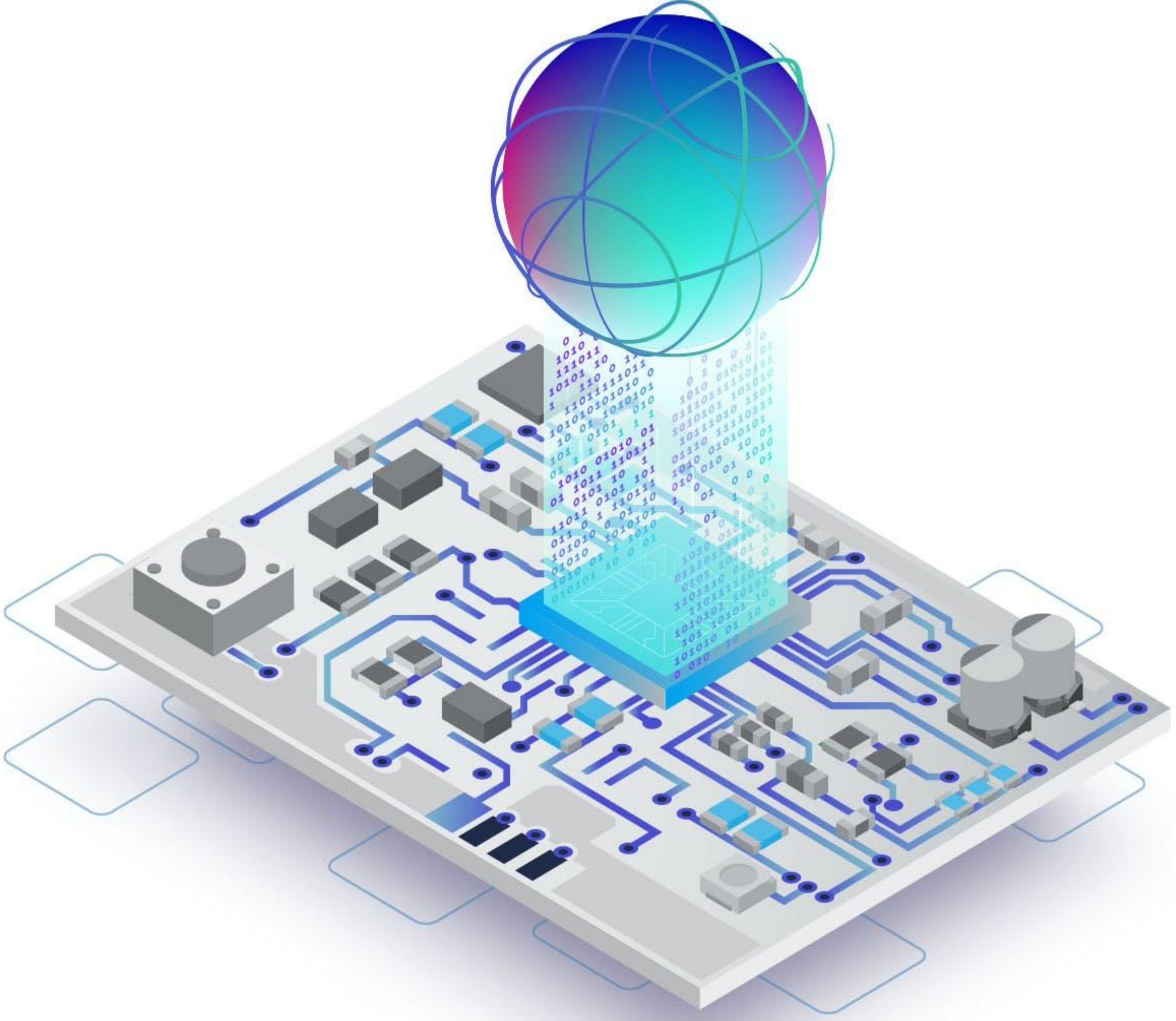
Real-time operating systems

Created & presented by:
Mohammadhossein Allahakbari

Shiraz University, department of Electrical and computer engineering



Outline

- Bare-Metal programming and it’s constraints
- Why use operating systems in embedded platforms?
- Real-time vs general purpose operating systems
- Deeper dive into RTOS concepts
- FreeRTOS examples

Prerequisites

- Having a basic familiarity with embedded systems
- Basic knowledge of operating systems concepts
- Intermediate acquaintance with C/C++ langs.

Bare-metal programming

Programming on hardware without any abstractions!

- No operating system is involved
- Interaction with the system is at a hardware level
- Hardware specifics should be considered

A bare-metal Arduino sketch

```
#include <Arduino.h>

void setup()
{
  // initialize LED digital pin as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop()
{
  // turn the LED on (HIGH is the voltage level)
  digitalWrite(LED_BUILTIN, HIGH);
  // wait for a second
  delay(1000);
  // turn the LED off by making the voltage LOW
  digitalWrite(LED_BUILTIN, LOW);
   // wait for a second
  delay(1000);
}
```

Figure0 – Arduino Blink sketch

A more sophisticated sketch

```
#include <Arduino.h>
void task1();
void task2();

void setup()
{
  // initialize stuff
}

void loop()
{
  task1();
  delay(1000);

  task2();
  delay(1000);
}

void extInterrupt(){
  //Respond to async keypress events
}

ISR(TIMER2_COMPA_vect){
  //Do some time critical stuff here
}

ISR(TIMER1_COMPA_vect){
  //Do some other time critical stuff here
}
```

Figure1 – Arduino sketch concerning interrupts

Bare-metal programming pros & cons

- Pros:
  - Faster code
  - Less memory and storage usage since no OS is involved
- Cons:
  - Hard to implement!
  - No scheduling system
  - Few capabilities for priority-based code execution

Why use OS in embedded platforms?

➢ Tasks of a typical flight controller

- Keeping the balance of the drone
- Responding to control signals
- Reading various sensor data
- Monitoring the drone’s health
- …

Figure 2 – DJI Matrice 300 RTK

General-purpose Operating Systems

- Uses a fairness policy to dispatch threads and processes
- The more threads that are running in a GPOS, the longer it will take to schedule and start executing a thread.
- They usually consume a high amount of hardware resources

Real-time Operating Systems

- Priority-based scheduling
- In contrast to GPOS, RTOS kernel objects can be selectively linked
- Number of tasks doesn’t determine the scheduling overhead

Real-time Operating implementations

- Popular Real-time operating systems:

  1. FreeRTOS
  2. Mbed
  3. Keil RTX

RTOS Concepts

- Task

  A real time application that uses an RTOS can be structured as a set of independent tasks. Each task executes within its own context with no coincidental dependency on other tasks within the system or the RTOS scheduler itself.

RTOS Concepts(cont'd)

- Multitasking

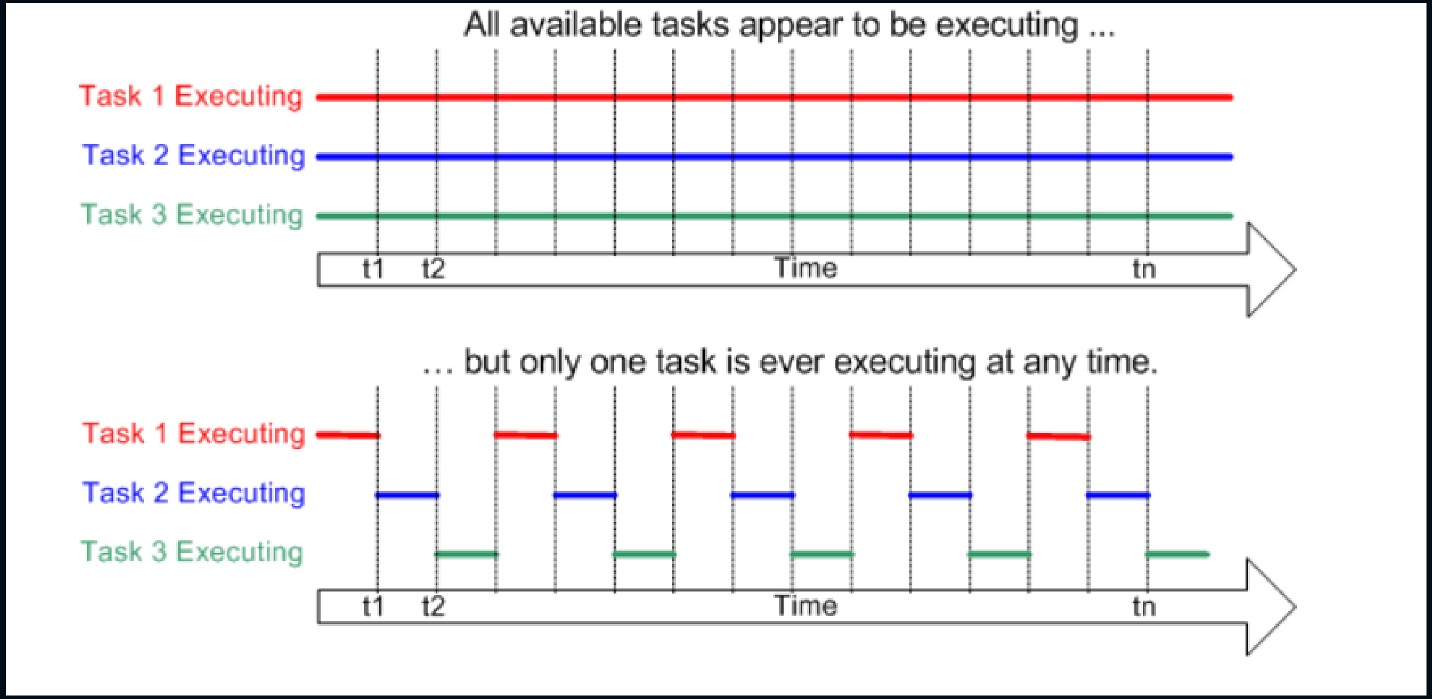


Figure 3 – Context Switching in an RTOS

RTOS Concepts(cont'd)

- Scheduling policy (specific to FreeRTOS)
  - Time Slicing Scheduling Policy for tasks with equal priority(round-robin)
  - Fixed priority preemptive scheduling for tasks with different priorities

RTOS Concepts(cont'd)

```
// And on to the things the same no matter the AVR type...
#define configUSE_PREEMPTION                1
#define configUSE_IDLE_HOOK                 1
#define configUSE_TICK_HOOK                 0
#define configCPU_CLOCK_HZ                  ( ( uint32_t ) F_CPU )
#define configMAX_PRIORITIES                4
#define configMINIMAL_STACK_SIZE            ( 192 )
#define configMAX_TASK_NAME_LEN             ( 8 )
#define configUSE_TRACE_FACILITY            0
#define configUSE_16_BIT_TICKS              1
#define configIDLE_SHOULD_YIELD             1

#define configUSE_MUTEXES                   1
#define configUSE_RECURSIVE_MUTEXES         1
#define configUSE_COUNTING_SEMAPHORES       1
#define configUSE_QUEUE_SETS                1
#define configQUEUE_REGISTRY_SIZE           1
#define configUSE_TIME_SLICING              1
#define configCHECK_FOR_STACK_OVERFLOW      1
#define configUSE_MALLOC_FAILED_HOOK        1
```

Figure 3 – Configuring scheduler parameters

RTOS Concepts(cont'd)

- Task synchronization
  - Mutex
  - Semaphore
    - Binary semaphore
    - Counting semaphore

RTOS Concepts(cont'd)

- Inter-task communication

  - Global variables (Not recommended at all!)
  - Queues

Thank you for your attendance!