

# RTOS

SESSION 1

**AMIT**

# ■ Content

- OPERATING SYSTEM MODELS
- Types of Systems.
- Software architecture concepts.
- Type of RTS.
- What is RTOS?
- Difference between GPOS and RTOS.
- Software architecture concepts.
- Why do we need RTOS?
- Let's Create the Scheduler APIs

## ■ OPERATING SYSTEM MODELS

There are many techniques for writing embedded software.

For low-complexity systems, the classic way does not use an RTOS but is designed as super loops or foreground/background systems.

**(super-loop and background)** An application consists of an infinite loop that calls one or more functions in succession to perform the desired operations

**(foreground)** Interrupt service routines (I S Rs) are used to handle the asynchronous, real-time parts of the application

In this architecture, the functions implementing the various functionalities are inherently, even if not formally declared as such, some kind of finite state machines, spinning around and switching states based on inputs provided by the ISRs.

# Needed Definitions :

- **Deterministic:**  
any system normally described as deterministic if it has a predictable execution pattern for Tasks .
- **Responsiveness:**  
how fast the System will handle a sudden event.
- **Periodicity:**  
how often the System will Repeat the handling of a specific Task.

# Types of Systems:

## 1- SUPER LOOP SYSTEM:

It is the simplest system at which all tasks performed within a super loop. **The ADVANTAGES of super loop:**

- **High Determinism.**  
all tasks runs sequentially and predictably as Example.
- **Easy to implement**  
no need for extra Software parameters to control Execution
- **No need for extra Hardware for handling the tasks.**

```
while(1)
{
    DIO();
    ADC();
    UART();
    Timer();
}
```

## The DISADVANTAGES of super loop:

- **Less Responsiveness**

if UART Received a data during the execution of Timer, it would not be handled

Until finishing Timer, DIO, and ADC tasks.

```
while(1)
{
    DIO();
    ADC();
    UART();
    Timer();
}
```

- **High Periodicity**

As it is fixed periodicity. The execution time of DIO is 10 ms, ADC is 15 ms, UART is 10 ms and Timer is 40 ms, So as an example the periodicity of DIO will be:

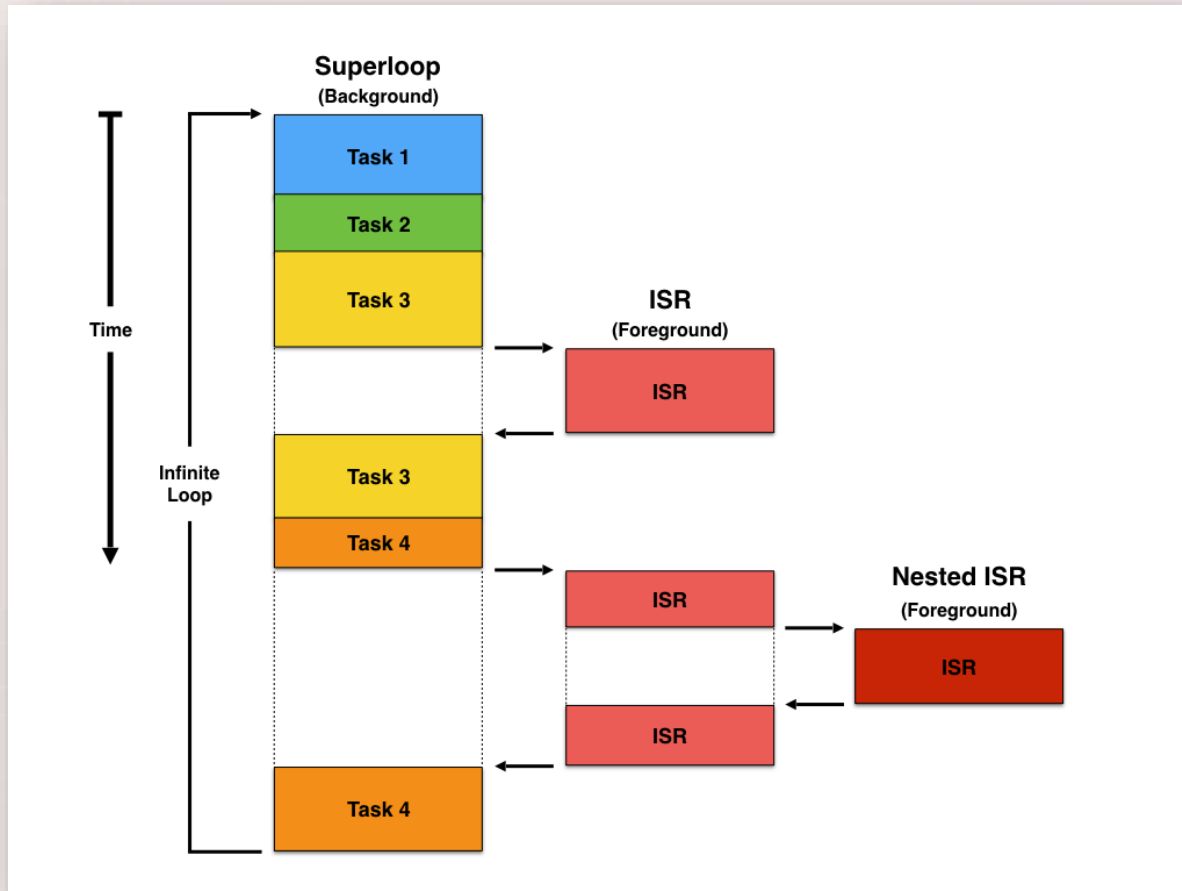
$10 + 15 + 10 + 40 = 75$  ms, the DIO time is calculated also because the time is counted from starting execution of DIO.

- **High Power consumption**

For example, the ADC task will be run even if there are no events from ADC.

## 2 - FORE-GROUND BACK-GROUND SYSTEM:

the system at which a higher Priority task could interrupt the execution of a lower Priority task.



```
while(1)
{
    DIO();
    ADC();
    UART();
    Timer();
}

ISR(ADC)
{
    //code of ADC
}
```

## The ADVANTAGES of Fore-ground Back-ground System:

- **higher responsiveness**  
because some tasks will be Executed unpredictably once it has an event or Data.
- **Low power consumption**  
because ADC's task will only run when its event is fired.
- **High Portability**



## The DISADVANTAGES of Fore/Background System:

- **low determinism.**

because there is no way to know when exactly the ADC task will be run, so there is no insurance that this task will meet its deadline.

- **needs extra Hardware**

as programmable interrupt controller (PIC) and some extra memory for IVT “Interrupt Vector Table”.

- **Complexity of implementation.**

because the code is divided into several TASKs and ISRs

- **The periodicity of this system can not be calculated.**

### 3- REAL-TIME OPERATING SYSTEM:

It consists of two parts: Operating System and Real Time

#### Operating System:

a software program that manages and operates Tasks and Devices considered as a platform between the user and the computer hardware

#### Definition Of RTS :

is a system in which the correct performance of its functions not Only depends on the results produced by the system but also the time at which these results are produced.

**i.e.** system that achieves its time constraints and Produces correct outputs at the correct Time.

# SOFTWARE ARCHITECTURE CONCEPTS :

## **MULTI-TASKING :**

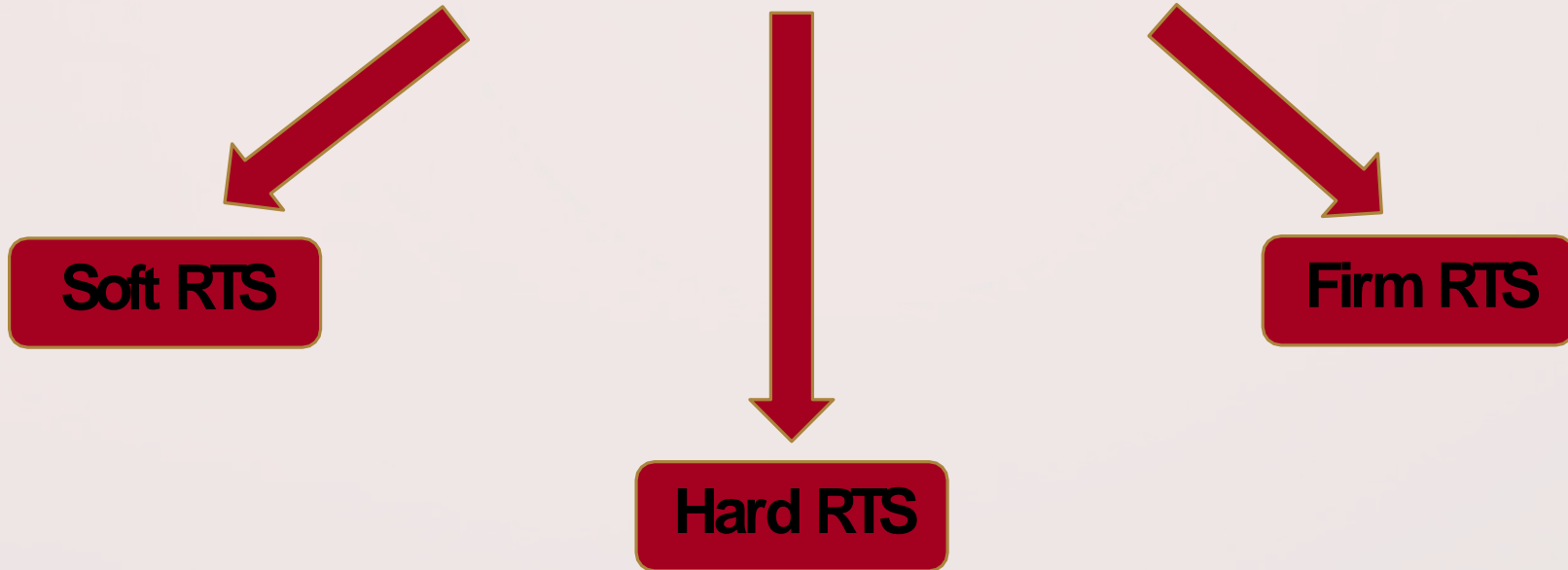
The process of scheduling and switching between tasks with a very high speed

making the user feel that tasks are performed at the same time.

## **MULTI-CORE :**

The Feature of executing several tasks at the same time using several CPUs built-in some kind of processor.

## TYPES Of RTS :



## Hard RTS

- it must be very accurate in timing because any small failure will cause a huge danger on a human or a machine. so, the deadline is handled very strictly
- Tasks can't miss deadline
- must start executing on specified scheduled time
- must be completed within the assigned time duration.
- Used at safety critical systems as

### Example :

- Car Airbag system
- Aircraft systems.
- Medical critical care system.
- Anti-Lock braking system

## Firm RTS

- It is used into systems that it does not cause a danger on human, but any failure into it may cause a big lost in money or damaging in machine. These type of RTOS also need to follow the deadlines.
- Few Tasks allowed to miss deadline
- More than that lead to system failure

### Example:

- Tracking and navigation systems
- Various types of Multimedia applications.
- Various types of Production lines.

## Soft RTS

- A system that doesn't cause any danger on human or money lose machine damaging .  
Soft Real time RTOS, accepts some delays by the Operating system. In this type of RTOS, there is a deadline assigned for a specific job, but a delay for a small amount of time is acceptable. So, deadlines are handled softly by this type of RTOS.
- Task May operate with some latency
- missing deadline affect Quality of system but not fatal or considered a failure

### Example :

- video Streaming
- WhatsApp chatting
- Online Games

# WHAT IS RTOS ?

The system that achieve the real-time concepts and the system which used at time critical systems to achieve its timing requirement .  
By managing Application and HW layer.



CMSIS

NUCLEUS

Based on ARM architectures



VxWorks supports AMD/Intel architecture , ARM architectures



Open source One code base for 40+ MCU architectures



# DIFFERENCE BETWEEN GPOS AND RTOS :

## General Purpose Operating Systems:

- Not Deterministic
- Un-predictable response times
- No time bound
- High priority tasks can't nesting low priority tasks
- Unlimited memory resources



## **Real-Time Operating Systems:**

- **Deterministic**
- **Predictable response times**
- **Time bound**
- **High priority tasks nesting low priority tasks**
- **Priority based scheduling**
- **Limited memory resources**

# **RTOS**

# WHAT IS THE COMMON BETWEEN GPOS AND RTOS ?



- **Multi-tasking Support**
- **Software and hardware Management**
- **Provide Hardware abstraction from App programs**

## SOFTWARE ARCHITECTURE CONCEPTS :

**TIME-Driven Systems** is a Software programming technique, where the control flow of the computer program is driven by a Timer clock which is used in Real-time systems

**EVENT-Driven Systems** is a Software programming technique in which the flow of the program is determined by events such as button presses, key presses, sensor outputs,...etc

# WHY DO WE NEED RTOS ?

- High Deterministic
- High Responsiveness
- Multitasking, alone, is enough

Giving the illusion to the user that tasks are performed at the same time.

- Lower development time.

Focus on the development, not on the timing and scheduling of each task.

- Break a complex problem into smaller pieces.

- possibility of adding new tasks without re-calculating and tweaking the timing as super loops

- Avoid spaghetti code

- **DISSADVANTAGES:** Extra processing and extra memory

HOW TO TOGGLE LED1, LED2, LED3 EACH 25mS,  
20mS, 30mS?



**Try to implement using a timer with an interrupt  
by using the concept of pointer to function**



## Lets CREATE THE SCHEDULER APIs

- Now, we will create a software module Called Scheduler, contains the following APIs:

**void** Scheduler\_Init(**void**);

**void** Scheduler\_CreateTask (**void** (\*ptr)(**void**), **u16** periodicity, **u8** priority );

**void** Scheduler\_Start(**void**);

**void** Scheduler\_KillTask( **u8** priority);

**void** Scheduler\_SuspendTask( **u8** priority);

**void** Scheduler\_ResumeTask( **u8** priority);

**void** Scheduler(**void**);



## 2- CALCULATION AND CONFIGURATION OF TICK TIME:

- First, we will define the type of timer that we will use, as a configuration definition.
- The initialization of scheduler will be same the initialization of timers, but the timer will be a solid configuration to Compare-Match mode to be easier to set an integer number of milliseconds.
- Also, the prescaler of timer will be the highest available value “1024”, to generate bigger time if the OCRx value becomes near to overflow value.
- As an example, if we choose Timer0 or Timer2 “there are same”, on Compare-Match mode and a “1024” prescaler:

so if the OCRx value becomes near to overflow value, the Time will be:

$$\text{Time} = \frac{\text{Counts} \times \text{Prescaler}}{\text{System frequency}} = \frac{255 \times 1024}{16,000,000} \times 1000 = 16.32 \text{ milliseconds}$$

- As an example, if we choose Timer1, on Compare-Match mode and a “1024” prescaler:

so if the OCRx value becomes near to overflow value, the Time will be:

$$\text{Time} = \frac{\text{Counts} \times \text{Prescaler}}{\text{System frequency}} = \frac{65,535 \times 1024}{16,000,000} \times 1000 = 4,194.24 \text{ milliseconds}$$

## CALCULATION AND CONFIGURATION OF TICK TIME:

- we have two possibilities to choose the value of “Tick Time”:
  - If the Tick Time value is lower or equals the maximum value of OCRx:
    - 16 milliseconds when Timer0 or Timer2 are chosen.
    - 4194 milliseconds when Timer1 is chosen.
  - If the Tick Time value is higher than maximum value of OCRx.
- If the tick time is lower than or equals 16 “Timer0 or Timer2”, the TimerX is already configured as a Compare-Match mode, so the OCRx value will be set by:

$$\text{OCR}_x = \frac{\text{Tick Time} \times \text{System frequency}}{\text{Prescale}}, \text{ as an example the tick time is 3 ms,}$$

so

$$\text{OCR}_x = \frac{0.003 \times 16,000,000}{1024} = 46 \text{ counts}$$

## CALCULATION AND CONFIGURATION OF TICK TIME:

- If the tick time is higher than 16 “Timer0 or Timer2”, the TimerX is already configured as a Compare-Match mode, but the time of Tick time is higher than the maximum time, so the system must enter the ISR several times to achieve the desired tick time, but How much is the OCRx value set?
  - It can be set by 1 ms as an easier way, but it will be heavy on the CPU to enter the ISR several times than it needs, so before setting the OCRx value, we will calculate the Greatest Common Factor between the ticking time and the value between 1 and 16 “available values can be set to the OCRx”.
  - To understand the meaning of the last paragraph, we will take a pack of examples:
    - If the ticking time is 20 ms, the timer must be set to compare at 10 ms and the system will enter the ISR twice.
    - If the tick time is 35 ms, the timer must be set to compare at 7ms and the system will enter the ISR 5 times and so on.

# 1- CREATE THE TASK CONTROL BLOCK

➤ To create the task characteristics:

- C function.
- Periodicity.
- State of task.

*To collect them into one variable, we can use a structure in C “**struct**”:*

```
typedef struct
{
    void (* ptr ) (void);
    u16 periodicity;
    u8  state;
    u16 count;
}TCB_t;
```

## SCHEDULER APIs

```
void Scheduler_Init(void){
    TCCR0 = 0x0D;
    #if TICK_TIME <= 16 && TICK_TIME >=0

        OCR0= ( (TICK_TIME * F_CPU)/(1024ul * 1000ul) );
        Num_entry_ISR=1 , counter=1;
    #elif TICK_TIME > 16

        for (u8 i=1; i<=16; i++)    {
            if (TICK_TIME % i == 0) max=i;
        }

        OCR0= ( (max * F_CPU)/(1024ul * 1000ul) );
        Num_entry_ISR = (TICK_TIME)/max ; counter=
        Num_entry_ISR;
    #endif
}
```

# SCHEDULER APIs

```
void Scheduler_CreateTask(void (*ptr)(void), u16 periodicity, u8 priority )
{
    if ( ptr != NULL)
    {
        tasks[Copy_u8Priority].ptr = ptr;
        tasks[Copy_u8Priority].periodicity = Copy_u16Periodicity;
        tasks[Copy_u8Priority].count = Copy_u16Periodicity-1;
        tasks[Copy_u8Priority].state = READY;
    }
}
```

# SCHEDULER APIs

```
➤ void Scheduler_Start(void)
{
    TIMSK |= (1<<1);
    asm("SEI");
    while(1);
}

➤ void Scheduler_KillTask( u8 priority)
{
    if (priority < MAX_TASKS)
    {
        tasks[priority].state=KILLED;
        tasks[priority].ptr =NULL;
    }
}
```

# SCHEDULER APIs

- **void** Scheduler\_SuspendTask( **u8** priority)  
{  
    **if** (priority < MAX\_TASKS) tasks[priority].state = SUSPEND;  
}
- **void** Scheduler\_ResumeTask( **u8** priority)  
{  
    **if** (priority < MAX\_TASKS)  
    {  
        tasks[priority].state = READY;  
        tasks[priority].count = 0; // tasks[priority].periodicity-1;  
    }  
}




# SCHEDULER APIs

```
➤ void Scheduler()  
{  
    counter --;  
    if(counter==0){  
        for (u8 i=0 ; i<MAX_TASKS ; i++) {  
            if (tasks[i].ptr != NULL && tasks[i].count ==0 && tasks[i].state == READY)  
            {  
                tasks[i].ptr ();  
                tasks[i].count = tasks[i].periodicity -1;  
            }  
            else{  
                tasks[i].count --;  
            }  
        }  
        counter = Num_entry_ISR;  
    }  
}
```

# SCHEDULER APIs

```
➤ ISR(TIMERO0 CTC_vec)
{
    Scheduler();
}
```

The background features a light gray gradient with a dark red border. Faint, stylized circuit lines in red and white are visible, particularly along the left and right edges, with small circles at the end of the lines.

# THANK YOU !

## AMIT