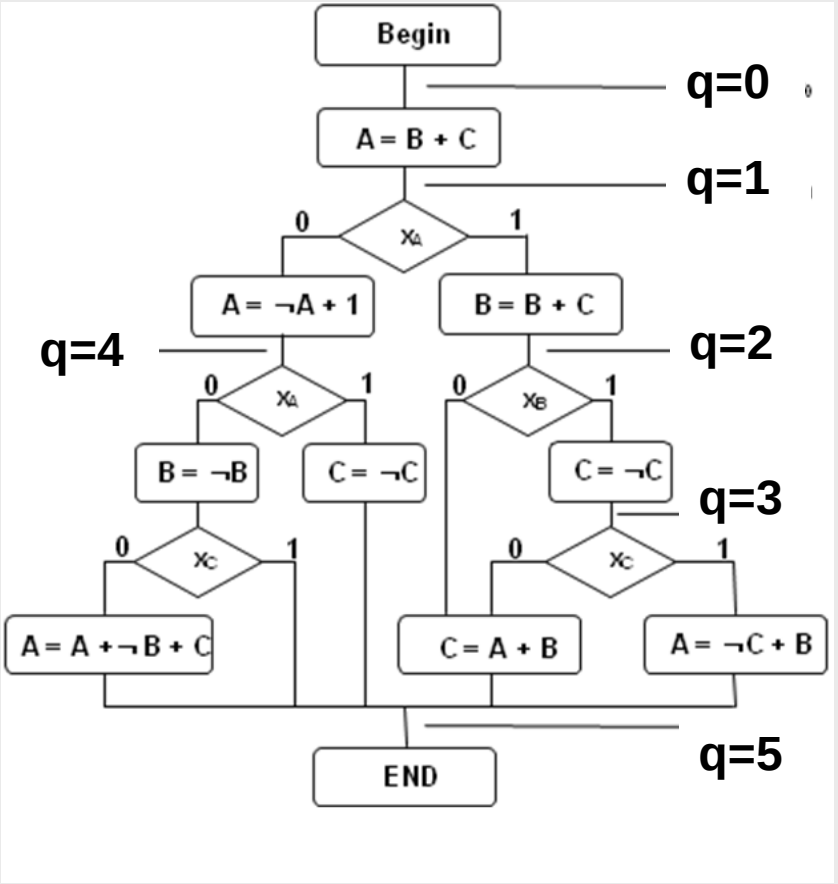
HLDD can be designed for a program written in any language describing either SW or HW

Digital System and Data Flow Diagram

Digital system



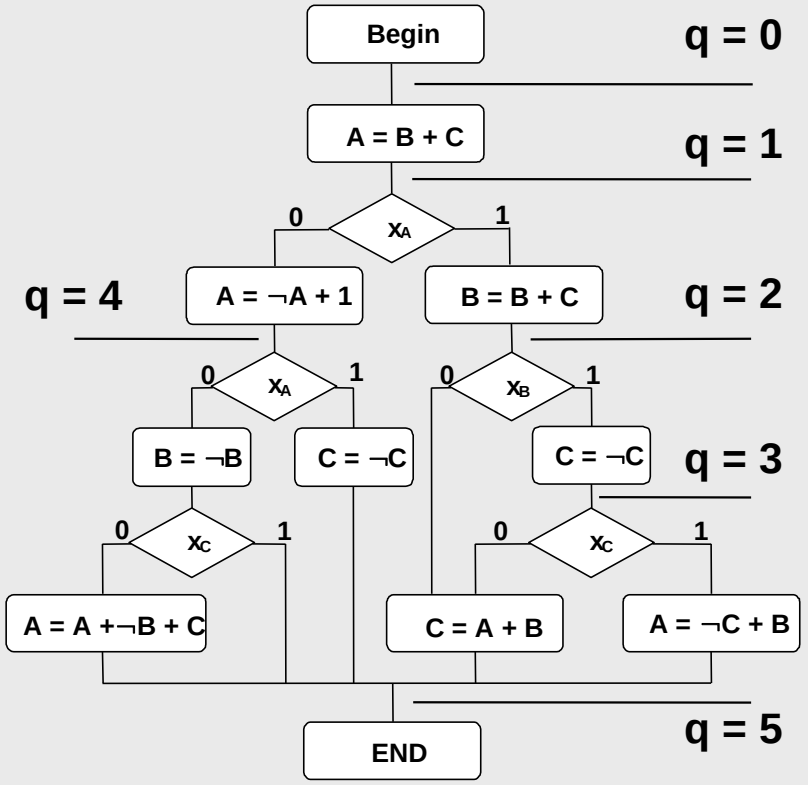
Data Flow Diagram



Synthesis of Functional HLDDs

Data Flow Diagram/FSMD

Insertion of states into the diagram



Results of the cycle based symbolic simulation:

Constraints				Assignment statements
q	x_A	x_B	x_C	
0				A = B + C; q = 1
1	0			A = ¬A + 1; q = 4
1	1			B = B + C; q = 2
2		0		C = A + B; q = 5
2		1		C = ¬C; q = 3
3			0	C = A + B; q = 5
3			1	A = ¬C + B; q = 5
4	0			B = ¬B
4	0		0	A = A + ¬B + C; q = 5
4	0		1	q = 5
4	1			C = ¬C; q = 5

Synthesis of HLDDs

Results of symbolic simulation:

Constraints				Assignment statements
q	x_A	x_B	x_C	
0				A = B + C; q = 1
1	0			A = ¬A + 1; q = 4
1	1			B = B + C; q = 2
2		0		C = A + B; q = 5
2		1		C = ¬C; q = 3
3			0	C = A + B; q = 5
3			1	A = ¬C + B; q = 5
4	0			B = ¬B
4	0		0	A = A + ¬B + C; q = 5
4	0		1	q = 5
4	1			C = ¬C; q = 5

Extraction of the behaviour for A:

q	x_A	x_B	x_C	A
0				B + C
1	0			¬A + 1
3			1	¬C + B
4	0		0	A + ¬B + C

Predicate equation for A:

$$A = f(q, A, B, C, x_A, x_C) =$$

$$= (q=0)(B+C) \vee$$

$$(q=1)(x_A=0)\,(\neg A + 1) \vee$$

$$(q=3)(x_C=1)(\neg C+B) \vee$$

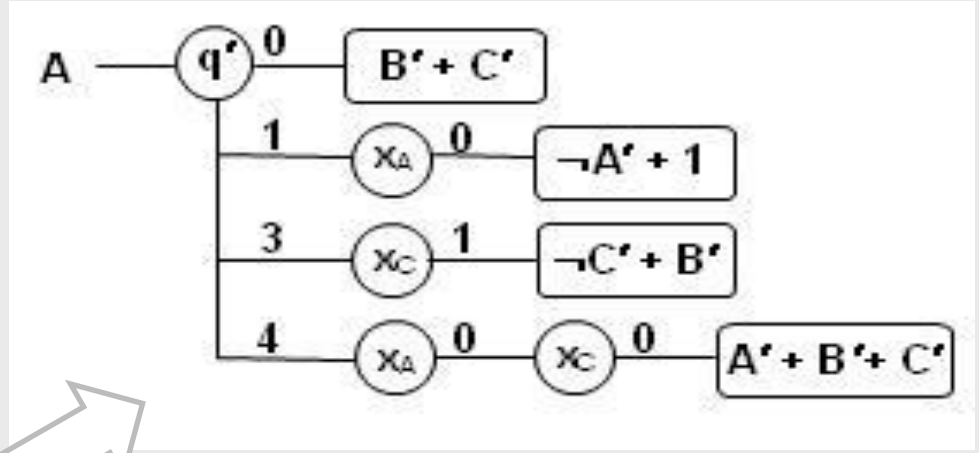
$$(q=4)(x_A=0)(x_C=0)(A+ \neg B + C + 1)$$

Synthesis of HLDDs

Extraction of the behaviour for A:

q	x_A	x_B	x_C	A
0				B + C
1	0			¬A + 1
3			1	¬C + B
4	0		0	A + ¬B + C

Decision diagram for A:



Predicate equation for A:

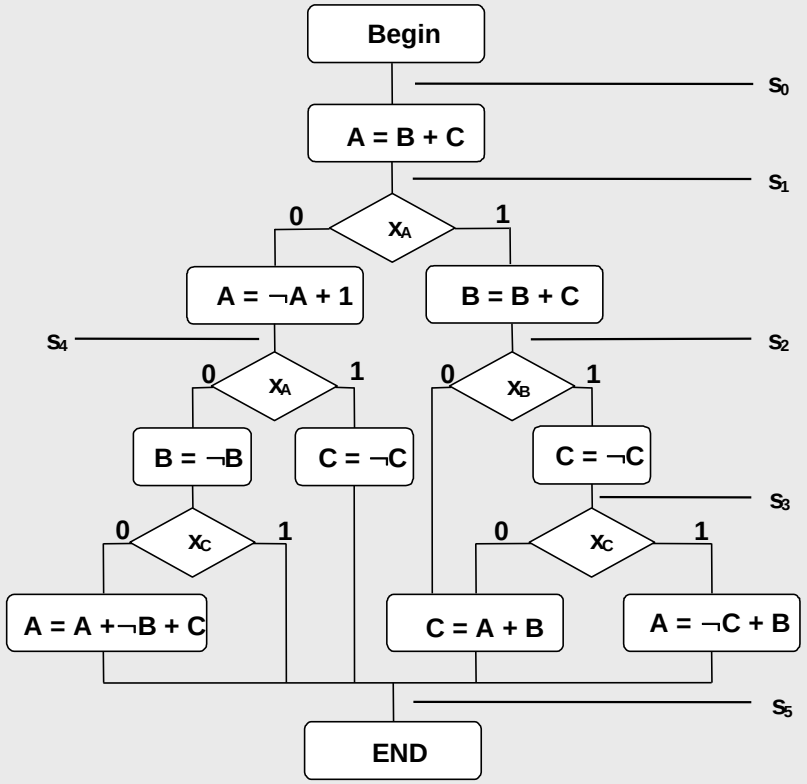
$$A = (q=0)(B+C) \vee (q=1)(x_A=0)(\neg A + 1) \vee$$
$$(q=3)(x_C=1)(\neg C+B) \vee$$
$$(q=4)(x_A=0)(x_C=0)(A+ \neg B + C + 1)$$

Synthesis method: similar to Shannon's expansion theorem:

$$y = F(X) = x_k F(X)\big|_{x_k=1} \vee \overline{x_k} F(X)\big|_{x_k=0}$$

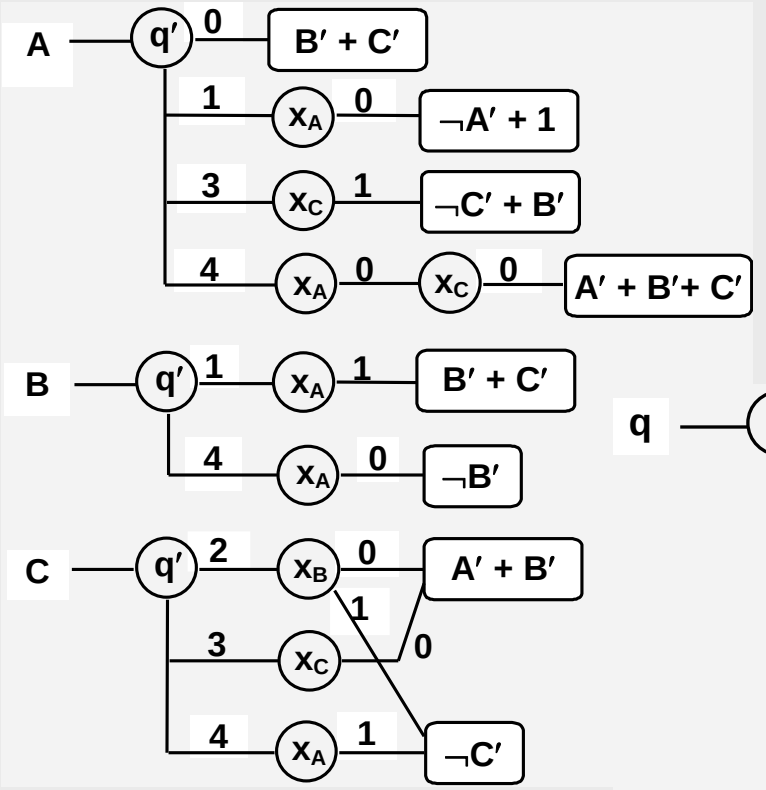
Functional HLDDs

Data Flow Diagram

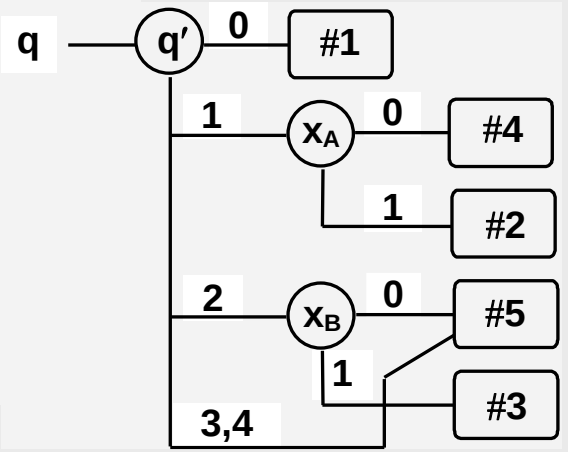


Decision Diagrams

Register variables

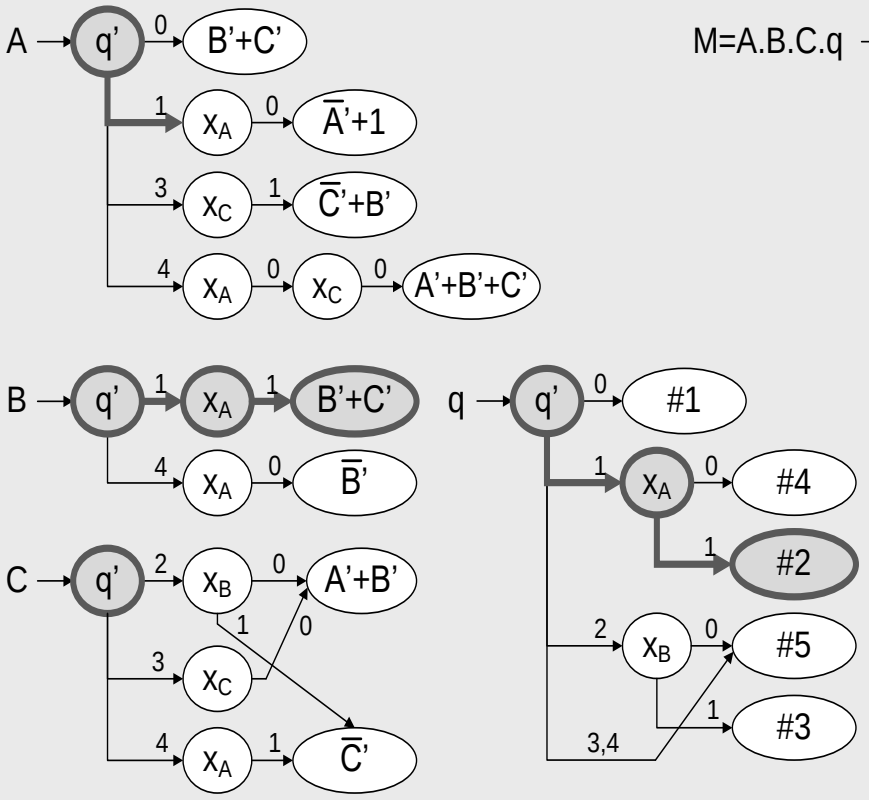


State variable

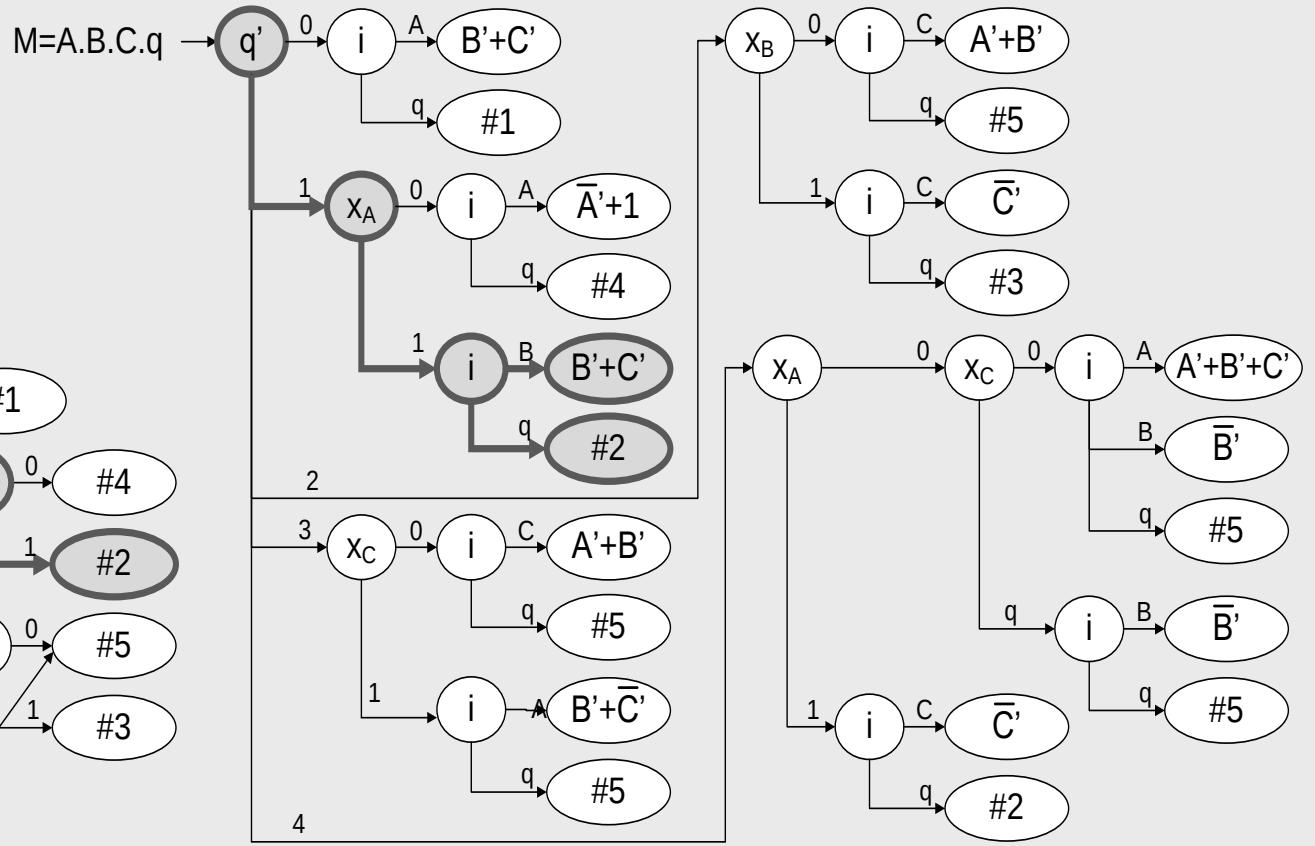


High-Level Vector Decision Diagrams

System with 4 HLDDs

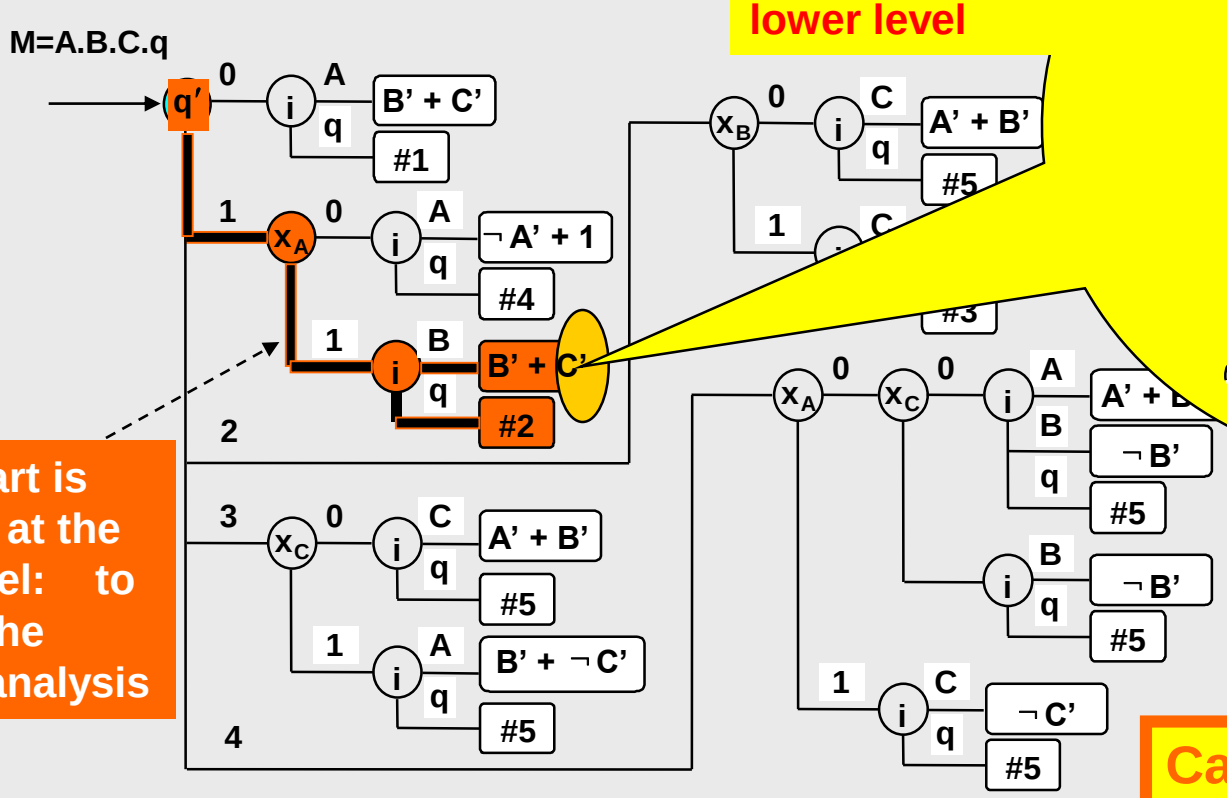


Vector HLDD

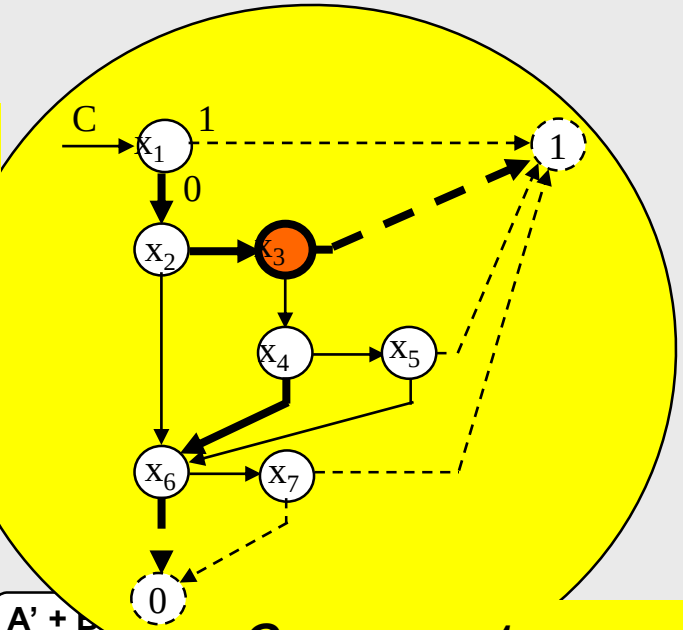


Hierarchical Test Generation with DDs

System:
High-level decision diagram



A small part is simulated at the lower level

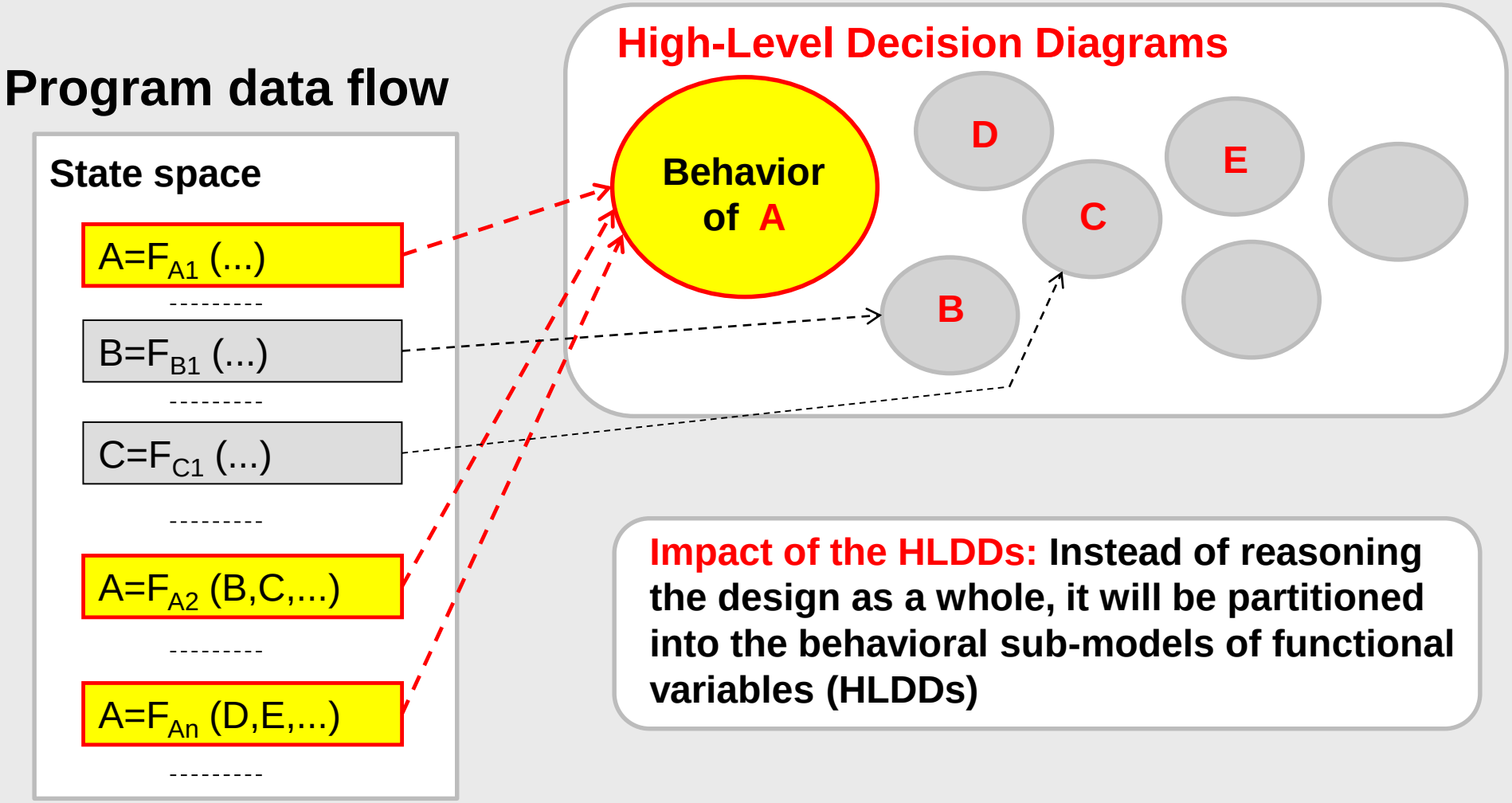


Component:
Binary Decision Diagram

A small part is simulated at the higher level: to increase the speed of analysis

Cause-effect analysis well formalized

Managing of the Complexity of HL Reasoning



Impact of the HLDDs: **Instead of reasoning the design as a whole, it will be partitioned into the behavioral sub-models of functional variables (HLDDs)**

DD Synthesis from VHDL Descriptions

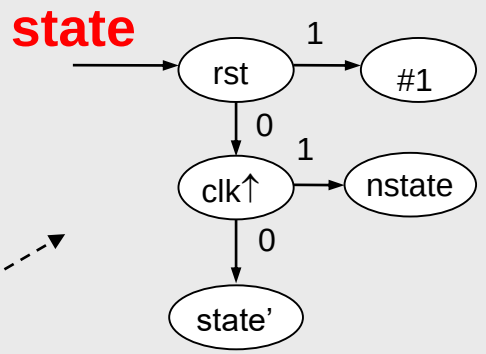
VHDL description of 4 processes which represent a simple control unit

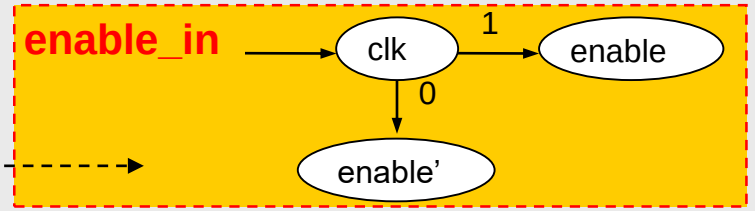
```
entity rd_pc is
  port
    ( clk, rst : in bit; rb0 : in bit; enable : in bit; reg_cp : out bit ;
    reg : out bit ; outreg : out bit ; fin : out bit ) ;
end rd_pc ;
architecture archi_rd_pc of rd_pc is type STATETYPE is
  (state1, state2);
  signal state, nstate: STATETYPE ; signal enable_in : bit ;
  signal reg_cp_comb : bit ;
begin
  seq: process(clk, rst)
  begin
    if rst='1' then state <= state1 ;
    elsif (clk'event and clk='1') then state <=  nstate ;
    end if ;
  end process ;
  process(clk, enable)
  begin
    if clk='1' then   enable_in <= enable ;
    end if ;
  end process ;
  process(clk, reg_cp_comb)
  begin
    if clk='0' then   reg_cp <= reg_cp_comb ;
    end if ;
  end process ;
  comb: process (state, rb0, enable_in)
  begin
    case state is
      when state1 => outreg <= '0' ; fin <= '0' ;
        if (enable_in='0') then   nstate <= state1 ;
                          reg <= '1' ; reg_cp_comb <= '0' ;
                     else  nstate <= state2 ; reg <= '1' ;
                          reg_cp_comb <= '1' ;
        end if ;
      when state2 =>
        if (rb0='1') then   nstate <= state2 ; reg <= '0' ;
                       reg_cp_comb <= '1'; outreg <= '0'; fin <= '0';
          elsif (enable_in='0') then   nstate <= state1 ; reg <= '0' ;
                       reg_cp_comb <= '0'; outreg <= '1'; fin <= '1';
                       else  nstate <= state2 ; reg <= '0' ;
                       reg_cp_comb <= '0'; outreg <= '0'; fin <= '1';
        end if ;
    end case ;
  end process ;
end archi_rd_pc ;
```

Synthesis of HLDDs from VHDL

```
entity rd_pc is
   port
      ( clk, rst : in bit; rb0 : in bit; enable : in bit; reg_cp : out bit ;
      reg : out bit ; outreg : out bit ; fin : out bit ) ;
end rd_pc ;
architecture archi_rd_pc of rd_pc is type STATETYPE is
   (state1, state2);
   signal state, nstate: STATETYPE ; signal enable_in : bit ;
   signal reg_cp_comb : bit ;
begin
   seq: process(clk, rst)
   begin
      if rst='1' then state <= state1 ;
      elsif (clk'event and clk='1') then state <=  nstate ;
      end if ;
   end process ;
   process(clk, enable)
   begin
      if clk='1' then   enable_in <= enable ;
      end if ;
   end process ;
```

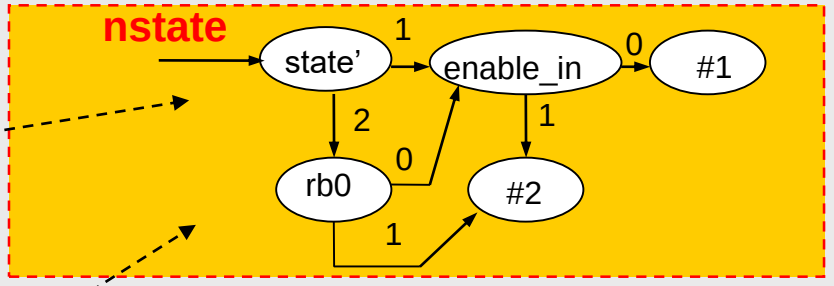
DDs for **state, enable_in** *and* **nstate**





Synthesis of HLDDs from VHDL

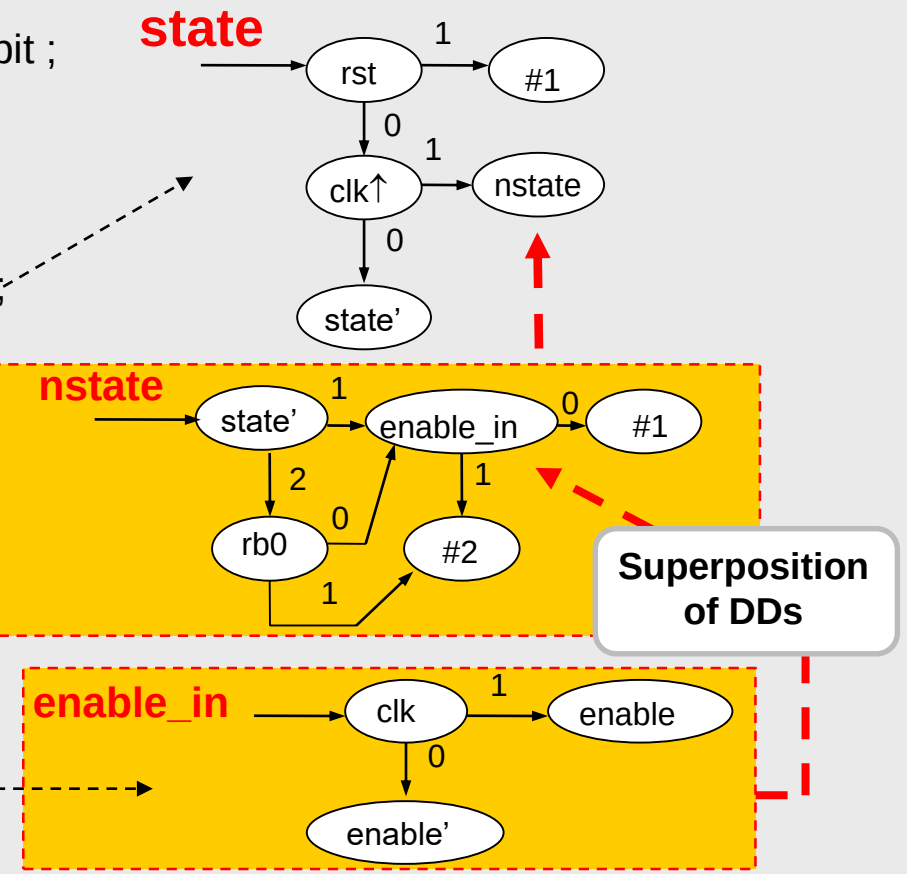
```
process(clk, reg_cp_comb)
begin
  if clk='0' then   reg_cp <= reg_cp_comb ;
  end if ;
end process ;
comb: process (state, rb0, enable_in)
begin
  case state is
    when state1 => outreg <= '0' ; fin <= '0' ;
      if (enable_in='0') then   nstate <= state1 ;
                          reg <= '1' ; reg_cp_comb <= '0' ;
                    else  nstate <= state2 ; reg <= '1' ;
                          reg_cp_comb <= '1' ;
      end if ;
    when state2 =>
      if (rb0='1') then   nstate <= state2 ; reg <= '0' ;
                    reg_cp_comb <= '1'; outreg <= '0'; fin <= '0';
        elsif (enable_in='0') then   nstate <= state1 ; reg <= '0' ;
                    reg_cp_comb <= '0'; outreg <= '1'; fin <= '1';
                    else  nstate <= state2 ; reg <= '0' ;
                    reg_cp_comb <= '0'; outreg <= '0'; fin <= '1';
      end if ;
  end case ;
end process ;
end archi_rd_pc ;
```



Synthesis of HLDDs from VHDL

```
entity rd_pc is
   port
      ( clk, rst : in bit; rb0 : in bit; enable : in bit; reg_cp : out bit ;
      reg : out bit ; outreg : out bit ; fin : out bit ) ;
end rd_pc ;
architecture archi_rd_pc of rd_pc is type STATETYPE is
   (state1, state2);
   signal state, nstate: STATETYPE ; signal enable_in : bit ;
   signal reg_cp_comb : bit ;
begin
   seq: process(clk, rst)
   begin
      if rst='1' then state <= state1 ;
      elsif (clk'event and clk='1') then state <=  nstate ;
      end if ;
   end process ;
   process(clk, enable)
   begin
      if clk='1' then   enable_in <= enable ;
      end if ;
   end process ;
```

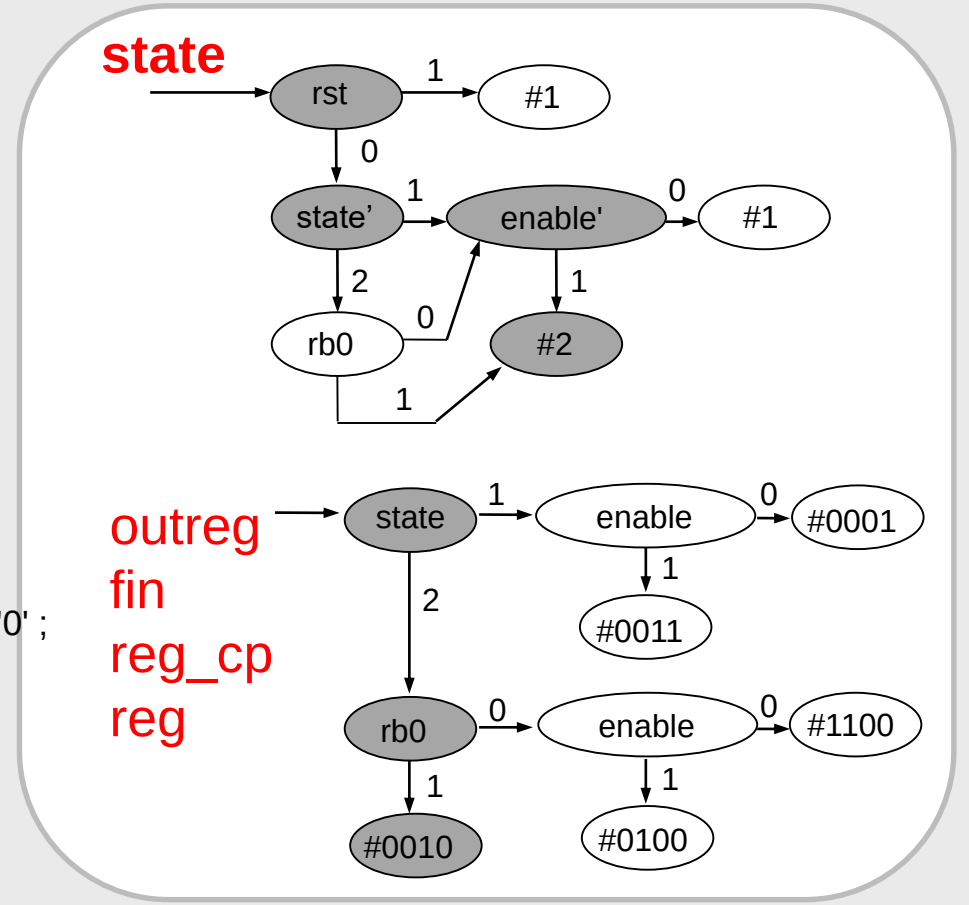
DDs for **state, enable_in** *and* **nstate**



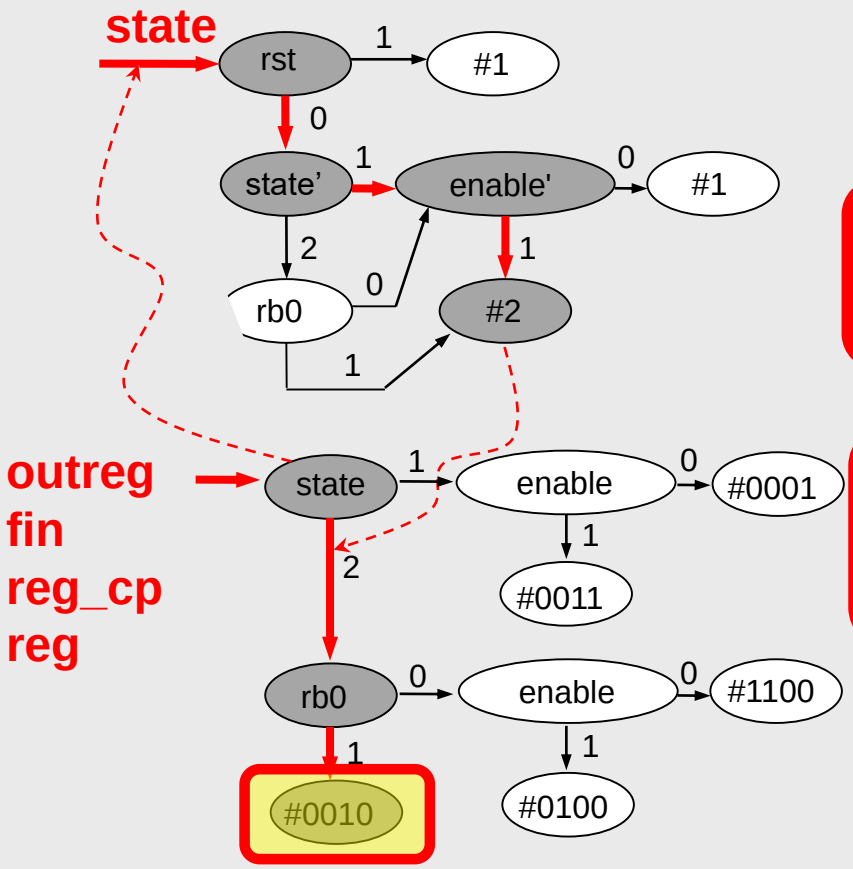
Synthesis of HLDDs from VHDL

```
process(clk, reg_cp_comb)
begin
  if clk='0' then   reg_cp <= reg_cp_comb ;
  end if ;
end process ;
comb: process (state, rb0, enable_in)
begin
  case state is
    when state1 => outreg <= '0' ; fin <= '0' ;
      if (enable_in='0') then   nstate <= state1 ;
                          reg <= '1' ; reg_cp_comb <= '0' ;
                    else  nstate <= state2 ; reg <= '1' ;
                          reg_cp_comb <= '1' ;
      end if ;
    when state2 =>
      if (rb0='1') then   nstate <= state2 ; reg <= '0' ;
                    reg_cp_comb <= '1'; outreg <= '0'; fin <= '0';
        elsif (enable_in='0') then   nstate <= state1 ; reg <= '0' ;
                    reg_cp_comb <= '0'; outreg <= '1'; fin <= '1';
                    else  nstate <= state2 ; reg <= '0' ;
                    reg_cp_comb <= '0'; outreg <= '0'; fin <= '1';
      end if ;
  end case ;
end process ;
end archi_rd_pc ;
```

HLDD model for the Control Unit



Simulation & Fault Backtracing with HLDD



Inputs

time	1	2	3	4	5	6
rst	1	0	0	0	0	0
enable	0	1	1	1	0	0
rb0	x	x	1	0	0	0
state	1	1	2	2	2	1
outreg	0	0	0	0	1	0
fin	0	0	0	1	1	0
reg_cp	0	1	1	0	0	0
reg	1	1	0	0	0	1

Outputs

Simulated vector

Motivations for High-Level Fault Models

Current State-of-the-Art:

- The efficiency of test generation (quality, speed) is highly depending on
  - the description method (level, language), and
  - fault models
- Because of the growing complexity of systems, gate level methods have become obsolete
- High-Level methods for diagnostic modeling are today emerging, however they are not still mature

Main disadvantages:

- The known methods for fault modeling are
  - dedicated to special classes (i.e. for microprocessors, for RTL, VHDL etc. languages...), not general
  - not well defined and formalized

State of Art: Microprocessor Fault Model

Source decoding (MUX):
F1: no source is selected
F2: wrong source is selected;
F3: more than one source is selected and the multiplexer output is either a wired-AND or a wired-OR function of the sources, depending on the technology.

Destination decoding (DMUX)
F4: no destination is selected
F5: instead of, or in addition to the selected correct destination, one or more other destinations selected

Instruction exec. faults
F6: one or more microorders not activated
F7: microorders are erroneously activated
F8: a different set of micro-instructions is activated instead of, or in addition to

Data storage/bus/ALU faults:
F9: one or more cells SAF0 /1;
F10: one or more cells fail 0→1/1→0
F11: two or more cells coupled;
F12: one or more lines SAF0 /1;
F13: one or more lines wired-OR/AND
F14: data manipulation faults

Thatte'1980

Register Level Fault Models

RTL statement:

$$K: (If\ T,C)\ \ R_D \leftarrow F(R_{S1}, R_{S2}, \ldots R_{Sm}),\ \rightarrow\ N$$

Components (variables) of the statement:

K	- label
T	- timing condition
C	- logical condition
R_D	- destination register
R_S	- source register
F	- operation (microoperation)
$\leftarrow$	- data transfer
$\rightarrow$ N	- jump to the next statement

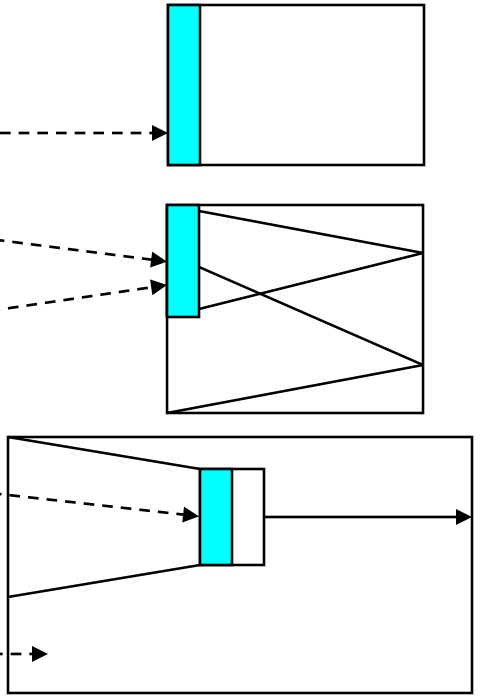
RT level faults:

$K \rightarrow K'$	- label faults
$T \rightarrow T'$	- timing faults
$C \rightarrow C'$	- logical condition faults
$R_D \rightarrow R_D$	- register decoding faults
$R_S \rightarrow R_S$	- data storage faults
$F \rightarrow F'$	- operation decoding faults
$\leftarrow$	- data transfer faults
$\rightarrow$ N	- control faults
$(F) \rightarrow (F)'$	- data manipulation faults

Universal Functional Fault Models

Exhaustive combinational fault model:

- **exhaustive test patterns**
- **pseudoexhaustive test patterns**
  - **exhaustive output line oriented test patterns**
  - **exhaustive module oriented test patterns**



Hierarchical test generation:

Fault Models for High-Level Components

Decoder:

- instead of correct line, incorrect is activated
- in addition to correct line, additional line is activated
- no lines are activated

Multiplexer (n inputs $log_2 n$ control lines):

- stuck-at - 0 (1) on inputs
- another input (instead of, additional)
- value, followed by its complement
- value, followed by its complement on a line whose address differs in 1 bit

Memory fault models:

- one or more cells stuck-at - 0 (1)
- two or more cells coupled

Fault Model vs. Test Description

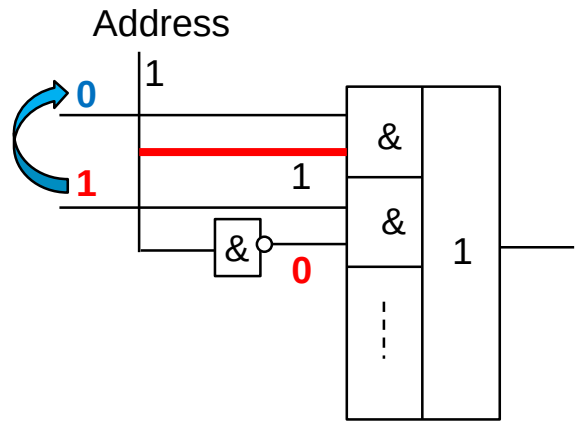
Dedicated functional fault model for multiplexer:

- **stuck-at-0 (1) on inputs,**
- **another input (instead of, additional)**
- **value, followed by its complement**

Functional fault model

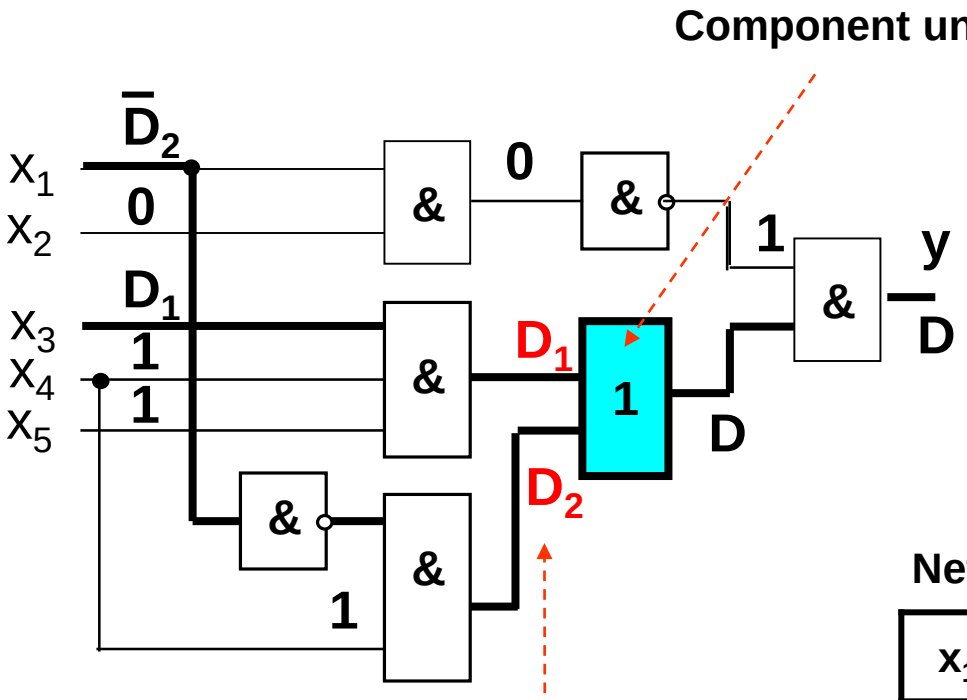
- **value, followed by its complement on a line whose address differs in one bit**

Test description



This functional model corresponds to two SAF models for all bit-slices in the MUX-circuit

Hierarchical Test Generation



Component level test:

D_1	D_2	D
0	0	0
0	1	1
1	0	1

Network level test (description):

x_1	x_2	x_3	x_4	x_5	y
$\overline{D}_2$	0	D_1	1	1	D

Input pattern – as a fault model

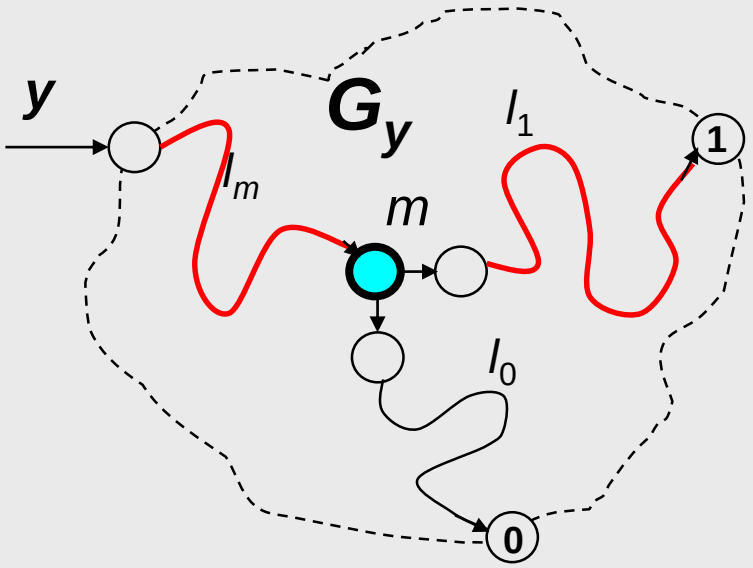
Symbolic test: contains 3 patterns

Fault model (embedded in the test description)

Generalization of BDDs

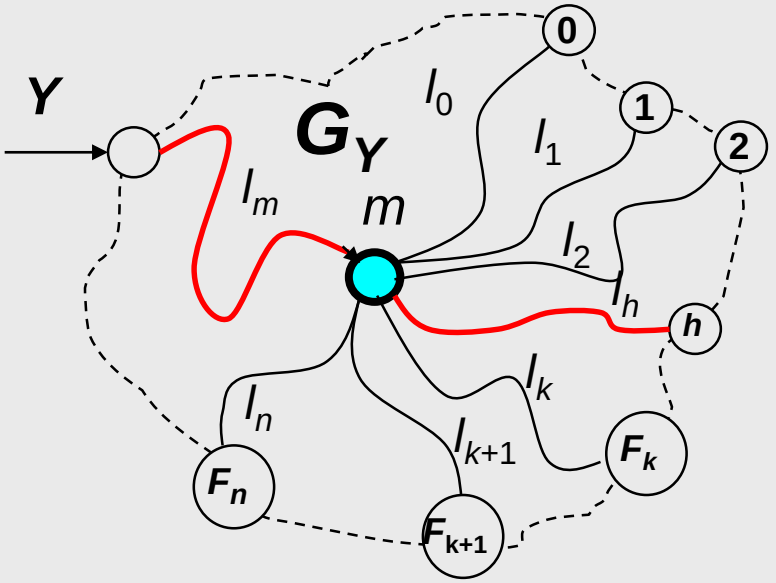
Binary DD

2 terminal nodes and
2 edges from each node



General case of DD

n ≥ 2 terminal nodes and
n ≥ 2 edges from each node



Novelty: **Boolean methods can be generalized in a straightforward way to higher functional levels**

HLDDs and Faults

RTL-statement:

K: (*If* T,C) $R_D \leftarrow F(R_{S1},R_{S2},\ldots R_{Sm}), \rightarrow N$

Nonterminal nodes

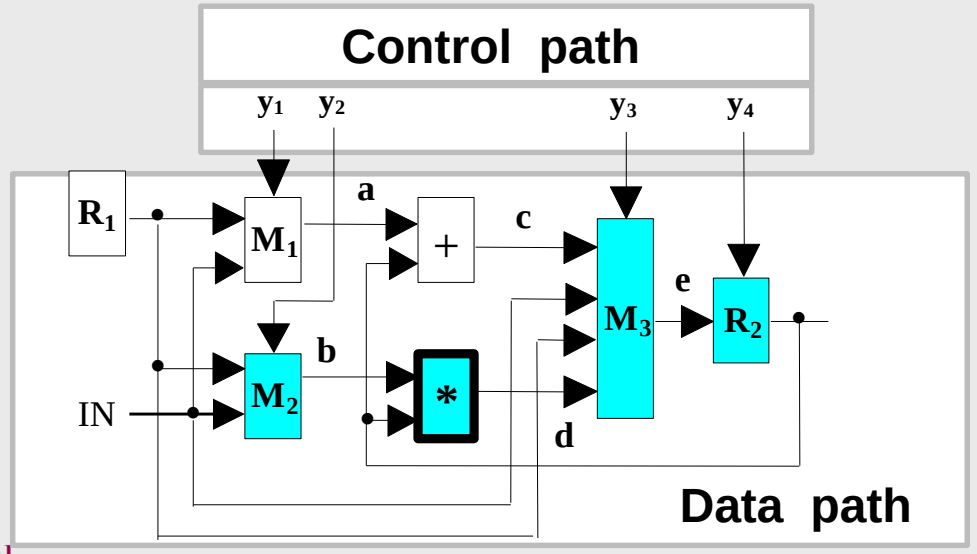
RTL-statement faults:

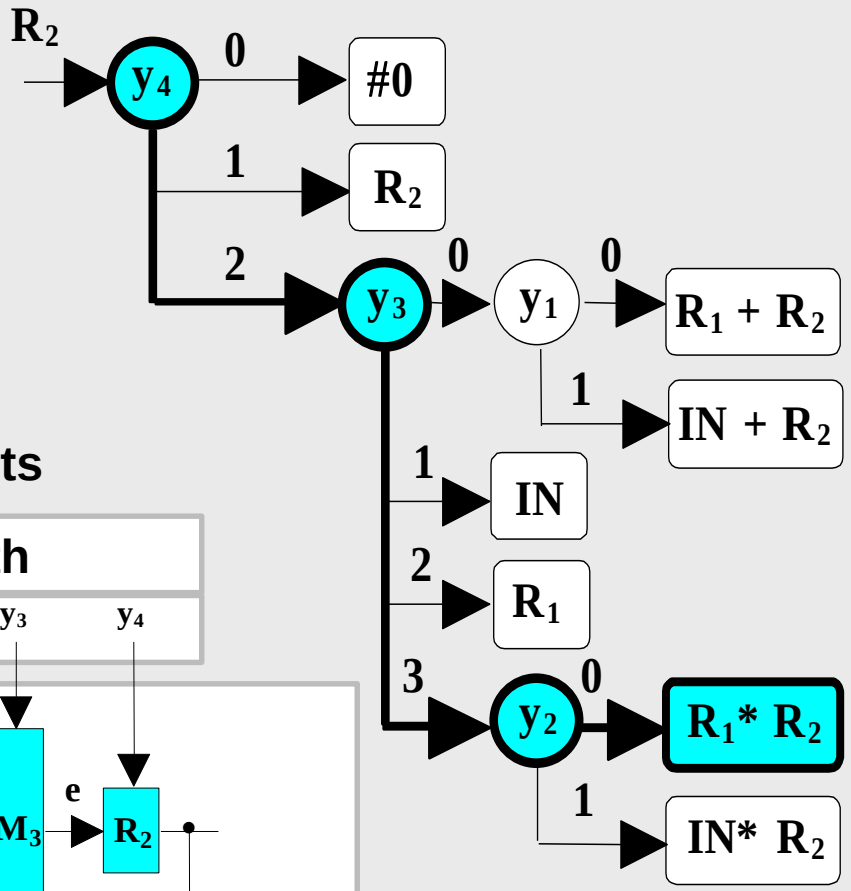
label,
timing condition,
logical condition,
register decoding,
operation decoding,
control faults

Terminal nodes

RTL-statement faults:

data storage,
data transfer,
data manipulation faults





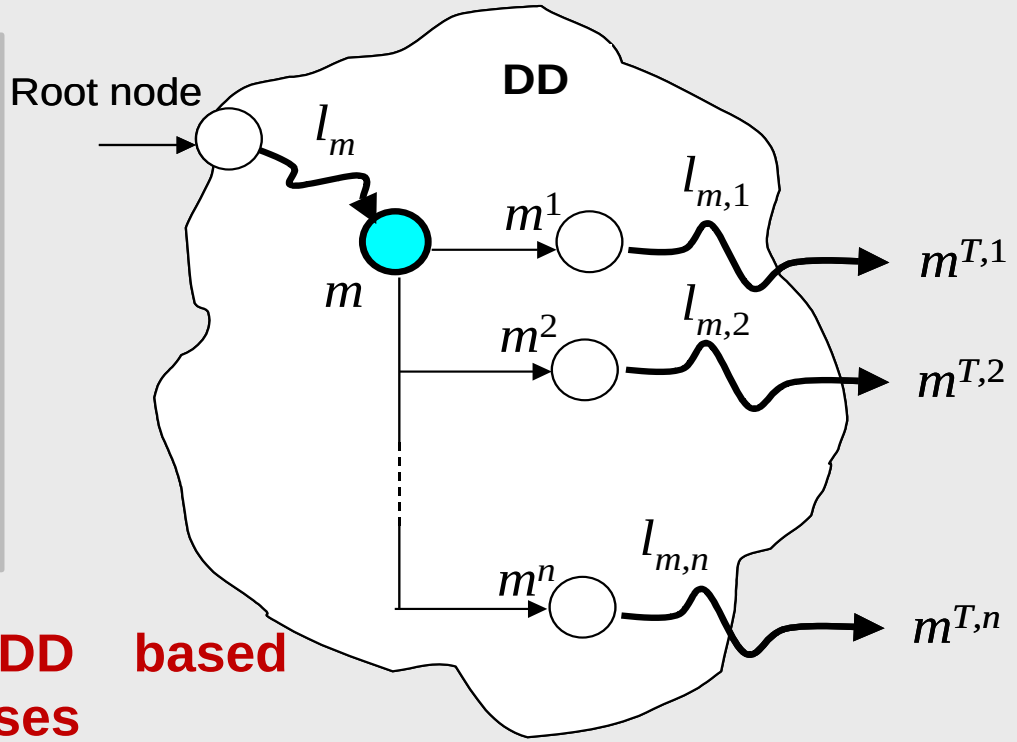
From Trad. MP Fault Model to HLDD Faults

Each path in a DD represents a working mode of the system
The faults having effect on the behaviour can be associated with nodes along the path

D1: node output is **always activated (SAF-1)**

D2: node output is **broken (SAF-0)**

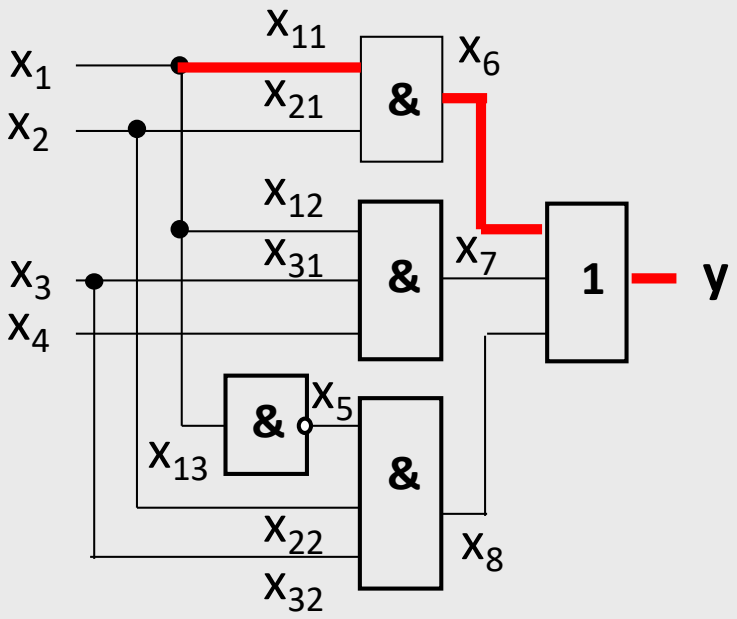
D3: instead of the given output **another output set** is **activated**

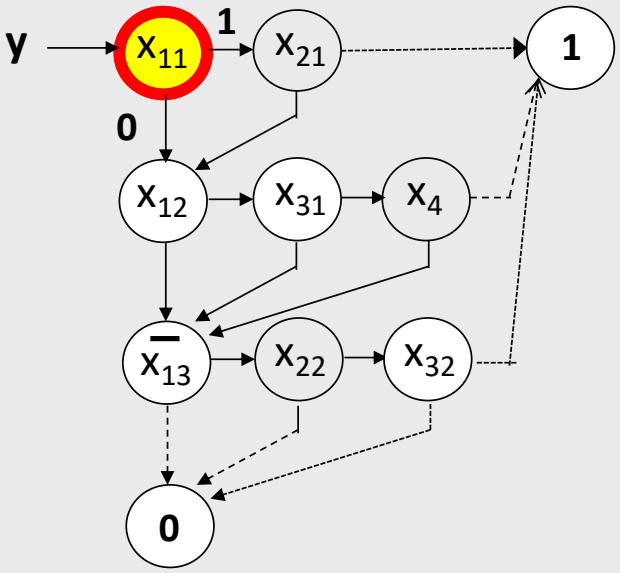


Instead of the 14 fault classes,the HLDD based fault model includes only 3 fault classes

Fault Collapsing with SSBDDs

Each node in SSBDD represents a signal path:



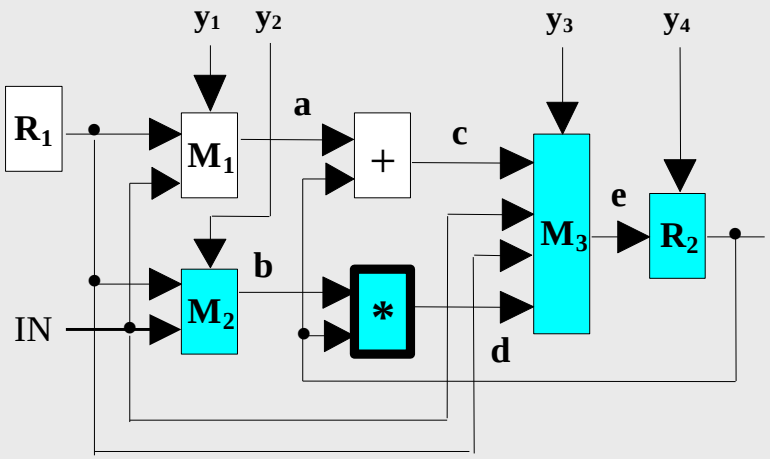


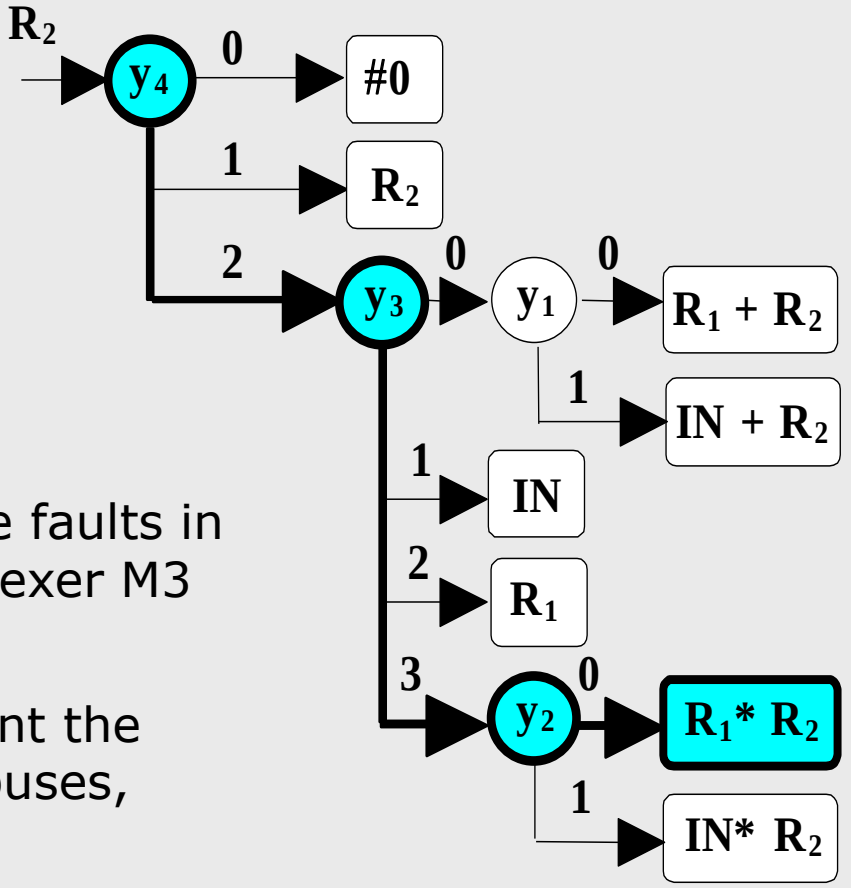
Two SSBDD faults $x_{11}\equiv 1$, $x_{11}\equiv 0$ represent a set of six faults in the circuit:

$$\{x_{11}\equiv 1,\ x_6\equiv 1,\ y\equiv 1;\ x_{11}\equiv 0,\ x_6\equiv 0,\ y\equiv 0\}$$

Fault Collapsing with Structural HLDDs

Each node in SSBDD represents a signal path:





The faults at **y_3** in HLDD represent the faults in the control circuitry and in the multiplexer M3 of the RTL circuit

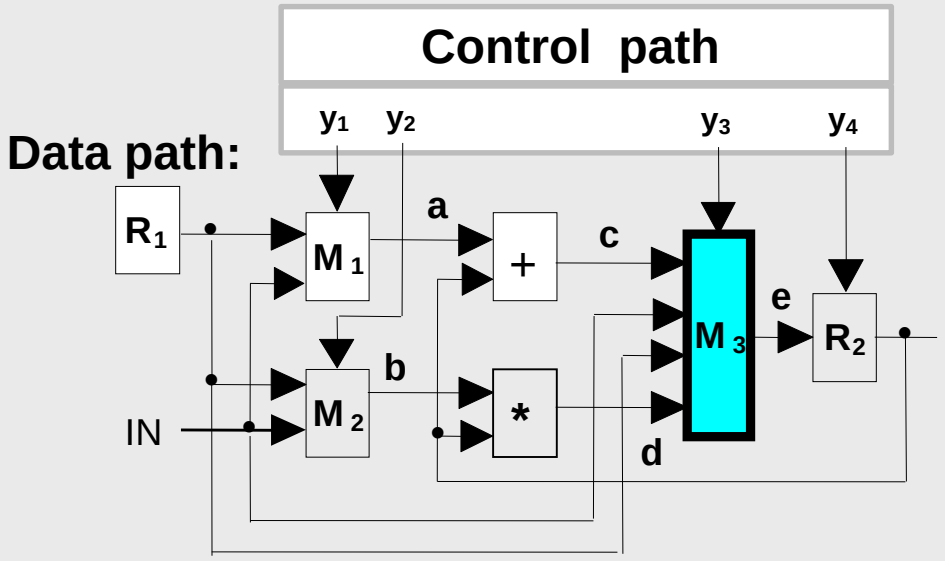
The faults at **R1*R2** in HLDD represent the faults in multiplier, input and output buses, and in the registers

Fault collapsing – not investigated at high-level

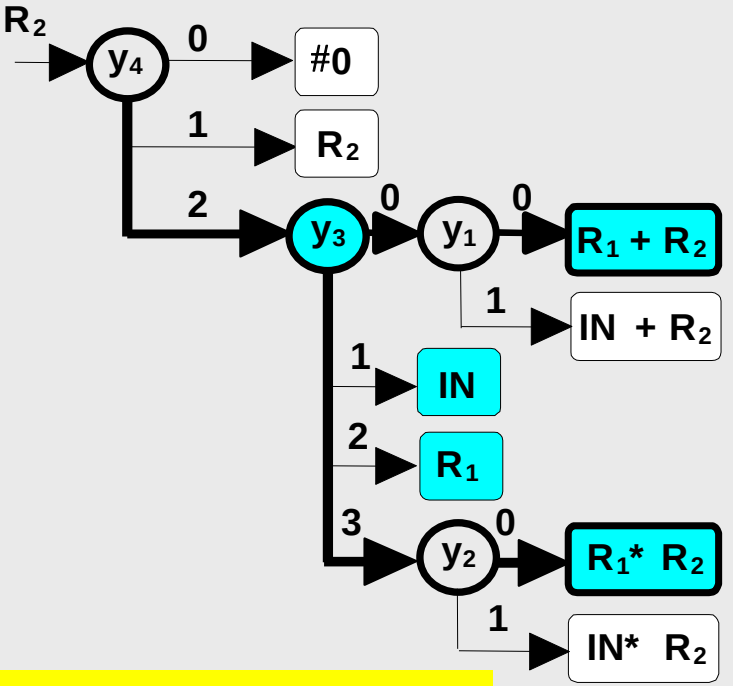
Fault Activation in Digital Systems

Multiple paths activation in a DD

Control function y_3 is tested



Decision Diagram

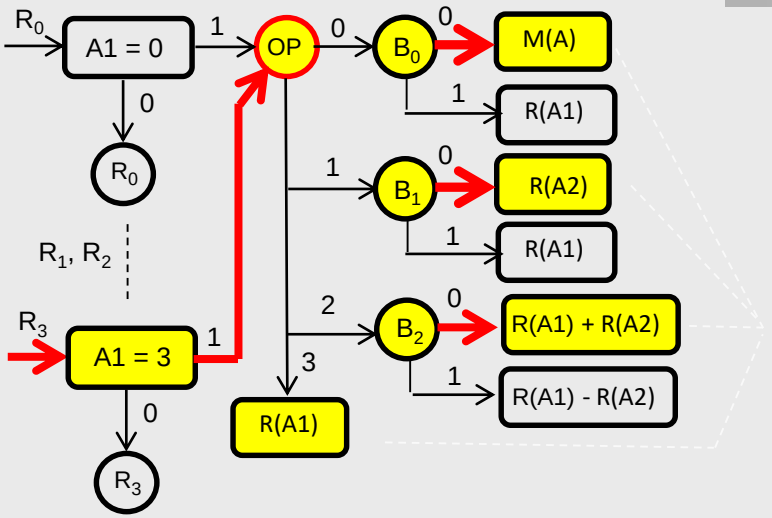


Test cycle:

Control: *For* **D = 0,1,2,3:** $y_1\, y_2\, y_3\, y_4 = 00D2$

Data: *Solution of* $R_1 + R_2 \neq IN \neq R_1 \neq R_1^* R_2$

Uniform Conditional Node Fault Model



Test data calculation rules:

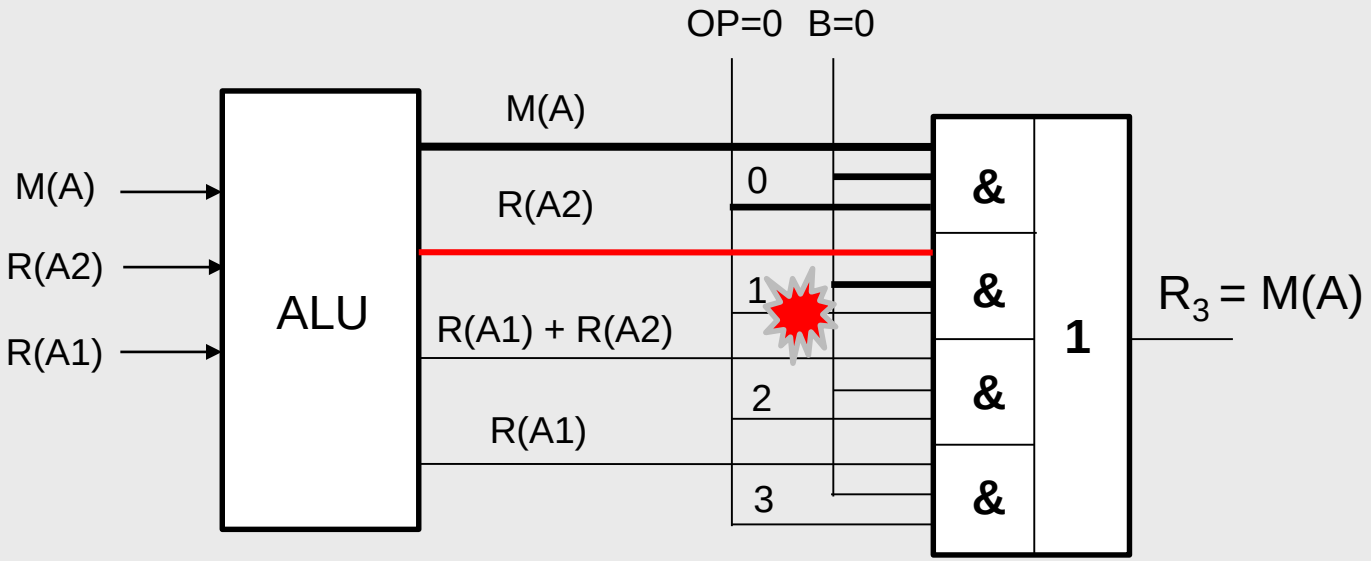
$\forall m^T \in M^T(OP)$: $[f(m^T) \neq \text{ZERO})]$

$\forall m_i, m_j \in M^T(OP)$: $[(f(m_i) < f(m_j)]$

Terminal nodes:

$M^T(OP)$ = {M(A), R(A2), R(A1)+R(A2), R(A1))}

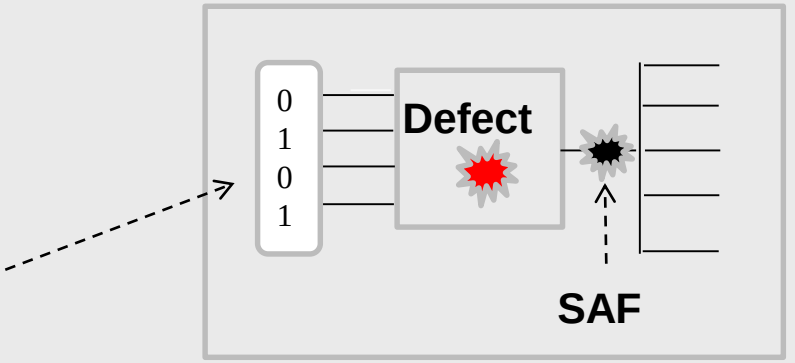
Explanation of the meaning of constraints (rules):



To detect the fault: R3 = M(A) ∨ R(A2) ⟶ M(A) < R(A2) is needed

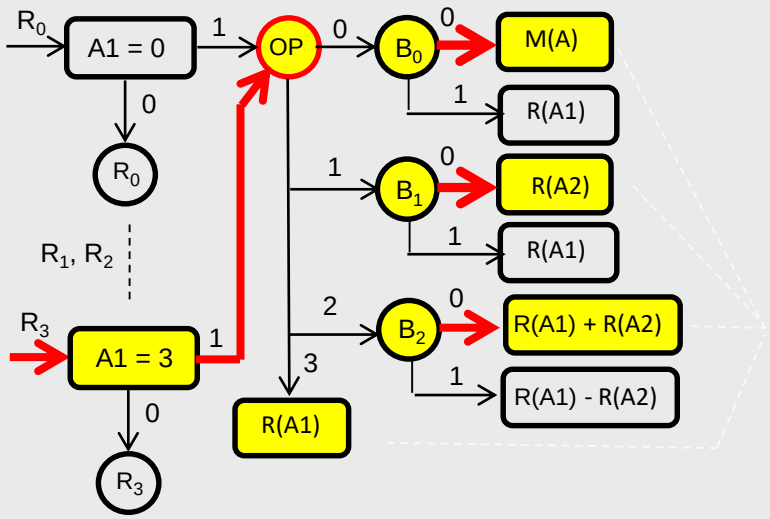
Uniform Conditional Node Fault Model

Logic level analog:
Conditional SAF model



Condition (constraint)

High level analog: Constraints for testing a node *OP* in HLDD:



High-level fault model:

Test data calculation rules:

$\forall m^T \in M^T(OP)$: $[f(m^T) \neq \text{ZERO})]$

$\forall m_i, m_j \in M^T(OP)$: $[(f(m_i) < f(m_j)]$

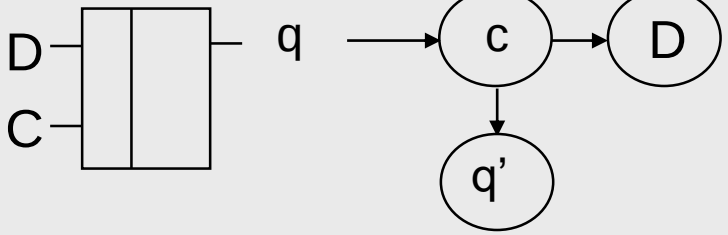
Terminal nodes:

$M^T(OP)$ = {M(A), R(A2), R(A1)+R(A2), R(A1))}

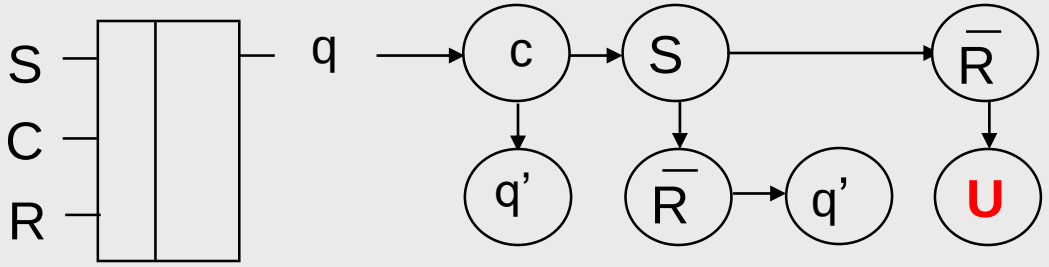
Multi-Terminal BDDs

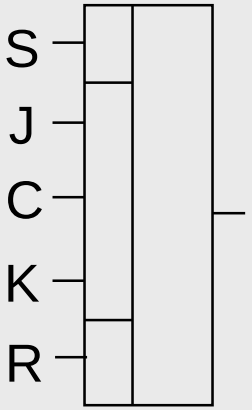
Multi-Terminal BDDs for representing uncertainties:

D Flip-Flop

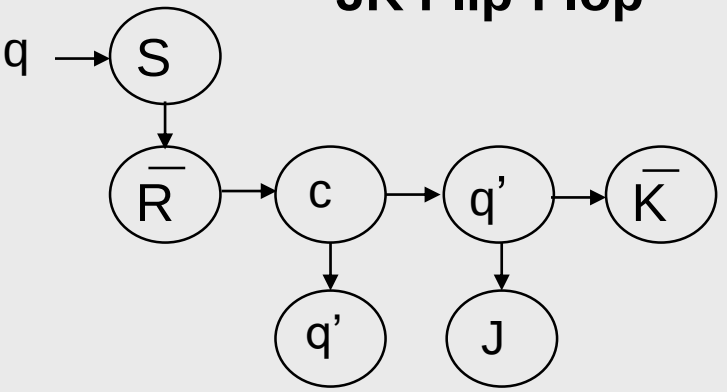


RS Flip-Flop





JK Flip-Flop



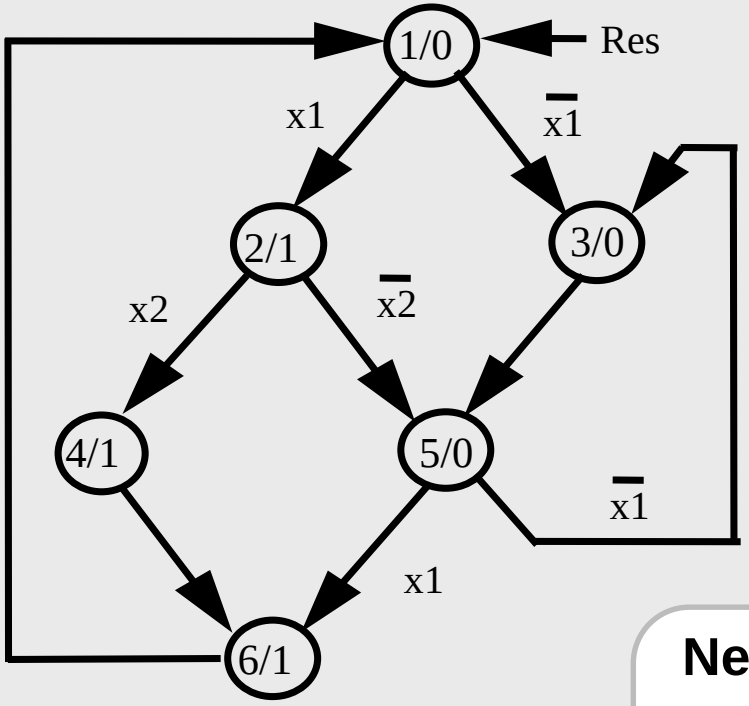
U - unknown value

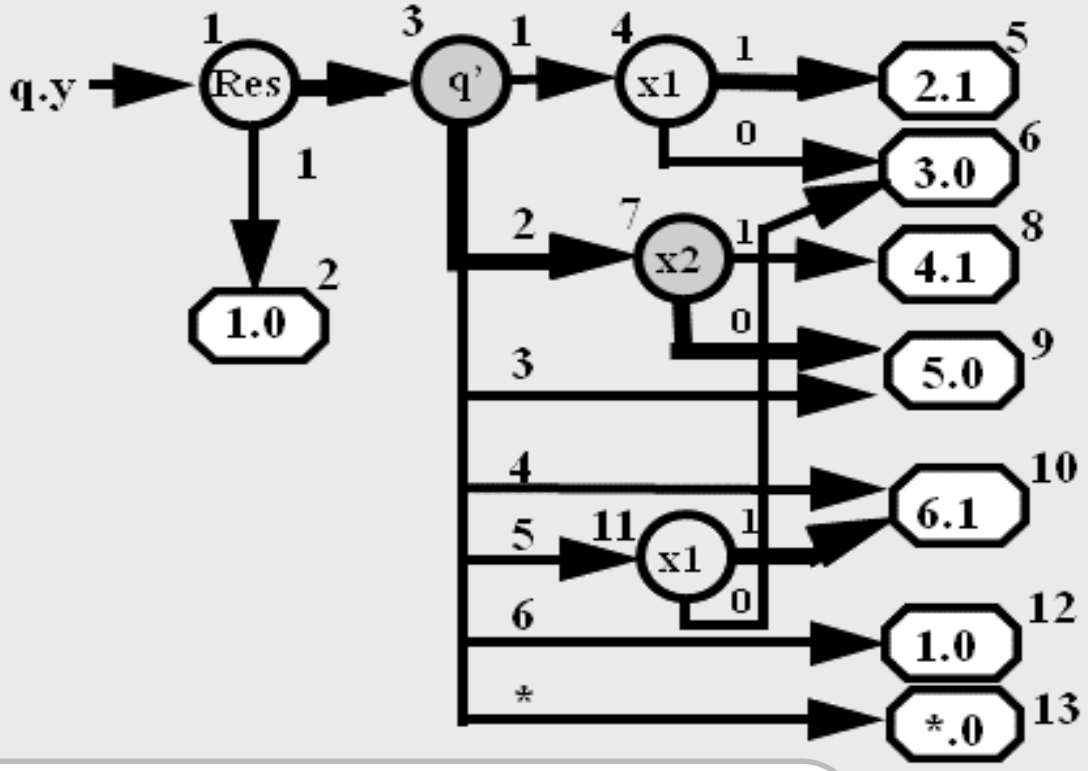
$$q = c(S \vee q'\overline{R}) \vee \overline{c}q'$$

$$SR = 0$$

Generalization of MTBDDs for FSMs

State Transition Diagram:

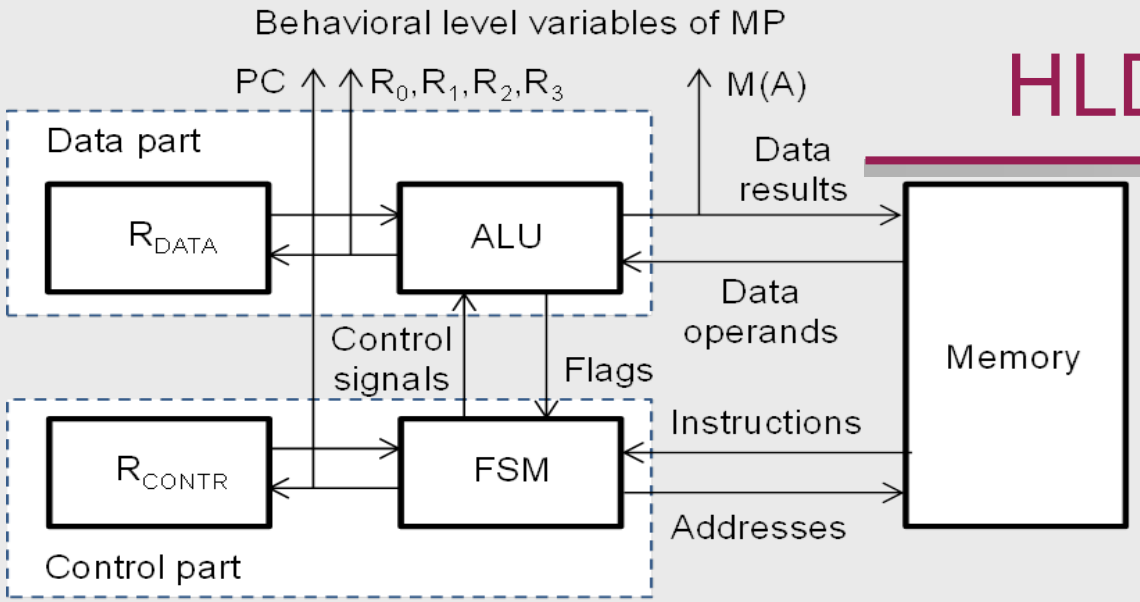




New features:

- ✓ representing vectors (vector DD)
- ✓ multi-output internal nodes
- ✓ multi-terminal DDs

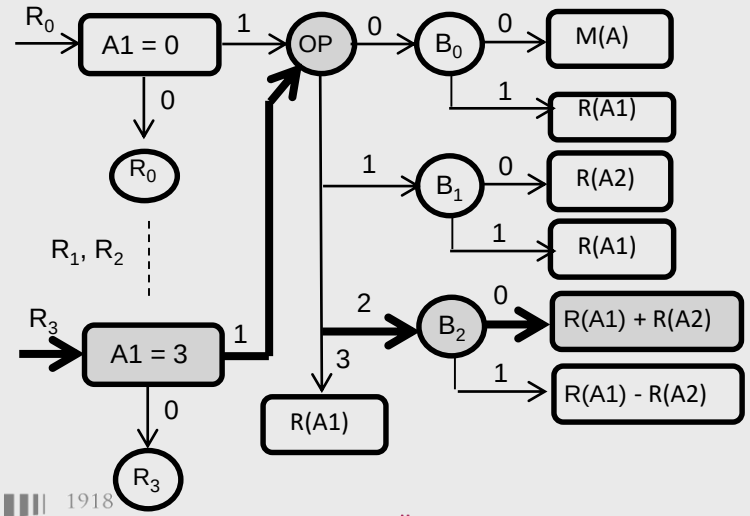
HLDDs for MP InstrSet



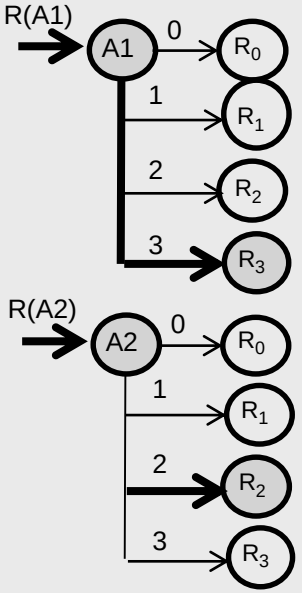
Instruction code:
ADD A1 A2
OP=2. B=0. A1=3. A2=2

$R_3 = R_3 + R_2$
PC = PC+1

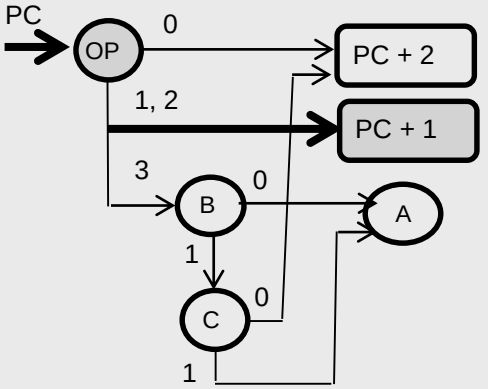
Registers and ALU



Register Decoding



Program Counter



Memory Access