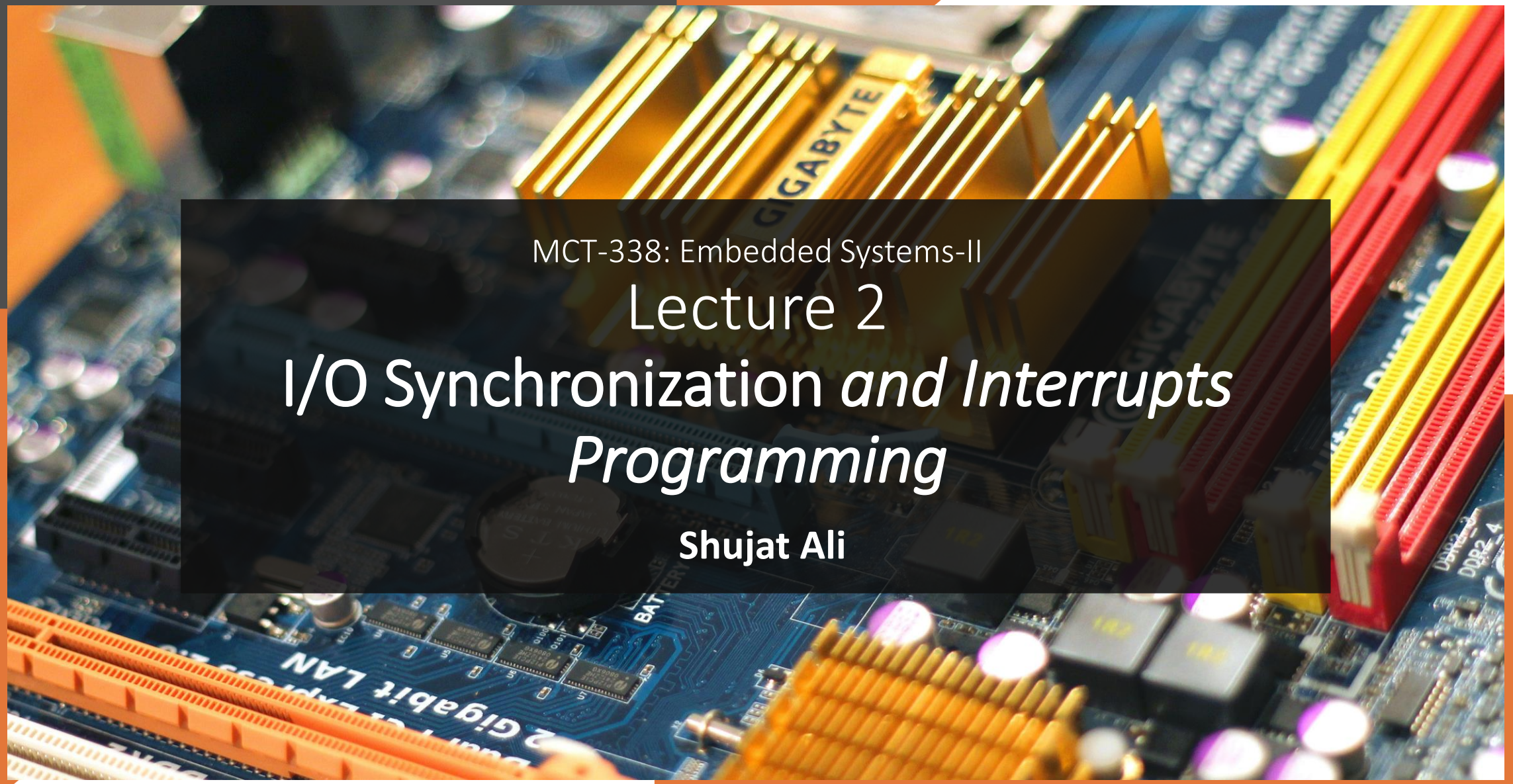
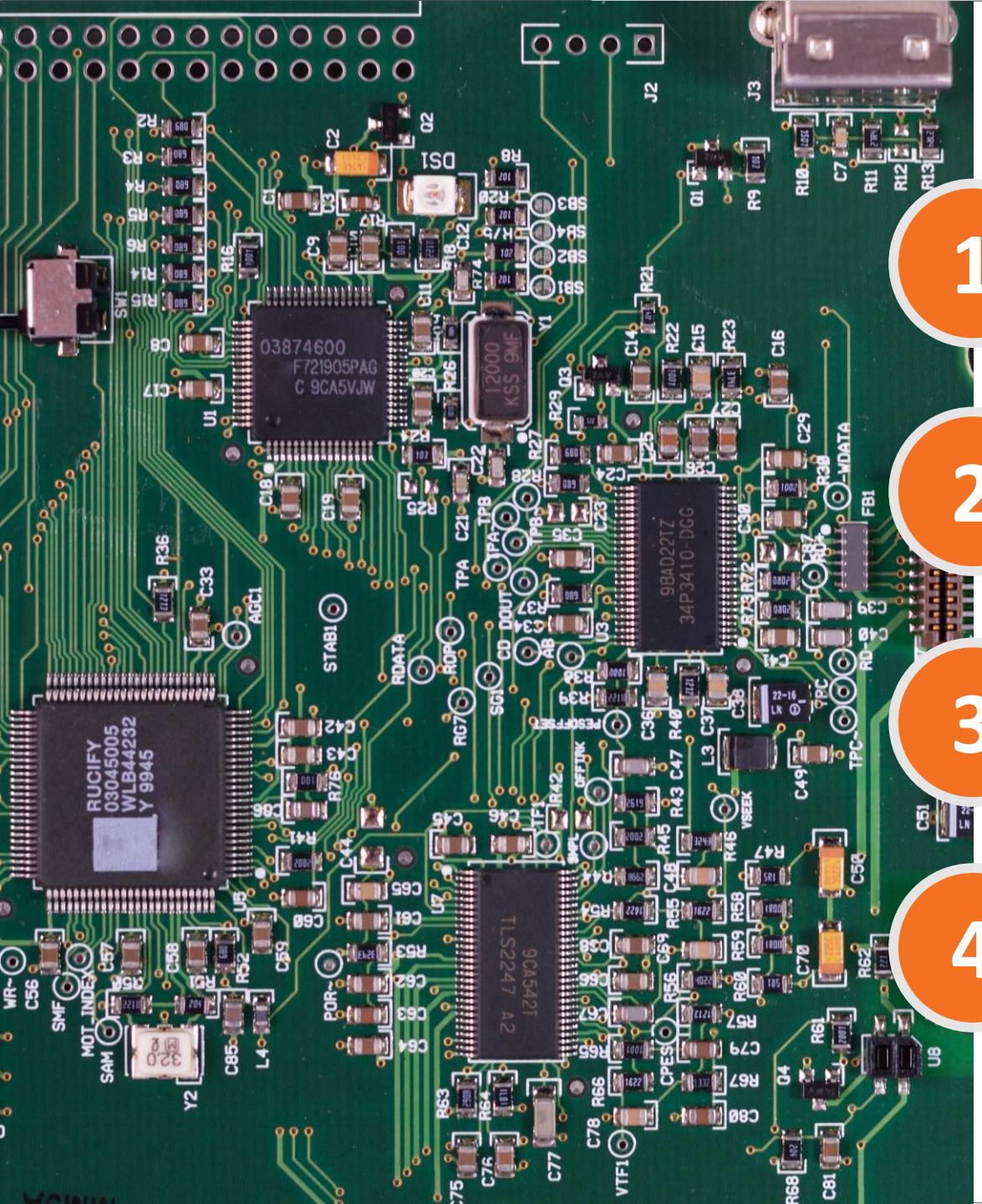


MCT-338: Embedded Systems-II

Lecture 2

I/O Synchronization *and Interrupts Programming*

Shujat Ali



1. INTRODUCTION TO I/O SYNCHRONIZATION
2. METHODS FOR I/O SYNCHRONIZATION
3. TYPES OF EXCEPTIONS OR INTERRUPTS
4. CONFIGURING INTERRUPTS FOR CORTEX-M DEVICES

INTRODUCTION TO I/O SYNCHRONIZATION

Peripheral Devices Operational States and State Transition, Input/Output Device Synchronization

Why is I/O Synchronization is Needed?

INTRODUCTION TO I/O SYNCHRONIZATION

- In an embedded system, speed of I/O devices mismatches that of the microprocessor.
- Therefore, there is an inherent need of synchronization for an efficient embedded system
- In the absence of synchronization,
  - the processor is highly underutilized when the I/O is too slow
  - potential data loss in case of too fast I/O device

Input/Output Devices

INTRODUCTION TO I/O SYNCHRONIZATION

Based on data handling a microcontroller may have following categories of peripherals (or devices).

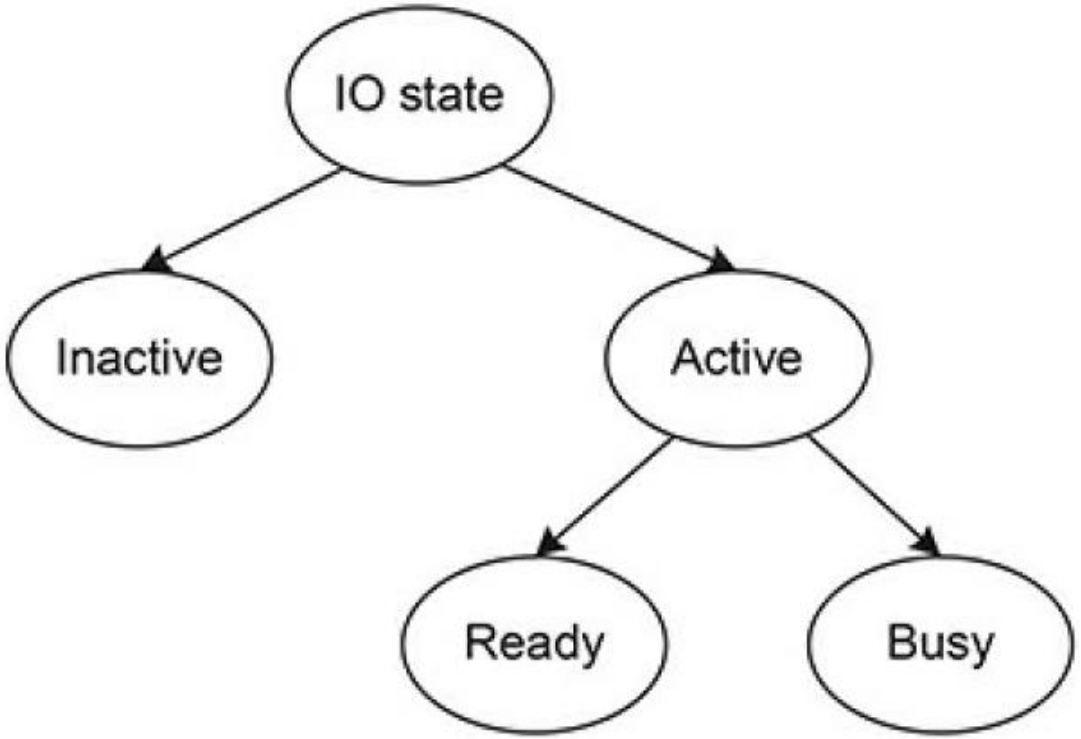
Input Device: A peripheral operates as an input device if it is used for data acquisition/reception, e.g., ADC, CCP of timer, GPIO Pin used as input pin, etc.

Output Device: A peripheral is termed as an output device if it is used for data generation/ transmission, e.g., DAC, PWM, GPIO Pin used as output pin, etc.

Communication interfaces (e.g., UART, USB, Ethernet, etc.) are used for data transmission as well as reception among different devices, therefore these are both input and output devices.

Peripheral Device Operation States

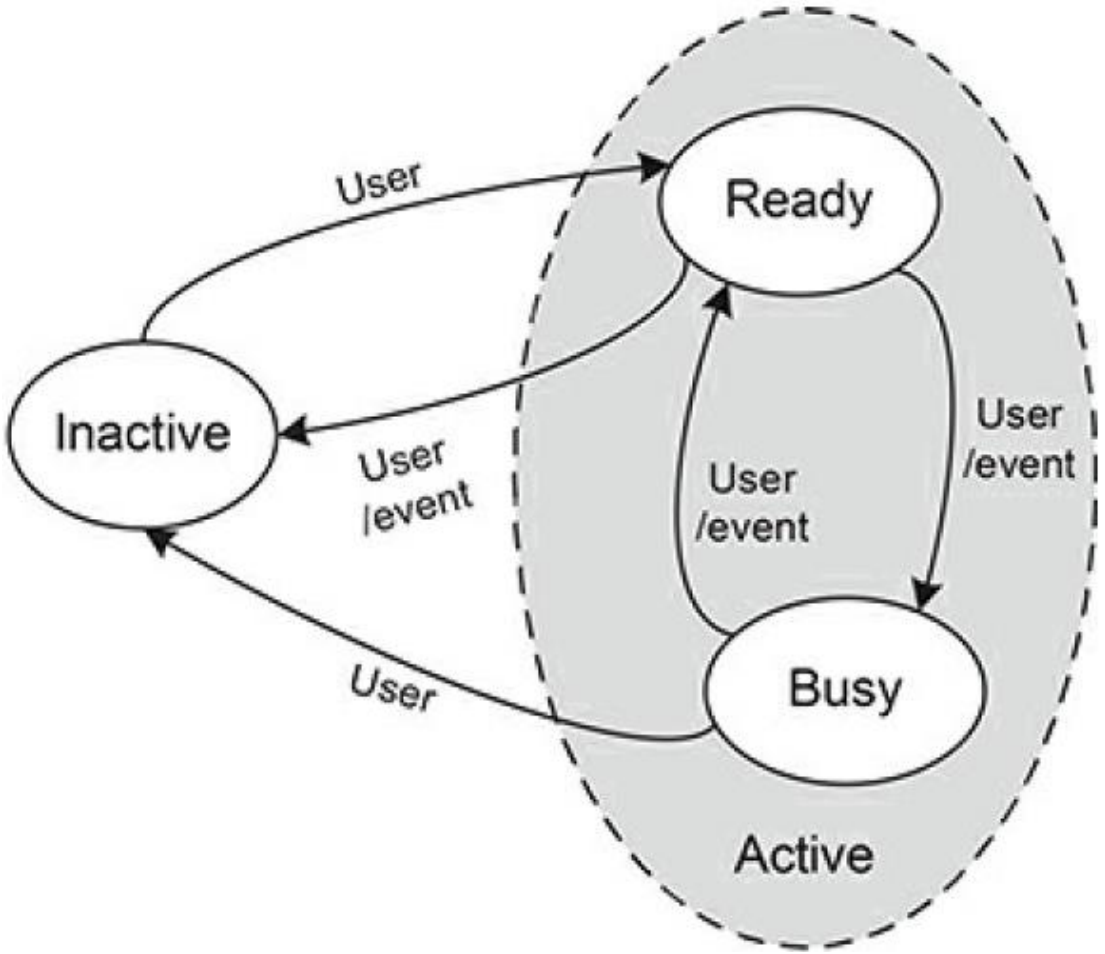
INTRODUCTION TO I/O SYNCHRONIZATION



- From an operational perspective, the hardware device can be in one of the two possible states
  1. ***Inactive state****:* No input-output operation can be performed by the device in an inactive state.
  2. ***Active state****:* When the device is active, it can be either in ***busy state*** or in ***ready state***.

Peripheral Device State Transition

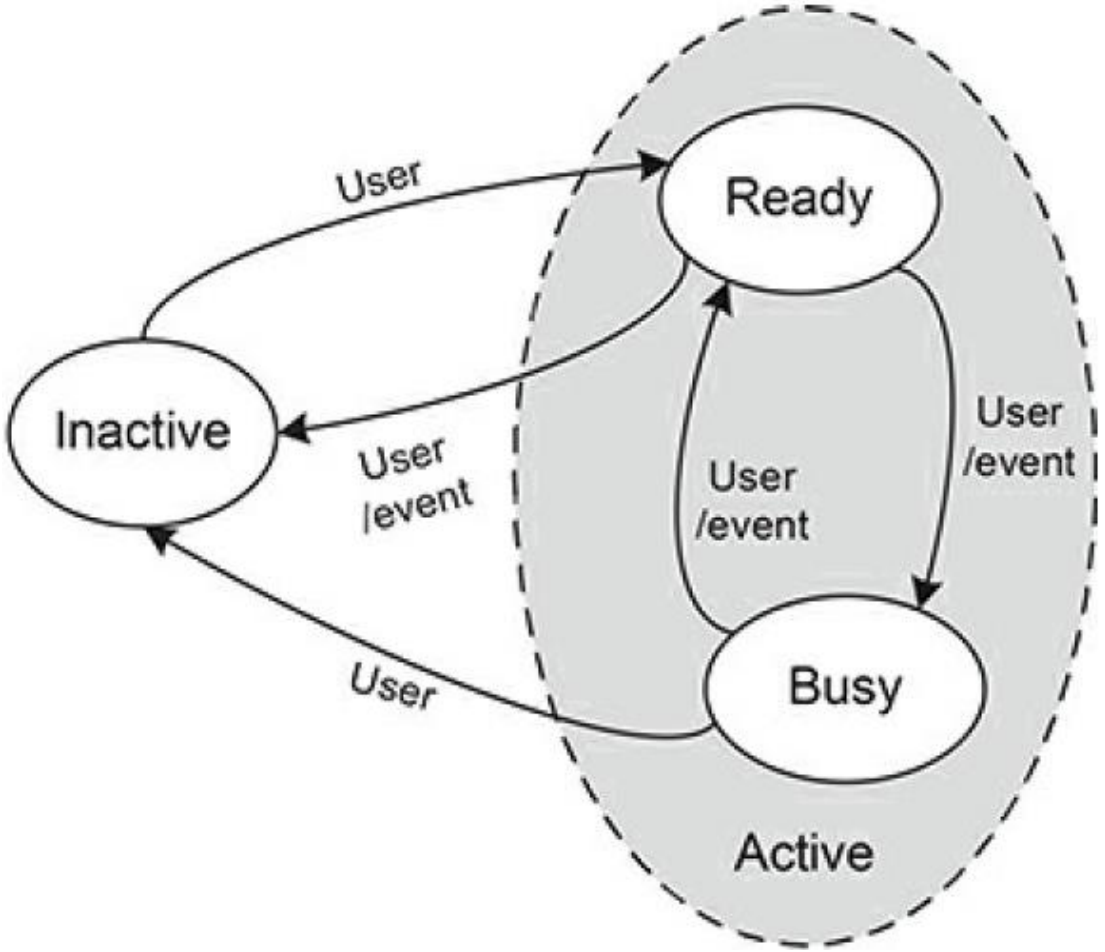
INTRODUCTION TO I/O SYNCHRONIZATION



- A transition from inactive to active state can be generated by the software (i.e., **user application program**)
- A transition from ready to busy state occurs either due to **user task assignment** or due to an **event**.
- The device remains in the busy state when performing the assigned task.
- When the device is finished with the current task a transition from busy to ready state occurs.
- The device remains in ready state, waiting for another task assignment.

Peripheral Device State Transition

INTRODUCTION TO I/O SYNCHRONIZATION



- A transition from ready to inactive state can be triggered either by the **user program** or it can happen due to **expiration of predefined inactivity time interval**.
- A state transition from busy state to inactive state can be triggered by the software (user program), which can lead to data loss.
- The state transitions are equally valid for both input as well as output modes of operation of a peripheral device.

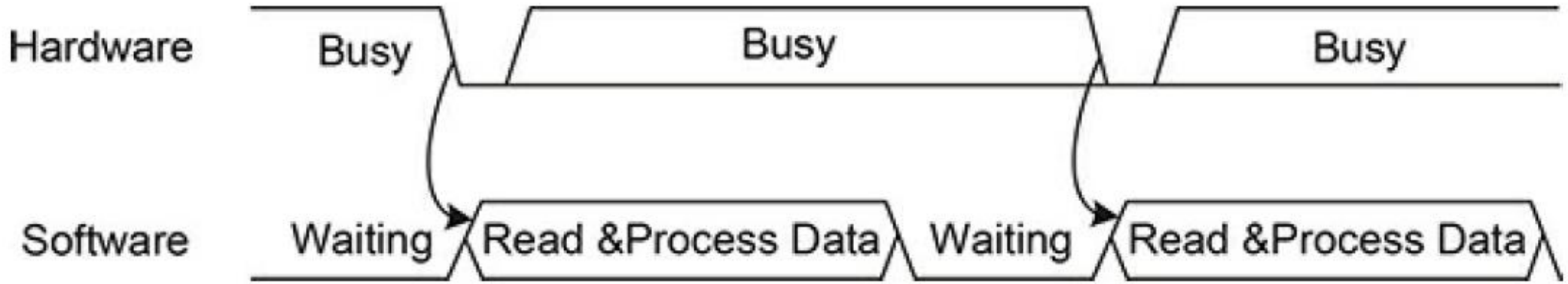
INTRODUCTION TO I/O SYNCHRONIZATION

- I/O synchronization ensures efficient communication between system software and peripheral hardware to prevent delays and data loss.
- If software executes faster than hardware, it must wait, while hardware generating data faster than software can process it risks data overflow.
- Synchronization mechanisms like interrupts and buffering help balance these speeds

Input Device Synchronization

INTRODUCTION TO I/O SYNCHRONIZATION

In case of an **input device**, hardware is the **data producer**, while software is responsible for receiving and processing data and is labeled as **data consumer**.



Once the hardware has produced data, the software reads it and allows the hardware device to start producing new data, while the software is busy in processing that data.

If the time required by the hardware to produce data is higher than the time it takes the software to consume data then software needs to wait for the hardware, i.e., ***time loss***

The arrows from hardware to the software, in timing diagram, mark the synchronizing instances.

Input Device Synchronization

INTRODUCTION TO I/O SYNCHRONIZATION

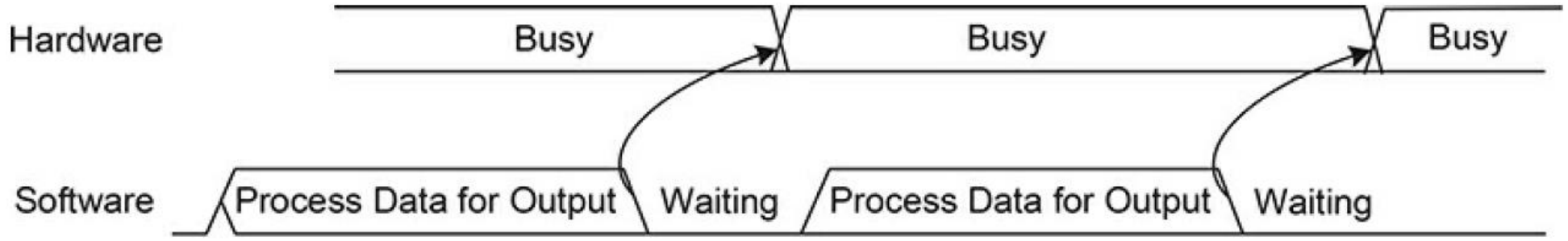
What if hardware produces data faster than the software can consume? There is always a possibility of ***data loss***. Different mechanisms, including buffering, hand shaking, etc. are used to avoid any data loss.

In an ideal scenario, the hardware and software data producing and consuming speeds should match exactly, which is highly unlikely in practical situation and some methodology for their synchronization is required.

Output Device Synchronization

INTRODUCTION TO I/O SYNCHRONIZATION

In case of an output device, hardware is the **data consumer**, while software is responsible for processing and producing data, i.e., **data producer**.



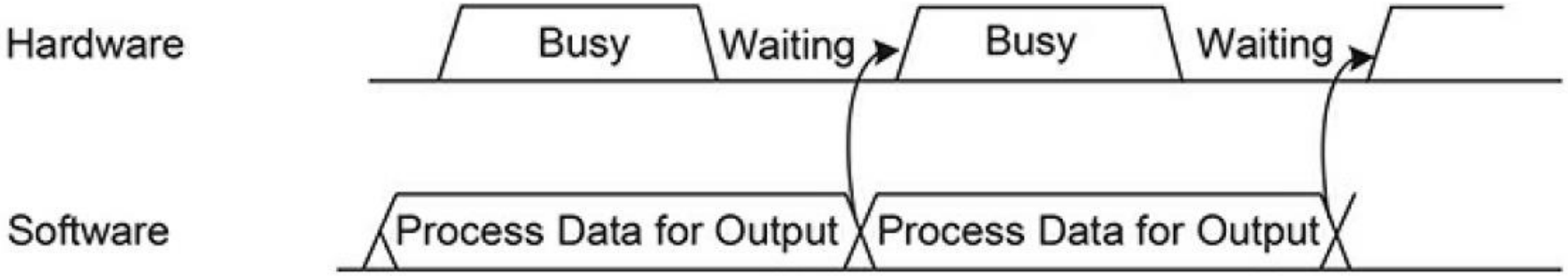
If the time taken by the software to produce data is less than the time required by the hardware to consume that data (i.e., to transmit or generate corresponding signal for that data) then software needs to wait for synchronization.

The synchronizing instances are marked by the arrows from the software to the hardware.

Output Device Synchronization

INTRODUCTION TO I/O SYNCHRONIZATION

In case of an output device, hardware is the **data consumer**, while software is responsible for processing and producing data, i.e., **data producer**.



The case, when hardware is faster and needs to wait for the software. The synchronizing instances are marked by the arrows from the software to the hardware.

From the both cases, we can observe that for output device synchronization, the mismatch in the hardware and software speeds results in **time loss** only and unlike an input device, there is no possibility of **data loss**.

CONFIGURING INTERRUPTS FOR CORTEX-M DEVICES

GPIO Interrupts

Fundamentals

CONFIGURING INTERRUPTS FOR CORTEX-M DEVICES

Configuring different peripheral devices to use interrupts for synchronization is performed during their initialization and is considered as one aspect of device initialization.

To enable interrupts for a peripheral device requires configuration at the following three different levels.

1. ***Processor interrupt configuration*** – provides global enabling and disabling of interrupts
2. ***NVIC configuration*** – allows device level interrupt configuration, priority assignment and pending behavior management
3. ***Device interrupt configuration*** – provides device level interrupt configurations

The NVIC and processor interrupt configurations are quite similar for different peripheral devices. However, the device interrupt configuration varies from device to device and the configuration for one device may not be used for a different peripheral device.

1. Processor Interrupt Configuration

CONFIGURING INTERRUPTS FOR CORTEX-M DEVICES

There are three processor registers that are used for interrupt masking:

1. **Priority Masking Register** (PRIMASK) – enable/disable interrupts and exceptions with programmable priority
2. **Fault Mask Register** (FAULTMASK) – enable/disable interrupts and exceptions except Reset and NMI
3. **Base Priority Masking Register** (BASEPRI) – blocks all the interrupts of either the same or lower priority

The default values of these registers are such that all interrupts are enabled.

However, in case of critical code execution, that should not be interrupted, global interrupt enabling and disabling can be performed.

2. NVIC Configuration

CONFIGURING INTERRUPTS FOR CORTEX-M DEVICES

Following are that are used for NVIC Configurations:

1. **Interrupt Enable Register** (ENx) – enables device interrupts
2. **Interrupt Disable Register** (DISx) – disables devices interrupts
3. **Priority Configuration Register** (PRIy) – assign priority to device interrupts

Depending on the specific peripheral interrupt allocation in the vector table

- x can take values from 0 to 4
- y can take values from 0 to 34

Separate NVIC registers are allocated for enabling and disabling of interrupts (IRQ0-IRQ138 or correspondingly exceptions 16 to 154).

2. NVIC Configuration

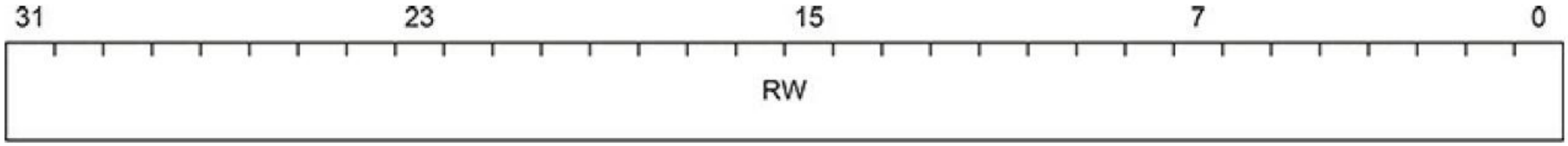
CONFIGURING INTERRUPTS FOR CORTEX-M DEVICES

Register label	Address	Reset value	Brief description
Interrupt enable registers			
EN0	0xE000E100	0x00000000	Used to enable interrupts 0 to 31.
EN1	0xE000E104	0x00000000	Used to enable interrupts 32 to 63.
⋮	⋮	⋮	⋮
EN4	0xE000E110	0x00000000	Used to enable interrupts 128 to 138.
Interrupt disable registers			
DIS0	0xE000E180	0x00000000	Used to disable interrupts 0 to 31.
DIS1	0xE000E184	0x00000000	Used to disable interrupts 32 to 63.
⋮	⋮	⋮	⋮
DIS4	0xE000E190	0x00000000	Used to disable interrupts 128 to 138.
Priority configuration registers			
PRI0	0xE000E400	0x00000000	Priority for interrupts 0 to 3.
PRI1	0xE000E404	0x00000000	Priority for interrupts 4 to 7.
⋮	⋮	⋮	⋮
PRI34	0xE000E488	0x00000000	Priority for interrupts 136 to 138.

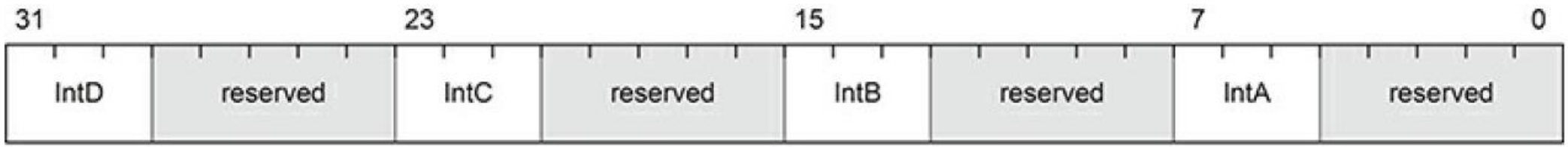
2. NVIC Configuration

CONFIGURING INTERRUPTS FOR CORTEX-M DEVICES

Interrupt Enable/Disable Registers Bit-field



Interrupt Priority Registers Bit-field



3. Device Interrupt Configuration

CONFIGURING INTERRUPTS FOR CORTEX-M DEVICES

Each device interrupt configuration has following two components.

1. **Interrupt Vector Table Configuration and ISR Implementation**
2. **Device Interrupt Register Configurations**

3. Device Interrupt Configuration

CONFIGURING INTERRUPTS FOR CORTEX-M DEVICES

1. Interrupt Vector Table Configuration and ISR Implementation

The interrupt vector table contains the addresses (labels or function names) of interrupt service routines corresponding to each source of interrupt.

To handle an interrupt from an arbitrary peripheral device, for instance say port F, we need to have either all the vector entries included in the table or reserve equivalent memory space before including port F ISR label in the vector table.

This is due to the fact that fixed exception/IRQ numbers are assigned to the sources of interrupts and correspondingly the relative addresses (with respect to the base address of the vector table) of ISRs are also fixed.

As a result, an interrupt vector table entry can not be skipped even if it is not required.

For Cortex-M, an ISR is implemented similar to an ordinary C function call.

3. Device Interrupt Configuration

CONFIGURING INTERRUPTS FOR CORTEX-M DEVICES

2. Device Interrupt Register Configurations for GPIO

Each device can have multiple possible sources of interrupts that can be enabled/disabled (unmasked/masked) individually using device interrupt related registers.

The device level configuration of GPIO as a source of external interrupts is discussed here, specifically, the configuration of port F pin 4 as source of external interrupt.

When a GPIO port pin is configured as an external interrupt, then interrupt triggering can be performed by one of the following two possible activations.

- Level triggered
- Edge triggered

3. Device Interrupt Configuration

CONFIGURING INTERRUPTS FOR CORTEX-M DEVICES

A **level-triggered interrupt** remains active until the peripheral deactivates it by de-asserting the interrupt signal.

In case of level triggered interrupt, the processor executes the ISR and on the exit if the interrupt condition is still valid, the processor immediately enters the ISR again, provided no higher priority interrupt occurs in the meantime.

In case of an **edge triggered interrupt**, the interrupt signal is generated by the rising or falling edge (or both edges) on the corresponding GPIO port pin.

There are seven 32-bit registers associated with GPIO port for interrupt configuration. Only least significant 8-bits are used by all these registers to configure each GPIO port pin individually.

3. Device Interrupt Configuration

CONFIGURING INTERRUPTS FOR CORTEX-M DEVICES

Interrupt Sense (IS) register is used to configure interrupt triggering as edge or level. When a bit in this register is cleared to 0 the corresponding GPIO port pin is configured as edge triggered interrupt.

Interrupt Event (IEV) register configures the type of the interrupt event. Setting to 1 configures the port pin for rising edge interrupt (when edge type interrupt is configured in IS register) or a logic level high triggers the interrupt when IS register is configured for level sensitive interrupt. Clearing it to 0 configures either the falling edge or logic low as the interrupt condition depending on the IS register.

Interrupt Both Edges (IBE), when set to 1, configures both rising as well as falling edges as the source of interrupt. For this to be valid, IS register should be configured for edge type interrupt. Configuring IBE register with a value of 1 overrides the IEV register configuration.

3. Device Interrupt Configuration

CONFIGURING INTERRUPTS FOR CORTEX-M DEVICES

Interrupt Mask (IM) register is used to enable or disable individual port pin interrupts. Writing 1 to this register enables the corresponding port pin interrupt.

Raw Interrupt Status (RIS) and **Masked Interrupt Status** (MIS) are the two registers, which indicate the occurrence of an interrupt condition. The RIS register indicates that a GPIO pin satisfies the requirements for an interrupt, but whether the interrupt is sent to NVIC or not depends on the IM register. On the other hand, the MIS register only indicates those GPIO pin interrupts that are enabled and sent to NVIC.

Interrupt Clear (ICR) register is used to clear an interrupt flag in the status registers. To clear an interrupt status flag in RIS as well as MIS registers, it is required to write '1' to the corresponding bit in the ICR register. If an interrupt is not cleared using ICR before exiting the corresponding interrupt service routine, then the ISR is immediately entered again, provided no other interrupt of higher priority has occurred in the meantime.

Thank You!

Any Questions?