

---

---

CS 152  
Computer Architecture and Engineering

Introduction to Architectures for Digital Signal Processing

Nov. 12, 1997

Bob Brodersen

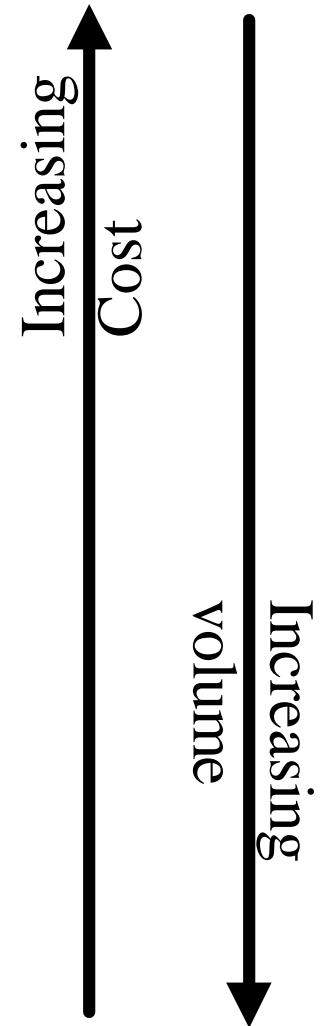
(<http://infopad.eecs.berkeley.edu>)

# Processor Applications

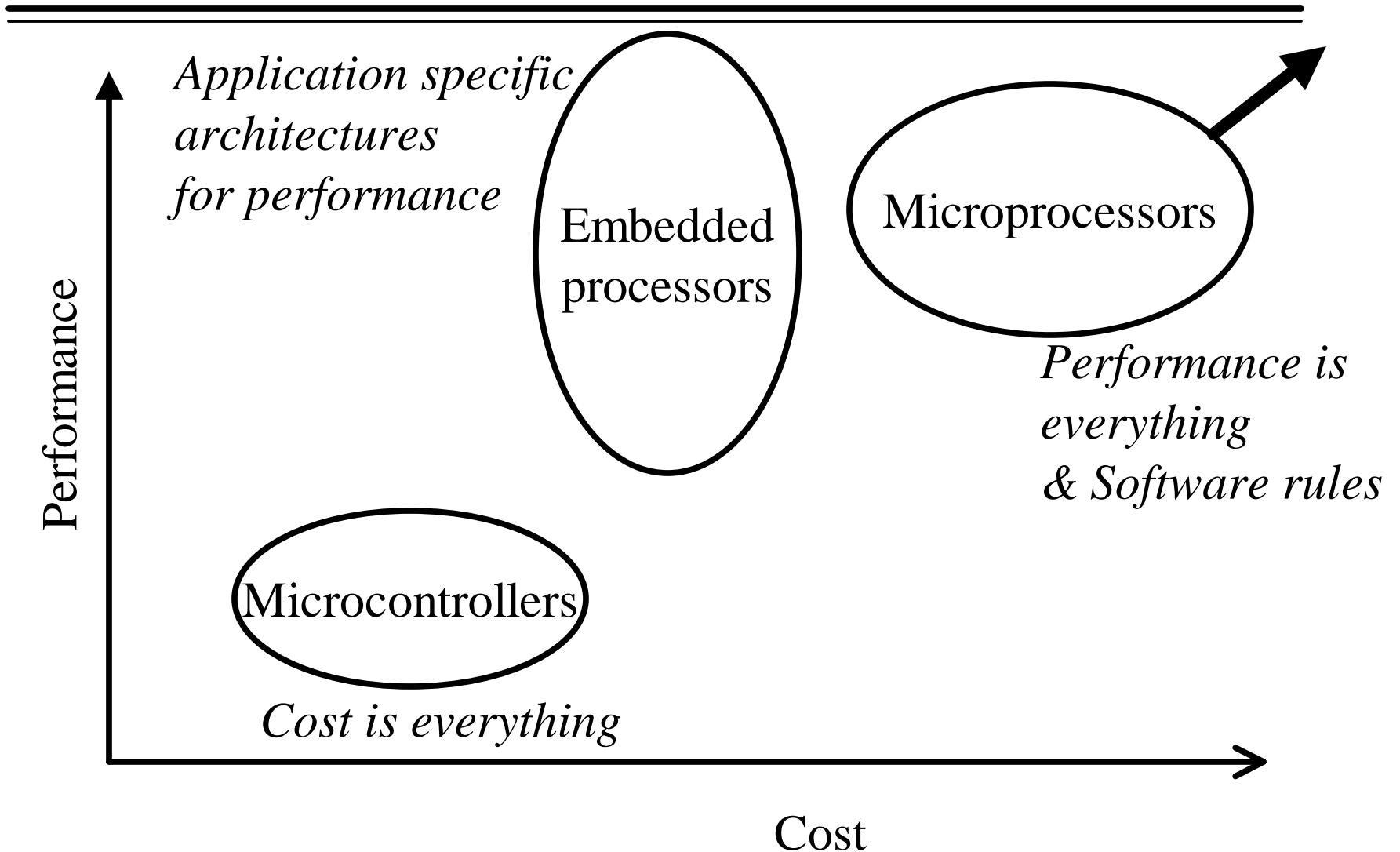
---

---

- General Purpose - high performance
  - Pentiums, Alpha's, SPARC
  - Used for general purpose software
  - Heavy weight OS - UNIX, NT
  - Workstations, PC's
- Embedded processors and processor cores
  - ARM, 486SX, Hitachi SH7000, NEC V800
  - Single program
  - Lightweight, often realtime OS
  - DSP support
  - Cellular phones, consumer electronics (e.g. CD players)
- Microcontrollers
  - Extremely cost sensitive
  - Small word size - 8 bit common
  - Highest volume processors by far
  - Automobiles, toasters, thermostats, ...

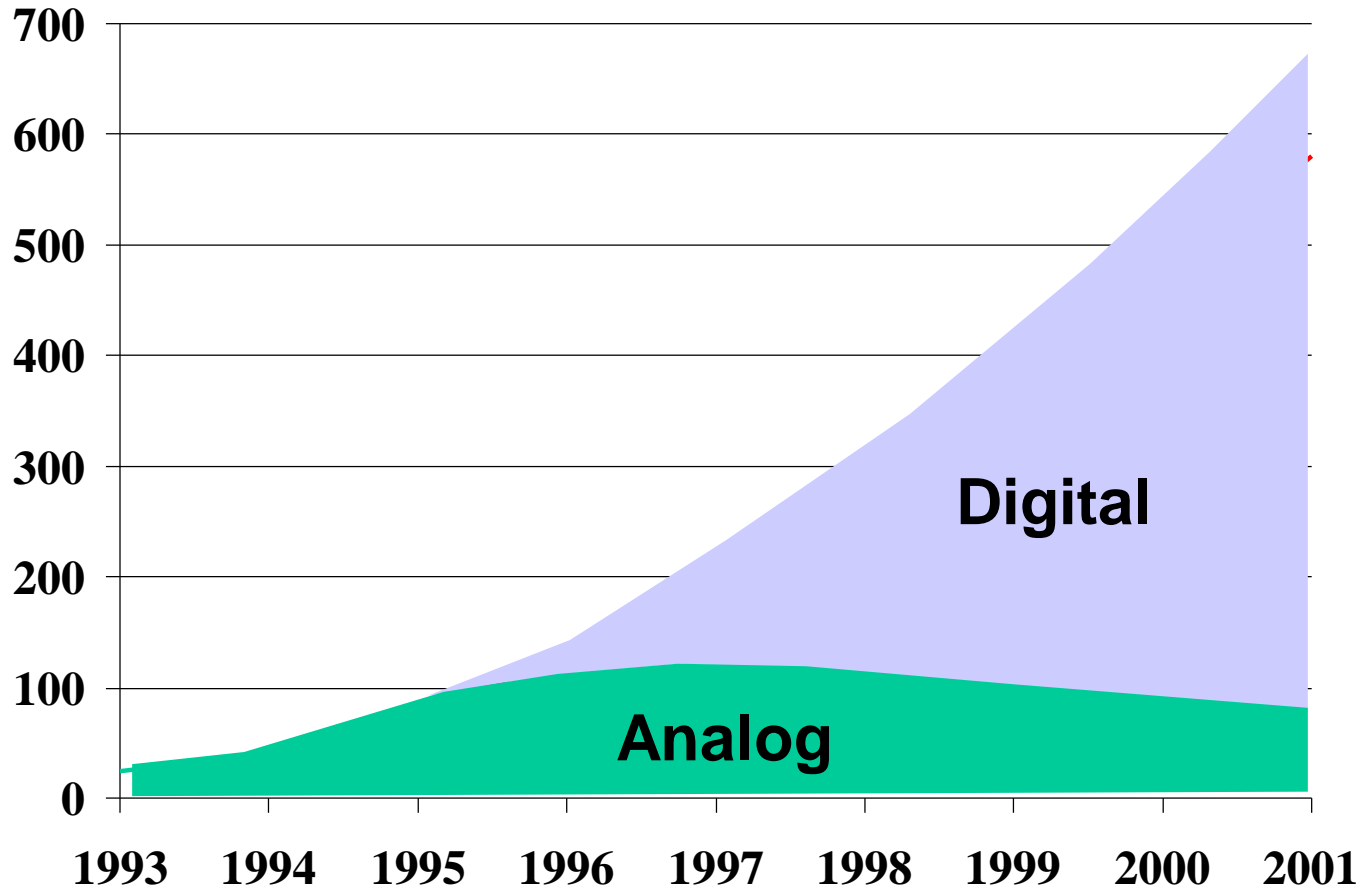


# The Processor Design Space



# World's Cellular Subscribers

Millions



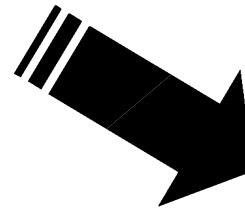
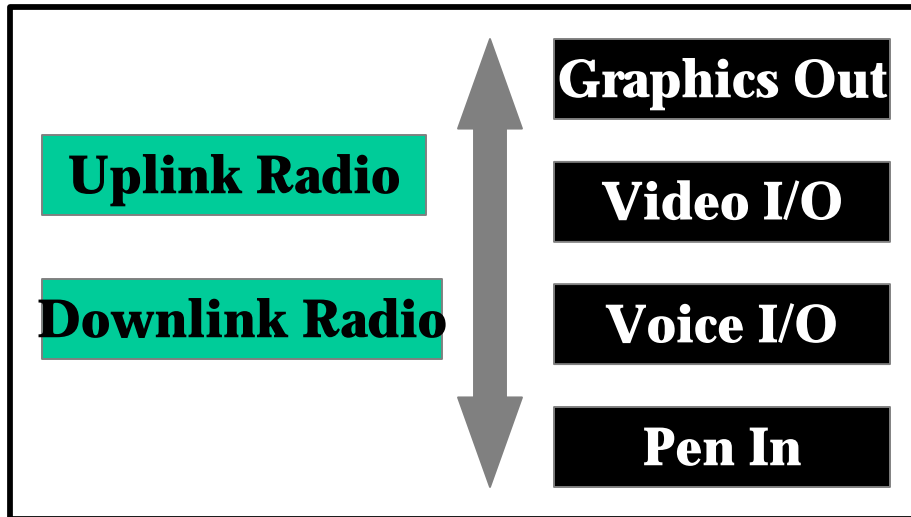
Will provide a ubiquitous infrastructure for wireless data as well as voice

Year

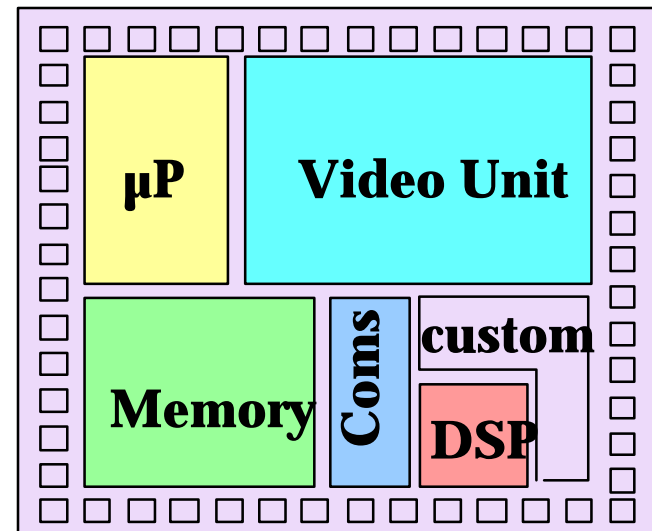


# Embedded applications

E.g. Multimedia terminal electronics



- Future chips will be a mix of processors, memory and dedicated hardware for specific algorithms and I/O



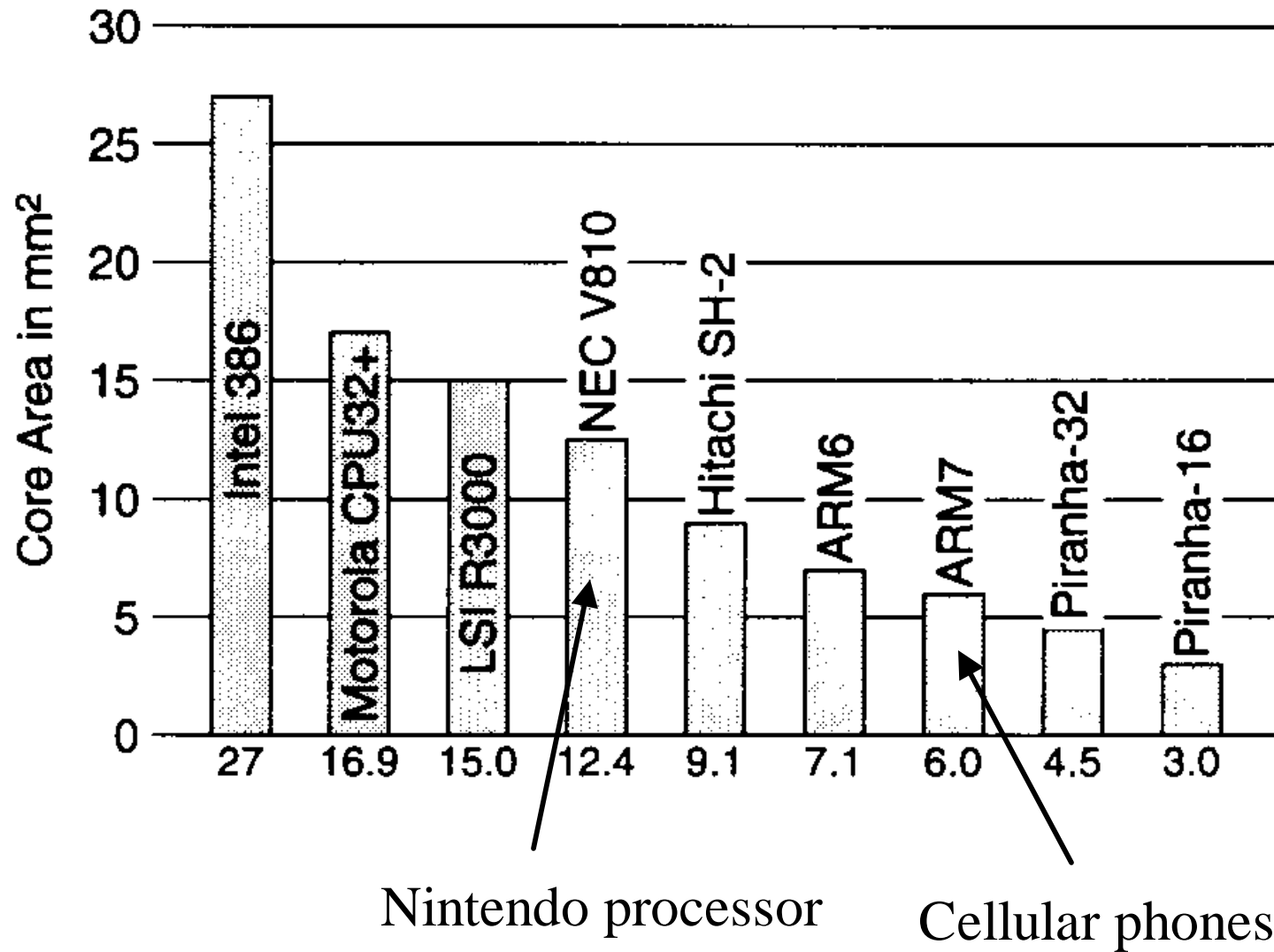
# Requirements of the Embedded Processors

---

---

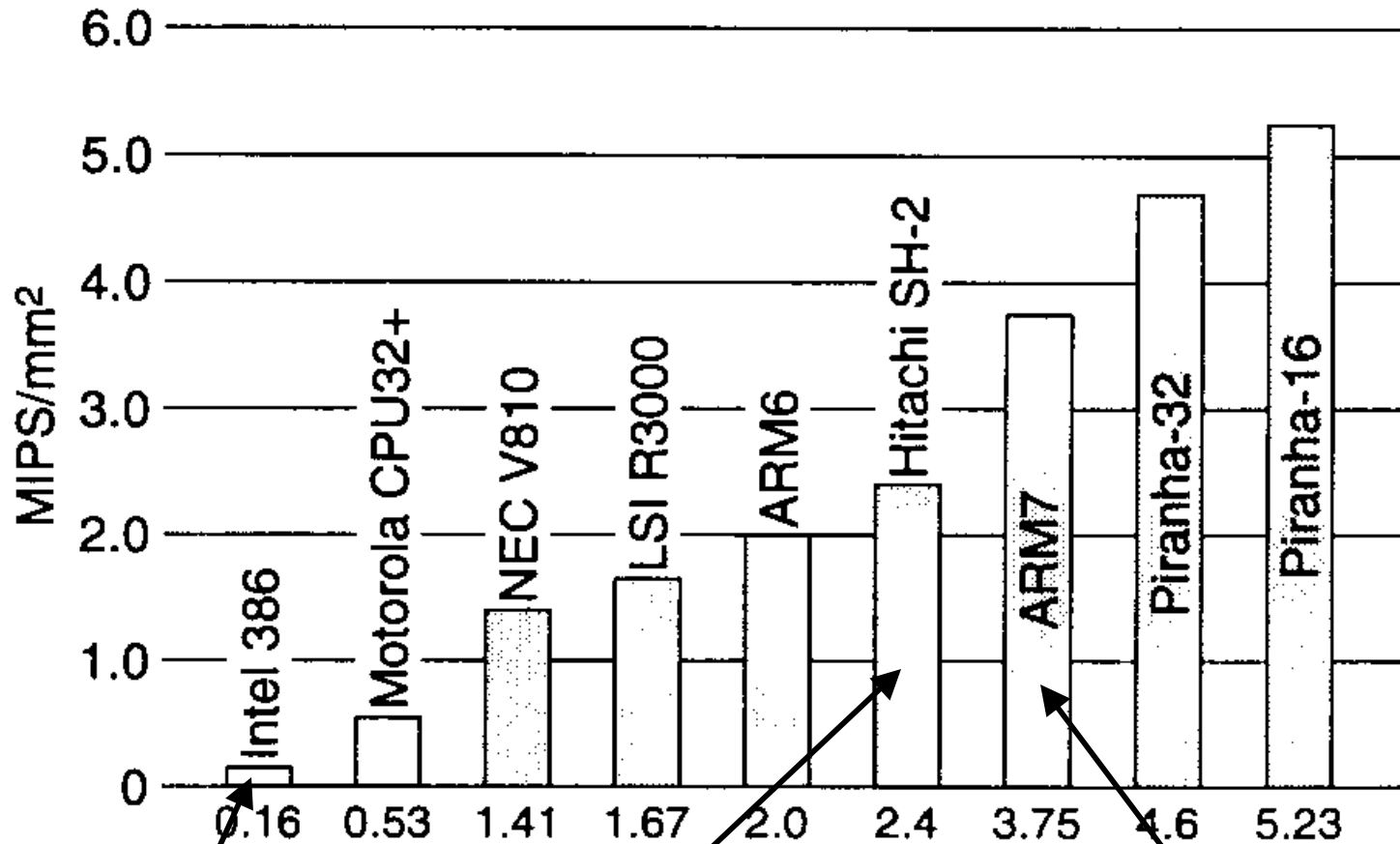
- Optimized for a single program - code often in on-chip ROM or off chip EPROM
- Minimum code size (one of the motivations initially for Java)
- Performance obtained by optimizing datapath
- Low cost
  - Lowest possible area
  - Technology behind the leading edge
  - High level of integration of peripherals (reduces system cost)
- Fast time to market
  - Compatible architectures (e.g. ARM) allows reuseable code
  - Customizable core
- Low power if application requires portability

# Area of processor cores = Cost



# Another figure of merit

## Computation per unit area



???

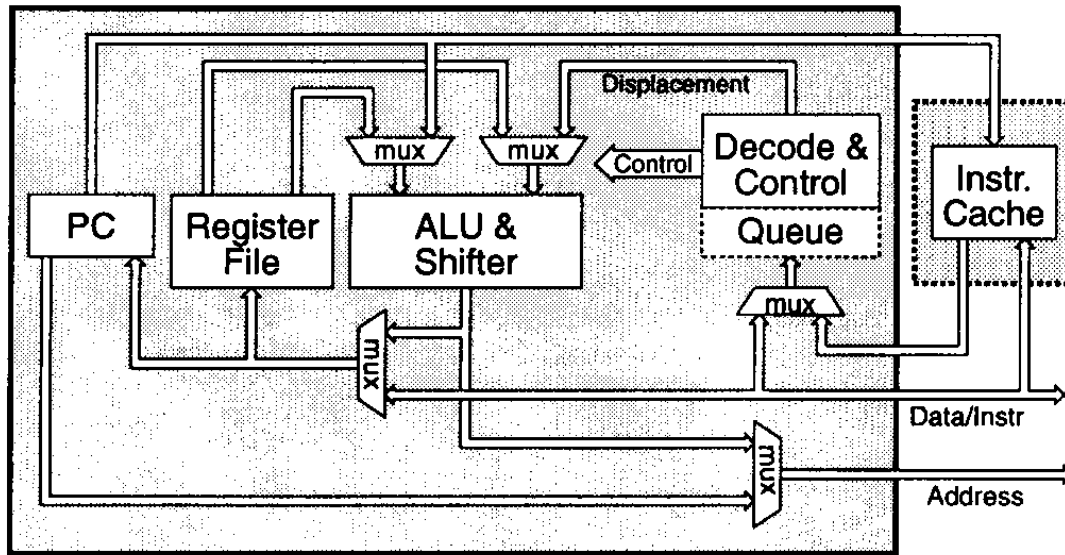
Nintendo processor

Cellular phones

# National Semiconductor - Embedded Processor Family

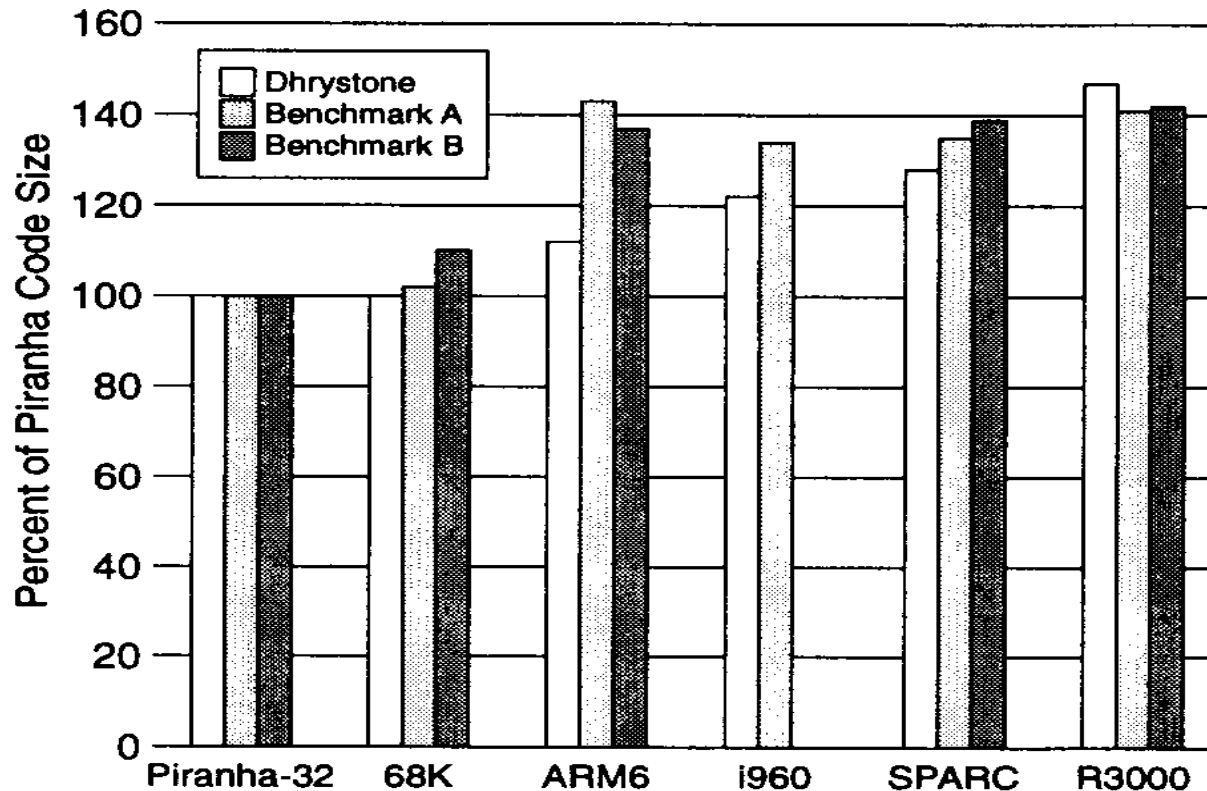
---

---



- Simple architecture
- 3 stage pipeline - fetch - decode - execute
- Minimum power and size
  - Short pipeline avoids branch prediction and bypass
  - Versions range from 8-64 bit - choose minimum that meets requirements

# Code size

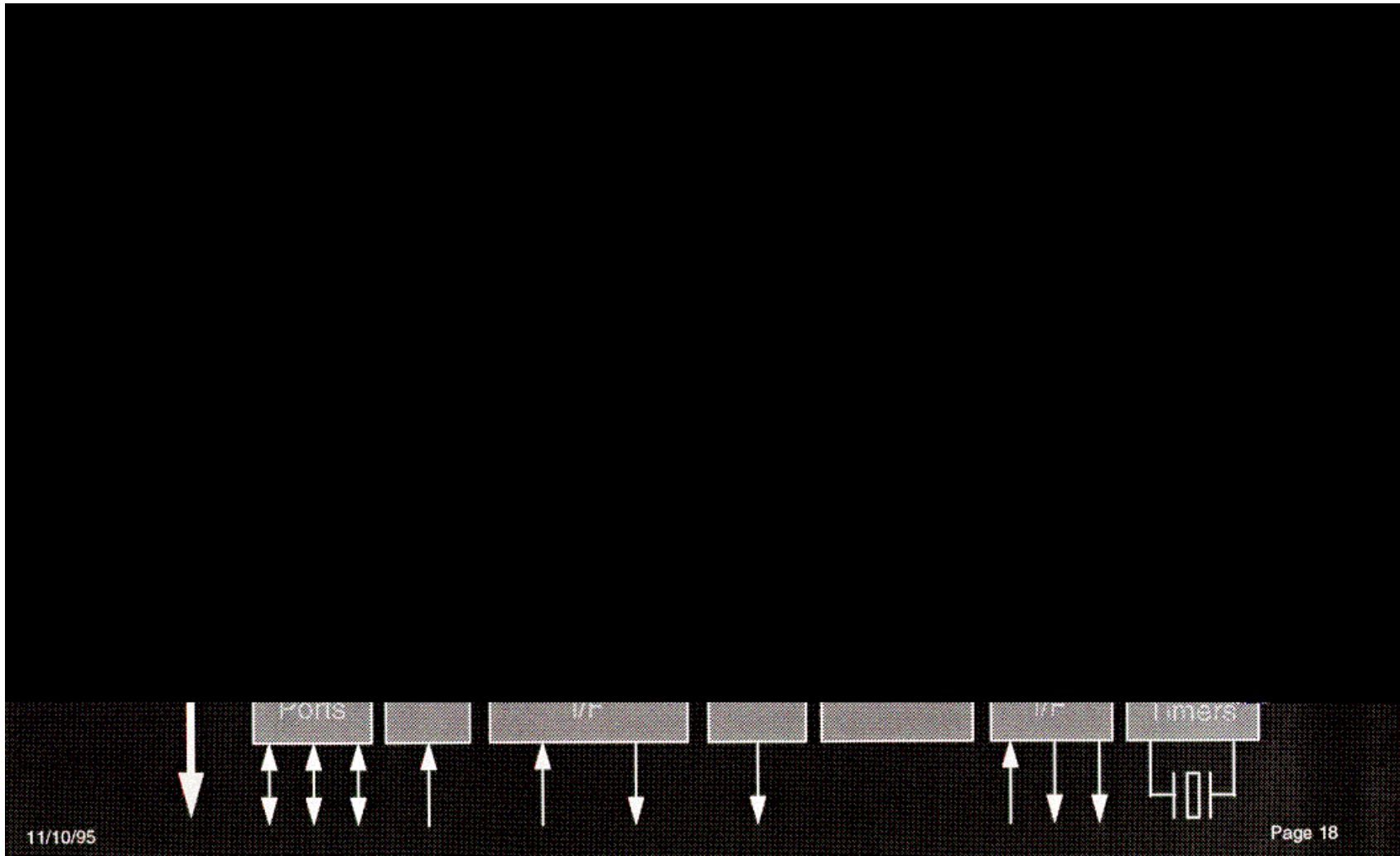


- If a majority of the chip is the program stored in ROM, then code size is a critical issue
- The Piranha has 3 sized instructions - basic 2 byte, and 2 byte plus 16 or 32 bit immediate

# Example application (single chip system)

---

---



## The DSP Module (DSPM)

---

---

- Vector instructions directly supported
- Pipelined datapath supports single cycle: Multiply, Add, Shift, Load/Store and Pointer adjustment
- Operates in parallel to processor core
- Saturation, overflow and rounding for ALU operations
- Automatic support for cyclic buffers (modulo arithmetic)

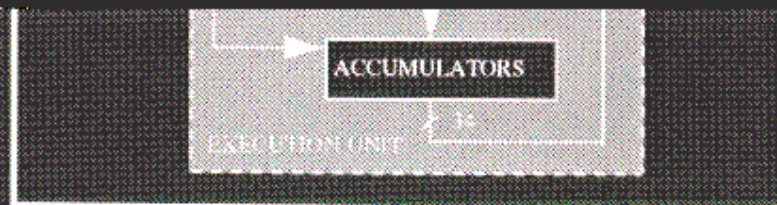
# The National DSP Module Architecture

Zero overhead  
repeat

Three simultaneous  
addresses

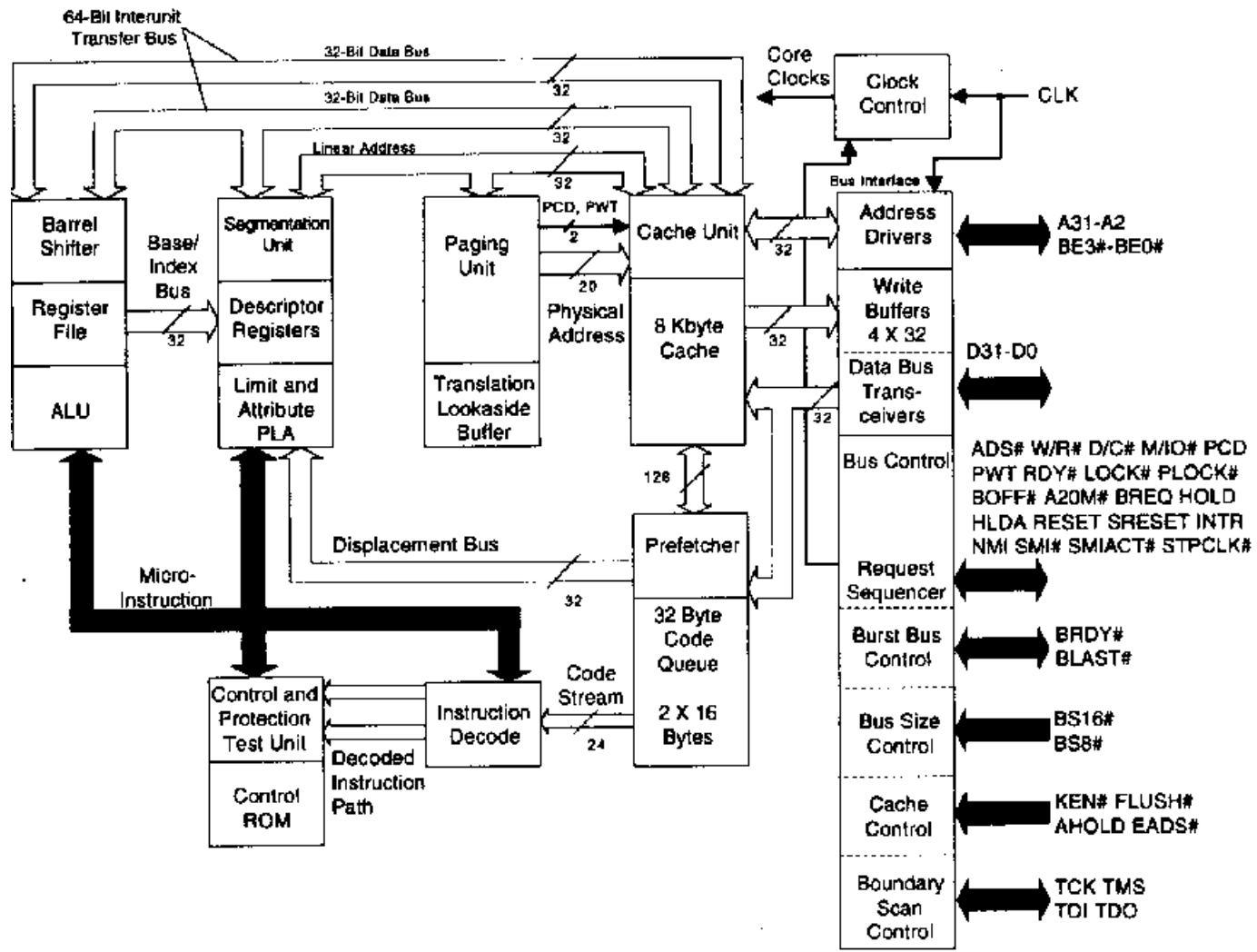
X Y Z

Single cycle MAC  
support is typical for  
DSP acceleration



# The 486 “Embedded Processor”

## Look familiar???



## The “Embedded” Features of the 486 GX

---

---

- Said to be designed “for embedded battery-operated and hand-held applications” (???)
- Fully static design (clock can stop and all state is kept)
- “Auto Clock Freeze” stops circuits which are not being used in a given instruction (gated clocks)
- Stop Clock (60  $\mu$ W), Stop Grant - clock runs but no program execution (40-85 mW)
- Split power supply - 2.0-3.3 Volt core, 3.3V. I/O,



$$\text{Power} = C V^2 f_{\text{clock}}$$

Table 17. Active I<sub>CC</sub> Values  
 T<sub>CASE</sub>=0 °C to +85 °C

| Symbol | Parameter | Frequency | Supply Voltage                | Typical I <sub>CC</sub> | Max. I <sub>CC</sub> | Power  |
|--------|-----------|-----------|-------------------------------|-------------------------|----------------------|--------|
|        |           | 16 MHz    | V <sub>CC</sub> = 2.0 ± 0.2 V | 65 mA                   | 105 mA               | 130 mW |
|        |           |           |                               |                         |                      | 350 mW |
|        |           |           |                               |                         |                      | 100 mW |
|        |           |           |                               |                         |                      | 430 mW |
|        |           |           |                               |                         |                      | 290 mW |
|        |           |           |                               |                         |                      | 540 mW |
|        |           |           |                               |                         |                      | 490 mW |
|        |           |           |                               |                         |                      | 17 mW  |
|        |           |           |                               |                         |                      | 20 mW  |
|        |           |           |                               |                         |                      | 23 mW  |
|        |           |           |                               |                         |                      | 30 mW  |

Note the clock rates

# Characterizing programs for their energy consumption

---

---

**Process Subframe 330mW**

**ComputeLag 107mW**

**IFilterCodebook 63mW**

**QuantizeGains 46mW**

**CodebookSearch 44mW**

**UpdateFilterState 8mW**

**OrthogonalizeCodebook 6mW**

**ComputeWeightedInput 22mW**

**ThetaToCodeword 8mW**

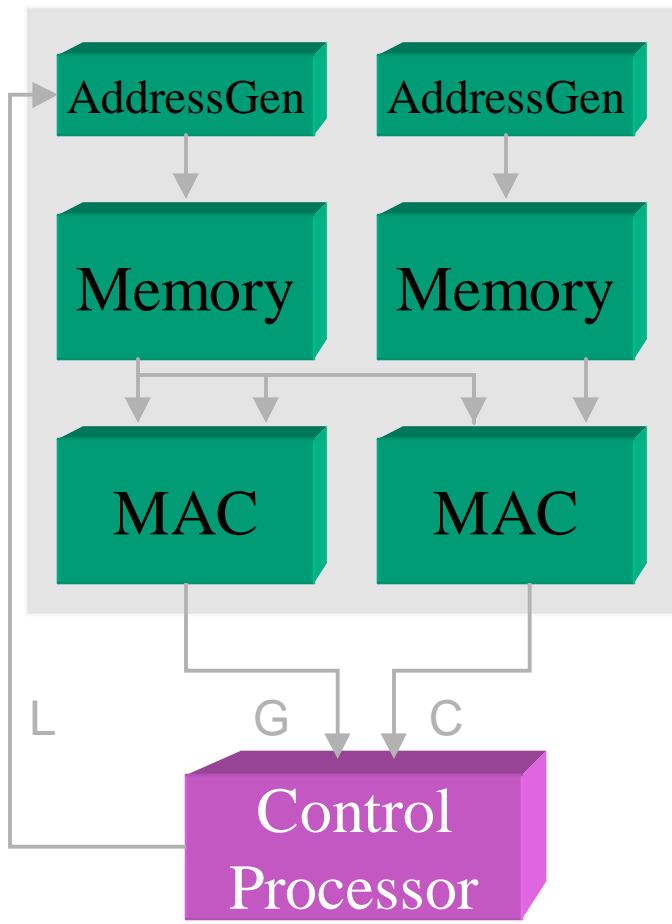
```
ComputeLag(...)  
{  
    R=dotprod(res,res);  
    for (lag=0..127)  
    {  
        lp=getLT(lt);  
        G = dotprod(lp, lp);  
    }  
}
```

Top four functions account for 90 % of the power  
65% of power dissipation in dot-vector products  
(data obtained from profiling of C++-code, weighted with  
estimated instruction energy costs)

# An architecture optimized for multiply-accumulate

---

---



## Energy/Flexibility Tradeoff's

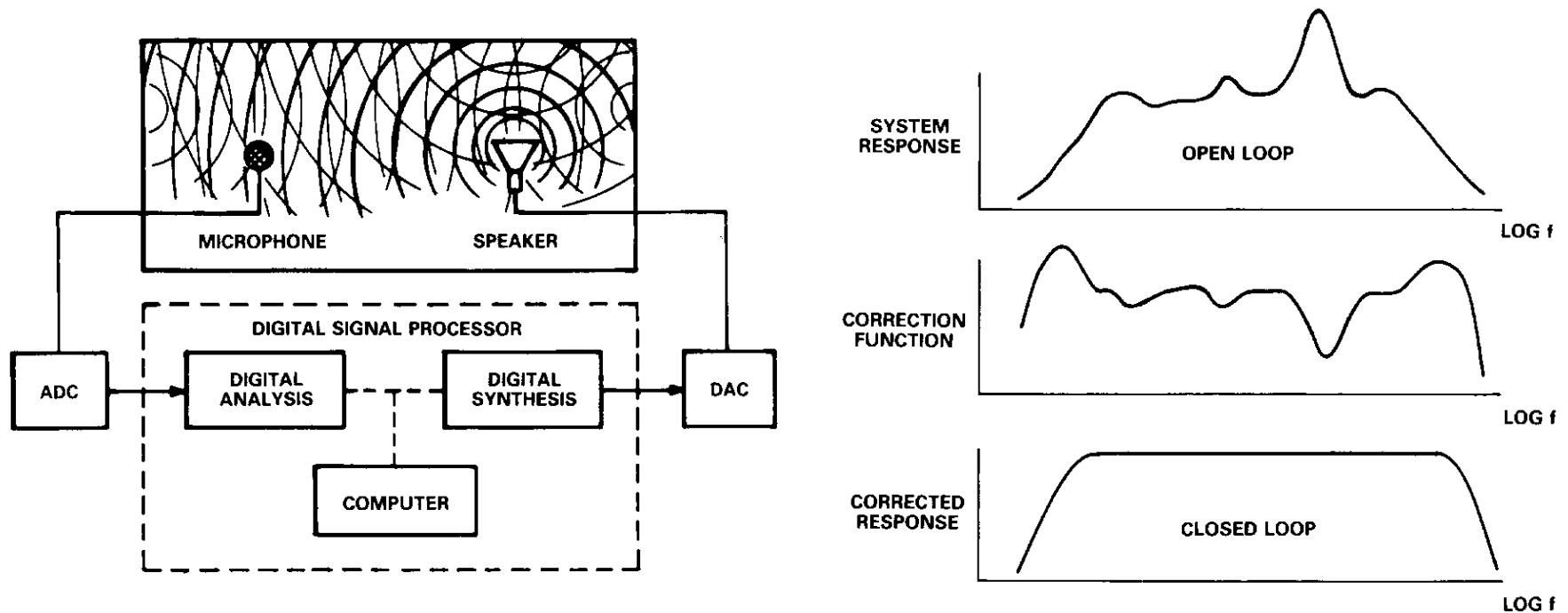
Arm 6 core (5V, 20 MHz):  
.02 MIPS/mW

ZSP DSP Superscaler (3V, 200 MHz)  
.3 MOPS/mW

Reconfigurable Dot-Vector Processor  
(1.5V, 30 MHz)  
5.9 MIPS/mW

\* MOPS = millions of operations/sec  
= millions of MACS/sec

# DSP Application - equalization



- The audio data streams from the source (computer) through the digital analysis and synthesis
- Hard realtime requirement - the processing must be done at the sample rate

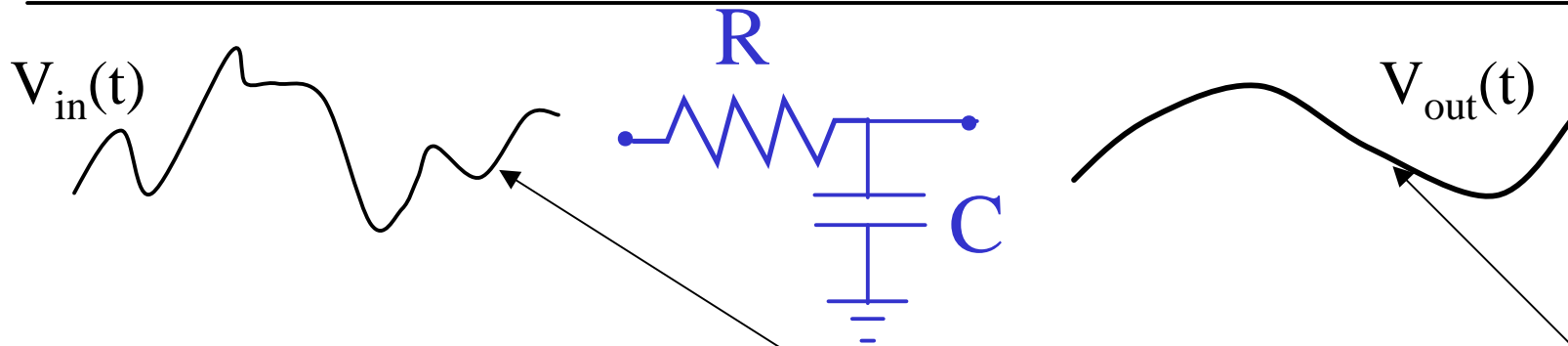
# Common DSP algorithms and applications

---

---

- Applications
  - Instrumentation and measurement
  - Communications
  - Audio and video processing
  - Graphics, image enhancement, 3-D rendering
  - Navigation, radar, GPS
  - Control - robotics, machine vision, guidance
- Algorithms
  - Frequency domain filtering - FIR and IIR
  - Frequency-time transformations - FFT
  - Correlation

# Sampled data processing



This RC low pass filter takes this time waveform (signal) and turns it into this filtered version

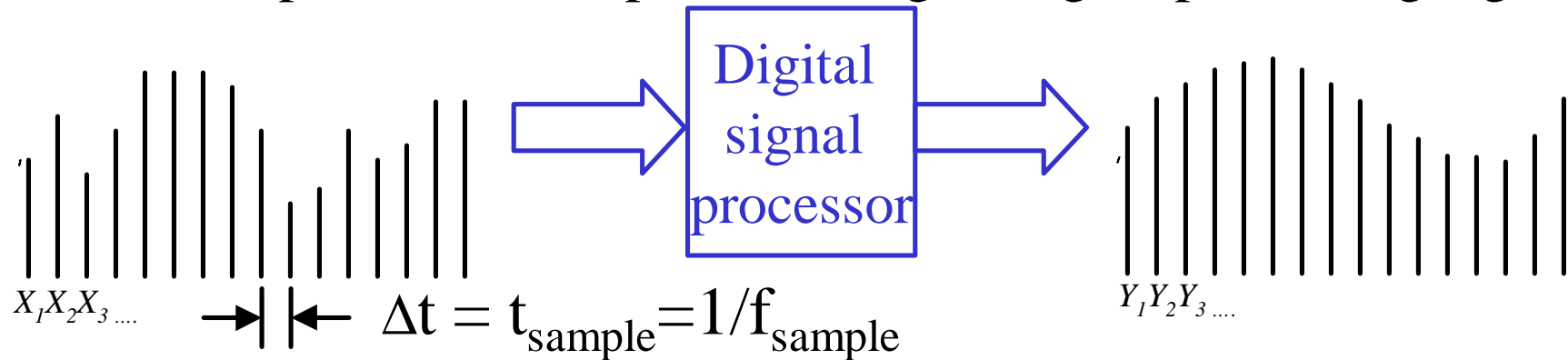
This analog circuit really is just an solution of the differential equation calculated using the physics of electric fields and currents:

$$RC \frac{dV_{out}}{dt} + V_{out}(t) = V_{in}(t)$$

To implement this digitally we need to convert this expression to discrete time. First we need to convert from a continuous time representation of the signal to discrete time sequences:  $V_{out}(t) \Rightarrow Y_1 Y_2 Y_3 \dots Y_n$  and  $V_{in}(t) \Rightarrow X_1 X_2 X_3 \dots X_n$

# Discrete time representation

The sampled version of  $V_{in}(t)$  is a sequence of numbers 6,8,4,12, ....  
This then provides the input to the digital signal processing algorithm



Now what is the processing that goes on to implement the filtering?

Using a discrete approximation to the derivative we obtain the discrete time equivalent of the continuous time differential equation:

$$RC \left( \frac{Y_n - Y_{n-1}}{\Delta t} \right) + Y_{n-1} = X_{n-1}$$

# A computational structure

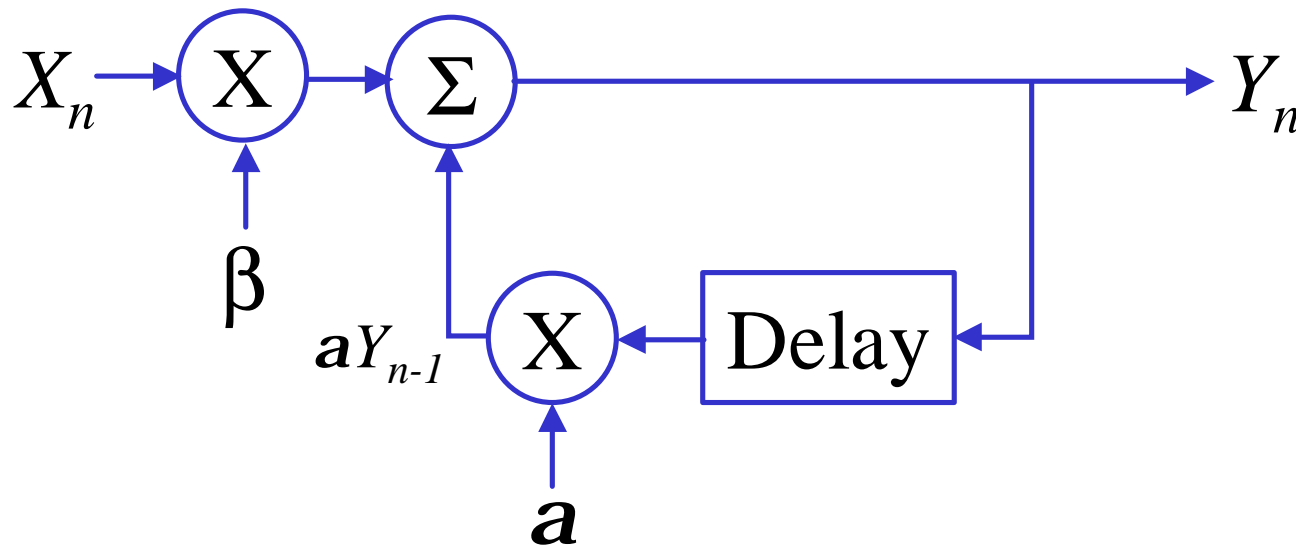
---

---

This can be rewritten as:

$$Y_n = \left(1 - \frac{\Delta t}{RC}\right)Y_{n-1} + \left(\frac{\Delta t}{RC}\right)X_n = \mathbf{a}Y_{n-1} + \mathbf{b}X_n$$

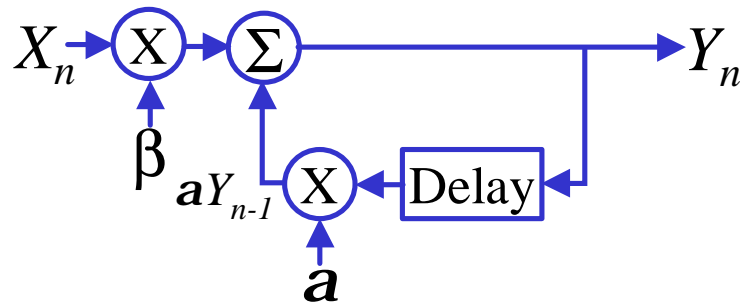
since the new sample is only a function of past samples it can be computed using the following procedure:



# Direct mapping architecture

---

---



- These calculations need to be finished after every sample period, since  $Y_n$  depends on  $Y_{n-1}$  and new data is continuously coming => hard real time requirement
- In each sample period there are 2 multiply adds and one accumulate.
- We could directly map this structure into hardware and then the delay becomes a pipeline register and we would need two multipliers and an adder - this is the most direct approach, almost no control, but also no flexibility

# Filter structures

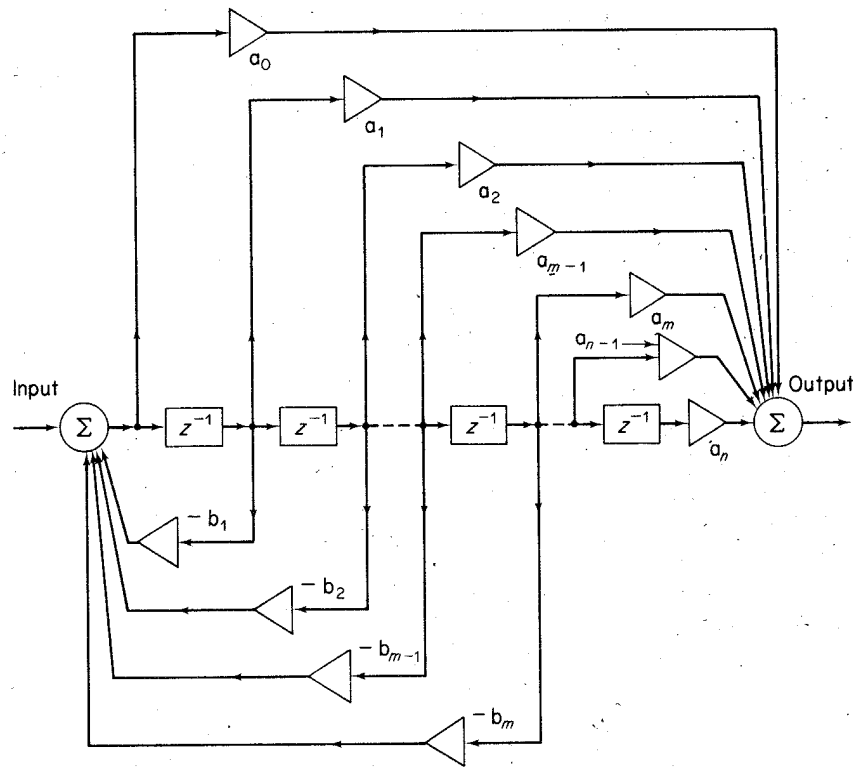


Figure 3.5 A canonic realization structure

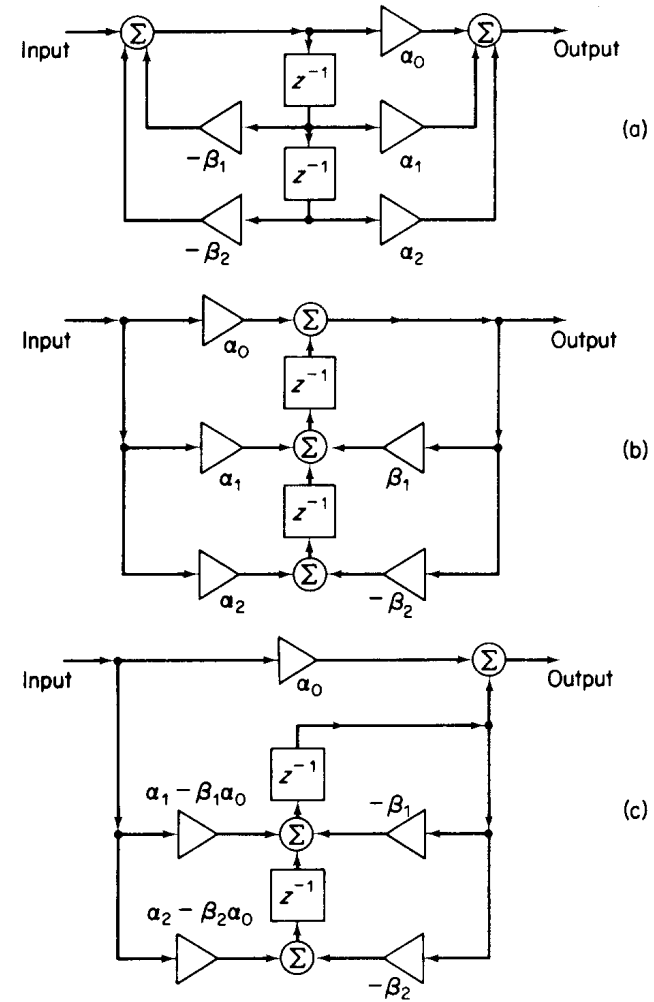
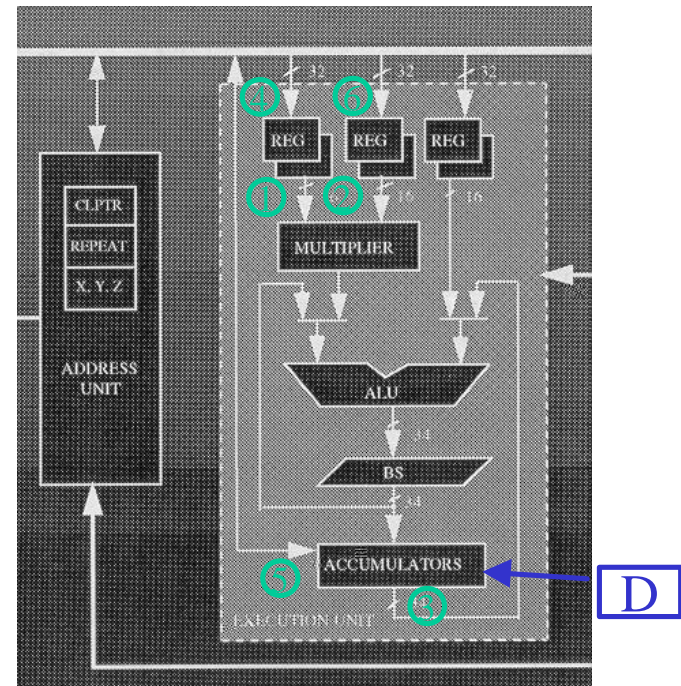
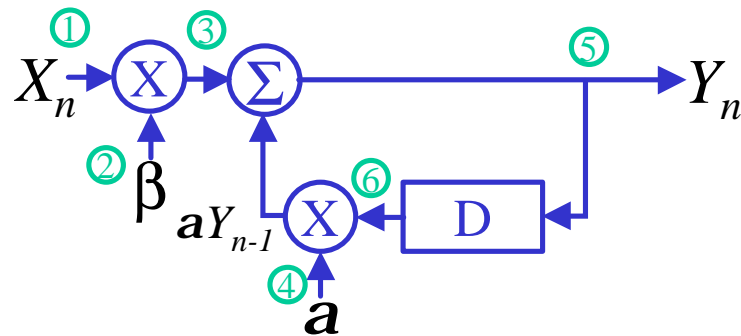


Figure 3.7 Some canonic realizations of the biquadratic section

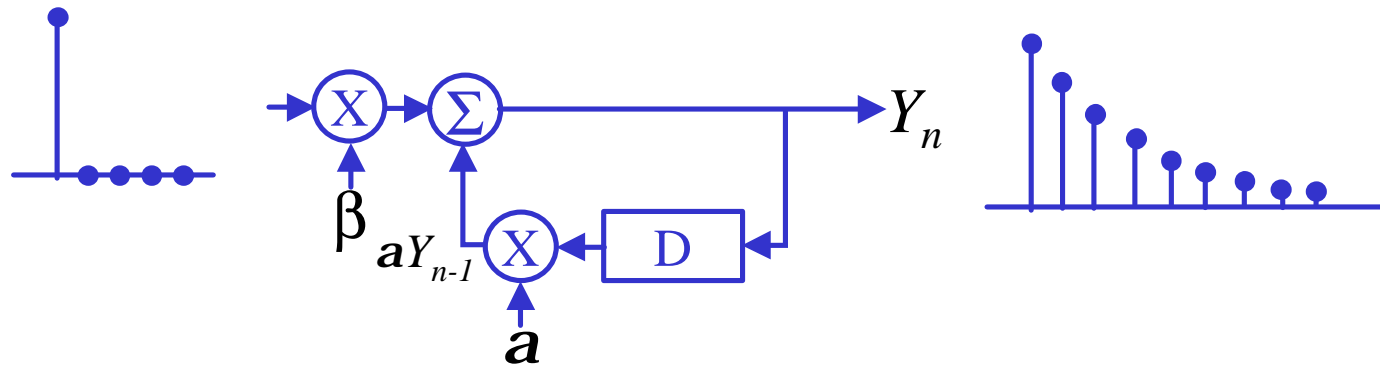
# Mapping of the filter onto a DSP execution unit



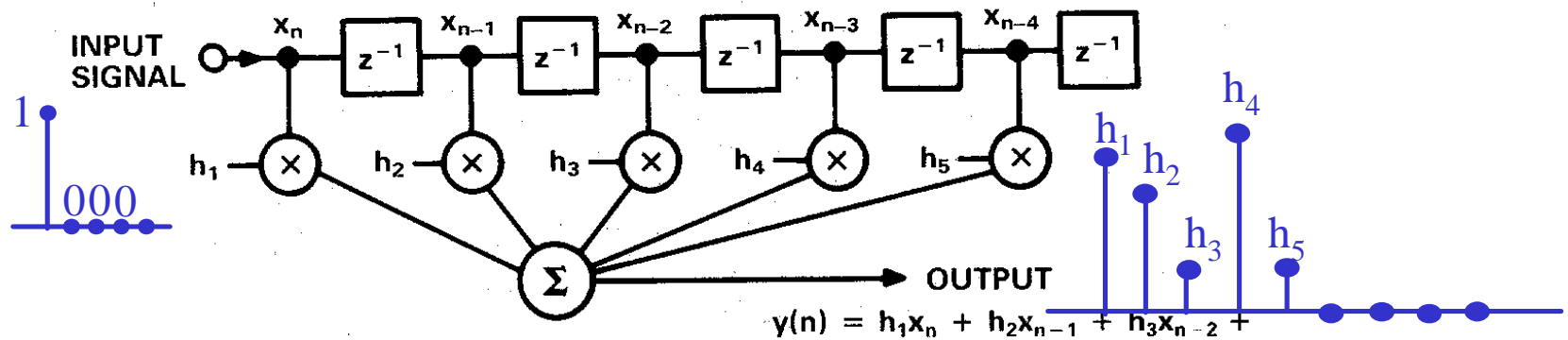
- The critical hardware unit in a DSP is the multiplier - much of the architecture is organized around allowing use of the multiplier on every cycle
- This means providing two operands on every cycle, through multiple data and address busses, multiple address units and local accumulator feedback

# IIR and FIR filters

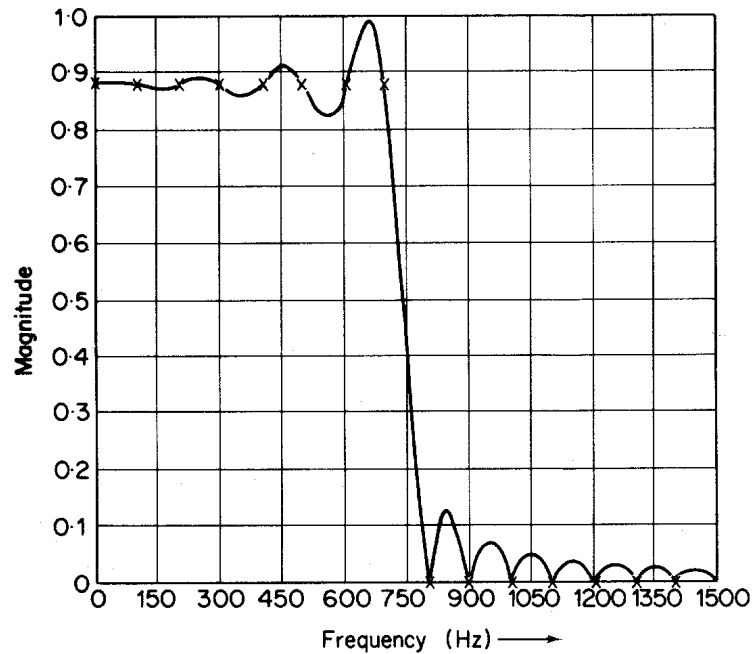
- Infinite Impulse Response (IIR) filter - has a feedback loop and the response to an impulse goes on forever



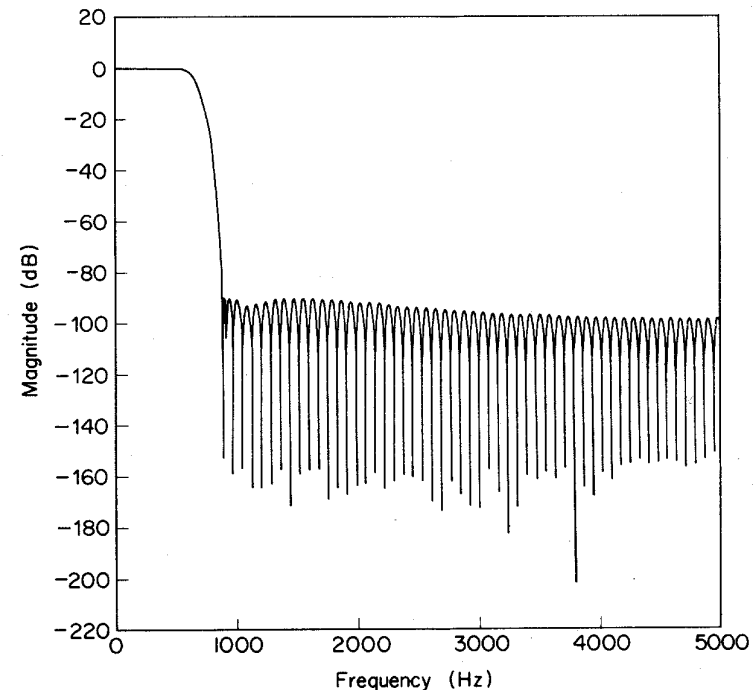
- The impulse response completely characterizes the filter response, so a more direct (purely digital) approach is the finite impulse response filter or FIR.



# FIR filter frequency response



15 stages



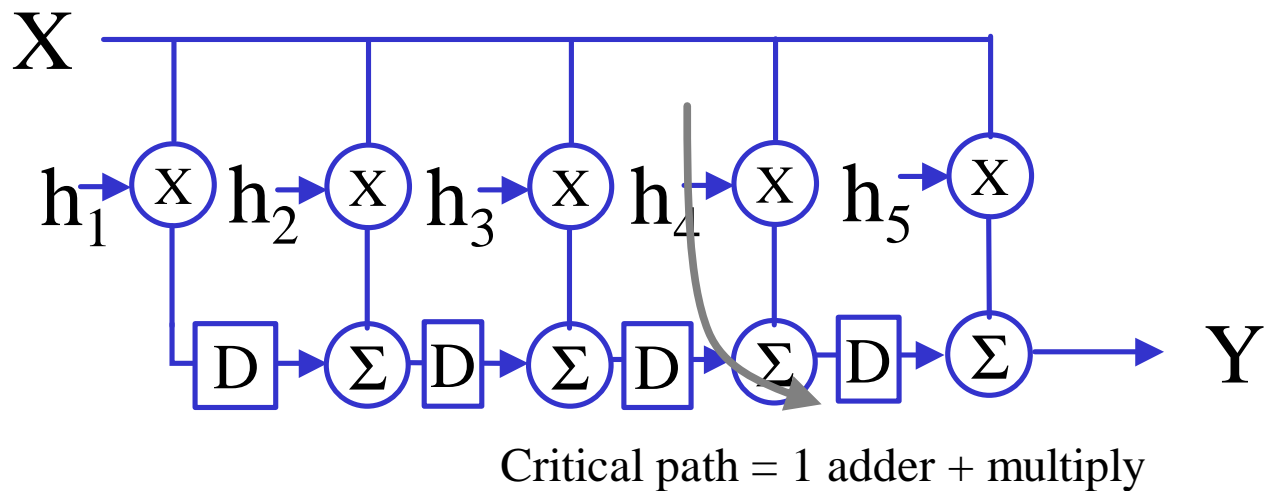
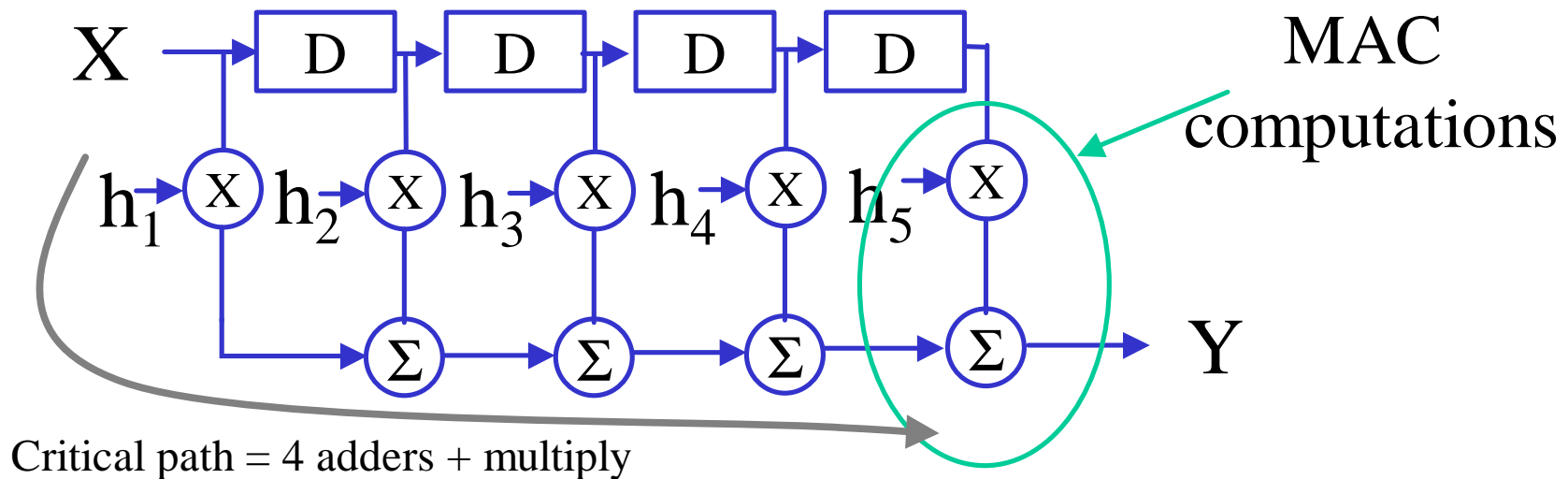
128 stages

- FIR filters are a very general structure and form the base of much more sophisticated processing, e.g. adaptive filters which make possible 56 kbit modems

# Transformations result in different critical paths for direct map architectures

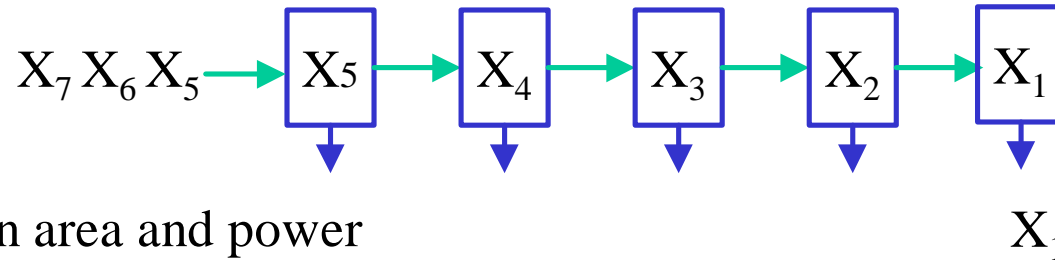
---

---



# Delay Lines

- Shift register



- Very inefficient in area and power

- since shift register cells are much larger than RAM
- ALL data must move every cycle

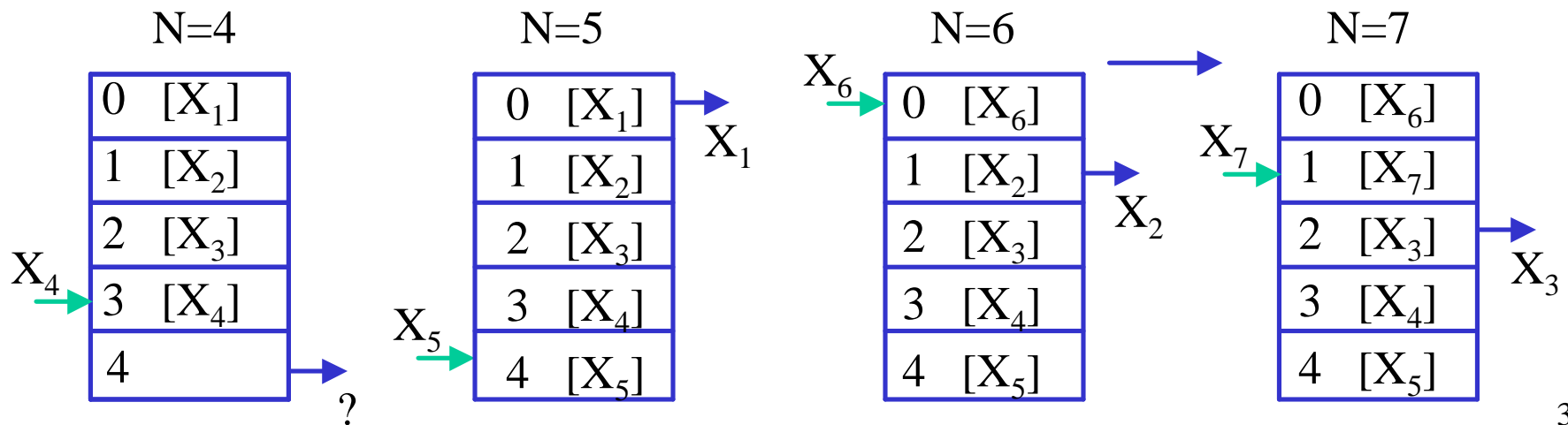
- Delay using circular buffers - use of modulo arithmetic

$N = \text{time index}$



Write address =  $(N \text{ modulo } 5) - 1$

Read address =  $N \text{ modulo } 5$

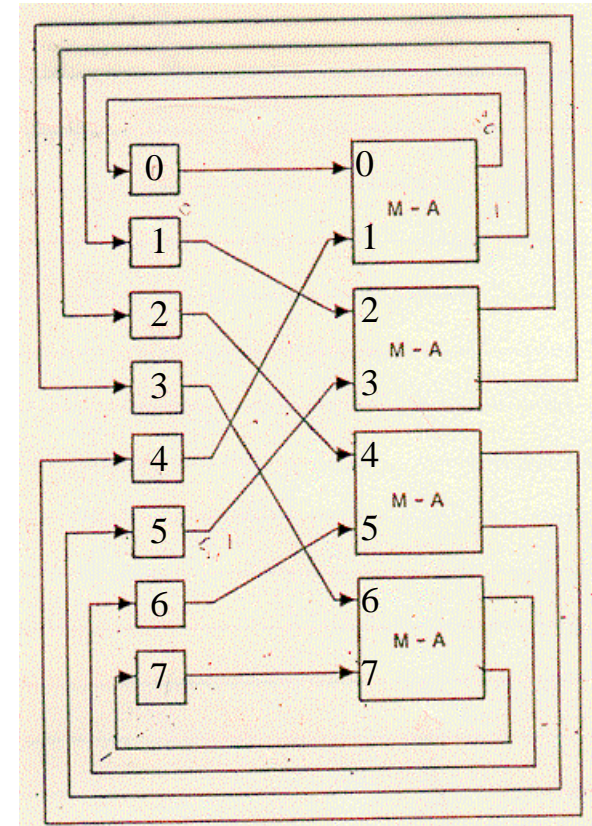
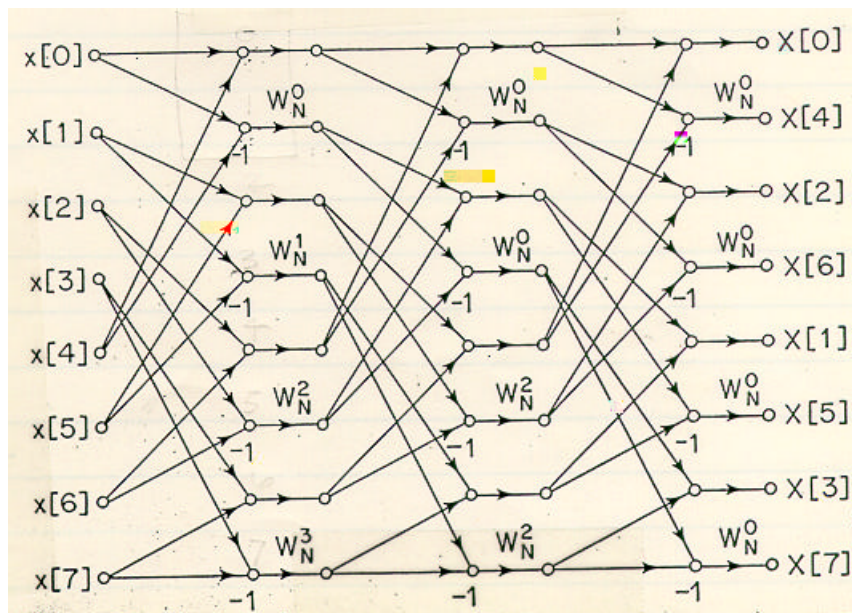
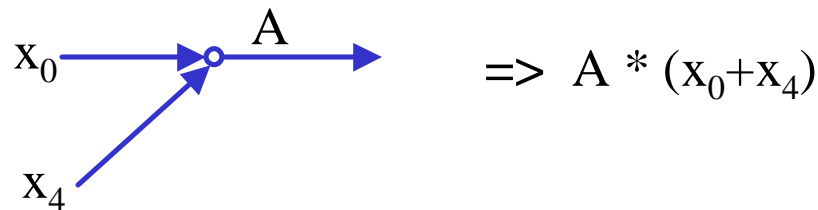


# FFT support

- “Flow diagram” of FFT algorithm - again based on multiply adds

Bit reversed addressing - what is the pattern?

000 000  
 001 010  
 010 100  
 011 110

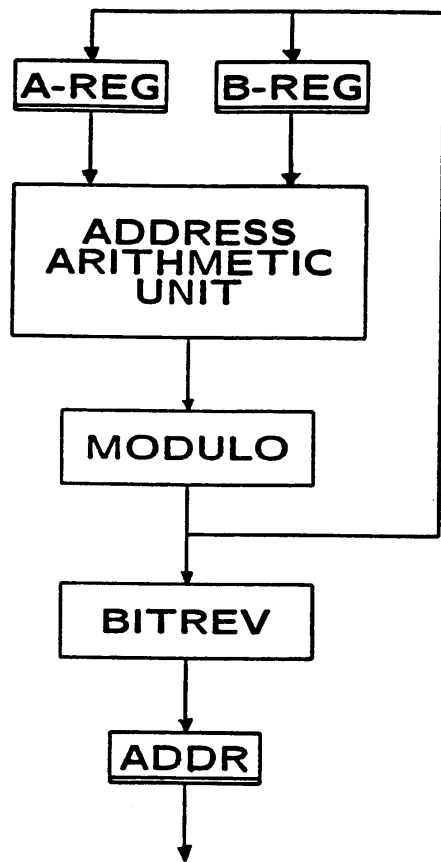


# Address calculation unit for DSP

---

---

## ADDRESS CALCULATION UNIT

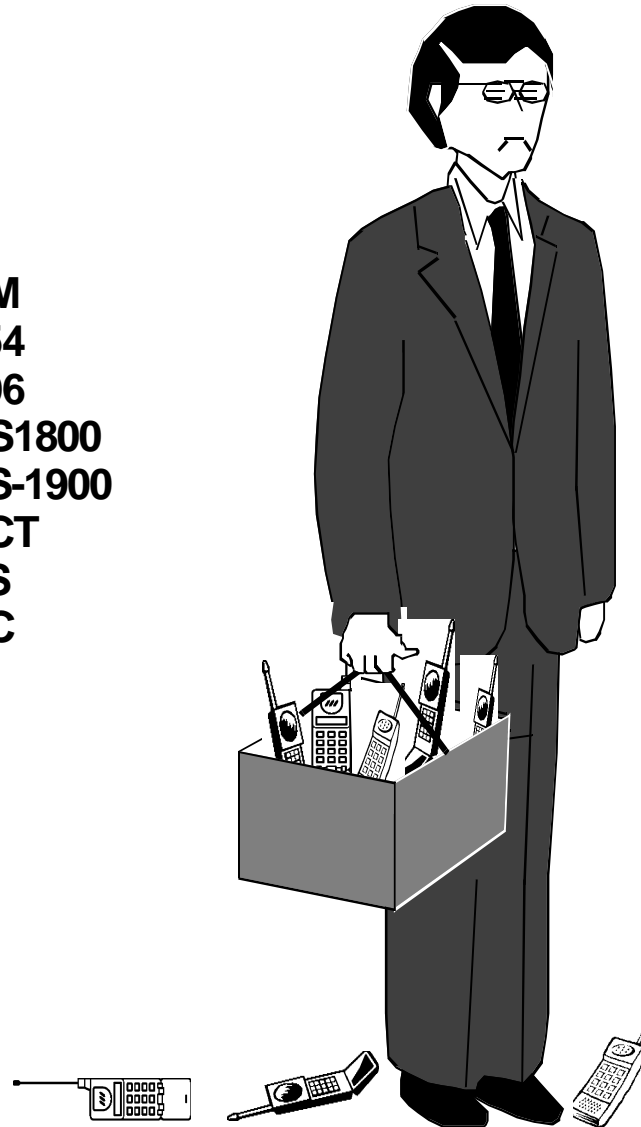


- Supports modulo and bit reversal arithmetic
- Often duplicated to calculate multiple addresses per cycle

# Lets look at an application - Supporting the Road Warrior of 1999

---

**GSM**  
**IS-54**  
**IS-96**  
**DCS1800**  
**PCS-1900**  
**DECT**  
**PHS**  
**PDC**  
**etc**



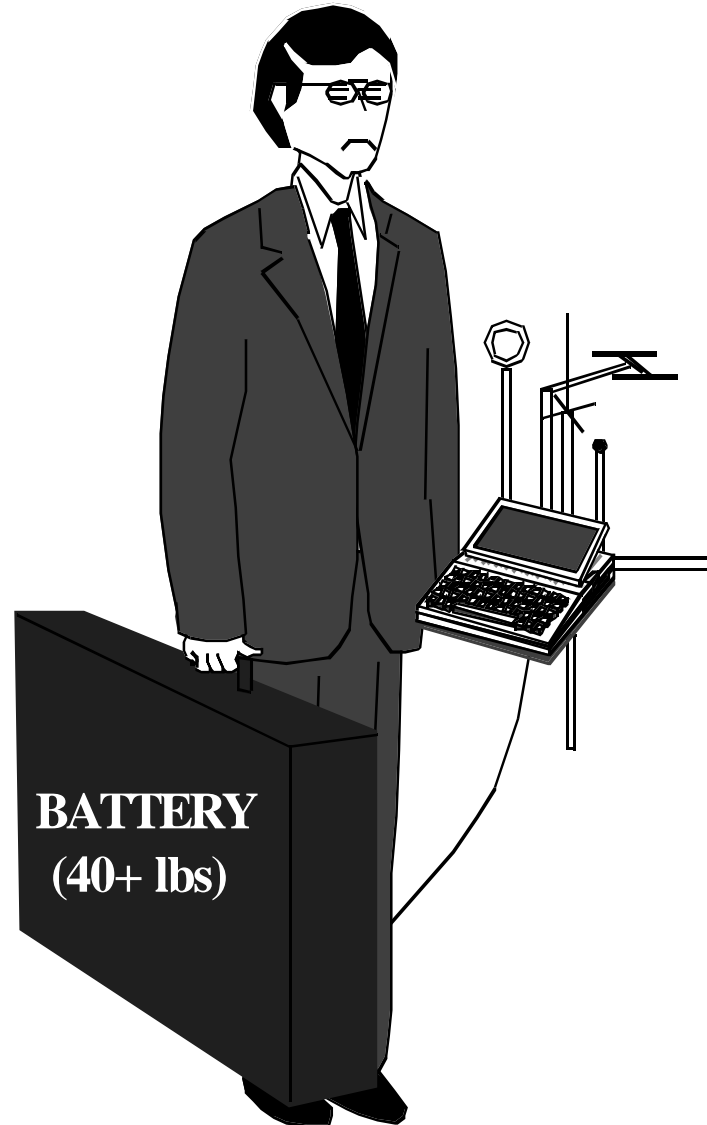
# Physical Layer Standards

| Parameter       | AMPS   | IS54               | GSM                          | JCP                | DECT            | CT2             | PHS            | 802.11FH |
|-----------------|--------|--------------------|------------------------------|--------------------|-----------------|-----------------|----------------|----------|
| Origin          | EIATIA | EIATIA             | ETSI                         |                    | ETSI            | UK              | Japan          | IEEE     |
| Access          | FDD    | FDM/FDD/TDM        | FDM/FDD/TDM                  |                    | FDM/TDM/TDD     | FDM/TDD         | TDM/TDD        | FH/FDM   |
| Modulation      | FM     | $\pi/4$ QPSK       | GMSK, diff                   | $\pi/4$ DQPSK      | GFSK            | GFSK            | $\pi/4$ -DQPSK | (G)FSK   |
| Baseband filter |        | Root raised cosine | Root raised cos. $\beta=0.3$ | Root raised cosine | Gaussian BT=0.5 | Gaussian BT=0.5 | Root Nyquist   |          |

# Software radio solution?

---

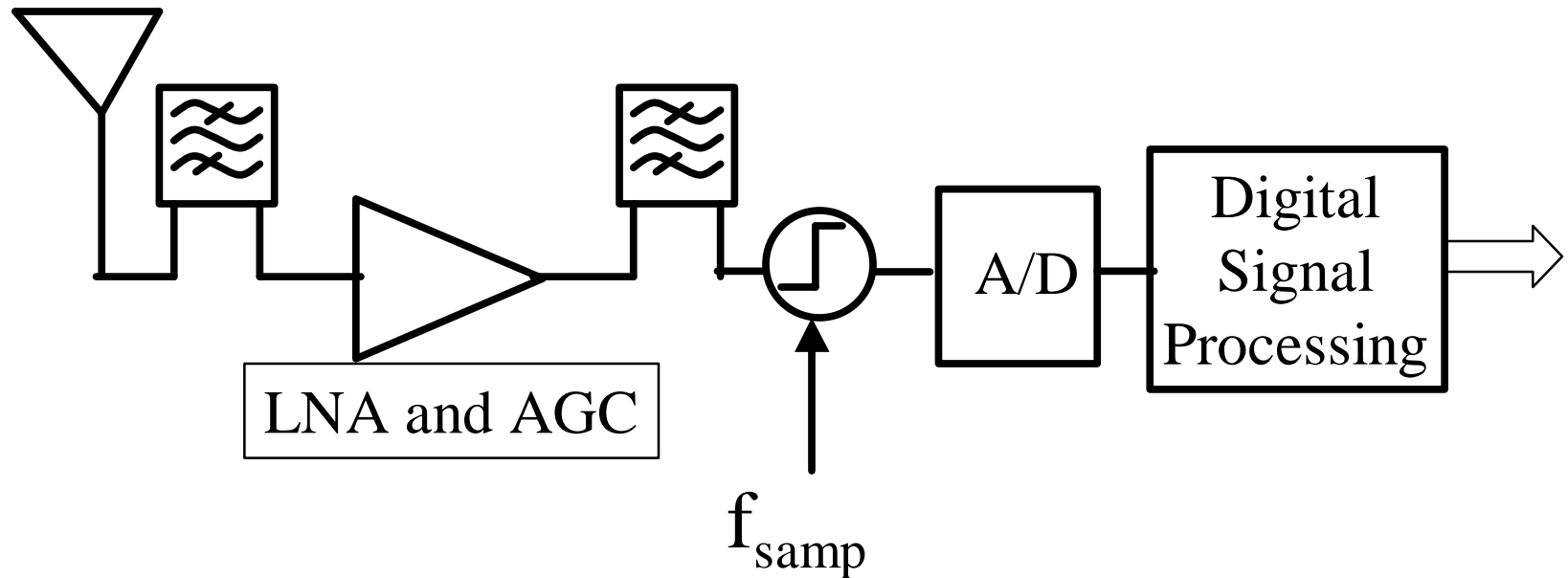
---



# How to implement a software radio

---

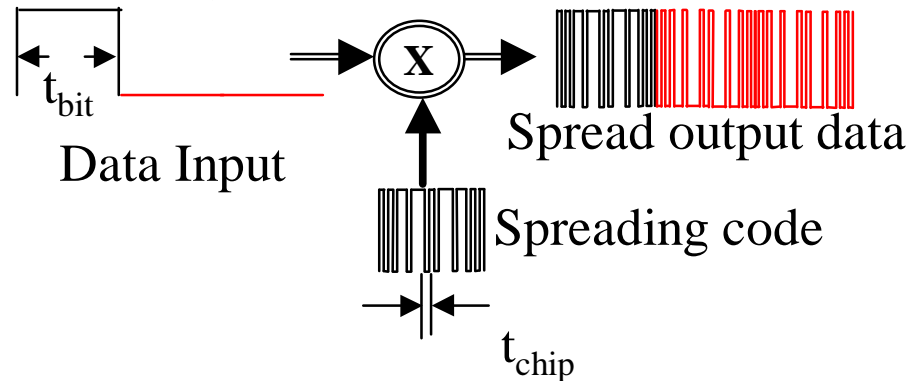
---



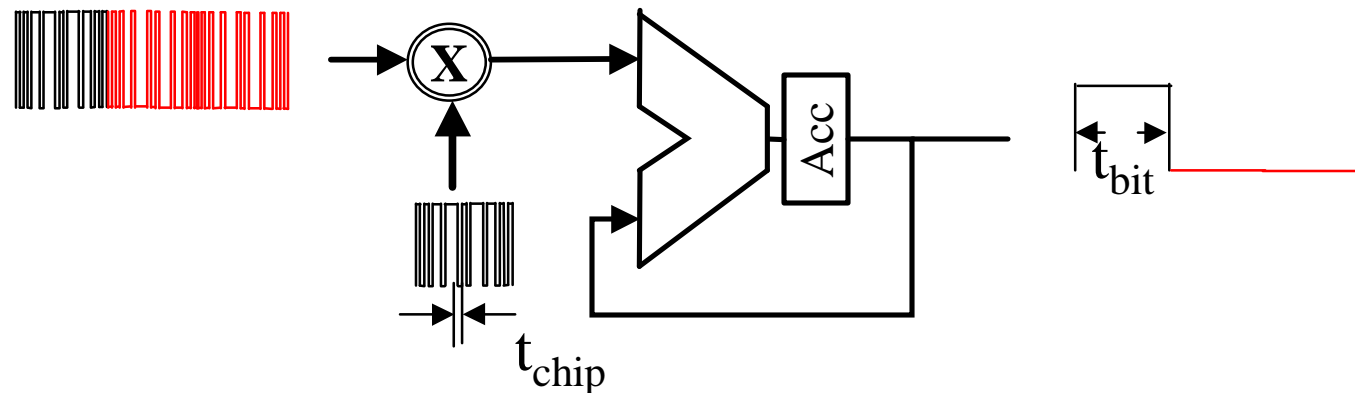
- Convert to digital representation as close to the antenna as possible
- Determine the best architecture to perform the DSP (FFT's, filters, correlators, ...)

# Example of the digital processing - Direct sequence spread spectrum (CDMA)

- Modulator (transmit side)

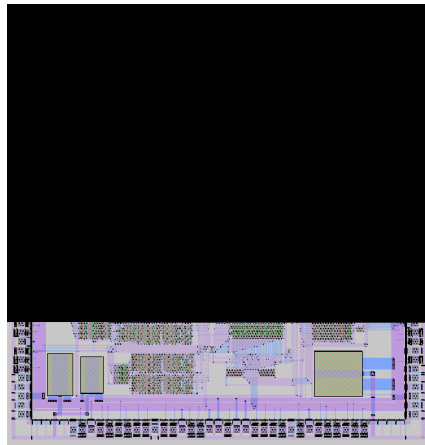


- Demodulator (transmit side) - a correlator is needed to decode the data

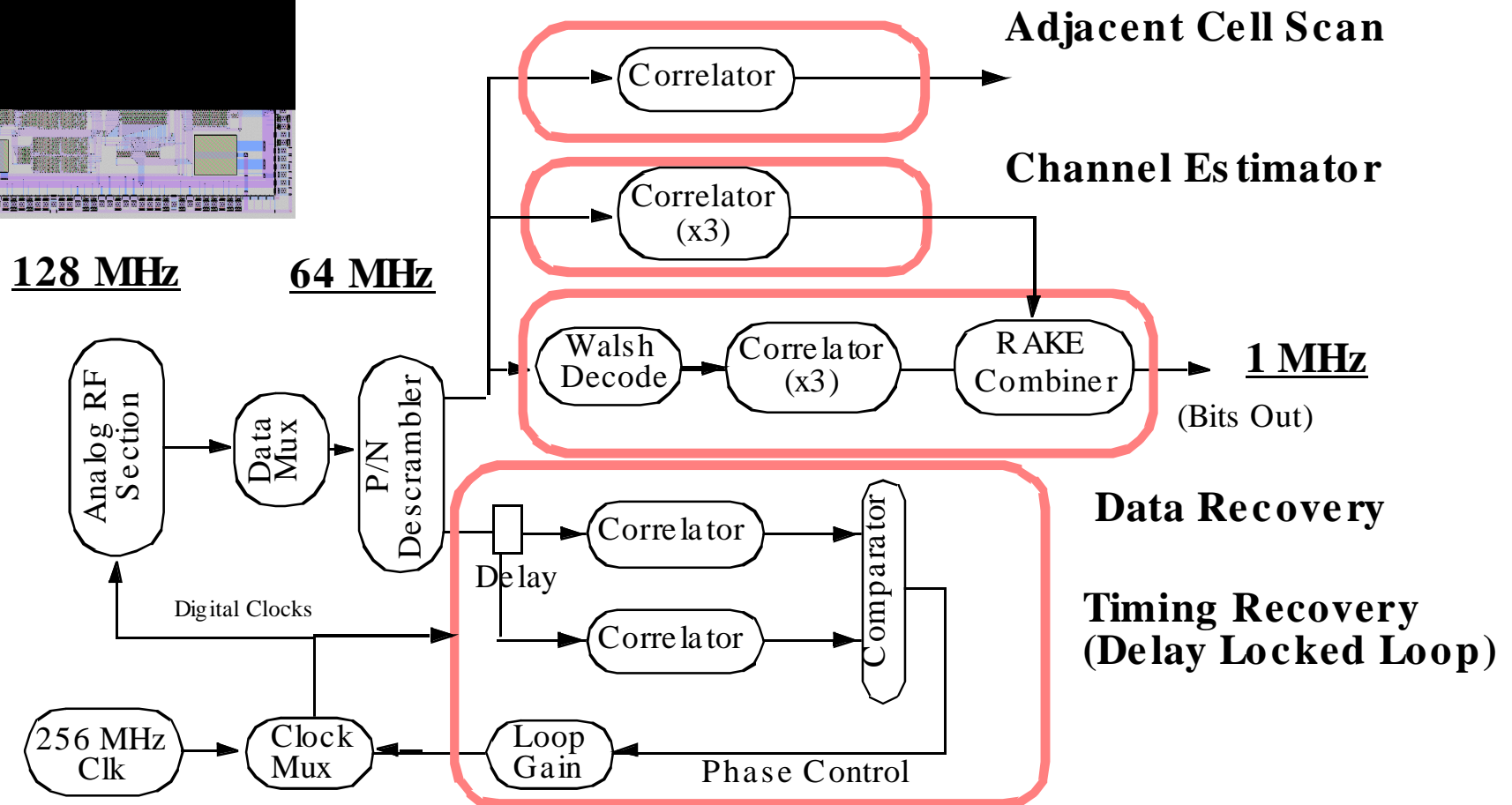


Again we have a MAC requirement,  
accumulations are performed at the chip rate

# Efficiency of direct mapping - CDMA digital baseband architecture



~1000 Mops using 27 mW at  
1.5 volts - 30Mops/mW



# Summary

## How is DSP different?

---

---

- Essentially infinite streams of data which need to be processed in real time
- Relatively small programs and data storage requirements
- Intensive arithmetic processing with low amount of control and branching (in the critical loops)
- High amount of I/O with analog interface
- Loosely coupled multiprocessor operation

# Summary

## How are DSP $\mu$ P's different

---

- Single cycle multiply accumulate (multiple busses and array multipliers)
- Complex instructions for standard DSP functions (IIR and FIR filters, convolvers)
- Specialized memory addressing
  - Bit reversal (FFT)
  - Modular arithmetic for circular buffers (delay lines)
- Zero overhead loops and repeat instructions
- I/O support
  - Serial and parallel ports
  - DMA
  - A/D and D/A interface
- Limited use of data and instruction caches
- Compiler support for hazard elimination

# Tradeoff off between high performance $\mu$ P and DSP's

---

---

- Advantages of General Purpose  $\mu$ P's
  - High volume production advantages
  - High level language and tool support
  - Efficient implementation of non-DSP tasks
  - Higher clock rates and more advanced technology
- Advantages of DSP  $\mu$ P's
  - Software and development support for signal processing applications (filters, FFT's, etc.)
  - Real Time OS and application libraries
  - Minimal support chips
  - Variety of versions allow cost/performance/power tradeoffs
  - Low cost