Connor Cimowsky
20427800

Below is the definition of the `init_timer` function I used for configuring the hardware timer. Since its frequency is 25 MHz, I scaled it by 25 000 in order to increment the `TC` register at a rate of 1000 Hz. This means that the value in `MR0` corresponds to the desired period of timer interrupts, in milliseconds. When the bottom 2 bits of `MCR` are set to 1, an interrupt will be generated when the value in the `TC` register reaches the value stored in `MR0`, and then the `TC` register will be reset. To start the timer, I set the `TCR`'s reset and enable bits accordingly in `start_timer`.

```
void init_timer(int period_ms)
{
    LPC_TIM0->PR = 25000;
    LPC_TIM0->MR0 = period_ms;
    LPC_TIM0->MCR = 0x03;
}
```

To enable interrupts for the `INT0` button, I enabled general-purpose I/O for P2.10 and subsequently configured it for input. Then, I set the appropriate bit of `IO2IntEnF` to 1 so that interrupts would be fired on the falling edge of the `INT0` signal. Finally, I enabled interrupts for `EINT3` on the interrupt controller.

```
void init_int0(void)
{
    LPC_PINCON->PINSEL4 &= ~(3 << 20);
    LPC_GPIO2->FIODIR   &= ~(1 << 10);

    LPC_GPIOINT->IO2IntEnF |= (1 << 10);
    NVIC_EnableIRQ(EINT3_IRQn);
}
```

To configure the LEDs, I set the appropriate GPIO direction bits for output and cleared the output pins.

```
void init_led(void)
{
    LPC_GPIO1->FIODIR |= 0xb0000000;
    LPC_GPIO2->FIODIR |= 0x0000007c;

    LPC_GPIO1->FIOCLR |= ~(0x0);
    LPC_GPIO2->FIOCLR |= ~(0x0);
}
```

To implement strict interrupt scheduling, I disabled `EINT3` interrupts and fired a one-shot timer whenever the `INT0` button was pressed:

```
void EINT3_IRQHandler(void)
{
    LPC_GPIOINT->IO2IntClr |= (1 << 10);

    NVIC_DisableIRQ(EINT3_IRQn);

    flash_message();
    start_timer();
}
```

When the timer interrupt is received after 5 seconds, I re-enable `EINT3` interrupts and disable timer interrupts:

```
void TIMER0_IRQHandler(void)
{
    LPC_TIM0->IR = 0x01; // clear interrupt

    NVIC_DisableIRQ(TIMER0_IRQn);
    NVIC_EnableIRQ(EINT3_IRQn);
}
```

For the bursty scheduler, I followed similar steps, except I only disabled `EINT3` interrupts when a counter variable reached a value of 3. Additionally, I used an asynchronous timer to periodically reset this counter and re-enable `EINT3` interrupts every 10 seconds.