Part 2: Timers and Interrupts

Code Overview

I start by displaying the initial stream of characters which are the elements in the *CURR_ROTATION* byte array on the HEX displays as well as the LEDs. Then, setup is required to enable interrupts on the private timer and the pushbuttons. Finally, we end up in the main body of the program, *IDLE*, where the pushbuttons interrupt flag, private timer interrupt flag, and switches are polled. Switches enable characters to change within the stream. For example, turning switch SW0 on will flip the character at position 0 inside *CURR_ROTATION*.

Performance Analysis

The subroutine responsible for servicing interrupts from the timer and the pushbuttons, *SERVICE_IRQ*, spans only 18 lines excluding comments. It calls 3 subroutines, *ARM_TIM_clear_INT_ASM, ARM_TIM_IS*, and *KEY_ISR,* that are 6, 5, and 8 lines respectively. This brings the total code size to 37 lines. Effectively, running *SERVICE_IRQ* once executes 33 instructions with slightly more memory accesses since we manipulate the stack. Given its small size, the total time spent within it is expected to be virtually insignificant. The switches are polled continuously in *IDLE*. Considering that the subroutine responsible for polling the switches, *check_switches*, is 27 lines long and executes 3 (pre-loop) + 20 * 10 (loop body) + 2 (last iteration of loop) + 1 (post-loop) = 206 instructions, the time spent within this subroutine is expected to be significant given the frequency at which we loop through *IDLE*. Finally, the user code (everything else in IDLE apart from *check_switches*) takes up the other half of the *IDLE* block at 28 lines. It polls the interrupt flags and executes several subroutines if any of the flags is raised. Therefore, the code size of the user code is a lot larger than the 28 lines contained in the *IDLE* block. The number of instructions executed, and memory accesses are expected to be far greater than 28 as well, though the frequency at which the interrupt subroutines are ran depends on how often interrupt flags are raised. Therefore, we can establish that there is a proportional relationship between time spent servicing interrupts and frequency of the interrupts as well as time spent in the user code and the frequency of the interrupts.

To showcase this, I have provided data on instructions executed and memory accesses for *SERVICE_IRQ*, *check_switches*, and *IDLE* (excluding *check_switches*). The table below contains the mentioned data and showcases the trend that the time spent servicing interrupts (i.e. in *SERVICE_IRQ* and subroutines in *IDLE*) increase as the number of interrupts increases. This can effectively be observed through varying the frequency of the private timer (lower frequency = more interrupts = more time spent servicing interrupts). The lower the load value and prescaler bits are, the more frequent interrupts will be and vice versa. My approach to gathering data was to insert a breakpoint at the first instruction of *SERVICE_IRQ* to find how many instructions are executed between the *_start* segment and the first timer interrupt. Then, I subtracted the number of instructions in the *_start* block to find how many of them can be attributed to polling the switches and user code respectively by using proportions. Lastly, I counted the number of instructions executed in the *SERVICE_IRQ* and user code associated to handling interrupts and factored them into the data as well. Thus, the data provided can give a good idea on the amount of time spent in each block.

Effective Frequency	*SERVICE_IRQ* (interrupts)		*check_switches* (polling switches)		*IDLE* (user code)	
	# Instructions Executed	# Memory Accesses	# Instructions Executed	# Memory Accesses	# Instructions Executed	# Memory Accesses
20 MHz	24	54	~ 129,000	~ 175,000	~ 5,000	~ 7,000
50 MHz	24	54	~ 642,000	~ 871,000	~ 25,000	~ 34,000
200 MHz	24	54	~ 3,148,000	~ 4,271,000	~ 122,000	~ 166,000

** Results were gathered by restarting each simulation and clearing the configuration bits, load value, and prescaler bits from the timer in between attempts.*

** The "~" signifies an approximate value. Even with the chosen methods, it was hard to get a consistent value. Therefore, an average of 5 attempts was taken for each result present in the table.*

From the data in the table above, we can deduce that per constant frame time, frequency determines how much time is spent servicing interrupts. Though the time spent within *SERVICE_IRQ* is very trivial compared to the scale of instructions executed and memory accesses of *check_switches* and *IDLE*, we can still say that as the effective frequency decreases, the more time will be spent servicing interrupts since the number of times an interrupt will be raised by the timer $T = 1 / f$ which dictates an inverse relation to the effective frequency. In sum, we can say that we spend the most amount of time polling the switches followed by executing user code and lastly servicing interrupts.