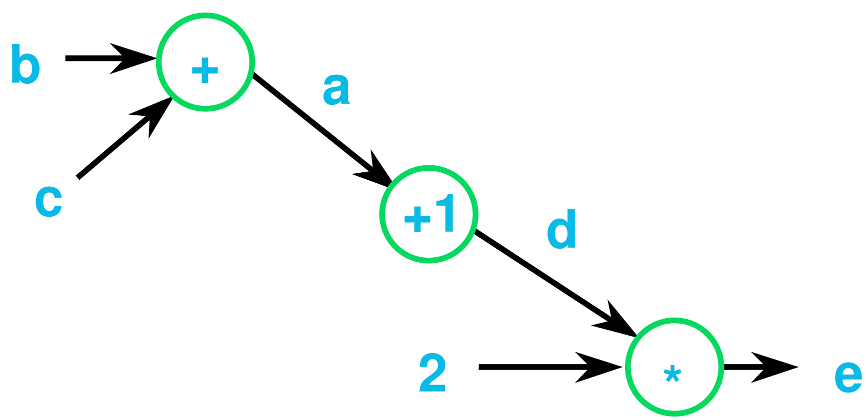
RTL versus Architectural Levels

An RTL description is structural whereas an architectural description is behavioral

a = b + c;
d = a + 1;
e = d * 2;

RTL implies



Behavioral implies

e = 2 * (b + c + 1)

RTL Synthesis

Two Steps:

1. **Translation from RTL description into gates**

2. **Optimization of logic**

Ideally, all of the intelligence should be in the optimization step

Unfortunately, starting point for optimization affects results

$\Rightarrow$ **Need to write a "good" RTL description**

VHDL

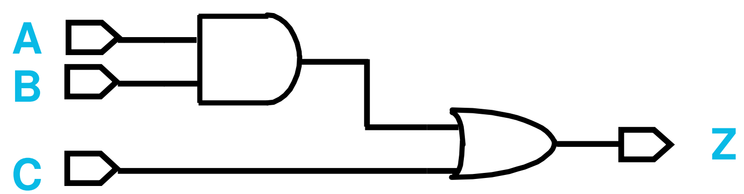
```
entity VHDL is
   port C (
          A, B, C  :  in BIT ;
          Z  : out BIT ;
   ) ;
end VHDL ;
```

entity and port declarations define interface

```
architecture VHDL_1 of VHDL is
begin
      Z ⇐   ( A and B )  or C  ;
end VHDL_1 ;
```

defines implementation

logic expression

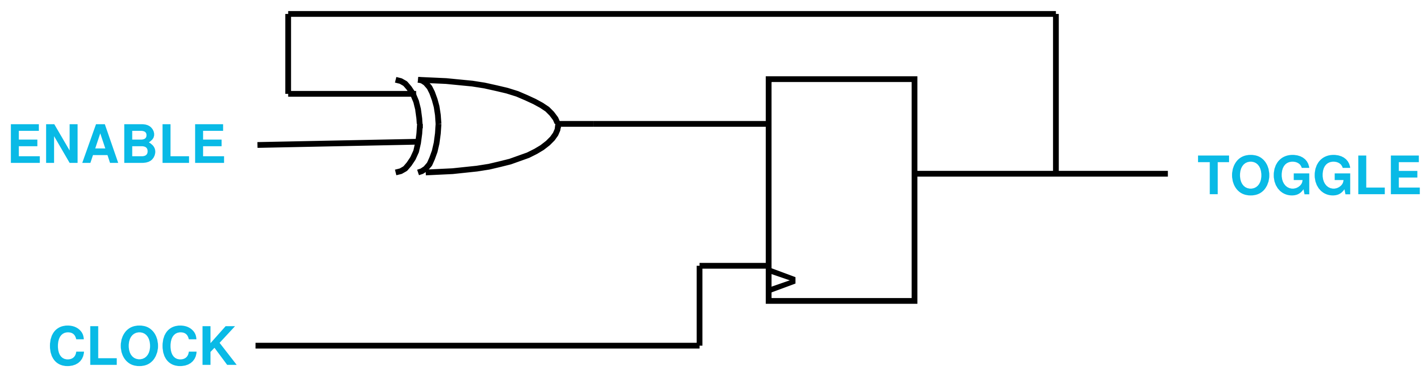


Wait Statements

```
process begin
    wait until CLOCK' event and CLOCK = '1' ;
    if  (ENABLE = '1') then
            TOGGLE = not TOGGLE;
    end if;

end process;
```

wait infers a flip-flop



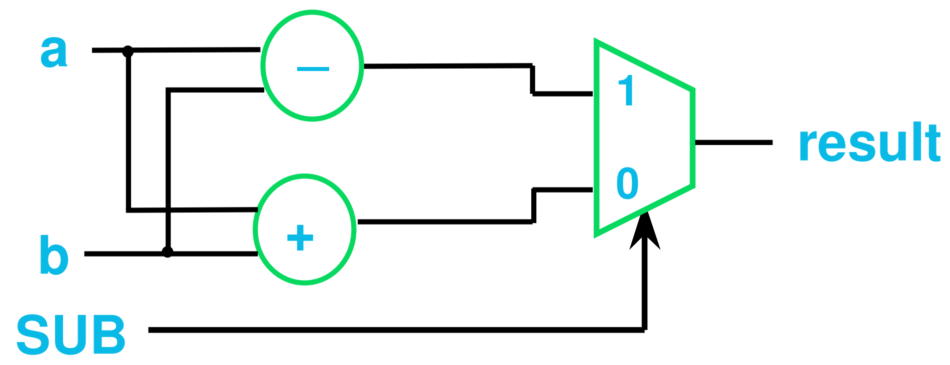
Writing RTL Descriptions

Current scenario: Designer iterates over RTL descriptions using simulation and synthesis tools to verify and evaluate design

PROBLEM: Equivalent RTL descriptions may result in dramatically different logic circuits in terms of area/performance EVEN AFTER OPTIMIZATION.

Equivalent RTL Descriptions

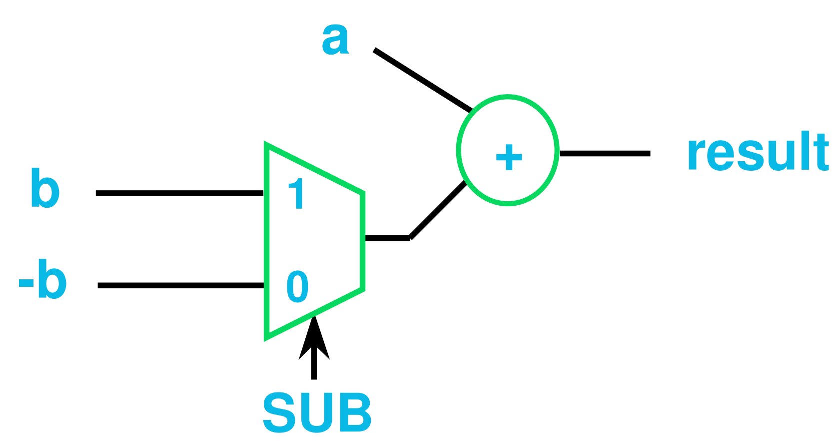
```
if (SUB)
  result = a - b;
else
  result = a + b;
```



push multiplexors to inputs

```
if (SUB)
  b = -b;

result = a + b;
```

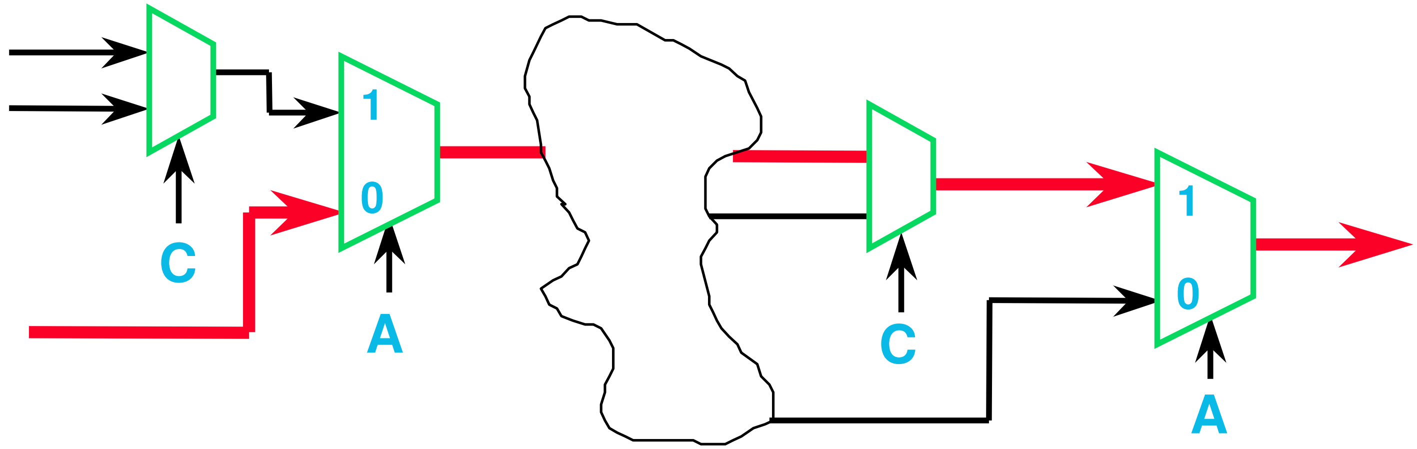


False Paths

```
if (A) then {
    if (C) then

      ...
}
if (A) then {
    if (C) then {
    }
      ...
}
```

avoid unnecessary nested if statements



Practical Solution

1. Increase power of optimization.
2. Define “good” input and restrict oneself to writing “good” input descriptions.
3. Manipulate input descriptions to make them “good”.

Rules for writing descriptions based on knowledge that a library of descriptions are good starting points for synthesis.

Behavioral Synthesis

Manipulating and optimizing RTL descriptions while maintaining required functionality but changing FSM behavior

Behavioral transformations

- **Parallelize, serialize or pipeline a behavioral description**
- **Allocate hardware; registers buses, ALUs to obtain a RTL description with a definite structure**

Behavioral Specification

Elliptic-filter (In, Out)

```
{
        i  = In
        a = i  +  t2
        b = a  +  t13
        g = t33  +  t39
        e = g  +  t26  +  b
        d = (m21 * e)  +  b
        f = (m24 * e)  +  g
            ...
            ...
        Out = o
}
```

Hardware Allocation

Treat the given description as a “dataflow” type specification.

Synthesize a datapath by first constructing a schedule of operations, that will require a certain amount of hardware, and then synthesize a controller given the datapath and associated schedule.

Control is viewed totally as a by-product of datapath synthesis.

Example Behavior

V3 = V1 + V2
V5 = V3 - V4
V7 = V3 * V6
V8 = V3 + V5
V9 = V1 + V7
V11 = V10 / V5
V13 = V3
V12 = V1
V14 = V11 and V8
V15 = V12 or V9
V1 = V14
V2 = V15

Schedule - I

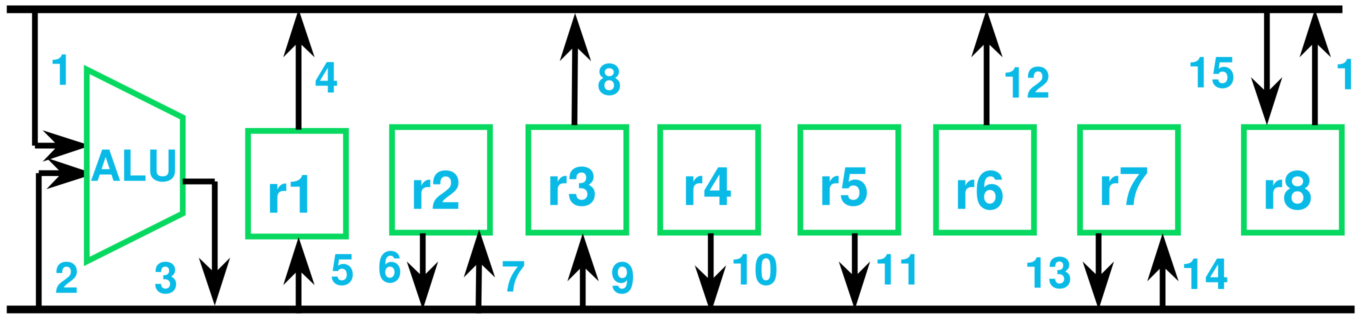
A schedule:

```
r3 = r1 + r2                    r8 = r1
r7 = r3 - r4
r2 = r3 * r5
r3 = r3 + r7
r2 = r1 + r2
r7 = r6 / r7
r1 = r3 and r7
r2 = r2 or r8
```

V1 to V15 merged into r1 to r8

Datapath - I

ALU performs +, -, *, /, and, or



r3 = r1 + r2
r7 = r3 - r4
r2 = r3 * r5
r3 = r3 + r7
r2 = r1 + r2
r7 = r6 / r7
r1 = r3 and r7
r2 = r2 or r8

r8 = r1

Controller - I

Controller:

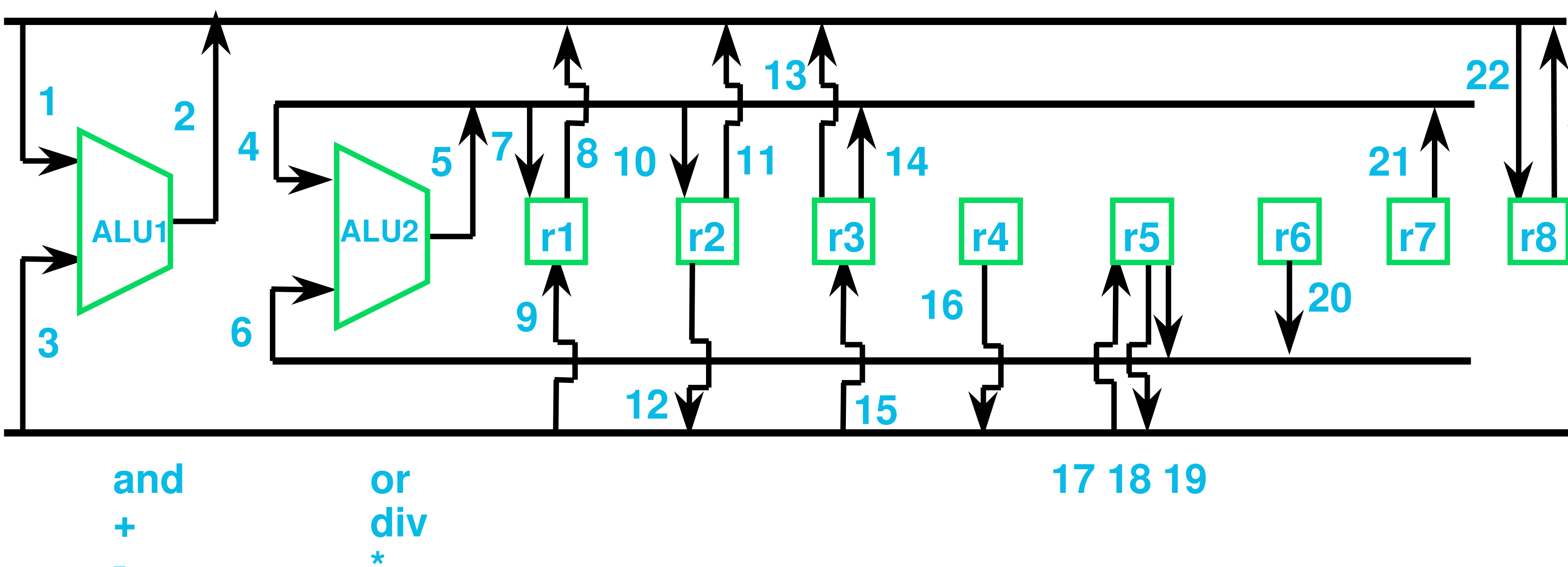
0	S0	S0	N0P	0000000000000000
1	S0	S1	N0P	0000000000000000
-	S1	S2	ADD	1111010010000010
-	S2	S3	SUB	1110000101000100
-	S3	S4	MUL	…
-	S4	S5	ADD	…
-	S5	S6	ADD	
-	S6	S7	DIV	
-	S7	S8	AND	
-	S8	S0	OR	…

Schedule and Datapath - II

r3 = r1 + r2
r5 = r3 - r4
r3 = r3 + r5
r2 = r1 and r2
r2 = r8 + r2

r8 = r1

r2 = r3 * r6
r5 = r5 * r7
r1 = r3 or r5



Controller - II

Controller

0	S0	S0	NOP	NOP	00000000000000000000000
1	S0	S1	NOP	NOP	00000000000000000000000
-	S1	S2	ADD	NOP	11100001000100100000010
-	S2	S3	SUB	MUL	...
-	S3	S4	ADD	DIV	...
-	S4	S5	AND	OR	
-	S5	S6	ADD	NOP	...

Allocation Problems

Arithmetic unit allocation: Decide the number of arithmetic units, as well as the grouping of arithmetic operators

e.g.,	ALU1: + , -
	ALU2: + , -, incr
	ALU3: * , shift
or	ALU1: + , - , incr , shift
	ALU2: *

Register allocation: Coalesce the variables into a minimum number of registers

Interconnect allocation: Allocate buses and links, to implement all the required data transfers between the registers and ALUs.

Placement Formulation

Call the X dimension (hardware) space and the Y dimension (execution) time

Space			
Time	Z1 = X1 + Y1	Z2 = X1 * Y2	Z3 = X2
	K1 = Z1 - X2	K2 = Z2 @ X1	
	L1 = Z2 or Z3		K3 = K2 + 1

Minimum number of ALUs:

- Maximum number of occupied space slots. Each ALU performs operations in its slot.

Minimum number of registers:

- Maximum density of variable lifetimes across all the time slots.

Interconnections related to the number of parallel data transfers in each time slot.

Register Allocation

Given a code sequence, coalesce the variables into a minimum number of registers.

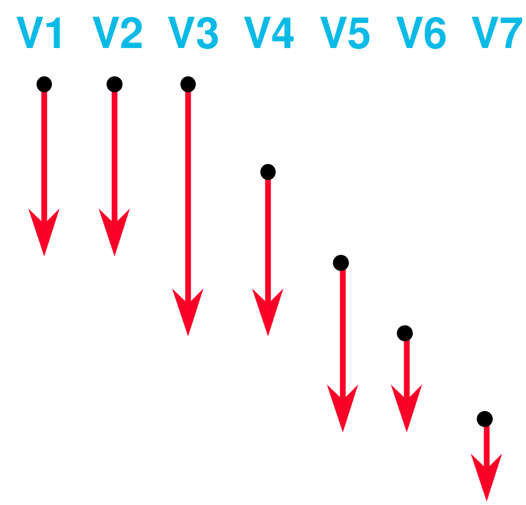
V1 = V2 + V3

V4 = V2 - V3

V5 = V1 * V2

V6 = V4 and V3

V7 = V5 or V6



Define the lifetime of a variable as the interval between its first assignment and last use.

Merge variables with non-overlapping lifetimes

Lifetimes are <u>open</u> intervals, at bottom end.

Register Allocation - II

Maximum density of variable lifetimes across all timeslots = minimum # registers required

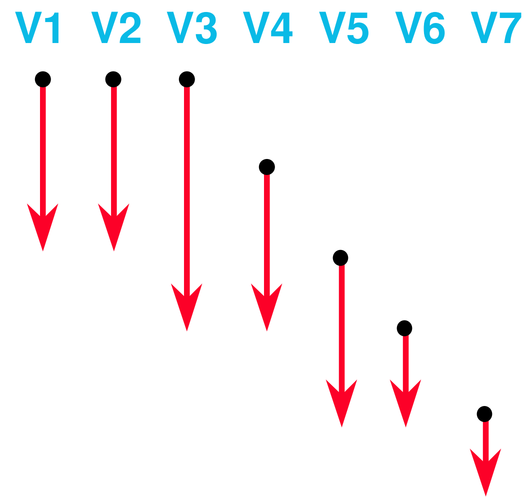
V1 = V2 + V3

V4 = V2 - V3

V5 = V1 * V2

V6 = V4 and V3

V7 = V5 or V6



Merge variables into # registers = maximum density

R1 = R2 + R3
R4 = R2 - R3
R1 = R1 * R2
R4 = R4 and R3
R4 = R1 or R4

(V1, V5) $\rightarrow$ R1   V2 $\rightarrow$ R2

(V4, V6, V7) $\rightarrow$ R4   V3 $\rightarrow$ R3

Different schedules have different max. densities.

Summary

RTL synthesis is mature and synthesis from RTL descriptions is a popular means of VLSI design

Behavioral synthesis is much less mature

- Some impact/use in DSP design (constrained problems: such as register allocation given a schedule)
- Zero impact on control dominated and general purpose computers