SPI Bitbang

Tyler Petty

CS 566 Laboratory 4 Report

1 Objective

The purpose of this experiment was to introduce the use of a Serial Peripheral Interface Expander chip by implementing a bit banging various addresses and setting up an interrupt when a button is pressed.

2 Apparatus

Experimentation used the Texas Instruments Tiva C Series EK-TM4C123GXL Launchpad, MCP23S17 I/O Expander, breadboard, and an assortment of leds, 1kΩ resisors.

3 Method

For setup the MCP23S17 and Tiva board were wired up on a breadboard following the pin layout described in table 1. In addition, some additional LEDs were wired in series as shown in figure 1. After the hardware was set up, a copy of lab 3 was used, with features stripped out except for _configureUART(), _heartbeat(), main() and _setupHardware().

Using this as a base, code was implemented to allow the Tiva board to both read and write to the MCP23S17, with LEDs tied to the SO and SI pins of the MCP23S17 to visually confirm that the chip was recieving and sending messages. Having confirmed that the Tiva board was able to read/write to the SPI, expanderInit() was implemented to configure the IOCON, IODIR, GPPU, IPOL and GPINTEN registers as required. This function can be referenced in both the simplified version in figure 2 as well as the full version in the Appendix.

Once the SPI was configured and the ability to read/write to it was confirmed, a LED was wired from pin GPA0 of the MCP23S17 and was configured to blink along side the Tiva board's internal LED, as controlled by _heartbeat(). Finally, pin INTB of the MCP23S17 was wired to PD0 on the Tiva board as well as to a LED as shown in 1. This interrupt pin was configured in expanderInit() 2 to be tied to the input of GB0, which was connected to a button. This allowed the MCP23S17 to send out an interrupt when the Tiva Board which after a short delay, clears the interrupt by reading the value of GPB, as controlled by both _interruptHandlerPortD() as well as _interrupt_polling() found in figures 4 and 5.

4 Data

SPI	Tiva Board
RESET	RESET
CS	PA2
SCK	PA4
SI	PA5
SO	PE0
INTB	PD0
GPA0	External LED
GPB0	External Button

Table 1: Pin layout

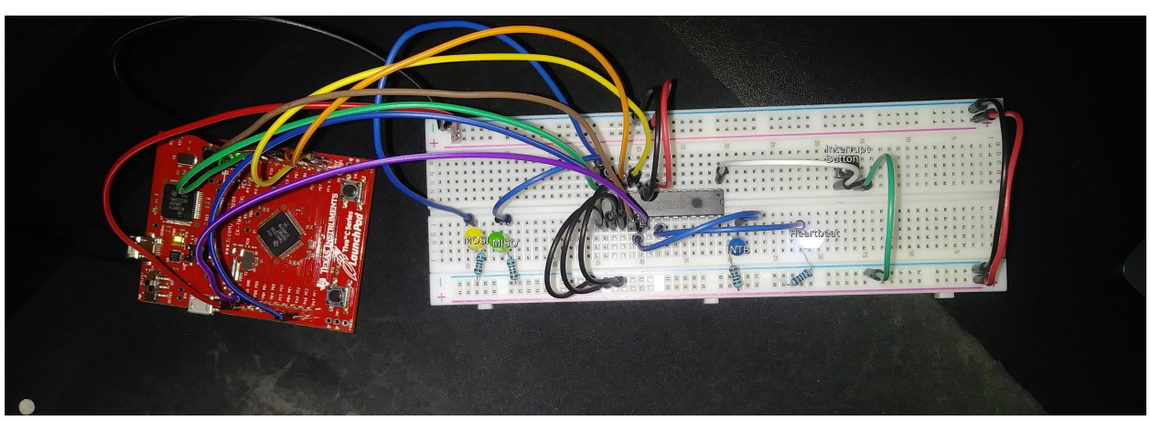

Figure 1: Breadboard Layout

5 Results and Analysis

It proved to be more difficult to work with the MCP23S17 than was expected, mostly due to an unfamiliarity to the device as well as external factors like the interrupt button being toggled from what might be external electrical interference or faulty wiring. Despite these complications, most of the key objectives of the lab were met. Code was implemented to enable read/write capabilities to the Tiva board, which was used to configure the registers of the MCP23S17 to set pins GPA0 and GPB0 as outputs, and tie GPB0 to the

```
void
expanderInit(void)
{
    uint8_t out = 0;
    //Configure IOCONA
    CS_HIGH;
    expanderWriteByte(0x0A, 0xA2);

    //Configure IODIRA
    CS_HIGH;
    expanderWriteByte(0x00, 0xfe);

    //Configure GPPUA
    CS_HIGH;
    expanderWriteByte(0x06, 0x01);

    out = 0;
    //Configure IOCONB
    CS_HIGH;
    expanderWriteByte(0x15, 0xA2);

    //Configure IODIRB
    CS_HIGH;
    expanderWriteByte(0x10, 0xff);

    //Configure IPOLB
    CS_HIGH;
    expanderWriteByte(0x11, 0x01);

    //Configure GPPUB
    CS_HIGH;
    expanderWriteByte(0x16, 0x01);

    //Configure GPINTENB
    CS_HIGH;
    expanderWriteByte(0x12, 0x01);

    //Configure INTCONB
    CS_HIGH;
    expanderWriteByte(0x14, 0x01);
}
```

Figure 2: Configuring the SPI

```
static void
_heartbeat( void *notUsed )
{
   uint32_t green500ms = 500; // 1 second for on off
   uint32_t ledOn = 1;

    while(true)
    {
        vTaskDelay(green500ms / portTICK_RATE_MS);
        uint8_t out = 0x0;
        LED(LED_G, ledOn);
        LED_REMOTE(ledOn);
        ledOn = !ledOn;
        #ifdef USB_SERIAL_OUTPUT
          UARTprintf("on:%i\n",ledOn);
        #endif
        CS_HIGH;
        out = expanderReadByte(0x19);
        #ifdef USB_SERIAL_OUTPUT
          UARTprintf("0x19:%x\n",out & (1));
        #endif
    }
}
```

Figure 3: Code responsible for the blinking of Tiva Board led as well as external led

```
static void
_interruptHandlerPortD(void)
{
    //
    // We have not woken a task at the start of the ISR.
    //
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;

    uint32_t mask = GPIOIntStatus(GPIO_PORTD_BASE, 1);
    #ifdef USB_SERIAL_OUTPUT
      UARTprintf ("mask: %x\n", (mask) & 0x1);
    #endif

    if (mask & (1<<0))
    {

      if (debouncing==0) {
        LED(LED_B, 1);
        #ifdef USB_SERIAL_OUTPUT
          UARTprintf ("External Button pressed\n");
          UARTprintf("Button pressed %i times\n", button_press_count);
        #endif
        debouncing = 1;
        button_press_count++;
        debounce();
      }
    }

    if (xHigherPriorityTaskWoken)
    {
        // should request a schedule update....
        // ....stay tuned
    }
    uint8_t value = expanderReadByte(0x19);
    #ifdef USB_SERIAL_OUTPUT
      UARTprintf ("value: %x\n", value);
    #endif

    GPIOIntClear(GPIO_PORTD_BASE, mask);
    // vTaskDelay(10);
}
```

Figure 4: Interrupt service handler

```
void
_interrupt_polling (void * notUsed)
{
  GPIOIntRegister(GPIO_PORTD_BASE, _interruptHandlerPortD);
  GPIOIntTypeSet(GPIO_PORTD_BASE, INT_B, GPIO_RISING_EDGE);

  IntPrioritySet(INT_GPIOD, 255);  // Required with FreeRTOS 10.1.1,
  GPIOIntEnable(GPIO_PORTD_BASE, INT_B);
  while(1) {
    vTaskDelay(10);
    LED(LED_B, 0);
  }
}
```

Figure 5: Interrupt task

interrupt pin INTB. This allowed for the wiring of an external led, which was set to blink alongside the Tiva board's internal led. These heartbeat leds were capable of blinking at 250hz before it was impossible to differentiate the states. Also, with the GPB0 and INTB set up, it was possible for an interrupt from the MCP23S17 to trigger an interrupt event on the Tiva board which in turn cleared the interrupt on the MCP23S17.

5.1 Questions

Q1: Why is the MCP23S17 unlike most SPI devices?
Having read the data sheet as well as the wiki page for Serial Peripheral Interface, the only major difference I found was the addition of three hardware address pins.

Q2: What does this mean for EE board layouts?
This allows for the chips to be set up in a way similar to the Daisy-chained SPI bus example on the wiki with the addition of the hardware address pins being wired in series. Since each devise can be given it's own slave address, configured using IOCON.HAEN, its possible to have 8 of these chips in a single daisy chain.

Q3: What is the purpose of the BANK bit?
The BANK bit changes how the addresses are mapped, allowing for the device to either segregate the two ports or have them paired. With the option of SEQOP, this could be used to speed up how the registers are read.

Q4: What are your limiting factors of driving this led on and off being one cycle?
Without having access to lab equipment I was not able to determine the exact speed that the chip was able to drive the pin repeatedly between high and low. However, my guess is that the limiting factors would likely be the led itself as well as the wiring as the MCP23S17 is capable of running at a clock speed of 10MHz while at 250Hz it was difficult to differentiate the led from blinking to being steadily on.

Q5: Describe the technique used to verify that the other pins are unaffected as the led is toggled
To verify that the other pins were not affected by the toggling led I set up code to read and write the values of the GPIO port as well as check the pins physically with a multimeter and leds.

Q6: What does the GPIO expander require to clear the interrupt?
Simply reading the register containing the data of GPB cleared the interrupt.

Q8: In the SPI Wiki, what is meant by calling SPI a 'de-facto standard'? How does this effect us?
What is ment by de-facto standard is that while the protocol stated in the wiki is not a 'written' standard, most manufacturers follow the protocol on their own chips. How this effects us is that while it is more likely than not that a randomly chosen SPI chip would behave like the one made by Motorola in 1979, there is no guarantee as manufacturers are free to design their SPIs however they choose to. Thus, it's important to read the chips data sheet to ensure the proper protocol is used.

Q9: What latency did you encounter in #17? Could you do better?
I was able to measure a latency of about 1s using a stopwatch as I did not have access to an oscilloscope or logic analyzer. As this is nowhere near the clock limits of neither the Tiva Board nor the MCP23S17 chip there is likely plenty of room for improvement.

Q10: List the steps from button-press to ISR clearing the Tiva Interrupt.

- Toggle Button to tie GPB0 to ground.
- GPB0 changes, causing INTB to go high and starting interrupt event on SPI.
- INTB sets PD0 high, causing the Tiva Board to start an interrupt event.
- _interruptHandlerPortD() detects change in PD0.
- _interruptHandlerPortD() starts debouncing, increments button_press_count, prints button press count to UART.
- _interruptHandlerPortD() gives _interrupt_polling() context.
- Tiva board enters _interrupt_polling() and then exits.
- _interruptHandlerPortD() clears interrupt on Tiva board and SPI.

Q11: Under what conditions could the latency be effected by normal task in your system?
The latency could be effected by the connection between INTB and PD0 having enough voltage to keep PD0 high, or the SPI fails to clear the interrupt, causing INTB to remain high.

6 Conclusion

As stated in Results and Analysis, this has been the most difficult lab so far. However, with the exception of the steps requiring lab equipment, the main objectives of the lab were completed. A successful bit banging protocol was implemented along with the successful tying of the SPI extender to external devices such as leds and buttons. Also, the SPI wasset up to cause an external interrupt for the Tiva board. All of this considered, I would say that the lab was a success.

7 Appendix

7.1 spi_master.c

```
/* Standard includes. */
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

/* Kernel includes. */
#include "FreeRTOS.h"
#include "task.h"

/* Hardware includes. */
#include "inc/tm4c123gh6pm.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "inc/hw_memmap.h"

/* Optional includes for USB serial output */
#ifdef USB_SERIAL_OUTPUT
#include "driverlib/rom.h"
#include "utils/uartstdio.h"
#include "driverlib/pin_map.h"
#include "driverlib/uart.h"
#endif

/*local includes*/
#include "assert.h"

// Included for GPIO_O_LOCK and others
#include "inc/hw_gpio.h" //for macro declaration of GPIO_O_LOCK and GPIO_O_CR
#include <inc/hw_types.h>

#include "timers.h"
#include "timing_verify.h"

#define SPEED_TEST 0


#define LED_R (1<<1)
#define LED_G (1<<3)
#define LED_B (1<<2)
#define SW1   (1<<4)
#define SW2   (1<<0)
#define CS (1<<3)
#define SCK (1<<4)
#define SI (1<<5)
#define SO (1<<0)
#define HIGH 1
#define LOW 0
#define INT_B (1<<0)

#define LED_ON(x) (GPIO_PORTF_DATA_R |= (x))
#define LED_OFF(x) (GPIO_PORTF_DATA_R &= ~(x))
#define LED(led,on) ((on)?LED_ON(led):LED_OFF(led))
#define pdTICKSTOMS( xTicks ) ((xTicks * 1000 ) / configTICK_RATE_HZ )

#define CS_HIGH (GPIO_PORTA_DATA_R |= CS)
#define CS_LOW (GPIO_PORTA_DATA_R &= ~CS)
#define setCS(x) ((x)?CS_HIGH:CS_LOW)

#define SCK_HIGH (GPIO_PORTA_DATA_R |= SCK)
#define SCK_LOW (GPIO_PORTA_DATA_R &= ~SCK)
#define setSCK(x) ((x)?SCK_HIGH:SCK_LOW)

#define SI_HIGH (GPIO_PORTA_DATA_R |= SI)
#define SI_LOW (GPIO_PORTA_DATA_R &= ~SI)
#define setMOSI(x) ((x)?SI_HIGH:SI_LOW)

uint32_t SystemCoreClock;
```

```
#ifdef USB_SERIAL_OUTPUT

//*****************************************************************************
//
//: Configure the UART and its pins.  This must be called before UARTprintf().
//
//*****************************************************************************


static void
_configureUART(void)
{
    //
    // Enable UART0
    //
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    //
    // Configure GPIO Pins for UART mode.
    //
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    //
    // Use the internal 16MHz oscillator as the UART clock source.
    //
    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);

    //
    // Initialize the UART for console I/O.
    //
    UARTStdioConfig(0, 115200, 16000000);
}

#endif

uint8_t debouncing = 0;
uint32_t button_press_count = 0;

void debounce()
{
  for (int i = 0; i < 255; i++)
  {
    __asm("    nop\n"
          "    nop\n"
          "    nop\n");
  }
  debouncing = 0;
}

uint8_t
getMISO()
{
  uint8_t value = 0;
  // for (int count=0; count < 8; count++)
  // {
  value = GPIO_PORTE_DATA_R & SO;
  // }
  return value;
}

static void
_setupHardware(void)
{
    //
    // Enable the GPIO port that is used for the on-board LED.
    // This is a TivaDriver library function
    //
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
    HWREG(GPIO_PORTF_BASE + GPIO_O_CR) = 0x01;
```

```
    HWREG(GPIO_PORTF_BASE + GPIO_O_AFSEL) |= 0x01;

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    HWREG(GPIO_PORTA_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
    HWREG(GPIO_PORTA_BASE + GPIO_O_CR) = 0x3c;
    HWREG(GPIO_PORTA_BASE + GPIO_O_DEN) = 0x3c;

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
    HWREG(GPIO_PORTE_BASE + GPIO_O_CR) = 0x03;
    HWREG(GPIO_PORTE_BASE + GPIO_O_DIR) = 0x03;
    HWREG(GPIO_PORTA_BASE + GPIO_O_DEN) = 0x03;
    HWREG(GPIO_PORTE_BASE + GPIO_O_PUR) |= 0x03;

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
    HWREG(GPIO_PORTD_BASE + GPIO_O_CR) = 0x01;
    HWREG(GPIO_PORTD_BASE + GPIO_O_DIR) = 0x01;
    HWREG(GPIO_PORTD_BASE + GPIO_O_DEN) = 0x01;
    HWREG(GPIO_PORTD_BASE + GPIO_O_PUR) |= 0x01;

    // Enable the GPIO pin for the LED (PF3).  Set the direction as output, and
    // enable the GPIO pin for digital function.
    // These are TiveDriver library functions
    //
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, (LED_G|LED_R|LED_B));
    GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, (CS|SCK|SI));
    GPIOPinTypeGPIOInput(GPIO_PORTE_BASE, SO);

    GPIOPinTypeGPIOInput(GPIO_PORTD_BASE, INT_B);

    GPIOPadConfigSet(GPIO_PORTD_BASE, INT_B, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);
    //
    // Set the clocking to run at (SYSDIV_2_5) 80.0 MHz from the PLL.
    //                            (SYSDIV_3) 66.6 MHz
    //                            (SYSDIV_4) 50.0 MHz
    //                            (SYSDIV_5) 40.0 MHz
    //                            (SYSDIV_6) 33.3 MHz
    //                            (SYSDIV_8) 25.0 MHz
    //                            (SYSDIV_10) 20.0 MHz
    //
    SystemCoreClock = 80000000;  // Required for FreeRTOS.

    SysCtlClockSet( SYSCTL_SYSDIV_10 |
                    SYSCTL_USE_PLL |
                    SYSCTL_XTAL_16MHZ |
                    SYSCTL_OSC_MAIN);

}

uint8_t
transfer(uint8_t out)
{
  uint8_t count, in = 0;
  setSCK(LOW);
  for (count = 0; count < 8; count++)
  {
    in <<= 1;
    setMOSI(out & 0x80);
    setSCK(HIGH);
    in += getMISO();
    setSCK(LOW);
    out <<= 1;
  }
  setMOSI(0);

  return (in);
}

static uint8_t
expanderReadByte(uint8_t address)
{
  uint8_t value, preRead = 0x41;
  setSCK(LOW);
```

```
  setCS(LOW);
  transfer(preRead);
  transfer(address);
  value = transfer(0);
  return value;
}

void
expanderWriteByte(uint8_t address, uint8_t value)
{

  uint8_t preRead = 0x40;
  setSCK(LOW);
  setCS(LOW);
  transfer(preRead);
  transfer(address);
  transfer(value);
  return;
}

static void
_interruptHandlerPortD(void)
{
    //
    // We have not woken a task at the start of the ISR.
    //
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;

    uint32_t mask = GPIOIntStatus(GPIO_PORTD_BASE, 1);
    #ifdef USB_SERIAL_OUTPUT
      UARTprintf ("mask: %x\n", (mask) & 0x1);
    #endif

    if (mask & (1<<0))
    {

      if (debouncing==0) {
        LED(LED_B, 1);
        #ifdef USB_SERIAL_OUTPUT
          UARTprintf ("External Button pressed\n");
          UARTprintf("Button pressed %i times\n", button_press_count);
        #endif
        debouncing = 1;
        button_press_count++;
        debounce();
      }
    }

    if (xHigherPriorityTaskWoken)
    {
        // should request a schedule update....
        // ....stay tuned
    }
    uint8_t value = expanderReadByte(0x19);
    #ifdef USB_SERIAL_OUTPUT
      UARTprintf ("value: %x\n", value);
    #endif

    GPIOIntClear(GPIO_PORTD_BASE, mask);
    // vTaskDelay(10);
}

void
LED_REMOTE(uint8_t on) {
  CS_HIGH;
  expanderWriteByte(0x9, on);
}

static void
_heartbeat( void *notUsed )
{
   uint32_t green500ms = 500; // 1 second for on off
```

```
   uint32_t ledOn = 1;

    while(true)
    {
        vTaskDelay(green500ms / portTICK_RATE_MS);
        uint8_t out = 0x0;
        LED(LED_G, ledOn);
        LED_REMOTE(ledOn);
        ledOn = !ledOn;
        #ifdef USB_SERIAL_OUTPUT
          UARTprintf("on:%i\n",ledOn);
        #endif
        CS_HIGH;
        out = expanderReadByte(0x19);
        #ifdef USB_SERIAL_OUTPUT
          UARTprintf("0x19:%x\n",out & (1));
        #endif
    }
}

void
expanderInit(void)
{
    uint8_t out = 0;
    #ifdef USB_SERIAL_OUTPUT
      UARTprintf("Configure IOCONA\n");
    #endif
    CS_HIGH;
    expanderWriteByte(0x0A, 0xA2);

    #ifdef USB_SERIAL_OUTPUT
      UARTprintf("Configure IODIRA\n");
    #endif
    CS_HIGH;
    expanderWriteByte(0x00, 0xfe);

    #ifdef USB_SERIAL_OUTPUT
      UARTprintf("Configure GPPUA\n");
    #endif
    CS_HIGH;
    expanderWriteByte(0x06, 0x01);

    out = 0;
    #ifdef USB_SERIAL_OUTPUT
      UARTprintf("Configure IOCONB\n");
    #endif
    CS_HIGH;
    expanderWriteByte(0x15, 0xA2);

    #ifdef USB_SERIAL_OUTPUT
      UARTprintf("Configure IODIRB\n");
    #endif
    CS_HIGH;
    expanderWriteByte(0x10, 0xff);

    #ifdef USB_SERIAL_OUTPUT
      UARTprintf("Configure IPOLB\n");
    #endif
    CS_HIGH;
    expanderWriteByte(0x11, 0x01);

    #ifdef USB_SERIAL_OUTPUT
      UARTprintf("Configure GPPUB\n");
    #endif
    CS_HIGH;
    expanderWriteByte(0x16, 0x01);

    #ifdef USB_SERIAL_OUTPUT
      UARTprintf("Configure GPINTENB\n");
    #endif
    CS_HIGH;
    expanderWriteByte(0x12, 0x01);
```

```
    #ifdef USB_SERIAL_OUTPUT
      UARTprintf("Configure INTCONB\n");
    #endif
    CS_HIGH;
    expanderWriteByte(0x14, 0x01);

    CS_HIGH;
    out = expanderReadByte(0x05);
    #ifdef USB_SERIAL_OUTPUT
      UARTprintf("IOCON:%x\n",out);
    #endif

    CS_HIGH;
    out = expanderReadByte(0x00);
    #ifdef USB_SERIAL_OUTPUT
      UARTprintf("IODIRA:%x\n",out);
    #endif
    CS_HIGH;
    LED_REMOTE(1);
    CS_HIGH;
    out = expanderReadByte(0x0A);
    #ifdef USB_SERIAL_OUTPUT
      UARTprintf("OLATA:%x\n",out);
    #endif
}

void
_interrupt_polling (void * notUsed)
{
  GPIOIntRegister(GPIO_PORTD_BASE, _interruptHandlerPortD);
  GPIOIntTypeSet(GPIO_PORTD_BASE, INT_B, GPIO_RISING_EDGE);

  IntPrioritySet(INT_GPIOD, 255);  // Required with FreeRTOS 10.1.1,
  GPIOIntEnable(GPIO_PORTD_BASE, INT_B);
  while(1) {
    vTaskDelay(10);
    LED(LED_B, 0);
  }
}

int main( void )
{
    _setupHardware();
    SCK_LOW;
    CS_HIGH;
    #ifdef USB_SERIAL_OUTPUT
            void spinDelayMs(uint32_t ms);
            _configureUART();
            spinDelayMs(1000);  // Allow UART to setup
            UARTprintf("\n\n\nHello from SPI Protocal main()\n");
    #endif

    expanderInit();

    xTaskCreate(_heartbeat,
                "green",
                configMINIMAL_STACK_SIZE,
                NULL,
                tskIDLE_PRIORITY + 1,  // higher numbers are higher priority..
                NULL );

    xTaskCreate(_interrupt_polling,
                "polling",
                configMINIMAL_STACK_SIZE,
                NULL,
                tskIDLE_PRIORITY + 5,  // higher numbers are higher priority..
                NULL );

    /* Start the tasks and timer running. */\
    vTaskStartScheduler();
```

```
    assert(0); // we should never get here..

    return 0;
}
```