A microcontroller-based delay generator for predicting shock arrival in the X2 expansion tube.

Mechanical Engineering Report 2011/05
P. A. Jacobs
School of Mechanical Engineering
The University of Queensland.

February 8, 2012

Contents

Abstract

This report describes the hardware and firmware for the custom-built timing box that synchronises the spectrometer with the X2 expansion tunnel flow. The box monitors the pressure signals at two locations in the acceleration tube and then predicts the time of arrival of the shock as it emerges from the acceleration tube into the test chamber. The output is a 5 volt (digital) pulse to let the spectrometer know when to record its data.

1 Introduction

The box, based on a small microcontroller, monitors the pressure signals at two locations close to the end of the acceleration tube and then predicts the time of arrival of the shock as it emerges from the acceleration tube into the test chamber. The box was built to support an experimental study [1] in which small shot-to-shot changes in shock speeds were making it difficult to record spectral data from the flow immediately following the shock. It was decided to monitor the last two pressure sensors in the acceleration tube make an estimate of the shock speed from the arrival times. A typical signal is shown in Fig.1. The time between the shock arrival at the second pressure sensor and its arrival at the end of the tube is about 50 microseconds.

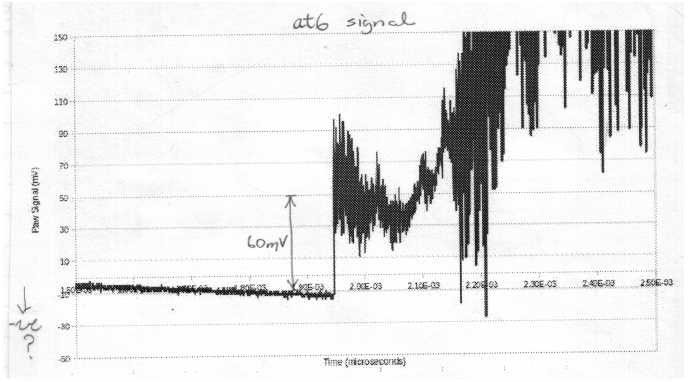


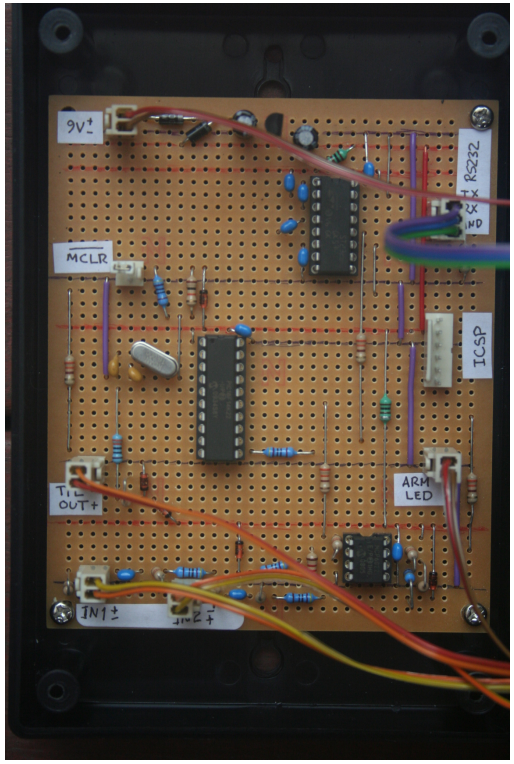
Figure 1: Raw pressure signal at sensor location AT6.

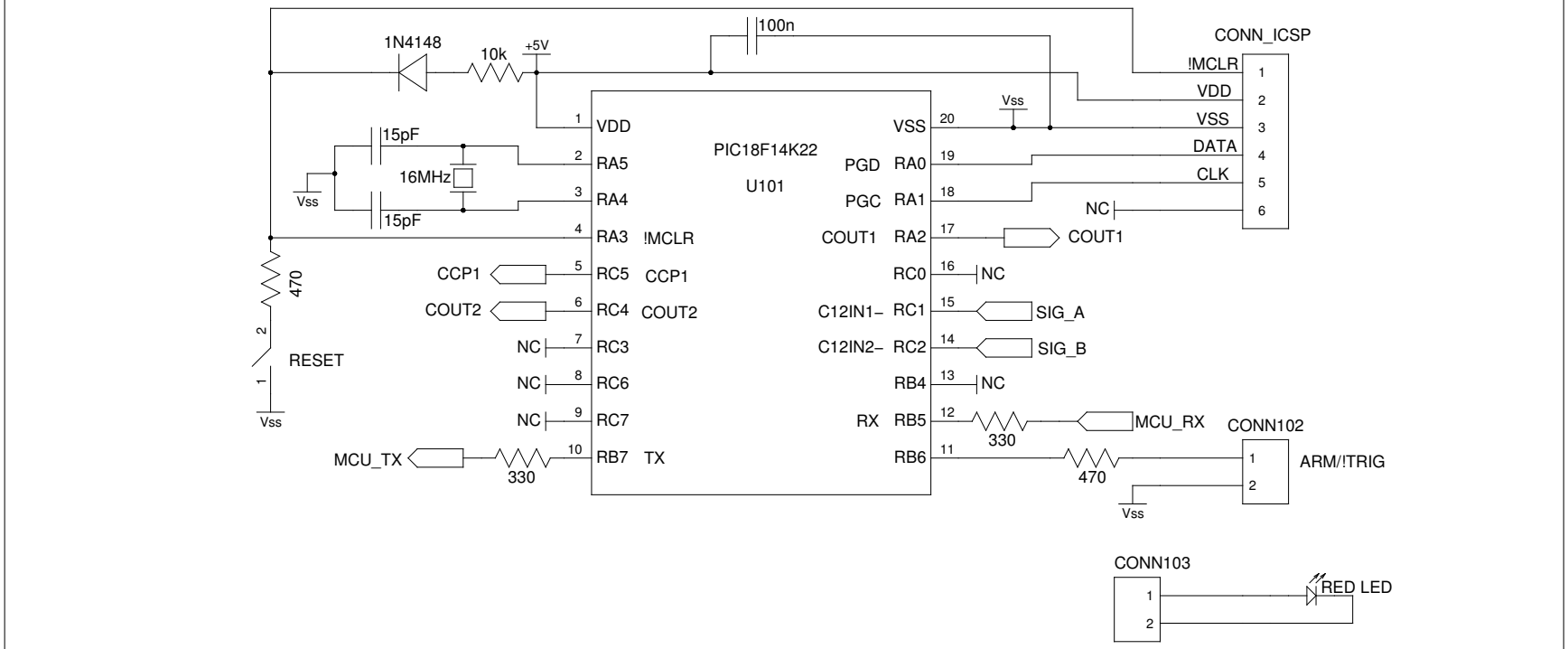
Configuration of the box is done via the serial-port connection. The firmware has built-in help and provides some guidance. Once configured, the box may be "armed" and it then waits for the shock arrival at the upstream pressure sensor. The time between arrival at the two pressure sensors is recorded as a count on the hardware timer. This timer is allowed to continue running while the predicted count of the timer, on shock arrival at the end of the tube, is estimated and put into the compare register. When this target count is reached the digital output signal is pulsed. Timer ticks are 62.5 ns so, allowing for a few cycles of house-keeping, the precision of the output pulse is within one microsecond.

2 Hardware

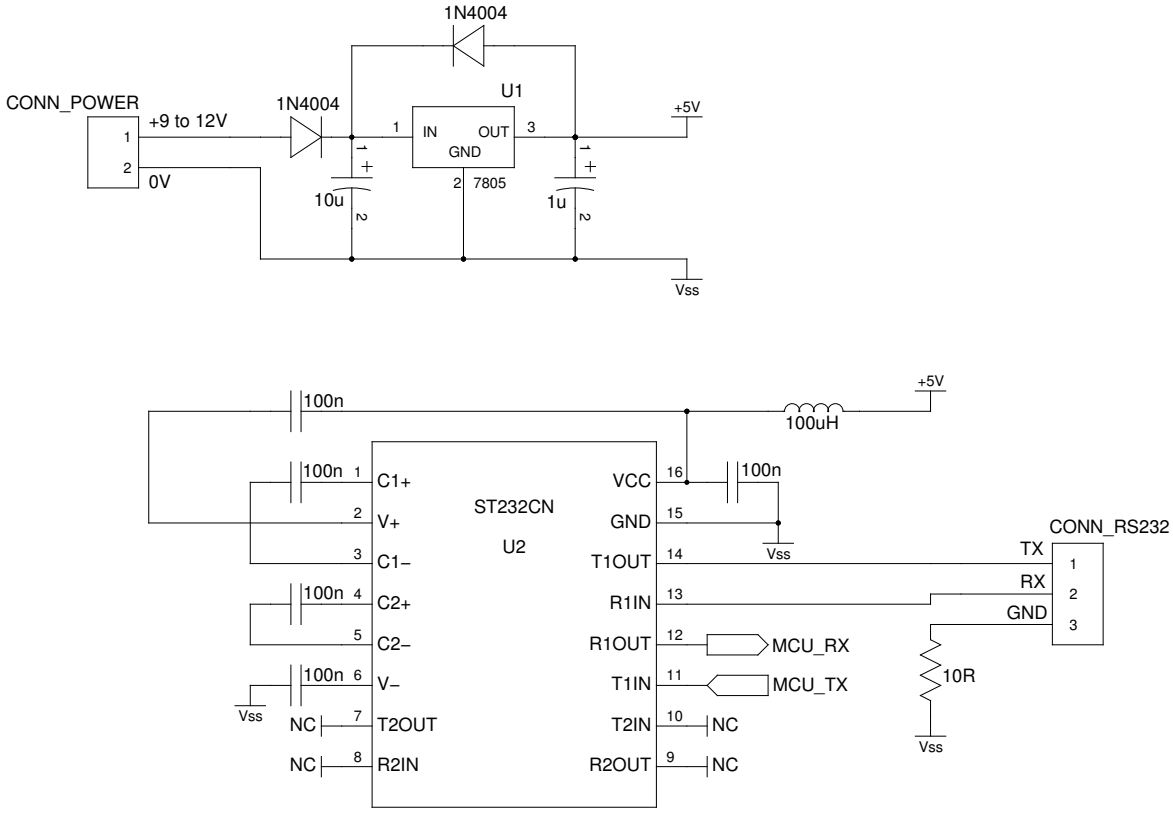
The main components of the box are:

- High-pass filters and buffer amplifiers (with a gain of about 20) for each pressure signal.
- A PIC18F14K22 microcontroller [2] that performs the timing and overall coordination. To get a sufficiently fast response, we make use of the hardware timers and comparators built into the microcontroller and do a simple calculation of the required timer count.
- An ST232 level converter to get the UART connected to the serial port of an external computer.

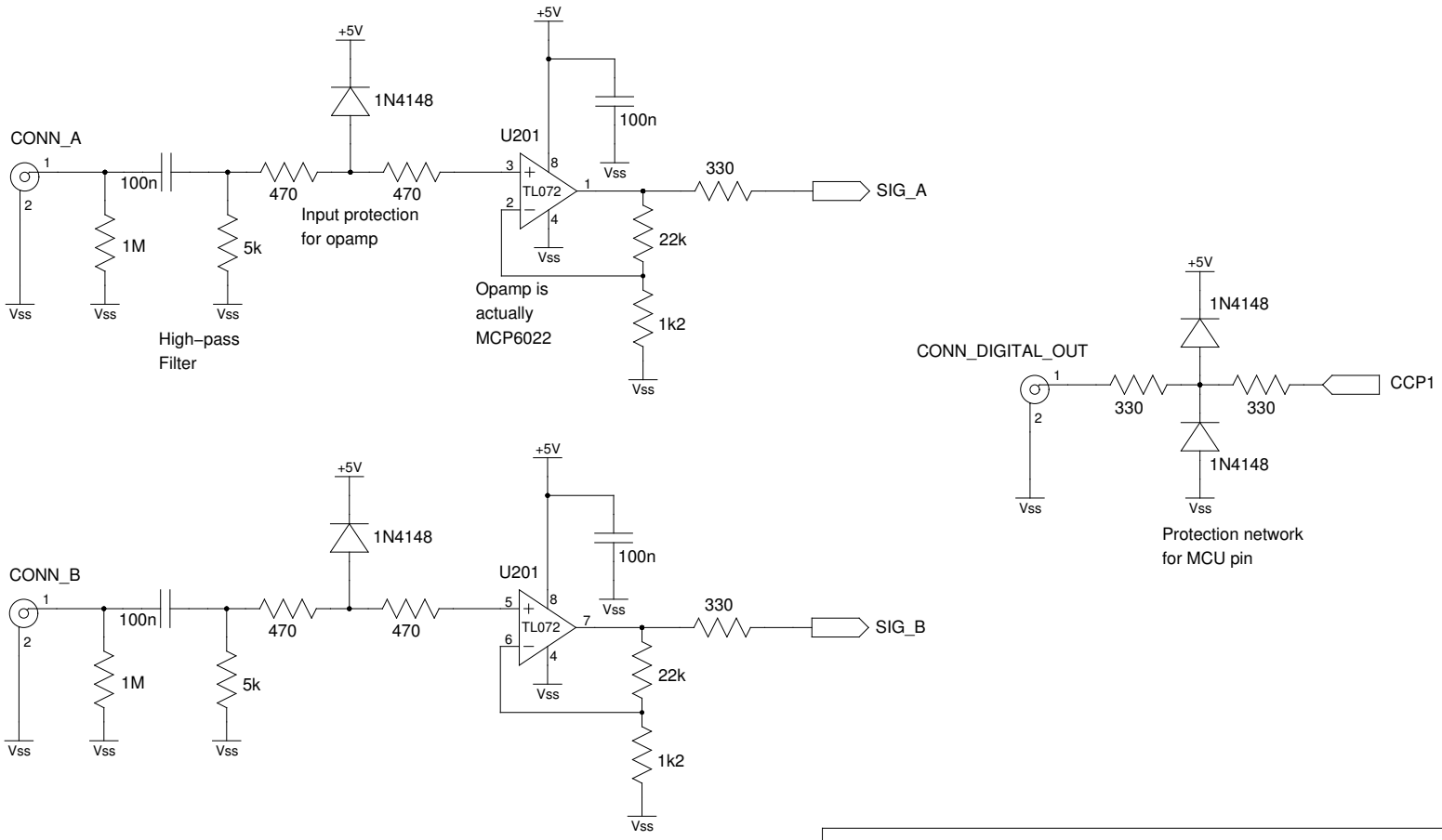




TITLE	X2 Shock timer –– MCU		
FILE:			REVISION:
PAGE	1	OF 3	DRAWN BY: Peter Jacobs



TITLE	X2 Shock Timer –– Power and Communication		
FILE:			REVISION:
PAGE	2	OF 3	DRAWN BY: Peter Jacobs



TITLE	X2 Shock Timer –– INPUT/OUTPUT Conditioning		
FILE:			REVISION:
PAGE	3	OF 3	DRAWN BY: Peter Jacobs

3 Firmware

The firmware running within the microcontroller has been compiled with Microchip's C18 compiler. The code uses quite a few Microchip-specific definitions and, also, the peripheral hardware library that comes with the C18 compiler. In-circuit programming was done via the MPLAB development environment.

```
// x2_shock_timer.c
// Deteck the passage of a shock at two pressure sensors spaces 500mm apart
// and output a TTL-level signal when the shock is expected to have travelled
// a further 450mm.
//
// PJ
// September 2010
//
//-------------------------------------------------------------
#define VERSION_STRING "0.4 09-Oct-2010"

#include "p18f14k22.h"
#include <stdio.h>
#include <string.h>
#include <usart.h>
#include <timers.h>
#include <delays.h>
#include <adc.h>
#include <stdlib.h>
#include <limits.h>

#pragma config FOSC = HS
#pragma config PLLEN = ON
#pragma config FCMEN = ON
#pragma config IESO = ON
#pragma config PWRTEN = ON
#pragma config BOREN = OFF
#pragma config WDTEN = OFF
#pragma config MCLRE = ON
#pragma config LVP = OFF
#pragma config XINST = OFF

// A buffer to contain incoming text commands.
#define TEXT_BUFFER_SIZE 20
static unsigned char text_buffer[TEXT_BUFFER_SIZE];
#define ACK 0x01
#define BS 0x08
#define CR 0x0D
#define LF 0x0A
```

```
#define EOT 0x04
#define ETX 0x03
#define RS232_DEBUG_MESSAGES  1

// Crystal frequency in Hz
#define F_OSC (64000000L)

// Names for some hardware pins
#define ARM_LED (LATBbits.LATB6)
#define LED_ON  1;
#define LED_OFF 0;

void initialize_hardware( void );
unsigned char set_CVR_value(unsigned char step);
void arm_and_wait( void );
void pulse_output( void );
void display_settings( void );

unsigned int time12;
unsigned int extra = 0;
unsigned int time23;

//-----------------------------------------------------------------------

void initialize_hardware( void )
{
    // Analogue pins AN0 through AN4, and reference voltages Vss—Vdd.
    ADCON1 = 0b00001010;

    // Set up digital ports.
    ANSELbits.ANS2 = 0; // enable digital buffer on RA2
    // Comparator 1 output on RA2
    TRISA = 0b11111011;
    LATA = 0;
    // Port B has serial communication: TX on RB7 and RX on RB5
    ANSELHbits.ANS11 = 0; // enable digital buffer on RB5
    // ARM_LED output on RB6
    TRISB = 0b10111111;
    LATB = 0;
    // Comparator 2 output on RC4, compare match on RC5
    TRISC = 0b11001111;
    LATC = 0;

    // Set up serial port for use via polling (not interrupts).
    // From page 69 of the C-library guide, we have the following formula.
    // baud-rate = F_OSC / (16 * (spbrg + 1))
#   define BAUD_RATE 38400
```

```
#   define SET_BRG ((F_OSC/BAUD_RATE/16) - 1)
    OpenUSART( USART_TX_INT_OFF & USART_RX_INT_OFF & USART_ASYNCH_MODE &
                USART_EIGHT_BIT & USART_CONT_RX & USART_BRGH_HIGH, SET_BRG );

    // Comparator 1, with input from C12IN1- and output on RA2
    CM1CON0bits.C1OE = 1;
    CM1CON0bits.C1POL = 1; // invert to get right-way up
    CM1CON0bits.C1SP = 1; // high speed
    CM1CON0bits.C1R = 1; // non-inverting input comes from C1Vref
    CM2CON1bits.C1RSEL = 0; // variable voltage reference
    // inverting input comes from C12IN1-
    CM1CON0bits.C1CH1 = 0; CM1CON0bits.C1CH0 = 1;
    CM1CON0bits.C1ON = 1;

    // Comparator 2, with input from C12IN2- and output on RC4
    CM2CON0bits.C2OE = 1;
    CM2CON0bits.C2POL = 1; // invert to get right-way up
    CM2CON0bits.C2SP = 1; // high speed
    CM2CON0bits.C2R = 1; // non-inverting input comes from C1Vref
    CM2CON1bits.C2RSEL = 0; // variable voltage reference
    // inverting input comes from C12IN2-
    CM2CON0bits.C2CH1 = 1; CM2CON0bits.C2CH0 = 0;
    CM2CON0bits.C2ON = 1;

    // Comparator voltage reference comes from fixed reference voltage of 1.024V.
    VREFCON0bits.FVR1S1 = 0; VREFCON0bits.FVR1S0 = 1;
    VREFCON0bits.FVR1EN = 1;
    while ( !VREFCON0bits.FVR1ST ) ; // wait until stable
    VREFCON1bits.D1PSS1 = 1; VREFCON1bits.D1PSS0 = 0; // from FVR output
    VREFCON1bits.D1NSS = 0; // from VSS
    set_CVR_value(8); //set it on a moderate value
    VREFCON1bits.D1EN = 1; // turn it on
} // end initialize_hardware()

unsigned char set_CVR_value(unsigned char step)
// Sets bits 5:0 in voltage-reference control register.
{
    if ( step > 31 ) step = 31; // maximum steps in R-ladder
    VREFCON2 = step;
    Delay100TCYx(100); // Allow it to settle.
    return step;
}

void arm_and_wait( void )
// Waits for the shock signals and then sets an output
// at the expected arrival time in the test-section.
{
```

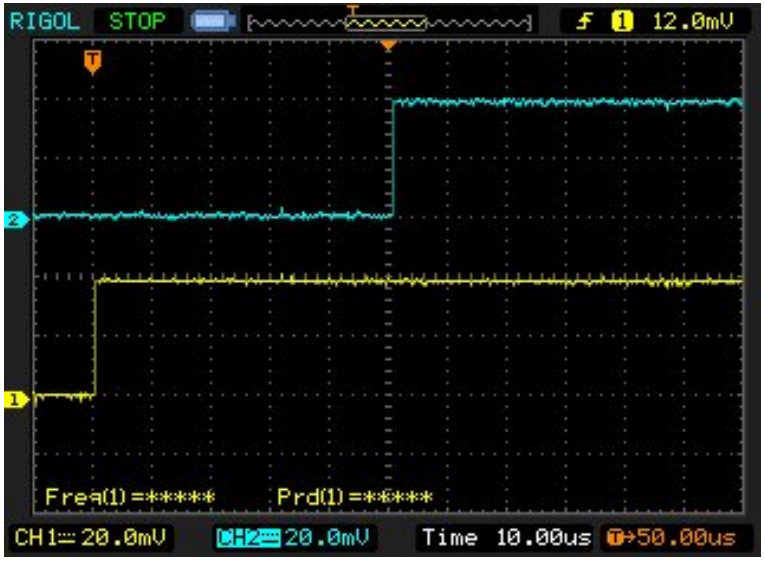
```
    unsigned char low, high;

    // Set up timer
    T1CONbits.TMR1CS = 0; // ticks at FOSC/4
    T1CONbits.RD16 = 1; // read all 16 bits at once
    TMR1H = 0; TMR1L = 0;
    ARM_LED = LED_ON;

    // Wait for arrival at first transducer, then start timing.
    while ( !CM1CON0bits.C1OUT ) ;
    T1CONbits.TMR1ON = 1;

    // On arrival at second transducer, look at timer and
    // predict timer value for arrival at test section.
    while ( !CM2CON0bits.C2OUT ) ;
    low = TMR1L; high = TMR1H;
    time12 = (((unsigned int) high) << 8) | low;
    time23 = (time12 * 2) + extra;
    CCPR1H = (unsigned char)(time23 >> 8); CCPR1L = (unsigned char)time23;
    // Compare mode: initialize CCP1 pin low, set output on compare match.
    CCP1CON = 0b00001000;
    PIR1bits.CCP1IF = 0;
    T3CONbits.T3CCP1 = 0; // look at timer 1

    // On compare match, we expect shock to arrive at test section.
    // Send our TTL signal (already on CCP1).
    while ( !PIR1bits.CCP1IF ) ;
    T1CONbits.TMR1ON = 0; // stop the timer
    ARM_LED = LED_OFF;

    // Wait a while and clean up.
    Delay1KTCYx(100);
    CCP1CON = 0; // Turn off compare module.
    PIR1bits.CCP1IF = 0;
    LATCbits.LATC5 = 0; // make sure that out output signal is off.
}

void pulse_output( void )
// Used for manual output, via command.
{
    ARM_LED = LED_ON;
    LATCbits.LATC5 = 1;
    Delay1KTCYx(100);
    LATCbits.LATC5 = 0;
    Delay10KTCYx(200);
    ARM_LED = LED_OFF;
    return;
```

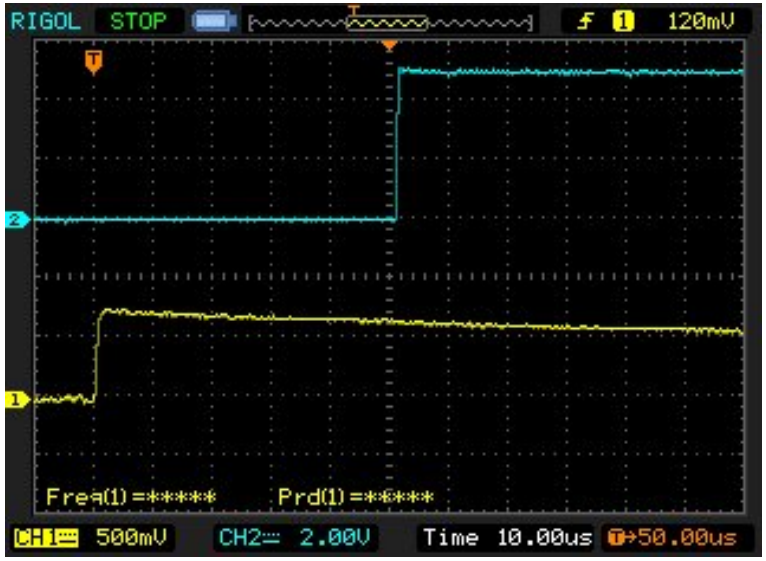
```
}

void display_settings( void )
{
    printf((const far rom char *)"extra= %d       (extra delay in counts, with 62.5ns/count)\r\n", extra);
    printf((const far rom char *)"VREFCON2= %d    (cvref steps from 31, over 1.024V)\r\n", VREFCON2);
    return;
}

//-----------------------------------------------------------------------
void main( void )
{
    static unsigned char text_length;
    static unsigned char character;
    unsigned char i, v;
    char small_text[10];

    // Let's begin...
    initialize_hardware();
    ARM_LED = LED_ON;
    Delay10KTCYx(200);
    ARM_LED = LED_OFF;

    while ( BusyUSART() );
    printf( (const far rom char *) "\r\nX2 shock speed timer " );
    printf( (const far rom char *) VERSION_STRING );
    printf( (const far rom char *) "\r\n" );
    while ( 1 ) {
        // Wipe text_buffer clean.
        for ( i = 0; i < TEXT_BUFFER_SIZE; ++i ) text_buffer[i] = 0;
        text_length = 0;
        // Accumulate characters until we get a Carriage-Return character.
#       define ECHO_CHARACTERS 1
        while ( 1 ) {
            while ( !DataRdyUSART() ); // wait for a character to arrive
            character = ReadUSART();
            if ( character == BS && text_length > 0 ) {
                // wipe out the previously received character
                text_length--;
                text_buffer[text_length] = 0;
#               if ECHO_CHARACTERS
                while ( BusyUSART() ); putcUSART( character ); // echo the backspace character
#               endif
                continue;
            }
            if ( (character != CR) && (text_length < TEXT_BUFFER_SIZE-1) ) {
                // normal character
```

```
                text_buffer[text_length] = character;
                text_length++;
#               if ECHO_CHARACTERS
                while ( BusyUSART() ); putcUSART( character ); // echo the character
#               endif
            } else {
#               if ECHO_CHARACTERS
                putrsUSART( (const far rom char *)"\r\n" );
#               endif
                break;
            }
        } // end while
        text_buffer[text_length] = 0; // terminate with NUL char
        if ( text_length == 0 ) continue; // must have received a CR with no command text
        // printf( (const far rom char*)"Command: %s\r\n", text_buffer );

        if ( !strncmppgm2ram((char *)text_buffer,(rom far char *)"status",6) ) {
            putrsUSART((const far rom char *)"Firmware version= ");
            putrsUSART((const far rom char *)VERSION_STRING);
            putrsUSART((const far rom char *)"\r\n");
            display_settings();
            printf((const far rom char *)"OK\r\n");
            continue;
        }
        if ( !strncmppgm2ram((char *)text_buffer,(rom far char *)"help",4) ||
             !strncmppgm2ram((char *)text_buffer,(rom far char *)"?",1) ) {
            putrsUSART((const far rom char *)"Available commands are:\r\n");
            putrsUSART((const far rom char *)"    status           (pings device, returns version string)\r\n");
            putrsUSART((const far rom char *)"    help             (or ? to get this list)\r\n");
            putrsUSART((const far rom char *)"    set cvref nn     (set comparator reference voltage in steps 0..31 of 1.024V\r\n");
            putrsUSART((const far rom char *)"    set extra nnnn   (set extra count, in 62.5ns steps\r\n");
            putrsUSART((const far rom char *)"    arm              (puts device into waiting mode)\r\n");
            putrsUSART((const far rom char *)"    pulse            (manual pulse output)\r\n");
            putrsUSART((const far rom char *)"OK\r\n");
            continue;
        }
        if ( !strncmppgm2ram((char *)text_buffer,(rom far char *)"set",3) ) {
            if ( !strncmppgm2ram((char *)&text_buffer[4],(rom far char *)"cvref",5) ) {
                v = atoi((char *)&text_buffer[10]);
                v = set_CVR_value(v);
                printf((const far rom char *)"setting cvref step to %d\r\n", v);
                printf((const far rom char *)"VREFCON2= 0x%x\r\n", VREFCON2);
            } else if ( !strncmppgm2ram((char *)&text_buffer[4],(rom far char *)"extra",5) ) {
                extra = atoi((char *)&text_buffer[10]);
                printf((const far rom char *)"setting extra (count) to %d\r\n", extra);
            } else {
                putrsUSART((const far rom char *)"\r\nUnknown option to set.\r\n");
```

```
            continue;
        }
        putrsUSART( (const far rom char *)"OK\r\n" );
        continue;
    }
    if ( !strncmppgm2ram((char *)text_buffer,(rom far char *)"arm",3) ) {
        display_settings();
        putrsUSART((const far rom char *)"Waiting for shock signal...\r\n");
        arm_and_wait();
        printf((const far rom char *)"time12= %d (decimal)\r\n", time12);
        printf((const far rom char *)"time23= %d (decimal)\r\n", time23);
        putrsUSART((const far rom char *)"\r\nOK\r\n");
        continue;
    }
    if ( !strncmppgm2ram((char *)text_buffer,(rom far char *)"pulse",5) ) {
        putrsUSART((const far rom char *)"Manual pulse output...\r\n");
        pulse_output();
        putrsUSART((const far rom char *)"\r\nOK\r\n");
        continue;
    }
    printf( (const far rom char *)"Unknown command: %s\r\n", text_buffer );
} // end while

// We actually don't ever expect to leave the while loop (until power-off).
CloseUSART(); // never get this far
} // end main
```

A second, smaller board has been made to simulate the shock signals. The left figure, below, shows the two simulated signals generated by this board. The right figure shows the amplified signal-2 plus positive-going edge of the digital output pulse.






```
// signal_generator.c
//
// Bare-bones signal generator for shock-speed timer.
// This simulates the 10 km/s shock passing the two
// upstream pressure sensors.
//
// PJ, 28-Sep-2010

#include "p18f14k22.h"
#include <delays.h>

#pragma config FOSC = IRC
#pragma config PLLEN = OFF
#pragma config FCMEN = ON
#pragma config IESO = OFF
#pragma config PWRTEN = ON
#pragma config BOREN = OFF
#pragma config WDTEN = OFF
#pragma config MCLRE = ON
#pragma config LVP = OFF
```

```
#pragma config XINST = OFF

// Clock frequency in Hz
#define F_OSC (16000000L)

#define OUT1 (LATBbits.LATB5)
#define OUT2 (LATBbits.LATB6)

void main( void )
{
    // Initialize hardware.
    // In the config bits above, we have selected the internal
    // high-speed oscillator, which defaults to 1MHz.
    OSCCON |= 0b01110000; // set IRFC<2:0> for 16MHz F_OSC
    ANSELHbits.ANS11 = 0; // enable digital buffer on RB5
    TRISB = 0b10011111; // RB5, RB6 output
    LATB = 0;
    Delay10KTCYx(2);

    // 50 microseconds = 200 instruction cycles @ F_OSC = 16 MHz
    OUT1 = 1;
    Delay10TCYx(20);
    OUT2 = 1;

    Delay10KTCYx(200);
    OUT1 = 0;
    OUT2 = 0;
    while ( 1 ) ; // hang around forever
}
```

References

[1] C.M. Jacobs. *Radiation in low density hypervelocity flows.* PhD thesis, The University of Queensland, Brisbane, and École Centrale, Paris., 2011.

[2] Microchip. PIC18(L)F1XK22 data sheet: 20-pin flash microcontrollers with nano watt XLP technology. Technical Report DS41365E, Microchip Technology Inc., www.microchip.com, 2011.