Microprocessors & Microcontrollers
: Arm Cortex M0+
(Using RP2040)

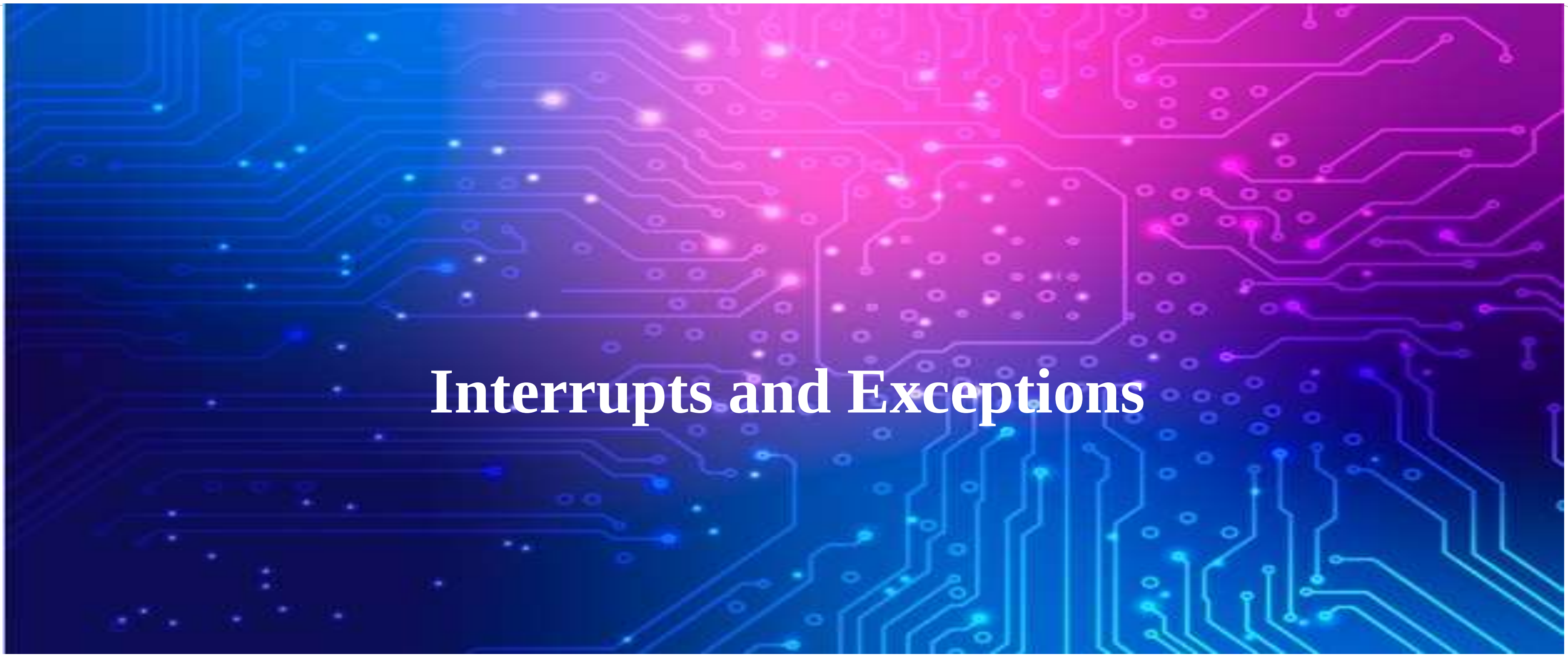
ESM_20

Interrupts, ISR and VTOR

Mouli Sankaran

Focus

- Interrupts
- Interrupt Service Routine (ISR)
- VTOR

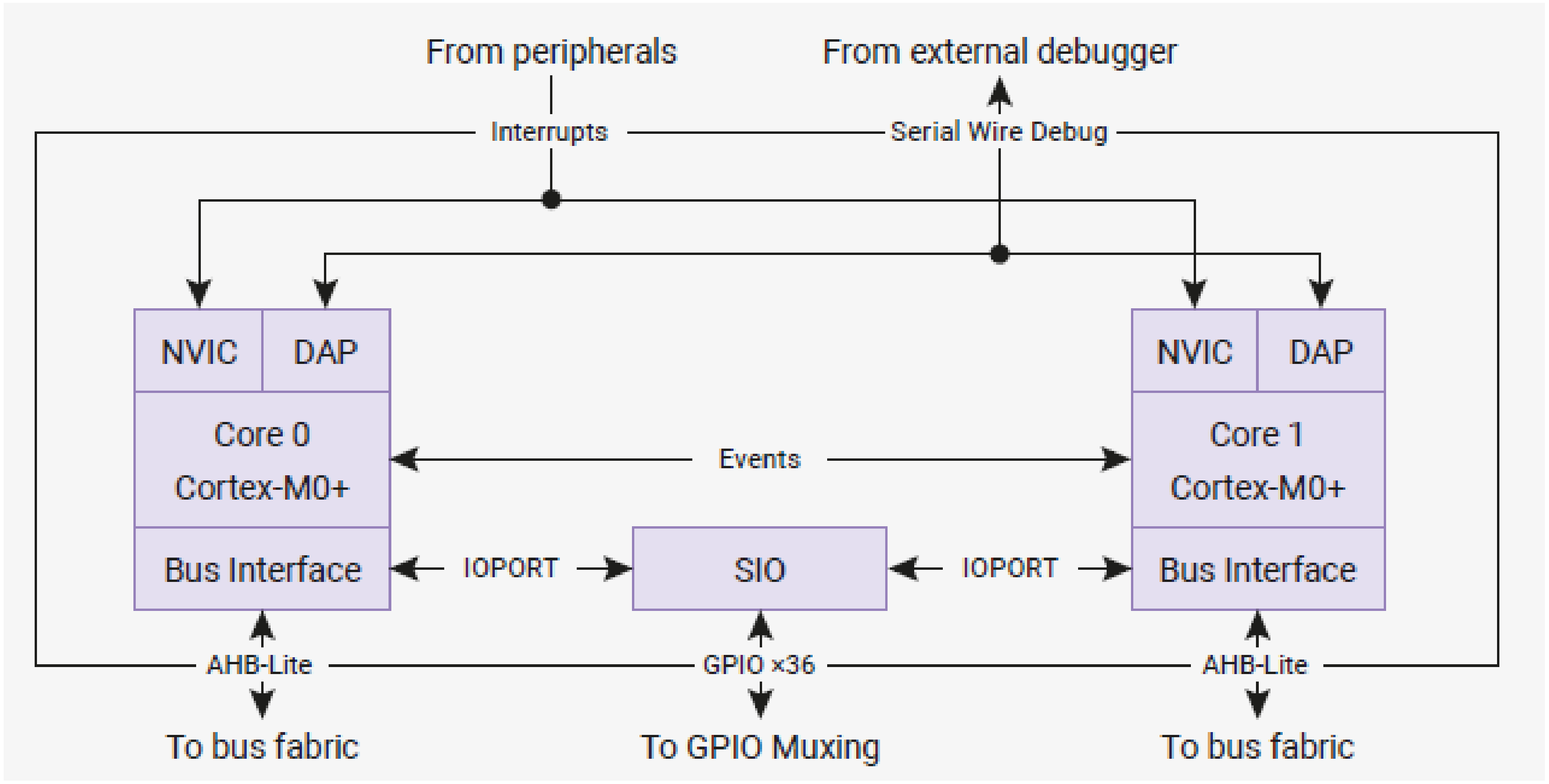


Interrupts and Exceptions

Interrupts and Exceptions

- **Interrupts** and **exceptions** are special kinds of control transfers
- They work somewhat like unprogrammed calls to subroutines
- They **alter** the **normal program flow** to handle **external events** or to **report errors** or **exceptional conditions**
- The **difference** between **interrupts** and **exceptions** is that **Interrupts** are used to handle asynchronous events external to the processor
  - **IRQ**s from Peripherals, **Reset**, etc.
- But **exceptions** handle conditions detected by the processor itself in the course of executing instructions
  - Hard Fault due to Data alignment mismatch or Undefined Instruction, etc.

RP2040: System Overview



NVIC: Nested Vectored Interrupt Controller

DAP: Debug Access Port

NVIC: Cortex-M0+

NVIC: Nested Vectored Interrupt Controller

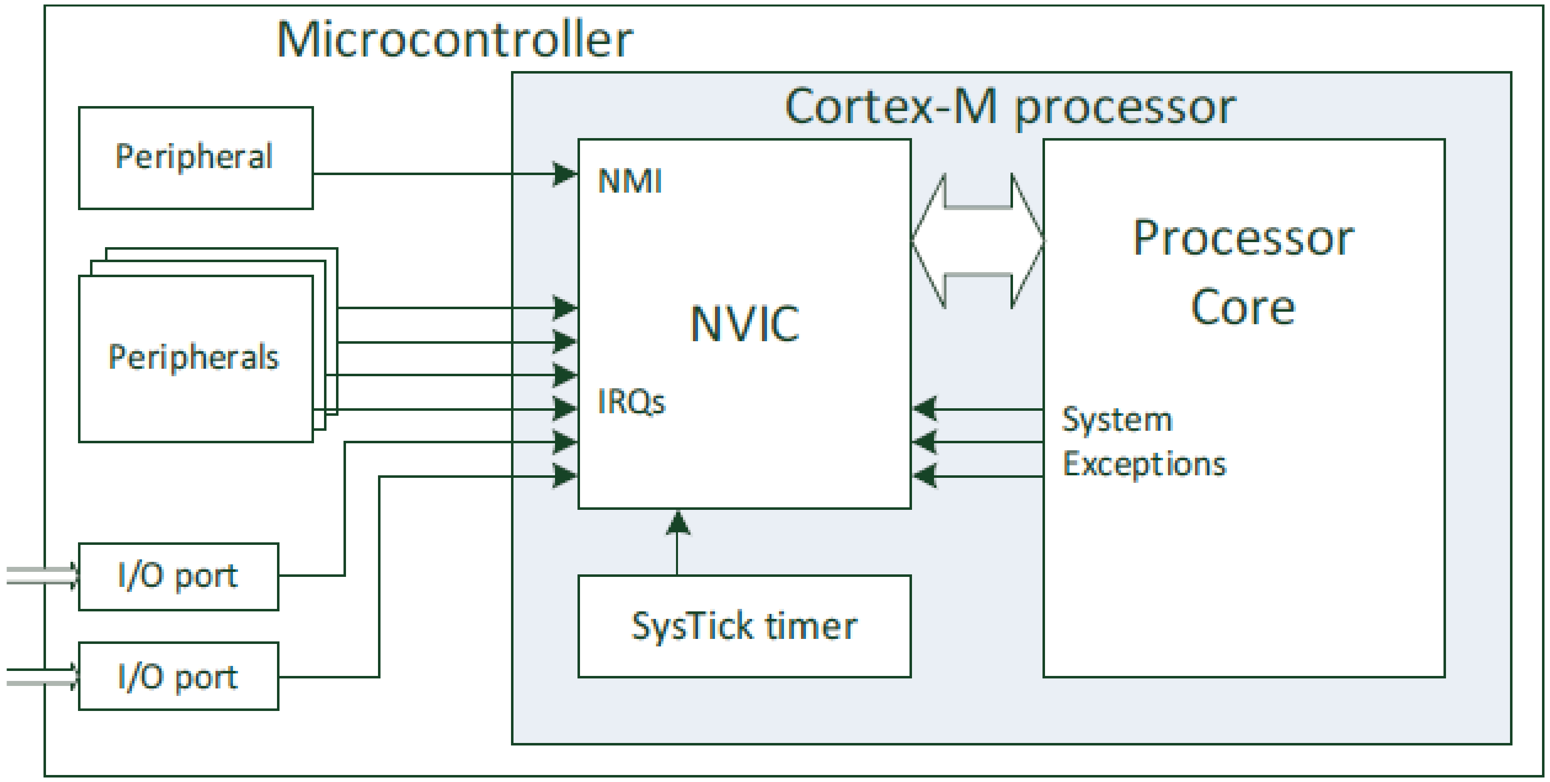


Figure 8.2

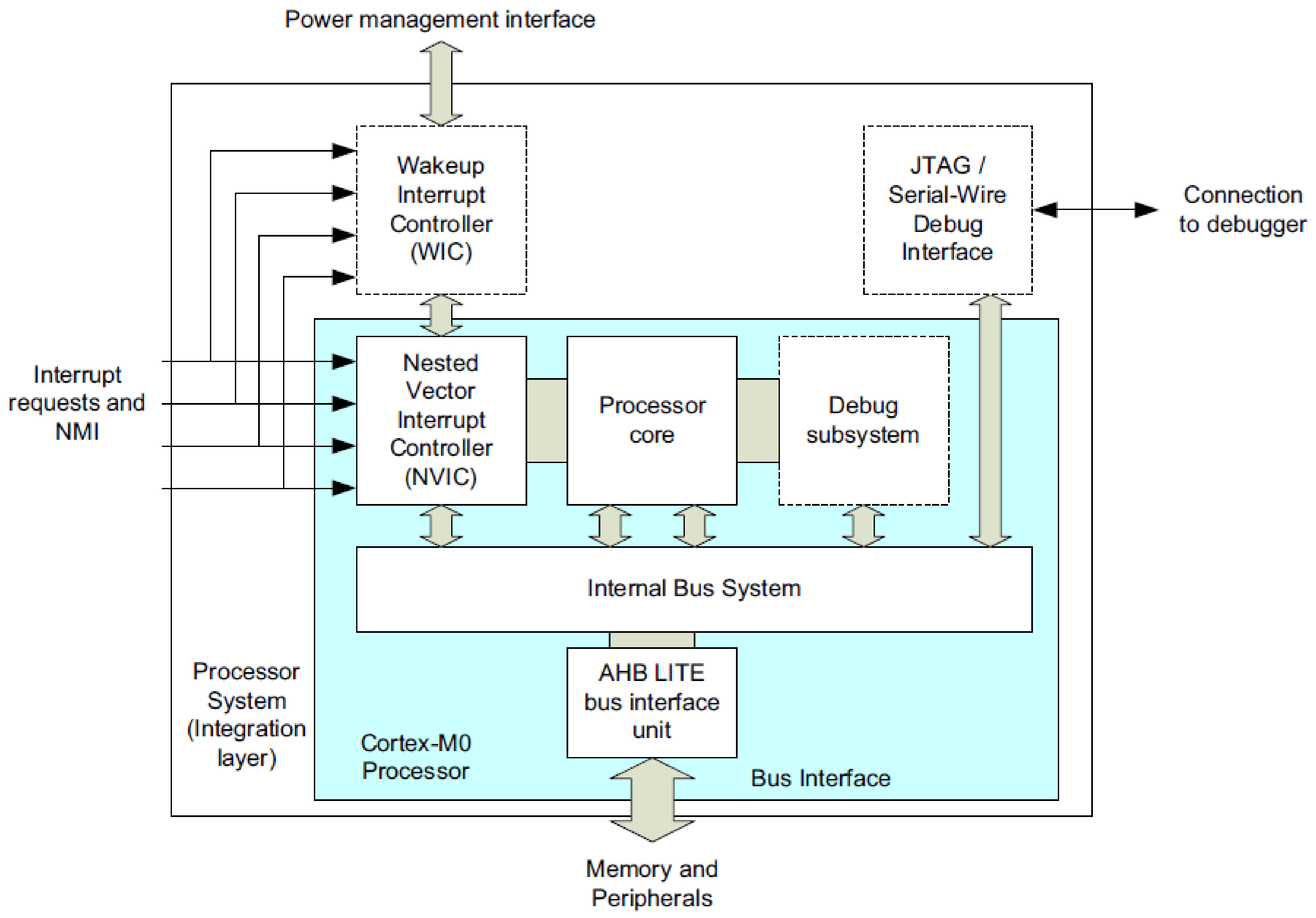
The NVIC in the Cortex®-M0 and Cortex-M0+ processors can deal with up to 32 IRQ inputs, an NMI, and a number of system exceptions.

NVIC in Cortex-M0+

- The Cortex-M processors are designed to provide very quick and deterministic interrupt responses.
- Processor’s execution control part is closely coupled with a built-in Nested Vectored Interrupt Controller (NVIC) to get a faster responses.
- The NVIC provides powerful and yet easy-to-use interrupt management.
- It supports up to 32 interrupt requests from various peripherals (chip design dependent).
- Nested" refers to the fact that interrupts can themselves be interrupted, by higher-priority interrupts.
- "Vectored“ refers to the hardware dispatching each interrupt to a distinct handler routine, specified by the vector table.
- NVIC registers (for enabling and configuring the priorities of interrupts) are only accessible using word transfers.

NVIC and WIC

WIC: Wake up Interrupt Controller

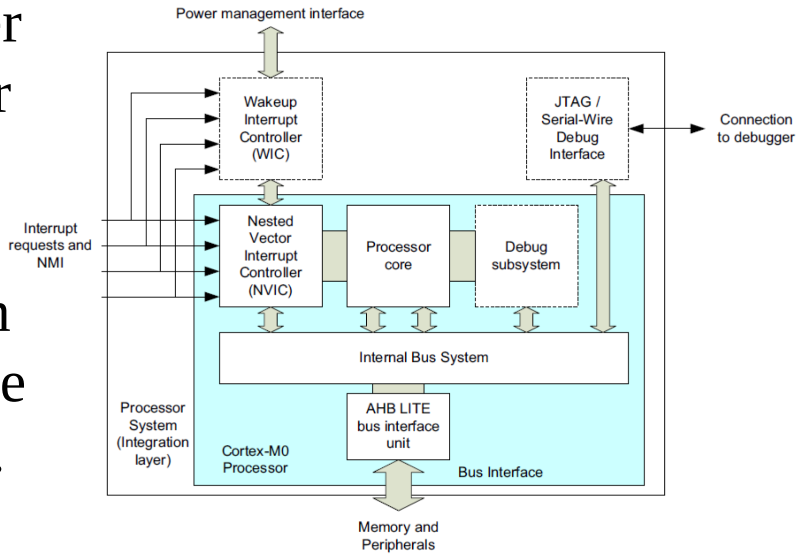


Power management interface
Wakeup Interrupt Controller (WIC)
JTAG / Serial-Wire Debug Interface
Connection to debugger
Interrupt requests and NMI
Nested Vector Interrupt Controller (NVIC)
Processor core
Debug subsystem
Internal Bus System
AHB LITE bus interface unit
Processor System (Integration layer)
Cortex-M0 Processor
Bus Interface
Memory and Peripherals

NVIC and WIC

WIC: Wake up Interrupt Controller

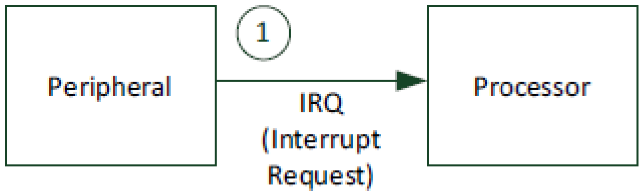
- The WIC is an optional unit. In low-power applications, the microcontroller can enter standby state with most parts of the processor powered down.
- Under this situation, the WIC can perform the function of interrupt masking while the NVIC and the processor core are inactive.

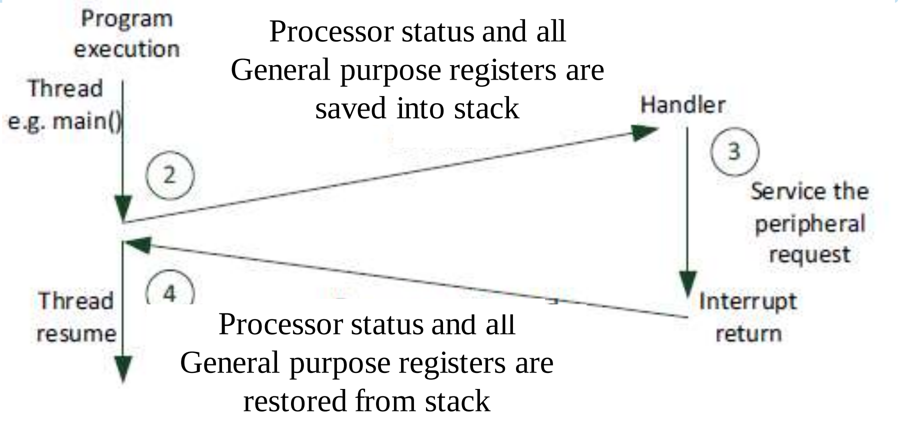


Power management interface
Wakeup Interrupt Controller (WIC)
JTAG / Serial-Wire Debug Interface
Connection to debugger
Interrupt requests and NMI
Nested Vector Interrupt Controller (NVIC)
Processor core
Debug subsystem
Internal Bus System
Processor System (Integration layer)
AHB LITE bus interface unit
Cortex-M0 Processor
Bus Interface
Memory and Peripherals

- RP2040 implementation includes a WIC.
- When an interrupt request is detected, the WIC informs the power management to power up the system so that the NVIC and the processor core can then handle the rest of the interrupt processing.
- This enables the processor and NVIC to be put into a very low-power sleep mode leaving the WIC to identify and prioritize interrupts.
- The processor fully implements the Wait For Interrupt (WFI), Wait For Event (WFE) and the Send Event (SEV) instructions.

Interrupts and Exceptions





1. A peripheral generates an interrupt request (IRQ) to the processor.
2. The NVIC receives and decides based on the configuration whether to interrupt the CPU or not.
3. The CPU detects and accepts the interrupt coming in from the NVIC.
4. CPU completes the execution of the current instruction. If multi-cycle instruction (LDM/STM) the execution is terminated and continued from the beginning after the interrupt is serviced.
5. CPU saves the Status Reg (PSR), all registers along with PC into the stack.
6. CPU locates the starting address of the ISR or handler corresponding to the IRQ using the Interrupt Vector Table and executes it.
7. The processor finishes the handler execution, restores the information previously pushed to the stack, and resumes the interrupted program.

Interrupt Vector Table and VTOR

VTOR: Vector Table Offset Register

Interrupt Vector Table in ROM on Reset

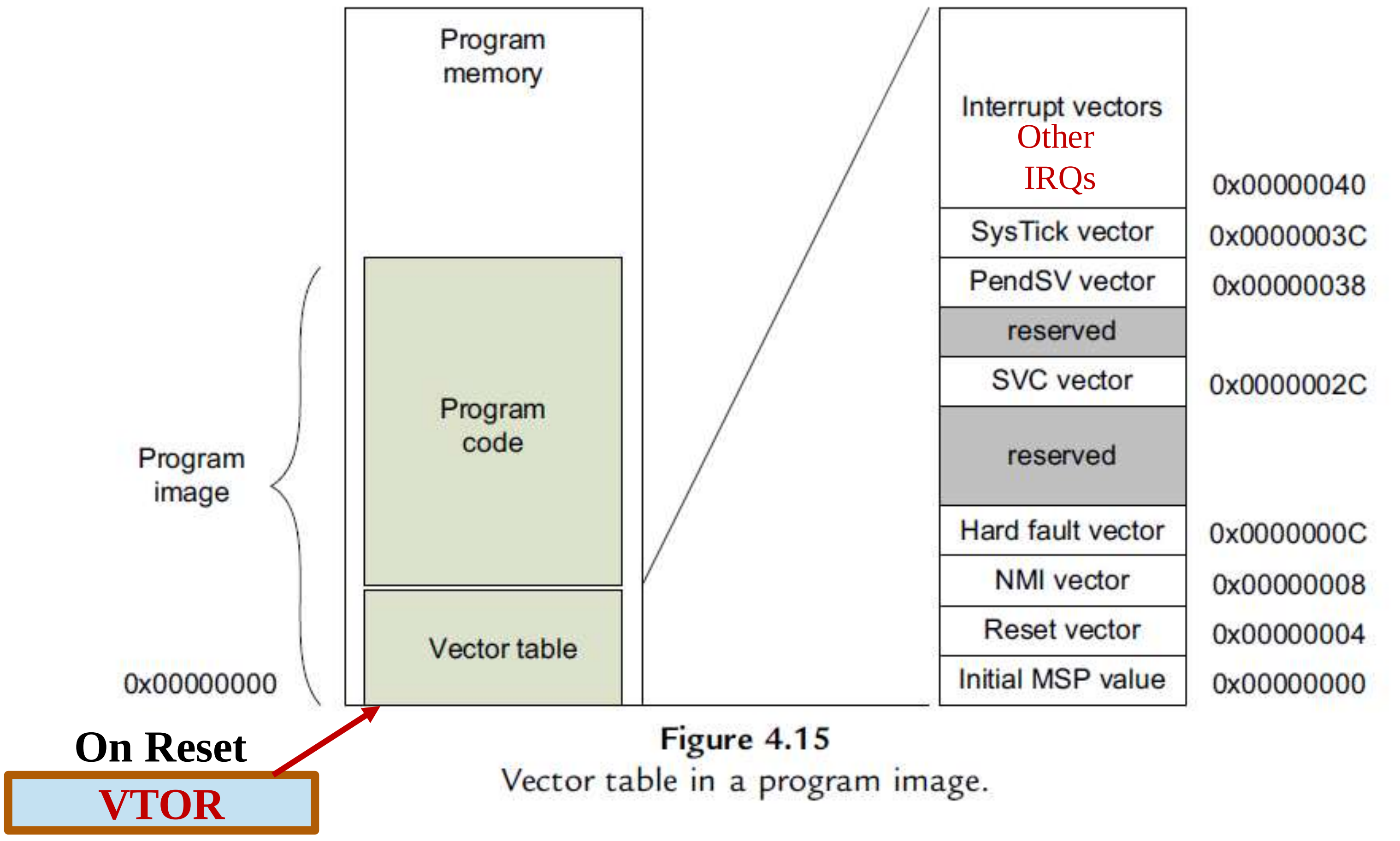


Figure 4.15
Vector table in a program image.

Each entry, points to the ISRs of that particular interrupt Or exception

- The Interrupt vector table is initially located at reset location 0x0 in the ROM, later this will be moved to Flash or SRAM based on the boot ROM code.

Interrupt Vector Table Relocation

Additional Ref1: Page 236/742

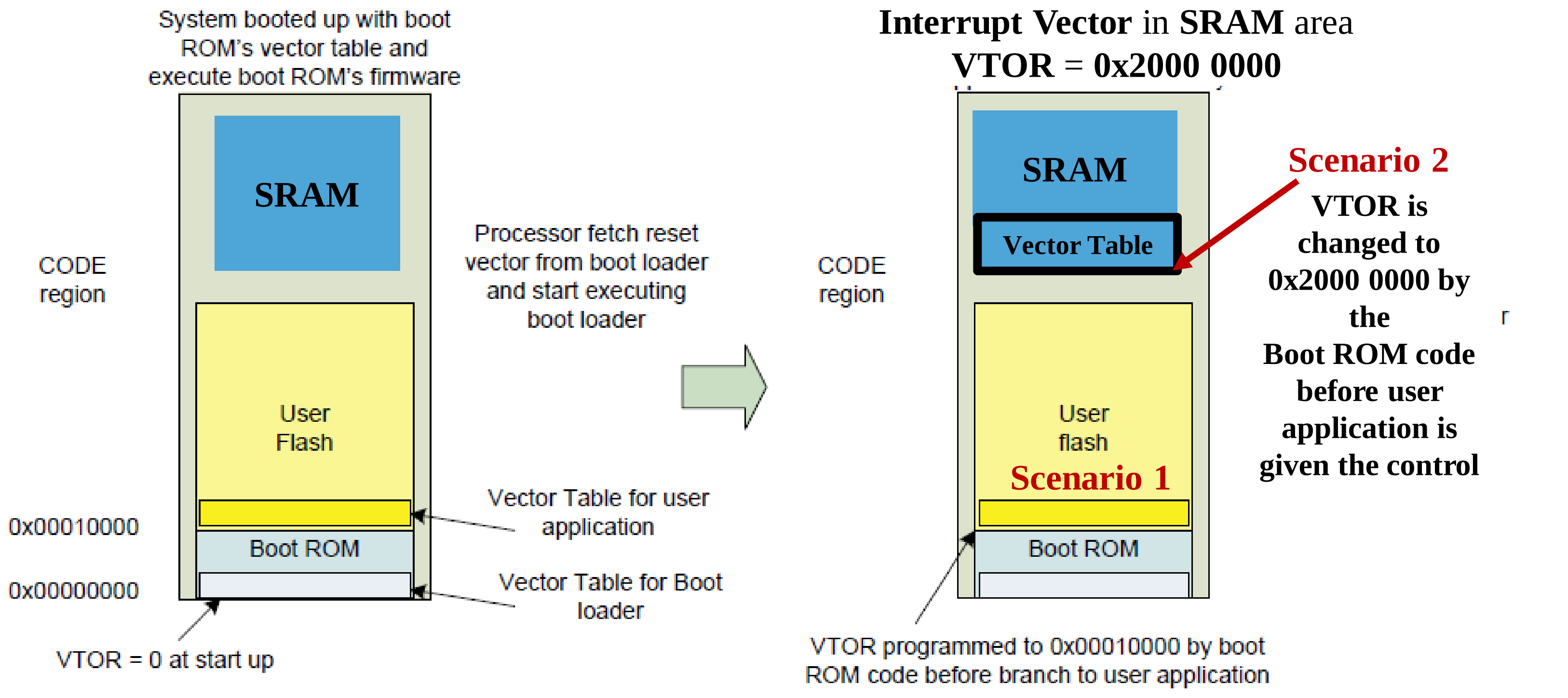

System booted up with boot ROM's vector table and execute boot ROM's firmware
SRAM
CODE region
User Flash
Boot ROM
0x00010000
0x00000000
VTOR = 0 at start up
Processor fetch reset vector from boot loader and start executing boot loader
Vector Table for user application
Vector Table for Boot loader
Interrupt Vector in SRAM area
VTOR = 0x2000 0000
SRAM
Vector Table
Scenario 2
VTOR is changed to 0x2000 0000 by the Boot ROM code before user application is given the control
CODE region
User flash
Scenario 1
Boot ROM
VTOR programmed to 0x00010000 by boot ROM code before branch to user application

Figure 9.3
Use of VTOR by boot loader.

- Initially on reset VTOR is pointing at the ROM address 0x0 where the processor accesses bootloader address in ROM from the table.
- Before the user program is started the VTOR is updated to a new location in SRAM if the interrupt table is to be updated with handlers.

Summary

- Interrupts
- Interrupt Service Routine (ISR)
- VTOR

References - 1

Ref 0

[illegible]

Getting started with Raspberry Pi Pico

C/C++ development with Raspberry Pi Pico and other RP2040-based microcontroller boards

Ref 1

arm Education Media

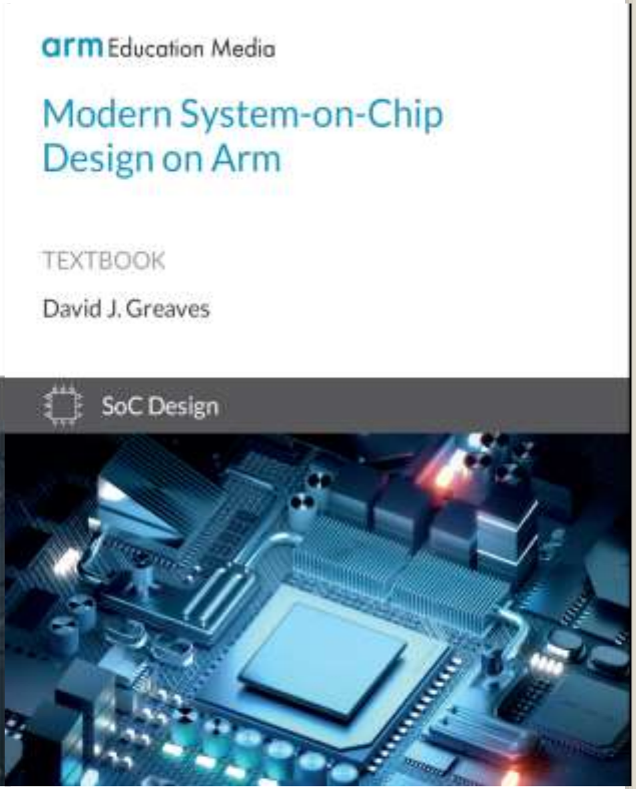
Fundamentals of System-on-Chip Design on Arm Cortex-M Microcontrollers

TEXTBOOK

René Beuchat, Florian Depraz, Andrea Guerrieri, Sahand Kashani

SoC Design

Ref 2



arm Education Media

Modern System-on-Chip Design on Arm

TEXTBOOK

David J. Greaves

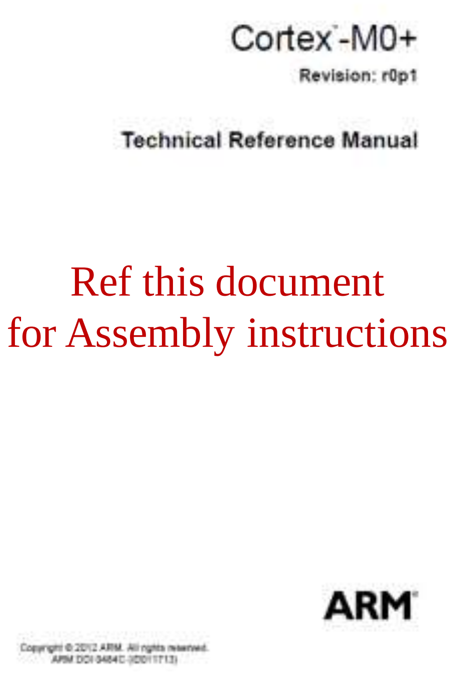
SoC Design

References - 2

Ref 3

Ref 4

Ref 5

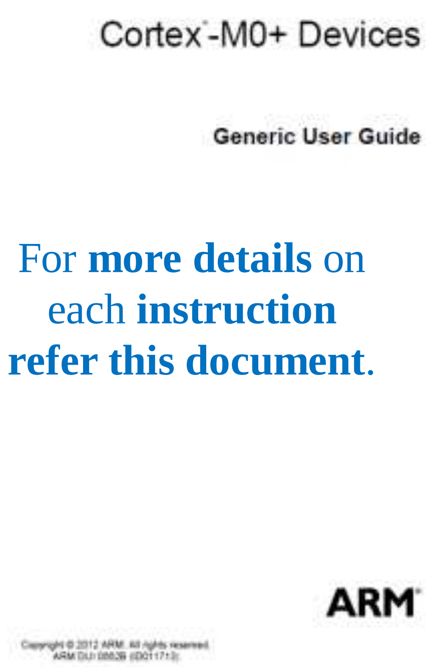

Cortex®-M0+

Revision: r0p1

Technical Reference Manual

ARM®

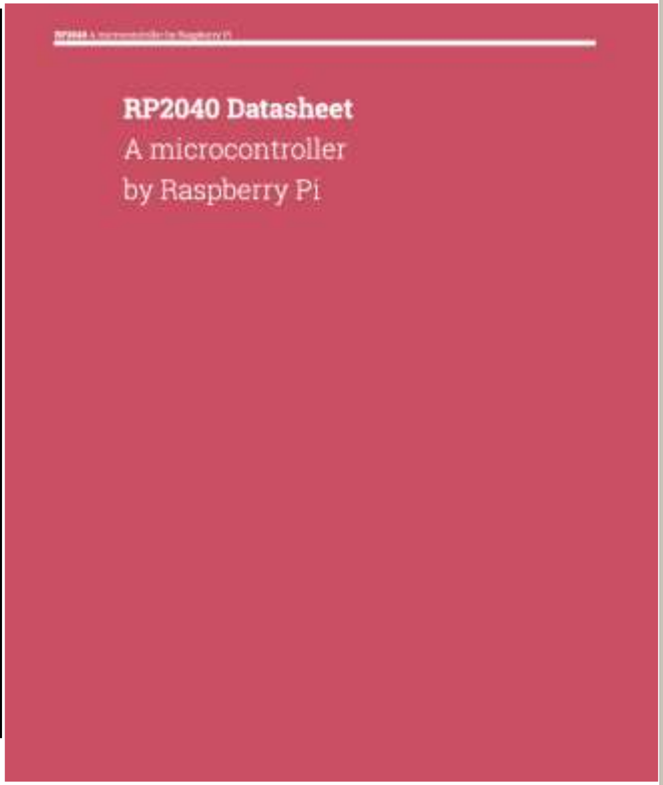
Ref this document
for Assembly instructions


Cortex®-M0+ Devices

Generic User Guide

ARM®

For **more details** on
each **instruction**
refer this document.


RP2040 A microcontroller by Raspberry Pi

RP2040 Datasheet
A microcontroller
by Raspberry Pi

Ref 6

RP2040 A microcontroller by Raspberry Pi

Raspberry Pi Pico C/C++ SDK
Libraries and tools for
C/C++ development on
RP2040 microcontrollers