9/1/23

ASIC : Application Specific Integrated Circuit

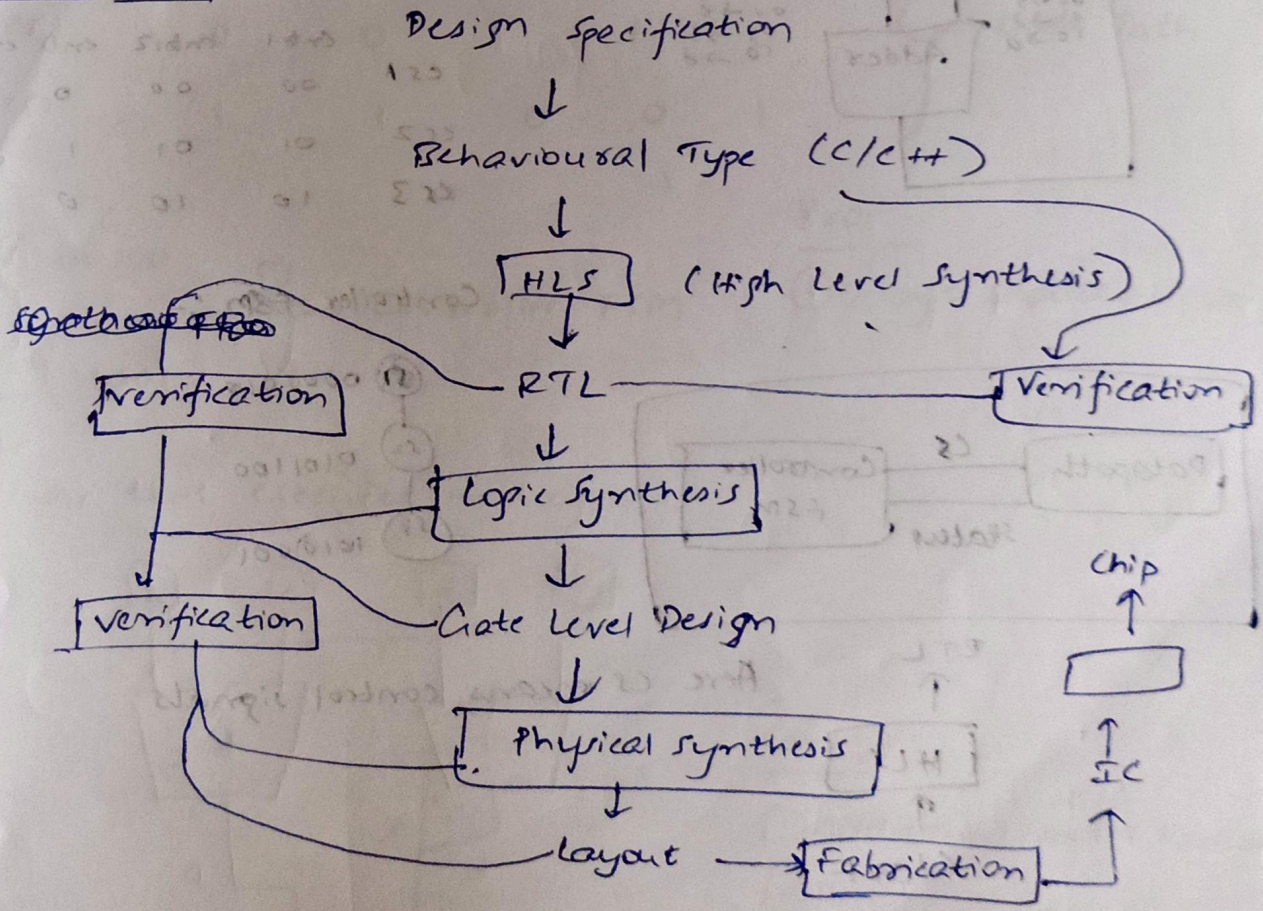
FPGA : Field Programmable gates Array

Difference blw the usage of ASIC and FPGA comes due to the issue of scalability,

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

FPGA is . . . . . few . . .

Synthesis Flow :-

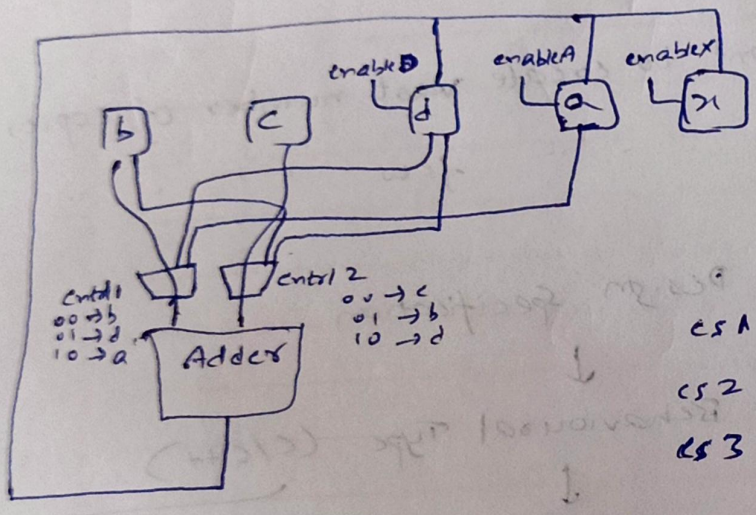


Fun (b, c, d) :

* a = b + c
d = d + b
x = a - d

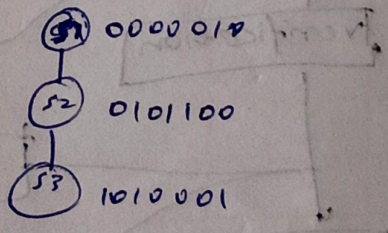
```
a = b+c
d = d+b
x = a-d
```

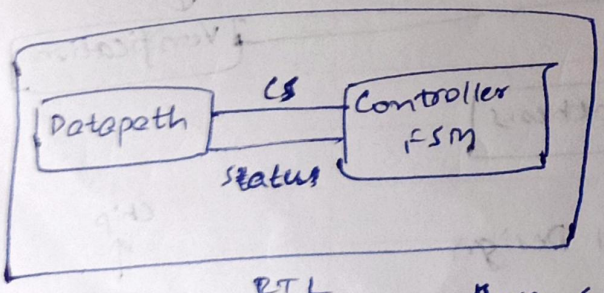
Hardware that executes above fun' in 3 clocks :-



	cntrl	cntrl2	enD	enA	enX
cs1	00	00	0	1	0
cs2	01	01	1	0	0
cs3	10	10	0	0	1

Controller FSM :-





RTL

↑

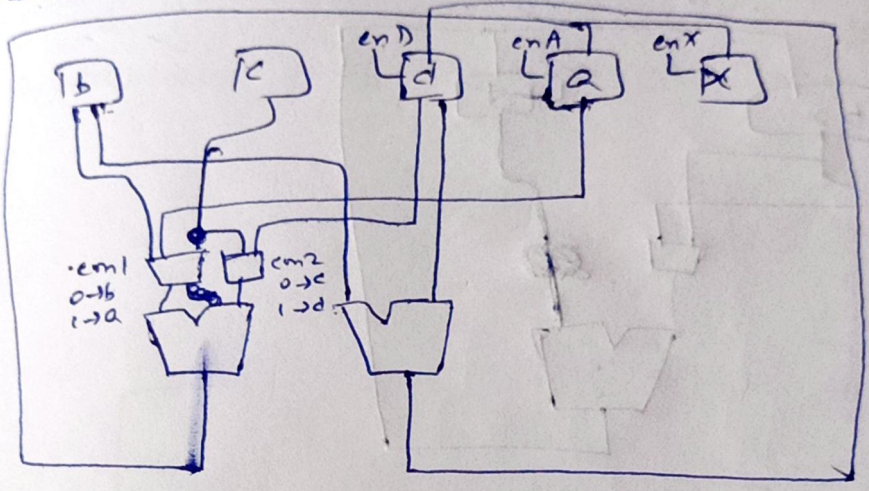
HLS

↑

C/C++

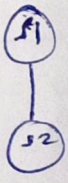
Here CS means control signals

Hardware that executes in 2 clocks :-



	cm1	cm2	enD	enA	enX
cs1	0	0	1	1	0
cs2	1	1	0	0	1

} Data Path

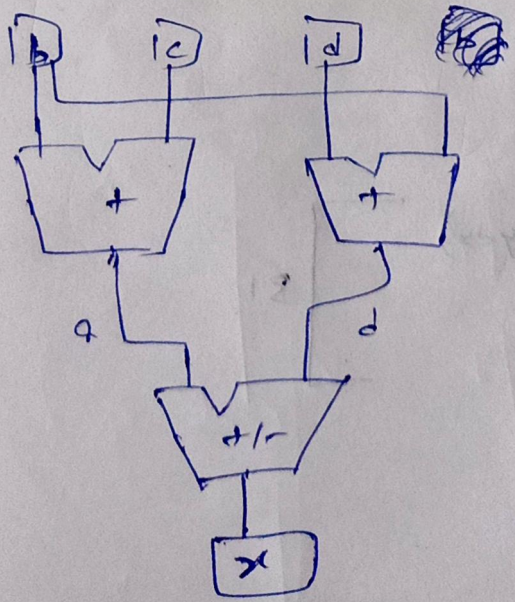


} → Controller FSM

Pros :-

① Delay = 1 Adder + 1 mux

Hardware that executes in 1 cycle :-



Cons :-

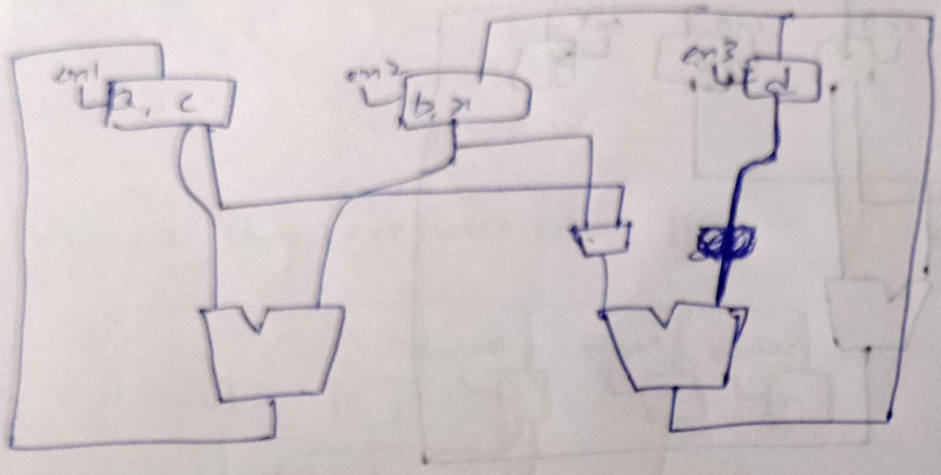
① Additional Adder Hardware

② Delay = 2 Adders

⇓

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

Design with few number of registers :-



2-cycle with 3 registers

(Along with interconnection optimisation)

HLS steps :-

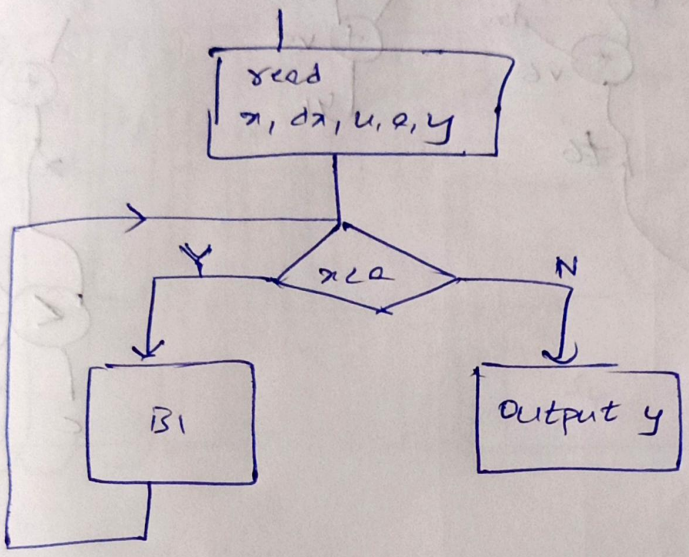
1. Construct control and data flow graph.
2. Split the operations into 3-address format
3. Identify data dependency among the operations

```
Diffeq(x, dx, u, a, y) {
    while (x < a) {
        u1 = u - (3*x*u*dx) - (3*y*dx)
        y1 = y + (u*dx)
        x = x + dx
    } u=u1  y=y1
    return y
}
```

B1

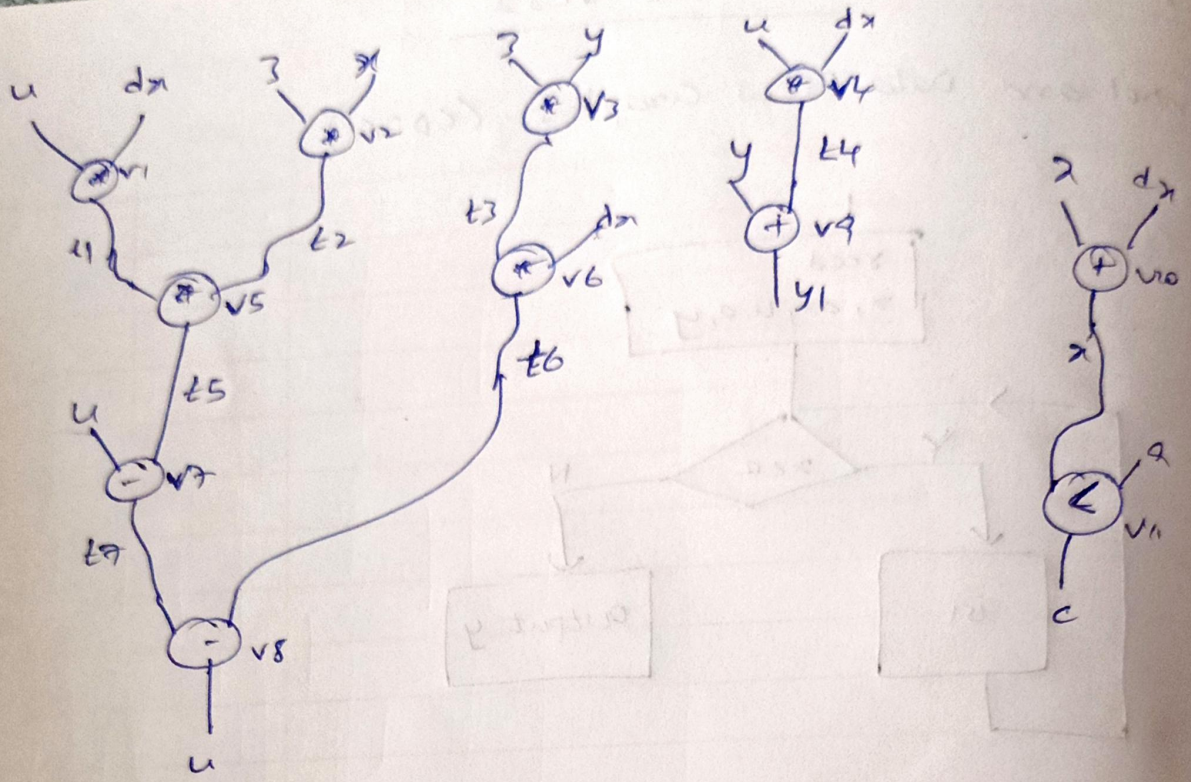
C-Based VLSI

Control and Data Flow Graph :- (CDFG)



B1

V1: t1 = u * dx
V2: t2 = 3 * x
V3: t3 = 3 * y
V4: t4 = u * dx
V5: t5 = t1 * t2
V6: t6 = t3 * dx
V7: t7 = u - t5
V8: t8 = t7 - t6
V9: y1 = y + t4
V10: x = x + dx
V11: c = x < a
V12: y = y1
V13: u = u1

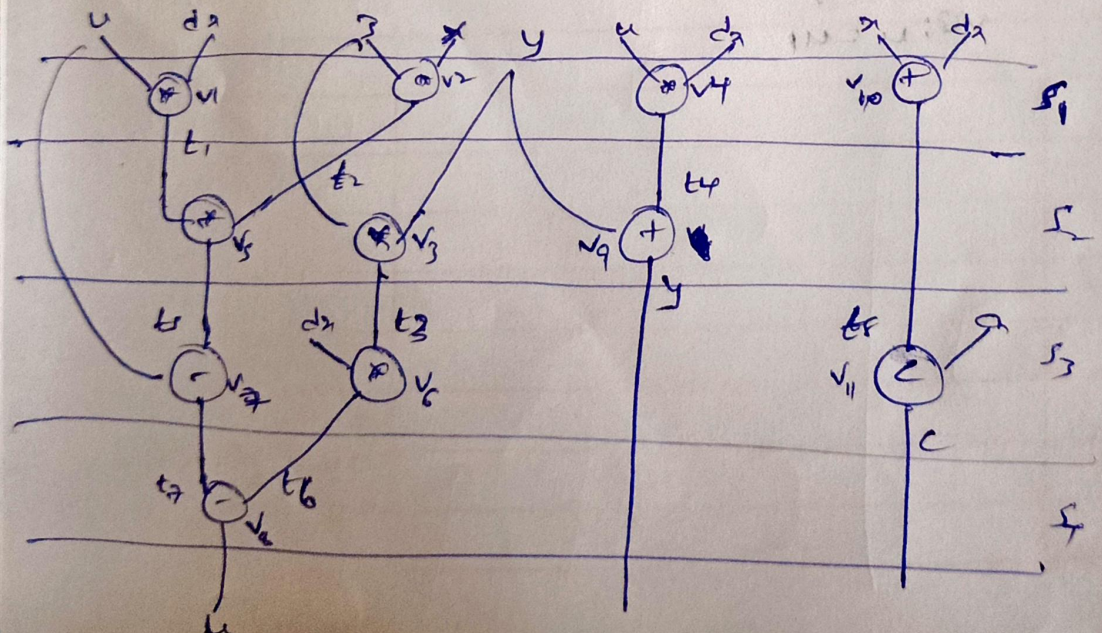


Data Dependency graph

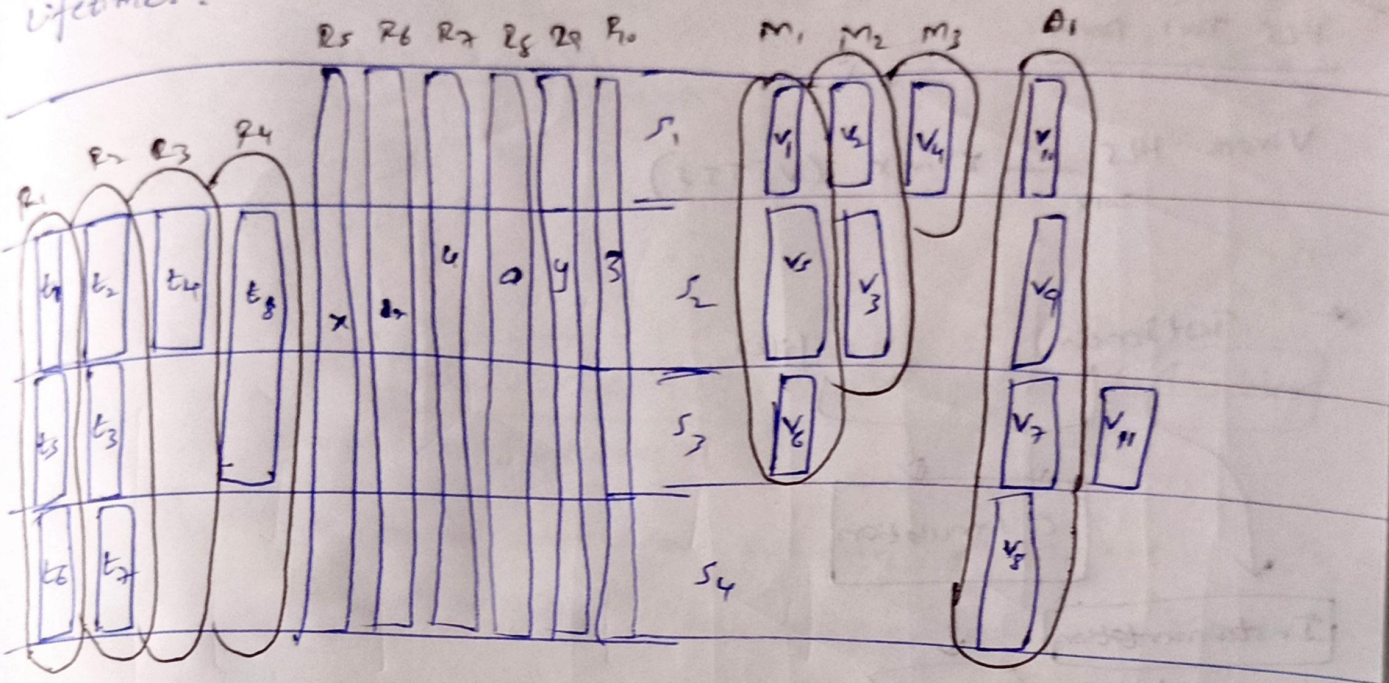
13/1/23

Scheduling : Assigning time step to operation

Allocation and Binding



Lifetimes :-



16/1/23

* HDL Verilog

function ⟷ module

variable ⟷ reg

Transistor level

↓

Gate level

↓

RTL / Data Flow

↓

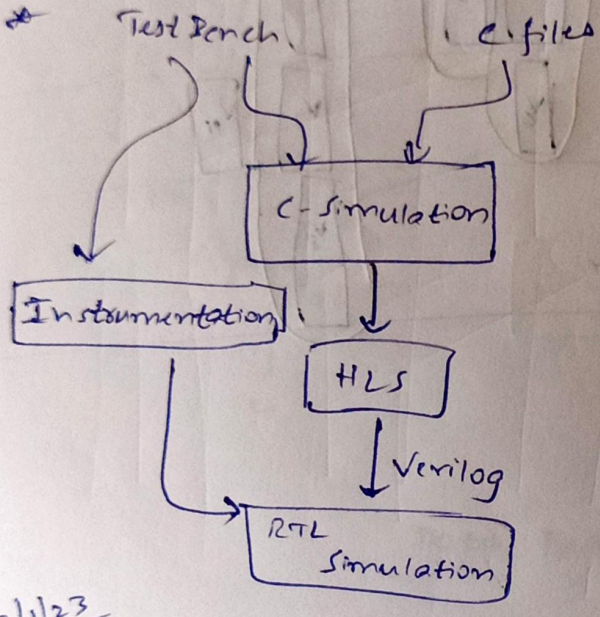
Behavioural level

⇒ Verilog can be written at any of these levels

19/1/23

HLS Tool Demo

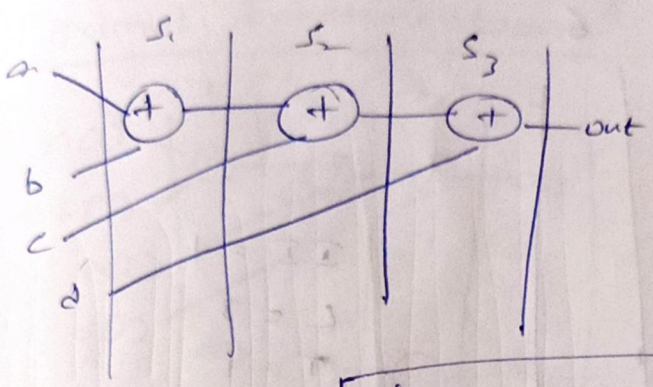
Vivado HLS — Xilinx (VITIS)
2019



23/1/23

Functional pipeline in HLS :-

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



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)

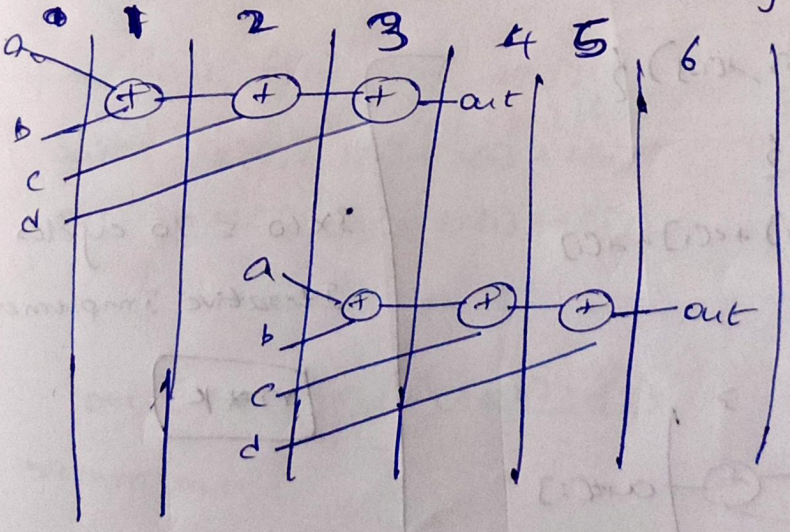
L = Latency

↳ Time required to process one set of inputs

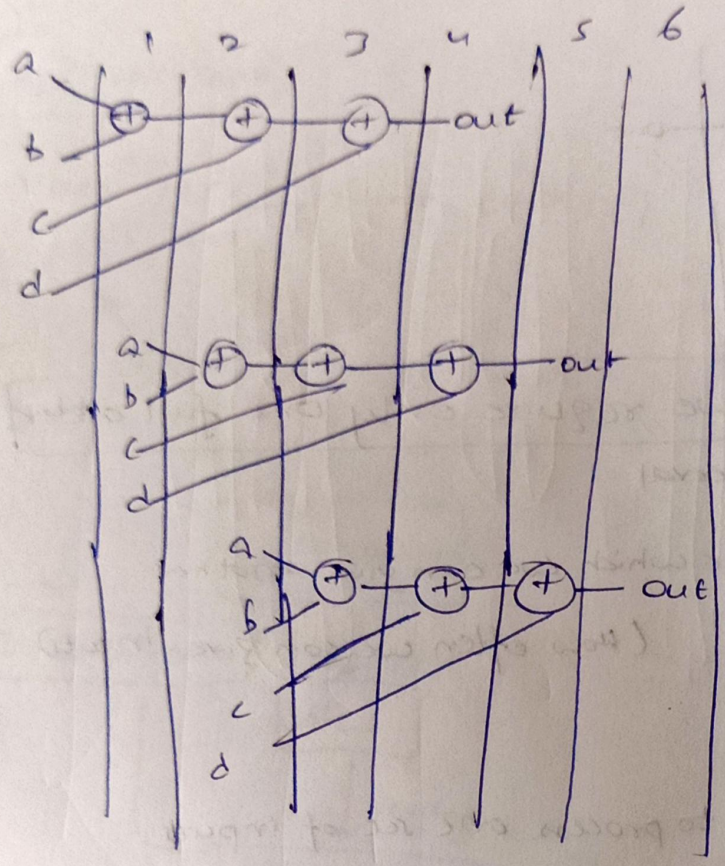
T = Throughput

↳ How often you get output from a design

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



II = 2
L = 3
T = 2



II = 1

L = 3

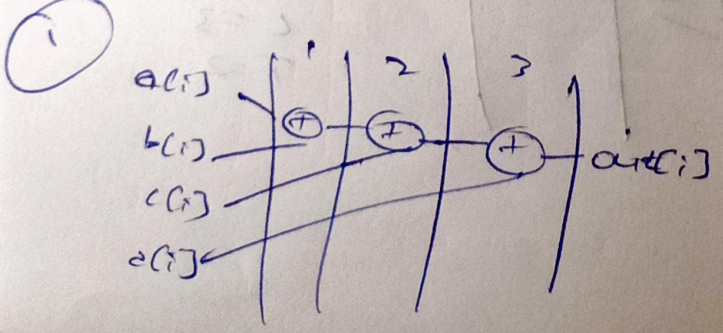
T = 1

Fully Pipelined Design

24/1/25

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
}
```

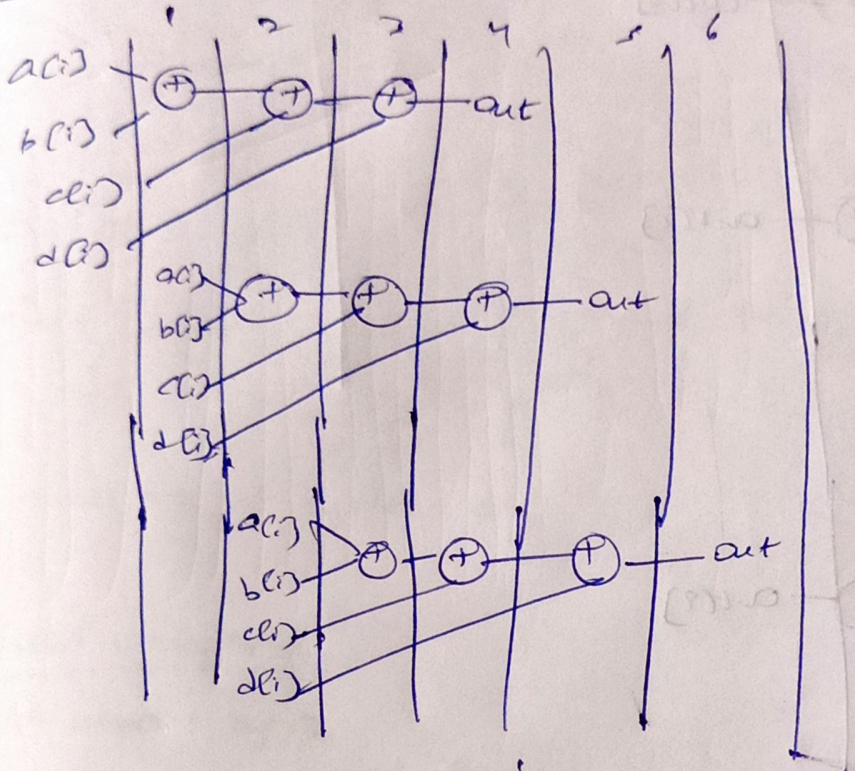


3 x 10 = 30 cycles

Iterative Implementation

n * k

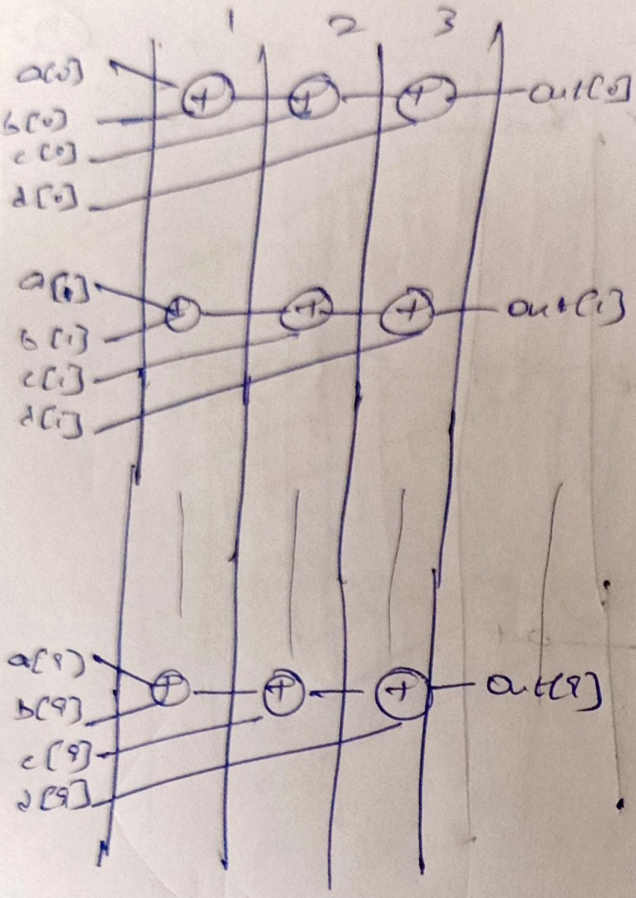
(2) Pipelined Implementation



Total number of cycles = $n+k-1$ = 12 cycles

(3) Unrolled Implementation (Loop unrolling)

```
void fun(a[i], b[i], c[i], d[i]){
    out[0] = a[0] + b[0] + c[0] + d[0]
    out[1] = a[1] + b[1] + c[1] + d[1]
        |          |          |
    out[9] = a[9] + b[9] + c[9] + d[9]
    return out
}
```



Total number of cycles = 3

Best case | k |

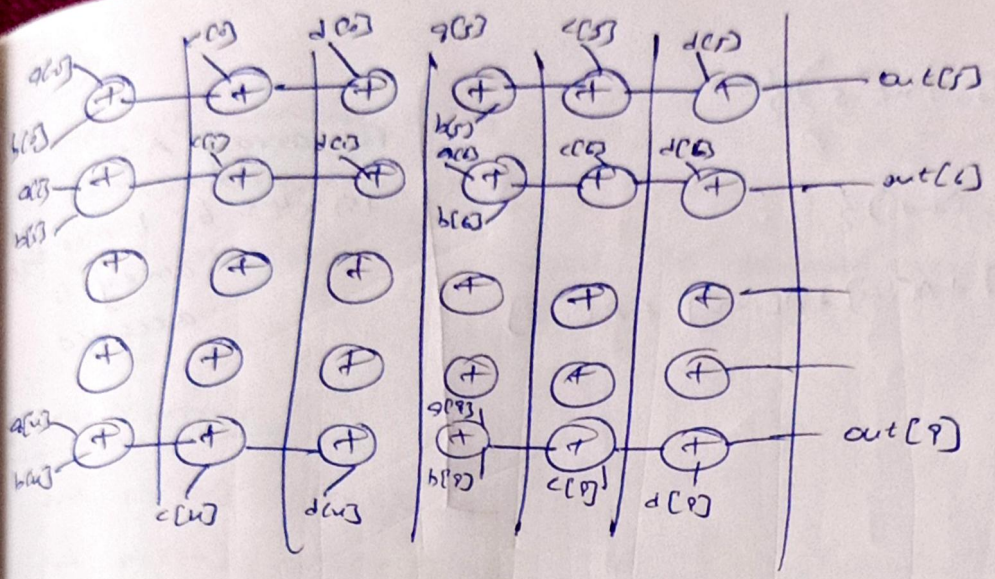
	Time	Resource
Iterative	30	1
Pipelined	12	3
Unrolled	3	10
Partial Unroll	15	2

⇒ Sorta good tradeoff b/w performance and resource

(3a) Unrolled with Resource Constraint

Adders = 5

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



Total no. of cycles = 6

(4) Partial Unrolling

unroll by 2

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

28/1/23

HLS of arrays :-

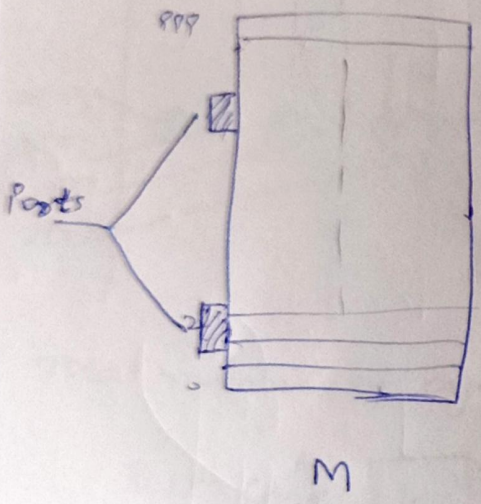
int a[1000] ⟹ RAM/ROM
(Random Access Memory) (Read only Memory)

mapped

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]
    }
}
```

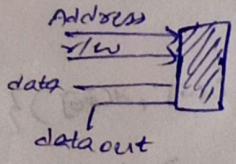
For array A:
17 × 4 = 68 times array is accessed



M

Memory would have single/dual ports

For each port :-




```
void fun (int A[20], int *s){
    t1 = A[0]   t2 = A[1]   t3 = A[2]
    for (i=3; i<20; i++){
        x = A[i]
        s[i] = t1 + t2 + t3
        t3 = x   t2 = t3   t1 = t2
    }
}
```

For array A:-
Total memory access
= 20

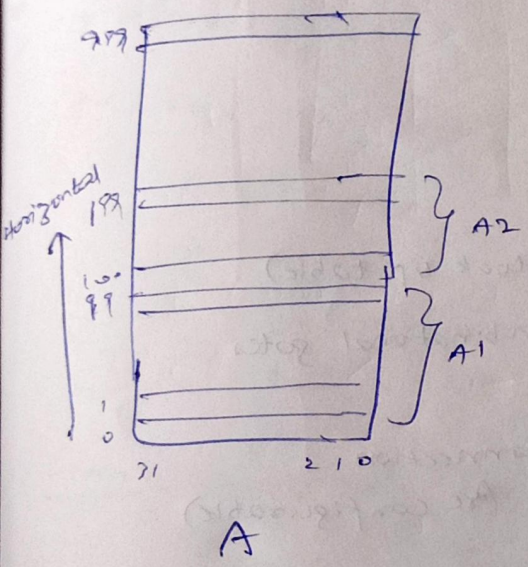
Array

1) Reduce memory access
2) Remove redundant memory access
3) Read once and store locally to reuse

Array Merging

int A1[100] = 100 x 32 bits

int A2[100] = 100 x 32 bits



A

A2[0] ⟷ A[100]

A2[99] ⟷ A[199]

A1[0] ⟷ A[0]

A1[99] ⟷ A[99]

#pragma HLS ARRAY_MAP

variable A1 instance A horizontal

#pragma HLS ARRAY_MAP

variable A2 instance A horizontal

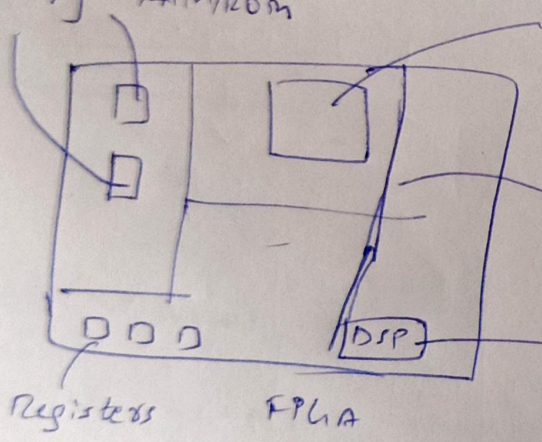
```
int A1[1000] , A2[500]
// map A1 to A , map A2 to A
for (int i=0 ; i<1000; i++) {

    A1[i] = value1

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

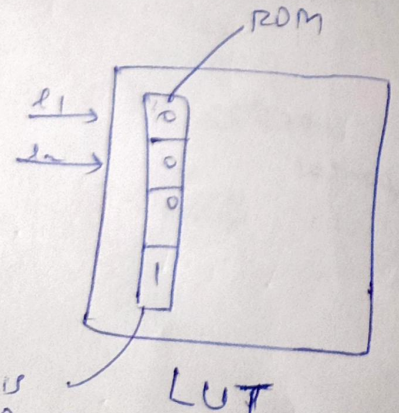
Given: One Dual Port RAM (2000 x 32)

Given: One single port RAM (1000 x 20)

30/1/23

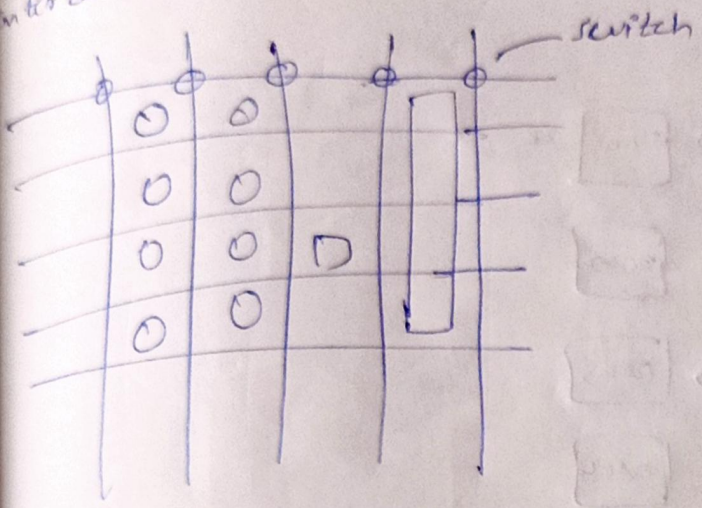
* Memory RAM/ROM





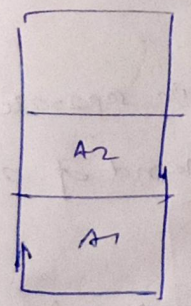
We can do this for any combinational gates

Interconnections:



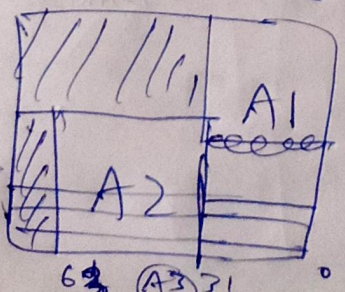

```
int A1[1000], A2[500]

for (i=0; i<1000; i++){
    A1[i] = value1;
    if (i<500)
        A2[i] = value1;
}
```

Given: One BRAM (1000 x 70) single port



⟹ Not possible

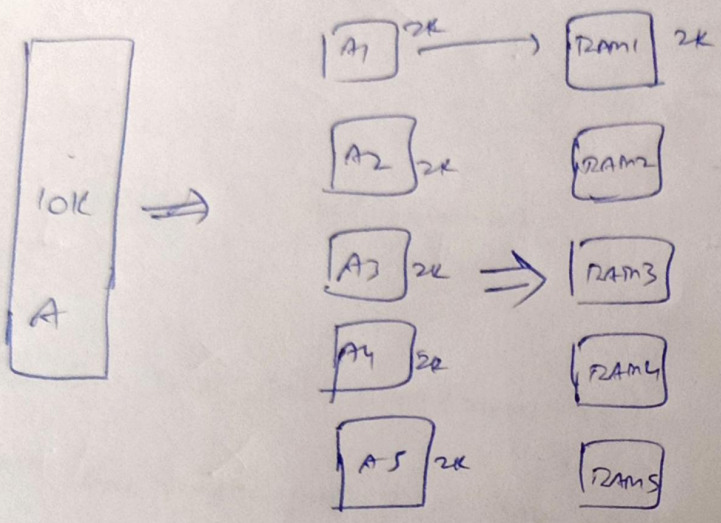
Placed them vertically



A3[i][31:0] = A1[i]

A3[i][69:32] = A2[i]

Array Partition




```
for (i=1; i<10K; i=i+2) {
    B[i] = A[i] + A[i+1]
}
```

A[i] → even location in RAM1

A[i+1] → odd location in RAM2

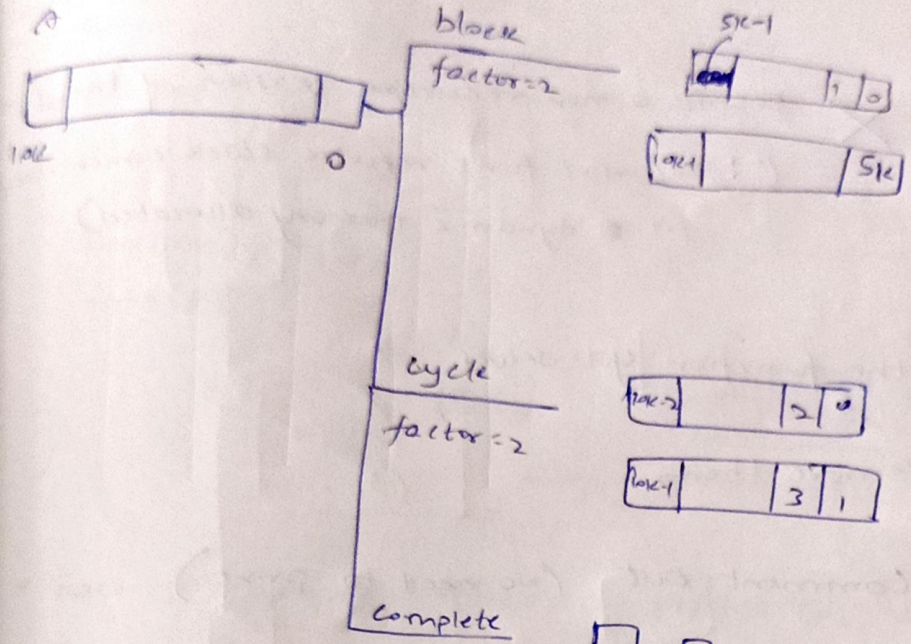
Objective is keeping the parallel access memory in separate RAM.

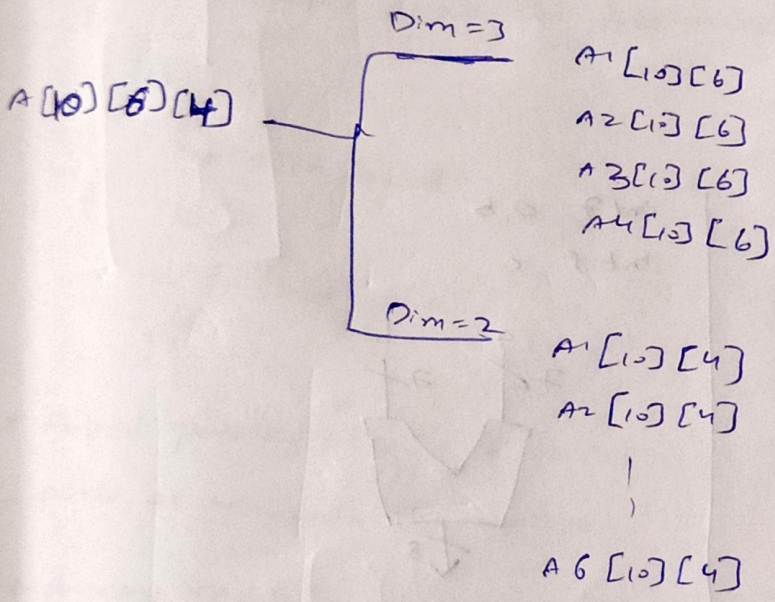
The command for the HLS tool to do the above kind of array splitting is

pragma HLS array-partition Array-Name <type> factor= ,

Dim =

type: block/cycle/complete





* Unsupported 'c' constructs :-

Data types — int, float, char ✓

structure, union ✓

pointers ✓

Array ✓

Dynamic Memory Allocation (malloc/calloc) ✗ Rewrite program by statically allocating memory

Function ✓

Recursive Function ✗ Rewrite a non-recursive version of the fun
(∵ 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)

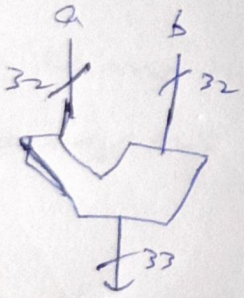
31/1/23

Coding style for RTL :-

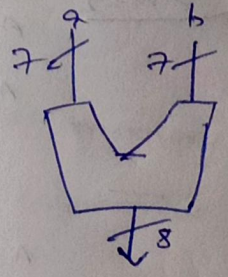
Data types

int 32 bits a, b, c

c = a+b

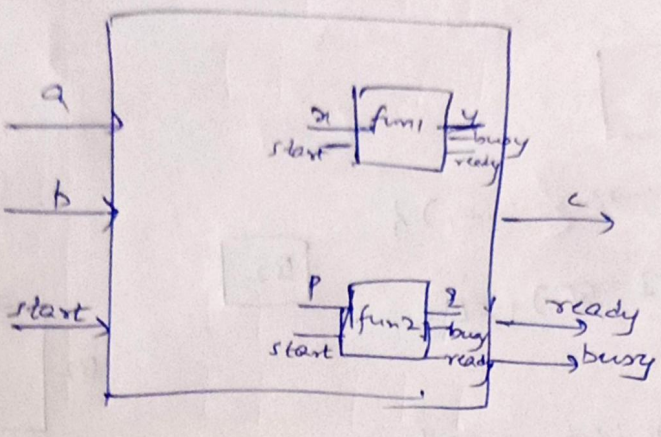


int7 a, b
int8 c



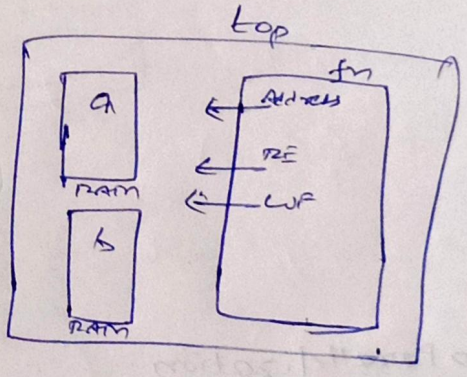
* Data width impacts the area, delay of the design heavily.
* So we must use precise data widths of the inputs.
* Input data width propagates and determine the datapath width in RTL.

```
top (a, b, c){
    |
    |
  func1(x,y)
  func2(p,q)
    |
    |
}
```



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

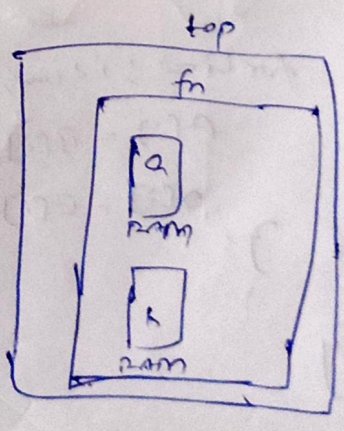
```
top(){
  int a[10], b[10]
    |
  fn(a[0], b[0])
}
```



* Avoid passing arrays to function as they create unnecessary ports at module boundaries.
* Array may be duplicated to avoid these.

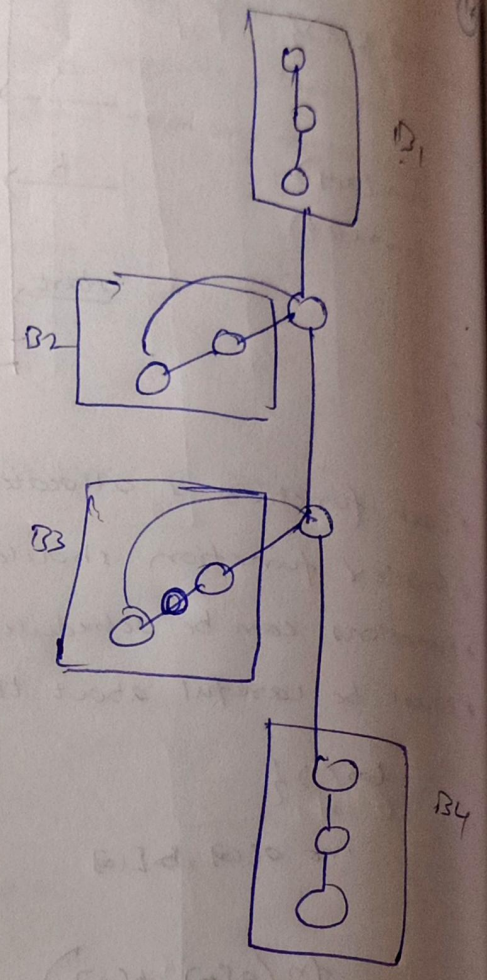
```
top(){
  |
 fn()
  |
}
```

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



Loops

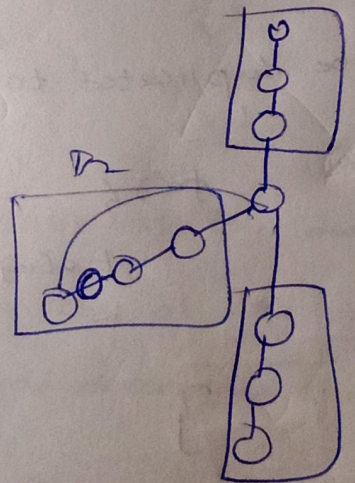
```
fn( ){
    B1
    for(i=1 ; i<100 ; i++){
        A[i] = B[i] + C[i]        B2
    }
    for(i=1; i<100; i++){
        D[i] = E[i] + F[i]        B3
    }
    B4
}
```



Total time = B1 + 100 × B2 + (100 × B3) + B4
= 3 + 200 + 300 + 3
= 506

Loop Parallelization

```
fun() {
    |
    |
    for(i=1 ; i<100; i++){
        A[i] = B[i] + C[i]
        D[i] = E[i] + F[i]
    }
    |
}
```



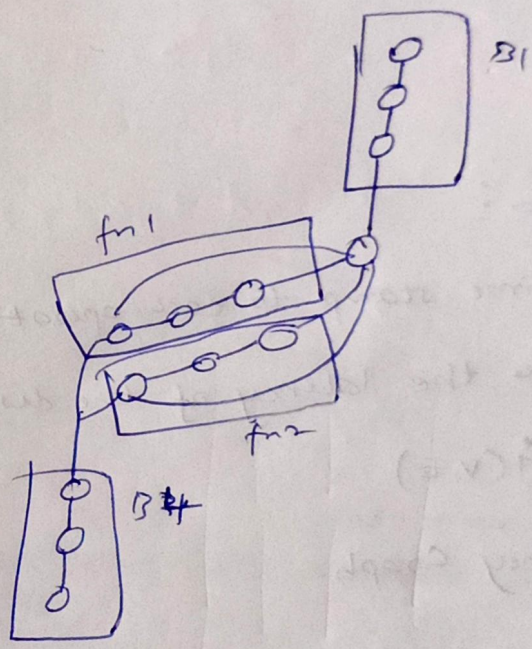
Total time = 3 + 100 × 3 = 406

Limit1 & Limit2

```
fun() {
  |
  for(i=1 ; i<100 ; i++) {
    A[i] = B[i] + C[i]
  }
  for(i=1 ; i<200 ; i++) {
    D[i] = E[i] + F[i]
  }
  |
}
```

⇒

```
fun() {
  |
  fun1()  //Loop1
  fun2()  //Loop2
  |
}
```



Putting the loops in functions will make the HLS 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 : ⟸ more Preffered

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

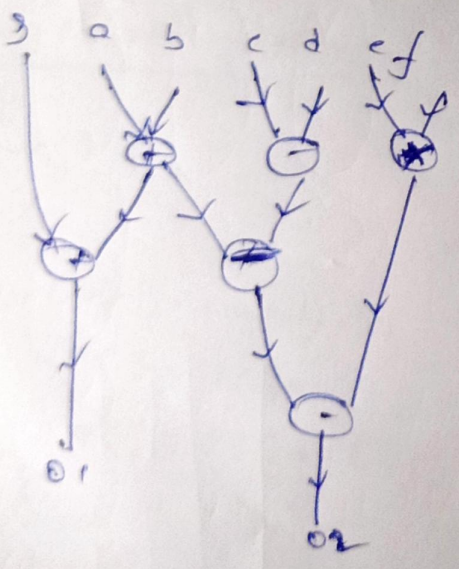
13/2/23

Scheduling :

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

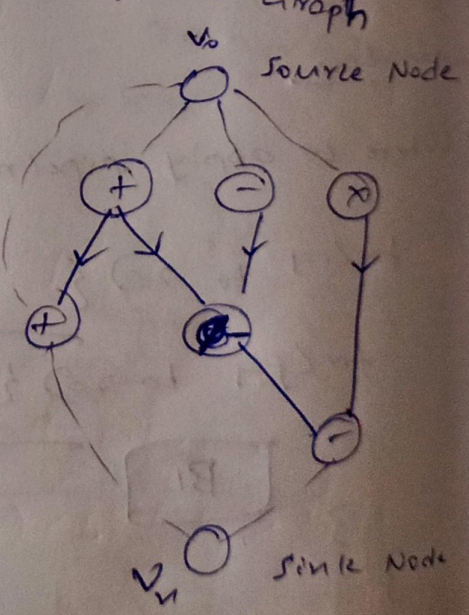
Input :- $G(V, E)$

Dependency Graph



⇒

Sequence Graph



Input:

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

(ii) $D = \{d_1, \ldots, d_n\}$

d_i is the delay of operation v_i

(iii) $r, V \rightarrow \{1, \ldots, n_{res}\}$ here n_{res} is type of operation

(iv) Resource Constraints $a_k : k = 1, 2, \ldots, n_{res}$

(v) Latency Constraints λ

Output:

(i) $T \{t_i \mid i = 0, 1, \ldots, n\}$ t_i represents the START time of node v_i

(ii) $(t_n - 1)$ is the latency $t_0 = 0$

$\phi : V \rightarrow Z^+$

$\phi(v_i) = t_i$

1. Unconstrained scheduling : (US)

Given G and D find $\phi : V \rightarrow Z^+$ 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

(II) Minimize latency under Resource Constraints (ML-RC)

Given:

(I) G (II) D (III) γ (IV) a_k

Output:

$\phi : V \rightarrow z^+$ i.e., $\phi(v_i) = t_i$ s.t ϕ satisfies

(I) Dependency constraints

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

(II) $\{v_i : \gamma(v_i) = k$ and $t_i \leq \lambda \leq t_i + d_i - 1\} \leq a_k$ for each $k = \{1, 2, \rightarrow n_{res}\}$ and for each time step $\lambda = 1, 2, \rightarrow t_n$

$\Rightarrow$ Resource constraint

(III) t_n is minimum

(III) Minimize Resource under Latency Constraint (MR-LC)

Given:

(I) G (II) D (III) γ (IV) λ (Latency Limit)

Output:

$\phi : V \rightarrow z^+$ i.e., $\phi(v_i) = t_i$ s.t ϕ satisfies

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

(II) Latency constraint $t_n \leq \lambda + 1$

$a_1, a_2, \ldots, a_m$ is minimum ⇒ Each resource type is minimized

- ML-RC
- MR-LC

} NP-Complete Problem

US is not.

Unconstraint scheduling (US)

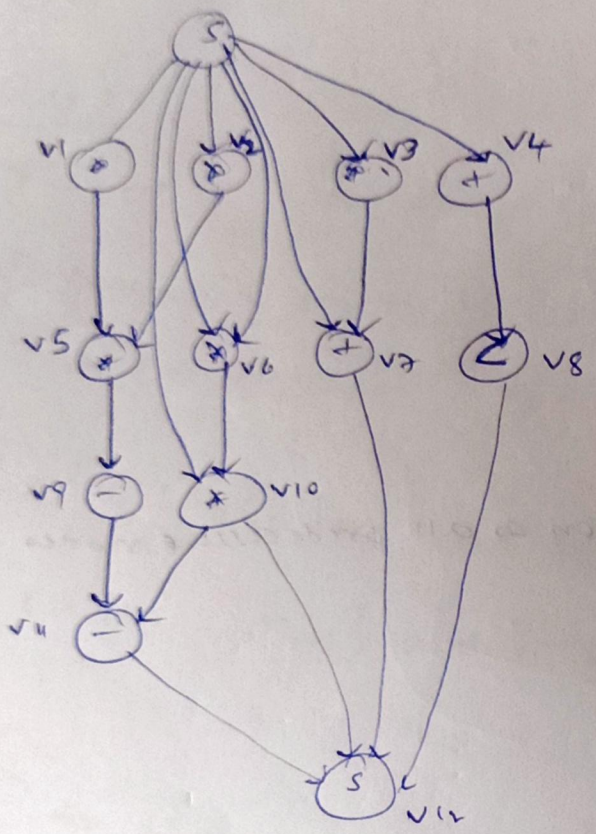
(i) ASAP (As Soon As Possible)

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

```
ASAP (G, D, r) {
    to = 0
    while (unscheduled node exist) {
        1. Select a node Vj whose predecessors
           have already been scheduled
        2. Schedule node Vj to tj where
           tj = MAX (di + ti)  ∀ (vi, vj) ∈ E
    }
}
```

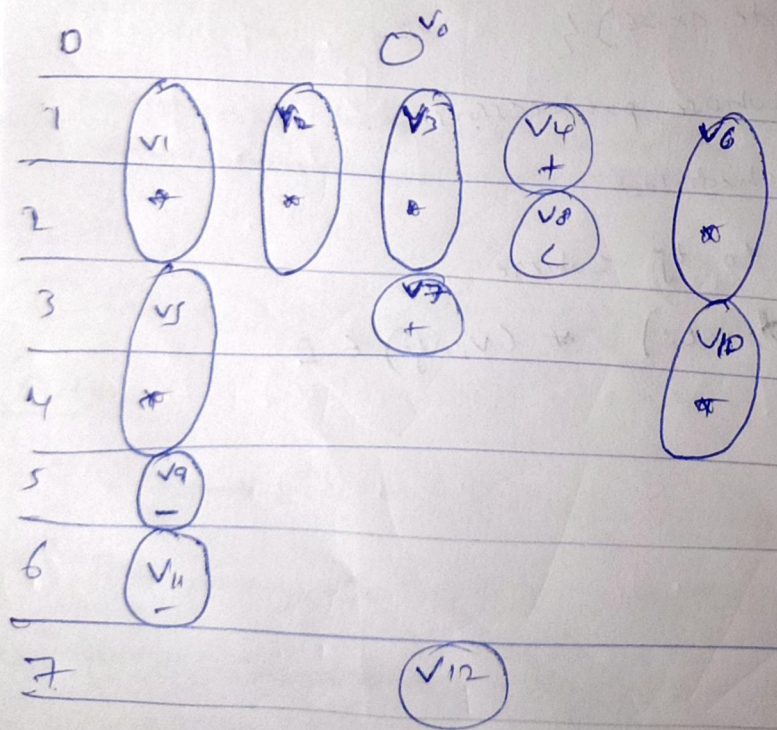
14/8/23

ASAP Scheduling (9, 7, 1)



$d(*) = 2$

$d(+, -, <) = 1$



$\lambda = 6$

ALAP (As Late As Possible)

OLAP (G, γ, D, λ) {

↳ Latency based

$t_n = \lambda + 1$

while (any node unscheduled) {

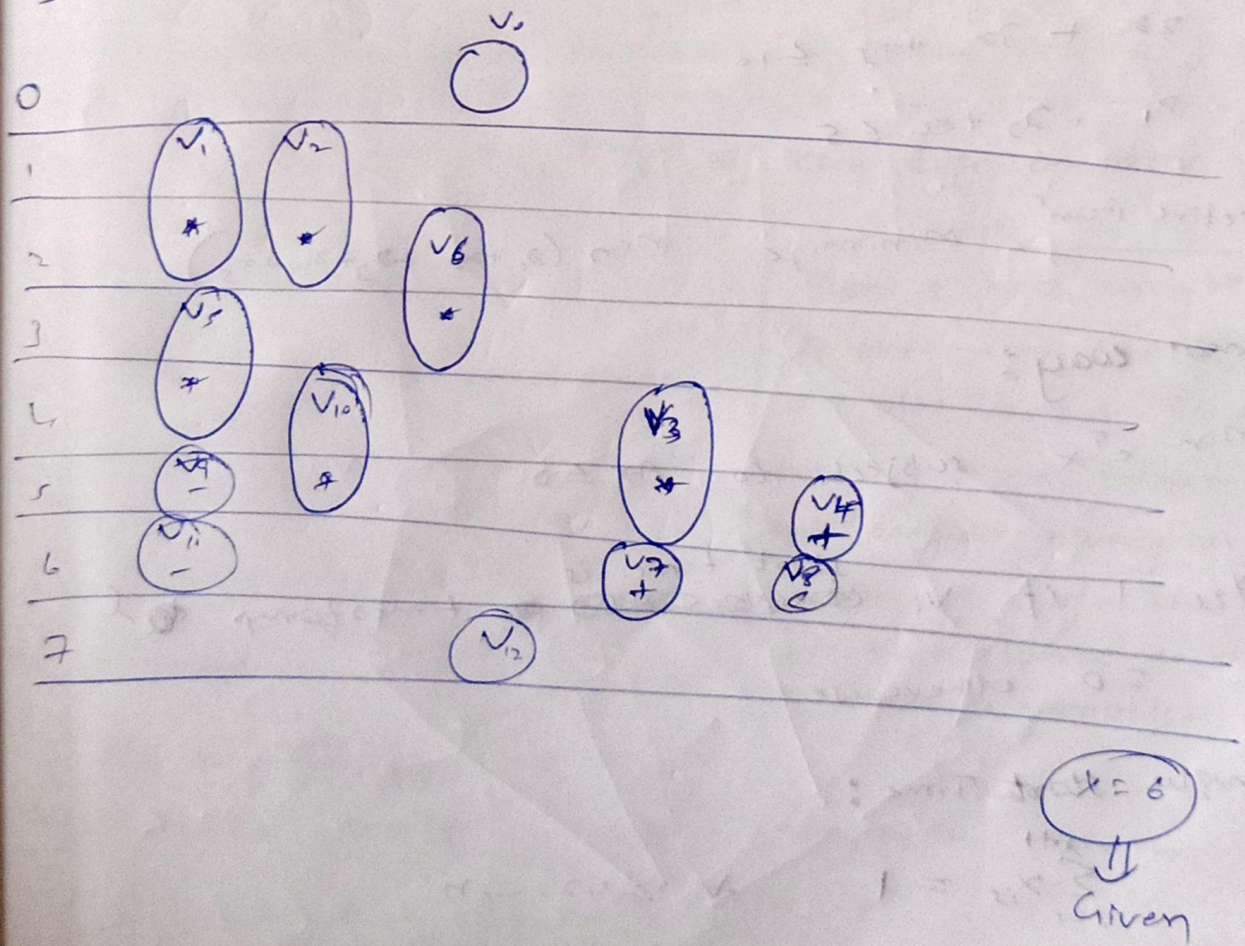
1. Select a node V_i whose successor are already scheduled.

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

}

if ($t_0 < 0$) "λ is not sufficient" error

}



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

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

Mobility of node v_i

$$= t_i^{ALAP} - t_i^{ASAP}$$

ILP formulation of Scheduling

ILP → Integer Linear Programming

$x_1, x_2 \dots x_n$ unknown integer variables

Constraints

$$2x_1 + 3x_2 + x_5 \leq 10$$

$$x_1 - x_3 + x_4 \leq 5$$

Objective Fun^c — Minimize $\min(x_1 + x_2 + x_3 + x_4 + x_5)$

Formal way:

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

$x_{il} = 1$ if v_i start time is l timestamp

$= 0$ otherwise

(1) Unique start Time:

$$\sum_{l=1}^{ALAP+1} x_{il} = 1 \quad \forall i = 1, 2, \dots, n$$

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

(2) Data dependencies / Precedence constraints :

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

i.e.,

$$\sum_{l=1}^{\lambda} x_{jl}\, l \geq \sum_{l=1}^{\lambda} x_{il}\, l + d_i$$

(3) Resource Constraints :

$$\sum_{i \cdot \gamma(v_i)=k} \; \sum_{m=l-d_i+1}^{l} x_{im} \leq a_k$$

$k = 1, 2, \ldots, n_{res}$

$l = 1, 2, \ldots, \lambda$

(4) Latency constraints :

$$\sum_{l=1}^{\lambda+1} l \cdot x_{nl} \leq \lambda + 1$$

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

Given a_k : resource limit of each type operator

1. Unique start Time
2. Data dependency constraints
3. Resource Constraints

Obj° : Minimize λ

i.e., Minimize t_n

Objective fun^c : $c^T t$ $\Rightarrow c_1 t_1 + c_2 t_2 + \ldots + c_n t_n$

$c^T = [\underset{c_1}{0} \; \underset{c_2}{0} \; \underset{c_3}{0} \; \ldots \; \underset{c_n}{1}]$

Here l is a variable. So the no. of variables x_{il} are also variable. In order for this to not happen we will consider a very large value of λ for the number of time stamps so that it is not a variable.

* ASAP :

Objective func^{c} : $c^T t$

$\vec{c}^T = [1 \quad 1 \quad 1 \quad \dots \quad 1]$

1. Unique Start Time
2. Data dependency Constraints

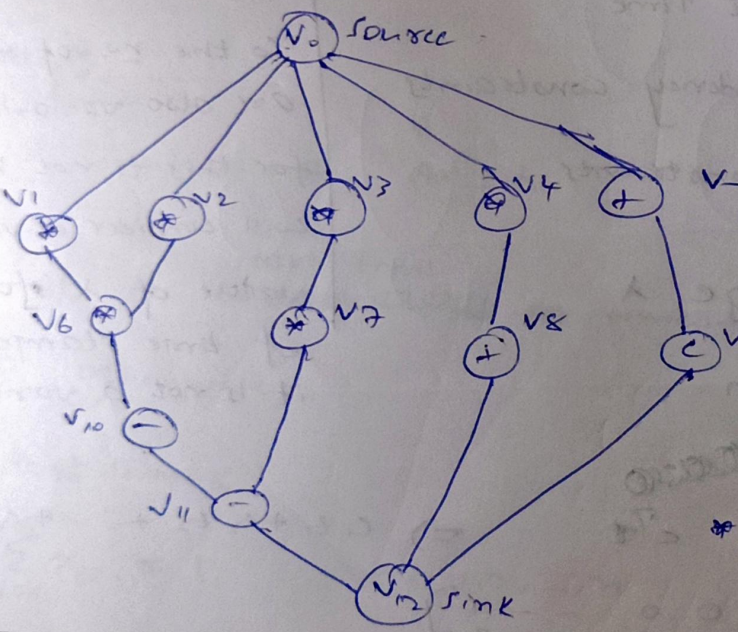
* MR-LC :

Objective func^{c} : $c^T . a$

$c^T = [c_1 \; c_2 \; \dots \; c_k]$ $\quad c_i \Rightarrow$ Value corresponding to the appropriate area occupied by the hardware

1. Unique Start Time
2. Data dependency Constraints
3. Resource Constraints where a_{ik} is a variable
4. Latency Constraints

*



*, −, +, < ⇒ Single cycle

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

MP-2c :

$x = 4$

Mobility of V_1, V_{21}, V_6, V_{10} and V_{11} is zero (∵ Given $\lambda = 4$)

$V_1 \Rightarrow x_{11}$ $V_2 \Rightarrow x_{21}$ $V_6 \Rightarrow x_{62}$ $V_{10} \Rightarrow x_{10,3}$ $V_{11} \Rightarrow x_{11,4}$

$V_3 \Rightarrow x_{31}, x_{32}$ $V_7 \Rightarrow x_{73}, x_{72}$

$V_4 \Rightarrow x_{41}, x_{42}, x_{43}$ $V_8 \Rightarrow x_{82}, x_{83}, x_{84}$

$V_5 \Rightarrow x_{51}, x_{52}, x_{53}$ $V_9 \Rightarrow x_{92}, x_{93}, x_{94}$

Unique start time :

$x_{11} = 1$ $x_{21} = 1$

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

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

Dependency Constraints :

(V_1, V_6) : $2 \cdot x_{62} \geq 1 \cdot x_{61} + 1$

(V_4, V_8) : $2x_{82} + 3x_{83} +$

Resource Constraints:

For time stamp 1:

Mult: $x_{11} + x_{21} + x_{31} + x_{41} \leq a_1$

ALU: $x_{51} \leq a_2$

For time stamp 2:

mult: $x_{62} + x_{32} + x_{42} + x_{92} \leq a_1$

ALU: $x_{82} + x_{52} + x_{92} \leq a_2$

Latency Constraint:

$1 \cdot x_{12,1} + 2 \cdot x_{12,2} + \cdots + 5 \cdot x_{12,5} \leq \lambda + 1$

2nd 2/23

Multiprocessor Scheduling:

Assumption

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

Earlier $a_1, a_2, \ldots, a_{n_{res}}$ ⇒ Now a (mixture of ALU) → No. of processing units

(A2) All operations have unit execution time

Scheduling problem is really NP-complete

ILP ⇒ Multiprocessor ML-RC :

1. Unique start Time ⇒ SAME

2. Data dependency

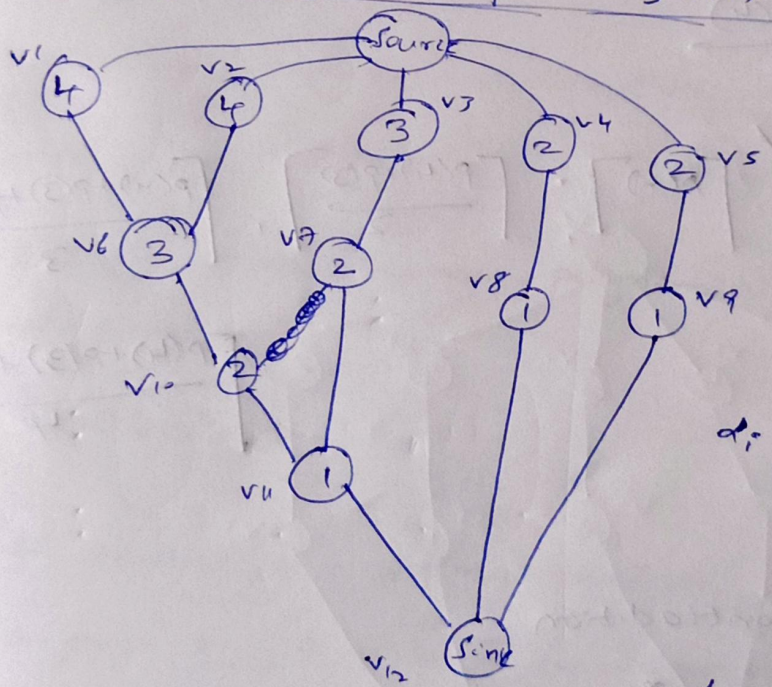
$$\sum_{l=1}^{\lambda+1} l \cdot x_{jl} \geq \sum_{l=1}^{\lambda+1} l \cdot x_{il} + 1 \qquad \forall (v_i, v_j) \in E$$

3. Resource Constraints

$$\sum_{i} x_{il} \leq a \qquad l = 1, 2, \ldots \lambda+1$$

4. Objective Function ⇒ SAME

Resource Bound under A1 < A2 for a given latency λ :



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

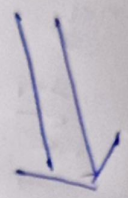
α_i = Label of node v_i denotes the largest path from v_i to sink

$\alpha = \max_{1 \leq i \leq n} \alpha_i$

$\alpha = 4$ $\quad \lambda \geq \alpha$

* Hu's Theorem : A lower bound on the number of resources to complete a schedule with latency λ is

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



Problem is NP-complete even with the two assumptions

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

If $\gamma=1$:

$$\bar{a} = \frac{p(4)}{1} = p(4)$$

If $\gamma=2$:

$$\bar{a} = \frac{p(3)+p(4)}{2}$$

So $\bar{a} = \max\left(\lceil p(4) \rceil, \left\lceil \frac{p(4)+p(3)}{2} \right\rceil, \left\lceil \frac{p(4)+p(3)+p(2)}{3} \right\rceil, \left\lceil \frac{p(4)+p(3)+p(2)+p(1)}{4} \right\rceil \right)$

$= 3$

(* Proof:

Let's prove by contradiction

Let $\bar{a} = \arg\max \gamma^*$

Let's assume that there exist a schedule λ with $< a_{\gamma^*}$

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

$a \cdot (r^* + \lambda - a) < \sum_{j=1}^{r^*} p(a+1-j) = p(a) + p(a-1) + \cdots + p(a+1-r^*)$

$\Downarrow$

No. of operations scheduled till time stamp $(r^* + \lambda - a)$

Atleast some node with label $(a+1-r^*)$ remain to be scheduled

$\Rightarrow$ Atleast $(a+1-r^*)$ steps needed to complete the schedule

$\therefore$ Total schedule length $\geq (a+1-r^*) + (r^* + \lambda - a)$

$= \lambda + 1$

(Contradiction)

21/2/23

* With the addition of another assumption,

(A3) Sequence Graph is a tree

the HU's algorithm becomes polynomial time solvable from NP-Complete

i.e. HU's algo $\Longrightarrow O(n)$

* HU's algo always give optimal solution.

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

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

a → Resource Bound

* HU's algo (G, a) {

Calculate d_i for each V_i

$l = 1$

while (there is a unscheduled node) {

① U = set of nodes ready to be scheduled (i.e., all predecessors are scheduled)

② Select $S \subseteq U$ such that $|S| \leq a$ and d_i's of S are maximal

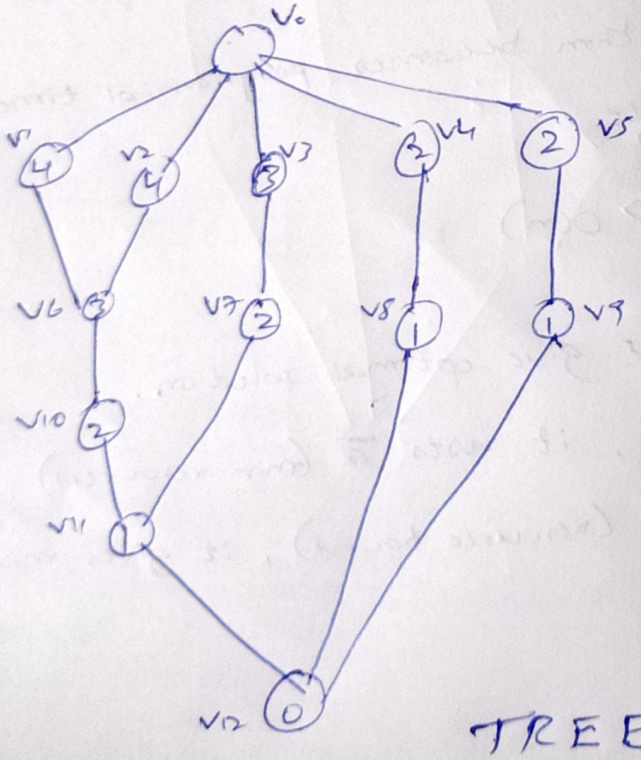
③ Schedule all V_i's in "l" timestamp

④ l++

}

}

* Example solving of Diffeq:



TREE

(a=3) ⇒ Calculated by using HU's formula

λ=1

U = {V1, V2, V3, V4, V5}

S = {V1, V2, V3} ⟹ in TS=1

λ=2

U = {V4, V5, V6, V7}

S = {V6, V7, V4} ⟹ in TS=2

λ=3

U = {V10, V8, V5}

S = {V10, V8, V5} ⟹ in TS=3

λ=4

U = {V11, V9}

S = {V11, V9} ⟹ 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)

× × ×