Project #3

EGR 424 - The Design of Microcontroller Applications

Instructor: Dr. Chirag Parikh

Renzo Garza Motta

December 6th, 2022

Objectives

The objective of this project is to develop the following skills:

- Reading and interpreting technical documentation and sample code
- Writing well-structured, low-level device drivers
- Development and debugging of interrupt-driven code on an actual hardware system

Project Requirements

- Implement and document a working kernel for the MSP432
- The kernel is to support multiple concurrent threads of execution
- The kernel is to be pre-emptive such that each thread may have its execution suspended at any time.
- The kernel is to pre-empt each thread at 2ms intervals using the SysTick timer peripheral.
- A thread may give up its 2ms "time slice" at any time by calling a `yield()` function.

Summary

This project aims to implement a simple kernel that allows the initialization and handling of a simple operating system, required onboard peripherals, and GPIO. This operating system consists of three threads that the operating system cycles through at 2ms intervals. This means that regardless of the task in the thread, this thread can only keep the microcontroller's resources for a total length of time of 2ms before control is taken from it and passed to the next thread.

To properly set up the operating system, three slices of memory were allocated to act as the core registers for each of the threads. Given that the core registers of the Cortex M4 are occupied by the "task at hand," these memory slices serve as placeholders for each thread to store the value of the core registers before passing control to the following thread. This allows each thread to recall where in its individual task it left off before control was taken from it (or was willingly yielded).

This project contains three threads (Thread0, Thread1, and Thread2). Thread0 is in charge of blinking the Red on-board LED whenever the thread is entered. Thread1 and Thread2 both control the onboard UART module to print to the PC's Terminal Window.

Thread0 uses bit banding to toggle this LED. This bit banding approach consists of using the CortexM4 internal memory which maps to specific pin locations. When a particular value is written to such memory locations, the corresponding pin can be driven to whatever value is desired. With the particular registers determined, a simple value assignment is needed to address that bit as desired.

Alternatively, since Thread1 and Thread2 both use UART, the problem arises when both threads want to use the UART module. If Thread1 is in the middle of printing a message to the console, and its timeslice expires, Thread2 then gains control and starts printing its message. This problem can be solved using locks.

A lock was implemented using a variable that holds the value 1 when the lock is unlocked, and a value of 0 when the lock is locked. This allows for each of the Threads to retain control of the UART module even after its timeslice is up temporarily. Say Thread1 has the UART module locked, Thread2 is unable to do anything with it until Thread1 regains control and finishes its task. Only when Thread1 has finished its task and unlocked the lock can Thread2 print to the UART module.

The control for these slices is handled by the Systick Timer. The Systick timer is set up to trigger an interrupt every 2ms. When the interrupt is triggered, the interrupt service routine (ISR) , Sytick_Handler(), is called which then takes a snapshot of the current Thread's registers by storing copies of these in their respective memory slices and giving control to the next thread in line.

Extra Credit - Context Switch Measurement

Context switch is the process of storing and restoring the state (i.e., the context) of a CPU or a thread so that execution can be resumed from the same point at a later time. However, this switch takes time and cannot be discounted when high efficiency is sought.

This context switch measurement was achieved by setting a GPIO pin high when the Systick Handler is entered, and set low when the handler is about to exit. Given that this section of the program was written in assembly, bit banding was used. In a similar fashion to Thread0's bit banding, the handler's bit banding was achieved by setting the value at a specific address in memory corresponding to P2.2. The oscilloscope was connected to the P2.2 and measured as shown in Figure 1 and Figure 2. The context switch was measured to be 14.6μs.

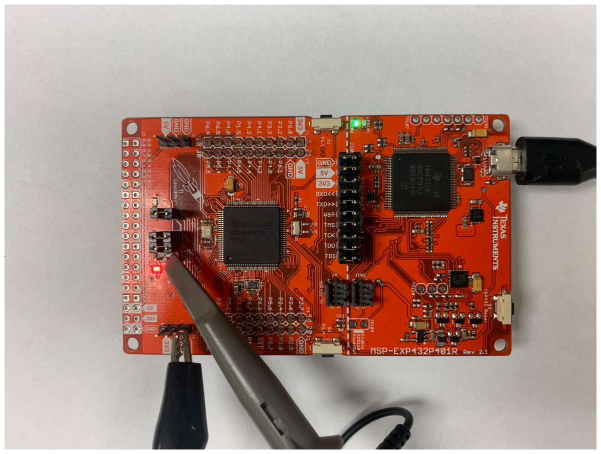

Figure 1. Context Switch Measurement Setup

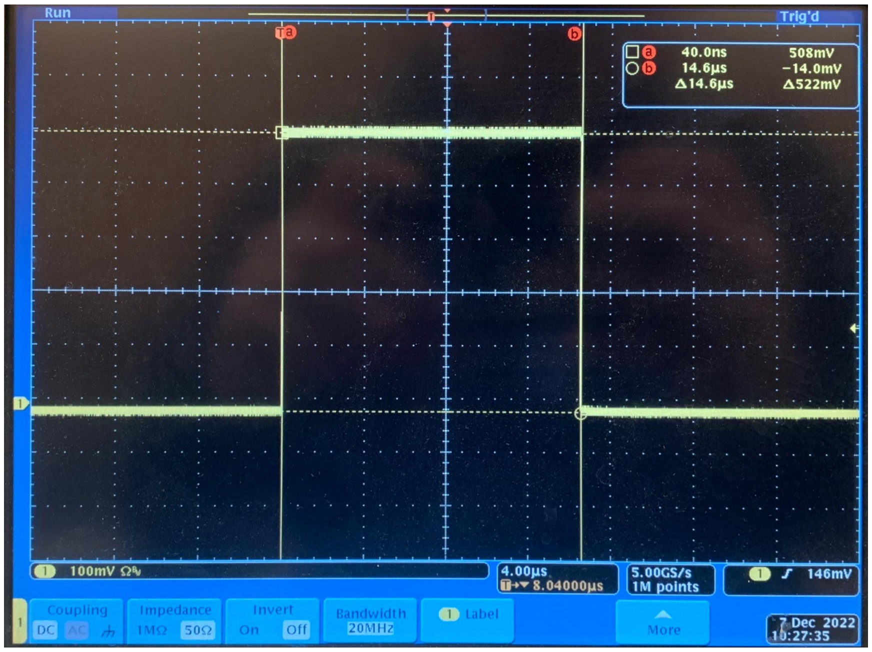

Figure 2. Context Switch Measurement Oscilloscope Capture