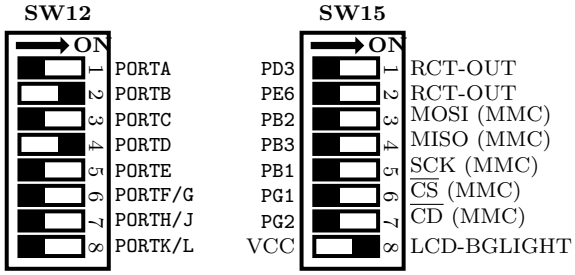
B Debugging: PWM / Timer / ADC

Please set up the switches and jumpers in the following way:



- Connect *J12* to *GND*.
- Set *J15* to *PF0*.
- Set *J18* to *VCC*.

Task Description:

A colleague of yours heard that you are attending the Microcontroller Lab at TU and asks you for help with a problematic microcontroller application. Your colleague uses a ATmel AVR microcontroller with 4KB RAM and clocked with 16MHz.
The goal is to implement an application which outputs a PWM signal to control a motor and uses an ADC to get an Input value. As resources are really limited all of this has to be done with a single Timer.
Thus Timer 1 is configured to run in phase correct PWM mode with a frequency of 2 Hz with a prescaler of 64. ICR1 is used as TOP and OC1A as update with non-inverting mode for waveform generation. The PWM signal needs to have a 20% duty cycle.
As the ADC needs to be triggered with the same frequency as the timer, your colleague runs it in auto trigger mode with Timer 1 as its source. The ADC samples channel 0 with AREF as reference and a prescaler of 128. In the ADC interrupt a callback function is called with the 10-bit value of the conversion result. Further, a recursive calculation, with interrupts reenabled, has to be called with half of the conversion result.
For development purpose, your colleague also included a liveness counter using Timer 2 running at the bottom left corner of the LCD. Since the liveness counter is taken from a library it should work fine. Further PORTD gets increased by Timer 2 at the frequency of 2Hz (same as the target PWM frequency), to allow you a rough estimate if your program is correct (note that the phase may be shifted).
When your colleague moved from his bigger development platform to the target platform, thing stopped when the ADC reads larger values, at least that is what you were told. As your colleague tried to fix this bug in a rush it got even worse. To simulate the target platform, uncomment the assignment of SP in `main`.

Summary of Specification:

- Phase correct PWM Signal on OC1A, with 2 Hz, 20% duty cycle, non-inverting output, and a prescaler of 64.
- Timer 1 auto triggers the ADC, which converts channel 0 with a prescaler of 128. The ISR executes callback and a recursive function with the conversion result.

Note:

- There are 5 bugs in the code, all of which can be fixed in at most 2 consecutive lines. "Excessive" fixes for these five bugs (4+ lines) will not be considered correct.
- The 5 bugs are divided to 3 difficulties:
  - There are 2 easy to find bugs each of which worth **0.5** points.
  - Finding the 2 advanced bugs will earn you additional **1** points each.
  - The hardest bug will be rewarded with **2.5** points.
- The sample solution has not all bugs fixed!

B.1 Task 2 source

```
//*****************************************************************************
// Task 1: Debugging: External Interrupt / Timer / callback
//*****************************************************************************

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>

#include <stdint.h>
#include <stdlib.h>

#include <util/atomic.h>

// prototypes
// NO NEED TO TOUCH
void adcInit(void (*_callback)(uint16_t));
void timer1Init();

extern void adjustValue(uint16_t);
extern void lab_init();
extern void display_value(uint8_t value);
extern void livenessInit();
extern uint8_t g(uint16_t v);
```

```
// store callback function
static void (*adcCallback)(uint16_t);

//*****************************************************************************
// main
//*****************************************************************************
int main() {
    //***************************************
    // DO NOT TOUCH
    lab_init();        // must be first statement!
    //***************************************

    // TODO: Uncomment for target simulation
    // set stack pointer to simulate small target with 4K RAM
    //SP = 4096;

    // must be set for an acceptable solution

    SP = 4096;


    adcInit(adjustValue);

    timer1Init();


    sei();

    set_sleep_mode(SLEEP_MODE_IDLE);

    while (1) {
        cli();
        sleep_enable();
        sei();
        sleep_cpu();
        sleep_disable();
        sei();
    }

    return 0;
}

void timer1Init() {
    // init timer
```

```
    // WGM13:WGM10 = "1010" is PC PWM mode with ICR1 as TOP value
    TCCR1B |= (1<<WGM13);
    TCCR1B &= ~(1<<WGM12);
    TCCR1A |= (1<<WGM11 | 1<<COM1A1);
    TCCR1A &= ~(1<<WGM10 | 1<<COM1A0);

    ICR1 = 8000L;
    OCR1A = 10 * 020;
```

```
    // === CORRECT === Medium M1

    // ((16000000Hz/64)*(1/2Hz)) / 2 = 62500

    ICR1 = 62500;

    OCR1A = 625 * 20; // ICR1 * (20/100) = 20% duty cycle
```

```
    // Configure output pin
    DDRB |= (1<<PB5);

    // start timer
    TCCR1B &= ~(1<<CS12);
    TCCR1B |= (1<<CS10 | 1<<CS11);
}

void adcInit(void (*_callback)(uint16_t)) {
```

```
    // === CORRECT === Easy E1

    if (_callback == NULL) {
```

```
    if (_callback == NULL); {
```

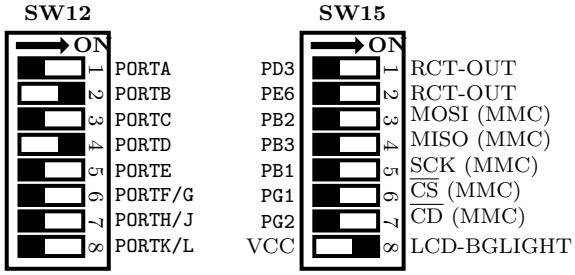
```
        abort();

    } else {

        adcCallback = _callback;

    }
    ADMUX = (1 << ADLAR);
```

```
    // === CORRECT === Easy E2

    ADMUX = 0; // no left adjustment ADLAR


    //  AUTO trigger with Timer1 TOV
    ADCSRA = (1<<ADIE | 1<<ADEN | 1<<ADSC | 1<<ADPS2 | 1<<ADPS1 | 1<<ADPS0 |
              1<<ADATE);


    // === CORRECT === Medium M2

    // BUG: no  Timer 1 TOV ISR installed => reset


    // enable the TOV for ADC

    TIMSK1 = (1<<TOIE1);


    ADCSRB = (1<<ADTS2 | 1<<ADTS1); // Timer1 TOV
}


// === CORRECT === Hard H1

static uint16_t f(uint16_t v) {


static uint16_t f(uint32_t v) {


    if (v <= 0) {
        return 1;
    } else {
        return f(v-1) - g(v);
    }
}

ISR(ADC_vect) {
    uint16_t val;

    val = ADC;
```

```
    adcCallback(val);

    NONATOMIC_BLOCK(NONATOMIC_FORCEOFF) {
        // ADC value must be right shifted by 1!
        f(val >> 1);
    }
}
```

B Debugging: PWM / Timer / ADC

Please set up the switches and jumpers in the following way:



- Connect *J12* to *GND*.
- Set *J15* to *PF0*.
- Set *J18* to *VCC*.

Task Description:

A colleague of yours heard that you are attending the Microcontroller Lab at TU and asks you for help with a problematic microcontroller application. Your colleague uses a ATmel AVR microcontroller with 4KB RAM and clocked with 16MHz.
The goal is to implement an application which outputs a PWM signal to control a motor and uses an ADC to get an Input value. As resources are really limited all of this has to be done with a single Timer.
Thus Timer 1 is configured to run in phase correct PWM mode with a frequency of 2 Hz with a prescaler of 64. ICR1 is used as TOP and OC1A as update with non-inverting mode for waveform generation. The PWM signal needs to have a 20% duty cycle.
As the ADC needs to be triggered with the same frequency as the timer, your colleague runs it in auto trigger mode with Timer 1 as its source. The ADC samples channel 0 with AREF as reference and a prescaler of 128. In the ADC interrupt a callback function is called with the 10-bit value of the conversion result. Further, a recursive calculation, with interrupts reenabled, has to be called with half of the conversion result.
For development purpose, your colleague also included a liveness counter using Timer 2 running at the bottom left corner of the LCD. Since the liveness counter is taken from a library it should work fine. Further PORTD gets increased by Timer 2 at the frequency of 2Hz (same as the target PWM frequency), to allow you a rough estimate if your program is correct (note that the phase may be shifted).
When your colleague moved from his bigger development platform to the target platform, thing stopped when the ADC reads larger values, at least that is what you were told. As your colleague tried to fix this bug in a rush it got even worse. To simulate the target platform, uncomment the assignment of SP in `main`.

Summary of Specification:

- Phase correct PWM Signal on OC1A, with 2 Hz, 20% duty cycle, non-inverting output, and a prescaler of 64.
- Timer 1 auto triggers the ADC, which converts channel 0 with a prescaler of 128. The ISR executes callback and a recursive function with the conversion result.

Note:

- There are 5 bugs in the code, all of which can be fixed in at most 2 consecutive lines. "Excessive" fixes for these five bugs (4+ lines) will not be considered correct.
- The 5 bugs are divided to 3 difficulties:
  - There are 2 easy to find bugs each of which worth **0.5** points.
  - Finding the 2 advanced bugs will earn you additional **1** points each.
  - The hardest bug will be rewarded with **2.5** points.
- The sample solution has not all bugs fixed!

B.1 Task 2 source

```
//*****************************************************************************
// Task 1: Debugging: External Interrupt / Timer / callback
//*****************************************************************************

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>

#include <stdint.h>
#include <stdlib.h>

#include <util/atomic.h>

// prototypes
// NO NEED TO TOUCH
void adcInit(void (*_callback)(uint16_t));
void timer1Init();

extern void adjustValue(uint16_t);
extern void lab_init();
extern void display_value(uint8_t value);
extern void livenessInit();
extern uint8_t g(uint16_t v);
```

```
// store callback function
static void (*adcCallback)(uint16_t);

//*****************************************************************************
// main
//*****************************************************************************
int main() {
    //***************************************
    // DO NOT TOUCH
    lab_init();        // must be first statement!
    //***************************************

    // TODO: Uncomment for target simulation
    // set stack pointer to simulate small target with 4K RAM
    // SP = 4096;

    // must be set for an acceptable solution

    SP = 4096;


    adcInit(adjustValue);

    timer1Init();


    sei();

    set_sleep_mode(SLEEP_MODE_IDLE);

    while (1) {
        cli();

        sleep_enable();

        sei();  // === CORRECT === Easy E1

        sleep_cpu();
        sleep_disable();

        sei();
    }
```

```
    return 0;
}

void timer1Init() {
    // init timer
    // WGM13:WGM10 = "1010" is PC PWM mode with ICR1 as TOP value
    TCCR1B |= (1<<WGM13);
    TCCR1B &= ~(1<<WGM12);
    TCCR1A |= (1<<WGM11 | 1<<COM1A1);
    TCCR1A &= ~(1<<WGM10 | 1<<COM1A0);

    ICR1 = 80000L;
    OCR1A = 10 * 020;

    // === CORRECT === Medium M1

    // ((16000000Hz/64)*(1/2Hz)) / 2 = 62500

    ICR1 = 62500;

    OCR1A = 625 * 20; // ICR1 * (20/100) = 20% duty cycle


    // Configure output pin OCA1
    DDRB |= (1<<PB5);

    // start timer
    TCCR1B &= ~(1<<CS12);
    TCCR1B |= (1<<CS10 | 1<<CS11);
}

void adcInit(void (*_callback)(uint16_t)) {

    // === CORRECT === Easy E2

    if (_callback == NULL) {


    if (_callback = NULL) {

        abort();

    } else {
```

```
        adcCallback = _callback;

    }

    ADMUX = 0;

    //  AUTO trigger with Timer1 TOV
    ADCSRA = (1<<ADIE | 1<<ADEN | 1<<ADSC | 1<<ADPS2 | 1<<ADPS1 | 1<<ADPS0 |
              1<<ADATE);

    ADCSRB = (1<<ADTS2 | 1<<ADTS1); // Timer1 TOV
}
```

```
// === CORRECT === Hard H1

static uint16_t f(uint16_t v) {
```

```
static uint16_t f(uint32_t v) {

    if (v <= 0) {
        return 1;
    } else {
        return f(v-1) - g(v);
    }
}

ISR(ADC_vect) {
```

```
    // === CORRECT === Medium M2

    uint16_t val;
```

```
    uint8_t val;

    val = ADC;

    adcCallback(val);
```

```
    NONATOMIC_BLOCK(NONATOMIC_FORCEOFF) {
        // ADC value must be right shifted by 1!
        f(val >> 1);
    }
}
```