# USS Tanglewire: An experiment in networking for an embedded computer applications class

**C. E. Wick, K. A. Knowles**
**United States Naval Academy**

Abstract

Small embedded microprocessors are used routinely as controllers in many types of consumer, industrial and military electronic devices. In many cases more than one of these small computers are used in a product to affect distributed control, or to localize specific system functions. This tends to modularize the system and helps to make it more reliable, survivable, configurable and upgradable. Networks provide the "glue" that connect each of the localized modules together into a functioning whole. We believe that students who take courses in microprocessor embedded control should have some exposure to network-connected control systems, and if possible they should also have experience in their implementation. This paper describes a project that we undertook at the U.S. Naval Academy in our computer engineering track where our students used an $I^2C$ network and PIC16C84 microprocessors to construct a model distributed shipboard damage control system.

## 1. Introduction

The Weapons and Systems Engineering Department at the United States Naval Academy offers an ABET accredited Systems Engineering degree to about one hundred students annually. As part of their studies our students may concentrate in two or more areas as tracks during their senior year. Included in the tracks offered to our students is one that offers specialized studies in computer engineering. In this track our students take a course in digital fundamentals, followed by a course in embedded microprocessor applications. In this second course we introduce microprocessor architecture, assembly language, and apply embedded microprocessors to various applications with a focus on their use in controls. For the past several years we have used PIC16C71 and PIC16C84 microcontrollers as the target for these applications because of their speed, versatility, and ease of programming. An in-house simulation and programming system, *PICSIM*, has been used successfully to teach the architecture of the processors, assembly language programming, and debugging techniques. The laboratory and final exam projects for this course vary from year to year, but have historically been examples of stand-alone microprocessor based systems.

In order to appeal to the professional aspirations of our student population we began a study of potential microprocessor based projects that could be seen to directly relate to their chosen careers. A particularly fertile area was seen to be in the use of networked embedded computers in modern weapons systems. Among current military projects in the area is on-going research

by the United States Navy to develop sensor-based local area network systems that will enhance the efficiency and survivability of its ships, while at the same time reducing the manpower needed to operate them. These so-called "Smart Ships" will rely on high-bandwidth networks and local processors to affect rapid determination of problems, and to permit autonomous or remote remediation of the problem. We decided to tackle this problem in small scale in our class laboratory with a stylized ship model and a networked system consisting of multiple PIC processors.

## 2. USS Tanglewire model

We began by constructing a four by eight foot plywood ship model frame, which was named USS Tanglewire, that served as a platform to hold student designed and constructed prototype damage control systems. The infrastructure consisted of a ship-outline platform that was separated into four two-foot cube compartments by plywood walls. The compartments were arranged two to a side. Each section of the course was assigned a compartment area for their work. Two additional areas, one in the bow and one in the stern were used for auxillary functions. The ship platform was raised about six inches by cross members that were cored so that major wire runs and power supplies would be outside of the compartment areas.

Students in a section were responsible for designing, building and interfacing damage control modules for their assigned model ship compartment. Their control modules were to sense some undesirable condition in the compartment (smoke, excessive heat, flooding or loss of power) and provide some remedy by an actuator or actuators under program and network control. Each damage control module was to report its status via a simple network back to a central control station at the ship's bridge. Additionally, each damage control module was to be capable of remediating the condition autonomously, under local control (a switch), or remotely from the bridge by the network. The compartments were pre-wired with a three wire network bus, a five volt power bus for their systems, and two twelve-volt busses for sensing power failure and circuit switching. Each compartments was also provided with a large metal pan to allow for testing systems under simulated damage conditions.

## 3. Damage control systems

In our first attempt at this project we were uncertain about its eventual complexity, so we asked student teams in each section to select from four sensor - actuator pairs as a foundation for their embedded computer damage control system. The instructors had pre-tested the sensors/actuators so we knew their properties and we knew that they were readily available. Additional volunteer student teams from some of the larger sections worked on bridge control systems that would provide a LCD display of status information from each compartment system and would provide overall system control. The sensor-actuator pairs for this first implementation were (condition sensor and remediating actuator):

A. A temperature sensor (sense the presence of fire) and a sprinkler pump.
B. A water level sensor (sense flooding) and a water pump.
C. A smoke sensor (sense fumes) and a fan.
D. A voltage sensor (condition of power busses) and a circuit switching relay.

Section teams were provided with analog temperature sensors (Analog Devices AD22100), magnetic switch level sensors, modified smoke detectors, various small pumps and fans. Microprocessors and other parts to complete their design were provided by the department if they were in existing stocks.

Other design conditions given to the student teams were: 1) That each damage control system was to be designed with a PIC16C84 microcontroller or with a PIC16C71 microcontroller, when analog-to-digital converters were needed and the sensors and actuators were to be controlled by available ports on the device; and 2) The students were to use locally written PIC processor software routines that would make their controller a slave unit on an $I^2C$ network ( a personal computer was used as the master controller for the network).

Along with the pre-tested slave $I^2C$ routines, a Windows -based PC was also supplied with locally written network routines that could poll the processors connected to the network, read status information and write commands to each polled network address. The $I^2C$ protocol was chosen because of its electrical simplicity, its ability to address multiple units, and because it represents a system that is in current use for communication with computers and devices in small embedded systems.

## 4. $I^2C$ network protocol and subsystems

The $I^2C$ network protocol was developed by the Phillips corporation as a simple method for affecting control of multiple large-scale integrated circuits[1]. Microchip, the manufacturer of PIC computers has provided some additional implementation details for their architecture[2]. Although it is a mechanically simple network, it is nonetheless an effective tool for teaching about connected embedded systems. An $I^2C$ serial network consists of (typically) a single master controller and multiple slave controllers that share a three-wire bus. Two active bus wires (clock and data ) are pulled up to a convenient voltage (+5 volts). The third wire provides a ground return. The master control produces the clock signal for the network by pulling the clock wire down. Each master and slave controller pull the data bus wire down to pass data according to network protocol.

The $I^2C$ protocol is also quite simple. Transmissions are started, synchronized and stopped by the master controller. Message start and stop conditions are signaled by transitions of the data bus line when the clock line is high. Changes in data can occur only when the clock line is low. Slave units separate data from framing commands by checking the state of the clock line when a data line change is observed. Each transmission from the master station starts with a seven-bit address field, which allows for independent addressing of up to 128 slave controllers or sub-functions on a slave controller. The address is followed by a read/write bit, which controls the direction of transmission, and an acknowledge bit, which the slave controller forces low to acknowledge receipt. The address is followed by one or more bytes of data, each byte having an extra bit position for acknowledging receipt. In our implementation the address header generated by the master processor was followed by one byte of data to, or a request of one byte of data from the addressed slave processor.

PIC16C84 and PIC16C71 processors were interfaced to the network through interrupt capable pins (these processors do not have intrinsic $I^2C$ capabilities). Interrupts were generated on clock and on data edges, which are important timing marks in this type of network. The PC master station was interfaced to the network through two output bits and two input bits of its printer port. The output bits of the PC were connected to simple NPN transistor switches that pulled high bus lines low. The input pins were used to sense the actual state of the data lines. With this configuration it is possible to have multiple master stations, although none were used in this project.

The $I^2C$ software in the PIC processor was implemented as an assembly language interrupt service routine. This routine uses twelve processor registers. The user interface is through four registers; one register to hold the address of messages being passed on the bus, a register to hold inbound data that is stored when the bus address matches the unique address of an individual controller, and a register to hold outbound data when a read is requested by the slave processor. The final register holds bit flags that indicate when data has been placed in the inbound register and when data has been taken from the outbound register by the master processor. An initialization routine for setting up the network variables and interrupt routine was provided as part of the software package. The software for this package was isolated as much as possible from any other software that may be applied to the PIC package by students. These assembly language routines were "included" in a program template file that was provided to the students for use in their software designs where they added and assembled their control application code (see Appendix B). Byte message protocols were established for the data portion of an inbound or outbound transmission from the master controller to a slave processor. The outbound message contained bit fields that could cause the damage control system to switch between remote, automatic and local operation, and to command the actuator on and off. Inbound messages to the master controller contained information about the current mode of the controller, sensor and actuator status. Unused bit fields were available for compartment or system unique information. Details on our PIC16C84/71 implementation may be found in Appendix A.

PC software was written for testing and control of the system was written as a graphical user interface in Windows with Borland Builder C programming software. A virtual device driver was also written to allow for direct pin control of the printer port. The software allowed for polling all possible addresses and looking for responses to determine net addresses that were active. A one byte message could be sent to any or all of the active addresses and the responses to the messages were kept in a list. In this way students could send activation commands to their systems and they could see the results at their compartments and the status information that was returned from their embedded processors.

## 5. Student designs

Students were given a set of abbreviated specifications for their damage control systems and the network implementation. The network specification required that they provide a unique network address for their system and that they use pins B0 and B7 of their processor for interfacing to the network. Their system was to detect a particular damage condition and operate a remediating

actuator under three possible modes:

A. Local mode, where the station presents status information when commanded, but corrective action occurs only when a local switch is thrown.

B. Automatic mode, where the station initiates correction activity upon sensing a damage condition.

C. Global mode, where the station reports status and operates its actuator under control of a bridge station.

Students were free to decide upon how to best process their sensor information, and when, and how to affect action when required.

## 6. Lessons learned and conclusions

This exercise was very effective in teaching concepts in embedded computing and networking and we will continue to use this project in this course during the coming year, but there were also some important lessons that were learned from the first year that will undoubtedly improve our subsequent efforts:

A. It is very easy in this network for a malfunctioning station, or a loose wire in one compartment to disable the network by pulling a clock or data line low. Our PC software could detect these faults by monitoring the clock and data lines, but we had trouble isolating the unit that was causing the difficulty. We will provide switched isolation at each compartment to allow us to better isolate faults.

B. The network bus lines must have protection against accidental contact with power supplies. We lost several PIC processors and a printer port because of accidental miss-wiring and sloppy construction practices. This problem will be corrected by applying fuses and zener diodes to the network data and clock lines in each compartment. This additional protection will be added to the positive disconnect switch discussed above.

C. Although the ship model was large, it still became crowded when several sections were working in the laboratory after hours. In a new design of the ship model we will make the individual compartments removable boxes so that work can be accomplished away from the basic ship platform.

Our students became very interested in this project because they could see the application of networked embedded processors in their careers. In spite of the difficulties mentioned above, most teams designed and built systems that operated partially to requirements and we had several teams build systems that completely fulfilled all design parameters. We can see that the modifications discussed above will increase the number of student completions significantly. This was a very worthwhile project having much educational value, and we encourage others to try this idea with this, or other scenarios.

Bibliography

1. The $I^2C$-bus and how to use it, Phillips Semiconductor Corp, April 1995. This document is still available at several Internet locations, see for example, http://www.mcc-us.com/i2chowto.htm

2. Communicating with the $I^2C$ Bus Using the PIC16C5X, Microchip Technology application note AN515, 1993,

available from URL: http://www.microchip.com/10/Appnote/Category/16C5X/index.htm

KENNETH KNOWLES
Kenneth Knowles is a Professor of Systems Engineering with the Weapons and Systems Engineering Department at the U.S. Naval Academy. Dr. Knowles received his BME,MME and Ph.D. degrees from the University of Virginia. His research interests are in machine vision and robotic system applications. He supports the NASA Hubble Space Telescope periodic servicing missions as Goddard Space Flight lead for the EVA support team.

CARL WICK
Carl Wick is an Associate Professor of Systems Engineering with the Weapons and Systems Engineering Department at the U.S. Naval Academy. Dr. Wick received a B.S. degree from the U.S. Naval Academy, A M.S. degree from the Naval Postgraduate School, and the D.Sc. degree from The George Washington University. His research interests are in embedded computer control systems, vision, and other sensor systems.

Appendix A.
PIC16F84/PIC16C71 Assembly language implementation of $I^2C$ slave processor

```
;/* These registers are private to the network routines */
FLAGS   equ     20h
startf  equ     0       ;start = 0
addrf   equ     1       ;address = 1
ackf    equ     2       ;acknowledge = 2
dtaf    equ     3       ;data buffer full = 3
rw      equ     4       ;read/write acces = 4
vaddr   equ     5       ;valid address = 5
ackset  equ     6       ;acknowledge set
oldb7   equ     7       ;last bit value for rb7
SNCTR   equ     1fh
INADR   equ     1eh
INDTA   equ     1dh
DDTA    equ     1ch
CNTR    equ     1bh
SAVEW   equ     1ah     ;save w context*
SAVES   equ     19h     ;save status context*
SAVEF   equ     18h     ;save fsr context*

;/*  User has access to these */
DADDR   equ     17h     ;display address *
OUTDTA  equ     16h     ;user output data*
DDTAI   equ     15h     ;user input data*
FLAGS1  equ     14h
DTARDY  equ     0       ;data is ready for pickup

; USER MUST SUPPLY MATCH ADDRESS MATCH = ADDRESS*2

sn_isr          movwf   SAVEW           ;save context, W register
                swapf   STATUS,W        ;save context, Status register
                movwf   SAVES
                movf    FSR,W           ;save context, FSR register
                movwf   SAVEF
```

```
        btfsc   INTCON,INTF
        goto    snisrd          ;data change
        btfsc   INTCON,RBIF
        goto    snisrc          ;clock change
        call    user_int
        goto    isr_end

snisrc  movf    FLAGS,W         ;see if any change since last value
        xorwf   PORTB,W         ;could be some other bit that changed
        andlw   80h
        btfsc   STATUS,Z
        goto    isr_end         ;some other bit

        ; Interrupt due to clock change

        btfss   PORTB,7         ;lo to hi transitions here
        goto    ackchk          ;hi to lo transitions
        btfss   FLAGS,startf    ;go only if start seen
        goto    isr_end
        decfsz  SNCTR,F         ;see if time for acknowledge
        goto    nicdta          ;otherwise collect data
        movlw   9
        movwf   SNCTR           ;reload with nine
        btfss   FLAGS,ackf
        goto    isr_end
        movlw   TRISB           ;change direction of port B0
        movwf   FSR
        bcf     PORTB,0         ;data bit to zero
        bcf     INDIR,0         ;change direction to output
        bcf     FLAGS,ackf
        bsf     FLAGS,ackset
        goto    isr_end

        ;Turn off acknowledge if on at next clock edge
ackchk  btfss   FLAGS,ackset
        goto    outchk
        bsf     PORTB,0         ;preset port b
        bcf     FLAGS,ackset    ;end of acknowledge
        btfsc   FLAGS,rw        ;change port dir if write
        goto    outchk
        movlw   TRISB
        movwf   FSR
        bsf     INDIR,0         ;change direction to input
        goto    isr_end

outchk  ; Send out data when read is requested
        btfss   FLAGS,startf    ;must have start flag
        goto    isr_end
        btfss   FLAGS,vaddr     ;must have valid address
        goto    isr_end
        btfsc   FLAGS,ackf      ;must have started acknowledge
        goto    isr_end
        btfss   FLAGS,rw        ;must be write request
        goto    isr_end
        movlw   1
        subwf   SNCTR,w
```

```
            btfsc   STATUS,Z
            goto    isr_end
            rlf     DDTA,F
            btfss   STATUS,C
            goto    zeroout
            bsf     PORTB,0
            goto    isr_end
zeroout     bcf     PORTB,0
            goto    isr_end
            ; Collect address and data at each up clock transition
nicdta
            bcf     STATUS,C        ;preclear carry
            btfsc   PORTB,0         ;check data pin
            bsf     STATUS,C
            btfss   FLAGS,addrf     ;see if collecting address or data
            goto    nicadr
            btfsc   FLAGS,rw        ;see if request to read or to write
            goto    isr_end         ;read handled by other routine
            btfss   FLAGS,vaddr     ;see if valid address
            goto    isr_end
            btfsc   FLAGS,dtaf      ;see if data already collected
            goto    isr_end
            rlf     INDTA,F
            movlw   1
            subwf   SNCTR,w
            btfss   STATUS,Z
            goto    isr_end
            bsf     FLAGS,ackf      ;acknowledge
            bsf     FLAGS,dtaf      ;good data
            goto    isr_end

nicadr
            rlf     INADR,F
            movlw   1
            subwf   SNCTR,w
            btfss   STATUS,Z
            goto    isr_end
            bsf     FLAGS,addrf     ;address portion done
            movf    INADR,W         ;check for valid address
            movwf   DADDR           ;save complete address
            andlw   0f8h
            movwf   INADR
            movlw   MATCH
            subwf   INADR,W
            btfss   STATUS,Z
            goto    isr_end
            bsf     FLAGS,vaddr     ;valid address
            bsf     FLAGS,ackf      ;acknowledge good address
            bcf     FLAGS,rw
            btfsc   DADDR,0         ;see if read or write access
            bsf     FLAGS,rw
            goto    isr_end

            ;Determine Start and Stop Transitions by watching data
snisrd      bcf     INTCON,INTF     ;reset data change flag
            btfss   PORTB,7         ;see if data change while clock hi
```

```
        goto    isr_end         ;no change while high, ignore
        btfsc   PORTB,0         ;see if start transition
        goto    snstop
        bsf     FLAGS,startf    ;set start flag
        bcf     FLAGS,rw        ;force a read for address
        bcf     FLAGS,ackf      ;reset acknowledge flags
        bcf     FLAGS,ackset
        bcf     FLAGS,dtaf      ;clear data flag
        bcf     FLAGS,vaddr     ;clear valid id
        bcf     FLAGS1,DTARDY   ;clear data ready flag
        movlw   9
        movwf   SNCTR           ;nine bits until end
        clrf    INADR
        clrf    INDTA
        movf    OUTDTA,W
        movwf   DDTA            ;copy data buffer in case of output
        movlw   OPTN            ;change int direction lo to hi
        movwf   FSR
        bsf     INDIR,6
        goto    isr_end

snstop  bcf     FLAGS,startf    ;reset start flag
        bcf     FLAGS,addrf     ;reset address flag
        btfss   FLAGS,vaddr     ;see if this is a valid id
        goto    snstop1         ;if not don't adjust flags
        bcf     FLAGS,vaddr     ;reset valid address
        btfsc   FLAGS,rw        ;see if last operation was write
        goto    snstop1         ;was a read
        movf    INDTA,W         ;copy good data over
        movwf   DDTAI
        bsf     FLAGS1,DTARDY   ;data is ready for pickup
snstop1 movlw   OPTN            ;change int direction hi to lo
        movwf   FSR
        bcf     INDIR,6
        movlw   TRISB           ;ensure direction is input
        movwf   FSR
        bsf     INDIR,0
        goto    isr_end

        ;Reset flags and restore context
isr_end bcf     FLAGS,7
        btfsc   PORTB,7
        bsf     FLAGS,7
        bcf     INTCON,RBIF
        bcf     INTCON,INTF
        movf    SAVEF,W         ;restore fsr
        movwf   FSR
        swapf   SAVES,W         ;restore context
        movwf   STATUS
        swapf   SAVEW,F
        swapf   SAVEW,W
        retfie

init_snet movlw TRISB
        movwf   FSR
        bsf     INDIR,7         ;bit B7 is input to start (clock)
```

```
        bsf     INDIR,0         ;bit B0 is input to start (data)
        movlw   OPTN
        movwf   FSR
        bcf     INDIR,6         ;interrupt on hi to low data to start
        clrf    FLAGS           ;preclear
isn1      btfss   PORTB,7
        goto    isn1
isn2      btfss   PORTB,0
        goto    isn2
        movlw   98h             ;gie + inte+rbie to start
        movwf   INTCON
        return
```

Appendix B.
Program template for use in network controlled programs

```
;       Template for s-net pic programs in Tanglewire project
;       The minimum requirements are listed in BOLD type

MATCH       equ     40*2    ;match address = your specific address *2
                            ;this program will match an address of 40

; put variable definitions here.  You can use registers 12-19 for your functions in a PIC16C84
; you can use additional registers if you do not include math or LCD routines

        org 0
        goto start
        org 4
        include "snet_int.asm"  ; interrupt routine and initialization routine found in Appendix A
user_int    return              ; can place a user interrupt routine here, finish with a return, not retfie

start
        ;initialize TRIS registers as necessary
        call init_snet          ;snet will change TRISB bits 0,7 and will change OPTIONS register.

loop    ;operational program loop
        ; move status information into W register
        movwf   OUTDTA          ;this variable holds outgoing snet data

chkindata
        btfss   FLAGS1,DTARDY   ;this flag is set when incoming data has arrived
        goto    chkindata
        bcf     FLAGS1,DTARDY   ;clear the flag in your program
        movf    DDTAI,W         ;get the input command data from the network

        ;perform operations as reqired with the data
        goto    loop
```