

9/11/23

ASIC : Application Specific Integrated Circuit

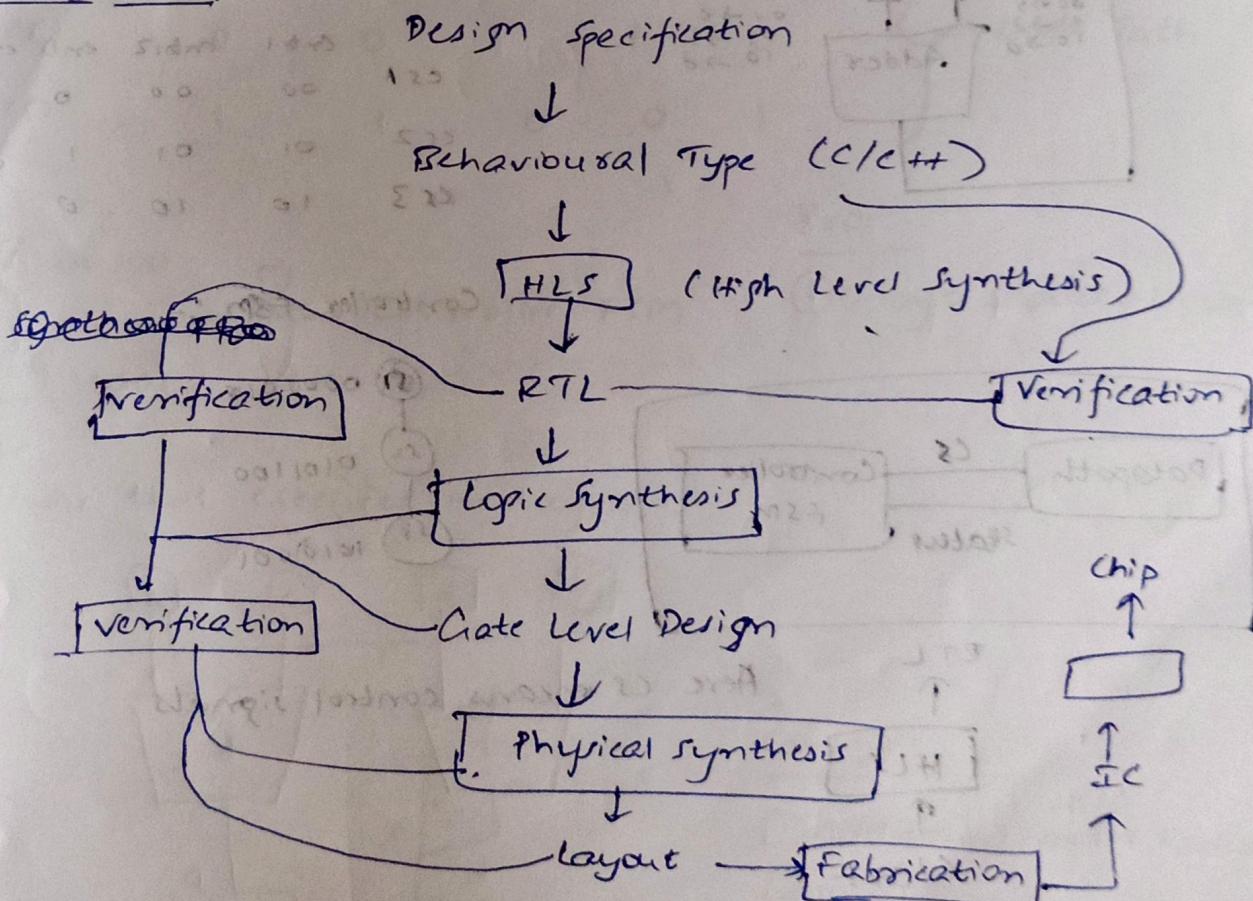
FPGA : Field Programmable gate ~~Analogy~~ Array

Difference b/w the usage of ASIC and FPGA comes due to the issue of scalability.

ASIC is used if we want to create ~~wast~~ number of copies.

FPGA is " " " " " few "

Synthesis Flow :-



$\text{fun}(b, cd) :$

$$* a = b + c$$

$$d = d + b$$

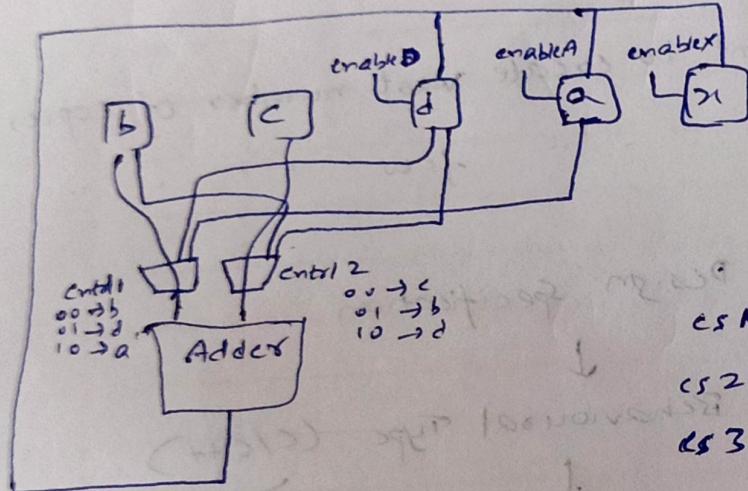
$$x = a - d$$

$$a = b + c$$

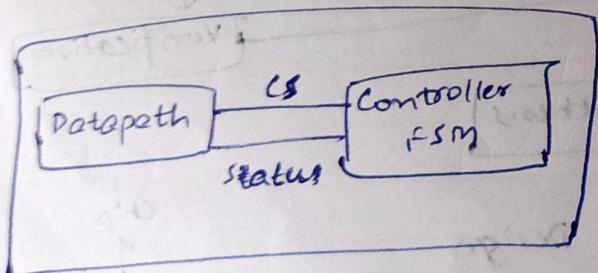
$$d = d + b$$

$$x = a - d$$

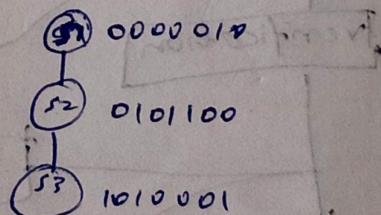
Hardware that executes above fun' in 3 clocks:-



	ctrl1	ctrl2	enD	enA	enX
cs1	00	00	0	1	0
cs2	01	01	1	0	0
cs3	10	10	0	0	1



Controller FSM :-



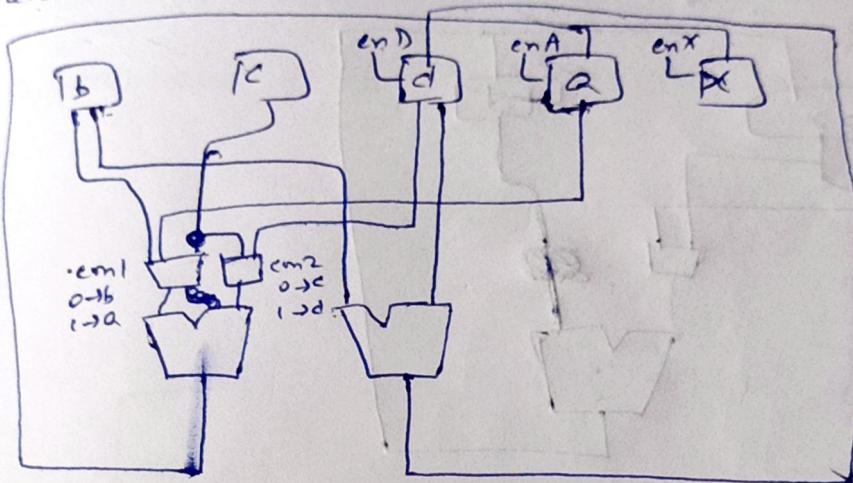
RTL

HLS

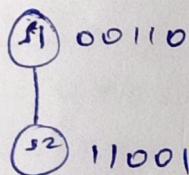
C/C++

Here CS means control signals

Hardware that executes in 2 clocks :-



	en1	en2	enD	enA	enX	
cs1	0	0	1	1	0	Data Path
cs2	1	1	0	0	1	

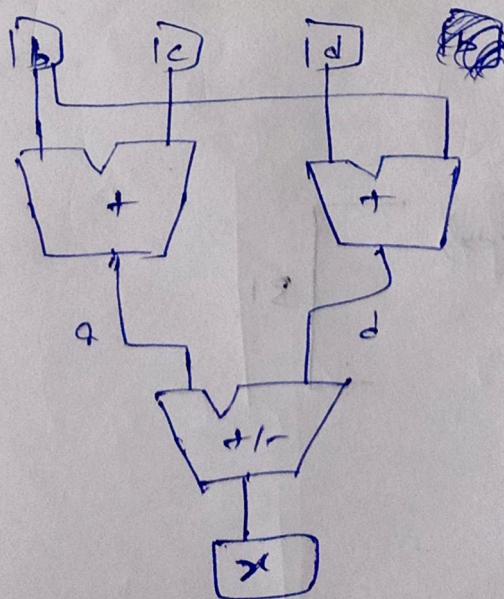


→ Controller FSM

Pros:-

- ① Delay = 1 Address + 1 mux

Hardware that executes in 1 cycle :-



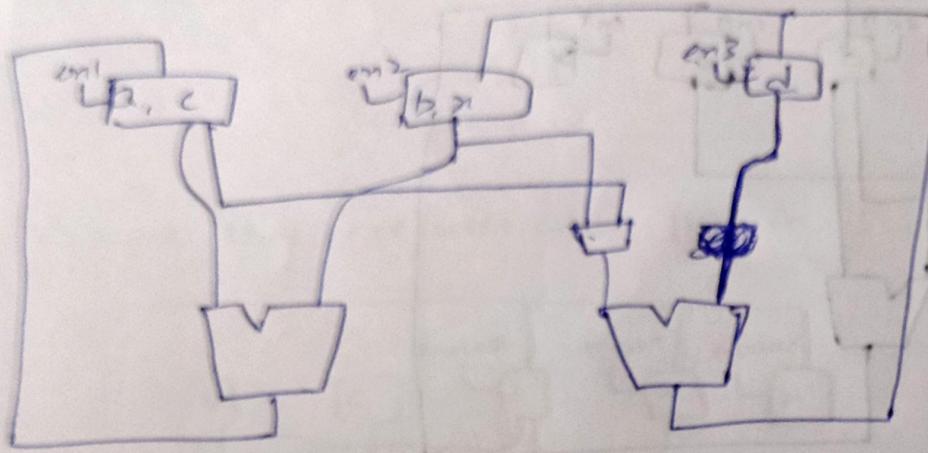
Cons:-

- ① Additional Adder Hardware
- ② Delay = 2 Address



So the time period
of the clock must
be > 2 Address

Design with few number of registers :-



2-cycle with 3 registers

(Along with interconnection optimisation)

ALS steps :-

- ① Construct control and data flow graph
- ② Split the operations into 3-address format
- ③ Identify data dependency among the operations

* Diffeg(x, dx, u, a, y) {

~~output exp~~

 while ($x \neq a$) {

$$u_1 = u - (3xu dx) - (3y dx)$$

$$y_1 = y + (u dx)$$

$$x = x + dx$$

$$y = u_1 \quad y = y_1$$

 return y

3

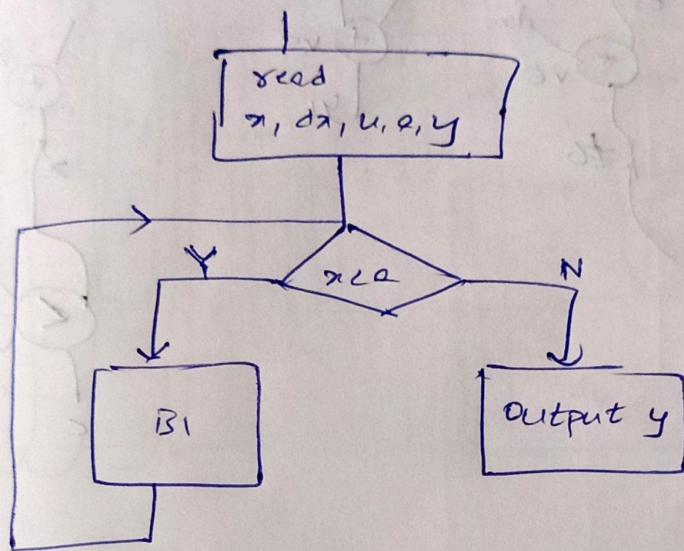
B1

C-Based VLSI

10/1/23

①

control and data flow graph :- (CDFG)



B1

$$v_1: t_1 = u * dx$$

$$v_2: t_2 = 3 * x$$

$$v_3: t_3 = 3 * y$$

$$v_4: t_4 = u * dx$$

$$v_5: t_5 = t_1 + t_2$$

$$v_6: t_6 = t_3 * dx$$

$$v_7: t_7 = u - ts$$

$$v_8: t_8 = t_7 - t_6$$

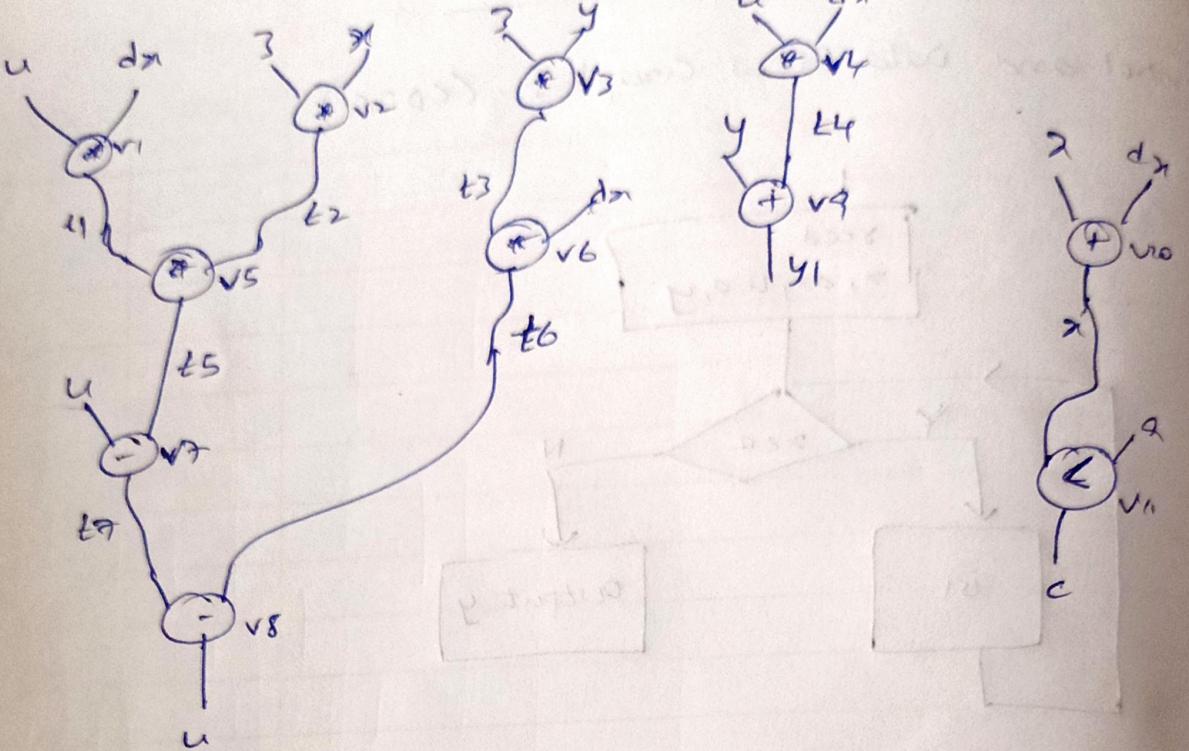
$$v_9: y_t = y + t_4$$

$$v_{10}: x = x + dx$$

$$v_{11}: c = x < a$$

$$v_{12}: y = y_1$$

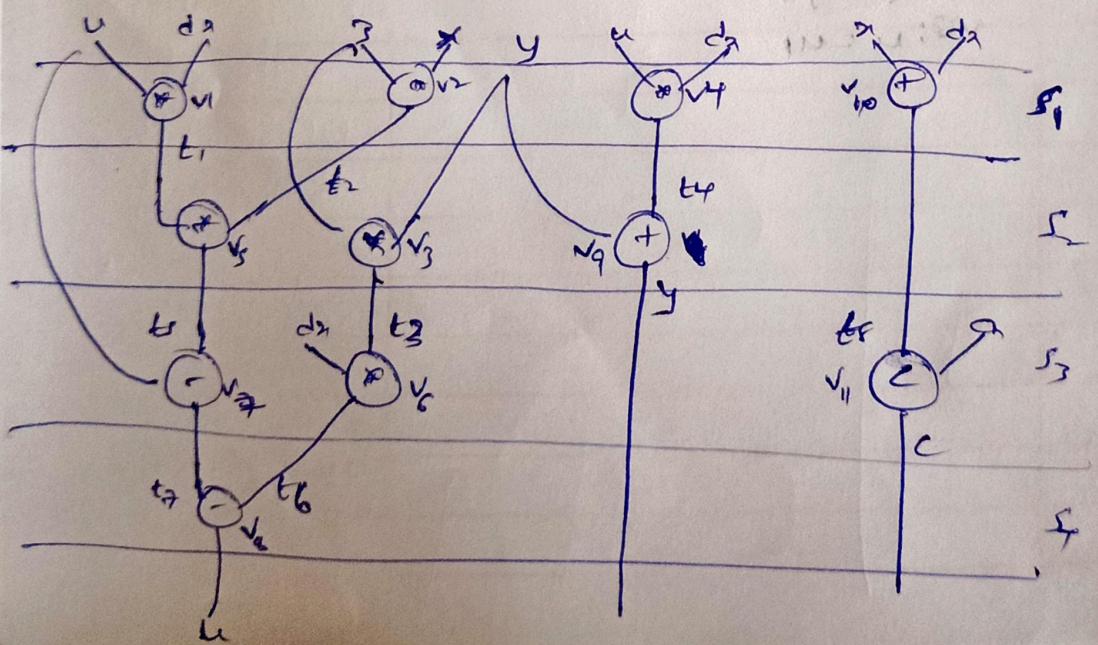
$$v_{13}: u = u_1$$



Data Dependency Graph

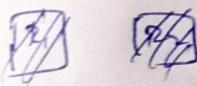
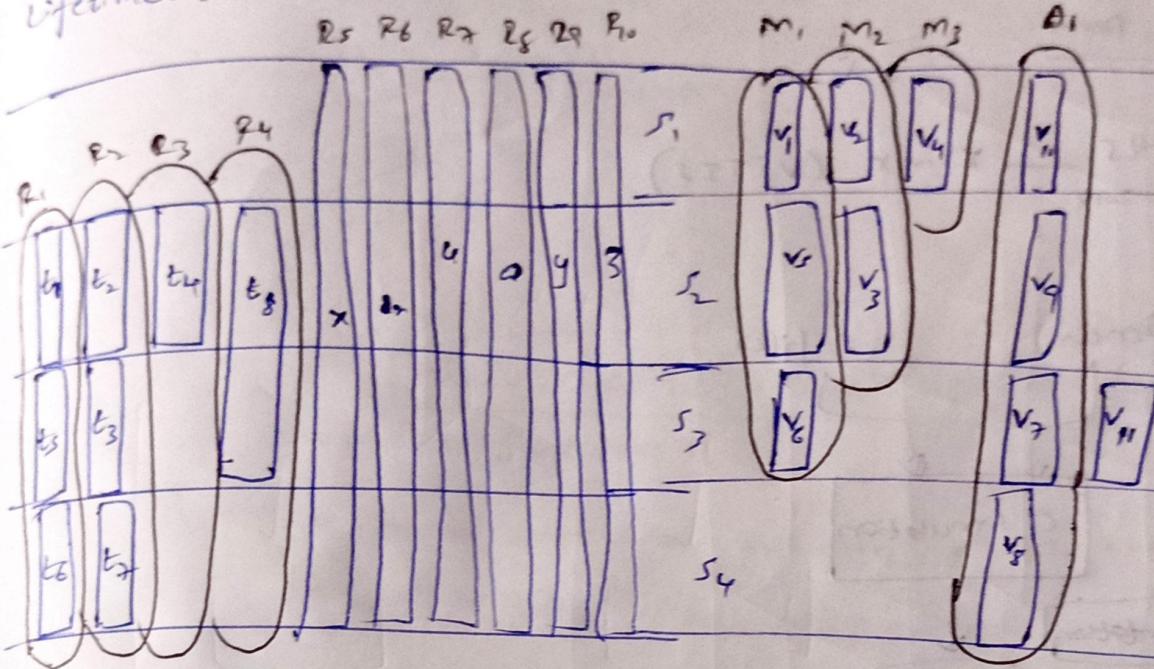
13/14/23

Scheduling: Assigning time step to operation
Allocation and Binding



③

lifetimes :-



16/11/23

* HDL Verilog

function \leftrightarrow modulevariable \leftrightarrow reg

Transistor Level



Gate Level

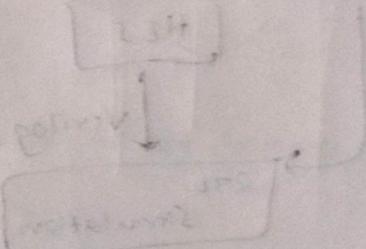


RTL / Data Flow



Behavioural Level

Verilog can be
written at any
of these levels

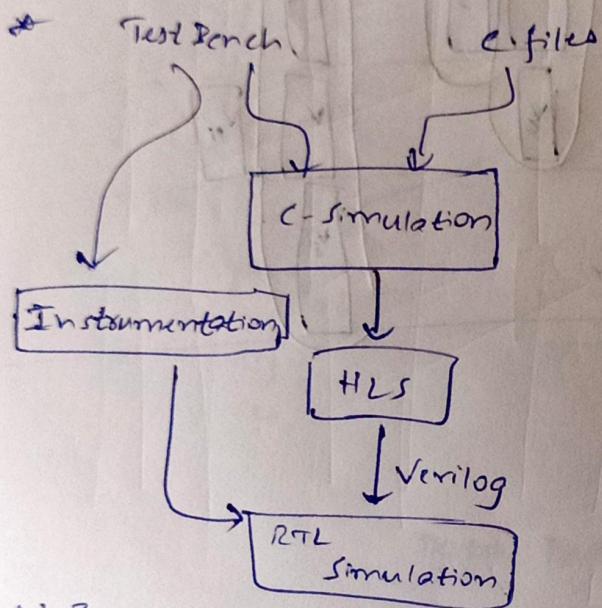


→ kind of statement
→ like verilog will support if
else, for loop
switch case
break = true
true number

17/11/23

HLS Tool Demo

Vivado HLS Xilinx (VITIS)
2019

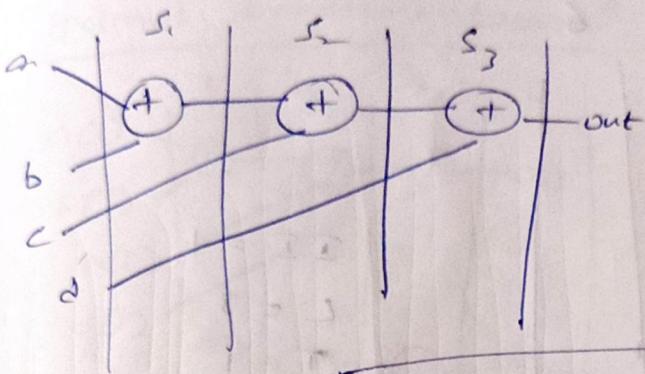


23/11/23

Functional Pipeline in HLS :-

```
accumulate(a,b,c,d){  
# Pregme HLS Pipeline II=2  
d1 = a+b  
d2 = d1+c  
out = d2+d  
return out
```

3



Now we require only one full adder

II = Initiation Interval

↳ Time period after which we can give another set of inputs (How often we can give inputs)

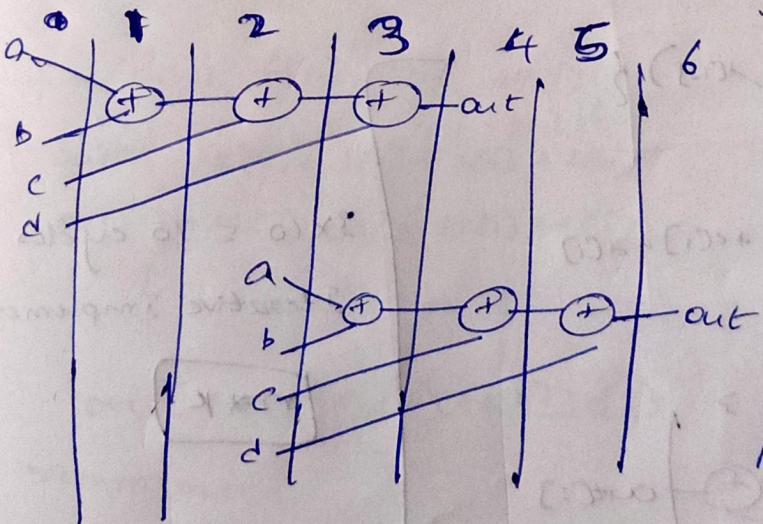
L = Latency

↳ Time required to process one set of inputs

T = Throughput

↳ How often you get output from a design

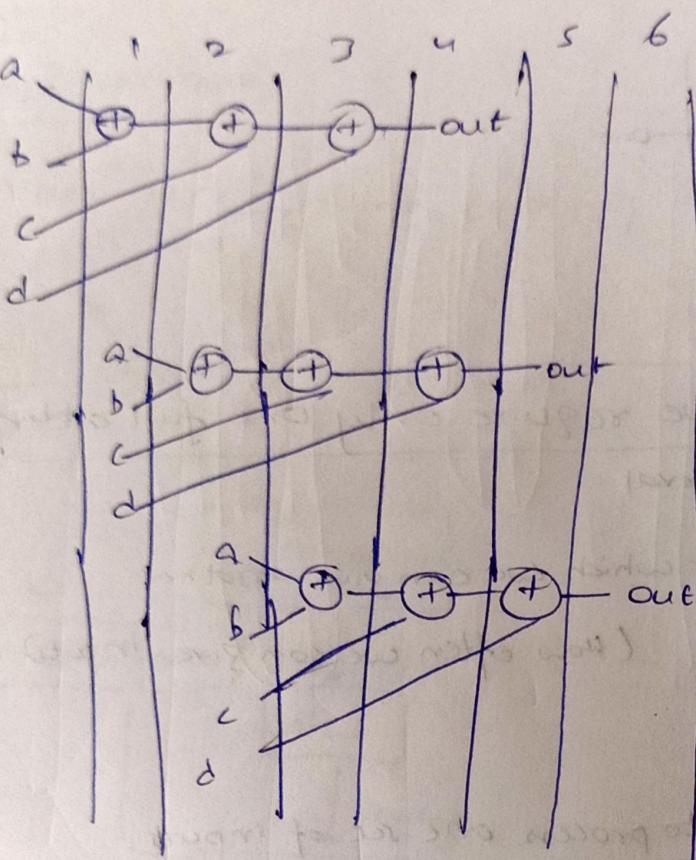
For the above design $II = L = T = 3$ cycles



$$II = 2$$

$$L = 3$$

$$T = 2$$



$$\Sigma I = 1$$

$$L = 3$$

$$T = 1$$

Fully Pipelined Design

24/1/23

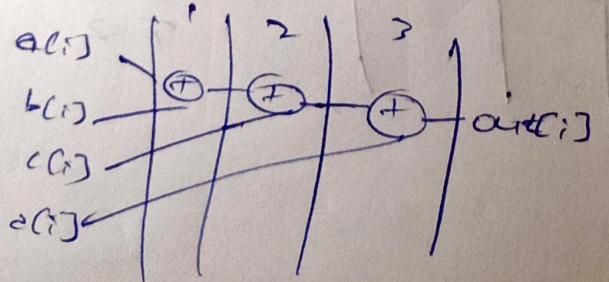
HLS for loop

```
void fn(a[10], b[10], c[10], d[10]) {
    for(i=0; i<10; i++) {
        out[i] = a[i] + b[i] + c[i] + d[i]
    }
    return out
}
```

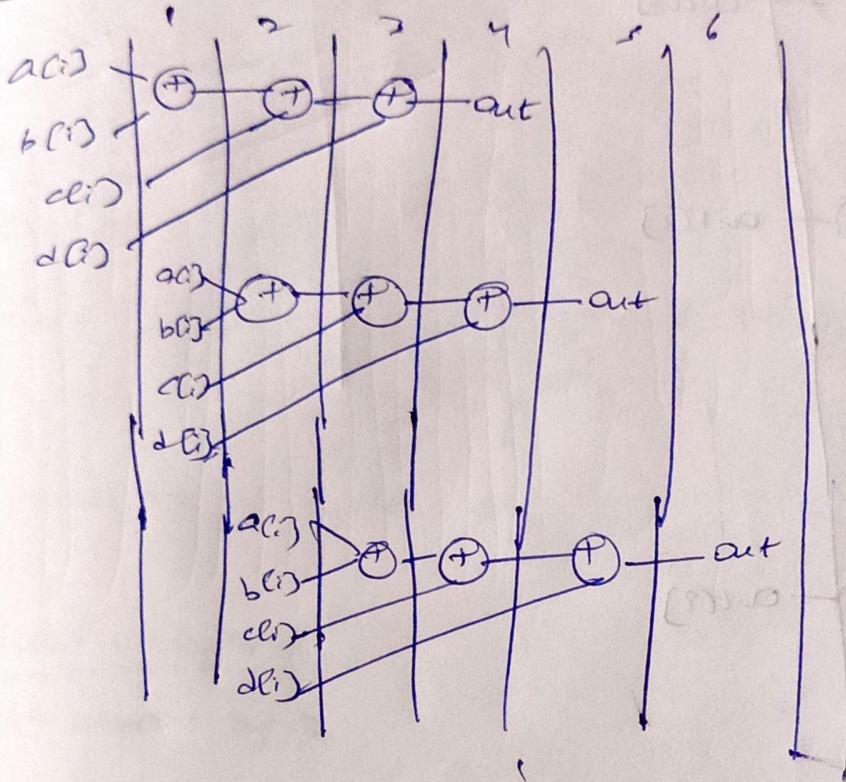
$$3 \times 10 = 30 \text{ cycles}$$

Iterative Implementation

$n * k$



② Pipelined Implementation



$$\text{Total number of cycles} = \boxed{n+k-1} = 12 \text{ cycles}$$

③ Unrolled Implementation (Loop unrolling)

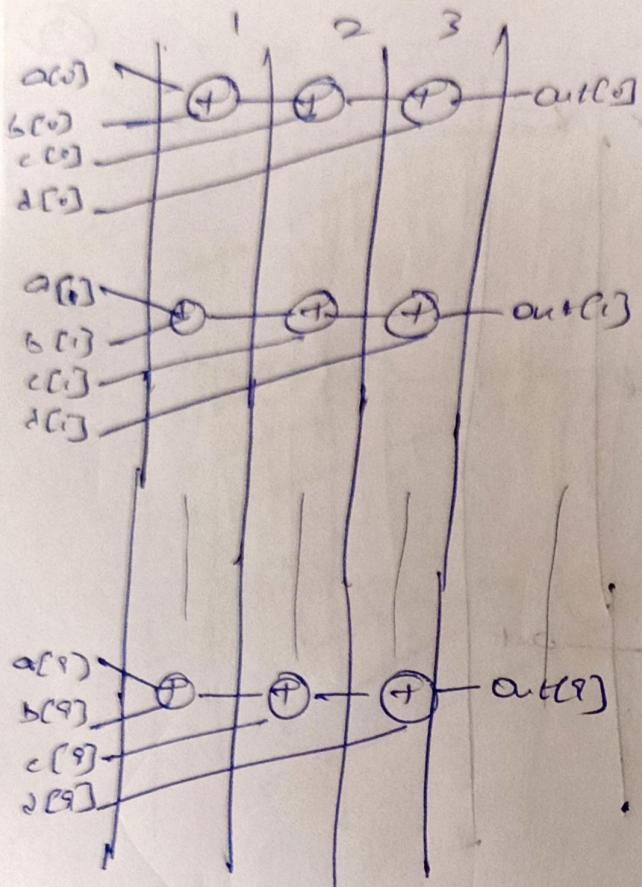
```
void fun(a[10], b[10], c[10], d[10]) {
```

$$out[0] = a[0] + b[0] + c[0] + d[0]$$

$$out[1] = a[1] + b[1] + c[1] + d[1]$$

$$out[2] = a[2] + b[2] + c[2] + d[2]$$

return out



Total number of cycles = 3

Best case k

	Time	Resource
Iterative	30	1
Pipelined	12	3
Unrolled	3	10
Partial unroll	15	2

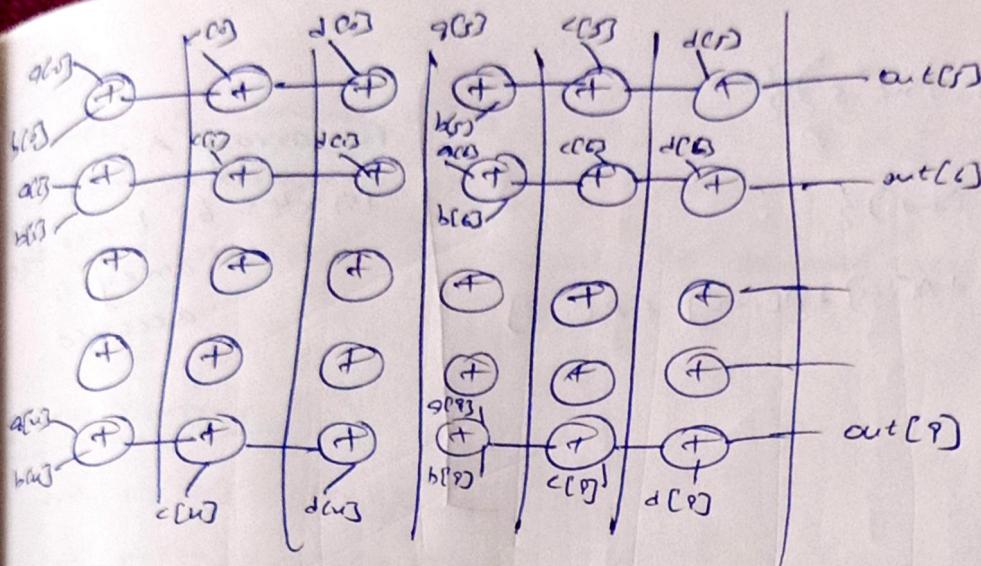
\Rightarrow Sorta good tradeoff b/w performance and resource

(3) Unrolled with Resource Constraint

Adders = 5

Obj: Schedule in min time-off w/o violating RC

9



Total no. of cycles = 6

④ Partial Unrolling

unroll by 2

```

void fn ( a[i,j], b[i,j], c[i,j], d[i,j] ) {
    for ( i<0 ; i<10 ; i+=2 ) {
        out out[i] = a[i] + b[i] + c[i] + d[i]
        out out[i+1] = a[i+1] + b[i+1] + c[i+1] + d[i+1]
    }
    return out
}

```

2/1/23

HLS of arrays :-

int a[loop] \Rightarrow RAM/ROM
 ↑ Random Access Memory
 mapped (Read only Memory)

m[999:0][31:0]

```
void fun (int A[20], int *s) {
```

```
    for (i=3; i<20; i++) {
```

$$s[i] = A[i] + A[i-1] + A[i-2] + A[i-3]$$

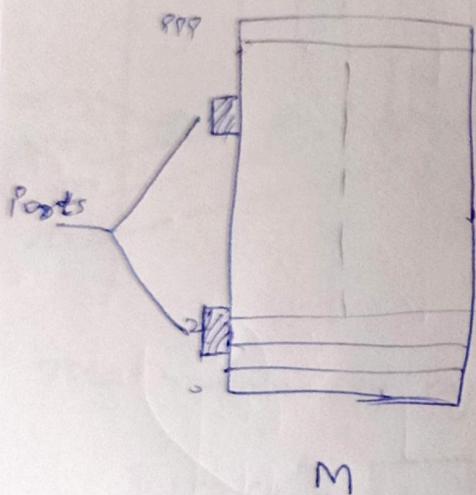
```
}
```

```
}
```

For array A:

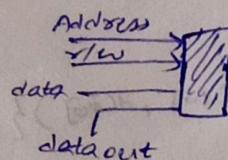
$$17 \times 4 = 68 \text{ times}$$

array is accessed



M

Memory would have single/dual port
For each port :-



```
void fun (int A[20], int *s) {
```

$$t_1 = A[0]$$

$$t_2 = A[1]$$

$$t_3 = A[2]$$

```
for (i=3; i<20; i++) {
```

$$x = A[i]$$

$$s[i] = \cancel{A[0]} \cancel{x} + t_1 + t_2 + t_3$$

$$t_3 = \cancel{x}$$

$$t_2 = \cancel{3}$$

$$t_1 = \cancel{2}$$

```
}
```

```
}
```

For array A:-

Total memory access
= 20

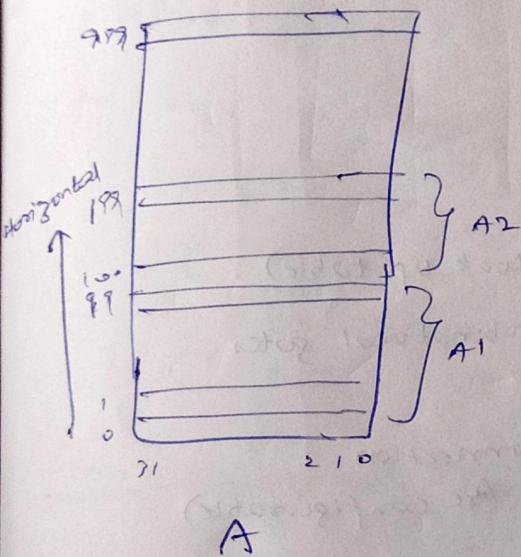
array

- 1) reduce memory access
- 2) remove redundant memory access
- 3) read once and store locally to ~~reuse~~ reuse

Array Merging

int A[100] - 100x32 bits

int A2[100] < 100x32 bits



$$\begin{aligned} A_2[0] &\leftrightarrow A[100] \\ A_2[99] &\leftrightarrow A[199] \\ A_1[0] &\leftrightarrow A[0] \\ A_1[99] &\leftrightarrow A[99] \end{aligned}$$

#pragma HLS ARRAY_MAP

variable A1 instance A horizontal

pragma HLS ARRAY_MAP

variable A2 instance A horizontal

int A1[1000], A2[500]

map A1 to A , map A2 to A

```
for (int i=0; i<1000; i++) {
```

```
    A1[i] = value1
```

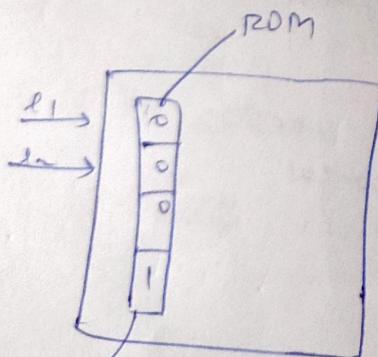
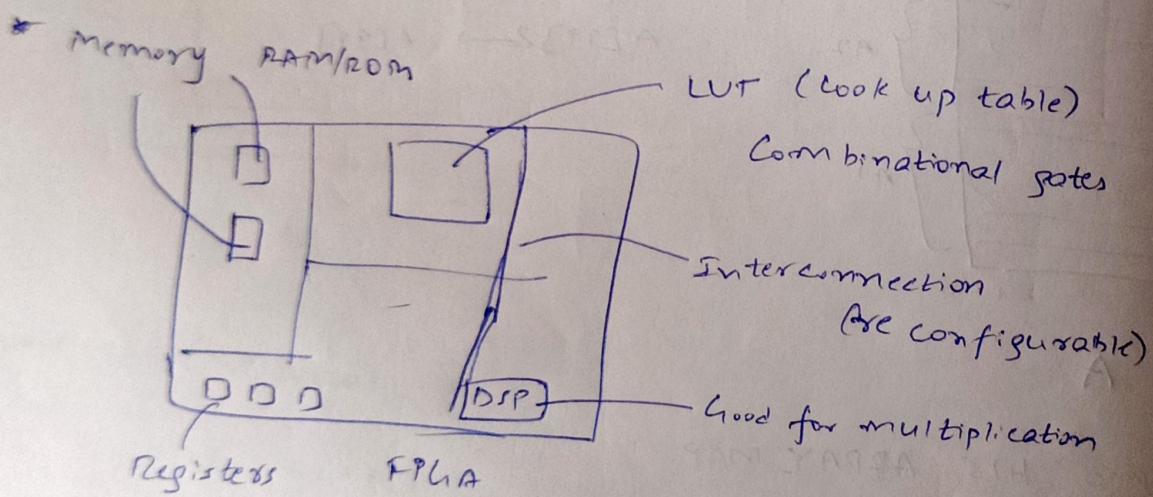
```
    if (i<500) A2[i] = value2
```

}

Given: One Dual Port RAM (2000×32)

Given: One single port RAM (1000×70)

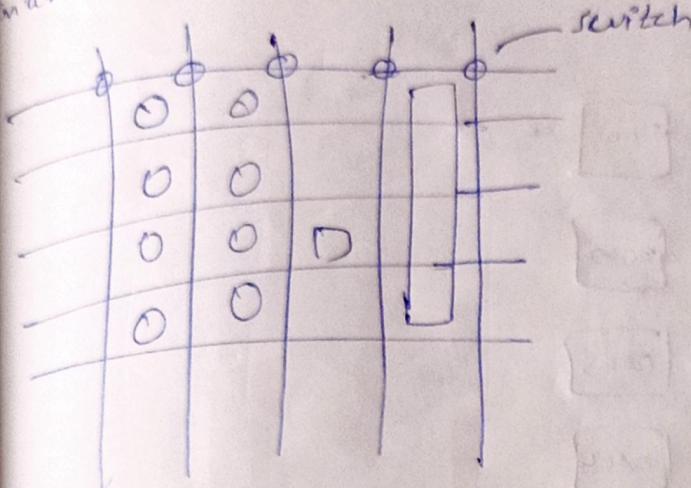
sol 1/23



This is for AND gate
LUT

We can do this for any combinational gates

inter connections:



`int A[1000], A2[100]`

`for (i=0; i<1000; i++) {`

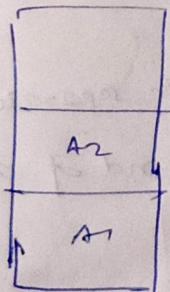
`A[i] = value1,`

`if (i < 500)`

`A2[i] = value1,`

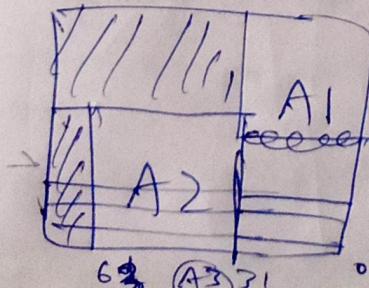
}

Given: One BRAM (1000 x 70) single port



⇒ Not possible

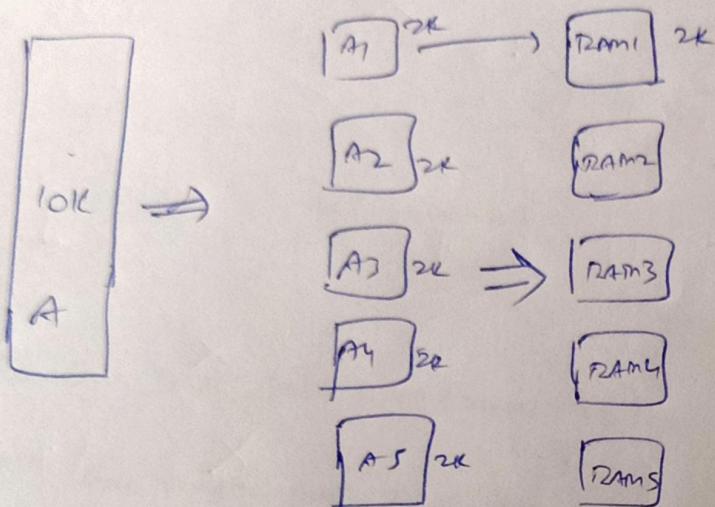
Placed them vertically



$$A3[i][31:0] = A[i]$$

$$A3[i][39:32] = A2[i]$$

Array Partition



```
for (i=1; i < 10K; i = i+2) {
```

$$B[i] = A[i] + A[i+1]$$

↓ ↓

even location odd location
in RAM1 in RAM2

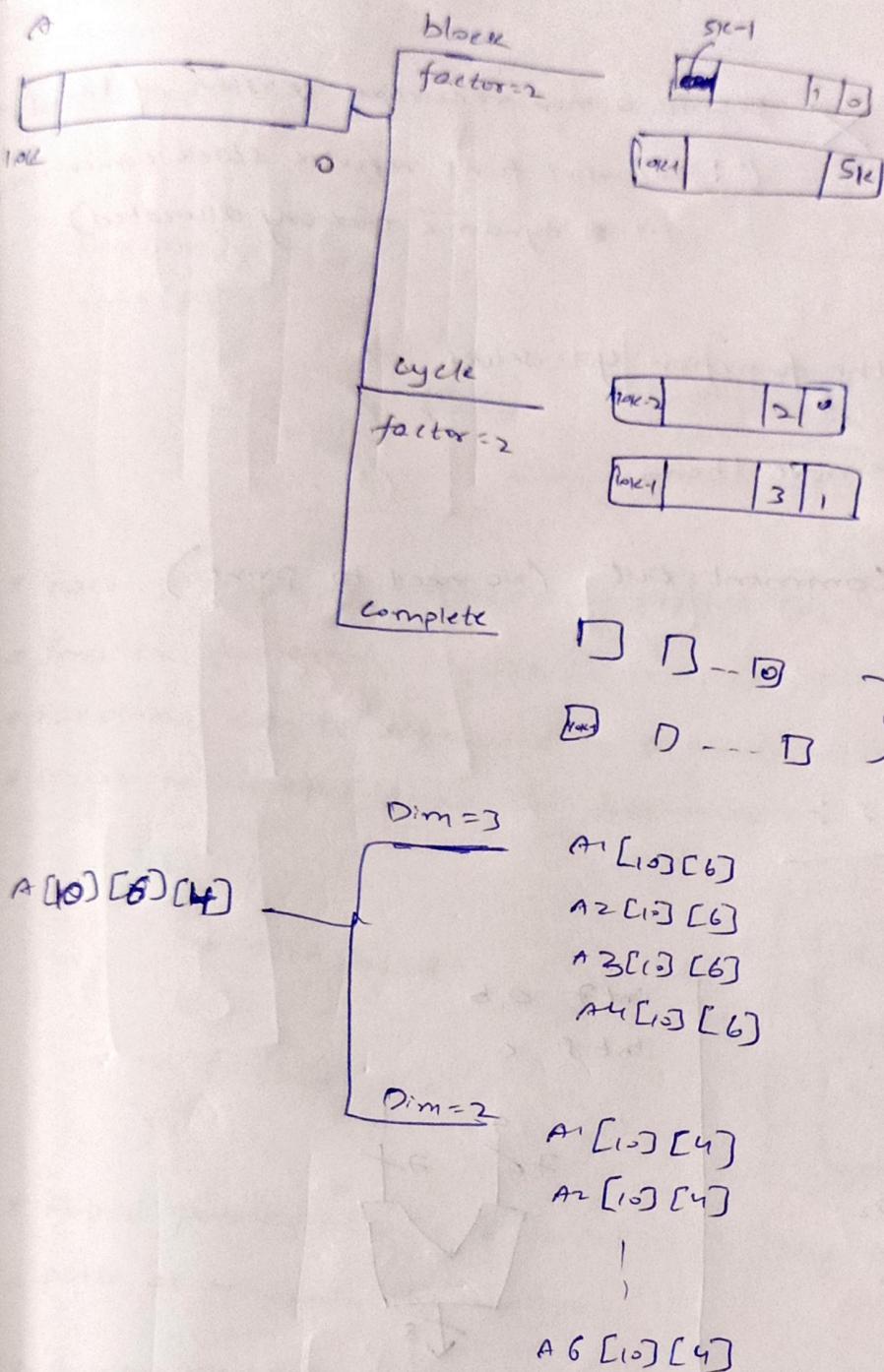
Objective is keeping the parallel access memory in separate RAM.
The command for the HLS tool to do the above kind of array splitting is

```
# program HLS array-partition Array-Name <type> factor ,
```

type: block/cycle/complete

Dim =

15



* Unsupported 'c' constructs :-

Data types — int, float, char ✓

structure, union ✓

pointers ✓

array ✓

Dynamic Memory Allocation (malloc/calloc) X

Rewrite program
by statically allocating
memory

Function ✓

Recursive Function ✗

Rewrite a non-recursive version of the function
(*: Recursive fun's require stack which is dynamic memory allocated)

STL ✗ Write the function yourselves

System calls ✗ Remove them

printf / scanf ✗ Comment out (no need to print)

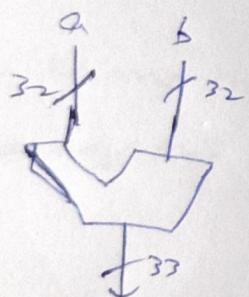
31/1/23

Coding style for HLS :-

Data types

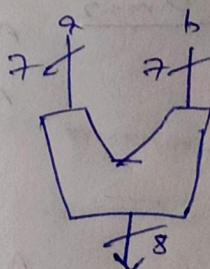
int 32 bits a, b, c

$$c = a + b$$



int f a, b

int g c



* Data width impacts the area, delay of the design heavily.

* So we must use precise data widths of the inputs.

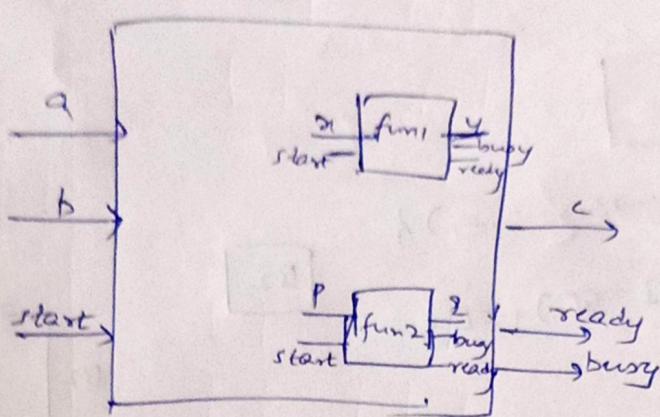
* Input data width propagates and determine the datapath width in RTL.

function

```
top(a,b,c){
```

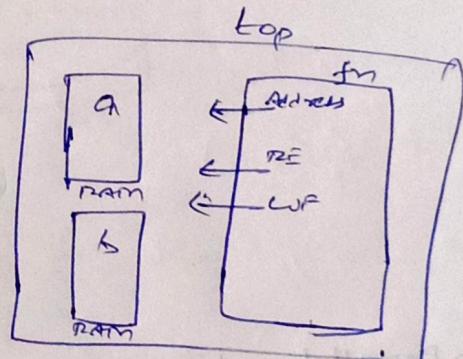
```
{  
    fun1(a,y)  
    fun2(b,z)  
}
```

}



- * Each function is allocated a dedicated hardware.
- * Smaller function should be inlined to reduce area overhead.
- * Functions can be scheduled in parallel \Rightarrow Improves the performance
- * Must be careful about the arguments.

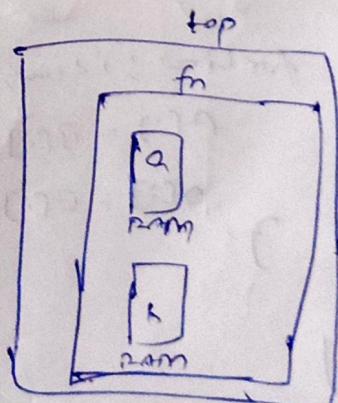
```
top(){  
    int a[100], b[100]  
    fn(a[0], b[0])  
}
```



- * Avoid passing arrays to function as they create unnecessary ports at module boundaries.

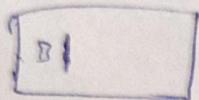
- * Array may be duplicated to avoid these.

```
top(){  
    fn()  
}  
fn(){  
    int a[100] b[100]  
}
```



Loops

fun() {



for($i=1$; $i \leq 100$; $i++$) {

$$A[i] = B[i] + C[i]$$

}

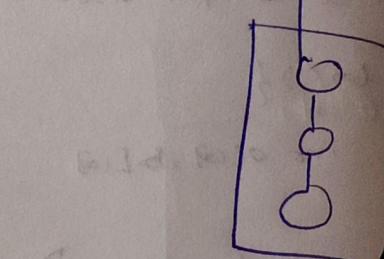
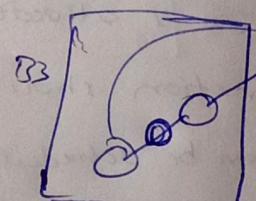
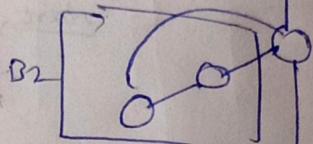
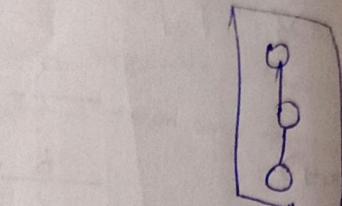
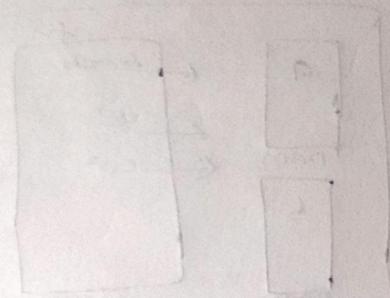
for($i=1$; $i \leq 100$; $i++$) {

$$D[i] = E[i] + F[i]$$

}



}



B4

Loop Parallelization

fun() {

1

for($i=1$; $i \leq 100$; $i++$) {

$$A[i] = B[i] + C[i]$$

$$D[i] = E[i] + F[i]$$

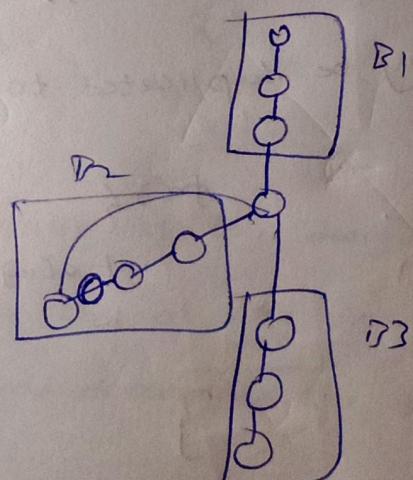
}

}

$$\text{Total Time} = B1 + 100 \times B2 + (20 \times) B3 + B4$$

$$= 3 + 200 + 300 + 3$$

$$= 506$$



$$\text{Total Time} = 3 + 100 \times 2 + 3 = 406$$

Limit of Loops

```

fun1 {
|
for (i=1 ; i<200 ; i++) {
    A[i] = B[i] + C[i]
}
|
for (i=1 ; i<200 ; i++) {
    D[i] = E[i] + F[i]
}
|
}

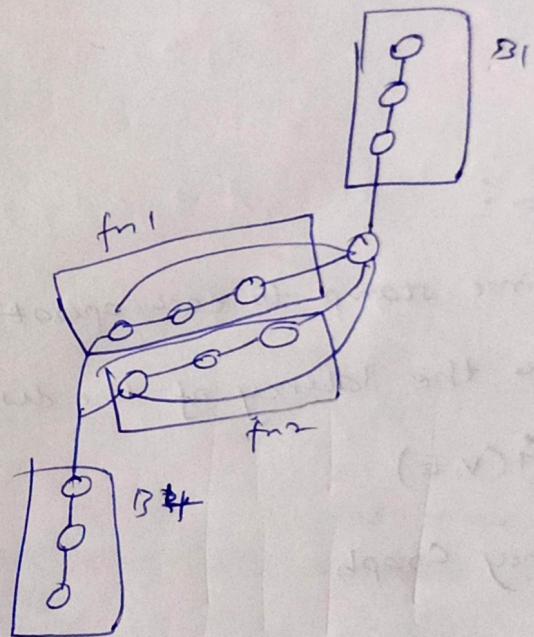
```

⇒

```

fun1 {
|
fun1() //loop1
fun2() //loop2
|
}

```



Putting the loops in functions
will make the CPU to run
them in parallel

Nested Loop

* Where to apply pipeline

```

for (i=1 to 20) {
    for (j=1 to 20) {
        |
        B1
        |
    }
}

```

→ Pipeline

→ Pipeline

Outer Loop :-

- 1) Inner loop will be unrolled
- 2) Run faster but with lot of area overhead

Inner Loop :-

- 1) Pipeline the inner loop  More Preferred
- 2) Require less resource
- 3) Pipelined for 400 times

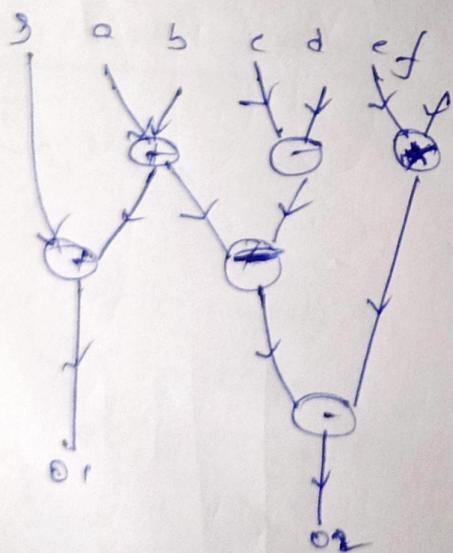
13/2/23

Scheduling :-

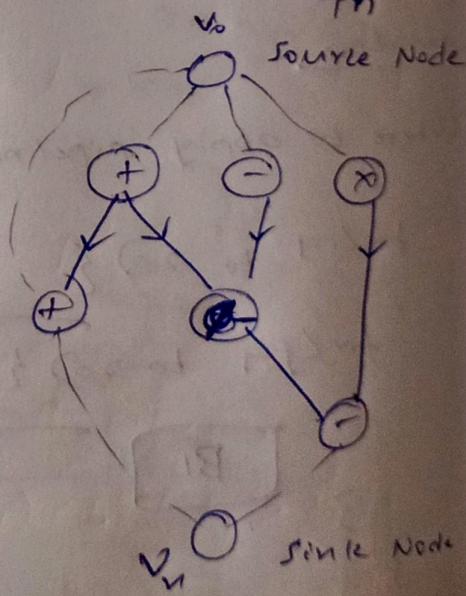
- * Assign time stamp to each operation
- * Determine the latency of the design

Input :- $G(V, E)$

Dependency Graph



Sequence Graph



Input:

① $G = (V, E)$, $\{v_1, v_2, \dots, v_n\}$ are the operations $E(v_i, v_j) : \text{Dependencies of the operations}$

② $D = \{d_1, \dots, d_n\}$

d_i is the delay of operation v_i

③ $r, V \rightarrow \{1, \dots, m_{ro}\}$ here m_{ro} is type of operation

④ Resource Constraints $a_k : k=1, 2, \dots, m_{ro}$

⑤ Latency constraints λ

Output:

⑥ $\{t_i : i=0, \dots, n\}$ t_i represents the ~~start time~~ of node v_i

⑦ (t_{n-1}) is the latency $t_0 = 0$

START time

$$\phi: V \rightarrow \mathbb{Z}^+ \\ \phi(v_i) = t_i$$

Unconstrained Scheduling : (US)

Given G and D find $\phi: V \rightarrow \mathbb{Z}^+$ i.e., $\phi(v_i) = t_i$ s.t.

i) It is a valid schedule i.e., it satisfies all dependencies i.e.,

$$\forall (v_i, v_j) \in E, t_j \geq t_i + d_i$$

ii) t_n is minimum \Rightarrow Optimum schedule

① Minimize latency under resource constraints (ML-RC)

Given:

① G ② D ③ Y ④ R

Output:

$\phi: V \rightarrow \mathbb{Z}^+$ i.e. $\phi(v_i) = t_i$ s.t. ϕ satisfies

① Dependency Constraints

$\forall (v_i, v_j) \in E, t_j \geq t_i + d_i$

② $\{v_i : \phi(v_i) = l_i \text{ and } t_i \leq l_i \leq t_i + d_i - 1\} \leq q_k$ for each

$k = \{1, 2, \dots, m_{\text{req}}\}$ and for each time step $l = 1, 2, \dots, t_n$

\Rightarrow Resource constraint

③ t_n is minimum

② Minimize Resource under Latency Constraint (MR-LC)

Given:

① G ② D ③ Y ④ l (latency limit)

Output:

$\phi: V \rightarrow \mathbb{Z}^+$ i.e. $\phi(v_i) = t_i$ s.t. ϕ satisfies

① Dependency Constraints $\forall (v_i, v_j) \in E, t_j \geq t_i + d_i$

② Latency constraint $t_n \leq l + 1$

$a_1, a_2, \dots, a_{\text{resources}}$ is minimum \Rightarrow Each resource type is minimized

ML-RC
MR-LC } NP-Complete
Problem

US is not.

Unconstraint scheduling (US)

i) ASAP (As soon As Possible)

\Rightarrow schedule each node as soon as all predecessor nodes completed the execution.

ASAP (G, D, r) {

~~definite~~ to = 0

while (uncheduled node exist) {

1. Select a node v_j whose predecessors have already been scheduled

2. Schedule node v_j to t_j where

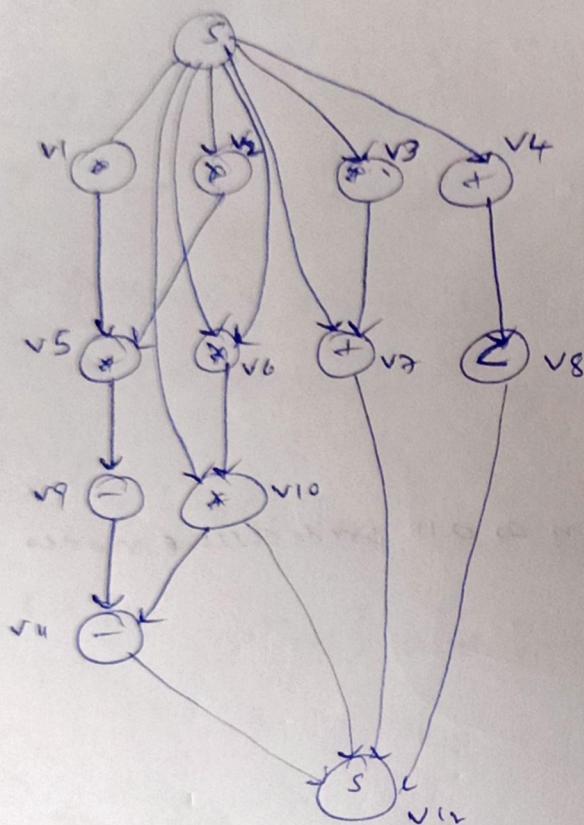
$$t_j = \max(d_i + t_i) \quad \forall (v_i, v_j) \in E$$

}

}

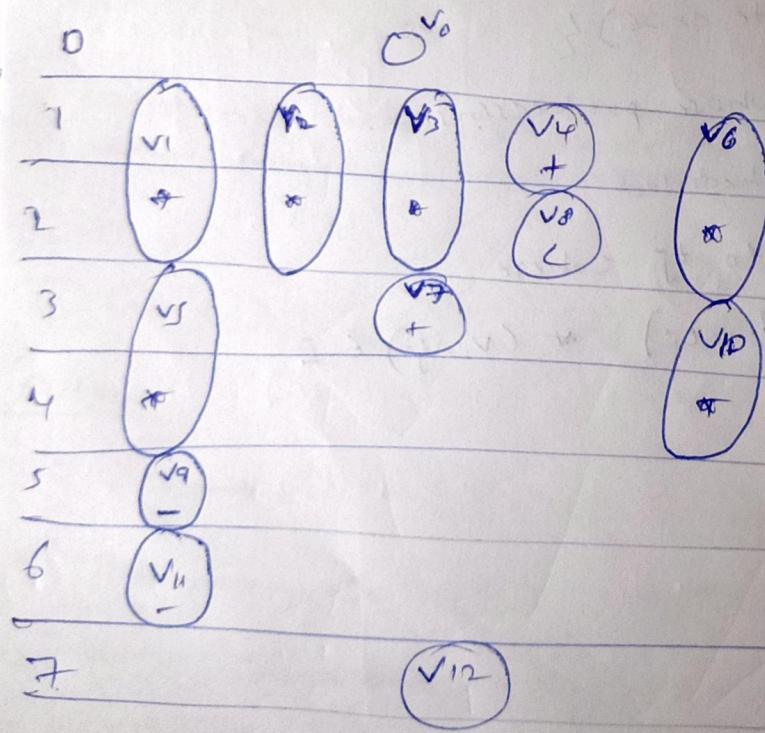
11/10/23

ASAP Scheduling (G, r, D)



$$d(*) = 2$$

$$d(+, -, \times) = 1$$



$$\boxed{x=6}$$

ALAP (As Late As Possible)

ALAP(a, δ, D, λ) {
 Latency based

$$tn = \lambda + 1$$

while (any node unscheduled) {

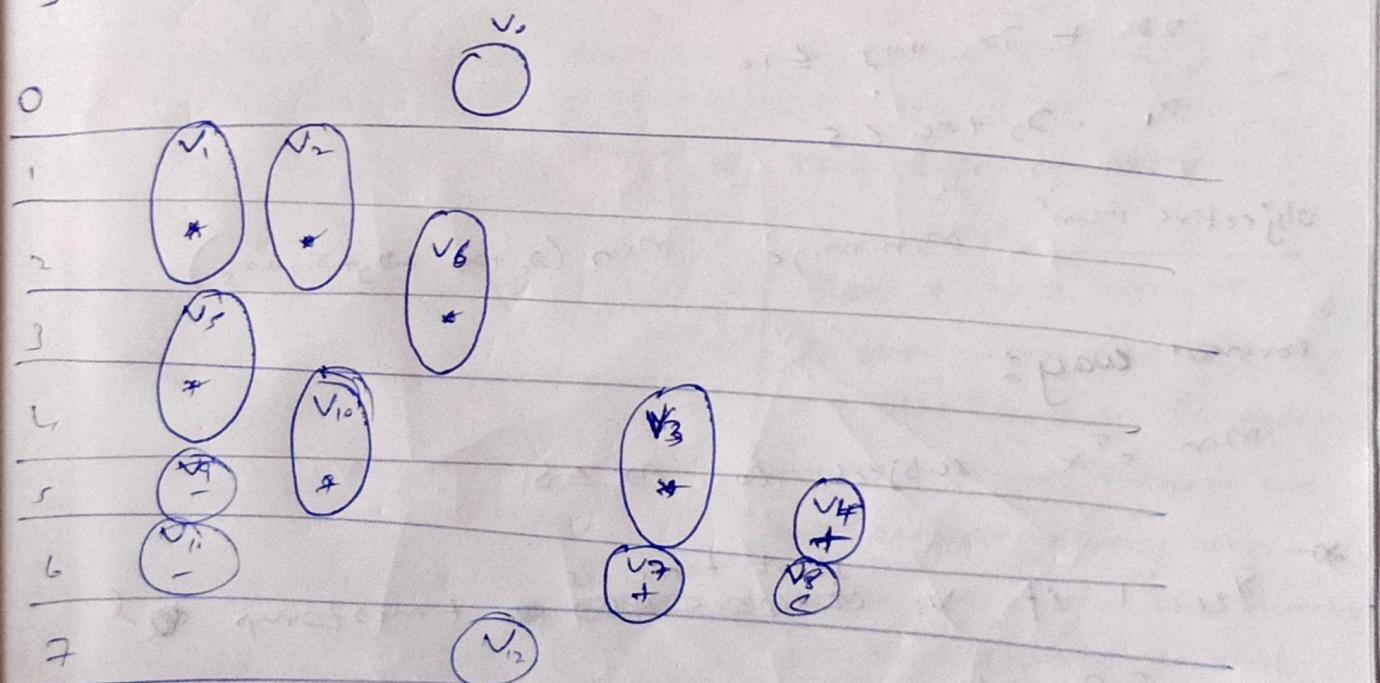
1. Select a node v_i whose successor are already scheduled.

2. Schedule v_i to $t_i = \min(t_j - d_i) \quad \forall (v_i, v_j) \in E$

}

if ($t_i < 0$) "λ is not sufficient" error

}



$x = 6$
 Given

* t_i^{ATAP} : Lower bound of schedule of node v_i

* t_i^{ALAP} : Upper bound of schedule of node v_i

Mobility of node v_i

$$\leq t_i^{\text{ALAP}} - t_i^{\text{ATAP}}$$

ILP Formulation of Scheduling

ILP \rightarrow Integer Linear Programming

x_1, x_2, \dots, x_n unknown integer variables

Constraints

$$2x_1 + 3x_2 + x_3 \leq 10$$

$$x_1 - 2x_2 + x_4 \leq 5$$

Objective Function Minimize $\min (x_1 + x_2 + x_3 + x_4)$

Formal way:

$$\min c^T x \text{ subject to } Ax \leq b$$

~~* $x_{ij} = 1$ if v_i starts at timestamp j~~
 $x_{ij} = 1$ if v_i starts at timestamp j
 $= 0$ otherwise

① Unique start time:

$$\sum_{j=1}^{n+1} x_{ij} = 1 \quad \forall i = 1, 2, \dots, n$$

$$t_i = \sum_{l=1}^L x_{il} l$$

② Data dependencies / Precedence constraints :

$$\forall (v_i, v_j) \in E \quad t_j \geq t_i + d_i$$

$$\text{i.e. } \sum_{l=1}^L x_{jl} l \geq \sum_{l=1}^L x_{il} l + d_i$$

③ Resource Constraints :

④ Latency constraints :

$$\sum_{l=1}^{L+1} x_{ml} \leq \alpha_k$$

$$\sum_{\substack{i: \delta(v_i)=k \\ m=l-d_i+1}} x_{im} \leq \alpha_k$$

$$k = 1, 2, \dots, m_{\text{res}}$$

$$l = 1, 2, \dots, L$$

* Minimize Latency under Resource Constraint (ML-RC) :-

Given α_k : resource limit of each type operator

1. Unique start time

Here λ is a variable.

2. Data dependency constraints

so the no. of variables x_{il} are also variable. In order

3. Resource Constraints

for this to not happen we will consider a very large

Obj : minimize λ

value of λ for the number of time stamps so that it is not a variable.

i.e., Minimize λ

General Solution

Objective func: $c^T \mathbf{x}$ $\Rightarrow c_1 t_1 + c_2 t_2 + \dots + c_n t_n$

$$c^T = [0 \ 0 \ 0 \ \dots \ 1] \\ c_1 \ c_2 \ c_3 \ \dots \ c_n$$

* ASAP:

Objective func: $c^T t$

$$t = [t_1 \ t_2 \ t_3 \dots \ t_n]$$

① Unique start Time

② Data dependency constraints

* MR-LC:

Objective func: $c^T a$

$$a = [a_1 \ a_2 \dots \ a_n]$$

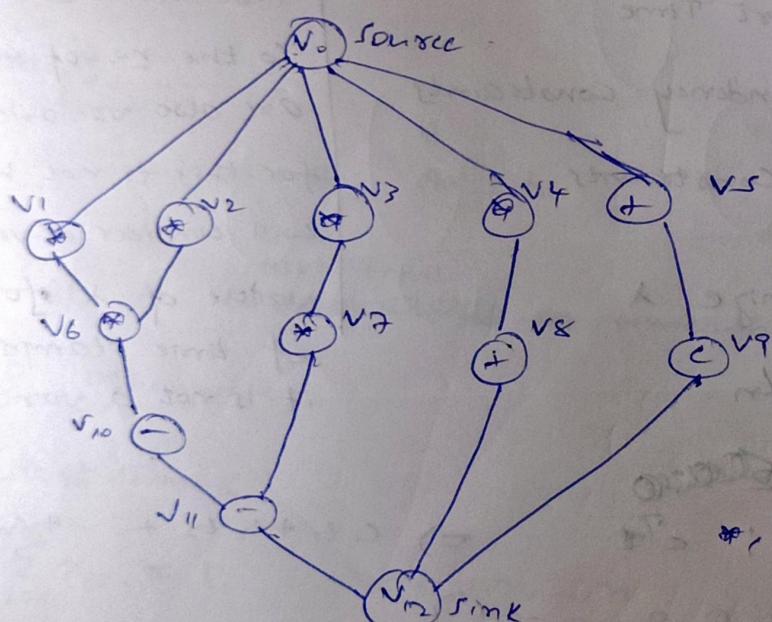
① Unique start Time

② Data dependency constraints

③ Resource constraints where a_{ik} is a variable

④ Latency constraints

$c_i \Rightarrow$ Value corresponding to the appropriate area occupied by the hardware



*, -, +, < \Rightarrow Single Cycle

i.e., $d_i = 1 \forall i$

MR-LC :

$$x = 4$$

mobility of v_1, v_b, v_{10} and v_{11} is zero (\because given $\lambda = 4$)

$$v_1 \Rightarrow x_{11} \quad v_2 \Rightarrow x_{21} \quad v_6 \Rightarrow x_{62} \quad v_{10} \Rightarrow x_{10,3} \quad v_{11} \Rightarrow x_{11,4}$$

$$v_3 \Rightarrow x_{31}, x_{32} \quad v_7 \Rightarrow x_{71}, x_{73}, x_{72}$$

$$v_4 \Rightarrow x_{41}, x_{42}, x_{43} \quad v_5 \Rightarrow x_{82}, x_{83}, x_{84}$$

$$v_5 \Rightarrow x_{51}, x_{52}, x_{53} \quad v_9 \Rightarrow x_{92}, x_{93}, x_{94}$$

Unique Start Time :

$$x_1 = 1 \quad x_{21} = 1$$

$$x_{31} + x_{32} = 1 \quad x_{73} + x_{72} = 1$$

$$x_{41} + x_{42} + x_{43} = 1$$

Dependency Constraints :

$$(v_1, v_6) : 2 \cdot x_{62} \geq 1 \cdot x_{11} + 1$$

$$(v_4, v_8) : 2x_{12} + 3x_{83} +$$

Resource Constraints:

For time stamp 1:

$$\text{MUL: } x_{11} + x_{21} + x_{31} + x_{41} \leq 0_1$$

$$\text{ALU: } x_{51} \leq 0_2$$

For time stamp 2:

$$\text{MUL: } x_{62} + x_{32} + x_{42} + x_{72} \leq 0_1$$

$$\text{ALU: } x_{82} + x_{52} + x_{92} \leq 0_2$$

Latency Constraint:

$$10x_{12,1} + 2x_{22,2} + \dots + 5x_{12,15} \leq \lambda + 1$$

2012/23

Multiprocessor Scheduling:

Assumption

- (A1) All operations can be created by same resource (ALU)

Earlier $a_1, a_2, \dots, a_{n_{\text{ops}}} \Rightarrow$ Now a (mixture of)
ALU

- (A2) All operations have unit execution time
No. of processing units

specifying problem
is //NP-complete

MLP \Rightarrow Multiprocessor ML-RC.

(31)

1. Unique Start Time \Rightarrow SAME

2. Data dependency

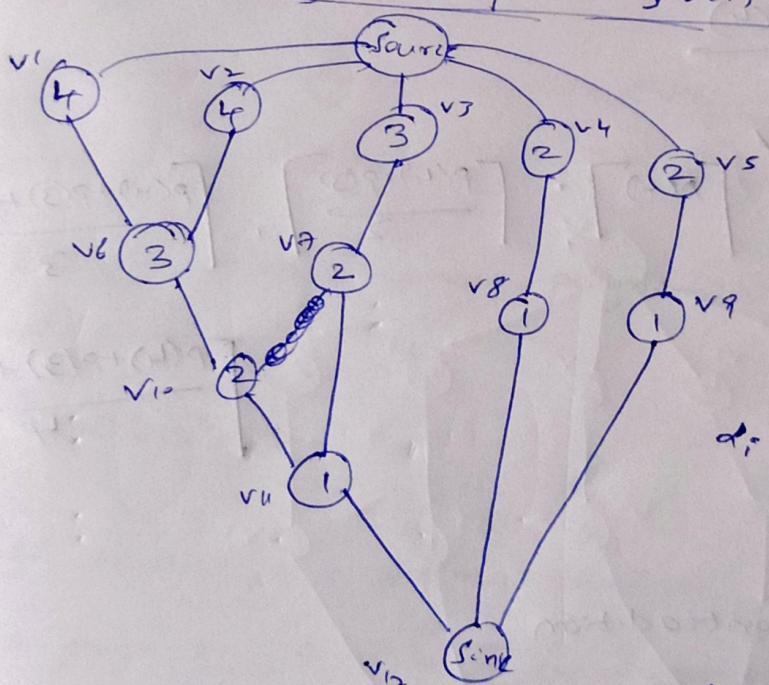
$$\sum_{l=1}^{A+1} \lambda \cdot x_{jl} \geq \sum_{l=1}^{A+1} \lambda \cdot x_{il} + 1 \quad \forall (v_i, v_j) \in E$$

3. Resource Constraints

$$\sum_l x_{il} \leq a \quad l = 1, 2, \dots, A+1$$

4. Objective Function \Rightarrow SAME

Resource Bound under $A_1 < A_2$ for a given latency λ :



$d_i =$ Label of node

v_i denotes the largest path from v_i to sink

$$\lambda = \max_{1 \leq i \leq n} d_i$$

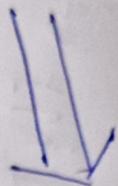
$d \approx 4$

$\lambda \geq d$

$P(j) = \# v_i$ with label j

* Matis Theorem: A lower bound on the number of resources to complete a schedule with latency δ is

$$\bar{\alpha} = \max_{\gamma} \left\lceil \frac{\sum_{j=1}^{\gamma} p(\delta+1-j)}{\gamma + 1 - \delta} \right\rceil$$



Problem is NP-complete even with the two assumptions

Example: $\lambda=4, \alpha=4$

if $\gamma=1$:

$$\bar{\alpha} = \frac{p(4)}{2+1} = p(4)$$

if $\gamma=2$:

$$\bar{\alpha} = \frac{p(2) + p(4)}{2}$$

$$\text{so } \bar{\alpha} = \max \left(\lceil p(4) \rceil, \lceil \frac{p(4) + p(3)}{2} \rceil, \lceil \frac{p(4) + p(3) + p(2)}{3} \rceil \right)$$

$$\lceil \frac{p(4) + p(3) + p(2) + p(1)}{4} \rceil$$

(c) Proof:

Let's prove by contradiction

Let $\bar{\alpha} = \arg \max_{\gamma} \gamma^*$

Let's assume that there exist a schedule λ with $\lambda \leq \bar{\alpha}$

$$\lambda < \frac{\sum_{j=1}^{\gamma^*} p(\delta+1-j)}{\gamma^* + 1 - \delta}$$

$$2 \cdot (\alpha^* + \lambda - \alpha) < \sum_{j=1}^{2^*} p(\alpha+1-j) = p(\alpha) + p(\alpha-1) + \dots + p(0)$$

\downarrow

$$P(\alpha+1-\alpha^*)$$

max no. of operations scheduled

HUB time stamp $(\alpha^* + \lambda - \alpha)$

At least some node with label $(\alpha+1-\alpha^*)$ remain to be scheduled

\Rightarrow At least $(\alpha+1-\alpha^*)$ steps needed to complete the schedule

\Leftrightarrow Total schedule length $\geq (\alpha+1-\alpha^*) + (\alpha^* + \lambda - \alpha)$
 $= \lambda + 1$
 (Contradiction)

21/2/23

* with the addition of another assumption,

(A3) sequence graph is a tree

the HUB's algorithm becomes polynomial time solvable
 from NP-complete

in HUB's algo $\Rightarrow O(n)$

* HUB's algo always give optimal solution.

① For a given λ , it uses $\bar{\alpha}$ (min resources)

② For a given $\bar{\alpha}$ (resource bound), it gives min λ

a → Resource Bound

* MU's algo (q, a) {

calculate d_i for each v_i

$l=1$

while (there is a unscheduled node) {

① $U = \text{set of nodes ready to be scheduled}$
(i.e., all predecessors are scheduled)

② select $s \subseteq U$ such that $|s| \leq q$ and d_i 's of s are maximal

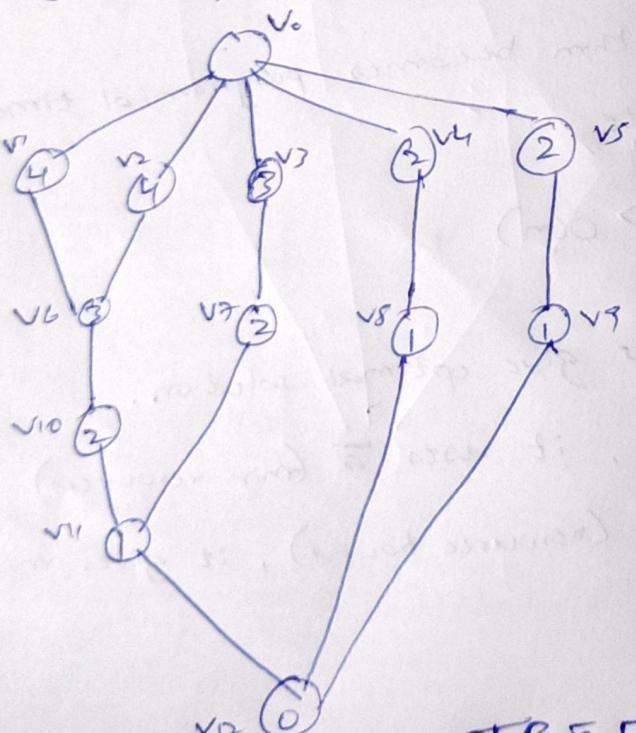
③ schedule all $v_i \in s$ in " l " timestamp

④ $l++$

}

3

Example solving of DiffQ:



1-3 \Rightarrow Calculated by using HU's formula

(25)

$$U = \{v_1, v_2, v_3, v_4, v_5\}$$

$$S = \{v_1, v_2, v_3\} \Rightarrow \text{in } TS = 1$$

1-2

$$U = \{v_4, v_5, v_6, v_7\}$$

$$S = \{v_6, v_7, v_4\} \Rightarrow \text{in } TS = 2$$

1-3

$$U = \{v_{10}, v_8, v_5\}$$

$$S = \{v_{10}, v_8, v_5\} \Rightarrow \text{in } TS = 3$$

1-4

$$U = \{v_{11}, v_9\}$$

$$S = \{v_{11}, v_9\} \Rightarrow \text{in } TS = 4$$

* List Based Scheduling

- Heuristic based algo based on HU's algorithm
- Maintain a list of available nodes
- Select a subset based on some priority value (longest path length etc.)
- Gives near optimal result in most of the cases
- Works for Generic Scheduling (No A1, A2, A3)
$$\begin{matrix} \times & \times & \times \end{matrix}$$