Embedded Systems Design Project Report

Abhishek Milind Tambe
Instrumentation And Applied Physics
Indian Institute of Science, Bangalore
abhishekmt@iisc.ac.in

Munipalle Vara Sai Charan
Instrumentation And Applied Physics
Indian Institute of Science, Bangalore
charansaim@iisc.ac.in

***Abstract*—The objective of the project is to create a simple Real Time Operating System (RTOS) for the Cortex-M4 microprocessor by implementing a task scheduler and Operating System APIs to make ISR-OS interface. The system should be capable of handling multiple tasks with varying priorities and should switch the tasks as that of in RTOS by round-robin algorithm according to their priorities, accommodating other system Interrupt Service Routines (ISRs). The goal is to minimize the overhead on processor with efficient system performance.**

I. Introduction

A mini RTOS is designed to handle multiple tasks with different priorities and system Interrupt Service Routines (ISRs) by algorithms such as round-robin algorithm. One important aspect of an RTOS is its context switching, which allows the system to save the current running task state in stack and then switch to a higher-priority task when it is available. Tasks running in RTOS system will be appear as shown in fig 1. Task priorities are used to determine the order in which tasks are executed. The OS calls are implemented using SVC call, PendSV and SysTick interrupts. The SVC and PendSV interrupt is used to perform context switching while the SysTick interrupt is used to run the scheduler, ensuring that the highest priority task is always executed first. The goal of this mini RTOS is to minimize overhead on the processor by efficient task scheduling.

A. Architecture

This section describes the architecture of this project. The preemptive task scheduler discussed here is implemented in ARM Cortex-M microcontroller, specifically the TM4C123GH6PM. It uses the SVC, PendSV exception handler to perform context switching between tasks. When a context switch is required, the current task's context is saved, and the context of the next task to be executed is restored. The pendsv function should trigger a PendSV exception and request a context switch.

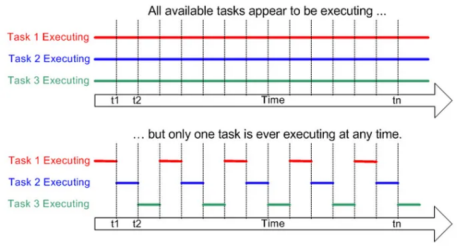

Fig. 1. Tasks scheduled by preemptive scheduler

In addition to PendSv, It should consist of SysTick exception handler to periodically select the next task to be executed based on its priority. we need to have a function for Systick Handler to update the delay values of all waiting tasks and moves tasks that have completed their delay to the ready state. If necessary, it need to set up a context switch by calling the pendsv function.

We need to set up a function to launch the Scheduler to first initialize the scheduler by dummy values stored in the stack and then once it is set up we need to get the first task to be executed according to the priority. The scheduler uses a simple round-robin approach to break ties between tasks with the same

priority. If no task is ready to run, the scheduler is set to run the current task only.

B. header file structure

The header file contains structures and functions related for managing tasks. This uses to run the tasks using a priority-based scheduling algorithm. The various structures that were used in this file are given below.

• **Task State** - This is used to define the state of a task. It has two values - ready and block. These values are used to represent the current state of a task.
• **TCBt** - This is a structure which represents a task control block (TCB). The TCB contains information about a task such as its stack pointer, priority, delay, task IDs and the address of the task function. This information is used by the OS to manage the task.
• **PendSv setup** - This is used to set up the PendSV exception. This exception is used by the OS to perform a context switch.
• **PendSv Handler** - This function is the exception handler for the PendSV exception. It performs a context switch by saving the current task's context and restoring the next task's context that needs to be executed.
• **schedule** - This function triggers the PendSv exception, which is used for context switching.
• **init systick timer** - This function is to initialize the systick timer by enabling the register. It is used by the OS to keep track of time and perform time-based operations.
• **Systick Handler** - This function is the exception handler for the Systick timer interrupt. It is used by the OS to keep track of time and perform time-based operations.
• **LaunchScheduler** - This function launches the scheduler, which is responsible for launching the tasks according to their priority.
• **create task** - This function creates a new task with the specified priority and task handler function.
• **delete task**- This function deletes the task with the specified priority and task handler.
• **init sched stack** - This function is used to store the kernel related registers on stack.
•**save psp value** - This function saves the psp value of the current task.
•**get psp value** - This function gets the psp value of current task.
•**update next task**- This function updates the next task that needs to be switched.
•**switch sp psp**- This function switches sp from MSP to PSP for the user tasks.
•**schedule**- This function calls the calls the pendSv interrupt for context switching.
•**task delay**- This function is used to block the task for specified delay and then moves it to blocking state until the specified delay is completed.
•**unblock tasks**- This function is used to unblock the task once the specified delay has been completed and moves it to ready state to schedule it.

• MAX NUM OF TASKS and STACKSIZE are pre processor macros that define the maximum number of tasks that can be created and the size of each task stack respectively.
• The task state defines a set of states that a task can be in: ready and block.

• **SysTick Handler** - The timer interrupt handler, responsible for updating the delay of waiting tasks and finding the next ready task with the next highest priority. When a higher-priority task becomes ready, it will trigger the PendSV interrupt with the schedule call function.

• **pendsv Handler** - The context switch interrupt handler is responsible for saving the context of the currently running task (registers R4-R11 and the stack pointer) onto the stack of its TCB, and then restoring the context of the next task from its TCB. It also updates the state of the TCBs and the running task ID. Finally, it enables interrupts and returns to the interrupted code with the special instructions like BX LR instruction. The function does not need to save or restore the registers R0-R3, PC,XPSR and LR (which are saved automatically by the Cortex-M core), and

that the function has no prologue or epilogue code (which is generated by the compiler by default).

C. source file structure

- The source file contains the functions whose prototypes are declared in the header files.

• **init tasks stack** - This is implemented in schedule.c source file. this function is used to initialize the dummy values in the stack for starting the scheduler.

• **unblock tasks** - This is implemented in schedule.c source file. this function is used to unblock the tasks. the unblocking is done by reducing the block count for each systick interrupt and once the block count reaches to 0 and it is in blocked state then it moves the task to ready state.

• **task delay** - This is implemented in schedule.c source file. this function is used to block the tasks. the blocking is done by taking the block count from the argument of task delay.

• **update next task** - This is implemented in schedule.c source file. this function is used to update the next task. It is done by comparing other tasks priority with all other tasks and takes the next highest priority and will schedule that task.

• **create task** - This is implemented in schedule.c source file. this function is used to create task and will assign the priority as given. It will create task based on the arguments passed to it, one is function pointer to task handler and other is priority. It will create task with the corresponding task handler and priority.

• **delete task** - This is implemented in schedule.c source file. this function is used to delete task based on function pointer to task handler. It will remove the task from the user task array.

D. Interrrupt services

- The section contains the interrupt services used in the project and the function prototypes were declared in the header file.

• **pendsv Handler** - This is implemented in schedule.c source file. this function is used for context switching whenever pendsv interrupt is triggered.

• **schedule** - This is implemented in schedule.c source file. this function is used to trigger pendsv by enabling it.

• **systick handler** - This is implemented in systick.c source file. this function is used for handling systick interrupt. the interrupt occurs for every 1ms. In this function we reduce the block count for each time interrupt occurs and calls scheduler to trigger the pendsv.

• **init systick timer** - This is implemented in systick.c source file. this function is used for intializing the systick interrupt. we load the value of no of ticks for which the interrupt needs to raise.