NUMERO: 2275A ANNO: 2017

A P P U N T I

STUDENTE: Preatto Stefania

MATERIA: Integrated - Systems Architecture - Prof. Masera

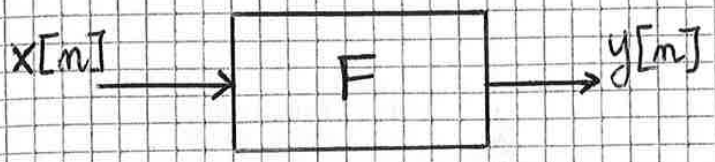
ALGORITHM TO ARCHITECTURE MAPPING

02/10/17

WE WANT TO MOVE FROM ONE DESCRIPTION OF THE PROBLEM AND TO MAP IT INTO AN ARCHITECTURE

THIS CAN BE DONE THROUGH A FORMAL METHOD

LET'S CONSIDER A **DIGITAL FILTER**:



GIVEN AN INPUT STREAM $x[n]$ WE OBTAIN $y[n]$ AS A RESULT OF THE PROCESSING DONE BY THE FILTER.

A POSSIBLE OUTPUT IS: $y[n] = a \cdot x[n] + \underbrace{b \cdot x[n-1] + c \cdot x[n-2]}$

ADDITIONAL SAMPLES OBTAINED IN PREVIOUS TIMES

TO REPRESENT THE FILTER WE USE THE **DATA FLOW GRAPH**

IT IS A REPRESENTATION COMPOSED BY

- **NODES** → ASSOCIATED WITH COMPUTATION
- **ARCS** → WHICH REPRESENT DIFFERENT ELEMENTS
  - A FLOW OF DATA → HOW COMPONENTS ARE ALLOCATED
  - DATA CONSTRAINTS → EG: THERE ARE 2 OPERATIONS AND THE 2nd OPERATION HAS TO BE DONE AFTER THE 1st ONE (THIS COMES OUT FROM CONSTRAINTS)
  - **DELAY** → 2 DIFFERENT KINDS
    - COMBINATIONAL DELAY → eg: OF A COMBINATIONAL CIRCUIT, LIKE AN ADDER
    - SEQUENTIAL DELAY → EG: A REGISTER THAT STORES A VALUE AND RESETS IT

METRICS RELATED TO SPEED

LET'S CONSIDER 2 DIFFERENT

LATENCY

IT'S A DELAY INFO
IT REPRESENTS HOW MUCH WE NEED TO WAIT AT THE OUTPUT OF THE CIRCUIT TO RECEIVE A VALID DATA FROM THE INPUT

YOU NEED TO MEASURE THE PATH FROM THE INPUT x TO THE OUTPUT y

⇓

IS A POSSIBLE PATH

IN THE CASE OF THE FILTER IT'S: $T_m + 2T_a$

THROUGHPUT

IT'S A NUMBER WHICH SAYS HOW MANY SAMPLES ARE GENERATED PER UNIT TIME AS OUTPUT

⇓

YOU CAN GENERATE AN OUTPUT AT EVERY CLOCK CYCLE

FOR THIS SPECIFIC CASE WE HAVE

$$th = \frac{1}{T_{ck}}$$

⇓

WE CAN EVALUATE THE MAXIMUM T_{ck} OF THE FILTER:
THE LONGEST PATH IS THE **CRITICAL PATH**

⇓

IT'S THE PATH WITH THE LONGEST DELAY FROM 1 FF TO AN OTHER FF (REGISTER/MEMORY ELEMENT)

DUMMY REGISTERS ARE NECESSARY

⇓

LET'S SUPPOSE THAT INPUT ELEMENTS ARE SENT THROUGH REGISTERS AND THE SAME CONCURRENCY AT THE OUTPUT

PATHS:
- A→B, IN→A } DELAY IS IDEALLY ∅
- B→OUT : $T_m + T_a$
- A→OUT : $T_m + 2T_a$
- IN→OUT : $T_m + 2T_a$

→ THE CRITICAL PATH IS

$T_{cp} = T_m + 2T_a$

⇓

SO $T_{ck} \geq T_{cp} = T_m + 2T_a$

ASSUMING: $\frac{T_a = 1}{T_m = 2} \Rightarrow T_{cp} = 4$

⚠ OUR CIRCUIT IS EXPECTED TO WORK FOR A MAXIMUM FREQUENCY

$$th \leq \frac{1}{T_m + 2T_a}$$

IT'S JUST AN APPROXIMATION BECAUSE WE ARE NEGLECTING A FEW ADDITIONAL CONTRIBUTIONS

- $T_{ck \to Q}$ CLOCK-TO-OUTPUT DELAY FROM THE STARTING FF
- T_{su} SET-UP TIME AT THE ENDING FF
- ADDITIONAL DELAYS ALONG THE PATH

WE CONSIDER $< \frac{T_m}{T_a}$ } THAT ARE MORE AFFECTING DELAYS

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 5 di 190

IN THIS WAY THE FILTER IS WORKING IN A CORRECT WAY

$$y[n] = c \cdot x[n-3] + b \cdot x[n-2] + a \cdot x[n-1]$$

THE COMPUTATION IS THE SAME AS THE ORIGINAL EXCEPT FROM A DELAY OF ONE SAMPLE

⇓

SO IT'S A CORRECT MODIFICATION

NOW WHAT IS THE LONGEST PATH?

$$T_{cp} = \max\{T_m, 2T_a\} \longrightarrow$$

ASSUMING $T_a = 1$, $T_m = 2$

$T_{cp} = 2$ THAT IS BETTER THAN BEFORE ($T_{cp_{pre}} = 4$)

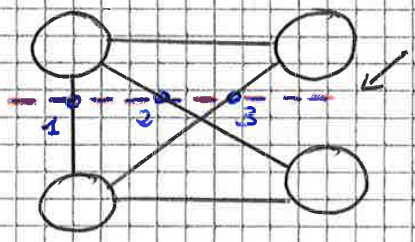
⇒ IN THIS CASE $throughput = \frac{1}{2}$ (vs $th_{orig} = \frac{1}{4}$)

WE IMPROVED PERFORMANCES

⚠ NB WE DON'T HAVE TO PLACE REGISTERS IN A RANDOM WAY

LET'S INTRODUCE 2 DEFINITIONS:

- **CUT-SET** = SUBSET OF ARCS OF THE GRAPH THAT WE CAN ELIMINATE BY SEPARATING THE GRAPH INTO 2 DIFFERENT PARTS THAT ARE SEPARATED BY EACH OTHERS.



BY REMOVING THESE 3 ARCS

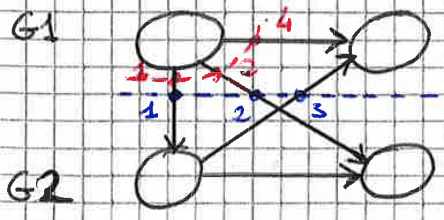
⇓

THERE ARE NO MORE CONNECTIONS, THEY'RE ISOLATED FROM EACH OTHERS

⇓

YOU CUT THE GRAPH INTO 2 PARTS

- **FEED FORWARD CUT-SET** = IT'S A CUT-SET WITH AN ADDITIONAL PROPERTY, WHERE EACH CUT HAS THE SAME DIRECTION



ARCS 1-2-3 FORM A CUT-SET BUT IT'S NOT FEED FORWARD CUTSET

BECAUSE CUT 3 IS IN THE OPPOSITE DIRECTION

1-2 COME FROM G1 TO G2
3 COMES FROM G2 TO G1

A GOOD CUT-SET CAN BE --- ⇒ THAT CONCERNS ARCS 1-2-4

⇓

EACH ARC HAS THE SAME DIRECTION!

YOU CAN APPLY THESE 2 DEFINITIONS TO THE DFG AND CONSIDER IF IT'S ACCEPTABLE

⇓

SO IF IT'S MODIFYING THE DFG IN A CORRECT WAY WITHOUT AFFECTING THE ALGORITHM WHICH IS IMPLEMENTED BY THE FILTER

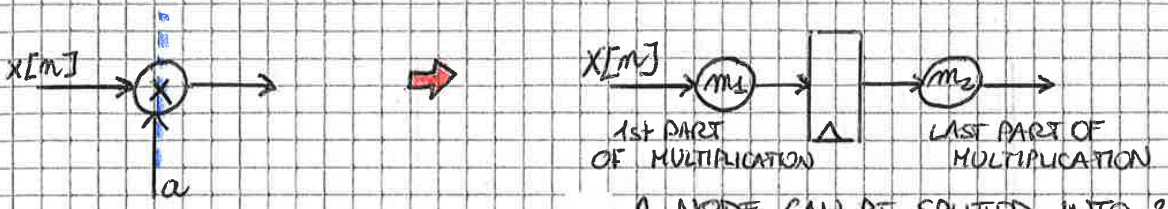
06/10/17

① 2 VERSIONS OF PIPELINING → COARSE GRAIN → WITH CUT-SET WE IDENTIFY PROPER POSITIONS (ARCS) OF THE PIPE WHERE REGISTERS COULD BE INSERTED

→ FINE GRAIN → WE INSERT A PIPE REGISTER INSIDE A PROCESSING NODE

⇓

WITH A GIVEN NODE IN THE GRAPH WE MAY IMAGINE TO SPLIT ITS ARITHMETIC IN MULTIPLE STEPS, EACH ALLOCATED TO A CLOCK CYCLE



A NODE CAN BE SPLITED INTO 2 OR MULTIPLES SUB-NODES BY MEANS OF REGS

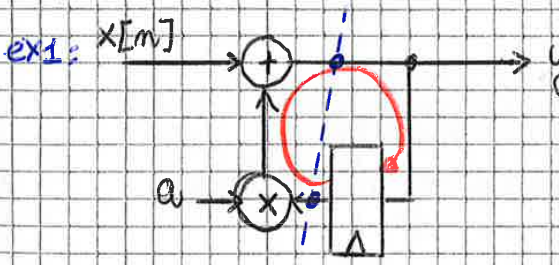
THIS PROCESS IS USUALLY APPLIED TO MULTIPLICATIONS (MULTIPLE ADDITIONS ARE DIVIDED INTO 2 CLOCK CYCLES)

⚠ IN THIS CASE WE DIVIDE A NODE AND WE DON'T PLACE REGISTERS ALONG ARCS.

FILTERS WITH LOOPS

GIVEN A DIGITAL FILTER THAT HAS A LOOP:

ex1:



IIR = INFINITE IMPULSE RESPONSE FILTER

$$y[n] = x[n] + a\,y[n-1]$$

CAN WE APPLY PIPELINE?

NO BECAUSE WE MUST BE ABLE TO FIND A FEED FORWARD CUT-SET FOR ANY DIRECTION YOU CAN IDENTIFY IN THIS CASE

⇓

IF YOU TRY TO INSERT A REGISTER IN ● YOU OBTAIN A TOTALLY DIFFERENT FILTER

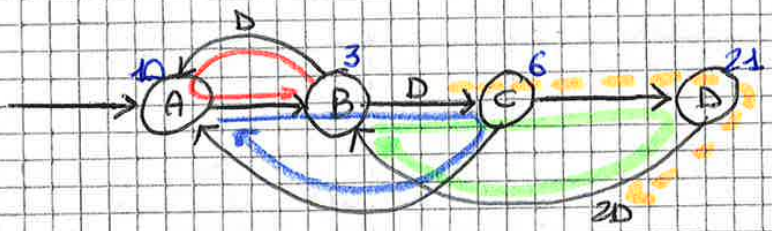
⇓

NOT ALLOWED !

PIPELINING TECHNIQUE CANNOT BE USED

AN OTHER TECHNIQUE THAT IS POSSIBLE IS RETIMING

EX2: GIVEN A DFG LIKE THIS



EACH NODE HAS A NUMBER THAT STANDS FOR ITS COMBINATIONAL DELAY (D costs 21 unit times)

IN ORDER TO COMPUTE Lb AND Ib → LET'S IDENTIFY LOOPS

$loop_1$: A-B-A $\qquad Lb_1 = \frac{10+3}{1} = 13$

$loop_2$: A-B-C-A $\qquad Lb_2 = \frac{10+3+6}{1} = 19$

$loop_3$: B-C-D-B $\qquad Lb_3 = \frac{3+6+21}{3} = \frac{30}{3} = 10$

$$T_\infty = \max\{Lb_1, Lb_2, Lb_3\} = 19$$

YOU CAN SEE THAT THIS AMOUNT IS THE EFFORT WE HAVE IN A SINGLE CYCLE

⇓

IF WE ARE ABLE TO DIVIDE THE TOTAL EFFORT IN A UNIFORM WAY → THIS CAN BE THE BEST OF MAXIMIZATIONS OF THE CLOCK THROUGHPUT

HINT: TRY TO DISTRIBUTE THE EFFORT IN A UNIFORM WAY

⇓

BY REARRANGING OUR COMPUTATION

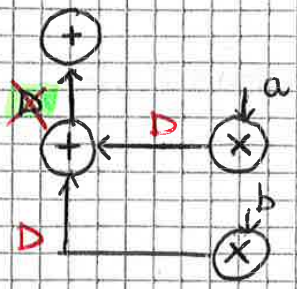
In EX2: $T_{cp} = 6+21 = 27 = T_{clk}$

$T_\infty = 19 \Rightarrow T_\infty < T_{cp}$

⇓

IT'S POSSIBLE TO IMPROVE PERFORMANCES TO MAKE THE CRITICAL PATH BECOMING EQUAL TO THE ITERATION BOUND

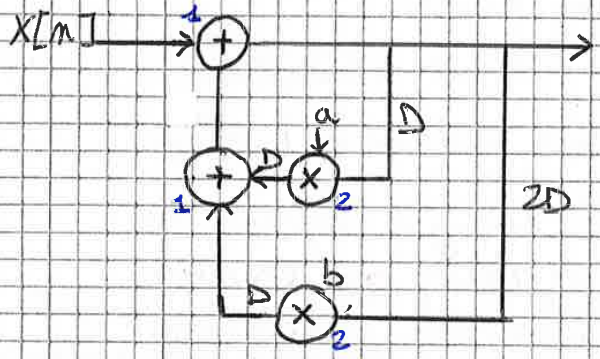
in ex3: IN THE INNER LOOP NOTE THAT IF YOU MOVE D AFTER THE MULTIPLIER ↓ YOU CREATE AN OTHER CRITICAL PATH ↓ SO AN OTHER POSSIBILITY CONSISTS IN CONSIDERING THE REGISTER D



RETIMING ON ⊕ NODE HAS TO BE DONE IN THIS WAY

BY DOING THIS OPERATION YOU DON'T AFFECT THE BEHAVIOUR OF THE FILTER

IN THIS WAY WE OBTAIN THE RETIMED DFG OF ex3



$T_{cp} = T_{ma} + T_a = 2 = T_m = T_\infty$

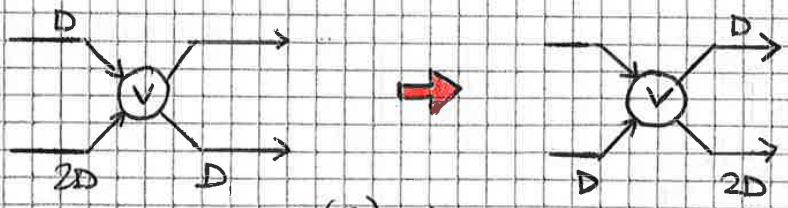
$th = \frac{1}{T_{cp}} = \frac{1}{2}$

IT'S THE BEST WE CAN DO WITH THIS PROCESS

GENERAL DEFINITION OF RETIMING:

FOR NODE $v \in DFG$ RETIMING $r(v)$ IS DEFINED AS THE NUMBER OF REGISTERS WE MOVE FROM INPUTS OF v TO ITS OUTPUTS.

ex4.



$r(v) = 1$ MEANS THAT WE WANT TO ELIMINATE ONE REGISTER AT THE INCOMING ARC AND ADD IT AT EACH OUTCOMING ARC

NOTE THAT ALL THE EDGES HAVE TO BE CONSIDERED

IF ! $r(v) < 0 \Rightarrow$ REGS ARE MOVED FROM OUTPUTS TO INPUTS

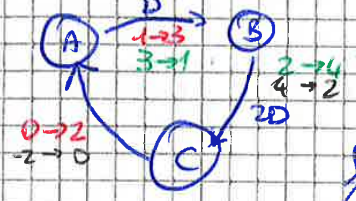
BY REPLACING EACH VALUE OF $W(e_{ij})$ IN $W_r(e_{ij})$:

$$\begin{cases} 1+r(2)-r(1) \geq 0 \\ 1+r(1)-r(3) \geq 1 \\ 2+r(1)-r(4) \geq 1 \\ r(3)-r(2) \geq 1 \\ r(4)-r(2) \geq 1 \end{cases}$$

BY SOLVING THIS SYSTEM OF EQUATIONS ⇓ WE CAN IMPROVE THE THROUGHPUT

ONE POSSIBLE TRICK DERIVES BY THE FACT THAT EACH EQUATION COMES FROM THE DEFINITION OF RETIMING ⇓ SO YOU CAN ADD ANY VALUE WITHOUT MODIFYING ANY RESULT!

example



Assume you have $r(A)+2$, $r(B)+2$, $r(C)+2$ ⇒ NOTHING CHANGES! YOU CAN ADD A COSTANT TO EACH VALUE ⇓ NO MODIFICATION AT ALL!

for example: $r(A)=2 \rightarrow$, $r(B)=2 \rightarrow$, $r(C)=2 \rightarrow$

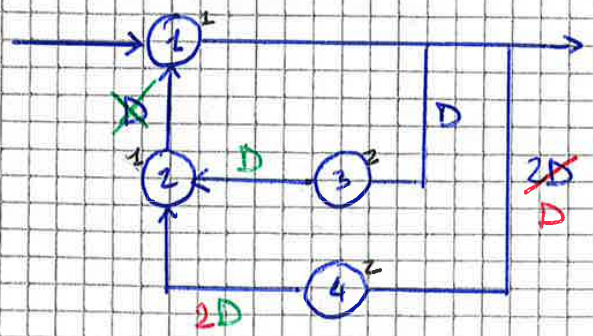
ALL THIS SHOWS THAT ALL r ARE DEFINED IN A REDUNDANT WAY SO IT'S POSSIBLE TO ADD COSTANTS!

IN OUR CASE WE CAN APPLY ⇒ $r(1)=0$ ⇓ BECAUSE IT APPEARS MOST OF TIMES

$$\begin{cases} r(2)+1 \geq 0 \\ 1-r(3) \geq 1 \\ 2-r(4) \geq 1 \\ r(3)-r(2) \geq 1 \\ r(4)-r(2) \geq 1 \end{cases} \Rightarrow \text{2 POSSIBLE SOLUTIONS}$$

1) $r(1)=0 \quad r(2)=-1 \quad r(3)=0 \quad r(4)=0$

2) $r(1)=0 \quad r(2)=-1 \quad r(3)=0 \quad r(4)=1$



NODES	r'	r''
1	0	0
2	-1	-1
3	0	0
4	0	1

NOTHING CHANGES BETWEEN THE 2 SOLUTIONS

WE OBTAIN: $T_{cp}=2 \Rightarrow th=\frac{1}{2}$ SAME RESULT FOR BOTH SOLUTIONS

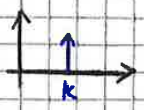
DIGITAL FILTERS

03/10/17

IT'S A SYSTEM RECEIVING A NUMERICAL STREAM OF SAMPLES AND ABLE TO GENERATE A FILTERED OUTPUT STREAM

IF WE APPLY AS INPUT AN IMPULSE

(1) $\delta[n-k]$ ⟶ $h_k[n]$ ⇒ ITS RESPONSE IS DIFFERENT SHAPES DEPEND ON THE SYSTEM WE'RE IMPLEMENTING



WITH A REAL SIGNAL → THE INPUT IS THE OVERLAPPING OF PULSES OF GENERAL AMPLITUDE

IN: $x[n] = \sum_k x[k]\,\delta[n-k]$ ⇒ THE FILTER RECEIVES $x[n]$ SAMPLES

THE OUTPUT OF A SINGLE PULSE

$$h_k[n] = F(\delta[n-k])$$

WE ASSUME < LINEARITY, TIME INVARIANCE

$$y[n] = F(x[n]) = F\left(\sum_k x[k]\,\delta[n-k]\right) \overset{\text{LINEARITY}}{=} \sum_k x[k]\,F(\delta[n-k]) \overset{(1)}{=} \sum_k x[k]\,h_k[n] =$$

$$\overset{\text{TIME INVARIANCE}}{=} \sum_k x[k]\,h[n-k] = x[n] * h[n]$$

(= CONVOLUTIONAL PRODUCT)

THE DIGITAL FILTER → HAS THE FOLLOWING PROPERTIES

LINEARITY

↓

X CAN BE DIVIDED INTO DIFFERENT CONTRIBUTIONS AND WE CAN APPLY THE FILTER TO EACH CONTRIBUTION

TIME INVARIANCE

↓

OUR RESPONSE IS ALMOST INDEPENDENT FROM THE TIME WE SEND IT EXCEPT FROM THE SHIFT IN TIME

↓

EVERY SAMPLE HAS THE SAME CHARACTERISTICS EXCEPT FROM A SHIFT IN TIME

$$h_k[n] = h[n-k]$$

CAUSALITY

IF WE APPLY AN INPUT AT TIME K WE EXPECT THAT THE OUTPUT DOESN'T COME BEFORE TIME K

$x[n] = 0 \quad n<0$

⇓

$y[n] = 0 \quad n<0$

STABILITY

⇓

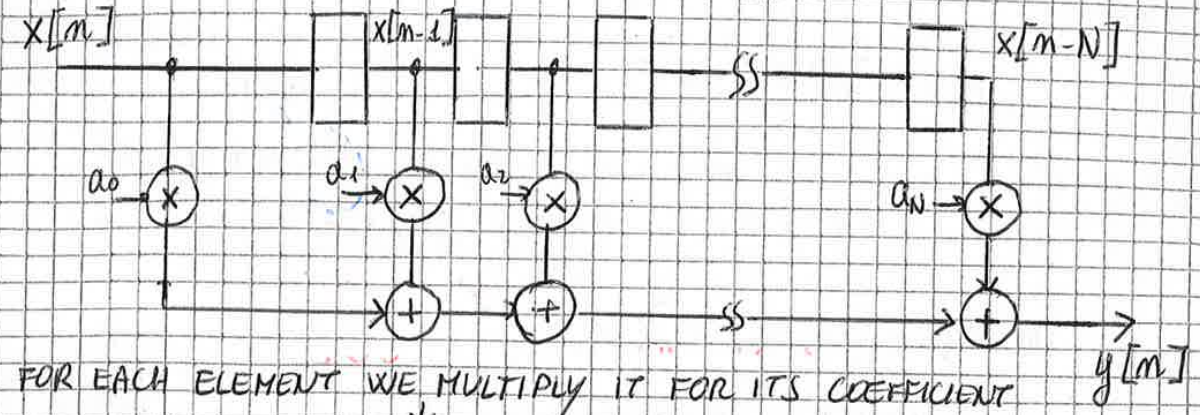
BIBO STABILITY

↓

THE SUM OF THE PEAKS HAS TO BE LIMITED

$$\sum_i |h[k]| < \infty$$

FIR FILTER → DIRECT FORM = DIRECT TRANSLATION OF A FIR FILTER INTO A COMPUTATIONAL STRUCTURE



FOR EACH ELEMENT WE MULTIPLY IT FOR ITS COEFFICIENT

WE CALCULATE THE PRODUCT OF A DELAYED VERSION OF THE INPUT AND ITS COEFFICIENT AND THEN WE SUM EACH CONTRIBUTION

AN OTHER POSSIBILITY → THANKS TO TRANSPOSITION THEOREM

IT'S A SERIES OF RULES

1) $\forall w \in EDFG$ TAKE THE OPPOSITE

   WE CHANGE THE DIRECTION FOR EVERY ARC IN THE DFG

2) EXCHANGE I/O PORT (IN <=> OUT)

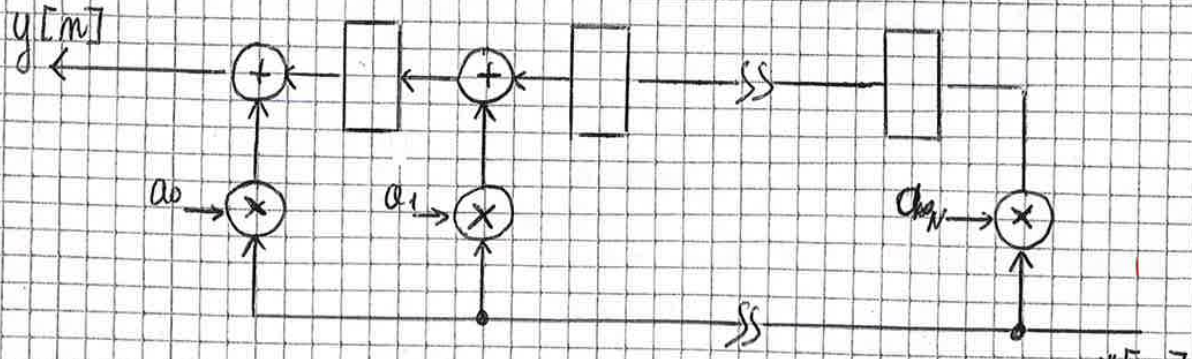
3) KEEP THE SAME GAIN FOR EVERY ARC

   = EITHER AMOUNT OF DELAY (REGS) OR VALUES OF MULTIPLIER (SO COEFFICIENTS)

4) ANY ⊕ ADDITION NODE BECOMES A FORK NODE (AND VICEVERSA)

FORK NODE = NODE WHICH DUPLICATES THE INPUT SAMPLE

LET'S OBTAIN THE TRANSPOSED FORM



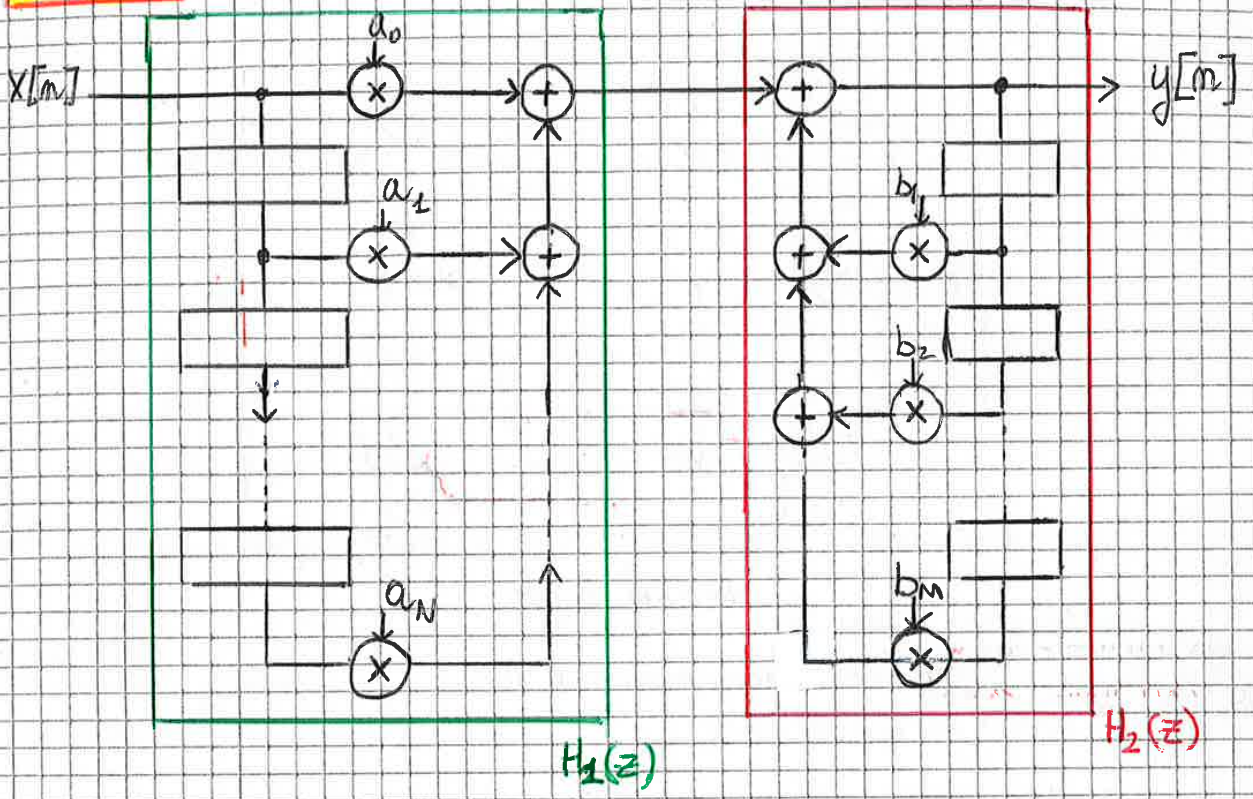
THE NEW STRUCTURE IS COMPLETELY DIFFERENT FROM THE ORIGINAL ONE.
WE TAKE A SINGLE SAMPLE AND WE APPLY IT IN PARALLEL TO EACH MULTIPLIER. THAN WE PERFORM ALL THE ADDITIONS THAT ARE PREDISTORTED.

NOW THE CRITICAL PATH IS SHORTER! IT'S IMPROVED!

IIR FILTER: $y[n]$ IS MADE OF 2 PARTS

$$y[n] = \sum_{i=0}^{N} a_i\, x[n-i] + \sum_{j=1}^{M} b_j\, y[n-j]$$

DIRECT FORM - II



WE HAVE 2 FILTERS → CONNECTED IN A CHAIN

$$H(z) = H_1(z) \cdot H_2(z)$$

[LIKE FOR ANY CASCADE OF FILTERS]

WE HAVE THE POSSIBILITY TO CHANGE THE ORDER

↓

BY DOING THAT WE SHOULD BE SURE THAT IT WORKS

↓

IF WE EXCHANGE → WE HAVE THAT REGISTERS ARE DELAYING THE SAME SIGNAL!

BY HAVING H_2 AND THEN H_1 S.T. WE HAVE HALF OF REGISTERS!

WHEN YOU DESIGN DIGITAL FILTER A PROBLEM IS TO DEFINE THE FILTER WITH A HUGE NUMBER OF BITS

↓

THIS IS TOO MUCH IN HARDWARE → SO YOU APPROXIMATE YOUR COEFFICIENTS FOR A MINIMUM NUMBER OF BITS

↓

THE BEHAVIOUR IN FREQUENCY DOMAIN CAN BE CHARACTERIZED BY MEANS OF ZEROS AND POLES WHICH ARE DEPENDING ON THE ZEROS OF THEIR COEFFICIENTS

$$H(z) = \frac{N(z)}{D(z)} \rightarrow \text{ZEROS}, \rightarrow \text{POLES}$$

} KEY TO DETERMINE THE BEHAVIOUR IN FREQUENCY DOMAIN

↓

BY MODIFYING THE COEFFICIENTS YOU MODIFY THE ZEROS AND POLES

YOU CAN TRY TO TUNE THE VALUE OF ONE COEFFICIENT THAT'S OK FOR THE 1ST ONE BUT IF YOU MODIFY THE 2ND COEFFICIENT YOU MODIFY EVERYTHING AGAIN!

IT'S VERY DIFFICULT TO COMPUTE THIS WORK!

IT'S POSSIBLE TO DERIVE THE SENSITIVITY OF ZEROS / POLES

IF THEY TEND TO BE VERY CLOSE TO EACH OTHERS

↓

SENSITIVITY IS VERY HIGH

IF POLES AND ZEROS ARE SEPARATED FROM EACH OTHERS AND DISTRIBUTED IN A LARGER DOMAIN

↓

SENSITIVITY IS MUCH LOWER!

THIS DETERMINES DIFFERENT CHOICES IN FIR/IIR FILTERS

- FIR FILTER → HAS ONLY A POLE Z=0
  ZEROS TEND NOT TO BE GROUPED BUT TO BE DISTRIBUTED
  ↓
  IT'S SIMPLER TO MODIFY COEFFICIENTS SINCE THE SENSITIVITY IS VERY LOW
  THAT'S WHY WE TREND TO USE THE DIRECT FORM
- IIR FILTER → ZEROS / POLES ARE GROUPED
  BEING VERY CLOSE TO EACH OTHERS SENSITIVITY IS EXTREMELY HIGH
  ↓
  SO IT'S DIFFICULT TO OPTIMIZE COEFFICIENTS
  ↓
  CASCADE FORM IS PREFERRED
  ↓
  THAT IS A DEGREE-II DIRECT IMPLEMENTATION
  WE MODIFY INDIPENDENTLY EVERY SINGLE STAGE

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 23 di 190

UNFOLDING (= LOOP UNROLLING)

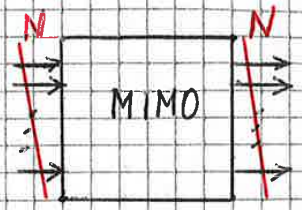
13/10/17

IT'S A PROCEDURE WE CAN EXPLOIT TO MOVE FROM A PURE SEQUENTIAL COMPUTATION INTO A PARALLEL / SEMIPARALLEL COMPUTATION

WE MOVE FROM A SYSTEM

SINGLE INPUT SINGLE OUTPUT

IF WE'RE ABLE TO IMPLEMENT IT INTO A PARALLEL FORM



MULTIPLE INPUT MULTIPLE OUTPUT } INPUTS ARE DIVIDED INTO SEVERAL STREAMS AND THEY PROVIDE THE SAME INFO AS x[n]

$T_{cp} = T_{clk}$

$th = \frac{1}{T_{clk}}$

THE THROUGHPUT CHANGES

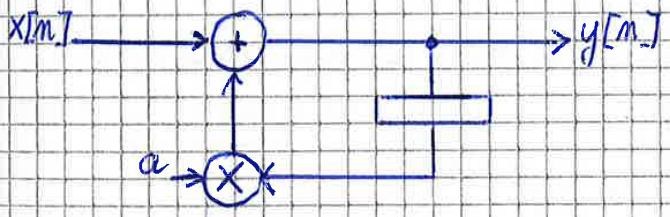
$T_{cp} = T_{clk}$

$th = \frac{N}{T_{clk}}$

WE OBTAIN AN HIGHER THROUGHPUT BY SUMMING IN PARALLEL OUR PROCESS

EXAMPLE

ex 1:



$y[n] = x[n] + a\,y[n-1]$

WE WANT TO APPLY UNFOLDING

FOR NOW WE SKIP THE 1ST STEP THAT CONSISTS IN EVALUATING T_∞

I METHOD → WE WRITE OUR EQUATION IN A PARALLEL FORM

$x[n]$ → $x[2k]$ EVEN ELEMENTS

$x[2k+1]$ ODD ELEMENTS

BY DOING THAT WE DIVIDE OUR EQUATION INTO A COUPLE OF EQUATIONS:

ODD POSITIONS

$$\begin{cases} y[2k] = x[2k] + a\,y[2k-1] = x[2k] + a\,y[2(k-1)+1] \\ y[2k+1] = x[2k+1] + a\,y[2k+1-1] = x[2k+1] + a\,y[2k] \end{cases}$$

EVEN POSITIONS

⇓

EVERY ELEMENT IN OUR EQUATION IS WRITTEN IN A FORM WHERE INDEX IS

EITHER A EVEN NUMBER

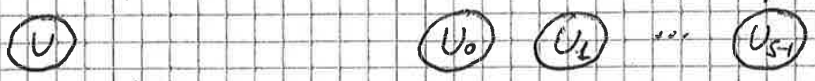
OR AN ODD NUMBER

II METHOD

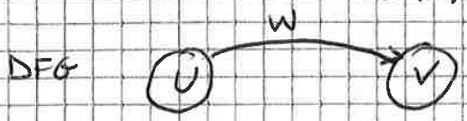
STARTING FROM DFG —(WE CREATE A J-DEGREE)→ DFG' BY EMPLOYING A J-DEGREE PARALLELISM

IT FOLLOWS 2 RULES:

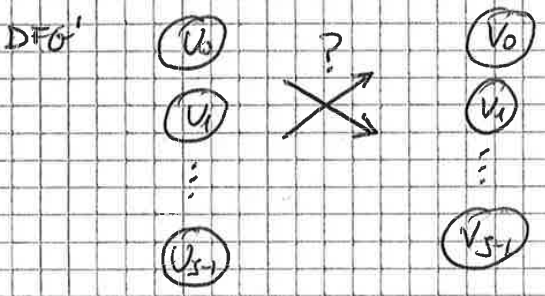
1) ∀NODE ∈ DFG, ALLOCATE J NODES OF THE SAME TYPE IN DFG'



2) ∀EDGE ∈ DFG, WITH W REGS, ALLOCATE J EDGES AND DECIDE FOR EACH OF THEM → THE FINAL DESTINATION, → THE NUMBER OF REGISTERS PLACED — USING A FORMULA



EDGE LINKING U → V



i = INDEX OF U TYPES NODES

LET'S CALL:
- J = DEGREE OF PARALLELISM
- W = NUMBER OF REGS PLACED IN THE ORIGINAL PATH

FOR EACH COUPLE WE COMPUTE $\frac{i+W}{J}$ → TO OBTAIN IN THE NEW PATH

- Q = QUOTIENT → $\left\lfloor \frac{(i+W)}{J} \right\rfloor$ NUMBER OF REGS
- R = REMINDER → $(i+W)\ \%\ J$ → FINAL DESTINATION

BY APPLYING THIS RULE WE GENERATE THE PARALLEL GRAPH

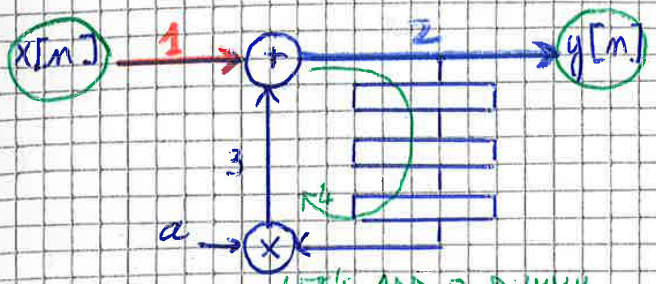
NOTE: IN PRINCIPLE YOU NEED TO APPLY THIS RULE TO EVERY SINGLE EDGE

BUT IF I HAVE AN ARC WITHOUT REGS → YOU CAN SIMPLY REPLICATE THE SELECTED ARCS WITH SAME SOURCE AND DESTINATION

SO DIVIDE YOUR GRAPH IN 2 PARTS → WITH ARCS WITH NO REGISTERS → DFG': SAME AS DFG (SAME SOURCE, SAME DESTINATION)

→ WITH ARCS WITH REGISTERS → CONSIDER EQUATION $\frac{i+W}{J}$

EX2 LET'S APPLY THE LOOP TO AN OTHER FILTER WHERE THERE'S A ROOM FOR IMPROVEMENT



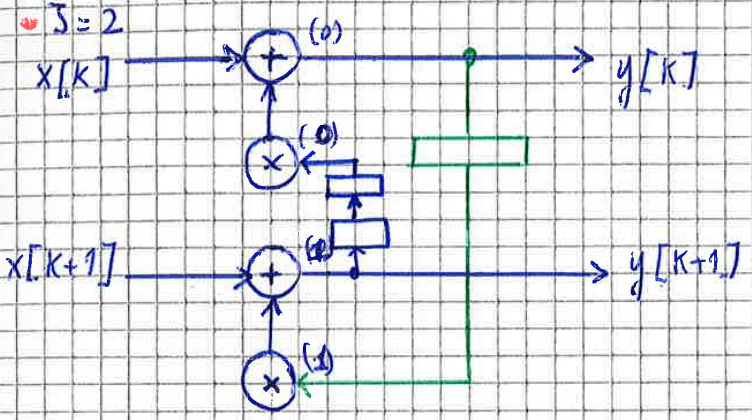
$T_{cp} = T_{clk}$

$T_{cp} = T_m + T_a$

$I_b = T_\infty = \frac{T_m + T_a}{3}$

$T_\infty < T_{clk}$ → WE CAN IMPROVE THE BEHAVIOR OF OUR FILTER OF 3 TIMES

LET'S ADD 2 DUMMY NODES FOR INPUT AND OUTPUT



CONCERNING ARCS 1, 2, 3 THERE ARE NO REGS
↓
SO FOR EACH OF THEM WE DON'T NEED TO CHANGE SOURCE AND DESTINATION

CONCERNIG EDGE 4 →

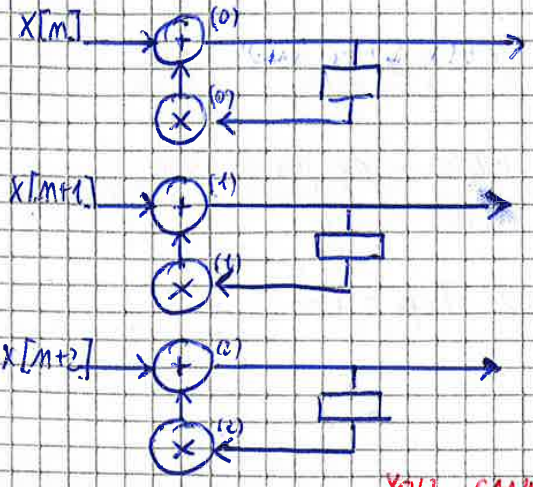
(0) $\frac{i+W}{3} = \frac{0+3}{2} = \frac{3}{2}$ → $Q = 1$ 1 REG. $R = 1$ DESTINATION } PATH —

(1) $\frac{1+3}{2} = \frac{4}{2}$ → $Q = 2$ 2 REGS $R = 0$ DESTINATION } PATH —

NEW DFG → SAME NUMBER OF REGISTERS

$T_{cp} = T_a + T_m$ → $th = \frac{2}{T_a + T_m}$ → PERFORMANCE IS ACHIEVED BY A FACTOR 2 ↓ IMPROVING OF PERFORMANCES!

J = 3 → WILL ALLOW YOU TO ACHIEVE THE BEST PERFORMANCES!



$i = 0$ $\frac{i+W}{3} = \frac{0+3}{3} = \frac{3}{3}$ → $Q = 1$, $R = 0$

$i = 1$ $\frac{1+3}{3} = \frac{4}{3}$ → $Q = 1$, $R = 1$

$i = 2$ $\frac{2+3}{3} = \frac{5}{3}$ → $Q = 1$, $R = 2$

IT'S A SPECIAL CASE WITH 3 DFGS SEPARATED BY EACH OTHERS AND THERE'S A REG FOR EACH PATH

$T_{cp} = T_m + T_a$ → $th = \frac{3}{T_m + T_a} = \frac{1}{T_\infty}$!

YOU CANNOT IMPROVE THE DFG ANY MORE!

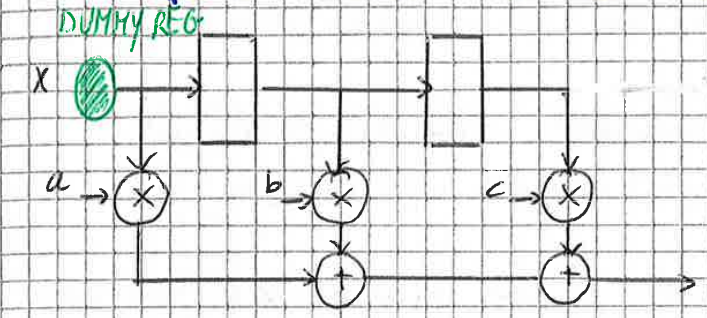
IF YOU APPLY J=4 → YOU HAVE LOOPS WHICH HAVE A CRITICAL PATH THAT IS LONGER

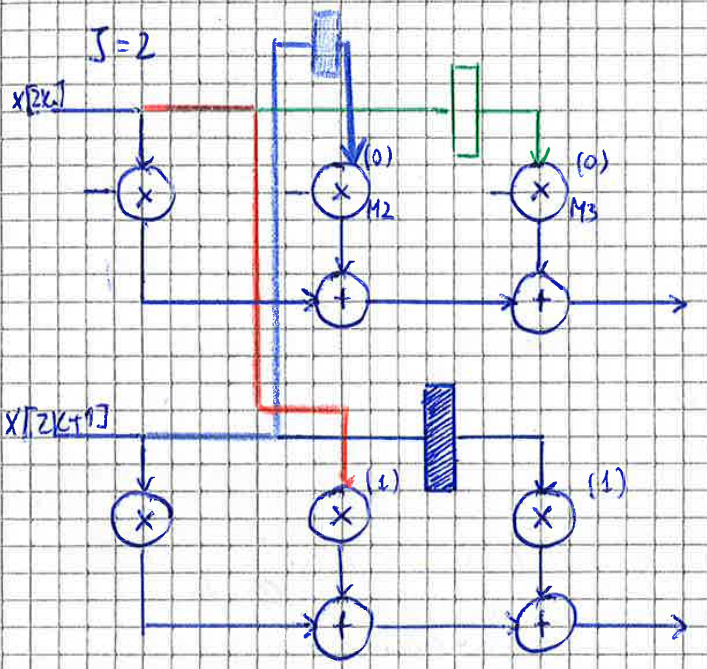
NB UNFOLDING → CAN ALSO BE APPLIED TO A DFG WHICH HAS NO LOOPS!

⇓

THE METHOD TO MOVE FROM A SERIAL TO A PARALLEL IMPLEMENTATION CAN BE ALSO APPLIED TO A FIR FILTER (TO A FEED FORWARD DFG)

$$y[n] = a\,x[n] + b\,x[n-1] + c\,x[n-2]$$





$i=0$, M2: $\frac{i+w}{J} = \frac{0+1}{2} = \frac{1}{2}$ → $Q=0$, $R=1$ →

$i=1$, M2: $\frac{1+1}{2} = \frac{2}{2}$ → $Q=1$, $R=0$ →

$i=0$, M3: $\frac{0+2}{2} = \frac{2}{2}$ → $Q=1$, $R=0$ REG → DESTINATION

$i=1$, M3: $\frac{1+2}{2} = \frac{3}{2}$ → $Q=1$, $R=1$ →

THERE ARE 3 REGISTERS INSTEAD OF 2!

IN REALITY → IF YOU OBSERVE CAREFULLY BLUE REGISTERS ARE DELAYING $X[2k+1]$ OF THE SAME PERIOD

⇓

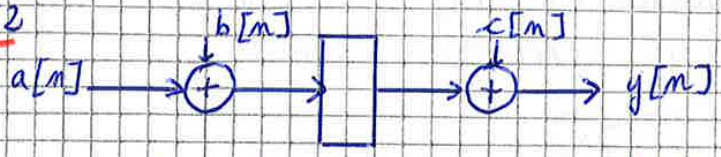
WE CAN USE A SINGLE REGISTER INSTEAD OF 2

⇓

$N_{regs} = 2$

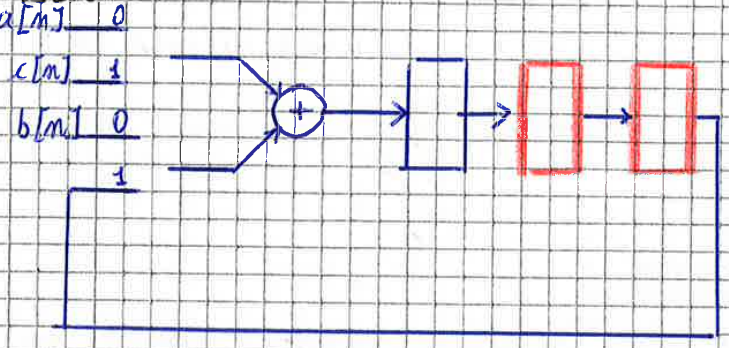
THIS CAN BE MORE DIFFICULT IN A MORE COMPLEX CIRCUIT

ex2



$$y[n] = a[n-1] + b[n-1] + c[n]$$

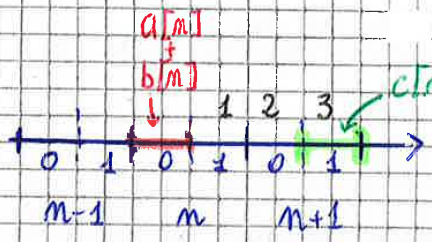
IF WE WANT TO PERFORM THE SAME OPERATION WE CAN TRY TO PROCEED AS BEFORE:



IF WE DON'T CHANGE ANYTHING WE'RE GOING TO HAVE A WRONG RESULT

THE SAMPLING PERIOD IS STILL THE SAME ⇓ IT'S STILL n

IN ORDER TO RESPECT TIMING WE NEEDS TO ADD 2 REGISTERS ⇓ TO HAVE A CORRECT SYNCHRONIZATION OF DATA



IF WE START @ SAMPLING PERIOD $n \to 0$

WE WANT TO COMBINE THE RESULT OF THE ADDER WITH

$$y[n] = c[n+1] + (a[n] + b[n])$$

⇓

SO WE NEED 3 CLOCK CYCLES TO HAVE A PROPER SYNCHRONIZATION

! HINT: WITH UNFOLDING YOU CAN HAVE ↑T_{CP} THAN THE ORIGINAL ONE BUT YOU NEED TO DIVIDE IT BY THE DEGREE OF UNFOLDING

$$th = \frac{N}{T_{CP}}$$

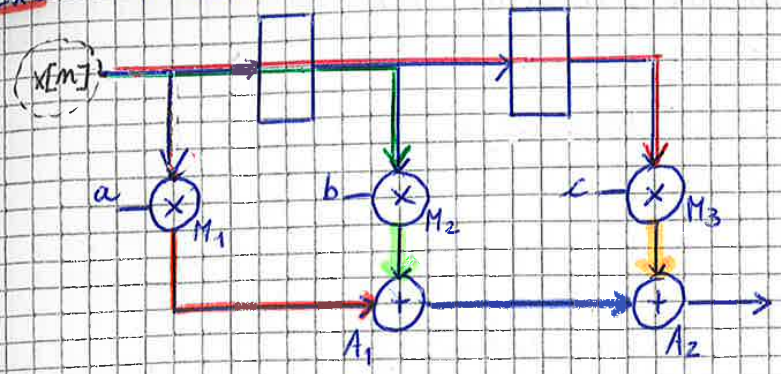
WITH N = DEGREE OF UNFOLDING

WITH FOLDING YOU CAN HAVE ↓T_{CP} THAN THE ORIGINAL ONE BUT YOU NEED TO MULTIPLY IT BY THE DEGREE OF FOLDING

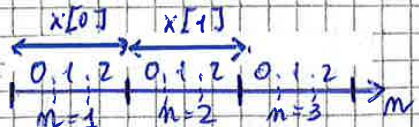
$$th = \frac{1}{N} \cdot \frac{1}{T_{CP}}$$

N = DEGREE OF FOLDING

ex3: FIR Filter



N=3 IMPLIES
- 3 MULTIPLICATIONS PERFORMED BY 1 MULIPLIER
- 2 ADDITIONS THAT USES 2 CLOCK CYCLES OF 3



WE NEED TO DECIDE THE SCHEDULING TIME

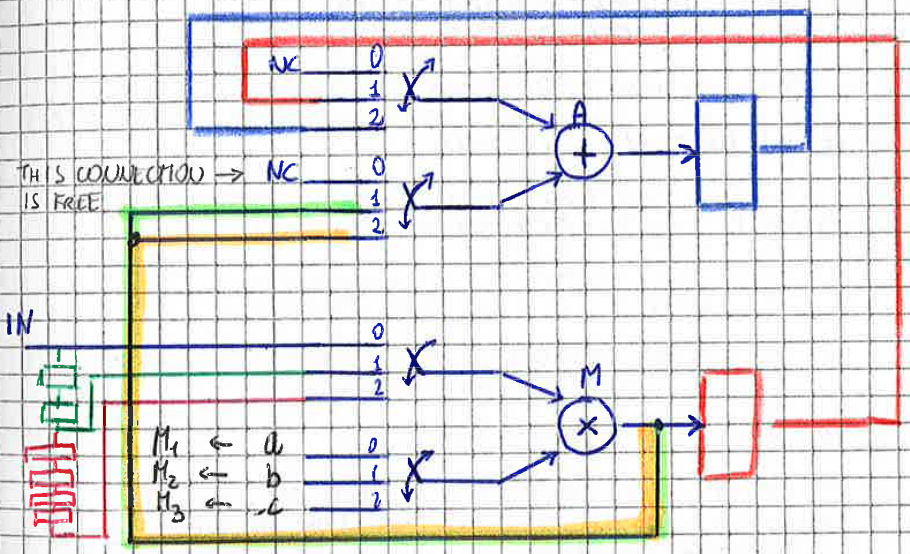
SCHEDULING @ TIME:

$M_1 \rightarrow 0$ $M_2 \rightarrow 1$ $M_3 \rightarrow 2$

$A_1 \rightarrow 1$ $A_2 \rightarrow 2$

x[n] IS INPUT NODE
IT IS STABLE FOR 3 CLOCK CYCLES SO AVAILABLE!
SCHEDULING TIME [0÷2]
IT CAN BE SELECTED WITHOUT CONFLICTS

OUR NEW ARCHITECTURE HAS 1 ADDER AND 1 MULTIPLIER



FOR EACH EDGE WE NEED TO CONSIDER THE FORMULA $W_F = Nw + U - u$

$M_1 \rightarrow A_1$ $W_F = 3 \cdot 0 + 1 - 0 = 1$
- 0: NO REGS IN THE ORIGINAL PATH
- 1: SCHEDULING TIME OF A_1
- 0: SCHEDULING TIME OF M_1

→ WE NEED 1 REGISTER IN THE PATH WHOSE DESTINATION IS LINKED TO THE SCHEDULING TIME OF U

$M_2 \rightarrow A_1$ $W_F = 3 \cdot 0 + 1 - 1 = 0$ → ACTIVATION ON SCHEDULING TIME 1, WITHOUT REGS ON THE PATH

IN THIS WAY WE'LL RECEIVE @ SAME TIME
- M_1: FROM PREVIOUS CLOCK CYCLE
- M_2: FROM CURRENT CLOCK CYCLE

$M_3 \rightarrow A_2$ $W_F = 3 \cdot 0 + 2 - 2 = 0$ ACTIVATED AT SCHEDULING TIME 2

$A_1 \rightarrow A_2$ $W_F = 3 \cdot 0 + 2 - 1 = 1$ ACTIVATED AT SCHEDULING TIME 2

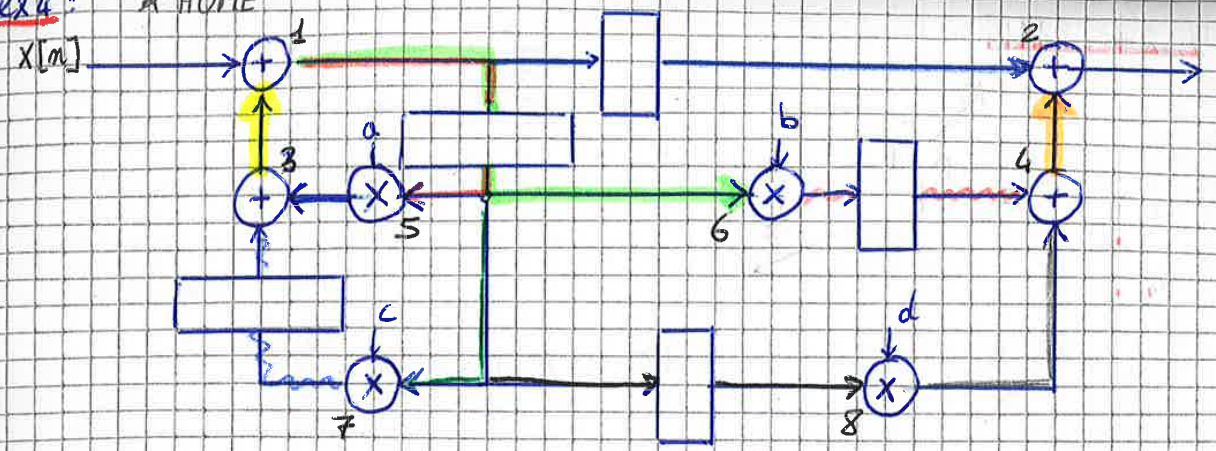
$IN \rightarrow M1$ $W_F = 3 \cdot 0 + 0 - 0 = 0$ ← MUST BE SELECTED AS 0 → WE CANNOT HAVE CONFLICTS!

$IN \rightarrow M2$ $W_F = 3 \cdot 1 + 1 - 2 = 2$

$IN \rightarrow M3$ $W_F = 3 \cdot 2 + 2 - 2 = 6$

SELECTED IN ORDER TO HAVE THE LOWER NUMBER OF REGS POSSIBLE

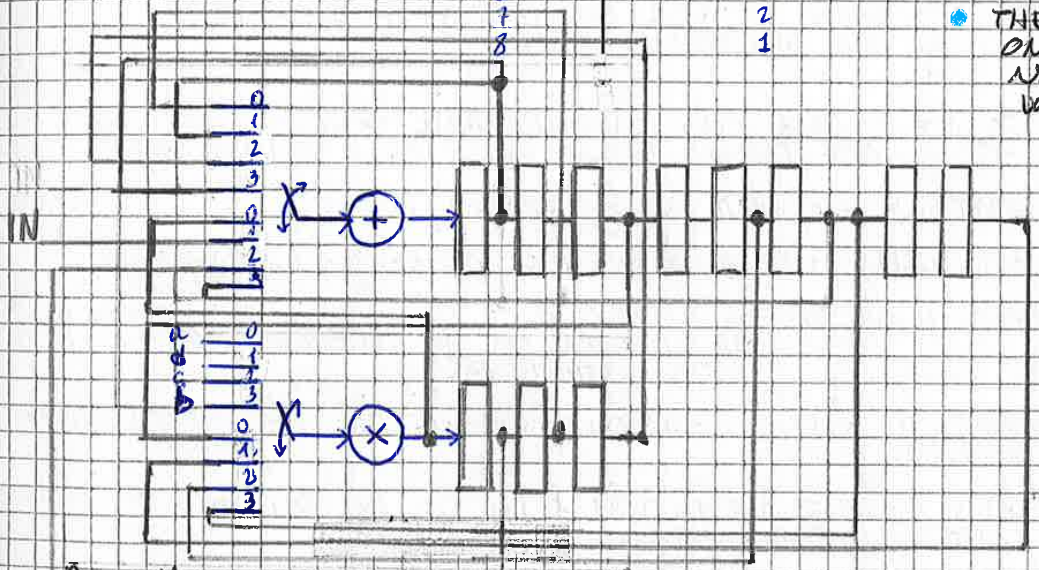
ex 4: ★ HOME



LET'S SET N = 4

OPERATION	SCHEDULING TIME
1	1
2	3
3	0
4	2
5	0
6	3
7	2
8	1

IF:
- THERE ARE NO REGISTERS IT'S CONVENIENT TO USE INCREASING INDEX WITH RESPECT TO DATA DEP.
- THERE IS AT LEAST ONE REGISTER NO PROBLEM WITH DATA DEPENDENCY



$w_F = N \cdot w + v - u$

$A_1 \to M_5$	$w_F = 4 \cdot 1 + 0 - 1 = 3$	AT SCHEDULING TIME 0
$A_1 \to A_2$	$w_F = 4 \cdot 1 + 3 - 1 = 6$	AT SCHEDULING TIME 3
$A_1 \to M_6$	$w_F = 4 \cdot 1 + 3 - 1 = 6$	AT SCHEDULING TIME 3
$A_1 \to M_7$	$w_F = 4 \cdot 1 + 2 - 1 = 5$	AT SCHEDULING TIME 2
$A_1 \to M_8$	$w_F = 4 \cdot 2 + 1 - 1 = 8$	AT SCHEDULING TIME 1
$A_3 \to A_1$	$w_F = 4 \cdot 0 + 1 - 0 = 1$	AT SCHEDULING TIME 1
$A_4 \to A_2$	$w_F = 4 \cdot 0 + 3 - 2 = 1$	AT SCHEDULING TIME 3
$M_5 \to A_3$	$w_F = 4 \cdot 0 + 0 - 0 = 0$	AT SCHEDULING TIME 0
$M_6 \to A_4$	$w_F = 4 \cdot 1 + 2 - 3 = 3$	AT SCHEDULING TIME 2
$M_7 \to A_3$	$w_F = 4 \cdot 1 + 0 - 2 = 2$	AT SCHEDULING TIME 0
$M_8 \to A_4$	$w_F = 4 \cdot 0 + 2 - 1 = 1$	AT SCHEDULING TIME 2
$IN \to A_1$	$w_F = 4 \cdot 0 + 1 - 1 = 0$	AT SCHEDULING TIME 1

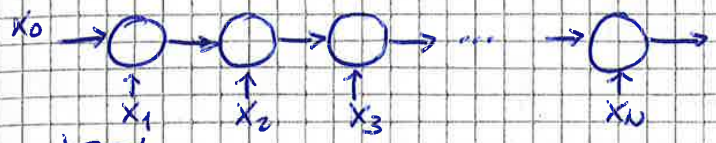
THESE ARE → UNIVERSAL TECHNIQUES

YOU CAN APPLY THEM TO ANY ALGORITHM WITHOUT ANY ASSUMPTION ON SPECIFIC PROPERTY (EXCEPT FROM PIPELINING)

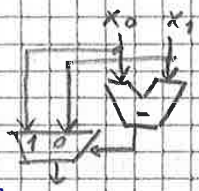
THERE ARE MANY OTHER TECHNIQUES → THAT REQUIRE SPECIFIC ASSUMPTION

eg: search of minimum among x_i samples

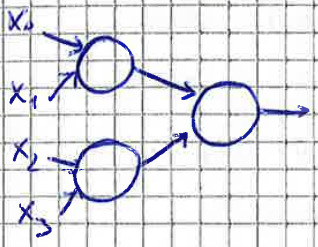
$m = \min_i \{x_i\}$



select the minimum between 2 inputs → through subtractor + mux



An other possible solution → concern a BINARY TREE → to obtain better implementation in terms of performances



CANNOT BE USED AS UNIVERSAL TECHNIQUE

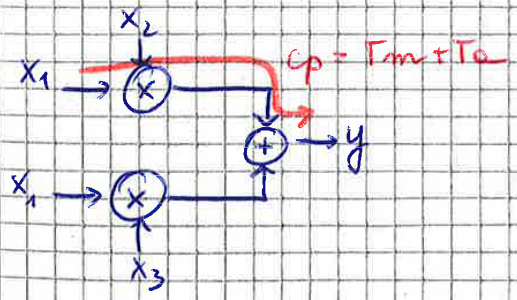
⇓

time increases logarithmic with respect to # nodes and no more linearly

- $y = x_1 x_2 + x_3 x_1$

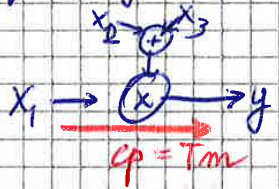
$x_1 \xrightarrow[cp]{} y$

Let's imagine the path from x_1 to y is the critical path



EXPLOITING THE DISTRIBUTIVE PROPERTY

$y = x_1 (x_2 + x_3)$



$T_{op_{max}} = \max\{T_m, T_a\} = T_m$ → > THROUGHPUT AT THE COST OF ADDITIONAL OPERATIONS

LET'S IMAGINE I WANT TO ACHIEVE A FASTER IMPLEMENTATION OF THE FILTER

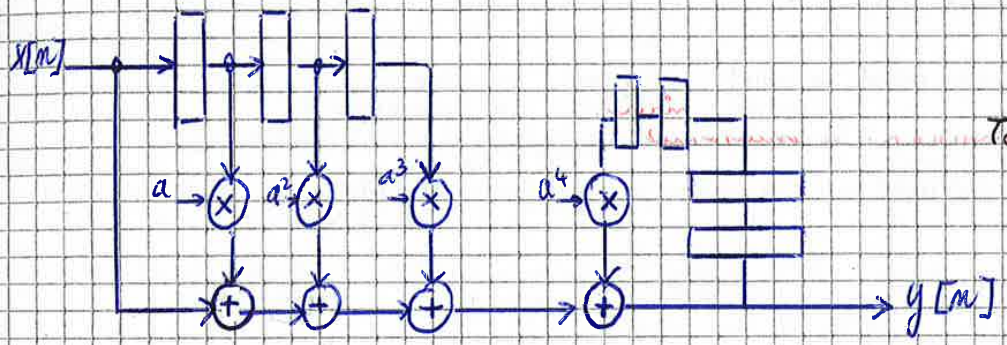
LOOK-AHEAD AGAIN

$$y[n] = x[n] + a \cdot x[n-1] + a^2 y[n-2]$$

$$y[n-2] = x[n-2] + a y[n-3] = x[n-2] + a \cdot x[n-3] + a^2 y[n-4]$$

IMPROVE THROUGHPUT BY A FACTOR

$$y[n] = x[n] + a \cdot x[n-1] + a^2 \cdot x[n-2] + a^3 \cdot x[n-3] + a^4 y[n-4]$$



$T_\infty = \frac{T_m + T_a}{4}$

IF YOU APPLY 1 LOOK-AHEAD => BETTER THROUGHPUT

WE CAN EXPLAIN IT IN Z-TRANSFORMED DOMAIN:

$$y[n] = x[n] + a \cdot y[n-1]$$

$$Y(z) = X(z) + a\, Y(z)\, z^{-1} \rightarrow H(z) = \frac{Y(z)}{X(z)} = \frac{1}{1 - a z^{-1}}$$

FROM THE ALGEBRIC POINT OF VIEW:

$$\frac{1}{(1 - a z^{-1})} \cdot \frac{1 + a z^{-1}}{1 + a z^{-1}} = \frac{1 + a z^{-1}}{1 - a^2 z^{-2}}$$

NEW DENOMINATOR IS OF ORDER 2 THAT IS A BETTER FROM THE IMPLEMENTATION POINT OF VIEW

LOOK-AHEAD → CAN BE APPLIED TO LOOPS → WHEN YOU'RE LIMITED BY $T_\infty = T_m$

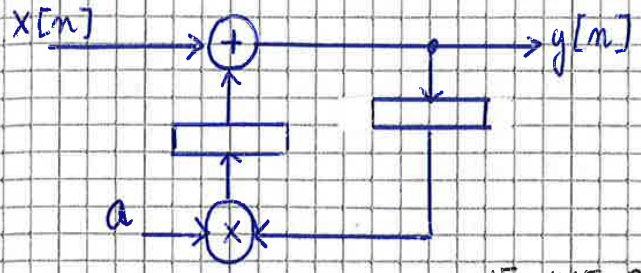
IF YOU'RE WORKING WITH COSTANT COEFFICIENT a IN A NEW EQUATION YOU HAVE a^2 → IT COSTS ONLY ONE ADDER IF a IS COSTANT

(IF a ≠ COSTANT IT'S NO MORE TRUE YOU CAN FORECAST a^2)

LET'S CONSIDER THE ORIGINAL FILTER

$$y[n] = x[n] + a\,y[n-1]$$

WE MODIFY THE ORIGINAL DFG BY INSERTING A NEW REGISTER HERE ✗ ↓ ABLE TO SPLIT THE CRITICAL PATH



$$y[n] = x[n] + a\,y[n-2]$$

THE NEW EQUATION SEEMS TO IMPLEMENT A DIFFERENT FILTER ↓ IF WE REWRITE IT FOR < EVEN / ODD POSITIONS IT'S THE CORRECT ONE

EVEN
$$\begin{cases} y[2k] = x[2k] + a\,y[2k-2] \\ y'[n] = x'[n] + a\,y'[n-1] \end{cases}$$

ODD
$$\begin{cases} y[2k+1] = x[2k+1] + a\,y[2k+1-2] = x[2k+1] + a\,y[2(k-1)+1] \\ y''[n] = x''[n] + a\,y''[n-1] \end{cases}$$

⇒ WE RE OVERLAPPING 2 FRAMES → TO INCREASE THE CLOCK FREQUENCY WE PROCEED IN PARALLEL WITH A COUPLE OF FRAMES

✗ WE PAY IN
- LATENCY → LONGER DELAY IN THE PROCESS
- MEMORY → IT'S NECESSARY MORE MEMORY IN ORDER TO PROCESS ONE TIME K_1 AND THE OTHER K_2

7

CRITICAL PATH → CARRY-BIT IS PROPAGATED FROM THE FIRST TO THE LAST FA

THE PATH CROSSES ALL THE FULL ADDERS AND THEN THERE ARE 2 POSSIBILITIES

EITHER C_N OR S_{N-1}

$$t_{RCA} = (n-1)\,t_c + \max\{t_c, t_s\}$$

FOR $(n-1)$ CASES WE HAVE THE PROPAGATION OF THE CARRY

CONCERNING THE LAST FA WE NEED TO DISTINGUISH BETWEEN S AND C_{i+1}

USUALLY WE HAVE $t_s > t_c$ → IT'S MORE CONVENIENT TO MINIMIZE t_c BECAUSE IT CROSSES ALL THE ADDERS

RCA ⇄ COMPLEXITY / PERFORMANCE } INCREASES LINEARLY WITH n

WE WANT TO IMPROVE PERFORMANCES TO OBTAIN FASTER ADDERS

FOR ANY ADDER → IN ADDITION TO THE OUTPUT BITS WE HAVE A NUMBER OF FLAGS (USEFUL FOR PROCESSORS)

- OVERFLOW FLAG
- SIGN FLAG IS THE MSB OF THE RESULT
- ZERO FLAG IS = 1 IF ALL THE $S_i = 0$

IS PRESENT IN 2 SITUATIONS:

1) $a_{n-1} = b_{n-1} = 0$ & $s_{n-1} = 1$ → THERE'S AN ERROR BECAUSE THE SUM BETWEEN 2 POSITIVE NUMBERS MUST BE POSITIVE

$C_n = 0$ IT CANNOT BE $\neq 0$
$C_{n-1} = S_{n-1} = 1$ → WITH $a, b = 0$ ⇒ $C_{n-1} \oplus C_n = 1$

2) $a_{n-1} = b_{n-1} = 1$ & $S_{n-1} = 0$

$C_n = 1$ FOR SURE!
$C_{n-1} = S_{n-1} = 0$ WITH $a, b = 1$ ⇒ $C_{n-1} \oplus C_n = 1$

IN ALL OTHER CASES: $a_{n-1} \neq b_{n-1}$ → $C_n = C_{n-1}$ (assume $a_{n-1} = 1$ $b_{n-1} = 0$; $S_{n-1} = 1 \oplus 0 \oplus 0 = 1$; $S_{n-1} = C_{n-1} \Rightarrow$ so $C_n = C_{n-1}$)

COMPUTATION IS CORRECT! $C_n \oplus C_{n-1} = 0$

⇒

$$O = C_n \oplus C_{n-1}$$

$$S = S_{n-1} \oplus O$$

OVERFLOW FLAG

SIGN FLAG → TO GET A CORRECT SIGN RESULT EVEN WITH OVERFLOW

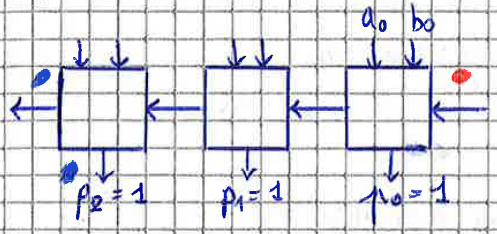
© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 45 di 190

c_{i+1} IS INDEPENDENT OF c_i IF $a_i = b_i$ → WE DON'T NEED TO WAIT FOR c_i

- $a_i = b_i = 0 \Rightarrow c_{i+1} = 0$ ALWAYS ⇒ KILL CONDITION
- $a_i = b_i = 1 \Rightarrow c_{i+1} = 1$ ALWAYS ⇒ GENERATE CONDITION (WE GENERATE A CARRY)

OTHERWISE $c_{i+1} = c_i$ IF $a_i \neq b_i$ ⇒ PROPAGATE CONDITION (CARRY IS PROPAGATED)

LET'S OBSERVE THE CRITICAL PATH AND FIND A METHOD TO SHORTEN THE PATH A LITTLE BIT



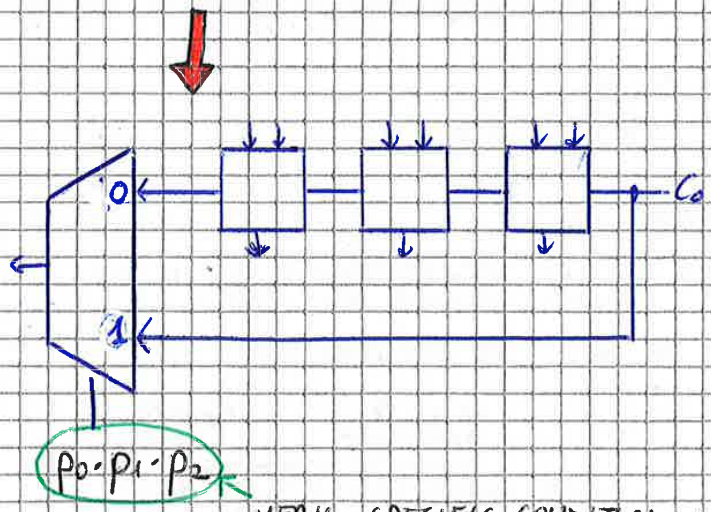
THE CARRY MUST PROPAGATE FROM • TO •

FOR EVERY STAGE THE PROPAGATE CONDITION HOLDS

↓

THIS IS THE ONLY CONDITION THE CARRY WILL PROPAGATE

$p_i = a_i \oplus b_i$



VERY SPECIFIC CONDITION → ALL THE OTHER CASES ARE DIFFERENT FROM THE CP

ES

QU

- $p_i = 0$ FOR $i = n-1$

$$t_{CSKA} = \left(\frac{n}{b} - 1\right) t_{MUX} + b\, t_{FA}$$ ← THERE'S NO POSSIBILITY TO ACTIVATE THE LAST BYPASS ⇓ CARRY PROPAGATES ALONG ALL FULL ADDERS

- IF $p_1 = 0$ → IN THE FULLADDER 1 OF THE FIRST BLOCK GENERATE/KILL

$$t_{CSKA} = \underbrace{\left(\frac{n}{b} - 2\right) t_{MUX}}_{\text{INTERMEDIATE BLOCKS}} + \underbrace{b\, t_{FA}}_{\text{LAST BLOCK}} + \underbrace{(b-1)\, t_{FA}}_{1^{st}\text{ BLOCK}}$$

HOW TO CHOOSE A GOOD VALUE FOR b?

$$t_{CSKA} = \left(\frac{n}{b} - 2\right) t_{MUX} + 2b\, t_{FA} - t_{FA} \quad (3)$$

$$\frac{\partial t_{CSKA}}{\partial b} = 0 \rightarrow 2 t_{FA} - \frac{n}{b^2} t_{MUX} = 0 \rightarrow \boxed{b = \sqrt{\frac{n\, t_{MUX}}{2\, t_{FA}}}}$$

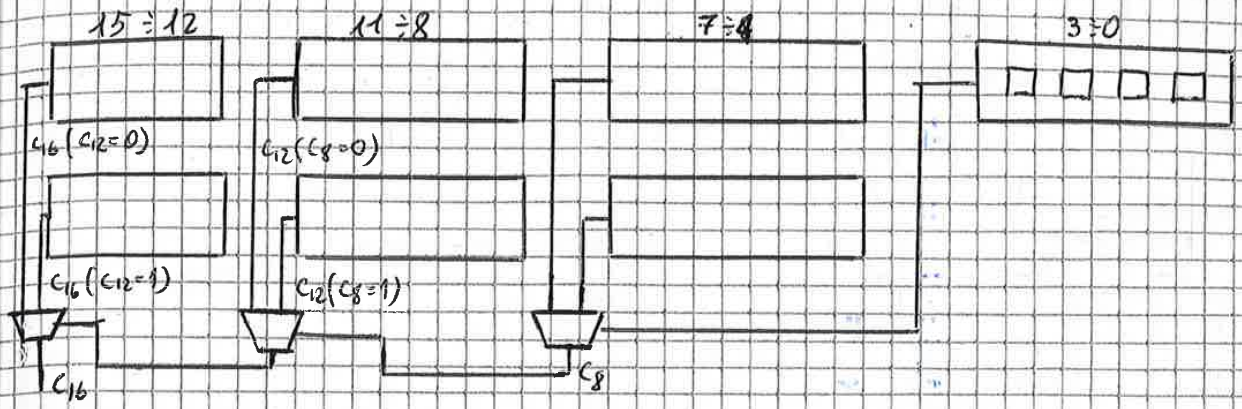
WE NEED TO APPROXIMATE IT TO THE CLOSEST INTEGER VALUE

REPLACING b IN (3)

$$t_{CSKA} = 2\sqrt{2n\, t_{MUX}\, t_{FA}} - t_{FA} - 2t_{MUX}$$ IT'S ALMOST FOR FREE! AND BETTER IN PERFORMANCE

- **LINEAR APPROACH**: LET'S SUPPOSE $n = 16$ $b = 4$

$k = \frac{n}{b} = 4$ # OF BLOCKS WE NEED TO ALLOCATE



$$t_{CSEA_{16}} = 4\,t_{FA} + 3\,t_{MUX}$$

$$t_{CSEA} = b\,t_{FA} + \left(\frac{n}{b} - 1\right) t_{MUX}$$

- b: # FA IN EACH BLOCK
- -1: 1ST BLOCK HAS A FIXED WELL-KNOWN C_{IN}

WE CAN COMPUTE THE OPTIMAL VALUE FOR b

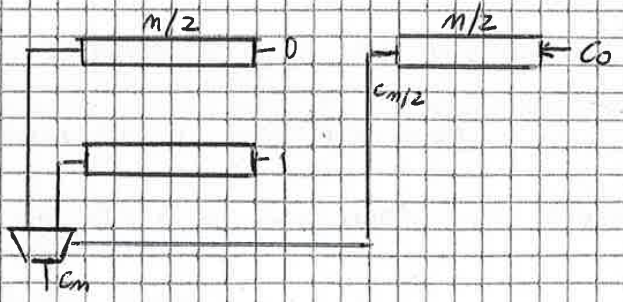
IT HAS THE SAME FORM OF CSKA

$\frac{\partial t_{CSKA}}{\partial b} = 0$ → YOU WILL GET A DELAY EXPRESSION THAT IS PROPORTIONAL TO $\sqrt{n}$

$$t_{CSEA} = \Theta(\sqrt{n})$$

- **LOGARITHMIC SOLUTION** → LET'S EXPLOIT THE METHOD IN A DIFFERENT FORM:

1ST LEVEL) $n \to n/2$ LET'S ASSUME n BITS FOR 2 OPERANDS THAT ARE SPLITTED IN 2 PARTS OF $n/2$ BITS



AT EACH APPLICATION OF THE DECOMPOSITION WE JUST INTRODUCE A SINGLE LEVEL OF MULTIPLEXING

⇓

DELAY GROWS WITH $\log_2(n)$

		SINGLE CONTRIBUTIONS	
AT LEVEL → 1 :	$n \to n/2$	$\frac{n}{2} t_{FA}$	t_{MUX}
2 :	$n/2 \to n/4$	$\frac{n}{4} t_{FA}$	$2 t_{MUX}$
3 :	$n/4 \to n/8$	$\frac{n}{8} t_{FA}$	$3 t_{MUX}$
⋮			
i :		$\frac{n}{2^i} t_{FA}$	$i t_{MUX}$

WHAT IS THE MAXIMUM VALUE FOR i? REVIEW THE SIZE OF BLOCKS UP TO HAVE A BLOCK THAT CONTAINS JUST A F

$$\frac{n}{2^i} = 1$$

↓ THIS MEANS

$$2^i = n \quad \leftarrow \text{LOGARITHMIC RELATIONSHIP}$$

$$i = \log_2 n$$

$$t_{FA_i} = \frac{n}{2^i} t_{FA} \qquad t_{MUX_i} = \log_2 n \cdot t_{MUX}$$

⇓

EXPLOITING THIS STRATEGY UP TO A BLOCK COMPOSED OF 1 BIT:

$$\boxed{t_{SCEA}(n) = t_{FA} + \log_2 n \cdot t_{MUX}}$$

✓ IT'S A VERY LIMITED INCREASING!

IT'S A VERY POWERFUL APPROACH

⇓

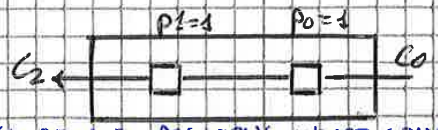
GOOD FOR A LARGE NUMBER OF BITS

✗ HIGH AREA OVERHEAD

HOW YOU ORGANIZE YOUR IDEA HAS A GREAT IMPACT ON YOUR RESULTS

BUT WE CAN ORGANIZE COMPUTATION IN A TREE-LIKE MANNER

LET'S DEFINE BLOCK < GENERATE / PROPAGATE

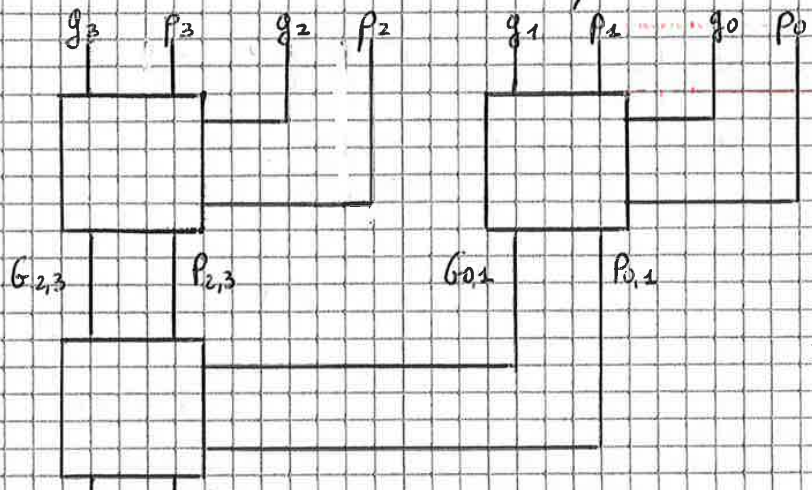


BEGINNING / END } OF RANGE OF THE POSITIONS WE'RE CONSIDERING

$$P_{0,1} = p_1 \cdot p_0$$
$$G_{0,1} = g_1 + p_1 g_0$$

} PROPAGATE/GENERATE ARE ASSUMED TO 0:1 STAGES

⇒ WE CAN DEFINE < PROPAGATE / GENERATE FOR ANY RANGE OF BIT POSITIONS



WITH 3 BLOCKS THAT PERFORM EXACTLY THE SAME COMPUTATIONS

$G_{0,3}$ $P_{0,3}$ → WITH $P_{0,3} = P_{0,1} \cdot P_{2,3}$

$G_{0,3} = G_{2,3} + G_{0,1} \cdot P_{2,3}$

IF YOU HAVE MORE BITS → YOU CAN COMPUTE THE FINAL MOST SIGNIFICANT CARRY

⇓

YOU OBTAIN A FINAL RESULT WHICH DEPENDS ON THE LOGARITHM OF THE NUMBER OF BITS

↓

BECAUSE YOU HAVE A TREE-LIKE ORGANIZATION

LET'S COMPUTE THE DIFFERENT DELAY CONTRIBUTIONS:

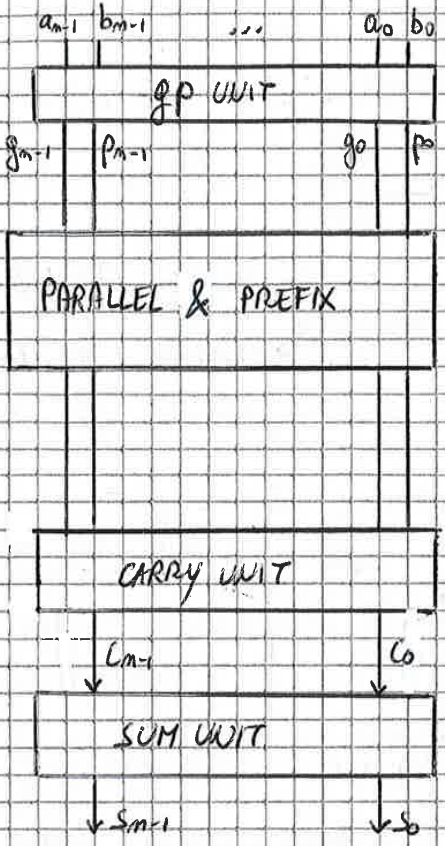
- t_{gp} (OR t_{AL}) → FROM a_i, b_i TO g_i, p_i
- $(\log_2 n - 1)\, t_{BL}$ → FROM g_i, p_i UP TO G, P OVER RANGES $0 \div 3$ AND $4 \div 7$ IN THE FORWARD DIRECTION
  $(\log_2 n - 1)$ STEPS ARE ENOUGH
  ONCE WE HAVE $G_{0,3}$, $P_{0,3}$, $G_{4,7}$, $P_{4,7}$ PLUS c_0, WE CAN IMMEDIATELY START COMPUTING $c_4 = c_0 P_{0,3} + G_{0,3}$ WITH NO NEED TO WAIT FOR c_8 WHICH IS COMPUTED IN PARALLEL
- $\log_2 n \cdot t_{BA}$ → FROM c_0 UP TO c_i, V_i → IN THE BACKWARD DIRECTION IT'S NECESSARY TO COMPUTE UP TO n
- t_S (OR t_{AA}) → TO COMPUTE THE SUM STARTING FROM THE RECEIVED c_{in}

ASSUMING $t_{BA} \simeq t_{BL} \simeq t_B$

$$t_{CLA} = t_{gp} + t_S + t_B (2\log_2 n - 1)$$

✕ MAIN DISADVANTAGE → WE NEED TO CROSS THE PATH TWICE
THIS CAN BE AVOIDED BY MIXING p, g PARTIAL SIGNALS

THE GENERAL ARCHITECTURE OF AN ADDER WITH THE PARALLEL PREFIX PROBLEM



⇒ g,p COMPUTATION

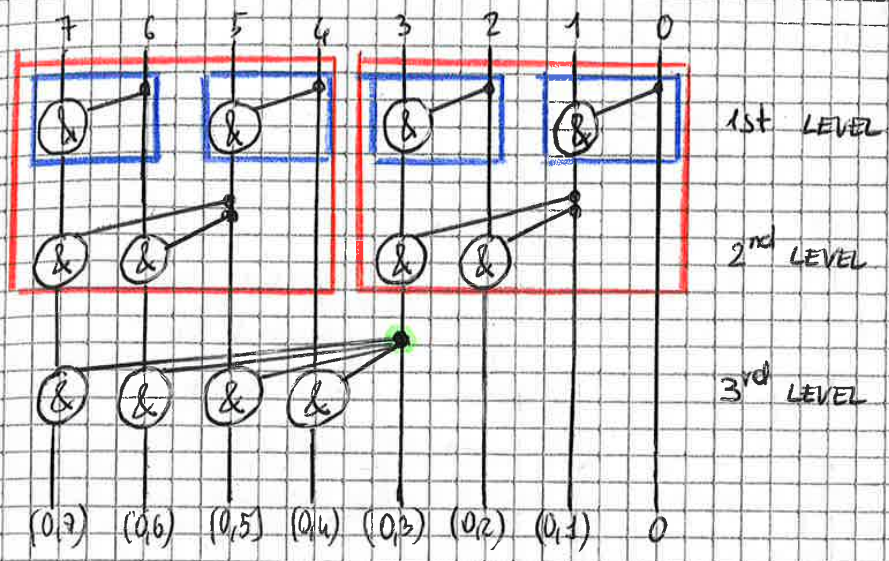
⇒ IT'S THE MOST COMPLEX PART
IT'S A NETWORK WHERE THE OPERATOR IS COMBINED IN A WAY TO GIVE ALL GENERATE AND PROPAGATE WE NEED

⇒ TO COMPUTE ALL THE CARRIES

→ SUM COMPUTATION

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 59 di 190

eg: CASE $n=8$



THIS SCHEME CAN BE GENERALIZED TO ANY n

- IMPLEMENTATION COST

$$C(n) = \frac{1}{2} n \cdot \log_2(n)$$

(4 · 3)

EVERY ROW REQUIRES A NUMBER OF OPERATIONS THAT IS $\frac{1}{2}n$ BITS OF EACH OPERAND MULTIPLIED BY THE NUMBER OF STAGES

- LONGEST PATH

$$t(n) = \log_2(n)$$

→ IT GOES THROUGH A NUMBER OF OPERATORS EQUAL TO THE NUMBER OF LEVELS (ROWS) WE HAVE

- FAN-OUT

$$f(n) = \frac{1}{2} n$$

→ A NODE IS VERY HEAVILY LOADED BY $n/2$
#CONNECTIONS = $\frac{1}{2}$ NUMBER OF BITS
↓
IT CAN INTRODUCE A VERY RELEVANT DELAY ON THE CRITICAL PATH
↓
CORRESPONDING TO PROPAGATION TIME THAT CAN BE RELEVANT

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 61 di 190

3. KOGGE-STONE → YOU HAVE OVERLAPPED INTERVALS IN THE ARCHITECTURE FOR A FASTER ARCHITECTURE

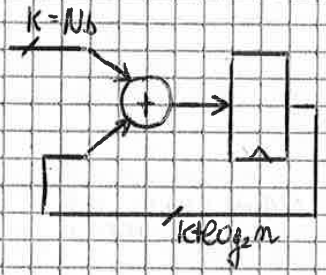
	$C(n)$	$t(n)$	$f(n)$	
L-F	$\frac{1}{2} n \log_2 n$	$\log_2 n$	$n/2$	→ BEST ONE FOR COMPLEXITY
B-K	$2n - \log_2 n - 2$	$2\log_2 n - 2$	$\log_2 n$	
K-S	$n\log_2 n - n + 1$	$\log_2 n$	$\log_2 n$	BEST

↘ BEST FOR SPEED BUT VERY LARGE COMPLEXITY

MULTI-OPERAND ADDERS → ARE EMPLOYED FOR FIR FILTER MULTIPLICATION WHICH IS IMPLEMENTED IN MULTIPLE ADDITIONS

WE CAN HAVE:

- **LINEAR** ORGANIZATION

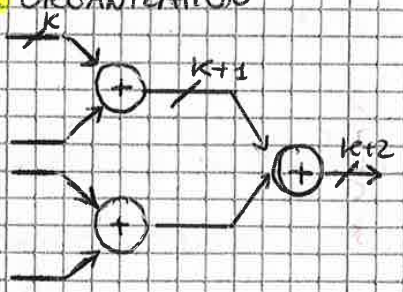


WE HAVE n OPERANDS

YOU'LL NEED n CYCLES WITH A CORRESPONDING DELAY WHICH IS PROPORTIONAL TO THE MAXIMUM NUMBER OF ADDITIONAL BITS

LATENCY: $t = n \cdot (k + \log_2 n) t_{FA}$

- **TREE-LIKE** ORGANIZATION

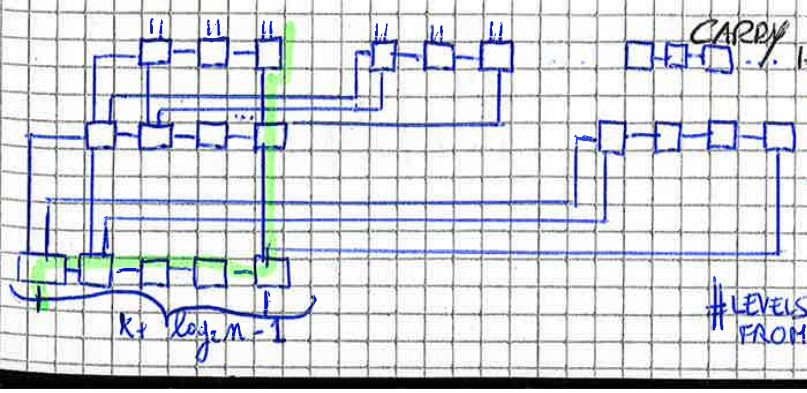


IT GOES THROUGH A LOWER NUMBER OF LEVELS

NB IN ORDER TO START THE COMPUTATION OF THE LSBs WHICH ARE COMING FROM LSBs OF THE PREVIOUS LAYERS ↓ WE DON'T NEED TO CALCULATE THE WHOLE ADDITION:

LATENCY: $t = \log_2 n \cdot t_{FA} + t_{FA}(k + \log_2 n) - 2t_{FA}$ (#LAYERS)

EXAMPLE: $n = 8$ $k = 3$



CARRY SELECT ADDERS WILL HAVE BETTER PERFORMANCES

$\log_2(n)$ LEVELS

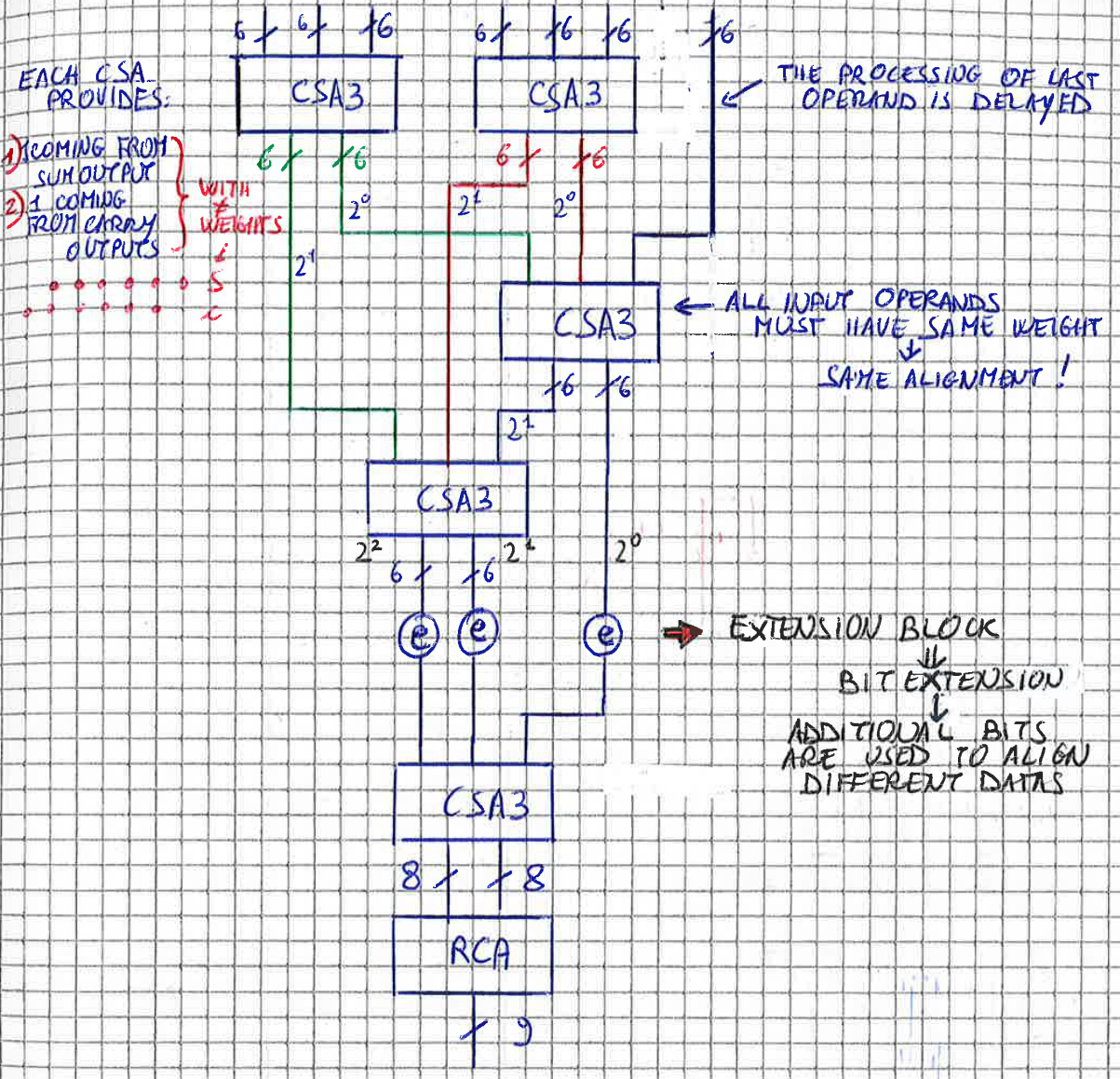
cp:

$\log_2 n - 1 + k + \log_2 n - 1$

LEVELS EXCEPT FROM LAST ONE

FA IN LAST BLOCK

WITH $N = 7$; EACH OPERAND WORKS WITH 6 BITS



EACH CSA PROVIDES:
1) COMING FROM SUM OUTPUT
2) 1 COMING FROM CARRY OUTPUTS
} WITH WEIGHTS i ≤ S ≤ i

THE PROCESSING OF LAST OPERAND IS DELAYED

← ALL INPUT OPERANDS MUST HAVE SAME WEIGHT
↓
SAME ALIGNMENT !

⇒ EXTENSION BLOCK
⇓
BIT EXTENSION
↓
ADDITIONAL BITS ARE USED TO ALIGN DIFFERENT DATAS

CSA → MADE UP WITH { SEVERAL STAGES OF CSA / FINAL STAGE OF RCA

$N = 7 \rightarrow R(N) = 4$

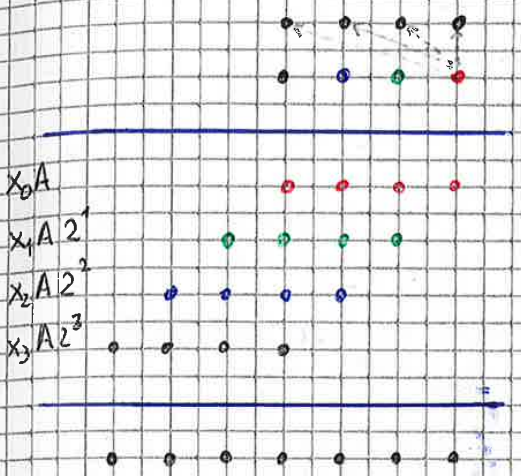
© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 65 di 190

MULTIPLIERS

GIVEN OPERANDS ON K BITS
- A → MULTIPLICAND $a_{K-1}, \dots a_0$
- X → MULTIPLIER $x_{K-1}, \dots x_0$
- P → PRODUCT $p_{2K-1}, \dots p_0$

LET'S APPLY THE DOT NOTATION

WHEN WE PERFORM A MULTIPLICATION → WE DO ALL THE PARTIAL PRODUCTS
WE MULTIPLY EACH BIT OF THE MULTIPLIER BY ALL THE BITS OF THE MULTIPLICAND (A)
AND WE SUM ALL THE CONTRIBUTIONS TOGETHER



WE CAN APPLY ONE BETWEEN 2 METHODS, DEPENDING ON THE DIRECTION WE PROCEED

SUM FROM THE TOP TO THE BOTTOM
⇓
RIGHT-SHIFT MULTIPLICATION

SUM FROM THE BOTTOM TO THE TOP
⇓
LEFT-SHIFT MULTIPLICATION

⚠ THE CRITICAL POINT IS THE MULTIPLICATION BY THE WEIGHT (2^i)
⇓
A BARREL SHIFTER IS NECESSARY
BUT IT'S TOO HEAVIER / EXPENSIVE

THE 2 ALGORITHMS ARE EXPLOITED IN A WAY THAT AVOIDS THE EMPLOYMENT OF THE BARREL SHIFTER

✓ EVEN IF THE OUTPUT IS ON 2K BITS => THE COMPLEXITY OF THE ADDER IS REDUCED TO K BITS

✓ WE CAN SHARE THE $\overset{P}{X}$ REGISTER
RIGHT PART OF P REGISTER
⇓
BECAUSE EACH TIME WE SHIFT P WE'RE SHIFTING X, TOO
⇓
REGISTER SHARING → TO SAVE SOME COMPLEXITY

2) LEFT-SHIFT MULTIPLICATION → IT'S THE SAME METHOD BUT IN THE OPPOSITE DIRECTION
⇓
WE START FROM THE BOTTOM TO THE TOP

PARTIAL PRODUCT: $\boxed{p^{(j)} = 2p^{(j-1)} + x_j A}$

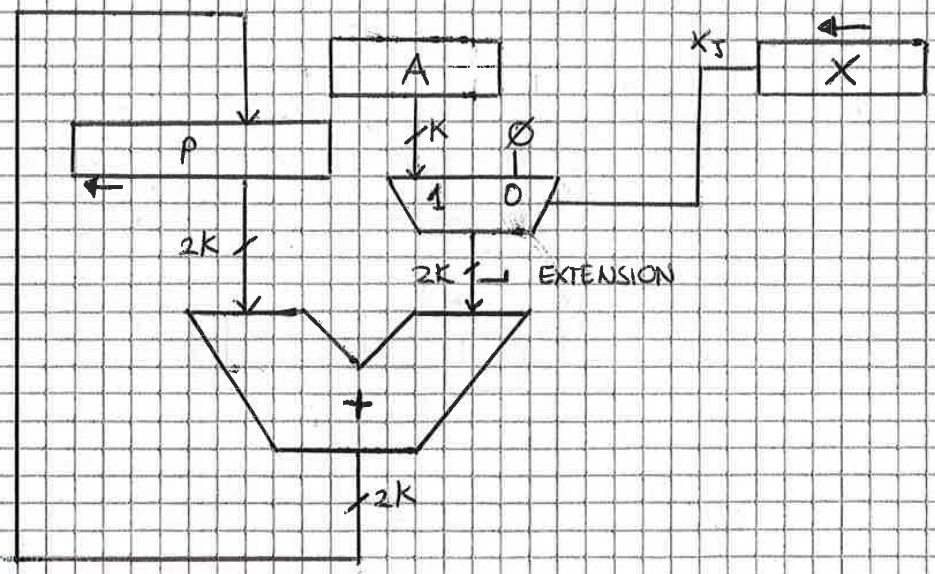
$p^{(k)} = 0$

$\forall j = k-1, \dots, 0$

WITH $k = 4$:

- $j = 4$ $\quad p^{(4)} = 0$
- $j = 3$ $\quad p^{(3)} = 0 + x_3 A = x_3 A$
- $j = 2$ $\quad p^{(2)} = 2x_3 A + x_2 A$
- $j = 1$ $\quad p^{(1)} = 2^2 x_3 A + 2x_2 A + x_1 A$
- $j = 0$ $\quad p^{(0)} = 2^3 x_3 A + 2^2 x_2 A + 2x_1 A + x_0 A$

LET'S IMPLEMENT ITS ARCHITECTURE:



✗ SHARING BETWEEN $\overset{P}{X}$ ISN'T POSSIBLE

✗ HIGHER INTERNAL PARALLELISM

BOOTH MULTIPLIER

IT'S POSSIBLE TO REPRESENT A DIGITAL VALUE INTO A LOWER NUMBER OF OPERATIONS

IF WE ADAPT THIS NEW CODING:

$(0,1) \longrightarrow (0,1,\bar{1})$ WITH $\bar{1} = -1$

IF WE ACCEPT THIS ADDITIONAL COMPLEXITY WE HAVE THE POSSIBILITY TO SIMPLIFY THE OPERATIONS

$X = 0001111$

$P = X \cdot A = A + A2 + A \cdot 2^2 + A \cdot 2^3$

$X = 0010001$

WE REPLACE THE LSB WITH -1 AND WE ADD TO THE TOP A 1

⇓

THIS REDUCES THE NUMBER OF ADDITIONS WE'LL PERFORM

$P = -1 \cdot A + A \cdot 2^4$ → ✓ JUST ONE SUBTRACTION INSTEAD OF 3 ADDITIONS!

LET'S TAKE EACH COMBINATION OF A COUPLE OF BIT OF X AND TRY TO UNDERSTAND WHICH OPERATION THAT HAS TO BE DONE

X_j	X_{j-1}	
0	0	NoP
0	1	$+A \cdot 2^j$
1	0	$-A \cdot 2^j$
1	1	NoP

NONE OPERATION HAS TO BE PERFORMED

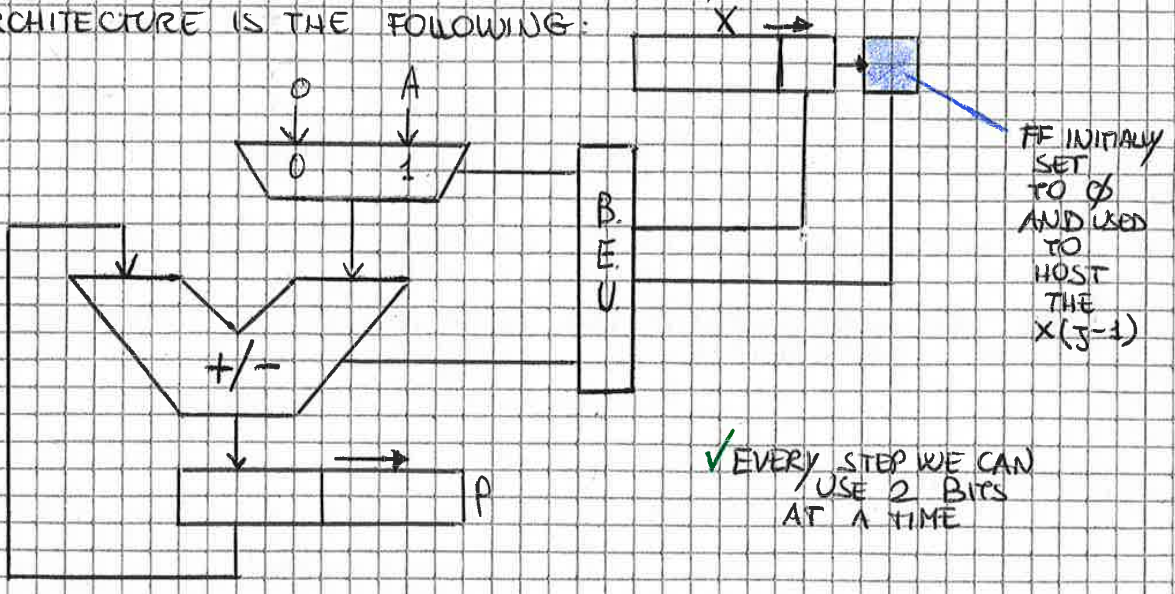
→ in our example

$X = 0001111$ (bits 4 3 2 1 0)

$+2^4 A$ $\quad -2^0 A = -A$

OUR ADDER IS ORGANIZED IN SUCH A WAY TO PERFORM ADDITION/SUBTRACTION CONTROLLED BY THE VALUE OF X

OUR ARCHITECTURE IS THE FOLLOWING:

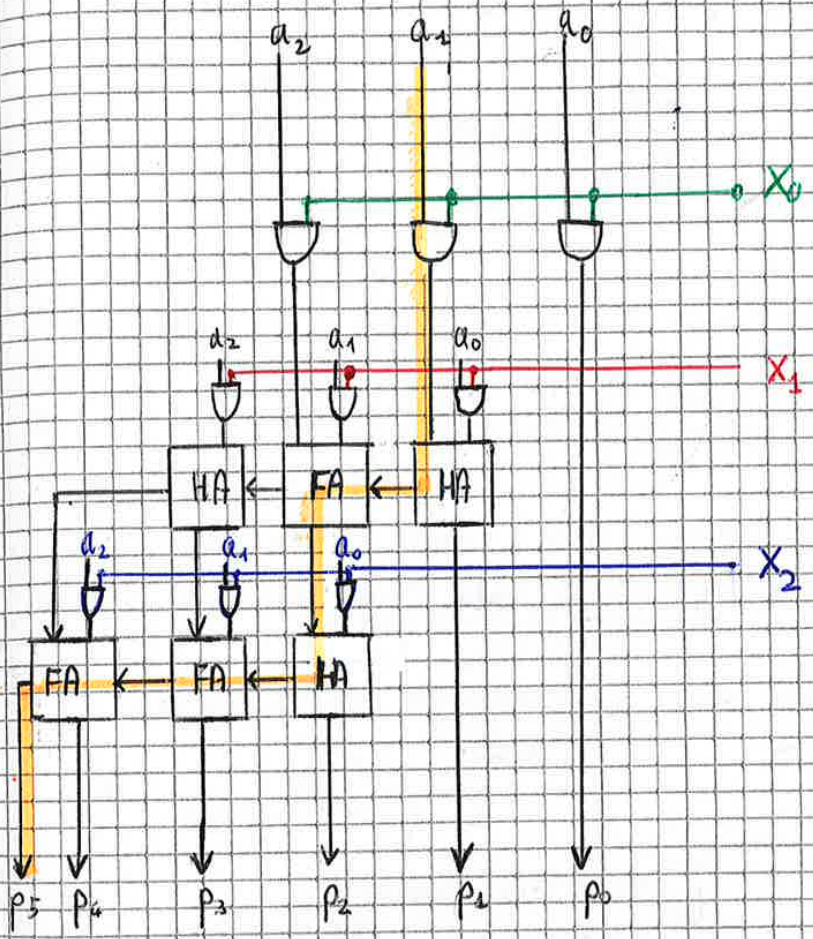


FF INITIALY SET TO 0 AND USED TO HOST THE X(j-1)

✓ EVERY STEP WE CAN USE 2 BITS AT A TIME

ARRAY ARCHITECTURE

IF WE WANT TO PERFORM THE MULTIPLICATION IN ONE STEP, WE NEED TO IMPLEMENT AN ARRAY STRUCTURE



cp:

ROWS $k-1$

COLUMNS $k + k-2 = 2k-2$

(k: LAST ROW; $k-2$: PREVIOUS ROWS)

→ IT DEPENDS ON THE ELEMENTS IT CROSSES → WE NEED TO DISTINGUISH BETWEEN → HA / FA → s_i / c_i

⚠ IN THIS CASE MULTIPLE CRITICAL PATHS ARE GENERATED, ALL OF THEM WITH ROUGHLY THE SAME DELAY

⇓ THERE ARE OTHER PATHS THAT GO THROUGH THE SAME NUMBER OF COMPONENTS

⇓

✗ THIS IS NOT ACCEPTED IN DESIGN!

IT'S BETTER TO WORK WITH JUST ONE C.P.!

WITH MULTIPLE CPS → IT'S NOT FEASIBLE

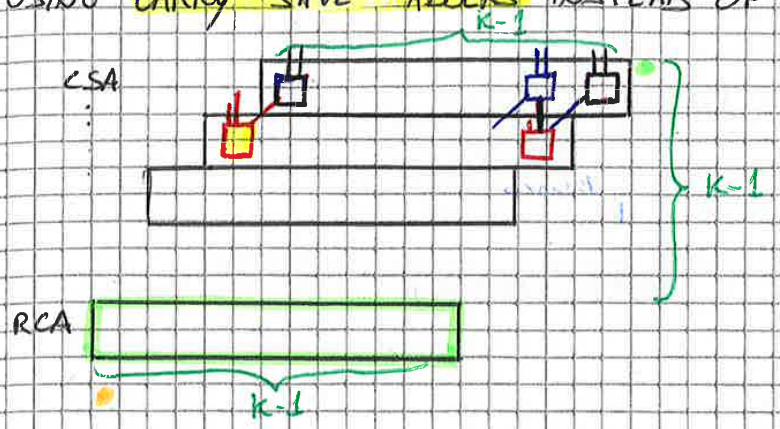
⇓ THAT'S WHY IT'S NOT A VERY GOOD ARCHITECTURE

↓

! A BETTER ARCHITECTURE REORGANIZES IT WITH A UNIQUE CRITICAL PATH

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 73 di 190

USING CARRY SAVE ADDERS INSTEAD OF RCAs WE OBTAIN:



WE TAKE JUST 2 PARTIAL PRODUCTS AT A TIME

FOR EACH CALCULATION < SUM BIT
CARRY BIT → IS PROPAGATED ALONG THE DIAGONAL LINE IN THE FOLLOWING ROW

LET'S ANALYZE THE STRUCTURE:

- 1st ROW → ALL HAs
- FROM 2nd TO (K-1) ROW → FULL ADDER WE HAVE

CARRY BIT COMING FROM PREVIOUS ROW PREVIOUS COLUMN

SUM BIT FROM PREVIOUS ROW SAME COLUMN

→ PARTIAL PRODUCT

EXCEPTION → FOR THE LAST ELEMENT OF EACH ROW THAT RECEIVES

- LAST ROW → RCA → BECAUSE AT THE BOTTOM OF THE ARRAY WE HAVE 2 OPERANDS AND WE NEED TO PERFORM THE SUM TO COMPLETE THE COMPUTATION

SIZE OF THIS STRUCTURE:

#ROWS: $K-1+1=K$

#COLUMNS: $K-1+K-1=2K-2$

GLOBAL DELAY: CONSIDERING t_{SUM} ↓ IN VERTICAL DIRECTION

t_{CARRY} ↙ IN THE DIAGONAL DIRECTION

t_{CARRY} ← IN THE ORIZONTAL DIRECTION FOR THE LAST LINE

$$t_{DELAY} = t_{SUM}(K-1) + (K-2+K-1)\,t_{CARRY}$$

(K-2+K-1): WE NEED TO EXCLUDE THE LAST ELEMENT IN THE HORIZONTAL DIRECTION

(K-1): IN THE DIAGONAL DIRECTION

NOW WE'VE JUST ONE CP!

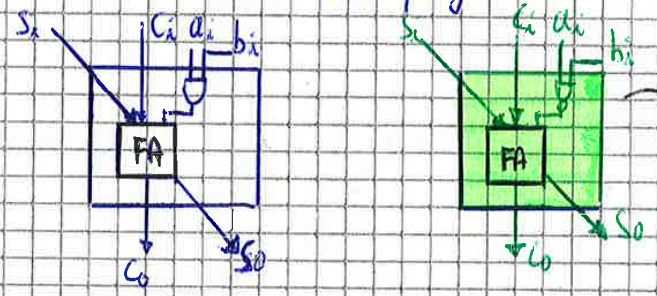
Example: $n=4$

$$P = a_3 b_3 2^6 - 2^3 \sum_{j=0}^{2} a_3 b_j 2^j - 2^3 \sum_{i=0}^{2} b_3 a_i 2^i + \sum_{i=0}^{3}\sum_{j=0}^{3} a_i b_j 2^{i+j}$$

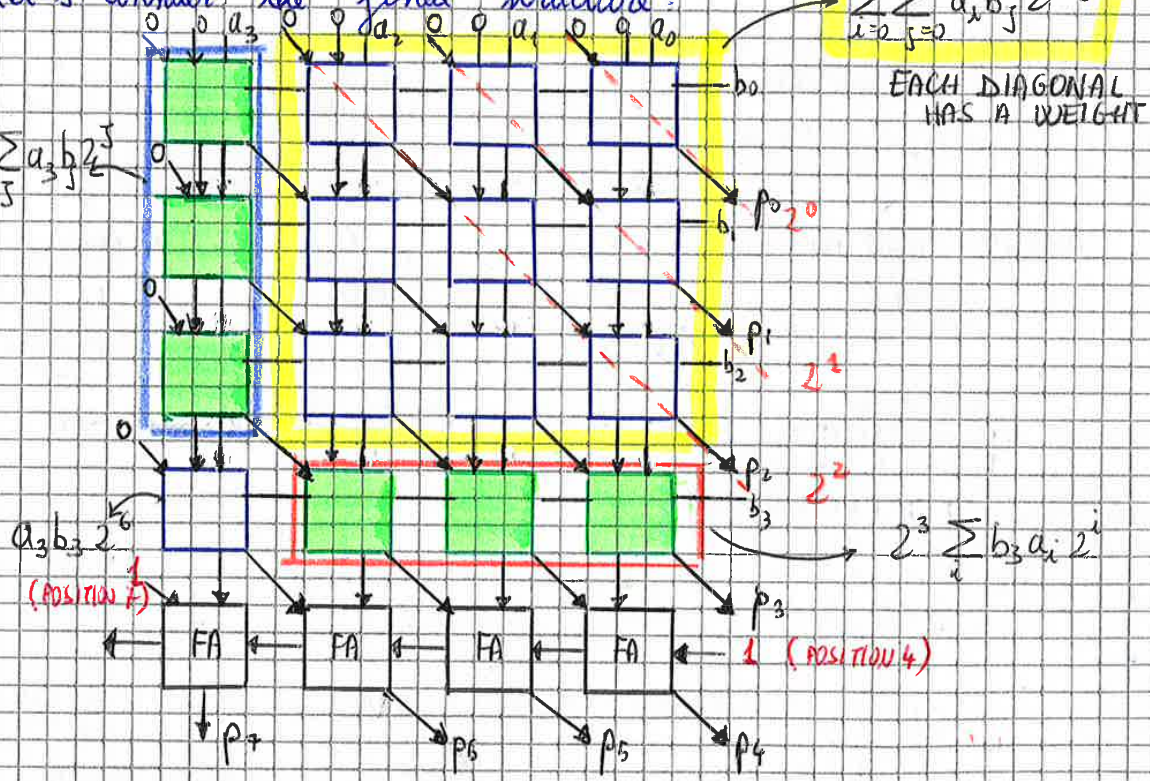
LET'S TAKE THEIR COMPLEMENT AND ADD 1 IN POSITION n AND $2n-1$

$$P = a_3 b_3 2^6 + 2^3 \sum_{j=0}^{2} \overline{a_3 b_j} 2^j + 2^3 \sum_{i=0}^{3} \overline{b_3 a_i} 2^i + \sum_{i=0}^{3}\sum_{j=0}^{3} a_i b_j 2^{i+j} + 1\cdot 2^4 + 2^7$$

Our architecture will employ these 2 blocks:



Let's consider the final structure:



CSA → PROVIDES US BETTER PERFORMANCES

⇓

APPLIVE CAN IMPROVE THE THROUGHPUT

OVERALL WE HAVE TO:

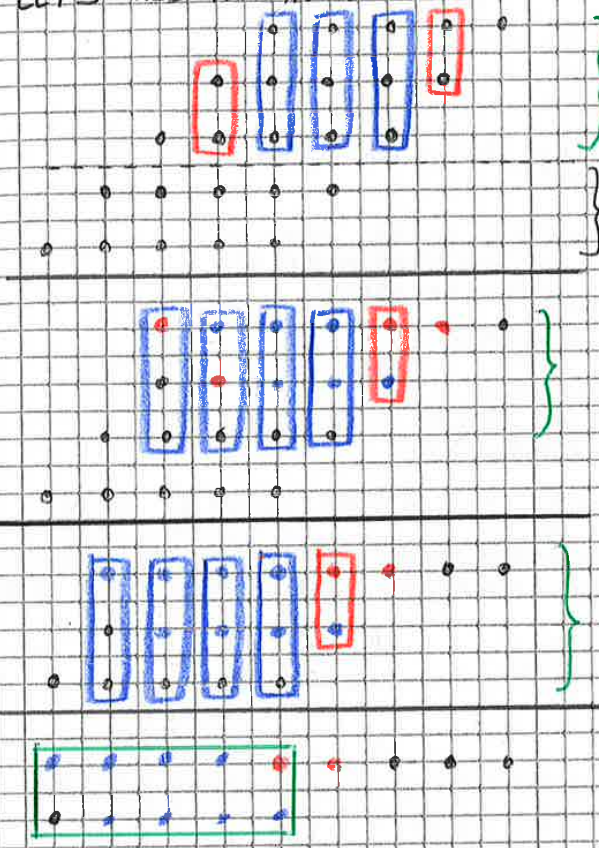
- TAKE THE INVERSE FOR EVERY BIT IN CONTRIBUTIONS AND
- ADD A 1 < IN POSITION n
  IN POSITION $2n-1$ (SIGN BIT)

1. WALLACE TREE → FOLLOW THESE STEPS

1) DIVIDE YOUR PARTIAL PRODUCTS IN GROUPS OF 3 → EACH TIME YOU CAN FIND 3 LINES OF PRODUCTS TO APPLY COMPRESSORS
2) ∀ GROUP ALLOCATE < HA → WITH 2 DOTS / FA → WITH 3 DOTS

IF YOU HAVE 1 DOT PROPAGATE IT TO THE NEXT CHANGE

LET'S ASSUME $n = 5$



HA	FA
2	3
1	4
1	4
1	4

TOTAL COMPLEXITY = 5HA + 15FA

(YOU SHOULD ALSO CONSIDER THE COMPLEXITY OF AND GATES THAT WE CONSIDER NEGLIGIBLE)

IN THIS CASE → WE ADD FA, HA → ASAP

WITH DADDA → WE ADD FA, HA → ALAP

ESTIMATION OF DELAY: $t = 3t_{FA} + 4t_{FA} + t_{HA} = 7t_{FA} + t_{HA}$

($3t_{FA}$: TREE; $4t_{FA} + t_{HA}$: FINAL RCA)

IN THE INITIAL SITUATION → WE NEED TO REDUCE THE NUMBER OF OPERANDS FROM 5 TO 4 ROWS

⇓

100 WHEN YOU HAVE A COLUMN ≤ 4 DOTS NOTHING HAS TO BE DONE

eg: COLUMN 1-2-3-4 → WE HAVE 4 DOTS PER COLUMN SO WE KEEP THE SAME DOTS IN THE NEXT LEVEL

COLUMN 5 → WE NEED TO REDUCE THE NUMBER OF DOTS FROM 5 TO 4 SO THE MINIMUM THAT COULD BE DONE IS USING AN HA

↘ THE OTHER 3 DOTS ARE MOVED TO THE NEXT STAGE

COLUMN 6 → 100 ⚠ CARRY BIT WHICH WILL RESULT FROM HA IN THE NEXT LEVEL OTHER COLUMNS → ARE COPIED TO THE NEXT STAGE

⇓

LET'S APPLY THE SAME PROCEDURE FOR ALL THE OTHER LEVELS

LEVEL 3 → COLUMN 5 - PAY ATTENTION! YOU HAVE 4 DOTS + 1

⇓

YOU NEED TO HAVE 3 DOTS IN THE NEXT LEVEL SO A COMPRESSOR FACTOR OF 2 (FA) IS NECESSARY

n=5

	TREE		RCA		TOTAL	
	#FA	#HA	#FA	#HA	#FA	#HA
WALLACE	11	4	4	1	15	5
DADDA	8	4	7	1	15	5

THE TOTAL NUMBER OF FULL ADDERS/HALF ADDERS IS THE SAME

DADDA TREE → TENDS TO REQUIRE

↙ A LOWER NUMBER OF FA/HA IN THE TREE

↘ A HIGHER NUMBER OF FA IN THE RCA

DIVISION → IT'S NOT A SIMPLE OPERATION
SIGNIFICANTLY MORE DIFFICULT THAN MULTIPLICATION

LET'S CONSIDER THE FOLLOWING AMOUNTS

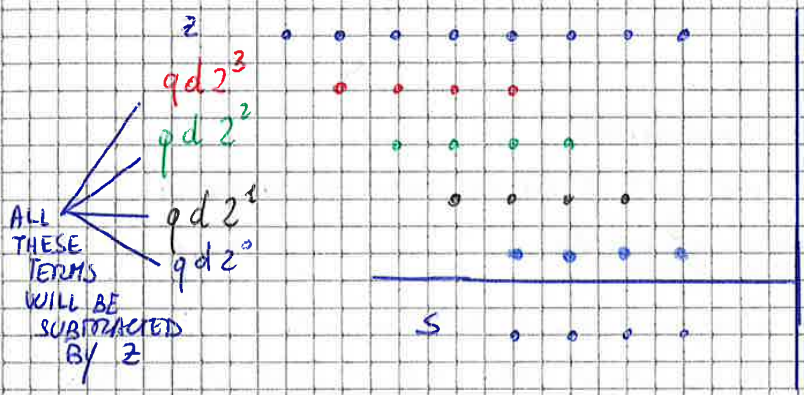
- z : DIVIDEND OF $2k$ BITS
- d : DIVISOR OF k BITS
- s : REMINDER OF k BITS
- q : QUOTIENT OF k BITS ⇒ THIS ISN'T ALWAYS TRUE
  IF $d=1 \rightarrow q=z$ ON $2k$ BITS
  ⇓
  THERE IS A CONSTRAINT

SCALING: $\frac{z}{d} = q$ → WHERE q CANNOT BE REPRESENTED ON k BITS

$\frac{z}{d \cdot 2^k} = q'$ THEN WE APPLY SOME SCALING TO HAVE A CORRECT q

$q = q' 2^k$

LET'S SUPPOSE $k=4$



AS FOR THE MULTIPLICATION THE DIRECT IMPLEMENTATION ISN'T SUGGESTED
⇓
YOU SHOULD ALLOCATE A BARREL SHIFTER
BUT WE DON'T WANT TO USE IT SO
WE EMPLOY A RECURSIVE METHOD
WITH JUST 2 SHIFTS

$$s^{(j)} = 2 \cdot s^{(j-1)} - q_{k-j} 2^k d$$

WITH $s^{(0)} = z$

$j=0$ $s^{(0)} = z$

$j=1$ $s^{(1)} = 2 \cdot z - q_3 2^4 d$ ← WE'RE SHIFTING BOTH z AND d TO LEFT BUT OF DIFFERENT AMOUNTS

$j=2$ $s^{(2)} = 2 s^{(1)} - q_2 2^4 d$ ← IT'S A RELATIVE SHIFT OF 2 POSITIONS WITH RESPECT TO z AND SO ON...

$j=3$ $s^{(3)} = 2 s^{(2)} - q_1 2^4 d$

$j=4$ $s^{(4)} = 2 s^{(3)} - q_0 2^4 d = s \cdot 2^4$ ← EVERY VALUE IS SHIFTED TO LEFT FOR AN AMOUNT THAT IS EQUIVALENT TO THE NUMBER OF BITS

⇓
WE NEED TO SHIFT IT BACK OF k BITS (IN OUR CASE $k=4$)

IF WE WANT TO SPEED UP THE PROCESS WE NEED TO CHANGE THE APPROACH

USING A **NON-RESTORING ALGORITHM** WE PERFORM OPERATIONS (3) AND (4) AT THE SAME TIME

LET'S CONSIDER THE CASE $q=1 \rightarrow$ WRONG

$$2s^{(1)} - 2^4 d < 0$$

RESTORING ALGORITHM

TAKE ITS VALUE AND REPLACE IT:

$$s^{(2)} = 2s^{(1)} - 2^4 d + 2^4 d = 2s^{(1)} \quad (a)$$

THEN TAKE

$$2s^{(2)} - 2^4 d = 4s^{(1)} - 2^4 d$$

$$s^{(2)} = 2s^{(1)}$$

NON-RESTORING ALGORITHM

WE AVOID OPERATION (a) OF GOING BACK TO THE PREVIOUS VALUE → EVEN IF IT'S WRONG WE KEEP THIS VALUE

$$s^{(2)} = 2s^{(1)} - 2^4 d$$

WE DON'T HAVE ANY DEPENDENCE ON SIGN! EVEN IF IT'S WRONG WE KEEP THIS VALUE

$$2s^{(2)} + 2^4 d = 2\left(2s^{(1)} - 2^4 d\right) + 2^4 d =$$

$$= 4s^{(1)} - 2 \cdot 2^4 d + 2^4 d = 4s^{(1)} - 2^4 d$$

THE TEST ON THE SIGN ISN'T REQUIRED TO STORE INTO THE REGISTER

↓

BUT IT'S NECESSARY AT THE FOLLOWING OPERATION TO DECIDE IF IT'S A SUM < SUBTRACTION

KEY ADVANTAGE OF THIS APPROACH → OPERATION TIME: **SHORTER C.P.!**

THE ONLY DIFFERENCE WITH THE PREVIOUS ARCHITECTURE IS A PROGRAMMABLE ADDER/SUBTRACTOR

IF THE PARTIAL REMINDER IS

- POSITIVE (Cout = 1) NOTHING CHANGES
- NEGATIVE (Cout = 0) YOU DON'T COME BACK TO THE PREVIOUS OPERATION YOU KEEP THE VALUE WHICH IS SHIFTED TOWARDS LEFT BUT IN THE FOLLOWING OPERATION WE SUBTRACT $-d$ NOT $+d$

NB: WITH NON-RESTORING APPROACH → WE CANNOT EXCLUDE THE POSSIBILITY THAT A NEGATIVE Cout CAN BE COMPUTED AT THE LAST STEP

↓

IF YOU GET A FINAL $S < 0$ YOU CANNOT COMPENSATE IT

TO FIX THIS PROBLEM WE NEED TO ADD ONE MORE STAGE

IF $S^{(4)}$ IS POSITIVE OR

↓

IF $S^{(4)}$ IS NEGATIVE WE NEED A NEW OPERATION TO ADD BACK d

SIGNED DIVISION

$$z = d \cdot q + s$$

HAS TO BE

$$SIGN(s) = sign(z)$$

$$SIGN(q) = sign(z) \oplus sign(d)$$

$$s^{j} = 2\,s^{(j-1)} - q_{k-j}\,2^{k}d$$

TO CHOOSE $q_{k-j} \in \{-1, 1\}$ WE NEED TO COMPARE d AND s

- $d \geq 0$
  - $s \geq 0$ → NORMAL CONDITION → SUBTRACTION HAS TO BE IMPLEMENTED → $q_{k-j} = 1$
  - $s < 0$ → NEGATIVE PARTIAL REMINDER WE NEED TO REPLACE SUB → ADDITION ⇒ $q_{k-j} = -1$
- $d \leq 0$ (EXTENSION)
  - $s \geq 0$ → ADDITION ⇒ $q_{k-j} = -1$
  - $s < 0$ → APPLY SUBTRACTION AND TAKE THE VALUE FOR $q_{k-j} = 1$

⚠ COMPARE AT EVERY STEP THE SIGN OF d AND s

- IF EQUAL → PERFORM SUBTRACTION
- IF ≠ → PERFORM ADDITION

AFTER THE FINAL STEP WE HAVE AN OPTIONAL FINAL CORRECTION

IF $SIGN(s^{(k)}) \neq SIGN(z)$ THEN WE NEED A CORRECTION STEP

IF $z > 0$ & $s^{(k)} < 0$ THEN $z = (q+1)d + (s-d)$

IF $z < 0$ & $s^{(k)} > 0$ THEN $z = (q-1)d + (s+d)$

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 87 di 190

NOW WE HAVE THE TEMPTATIVE VALUES OF

- QUOTIENT $q' = -1\ \ 1\ \ -1\ \ 1$ → LET'S TRANSLATE THIS BSD REPRESENTATION INTO A BINARY REPRESENTATION

$0\ \ 1\ \ 0\ \ 1$

$0\ 1\ 0\ 1\ 0$

$0\ 1\ 0\ 1\ 1$

$1\ 1\ 0\ 1\ 1$ → $q' = -5$ IN BINARY FORM

- REMINDER: $S' \rightarrow S'2^k = S^{(4)} \rightarrow S' = 11110$

NOW LET'S COMPARE $sign(S) \neq sign(Z)$ → THEY MUST BE THE SAME!
↓
WE NEED A COMPENSATION STEP

- IF $Z < 0 \rightarrow S = S' + d$
  $q = q' - 1$
- IF $Z > 0 \rightarrow S = S' - d$
  $q = q' + 1$

} ⇒ THAT'S OUR CASE: $S^{(4)} < 0$, $Z > 0$
⇓
A COMPENSATION STEP IS NEEDED

q' 11011 +
1
11100 → $q = -4$

S' 11110 +
$-d$ 00111 =
00101 → $S = 5$

IF WE WANT A FASTER IMPLEMENTATION → WE CAN ADOPT A HIGHER RADIX

- RADIX 2 → WE HANDLE 1 BIT AT A TIME
- RADIX 4 → WE HANDLE 2 BITS AT A TIME
- RADIX 8 → WE HANDLE 3 BITS AT A TIME

} WITH HIGH RADIX IMPLEMENTATION WE CAN HAVE

$$S^{(s)} = r \cdot S^{(s-1)} - q_{k-s}(r^k \cdot d)$$

DIVISOR THAT MUST BE PARTIALLY ALIGNED TO THE REMINDER

WITH RADIX 4 → 2 SHIFT AT TIME

q IS ASSUMING A VALUE IN THE EXTENDED RANGE

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 89 di 190

AN OTHER OPTION → SRT DIVIDER = SWEENAG ROBERTSON TOCHER

LET'S IMAGINE 2 NUMBERS IN BASE 10

$z = 12257968$ $\quad d = 2043$

QUOTIENT IS 5 → NORMALLY WE NEED TO TRY WITH DIFFERENT CASES

BUT LET'S COUSIDER MSB IN $< d^2 >$ WHY NOT USING 6? THAT'S NOT CORRECT

12257968	2043
12258	600 (-1) ⇒ 5999
-32	
-2043	

INSTEAD OF REPLACING 6 WITH 5 AND REDO THE CALCULATION WE WANT TO CONTINUE

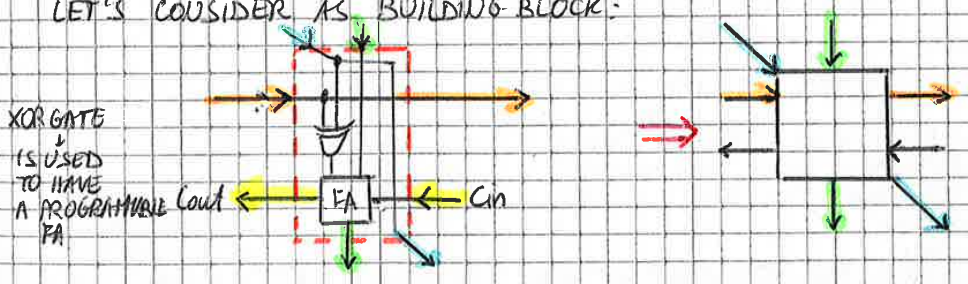
⇓

BY USING A REDUNDANT DIGIT SET ADMITTING -1 WE CAN COMPENSATE FOR THE 6 IN THE MOST SIGNIFICANT POSITION

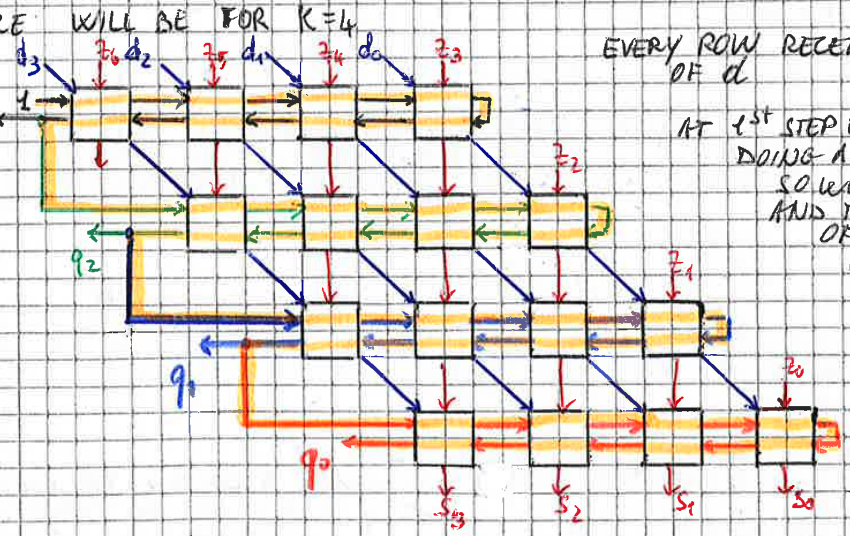
KEY IDEA: ADMIT IN THE CALCULATION AN EXTENSION TO NEGATIVE VALUES BECAUSE WE HAVE THE POSSIBILITY TO COMPENSATE THE WRONG VALUE

ARRAY DIVIDER → WE PERFORM DIVISION IN A SINGLE STEP

LET'S COUSIDER AS BUILDING BLOCK:



OUR STRUCTURE WILL BE FOR K=4



1ST ROW: $2z - 2^4 d$

2ND ROW: $2s^{(1)} \pm 2^4 d$ ← DEPENDING ON PREVIOUS OUTCOME

EVERY ROW RECEIVES THE VALUE OF d

AT 1ST STEP WE'RE ALWAYS DOING A SUBTRACTION SO WE HAVE A 1 AND THE CARRY OUT OF EACH ROW IS USED AS INPUT IN THE FOLLOWING ITERATION

CP → IS LONG IT CROSSES EVERY ELEMENT IN THE ARRAY

⇓

IT GROWS QUADRATICALLY WITH THE NUMBER OF BITS → $CP \propto K^2$

IT'S POSSIBLE TO IMPLEMENT THIS STRUCTURE BY USING PIPLENING TO IMPROVE THROUGHPUT

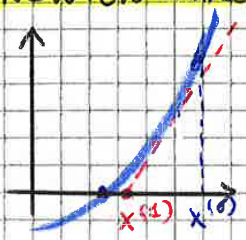
17/11/17

RECIPROCATION

$q = \left\lfloor \frac{2}{d} \right\rfloor$ TO OBTAIN $q \to \left\lfloor \frac{1}{d} \right\rfloor \cdot 2$ WITH INTEGER

2 METHODS:

1) **NEWTON-RAPSON** METHOD



GIVEN $f(x)$ I WANT TO FIND THE ROOT X_{target}

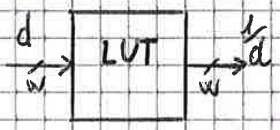
$$X^{(i+1)} = X^{(i)} - \frac{f(x^{(i)})}{f'(x^{(i)})}$$

YOU IDENTIFY THE TANGENT OF THE INITIAL CURVE EACH TIME

WE CAN DEFINE THE FUNCTION AS:

$$f(x) = \frac{1}{x} - d$$ → THERE ARE ALSO OTHER METHODS TO COMPUTE THE ROOT OF A FUNCTION

2) **LUT-BASED APPROACH** → USES THE MEMORY AS A LUT

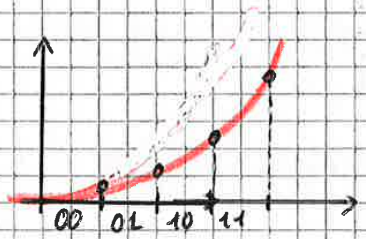


WHERE $\frac{1}{d}$ WAS PREVIOUSLY STORED

× MEMORY SIZE → $w \cdot 2^w$ → IT HAS AN EXPONENTIAL GROWING WITH w

↓

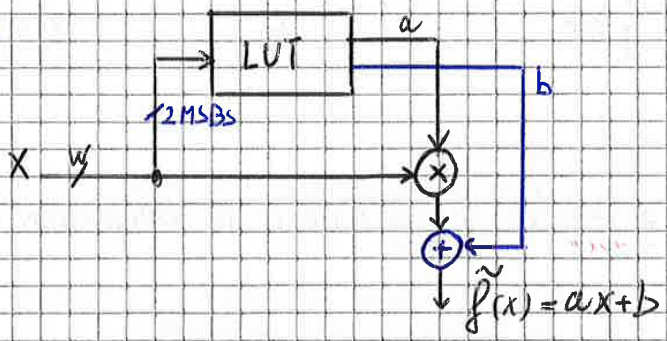
TO LIMIT THE SIZE OF MEMORY → WE NEED TO ADOPT A **PIECEWISE LINEAR APPROXIMATION** PWL



WE FIND A LINEAR APPROXIMATION TO ANY SEGMENT

↓

GIVEN X OF w BITS, WE DERIVE 2MSBs THAT IDENTIFY THE CORRESPONDING SEGMENT AND ENTER THEM IN A LUT



LET'S DIVIDE THE INTERVAL INTO A LARGER NUMBER OF SEGMENTS AND HAVE A BETTER APPROXIMATION

↓

IN SOME CASES IT ISN'T EFFICIENT → WHEN WE HAVE NO SHARP VARIATIONS

FLOATING - POINT REPRESENTATION

FOR NON-INTEGER VALUES → THERE ARE 2 POSSIBILITIES

FIXED-POINT REPRESENTATION
↓
NO IMPACT ON HARDWARE
BUT JUST A PROPER EXTENSION
ON IN/OUT DATA

FLOATING-POINT REPRESENTATION
↓
EACH NUMBER IS DIVIDED INTO 3 PARTS:
1) SIGN
2) EXPONENT
3) SIGNIFICAND

$+1,6 \cdot 2^{-3} = 0,2$

IT IS IMPORTANT TO HAVE A STANDARD REPRESENTATION
↓
IEEE STANDARDS 754-1985 THAT EXISTS IN 2 FORMATS:
1) SINGLE PRECISION → 32 BITS
2) DOUBLE PRECISION → 64 BITS

LET'S CONSIDER SINGLE PRECISION → THAT IS REPRESENTED ON 32 BITS
- SIGN: 1 BIT
- EXPONENT: 8 BITS
- FRACTION: 23 BITS
  ↓
  IS CLOSE TO THE SIGNIFICAND EXCEPT FROM THE MSB = 1

$1.6 \cdot 2^{-3}$ (1.6 = FRACTION)

SIGNIFICAND → 1. FRACTION

LET'S FOCUS ON THE EXPONENT → THERE ARE $2^8 = 256$ DIFFERENT VALUES TO REPRESENT BOTH POSITIVE AND NEGATIVE VALUES
↓
WE'LL USE THE **BIASED REPRESENTATION**

BIASED EXPONENT = UNBIASED EXPONENT + 127

$127 = 2^{K-1} - 1$

WHY THIS REPRESENTATION?
IT'S A CONVENTION → ✓ THE KEY ADVANTAGE: IS THAT USING BIAS WE CAN COMPARE 2 NUMBERS IN A SIMPLER WAY THEN IN 2s COMPLEMENT REPRESENTATION

IMAGINE TO HAVE 2 NUMBERS:

1) 2's COMPLEMENT REPRESENTATION

$1.0 \cdot 2^{-1}$	0	11111111	0...0
$1.0 \cdot 2^{+1}$	0	00000001	0...0

} TO COMPARE THEM WE SHOULD USE A NORMAL COMPARATOR

2) BIASED REPRESENTATION

$1.0 \cdot 2^{-1}$	0	01111110	0...0
$1.0 \cdot 2^{+1}$	0	10000000	0...0

} YOU CAN START FROM THE MSB AND GO TO RIGHT AS FAR AS YOU FIND A
- ZERO → IN ONE NUMBER
- ONE → IN THE OTHER
  ↓
  YOU ALREADY KNOW THE GREATER NUMBER

SIGN | EXPONENT: 127 − 1 = 126, 127 + 1 = 128 | FRACTION ON 23 BITS

Example 2 → 1 10000001 01...0
BIAS EXP: 128 → UNBIASED = 128 − 127 = 2, FRACTION $(0,01)_2 = (0,25)_{10}$ ⇔ $-1,25 \cdot 2^{2} = 5$

LET'S CONSIDER SINGLE PRECISION

EXP = 0 → FRACTIONAL PART
- f = 0 → S=0 / S=1 → 0 / 0 (VALUE 0 HAS 2 REPRESENTATIONS)
- f ≠ 0 → DENORMAL VALUES → WE REPLACE CORRECT EXPONENT AND LOOK AT f: $0.f \cdot 2^{-126}$, INTEGER PART = 0

$1 \le EXP \le 254$ → NORMAL REPRESENTATION → RANGE $(2^{-126} \div 2^{127})$, INTEGER PART = 1

EXP = 255 → FRACTIONAL PART
- f = 0 → SIGN BIT: S=0 → +∞, S=1 → −∞
- f ≠ 0 → NaN

MULTIPLE REPRESENTATIONS WITH A COMMON EXPONENT = 11111111

eg: −∞ 1 11111111 0…0 (23)

Example: Let's consider a representation ← 1 BIT: SIGN; 3 BITS: EXPONENT; 4 BITS: FRACTIONAL

bias = 3, $E_{MIN} = -2$, $E_{MAX} = 3$

S	EXP	FRACTIONAL	VALUE
0	000	0000	0
0	000	0001	$2^{-4} \cdot 2^{-2} = 2^{-6}$
	...		
0	001	0000	$1.0 \cdot 2^{-3+1} = 1.0 \cdot 2^{-2}$
	...		
0	110	1111	$(1.111) \cdot 2^{-3+6} = 1.111 \cdot 2^{3}$
0	111	0000	+∞
0	111	0001	NaN
	...		⋮
0	111	1111	NaN
1			→ same for all corresponding negatives

$2^{-(EXP-1)}$