CSE4352/CSE5442 3.0 Real-Time Systems Practice

Lab Manual

Table of Contents

DISCLAIMER

Although all care has been taken to obtain correct information, we cannot be liable for any sort of incidental or consequential damages (including damages for loss of business, profits or the like) arising out of the use of the information provided here.

LABS

LAB 1 - HelloWorld on Dragon12 Development Board

Lab Objectives

The purpose of this lab is to familiarize you with the development process of applications using CodeWarrior IDE for the MC9S12DP256B microcontroller.

Background information

Embedded Systems

An embedded system, as opposed to a general purpose computer, is a special purpose system in which the computer is completely encapsulated by or dedicated to the device or system it controls. Unlike a general purpose computer, such as a personal computer, an embedded system performs one or a few predefined tasks, usually with very specific requirements. Since the system is dedicated to specific tasks, design engineers can optimize it, reducing the size and cost of the product. Embedded systems are often mass produced, benefiting from economies of scale.

Usually intermediate developers face the following issues while working on embedded platforms:

- Finding affordable hardware platform
- Finding easy to follow and organized documentation
- Finding open source and customizable compilers and assemblers
- Finding a simple and affordable Integrated Development Environment (IDE)
- Difficulties with testing, transferring and debugging code on target platform
- Finding simple and understandable examples for hardware modules
- Need of electrical knowledge and general understanding of target platform
- Need of high tech and expensive equipments (i.e. precision oscilloscopes and logic analyzers)
- Finding simple and affordable embedded operating system

Freescale Semiconductor Inc.

Freescale Semiconductor, Inc. is an semiconductor manufacturer created by the divestiture of the Semiconductor Products Sector of Motorola in 2004. Freescale focuses on the automotive, embedded and communications markets for their semiconductor products. Freescale is among the Worldwide Top 50 Semiconductor Sales Leaders.

The 68HC12 (6812 or HC12 for short) is a 16 bit micro-controller family from Freescale Semiconductor. Originally introduced in 1994, the architecture is an enhancement of the previous

Freescale 68HC11. Programs written for the HC11 are usually compatible with the HC12, which has a few extra instructions. The first 68HC12 derivatives had a maximum bus speed of 8 MHz and flash memory sizes up to 128 KBytes.

Like the 68HC11, the 68HC12 has two 8 bit accumulators A and B (sometimes referred to as a single 16 bit register D), also two 16bit registers X and Y, a 16 bit program counter, a 16bit stack pointer and an 8 bit Condition Code Register.

Motorola has launched the new HCS12 (also known as MC9S12) product line in 2000. The bus speed was improved up to 25MHz and flash sizes up to 512 KBytes. The MC9S12NE64 was introduced by Freescale in September 2004, claiming to be the "industry's first single-chip fast-Ethernet Flash micro-controller." It features a 25 MHz HCS12 CPU, 64 KBytes of FLASH EEPROM, 8 KBytes of RAM, and an Ethernet 10/100 Mbit/s controller. Since the HCS12 was a direct upgrade to the existing HC12 family, most of the links and information provided here are suitable for both lines.

Latest addition in 2004 was the advanced HCS12X, providing even more features, including the XGATE DMA co-processor. HCS12X is fully backward-compatible with HCS12 CPU. The S12X family utilizes the latest process technology. It boasts higher speed (40 MHz), more functionality, reduced power consumption and cost, and enhanced performance with the new XGATE on-chip memory-management and DMA module. The XGATE peripheral co-processor allows for some tasks to be offloaded from the CPU also allows several new instructions to increase performance.

Freescale announced the MC9S12XEP100 in May 2006 to further extend the S12X family to 50MHz bus speed and add a Memory protection unit (based on segmentation) and a hardware scheme to provide Emulated EEPROM.

Freescale HC1x platform has been around from 1990s which makes it affordable, well tested and stable candidates for embedded projects. Documentations can be downloaded free of charge from Freescale Semiconductor's website. GNU tool chain has been ported and tested on HC1x MCUs [1]. BDM features of HC12 series provide ease of transfer, testing and debugging codes to MCUs. We have decided to use HCS12 hardware as the host system for μC/OS.

Dragon12 Evaluation Board

Dragon12 is the name of the evaluation board made by evbplus.com. The board is using 9S12 family of Freescale 16-bit microcontrollers. Revision D of the Dragon12 board is using MC9S912DP256B processor with the following capabilities:

- CAN controller
- 16X2 LCD display module with LED backlight
    - it can be replaced by any size of LCD display module via a 16 pin cable assembly
- 4-digit, 7-segment display module
    - Can be used to learn multiplexing techniques
- 4 X 4 keypad

- Eight LEDs connected to port B
- An 8-position DIP switch connected to port H
- Four pushbutton switches
- IR transceiver with built-in 38KHz oscillator
- RS485 communication port
- speaker driven by timer, or PWM, or DAC for alarm, voice and music applications
- Potentiometer trimmer pot for analog input
- Dual SCIs with DB9 connectors
- Dual 10-bit DAC for testing SPI interface and generating analog waveforms
- I2C expansion port for interfacing external I2C devices
- Fast SPI expansion port for interfacing external SPI devices
- Abort switch for stopping program when program is hung in a dead loop
- MC9S12DP256 MCU includes the following on-chip peripherals:
    - 3 SPIs
    - 2 SCIs
    - 2 CANs
    - I2C interface
    - 8 16-bit timers
    - 8 8-bit PWMs or 4 16-bit PWMs
    - 16-channel 10-bit A/D converter
- Bus speed up to 25 MHz
- The 112-Pins on-board MCU (MC9S12DP256B) with 89 I/O-Pins is included
- BDM-in connector to be connected with a BDM from multiple vendors for debugging.
- BDM-out connector for making this board as a HCS12 / 9S12 BDM or programmer.
- PC board size 8.4" X 5.3"

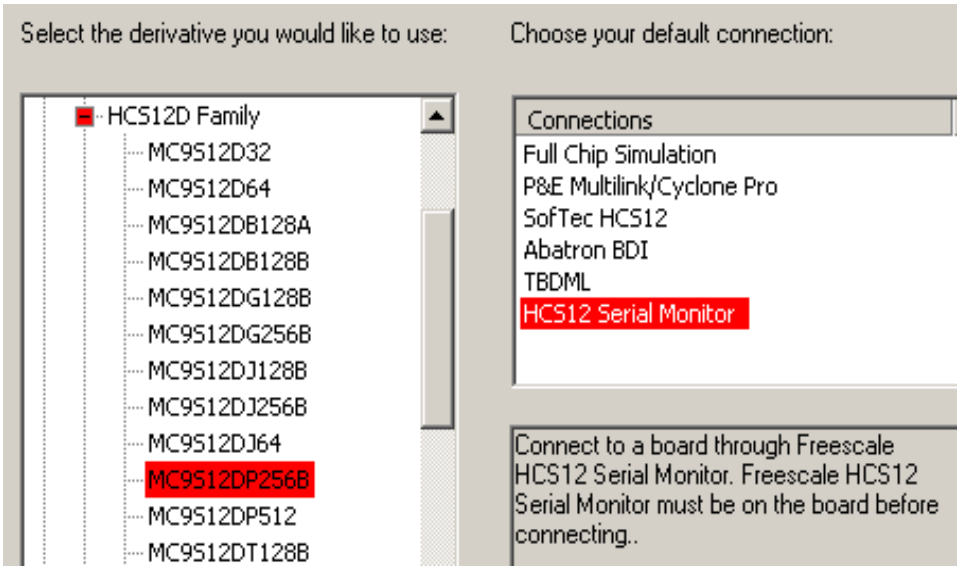
Evaluation

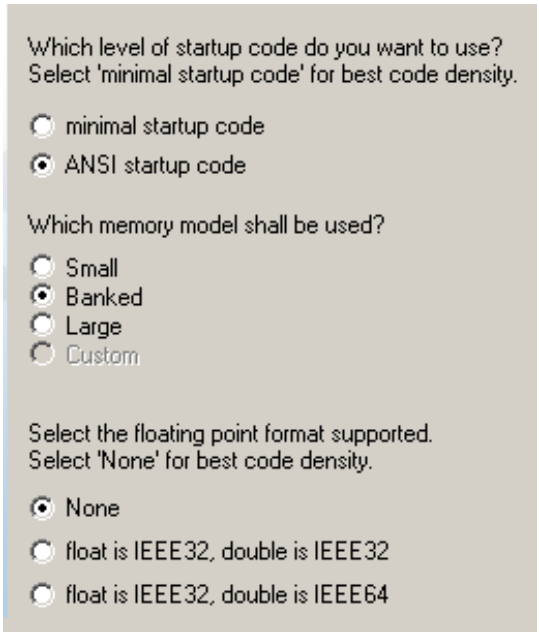
Demonstrate your program to your T.A.

Procedure

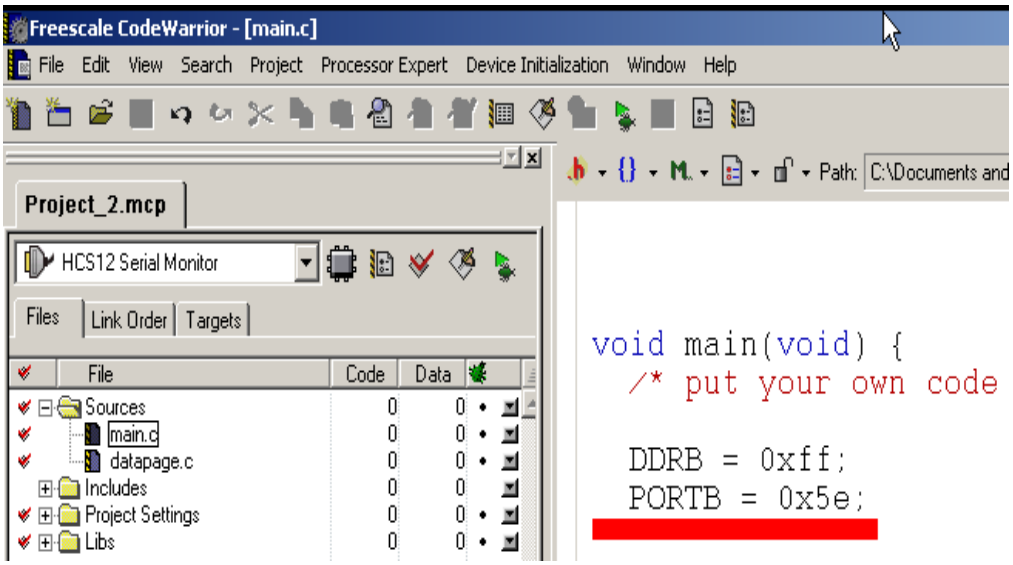
In this lab we are going to make a simple C program to turn on the LEDs on the 7-segment display module of Dragon12 board. The 7-segment display is connected to PORTB of microprocessor. We will use Data Direction Register for Port B (DDRB) to mark all the Port B's pins to output. This can be done by calling DDRB = 0xff

- Launch the Freescale CodeWarrior IDE. The shortcut may look like: 
- Start a new project and use the following settings:

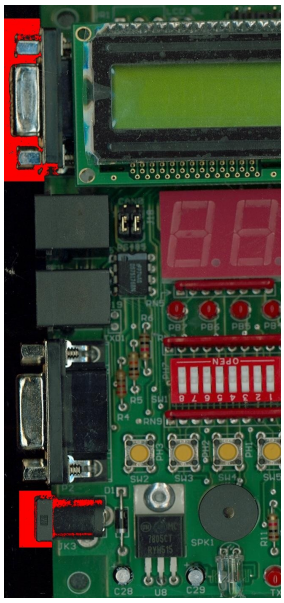




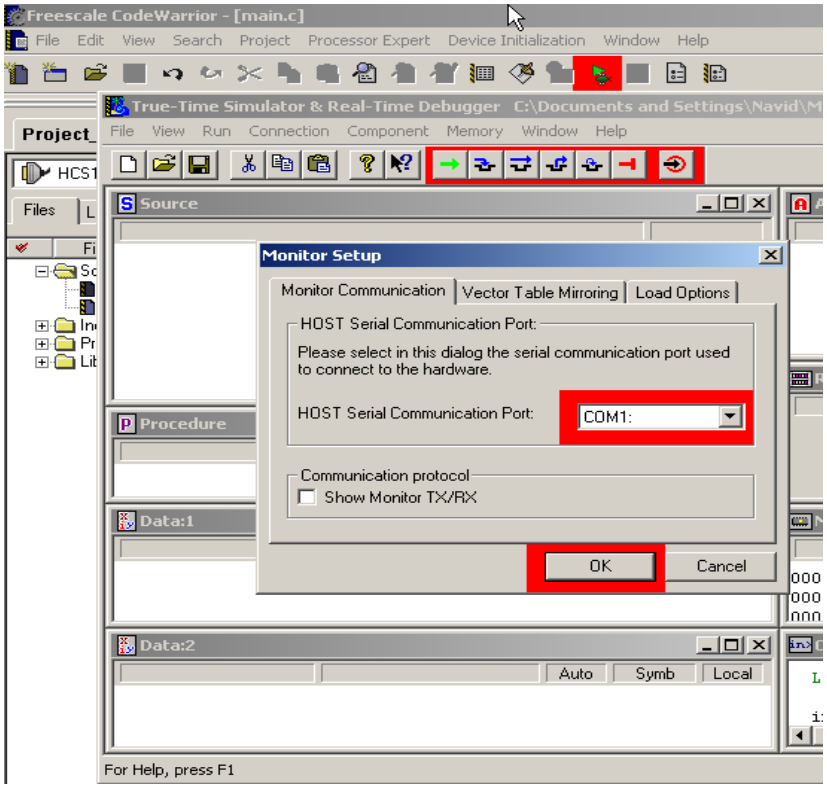
- We will use Data Direction Register for Port B (DDRB) to mark all the Port B's pins to output which can be done by calling DDRB = 0xff. On the main.c file add the followings:
  - DDRB=0xff;
  - PORTB=0x5e; // sample value just to turn some of the LEDs on



- To compile the project use: 
- To upload the compiled program to the board use: 
    - Make sure the board is powered up and connected to serial port on the PC (COM1). Please use only the top-left serial port as shown in the picture below.



    - If it is the first time you are running the the program you may need to setup the PC's serial port:



- Please demonstrate the program to your TA and make sure to understand how use simulation and debugging tools as shown in the above picture.

Reference Manuals

Please note all the drivers to run the subsystems of the Dragon12 board will be provided and you don't need to write any driver code. However, in case you want to look at the official manuals of MC9S12DG256B they can be found on-line or in the course directory:

http://gcc-hcs12.com/documnets/Freescale/motorolaindex.html

- MC9S12DP256 Overview
- MC9S12DP256 Device Users Guide (Electrical characteristics, Registers' Map ...)
- MC9S12DP256 Advance Information
- HCS12 Core Users Guide (CPU Core, BDM interface)
- HCS12 Reference Manual (CPU Instructions - this is the programmers' manual)

- ATD_10B8C Block Users Guide (Analog to Digital Converter)
- An Overview of the HCS12 ATD Module
- ADC Definitions and Specifications
- PWM_8B8C Block Users Guide (Pulse Width Modulator)
- MC9S12DP256 Port Integration Module Block Users Guide (External interface for ports A,B,E,H,J,K,M,P,S,T)

- HCS12 Serial Communications Interface Block Guide
- SPI Block Users Guide (Serial Peripheral Interface)
- HSC12 Inter-Integrated Circuit Block Guide (I2C)
- CRG Block Users Guide (Clock and Reset Generator)
- ECT_16B8C Block Users Guide (Enhanced Timer Capture)
- BDLC Block Guide
- MSCAN Block Guide (CAN interface)
- VREG Block Users Guide (Voltage Regulator)

- HCS12 Non-Volatile Memory (NVM) Guidelines
- EETS4K Block Users Guide (4K EEPROM)
- FTS256K Block Users Guide (256K Flash EEPROM)

- Migration from Assembly to C Language

- D-Bug12 Reference Guide
- Using The Callable Routines In D-Bug12
- Using and Extending D-Bug12 Routines

- Using Background Debug Mode for the M68HC12 Family

- A Serial Bootloader for Reprogramming

- MC34164 Undervoltage Sensor
- HIP7020 J1850 Bus Transceiver (for BDLC)
- PCA82C250 CAN Controller Interface
- BOSCH CAN Specification Version 2.0
- Precison Sine-Wave Tone Synthesis using 8-Bit MCUs
- Audio Reproduction on the HCS12 Microcontrollers
- HSC12 External Bus Design
- HSC12 External Bus Design, companion note to AN2287
- VPW J1850 Multiplexing Controller (BDLC) Module

- MC68HC812A4 Reference (older 68HC12 microcontroller)
- Transporting M68HC11 Code to M68HC12 Devices
- MC9S12DP256 Software Development Using Metrowerks Codewarrior

LAB 2 – Introduction to µC/OS-II

Lab Objectives

The purpose of this lab is to provide you with an introduction to µC/OS-II and its source code structure.

Background Information

µC/OS-II Overview

µC/OS-II (read as MicroC/OS-II) is the second generation of µC/OS which is a priority-based, preemptive and real-time multitasking operating system written mainly in the C programming language. It is originally published in a book by Jean J. Labrosse, µC/OS The Real-Time Kernel, which purpose was to describe the internals of a portable operating system with a small footprint. It is now a product which is maintained by Micrium Inc. and licenses are issued per product or royalty free for non-commercial educational uses. Even though the source code of µC/OS is available, it is not by any means considered free or open source software.

µC/OS-II is an extremely detailed and highly readable design study which is particularly useful to the embedded systems student. While documenting the design and implementation of the kernel, the book also walks through the many related development issues such as how to adapt the kernel for a new microprocessor, how to install the kernel, and how to structure the applications that run on the kernel.

µC/OS-II Important Features

Important features of µC/OS-II are [2]:

- Highly portable, scalable and preemptive real-time multitasking kernel that you only build what you need.
- It can manage a predefined maximum number of tasks .
- It can be expanded and connected to addons such as µC/GUI and µC/FS which are GUI and File Systems for µC/OS-II
- It supports all type of processors from 8-bit to 64-bit

µC/OS-II like most modern operating systems has the following components:

- Task Management (i.e. Create, Delete, Change Priority and Suspend/Resume tasks)
- Time and Timer Management
- Fixed Sized Memory Block management.
- Inter-Task Communication (i.e. Message Mailboxes and Message Queues)
- Semaphores, Mutual Exclusion Semaphores
- Many external modules are available as the real-time addons to the core (µC/GUI, µC/FS, µC/CAN, µC/USB, µC/TCP-IP and many more.).

µC/OS-II allows one to create new tasks and check the existing status of the tasks stack. Tasks can be deleted or their priority can be changed. Also µC/OS-II provides general information about a specific task and allows one to suspend or resume operation as well on a task.

The current version of µC/OS-II can manage up to 64 tasks. The four highest priority tasks and the four lowest priority tasks are reserved for the OS itself. The lower the value of the priority, the higher the priority of the task. The task priority number also serves as the task identifier

µC/OS-II uses rate preemptive monotonic scheduling such that the highest rate of execution is given to the highest priority task which is ready. Tasks are periodic and do not synchronize with one another and

µC/OS-II Frequently Used Functions

OSInit

OSInit function is used to initialize the internals of µC/OS-II and MUST be called prior to creating any µC/OS-II object and, prior to calling OSStart().

OSStart

OSStart function is used to start the multitasking process which lets µC/OS-II manages the task that you have created. Before you can call OSStart(), you MUST have called OSInit() and you MUST have created at least one task.

OSIntEnter

OSIntEnter function is used to notify µC/OS-II that you are about to service an interrupt service routine (ISR). This allows µC/OS-II to keep track of interrupt nesting and thus only perform rescheduling at the last nested ISR. You are allowed to nest interrupts up to 255 levels deep.

OSIntExit

OSIntExit function is used to notify µC/OS-II that you have completed servicing an ISR. When the last nested ISR has completed, µC/OS-II will call the scheduler to determine whether a new, high-priority task, is ready to run.

You MUST invoke OSIntEnter() and OSIntExit() in pairs. In other words, for every call to OSIntEnter() at the beginning of the ISR you MUST have a call to OSIntExit() at the end of the ISR.

Please note that rescheduling is prevented when the scheduler is locked (see OS_SchedLock())

OSTaskCreate

OSTaskCreate function is used to have µC/OS-II manage the execution of a task. Tasks can either be created prior to the start of multitasking or by a running task. A task cannot be created by an ISR.

Signature:

- `INT8U OSTaskCreate (void (*task)(void *p_arg), void *`**`p_arg`**`, OS_STK *`**`ptos`**`, INT8U `**`prio`**`)`

Arguments:

- Task argument is a pointer to the task's code
- P_arg argument is a pointer to an optional data area which can be used to pass parameters to the task when the task first executes. Where the task is concerned it thinks it was invoked and passed the argument 'p_arg' as follows:

```
void Task (void *p_arg)
{
      for (;;)
      {
       Task code;
      }
}
```

- Ptos argument is a pointer to the task's top of stack. If the configuration constant OS_STK_GROWTH is set to 1, the stack is assumed to grow downward (i.e. from high memory to low memory). 'pstk' will thus point to the highest (valid) memory location of the stack. If OS_STK_GROWTH is set to 0, 'pstk' will point to the lowest memory location of the stack and the stack will grow with increasing memory locations.

- Prio argument is the task's priority. A unique priority MUST be assigned to each task and the lower the number, the higher the priority.

Returns:

- OS_ERR_NONE: if the function was successful.
- OS_PRIO_EXIT: if the task priority already exist (each task MUST have a unique priority).
- OS_ERR_PRIO_INVALID: if the priority you specify is higher that the maximum allowed (i.e. >= OS_LOWEST_PRIO)
- OS_ERR_TASK_CREATE_ISR: if you tried to create a task from an ISR.

OSTaskDel

OSTaskDel function allows you to delete a task. The calling task can delete itself by its own priority number. The deleted task is returned to the dormant state and can be re-activated by creating the deleted task again.

Signiture:

```
INT8U  OSTaskDel (INT8U prio)
```

Argument:

- prio argument is the priority of the task to delete. Note that you can explicitely delete the current task without knowing its priority level by setting 'prio' to OS_PRIO_SELF.

Returns:

- OS_ERR_NONE: if the call is successful
- OS_ERR_TASK_DEL_IDLE: if you attempted to delete µC/OS-II's idle task
- OS_ERR_PRIO_INVALID: if the priority you specify is higher that the maximum allowed (i.e. >= OS_LOWEST_PRIO) or, you have not specified OS_PRIO_SELF.
- OS_ERR_TASK_DEL: if the task is assigned to a Mutex PIP.

- OS_ERR_TASK_NOT_EXIST: if the task you want to delete does not exist.
- OS_ERR_TASK_DEL_ISR : if you tried to delete a task from an ISR

Please note:

- To reduce interrupt latency, OSTaskDel() 'disables' the task:
  - by making it not ready
  - by removing it from any wait lists
  - by preventing OSTimeTick() from making the task ready to run.

  The task can then be 'unlinked' from the miscellaneous structures in μC/OS-II.

- The function OS_Dummy() is called after OS_EXIT_CRITICAL() because, on most processors, the next instruction following the enable interrupt instruction is ignored.

- An ISR cannot delete a task.

- The lock nesting counter is incremented because, for a brief instant, if the current task is being deleted, the current task would not be able to be rescheduled because it is removed from the ready list. Incrementing the nesting counter prevents another task from being scheduled. This means that an ISR would return to the current task which is being deleted. The rest of the deletion would thus be able to be completed.

OSTaskDelReq

OSTaskDelReq function is used to notify a task to delete itself and to see if a task requested that the current task delete itself. This function is a little tricky to understand. Basically, you have a task that needs to be deleted however, this task has resources that it has allocated (memory buffers, semaphores, mailboxes, queues etc.). The task cannot be deleted otherwise these resources would not be freed. The requesting task calls OSTaskDelReq() to indicate that the task needs to be deleted. Deleting of the task is however, deferred to the task to be deleted. For example, suppose that task #10 needs to be deleted. The requesting task example, task #5, would call OSTaskDelReq(10). When task #10 gets to execute, it calls this function by specifying OS_PRIO_SELF and monitors the returned value. If the return value is OS_ERR_TASK_DEL_REQ, another task requested a task delete. Task #10 would look like this:

```
void Task(void *p_arg)
{
      .
      .
      .

      while (1)
      {
            OSTimeDly(1);
            if (OSTaskDelReq(OS_PRIO_SELF) == OS_ERR_TASK_DEL_REQ)
            {
                  Release any owned resources;
                  De-allocate any dynamic memory;
                  OSTaskDel(OS_PRIO_SELF);
            }
      }
}
```

Argument:

- Prio argument is the priority of the task to request the delete from

Returns :

- OS_ERR_NONE: if the task exist and the request has been registered
- OS_ERR_TASK_NOT_EXIST: if the task has been deleted. This allows the caller to know whether the request has been executed.
- OS_ERR_TASK_DEL: if the task is assigned to a Mutex.
- OS_ERR_TASK_DEL_IDLE: if you requested to delete µC/OS-II's idle task
- OS_ERR_PRIO_INVALID: if the priority you specify is higher that the maximum allowed (i.e. >= OS_LOWEST_PRIO) or, you have not specified OS_PRIO_SELF.
- OS_ERR_TASK_DEL_REQ: if a task (possibly another task) requested that the running task be deleted.

OSTaskSuspend

OSTaskSuspend function is called to suspend a task. The task can be the calling task if the priority passed to OSTaskSuspend() is the priority of the calling task or OS_PRIO_SELF.

You should use this function with great care. If you suspend a task that is waiting for an event (i.e. a message, a semaphore, a queue ...) you will prevent this task from running when the event arrives.

Argument:

- Prio arguments is the priority of the task to suspend. If you specify OS_PRIO_SELF, the calling task will suspend itself and rescheduling will occur.

Returns:

- OS_ERR_NONE: if the requested task is suspended
- OS_ERR_TASK_SUSPEND_IDLE: if you attempted to suspend the idle task which is not allowed.
- OS_ERR_PRIO_INVALID: if the priority you specify is higher that the maximum allowed (i.e. >= OS_LOWEST_PRIO) or, you have not specified OS_PRIO_SELF.
- OS_ERR_TASK_SUSPEND_PRIO: if the task to suspend does not exist
- OS_ERR_TASK_NOT_EXITS: if the task is assigned to a Mutex PIP

OSTaskResume

OSTaskResume function is called to resume a previously suspended task. This is the only call that will remove an explicit task suspension.

Returns:

- OS_ERR_NONE: if the requested task is resumed
- OS_ERR_PRIO_INVALID: if the priority you specify is higher that the maximum allowed (i.e. >= OS_LOWEST_PRIO)

- OS_ERR_TASK_RESUME_PRIO: if the task to resume does not exist
- OS_ERR_TASK_NOT_EXIST: if the task is assigned to a Mutex PIP
- OS_ERR_TASK_NOT_SUSPENDED: if the task to resume has not been suspended

OSTaskChangePrio

OSTaskChangePrio function allows you to change the priority of a task dynamically. Note that the new priority MUST be available.

Returns:

- OS_ERR_NONE: is the call was successful
- OS_ERR_PRIO_INVALID: if the priority you specify is higher that the maximum allowed (i.e. >= OS_LOWEST_PRIO)
- OS_ERR_PRIO_EXIST: if the new priority already exist.
- OS_ERR_PRIO: there is no task with the specified OLD priority (i.e. the OLD task does not exist.
- OS_ERR_TASK_NOT_EXIST: if the task is assigned to a Mutex PIP.

OSSchedLock

OSSchedLock function is used to prevent rescheduling to take place. This allows your application to prevent context switches until you are ready to permit context switching. You MUST invoke OSSchedLock() and OSSchedUnlock() in pair. In other words, for every call to OSSchedLock() you MUST have a call to OSSchedUnlock().

OSSchedUnlock

OSSchedUnlock function is used to re-allow rescheduling. You MUST invoke OSSchedLock() and OSSchedUnlock() in pair. In other words, for every call to OSSchedLock() you MUST have a call to OSSchedUnlock().

OSTimeDly

OSTimeDly function is called to delay execution of the currently running task until the specified number of system ticks expires. This, of course, directly equates to delaying the current task for some time to expire. No delay will result If the specified delay is 0. If the specified delay is greater than 0 then, a context switch will result.

Argument "ticks" is the time delay that the task will be suspended in number of clock 'ticks'. Note that by specifying 0, the task will not be delayed.

OSTimeDlyHMSM

OSTimeDlyHMSM function is called to delay execution of the currently running task until some time expires. This call allows you to specify the delay time in HOURS, MINUTES, SECONDS and MILLISECONDS instead of ticks.

OS_ENTER_CRITICAL

OS_ENTER_CRITICAL() is a macro inserts the machine instruction into your code to block all interrupts .

OS_EXIT_CRITICAL

OS_EXIT_CRITICAL() is a macro inserts the machine instruction to enable interrupts.

OSSemCreate

OSSemCreate function creates a semaphore.

Argument "cnt" is the initial value for the semaphore. If the value is 0, no resource is available (or no event has occurred). You initialize the semaphore to a non-zero value to specify how many resources are available (e.g. if you have 10 resources, you would initialize the semaphore to 10).

Return will be (void *)0 if no event control blocks were available. Otherwise return is a pointer to the event control block (OS_EVENT) associated with the created semaphore

OSSemPost

OSSemPost function signals a semaphore Argument pevent is a pointer to the event control block associated with the desired semaphore.

Returns:

- OS_ERR_NONE: The call was successful and the semaphore was signaled.
- OS_ERR_SEM_OVF: If the semaphore count exceeded its limit. In other words, you have signalled the semaphore more often than you waited on it with either OSSemAccept() or OSSemPend().
- OS_ERR_EVENT_TYPE: If you didn't pass a pointer to a semaphore
- OS_ERR_PEVENT_NULL: If 'pevent' is a NULL pointer.

OSSemPendAbort

OSSemPendAbort function aborts & readies any tasks currently waiting on a semaphore. This function should be used to fault-abort the wait on the semaphore, rather than to normally signal the semaphore via OSSemPost().

Prelab studies

- Please make sure to read and understand the license agreements located at
  - `LAB02/Micrium/ReadMe/LegalNotice/`

- Also read the following study guides and application notes:
  - `LAB02/Micrium/ReadMe/AN1004 (The 10-Minute Guide to RTOS).pdf`
  - `LAB02/Micrium/ReadMe/uCOS-II-RefMan.pdf (to be used as reference)`
  - `LAB02/Micrium/ReadMe/QuickRefChart.pdf`
  - `LAB02/Micrium/ReadMe/Task-State-Diagram.pdf`
  - `LAB02/Micrium/ReadMe/AN1002 (Mutual Exclusion Semaphores).pdf`
  - `LAB02/Micrium/ReadMe/AN1005 (Inter-Process Communication) .pdf`
  - `LAB02/Micrium/ReadMe/AN1007A (µC OS-II and Event Flags) .pdf`

Evaluation

µC/OS-II source code is divided into platform dependent and platform independent files. Platform independent files can be found in `Micrium/Software/uCOS-II/Source/`. Platform independent files can be found in `Micrium/Software/uCOS-II/Ports/HCS12/Paged/Metrowerks/SerialMonitor/`.

By study the source structure and `Micrium/ReadMe/uCOS-II-RefMan.pdf` briefly answer the following questions:

1. Write a pseudo code using µC/OS-II to create two tasks and protect their critical sections that is accessing a common variable called XYZ.
2. In your own words, explain how multitasking is achieved in the µC/OS-II (pay special attention to `os_cpu_a.s`)?

Reference Manuals

The following application notes can be found in `LAB02/Micrium/ReadMe/`:

- AN1004 (The 10-Minute Guide to RTOS)
- uCOS-II-RefMan
- QuickRefChart
- Task-State-Diagram
- AN1002 (Mutual Exclusion Semaphores)
- AN1005 (Inter-Process Communication)
- AN1007A (µC OS-II and Event Flags)
- uCOS-II-CfgMan

LAB 3 - Compile and Boot µC/OS-II on Dragon12 Development Board

Lab Objectives

The purpose of this lab is to demonstrate the compilation, upload and boot process of the MC9S12DP256 port of the µC/OS-II using CodeWarrior compiler.

Prelab studies

- Please make sure to read and understand the license agreements located at
  - `LAB03/Micrium/ReadMe/LegalNotice/`
- Make sure you fully understand the previous labs (lab1 and lab2)
- Also read the application note 1456 which provides a general information about lab3 stationary project.
  - `LAB03/Micrium/ReadMe/AN-1456 (µC OS-II Dragon12 Development Board).pdf`

Evaluation

Demonstrate the uploaded program to your T.A. Provide brief answers to the following questions:

- What is the process of creating a new task in µC/OS-II? Provide a pseudo sample code by study "`KeypadTask`" code block used in the lab.
- Explain how `OSFlagPend` function and how it prevents the "`KeypadTask`" from accessing the critical LCD resource?

Procedure

- Open the lab stationary using CodeWarrior located in:

  `LAB03/Micrium/Software/EvalBoards/Freescale/MC9S12DG256B/Wytec Dragon12/Metrowerks/Paged/OS-Probe-LCD/OS-Probe-LCD.mcp`

- Recompile the project and transfer the binaries to Dragon12 board as explained in lab1. 

Reference Manuals

The following application notes can be found in `LAB03/Micrium/ReadMe/`:

- **AN1004 (The 10-Minute Guide to RTOS)**
- **AN1456 (µC OS-II Dragon12 Development Board)**
- **uCOS-II-RefMan**
- **QuickRefChart**
- Task-State-Diagram

- AN1002 (Mutual Exclusion Semaphores)
- AN1005 (Inter-Process Communication)
- AN1007A (μC OS-II and Event Flags)
- AN2000 (C Coding Standard)
- uCOS-II-CfgMan
- TaskAssignmentWorksheet
- Dragon12 LCD-Manual
- WhatsNewSince-V200
- ReleaseNotes

LAB 4 – Adding customized tasks to µC/OS-II

Lab Objectives

The purpose of this lab is to write additional customized tasks to the µC/OS-II.

Background

Servo Motor

A Servo [3] is a device that has an output shaft that can be positioned to a specific angular positions by sending the servo a coded signal. As long as the coded signal exists on the input line, the servo will maintain the angular position of the shaft. As the coded signal changes, the angular position of the shaft changes. Servos are topically have 3 wires; one is for power (+5 volts), ground, and the yellow wire is the control wire.

The servo motor has a control circuit and a potentiometer (a variable resistor also known as pot) that is connected to the output shaft. The pot allows the control circuitry to monitor the current angle of the servo motor. If the shaft is at the correct angle, then the motor shuts off. If the circuit finds that the angle is not correct, it will turn the motor the correct direction until the angle is correct. The output shaft of the servo is capable of traveling somewhere around 180 degrees. A normal servo is mechanically not capable of turning any farther due to a mechanical stop built on to the main output gear.

Infrared Range Finder

Infrared Range Finder [4] works by the process of triangulation. A pulse of infrared light is emitted and then reflected back (or not reflected at all). When the light returns it comes back at an angle that is dependent on the distance of the reflecting object. The infrared range finder receiver has a special precision lens that transmits the reflected light onto an enclosed linear CCD array based on the triangulation angle. The CCD array then determines the angle and causes the rangefinder to then give a corresponding analog value to be read by external microcontroller.

Prelab studies

- Please make sure to read and understand the license agreements located at
  - `LAB04/Micrium/ReadMe/LegalNotice/`
- Make sure you fully understand the previous labs (lab1, lab2 and lab3)
- Also read the manuals for the parts used in this lab:
  - `LAB04/Micrium/ReadMe/SHARP GP2D12 IR Range Finder Manual.pdf`
  - `LAB04/Micrium/ReadMe/Hitec HS422 Servo Motor Manual.pdf`

Evaluation

Demonstrate the uploaded program to your T.A. Make sure you can trace the `ir_range` function.

Procedure

The goal is to have a task that every 300 ms checks the distance of the IR Rangefinder and change the servo's position accordingly.

- Open the lab stationary using CodeWarrior located in:

  ```
  LAB04/Micrium/Software/EvalBoards/Freescale/MC9S12DG256B/Wytec
  Dragon12/Metrowerks/Paged/OS-Probe-LCD/OS-Probe-LCD.mcp
  ```

- Add a task that every 300 ms calls `ir_range(ATD0DR0)`

- Connect the VCC and ground wires of Servo and IR Range Finder. Make sure you have +5 volts and have a common ground between the microprocessor and your power supply.

- Connect PP7 pin to servo's input signal and PAD04 pin to R Range Finder output signal.

- Double check your circuitry with your TA before powering up anything.

- Recompile the project and transfer the binaries to Dragon12 board. 

Reference Manuals

The following application notes can be found in `LAB04/Micrium/ReadMe/`:

- **SHARP GP2D12 IR Range Finder Manual**
- **Hitec HS422 Servo Motor Manual**
- **AN1004 (The 10-Minute Guide to RTOS)**
- **AN1456 (µC OS-II Dragon12 Development Board)**
- **uCOS-II-RefMan**
- **QuickRefChart**
- Task-State-Diagram
- AN1002 (Mutual Exclusion Semaphores)
- AN1005 (Inter-Process Communication)
- AN1007A (µC OS-II and Event Flags)
- AN2000 (C Coding Standard)
- uCOS-II-CfgMan
- TaskAssignmentWorksheet
- Dragon12 LCD-Manual
- WhatsNewSince-V200
- ReleaseNotes

References

[1] http://gcc-hcs12.com

[2] Based on a presentation by Enric Pastor (http://studies.ac.upc.edu/EPSC/SED/Apuntes/uCOS-II.pdf)

[3] Based on http://www.seattlerobotics.org/guide/servos.html

[4] Based on http://www.societyofrobots.com/sensors_sharpirrange.shtml