Microcontrollers and DSPs

Contents

Microcontrollers and DSPs

Some references

1. D. A. Patterson, J. L. Hennessy, “Computer Organization and Design”, Morgan Kaufmann, cap. 5 pagg. 338-416.
2. A. Clements, "The principles of computer hardware", Oxford, 2000, cap. 5, pagg. 231-244.
3. P. Lapsley, J. Bier, A. Shoham, E.A. Lee, "DSP Processor Fundamentals - Architectures and Features", IEEE Press, New York, 1997, cap. 1.
4. K. Hintz, D. Tabak, "Microcontrollers - Architecture, Implementation and Programming", Mc Graw - Hill, 1992, par. 1.1.4, pp. 16-26.

Microcontrollers (mCs)

A microcontroller is a processor specifically designed and optimized to perform control, timing, supervising tasks on various target devices. It is characterized by the availability of relatively large amounts of “on chip” memory (ROM, EEPROM, Flash ...) and of several peripheral units, for different functions (I/O, A/D conversion, timer, counters, PWM, …). It is normally characterized by reduced complexity and low cost.

Microcontrollers (mCs)

Peripheral units in mCs:

- A/D converters (number of bit, conversion speed, linearity vary a lot among different devices)
- Timer and counters
- PWM modulators
- External memories (ROM, EEPROM, FLASH)
- Communication ports (serial, I2C, field bus e.g. CAN)

Microcontrollers (mCs)

The use of mCs is very common for the implementation of:

- portable measurement instruments;
- PC peripherals;
- fax/photocopiers;
- home appliances;
- cell phones;
- industrial applications, in particular in the automotive and electrical drives fields.

Digital Signal Processors (DSPs)

DSPs are microprocessors specifically designed and optimized to efficiently perform real time signal processing tasks. They are characterized by high computational power and relatively low cost (if compared to general purpose processors). Particular care is taken in minimizing the power consumption (e.g. in embedded portable applications).

Digital Signal Processors (DSPs)

Several different DSP families are available on the market. They all exhibit some common features:

- availability of a built-in multiplier circuit (MAC instruction);
- capability to operate multiple memory accesses in a single clock cycle;
- specific addressing modes for circular registers and stacks;
- sophisticated program flow control instructions;
- availability of DMA circuitry (top level).

Digital Signal Processors (DSPs)

The major application areas for DSPs are related to:

- coding/decoding of speech, hi-fi audio segnals, video signal processing;
- compression/decompression of data;
- encryption/decryption of data;
- mixing of audio and video signals;
- sound synthesis.

DSPs vs mCs

Traditionally, mCs were used in the implementation of control functions, thanks to the wide range of peripheral units available on-chip. The computational power was limited (CPUs had 8 bits or less, no hardware multiplier).

DSPs were used, instead, almost only for signal-processing applications, where the key parameter is computational power.

Currently, the differences in the application fields of mCs and DSPs are a lot fuzzier.

DSPs vs mCs

More recent DSPs include peripheral units traditionally typical of mCs. On the other hand, mCs present, at least in top level models, hardware organizations and computational powers closer and closer to those typical of DSPs. Costs and performance may be very close and, for particular applications, the choice of the device may be quite difficult.

We definitely need criteria to compare different devices.

DSPs vs mCs

The fundamental parameters for the comparison are, of course, cost and performance.

To minimize the cost parameter, for given specifications, it is normally required to take into account not only the device cost, but also the estimated development time, the so called time to market.

DSPs vs mCs

The cost of device is largely dependent on the expected production volume.

Time and resources required by the development of the application are a function of several factors, like:

- availability of high quality and high reliability development tools;
- effective technical support from the device manufacturer.

DSPs vs mCs

The application specifications determine the performance level required for the selected microprocessor in terms of:

- required peripheral units and their basic parameters (e.g. A/D converter with 8, 10 or 12 bits);
- operating conditions (e.g. maximum allowable power consumption, temperature range);
- required computational power (real time control, signal processing …).

Performance measurement

The perfomance level of any processor can be measured only in terms of time required to excute a given program.

In the case of mCs or DSPs this is the same time the processor effectively spends on the program instructions (computation time), unless an operating system coordinating several tasks in time sharing is running on the device.

Estimation of computation time

The computation time of a program is a key parameter in real time applications (both in control and signal processing). This can be estimated based on three parameters:

- processor clock period;
- number of clock cycles required by the instructions in the program;
- number of instructions required by the program.

Estimation of computation time

The clock period and the number of clock cycles required by the various program instructions can be read on the processor datasheet/user manual.

The number of instructions required by a given algorithm is a function of the processor architecture.

By architecture we mean the set of resources that are available to the programmer for the implementation of the algorithm (instruction set).

Estimation of computation time

Any given architecture can be implemented in different ways at the hardware level.

We must therefore distinguish processor organization and architecture: the former is the particular hardware implementation of the latter.

The architecture has a direct effect on the number of instructions required by a given program. The organization determines the clock period and the number of clock cycles required by any instruction.

Estimation of computation time

The computation time of a program can be estimated by using the following formula:

$$T_{cal} = T_{clk} \cdot \sum_{i=1}^{N_{cl}} N_i \cdot NC_i \qquad (1)$$

where T_{clk} is the processor clock period, N_i is the number of class *i* instructions in the program, NC_i is the average number of clock cycles required by class *i instructions*, N_{cl} is the number of considered instruction classes.

Estimation of computation time

Relation (1) assumes that the program is not interrupted by other processes and neglects the delays due to memory accesses.

To increase the speed of a processor, we therefore need to:

- reduce the clock cycle (T_{clk});
- reduce the number of cycles required by the more commonly used instructions (NC);
- reduce the number of instructions required by a given algorithm.

Speed limits!

Reducing the clock cycle duration always implies the increase of power consumption for the processor.

This can be limited by reducing also the supply voltage.

Which tells us why there is a strong need for lower and lower power supply voltages (<1V) in computer applications.

The limitations are basically technological (we need new processes/materials).

Speed limits!

The number of instructions required by a given algorithm is a function of the processor architecture, i.e. of its instruction set, as seen by the programmer/compiler.

The reduction of this parameter leads to complex instruction set computers (CISC), instead of reduced (and simple) instruction set computers (RISC). This again affects the processor organization and its cost. That´s why RISC processors are a lot more used than CISC processors.

Speed limits!

The reduction of the number of clock cycles required by an instruction calls for a more sophisticated hardware organization of the processor, e.g. wired control instead of micro-programmed control, higher degree of parallelism (achievable in several different ways: VLIW, SIMD, etc.) or the use of pipelines.

This trend leads to complex processors, with high cost. The limitation in this case is basically "economical".

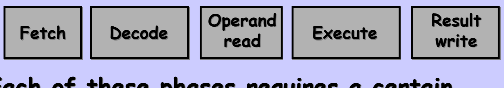
Processor control

The execution of any instruction is normally divided into several different phases. For instance, the instruction

Add R1, num;

That is [R1] = [R1] + M[num].

could require the following actions:



Each of these phases requires a certain amount of time, depending on the speed of response of the processor functional units.

Processor control

In a single cycle organization, the processor clock period will have to be long enough to allow the execution of all phases, so it will be equal to, at least, the sum of the response times of all the functional units.

In a multicycle organization, each phase is executed in, at least, a clock period. The instruction in the example will then require, at least, 5 clock periods. The execution time of some instructions can, in some cases, be longer in a multi cycle organization with respect to a single cycle one.

Single vs multi cycle organization

Unless we are considering very simple processors with extremely small instruction sets, as some RISC processors, the single cycle organization tends to be inefficient.

The most complex instruction, i.e. that requiring the longest time, determines the minimum possible clock period.

This strongly penalizes the execution of faster instructions, that only require a fraction of that period. For the remaining time, the processor is not doing anything.

Comparison single cycle/multicycle

Let's consider a processor with the following response times:

- memory: 2 ns
- ALU: 2 ns
- registers: 1 ns

Let's also consider a typical program made up of instructions like the following:

- memory reads (load): 24%
- memory writes (store): 12%
- ALU operations on registers: 44%
- jumps (or branches): 20%

Comparison single cycle/multicycle

Any single cycle implementation of a processor control unit must be designed to allow the execution of the slowest instruction.

Usually, this is the memory read (like "load R1, num") which requires:

a. fetch phase: 2 ns;
b. possible ALU operation (offset): 2 ns;
c. register write (memory address): 1 ns;
d. data memory access: 2 ns;
e. register write: 1 ns.

Comparison single cycle/multicycle

A multicycle implementation of the control unit is, instead, limited only by the response time of the slowest CPU functional unit, that is the memory access unit (in our example).

Supposing that we can subdivide the clock period into 4 segments (depending on the processor organization), any of these will have to be 2 ns long.

In the two implementations, the *load* instruction is going to have the same duration (8 ns). Simpler instructions (e.g. jumps) will be faster in the multicycle case.

Comparison single cycle/multicycle

Supposing that the processor instructions have the following durations (in terms of clock period segments and in absolute value):

load: 4 segments, 8 ns;
store: 4 segments, 8 ns;
ALU operations: 3 segments, 6 ns;
jumps: 2 segments, 4 ns;

The typical program will have an average number NC equal to:

$4 \cdot (0.24 + 0.12) + 3 \cdot 0.44 + 2 \cdot 0.2 = 3.16$

so well below 4.

Pipeline

Both in single and in multi cycle organizations, during the execution of any instruction the different processor units operate only for a fraction of the total execution time, which is highly inefficient.

The pipeline organization tends to remove this inefficiency. The different functional units are operated simultaneously, but on parts of different instructions, as in any industrial pipeline.

The only problem with this organization is in the resolution of data/structural conflicts.

Comparison pipeline/multicycle

Let's consider again the same processor with the following response times:

- memory: 2 ns
- ALU: 2 ns
- registers: 1 ns

Let's also consider again the typical program made up of instructions like the following:

- memory read (load): 24%
- memory write (store): 12%
- ALU operations on registers: 44%
- jumps (or branches): 20%

Comparison pipeline/multicycle

Supposing our pipeline organization allows to:

- execute without interlocking a half of the load operations, with 1 clock cycle penalty the other half.
- execute, without stalling, a half of jumps and, for the other half to get a 2 clock cycle penalty.

It follows from this, that number of clock cycles required by load instructions is on average equal to 1.5, while it is equal to $0.5\cdot1+0.5\cdot(1+2) = 2$ for jump instructions.

Comparison pipeline/multicycle

Our typical program will present an average NC value equal to:

$$1.5\cdot0.24 + 1\cdot(0.12 + 0.44) + 2\cdot0.2 = 1.32$$

We saw that, in the multicycle organization, with 4 segments and no pipeline, the average number of clock cycles per instruction, NC, was equal to 3.16.

The use of a pipeline with 4 stages allows to reduce by almost 60% the execution time of the program. The acceleration allowed by the pipeline is then equal to 2.4.

Comparison pipeline/multicycle

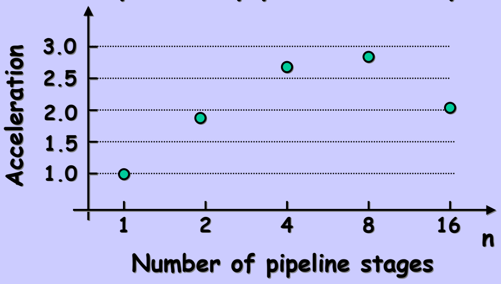


Diagram relating the increase in performance and the number of stages n (also known as "depth") of the pipeline.

Comparison pipeline/multicycle

The factors limiting the acceleration any pipeline can guarantee are:

- data conflicts: the higher the number of stages the higher the penality any stall condition determines;
- decrease in the execution speed of jumps: the higher the number of stages the bigger the delay;
- increase in the CPU control complexity that requires clock frequency reduction: this is due to auxiliary and control circuitry for the pipeline (registers, logic, …).

Superscalar architectures

In last generation DSPs, we are now seeing even more complex organizations, typically taken from the world of general purpose processors (GPPs).

Among these, it is quite common to find the so called superscalar architectures.

These are based on the replication of functional units within the CPU, so as to allow the execution of several instructions in parallel.

It is typical to find a replication factor between 2 and 4.

Superscalar architectures

The control strategy for this type of processors usually shows very high complexity:

- branch condition prediction;
- advanced memory organization (dynamic RAM, multi level cache, etc.);
- dynamic pipeline.

The instruction set is usually RISC type, but SIMD architectures are also possible to further increase the level of parallelism.

Superscalar architectures

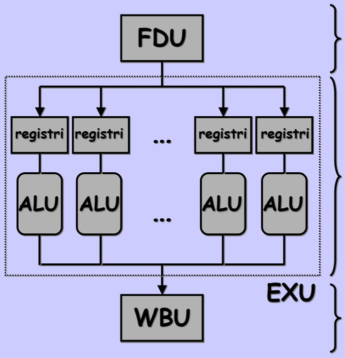
A dynamic pipeline is capable of organizing the execution of instructions in an out of order manner: this way it is possible to reduce the penalties due to stall conditions to a minimum.

The typical structure of a dynamic pipeline is made up of three main units:

- fetch and decode unit (FDU);
- execution unit (EXU);
- write-back unit (WBU).

The first and the last unit operate in order, the second does not.

Superscalar architectures



Fetch and decode: in order, sometimes on several instructions simultaneaously.

Execution in parallel. The FDU distributes the instructions to execute among the functional units of the EXU, using prediction techniques for conditioned branches.

Results write back: the WBU takes care of data conflicts and only writes the "safe" ones.

Superscalar architectures

The use of these complex organizations with functional unit parallelism, dynamic pipelines, branch predictions, normally implies very high implementation costs.

The DSPs that adopt them, are designed for high performance, reduced scale applications, where the computational power is the key issue.

These devices are characterized by a pretty high power consumption and are not suited to *embedded control* applications for large volume productions.

Superscalar architectures

The DSPs with superscalar architectures are also direct competitors of general purpose processors (GPPs), which often offer signal processing capabilities (in particular for audio and video applications) that are quite close to the DSPs' ones.

DSPs are still preferable because of lower cost and lower power consumption. Besides, being slightly less complex, it is normally easier to estimate the computation time of a program (fundamental for possible real-time applications).

Superscalar architectures vs VLIW

Some DSPs adopt the VLIW (Very Long Instruction Word) strategy to increase the internal level of parallelism.

This technique combines a big number (e.g. 8) of simple instructions in a single instruction memory word, that is fetched in a single clock cycle.

The decoder decomposes the long instruction in its basic components and, exploiting a given degree of hardware parallelism, distributes each component to a different execution unit.

Superscalar architectures vs VLIW

The VLIW approach is not equivalent to the superscalar one because:

- only some particular instruction sequences can be combined in long instruction words that completly exploit the CPU (e.g. that of a FIR filter tap);
- the adopted level of hardware parallelism is normally not too high (two execution units as a maximum);
- the pipeline is static;
- the instruction bus has a lot of bits (e.g. 256).

Maximizing performance

The processor performance is a function of both its architecture and of its organization, at the hardware level.

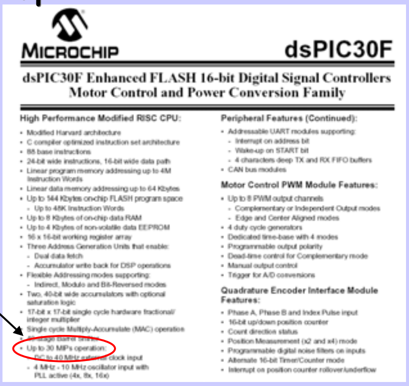
The maximization of performance calls for a co-ordinated design of hardware and software.

The problem is further complicated by the action of several design constraints such as:

- cost;
- electric power consumption.

Example: Data-sheet

The manufacturer claims a 30 MIPS (F_{clk} = 40 MHz) operating speed.


MICROCHIP dsPIC30F

dsPIC30F Enhanced FLASH 16-bit Digital Signal Controllers Motor Control and Power Conversion Family

High Performance Modified RISC CPU:

- Modified Harvard architecture
- C compiler optimized instruction set architecture
- 88 base instructions
- 24-bit wide instructions, 16-bit wide data path
- Linear program memory addressing up to 4M Instruction Words
- Linear data memory addressing up to 64 Kbytes
- Up to 144 Kbytes on-chip FLASH program space
  - Up to 48K Instruction Words
- Up to 8 Kbytes of on-chip data RAM
- Up to 4 Kbytes of non-volatile data EEPROM
- 16 x 16-bit working register array
- Three Address Generation Units that enable:
  - Dual data fetch
  - Accumulator write back for DSP operations
- Flexible Addressing modes supporting:
  - Indirect, Modulo and Bit-Reversed modes
- Two, 40-bit wide accumulators with optional saturation logic
- 17-bit x 17-bit single cycle hardware fractional/ integer multiplier
- Single cycle Multiply-Accumulate (MAC) operation
- 40-stage Barrel Shifter
- Up to 30 MIPs operation:
  - DC to 40 MHz external clock input
  - 4 MHz - 10 MHz oscillator input with PLL active (4x, 8x, 16x)

Peripheral Features (Continued):

- Addressable UART modules supporting:
  - Interrupt on address bit
  - Wake-up on START bit
  - 4 characters deep TX and RX FIFO buffers
- CAN bus modules

Motor Control PWM Module Features:

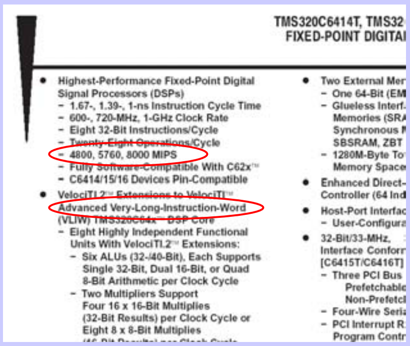
- Up to 8 PWM output channels
  - Complementary or Independent Output modes
  - Edge and Center Aligned modes
- 4 duty cycle generators
- Dedicated time-base with 4 modes
- Programmable output polarity
- Dead-time control for Complementary mode
- Manual output control
- Trigger for A/D conversions

Quadrature Encoder Interface Module Features:

- Phase A, Phase B and Index Pulse input
- 16-bit up/down position counter
- Count direction status
- Position Measurement (x2 and x4) mode
- Programmable digital noise filters on inputs
- Alternate 16-bit Timer/Counter mode
- Interrupt on position counter rollover/underflow

Example: Data-sheet

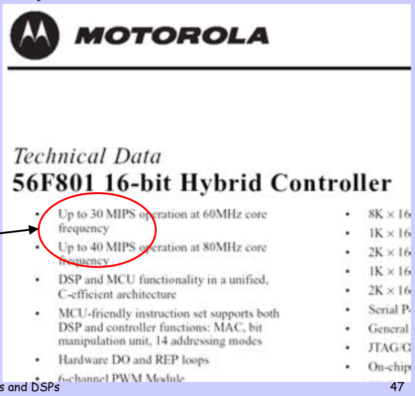
The manufacturer claims a maximum speed of 8000 MIPS! But here F_{clk} = 1000 MHz!


TMS320C6414T, TMS32
FIXED-POINT DIGITAL

- Highest-Performance Fixed-Point Digital Signal Processors (DSPs)
  - 1.67-, 1.39-, 1-ns Instruction Cycle Time
  - 600-, 720-MHz, 1-GHz Clock Rate
  - Eight 32-Bit Instructions/Cycle
  - Twenty-Eight Operations/Cycle
  - 4800, 5760, 8000 MIPS
  - Fully Software-Compatible With C62x™
  - C6414/15/16 Devices Pin-Compatible
- VelociTI.2™ Extensions to VelociTI™ Advanced Very-Long-Instruction-Word (VLIW) TMS320C64x™ DSP Core
  - Eight Highly Independent Functional Units With VelociTI.2™ Extensions:
    - Six ALUs (32-/40-Bit), Each Supports Single 32-Bit, Dual 16-Bit, or Quad 8-Bit Arithmetic per Clock Cycle
    - Two Multipliers Support Four 16 x 16-Bit Multiplies (32-Bit Results) per Clock Cycle or Eight 8 x 8-Bit Multiplies
- Two External Mer
  - One 64-Bit (EM
  - Glueless Interf Memories (SRA Synchronous M SBSRAM, ZBT
  - 1280M-Byte To Memory Space
- Enhanced Direct- Controller (64 Ind
- Host-Port Interfac
  - User-Configura
- 32-Bit/33-MHz, Interface Conforr [C6415T/C6416T]
  - Three PCI Bus Prefetchable Non-Prefetcl
  - Four-Wire Seria
  - PCI Interrupt R Program Contr

Example: Data-sheet

The manufacturer claims a maximum speed of 40 MIPS (F_{clk} = 80 MHz).


MOTOROLA

Technical Data

56F801 16-bit Hybrid Controller

- Up to 30 MIPS operation at 60MHz core frequency
- Up to 40 MIPS operation at 80MHz core frequency
- DSP and MCU functionality in a unified, C-efficient architecture
- MCU-friendly instruction set supports both DSP and controller functions: MAC, bit manipulation unit, 14 addressing modes
- Hardware DO and REP loops
- 6-channel PWM Module
- 8K × 16
- 1K × 16
- 2K × 16
- 1K × 16
- 2K × 16
- Serial P
- General
- JTAG/C
- On-chip