**Design and development of Real-time multitasking Microkernel for ARM9TDMI**

**ABSTRACT**

Real-time systems are those systems whose response is deterministic in time. In computer science, a **real-time microkernel** is the near-minimum amount of software that can provide the mechanisms needed to implement a real-time operating system. These mechanisms include low-level address space management, thread management, and inter-process communication (I.P.C). As an operating system design approach, microkernels permit typical operating system services, such as device drivers, protocol stacks, file systems and user interface code, to run in user space. If the hardware provides multiple rings or CPU modes, the microkernel is the only software executing at the most privileged level (generally referred to as supervisor).

In this project a 32-task Real Time Microkernel is designed using which multi-tasking can be done on the targeted processor ARM7TDMI from ARM limited. The micro kernel includes a preemptive priority scheduler and context switching modules for carrying out multi-tasking. Routines to create and manage tasks will be developed. Once created, the tasks will be scheduled by our own scheduler automatically. Subsequently, intertask communication mechanism is added to this scheduler, to make it a small real-time kernel.

Two sets of functions are developed in this project. First one is OS functions and second is application functions. OS functions are mainly for carrying out task creation, multi-tasking and Inter task communication. The number of application functions can vary between 1 to 32. Each of these application functions is created as a task by the microkernel and scheduled by the pre-emptive priority scheduler. Multi tasking of these application tasks will be demonstrated in this project. The function prototypes of some of the OS functions are shown below.

**osrc\_t thread\_create (int 32\*ptid, int32 pri, void (\*pfunc)(void))**

Creates a thread with the entry point pointed to by pfunc, and makes it ready to run. If the priority is higher than the current thread’s priority, it schedules the created thread. Returns OK if successful, or ERR on error.

**void thread\_suspend (int32 tid)**

Suspends the specified task (puts it in the wait state).

**void thread\_resume (int32 tid)**

Resumes the specified task (puts it in the ready/running state depending on priority).

**int32 thread\_self (void)**

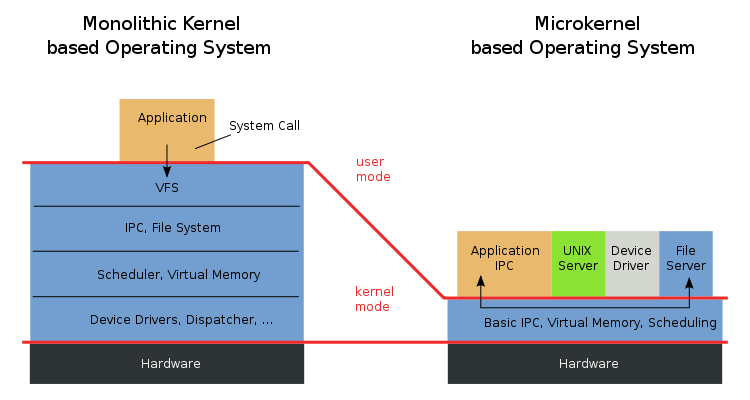
Returns the id of the current task.

**int32 thread\_getpri (int32 tid)**

Returns the priority of the specified task.

**myrc\_t thread\_setpri (int32 tid, int32 pri)**

Sets the priority of the specified task. Additionally, performs scheduling if required due to the change in the task priority.



**MicroKernel Structure Diagram**

**Tools and hardware:**

Simulation tool : **Keil Micro Vision**

Embedded board : **Micro controller based on ARM9**

Flash loading : **Flash Magic**