ARM Cortex-M0 DesignStart
Processor and v6-M Architecture

Joe Bungo
ARM University Program Manager – Americas/Europe
R&D Division
The ARM University Program — Preparing Today’s Students for Tomorrow’s Technology


The World's Leading Semiconductor IP Supplier
Agenda

Introduction to ARM Ltd
- Cortex-M0 DesignStart Processor
- ARM v6-M Programmers Model
- ARM v6-M Exception Handling
- ARM v6-M Instruction Set Overview
- Pipeline
- M0_DS EDK Example System
ARM Ltd

ARM founded in November 1990
- Advanced RISC Machines

Company headquarters in Cambridge, UK
- Processor design centers in Cambridge, Austin, and Sophia Antipolis
- Sales, support, and engineering offices all over the world

Best known for its range of RISC processor cores designs
- Other products – fabric IP, software tools, models, cell libraries -
- to help partners develop and ship ARM-based SoCs

ARM does not manufacture silicon

More information about ARM and our offices on our web site: www.arm.com
ARM Offices Worldwide
ARM’s Activities

Connected Community
Development Tools
Software IP

Processors
System Level IP:
Data Engines
Fabric
3D Graphics

Physical IP
ARM Connected Community – 800+
Huge Range of Applications

Equipment Adopting 32-bit ARM Microcontrollers
Huge Opportunity For ARM Technology

30+ billion cores to date

100+ billion cores accumulated after next 9 yrs

1998 2013 2020

The Architecture for the Digital World®
ARM Cortex Advanced Processors

- **ARM Cortex™-A family:**
  - Applications processors for feature-rich OS and 3rd party applications

- **ARM Cortex-R family:**
  - Embedded processors for real-time signal processing, control applications

- **ARM Cortex-M family:**
  - Microcontroller-oriented processors for MCU, ASSP, and SoC applications
Relative Performance*

*Represents attainable speeds in 130, 90, 65, or 45nm processes

<table>
<thead>
<tr>
<th></th>
<th>Max Freq (MHz)</th>
<th>Min Power (mW/MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cortex-M0</td>
<td>50</td>
<td>0.012</td>
</tr>
<tr>
<td>Cortex-M3</td>
<td>150</td>
<td>0.06</td>
</tr>
<tr>
<td>ARM7</td>
<td>184</td>
<td>0.35</td>
</tr>
<tr>
<td>ARM926</td>
<td>470</td>
<td>0.235</td>
</tr>
<tr>
<td>ARM1026</td>
<td>540</td>
<td>0.36</td>
</tr>
<tr>
<td>ARM1136</td>
<td>610</td>
<td>0.335</td>
</tr>
<tr>
<td>ARM1176</td>
<td>750</td>
<td>0.568</td>
</tr>
<tr>
<td>Cortex-A8</td>
<td>1100</td>
<td>0.43</td>
</tr>
<tr>
<td>Cortex-A9 Dual-core</td>
<td>2000</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Max Frequency (MHz)
Agenda

- Introduction to ARM Ltd
- Cortex-M0 DesignStart Processor
  - ARM v6-M Programmers Model
  - ARM v6-M Exception Handling
  - ARM v6-M Instruction Set Overview
  - Pipeline
  - M0_DS EDK Example System
Cortex family

**Cortex-A8**
- Architecture v7A
- MMU
- AXI
- VFP & NEON support

**Cortex-R4**
- Architecture v7R
- MPU (optional)
- AXI
- Dual Issue

**Cortex-M3**
- Architecture v7M
- MPU (optional)
- AHB Lite & APB
Cortex-M0 DesignStart Processor

Cortex™-M0
DesignStart

Nested Vectored Interrupt Controller

CPU

AHB-lite Interface
### Cortex-M0 DesignStart Differences

<table>
<thead>
<tr>
<th>ARM Cortex-M0 processor features</th>
<th>Full product options</th>
<th>“M0_DS” implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verilog core</td>
<td>✓</td>
<td>Flattened and Obfuscated</td>
</tr>
<tr>
<td>AMBA AHB-lite interface</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ARMv6-M instruction set architecture</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>NVIC Interrupt controller</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Interrupt line configurations</td>
<td>1 to 32</td>
<td>16 only</td>
</tr>
<tr>
<td>Debug (SWD, JTAG) option</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Up to 4 breakpoints, 2 watchpoints</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Low power optimisations (ACG)</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Multiple power domain support with WIC</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Fast multiplier (1 cycle) option</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>System timer</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Area (gates)</strong></td>
<td>12k – 25k</td>
<td>16K</td>
</tr>
</tbody>
</table>
Agenda

- Introduction to ARM Ltd Cortex-M0 DesignStart Processor
  - ARM v6-M Programmers Model
  - ARM v6-M Exception Handling
  - ARM v6-M Instruction Set Overview Pipeline
  - Basic System Design
v6-M Data Types

- ARM v6-M is a 32-bit architecture.

- When used in relation to the ARM:
  - **Byte** means 8 bits
  - **Halfword** means 16 bits (two bytes)
  - **Word** means 32 bits (four bytes)
  - **Doubleword** means 64 bits (eight bytes)
The Cortex-M0 is designed to be programmed fully in C
  - No need to write assembly code

- Full Thumb technology, and subset of Thumb2
  - 16-bit and 32-bit instructions

- Set of processor core and memory-mapped registers are provided

- Forwards compatible with other M-profile processors

- In-order execution of instructions

- All instructions are treated as restartable
  - Including LDM/STM
Cortex-M Differences

- Fully programmable in C
- Stack-based exception model
- Only two processor modes
  - Thread Mode for User tasks
  - Handler Mode for OS tasks and exceptions
- Vector table contains addresses
Cortex-M0 Memory Map

Core Memory Mapped Register Space, i.e. NVIC (XN)

External Peripherals (XN)

External Memory

External Peripherals (XN)

Data Memory (code can also be placed here)

Executable region for program code and data (vector table is fixed at address 0x0000 0000)

Device 511MB 0xE010 0000

Private Peripheral Bus 0xE000 0000

External Device 1GB 0xA000 0000

External RAM 1GB 0x6000 0000

Peripheral 500MB 0x4000 0000

SRAM 500MB 0x2000 0000

Code 500MB 0x0000 0000

XN – execute never
Cortex-M0 Register Set

- 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
## Register Usage

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0</td>
<td>Arguments into function</td>
</tr>
<tr>
<td>r1</td>
<td>Result(s) from function</td>
</tr>
<tr>
<td>r2</td>
<td>otherwise corruptible (Additional parameters passed on stack)</td>
</tr>
<tr>
<td>r3</td>
<td>Scratch register (corruptible)</td>
</tr>
<tr>
<td>r4</td>
<td>Stack Pointer</td>
</tr>
<tr>
<td>r5</td>
<td>Link Register</td>
</tr>
<tr>
<td>r6</td>
<td>Program Counter</td>
</tr>
<tr>
<td>r7</td>
<td></td>
</tr>
<tr>
<td>r8</td>
<td></td>
</tr>
<tr>
<td>r9</td>
<td></td>
</tr>
<tr>
<td>r10</td>
<td></td>
</tr>
<tr>
<td>r11</td>
<td></td>
</tr>
<tr>
<td>r12</td>
<td></td>
</tr>
<tr>
<td>r13/sp</td>
<td></td>
</tr>
<tr>
<td>r14/1r</td>
<td></td>
</tr>
<tr>
<td>r15/pc</td>
<td></td>
</tr>
</tbody>
</table>

The compiler has a set of rules known as a Procedure Call Standard that determine how to pass parameters to a function (see [AAPCS](#)).

CPSR flags may be corrupted by function call.

Assembler code which links with compiled code must follow the AAPCS at external interfaces.

The AAPCS is part of the ABI for the ARM Architecture.

- r14 can be used as a temporary once value stacked.

**Register variables**

- Must be preserved

**Additional parameters**

Passed on stack
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

- **IEPSR**
  - Composite register of IPSR and EPSR
Processor Mode Usage

- Processor mode may change when exceptions occur
  - Thread Mode is entered on Reset
  - Handler Mode is entered on all other exceptions
- Both modes have full access to all system resources
  - No concept of privilege. Mechanism exists but has no meaning to M0

ARM Cortex-M0 Processor

![Diagram showing processor modes and exceptions]
Agenda

- Introduction to ARM Ltd
- Cortex-M0 Design
- Start Processor
- ARM v6-M Programmers Model
  - ARM v6-M Exception Handling
- ARM v6-M Instruction Set Overview
- Pipeline
- M0_DS EDK Example System
## Supported Exceptions

- **Seven exception types are supported**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset</td>
<td>Processor reset input is asserted</td>
</tr>
<tr>
<td>HardFault</td>
<td>Any type of fault occurred</td>
</tr>
<tr>
<td></td>
<td>e.g. Bus fault or undefined instruction</td>
</tr>
<tr>
<td>NMI</td>
<td>Non-Maskable Interrupt occurred</td>
</tr>
<tr>
<td>IRQs</td>
<td>IRQ Interrupts occurred</td>
</tr>
<tr>
<td>PendSV</td>
<td>Software generated interrupt</td>
</tr>
<tr>
<td>SVCall</td>
<td>Execution of a SVC instruction</td>
</tr>
<tr>
<td>SysTick</td>
<td>Internal system timer caused interrupt</td>
</tr>
</tbody>
</table>

**Using Handler Mode**

- HardFault: Any type of fault occurred, e.g. Bus fault or undefined instruction
- NMI: Non-Maskable Interrupt occurred
- IRQs: IRQ Interrupts occurred
- PendSV: Software generated interrupt
- SVCall: Execution of a SVC instruction
- SysTick: Internal system timer caused interrupt
Exception Properties

- Each exception has associated properties

**Exception number** – Identification for the exception

**Vector address** – Exception entry point in memory

**Priority level** – Determines the order in which multiple pending exceptions are handled
## Vector Table

- Vector table contains the following information:
  - Handler vector addresses
  - Initial value of the Main Stack Pointer (MSP)

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xBF</td>
<td>Interrupt #31 Handler Vector</td>
</tr>
<tr>
<td>0x40</td>
<td>Interrupt #0 Handler Vector</td>
</tr>
<tr>
<td>0x3C</td>
<td>SysTick Handler Vector</td>
</tr>
<tr>
<td>0x38</td>
<td>PendSV Handler Vector</td>
</tr>
<tr>
<td>0x2C</td>
<td>reserved</td>
</tr>
<tr>
<td>0x2C</td>
<td>SVCall Handler Vector</td>
</tr>
<tr>
<td>0x0C</td>
<td>reserved</td>
</tr>
<tr>
<td>0x08</td>
<td>HardFault Handler Vector</td>
</tr>
<tr>
<td>0x04</td>
<td>NMI Handler Vector</td>
</tr>
<tr>
<td>0x00</td>
<td>Reset Handler Vector</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>Initial value of MSP</td>
</tr>
</tbody>
</table>
## Exception Numbers & Vector Addresses

<table>
<thead>
<tr>
<th>Name</th>
<th>Exception Number</th>
<th>Vector Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupts #0 - #31 (N interrupts)</td>
<td>16 to 16 + N</td>
<td>0x40 - 0xBF</td>
</tr>
<tr>
<td>SysTick (SysTick Extension)</td>
<td>15</td>
<td>0x3C</td>
</tr>
<tr>
<td>PendSV</td>
<td>14</td>
<td>0x38</td>
</tr>
<tr>
<td>SVCurrect (SVCurrect)</td>
<td>11</td>
<td>0x2C</td>
</tr>
<tr>
<td>HardFault</td>
<td>3</td>
<td>0x0C</td>
</tr>
<tr>
<td>Non Maskable Interrupt (NMI)</td>
<td>2</td>
<td>0x08</td>
</tr>
<tr>
<td>Reset</td>
<td>1</td>
<td>0x04</td>
</tr>
</tbody>
</table>

- Exception number is used to calculate the vector address
  - Exception number defines the word offset from 0x0

- Exception numbers which are not shown are reserved
Exception Entry Overview

- The Exception Entry sequence is as follows

  - Perform Stack Push (R0-R3, R12, R14, PC (return address), and xPSR)
  - Select Handler Mode
  - Update registers
    - Set LR to EXC_RETURN (architectural value)
      - Generated automatically when exception accepted
      - Special value that indicates exception return vs. subroutine return
      - Also indicates mode to return to
    - Set IPSR<5:0> to the Exception Number
    - Set EPSR.T bit
    - Set PC to the vector address
Vector Table Usage

- In the case of an exception, the core
  - Reads the vector handler address for the exception from the vector table
  - Branches to the handler

```
+--------------------------------+
| Handler Vector                |
| points to the Handler code   |
+--------------------------------+
|                               |
+--------------------------------+
| Handler Vector                |
+--------------------------------+
| Handler code                  |
+--------------------------------+```
Exception Return

- Exception returns can be accomplished using the following instructions
  - POP which includes loading the PC
  - BX with any register
    - Usually LR is used

- Exception return is performed in the following steps
  - LR value (EXC_RETURN) is loaded into the PC to signal an exception return
    - Since no special instruction for exception return needed, handlers can be written as normal C functions
  - Change mode & stack
    - Dependent on EXC_RETURN
  - Stack Pop of AAPCS registers is performed
    - Includes load of return address into the PC
Reset Behavior

1. A reset exception occurs (Reset input was asserted)
2. Load MSP (Main Stack Pointer) register initial value from address 0x00
3. Load reset handler vector address from address 0x04
4. Reset handler executes in Thread Mode
5. Optional: Reset handler branches to the main program
Interrupt Handling

- One Non-Maskable Interrupt (INTNMI) supported
- 16 prioritizable interrupts supported
  - Interrupts can be masked
  - Implementation option selects number of interrupts supported
- Nested Vectored Interrupt Controller (NVIC) is tightly coupled with processor core
- Interrupt inputs are active HIGH
- Tail-chaining supported
Interrupt Enable Registers

- ISER - Interrupt Set-Enable Register
  - Used to enable interrupts

- ICER - Interrupt Clear-Enable Register
  - Used to disable interrupts
Priority Register Fields

- 8 Priority registers are available

![Diagram showing priority register fields]

- Priority IRQ31
- Priority IRQ30
- Priority IRQ29
- Priority IRQ28
- Priority IRQ27
- Priority IRQ26
- Priority IRQ25
- Priority IRQ24
- Priority IRQ23
- Priority IRQ22
- Priority IRQ21
- Priority IRQ20
- Priority IRQ19
- Priority IRQ18
- Priority IRQ17
- Priority IRQ16
- Priority IRQ15
- Priority IRQ14
- Priority IRQ13
- Priority IRQ12
- Priority IRQ11
- Priority IRQ10
- Priority IRQ9
- Priority IRQ8
- Priority IRQ7
- Priority IRQ6
- Priority IRQ5
- Priority IRQ4
- Priority IRQ3
- Priority IRQ2
- Priority IRQ1
- Priority IRQ0
Use of a Priority Register

- A Interrupt Priority Register (IPR) contains 4 priority fields
  - 4 IRQ priorities per register
  - 2 bits each – 4 priority levels

```
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| Priority 3      | Priority 2     | Priority 1     | Priority 0     |                |                |                |                |                |                |                |                |                |                |                |                |                |                |                |                |                |                |                |                |                |                |                |                |
```
Agenda

Introduction to ARM Ltd
Cortex-M0 DesignStart Processor
ARM v6-M Programmers Model
ARM v6-M Exception Handling
  - ARM v6-M Instruction Set Overview
    Pipeline
M0_DS EDK Example System
ARM and Thumb Performance

Dhrystone 2.1/sec @ 20MHz

Memory width (zero wait state)

<table>
<thead>
<tr>
<th></th>
<th>ARM</th>
<th>Thumb</th>
</tr>
</thead>
<tbody>
<tr>
<td>32-bit</td>
<td>30000</td>
<td>25000</td>
</tr>
<tr>
<td>16-bit</td>
<td>15000</td>
<td>10000</td>
</tr>
<tr>
<td>16-bit with 32-bit stack</td>
<td>15000</td>
<td>10000</td>
</tr>
</tbody>
</table>
The Thumb-2 instruction set

- Variable-length instructions
  - ARM instructions are a fixed length of 32 bits
  - Thumb instructions are a fixed length of 16 bits
  - Thumb-2 instructions can be either 16-bit or 32-bit

- Thumb-2 gives approximately 26% improvement in code density over ARM

- Thumb-2 gives approximately 25% improvement in performance over Thumb
Instruction Set Introduction

- ARMv6-M supports a subset of 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

- Conditional execution is supported
  - Only one conditional instruction

- Optimized for compilation from C
  - Thumb-2 instructions are not designed to be written by hand
  - But easy to learn due to small number of mnemonics
Binary Upwards Compatibility
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

![Branch Diagram]

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

![Branch with Link Diagram]
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 and Rotate Instructions
  - ASR
  - LSL, LSR
  - ROR

- Multiply Instruction
  - MUL (Rd = Rn)

- Sign/Unsign Extend Instructions
  - SXTB, SXTH, UXTB, UXTH

- Miscellaneous Data Processing
  - REV, REV16, REVSH

Examples

- SUBS r0,#1     (r0 ← r0 - 1)
- ORRS r0,r1     (r0 ← r0 | r1)
- MOVs r0,#1     (r0 ← r0 + 1)
- CMP r0,r1      (r0 ← r0 - r1)
- RSBS r0,r1,#0   (r0 ← -r1)
- ADR r0, Start  (r0 ← [Start])

- ASRS r0,r1,#7  (r0 ← r1 >> 7)
- LSLS r0,r1,#3  (r0 ← r1 << 3)
- RORS r0,r1     (r0 ← r0 >> r1)
- MULS r0,r1,r0   (r0 ← r1 * r0)
- UXTB r0,r1     (r0 ← r1[7:0])
- REV r0,r1      Byte Swap
Load and Store Instructions

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

- **Signed Loads**
  - LDRSH
  - LDRSB

- **Load/Stores Multiple**
  - LDM, LDMIA/LDMFD (same as LDM, but with base register update option)
  - STM, STMIA/STMEA (same as STM, but with base register update option)
  - PUSH, POP
    - Uses “Full Descending (FD)” stack, SP always points to last filled data and SP predecrements for each PUSH)

**Examples**

- LDR r0, [r1]  
  (r0 ← [r1])

- STM r0, {r1, r2}  
  (r1 → [r0])  (r2 → [r0+4])

- LDM r0, {r1, r2}  
  (r1 ← [r0])  (r2 ← [r0+4])

- PUSH {r1, r2}  
  (r1 → [SP], r2 → [SP+4])

- POP {r1, r2}  
  (r1 ← [SP], r2 ← [SP-4])
LDM / STM operation

- Syntax:
  \[
  \langle\text{LDM}\mid\text{STM}\rangle\langle\text{cond}\rangle\langle\text{addressing}\_\text{mode}\rangle\text{ Rb!}, \langle\text{register}\text{ list}\rangle
  \]

- 4 addressing modes (NOT ALL SUPPORTED IN v6-M!!):
  - \text{LDMIA / STMIA} increment after
  - \text{LDMIB / STMIB} increment before
  - \text{LDMDA / STMDA} decrement after
  - \text{LDMDB / STMDB} decrement before

\[
\begin{align*}
\text{LDMxx r10, \{r0,r1,r4\}} \\
\text{STMxx r10, \{r0,r1,r4\}}
\end{align*}
\]
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 (except NMI and Hard Fault))
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<cond> 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>
## Instruction Cycle Timing

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

- Zero wait state memory system assumed
Agenda

- Introduction to ARM Ltd
- Cortex-M0 DesignStart Processor
- ARM v6-M Programmers Model
- ARM v6-M Exception Handling
- ARM v6-M Instruction Set Overview

- Pipeline

  M0_DS EDK Example System
Integer Core Pipeline

- 3-stage pipeline core with von Neumann architecture

- Pipeline operates in lockstep all the time

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

Fetch → Decode → Execute
Agenda

Introduction to ARM Ltd
Cortex-M0 DesignStart Processor
ARM v6-M Programmers Model
ARM v6-M Exception Handling
ARM v6-M Instruction Set Overview
Pipeline
- M0_DS EDK Example System
An Example AMBA System

- ARM processor
- External Memory Interface
- on-chip RAM
- DMA Bus Master
- AHB-Lite
- APB Bridge
- UART
- Timer
- Keypad
- PIO

The Architecture for the Digital World®
## Memory Map

<table>
<thead>
<tr>
<th></th>
<th>START ADDRESS</th>
<th>END ADDRESS</th>
<th>SIZE</th>
<th>INTERRUPT CAPABILITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSRAM</td>
<td>0x0000_0000</td>
<td>0x007F_FFFF</td>
<td>8MB</td>
<td>NO</td>
</tr>
<tr>
<td>FLASH</td>
<td>0x0080_0000</td>
<td>0x00FF_FFFF</td>
<td>8MB</td>
<td>NO</td>
</tr>
<tr>
<td>LED</td>
<td>0x5000_0000</td>
<td>0x50FF_FFFF</td>
<td>16MB</td>
<td>NO</td>
</tr>
<tr>
<td>UART</td>
<td>0x5100_0000</td>
<td>0x51FF_FFFF</td>
<td>16MB</td>
<td>YES</td>
</tr>
<tr>
<td>DUMMY1</td>
<td>0x5200_0000</td>
<td>0x52FF_FFFF</td>
<td>16MB</td>
<td>NO</td>
</tr>
<tr>
<td>DUMMY2</td>
<td>0x5300_0000</td>
<td>0x53FF_FFFF</td>
<td>16MB</td>
<td>NO</td>
</tr>
<tr>
<td>GPIO</td>
<td>0x5400_0000</td>
<td>0x54FF_FFFF</td>
<td>16MB</td>
<td>NO</td>
</tr>
<tr>
<td>VGA</td>
<td>0x5500_0000</td>
<td>0x55FF_FFFF</td>
<td>16MB</td>
<td>NO</td>
</tr>
<tr>
<td>KB</td>
<td>0x5600_0000</td>
<td>0x56FF_FFFF</td>
<td>16MB</td>
<td>YES</td>
</tr>
<tr>
<td>TIMER</td>
<td>0x5700_0000</td>
<td>0x57FF_FFFF</td>
<td>16MB</td>
<td>YES</td>
</tr>
<tr>
<td>7SEG</td>
<td>0x5800_0000</td>
<td>0x58FF_FFFF</td>
<td>16MB</td>
<td>NO</td>
</tr>
</tbody>
</table>
Design Flow (Nexsys3)

1. HARDWARE (Verilog or VHDL) → XILINX ISE → .BIT FILE → ADEPT SUITE (FPGA CONFIG)
2. SOFTWARE (C & Assembly) → ARM KEIL → .BIN FILE → ADEPT SUITE (RAM CONFIG)
Outcome
Cortex-M0 Documentation


- Cortex-M0 DesignStart Release Notes (part of M0_DS deliverables)

- “Definitive Guide to the ARM Cortex-M0” – Joseph Yiu
AUP Registration

- To keep up with AUP offerings and ARM developments overall, register to the ARM University Programme online:

  www.arm.com/university

- Tell us about your ARM-based activities and feedback on our various offerings
ARM University Program

Students

Educators

Researchers

The ARM University Program – Preparing Today’s Students for Tomorrow’s Technology

www.arm.com/university


The World's Leading Semiconductor IP Supplier

The Architecture for the Digital World®