HANDOUT 1

Embedded Systems

- microcomputer system embedded in larger system.
- computer system = processor + memory + I/O peripheral
- modern embedded system
  - + based on microcontrollers.
  - + microprocessors use in more complex systems. (External chips)
- reduce size and cost, increase reliability and performance
- specific tasks

Classification

- microprocessors OR microcontrollers : 8-bit, 16-bit, 32-bit or 64-bit.
- classify by based on COMPLEXITY and PERFORMANCE
  - + small-scale
    - └ simple task
    - └ no OS
  - + medium-scale
  - + large-scale

Embedded and OS

- general-purpose system = full-scale OS.
- Embedded system may or may not have OS.
- Many real-time embedded systems have real-time OS or real-time kernel.

Reactive Embedded

- Embedded systems are reactive systems in nature.
- Triggering
  - + Time-triggered
  - + Event-triggered

Real-time Embedded

- Many embedded systems are in reactive systems class.
  - + continuously react to inputs from environment.
  - + must RESPOND to inputs within short period of time.
  - + EXPECTED to execute all tasks in their deadlines.
  - + Real-time Embedded Systems = RTES
    - o must have timing constraints.
    - o must have deadline-driven.

NOT MEAN FASTER IS BETTER

Hard & Soft Real-time

- Subclasses of real-time constraints
  - + HARD RT
    - o HIGHLY critical time
    - o results in SYSTEM FAILURE
    - o important to SAFETY
  - + SOFT RT
    - o DESIRABLE in timing
    - o NOT CRITICAL but can loss.

Predictability and Determinism

- predictable in terms of all timing requirements by mathematics.
  - + system workload
  - + capability of processors = memories + cache + bus systems
  - + run-time OS support
  - + process and task priorities
  - + scheduling algorithm
  - + so on ...

Highly Constrained Environments

- RT embedded systems are often run in HIGHLY resource Contrained enviroments.
  - + process or speed + memory capacity + user interface

Concurrency in RT Systems

- several computations are executing
  - + simultaneously
  - + potentially
- multiple processors or events may occur at the same time

Critical Real time Systems

- split 2 categories:
  - + critical system : consequences of a failure in system is the worst case to loss.
  - + non-critical system : missing deadline wont harm much but useless.

Safety and Reliability

- safety means no accidents or no losses.
- reliability is ability or component to perform functions for specified time.
  - + often measured in FAILURES / million operating hours.
- embedded systems are EXPECTED to run continuously for years without ERRORS.
- Software and Hardware must DESIGN and DEVELOP with TESTING more carefully than general purpose computing systems.

Structures & Components

- The brain is CONTROLLER
  - + one or more microprocessors.
  - + memory
  - + some peripherals
  - + real-time software application
    - o run real-time tasks concurrently
    - o may or may not be with support RTOS
    - o Depend on complexity of Embedded system.
  - + processor must read, understand and manipulate the data which are the most of data is ANALOG SIGNALS.
  - + need ADC between a SENSOR and CONTROL
  - + controller acts to target system through ACTUATORS
    - o hydraulic, electric, thermal, magnetic or mechanid
    - o need DAC to apply digital to analog output for ACTUATOR.

Typical H/W Components

- processing units
  - + general-purpose or special-purpose
  - + special-purpose processors
    - o MCU or SoC (System on Chip), ASIC, ASIP, FPGA, DSP, Multi-core
  - + MCU versus MPU : different of MCU and MPU is software and development.
    - o MCU
      - → self-contained system peripherals
      - → memory
      - → CPU for specific tasks
    - o MPU
      - → support as
      - → related software
- memories
  - + On-chip or off-chip(external)
  - + volatile or non-volatile
  - + main memory or cache memory
  - + ROM (Read-only Memory)
    - o EEPROM
    - o Flash
  - + RAM (Random Access Memory)
    - o SRAM
    - o DRAM
- on-chip peripheral
  - + timer/counter
  - + watchdog timer
- I/O Interfaces
  - + GPIO, ADC, DAC, UART, I2C, SPI, ...
- sensors and actuators

HANDOUT 2

Super-loop Concept

- application executes each function in a fixed order.
- time-critical operations must be processed within an ISR.
- an interrupt is a special signal that triggers a change in execution.

Definitions

- Interrupt Service Routine (ISR) is a program function.
- Interrupt vector is a fixed address that contains the start address of ISR.
- Interrupt flag is one bit in a register that shows does interrupt has been triggered or not.
- Interrupt mask is one bit in a register that controls does interrupt can be triggers or not.
- Non-Maskable Interrupt is an interrupt that is always active.
- Asynchronous event is an event that can happen any time.
- Even-triggered interrupt is an interrupt that is triggered by a timer in period.
- Time-triggered interrupt is an interrupt that is triggered by asynchronous event.

Super-loop

- Super-loop is a program composed of an INFINITE LOOP with all tasks of system.
- each of functinal blocks is coded as a separate BLOCK OF CODE.

```
void main() {
    // initializations
    for(;;) {
        // different tasks
    }
}
```

- A complex time-consuming functional block
  - + can be split into smaller tasks
  - + can be handled by FINITE STATE MACHINE.

Event Detection

- Polling versus Interrupt-Driven
- Assume event

Lab Directions

- Install Microchip Studio software
- use microchip studio IDE to build C code for AVR chip.
- Exercise
  - + Write C code to detect a button-press event on an external-interrupt pin.
  - + wake up MCU from sleep mode to toggle LED in pin 6
  - + Do not use Arduino libs.

HANDOUT 3

Basic OS Concepts

- manage hardware resources and activities.
  - + scheduling application programs
  - + scheduling processes
  - + writing files to disk
  - + sending data across a network.
  - + so on ...
- employ KERNEL
  - + allow users access to the computer
  - + multiple users can execute multiple programs
- is a computer program
  - + software between software app and hardware.
- must execute several processes to maximum of CPU usage
- support multitasking, IPC, process synchronization and other system services.
- manage and allocate resources to processes
- process management
  - + process loading, process creation, execution control, process monitoring
  - + interaction of the process with signal or interrupt
  - + CPU allocation and process termination
- inter-process communication
  - + synchronization
  - + process protection
  - + deadlock and live-lock detection
  - + handling and data exchange mechanisms.
- memory and file management
  - + services of memory allocation
  - + file creation, deletion, reposition, protection
- I/O management handles request and release the subroutines

Tasks

- processes may consist of several concurrent activities
  - + concurrent activities can be split into smaller called TASKS.
    - o spliting can help to reduce the system complexity, errors and facilitate test.
    - o easier to understand and manage the small pieces of code than larger.

Multi Tasking

- OS can execute multiple tasks called multi-tasking
- share the CPU between tasks in system

RTOS

- an OS that manages
  - + hardware resources
  - + runs applications
  - + processes events or data on a real-time basis

RTOS

- a program that schedules task execution
- a combination of modules in RTOS
  - + real-time kernel
  - + file system
  - + networking protocol stacks
  - + other components required for a particular application
- built to be predictable and RESPONSIVE

Architectures of RTOS

- 2 main architectures of RTOSes
  - + Monolithic
    - o All system services are bundling together into kernel
    - o kernel runs all OS components
    - o device drivers, file management, networking and graphic stack
  - + Micro-kernel
    - o separate processes called servers
      - → some run in kernel
      - → some run in user
    - o communicate by passing messages

Real-time kernel

- minimal implementation of an RTOS
- core supervisory software privides
  - + minimal logic
  - + scheduling
  - + resource-management algorithms
- consist of task scheduler and a context switch handler

Task Management

- tasks are scheduled by the RTOS based on their priority
- task management and scheduling are parts of the CORE functions of an RTOS kernel.
- task switcher versus task scheduler
  - + task switcher is for switching from one task to another based on
    - → interrupt
    - → software driven
  - + task scheduler defines which task should be running based on PRIORITY.
    - → control the execution of tasks
    - → provide the algorithms to define what task should execute when.

Task Context switching

- While task is suspended, other tasks will execute and modify the CPU register values
- saving the context of a task being suspended
- restoring the context of a task being resumed

Components of RTOS

- middleware
  - + network Ethernet, wifi, TSL/SSL
  - + USB host and device
  - + bluetooth
  - + LoRaWAN
  - + File System
  - + 6 LOWPAN
  - + GUI
  - + Command line
  - + logging

Functions VS Tasks

- Functions
  - + a set of program instructions
  - + run sequentially
- Tasks
  - + is a function
  - + run in a specific "context"
  - + tasks are structured in one of 2 ways
    - o run to completion
    - o endless loop
  - + all tasks are not equally important
  - + can have different priorities

Task Priority Assignment

- most RT systems use a priority
- 2 classes of priorities
  - + static priority
    - o when a task is created and remains CONSTANT throughout execution
  - + dynamic priority
    - o when a task is created, but can be changed at any time during execution

Preemptivity

- action of switching to higher-priority
- transparently without having to wait for completion of lower-priority

Classification of Tasks

- periodic tasks are repeated once a period
- aperiodic tasks are one-shot and event-driven
- sporadic tasks are also event-driven

Temporal Parameters Of Tasks

- release time
  - + the time when a task becomes available for execution
- task deadline
  - + the instant of the time which its execution must be completed
- task execution time
  - + the amount of time that is required to complete the execution
  - + depends on the complexity of the task and speed of the processor
- response time
  - + the length of time passed from the task is released to the execution is completed
- period, phase, and utilization
  - + period is the time between the release times of 2 consecutive instances
  - + phase is the release time of its first instance
  - + utilization is the ratio of its execution time over its period

Context switching

- new task's context is restored from its storage area and then resumes execution new
- a context is the current execution state of a task
- each task has its own context
  - + the state of the CPU registers
- task context consists of
  - + program counter
  - + CPU registers
  - + stack pointer

Task Creation

- every task must have a Task Control Block (TCB)
  - + contains system information
- when a task is created, several attributes need to be specified
- every task is assigned a task name or task ID

Task States

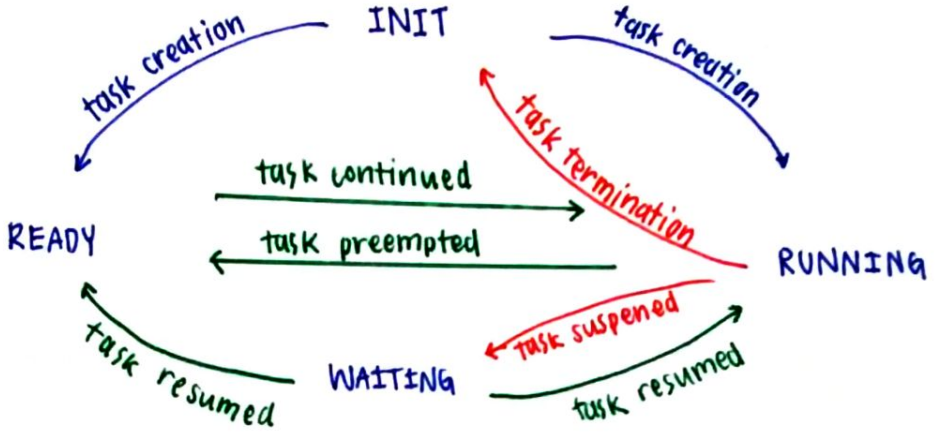
- a finite state machine (FSM)
  - + 3 main states
    - o ready state
    - o blocked state
    - o running state
  - + some RT kernels
    - o suspened
    - o pended
    - o delayed

→ READY state
- o task is ready to run but cannot run

→ BLOCKED or waiting state
- o task has requested a resource that is not AVAILABLE

→ RUNNING state
- o the task is the highest priority task and is running

Task states and transitions



Basic Scheduling Algorithms

- scheduler determines which task runs
  - + Preemptive priority-based scheduling
  - + Round-robin scheduling
  - + Cooperative scheduling

HANDOUT 4

RT Event Categories

- asynchronous events are entirely unpredictable
- synchronous events are preditable events

Fore / Back Ground Sys.

- an application consists of an infinite loop called TASKs (BACKGROUND)
- ISR are designed to handle asynchronous events (FOREGROUND)
- FOREGROUND is called the interrupt level
- BACKGROUND is called the task level

Pending Tasks

- a task waits for an event by calling one of functions that brings the task to the pending state if the event has not occurred

Interrupt Management

- interrupt is a hardware mechanism
  - + used to inform the CPU that an asynchronous event occurred
- when an interrupt is recognized, the CPU saves part of its context and JUMP to ISR.

Interrupt Handling

- if interrupt has occurred
  - + CPU suspends the execution of the current task
  - + then executes and ISR
- in real-time environment, interrupt must be disable as little as possible
  - + if disable the interrupt, it affects interrupt latency
- ISRs should be as short as possible
  - + most of work of handling the interrupting devices should be done at the task level

Classification of Interrupts

- maskable vs. non-maskable
- hardware vs. software
- nested-interrupt handling / priority-based preemption

Interrupt Controller

- captures all of the different interrupts presented to the processor
- the Interrupt devices signal to the interrupt controller, which
  - + priorities the interrupts
  - + presents the HIGHEST priority interrupt to CPU

Nested Vector Interrupt Control

- a method of prioritizing interrupts
  - + help improving the CPU performance
  - + reducing interrupt latency
- NVIC ensures that the higher priority interrupts are completed before lower
- NVIC uses a vector table that contains the addresses of ISR for each interrupt
- when an interrupt is triggered, the processor gets the address for the vector table

Idle Task

- an internal task
- runs when no other application task is able to run because none of events
- lowest priority task

Hook Functions

- also known as a callback
- CPU in low-power mode
  - + most processors exit low-power mode when an interrupt occurs
  - + depending on the processor, ISR may have to write to special register

Reentant Thread-Safe Functions

- Reentantcy and thread-safe are 2 separate concepts
- Code written for multi-threaded programs must be reentrant an thread-safe
- thread-safe functions
  - + if multiple thread can execute the same function at the same time safely without interfacing with each other
  - + protects shared resources from concurrent access by serializing the access using a lock (recursive mutex) or using atomic operations
  - + disable interrupt before critical section
  - + enable interrupt after critical section
- Reentrant functions
  - + can be interrupted while being executed thread and then safely resumed again
  - + does not hold static data
  - + does not use non-constant global variables
  - + must not call non-reentrant functions
  - + 3 main reasons: recursion, interruption and multi-threading
  - + the function should not use global data, return pointer

HANDOUT 5

FreeRTOS

- the FreeRTOS kernel was originally developed by Richard Barry around 2003
- created and maintained by Real Time Engineers Ltd. (UK)
- MIT open source license
  - + free, open-source
  - + portable
  - + FreeRTOS is one of the most popular RTOSes for small embedded systems
- is a combination of one of the supported compilers and processors architectures
- there are many add-on software products

FreeRTOS Portability

- has 3 files in the source directory
  - + task.c, queue.c, list.c
- additional files are required software timer, event group and co-routine function
  - + event_group.c, timer.c, croutine.c
- FreeRTOS needs compiler and architecture specific code called FreeRTOS port

FreeRTOS Scheduling modes

- in co-operative scheduling mode, context switch occurs only when
  - + a task is in a RUNNIG STATE enters to BLOCKED STATE
  - + calling by YEILDS ⇒ taskYIELD()
  - * tasks are never preempted

Free RTOS Memory Management

(kernel memory allocation)

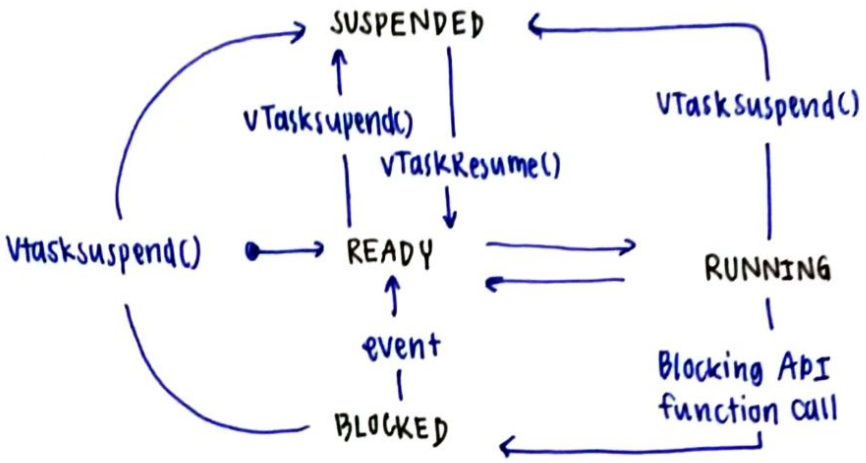
- needs RAM each time a task or other RTOS object is created
- the section that is allocated for a task or an object is called stack
  - + stack contains TASK FILE and TCB that allows the kernal to handle the stack
  - + all stacks are stored in a section called heap
  - + when applications need memory, they can allocate it from RTOS heap
- FreeRTOS offers up to 5 heaps management schemas in source/portable/memmang
  - + heap1 : simplest implementation, does not permit memory to be freed once allocated
  - + heap2 : use best-fit algorithm, allow allocated blocks to be freed
  - + heap3 : a simple wrapper for the standard C lib malloc() and free()
  - + heap4 : use first-fit algorithm, combines free memory blocks into a large block
  - + heap5 : similar to heap4, can span the heap

FreeRTOS Low Power Support

- FreeRTOS kernel uses a hardware timer to generate periodic tick interrupts which are used to measure time
- use idle task hook function to enter low-power state

FreeRTOS Debugging Support

- provides a mechanism for stack overflow detection
  + with specific prototype and name
  + the kernel calls the hook function if the stack pointer has a value outside valid
- TASK STATES



Synchronization Communication

- tasks may synchronize and communication among themselves
  + using KERNEL OBJECTS provided by real time kernel
  + SYNCHONIZATION : a task is waiting for a specific event from another task
  + COMMUNICATION : a task sends/receives data to/from another task

Semaphores

- or COUNTING
- is a protected variable used for restricting access to share resources
  + use for EVENT NOTIFICATION, INTER-TASK synchronization and mutual exclusion
- is an integer variable
  + never allowed to fall below zero
- 2 operations : wait/down and signal/up

Binary Semaphores

- is a SPECIAL CASE of counting semaphore
  + where count is restricted to the values 0 and 1
- do not support RECURSION

Mutexes

- is an object that acts like a token or gatekeeper
- provide mutual exclusion
- is a locking mechanism used to control the access of tasks to critical section or share resources
- only one task can own a given mutex at the same time

Critical Sections

- is a code segment which instructions must be executed in sequenc without interrupt
- no two threads can be in a critical section at the same time
- contains shared resources
- when the thread is ready to execute the thread code segment
  - + it first ATTEMPTS to ACQUIRE that MUTEX
  - + after the thread has aquire mutex, it executes code segment and the release

Mutex vs. Semaphore

- speed
  - + mutex is slower than a semaphore
  - + semaphore requires fewer system resources
- thread ownership
  - + mutex has only one thread can own
- priority inheritance
  - + mutex is available only
- inter-thread synchronization
  - + semaphore can be performed, but an event flags should be considered before
- event notification
  - + semaphore can be performed
- thread suspension
  - + mutex of by thread can suspend if another thread already owns the mutex
  - + semaphore by thead can suspend if the value of a counting semaphore is zero

HANDOUT 6

Learning Guidelines

- using Arduino IDE with FreeRTOS library for different MCU boards
  - + Advantages
    - ease of use, suitable for beginners
    - available of low-cost MCU boards
- migrating to Non-Arduino Software Development
  - + ESP32
  - + STM32
  - + SAM/SAMD21

Arduino FreeRTOS ports

- Richard Barry has ported FreeRTOS to Arduino boards
  - + AVR Mega board (Uno/Nano)
  - + Arduino Mega2560 boards
  - + FreeRTOS library (v.10.4.3)
- uses WDT timer to implement the system tick interrupt

Task states

- tasks are created and provided by FreeRTOS kernel
  - + when task is created, it is in READY state or use taskYIELD() or stop immediately
  - + if task schedwler choose task, task will change state to RUNNING state
  - + change to BLOCKED state when use vTaskDelay() or wait some conditions
  - + change to SUSPENDED when use vTaskSuspend()

Task Priority level

- an argument of xTaskCreate() = UBaseType_t uxPriority
- in generally, the priority in FreeRTOS is more than zero
  - + minimum value is zero, it is used by IDLE TASK (tskIDLE_PRIORITY = 0)
  - + maximum value is configMAX_PRIORITY − 1
- FreeRTOS's task is running in Preemptive Scheduling mode
  - + high priority of task and it is in ready state, this task will run before low priority
  - + in case of many tasks have the same priority, allocate CPU by Round-Robin

Arduino Tick Timer

- rising up or frequency of Tick Timer is set 62 Hz
  - + period is 16 ms (around)
  - + use WDT timer for OS timer