Class 3. IRQ and DMA

Exception Processing and Interrupt

Basic Concepts of Interrupt Processing

- A variety of unexpected events in a computer system
  - I/O events, error conditions, network events etc
- These events are handled by interrupt processing
  - Speed disparity of various devices in a computer
  - Allow multiple and parallel processing of tasks
- An analogous example: A reading process
  - Phone rings--→ Recognition of an event
  - Answer the phone or not? → Priority
  - Book mark the page → store context
  - Answer the phone → handler
  - Continue the reading process after phone conversation → done

Interrupts

- Motivations
  - Inform a program of some external events timely
    - Polling *vs* Interrupt
  - Implement multi-tasking with priority support

Merriam-Webster:
"to break the uniformity or continuity of"

Polling *vs* Interrupt

Polling:
You pick up the phone every three seconds to check whether you are getting a call.

Interrupt:
Do whatever you should do and pick up the phone when it rings.

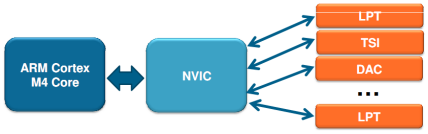
Image from http://vecto.rs

Necessary Procedure of an Interrupt Process

- Interrupt requester ←→ CPU
- Recognition of an interrupt request:
  - Interrupt requester makes an interrupt request
  - CPU recognizes the interrupt request
- Prioritization: Determining whether granting the request or not
  - Requester provides its priority
  - CPU compares it with the priority of its current process
- Context saving to be able to come back after interrupt
  - Program counter marks where interrupt happened
  - Status register and possible other necessary context information

SoC interconnect diagram

- Any module capable of generating an interrupt will depend on NVIC operation
- NMI can be generated from:
  - External pin (must configure the MUX)
  - CoreSight Embedded Trace Buffer (ETB)



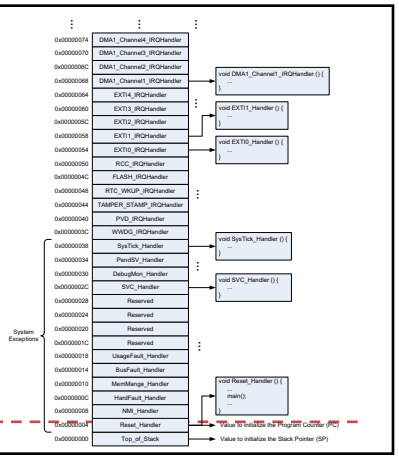
Interrupt Service Routine Vector Table

- Start address for the exception hander for each exception type is fixed and pre-defined
- Processor loads PC with this fixed, pre-defined address
- Exception Vector Table starts at memory address 0
- Program Counter pc = 0x00000004 initially

Address	Priority	Type of priority	Acronym	Description
0x0000_0000	-	-	-	Stack Pointer
0x0000_0004	-3	fixed	Reset	Reset Vector
0x0000_0008	-2	fixed	NMI_Handler	Non maskable interrupt. The RCC Clock Security System (CSS) is linked to the NMI vector.
0x0000_000C	-1	fixed	HardFault_Handler	All class of fault
0x0000_0010	0	settable	MemManage_Handler	Memory management
0x0000_0014	1	settable	BusFault_Handler	Pre-fetch fault, memory access fault
0x0000_0018	2	settable	UsageFault_Handler	Undefined instruction or illegal state
0x0000_001C-0x0000_0028	-	-	-	Reserved
0x0000_002C	3	settable	SVC_Handler	System service call via SWI instruction
0x0000_0030	4	settable	DebugMon_Handler	Debug Monitor
0x0000_0034	-	-	-	Reserved
0x0000_0038	5	settable	PendSV_Handler	Pendable request for system service
0x0000_003C	6	settable	SysTick_Handler	System tick timer
...				

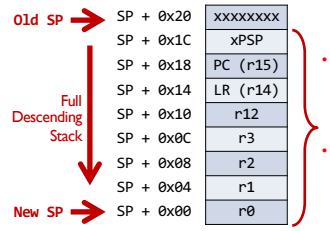
7

ISR Vector Table



8

Stacking & Unstacking

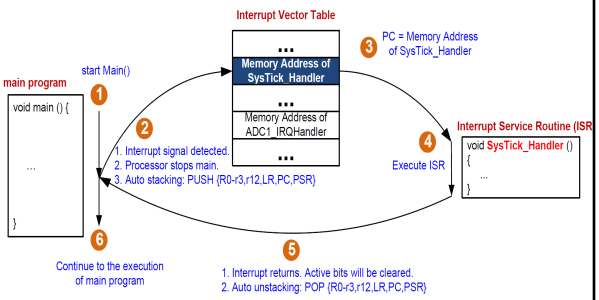


- **Stacking:** The processor automatically pushes these eight registers into the main stack before an interrupt handler starts
- **Unstacking:** The processor automatically pops these eight register out of the main stack when an interrupt hander exits.

- Two stack pointers: Main SP (MSP) and Process SP (PSP)
- Determined by operating mode, and bit 0 of the CONTROL register
  - Handler mode → SP = MSP
  - Thread mode → SP = MSP, if CONTROL[0] = 0 (i.e., privileged thread mode);
    SP = PSP, if CONTROL[0] = 1 (i.e., unprivileged thread mode)

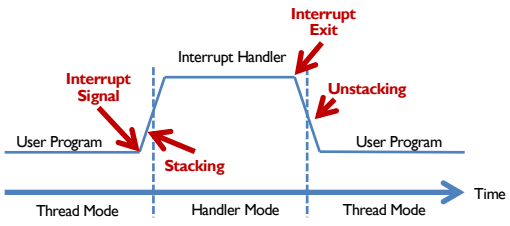
9

Interrupt



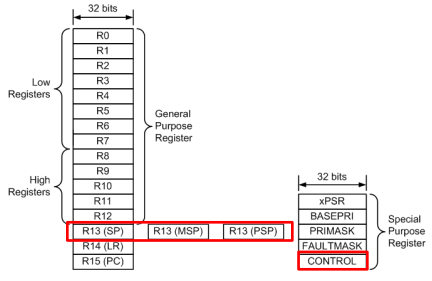
10

Stacking & Unstacking



11

Registers



MSP: Main Stack Pointer
PSP: Process Stack Pointer

12

Processor Mode: Handler Mode *vs* Thread Mode

- Handler mode and Thread mode
  - Handler mode always use MSP (Main Stack Pointer)
  - Thread Mode uses either PSP (Process Stack Pointer) or MSP
    - Control[1] = 0, SP = MSP (default)
    - Control[0] = 1, SP = PSP
- When the processor is reset, the default is the thread mode.
- The processor enters the handler mode when an exception occurs.

13

Sequence of register setups

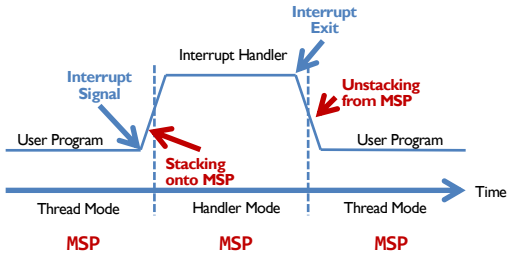
The steps for enabling an interrupt on NVIC:

- 1. Enable the peripheral to be used
- 2. Set the proper bit on the NVICSERx to enable the interrupt on the NVIC
- 3. Clear any pending interrupt by writing to the NVICCPRx to avoid any spurious interrupt
- 4. Configure the interrupt priority by writing to the NVICIPxx
- 5. Write the ISR
- 6. Enable global interrupts

14

Stacking & Unstacking

Control[1] = 0 ⟹ User program uses MSP.

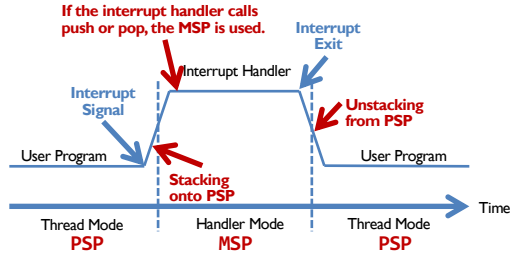


LR = 0xFFFFFFF9

15

Stacking & Unstacking

Control[1] = 1 ⟹ User program uses PSP.

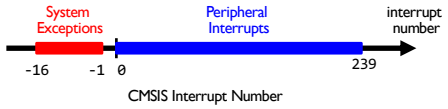


LR = 0xFFFFFFFD

16

Interrupt Number

- Cortex-M supports up to 256 interrupts.



CMSIS Interrupt Number

- First 16 are system exceptions
  - CMSIS defines their interrupt numbers as negative
  - Defined by ARM core
- The rest 240 are peripheral interrupts
  - Peripheral interrupt number starts with 0.
  - Defined by chip manufacturers.

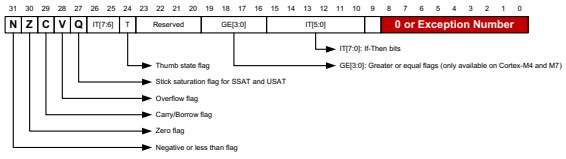
17

Interrupt Number in CMSIS *vs* in PSR

Interrupt number for CMSIS functions

```
NVIC_DisableIRQ (IRQn);            // Disable interrupt
NVIC_EnableIRQ (IRQn);             // Enable interrupt
NVIC_ClearingPending (IRQn);       // clear pending status
NVIC_SetPriority (IRQn, priority); // set priority level
```

Interrupt number in Program Status Register (PSR)



Interrupt Number in PSR = 16 + Interrupt Number for CMSIS

18

CMSIS Interrupt Number

```
/******  Cortex-M4 System Exceptions ********************************************************/
  NonMaskableInt_IRQn       = -14,    /* 2 Cortex-M4 Non Maskable Interrupt                */
  HardFault_IRQn            = -13,    /* 3 Cortex-M4 Hard Fault Interrupt                  */
  MemoryManagement_IRQn     = -12,    /* 4 Cortex-M4 Memory Management Interrupt           */
  BusFault_IRQn             = -11,    /* 5 Cortex-M4 Bus Fault Interrupt                   */
  UsageFault_IRQn           = -10,    /* 6 Cortex-M4 Usage Fault Interrupt                 */
  SVCall_IRQn               = -5,     /* 11 Cortex-M4 SV Call Interrupt                    */
  DebugMonitor_IRQn         = -4,     /* 12 Cortex-M4 Debug Monitor Interrupt              */
  PendSV_IRQn               = -2,     /* 14 Cortex-M4 Pend SV Interrupt                    */
  SysTick_IRQn              = -1,     /* 15 Cortex-M4 System Tick Interrupt                */
/******  Peripheral Interrupt Numbers *******************************************************/
  WWDG_IRQn                 = 0,      /* Window WatchDog Interrupt                         */
  PVD_PVM_IRQn              = 1,      /* PVD/PVM1,2,3,4 through EXTI Line detection Interrupts */
  TAMP_STAMP_IRQn           = 2,      /* Tamper and TimeStamp interrupts through the EXTI line */
  RTC_WKUP_IRQn             = 3,      /* RTC Wakeup interrupt through the EXTI line        */
  FLASH_IRQn                = 4,      /* FLASH global Interrupt                            */
  RCC_IRQn                  = 5,      /* RCC global Interrupt                              */
  EXTI0_IRQn                = 6,      /* EXTI Line0 Interrupt                              */
  ...
```

System Exceptions

Peripheral Interrupts

stm321476xx.h

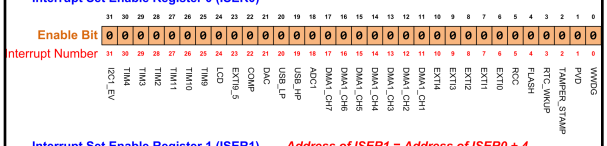
Enable an Interrupt

- Enable a system exception
  - Some are always enabled (cannot be disabled)
  - No centralized registers for enabling/disabling
  - Each are control by its corresponding components, such as SysTick module
- Enable a peripheral interrupt
  - Centralized register arrays for enabling/disabling
  - **ISER** registers for enabling
  - **ICER** registers for disabling

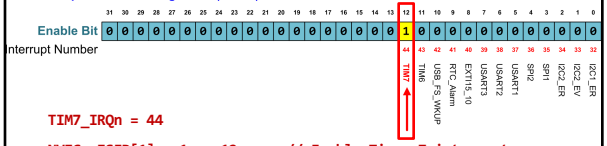
20

Enabling Peripheral Interrupts

Interrupt Set Enable Register 0 (ISER0)



Interrupt Set Enable Register 1 (ISER1) *Address of ISER1 = Address of ISER0 + 4*

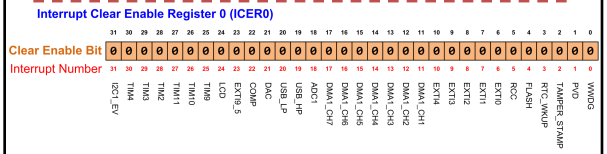



```
TIM7_IRQn = 44
NVIC->ISER[1] = 1 << 12;     // Enable Timer 7 interrupt
```

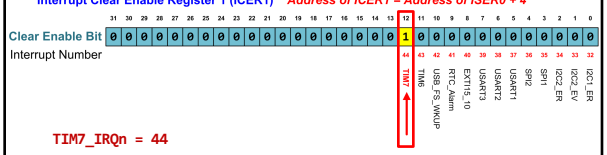
21

Disabling Peripheral Interrupts

Interrupt Clear Enable Register 0 (ICER0)



Interrupt Clear Enable Register 1 (ICER1) *Address of ICER1 = Address of ISER0 + 4*




```
TIM7_IRQn = 44
NVIC->ICER[1] = 1 << 12;     // Diable Timer 7 interrupt
```

22

Disable/Enable Peripheral Interrupts

- For all peripheral interrupts: `IRQn ≥ 0`
- Method 1:
  - `NVIC_EnableIRQ (IRQn);  // Enable interrupt`
  - `NVIC_DisableIRQ (IRQn); // Disable interrupt`
- Method 2:
  - Enable:
    - `NVIC->ISER[ IRQn / 32] = 1 << (IRQn % 32);`
    - Better solution:
    - `NVIC->ISER[ IRQn >> 5] = 1 << (IRQn & 0x1F);`
  - Disable:
    - `NVIC->ICER[ IRQn >> 5] = 1 << (IRQn & 0x1F);`

23

Interrupt Priority

- Inverse Relationship:
  - Lower priority value means higher urgency.
    - Priority of Interrupt A = 5,
    - Priority of Interrupt B = 2,
    - B has a higher priority/urgency than A.
- Fixed priority for Reset, HardFault, and NMI.

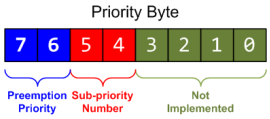
Exception	IRQn	Priority
Reset	N/A	-3 (the highest)
Non-maskable Interrupt (NMI)	-14	-2 (2nd highest)
Hard Fault	-13	-1

- Adjustable for all the other interrupts

24

Interrupt Priority

- Interrupt priority is configured by Interrupt Priority Register (IP)
- Each priority consists of two fields, including preempt priority number and sub-priority number.
  - The preempt priority number defines the priority for preemption.
  - The sub-priority number determines the order when multiple interrupts are pending with the same preempt priority number.

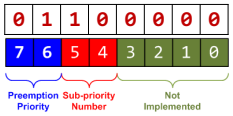


default setting

25

Interrupt Priority Levels

```
NVIC_SetPriority(7, 6);
```




```
IP = 0x60 = 96
```

core_cm4.h or core_cm3.h

```
typedef struct {
  ...
  // Interrupt Priority Register
  volatile uint8_t IP[240];
  ...
} NVIC_Type;
```

It is equivalent to:

```
NVIC->IP[7] = (6 << 4) & 0xff;
```

26

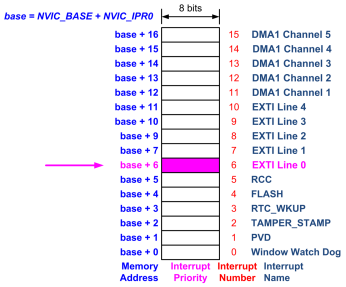
Preemption and Sub-priority Configuration

- `NVIC_SetPriorityGrouping(n)`
  - Perform unlock, and update AIRCR register

n	# of bits in preemption priority	# of bits in sub-priority
0	0	4
1	1	3
2 (default)	2	2
3	3	1
4	4	0

27

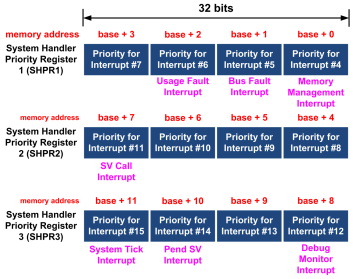
Priority of Peripheral Interrupts




```
// Set the priority for EXTI 0 (Interrupt number 6)
NVIC->IP[6] = 0xF0;
```

28

Priority of System Interrupts




```
// Set the priority of a system interrupt IRQn
SCB->SHP[(IRQn) & 0xF) - 4] = (priority << 4) & 0xFF;
```

29

Masking Priority

- Disable all interrupts with less urgency
  - For critical code, only certain interrupts are allowed.
  - BASEPRI: Disable all interrupts of specific priority level or higher priority level

```
// Disable interrupts with priority value same or higher
__set_BASEPRI( 5 << 4 )

// Critical code
...

// Remove BASEPRI masking
__set_BASEPRI(0);
```

30

Exception-masking registers (PRIMASK, FAULTMASK and BASEPRI)

- PRIMASK: Used to disable all exceptions except for Non-maskable interrupt (NMI) and hard fault.
  - Write 1 to PRIMASK to disable all interrupts except NMI

```
MOV R0, #1
MSR PRIMASK, R0
```

  - Write 0 to PRIMASK to enable all interrupts

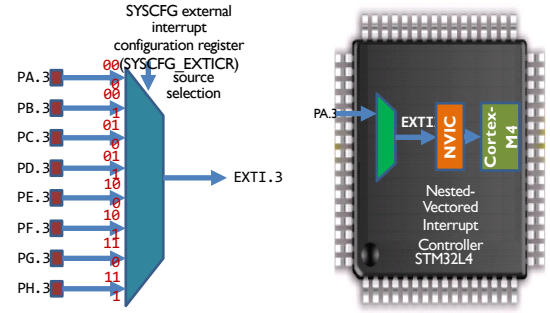
```
MOV R0, #0
MSR PRIMASK, R0
```

- FAULTMASK: Like PRIMASK but change the current priority level to -1, so that even hard fault handler is blocked
- BASEPRI: Disable interrupts only with priority lower than a certain level
  - Example, disable all exceptions with priority level higher than 0x60

```
MOV R0, #0x60
MSR BASEPRI, R0
```

31

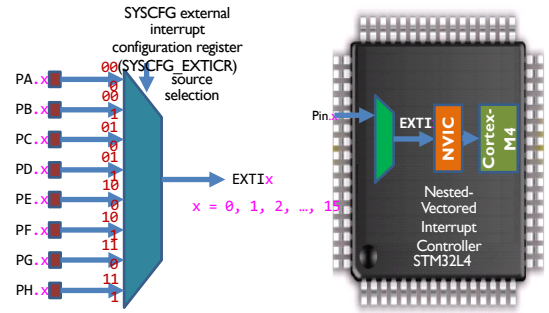
External Interrupt (EXTI) Sources



Select pin x from Port y as EXTIx

32

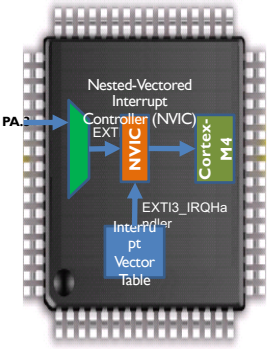
External Interrupt (EXTI) Sources



One mux for each of EXTI0, EXTI1, …, EXTI15

33

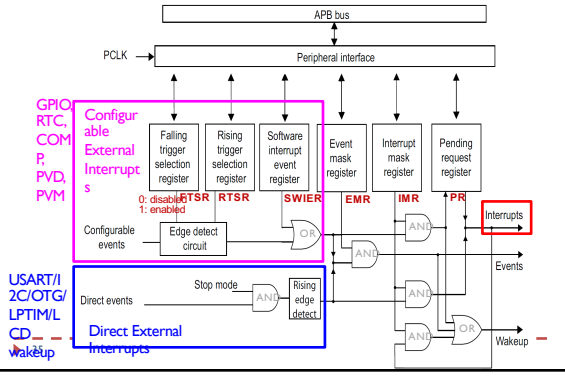
Interrupt Vector Table



Interrupt Vector Table

Interrupt Number (8 bits)	Memory Address of ISR (32 bits)
...	...
EXTI0_IRQn = 6	EXTI0_IRQHandler
EXTI1_IRQn = 7	EXTI1_IRQHandler
EXTI2_IRQn = 8	EXTI2_IRQHandler
EXTI3_IRQn = 9	EXTI3_IRQHandler
EXTI4_IRQn = 10	EXTI4_IRQHandler
...	...
EXTI9_5_IRQn = 23	EXTI9_5_IRQHandler
...	...
EXTI15_10_IRQn = 40	EXTI15_10_IRQHandler

External Interrupt (EXTI) Controller



DMA: Direct Memory Access

High Speed and Direct Data Transfer Between Memory and Peripherals

36

Basic Concepts of DMA

- Limitations of Interrupt Processing
  - CPU involvements
  - Good for discrete events with small amount of data
  - Inefficient for large data transfers
- Needs for High Speed Data Transfer between
  - disk and RAM;
  - NIC and RAM;
  - more
- An analogous example: Program of Study
  - Student asks dean for advising and signature → Request
  - Dean directs student to the advisor → address, tasks, and go
  - Student talks with the advisor → communication/data tranfser
  - Advisor signed the program of study after completing advising and send student back o dean for final signature→ report completion

37

General Procedure of DMA

- DMA Request
  - Request from peripheral through hardware
  - Explicit software initiation
  - Channel to channel linking for continual transfer
- Source/Destination and amount of Data Transfer
  - CPU write registers in DMA controller to define
    - Source address, destination address, and byte count
- Direct and Continuous Data Transfer
  - Data transfer is done directly between memory and peripheral device without CPU involvement
- Report Completion
  - When transfer is done, reporting completion through interrupt

38

Direct Memory Access (DMA)

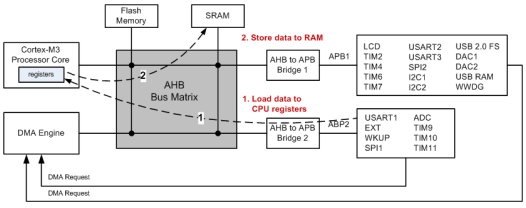
- DMA releases CPU from moving data
  - between peripherals and memory, or
  - between one peripheral and another peripheral.
- DMA uses bus matrix to allow concurrent transfers

Peripherals	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7
ADC1	ADC1						
SPI		SPI1_RX	SPI1_TX	SPI2_RX	SPI2_TX		
USART		USART3_TX	USART3_RX	USART1_TX	USART1_RX	USART2_RX	USART2_TX
I2C				I2C2_TX	I2C2_RX	I2C1_TX	I2C1_RX
TIM2	TIM2_CH3	TIM2_UP			TIM2_CH1		TIM2_CH2 TIM2_CH4
TIM3		TIM3_CH3	TIM3_CH4 TIM3_UP			TIM3_CH1 TIM3_TRIG	
TIM4	TIM4_CH1			TIM4_CH2	TIM4_CH3		TIM4_UP
TIM6 DAC_Ch1		TIM6_UP DAC_Ch1					
TIM7 DAC_CH2			TIM7_UP DAC_CH2				

39

Programmed I/Os

- Processor executes a lot Loads/Stores to move data
- High overhead and slow

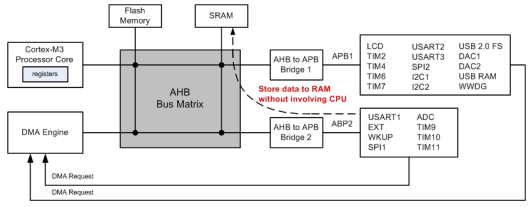


Receiving data from USART serial port without using DMA

40

DMA Sets Core Free

- CPU delegates reads/writes to DMA controller
- Low overhead and fast



Receiving data from USART serial port using DMA

41

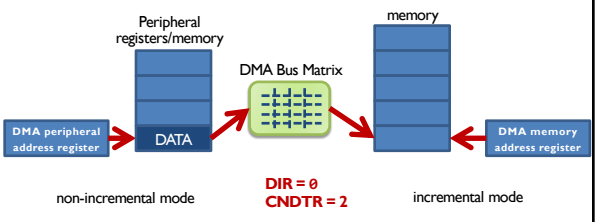
DMA Controller

- Basic Procedures
  - DMA device requests bus
  - CPU grants bus request
  - CPU takes its signals to HiZ
- Key DMA Controller Registers
  - DMA memory address register (CMAR)
  - DMA peripheral address register (CPAR)
  - DMA number of data register (CNDTR)
  - DMA configuration register (CCR)
- DMA are often used together with interrupts

42

DMA Mode: Incremental Mode

DIR: Data transfer direction: 0 = Read from peripheral; 1 = Read from memory

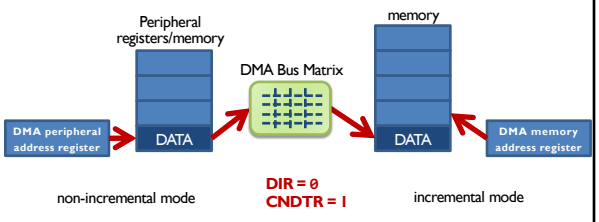


DIR = 0
CNDTR = 2

43

DMA Mode: Incremental Mode

DIR: Data transfer direction: 0 = Read from peripheral; 1 = Read from memory



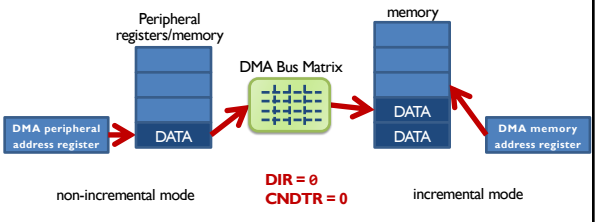
DIR = 0
CNDTR = 1

1st DMA Transfer

44

DMA Mode: Incremental Mode

DIR: Data transfer direction: 0 = Read from peripheral; 1 = Read from memory



DIR = 0
CNDTR = 0

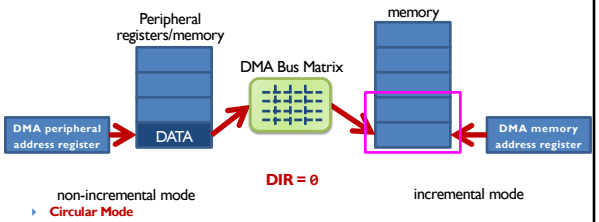
2nd DMA Transfer

45

DMA stops since CNDTR is zero now.

DMA Mode: Circular Mode

DIR: Data transfer direction: 0 = Read from peripheral; 1 = Read from memory



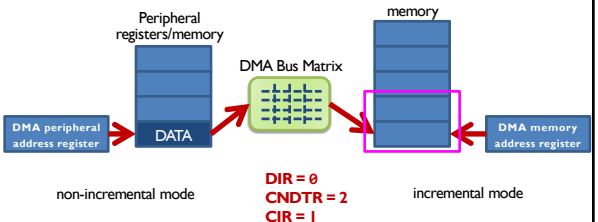
DIR = 0

- **Circular Mode**
  - handle circular buffers and continuous data flows
  - The number of data to be transferred (CNDTR) is automatically reloaded and DMA requests continue to be served

46

DMA Mode: Incremental Mode

DIR: Data transfer direction: 0 = Read from peripheral; 1 = Read from memory

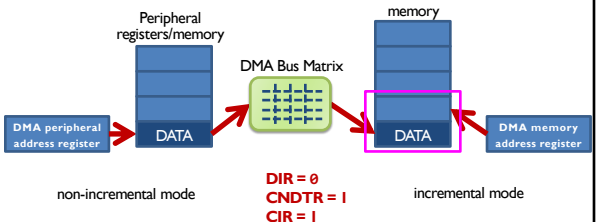


DIR = 0
CNDTR = 2
CIR = 1

47

DMA Mode: Incremental Mode

DIR: Data transfer direction: 0 = Read from peripheral; 1 = Read from memory



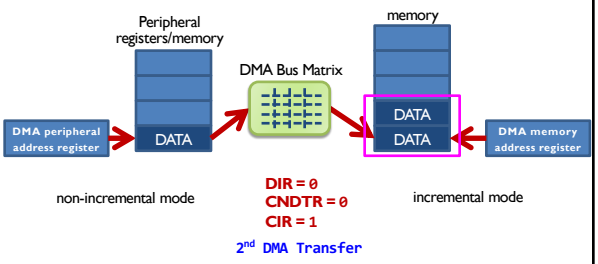
DIR = 0
CNDTR = 1
CIR = 1

1st DMA Transfer

48

DMA Mode: Incremental Mode

DIR: Data transfer direction: 0 = Read from peripheral; 1 = Read from memory



DIR = 0
CNDTR = 0
CIR = 1

2nd DMA Transfer

DMA resets and continues to run!

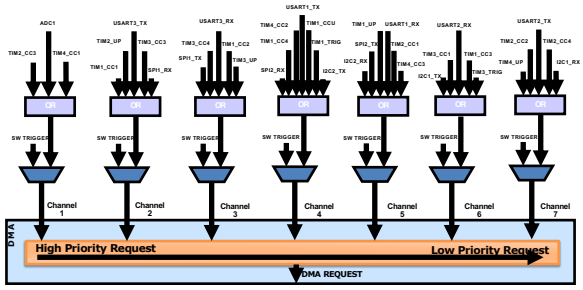
49

DMA Interrupts

- Programmable and Independent source and destination transfer data size: Byte, Halfword or Word
- Three event flags: DMA Half Transfer, DMA Transfer complete and DMA Transfer Error
- Software programmable priorities: Very high, High, Medium or Low

50

DMA Request Mapping



51

From stm

DMA Summary

- Without DMA, CPU has to execute many load and store instructions, leading to slower performance.
- DMA, which makes an automatic data transfer when received a DMA request without involving CPU, accelerates the overall performance.

52