ARM® Cortex M0
Design Start

Karthik Shivashankar
Senior Engineer, ARM R&D
Cambridge, UK
ARM Ltd

- Founded in November 1990
  - Spun out of Acorn Computers
  - Initial funding from Apple, Acorn and VLSI

- Designs the ARM range of RISC processor cores
  - Licenses ARM core designs to semiconductor partners who fabricate and sell to their customers
  - **ARM does not fabricate silicon itself**

- Also develop technologies to assist with the design-in of the ARM architecture
  - Software tools, boards, debug hardware
  - Application software
  - Bus architectures
  - Peripherals, etc
ARM’s Activities

Processors
System Level IP:
  Data Engines
  Fabric
  3D Graphics

Physical IP

Connected Community
Development Tools
Software IP
ARM Processor

- Classic ARM Processors
  - ARM7
  - ARM9
  - ARM11

- Embedded Cortex Processors
  - Cortex-M0
  - Cortex-M0+
  - Cortex-M1
  - Cortex-M3
  - Cortex-M4
  - Cortex-R4
  - Cortex-R5
  - Cortex-R7

- Application Cortex Processors
  - Cortex-A7
  - Cortex-A8
  - Cortex-A9
  - Cortex-A15

The Architecture for the Digital World®
How many ARM’s Do You Have?

Mobile phones
- ~100% market share

Smartphones
- 3x 100% market share

Mobile Computers
- 5x 100% market share

Digital TVs
- 30% market share

Disk Drives
- ~70% market share

PC Peripherals
- 30% market share

Cars
- 5x 40% market share

Microcontrollers
- 10% market share
M-Class ARM Processors

- Cortex-M0
- Cortex-M1
- Cortex-M3
- Cortex-M4
Cortex-M0 DesignStart

- Fixed and simplest configuration
- Obfuscated, but synthesizable verilog netlist
- Available through ARM Design Start portal
- No initial licensing fee
How to Access

- **Cortex M0 – Design Start**

- **Keil MDK (For Software Development)**

- **Example Design Kit**
  - A full working system as a starting point (HW + SW)
  - Essential AMBA interconnects and peripherals (with source code)
  - Example programs for Keil MDK
  - Working prototypes on an FPGA board
Hello World !!
Complete SoC

ARM Processor (CM0-DS)

VGA
UART
GPIO
PS2-KB
LED
7SEG

VGA
UART
Switches & LED
PS2
LEDs
7-SEG Display

PSRAM (16MB)
FLASH (16 MB)
An Example AMBA AHB-Lite System
ARM R&D Testchip using CM0-DS
MIT Project

FPGA - High Level Block Diagram

Arm Core

AHB Control Logic

PS Ram Controller

Fast Ram

FFT

7 Seg Decoder

GPIO Controller

UART/USB

VGA Controller

UART

Timer

Peak Detection

SPI Controller
Cortex M0 Products – LPC1100

LPC1100L series

The LPC1100L series is the lowest-priced 32-bit MCU solution on the market. It is a high-value, easy-to-use upgrade for existing 8/16-bit designs, delivering unprecedented performance, simplicity, and power. The optimized Thumb instruction set also enables dramatic reductions in code size for most 8/16-bit applications. The LPC1100L is a seamless entry point for 8/16-bit designers looking to start using the scalable ARM architecture.

Features

- 50 MHz ARM Cortex-M0 core
- Up to 32 KB Flash
- Up to 8 KB SRAM
- Serial peripherals: I^2C Fast-mode Plus, two SPI, UART
- 8-channel, 10-bit ADC
- Lowest active power consumption (only 130 µA/MHz)
- Up to 42 high-speed GPIO
- Superior code density compared to traditional 8/16-bit MCUs
- Supported by NXP’s low-cost LPCXpresso toolchain
Cortex M0 Products – LPC1102

LPC1102 block diagram

LPC1102: 5 mm² footprint

The LPC1102, the first in a series of devices housed in Wafer Level Chip Scale Packages (WL-CSPs), offers unprecedented computing power in just 5 mm² of PCB area. Designed for applications requiring an ultra-miniature board layout, it offers true 32-bit performance (50 MHz) and offers a far higher memory configuration (32 KB Flash) than typical 8/16-bit solutions.

Features

- 50 MHz ARM Cortex-M0 core
- 32 KB Flash
- 8 KB SRAM
- Serial peripherals: SPI, UART
- 5-channel, 10-bit ADC
- Two 32-bit timers, two 16-bit timers, Systick timer, WDT
- Lower dynamic power, leading to overall reduced power consumption
- 11 high-speed GPIO
- WL-CSP package (2.17 x 2.32 mm, 0.5 mm pitch)
- Superior code density compared to traditional 8/16-bit MCUs
- Supported by NXP’s low-cost LPCXpresso toolchain
- Best Product for Embedded Systems & ICs (2010 EDN China Innovation)

World’s Smallest 32 bit MCU
ARMSOC_1 (Hello World)
Section 1

CORTEX MO
ARM Cortex-M0 Processor

- ARMv6-M Architecture
  - Von-Neumann Architecture
  - 32-bit Architecture
  - Thumb technology
- Nested Vector Interrupt Controller (NVIC)
- AHB-Lite Master Interface
- Optional CoreSight-compliant Debug
- Ultra-low Power Support
- RTL is configurable
- Synthesizable
  - Gate count 12 ~ 25K
Cortex-M0 DesignStart Processor

- ARMv6-M Architecture
  - Von-Neumann Architecture
  - 32-bit Architecture
  - Thumb technology
- Nested Vector Interrupt Controller (NVIC)
- AHB-Lite Master Interface
  - Optional CoreSight-compliant Debug
  - Ultra-low Power Support
  - RTL is configurable
- Synthesizable
  - Gate count 12 ~ 25K
**Integer Core Pipeline**

- 3-stage pipeline core with von Neumann architecture

- An instruction is advanced in the pipeline only when another instruction is executed

- PFU fetches instructions
  - Fetches instructions in 32-bit quantities
  - 16 bit instructions are buffered for later execution
  - Instruction data is usually only read from AHB every other cycle
Exceptions & Interrupts

- **NVIC enables efficient and low latency exception handling**
  - Nested Vectored Interrupt Controller
  - Hardware stacking of state on interrupt entry
  - 16 clock interrupt latency

- **The NVIC includes support for**
  - Prioritization of exceptions
  - Tail-chaining & Late arriving interrupts
  - Level and Pulse interrupts

- **Configurable deterministic exception handling timing behavior**
  - Takes the same number of cycles to handle an exception*
  - Provides jitter-free interrupt response to one high-priority interrupt
  - Useful in some state machine replacement scenarios

*Assumes the same conditions and a deterministic memory system
# Instruction Cycle Timing

<table>
<thead>
<tr>
<th>Duration</th>
<th>Description</th>
</tr>
</thead>
</table>
| 1 cycle    | All data-processing operations (without PC as destination - ADD, SUB, MOV, NOP)  
                      All 16-bit Thumb branch instructions (when not taken) |
| 2 cycles   | All single-element load or store operations (LDR/STR)                      
                      Wait for interrupt or event (WFI, WFE) |
| 3 cycles   | All 16-bit Thumb branch instructions (when taken)                          
                      Data-processing operations where PC is the destination register |
| 4 cycles   | All 32-bit Thumb instructions (BL, DMB, DSB, ISB, MSR, MRS)               |
| 1+N        | Multiple load and stores containing N elements (without POP with PC in list)  
                      LDM, STM, POP and PUSH |
| 4+N        | POP with PC in list                                                        |
| 1 or 32 cycles | Multiplication (MULS)                                                   |

- Zero wait state memory system assumed
ARM-V6M PROGRAMMERS MODEL
Programmers Registers

- All registers are 32 bits wide

- 13 general purpose registers
  - Registers r0 – r7 (Low registers)
  - Registers r8 – r12 (High registers)

- 3 registers with special meaning/usage
  - Stack Pointer (SP) – r13
  - Link Register (LR) – r14
  - Program Counter (PC) – r15

- Special-purpose registers
  - xPSR shows a composite of the content of
    - APSR, IPSR, EPSR

Low Registers

High Registers

Program Status Registers
Stack Pointer (SP)

- Usage of a stack is to save register contents in memory
  - The content stored on a stack can be restored for later usage
  - The Stack Pointer (SP) points to a memory location – the stack

![Diagram of push and pop operations on a stack]

- Push Operation
  - Register with address 0x12345678
  - Memory location with address 0xABCDABCD

- Pop Operation
  - Register with address 0x12345678
  - Memory location with address 0xABCDABCD
Link Register (LR)

- The Link Register (LR) is used to enable returns from subroutines

```c
void func0 (void) {
    : func1();
    :
}
```

- Further usage of the LR
  - It has a special function for exception handling
Program Counter (PC)

- The Program Counter (PC) points to the instruction in memory which is to be loaded next.

- Changing the PC will change the flow of the program.
The PSR Registers

- APSR - Application Program Status Register

- Contains the Negative, Zero, Carry and Overflow flags from the ALU

- IPSR – Interrupt Program Status Register

- EPSR – Execution Program Status Register

- Thumb code is executed
The PSR Composite Registers

- **xPSR**
  - Composite register of APSR, IPSR and EPSR

  ![PSR Composite Registers](image)

- **IEPSR**
  - Composite register of IPSR and EPSR

  ![IEPSR Composite Registers](image)
Modes Overview

ARM Processor

Application Code

Thread Mode

Exception Entry

Exception Code

Handler Mode

Exception Return

Reset

Not shown: Handler mode can also be re-entered on exception return
Modes

- **Thread Mode**
  - Used for application execution
  - Selected on reset
  - Selected on exception return
    - Apart from nested exceptions

- **Handler Mode**
  - Used for exception handling
  - Entered on an exception

- Both modes have full access to all system resources
  - No concept of privilege
Exceptions

- A condition that changes the normal flow of control in a program

- 4 classes of exceptions are supported
  - Reset
  - Fault
    - HardFault
  - Interrupt
    - Non-Maskable Interrupt (NMI)
    - Interrupts (IRQs)
    - SysTick (optional)
    - PendSV
  - Supervisor Call
    - SVCall
Instruction Set Introduction

- ARMv6-M supports Thumb-2 technology
  - A subset of the full Thumb-2 instruction set is supported
  - The ARM instruction set is not supported

- Thumb-2 technology supports mixed 16-bit/32-bit instructions

- Small number of additional 32-bit instructions supported
  - Instructions introduced with the Thumb-2 technology

- Conditional execution is supported
  - Only one conditional instruction

- Optimized for compilation from C
  - Thumb-2 instructions are not designed to be written by hand
  - Easy to learn due to small number of mnemonics
CM0 Instruction Set

![CM0 Instruction Set Diagram](image)
Binary Upwards Compatibility

The Architecture for the Digital World®

ARM®
Instruction Classes

- Branch instructions
- Data-processing instructions
- Load and store instructions
- Status register access instructions
- Miscellaneous instructions
Branch Instructions

- **B** – Branch
  - Absolute branch to a target address, relative to Program Counter (PC)
  - +/- 256 bytes range, conditional execution supported
  - +/- 1MB range, no conditional execution supported

- **BL** – Branch with Link
  - Branch to a subroutine – Link register is updated
  - +/- 16MB range, relative to Program Counter (PC)
Branch Instructions with Exchange

- **BX** – Branch and Exchange
  - Branch to a target address
  - Any range within 4GB
- **BLX** – Branch with Link and Exchange
  - Branch to a subroutine – Link register (LR) is updated
  - Any range within 4GB

Interworking

BLX/BX

<table>
<thead>
<tr>
<th>Rm</th>
<th>1</th>
</tr>
</thead>
</table>

Thumb Instruction Set

ARM Instruction Set

BLX/BX

| Rm | 0 |

to the ARM Instruction Set results in an exception in ARMv6-M.
# Data Processing Instructions

- **Standard Data Processing Instructions**
  - ADD, ADC, SUB, SBC, RSB
  - AND, ORR, EOR, BIC
  - MOV, MVN
  - TST, CMP, CMN
  - ADR (Pseudo Instruction)

- **Shift Instructions**
  - ASR
  - LSL, LSR
  - ROR

- **Multiply Instruction**
  - MUL

- **Packing/Unpacking Instructions**
  - SXTB, SXTH, UXTB, UXTH

- **Miscellaneous Data Processing**
  - REV, REV16, REVSH

### Examples

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBS r0,#1</td>
<td>(r0 ← r0 - 1)</td>
</tr>
<tr>
<td>ORRS r0,r1</td>
<td>(r0 ← r0</td>
</tr>
<tr>
<td>MOVs r0,#1</td>
<td>(r0 ← r0 + 1)</td>
</tr>
<tr>
<td>CMP r0,r1</td>
<td></td>
</tr>
<tr>
<td>RSBS r0,r1,#0</td>
<td>(r0 ← -r1)</td>
</tr>
<tr>
<td>ASRS r0,r1,#7</td>
<td>(r0 ← r1 &gt;&gt; 7)</td>
</tr>
<tr>
<td>LSLS r0,r1,#3</td>
<td>(r0 ← r1 &lt;&lt; 3)</td>
</tr>
<tr>
<td>RORS r0,r1</td>
<td>(r0 ← r0 &gt;&gt; r1)</td>
</tr>
<tr>
<td>MULS r0,r1,r0</td>
<td>(r0 ← r1 * r0)</td>
</tr>
<tr>
<td>UXTB r0,r1</td>
<td>(r0 ← r1[7:0])</td>
</tr>
<tr>
<td>REV r0,r1</td>
<td>Byte Swap</td>
</tr>
</tbody>
</table>
Status Register Access Instructions

- **MRS/MSR** - Move data between a general purpose register and status register
  - **MRS** (Register ← Status Register)
  - **MSR** (Status Register ← Register)

  Examples
  
  ```
  MRS r0, IPSR  (r0 ← IPSR)
  MSR APSR, r0  (APSR ← r0)
  ```

- **CPS** – Change Processor State
  - Allows the enable/disable interrupts

  Examples
  
  ```
  CPSIE  i     (CPS Interrupt Enable)
  CPSID  i     (CPS Interrupt Disable)
  ```
Miscellaneous Instructions

- Memory Barriers Instructions
  - **DMB** Data Memory Barrier
  - **DSB** Data Synchronisation Barrier
  - **ISB** Instruction Synchronisation Barrier

- Hint instructions
  - **SEV** Send Event
  - **WFE** Wait for Event
  - **WFI** Wait for Interrupt
  - **NOP** No Operation

- Supervisor Call Instruction
  - **SVC** Generates a SVCall exception
    Number used for identification

- NOP Compatible Instructions
  - Executed as a NOP when not implemented

Example

```
SVC #0
```
Conditional Execution

- APSR condition code flags are used to decide if a branch instruction is executed
  - Condition code flags are updated from previous code execution
  - With the ‘S’ suffix included the APSR flags are updated

- Conditional execution is only supported using 16-bit branch instructions
  - B<c> addr
## Condition Codes

<table>
<thead>
<tr>
<th>Condition Code</th>
<th>Interpretation</th>
<th>Status Flag State</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQ</td>
<td>Equal / equals zero</td>
<td>Z set</td>
</tr>
<tr>
<td>NE</td>
<td>Not equal</td>
<td>Z clear</td>
</tr>
<tr>
<td>CS / HS</td>
<td>Carry set / unsigned higher or same</td>
<td>C set</td>
</tr>
<tr>
<td>CC / LO</td>
<td>Carry clear / unsigned lower</td>
<td>C clear</td>
</tr>
<tr>
<td>MI</td>
<td>Minus / negative</td>
<td>N set</td>
</tr>
<tr>
<td>PL</td>
<td>Plus / positive or zero</td>
<td>N clear</td>
</tr>
<tr>
<td>VS</td>
<td>Overflow</td>
<td>V set</td>
</tr>
<tr>
<td>VC</td>
<td>No overflow</td>
<td>V clear</td>
</tr>
<tr>
<td>HI</td>
<td>Unsigned higher</td>
<td>C set and Z clear</td>
</tr>
<tr>
<td>LS</td>
<td>Unsigned lower or same</td>
<td>C clear or Z set</td>
</tr>
<tr>
<td>GE</td>
<td>Signed greater than or equal</td>
<td>N equals V</td>
</tr>
<tr>
<td>LT</td>
<td>Signed less than</td>
<td>N is not equal to V</td>
</tr>
<tr>
<td>GT</td>
<td>Signed greater than</td>
<td>Z clear and N equals V</td>
</tr>
<tr>
<td>LE</td>
<td>Signed less than or equal</td>
<td>Z set or N is not equal to V</td>
</tr>
<tr>
<td>AL</td>
<td>Always (optional)</td>
<td>Any</td>
</tr>
</tbody>
</table>
Load and Store Instructions

- **Unsigned Loads/Stores**
  - LDR/STR
  - LDRH/STRH
  - LDRB/STRB

- **Signed Loads**
  - LDRSH
  - LDRSB

- **Load/Stores Multiple**
  - LDM, LDMIA, LDMFD
  - STM, STMIA, STMEA
  - PUSH/POP

Examples:
- LDR r0,[r1] \(\rightarrow [r1]\)
- STM r0!,{r1,r2} \((r1 \rightarrow [r0]) (r2 \rightarrow [r0+4])\)
- LDM r0!,{r1,r2} \((r1 \leftarrow [r0]) (r2 \leftarrow [r0+4])\)
- PUSH {r1,r2} \((r1 \rightarrow [SP], r2 \rightarrow [SP+4])\)
- POP {r1,r2} \((r1 \leftarrow [SP], r2 \leftarrow [SP-4])\)
ARM-V6M MEMORY ARCHITECTURE
### Address Map Overview

<table>
<thead>
<tr>
<th>Address Range</th>
<th>Name</th>
<th>Device Type</th>
<th>XN</th>
<th>Cache</th>
<th>Supported Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFFFFFFFF - 0xE0000000</td>
<td>System</td>
<td>Strongly Ordered</td>
<td>XN</td>
<td>-</td>
<td>System segment, PPB</td>
</tr>
<tr>
<td>0xDFFFFFFF - 0xC0000000</td>
<td>Device</td>
<td>Device</td>
<td>XN</td>
<td>-</td>
<td>Non-shareable memory</td>
</tr>
<tr>
<td>0xBFFFFFFF - 0xA0000000</td>
<td>Device</td>
<td>Device, Shareable</td>
<td>XN</td>
<td>-</td>
<td>Shareable memory</td>
</tr>
<tr>
<td>0x9FFFFFFF - 0x80000000</td>
<td>RAM</td>
<td>Normal</td>
<td></td>
<td>WT</td>
<td>Memory with WT cache attributes</td>
</tr>
<tr>
<td>0x7FFFFFFF - 0x60000000</td>
<td>RAM</td>
<td>Normal</td>
<td></td>
<td>WBWA</td>
<td>Write-back, Write-allocate L2/L3</td>
</tr>
<tr>
<td>0x5FFFFFFF - 0x40000000</td>
<td>Peripheral</td>
<td>Device</td>
<td>XN</td>
<td>-</td>
<td>On-chip peripheral address space</td>
</tr>
<tr>
<td>0x3FFFFFFF - 0x20000000</td>
<td>SRAM</td>
<td>Normal</td>
<td></td>
<td>WBWA</td>
<td>SRAM</td>
</tr>
<tr>
<td>0x1FFFFFFF - 0x00000000</td>
<td>Code</td>
<td>Normal</td>
<td></td>
<td>WT</td>
<td>ROM</td>
</tr>
</tbody>
</table>

**Notes:**
- WT: Write-through
- WBWA: Write-back, Write-allocate
- XN: Xenon ordering
- System: System segment
- Strongly Ordered: Strongly ordered memory
- Non-shareable: Non-shareable memory
- Shareable: Shareable memory
- Memory: Memory with WT cache attributes
- On-chip: On-chip address space
- SRAM: SRAM
- On-chip RAM: On-chip RAM
- ROM: ROM
- Flash Memory: Flash Memory
System Segment

- Segment for control & configuration of the processor
  - Including resources like NVIC, System Timer or Debug

- Processor has only limited view of this space

- Top of memory can be used for adding additional implementation defined system space
Private Peripheral Bus (PPB)

- Always accessed as little endian, regardless of the endianness state
- Only supports aligned word accesses
  - Byte/halfword or unaligned accesses are not supported
System Control Space (SCS)

- 4kB address space within the PPB
- Provides arrays of 32-bit registers
  - Configuration
  - Status
  - Control
System Control Block (SCB)

- Provides configuration registers for the processor

```
+----------------------------------+
|   0xE000ED90                    |
| System Control Block (SCB)      |
| 0xE000ED00                      |
+----------------------------------+

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>DFSR</td>
<td></td>
</tr>
<tr>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>SHCSR</td>
<td></td>
</tr>
<tr>
<td>SHPR3</td>
<td></td>
</tr>
<tr>
<td>SHPR2</td>
<td></td>
</tr>
<tr>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>CCR</td>
<td></td>
</tr>
<tr>
<td>SCR</td>
<td></td>
</tr>
<tr>
<td>AIRCR</td>
<td></td>
</tr>
<tr>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>ICSR</td>
<td></td>
</tr>
<tr>
<td>CPUID</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
## SCB Registers Overview

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>DFSR</td>
<td>Debug Fault Status Register</td>
</tr>
<tr>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>SHCSR</td>
<td>System Handler Control and State Register</td>
</tr>
<tr>
<td>SHPR3</td>
<td>System Handler Priority Register 3</td>
</tr>
<tr>
<td>SHPR2</td>
<td>System Handler Priority Register 2</td>
</tr>
<tr>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>CCR</td>
<td>Configuration &amp; Control Register</td>
</tr>
<tr>
<td>SCR</td>
<td>System Control Register</td>
</tr>
<tr>
<td>AIRCR</td>
<td>Application Interrupt and Reset Control Register</td>
</tr>
<tr>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>ICSR</td>
<td>Interrupt Control State Register</td>
</tr>
<tr>
<td>CPUID</td>
<td>CPU Identification</td>
</tr>
</tbody>
</table>

The Architecture for the Digital World®
Vector Table

- Vector Table Start at 0x0
- It Defines
  - Initial Value for MSP
  - Starting address of program execution
  - Starting address of exception handlers
  - Each Vector is located at address = Exception Number x 4
  - The size of vector table depends on number of exceptions implemented.
What happens on Reset

- Access first two words from the vector table
  - 0x0 → Main stack pointer
  - 0x4 → Reset Vector
- Initialise Main Stack Pointer (MSP)
- Initialise Program Counter (PC)
Special Note

- Vectors are address but not branch instructions. This is different from traditional ARM processors.
- The LSB of the vectors are **always** set to “1”, indicating that the exception handlers are Thumb code.
- If the LSB is set to “0” then the processor will raise a fault exception and locks up.
- First two vectors are must
  - Stack Pointer
  - Reset Handler
- But NMI and Fault Exception handler is recommended.
Keil Demo
Further Reading

THE DEFINITIVE GUIDE TO THE ARM® CORTEX™-M0

Joseph Yiu
Section 2

CORTEX MO – SYSTEM DESIGN