- Interrupt configuration is done in init_interrupts(). This is not done in init_gpio() to make marking easier

- Blinking is done using timer 0. It is a periodic timer that raises an interrupt when it times out. The ISR is timer0_handler()

- SW1 and SW2 use GPIO interrupts to detect button presses and timers 1 and 2 for debouncing. Since all of port F only has one timer, the ISR port_f_handler() checks which switch is pressed and takes action accordingly. SW1 and SW2 are configured to interrupt on negative edges

- Debouncing of the switches is done by ignoring all input for a small period of time after the first negative edge is detected. When SW1 raises an interrupt port_f_handler() disables interrupts for SW1 and starts timer 1. When timer 1 times out, timer1_handler() re-enables the SW1 interrupt. The same is done for SW2

- Timer 3 is used for the seven segment display counter. It is a periodic timer that raises an interrupt when it times out. timer3_handler() handler updates the count on the displays if counting is enabled

- time_square() is used to answer 2.b. It uses timer 4. It is commented out in main() but can be uncommented to check the results. If stack and heap sizes aren't properly set in the build configuration, the printf() inside time_square() will cause a hard fault. Stack of 2048 and heap of 1024 have been confirmed to work.

2.a.

i. The 74AC373 latch can source or sink +- 50 mA per pin at max

ii. This current does not come from the microcontroller or the latch. It comes from the 5V rail connected to each display.
The microcontroller just provides the data

iii. Each segment is connected to the latch through a 200 Ohm resistor
+5V input - 2.6V drop across an LED segment = 2.4V across the resistor
Current across resistor = 2.4 / 200 = 12 mA
Current through all 8 segments is 12 * 8 = 96 mA

2.b. square takes 33 clock cycles to run. There is a variation of a few clock cycles between runs. The profiling is done in the time_square() function. This function is called before any other configuration is done on the microcontroller so other interrupts don't affect the results.

2.c. The FaultISR() is the hard fault handler. It is called when a hard fault occurs. Hard faults are described in Table 2-11 on page 112 of the datasheet. These include things like an undefined instruction and divide by 0