Embedded System HW

Tajana Simunic Rosing
Department of Computer Science and Engineering
University of California, San Diego.
Embedded System Hardware

- Embedded system hardware is frequently designed as a part of a control system ("hardware in a loop"): cyber-physical systems
Hardware platform architecture
Microprocessors in Embedded Systems

Tajana Simunic Rosing
Department of Computer Science and Engineering
University of California, San Diego.
System-on-Chip platforms

Nvidia Tegra 2 die photo

Qualcomm Snapdragon block diagram

General processor
- Application processor (CPU)

Specialized units
- Graphics processing unit (GPU)
- Digital signals processing (DSP)
- Etc.
## Processor comparison metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Clock frequency</strong></td>
<td>Computation speed</td>
</tr>
<tr>
<td><strong>Memory subsystem</strong></td>
<td>Indeterminacy in execution&lt;br&gt;Cache miss: compulsory, conflict, capacity</td>
</tr>
<tr>
<td><strong>Power consumption</strong></td>
<td>Idle power draw, dynamic range, sleep modes</td>
</tr>
<tr>
<td><strong>Chip area</strong></td>
<td>May be an issue in small form factors</td>
</tr>
<tr>
<td><strong>Versatility/specialization</strong></td>
<td>FPGAs, ASICs</td>
</tr>
<tr>
<td><strong>Non-technical</strong></td>
<td>Development environment, prior expertise, licensing</td>
</tr>
</tbody>
</table>

Examples: ARM Cortex-A, TI C54x, TI 60x DSPs, Altera Stratix, etc.
## Processor comparison metrics

| Parallelism | Superscalar pipeline  
|             | • Depth & width $\rightarrow$ latency & throughput  
|             | Multithreading  
|             | • GPU workload requires different programming effort  
| **Instruction set architecture** | Complex instruction set computer (**CISC**):  
|             | • Many addressing modes  
|             | • Many operations per instruction  
|             | • E.g. TI C54x  
|             |  
|             | Reduced instruction set computer (**RISC**):  
|             | • Load/store  
|             | • Easy to pipeline  
|             | • E.g. ARM  
|             |  
|             | Very Long Instruction Word (**VLIW**)  

Processor comparison metrics

• How do you define “speed”?  
  – Clock speed – but instructions per cycle may differ  
  – Instructions per second – but work per instr. may differ

• Practical evaluation  
  – Dhrystone: Synthetic benchmark, developed in 1984  
    • Dhrystones/sec (a.k.a. Dhrystone MIPS) to normalize difference in instruction count between RISC/CISC  
    • MIPS: 1 MIPS = 1757 Dhrystones per second (based on Digital’s VAX 11/780)  
  – SPEC: set of more realistic benchmarks, but oriented to desktops  
  – EEMBC: EDN Embedded Benchmark Consortium, [www.eembc.org](http://www.eembc.org)  
    • Suites of benchmarks: automotive, consumer electronics, networking, office automation, telecommunications  
    • E.g. CoreMark, intended to replace Dhrystone  
  – 0xbench: Integrated Android benchmarks  
    • Covers C library, system calls, Javascript (web performance), graphics, Dalvik VM garbage collection

• What about power and energy?
Power and Energy Relationship

\[ E = \int P \, dt \]
Power Saving Strategies

- System and components are:
  - Designed to deliver peak performance, but ...
  - Not needing peak performance most of the time

- Dynamic Power Management (DPM)
  - Shut down components during idle times

- Dynamic Voltage Frequency Scaling (DVFS)
  - Reduce voltage and frequency of components

- System Level Power Management Policies
  - Manage devices with different power management capabilities
  - Understand tradeoff between DPM and DVFS
Dynamic Voltage Scaling (DVS)

Power consumption of CMOS circuits (ignoring leakage):

\[ P = \alpha \cdot C_L \cdot V_{dd}^2 \cdot f \]

\( \alpha \): switching activity
\( C_L \): load capacitance
\( V_{dd} \): supply voltage
\( f \): clock frequency

Delay for CMOS circuits:

\[ \tau = k \cdot C_L \cdot \frac{V_{dd}}{(V_{dd} - V_t)^2} \]

\( V_t \): threshold voltage
\( (V_t \ll V_{dd}) \)

DVFS vs. power down

(a) No power-down

(b) Power-down

(c) Dynamic voltage scaling

Time

Power

Deadline
Energy Savings with DVFS

\[ P_R = P_{c_1} - P_{c_2} \]  
**Reduction in CPU power**

\[ P_E = (P_{c_2} - P_{c_{idle}}) + (P_d - P_{d_{idle}}) \]  
**Extra system power**

\[ E_{DVFS} = P_R t_1 - P_E t_{delay} \]
Linux Frequency Governors

Governors:
- Performance
- Ondemand
- Powersave

In-between frequency range

Highest Frequency

Lowest Frequency

Time

Frequency
Parallel CPU Architectures
Parallelism extraction

• Static
  – Use compiler to analyze program code
  – Can make use of high-level language constructs
  – Cannot inspect data values
  – Simpler CPU control

• Dynamic
  – Use hardware to identify opportunities
  – Can make use of data values
  – More complex CPU

• Parallelism exists at several levels of granularity
  • Task
  • Data
  • Instruction
Superscalar

- Instruction-level parallelism
- Replicated execution resources
  - E.g. ALU components
- RISC instructions are pipelined
  - N-way superscalar: \( n \) inst/cycle \( \rightarrow \) \( n^2 \) HW

<table>
<thead>
<tr>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>MEM</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
</tbody>
</table>

2-way Superscalar
Single Issue Multiple Data - SIMD

- Multiple processing elements perform the same operation on multiple data points simultaneously (e.g. ADD)
  - exploit data level parallelism, but not concurrency: there are simultaneous (parallel) computations, but only a single process (instruction) at a given moment
  - It is similar to superscalar, but the level of parallelism in SIMD is much higher
- CPUs use SIMD instructions to improve the performance of multimedia
- Disadvantages:
  - Not all algorithms can be vectorized easily
    - It usually requires hand coding
  - Requires large register files -> higher power consumption and area
VLIW architecture

- Large register file feeds multiple function units (FUs)
- Compile time assignment of instructions to FUs

```
EBOX (execution unit)
Add r1,r2,r3; Sub r4,r5,r6; Ld r7,foo; St r8,baz; NOP
```

Diagram:
- Register file
- ALU
- ALU
- Load/store
- Load/store
- FU
Clustered VLIW architecture

- Register file, function units divided into clusters.
Why isn’t everything just massively parallel?

• Types of architectural hazards
  – Data (e.g. read-after-write, pointer aliasing)
  – Structural
  – Control flow

• Difficult to fully utilize parallel structures
  – Programs have real dependencies that limit ILP
  – Utilization of parallel structures depends on programming model
  – Limited window size during instruction issue
  – Memory delays

• High cost of errors in prediction/speculation
  – Performance: Stalls introduced to wait for reissue
  – Energy: Wasted power going down wrong execution path
## Embedded processor trends

<table>
<thead>
<tr>
<th></th>
<th>ARM11</th>
<th>ARM Cortex-A8</th>
<th>ARM Cortex-A9</th>
<th>Qualcomm Scorpion</th>
<th>Qualcomm Krait[1]</th>
<th>ARM Cortex-A15 MPCore</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Decode</strong></td>
<td>single-issue</td>
<td>2-wide</td>
<td>2-wide</td>
<td>2-wide</td>
<td>3-wide</td>
<td>3-wide</td>
</tr>
<tr>
<td><strong>Pipeline depth</strong></td>
<td>8 stages</td>
<td>13 stages</td>
<td>8 stages</td>
<td>10 stages</td>
<td>11 stages</td>
<td>15/17-25 stages</td>
</tr>
<tr>
<td><strong>Out of Order</strong></td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes, non-speculative[2]</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Execution</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FPU</strong></td>
<td>VFPv2 (pipelined)</td>
<td>VFPv3 (not pipelined)</td>
<td>VFPv3-D16 or VFPv3-D32 (typical) (pipelined)</td>
<td>VFPv3 (pipelined)</td>
<td>VFPv4 (pipelined)[3]</td>
<td>VFPv4 (pipelined)</td>
</tr>
<tr>
<td><strong>NEON</strong></td>
<td>None</td>
<td>Yes (Partially 128-bit wide)</td>
<td>Optional (Partially 128-bit wide)</td>
<td>Yes (128-bit wide)</td>
<td>Yes (128-bit wide)</td>
<td>Yes (128-bit wide)</td>
</tr>
<tr>
<td><strong>Process Technology</strong></td>
<td>90 nm</td>
<td>65/45 nm</td>
<td>45/40/32/28 nm</td>
<td>65/45 nm</td>
<td>28 nm</td>
<td>32/28 nm</td>
</tr>
<tr>
<td><strong>L0 Cache</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>L1 Cache</strong></td>
<td>Varying, typically 16 kB + 16 kB</td>
<td>32 kB + 32 kB</td>
<td>32 kB + 32 kB</td>
<td>32 kB + 32 kB</td>
<td>16 kB + 16 kB 4-way set associative</td>
<td>32 kB + 32 kB per core</td>
</tr>
<tr>
<td><strong>L2 Cache</strong></td>
<td>Varying, typically none</td>
<td>256 or 512 (typical) kB</td>
<td>1 MB</td>
<td>256 kB (Single-core)/512 kB (Dual-core)</td>
<td>1 MB 8-way set associative (Dual-core)/2 MB (Quad-core)</td>
<td>up to 4 MB per cluster, up to 8 MB per chip</td>
</tr>
<tr>
<td><strong>Core Configurations</strong></td>
<td>1</td>
<td>1</td>
<td>1, 2, 4</td>
<td>1, 2</td>
<td>2, 4</td>
<td>2, 4, 8(4×2)</td>
</tr>
<tr>
<td><strong>DMIPS/MHz speed per core</strong></td>
<td>1.25</td>
<td>2.0</td>
<td>2.5</td>
<td>2.1</td>
<td>3.3</td>
<td>3.5</td>
</tr>
</tbody>
</table>

### Parallelization
- Multiple cores
- Multiple-issue per core
- Out of order execution
- Single Issue Multiple Data - SIMD (NEON)

### Process technology
- Higher density allows more parallel structures
- Higher power density, thermal issues

---

*NEON: Advanced SIMD extension is a combined 64- and 128-bit instruction set that provides standardized acceleration for media and DSP apps*
General/applications processing

ARMv7 ARCHITECTURE
ARMv7

• ARM assembly language - RISC
• ARM programming model
• ARM memory organization
• ARM data operations (32 bit)
• ARM flow of control
• Hardware-based floating point unit
ARM programming model

Every arithmetic, logical, or shifting operation sets CPSR bits: N (negative), Z (zero), C (carry), V (overflow)
ARM pipeline execution

```
add r0, r1, #5
sub r2, r3, r6
cmp r2, #3
```

Time:
1  2  3  time
ARM data instructions

- ADD, ADC : add (w. carry)
- SUB, SBC : subtract (w. carry)
- MUL, MLA : multiply (and accumulate)
- AND, ORR, EOR
- BIC : bit clear
- LSL, LSR : logical shift left/right
- ASL, ASR : arithmetic shift left/right
- ROR : rotate right
- RRX : rotate right extended with C
ARM flow of control

• All operations can be performed conditionally, testing CPSR:
  – EQ, NE, CS, CC, MI, PL, VS, VC, HI, LS, GE, LT, GT, LE

• Branch operation:
  B #100
  – Can be performed conditionally.
ARM comparison instructions

- CMP : compare
- CMN : negated compare
- TST : bit-wise AND
- TEQ : bit-wise XOR
- These instructions set only the NZCV bits of CPSR.
ARM load/store/move instructions

- **LDR, LDRH, LDRB**: load (half-word, byte)
- **STR, STRH, STRB**: store (half-word, byte)
- **Addressing modes**:
  - register indirect: `LDR r0, [r1]`
  - with second register: `LDR r0, [r1, -r2]`
  - with constant: `LDR r0, [r1, #4]`
- **MOV, MVN**: move (negated)
  
  ```
  MOV r0, r1 ; sets r0 to r1
  ```
Addressing modes

- **Base-plus-offset addressing:**
  \[ \text{LDR } r0, [r1, #16] \]
  - Loads from location \( r1+16 \)

- **Auto-indexing increments base register:**
  \[ \text{LDR } r0, [r1, #16]! \]

- **Post-indexing fetches, then does offset:**
  \[ \text{LDR } r0, [r1], #16 \]
  - Loads \( r0 \) from \( r1 \), then adds \( 16 \) to \( r1 \)
ARM subroutine linkage

• Branch and link instruction:
  \[ \text{BL } \text{foo} \]
  – Copies current PC to r14.

• To return from subroutine:
  \[ \text{MOV } r15, r14 \]
Raspberry Pi – Microprocessor/CPU

- Designed for mobile applications
- Raspberry Pi2: Broadcom BCM2836 SoC
  - CPU: Quad-core Cortex-A7 (900 MHz)
  - GPU: Dual Core VideoCore IV® Co-Processor
  - Level 1 cache: 32 KB, Level 2 cache: 512 KB
- Raspberry Pi3: Broadcom BCM2837 SoC
  - Identical underlying architecture to BCM2836
  - CPU: Quad-core Cortex-A53 Processor
  - Better performance (around 50%) than Cortex-A7 of RPi2
- Raspberry Zero: Broadcom BCM2835 SoC
  - CPU: 1 GHz single-core ARM1176JZF-S
  - Cheapest Raspberry Pi
Rasp. Pi2 - BCM2836 Processor Architecture

- BCM2836
  - ARMv7 Architecture
  - Single Core
  - 32-Bit RISC
  - 700 MHz Clock Rate

- 8 Pipeline Stages:

Components:
- Core
- Load Store Unit
- Prefetch Unit
- Memory System
- Vector Floating Point
Rasp. Pi 2 Cortex-A7 Processor Architecture

• Cortex-A7 CPU architecture
  – ARMv7, Thumb-2
  – Also used for big.LITTLE
    • Feature-compatible with Cortex-A15 (big)
    • Rule of thumb: CPU power increases by 2x for an increase in performance by 50%
    • SW is the same; use the right CPU for the right task
  – VFPv4 Floating Point Unit, NEON SIMD
  – In-order dual-issue 8 stage pipeline
Cortex Processor Architecture: NEON SIMD

- **NEON SIMD**
  - SIMD: Simultaneous computation of large-size operands
  - Large NEON register file with its dual 128-bit/64-bit views
    - D register: 64 bit, Q register: 128 bit
  - Minimizes access to memory, enhancing data throughput
  - 75% higher performance for multimedia processing in embedded devices
  - "Near zero" increase in power consumption
  - Supports hardware execution of vector floating point instructions (VFP)
  - Works with a standard ARM compiler (`arm_neon.h`)

```
VADD.16 D0, D1, D2
VMUL.I32.S16 Q0, D2, D3;
```

- 64 bit ADD
- 64 bit multiply to 128 bit result
ARMv7 varieties

- ARMv7-A Application profile:
  - Implements a traditional ARM architecture with multiple modes.
  - Supports a Virtual Memory System Architecture (VMSA) based on a Memory Management Unit (MMU).
  - Supports the ARM and Thumb instruction sets.

- ARMv7-R Real-time profile:
  - Implements a traditional ARM architecture with multiple modes.
  - Supports a Protected Memory System Architecture (PMSA) based on a Memory Protection Unit (MPU).
  - Supports the ARM and Thumb instruction sets.

- ARMv7-M Microcontroller profile:
  - Implements a programmers' model designed for low-latency interrupt processing, with hardware stacking of registers and support for writing interrupt handlers in high-level languages.
  - Implements a variant of the ARMv7 PMSA.
  - Supports a variant of the Thumb instruction set.
ARMv7 Summary

• Load/store architecture
• Many instructions are RISCy, operate in one cycle
  – Some multi-register operations take longer.
• All instructions can be executed conditionally
• ARMv7-A is deployed in:
  – Cortex-A15 (Snapdragon Krait, Nvidia Tegra, TI OMAP)
  – Cortex-A7 (Raspberry PI 2)
  – Cortex-A5 (AMD Fusion)
  – Atmel microcontrollers
What is on RPi3?

- Broadcom BCM2837 64bit ARMv8 quad core Cortex A53 processor @ 1.2GHz with dual core VideoCore IV GPU @ 400 MHz
- Memory – 1GB LPDDR2, Storage – micro SD slot
- Video & Audio Out: HDMI 1.4, 4-pole stereo audio, composite video port
- Connectivity – 10/100M Ethernet, WiFi 802.11 b/g/n and Bluetooth 4.1 LE
- USB – 4x USB 2.0 host ports, 1x micro USB port for power
- Expansion
  - MIPI DSI for touch screen display
  - MIPI CSI for camera
  - 40-pin GPIO header
- Power Supply – 5V up to 2.4A

Source: cnx-software.com
RPi3: Cortex-A53

- Quad core CPU
- ARMv8
- High efficiency CPU for wide range of applications in mobile, DTV, automotive, networking,
- Can pair with any ARMv8 core in a big.LITTLE pairing: e.g. Cortex-A57, Cortex-A72, Cortex-A53 or Cortex-A35 clusters

Source: raspberrypi.org
RPi3: Cortex-A53

Architectural Details

• 8-stage pipelined processor with 2-way superscalar, in-order execution pipeline
• DSP & NEON SIMD extensions per core
• VFPv4 Floating Point Unit per core
  – All key floating point functions implemented in HW (e.g. sqrt, abs, fmul, etc)
• HW virtualization
• Trust Zone security extensions
  – Advanced Encryption Standard (AES) (d)encryption
  – Secure Hash Algorithm (SHA)
• 64-byte cache lines; 10-entry L1 TLB, and 512-entry L2 TLB
• 4 Kb conditional branch predictor, 256-entry indirect branch predictor

Raspberry Pi3: ARMv8

• Addition of 64-bit support
  – Larger virtual address space
• New instruction set (A64)
  – Fewer conditional instructions
  – New instructions to support 64-bit operands
  – No arbitrary length load/store multiple instructions
• Enhanced cryptography (both 32 and 64-bit)
• Mostly backwards compatible with ARMv7
• Enable expansion into higher performance markets
  – Mobile phones, servers, supercomputers
  – Cortex-A53, Apple Cyclone, Nvidia Denver
Raspberry Pi Zero

- Half the size of Raspberry Pi 2 & 3
  - Dimensions: 65mm × 30mm × 5mm
- Less computing power with similar peripherals
  - 1GHz single-core ARM11 CPU (RPi3: Quadcore Cortex A53 1.2GHz)
  - 512MB RAM (RPi3: 1GB)
- CPU architecture: ARM1176JZF-S
  - ARMv6 ISA, only single core
  - 8-stage pipeline (no superscalar)
  - No NEON SIMD
GRAPHICS PROCESSING (GPU)
• Primary architectural elements:
  – Vertex shaders, pixel shaders (fragment shaders)
• Performance measured in GFLOPS
GPU programming

• OpenCL – general language used for CPU, GPU, DSP programming
  – Open standard with an emphasis on portability
  – Supported by Intel, AMD, Nvidia, and ARM

• Compute Unified Device Architecture (CUDA)
  – Enables GPGPUs: GPUs can be used for general purpose processing
  – Specific to NVIDIA
  – Single program multiple data (SPMD)
    • Threading directives and memory addressing

    ```c
    __global__
    void saxpy_parallel(int n,
                         float a,
                         float *x,
                         float *y)
    {
        int i = blockIdx.x * blockDim.x + threadIdx.x;
        if (i<n) y[i] = a*x[i] + y[i];
    }
    
    void saxpy_serial( int n,
                       float a,
                       float *x,
                       float *y)
    {
        for (int i=0; i<n; i++)
            y[i] = x*a[i] + y[i];
    }
    
    // perform SAXPY on 1M elements
    saxpy_serial(4096*256, 2.0, x, y);
    // perform SAXPY on 1M elements
    saxpy_parallel<<<4096,256>>>(n,2.0, x, y);
    ```
GPU programming

- A kernel scales across any number of parallel processors
  - Contains grid of thread blocks
- Thread block shape can be 1D, 2D, 3D
  - All threads in a thread block run the same code, over the same data in shared memory space
  - Threads have thread ID numbers within block to select work and address shared data
  - Threads in different blocks cannot cooperate
- Threads are grouped by warps (scheduling units)
Nvidia GeForce Block Diagram

- Ultra low power (ULP) version of GeForce used in Tegra 3, 4
- Desktop GTX 280 shown; embedded versions have different number of TPCs, SMs, etc
Qualcomm *Adreno 2xx*

- Snapdragon S1-4 chipsets (e.g. MSM8x60)
  - Adreno 5xx released in 2015+
- Unified shaders
  - Same instruction set for fragment and vertex processing
  - More versatile hardware
- 5-way VLIW
- Also used for non-gaming apps, e.g. browser
  - Behavior in non-gaming is not as well-understood
Qualcomm Adreno 2xx

- Snapdragon S4 chipset (MSM8960)
- “Digital core rail power” (red) includes GPU, video decode, & modem digital blocks
- GLBenchmark: high-end gaming content
  - CPU power up to 750mW @ 1.5 GHz
  - GPU power up to 1.6W @ 400MHz
Raspberry Pi 3: GPU

- GPU: VideoCore 4 Dual-core architecture at 400MHz
- A low-power mobile multimedia processor architecture
- Highly integrated with CPU, memory and display circuitry
- High power efficiency in driving fast off-chip buses
- Encoder and Decoder support
  - Hardware block assisting JPEG
  - 1080p30 Full HD H.264 Video
- Runs common libraries and drivers
  - OpenGL ES 2.0, OpenVG 1.1, Open EGL, OpenMAX
  - [http://elinux.org/Raspberry_Pi_VideoCore_APIs](http://elinux.org/Raspberry_Pi_VideoCore_APIs)
  - OpenGL:
    - cross-language, cross-platform application programming interface (API) for rendering 2D and 3D vector graphics
Raspberry Pi 3: VideoCore 4 GPU Architecture

- **GPU Core**: Parallel computing of video data at low clock speed
  - Coprocessor extension to the ARM architecture
  - Grouped by slices: share common cache resources
- **Quad Processor Unit (QPU)**: 4-way SIMD machine
  - Four-way physical parallelism
  - 16-way virtual parallelism (4-way multiplexed over four successive clock cycles)
  - Dual-issue floating-point ALU: one add and multiply per cycle
  - Raspberry Pi: 12 QPUs
- **Special functions units (SPU)**
  - SQRT, LOG, RECIPSQRT, EXP
- **Texture and memory units**
  - Specialized memory for 3D texture data
Raspberry Pi: VideoCore 4 GPU Pipeline

- GPU Pipeline: 8 stages
DIGITAL SIGNAL PROCESSING (DSP)
Digital signal processing (DSP)

- Processing or manipulation of signals using digital techniques
- Interfacing with the physical world
  - E.g. audio, digital images, speech processing, medical monitoring (EKG)

![Diagram of signal processing](image_url)
Fundamental DSP Operations

• Filtering
  – Finite impulse response (FIR)
• Frequency transforms
  – Fast Fourier (FFT)
  – Discrete cosine (DCT)
  – Inverse discrete cosine (IDCT)

\[ y(n) = \sum_{i=0}^{L-1} a_i x(n-i) \]

Pseudo C code:
```c
for (n=0; n<N; n++)
{
  s=0;
  for (i=0; i<L; i++)
  {
    s += a[i] * x[n-i];
  }
  y[n] = s;
}
```
DSP architectural features

• Fixed-point vs floating point
• VLIW or specialized SIMD techniques
  – E.g. Qualcomm Hexagon DSP (VLIW) dispatches up to 4 instructions to 4 execution units per cycle
• No virtual memory or context switching
• Separate instruction and data storage
  – Harvard architecture vs Von Neumann
• Pipelined FUs are integrated into the datapath

• Main DSP Manufacturers:
  – Texas Instruments (http://www.ti.com)
  – Motorola (http://www.motorola.com)
  – Analog Devices (http://www.analog.com)
What are DSPs Used For?

Speech processing
Audio effects
Image compression
Video encoding
Noise cancellation
Virtual/augmented reality
Actuation error detection
Speech processing

- Encoding
- Compression
- Synthesis
- Recognition

The blue sound wave is on the key again.

"k" in "key"
Image processing

- Trade off “good enough” quality in “essential” regions
- Enable higher transmission bandwidth, minimal storage, media interactivity

- Still image encoding: JPEG (Joint Photographic Experts Group)
  - JPEG2000: Wavelet Transform based
- Video encoding: MPEG (Moving Pictures Experts Group)
  - MPEG-4 (aka H.264)
    - Variable macroblock sizes (4x4 to 16x16)
    - Enhanced to allow lossless regions
MPEG: Group of Pictures (GOP)

- A structure of consecutive frames that can be decoded without any other reference frames
- Transmitted sequence is not the same as displayed sequence
- **Inter-frame** prediction/compression exploits temporal redundancy between neighboring frames
- **Intra-frame** coding is applied only to the current frame

Figure 1: Prediction between MPEG-2 Frames
Types of frames

• I frame (intra-coded)
  – Coded without reference to other frames
  – Begins each GOP

• P frame (predictive-coded)
  – Coded with reference to a previous reference frame (either I or P)
  – Size is usually about 1/3rd of an I frame

• B frame (bi-directional predictive-coded)
  – Coded with reference to both previous and future reference frames (either I or P)
  – Size is usually about 1/6th of an I frame
MPEG-2 Codec

Original video

Encoding

DCT → Quantization → Variable length coder (VLC) → Bitstream out

Intraframe

Interframe

- Motion compensation

+ Motion estimation

- Frame store

Inverse quantization

IDCT

Loop filter

Intraframe

Interframe
DCT/IDCT

- Used for lossy compression
- Discrete cosine transform (DCT)
  - Represent a finite sequence of data points with a sum of cosine (even) functions of different frequencies
  - Similar to the discrete Fourier transform (DFT)
  - If a coefficient has a lot of variance over a set, then it cannot be removed without affecting the picture quality
- Inverse DCT
  - Reconstruct sequence from frequency coefficients

Source: Xilinx
2D DCT & IDCT

- Image divided into macroblocks of 8x8 pixels
- DCT of each group is an 8x8 transform coefficient array; entries represent spatial frequencies

Source: Xilinx
DCT & IDCT operations

DCT:

\[
F[u, v] = \frac{1}{N^2} \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} f[m, n] \cos\left(\frac{(2m + 1)u\pi}{2N}\right) \cos\left(\frac{(2n + 1)v\pi}{2N}\right)
\]

where:
- \(u, v\) = discrete frequency variables (0, 1, 2, ..., \(N - 1\))
- \(f[m, n]\) = \(N\) by \(N\) image pixels (0, 1, 2, ..., \(N - 1\))
- \(F[u, v]\) = the DCT result

IDCT:

\[
f[m, n] = \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} c[u] c[v] F[u, v] \cos\left(\frac{(2m + 1)u\pi}{2N}\right) \cos\left(\frac{(2n + 1)v\pi}{2N}\right)
\]

- Dedicated functional unit: *fused multiply-add*
  - Common to DCT/IDCT & many other DSP operations
Fixed-Point Design

- Digital signal processing algorithms
  - Early development in floating point
  - Converted into fixed point for production to gain efficiency
- Fixed-point digital hardware
  - Lower area
  - Lower power
  - Lower per unit production cost

Copyright Kyungtae Han [2]
Fixed-Point Representation

\[ x = 0.5 \times 0.125 + 0.25 \times 0.125 \]
\[ = 0.0625 + 0.03125 \]
\[ = 0.09375 \]

• For integer word length \(iwl=1\) and fractional word length \(fwl=3\) decimal digits

• Less significant digits can be rounded or truncated

• Similar to a floating point system with numbers \(\in (-1..1)\), with no stored exponent (bits used to increase precision).

• Automatic scaling:
  – shifting after multiplications and divisions in order to maintain the binary point.
Fixed-Point Design

- All variables have to be annotated manually
  - Value ranges are well known
  - Avoid overflow
  - Minimize quantization effects
  - Find optimum wordlength

- Manual process supported by simulation
  - Time-consuming
  - Error prone
ALU interacts with accumulator, memory and registers in three different ways
Conventional DSP Architecture

- Multiply-accumulate (MAC) in 1 instruction cycle
- Harvard architecture for fast on-chip I/O
  - Data memory/bus separate from program memory/bus
  - One read from program memory per instruction cycle
  - Two reads/writes from/to data memory per inst. cycle
- Instructions to keep pipeline (3-6 stages) full
  - Zero-overhead looping (one pipeline flush to set up)
  - Delayed branches
- Special addressing modes supported in hardware
  - Bit-reversed addressing (e.g. fast Fourier transforms)
  - Modulo addressing for circular buffers (e.g. filters)
Buffering in DSPs

- **Buffer of length $K$**
  - Used in finite and infinite impulse response filters

- **Linear buffer**
  - Sort by time index
  - Update: discard oldest data, copy old data left, insert new data

- **Circular buffer**
  - Oldest data index
  - Update: insert new data at oldest index, update oldest index

### Data Shifting Using a Linear Buffer

<table>
<thead>
<tr>
<th>Time</th>
<th>Buffer contents</th>
<th>Next sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n=N$</td>
<td>$x_{N-K+1} \quad x_{N-K+2} \quad \cdots \quad x_{N-1} \quad x_N$</td>
<td>$x_{N+1}$</td>
</tr>
<tr>
<td>$n=N+1$</td>
<td>$x_{N-K+2} \quad x_{N-K+3} \quad \cdots \quad x_N \quad x_{N+1}$</td>
<td>$x_{N+2}$</td>
</tr>
<tr>
<td>$n=N+2$</td>
<td>$x_{N-K+3} \quad x_{N-K+4} \quad \cdots \quad x_{N+1} \quad x_{N+2}$</td>
<td>$x_{N+3}$</td>
</tr>
</tbody>
</table>

### Modulo Addressing Using a Circular Buffer

<table>
<thead>
<tr>
<th>Time</th>
<th>Buffer contents</th>
<th>Next sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n=N$</td>
<td>$x_{N-2} \quad x_{N-1} \quad x_N \quad x_{N-K+1} \quad x_{N-K+2}$</td>
<td>$x_{N+1}$</td>
</tr>
<tr>
<td>$n=N+1$</td>
<td>$x_{N-2} \quad x_{N-1} \quad x_N \quad x_{N+1} \quad x_{N-K+2} \quad x_{N-K+3}$</td>
<td>$x_{N+2}$</td>
</tr>
<tr>
<td>$n=N+2$</td>
<td>$x_{N-2} \quad x_{N-1} \quad x_N \quad x_{N+1} \quad x_{N+2} \quad x_{N-K+3} \quad x_{N-K+4}$</td>
<td>$x_{N+3}$</td>
</tr>
</tbody>
</table>
DSP Pipelining

**Sequential** *(Freescale 56000)*

- Fetch
- Decode
- Read
- Execute

**Pipelined** *(Most conventional DSPs)*

- Fetch
- Decode
- Read
- Execute

**Superscalar** *(Pentium)*

- Fetch
- Decode
- Read
- Execute

**Superpipelined** *(TMS320C6000)*

- Fetch
- Decode
- Read
- Execute

**Pipelining**
- Process instruction stream in stages (as stages of assembly on a manufacturing line)
- Increase throughput

**Managing Pipelines**
- Compiler or programmer
- Pipeline interlocking
RISC vs. DSP: Instruction Encoding

**RISC: Superscalar, out-of-order execution**

- Memory
- Reorder
- Floating-Point Unit
- Integer Unit
- Load/store

**DSP: Horizontal microcode, in-order execution**

- Memory
- Load/store
- ALU
- Multiplier
- Address
RISC vs. DSP: Memory Hierarchy

RISC
- Registers
- Out of order
- I/D Cache
- Physical memory
- TLB
- DMA Controller

DSP
- Registers
- I Cache
- Internal memories
- External memories
- DMA Controller

TLB: Translation Lookaside Buffer
DMA: Direct Memory Access
Conventional DSP Processor Families

- **Floating-point DSPs**
  - Used in initial prototyping of algorithms
  - Resurgence due to professional and car audio
- **Different on-chip configurations in each family**
  - Size and map of data and program memory
  - A/D, input/output buffers, interfaces, timers, and D/A
- **Drawbacks to conventional DSP processors**
  - No byte addressing (needed for images and video)
  - Limited on-chip memory
  - Limited addressable memory on fixed-point DSPs (exceptions include Freescale 56300 and TI C5409)
  - Non-standard C extensions for fixed-point data type

**DSP Market (est.)**

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fixed-point</strong></td>
</tr>
<tr>
<td><strong>Floating-point</strong></td>
</tr>
</tbody>
</table>
Microprocessors in Embedded Systems

Tajana Simunic Rosing
Department of Computer Science and Engineering
University of California, San Diego.
TI C54x family (CISC)

- **Modified Harvard architecture**: separate buses for program code and data
  - PB: program read bus
  - CB, DB: data read busses
  - EB: data write bus
  - PAB, CAB, DAB, EAB: address busses
- Can generate two data memory addresses per cycle
  - Stored in auxiliary register address units
- High performance, reproducible behavior, optimized for different memory structures
TI C54x architectural features

• 40-bit ALU & Barrel shifter
  – Input from accumulator or data memory
  – Output to ALU

• 17 x 17 multiplier

• Single-cycle exponent encoder

• Two address generators with dedicated registers

• Accumulators
  – Low-order (0-15), high-order (16-31), guard (32-39)
TI C54x instruction set features

• Compare, select and store unit (CSSU) unit
  – Compares high and low accumulator words
  – Accelerates Viterbi operations

• Repeat and block repeat instructions

• Instructions that read 2, 3 operands simultaneously

• Three IDLE instructions
  – Selectively shut down CPU, on-chip peripherals, whole chip including phase-locked loop
TI C54x pipeline

- **Prefetch**: Send PC address on program address bus
- **Fetch**: Load instruction from program bus to IR
- **Decode**
- **Access**: Put operand addresses on busses
- **Read**: Get operands from busses
- **Execute**
# Addressing Modes

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Operand field</th>
<th>Register-file contents</th>
<th>Memory contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immediate</td>
<td>Data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Register-direct</td>
<td>Register address</td>
<td>Data</td>
<td></td>
</tr>
<tr>
<td>Register indirect</td>
<td>Register address</td>
<td>Memory address</td>
<td>Data</td>
</tr>
<tr>
<td>Direct</td>
<td>Memory address</td>
<td>Data</td>
<td></td>
</tr>
<tr>
<td>Indirect</td>
<td>Memory address</td>
<td>Memory address</td>
<td>Data</td>
</tr>
</tbody>
</table>
TIs C55x Pipeline

- Prefetch 1:
  - Send address to memory
- Prefetch 2:
  - Wait for response
- Fetch:
  - Get instruction from memory and put in IBQ
- Predecode:
  - Identify where instructions begin and end
  - Identify parallel instructions
- Decode:
  - Decode an instruction pair or single instruction.
- Address:
  - Perform address calculations.
- Access 1/2:
  - Send address to memory; wait.
- Read:
  - Read data from memory. Evaluate condition registers.
- Execute:
  - Read/modify registers. Set conditions.
- W/W+:
  - Write data to MMR-addressed registers, memory; finish.

[Diagram showing the pipeline stages: fetch, execute, with durations 4 and 7-8]
C55x organization

3 data read busses
3 data read address busses
program address bus
program read bus
2 data write busses
2 data write address busses

Instruction unit
Program flow unit
Address unit
Data unit

32
16
24
24
16
24
C55x hardware extensions

• Target image/video applications
  – DCT/IDCT
  – Pixel interpolation
  – Motion estimation

• Available in 5509 and 5510
  – Equivalent C-callable functions for other devices.
TI C62/C67 (VLIW)

- Up to 8 instructions/cycle
- 32 32-bit registers
- Function units
  - Two multipliers
  - Six ALUs
- Data operations
  - 8/16/32-bit arithmetic
  - 40-bit operations
  - Bit manipulation operations
TI C6000 DSP Architecture

Simplified Architecture

Program RAM or Cache

Data RAM

Internal Buses

Regs (A0-A15)

Regs (B0-B15)

Control Regs

CPU

External Memory
- Sync
- Async

Addr

Data

DMA

Serial Port

Host Port

Boot Load

Timers

Pwr Down
C6x functional units

- **.L**
  - 32/40-bit arithmetic
  - Leftmost 1 counting
  - Logical ops
- **.S**
  - 32-bit arithmetic
  - 32/40-bit shift and 32-bit field
  - Branches
  - Constants
- **.M**
  - 16 x 16 multiply
- **.D**
  - 32-bit add, subtract, circular address
  - Load, store with 5/15-bit constant offset
C6x system

- On-chip RAM
- 32-bit external memory: SDRAM, SRAM
- Host port
- Multiple serial ports
- Multichannel DMA
- 32-bit timer
- Families: All support same C6000 instruction set
  - C6200 fixed-pt. 150- 300 MHz ADSL, printers
  - C6400 fixed pt. 300-1,000 MHz video, wireless base stations
  - C6700 floating 100- 300 MHz medical imaging, pro-audio
### C6000 Instruction Set by Functional Unit

<table>
<thead>
<tr>
<th><strong>.S Unit</strong></th>
<th><strong>.L Unit</strong></th>
<th><strong>.D Unit</strong></th>
<th><strong>.M Unit</strong></th>
<th><strong>Other</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>ABS</td>
<td>ADD</td>
<td>MPY</td>
<td>NOP</td>
</tr>
<tr>
<td>ADDK</td>
<td>NOT</td>
<td>ADDA</td>
<td>SMPY</td>
<td>IDLE</td>
</tr>
<tr>
<td>ADD2</td>
<td>OR</td>
<td>SUB</td>
<td>MPYH</td>
<td></td>
</tr>
<tr>
<td>AND</td>
<td>SET</td>
<td>SADD</td>
<td>SMPYH</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>SHL</td>
<td>SAT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLR</td>
<td>SHR</td>
<td>SSUB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXT</td>
<td>SSHL</td>
<td>SUBC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MV</td>
<td>SUB</td>
<td>XOR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MVC</td>
<td>SUB2</td>
<td>ZERO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MVK</td>
<td>XOR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MVKH</td>
<td>ZERO</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Six of the eight functional units can perform integer add, subtract, and move operations.
## TI TMS320C6000 Instruction Set

### Arithmetic
- ABS
- ADD
- ADDA
- ADDK
- ADD2
- MPY
- MPYH
- NEG
- SMPY
- SMPYH
- SADD
- SAT
- SSUB
- SUB
- SUBA
- SUBC
- SUB2
- ZERO

### Logical
- AND
- CMPEQ
- CMPGT
- CMPLT
- NOT
- OR
- SHL
- SHR
- SSHL
- XOR

### Data Management
- LD
- MV
- MVC
- MVK
- MVKH
- ST

### Bit Management
- CLR
- EXT
- LMBD
- NORM
- SET

### Program Control
- B
- IDLE
- NOP

### C6000 Instruction Set by Category
- (un)signed multiplication
- saturation/packed arithmetic
# TI C6000 vs. C5000 Addressing Modes

<table>
<thead>
<tr>
<th>Immediate</th>
<th>TI C5000</th>
<th>TI C6000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ADD #0FFh</td>
<td>add .L1</td>
</tr>
<tr>
<td></td>
<td>(implied)</td>
<td>-13,A1,A6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Register</th>
<th>TI C5000</th>
<th>TI C6000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ADD 010h</td>
<td>add .L1 A7,A6,A7</td>
</tr>
<tr>
<td></td>
<td>(implied)</td>
<td>not supported</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Direct</th>
<th>TI C5000</th>
<th>TI C6000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ADD *</td>
<td>ldw .D1 *A5++[8],A1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Indirect</th>
<th>TI C5000</th>
<th>TI C6000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Programmable Logic Devices (PLDs)
Programmable Logic Devices

• Simple PLD (SPLD)
  – Programmable logic array (PLA)
  – Programmable array logic (PAL) – fixed OR plane
• Complex PLD (CPLD)
  – Building block: macrocells
• Field-Programmable Gate Arrays (FPGA) tech:
  – Fuse/Antifuse PLD
  – Erasable EPLD & EEPLD
  – SRAM based

<table>
<thead>
<tr>
<th>Name</th>
<th>Re-programmable</th>
<th>Volatile</th>
<th>Technology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fuse</td>
<td>No</td>
<td>No</td>
<td>Bipolar</td>
</tr>
<tr>
<td>EPROM</td>
<td>Yes – out of circuit</td>
<td>No</td>
<td>UVCMOS</td>
</tr>
<tr>
<td>EEPROM</td>
<td>Yes – in circuit</td>
<td>No</td>
<td>EECMOS</td>
</tr>
<tr>
<td>SRAM</td>
<td>Yes – in circuit</td>
<td>Yes</td>
<td>CMOS</td>
</tr>
<tr>
<td>Antifuse</td>
<td>No</td>
<td>No</td>
<td>CMOS+</td>
</tr>
</tbody>
</table>
Antifuse PLDs

- **Actel Axcelerator family**

  - **Antifuse**:  
    - open when not programmed  
    - Low resistance when programmed
Erasable Programmable Logic Devices

- Erasable programmable ROM (EPROM)
  - erased by UV light
- Altera’s building block is a MACROCELL

8 Product Term AND-OR Array + Programmable MUX’s

Programmable polarity

Programmable feedback

Seq. Logic Block

I/O Pin

pad

CLK

AND ARRAY

MUX

MUX

Q

F/B MUX

Output MUX

Invert Control

Clk MUX

Erasable programmable ROM (EPROM)
- erased by UV light

Altera’s building block is a MACROCELL

Erasable programmable ROM (EPROM)
- erased by UV light

Altera’s building block is a MACROCELL
Complex Programmable Logic Devices (CPLD)

- Altera *Multiple Array Matrix (MAX)* architecture
- AND-OR structures are relatively limited, cannot share signals/product terms among macrocells

**EPM5128:**
- 8 Fixed Inputs
- 52 I/O Pins
- 8 LABs
- 16 Macrocells/LAB
- 32 Expanders/LAB
Altera MAX 7k (EEPLD)
Logic Block

Electrically Erasable Programmable ROM: EEPROM
SRAM based PLD

- Altera Flex 10k Block Diagram
SRAM based PLD

- Altera Flex 10k Logic Array Block (LAB)
SRAM based PLD

- Altera *Flex 10k* Logic Element (LE)
SRAM parts with DSP

• Altera *Stratix II*: Block Diagram
FPGA with DSP

- Altera *Stratix II* DSP block
Application-specific integrated circuit (ASICs)

• Custom integrated circuits that have been designed for a single use or application
• Standard single-purpose processors
  – “Off-the-shelf”, pre-designed for a common task (e.g. peripherals)
  – serial transmission
  – analog/digital conversions
Combined system: CPU+FPGA+ASICs

- **Actel Fusion Family**
  - ARM7 CPU with FPGA and ASIC implementations of “smart peripherals” for analog functions
Summary

• Processor metrics, trends
• Architectures and functions
  – CPUs
  – GPU
  – DSP
• Implementations
  – Programmable logic – PLDs and FPGAs
  – Custom ASICs
Sources and References


• Brian L. Evans, Embedded Signal Processing Laboratory, UT Austin.