Design Representation

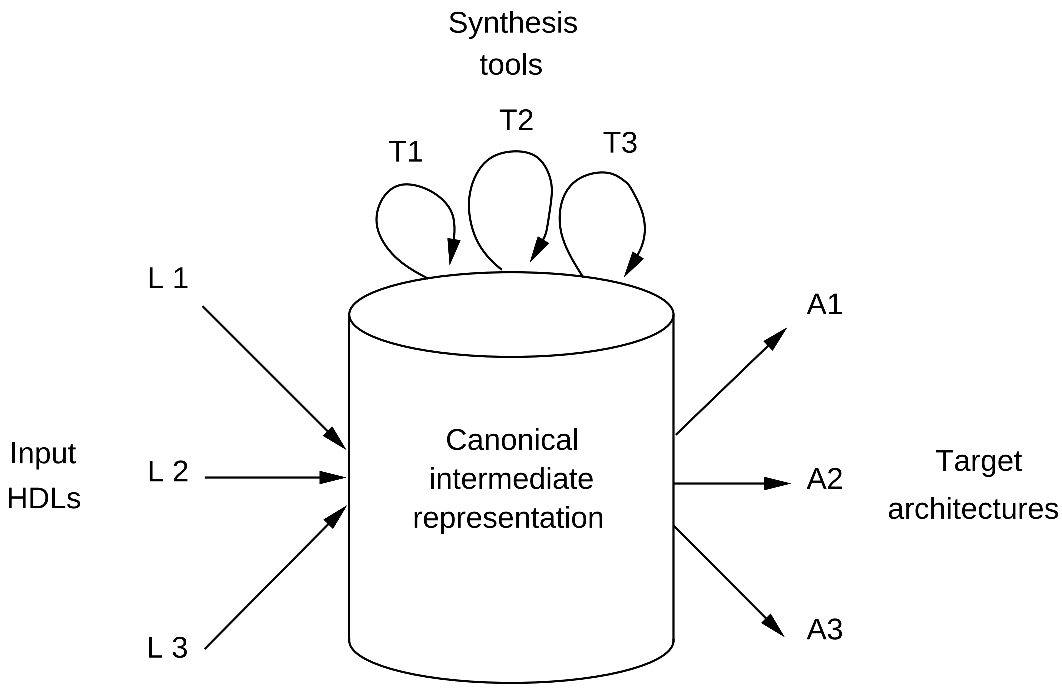
Chapter 5

Source: Gajski, Dutt, Wu, Lin

"High−Level Synthesis"

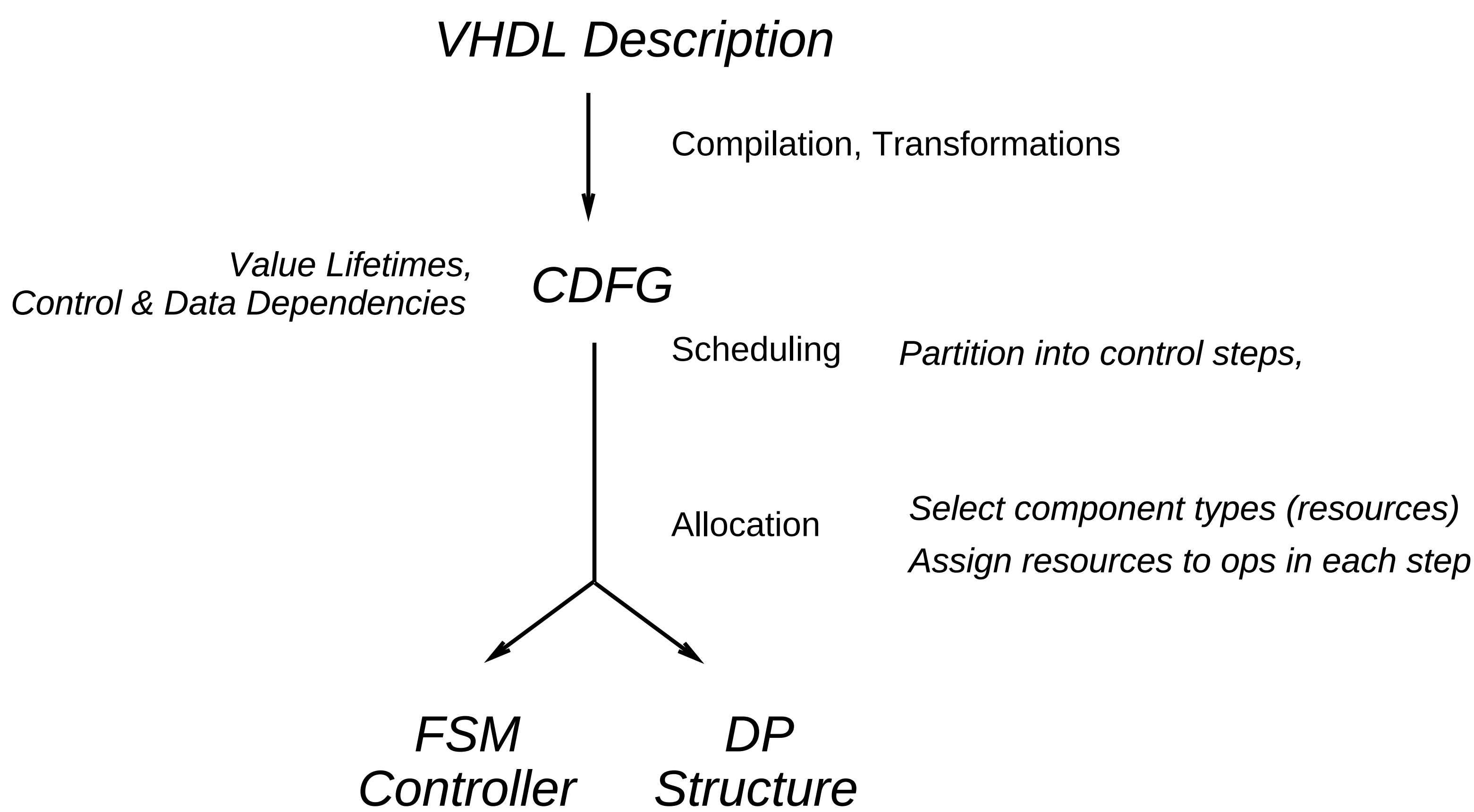
Kluwer Academic Publishers, 1992

ROLE OF INTERMEDIATE REPRESENTATION



- **Database for complete design information**
- **Uniform view across tools and users**
- **Language independent**
- **Support all architectural styles**

High−Level Synthesis Trajectory

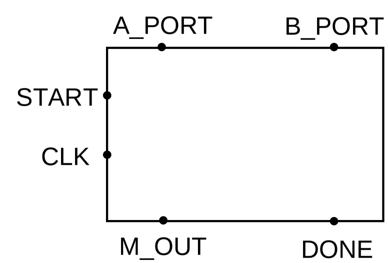


UC Irvine

SHIFT−MULTIPLIER: VHDL BEHAVIOR

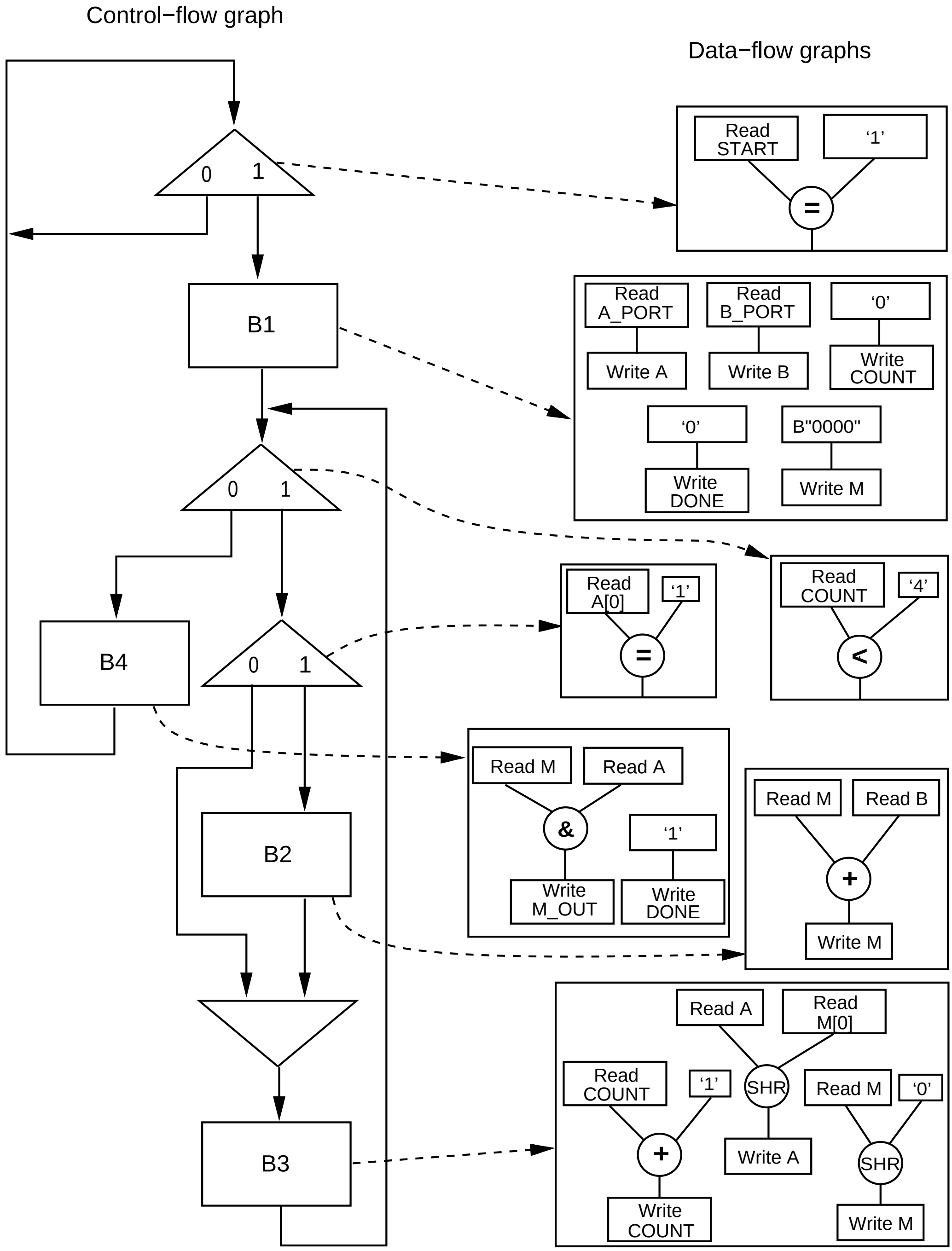
```
entity MULT is
 port ( A_PORT,
        B_PORT:  in bit_vector(3 downto 0);
        M_OUT:  out bit_vector(7 downto 0);
        CLK:  in CLOCK;
        START:  in BIT;
        DONE:  out BIT;
      );
end MULT;
```

```
architecture SHIFT_MULT  of  MULT  is
begin
    process
        variable A, B, M  :  BIT_VECTOR;
        variable COUNT  :  INTEGER;
    begin
        wait until (START = 1);
        A := A_PORT;   COUNT := 0;
        B := B_PORT;   DONE <= '0';
        M := B"0000";
        while  (COUNT < 4)  loop
            if (A(0) = '1')  then
                    M  :=  M + B;
            end if;
            A := SHR(A, M(0));
            M := SHR(M, '0');
            COUNT := COUNT + 1;
        end loop;
        M_OUT  <=  M & A;
        DONE <= '1';
    end process;
end SHIFT_MULT;
```



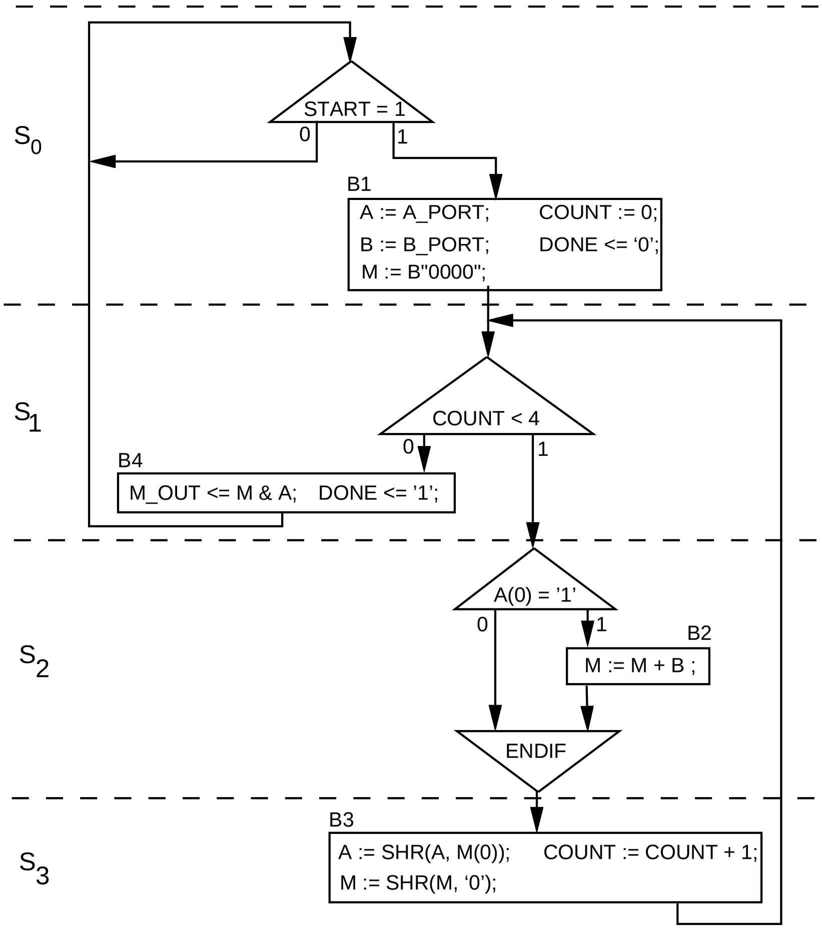
DESIGN FLOW IN HLS

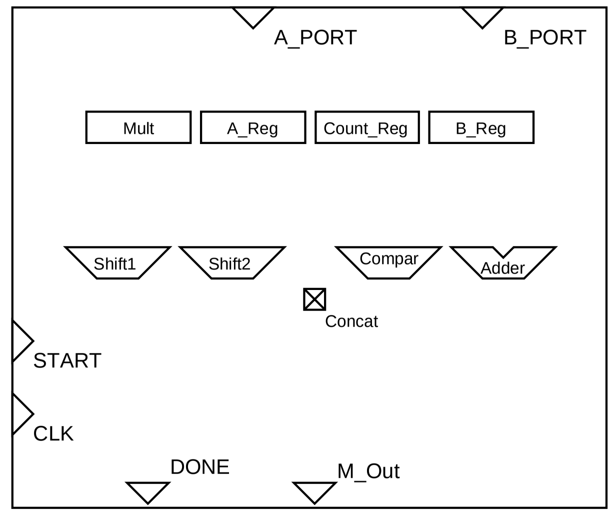
(Shift−Multiplier Example)



UC Irvine

SCHEDULED CDFG *(Shift−Multiplier Example)*



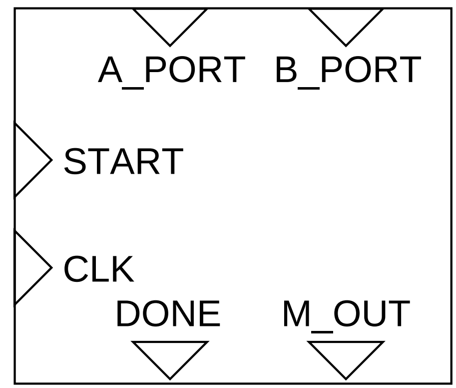


Initial Allocation

DESIGN VIEW: *Finite State Machine with DataPath (FSMD)*

Present State	Condition	Value	Actions	Next State
S0	START = 1	T	A := A_PORT; B := B_PORT; COUNT := 0; DONE := '0'; M := 0000";	S1
		F		S0
S1	COUNT < 4	T		S2
		F	M_OUT := M @ A; DONE := '1';	S0
S2	A(0) = 1	T	M := M + B;	S3
		F		S3
S3			A := SHR(A, M(0)); M := SHR(M. '0'); COUNT := COUNT + 1;	S1

FSMD state table



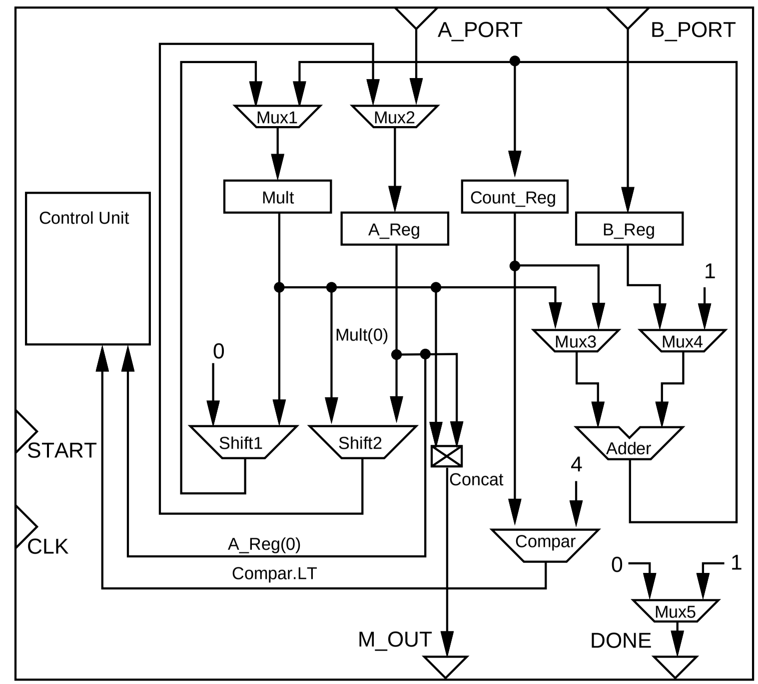
I/O ports

UC Irvine

SHIFT−MULTIPLIER: AFTER ALLOCATION

Present State	Condition	Value	Actions	Next State
S1	Compar.LT	1		S2
		0	Concat(OP: concat, INPS: Mult, A_Reg); M_OUT(OP: load, INPS: Concat); Mux5(OP: c1, INPS: '0', '1'); DONE(OP: load, INPS: Mux5);	S0
S2	A_Reg(0)	1	Mux3(OP: c0, INPS: Mult, Count_Reg); Mux4(OP: c0, INPS: B_Reg, "0001"); Adder(OP: add, INPS: Mux3, Mux4); Mux1(OP: c1, INPS: Shift1, Adder); Mult(OP: load, INPS: Mux1);	S3
		0		S3

Component−based state table

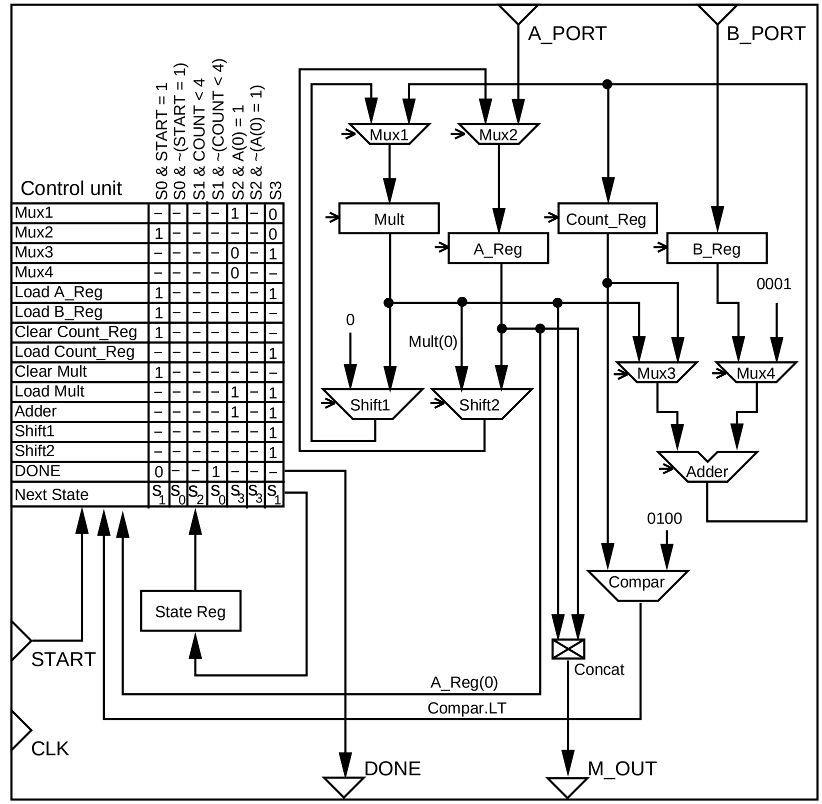


Partial design

SHIFT–MULTIPLIER: *After Control Generation*

Present State	Condition	Value	Actions	Next State
S1	Compar.LT	1		S2
		0	DONE := 1;	S0
S2	A_Reg(0)	1	Mux3.sel := 0; Mux4.sel := 0; Adder.add := 1; Mux1.sel := 1; Mult.load := 1;	S3
		0		S3

Symbolic Control Table



Control unit	S0 & START = 1	S0 & ~(START = 1)	S1 & COUNT < 4	S1 & ~(COUNT < 4)	S2 & A(0) = 1	S2 & ~(A(0) = 1)	S3
Mux1	–	–	–	–	1	–	0
Mux2	1	–	–	–	–	–	0
Mux3	–	–	–	–	0	–	1
Mux4	–	–	–	–	0	–	–
Load A_Reg	1	–	–	–	–	–	1
Load B_Reg	1	–	–	–	–	–	–
Clear Count_Reg	1	–	–	–	–	–	–
Load Count_Reg	–	–	–	–	–	–	1
Clear Mult	1	–	–	–	–	–	–
Load Mult	–	–	–	–	1	–	1
Adder	–	–	–	–	1	–	1
Shift1	–	–	–	–	–	–	1
Shift2	–	–	–	–	–	–	1
DONE	0	–	–	1	–	–	–
Next State	S_1	S_0	S_2	S_0	S_3	S_3	S_1

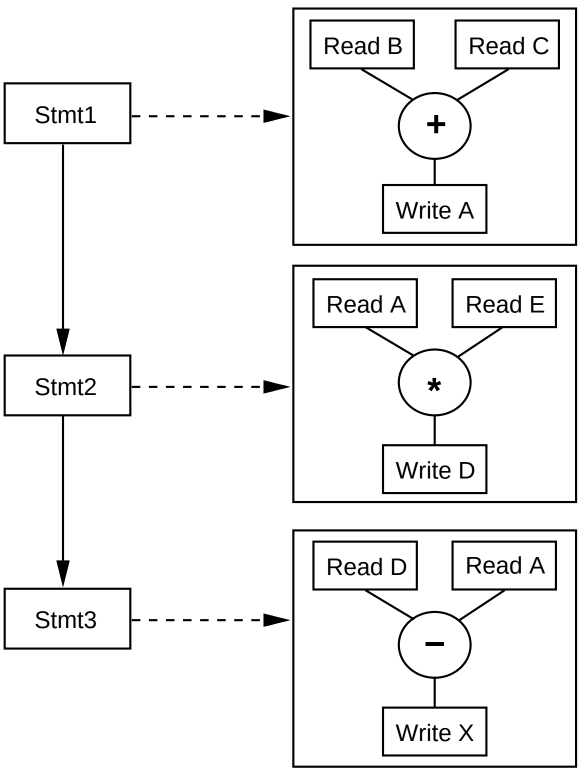
Complete Design

UC Irvine

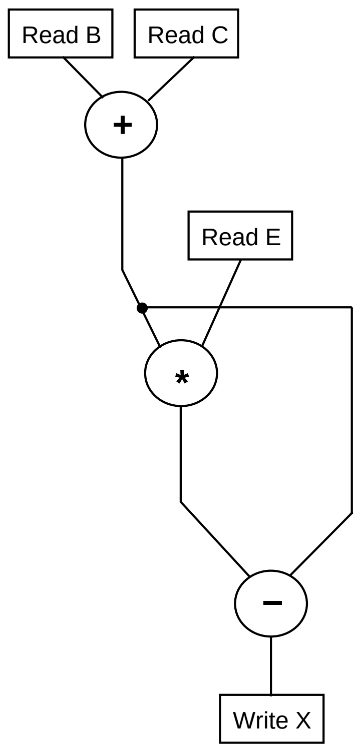
HDL COMPILATION

```
A := B + C;
D := A * E;
X := D − A;
```

HDL description



Parse tree

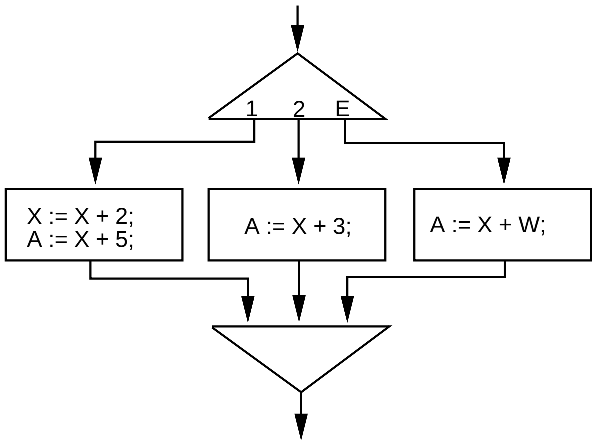


DFG

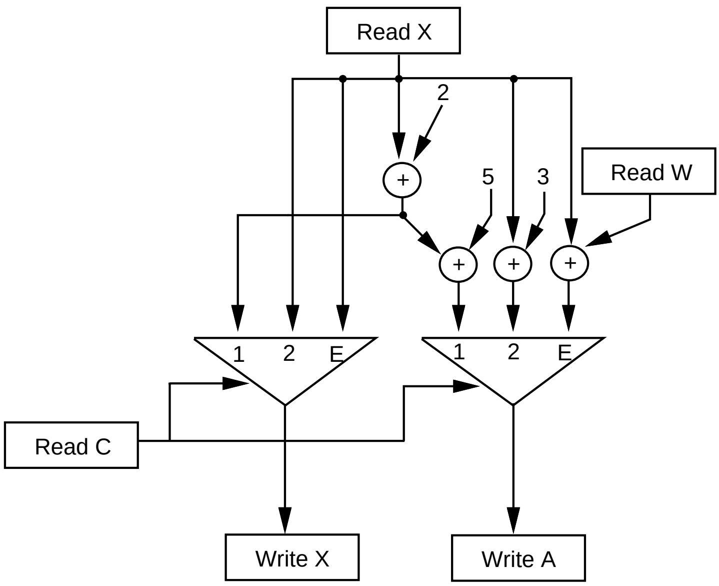
CONTROL AND DATAFLOW REPRESENTATION

```
case C is
     when 1 => X := X + 2;
               A := X + 5;
     when 2 => A := X + 3;
     when others => A := X + W;
end case;
```

VHDL Description



Control flow representation

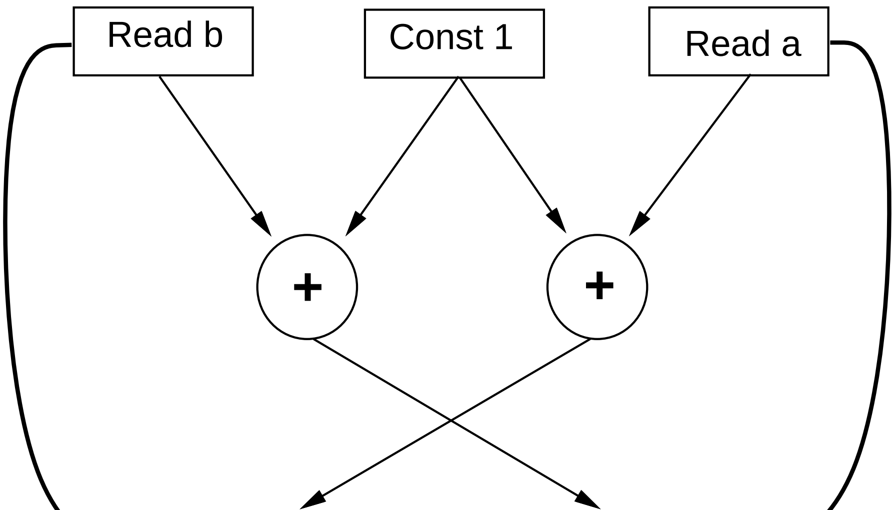


Dataflow representation

DATAFLOW WITH PRECEDENCE ARCS

```
b <= a + 1;
a <= b + 1;
```

Concurrent VHDL



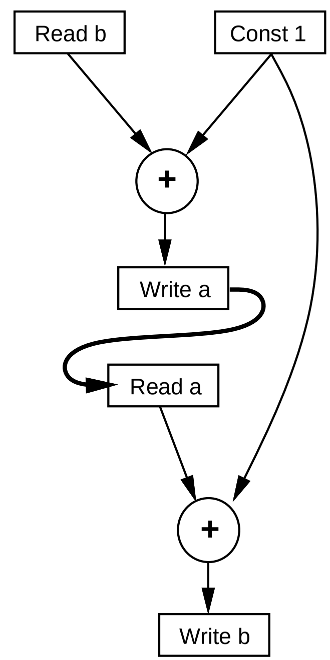
Representation

UC Irvine

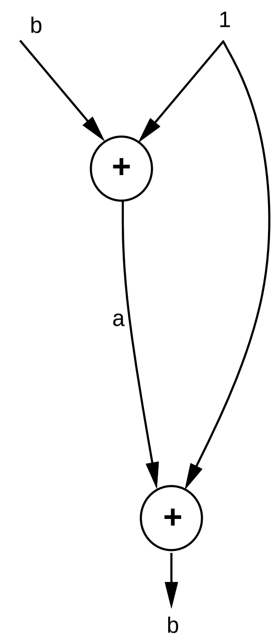
VARIABLE ACCESS REPRESENTATION

```
a := b + 1;
b := a + 1;
```

Seq. VHDL

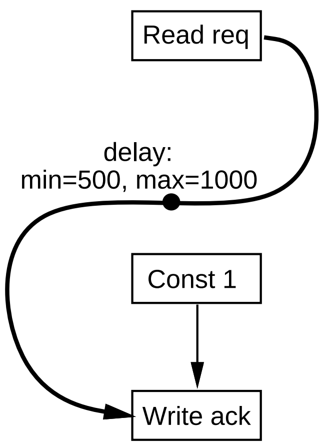


DFG with variable nodes

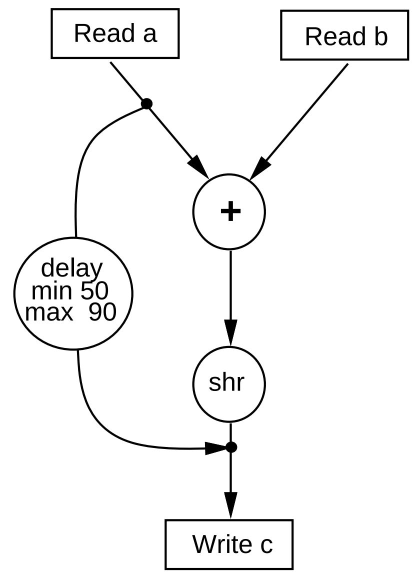


DFG with variable traces

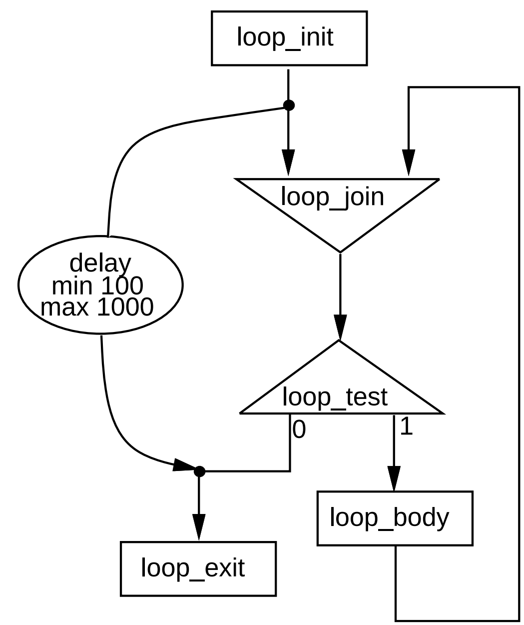
TIMING REPRESENTATION



Dataflow annotation



Timing in DFG

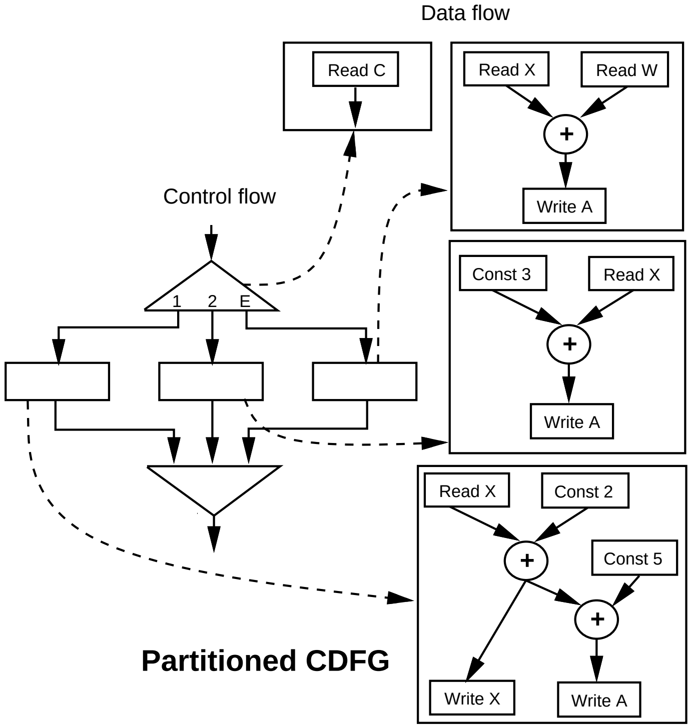


Timing in CFG

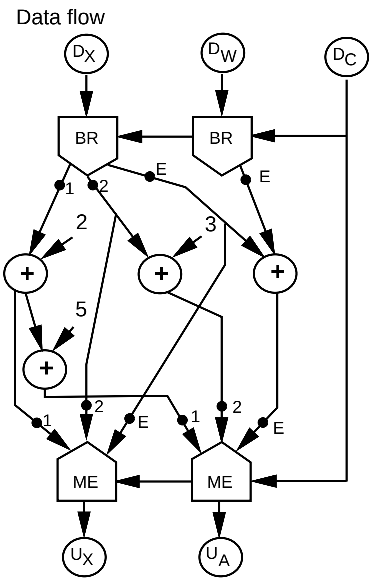
OTHER FLOW GRAPH REPRESENTATIONS

```
1. case C is
2.     when 1 =>  X := X + 2;
3.                A := X + 5;
4.     when 2 =>  A := X + 3;
5.     when others => A := X + W;
6. end case;
```

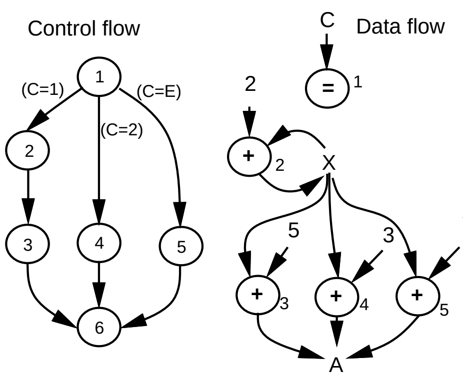
VHDL Description



Partitioned CDFG



DeJong's hybrid flow graph



SSIM flow graph

UC Irvine

VHDL PROCESS REPRESENTATION

Process <=> Control/Data Flow Graph (CDFG)

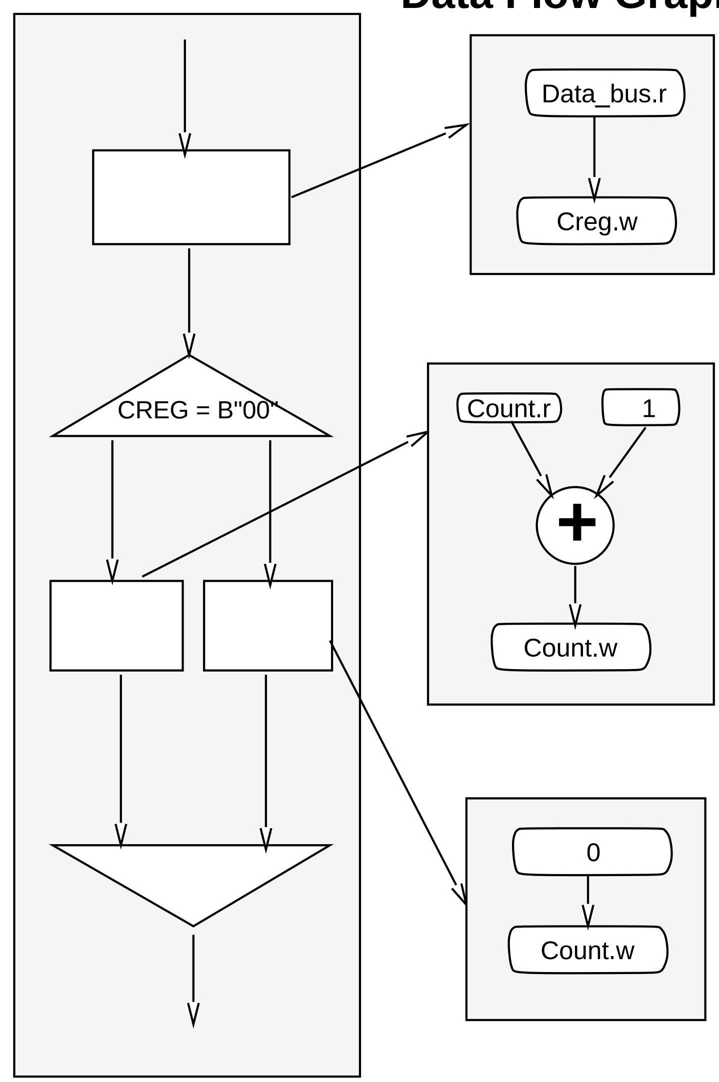
Sequential execution

VHDL process

```
P1: process
begin

   CREG := DATA_BUS;

   if  CREG = B"00"  then
        COUNT := COUNT + 1;
   else
        COUNT := 0;
   end if

end process;
```

Control Flow Graph

Data Flow Graph



VHDL BLOCK REPRESENTATION

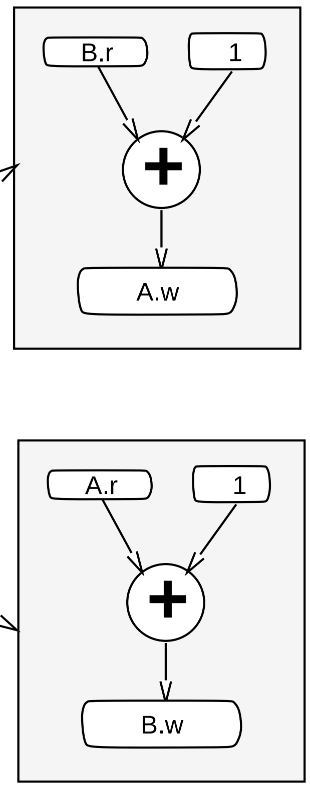
VHDL block <=> Directed acyclic Graph (DAG)

Parallel execution

VHDL Block

```
architecture swap of design is
begin
L: block (clock'event AND clock = '1')
    begin

    L1:  A  <=  guarded  B + 1;

    L2:  B  <=  guarded  A + 1;

    end L;
end swap;
```

DAGs



UC Irvine

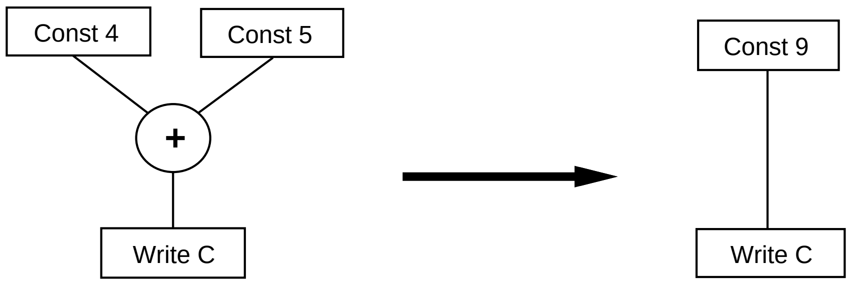
TRANSFORMATIONS

1. Compiler transformations

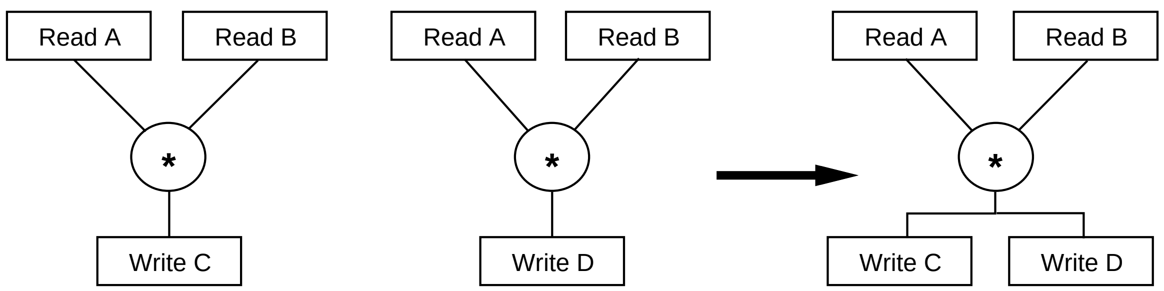
2. Flowgraph transformations

3. Hardware transformations

COMPILER TRANSFORMATIONS



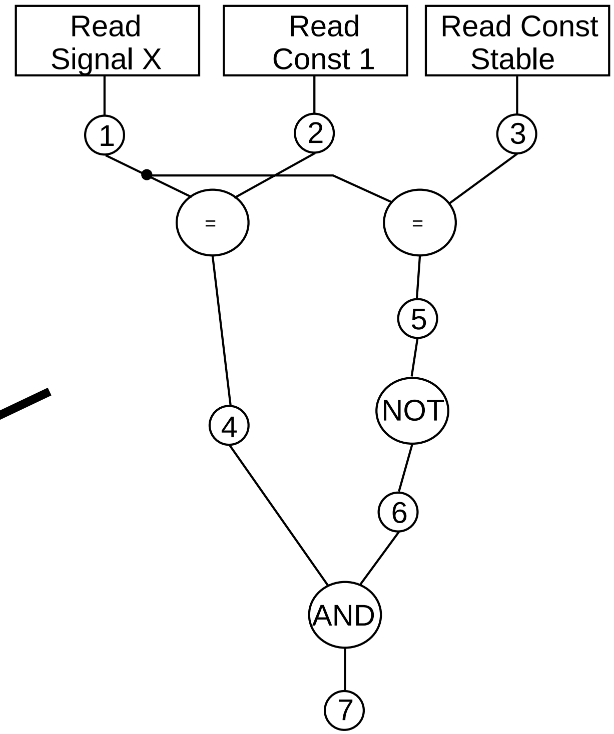
Constant folding

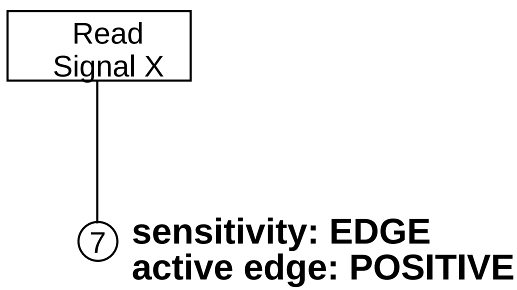


Redundant operator elimination

SIGNAL ATTRIBUTE TRANSFORMATIONS

X = 1 and not(X'stable)



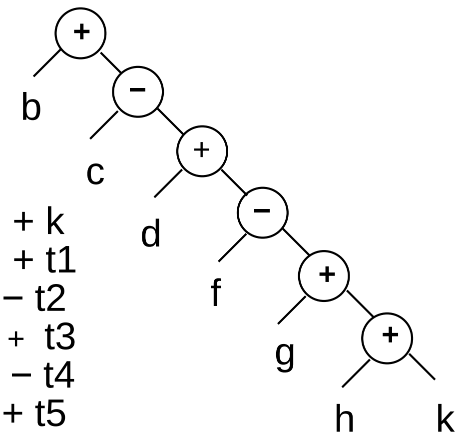


UC Irvine

TREE–HEIGHT TRANSFORMATION

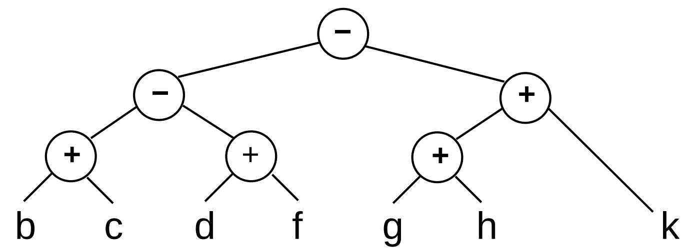
```
a := b + c − d + f − g + h + k;
```

VHDL code



t1 := h + k
t2 := g + t1
t3 := f − t2
t4 := d + t3
t5 := c − t4
a := b + t5

Initial parse tree



t1 := b + c
t2 := d + f
t3 := g + h
Potential parallelism

t4 := t1 − t2
t5 := t3 + k
a := t4 − t5
Potential parallelism

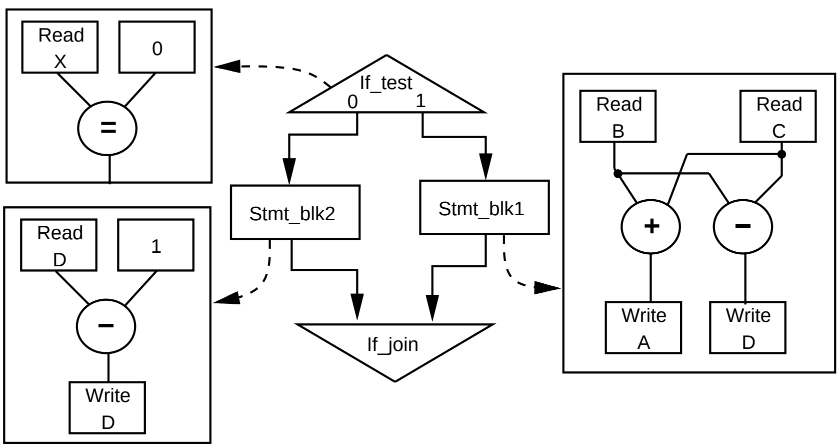
After tree height reduction

UC Irvine

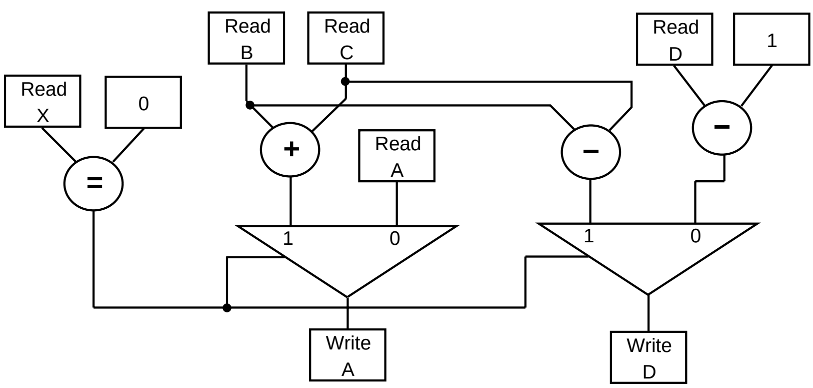
CONTROL–TO–DATAFLOW TRANSFORMATIONS

```
if (X = 0) then
      A := B + C;
      D := B − C;
else
      D := D − 1;
end if;
```

Textual representation



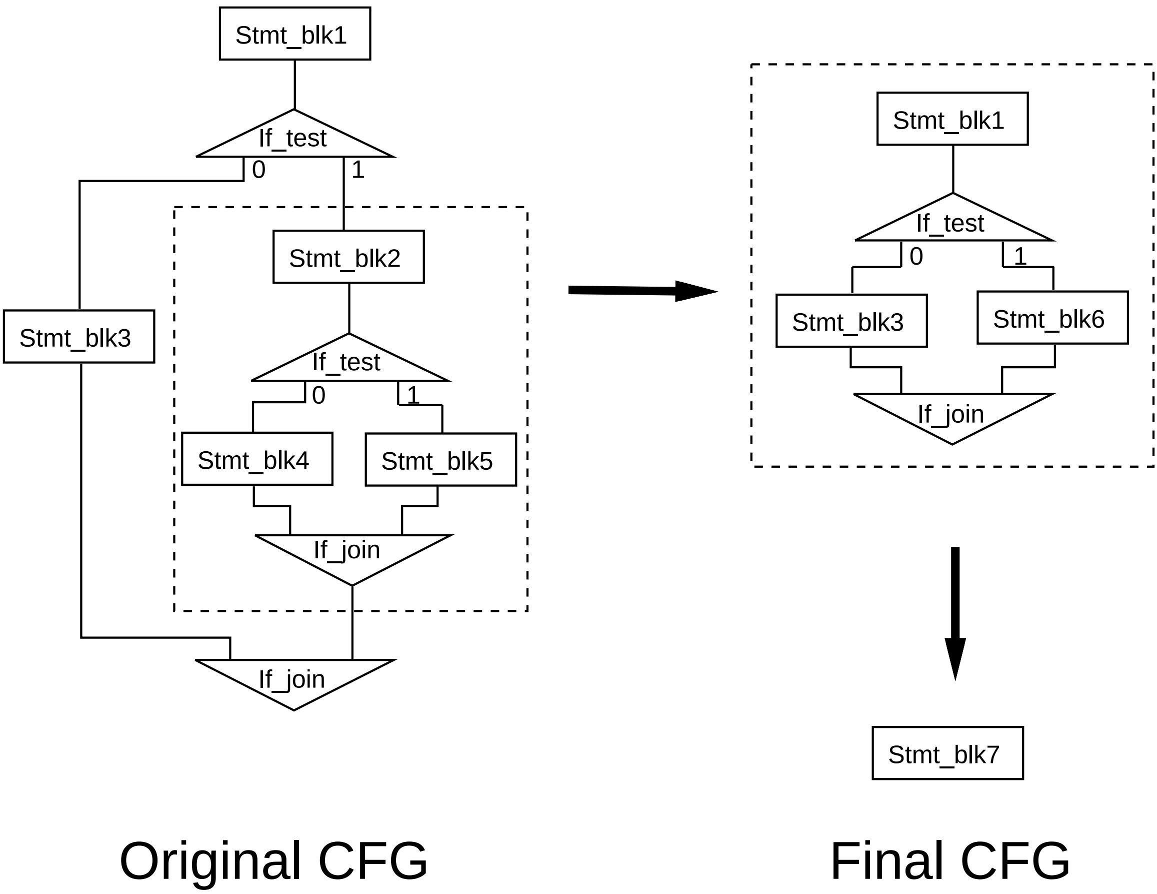
CF representation



DF representation

UC Irvine

CONTROL FLATTENING

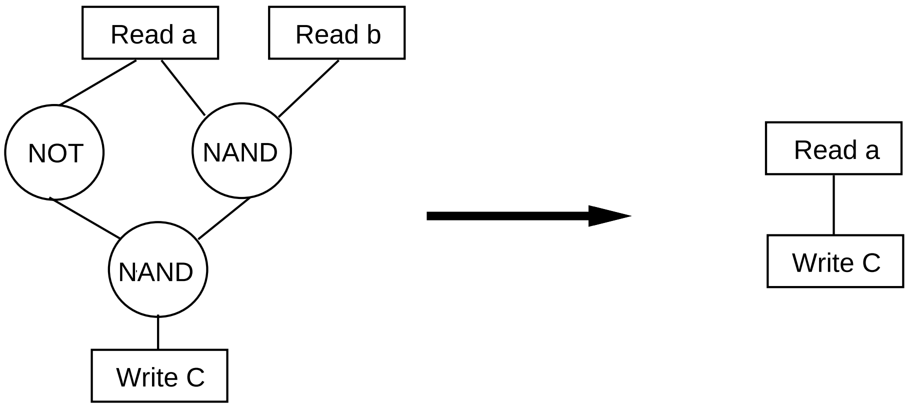


UC Irvine

LOGIC−LEVEL TRANSFORMATIONS

c = (a’ NAND (a NAND b)) = a

HDL boolean expression

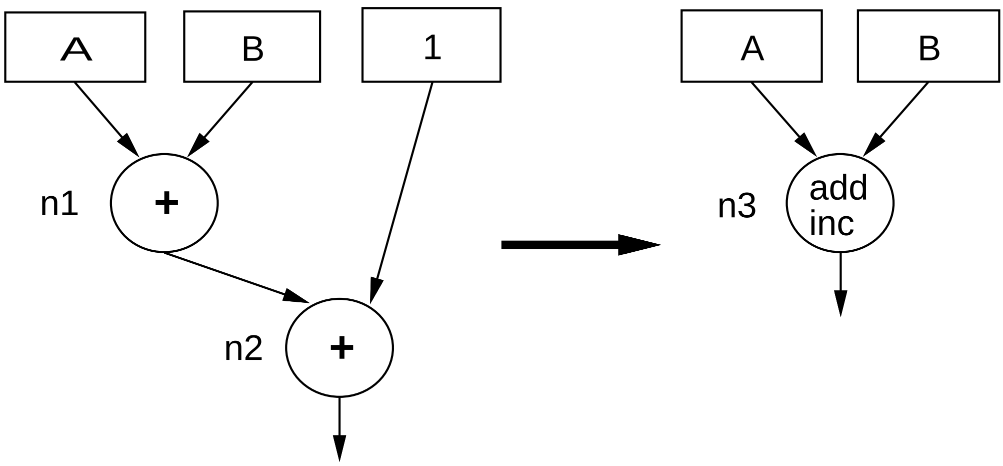


Original flow graph **After logic optimization**

RT TRANSFORMATION

RT−level function recognition

RT−Component specific transformation

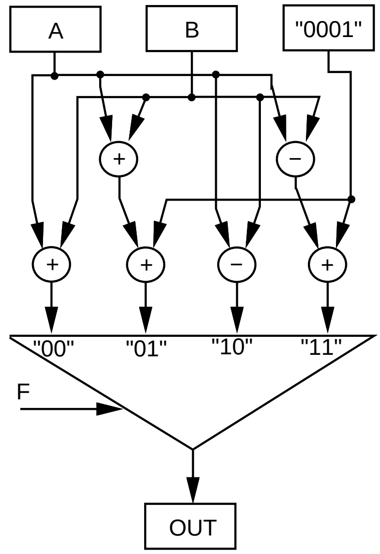


Adder/Incrementer Transformation

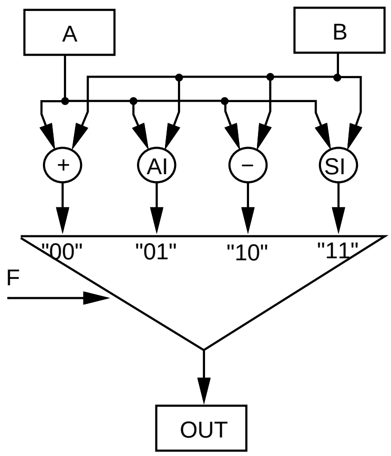
COMPLEX FUNCTION RECOGNITION

```
case F is
        when "00" => OUT <= A + B;
        when "01" => OUT <= A + B + "0001";
        when "10" => OUT <= A − B;
        when "11" => OUT <= A − B + "0001";
end case;
```

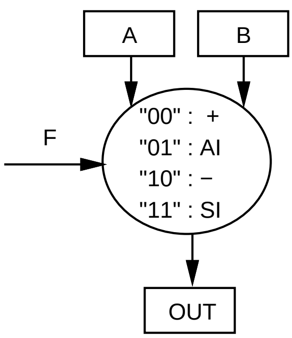
VHDL



DF graph



Simplified DF graph



Complex−node DF graph

FUTURE DIRECTIONS

1. Common representation
2. Different views
3. Different architectures
4. Description disambiguation
5. Layout−Driven Transformations
6. Transformation scripts
7. Representation for interactive design