

---

## **AC 2012-4211: NEW LIFE FOR PROCESS CONTROL TRAINERS IN A MICROCONTROLLER COURSE**

**Dr. Dale H. Litwhiler P.E., Pennsylvania State University, Berks**

Dale H. Litwhiler is an Associate Professor at Penn State, Berks, in Reading, Penn. He received his B.S. from Penn State University, his M.S. from Syracuse University, and his Ph.D. from Lehigh University, all in electrical engineering. Prior to beginning his academic career, he worked with IBM Federal Systems and Lockheed Martin Commercial Space Systems as a Hardware and Software Design Engineer.

# New Life for Process Control Trainers in a Microcontroller Course

## Abstract

To improve student enthusiasm and demonstrate the power of embedded control, laboratory process control trainers were adapted and repurposed for use in an introductory microcontroller course. While some students are able to extrapolate the ideas conveyed by making an LED flash using assembly code, other students need to see their code doing something more powerful and realistic. Control system training rigs are common in engineering and technology laboratories. These trainers, from manufacturers such as Feedback®, typically have some type of “plant” that is the controllable center of the system. The parameters of the plant are then measurable via several types of process sensors. To complete the control loop, a programmable or otherwise adjustable control unit is provided to be used to demonstrate various closed-loop control techniques such as PID control. The system used here is the Feedback Basic Process Rig model 38-100. This system is comprised of a plumbing network in which the water level in a tank is regulated by controlling the flow of water into and out of the tank using solenoid and/or servo valves. Tank level and system flow rate sensors are used as inputs to the control unit. For the microcontroller course, the manufacturer’s control unit was disconnected from the system and replaced with a microcontroller trainer board and a simple, custom interface box to make the appropriate interconnections. The students designed their own assembly code to read the system sensors and control the water level to a variable, user-determined set point. The required code components were developed throughout the semester as each peripheral of the microcontroller was discussed. The process control served as a culminating project for the course. This paper presents the system hardware and example student software. Course curriculum is also presented and discussed.

## Introduction

Microcontrollers are used extensively in process control applications. Courses in process control often use microcontrollers to implement various control techniques.<sup>1</sup> Low-cost custom-built process control trainers can be utilized to demonstrate many processes.<sup>2,3</sup> The experience presented here, employs under-utilized educational-grade process control equipment to crossover as training tools for a microcontroller course. The emphasis here is on the ability to sense and control the process with the microcontroller rather than the control theory itself.

The laboratory space used for the microcontroller laboratory classes at (school name here) also serves as the control and automation laboratory. Contained within this laboratory, are several Basic Process Rig model 38-100 trainer stations manufactured by Feedback™. Figure 1 shows a photograph of one of the trainer stations. The trainer is comprised of rugged mechanical (plumbing) and electrical hardware that can easily be reconfigured for various water process control demonstrations and experiments.<sup>4</sup> The accessibility of the electrical connections between the sensors and actuators also makes it easy for system customization. In particular, custom interface hardware was developed to facilitate embedded control of the water process rig using the microcontroller trainer board already used in the microcontroller course. In this way, students

could develop and implement code for the microcontroller that had a very visible and active application.

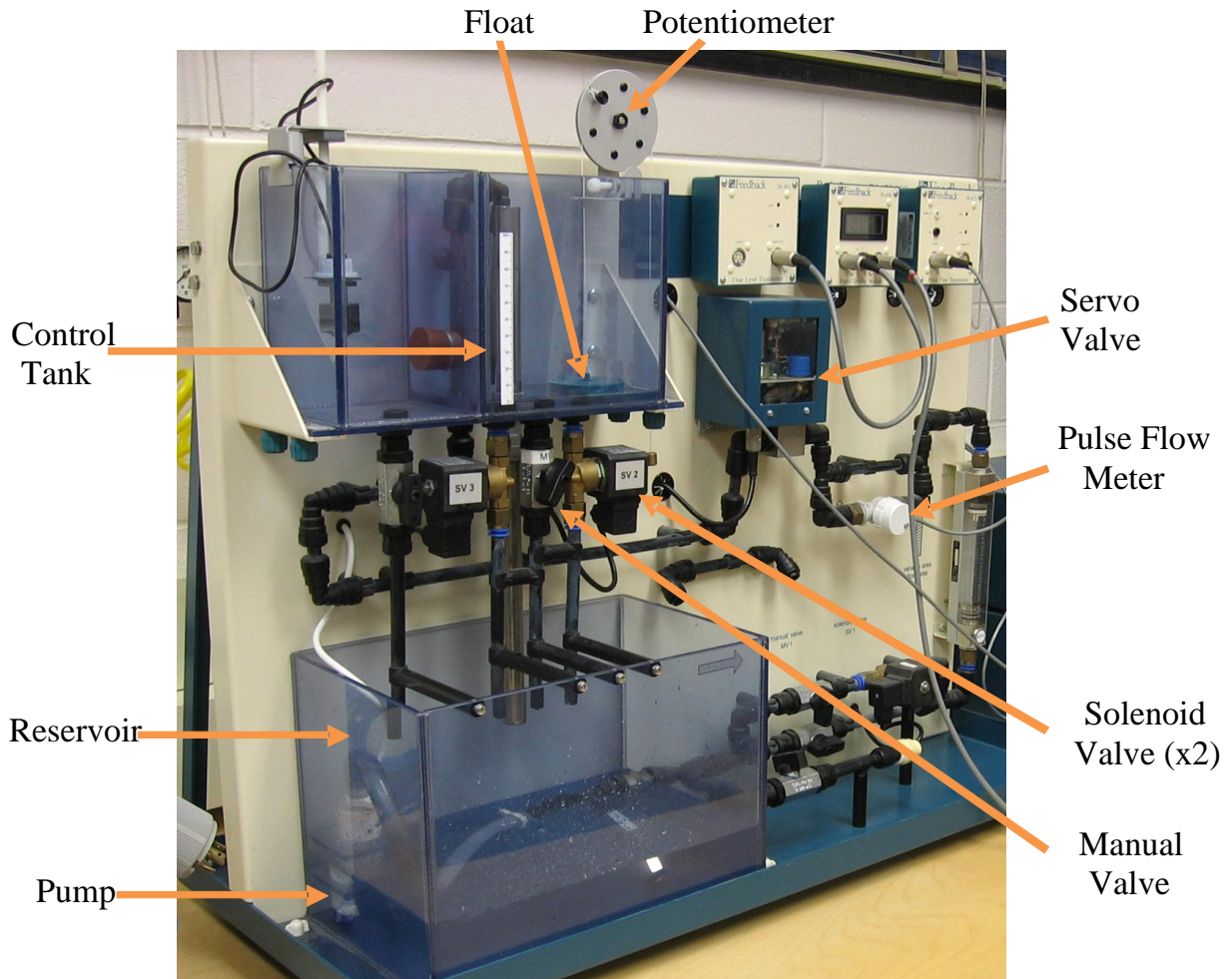


Figure 2 shows the block diagram for the water process trainer rig (shown inside the dashed box) with the interconnections to the microcontroller trainer board. All of the connections are accomplished with simple patch cords so no tools are required to substitute the microcontroller-based controller for the standard system controller. The system is then easily restored to its standard configuration at the end of each laboratory period. Because the lab space is shared, this was a very important consideration in the design of the interface unit.

The students use the microcontroller trainer board for each laboratory session and are very familiar with its use together with the MPLAB™ IDE and PICkit2™ programmer from Microchip.<sup>5</sup> Making the simple connections to the interface box was very similar to the type of wiring they had been doing in prior experiments. This was also a very important design consideration.

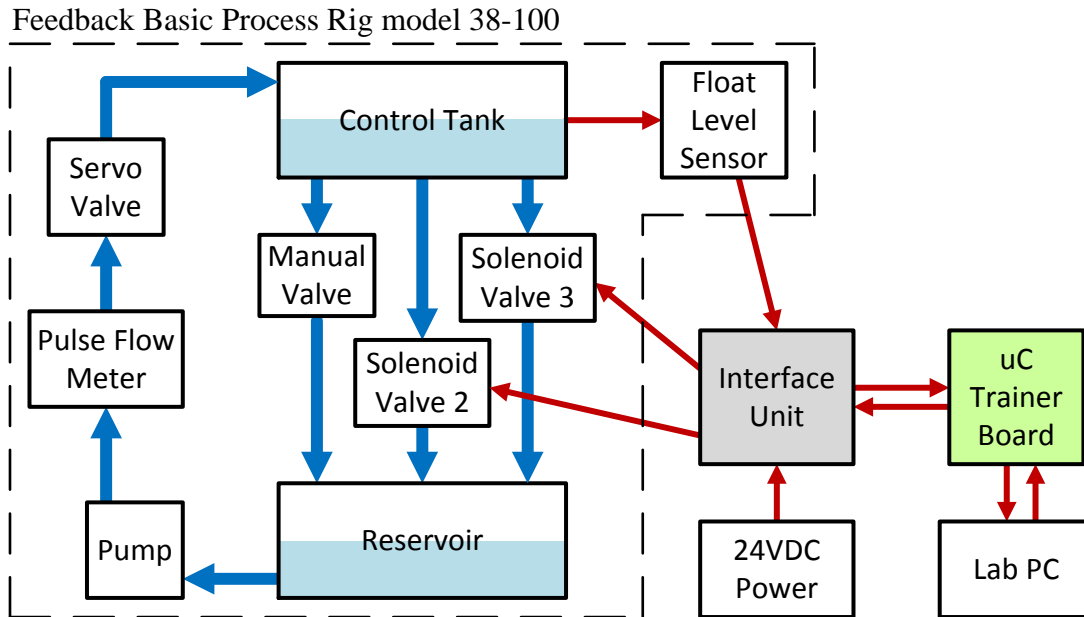


Figure 2. Modified System Block Diagram

## Course Curriculum

The microcontroller course is an introductory course taken by second or third year engineering technology students. The course uses the PIC™ 18F4520, 8-bit microcontroller as the primary device for demonstrating the capabilities of such devices. The devices are programmed using assembly language. The course topics are as follows:

- Arithmetic and logic commands and operations
- Decision branches
- Subroutines
- Time delays
- I/O port programming
- Serial (RS232) port communication and programming
- Timer programming
- A/D conversion
- Polling and interrupt handling
- SPI bus communication and programming
- Controlling high-power loads

The topics shown above are covered in detail with programming and laboratory exercises. Each of these concepts are key components in the assembly code developed by the students to control the water process. The water process control project ties together these concepts in a very tangible manner.

## System Overview: Hardware

The interface unit is shown in Figure 3 with and without its cover. The actual size of the plastic enclosure box is 2"x3"x1". The connections to the solenoid valves are made via 2-conductor patch cords with ¼" phone plugs that connect directly into the back of the process rig fixture. The control tank water level sensing potentiometer is connected to the interface box via a 3-conductor patch cord with a 5-pin DIN connector (2 unused pins). Connections to the microcontroller trainer are made with simple jumper wires that are inserted into the individual female connector pin sockets as needed. Figure 4 shows an assembly diagram of the interface unit.

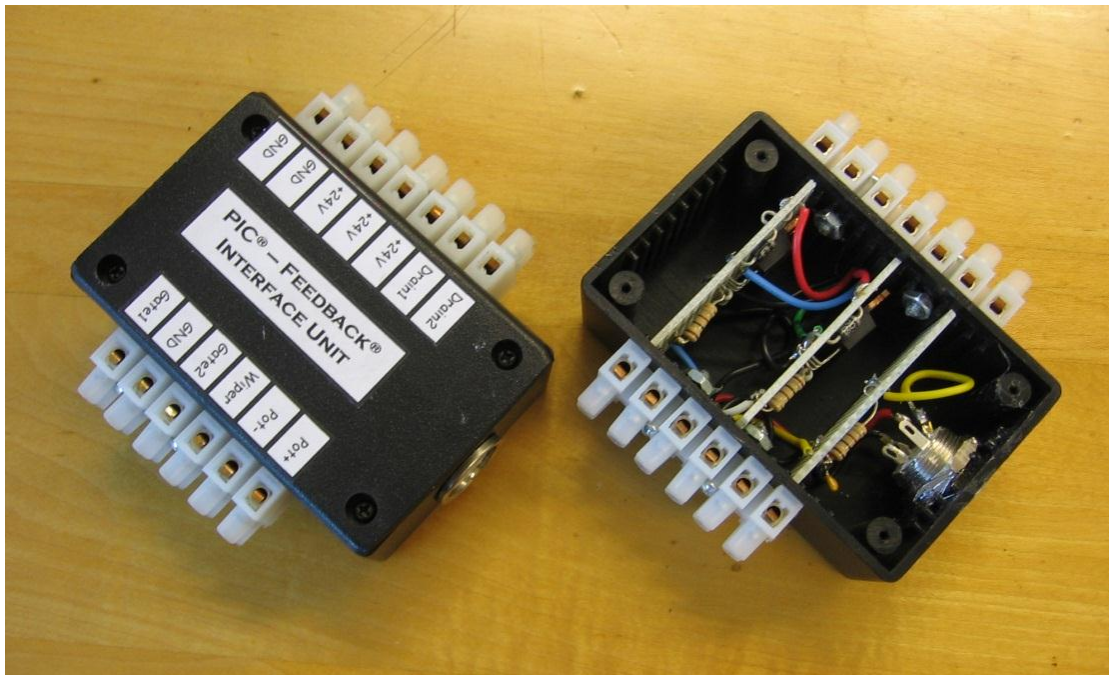


Figure 3. Interface unit

The solenoid valve requires a nominal current of 0.5A at 24V to hold it in the open position. When the solenoid is de-energized, the valve is closed by spring action. To control the high current of the solenoid valve, a MOSFET switch is used. Figure 5 shows a schematic of the MOSFET switch circuit contained in the interface unit.

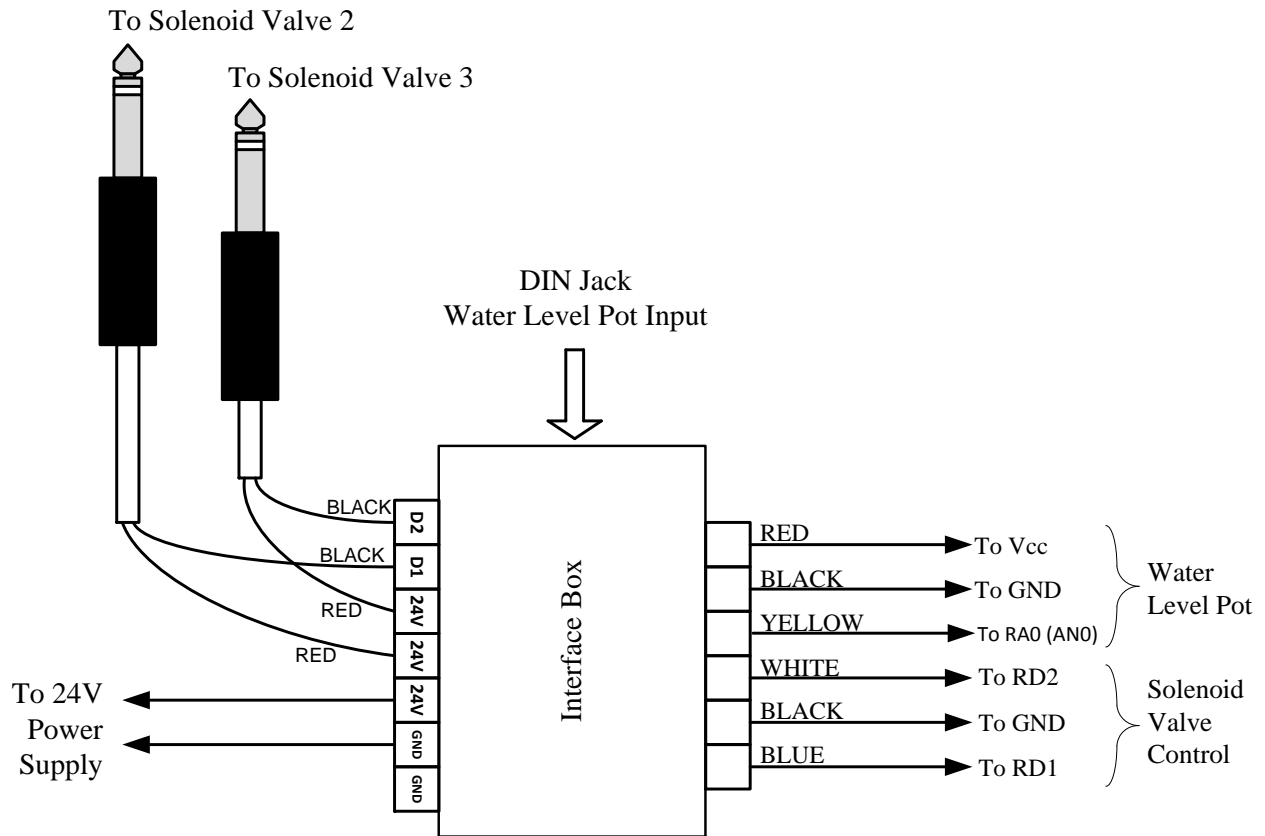


Figure 4. Interface unit connection diagram

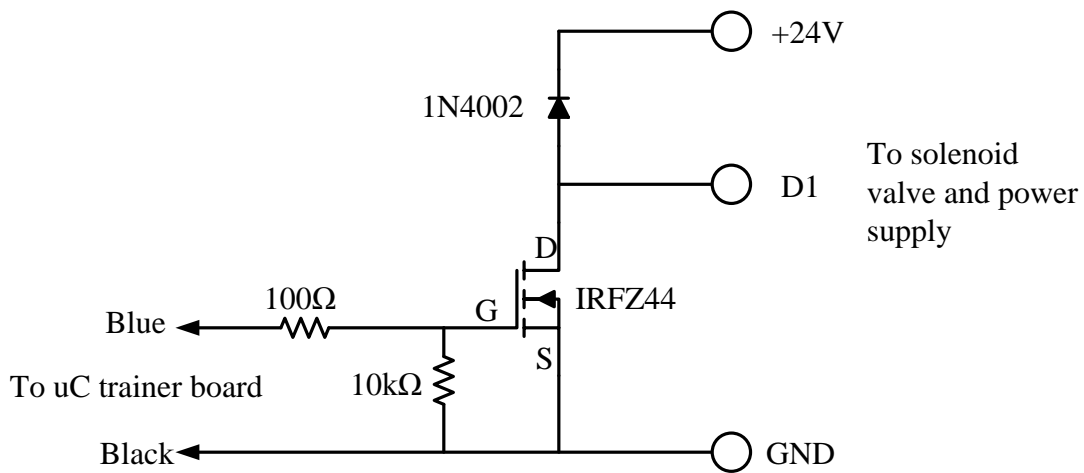


Figure 5. Solenoid valve MOSFET switch circuit

The control tank water level is measured with a float and 10kΩ potentiometer apparatus. An excitation voltage of 5VDC is passed from the microcontroller trainer board to the potentiometer

via the interface unit connections. The potentiometer wiper (output) voltage is noise-filtered with a simple RC low-pass filter circuit also housed in the interface box. The potentiometer connections are shown in Figure 6.

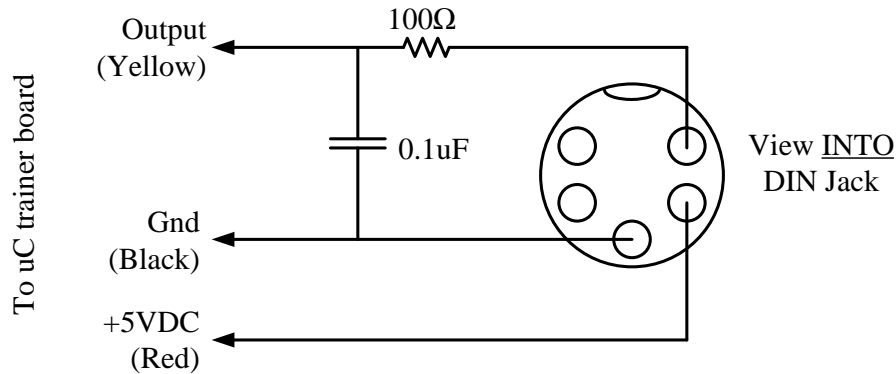


Figure 6. Water level sensor interface connector and signal filter circuit

The MDE PIC Trainer V2 microcontroller trainer board manufactured by MicroDigitalEd is shown in Figure 7.<sup>6</sup> This board provides easy access to all 40 pins of the PIC18F4520 microcontroller via male and female header connectors. The female headers are used to easily connect jumper wires to the interface unit. The board also contains some simple DIP switches, an LED driver IC and LEDs, and an RS232 interface/driver IC. In-circuit programming of the microcontroller is accomplished through a dedicated header designed to mate with the PICKit2 programmer. +5V power for the board can be obtained through the USB connection of the PICKet2 or through an external +5V supply (included with the trainer board). A photograph of the interface unit and microcontroller trainer connections is shown in Figure 8.



Figure 7. Microcontroller trainer board

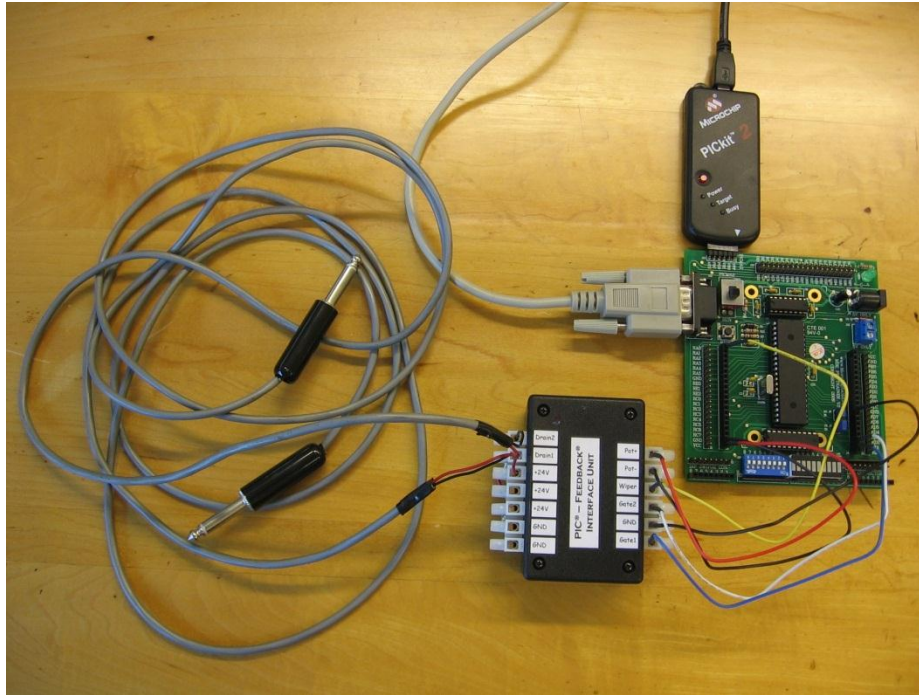


Figure 8. Interface unit and microcontroller trainer connections

### System Overview: Software

The tasks of the microcontroller software are to control the level of the control tank water level using hysteresis (bang-bang) control, periodically update the user with the current water level and be ready accept new desired water level set points from the user. The user interface is a LabVIEW application running on a PC that communicates with the microcontroller trainer board via RS232 serial link.

The user enters a desired water level as a percentage of tank fill (0 – 100). The output voltage of the water level sensor potentiometer is measured using the microcontroller's 10-bit A/D converter. To simplify the data manipulation, only the eight most significant bits of this measurement are used. This produces a water level value, referred to the potentiometer, ranging from 0 – 255.

The conversion between the percentage of tank fill (0-100) and potentiometer digitized voltage reading (0-255), and vice-versa, is accomplished using calibration tables stored in external EEPROM. The data for these calibration tables is obtained by the students for each trainer station. The EEPROM data is accessed by the microcontroller via the SPI communication bus. In this manner, the code developed by the students can be used with any of the water process training stations in the laboratory as long as the programmed EEPROM for that station is used.

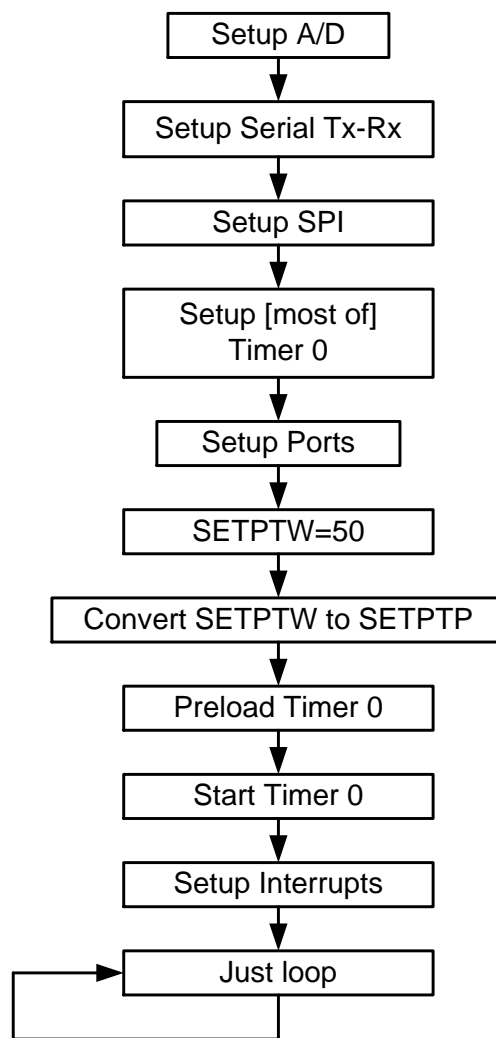
When the user inputs a desired tank fill level, the code retrieves the corresponding potentiometer digitized voltage value. The code then calculates a hysteretic dead-band by adding (upper limit) and subtracting (lower limit) 5 from the digitized voltage. When the water level is below the



lower limit, both solenoid drain valves are closed. When the water level is above the upper limit, one solenoid drain valve is opened. If the water level continues to rise to the value of 5 counts above the upper limit (10 counts above the set point), the second solenoid drain valve is opened.

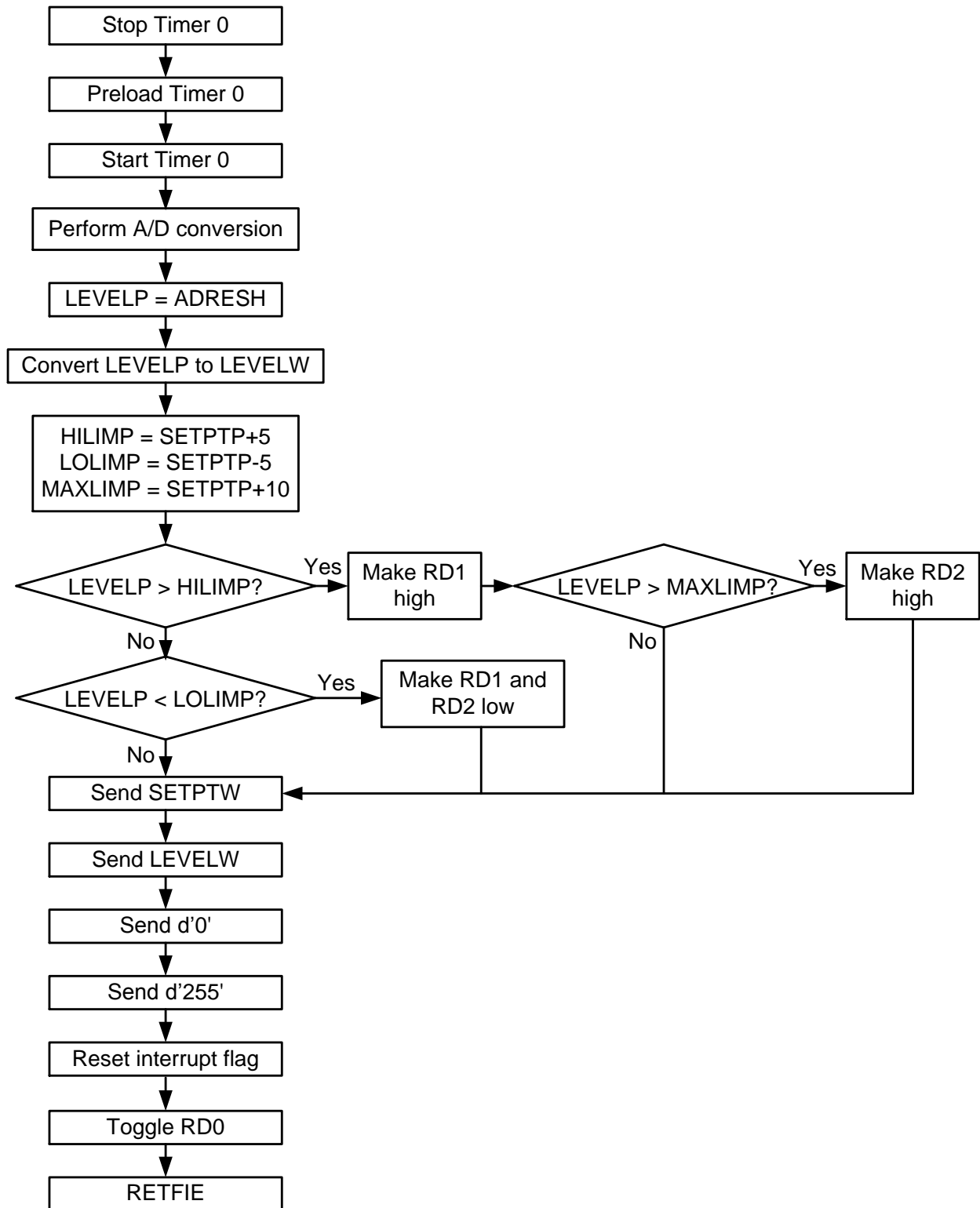
At one second intervals, as determined by one of the microcontroller timers, the current water level and current set point are sent to the user via the serial port. Prior to sending these values, they are first converted from digitized voltages to tank fill percentages using the EEPROM lookup table.

The flowchart from which the students design their assembly code is shown in Figure 9. The assembly code elements required to successfully implement the desired functions encompass most of the topics previously covered in the course. An example of the assembly code developed by the students is given in Appendix A.



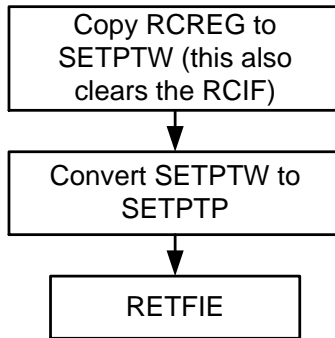
Main setup and loop

Figure 9. Water process control flowchart



Timer0 rollover (1s interval) interrupt code

Figure 9 continued. Water process control flowchart



Serial receive interrupt code

Figure 9 continued. Water process control flowchart

### Performance Results

During the last week of the semester, the students were required to demonstrate the performance of their water process control system. Several desired water levels were entered into the user interface. Some near the top of the control tank, others near the bottom. System performance data was collected and included in the students' final reports. Figure 10 shows a screen capture of the LabVIEW user interface which also displays the performance data for one of the water process control stations.

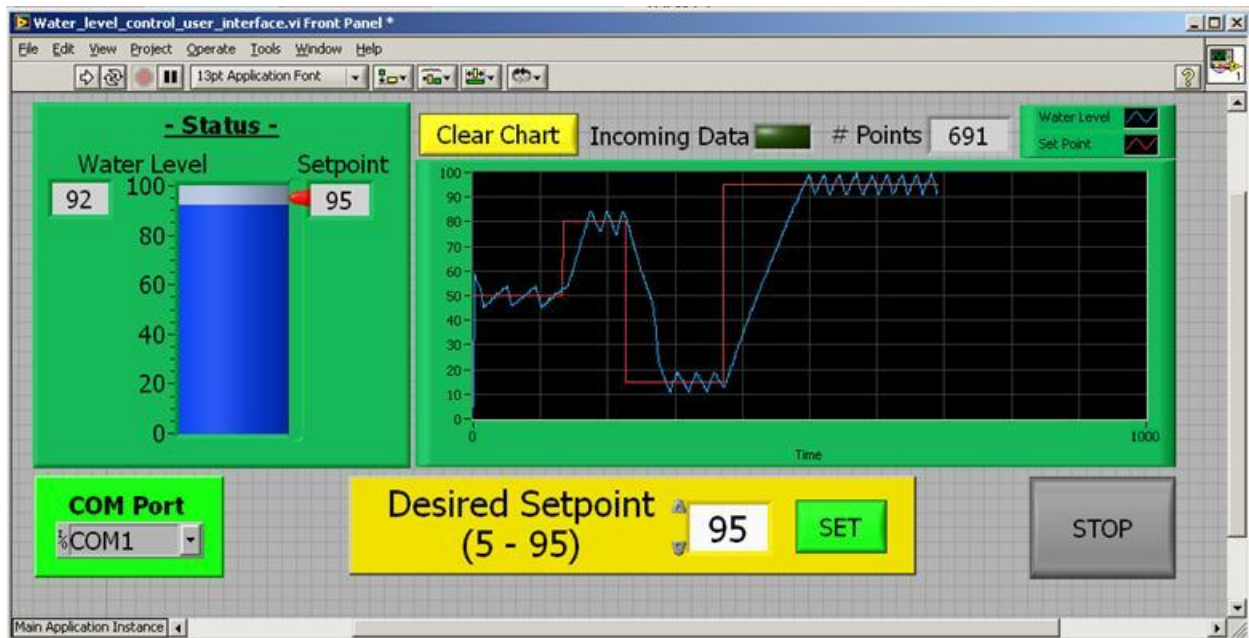


Figure 10. User interface and example system performance test data

## Conclusions and Outcomes

A simple and inexpensive custom interface unit allows for the insertion of a microcontroller trainer into an educational process trainer station. The interface unit facilitates this insertion in a quick and efficient manner without the need for any changes to the system hardwiring; only a few patch cords need to be switched. This allows the system to be quickly reconfigured for use by different courses.

Although this was not a control system course, the students were able to grasp the control ideas and work with the process control trainer equipment very effectively. Few technical problems were encountered that were unique to the process control stations. In the two years that this has been done, all of the student groups were able to achieve the objectives outlined in the project.

The use of flowing water, the whirring of the pump, the decisive thump of the solenoid valves opening and closing, and the physical size of the process trainer added a real-world industrial feel to the project. All members of the teams were actively engaged and involved in the debugging of their system's hardware and software.

## Acknowledgements

The author wishes to thank the electrical engineering laboratory supervisor, Mr. Jeff Wike, for his help with design ideas and the fabrication of the interface units.

## References

1. Lodge, K., "The programming of a micro-controller as the laboratory component in process control for undergraduates in chemical engineering," Proceedings of the American Society for Engineering Education Annual Conference and Exposition, 2006.
2. Rehg, J., "Low cost process control trainers," Proceedings of the American Society for Engineering Education Annual Conference and Exposition, 1998.
3. Moor, S., Piergiovanni, P., and Keyser, D., "Design – Build – Test: Flexible process control kits for the classroom," Proceedings of the American Society for Engineering Education Annual Conference and Exposition, 2003.
4. <http://www.feedback-group.com/product/level-and-flow-process-control-6060>.
5. <http://www.microchip.com>
6. <http://www.MicroDigitalEd.com>

## Appendix A

```
;assembly code file for the water process control project.
;
; - Water level is read from potentiometer on AN0 (0 - 255)
; - User sends desired water level via serial port (interrupt here) as
;   percent of tank fill(0 - 100)
; - Conversion between water level (pot) and desired are through a lookup
;   table in SPI EEPROM
; - Water level is controlled by opening one or both solenoid drain valves
;   (RD1 and RD2). There is a constant flow into the tank.
; - Cadence is set by 1 second rollover of Timer 0 (interrupt)
;   - Stop timer, preload timer, start timer
;   - Read water level
;   - Compare with desired water level.
;   - Make decision to open, close, or do nothing with the drain valves
;     based on water level and desired level.
;   - Send current water level, desired level, 0, 255 to PC via serial port
;   - Toggle LED on RD0 to show sign of life in software.
;   - Reset flags
; - Sit in "DumLoop" forever waiting for interrupts
;
;-----
;
;
#include P18F4520.inc
        CONFIG LVP = OFF, MCLRE = OFF, WDT = OFF, OSC = INTIO67, PBADEN = OFF

SETPTW      equ 0x00      ;setpoint referred to water level (0-100)
SETPTP      equ 0x01      ;setpoint referred to pot reading (0-255)
LEVELW      equ 0x02      ;current water level referred to water level (0-100)
LEVELP      equ 0x03      ;current water level referred to pot reading (0-255)
HILIMP      equ 0x04      ;upper water level limit referred to pot (0-255)
LOLIMP      equ 0x05      ;lower water level limit referred to pot (0-255)
MAXLIMP     equ 0x06      ;maximum water level limit referred to pot (0-255)
Couter      EQU 0x07      ;needed for DELAYMS subroutine
Cinner      EQU 0x08      ;needed for DELAYMS subroutine

org 0x00                                ;tells compiler where to start user code space
        movlw      b'01100011'
        movwf     OSCCON                  ;4MHz internal clock (2 LSB select the
; internal(11) or external(00))
        goto      Main                    ;jump over interrupt locations

org 0x08                                ;program flow jumps here when an
; interrupt occurs
        btfscc   INTCON, TMR0IF          ;check if TMR0 interrupt flag set
        goto     TMR0_ISR
        btfscc   PIR1, RCIF              ;check if received byte interrupt flag set
; flag is set
        goto     RC_ISR
        retfie                                ;just a ghost, I guess. Back to DumLoop.

Main
;Setup A/D ...(taken from A-D Conversion Lab Starter.asm)
```

```

MOVLW b'11000001' ;select AN0, NO Go, A/D ON
MOVWF ADCON0
MOVLW b'11001110' ;Vref=Vdd/Gnd, AN0=A, AN1-12=D
MOVWF ADCON1
MOVLW b'01100001' ;Left Justified, 8TAD, FOSC/8
MOVWF ADCON2

;setup serial port Tx-Rx...

MOVLW      B'00100100' ;enable transmit, choose high baud rate
MOVWF      TXSTA       ;write to register
MOVLW      D'25'       ;9600 baud with 4MHz clock, high setting
MOVWF      SPBRG       ;put it in Serial Port Baud Rate
                          ;Generator register
MOVLW      B'10010000' ;enable serial port, 8-bit, continuous
                          ;receive
MOVWF      RCSTA       ;write to register
BCF        TRISC, 6    ;make TX pin of PORTC an output pin
BSF        TRISC, 7    ;make RX pin of PORTC an input pin
BSF        PIE1, RCIE  ;enable interrupts from received byte
BSF        INTCON, PEIE ;enable peripheral interrupts

;setup SPI bus parameters including CSbar for EEPROM...

BSF        TRISC, SDI  ;make SDI pin of PORTC an input pin
BCF        TRISC, SDO  ;make SDO pin of PORTC an output pin
BCF        TRISC, SCK  ;make SCK pin of PORTC an output pin
BCF        TRISC, RC1  ;make RC1 pin an output pin, use as CSbar
BSF        PORTC, RC1  ;set RC1 so that it starts out high
MOVLW      b'01000000' ;need CKE=1 (bit 6) for proper data write
                          ;edge for EEPROM write

MOVWF      SSPSTAT
MOVLW      b'00100001' ;0,0,Enable SPI port,0,0,SPI Master with
                          ;clk = Fosc/16

MOVWF      SSPCON1

;Setup Timer 0...
MOVLW      b'00000011' ;TMR0 Off, 16-bit, internal clk,
                          ;prescaler ON, 1:16 prescaler

MOVWF      T0CON       ;
MOVLW      0x0B        ;with 1:16 prescaler, 62500 counts occur
                          ;in 1000ms. 65636 - 62500 = 3036 = 0x0BDC

MOVWF      TMR0H       ;load TMR0H with 0x0B
MOVLW      0xDC
MOVWF      TMR0L       ;load TMR0L with 0xDC
BSF        INTCON, TMR0IE ;enable Timer0 rollover interrupts

;setup remaining I/O pins...

BCF        TRISD, 0    ;make RD0 an output for blinking LED
BCF        TRISD, 1    ;make RD1 an output to control solenoid
                          ;valve 2
BCF        TRISD, 2    ;make RD2 an output to control solenoid
                          ;valve 3

;set initial water level set point to 50% and use lookup table in EEPROM to
;determine the corresponding SETPTP

```

```

movlw      d'50'
movwf     SETPTW

BCF       PORTC,RC1      ;make CSbar low   to select chip
movlw     b'00001011'   ;0,0,0,0, A8=1 for upper 256 byte area of
                        ;EEPROM, 011 (read operation) this is the
call      SPI           ;first SPI byte
movf      SETPTW,w      ;seond SPI byte indicates the location in
                        ;the upper 256 bytes to read from

call      SPI
call      SPI           ;read the value at that location (just
                        ;send any byte to force read from EEPROM)
BSF       PORTC,RC1     ;make CSbar high to de-select chip

movwf     SETPTP        ;copy the value read from EEPROM to the
                        ;setpoint-referred-to-pot register

;last few details and GIE...

BCF       INTCON, TMR0IF ;clear TMR0 overflow interrupt flag
BSF       T0CON, TMR0ON  ;start timer0 (could also use T0CON, 7)
BSF       INTCON, GIE    ;set the GIE bit to enable all
                        ;interrupts.

DumLoop
bra       DumLoop       ;something (nothing) to do

;Interrupt Service Routines...

TMR0_ISR  ;this is where almost everything happens...
;reload timer...
BCF       T0CON, TMR0ON  ;stop timer0 (could also use T0CON, 7)
MOVLW    0x0B           ;with 1:16 prescaler, 62500 counts occur
                        ;in 1000ms. 65636 - 62500 = 3036 = 0x0BDC
MOVWF    TMR0H          ;load TMR0H with 0x0B
MOVLW    0xDC           ;
MOVWF    TMR0L          ;load TMR0L with 0xDC
BSF      T0CON, TMR0ON  ;start timer0 (could also use T0CON, 7)

;perform A/D conversion...
BSF      ADCON0,1       ;start the conversion
ADLoop
BTFSC   ADCON0,1       ;wait for conversion to be done
goto    ADLoop
movff   ADRESH,LEVELP  ;copy [8-bit] A/D conversion result to
                        ;safe register

;convert LEVELP to a LEVELW value...

BCF      PORTC,RC1     ;make CSbar low   to select chip
movlw    b'00000011'   ;0,0,0,0, A8=0 for lower 256 byte area of
                        ;EEPROM, 011 (read operation) this is the
call     SPI           ;first SPI byte
movf     LEVELP,w      ;seond SPI byte indicates the location in
                        ;the lower 256 bytes to read value from

```

```

call        SPI
call        SPI                ;read the value at that location (just
                                ;send any byte to force read from EEPROM)
BSF         PORTC,RC1          ;make CSbar high to de-select chip

movwf      LEVELW              ;copy the value read from EEPROM to the
                                ;level-referred-to-water register

;calculate limits...
movlw     d'5'
addwf    SETPTP,w              ;add w TO f, put result in w
movwf    HILIMP                ;make upper limit 5 counts greater than
                                ;SETPTP

movlw     d'10'
addwf    SETPTP,w              ;add w TO f, put result in w
movwf    MAXLIMP              ;make maximum upper limit 10 counts
                                ;greater than SETPTP

movlw     d'5'
subwf    SETPTP,w              ;subtract w FROM f, put result in w
movwf    LOLIMP                ;make lower limit 5 counts less than
                                ;SETPTP

;compare water level with set points and turn valves on and off as needed...

HiTest
    movf   HILIMP,w
    cpfsgt LEVELP                ;is water level higher than upper limit?
    goto  LoTest
    BSF   PORTD,1                ;turn on MOSFET to open drain valve SV2

MaxTest
    movf   MAXLIMP,w
    cpfsgt LEVELP                ;is water level higher than maximum upper
                                ;limit?
    goto  LoTest
    BSF   PORTD,2                ;turn on MOSFET to open drain valve SV3

    goto  SendInfo

LoTest
    movf   LOLIMP,w
    cpfslt LEVELP                ;is water level lower than lower limit?
    goto  SendInfo
    BCF   PORTD,1                ;turn off MOSFET to close drain valve SV2
    BCF   PORTD,2                ;turn off MOSFET to close drain valve SV3

;transmit data to PC via serial port followed by two sync/framing bytes
;(0 and 255)

SendInfo
    movf   SETPTW,w
    call   SendIt
    movf   LEVELW,w
    call   SendIt
    movlw  d'0'                ;send ASCII null as part of data frame

```



```

call      SendIt
movlw    d'255'      ;send ASCII 255 as part of data frame
call      SendIt

;last few details before returning to DumLoop...

BTG      PORTD,0      ;toggle RD0 to blink LED to indicate the system
                    ; is alive
BCF      INTCON, TMR0IF ;clear TMR0 overflow interrupt flag
RETFIE

RC_ISR    ;serial byte received..

movff    RCREG,SETPTW ;copy received value to setpoint-
                    ;referred-to-water-level register (this
                    ;also clears the flag)

BCF      PORTC,RC1    ;make CSbar low to select chip
movlw    b'00001011' ;0,0,0,0, A8=1 for upper 256 byte area of
                    ;EEPROM, 011 (read operation) this is the
call     SPI           ;first SPI byte
movf     SETPTW,w      ;seond SPI byte indicates the location in the
                    ;upper 256 bytes from which to read value

call     SPI
call     SPI           ;read the value at that location (just send any
                    ;byte to force read from EEPROM)
BSF      PORTC,RC1    ;make CSbar high to de-select chip

movwf    SETPTP       ;copy the value read from EEPROM to the
                    ;setpoint-referred-to-pot register
RETFIE

;Subroutines...

SendIt
S1      BTFSS    PIR1, TXIF      ;wait until the last bit is sent
        BRA     S1              ;stay in loop
        MOVWF   TXREG
        RETURN

SPI     MOVWF    SSPBUF          ;copy contents of WREG to SPI buffer
SPILoop BTFSS    SSPSTAT,BF      ;wait for it to be sent (and/or something
                    ;to be fully received)
        BRA     SPILoop
        MOVF    SSPBUF,W        ;copy received byte to WREG before
                    ;returning
        RETURN

DELAYMS ;load desired millisecond delay into WREG before calling
        ;this (at 4MHz).
movwf   Couter

OutLoop
movlw   d'249'
movwf   Cinner

InLoop  nop

```

```
    decf    Cinner  
    bnz    InLoop  
    decf    Couter  
    bnz    OutLoop  
    return
```

END