ECE 354: Project Part 1

Timer

The first timer (timer0) was implemented using interrupts to generate a simple clock. The timer was set to fire an interrupt approximately every 10 ms. Two counters were used, one to count up to 100 to represent 1 full second and another to contain the total number of seconds elapsed.

In the main run loop, a local variable was used to store the last count of seconds sent to the UART1. A check is performed on each loop to detect if the current count is not equal to the last one. If the count has changed, then update the out string and prepare to output it to the UART. A rudimentary string library was implemented to handle converting the integers into strings. A integer was used both as a flag to indicate that the device needs to output a new string as well as to keep an index to the next character to print.

Printing was performed using an interrupt. A global variable, out_char, was used as both a flag and character storage. It was used to indicate in the main run loop that the UART is ready to accept the next character. The UART interrupt sets the value of out_char to '\0' to indicate readiness. If it is ready, the value is updated to the next character to output by the main run loop.

```
void c_timer_handler(void) {
    timer_count++;
    if (timer_count == 100) {
        counter++;
        reset timer_count;
    }
    acknowledge the interrupt
}

int main (void) {
    initialize local and global variables
    setup timer ISR vector
    setup UART ISR vector
    setup timer registers

    while(true) {
        if (counter value has changed) {
            convert counter value into hours, minutes, and seconds
            compile a string in the format: "hh:mm:ss"
            set flag for outputting the string
            save the current counter value
        }
        if (output flag is high
            AND the UART is ready
            AND there is a character to print) {
            set the output character to the next character to print
```

```
            call an interrupt on UART1
            increment index counter
        }
    }
    return 0;
}
```

Memory Management

The memory pool was implemented using a data structure known as a free list. A free list is a singly linked list used to coordinate unallocated memory blocks. Instead of storing a separate next point the first four bytes of each memory block in the list are used to store the address of the next free memory block. The only storage required is a pointer to the head of the list. The reason this data structure was chosen over a singly linked list is that the runtime for all the operations of the free list is $O(1)$ as opposed to $O(n)$.

Additionally, a field of thirty-two bits is used to track the status of each block, 0 for allocated and 1 for deallocated. This was implemented to prevent against double deallocation as well as allocating against blocks that are not currently free. The advantage of this is that it can be done in $O(1)$ time instead having to traverse the entire free list.

The pseudocode associated to the memory.c file is as follows:

```
declare start of free memory
declare start of free list
declare memory allocation field

int get_block_index(void* addr) {
    return (addr - start of free memory) / BLOCK_SIZE
}

void init_memory() {
    declare local variables

    set the head of free list = start of free memory + 31 blocks
    iterate over blocks in free list{
        store location of next free memory block in current memory block
    }

    set memory allocation field = 0
}

void* s_request_memory_block() {
    declare local variables

    if (the head of the free list not null) {
```

```
        pop a block of memory off the free list
        set corresponding bit in memory allocation field to 1
        return the address of the memory block
    }

    return NULL
}

int s_release_memory_block(void* memory_block) {
    declare local variables

    if (the memory block has been allocated) {
        push the block onto the free list
        clear the corresponding bit in the memory allocation field
        return success
    }

    return failure

}
```