Mapping Signal Processing Algorithms to Architecture

Sumam David S
Head, Dept of Electronics & Communication
National Institute of Technology Karnataka, Surathkal, India
sumam@ieee.org

Objectives

- *At the end of the lecture*
  - have gained an overview of implementation aspects of signal processing algorithms
  - have gained an understanding of the relationship between the parameters influencing the choice of implementation
  - appreciate the approaches in efficient mapping signal processing algorithms to architecture
  - be familiar with few transformations that help the designer to develop different solutions for a given signal processing algorithm

Organisation

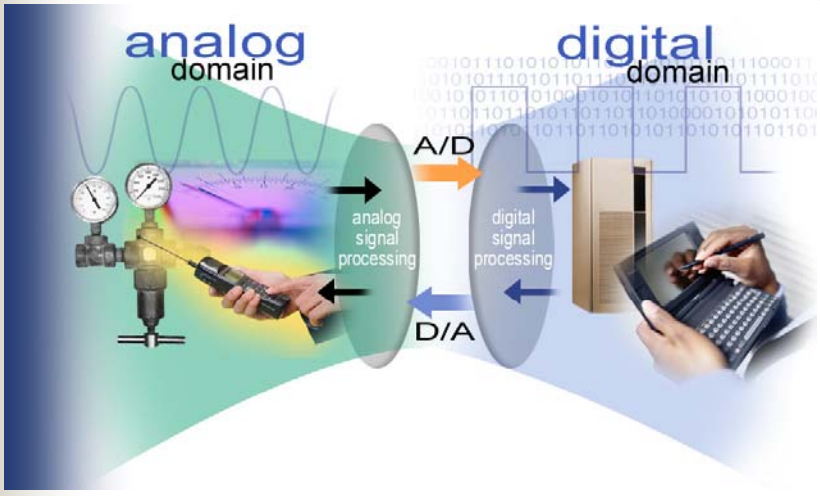
- What is SP ?
- Applications of SP
- Key algorithms
- Implementation options & trade-offs
- Key approaches in mapping algorithms to architecture

NITK Surathkal, India

- 12°N, 74°E
- West coast of India
- Arabian Sea
- Mangalore

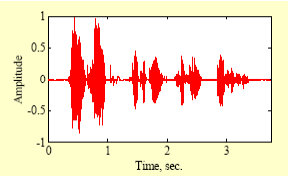
SP – where do u find it?

Typical DSP system

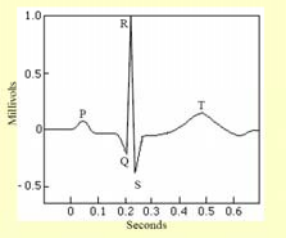


Examples of typical signals

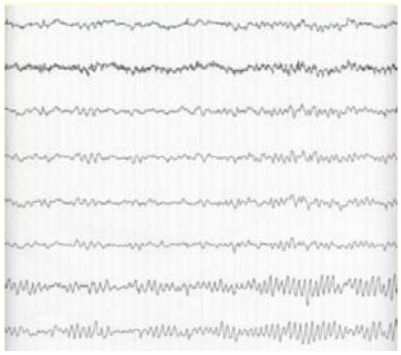
- **Speech and music signals**
  - Represent air pressure as a function of time at a point in space



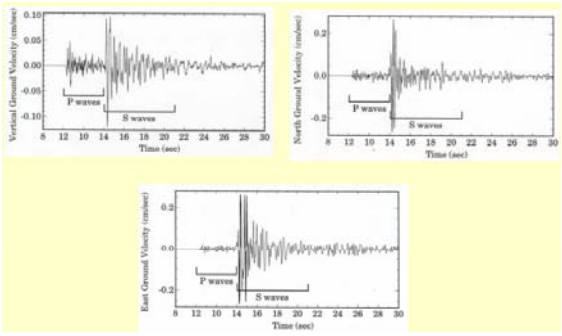
Biomedical signals



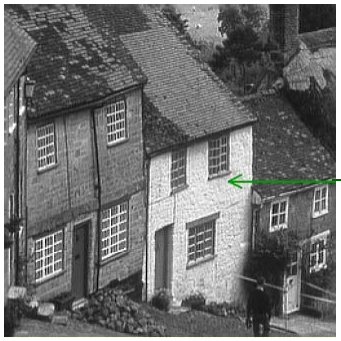
EEG



Seismic signal



Image



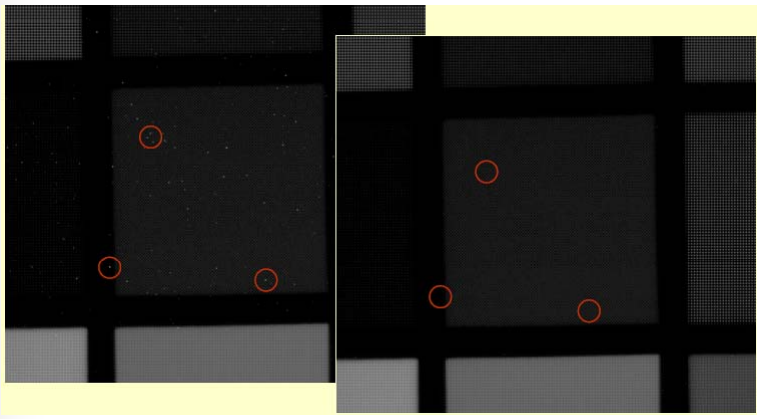
I(x,y)

Video

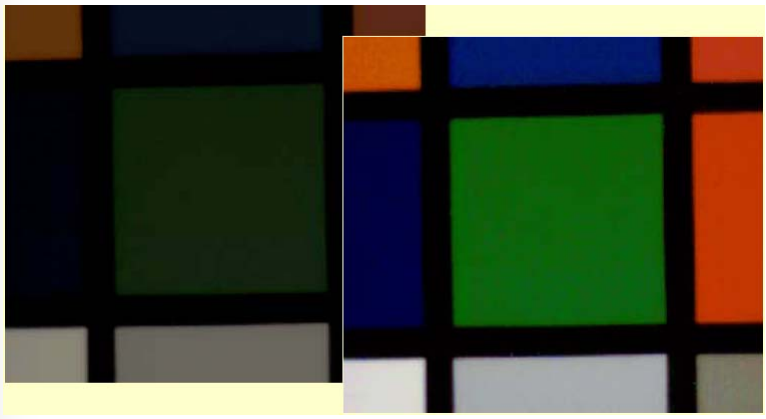
Digital Camera

- Image Processing Algorithms
  - Bad pixel detection and masking
  - Color interpolation
  - Color balancing
  - Contrast enhancement
  - False color detection and masking
  - Image and video compression

Bad pixel detection & masking



Color interpolation & balancing



Digital Sound Synthesis

- Guitar with nylon strings
- Marimba
- Tenor saxophone

Signal Compression

- Original music
  - Audio Format: PCM 16.000 kHz, 16 Bit
  - (Data size 66206 bytes)
- Compressed music
  - Audio Format: GSM 6.10, 22.05 kHz
  - (Data size 9295 bytes)

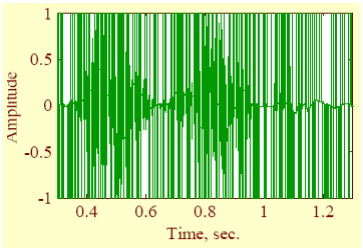
Image Compression

8 bpp

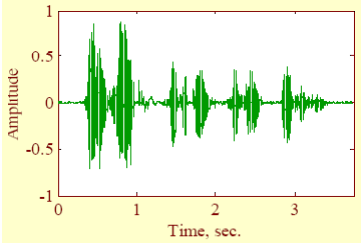
0.5 bpp

Signal Enhancement

Noisy speech



Noise removed



Contrast enhancement

Original Enhanced

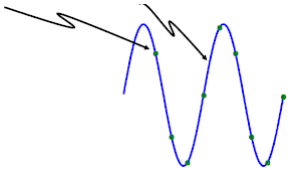
Noise removal

20% pixels corrupted with additive impulse noise

Noise-removed version

Signal Processing

- Works on discrete samples of a continuous time signal
- Real time requirement
- Data driven
- Programmable or custom DSPs



Implementation choice

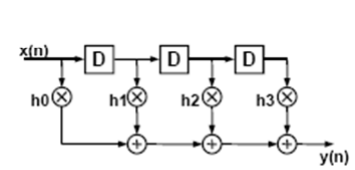
- Depends on
  - Sampling rate
  - Throughput
  - Power - energy
  - Area
  - Wordlength – precision
  - Flexibility
  - Time market
  - Volume

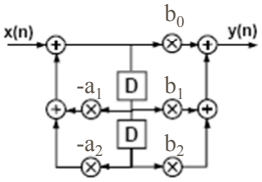
Examples of DSP Primitives

- Convolution
- Digital Filters
  - Finite Impulse response (FIR)
  - Infinite Impulse Response (IIR)
- Correlation
- Discrete Fourier Transform / Fast Fourier Transform (FFT)
- Discrete Cosine Transform (DCT)
- Least Mean Square (LMS)

Applications often comprised of many primitives

Two basic DSP structures





Finite impulse Response (FIR)
No feedback
Order -4

Infinite impulse Response (IIR)
Feedback
Order - 2

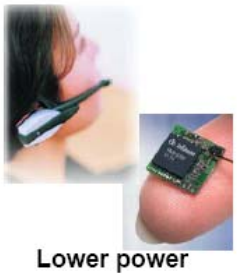
Different applns different demands

Flexibilty
Complexity

Processors
FPGAs

Low power
Low cost
Flexibilty

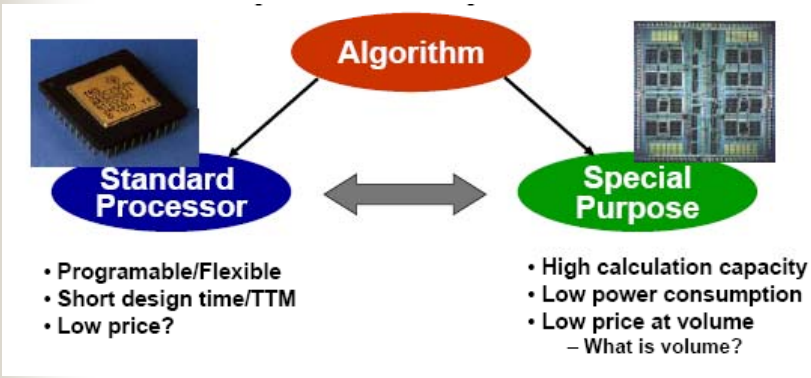
Processors
ASICs



Lower power
Lower cost

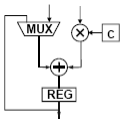
ASICs
Processors

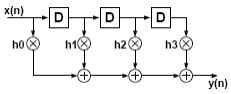
Standard processors or special purpose ?



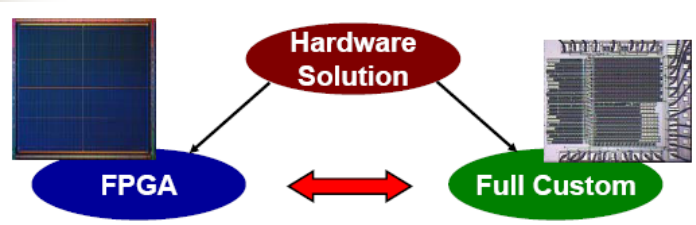
Architectural options

- OTS (Off The Shelf) processors
  - Programmable microprocessors or DSP
  - Based on generic computational units, for DSPs usually MAC
  - Prefabbed or IP cores
- Time-multiplexed application specific processors
  - Several algorithmic operations performed on same hardware unit
  - Trades reduced HW for longer computation time
- Hardware mapped architectures
  - One (or more) hardware unit per algorithmic operation
  - High hardware cost and high throughput



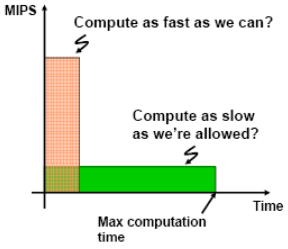


Hardware options



FPGA - Field Programmable gate arrays
ASIC – Application Specific Integrated Circuit ASIC

Key design issue today - energy



Utilising the computational time

- clock frequency
- supply voltage
- Sleep modes

Algorithm 2 Architecture

- Which structure gives optimal performance, energy and area?
- How to get from a signal processing algorithm to an EFFICIENT implementation using
  - Different numbering systems
  - Pipelining
  - Parallelism
  - Retiming, Scheduling
  - Strength reduction, i.e. complexity of operations.
  - etc, etc,...
- in a structured way!

An example

- Datapath and control path

DSP basic operations

FIR

$$y_n = \sum_{k=0}^{N-1} h_k \cdot x_{n-k}$$

IIR

$$y_n = \sum_{k=0}^{M} a_k \cdot x_{n-k} + \sum_{k=1}^{N} b_k \cdot y_{n-k}$$

Transforms
FFT,DCT

$$X_m = \sum_{n=0}^{N-1} x_n \cdot e^{-j2\pi mn/N}$$

Decomposition (SVD, LU, QR) - Matrix operations

Arithmetic operations ? x, +, shifts

Non terminating – data flow , sampling rate

Number representation

- Binary number system
  - Integers eg. 0110 → 6 (decimal)
    - Two's complement 1010 → -6 (decimal) (4 bit)
  - Fractional values
    - Divide by 2^{n-1} → range [-1, +1)
      - n bits : [2^{n-1} , 2^{n-1}-1]
      - 8 bits : ?
    - Fractional : -1 , +127/128
      - Resolution : 1/256 of range :[-1. +1)

Floating point numbers

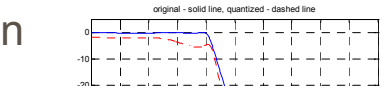
- 32 bits

S(1)	Exponent (8)	Mantissa (23)

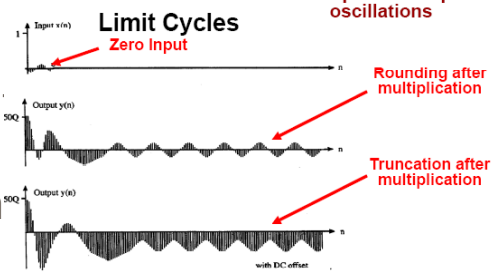
- Number magnitude: $(1\text{-mantissa}) \times 2^{\text{exponent-127}}$
  - Smallest +ve: $(1.00\ldots.0) \times 2^{-126}$
  - Largest +ve: $(1.11\ldots.1) \times 2^{127} \sim 2^{128}$
  - Mantissa – determines resolution
  - Exponent – determines dynamic range
- Much wider dynamic range at the cost of energy, area, computation time
- Energy efficient implementation fixed point used

Finite word length effects

- Coefficient quantisation
- Signal quantisation
  - Round off noise
  - Limit cycles
- Scaling
  - Adjust signal to hardware
- Saturation arithm



Example: zero input oscillations



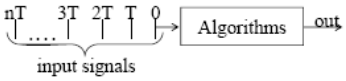
Implementation Flow

- High level description - Specifications
  - Equations
  - Matlab, C/C++
- Architecture
  - Building blocks, speed and area
  - Block diagram
  - Optimisations: #bits, # coefficients
  - Hardware Description languages – Verilog, VHDL
- Compile (Synthesize)
  - Mapping to gates
  - Place & Route

Implementation of FIR filters

$$y_n = \sum_{k=0}^{3} h_k \cdot x_{n-k}$$

Algorithm non-terminating

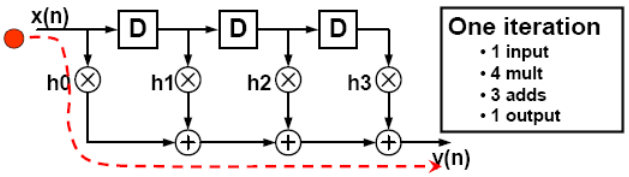


for $n = 0$ *to* ∞

$$y(n) = h_0 x(n) + h_1 x(n-1) + h_2 x(n-2) + h_3 x(n-3)$$

One execution of the loop – one iteration
Iteration period = time for one iteration

Direct form 4 tap Fir filter



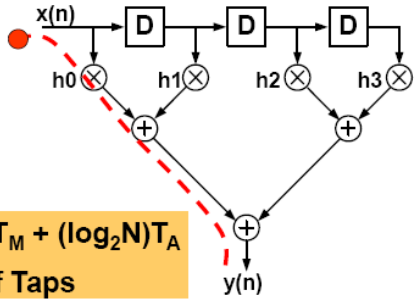
Critical path = longest path without delay element

Critical path sets bound on clock frequency

$$T_{clk} \geq T_{critical}$$

$T_{Critical} = T_M + (N-1)T_A$

FIR with adder tree



$T_{Critical} = T_M + (\log_2 N)T_A$

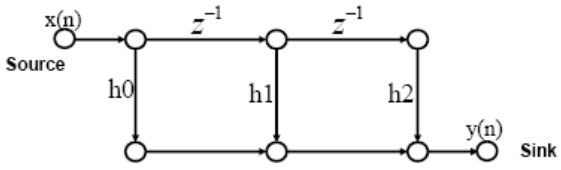
N = Nr. of Taps

Critical path

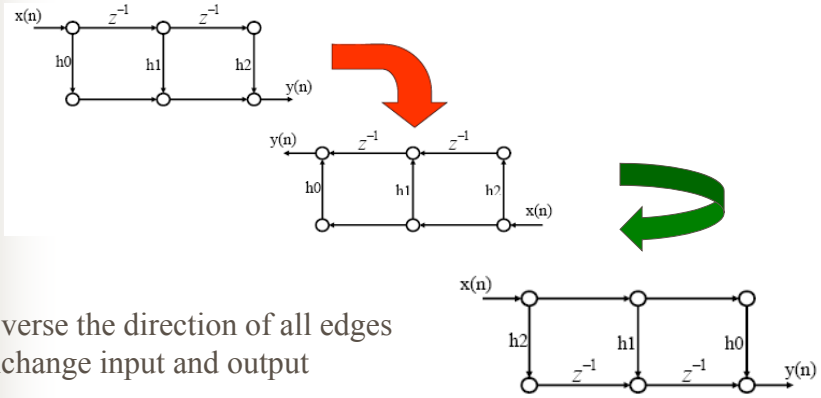
- Critical path
  - between delays
  - input to delay
  - delay to output
  - input to output
- Sampling rate = no of samples processed/sec
- Latency = time difference between output and corresponding input
  - Combinatorial logic = gate delays
  - Sequential logic = no of clock cycles

Signal Flow Graph

- Nodes: represent computations and/or task
- Directed edge *(j, k):* denotes a linear transformation from the input signal at node *j* to the output signal at node *k*
  - in digital usually limited to delays and constant multipliers
- Source = no entering edge
- Sink = only entering edges

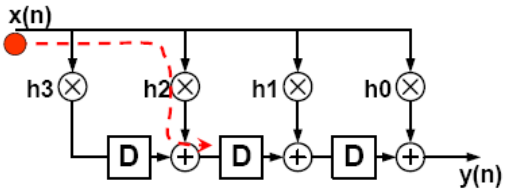


Transposition



Reverse the direction of all edges
Exchange input and output

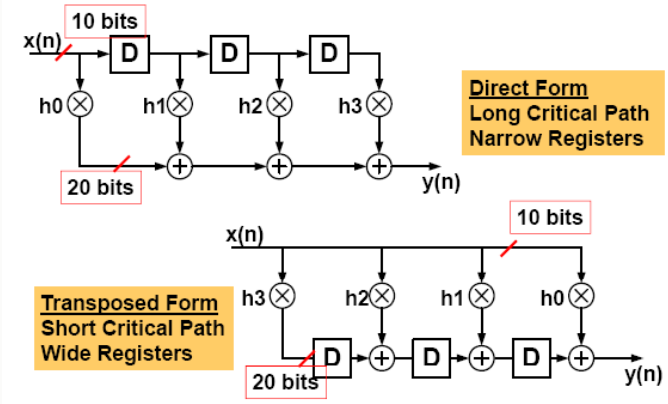
Transposed FIR filter



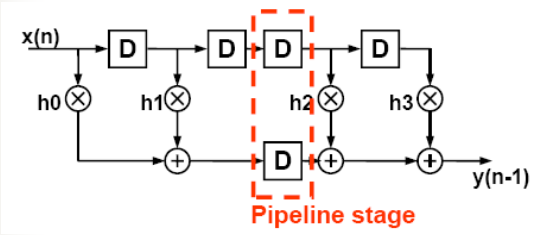
$T_{Critical} = T_M + T_A$

N = Nr. of Taps

Fixed point implementation



Pipelining

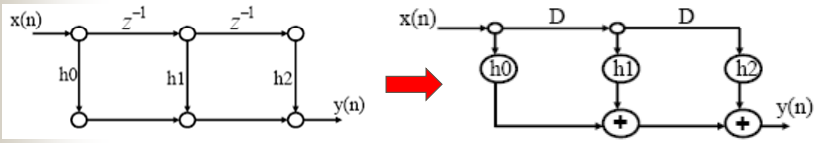


Pipelining increases latency

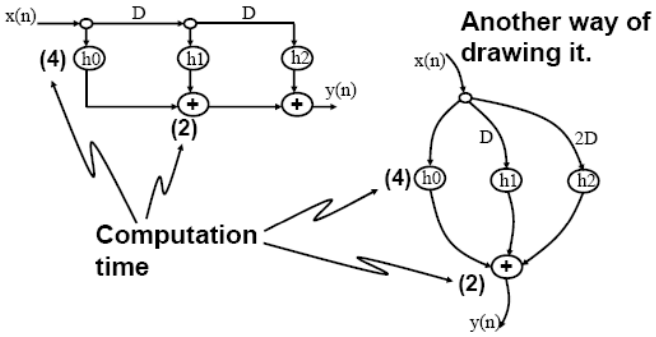
More registers can bring down Critical Path delay to $\max(T_M, T_A)$

Data Flow Graph

- DFGs capture the data-driven property of DSP algorithm
- Data-Flow Graph – nodes represent computations (or functions)
  - directed edges represent data flow with non-negative number of delays
  - a node can execute whenever all its input data are available.
  - Each edge describes a precedence constraint between two nodes in DFG:
- Intra-iteration precedence constraint: if the edge has zero delays
- Inter-iteration precedence constraint: if the edge has one or more delays



DFG

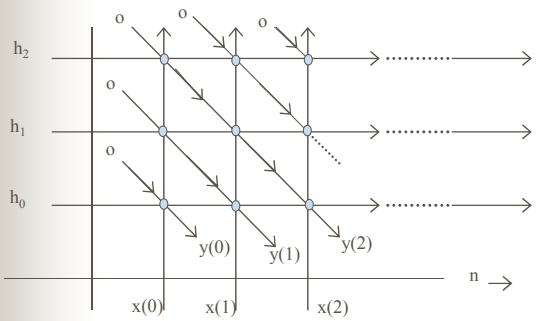


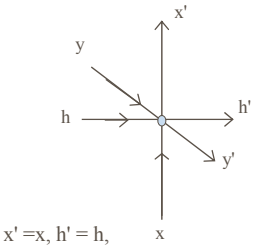
Dependence Graph

- Like DFG a combination of nodes (tasks) with directed edges which indicate precedence constraints
- Unlike DFG nodes are not reused. Instead in each iteration a new node is created
- DG has an explicit time axis, hence delays are not indicated
- Extremely modular regular arrays

FIR Filter - DG

$$y(n) = h_0\, x(n) + h_1 x(n-1) + h_2 x(n-2)$$



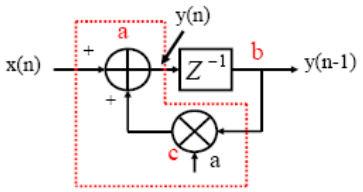


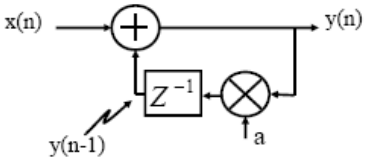
x' =x, h' = h,

y' = y + hx

Recurrence equations

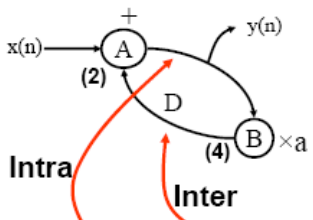
$$y(n) = a \cdot y(n-1) + x(n)$$

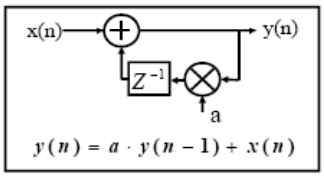




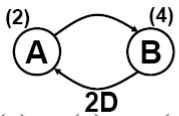
Time for one complete cycle = T_M+T_A – Iteration period
Sampling time $T_S \geq T_M+T_A$

Precedence constraints





$$A_0 \rightarrow B_0 \Rightarrow A_1 \rightarrow B_1 \Rightarrow A_2 \ldots$$

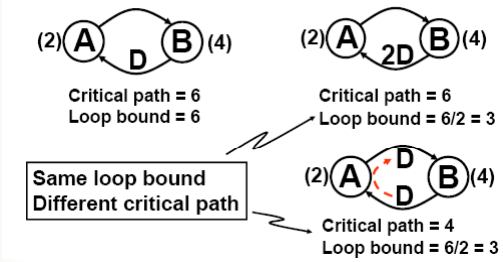


$$A_0 \rightarrow B_0 \Rightarrow A_2 \rightarrow B_2 \Rightarrow A_4 \ldots$$
$$A_1 \rightarrow B_1 \Rightarrow A_3 \rightarrow B_3 \Rightarrow A_5 \ldots$$

$$y(n) = x(n) + a\ y(n-2)$$

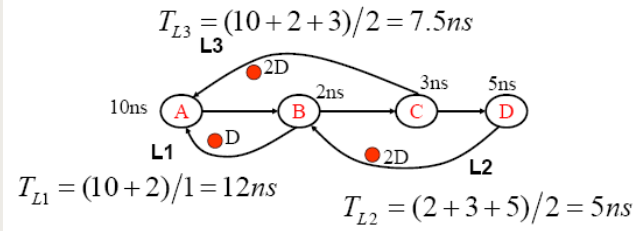
Loop Bound

$$\text{Loop bound} = \frac{\text{Total execution time on loop}}{\text{Total delays in loop}}$$



Iteration Bound

Iteration Bound = Maximum loop bound



Delay free loops cannot be computed

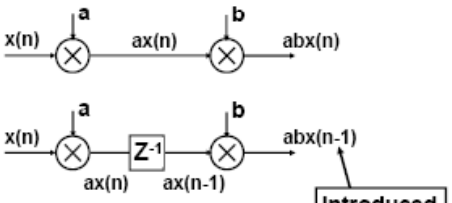
Iteration bound

- Clock period is lower bounded by the critical path computation time
- Sampling period is lower bounded by the iteration bound regardless of computational resources available
- Iteration bound
  - Depends on technology
  - Derivation of graph from equations

Pipelining and Parallelism

- Pipelining
  - Splits the logic path by introducing pipeline registers
  - leads to a reduction of the critical path but introduce latency
  - Either increases the clock speed (or sampling speed) or reduces the power consumption at same speed in a DSP system
- Parallel Processing
  - Multiple outputs are computed in parallel in a clock period
  - The effective sampling speed is increased by the level of parallelism
  - Can also be used to reduce the power consumption

Pipelining

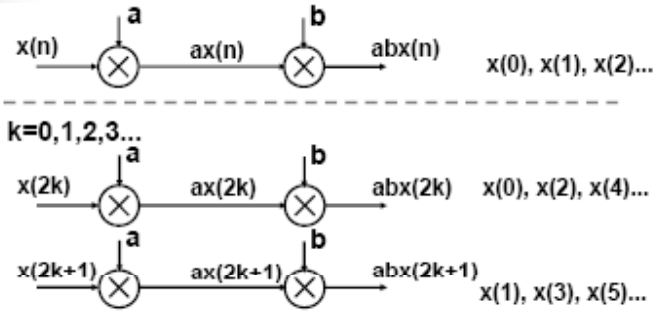


Critical path cut in half
- double clock speed

or

- lower power consumption due to reduced V_{DD}

Parallel processing



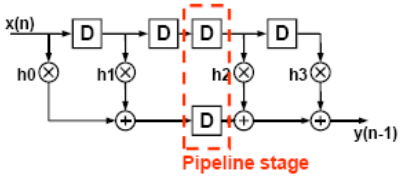
Two samples are processed in parallel
- double throughput

or

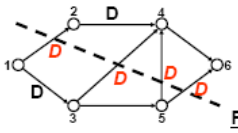
- lower power consumption due to reduced V_{DD}

Pipelining

- Cutset - A set of edges that if removed, or cut, results in two disjoint graphs.
- Feedforward Cutset - if data is moved in forward direction on all cutsets
- Pipelining – Adding delays at feedforward cutsets
- Reduces critical path delay
- Introduces latency, no change in functionality



Pipelining



$T_{critical}$ = 3⇨5⇨4⇨6

Feedforward cutset

Must place delays on all edges in the cutset

Tnode = 1

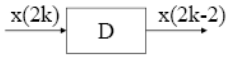
$CP_{old} = 4$

$CP_{new} = 2$

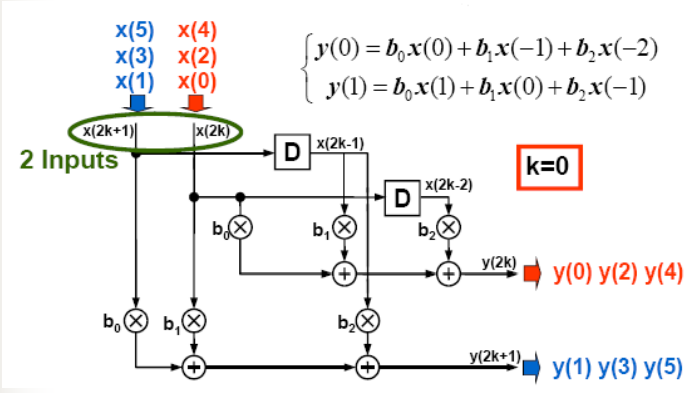
Parallel FIR filter

SISO FIR $y(n) = b_0 x(n) + b_1 x(n-1) + b_2 x(n-2)$

MIMO FIR
$$\begin{cases} y(2k) = b_0 x(2k) + b_1 x(2k-1) + b_2 x(2k-2) \\ y(2k+1) = b_0 x(2k+1) + b_1 x(2k) + b_2 x(2k-1) \end{cases}$$



Parallel 3 tap FIR



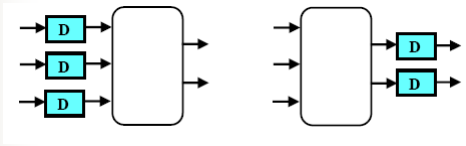
$$\begin{cases} y(0) = b_0 x(0) + b_1 x(-1) + b_2 x(-2) \\ y(1) = b_0 x(1) + b_1 x(0) + b_2 x(-1) \end{cases}$$

Pipelining & Parallelism

- Critical path unchanged in parallel system
  - $T_{clock} \geq T_M + 2T_A$
  - $T_{iter} = T_s = T_{clock}/L$
- Parallel system $T_{clock} \neq T_s$
- Pipeline system $T_{clock} = T_s$
- By combining parallel processing (block size: L) and pipelining (pipelining stage: M), the sample period can be reduced to
- $T_{iter} = T_s = T_{clock}/LM$

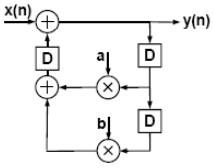
Retiming

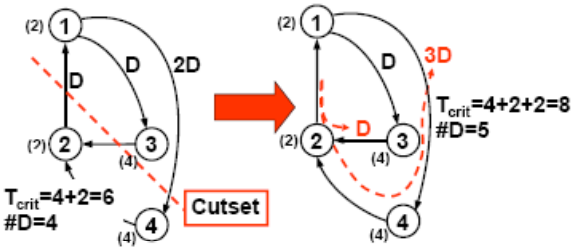
Moving delays in the system



- Modifies Critical path delay, no of registers
- Retiming does not change
  - delay in a loop
  - iteration bound

Cutset retiming

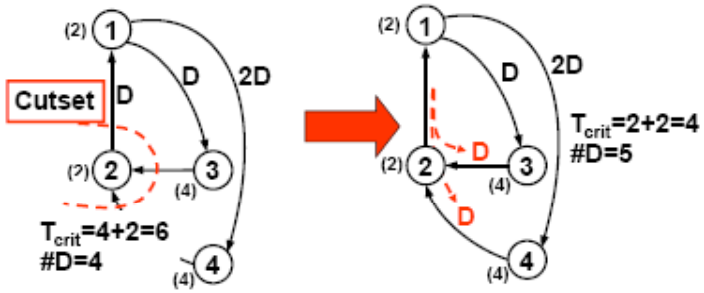




Cutset Retiming
Add delays to edges going one way and remove from ones going the other.

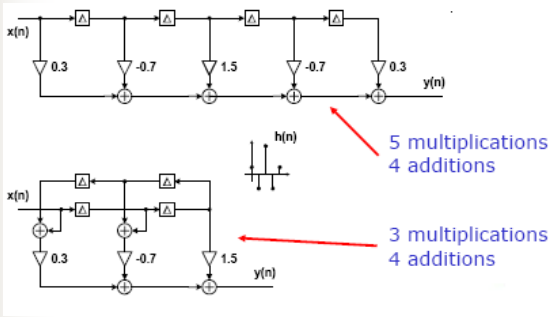
Node retiming

Cut set around a node



Algorithmic Strength reduction

- Reduces no of strong operations



Fixed coefficient multiplication

Conventional Multiplication

$Z = X \cdot Y$

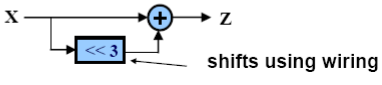
				X_3	X_2	X_1	X_0
				Y_3	Y_2	Y_1	Y_0
				$X_3 \cdot Y_0$	$X_2 \cdot Y_0$	$X_1 \cdot Y_0$	$X_0 \cdot Y_0$
			$X_3 \cdot Y_1$	$X_2 \cdot Y_1$	$X_1 \cdot Y_1$	$X_0 \cdot Y_1$	
		$X_3 \cdot Y_2$	$X_2 \cdot Y_2$	$X_1 \cdot Y_2$	$X_0 \cdot Y_2$		
	$X_3 \cdot Y_3$	$X_2 \cdot Y_3$	$X_1 \cdot Y_3$	$X_0 \cdot Y_3$			
Z_7	Z_6	Z_5	Z_4	Z_3	Z_2	Z_1	Z_0

Constant multiplication (become hardwired shifts and adds)

$Z = X \cdot (1001)_2$

				X_3	X_2	X_1	X_0
				1	0	0	1
				X_3	X_2	X_1	X_0
	X_3	X_2	X_1	X_0			
Z_7	Z_6	Z_5	Z_4	Z_3	Z_2	Z_1	Z_0

$Y = (1001)_2 = 2^3 + 2^0$



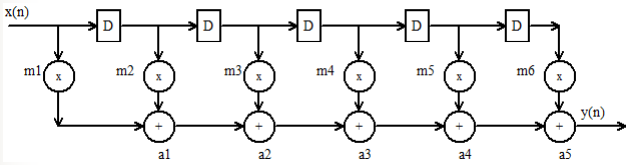
Complex multiplication

- *(a+jb)(x+jy)*
- *Can u do it with 3 multipliers and few extra adders?*

Architectural synthesis

- Given a DFG, resources
  - Allocation
    - Allocate enough resources to solve the problem
  - Binding
    - Which operation happens on which resource
  - Scheduling
    - When should each operation take place

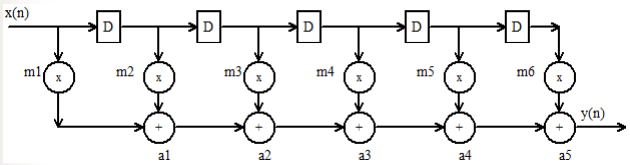
6 tap FIR Filter



Cycle	Operations	Resources
1	m1, m2,m3,m4,m5,m6	6 multi
2	a1	1 adder
3	a2	1 adder
4	a3	1 adder`
5	a4	1 adder
6	a5	1 adder

Resource Utilisation
Adder – 5/6 ~83%
Multiplier – 6/36 ~ 16.67%
m1, m2 have to be in cycle 1

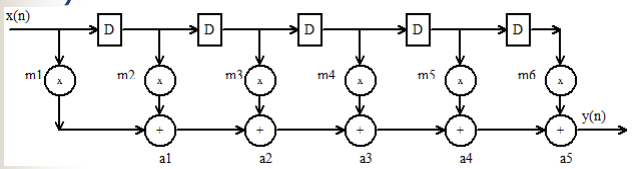
Alternate – 2 M +1 A



Cycle	Operations	Resources
1	m1, m2	2 multi
2	a1,m3,m4	1 adder+1multi
3	a2,m5,m6	1 adder+1 multi
4	a3	1 adder`
5	a4	1 adder
6	a5	1 adder

Adder – 5/6 ~83%
Multiplier – 6/12 = 50%

1M, 1A



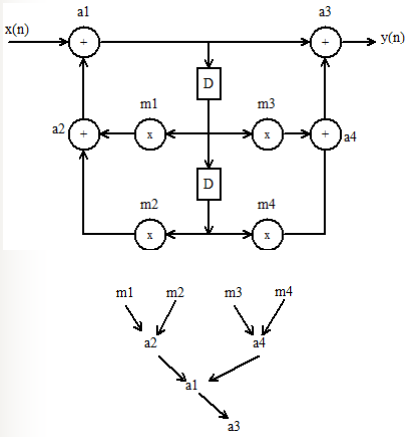
Cycle	Operations	Resources
1	m1	1 multi
2	m2	1 adder+1multi
3	m3, a1	1 adder+1 multi
4	m4, a2	1 adder+1 multi`
5	m5, a3	1 adder+1multi
6	m6,a4	1 adder+1multi
7	a5	1 adder

Adder – 5/7 ~84%
Multiplier – 5/7 = 73%

Scheduling

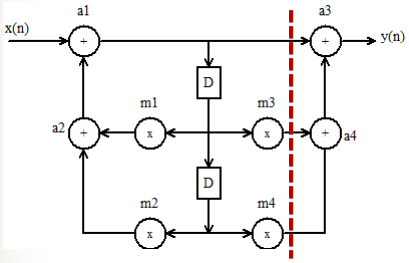
- ASAP or ALAP
- Mobility =$ALAP_{time}$ – $ASAP_{time}$ for each function
- Time bound = Total no of opns of a type / # resources
- System bound = max over all types of resources
- If resource constraint
  - (1M,1A) → max(6,5) = 6 time units
  - Pipelining / retiming
- 6M, 5 A → 2 cycles

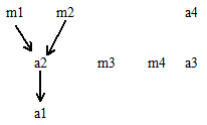
IIR filter



4 cycles
Iteration Bound = 3

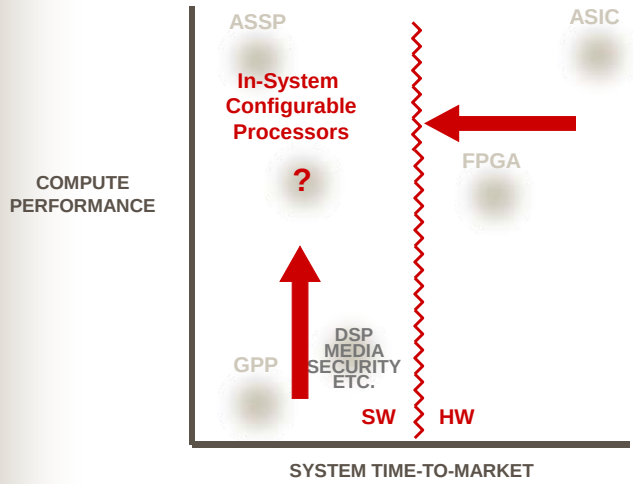
IIR filter



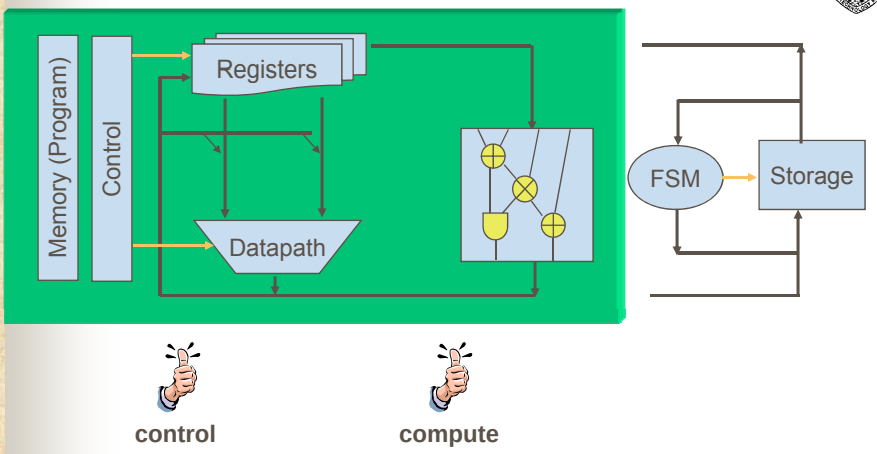


3 cycles
2A+3M

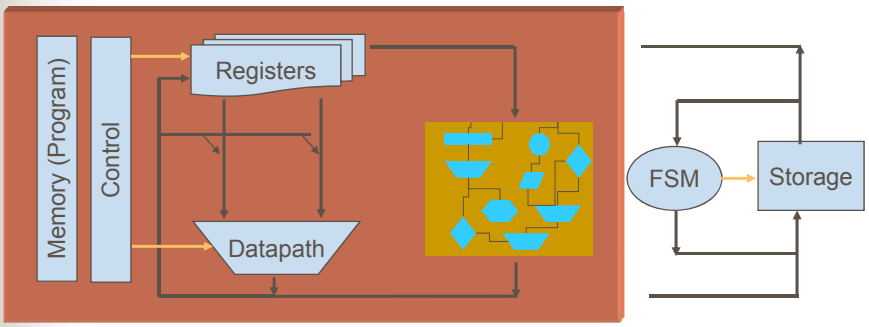
Embedded System Design Dilemma



Combining The Best of HW and SW



Software Configurable Processor Concept



Conclusion

- Overview of implementation signal processing algorithms
- Relationship between parameters influencing the choice of implementation
- Looked at some approaches to efficiently map signal processing algorithms to architecture
- Hopefully a thought on what happens at hardware level and how to optimise algorithms to suit the architecture

References

- K.K. Parhi, *VLSI Digital Signal Processing Systems*, Wiley 1999