Timers and Interrupts for Embedded System

Hemant Ghayvat

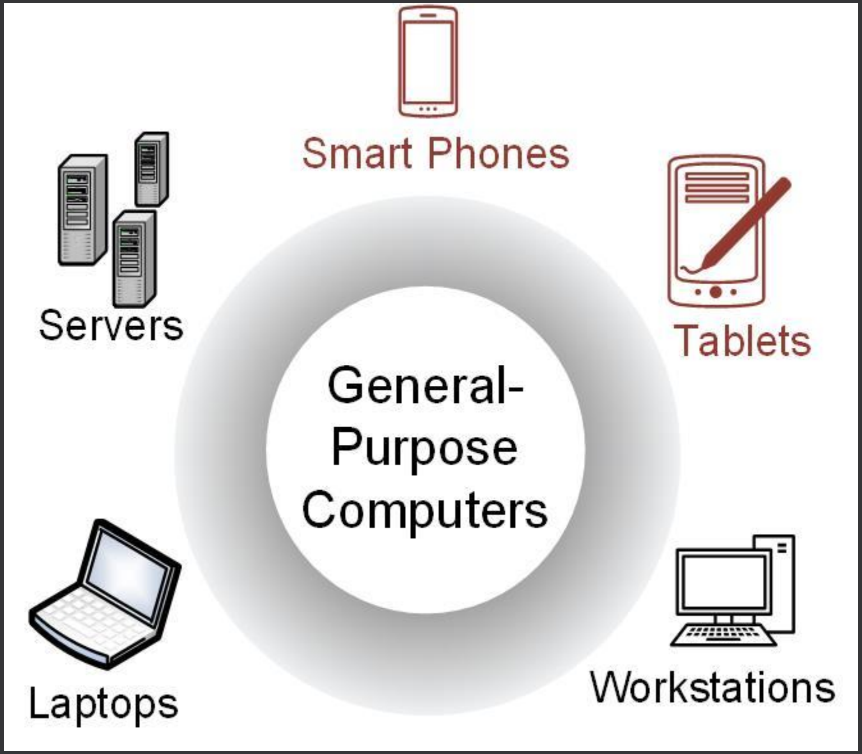
Associate Professor (Docent)

Department of Computer Science and Media Technology
hemant.ghayvat@lnu.se

-ES (from sensor to actuator)

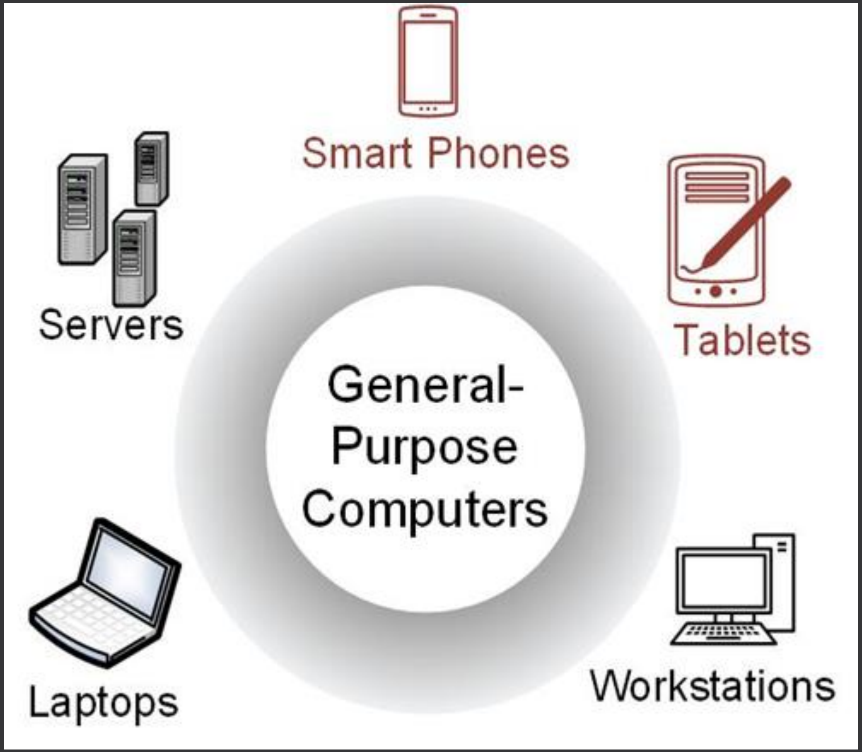
General-Purpose Computers

- Able to run a variety of software.
- Contain relatively high-performance hardware components (fast processors, data & program storage).
- Require an operating system (OS).



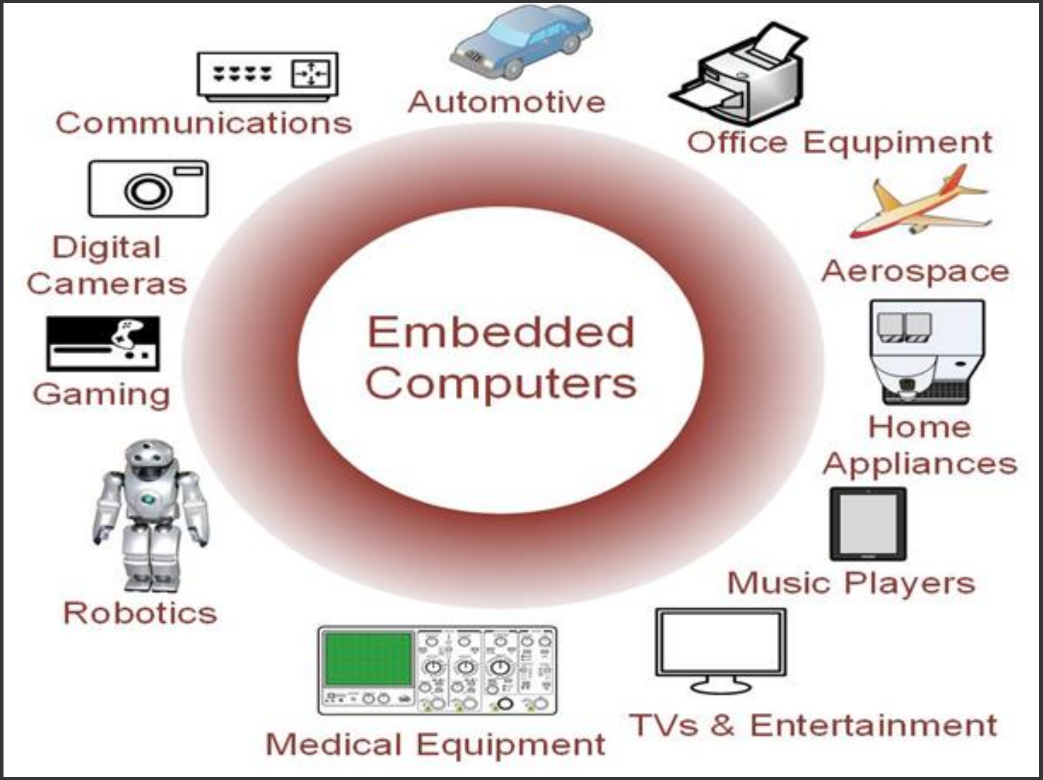
General-Purpose Computers

- Designed for heavy user interaction.
- Uses a variety of peripherals (displays, keyboards, mice, internet connections, wireless communication capability).
- Expensive ($100s - $1000s).
- Use a group of integrated circuits or chips (ICs).
  - One implements the central processing unit (CPU).
- Several implement data memory and program storage.



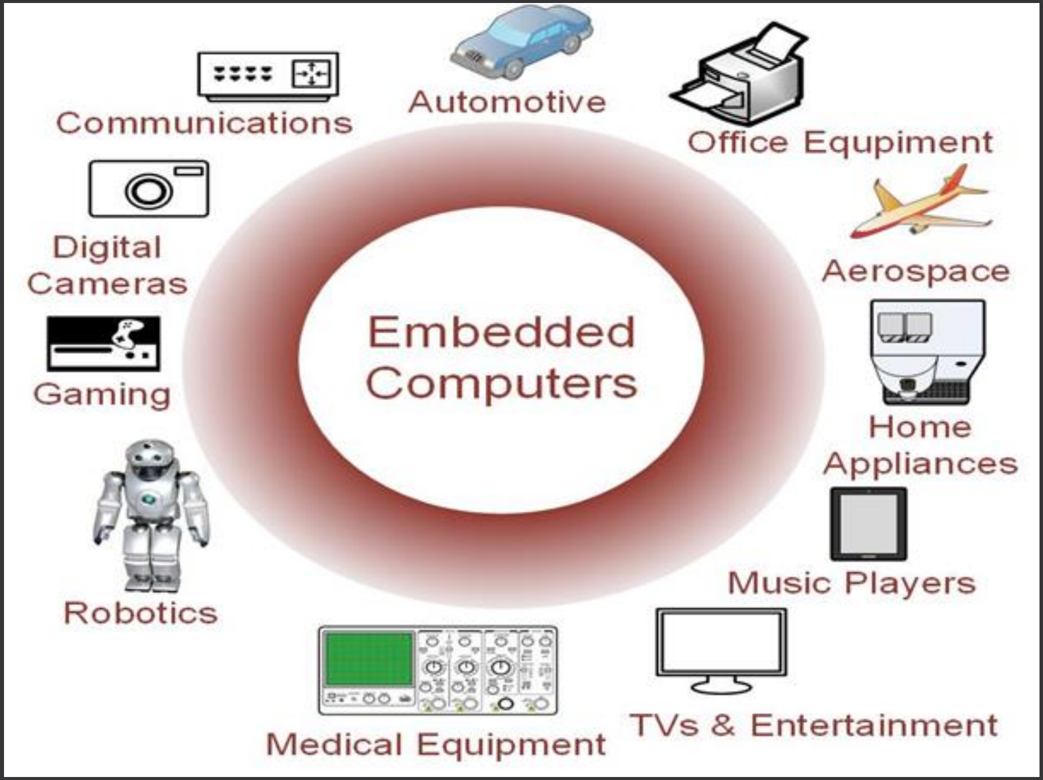
Embedded Computers

- Resources can be implemented on a single IC.
- Include a variety of peripherals (timers, analog-to-digital converters, digital-to-analog converters, serial interfaces).
- Small size makes them very versatile.



Embedded Computers

- Contains firmware (only the *needed* software which is not intended to be changed frequently).
- May contain Real Time Operating Systems (RTOS) which are used as a task scheduler.
- Low cost (10s of cents to a few dollars).



Kitchen analogy for the timer

- The Kitchen Setup:
- Timer as Kitchen Stopwatch:
- Different Types of Timers as Specialized Kitchen Tools:
- Coordination of Cooking Tasks:
- Watchdog Timer as the Sous Chef:
- Real-Time Clock (RTC) as the Kitchen Wall Clock:
- Handling Timer Overflows as Kitchen Reset:
- Fine-Tuning for Efficiency:
- Emergency Timers for Urgent Situations:

Types of Timer in ES

- **1. Watchdog Timer**
- **Purpose**: Malfunction or if the software fails to reset
- **Operation**: timeout value before being reset
- **Usage**: software hang-ups or crashes.
- **2. Interval Timer**
- **Purpose**: interrupts at regular intervals
- **Operation**: specific count value and runs until it reaches zero, followed by interrupt
- **Usage**: Used in real-time operating systems (RTOS) for periodic task execution.

Software Reliability

- Embedded systems must be able to cope with both hardware and software **anomalies** to be truly robust.
- In many cases, embedded devices operate in total **isolation** and are **not accessible** to an operator.
- **Manually resetting** a device in this scenario when its **software "hangs" is not possible**.
- In extreme cases, this can result in **damaged hardware or loss of life** and incur significant cost impact.

The Clementine

- In 1994, a deep space probe, the **Clementine**, was launched to make observations of the moon and a large asteroid (1620 Geographos).
- After months of operation, a **software exception caused** a control thruster to **fire for 11 minutes**, which **depleted most of the remaining fuel** and caused the probe to rotate at 80 RPM.
- Control was eventually regained, but it was too late to successfully complete the mission.

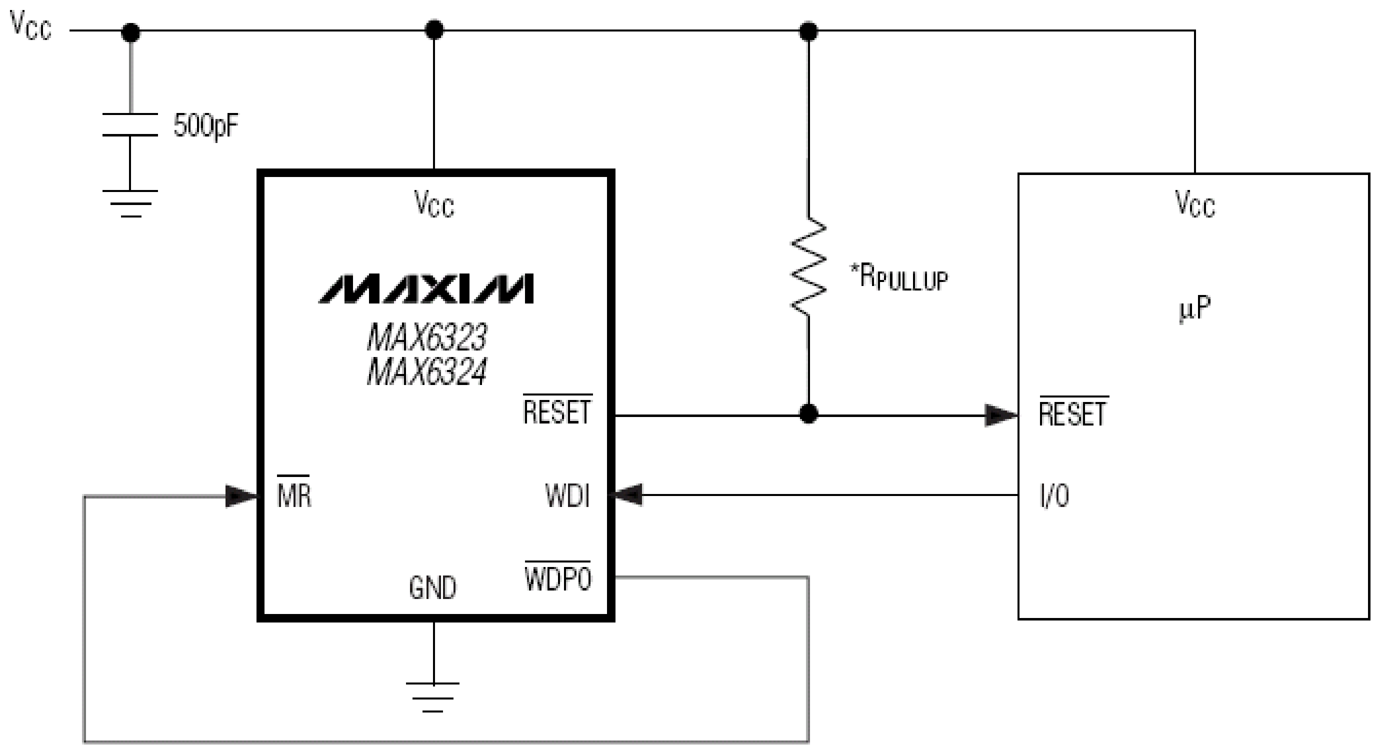
Watchdog Timers

- While it is not possible **to cope with *all*** hardware and software **anomalies**, the developer can employ the use of **watchdog timers** to help **mitigate the risks**.
- A watchdog timer is a **hardware timing device** that triggers a system reset, or similar operation, after a designated amount of **time has elapsed**.
- A watchdog timer can be either a **stand-alone hardware component** or **built in**to the processor itself.
- To **avoid a reset**, an application must **periodically reset the watchdog timer** before this interval elapses.   This is also known as "**kicking the dog**".

External Watchdogs

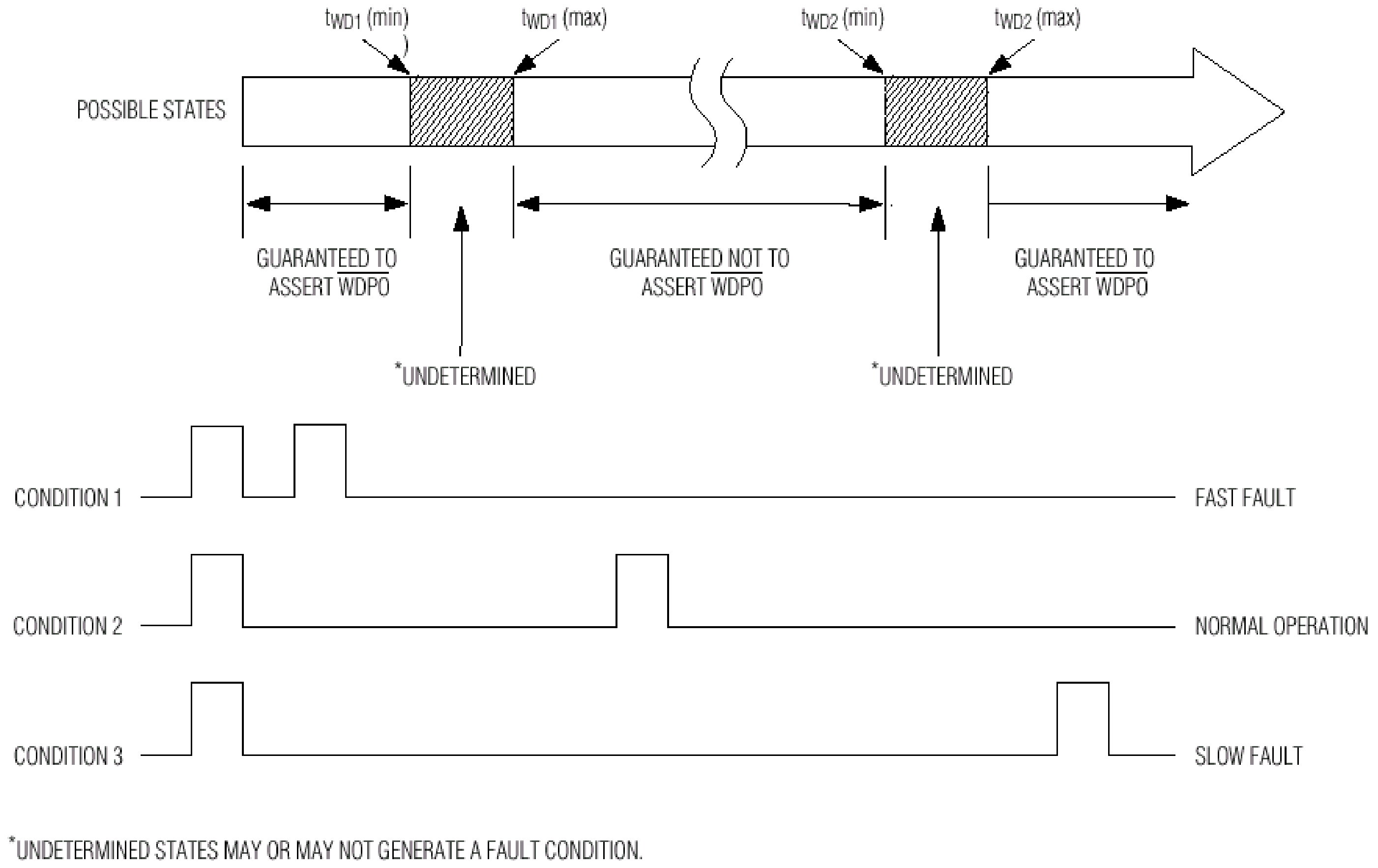
- External watchdog timers are integrated circuits that physically assert the reset pin of the processor.
- The Processor must assert an output pin in some fashion to reset the timing mechanism of the watchdog.
- This type of watchdog is generally considered the most appropriate because of the complete independence of the watchdog from the processor.
- Some external watchdogs feature a “windowed” reset.
  - **Enforces timing constraints** for a proper watchdog reset.
  - **Minimizes likelihood** of errant software resetting the watchdog.

External Watchdog Schematic



*MAX6324 ONLY

Windowed Watchdog Operation

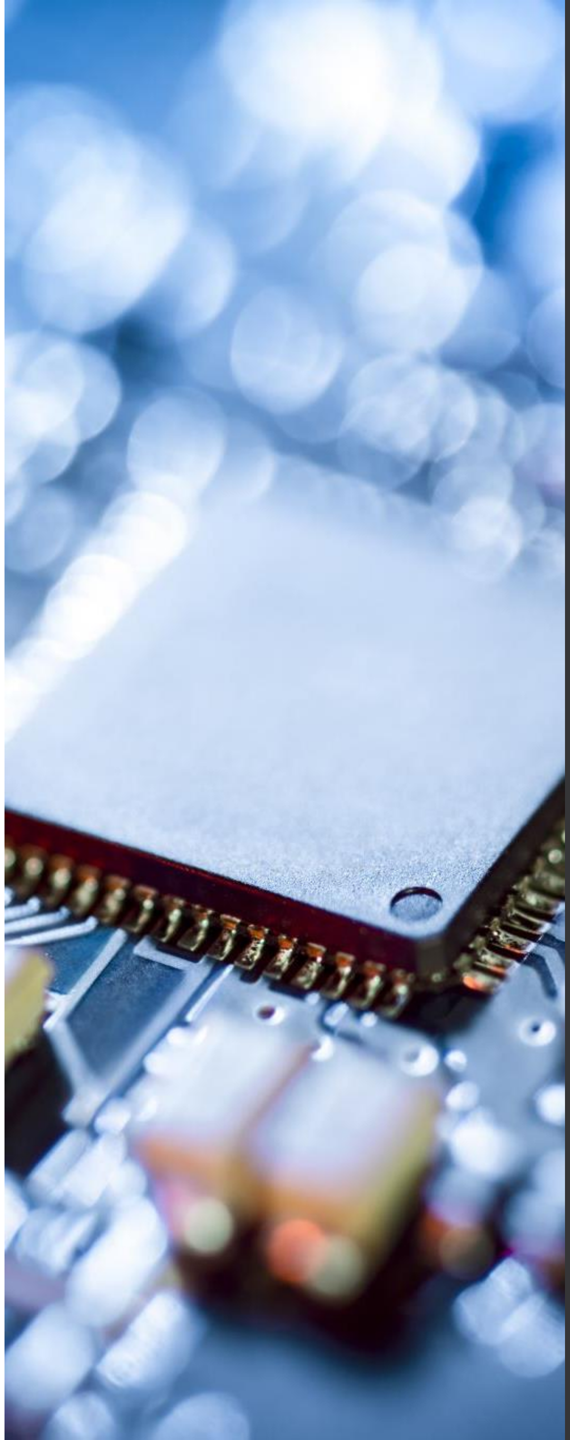


t_{WD1} (min)
t_{WD1} (max)
t_{WD2} (min)
t_{WD2} (max)
POSSIBLE STATES
GUARANTEED TO ASSERT $\overline{\text{WDPO}}$
GUARANTEED NOT TO ASSERT $\overline{\text{WDPO}}$
GUARANTEED TO ASSERT $\overline{\text{WDPO}}$
*UNDETERMINED
*UNDETERMINED
CONDITION 1
FAST FAULT
CONDITION 2
NORMAL OPERATION
CONDITION 3
SLOW FAULT
*UNDETERMINED STATES MAY OR MAY NOT GENERATE A FAULT CONDITION.

Maxim MAX6323

Internal Watchdogs

- Many processors and microcontrollers have built-in **watchdog circuitry available to the programmer**.
- This typically consists of a memory-mapped counter that **triggers a non-maskable interrupt (NMI)**, or reset, when the counter reaches a predefined value.
- **Instead** of issuing a reset via an I/O pin assertion, an internal counter of reset to an initial value.
- Watchdog **configuration** is controlled user software.
- Watchdog may even be used as a general-purpose timer in some cases.



Types of Timer in ES

3. Counter

Purpose: External events or pulses.

Operation: Increments or decrements

Usage: CNC (Computerized Numerical Control), PWM

4. Programmable Timer

Purpose: programmed for different modes and intervals to offer flexibility

Operation: square waves, PWM signals, or precise delays.

Usage: motor control and signal generation.

Types of Timer in ES

- **5. Real-Time Clock (RTC)**
- **Purpose**: current date and time.
- **Operation**: timekeeping functions.
- **Usage**: portable electronics, wearable devices, and energy-efficient system, RTOS
- **6. Capture Timer**
- **Purpose**: Records the timer value at the moment of an external event.
- **Operation**: storing this value in a register.
- **Usage**: PWM.

Types of Timer in ES

- **7. Compare Timer**
- **Purpose**: Compare the predefined value of timer, if matches it generates an output or interrupt
- **Operation**: Continuously compares the timer count
- **Usage**: Microwave oven, Commonly used in pulse-width modulation (PWM) for motor control, generating precise time delays, and creating periodic signals.
- **8. PWM (Pulse Width Modulation) Timer**
- **Purpose**: Generates PWM signals for controlling devices
- **Operation**: Modulates the width of the output pulse based on a timer
- **Usage**: Used in motor speed control, dimming LEDs, and controlling power to devices.

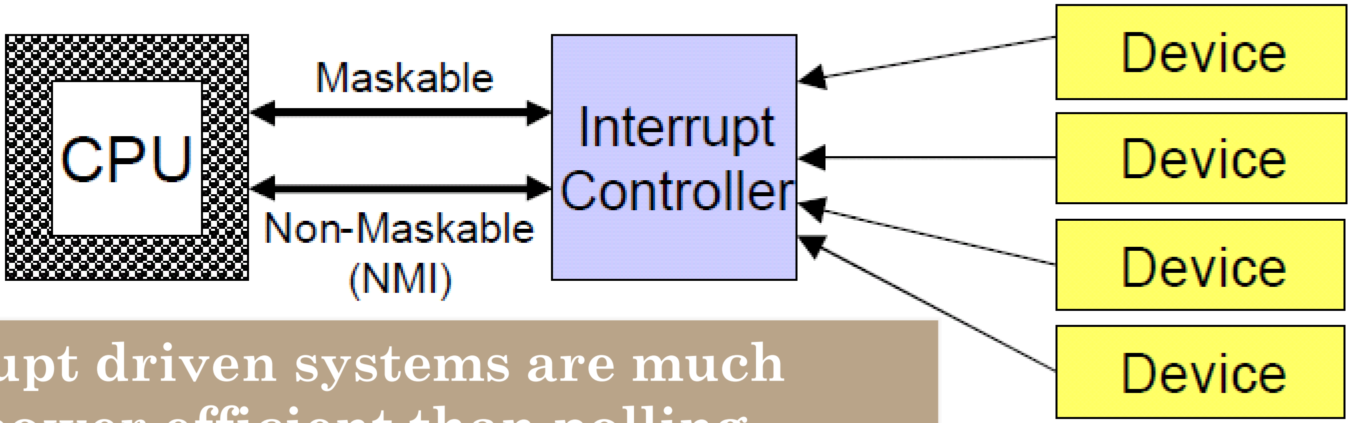
Dealing with asynchronous events

- Many sources of "events" during program execution
  - Application makes a system call
  - Peripheral needs attention or has completed a requested action
- How do we know that an event has occurred?
- Broadly, two options to "detect" events
  - Polling
    - We can repeatedly poll the app/processor/peripherals
    - When an event occurs, detect this via a poll and take action
  - Interrupts
    - Let the app/processors/peripheral notify us instead
    - Take action when such a notification occurs (or shortly later)

Dealing with asynchronous events

- Many sources of “events” during program execution
  - Application makes a system call
  - Software executes instruction illegally (e.g. divides by zero)
  - Peripheral needs attention or has completed a requested action
- How do we know that an event has occurred?
- Broadly, two options to “detect” events
  - Polling
    - We can repeatedly poll the app/processor/peripherals
    - When an event occurs, detect this via a poll and take action
  - Interrupts
    - Let the app/processors/peripheral notify us instead
    - Take action when such a notification occurs (or shortly later)

Introduction



Interrupt driven systems are much more power efficient than polling

In normal cases Interrupts save CPU time, allowing it to do more works while waiting for interrupt signals. Better

Interrupts preempt the current execution flow, disrupting CPU scheduling.

Interrupts introduce some overhead

Consume much more power

CPU periodically checks each device to see if it needs service

- ✗ takes CPU time even when no requests pending
- ✗ overhead may be reduced at expense of response time
- ✓ can be efficient if events arrive rapidly

"Polling is like picking up your phone every few seconds to see if you have a call. …"

Polling is 100% under control of CPU scheduler

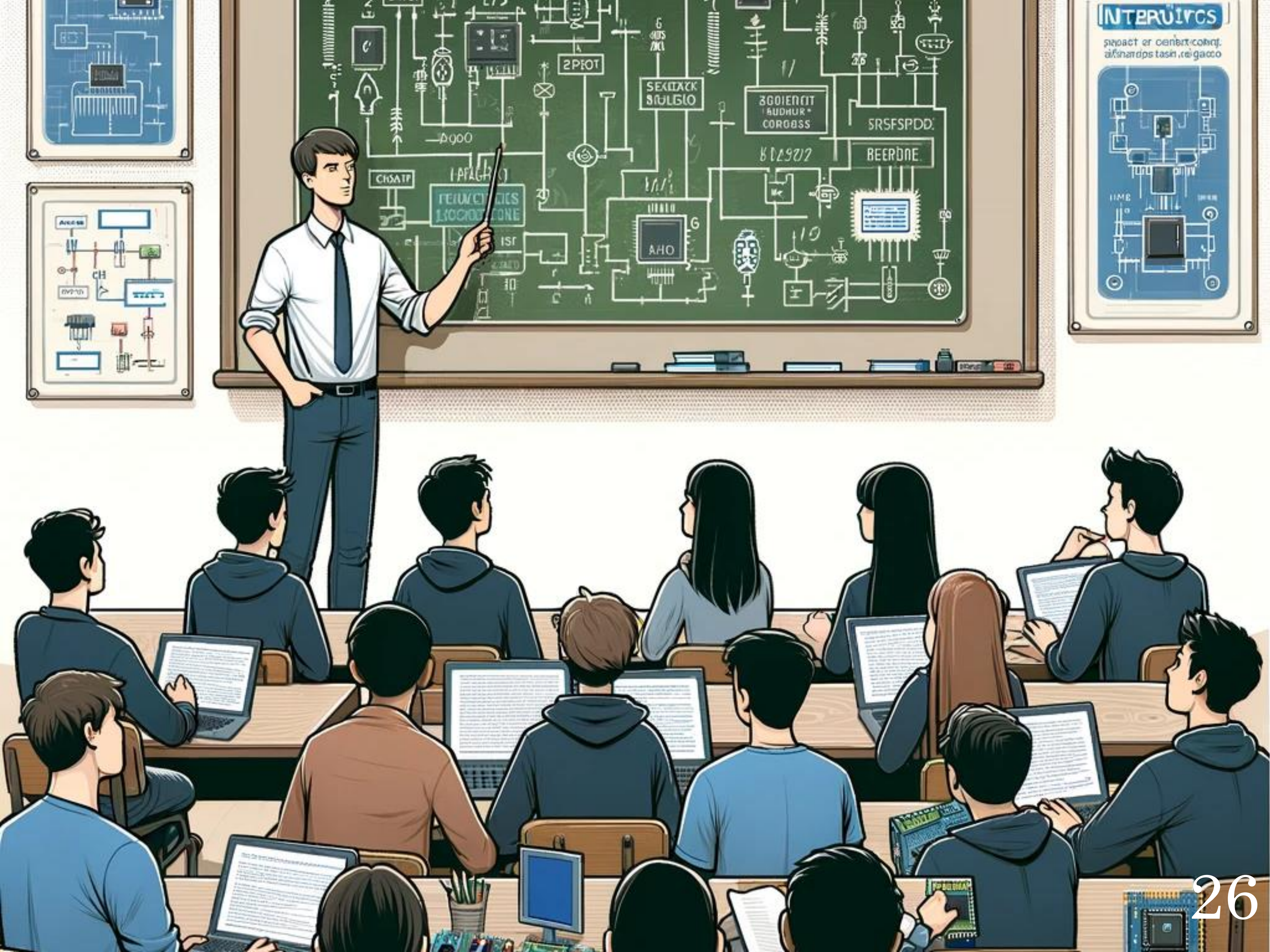
Interrupts VS Polling

"Polling is like picking up your phone every few seconds to see if you have a call. Interrupts are like waiting for the phone to ring."

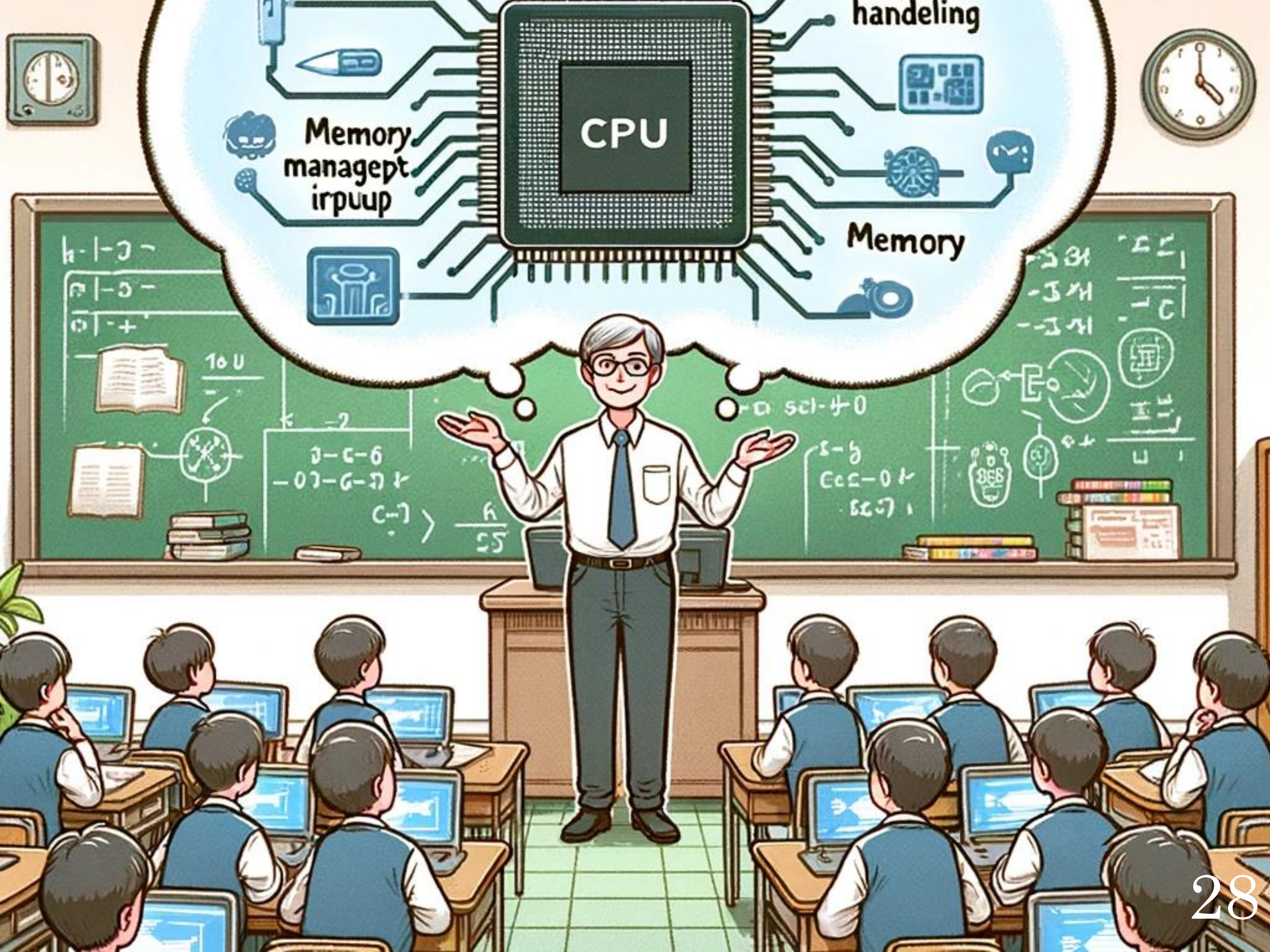
Interrupts win if processor has other work to do and event response time is not critical

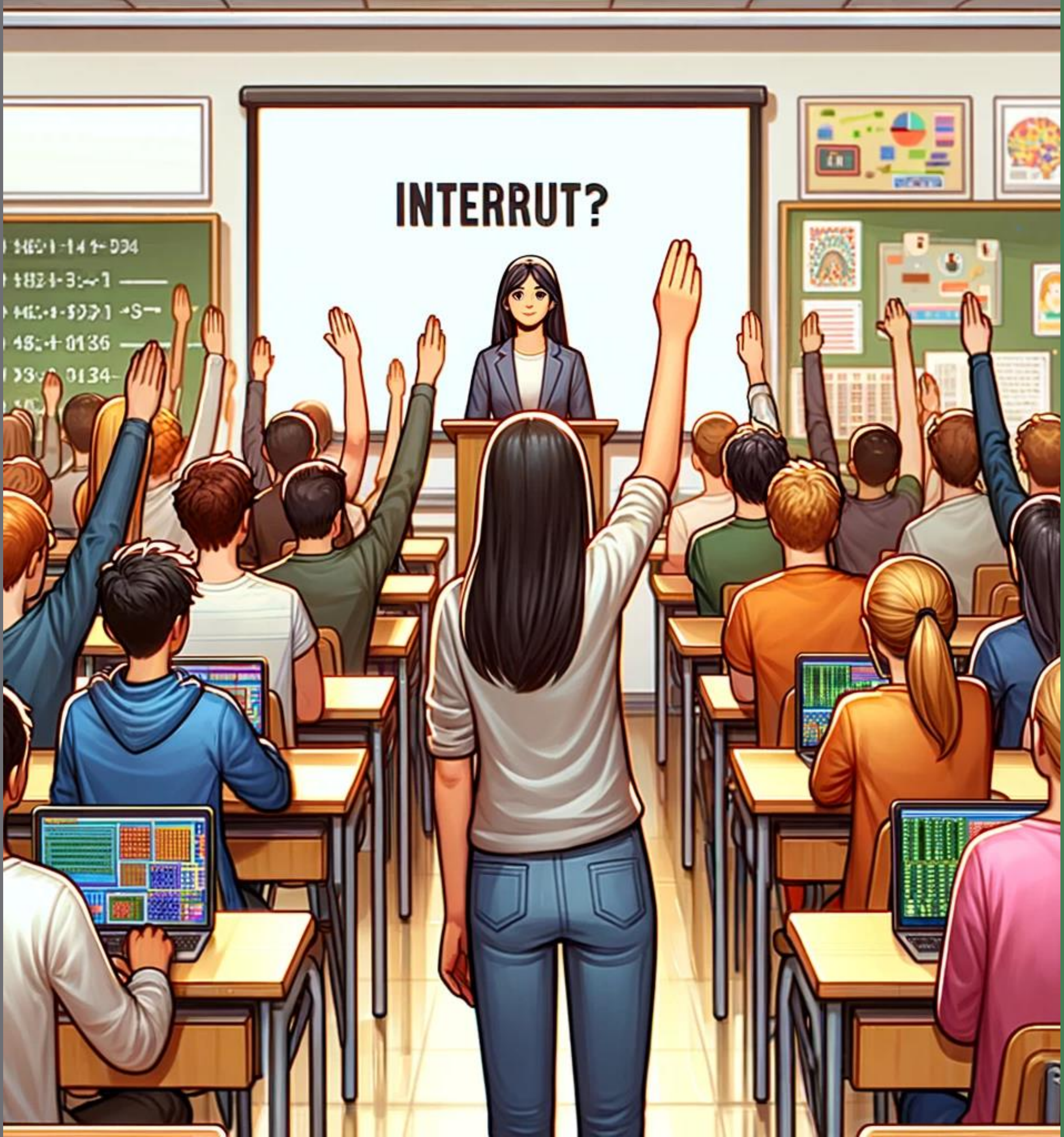
Polling can be better if processor has to respond to an event as soon as possible

Interrupts as the student teacher classroom analogy

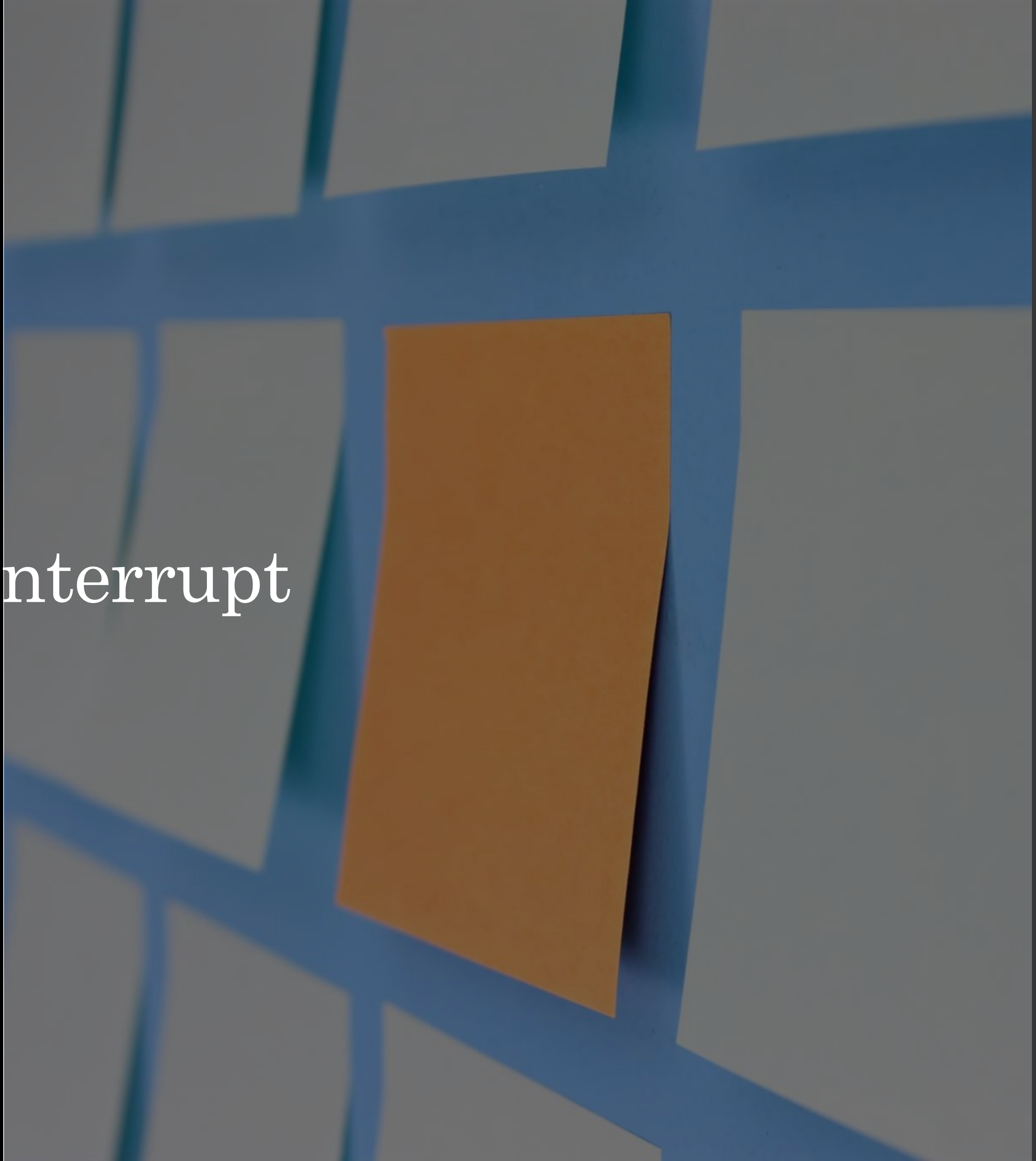


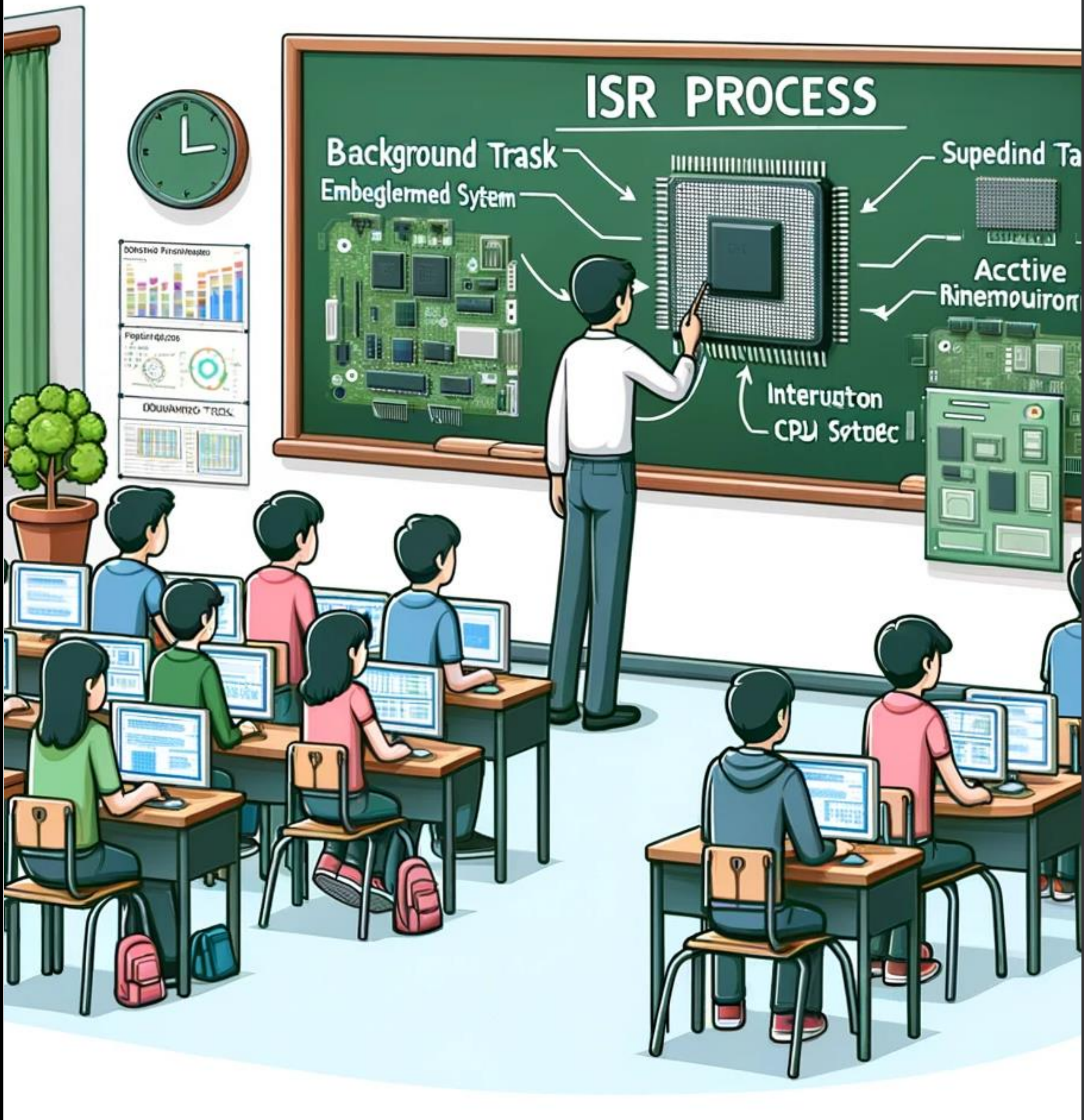
Dlema Processing
Senecor oupoees an UDorming
Soubetes Montiporting

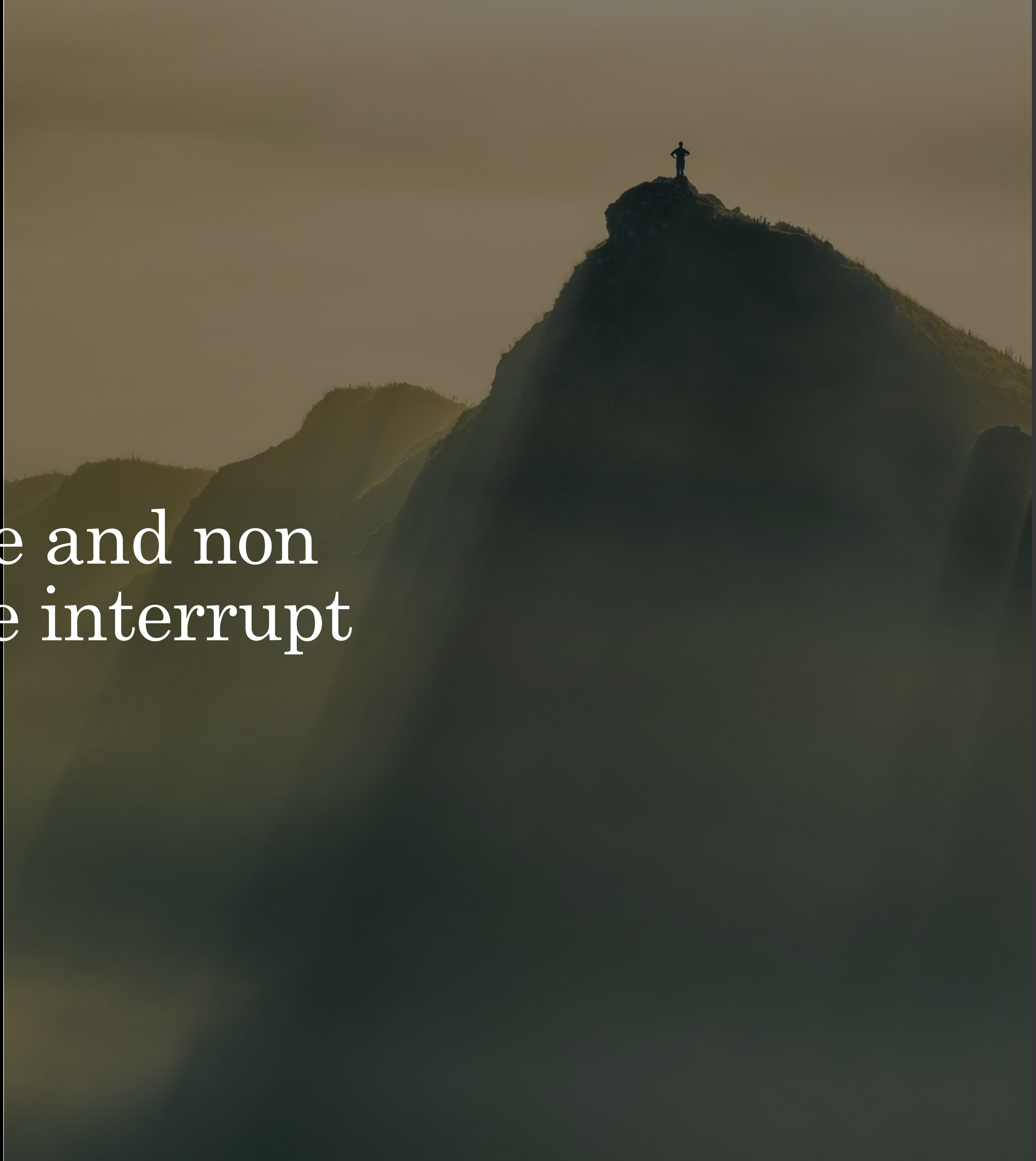

handeling
Memory managept irpuup
CPU
Memory


INTERRUT?

Priority interrupt



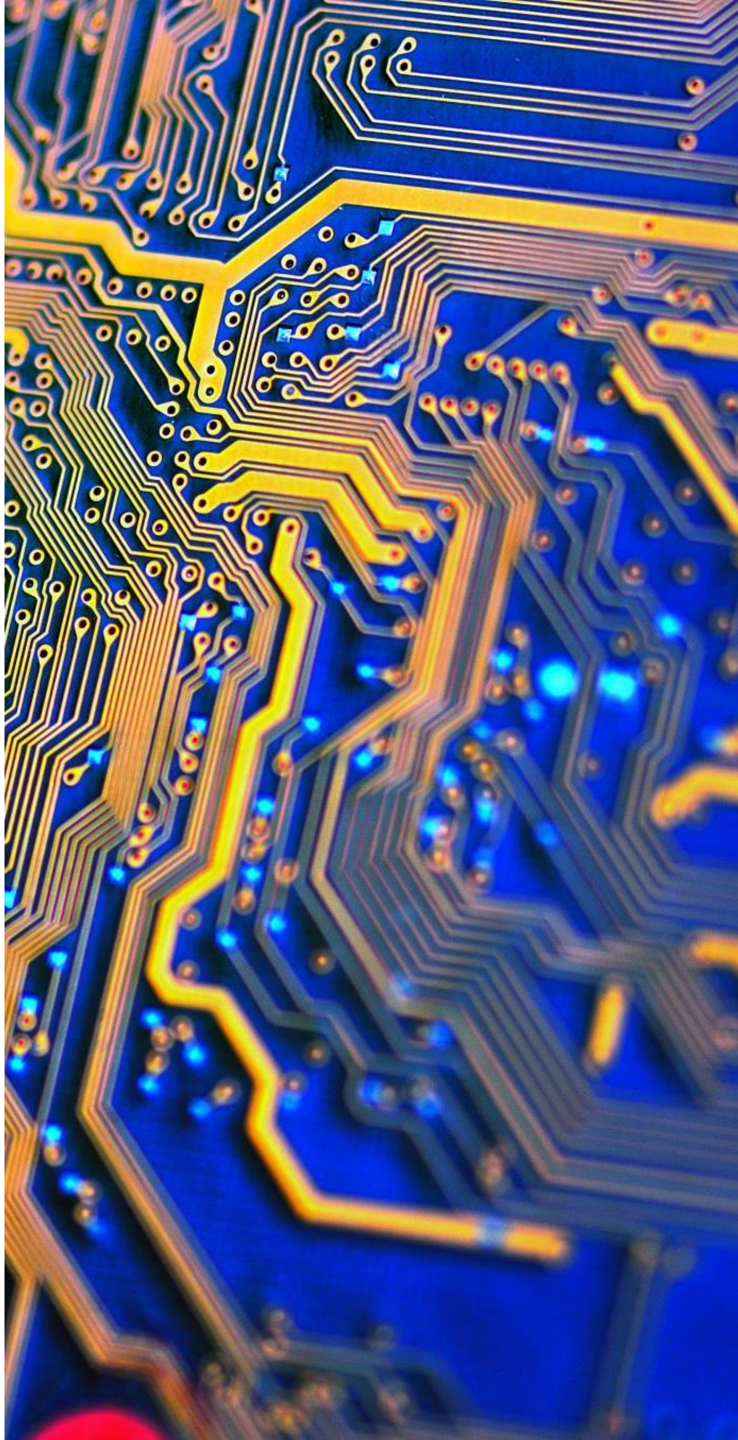

ISR PROCESS
Background Trask
Embeglermed Sytem
Supedind Ta
Acctive
Rinemouiron
Interugton
CPU Sytoec



Maskable and non maskable interrupt

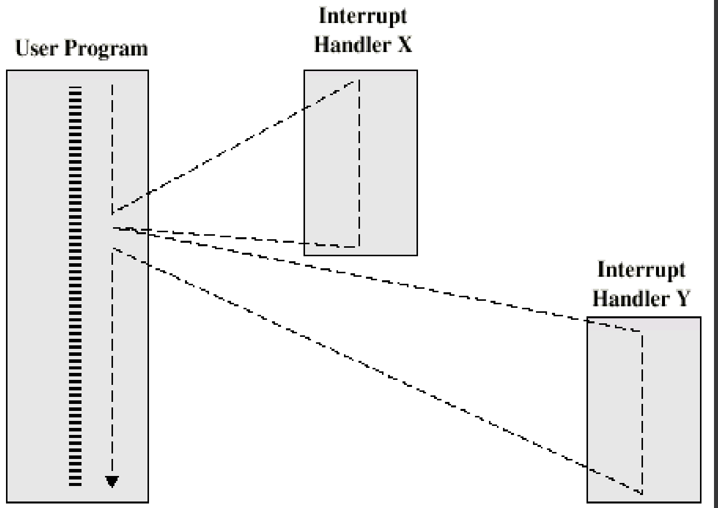
Interrupt Types

- Hardware
- Software
- Vectored: the address of the interrupt service routine (ISR) is predefined ; so, when interrupt occurs, the processor automatically jumps to this specific address to handle the interrupt.
- Non Vectored (like Trap) : interrupting device does not provide an ISR address. Instead, the processor must be directed to a fixed address where it can find the ISR for that interrupt.
- Maskable Interrupt (Low Priority)
- Non maskable (high priority)

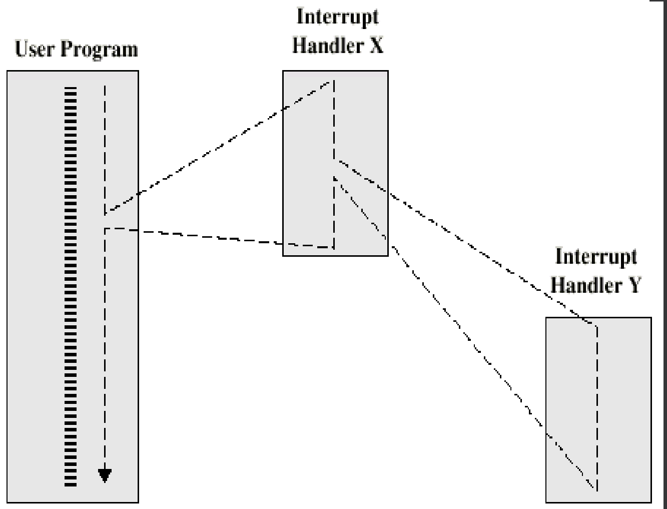


Handling Multiple **Interrupts**

- **Sequential approach** – once an interrupt handler has been started it runs to completion
  - (+) **Simpler**
  - (-) **Does not handle priority interrupts** well
    Example: Incoming data might be lost.



- **Nested approach** – a higher priority device can interrupt a **lower priority one**.
  - (+) **More complex**
  - (-) Interrupts get handled in order of priority.

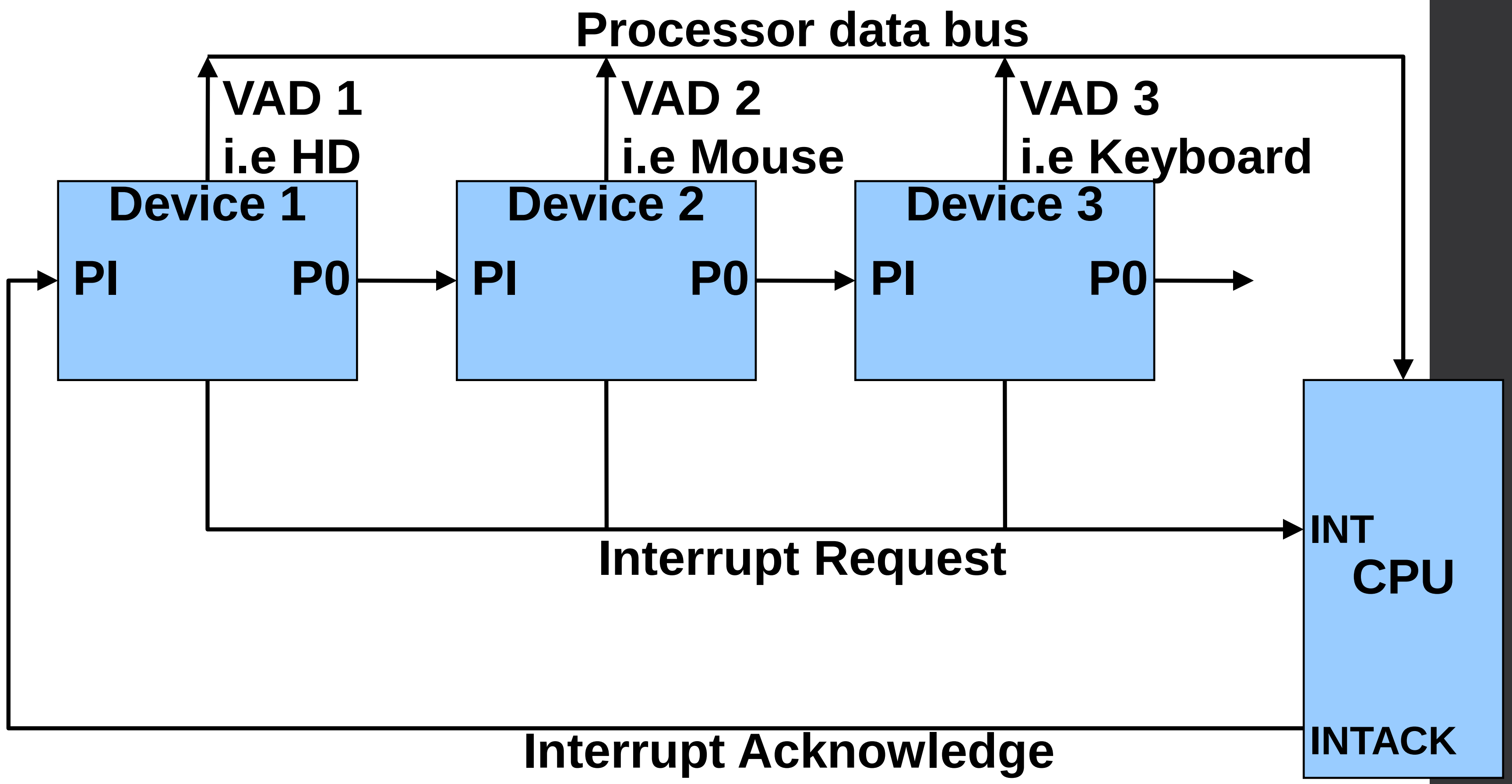


Priority Interrupts

- Daisy-Chain Priority
  - **Hardware** solution
  - **Serial connection** of all devices that request interrupts.
  - Device with the highest priority takes first position, 2nd highest takes 2nd position etc.
  - Interrupt request line shared by all devices.

Daisy-chain Priority Interrupt

A Serial Approach



PI: Priority Input, VAD: Vectored Address (i.e its device responsibility to give address of ISR to CPU)

Daisy-chain Priority Interrupt A Serial Approach

Advantages

- Simple Implementation
- Cost-Effective:

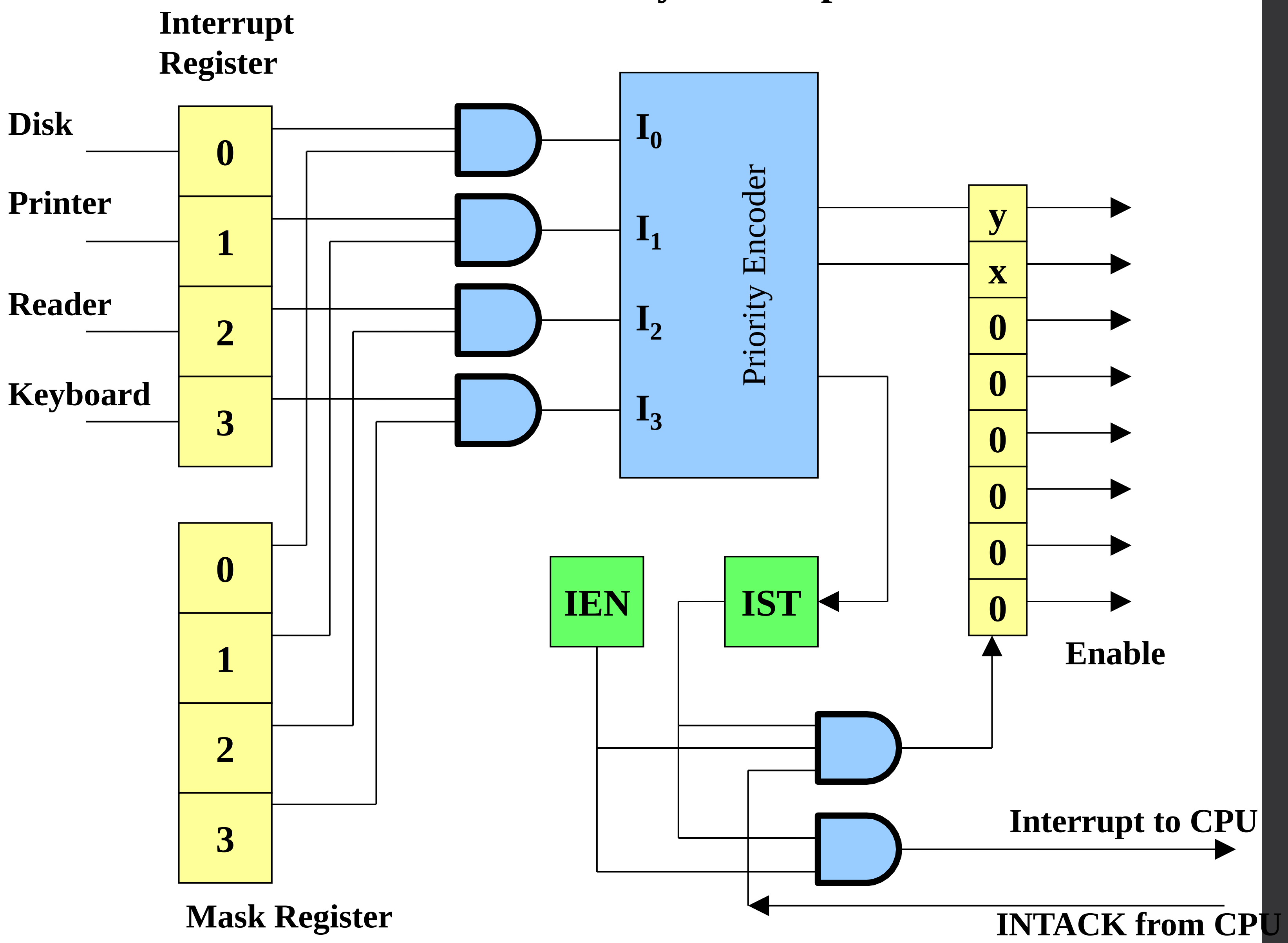
Disadvantages

- Fixed Priority
- Propagation Delay

Parallel Priority Interrupt

- Uses a register – whose bits are set separately by the interrupt signal from each device.
- Priority established according to the position of bits in the interrupt register.
- A mask register is used to control the status of each interrupt request. Mask bits set programmatically.
- Priority encoder generates low order bits of the VAD, which is transferred to the CPU.
- Encoder sets an interrupt status flip-flop IST whenever a non-masked interrupt occurs.
- Interrupt enable flip-flop provides overall control over the interrupt system.

Parallel Priority Interrupt Hardware



Interrupt Service Table (IST)

Priority Encoder

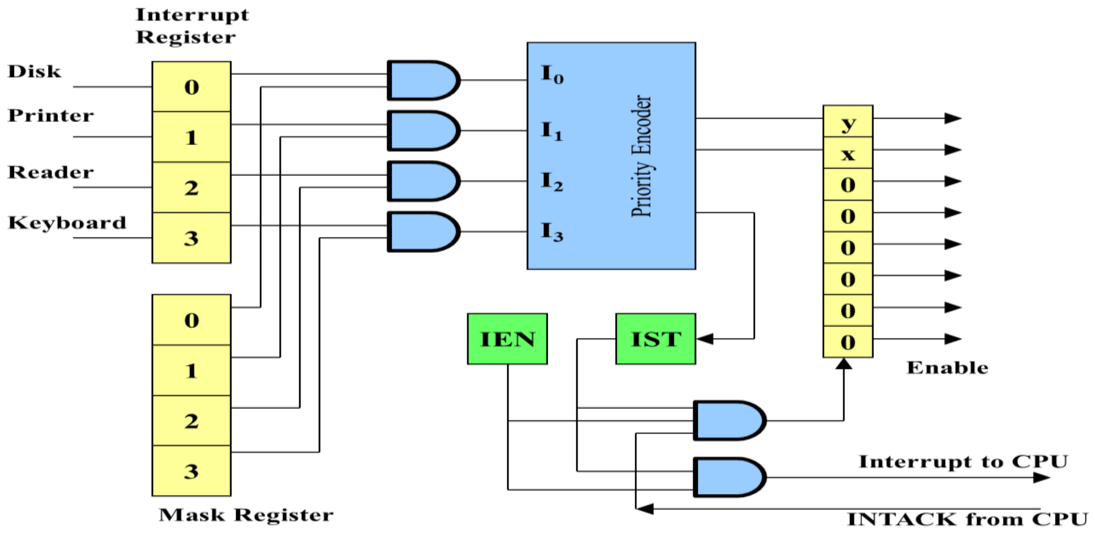
- Circuit that implements the priority function.
- Logic – if two or more inputs arrive at the same time, the input having the highest priority will take precedence.

Inputs				Outputs		
I_0	**I_1**	**I_2**	**I_3**	**X**	**Y**	**IST**
1	d	d	d	0	0	1
0	1	d	d	0	1	1
0	0	1	d	1	0	1
0	0	0	1	1	1	1
0	0	0	0	d	d	0

- Boolean functions

$X = I'_0I'_1$ $\quad$ $Y = I'_0I_1 + I'_0I'_2$ $\quad$ $IST = I_0 + I_1 + I_2 + I_3$

D stands for Disable



Inputs			
I_0	I_1	I_2	I_3
1	d	d	d
0	1	d	d
0	0	1	d
0	0	0	1
0	0	0	0

Outputs		
X	Y	IST
0	0	1
0	1	1
1	0	1
1	1	1
d	d	0

References:

1. Digital Design and Computer Architecture, 2nd Edition, David Money Harris and Sarah L. Harris, Elsevier.
2. https://link.springer.com/chapter/10.1007/978-1-349-06155-6_10#:~:text=Shift%20instructions%20allow%20the%20bits,when%20the%20shift%20is%20made.
3. William Stallings Computer Organization and Architecture 8th Edition. **Wen-mei & Patel: http://courses.ece.uiuc.edu/ece511/lectures/lecture3.ppt**
4. **Patterson: http://www.cs.berkeley.edu/~pattrsn/252S98/index.html**
5. **Rabaey: (He used lots of Patterson material): http://bwrc.eecs.berkeley.edu/Classes/CS252/index.htm**
6. **Katz: (Again, he borrowed heavily from Patterson): http://http.cs.berkeley.edu/~randy/Courses/CS252.F95/CS252.Intro.html**
7. **Mark Hill: (Follows text fairly well): http://www.cs.wisc.edu/~markhill/cs752/**
8. **Prabal Dutta, University of Michigan, Interrupts, Exceptions, Traps, Faults & ARM's Nested Vectored Interrupt Controller**
9. **Introduction to Computer Architecture, by based on slides from Gojko Babic**
10. www.youtube.com/c/DigitalLogicProgramming_LaMeres by **Brock J. LaMeres, Ph.D**
11. **In proceedings(regehr2005preventing) Regehr, J. & Duongsaa, U. Preventing interrupt overload ACM SIGPLAN Notices 2005, Vol. 40(7), pp. 50-58**
12. **http://www.cs.toronto.edu/~demke/469F.06/Lectures/Lecture6.pdf**
13. **In proceedings(lewandowski2007modeling) Lewandowski, M., Stanovich, M., Baker, T., Gopalan, K. & Wang, A. Modeling device driver effects in real-time schedulability analysis: Study of a network driver Real Time and Embedded Technology and Applications Symposium, 2007. RTAS'07. 13th IEEE 2007, pp. 57-68**
14. **J. Regehr's work titled, "Preventing Interrupt Overload".**
15. **Saud Wasly,** Interrupt Management In Real-time Systems

References

Shibu K V, “Introduction to Embedded Systems”, Tata McGraw Hill, 2009.

Raj Kamal, “Embedded Systems: Architecture and Programming”, Tata McGraw Hill, 2008.
Shrishail Bhat, Dept. of ECE, AIITM Bhalki

Additional Reading Material:

Smith, J., & Bhat, S. (2015). "Secure Bootloader Design for Embedded Systems." IEEE Transactions on Dependable and Secure Computing, 12(4), 398-410.

Gupta, A., Kamal, R., & Patel, H. (2017). "Machine Learning Applications in Embedded Systems: A Survey." IEEE Transactions on Emerging Topics in Computing, 5(3), 398-410.

Kumar, S., & Shibu, K. V. (2019). "Energy-Efficient Task Scheduling Algorithms for Multi-Core Embedded Systems." IEEE Transactions on Computers, 68(8), 1189-1202.

Bhat, S., & Singh, A. (2016). "Wireless Sensor Networks for Environmental Monitoring: A Survey." ACM Transactions on Sensor Networks, 13(2), Article 15.

Patel, R., & Kamal, R. (2018). "Design Space Exploration Techniques for Embedded System Architectures." ACM Transactions on Embedded Computing Systems, 17(3), Article 78.

Jain, S., & Bhat, S. (2020). "Fault Diagnosis Techniques in Real-Time Embedded Systems: A Comprehensive Review." ACM Transactions on Design Automation of Electronic Systems, 25(4), Article 40.

Lnu.se