AN214

The PICmicro® MCU as an IEEE 1451.2 Compatible Smart Transducer Interface Module (STIM)

Authors: *Richard L. Fischer*
Microchip Technology Inc.

Jeff Burch
Agilent Technologies

INTRODUCTION

This application note discusses the organization of the PICmicro MCU and related firmware for developing IEEE 1451.2 Smart Transducer Interface Modules (STIM). Although a detailed understanding of the 1451.2 is not required, a working knowledge of 1451.2 concepts like Transducer Electronic Data Sheet (TEDS), STIMs, data models, etc., is assumed. The purpose of this application note is to provide the developers of a PICmicro MCU based STIM with some key hardware and firmware building blocks, that may be utilized in the development of a STIM.

IEEE-1451.2 OVERVIEW

The IEEE 1451 standard is composed of four parts, 1451.1, 1451.2, 1451.3 and 1451.4. The combination of the four parts (some parts are approved and others are pending) define the signal chain from the analog sensor to the digital network (See Figure 1). The main objectives of the IEEE 1451 standard are to:

- Enable plug and play at the transducer level (sensor or actuator) by providing a common communication interface for transducers.
- Enable and simplify the creation of networked smart transducers.
- Facilitate the support of multiple networks.

The focus of this application note deals with the 1451.2 part of the standard.

The IEEE 1451.2 standard provides the means to connect sensors and actuators to a digital system, typically a network, via a Network Capable Application Processor (NCAP). Figure 2 depicts a general distributed measurement and control application and Figure 3 depicts the functional boundaries relating to the transducer interface standard, namely IEEE 1451.2.

In simplest terms, the IEEE-1451.2 standard does the following:

- Defines a digital interface for connecting transducers to microprocessors.
- Describes a TEDS, “electronic data sheet”, and its data formats.
- Defines an electrical interface, read and write logic functions to access the TEDS, and a wide variety of transducers.

This standard does not specify signal conditioning, signal conversion, or how the TEDS data is used in applications.

There is currently no defined independent digital communication interface between transducers and microprocessors. Each vendor builds its own interface. Without an independent, openly defined interface, transducer interfacing and integration are time consuming and duplicated efforts by vendors are economically unproductive. This interface provides a minimum implementation subset that allows self-identification and configuration of sensors and actuators, and also allows extensibility by vendors to provide growth and product differentiation.

FIGURE 1: IEEE 1451 DEFINES THE SIGNAL CHAIN FROM ANALOG SENSOR TO DIGITAL NETWORK

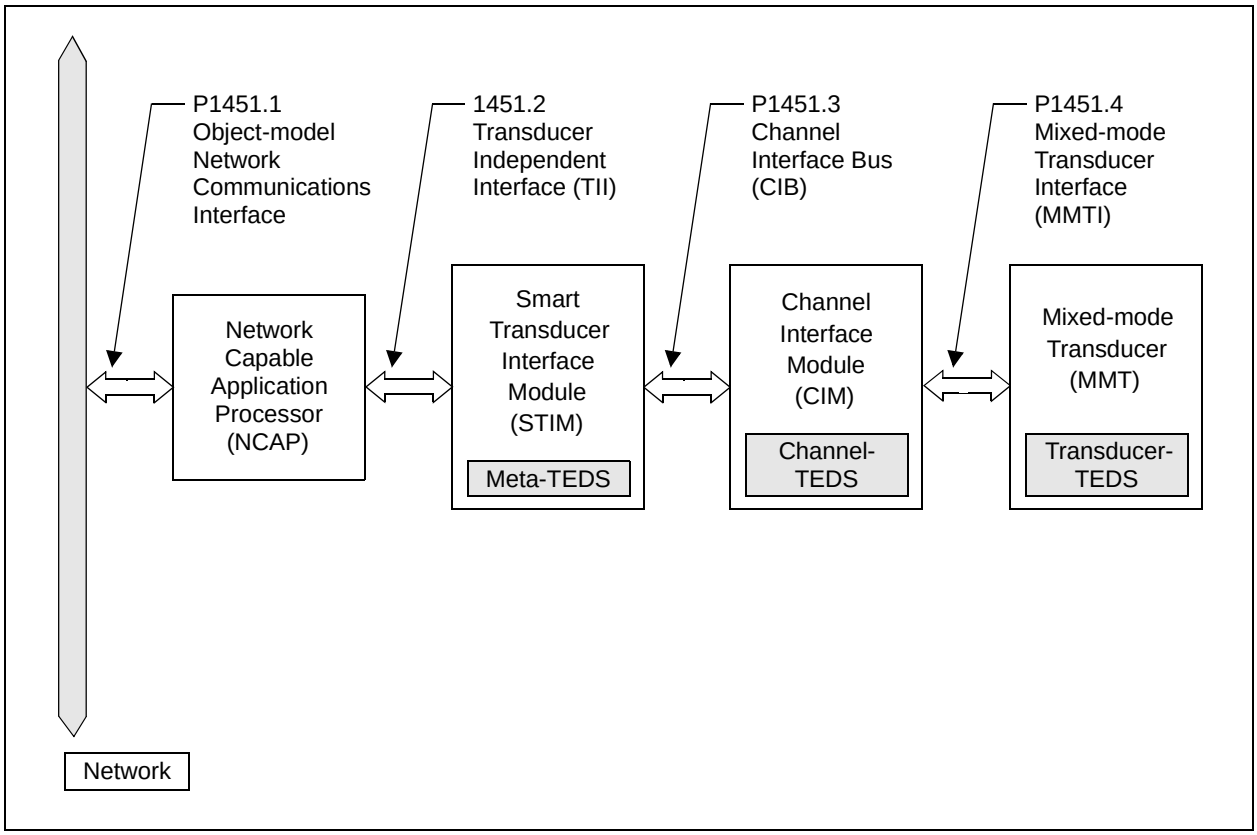
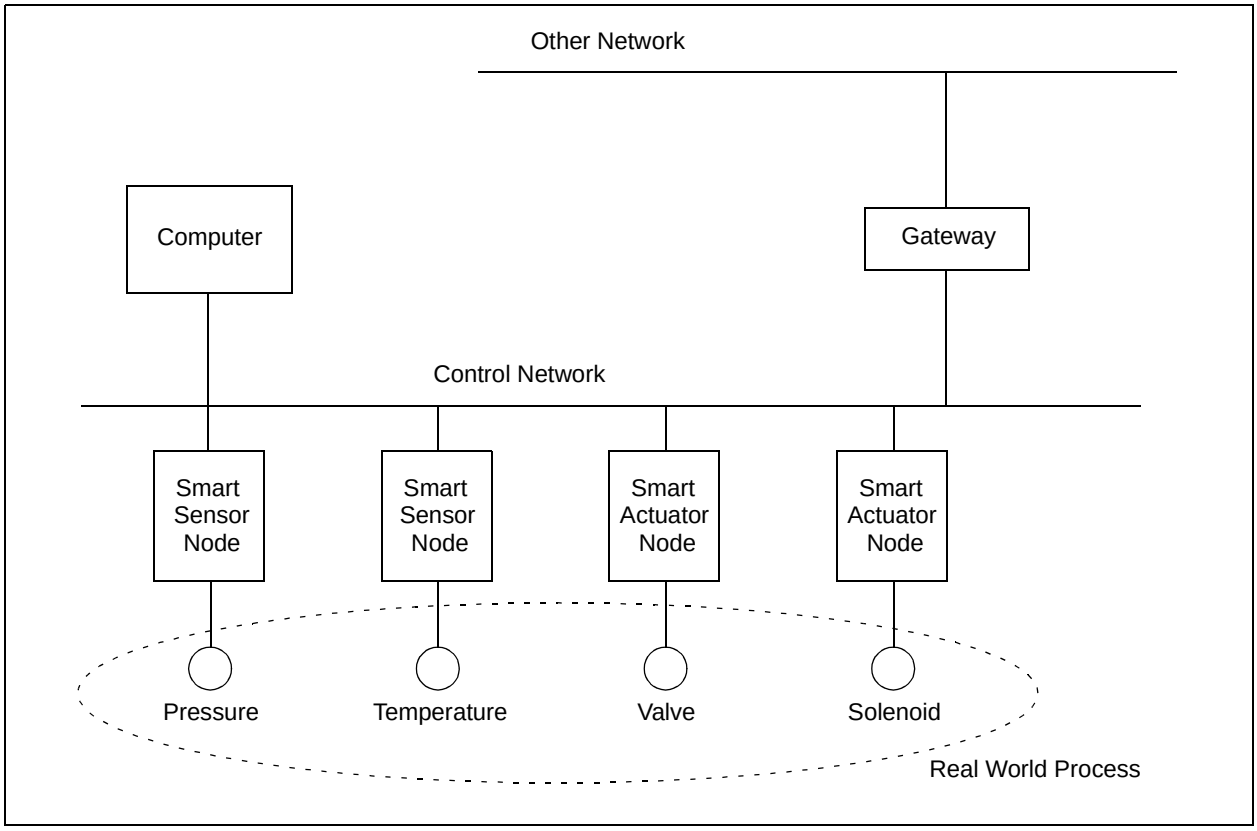


FIGURE 2: GENERAL DISTRIBUTED MEASUREMENT AND CONTROL APPLICATION



WHAT IS AN NCAP?

A STIM works in conjunction with a microprocessor based interface to a communication network. This module, termed a Network Capable Application Processor (NCAP) in the standard, primarily mediates between the STIM (whose interface is computer network independent) and a particular network.

The NCAP may also perform correction of the raw data from the STIM and may include application specific data processing and control functionality (see Figure 3 and Figure 4).

The Hewlett-Packard BFOOT-66501 Embedded Ethernet Controller is the NCAP for this application. The controller is a complete thin web server solution for manufacturers of smart sensors and actuators, or for manufacturers of products with embedded control who need a way to rapidly create smart Ethernet-ready devices that can scale to complete solutions. The BFOOT-66501 supports TCP/IP, FTP and HTTP protocols.

FIGURE 3: CONTEXT FOR THE TRANSDUCER INTERFACE SPECIFICATION

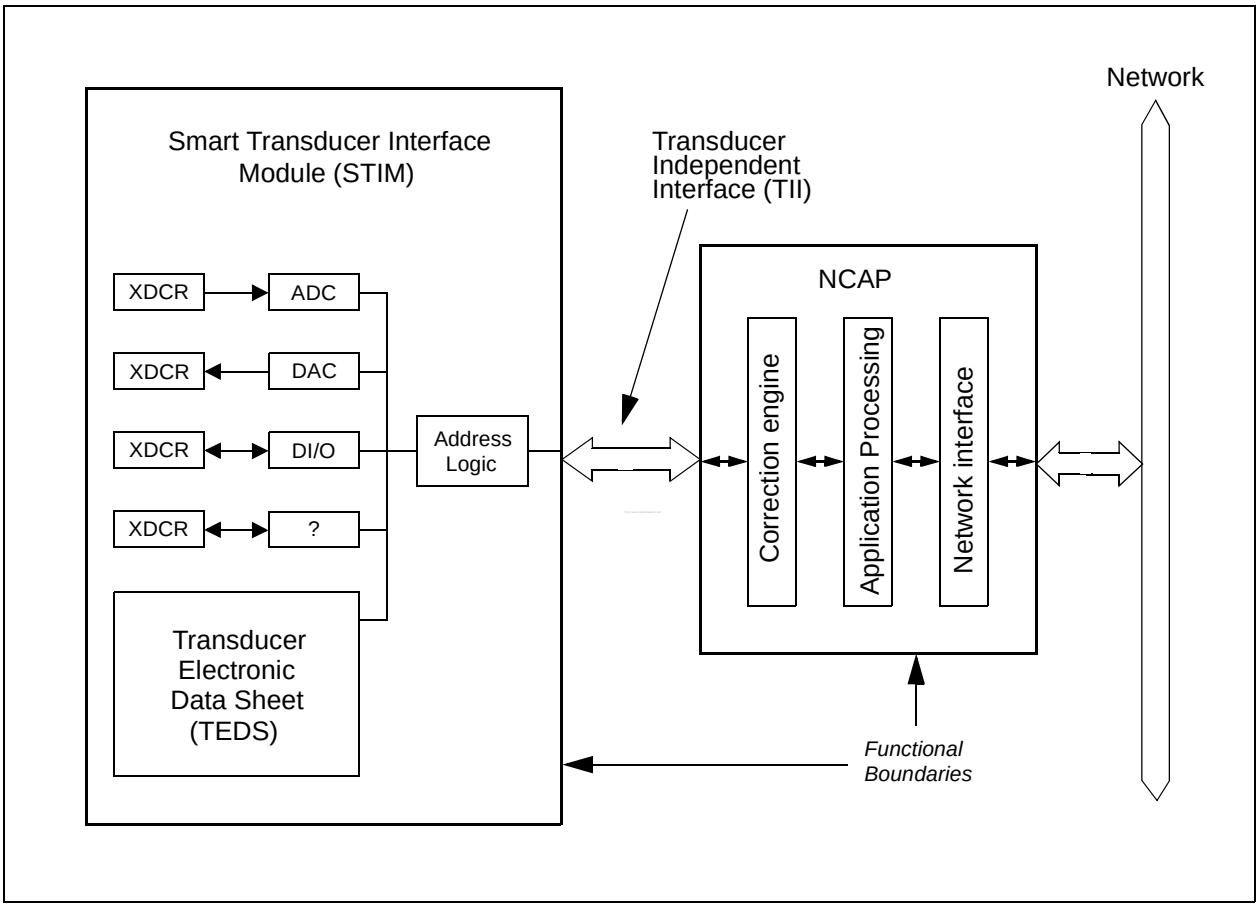
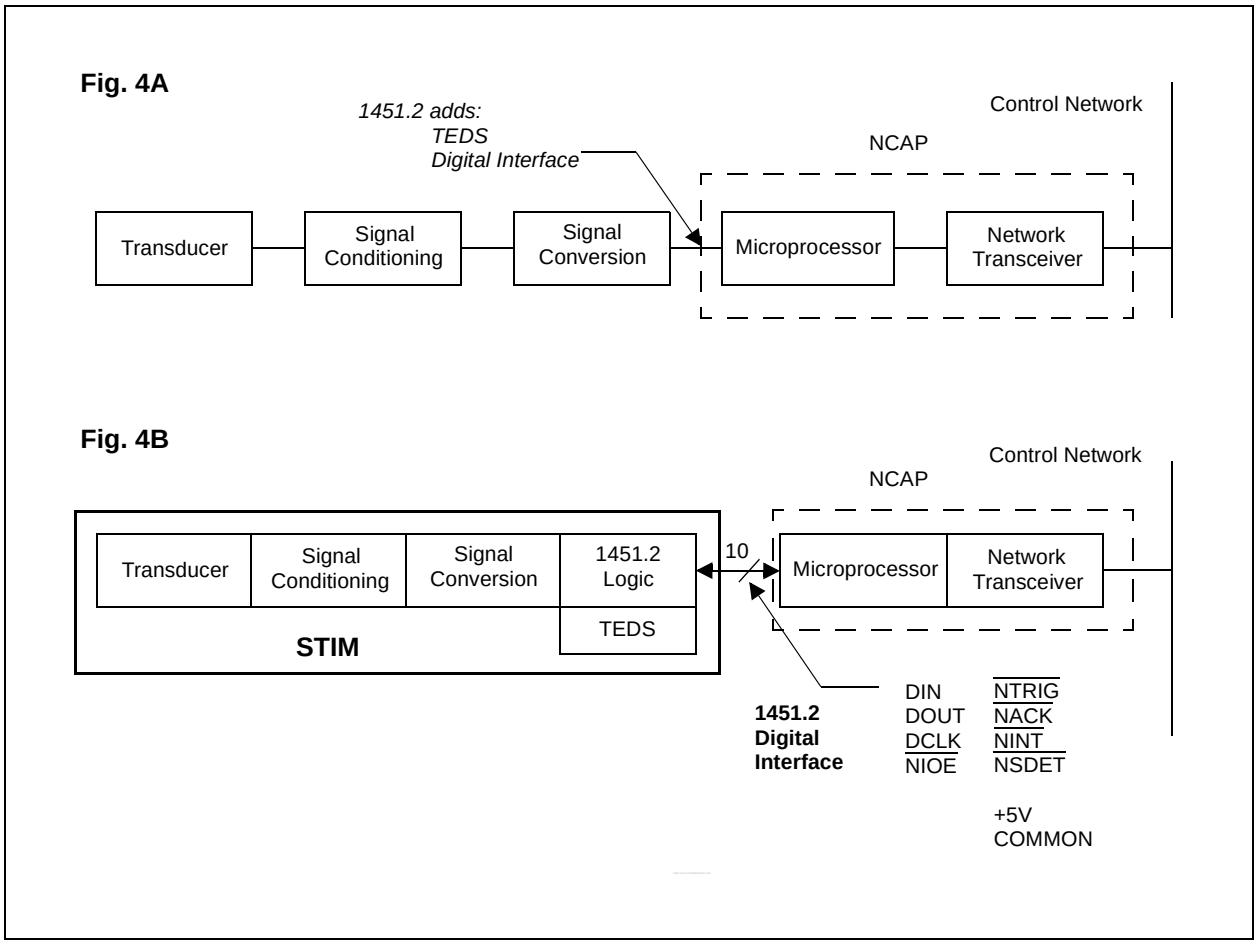


FIGURE 4: BLOCK DIAGRAMS FOR A GENERAL SMART TRANSDUCER AND A 1451.2 COMPATIBLE TRANSDUCER



WHAT IS A STIM?

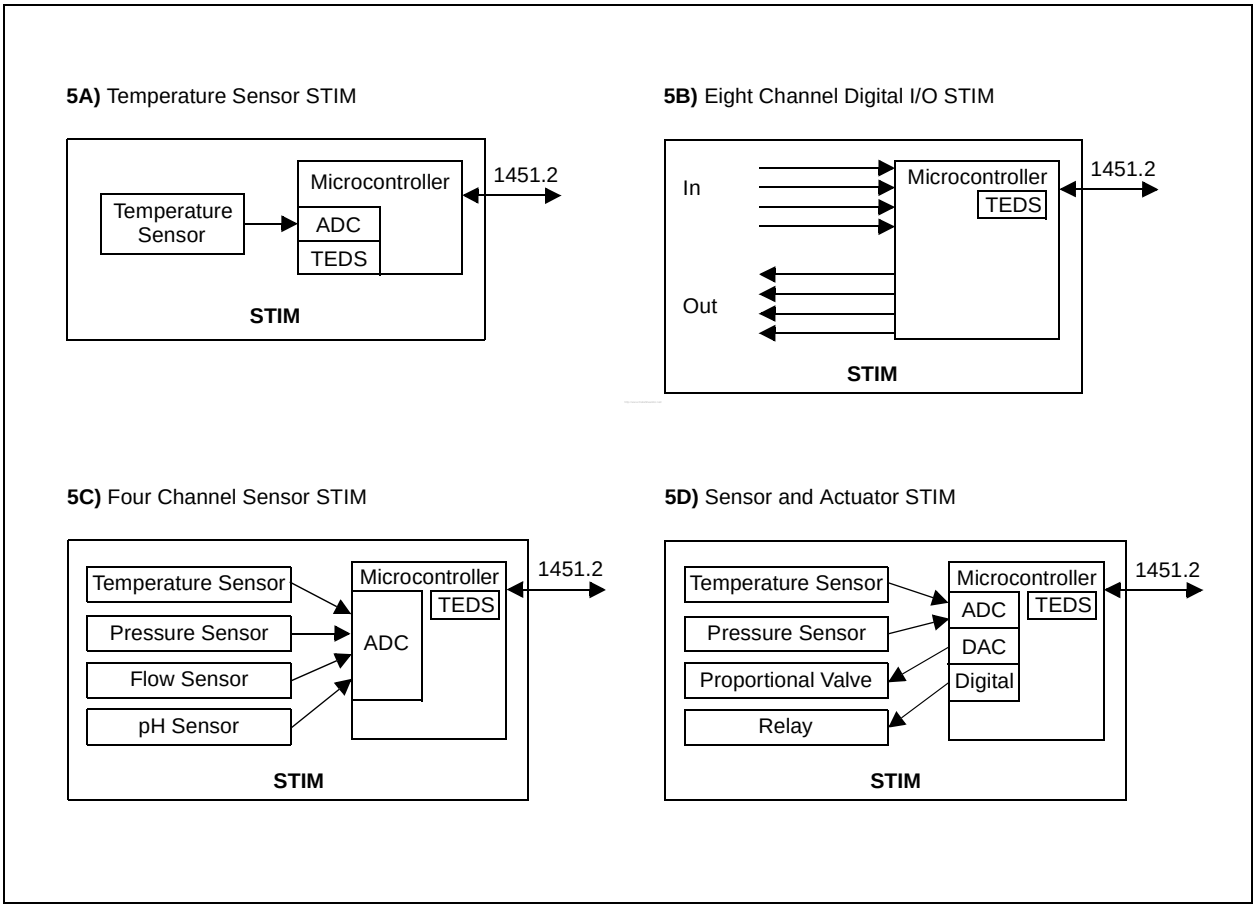
The IEEE 1451.2 standard introduces the concept of the STIM. A STIM can range in complexity from a single sensor or actuator, to many channels (up to 255 channels) of transducers (sensors or actuators). Examples of STIM's are shown in Figure 5.

A transducer channel is denoted "smart" in this context because of the following three features:

- It is described by a machine-readable Transducer Electronic Data Sheet (TEDS).
- The control and data associated with the channel are digital.
- Triggering, status, and control are provided to support the proper functioning of the channel.

A STIM contains the TEDS, logic to implement the transducer interface, the transducer(s) and any signal conversion or signal conditioning. A STIM is controlled by a NCAP module by means of a dedicated digital interface. This interface is not a network. The NCAP mediates between the STIM and a digital network, and may provide local intelligence. It is desirable that the STIM and NCAP add little size or cost to the transducer(s) they describe and interface. TEDS memory requirements are typically less than two kilobytes. These size and cost considerations also restrict the computing power available in the NCAP.

FIGURE 5: EXAMPLES OF 1451.2 SMART TRANSDUCER INTERFACE MODULES



STIM IMPLEMENTATION

The Microchip PIC16C62A and 93C86 EEPROM are the hardware base for the STIM in this application note. Other PICmicro devices could have been implemented for the STIM, such as the PIC16C773 with a 93C86. Another alternative might be to utilize a Microchip SPI™ protocol based EEPROM device in place of the Microwire® memory device. When developing a STIM, there are several parts that must be addressed:

- STIM Hardware
- NCAP Interface
- Creating the TEDS
- Organization of the Source Code
- Creating the Source Code using the Template

STIM Hardware

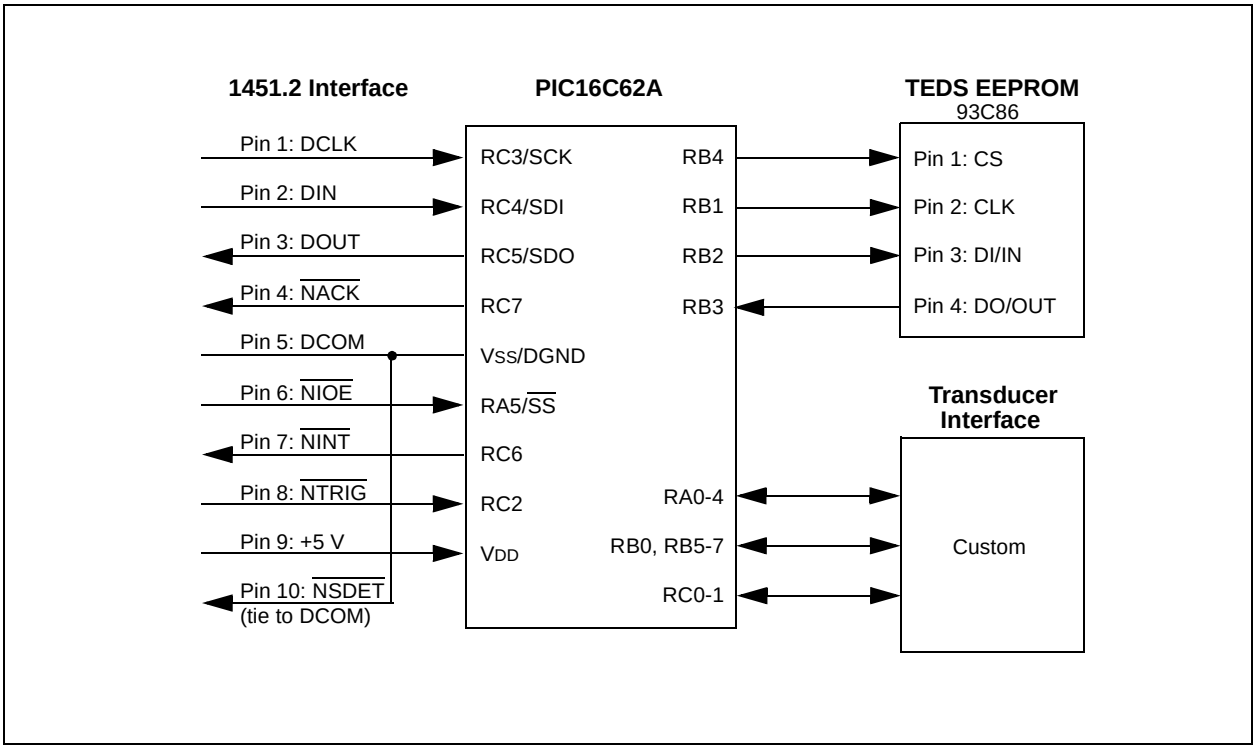
The STIM hardware must be capable of supporting the 1451.2 interface, the transducer(s) interface and the TEDS interface (see Figure 6). These three requirements should be considered when selecting a PICmicro device for the STIM. For example, the 1451.2 interface requires 8 pins (plus power and ground), the TEDS interface may require up to 4 pins and the transducer interface will most likely require all remaining pins and possibly more. The 1451.2 interface requires a synchronous data transfer with the respective control signals. Selecting a PICmicro MCU with a SPI peripheral will meet this interface requirement. The transducer interface will vary, depending upon the STIM functional requirements. For example, if the STIM requires an ADC, what are the requirements:

- 8- 10- or 12-bit ADC?
- What is the sample rate required?
- Can the ADC on a PICmicro MCU be utilized?
- Is an external stand-alone ADC required, e.g., the Microchip MCP3201.

These are questions which the STIM developer must answer. If the STIM requires a 12-bit ADC, then the PIC16C771 or PIC16C773 may be a solution. Likewise, if the STIM requires a 10-bit ADC, then you might use the PIC16C717 or FLASH PIC16F873 and eliminate the external EEPROM. Remember that a STIM can support up to 255 transducer channels. This STIM capability alone may dictate the selection of the PICmicro device for the STIM.

Finally, the TEDS interface must be evaluated. The size of the TEDS will, for the most part, depend on the number of transducer channels that the STIM supports. The TEDS is read, written and used by the NCAP. The TEDS holds information, such as: sensitivity, serial number, model number, date of last calibration, specific model number data characteristics, etc. If the STIM MCU is OTP based, then an external EEPROM device is required. If the STIM MCU is FLASH based, such as the PIC16F873, then no external EEPROM is required, since you may use the FLASH program memory for the TEDS.

FIGURE 6: STIM INTERFACE BLOCK DIAGRAM



NCAP Interface

The physical connection between the NCAP and the PIC16C62A (STIM) is a 10-pin Transducer Independent Interface (TII) (see Figure 6). The TII is built around a synchronous serial communications based on the Serial Peripheral Interface (SPI) protocol. The recommended TII pin assignments are listed in Table 1, which also indicates whether each physical signal is considered an input or output by the NCAP and the STIM. The TII logical signal definitions and functions are shown in Table 2, which also indicates whether a signal is positive or negative logic and whether it is level or edge sensitive. The SSP module of the PIC16C62A is used for SPI communications between the NCAP and STIM. When developing the TEDS, one parameter to consider is the maximum data rate for SPI transfers. The respective PICmicro MCU specification should also be considered for this TEDS entry.

The 1451.2 interface provides for the sequence of reading and writing between the NCAP and the STIM. The top level protocols are read frame, write frame and triggering. The general data transfer protocol is shown in Figure 7.

In addition to the logical signals described above, the TII also supplies power and a common ground reference to the STIM. The NCAP supplies up to 75 mA at 5V ± 0.20VDC to the STIM. The standard provides for supplemental power, independent of the NCAP, if needed for sensitive or high-power transducers, but only the NCAP can provide power for the STIM interface control circuitry.

Finally, the connector type is left up to the developer of the STIM. The IEEE-1451.2 standard does not call out a connector type. However, the standard does recommend that the STIM connector be:

- At least ten pins
- Male (because the STIM will not supply power)
- Polarized

For this STIM implementation, a 2x5 polarized connector is implemented.

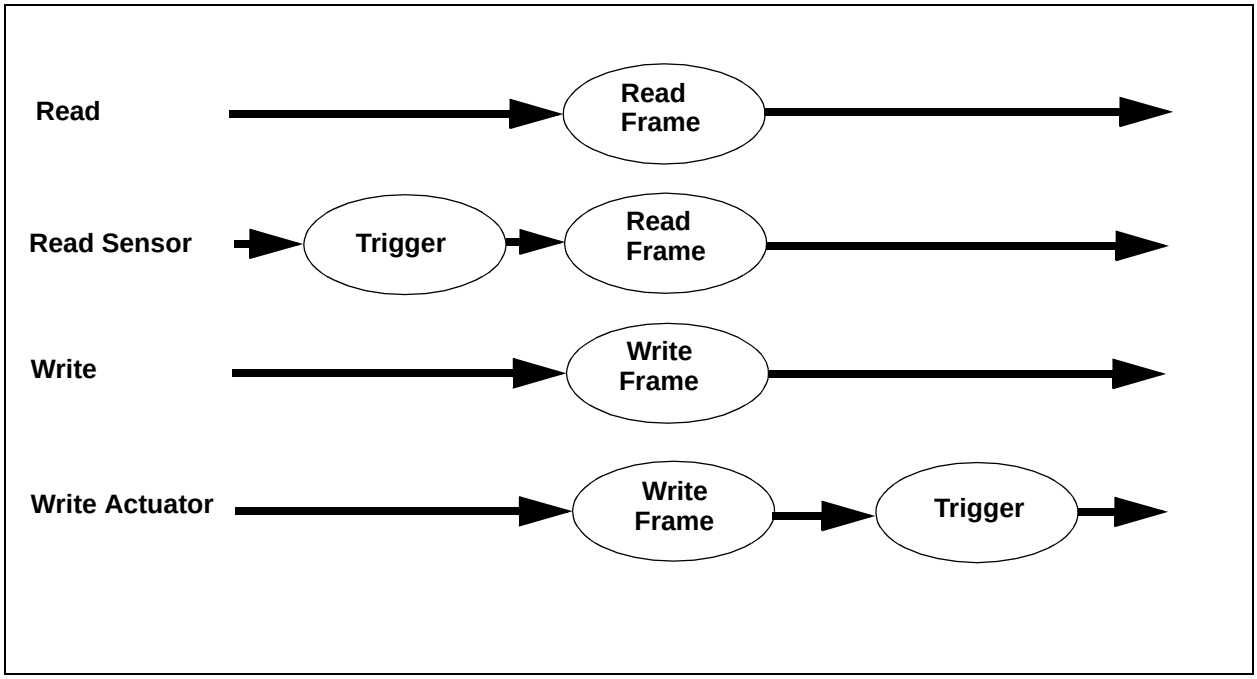
TABLE 1: RECOMMENDED TII PIN ASSIGNMENTS

Pin Number	Signal Name	Wire Color	Direction for NCAP	Direction for STIM
1	DCLK	brown	OUT	IN
2	DIN	red	OUT	IN
3	DOUT	orange	IN	OUT
4	$\overline{\text{NACK}}$	yellow	IN	OUT
5	COMMON (GROUND)	green	POWER	POWER
6	$\overline{\text{NIOE}}$	blue	OUT	IN
7	$\overline{\text{NINT}}$	violet	IN	OUT
8	$\overline{\text{NTRIG}}$	gray	OUT	IN
9	POWER (5 VDC)	white	POWER	POWER
10	$\overline{\text{NSDET}}$	black	IN	OUT

TABLE 2: TII SIGNAL DEFINITION

Line	Logic	Driven By	Function
DIN	Positive logic	NCAP	Transports address and data from NCAP to STIM
DOUT	Positive logic	STIM	Transports data from STIM to NCAP
DCLK	Positive edge	NCAP	Positive-going edge latches data on both DIN and DOUT
$\overline{\text{NIOE}}$	Active low	NCAP	Signals that the data transport is active and delimits data transport framing
$\overline{\text{NTRIG}}$	Negative edge	NCAP	Performs triggering function
$\overline{\text{NACK}}$	Negative edge	STIM	Serves two functions: 1. Trigger acknowledge 2. Data transport acknowledge
$\overline{\text{NINT}}$	Negative edge	STIM	Used by the STIM to request service from the NCAP
$\overline{\text{NSDET}}$	Active low	STIM	Used by the NCAP to detect the presence of a STIM
POWER	N/A	NCAP	Nominal 5V power supply
COMMON	N/A	NCAP	Signal common or ground

FIGURE 7: GENERAL DATA TRANSFER PROTOCOL



Creating the Transducer Electronic Data Sheet (TEDS)

Creating the TEDS is recommended before you start firmware development. Through this process, you will have sufficient information to develop the custom transducer interface firmware. The key decisions that must be made are outlined below:

- Define how many channels will be provided by the STIM
- Define the data type for each channel (e.g., sensor, actuator, event sensor, etc.)
- Define the data model for each channel (e.g., integer, float, doubles, etc.)
- Define which channels (if any) have data repetitions

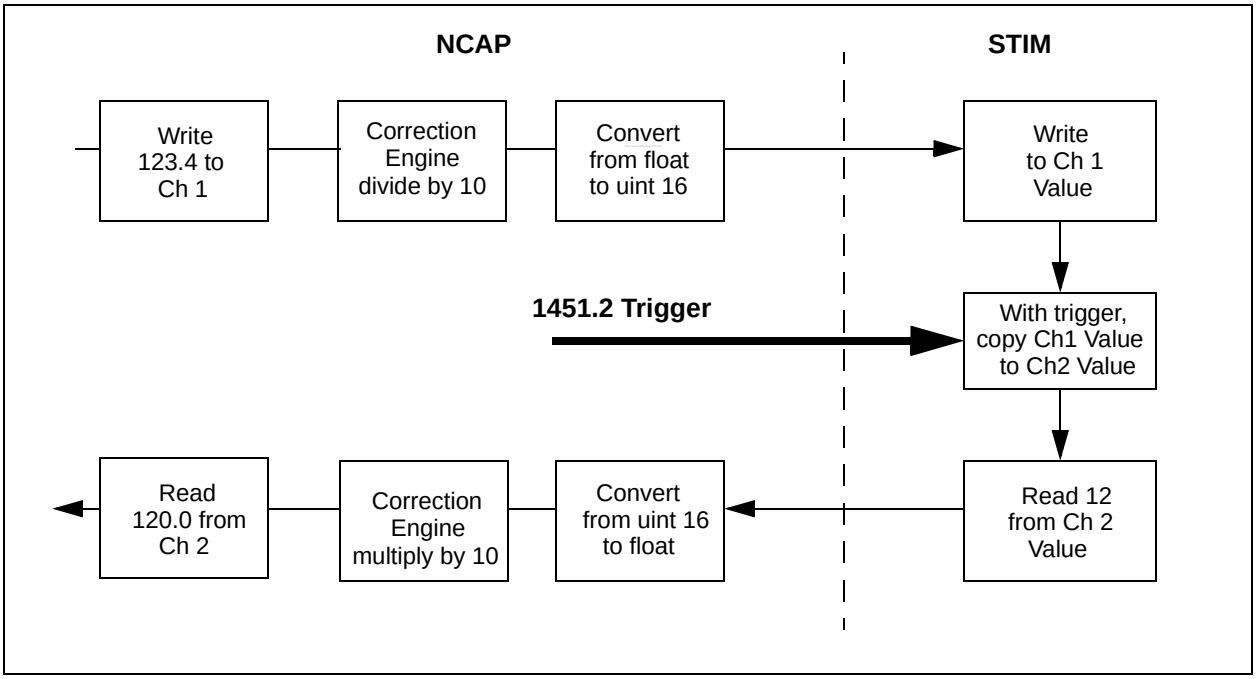
It is recommended to use a TEDS Editor when creating the actual TEDS. A template TEDS is provided as part of this application note (stimTemp.ted). Usually, you will load it into the editor and make appropriate modifications. The template TEDS (stimTemp.ted) specifies a two-channel loopback STIM. This is illustrated in Figure 8.

If desired, you can initially set each channel to CAL_NONE to disable the correction process. This will save you some time, because you will not need to specify the correction model. Later, after your STIM is functional, you may edit the TEDS file to provide this information.

Channel 1 is a 16-bit actuator and Channel 2 is a 16-bit sensor. When the device is triggered, the cached actuator value is copied to the sensor data buffer. The actuator has a “divide by 10” and the sensor has a “multiply by 10” correction. With the conversion from float to unsigned short integer (uint16), the following loopback transformation occurs: 1) Write channel 1 with 123.4; 2) Read 120 from channel 2. The intervening steps are as follows:

1. 123.4 gets converted to 12.34 by the correction engine.
2. The value is truncated to 12 before being written to the STIM channel 1.
3. Reading channel 2 receives 12 from the STIM.
4. This value is converted to 120.00 by the correction engine.

FIGURE 8: LOOPBACK STIM BLOCK DIAGRAM



TEDS Timing Parameters

The timing parameters in the template TEDS (stimTemp.ted) are appropriate for a PIC16C62A operating at 10 MHz.

The key timing parameters are provided in Table 3 below. Note that the stimTemp.c has very little 'real' work to do. Consequently, these values are about as fast as is practical, given for a PICmicro MCU running at 10 MHz. Of course, a PICmicro MCU operating at 20 MHz and hand-crafted assembly will yield additional time savings. Figure 9 presents a typical timing sequence for a triggered channel 0 read.

TABLE 3: STIM TIMING PARAMETERS

1451.2 Section	Description	Value
Meta TEDS		
5.1.3.17	Worst Case Channel Warm-up Time	1 msec.
5.1.3.18	Command Response Time (Maximum time to process any command)	100 µsec.
5.1.3.19	STIM Handshake time (Time to remove trigger acknowledge)	10 µsec.
5.1.3.20	End-of-Frame Detection Latency (Maximum time between 1451.2 transactions)	50 µsec.
5.1.3.21	TEDS Hold-off Time	1.2 msec
5.1.3.22	Operational Hold-off Time	50 µsec.
5.1.3.23	Maximum Data Rate	5 Mbps
Channel TEDS		
5.2.3.21	Channel Update Time (Maximum time to acknowledge a trigger)	6 µsec.
5.2.3.22	Channel Write Setup Time (Time between end of write frame and application of trigger)	15 µsec.
5.2.3.23	Channel Read Setup Time (Time between trigger back and start of read frame)	160 µsec.
5.2.3.24	Channel Sampling Period (Minimum time between back-to-back triggers)	160 µsec.
5.2.3.25	Channel Warm-up Time	1 msec
5.2.3.26	Channel Aggregate Hold-off Time (The total time for the data transport frame at maximum data rate. This includes byte-pacing delays added by the STIM)	75 µsec.
5.2.3.27	Timing Correction (Correction of timestamp for a given channel with channel 0 trigger)	1 µsec.
5.2.3.28	Trigger Accuracy (Accuracy of Timing Correction)	1 µsec.

FIGURE 9: TRIGGERED CHANNEL ZERO READ

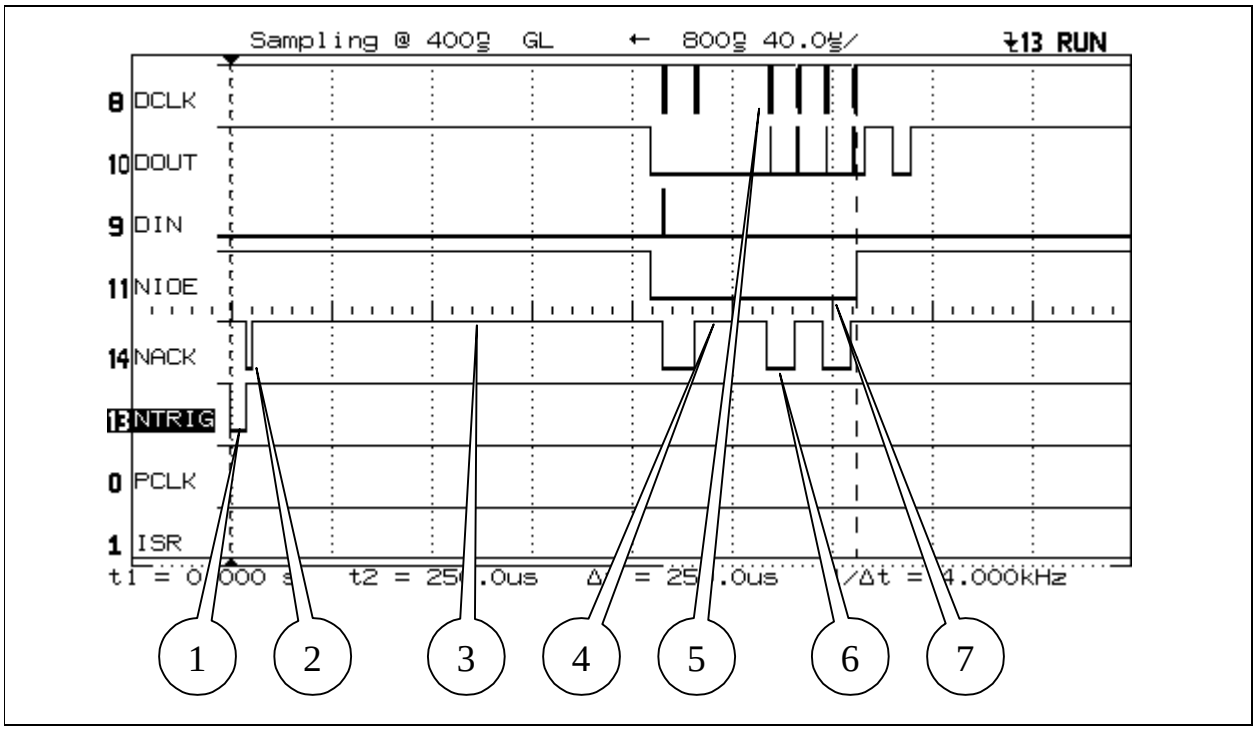


Figure 9 represents a 1451.2 bus data transport during a triggered channel 0 read. Note that six bytes are transferred across the 1451.2 bus for this transaction (i.e., function address of 128, channel of 0, MSB of channel 1, LSB of channel 1, MSB of channel 2 and LSB of channel 2). In this measurement, the logic analyzer trigger signal condition was the STIM acknowledgment of the NCAP trigger (first falling edge of the NACK signal). Various points of interest are identified on the figure and discussed below.

1. Channel Update Time.
2. STIM Handshake Time.
3. Channel Read Setup Time. The NCAP will wait this long before dropping $\overline{\text{NIOE}}$ to begin the read data transaction. In the stimtemp.c implementation, the NCAP delay gives the STIM time to perform any status updates and interrupt processing. In a STIM with measurement hardware, this time would most likely be dominated by ADC conversion delays.
4. The increased delay between the end of the channel selector byte and the transmission of the first byte of the data frame. This delay happens when the PICmicro device is processing the function code and dispatching to the `transducer_read()` function.
5. This is the 8 DCLK edges that clock the transfer of a single byte across the 1451.2 interface. In the STIM, the DCLK rate is 5Mbps.
6. This is the time for the PICmicro device to perform the byte-handshake.
7. The channel 0 read operation is longer than the Channel Aggregate Hold-off Time.

Organization of the Source Code

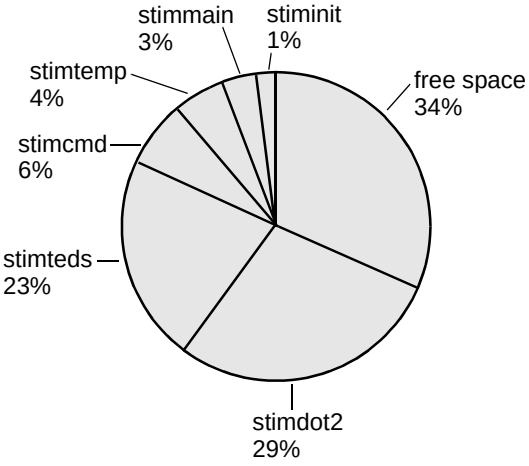
The source code is divided into the following files. All files are in a common subdirectory, unless noted. Table 4 lists the source code files and provides a brief functional description. The source code files are provided in Appendix A. The RAM and program memory requirements for this STIM implementation are shown in Figure 10.

TABLE 4: SOURCE CODE FILES

File Name	File Destination	Description
stimmain.c	PICmicro MCU	This is the main event processing loop for the firmware. You should not need to modify this file.
stim.h	PICmicro MCU	Global STIM variable definitions and function prototypes. You should not need to modify this file.
stimdot2.c	PICmicro MCU	This source file provides the functions to manage the 1451.2 communications. You should not need to modify this file.
stimdot2.h	PICmicro MCU	Variable references and macros utilized for 1451.2 communications and process support. You should not need to modify this file.
stimteds.c	PICmicro MCU	This source file provides functions to manage communication with the TEDS EEPROM and reading or writing TEDS blocks. No modification of this file is required unless you change the memory type for the TEDS.
stimteds.h	PICmicro MCU	Reference linkage for variables and functions and PICmicro MCU pin definitions for communicating with TEDS. No modification of this file is required unless you change the memory type for the TEDS.
stimtemp.c	PICmicro MCU	This is a source code template that you will modify for your STIM. Typically, this is the only file you will need to study. Make a copy of this file and rename as appropriate.
stimtemp.h	PICmicro MCU	Reference linkage for variables and functions for the template.c file. You should not need to modify this file.
stiminit.c	PICmicro MCU	This source file provides functions to manage the PICmicro MCU initialization. Depending on the STIM configuration, this file may require modifications.
stiminit.h	PICmicro MCU	PICmicro MCU PORT and TRIS definitions. Depending on the STIM configuration, this file may require modifications.
stimcmd.c	PICmicro MCU	This file is used to process the functional and channel address commands issued by the NCAP over the 1451.2 interface. The functions called are located in the stimtemp.c, stimteds.c and stimdot2.c files. You should not need to modify this file.
stimcmd.h	PICmicro MCU	Reference linkage for variables and functions.
pic.h	PICmicro MCU	PICmicro MCU processor definition file. This is a Hi-Tech specific file. No modification of this file is required.
common.h	PICmicro MCU	Common information for all source files, including 1451.2 interface pin definitions.
cnfig62a.h	PICmicro MCU	PICmicro MCU configuration bit definition file. You should not need to modify this file.
generateVersion.cpp	PC	Source code for the PC or HPUX tool to generate the STIM version string.
generateVersion	PC	HPUX executable.
generateVersion.exe	PC	WIN32 executable.

Note: The PICmicro MCU based source files were built with the Hi-Tech PICC Compiler tool, version 7.85.

FIGURE 10: PROGRAM MEMORY USAGE FOR STIMTEMP



Creating the Source Code using the Template

Create a new source code module using the supplied template (stimtemp.c). It is recommended that you first copy stimtemp.c to some new file (e.g. mystim.c). Perform the following steps (also outlined in stimtemp.c):

1. Define the maximum number of channels on the STIM with the #define MAX_NUM_CHANNELS macro. This is used to allocate memory for the status, mask, aux_status and aux_mask registers. This macro is located in the common.h file.
2. Run the “generateVersion” executable (on either the HPUX or WIN32) and provide a single command line parameter for your STIM. For example: generateVersion myStim_v1.0. Copy and paste the output into your source file. For this STIM, the output is copied into the stimtemp.c file.
3. If your STIM hardware uses different pins than those shown in Figure 6, modify the #define statements to override the pin assignments. For the 1451.2 interface pin definitions, see the common.h file. For the TEDS interface pin definitions, see the stimteds.h file.
4. Provide the #defines for the tri-state configuration and initial port values (e.g. TRIS_A_VAL and INIT_A_VAL). Since this is done on a byte basis (rather than bit), careful attention should be paid to not alter the values for the 1451.2 or TEDS EEPROM pins. The default values are presented in the stiminit.h file. A value of ‘1’ in the tri-state register will set the corresponding pin to an input.
5. If you need to override any default processing in the generic code, define the appropriate macro. For example, to implement a non-supported 1451.2 function for a channel 0 operation, provide the following declaration and macro (see Table 5). Note that most parameters are available as global parameters and are therefore, not passed as arguments:
6. Declare the following functions:
   - void transducer_initialize(void);
   - void transducer_trigger(void);
   - void transducer_read(unsigned char chnl);
   - void transducer_write(unsigned char chnl);
7. Implement the above functions:
   - transducer_initialize( ) - Set up the transducer to its RESET condition. This function will be called immediately following the configuration of the tri-state registers.
   - transducer_trigger( ) - Perform the 1451.2 trigger function. The trig_chnl_addr global variable holds the current trigger address. Normally, the standard_trigger( ) function can be called at the end of this function, to perform the 1451.2 trigger handshake operation.
   - transducer_read( ) - Send data to the NCAP by performing the 1451.2 read operation. The chnl_addr global variable holds the current channel address. Be prepared to generate a 1451.2 channel 0 read frame, if requested. The chnl_addr will be zero in this case. The wr_1451_byte( ) function should be used to send data to the 1451.2 bus.
   - transducer_write( ) - Receive data from the NCAP by performing the 1451.2 write operation. The chnl_addr global variable holds the current channel address. Be prepared to receive a 1451.2 channel 0 write frame if requested. The chnl_addr will be zero in this case. The rd_1451_byte( ) function should be used to read each data byte from the 1451.2 bus.

TABLE 5: MACROS TO MODIFY STANDARD STIM PROCESSING

Macro Name	Function	Default Value
DEFAULT_1451DOT2_FUNCTION	Default processing for 1451.2 function codes.	post_inval_cmd( )
DEFAULT_1451DOT2_COMMAND	Default processing for 1451.2 commands.	post_inval_cmd( )
RESET_1451DOT2_COMMAND	Perform the RESET function for a 1451.2 channel or STIM.	standard_reset( )
AFTER_1451DOT2_FRAME	After processing the payload portion of the transport. **Note:** The frame may still be active if the NCAP is trying to read too many bytes. For test purposes only.	
AFTER_PROCESS_TEDS_PROLOG	After processing the TEDS header. For test purposes only.	
CHECK_WRITE_PROTECT	Check if the TEDS is write protected. If true, this macro should invoke post_inval_cmd() and return.	

Managing Status, Mask and Interrupts

By default, this code provides full support for STIM status, interrupt mask, auxiliary status and auxiliary interrupt mask registers (function code 130, 132, 133 and 134). This is provided at both the global and channel levels.

Because the auxiliary registers are optional, a simple mechanism is provided to disable their implementation. Just define NO_AUX_STATUS_REGISTERS, which is located in the common.h file. This decreases RAM and ROM usage. For example, RAM decreases by 4*numChannels + 2 for the auxiliary status and mask registers.

The STIM will generate an interrupt as illustrated in Figures 10 and 11 of the 1451.2 standard. Note that for each channel, the auxiliary status can generate interrupts through two different paths: 1) through the auxiliary interrupt mask, and 2) through the auxiliary status bit in the channel status register, when combined with that channel's interrupt mask.

Similarly, each channel can generate an interrupt through two paths: 1) through its service request bit, and 2) through the STIM's global status register and the global interrupt mask.

Structure of the TEDS EEPROM Data

The following table describes the EEPROM header used by the current code. The actual data blocks follow the 1451.2 standard.

TABLE 6: STRUCTURE OF THE TEDS EEPROM DATA

TEDS Structure	Address (hex)	Length (bytes)	Field Comment
File Header	0	2	LITTLE ENDIAN: address of directory, currently 8
	2	2	LITTLE ENDIAN: number of channels (N)
	4	2	LITTLE ENDIAN: TEDS image format version number, currently 1
	6	2	LITTLE ENDIAN: number of TEDS blocks per channel, currently 4
Begin directory	8	2	LITTLE ENDIAN: address of meta-TEDS
	10	2	LITTLE ENDIAN: address of meta-ID-TEDS
	12	2	LITTLE ENDIAN: address of ch 1 channel TEDS
	14	2	LITTLE ENDIAN: address of ch 1 channel ID TEDS
	16	2	LITTLE ENDIAN: address of ch 1 calibration TEDS
	18	2	LITTLE ENDIAN: address of ch 1 calibration ID TEDS
	20	2	LITTLE ENDIAN: address of ch 2 channel TEDS
	22	2	LITTLE ENDIAN: address of ch 2 channel ID TEDS
	24	2	LITTLE ENDIAN: address of ch 2 calibration TEDS
	26	2	LITTLE ENDIAN: address of ch 2 calibration ID TEDS
	i(= 8N+4)	2	LITTLE ENDIAN: address of ch N channel TEDS
	i+2	2	LITTLE ENDIAN: address of ch N channel ID TEDS
	i+4	2	LITTLE ENDIAN: address of ch N calibration TEDS
	i+6	2	LITTLE ENDIAN: address of ch N calibration ID TEDS
Begin TEDS data	J(= i+8)	1	functional address of first TEDS block
	J+1	1	channel address of first TEDS block
	J+2	4	first TEDS block length header (long int whose value is L)
	J+6	L	first TEDS block plus checksum
	K(= J+6+L)	1	functional address of second TEDS block
	K+1	1	channel address of second TEDS block
	K+2	4	second TEDS block length header (long int whose value is M)
	K+6	M	second TEDS block plus checksum
	P	1	functional address of last TEDS block
	P+1	1	channel address of last TEDS block
	P+2	4	last TEDS block length header (long int whose value is Q)
	P+6	Q	last TEDS block plus checksum
	P+6+Q	2	two null bytes

Note 1: Numbers in the file header and pointers in the directory block are 16-bit unsigned LITTLE-ENDIAN. Multiple-byte TEDS data fields are BIG-ENDIAN.

2: The TEDS blocks are as defined in IEEE 1451.2 draft 3.02.

3: Each TEDS block is preceded by two bytes giving the functional address and the channel address by which the block is addressed in the STIM. This identifies the block in the same semantics as the standard and does not depend on absolute address.

4: The pointers in the directory point to the beginning of the block structure, NOT to the functional/channel address header. (The last entry in the directory is an exception to this in that it is defined to point to a functional address.)

5: The functional/channel address headers provide an alternate way of finding the TEDS block needed. The first two bytes following the last directory entry are the functional and channel address for the first TEDS block. If this block is not the one desired, read the next four bytes as a long integer (which is the length of the TEDS block) and add it to the current read pointer to get the address of the next functional/channel address header. In this way, you can step through memory until you find the TEDS you want. The null bytes after the last TEDS blocks signal the end of the linked list.

6: The linked list format allows the addition of other TEDS and TEDS extensions. These will not have pointers in the directory and will have to be addressed by walking through the linked list. Since these are likely to be less frequently accessed, the added time required to do this is of less consequence.

Limitations in the Current Implementation

The current code has the following limitations:

1. The TEDS writing mechanism uses the function code 32 and only excepts a write of the full TEDS file.It does not support writing individual TEDS blocks (except the channel CAL TEDS).
2. The write CAL ID TEDS mechanism is not supported. Currently, it returns post_inval_cmd( ).

SUMMARY

The goal of the IEEE 1451.2 standard is to provide an industry standard interface to efficiently connect transducers to microcontrollers and to connect microcontrollers to networks. Microchip Technology offers a large portfolio of smart microcontrollers that can be implemented as the STIM for compliance with the IEEE 1451.2 standard.

Note:	Information contained in the application note regarding device applications and the like, is intended for suggestion only and may be superseded by updates. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated, with respect to the accuracy or use of such information, or infringement of patents, or other intellectual property rights arising from such use or otherwise.

REFERENCE MATERIAL

IEEE Std 1451.2-1997, IEEE Standard for a Smart Transducer Interface for Sensors and Actuators - Transducer to Microprocessor Communication Protocols and Transducer Electronic Data Sheet (TEDS) Format

PIC16C6X Data Sheet, Microchip Technology Inc., Document # DS30234

PIC16C717/770/771 Data Sheet, Microchip Technology Inc., Document # DS41120

PIC16C77X Data Sheet, Microchip Technology Inc., Document # DS30275

PIC16F87X Data Sheet, Microchip Technology Inc., Document # DS30292, 1998

PICmicro™ Mid-Range MCU Family Reference Manual, Microchip Technology Inc., Document # DS33023

APPENDIX A: STIM SOURCE CODE FILES

```
/*****************************************************************
*                                                                *
*          The PICmicro as an IEEE 1451.2 Compatible Smart       *
*               Transducer Interface Module (STIM)               *
*                                                                *
*****************************************************************
*                                                                *
*   Filename:       stimmain.c                                   *
*   Date:           06/27/2000                                   *
*   Revision:       1.12                                         *
*                                                                *
*   Contributor:    Richard L. Fischer                           *
*   Company:        Microchip Technology Inc.                    *
*                                                                *
*   Contributor:    Jeff Burch                                   *
*   Company:        Agilent Technologies                         *
*                                                                *
*                                                                *
*   Tools:          Hi-Tech PIC C Compiler V7.85                 *
*                   MPLAB 5.00.00                                *
*                                                                *
*****************************************************************
*                                                                *
*   System files required:                                       *
*                                                                *
*                   stimmain.c                                   *
*                   stimdot2.c                                   *
*                   stiminit.c                                   *
*                   stimteds.c                                   *
*                   stimcmd.c                                    *
*                   stimtemp.c                                   *
*                                                                *
*                                                                *
*                   pic.h        (Hi-Tech file)                  *
*                   common.h                                     *
*                   stim.h                                       *
*                   stimdot2.h                                   *
*                   stiminit.h                                   *
*                   stimteds.h                                   *
*                   stimcmd.h                                    *
*                   cnfig62a.h                                   *
*                   delays.h     (Hi-Tech file)                  *
*                                                                *
*****************************************************************
*                                                                *
*   Notes:                                                       *
*                                                               *
*   Device Fosc -> 10.00MHz external crystal                    *
*   WDT -> off                                                  *
*    Brownout -> on                                              *
```

```
*   Powerup timer -> on                                                *
*   Code Protect -> off                                                *
*                                                                      *
*   Interrupt sources -> None (at this moment)                         *
*                                                                      *
*                                                                      *
************************************************************************/


    #include <pic.h>                  // processor if/def file
    #include “cnfig62a.h”             // configuration word definitions
    #include “common.h”               // project commons, etc.
    #include “stim.h”                 //

    __CONFIG   ( CONBLANK & BODEN_ON & PWRTE_ON & CP_OFF & WDT_OFF & HS_OSC );



/*************************************************************************/
void main( void )
{
    init_pic_ports();                 // set port directions, initialize output values
    transducer_initialize();          // initialize the transducer
    process_teds_prolog();            // read the TEDS prolog
    init_mask_registers( );           // initialize the STIM to generate all interrupts
    init_int_registers( );            // initialize the status registers
    update_nint();                    // on status or mask write update NINT

    aborted_1451_io = 0 ;             // initialize global variable

    for ( ; ; )
    {
      NACK_PIN = 1;                   // set NACK output to idle high

      // ensure de-assertion of NIOE unless abort noted in prev. frame
      // espec. important for 2nd, subseq. passes thru this loop
      // if aborted_1451_io (in prev. frame), do *not* wait for NIOE high

      init_spi( !aborted_1451_io );   // initialize SPI module (Section 6.3.8)
      aborted_1451_io = 0 ;           // reset variable
      wrNackDone = 0;                 // reset variable

      while ( NIOE_PIN )              // await assertion of NIOE (*SS) from NCAP
      {
        if ( !NTRIG_PIN )             // test if NCAP is asserting NTRIG to STIM
        {
          transducer_trigger();       // when trigger asserted, call the user’s function
        }
      }

      // At this point the NCAP has asserted NIOE (*SS) to STIM. This signals that the
      // data transport is active and delimits the transport framing

      // Read the 1451.2 functional and channel address
      // NOTE if NIOE (*SS) is de-asserted, abort the rest of frame


      func_addr = rd_1451_byte();   // read the 1451.2 functional address
      if ( aborted_1451_io )          // if global is true
        continue;                     // start the for loop again

      chnl_addr = rd_1451_byte();     // read the 1451.2 channel address
      if ( aborted_1451_io )          // if global is true
        continue ;                    // start the for loop again
```



```
      if ( chnl_addr > num_channels )  // test for valid channel address
      {
        post_inval_cmd();           // bad channel address, update status
      }
      else                             // valid channel address so ...
      {
        process_valid_cmd();           // process functional and channel address
      }
    }
}
```

```
/*********************************************************************
*                                                                    *
*   Filename:       stim.h                                           *
*   Date:           06/27/2000                                       *
*   Revision:       1.4                                              *
*                                                                    *
*   Tools:          Hi-Tech PIC C Compiler V7.85                     *
*                   MPLAB 5.00.00                                    *
*                                                                    *
*                                                                   *
*   Description:    Global STIM data and function prototypes         *
*                                                                    *
*********************************************************************/



extern voidprocess_valid_cmd();


// ======= Functions and data from stiminit.c =======
extern void  init_pic_ports( void );
extern void  init_spi( uchar await_nioe );


// ======= Functions and data from stimteds.c =======
// EEPROM I/O Functions
extern void  send_ee_bits( uchar databits, uchar bitcount ); // utility for next 3-4
uchar ee_read( unsigned int addr );         // returns status, data is via eeprom_data
uchar ee_write( unsigned int addr );        // returns status, data is via eeprom_data
uchar ee_wr_enable( uchar opcode ); // should receive: EE_EWEN_OP or EE_EWDS_OP


// Standard TEDS reads
void  process_teds_prolog();        // init e.g. teds_format_vsn, teds_blks_per_chnl
void  rd_meta_teds( void );
void  rd_meta_id_teds( void );
void  rd_channel_teds( void );
void  rd_channel_id_teds( void );
void  rd_calibration_teds( void );
void  rd_calibration_id_teds( void );


// Extension Read(s)
void   rd_fw_version( void );
void   rd_full_teds_eeprom( void );


// TEDS Write function(s)
void   wr_calibration_teds( void );
void   wr_calibration_id_teds( void );
void   wr_full_teds_eeprom( void );


// EEPROM globals
uchar   eeprom_data;              // data for read/write ops.
uchar   ee_rd_wr_bit_cntr;
unsigned int   teds_addr;
unsigned int   teds_addr2;


// STIM-dependent teds image metadata; these are primed from E^2 ‘prolog’
uchar   num_channels;
uchar   teds_format_vsn;
uchar   teds_blks_per_chnl;

ucharwrNackDone;// Write NACK control
```



```
// ======= Functions and data from stimtemp.c =======
extern void  transducer_initialize( void );
extern void  transducer_trigger( void );
extern void  transducer_read( void );
extern void  transducer_write( void );


// ======= Functions and data from stimdot2.c =======
// 1451 Byte I/O Functions
extern uchar  rd_1451_byte( void );
extern uchar  wr_1451_byte( uchar abyte );
extern void   frame_complete( void );
extern uchar  frame_aborted( void );     // tests NIOE and aborted_1451_io; sets latter

struct events {
       unsigned int time_expire  :1;
       unsigned int              :7;
} flag;


// 1451 Transducer, Global Functions
void   rd_status( void );
void   rd_aux_status( void );
void   rd_trig_chnl_addr( void );
void   wr_trig_chnl_addr( void );
void   rd_int_mask( void );
void   wr_int_mask( void );
void   rd_aux_int_mask( void );
void   wr_aux_int_mask( void );
void   wr_ctrl_cmd( void );
void   post_inval_cmd( void );
void   update_nint( void );
void   standard_trigger( void );
void   standard_reset( void );
void   init_mask_registers( void );
void   init_int_registers( void );


// 1451.2 Driver Globals
uchar  func_addr;
uchar  chnl_addr;
uchar  trig_chnl_addr = 1;


// side-channel status for parties interested in 1451 frame/byte i/o aborts
uchar  aborted_1451_io;


// --- status registers ---
// splitting bytes is probably safest (given compilers), easiest, tightest
uchar global_status_lo = 0;
uchar global_status_hi = 0;


uchar chan_status_lo[ MAX_NUM_CHANNELS ];
uchar chan_status_hi[ MAX_NUM_CHANNELS ];


// --- interrupt mask registers ---
uchar global_int_mask_lo;
uchar global_int_mask_hi;


uchar chan_int_mask_lo[ MAX_NUM_CHANNELS ];
```

```
uchar chan_int_mask_hi[ MAX_NUM_CHANNELS ];


// Note: the aux status registers and mask are optional.
// By default, they are implemented. Define the NO_AUX_STATUS_REGISTERS
// to disable this implementation
#ifndef NO_AUX_STATUS_REGISTERS
    // --- AUX status registers ---
    uchar global_aux_status_lo = 0;
    uchar global_aux_status_hi = 0;

    uchar chan_aux_status_lo[ MAX_NUM_CHANNELS ];
    uchar chan_aux_status_hi[ MAX_NUM_CHANNELS ];

    // --- AUX interrupt mask registers ---
    uchar global_aux_int_mask_lo;
    uchar global_aux_int_mask_hi;

    uchar chan_aux_int_mask_lo[ MAX_NUM_CHANNELS ];
    uchar chan_aux_int_mask_hi[ MAX_NUM_CHANNELS ];
#endif
```



```
/*********************************************************************
*                                                                    *
*   Filename:       common.h                                         *
*   Date:           06/27/2000                                       *
*   Revision:       1.4                                              *
*                                                                    *
*   Tools:          Hi-Tech PIC C Compiler V7.85                      *
*                   MPLAB 5.00.00                                    *
*                                                                    *
*                                                                   *
*   Description:    Common information for all source files,         *
*                   including 1451.2 pin definitions.                *
*                                                                    *
*********************************************************************/



typedef unsigned char uchar;           // create new type


#define MAX_NUM_CHANNELS  2
//#define NO_AUX_STATUS_REGISTERS


// Set these macros to handle additional global or channel control commands
#ifndef DEFAULT_1451DOT2_COMMAND
#define DEFAULT_1451DOT2_COMMAND post_inval_cmd();
#endif

#ifndef RESET_1451DOT2_COMMAND
#define RESET_1451DOT2_COMMAND standard_reset( );
#endif

#ifndef DEFAULT_1451DOT2_FUNCTION
#define DEFAULT_1451DOT2_FUNCTION post_inval_cmd();
#endif


#define  NOP()     asm (“NOP”);



/*********************************************************************/
// 1451.2 / NSP physical lines

#ifndef DCLK_PIN
#define DCLK_PIN   RC3            /* Pin 1 of 1451.2 10 pin */
#endif

#ifndef DIN_PIN
#define DIN_PIN    RC4            /* Pin 2 */
#endif

#ifndef DOUT_PIN
#define DOUT_PIN   RC5            /* Pin 3 */
#endif

#ifndef NACK_PIN
#define NACK_PIN   RC7            /* Pin 4 */
#endif

#ifndef NIOE_PIN
#define NIOE_PIN   RA5            /* Pin 6 */
#endif

#ifndef NINT_PIN
```

```
#define NINT_PIN   RC6            /* Pin 7 */
#endif

#ifndef NTRIG_PIN
#define NTRIG_PIN  RC2            /* Pin 8 */
#endif
```



```
/*****************************************************************
*                                                                *
*   Filename:       cnfig62a.h                                   *
*   Date:           06/27/2000                                   *
*   File Version:   1.00                                         *
*                                                                *
*   Compiler:       Hi-Tech PIC C Compiler V7.85                  *
*                                                                *
*****************************************************************/

/*****CONFIGURATION BIT DEFINITIONS FOR PIC16C62A PICmicro  *****/


#define     CONBLANK         0x3FFF

#define     BODEN_ON         0x3FFF
#define     BODEN_OFF        0x3FBF
#define     CP_ALL           0x00CF
#define     CP_75            0x15DF
#define     CP_50            0x2AEF
#define     CP_OFF           0x3FFF
#define     PWRTE_OFF        0x3FFF
#define     PWRTE_ON         0x3FF7
#define     WDT_ON           0x3FFF
#define     WDT_OFF          0x3FFB
#define     LP_OSC           0x3FFC
#define     XT_OSC           0x3FFD
#define     HS_OSC           0x3FFE
#define     RC_OSC           0x3FFF
```

```
/*********************************************************************
*                                                                    *
*          The PICmicro as an IEEE 1451.2 Compatible Smart           *
*               Transducer Interface Module (STIM)                   *
*                                                                    *
*********************************************************************
*                                                                    *
*   Filename:       stimdot2.c                                       *
*   Date:           06/27/2000                                       *
*   Revision:       1.8                                              *
*                                                                    *
*   Contributor:    Richard L. Fischer                               *
*   Company:        Microchip Technology Inc.                        *
*                                                                    *
*   Contributor:    Jeff Burch                                       *
*   Company:        Agilent Technologies                             *
*                                                                    *
*                                                                    *
*   Tools:          Hi-Tech PIC C Compiler V7.85                     *
*                   MPLAB 5.00.00                                    *
*                                                                    *
*********************************************************************
*                                                                    *
*   Files required:                                                  *
*                                                                    *
*                   stimdot2.c                                       *
*                                                                    *
*                                                                    *
*                   pic.h        (Hi-Tech file)                      *
*                   common.h                                         *
*                   stimdot2.h                                       *
*                                                                    *
*********************************************************************
*                                                                    *
*   Notes:                                                           *
*                                                                    *
*   Description:   Common STIM functions for 1451.2 communication    *
*                                                                    *
*********************************************************************/

       #include <pic.h>              // processor if/def file
       #include “common.h”
       #include “stimdot2.h”



void update_nint( void );
void post_inval_cmd( void );


////////////////////////////////////////////////////////////////
// Reading/Writing functions (Global)
////////////////////////////////////////////////////////////////

// Read a byte from the NCAP
uchar  rd_1451_byte( void )
{
    uchar result;                     // temp holding register for SSPBUF

    // NIOE is low/asserted on entry; SPI is quiescent; SSPIF is clear
    // global affected: aborted_1451_io; see description in header file

    // ensure frame alive; if not, exit w/o issuing NACK edge
    if ( aborted_1451_io )            // if global is true
      return ( 0x00 );                // return
```



```
    // Tell the NCAP that we are ready to read this byte.
    // do pacing handshake - enable flow of exactly one byte
    NACK_PIN ^= 1;                      // toggle NACK to NCAP

    for ( ; ; )                         // loop til frame done or byte received
    {                                   // global conveys which occurred
      if ( NIOE_PIN  )                  // test if *SS pin negated, frame done?
      {
        if ( SSPIF )                    // if SPI event true as well ...
          break;                        // addresses a race; frame DID finish
        else                            // if SPI event false then..
        {
          aborted_1451_io = 1;          // set global to true, indicates abort
          break;                        // exit for loop
        }
      }
      if ( SSPIF )                      // if SPI event true
        break;                          // exit for loop
    }

    // unload SPI buffer - good or not
    result = SSPBUF;                    // temp save of SSPBUF contents
    SSPIF = 0;                          // clear hardware event interrupt flag
    SSPBUF = 0x00;                      // set SSPBUF to known contents

    return ( result );                  // return with byte read from SPI
}


// Write a byte to the NCAP
uchar wr_1451_byte( uchar abyte )
{
    // NIOE is low/asserted on entry; SPI is quiescent; SSPIF is clear
    // global affected: aborted_1451_io ; see descr above

    // ensure frame alive; if not, exit w/o issuing NACK edge
    if ( aborted_1451_io )              // if global is true
      return ( 0x00 );

    // loop til previous frame done or byte done; global conveys which occurred
    if ( wrNackDone )                   //
    {
       for ( ; ; )
       {
         if ( NIOE_PIN  )               // test if *SS pin negated, frame done?
         {
           if ( SSPIF )                 // if SPI event true
             break;                     // addresses a race; frame DID finish
           else                         // if SPI event false then..
           {
             aborted_1451_io = 1;       // set global to true, indicates abort
             break;                     // exit for loop
           }
         }
         if ( SSPIF )                   // if SPI event true
           break;                       // exit for loop
      }
    }

    // load outbound byte
    SSPBUF = abyte;                     // load byte for NCAP to read
    SSPIF = 0;                          // clear hardware event interrupt flag

    // do pacing handshake - enable flow of exactly one byte
    NACK_PIN ^= 1;                      // toggle NACK to NCAP
```

```
    wrNackDone = 1;                    //
    return ( 0x00 );
}


// Test to see if the NCAP is aborting the 1451.2 transaction.
// Check NIOE and set the aborted_1451_io global
uchar frame_aborted( void )
{
    // check for NIOE deassertion, now or earlier in this frame
    if ( NIOE_PIN )                    // is NIOE (*SS) negated ( by NCAP )
      aborted_1451_io = 1;             // capture observance of frame abort

    return ( aborted_1451_io );        // return with condition
}


// Return the STIM's status to the NCAP (high byte then low byte)
void rd_status( void )
{
    if ( chnl_addr )                    // channels 1-255 status
    {
       wr_1451_byte( chan_status_hi[ chnl_addr-1 ] );

       // check for NIOE (*SS) deassertion, now or earlier this frame
      if ( aborted_1451_io )          // if global is true
         return;
       wr_1451_byte( chan_status_lo[ chnl_addr-1 ] );

       // clear bits after seen
       chan_status_lo[ chnl_addr-1 ] = 0x00;
       chan_status_hi[ chnl_addr-1 ] = STATUS_OPERATIONAL; // clear bits after seen, leave opera-
tional

    }
    else                                // else STIM global status
    {
       wr_1451_byte( global_status_hi );

       // check for NIOE (*SS) deassertion, now or earlier this frame
       if ( aborted_1451_io )          // if global is true
         return;
       wr_1451_byte( global_status_lo );

       // clear bits after seen
       global_status_lo = 0x00;     // clear bits after seen. Note: update_nint( ) will change
this.
       global_status_hi = STATUS_OPERATIONAL; // clear bits after seen, leave operational
    }

    update_nint();                     // on status or mask write update NINT
}


// Return the STIM's auxiliary status to the NCAP
void rd_aux_status( void )
{
#ifdef NO_AUX_STATUS_REGISTERS
    post_inval_cmd( );                 // call post_inval_cmd( )
#else
    if ( chnl_addr )                    // channels 1-255 auxiliary status
    {
      wr_1451_byte( chan_aux_status_hi[ chnl_addr-1 ] );

      // check for NIOE (*SS) deassertion, now or earlier this frame
      if ( aborted_1451_io )          // if global is true
```



```
        return;
      wr_1451_byte( chan_aux_status_lo[ chnl_addr-1 ] );

      // clear bits after seen
      chan_aux_status_hi[ chnl_addr-1 ] = 0x00;
      chan_aux_status_hi[ chnl_addr-1 ] = 0x00;

    }
    else                              // else STIM global auxiliary status
    {
       wr_1451_byte( global_aux_status_hi );

       // check for NIOE (*SS) deassertion, now or earlier this frame
       if ( aborted_1451_io )         // if global is true
         return;
       wr_1451_byte(global_aux_status_lo);

       // clear bits after seen. Note: update_nint( ) will change this.
       global_aux_status_lo = 0x00;
       global_aux_status_hi = 0x00;
    }

    update_nint();                    // on status or mask write update NINT
#endif
}


// Return the Trigger channel to the NCAP
void rd_trig_chnl_addr( void )
{
    wr_1451_byte( trig_chnl_addr );
}


// Set the Trigger channel. Check is performed to prevent setting the trigger address > than
// the number of valid channels.
void wr_trig_chnl_addr()
{
    uchar old_addr;

    old_addr = trig_chnl_addr;          // save off current trigger channel
    trig_chnl_addr = rd_1451_byte();    // read in new trigger channel

    if ( trig_chnl_addr > num_channels ) // test if new trigger is out of range
    {
      trig_chnl_addr = old_addr ;       // if so, restore trigger channel
      post_inval_cmd();                 // set STIM invalid command bit
    }
}



// Return the STIM firmware version. The global vsn_string is accessed.
// Note that this variable is in ROM. (access little slower then RAM, but saves RAM)
void rd_fw_version( void )
{
    unsigned uchar idx ;

    for ( idx=0; idx<sizeof(vsn_string) ; idx++ )
    {
     wr_1451_byte( vsn_string[idx] );

      // check for NIOE deassertion, now or earlier this frame
     if ( aborted_1451_io )          // if global is true
        return;
    }
```

```
}


// Initialize the STIM mask and auxiliary mask registers. Default power-up
// value for standard int mask register is all ones and all zeros for
// auxiliary int mask registers.
void init_mask_registers( void )
{
    uchar ch;
    for ( ch=0; ch<num_channels; ch++ )
    {
      chan_int_mask_hi[ ch ] = 0xff;   // all bits on
      chan_int_mask_lo[ ch ] = 0xfe;   // all bits on but LSB

#ifndef NO_AUX_STATUS_REGISTERS
      chan_aux_int_mask_hi[ ch ] = 0x00; // all bits off
      chan_aux_int_mask_lo[ ch ] = 0x00; // all bits off
#endif
    }

    global_int_mask_hi = 0xff;         // all bits on
    global_int_mask_lo = 0xfe;         // all bits on but LSB

#ifndef NO_AUX_STATUS_REGISTERS
    global_aux_int_mask_hi = 0x00;     // all bits off
    global_aux_int_mask_lo = 0x00;     // all bits off
#endif
}



// Initialize the STIM interrupt and aux interrupt registers
void init_int_registers( void )
{
    chnl_addr = 0x00;

    RESET_1451DOT2_COMMAND             // call standard_reset( );
}



// Return the Interrupt mask to the NCAP
void rd_int_mask( void )
{
    if ( chnl_addr )                   // channels 1-255 int mask
    {
      wr_1451_byte( chan_int_mask_hi[ chnl_addr-1 ] );

      // check for NIOE (*SS) deassertion, now or earlier this frame
      if ( aborted_1451_io )          // if global is true
         return;
      wr_1451_byte( chan_int_mask_lo[ chnl_addr-1 ] );

    }
    else                               // else STIM global int mask
    {
      wr_1451_byte( global_int_mask_hi );

      // check for NIOE (*SS) deassertion, now or earlier this frame
      if ( aborted_1451_io )          // if global is true
         return;
      wr_1451_byte( global_int_mask_lo );
    }
}
```



```
// Return the Auxiliary Interrupt mask to the NCAP
void rd_aux_int_mask( void )
{
#ifdef NO_AUX_STATUS_REGISTERS
    post_inval_cmd( );                // call post_inval_cmd( )
#else
    if ( chnl_addr )                  // channels 1-255 aux. int mask
    {
      wr_1451_byte( chan_aux_int_mask_hi[ chnl_addr-1 ] );

      // check for NIOE (*SS) deassertion, now or earlier this frame
      if ( aborted_1451_io )          // if global is true
         return;
      wr_1451_byte( chan_aux_int_mask_lo[ chnl_addr-1 ] );
    }
    else                              // else STIM global aux. int mask
    {
      wr_1451_byte( global_aux_int_mask_hi );

      // check for NIOE (*SS) deassertion, now or earlier this frame
      if ( aborted_1451_io )          // if global is true
         return;
      wr_1451_byte( global_aux_int_mask_lo );
    }
#endif
}


// Set the interrupt mask per NCAP command
void wr_int_mask( void )
{
    if ( chnl_addr )                  // channels 1-255 int mask
    {
      chan_int_mask_hi[ chnl_addr-1 ] = rd_1451_byte();

      // check for NIOE (*SS) deassertion, now or earlier this frame
      if ( aborted_1451_io )          // if global is true
         return;
      chan_int_mask_lo[ chnl_addr-1 ] = rd_1451_byte();
    }
    else                              // else STIM global int mask
    {
      global_int_mask_hi = rd_1451_byte();

      // check for NIOE (*SS) deassertion, now or earlier this frame
      if ( aborted_1451_io )          // if global is true
         return;
      global_int_mask_lo = rd_1451_byte();
    }

    update_nint();                // on status or mask write update NINT
    // maybe better (tho' slower) to update_nint at frame end
}


// Set the interrupt mask via NCAP
void wr_aux_int_mask( void )
{
#ifdef NO_AUX_STATUS_REGISTERS
    post_inval_cmd( );
#else
    if ( chnl_addr )                  // channels 1-255 aux. int mask
    {
      chan_aux_int_mask_hi[ chnl_addr-1 ] = rd_1451_byte();

      // check for NIOE (*SS) deassertion, now or earlier this frame
```

```
      if ( aborted_1451_io )          // if global is true
         return;
       chan_aux_int_mask_lo[ chnl_addr-1 ] = rd_1451_byte();
    }
    else                               // else STIM global aux. int mask
    {
      global_aux_int_mask_hi = rd_1451_byte();

      // check for NIOE (*SS) deassertion, now or earlier this frame
      if ( aborted_1451_io )          // if global is true
         return;
      global_aux_int_mask_lo = rd_1451_byte();
    }

    update_nint();                // on status or mask write update NINT
    // maybe better (tho' slower) to update_nint at frame end
#endif
}


// Process global commands (See specification secton 4.7)
void wr_ctrl_cmd( void )
{
    uchar cmd_lo, cmd_hi ;
    cmd_hi = rd_1451_byte();           // read command low byte

    // check for NIOE deassertion, now or earlier this frame
    if ( aborted_1451_io )          // if global is true
       return;

    cmd_lo = rd_1451_byte();           // read command high byte
    if ( cmd_hi )
    {
      DEFAULT_1451DOT2_COMMAND       // commands  > 255 not implemented at this time
    }                                  // call post_inval_cmd( )
    else
    {
      switch ( cmd_lo )
      {
        case COMMAND_NOP:              // "no operation" command (value 0)
            break;

        case COMMAND_RESET:            // reset STIM command (value 1)
           RESET_1451DOT2_COMMAND     // call standard_reset( );
            break;

        default:                       // else unimplmented command
            DEFAULT_1451DOT2_COMMAND   // call post_inval_cmd( )
            break;
      }
    }
}


// Post an invalid command
void post_inval_cmd( void )
{
    global_status_lo |= STATUS_INVALID_COMMAND;// set STIM invalid command bit
    update_nint();
}


// Update NINT depending on status and auxiliary status registers
void update_nint( void )
{
    uchar ch, start, stop;
```



```
    uchar tmp;

    if ( chnl_addr )                    // if channel 1-255
    {
      start = chnl_addr-1;
      stop = chnl_addr;
    }
    else                                // else, global channel 0
    {
      start = 0x00;
      stop = num_channels;              // maxmium channels on STIM
    }

    // We must walk through each channel to update the status bit for that channel
#ifdef NO_AUX_STATUS_REGISTERS

    // With no aux status registers, we must make sure this bit is clear
    chan_status_lo[ ch ] &= ~STATUS_AUX_STATUS;
#else
    /* Test if there are any auxiliary status bits set for a specific channel.
       If true then set the respective channel aux. status available bit. If
       false then clear the respective channel aux. status available bit. */
    for ( ch=start; ch<stop; ch++ )      // loop on required number of channels
    {
      if ( chan_aux_status_lo[ ch ] || chan_aux_status_hi[ ch ] ) // any bit set
      {
        chan_status_lo[ ch ] |= STATUS_AUX_STATUS;  // set bit
      }
      else
      {
        chan_status_lo[ ch ] &= ~STATUS_AUX_STATUS; // clear bit
      }
    }
#endif

    // Update the channel service request bit. We must 'and' with the mask to see
    // if this bit is enabled for interrupts (See specification section 4.9)
    for ( ch=start; ch<stop; ch++ )
    {
      if (( chan_status_lo[ ch ] & chan_int_mask_lo[ ch ] )
         | ( chan_status_hi[ ch ] & chan_int_mask_hi[ ch ] )
#ifndef NO_AUX_STATUS_REGISTERS
         | ( chan_aux_status_lo[ ch ] & chan_aux_int_mask_lo[ ch ] )
         | ( chan_aux_status_hi[ ch ] & chan_aux_int_mask_hi[ ch ] )
#endif
       )
      {
        chan_status_lo[ ch ] |= STATUS_SERVICE_REQUEST; // set bit
      }
      else
      {
        chan_status_lo[ ch ] &= ~STATUS_SERVICE_REQUEST; // clear bit
      }
    }

    // Update global status and auxiliary status registers by or'ing all the bits
    // For now, we treat the global bit to be an 'or' of the channel bits for:
    // auxiliary self-test, operational, reset, and trigger acknowledge.

    // Care is taken to preserve the invalid command bit.
    tmp = global_status_lo & STATUS_INVALID_COMMAND;
    global_status_lo = 0x00;
    global_status_hi = 0x00;
#ifndef NO_AUX_STATUS_REGISTERS
    global_aux_status_lo = 0x00;
    global_aux_status_hi = 0x00;
```

```
#endif

    // Caution: Because we are doing an ‘or’, we MUST loop through each channel
    for ( ch=0; ch<num_channels; ch++ )
    {
      global_status_lo |= chan_status_lo[ ch ];
      global_status_hi |= chan_status_hi[ ch ];
#ifndef NO_AUX_STATUS_REGISTERS
      global_aux_status_lo |= chan_aux_status_lo[ ch ];
      global_aux_status_hi |= chan_aux_status_hi[ ch ];
#endif
    }
    // restore invalid command bit
    global_status_lo |= tmp;

    /* Combine the status bits with the interrupt masks and see if there are
       valid matches. If true, set the global service request bit. If false,
       clear the global service request bit. */
    if ( ( global_status_lo & global_int_mask_lo )
         | ( global_status_hi & global_int_mask_hi )
#ifndef NO_AUX_STATUS_REGISTERS
         | ( global_aux_status_hi & global_aux_int_mask_hi )
         | ( global_aux_status_lo & global_aux_int_mask_lo )
#endif
       )
    {                                   // set global service request bit
      global_status_lo |= STATUS_SERVICE_REQUEST;
    }
    else
    {                                   // clear global service request bit
      global_status_lo &= ~STATUS_SERVICE_REQUEST;
    }

    // Now see if we need to generate an interrupt. Or all the service request bits
    tmp = global_status_lo & STATUS_SERVICE_REQUEST;

    if ( !tmp )
    {                       // Or in the individual channels only if needed. Stop on the first
‘true’
      for ( ch=0; ch<num_channels && !tmp; ch++ )
      {
        tmp |= chan_status_lo[ ch ] & STATUS_SERVICE_REQUEST;
      }
    }

    if ( tmp )                          // if there is a true compare state
    {
      NINT_PIN = 0;                     // generate the interrupt
    }
    else
    {
      NINT_PIN = 1;                     // negate the interrupt
    }
}


// Perform the standard trigger acknowledgement
void standard_trigger( void )
{
    uchar ch, start, stop;        // define auto type variables

    NACK_PIN = 0;                 // assert the acknowledge signal to NCAP
                                  // generate this edge *right after* sampling:

   // trigger ACK (NACK) is now asserted
   // let NACK assert until TRIG deasserts
```



```
   // if NCAP asserts TRIG forever, we stick here

// should we wait here forever? (Suggestion to use timer0 for loop break out)

    while( !NTRIG_PIN );              // wait until trigger is negated by NCAP
    NACK_PIN = 1;                     // negate the acknowledge signal to NCAP

   // Update the trigger acknowledge bit
   // We do this AFTER the trigger has been acknowledged
    if ( chnl_addr )                  // if channel 1-255
    {
        start = chnl_addr-1;
        stop = chnl_addr;
    }
    else                              // else, global channel 0
    {
        start = 0x00;
        stop = num_channels;
    }

    for ( ch=start; ch<stop; ch++ )
    {
        // Trigger acknowledge each channel
        chan_status_lo[ ch ] |= STATUS_TRIGGER_ACKNOWLEDGE ; // set trig. ack'd status bit
    }

    global_status_lo  |= STATUS_TRIGGER_ACKNOWLEDGE ; // set trig. ack'd status bit

    update_nint();                // on status write update NINT

}


// Perform the standard channel reset. Here we reset the status and
// auxiliary status to initial values. We return with the reset and
// operational bits set.
void standard_reset( void )
{
    uchar ch, start, stop;
    if ( chnl_addr )                  // if channel 1-255
    {
      start = chnl_addr-1;
      stop = chnl_addr;
    }
    else                              // else, global channel 0
    {
      start = 0x00;
      stop = num_channels;
    }

    for ( ch=start; ch<stop; ch++ )
    {
      chan_status_lo[ ch ] = STATUS_RESET;       // reset bit
      chan_status_hi[ ch ] = STATUS_OPERATIONAL; // leave operational
#ifndef NO_AUX_STATUS_REGISTERS
      chan_aux_status_hi[ ch ] = 0x00;           // all bits off
      chan_aux_status_lo[ ch ] = 0x00;           // all bits off
#endif
    }

    global_status_lo = STATUS_RESET;      // reset bit
    global_status_hi = STATUS_OPERATIONAL;// leave operational
#ifndef NO_AUX_STATUS_REGISTERS
    global_aux_status_hi = 0x00;
    global_aux_status_lo = 0x00;
#endif
```

```
    // note: spec says.. A STIM or channel reset command shall not
    // affect the value of the interrupt mask bits.
    update_nint();
}
```



```
/*********************************************************************
*                                                                    *
*   Filename:       stimdot2.h                                       *
*   Date:           06/27/2000                                       *
*   Revision:       1.4                                              *
*                                                                    *
*   Tools:          Hi-Tech PIC C Compiler V7.85                     *
*                   MPLAB 5.00.00                                    *
*                                                                    *
*                                                                    *
*                                                                   *
*   Description:    Variable references and macros utilized for      *
*                   1451.2 communications and process support.       *
*                                                                    *
*********************************************************************/




void standard_reset( void );


extern uchar frame_aborted(void);

extern struct events {
       unsigned int time_expire  :1;
       unsigned int              :7;
} flag;

extern uchar  aborted_1451_io;
extern uchar  wrNackDone;          // Write NACK control
extern uchar  chnl_addr;
extern uchar  chan_status_hi[ MAX_NUM_CHANNELS ];
extern uchar  chan_status_lo[ MAX_NUM_CHANNELS ];

extern uchar  global_status_lo;
extern uchar  global_status_hi;
extern uchar  chan_aux_status_lo[ MAX_NUM_CHANNELS ];
extern uchar  chan_aux_status_hi[ MAX_NUM_CHANNELS ];
extern uchar  global_aux_status_lo;
extern uchar  global_aux_status_hi;

extern uchar  trig_chnl_addr;

extern uchar  num_channels;
extern const  uchar vsn_string[4 + 0x13 ];

extern uchar  chan_int_mask_lo[ MAX_NUM_CHANNELS ];
extern uchar  chan_int_mask_hi[ MAX_NUM_CHANNELS ];

extern uchar  chan_aux_int_mask_lo[ MAX_NUM_CHANNELS ];
extern uchar  chan_aux_int_mask_hi[ MAX_NUM_CHANNELS ];

extern uchar  global_int_mask_lo;
extern uchar  global_int_mask_hi;

extern uchar  global_aux_int_mask_lo;
extern uchar  global_aux_int_mask_hi;



/*****************************************************************************/
// 1451.2 functional addresses

#define WRITE_TRANSDUCER_DATA          0
```

```
#define WRITE_CONTROL_CMD               1
#define WRITE_TRIGGERED_CHANNEL_ADDR    3
#define WRITE_INT_MASK                  5
#define WRITE_AUX_INT_MASK              6
#define WRITE_CALIBRATION_TEDS          64
#define WRITE_CALIBRATION_ID_TEDS       65

#define READ_TRANSDUCER_DATA            128
#define READ_STATUS                     130
#define READ_TRIGGERED_CHANNEL_ADDR     131
#define READ_AUX_STATUS                 132
#define READ_INT_MASK                   133
#define READ_AUX_INT_MASK               134
#define READ_FW_VERSION                 135

#define READ_TEDS                       160
#define READ_ID_TEDS                    161
#define READ_CALIBRATION_TEDS           192
#define READ_CALIBRATION_ID_TEDS        193

#define WRITE_APP_SPECIFIC_TEDS         96
#define READ_APP_SPECIFIC_TEDS          224
#define WRITE_FULL_TEDS_EEPROM          32
#define READ_FULL_TEDS_EEPROM           240



/*****************************************************************************/
// 1451.2 status bits (a 16 bit quantity) is stored in two 8-bit bytes for efficiency

// Lo byte
#define STATUS_SERVICE_REQUEST           0x01
#define STATUS_TRIGGER_ACKNOWLEDGE       0x02
#define STATUS_RESET                     0x04
#define STATUS_INVALID_COMMAND           0x08
#define STATUS_AUX_STATUS                0x10
#define STATUS_MISSED_DATA               0x20
#define STATUS_DATA_AVAILABLE            0x40
#define STATUS_HARDWARE_ERROR            0x80

// Hi byte
#define STATUS_OPERATIONAL               0x01


// 1451.2 interrupt bits, low byte
#define INT_MSK_SERVICE_REQUEST          0x01
#define INT_MSK_TRIGGER_ACKNOWLEDGE      0x02
#define INT_MSK_RESET                    0x04
#define INT_MSK_INVALID_COMMAND          0x08
#define INT_MSK_AUX_STATUS               0x10
#define INT_MSK_MISSED_DATA              0x20
#define INT_MSK_DATA_AVAILABLE           0x40
#define INT_MSK_HARDWARE_ERROR           0x80

// 1451.2 interrupt bits, high byte
#define INT_MSK_OPERATIONAL              0x01

// 1451.2 commands
#define COMMAND_NOP                      0x00
#define COMMAND_RESET                    0x01
#define COMMAND_INITIATE_SELF_TEST       0x02
#define COMMAND_CALIBRATE_CHANNELS       0x03
#define COMMAND_ZERO_ALL_CHANNELS        0x04
#define COMMAND_ENABLE_EVENT_SEQUENCE    0x05
#define COMMAND_DISABLE_EVENT_SEQUENCE   0x06
#define COMMAND_CONFIG_EVENT_SEQUENCE    0x07
#define COMMAND_ENABLE_DATA_SEQUENCE     0x09
#define COMMAND_DISABLE_DATA_SEQUENCE    0x0a
```



```
/*****************************************************************
*                                                                *
*          The PICmicro as an IEEE 1451.2 Compatible Smart       *
*               Transducer Interface Module (STIM)               *
*                                                                *
*****************************************************************
*                                                                *
*   Filename:       stiminit.c                                   *
*   Date:           06/27/2000                                   *
*   Revision:       1.2                                          *
*                                                                *
*                                                                *
*   Contributor:    Richard L. Fischer                           *
*   Company:        Microchip Technology Inc.                    *
*                                                                *
*   Contributor:    Jeff Burch                                   *
*   Company:        Agilent Technologies                         *
*                                                                *
*                                                                *
*   Tools:          Hi-Tech PIC C Compiler V7.85                 *
*                   MPLAB 5.00.00                                *
*                                                                *
*****************************************************************
*                                                                *
*   Files required:                                              *
*                                                                *
*                   stiminit.c                                   *
*                                                                *
*                   pic.h        (Hi-Tech file)                  *
*                   common.h                                     *
*                   stiminit.h                                   *
*                                                                *
*****************************************************************
*                                                                *
*   Notes:                                                       *
*                                                                *
*   Description:  Common STIM functions to initialize the PICmicro   *
*                                                                *
*   Port A: A5  input  NIOE     (SS, active low)                 *
*           A4  n/c                                              *
*          A3  n/c                                              *
*          A2  n/c                                              *
*          A1  n/c                                              *
*          A0  n/c                                              *
*                                                                *
*   Port B: B7  n/c                                              *
*           B6  n/c                                              *
*           B5  n/c                                              *
*           B4  output  EEPCS   initial value = 0                *
*           B3  input   EEPDOUT                                  *
*           B2  output  EEPDIN  initial value = 0                *
*           B1  output  EEPDCLK initial value = 0                *
*           B0  n/c                                              *
*                                                                *
*   Port C: C7  output  NACK    initial value = 1                *
*           C6  output  NINT    initial value = 1                *
*           C5  output  DOUT    initial value = 0                *
*           C4  input   DIN                                      *
*           C3  input   DCLK    (idle high)                      *
*           C2  input   NTRIG   (active low)                     *
*           C1  n/c                                              *
*           C0  n/c                                              *
*                                                                *
*****************************************************************/

        #include <pic.h>              // processor if/def file
```

```
       #include “common.h”
       #include “stiminit.h”


////////////////////////////////////////////////////////////////
//       PICmicro initialization
////////////////////////////////////////////////////////////////

// Initialize the H/W SPI port
void init_spi( uchar await_nioe )
{
    SSPCON = 0b00010100;               // set: CKP & SPI slave w/nioe (*SS)
    SSPBUF = 0;                        // set SSPBUF to known state
    SSPIF = 0;                         // reset H/W event interrupt flag
    wrNackDone = 0;                    // reset global

    if ( await_nioe )                  // test if requiring to test pin
    {
      while ( !NIOE_PIN );             // wait while NIOE (*SS) pin is low/asserted
    }                                  // this signal is negated by NCAP
    SSPCON = 0b00110100;               // set: same but SSPEN now added
}


// Initialize the PICmicro I/O Ports
void init_pic_ports( void )
{
    // To avoid any glitches, set port states before setting direction

    PORTA = INIT_A_VAL;                // Initialize PORTA data latch
    PORTB = INIT_B_VAL;                // Initialize PORTB data latch
    PORTC = INIT_C_VAL;                // Initialize PORTC data latch

// next instruction required for analog type parts
//  ADCON1 = 0b00000111;               // Set PORTA for inputs
    TRISA = TRIS_A_VAL;                // Initialize PORTA direction latch
    TRISB = TRIS_B_VAL;                // Initialize PORTB direction latch
    TRISC = TRIS_C_VAL;                // Initialize PORTC direction latch

    PIE1 &= PIE1_MSK;                  // squelch SPI based IRQ

    init_spi( 1 );                     // 1 =>  await NIOE deasserting
}
```



```
/*********************************************************************
*                                                                    *
*   Filename:       stiminit.h                                       *
*   Date:           06/27/2000                                       *
*   Revision:       1.4                                              *
*                                                                    *
*   Tools:          Hi-Tech PIC C Compiler V7.85                     *
*                   MPLAB 5.00.00                                    *
*                                                                    *
*                                                                    *
*   Description:    PICmicro PORT and TRIS definitions               *
*                                                                    *
*********************************************************************/



extern uchar wrNackDone;


#ifndef TRIS_A_VAL
#define TRIS_A_VAL      0b00111111
#endif

// Port A: all inputs
#ifndef INIT_A_VAL
#define INIT_A_VAL      0b00000000
#endif


#ifndef TRIS_B_VAL
#define TRIS_B_VAL      0b11101001
#endif

// Port B: all outputs low
#ifndef INIT_B_VAL
#define INIT_B_VAL      0b00000000
#endif


#ifndef TRIS_C_VAL
#define TRIS_C_VAL      0b00011111
#endif

// Port C: NINT and NACK high
#ifndef INIT_C_VAL
#define INIT_C_VAL      0b11000000
#endif


#define PIE1_MSK        0b11110111  /* squelches SPI IRQ e.g. */
```

```
/*********************************************************************
*                                                                    *
*          The PICmicro as an IEEE 1451.2 Compatible Smart           *
*               Transducer Interface Module (STIM)                   *
*                                                                    *
*********************************************************************
*                                                                    *
*   Filename:       stimteds.c                                       *
*   Date:           06/27/2000                                       *
*   Revision:       1.3                                              *
*                                                                    *
*   Contributor:    Richard L. Fischer                               *
*   Company:        Microchip Technology Inc.                        *
*                                                                    *
*   Contributor:    Jeff Burch                                       *
*   Company:        Agilent Technologies                             *
*                                                                    *
*                                                                    *
*   Tools:          Hi-Tech PIC C Compiler V7.85                     *
*                   MPLAB 5.00.00                                    *
*                                                                    *
*********************************************************************
*                                                                    *
*   Files required:                                                  *
*                                                                    *
*                   stimteds.c                                       *
*                                                                    *
*                   pic.h        (Hi-Tech file)                      *
*                   common.h                                         *
*                   stimteds.h                                       *
*                   delays.h     (Hi-Tech file)                      *
*                                                                    *
*********************************************************************
*                                                                    *
*   Notes:                                                           *
*                                                                    *
*   Description:     Common STIM functions for accessing TEDS        *
*                                                                    *
*********************************************************************/

       #include <pic.h>              // processor if/def file
       #include “common.h”           // project commons, etc.
       #include “stimteds.h”         //
       #include “c:\ht-pic\samples\delay.h”



extern uchar  eeprom_data;             // data for read/write ops.
extern uchar  ee_rd_wr_bit_cntr;


// EEPROM related constants - opcodes, w/start_bit

#ifndef EEPROM_BYTES
#define EEPROM_BYTES 2048              // x8 bit organization
#endif

#define LEN_OF_GLOBAL_EE_OPCODE   5



// utility for clocking up to one byte out the DI line to the eeprom
void send_ee_bits( uchar databits, uchar bitcount )
{
    // clock out on DI the ‘bitcount’ lsb’s of ‘databits’
```



```
    // note: 2MHz bit rate for Vdd >= 4.5Vdc

    // if using shift_left, must get first output bit left-justified
    if ( bitcount < 8 ) // careful: at least one caller uses > 8
        databits <<= ( 8 - bitcount ) ;

// loop dynamics based on 10MHz Fosc; if changed Nops may be required
    while ( bitcount-- )
    {
      EE_DI_PIN = 0;                  // set data pin low

      databits <<= 1;                 // shift left 1 bit position
      if ( CARRY )                    // test for carry
      {
        EE_DI_PIN = 1;                // set data pin high
      }

      EE_SK_PIN = 1;                  // Tckh min = 300nS
      EE_SK_PIN = 0;                  // Tckl min = 200nS
    }
}



// these are a bit conservative w.r.t. setup/hold/speed ...
uchar ee_read( unsigned int addr )
{
    //  eeprom_data carries back the 8-bit payload
    //  procedure:
    //  DI, SK are kept low between operations so they are low on entry here
    //  assert CS
    //  send start bit, 2 bits opcode
    //  send 3 msb's of address (for 2048x8 organization)
    //  send 8 lsb's of address
    //  lower DI
    //  discard leading 0 startbit
    //  read 8 bits of data
    //  deassert CS ; SK is lowered by final bit send, DI was lowered above

      EE_CS_PIN = 1;                  // assert eeprom chip select

    // issue opcode
    send_ee_bits( EE_READ_OP, LEN_OF_BYTEWISE_EE_OPCODE ); // args: data,count

    // msb's of address -> 3 bits
    send_ee_bits( (uchar)(addr>>8), EE_ADDR_LEN-8 ); // args: data,count

    // lsb's of address -> 8 bits
    send_ee_bits( (uchar)addr, 8 );

      EE_DI_PIN = 0;                  // set EE DI to logic 0
    // the discarding of a start bit is implicit; no extra SK edges
    // need be generated

    eeprom_data = 0x00;               // byte to be returned
    ee_rd_wr_bit_cntr = 8;            // init bit counter

    //  procedure to read each data bit MSb to LSb:
    //  raise SK
    //  wait >= Tpd (400ns for Microchip's 93C86)
    //  sample EE_DO pin, stuff/shift into appropriate capture bit
    //  lower SK

    while ( ee_rd_wr_bit_cntr-- )
    {
      EE_SK_PIN = 1;                  // set clock pin high
```

```
      eeprom_data <<= 1;           // shift composed byte by 1

      // Tpd provided inherently by prior instructions
      if ( EE_DO_PIN )                // test EE data out pin is high
      {
        eeprom_data |= 1;             // EE_DO is logic 1 so set LSb
      }

      EE_SK_PIN = 0;                  // set clock pin low
     }

    EE_CS_PIN = 0;                    // negate eeprom chip select
    return ( 0 );                     //
}


uchar ee_write( unsigned int addr )
{
    //  eeprom_data carries in the 8-bit payload

    //  procedure:
    //  DI, SK are kept low between operations so they are low on entry here
    //  assert CS
    //  send start bit, 2 bits opcode
    //  send 3 msb's of address (for 2048x8 organization)
    //  send 8 lsb's of address
    //  send 8 bits of data
    //  wait out "busy" phase of eeprom (on DO)
    //  deassert CS & lower DI; SK is lowered by final bit send

    EE_CS_PIN = 1;                    // assert eeprom chip select

    // issue opcode
    send_ee_bits( EE_WRITE_OP, LEN_OF_BYTEWISE_EE_OPCODE ); // args: data,count

    // msb's of address -> 3 bits
    send_ee_bits( (uchar)(addr>>8), EE_ADDR_LEN-8 ); // args: data,count

    // lsb's of address -> 8 bits
    send_ee_bits( (uchar)addr, 8 );

    // data bits
    send_ee_bits( eeprom_data, 8 );

    // wait out 'busy'; complete ; on NIOE up (NCAP timeout e.g.) abort
    // a Microchip 93C86 takes <= 10ms maximum, 4ms typical per word
    // Microchip self-timed programming cycle is initiated on rising edge of
    // SK as the last data bit (DO) is clocked out.

    // wait for EE done bit
    // RH 980803 - if frame is aborting, note it, but let EE finish,
    // allow ~30ms max or so here - 180us*166
    for ( ee_rd_wr_bit_cntr =166;  ee_rd_wr_bit_cntr;  ee_rd_wr_bit_cntr-- )
    {
      DelayUs( 180 );                 // too much => EOF latency suffers on aborted frame
        if ( EE_DO_PIN )              // test if EE device ready
          break;                      // byte completed

        // check for NIOE deassertion, now or earlier this frame:
        frame_aborted();              // sets aborted_1451_io if NIOE deasserts
    }

    // EE_DI = EE_CS = 0 ; // note: CS hold after final SK drop can even be 0

    EE_DI_PIN = 0;                    // compile trouble above
    EE_CS_PIN = 0;                    // deassert eeprom chip select
```



```
    return ( 0 );
}



uchar ee_wr_enable( uchar opcode )
{
    // ee_wr_enable arg1: EE_EWEN_OP or EE_EWDS_OP (enable or disable)
    // DI, SK are kept low between operations so they are low on entry here
    // first, assert CS, then
    // send: start bit, 2 bits opcode, 2 bits msb address, rest of dummy addr
    // finally, deassert CS & lower DI; SK is lowered by final bit send

    EE_CS_PIN = 1;                    // assert eeprom chip select

    send_ee_bits( opcode, LEN_OF_GLOBAL_EE_OPCODE );
    send_ee_bits( 0, EE_ADDR_LEN-2 );  // these (~9) bits have don't care values;
                                       // therefore the 8 bit "limit" of
                                       // send_ee_bits() is ok

    // EE_DI = EE_CS = 0 ; // note: CS hold after final SK drop can even be 0
    EE_DI_PIN = 0;                    // compile trouble above
    EE_CS_PIN = 0;

    return ( 0 );
}


// ?? Left off here

////////////////////////////////////////////////////////////////
// TEDS Reading Support
////////////////////////////////////////////////////////////////

// consult separate documentation regarding TEDS EEPROM layout
// to understand what follows

void rd_teds_block( uchar blocknum )
{
    // it's assumed that 65536 is max teds block size
    // teds address calculating could be shared better between rd/wr...
    // get addr of ptr to 0th block

    ee_read( 1 );            // eeprom_data := MSB
    teds_addr = eeprom_data;
    teds_addr <<= 8;

    ee_read( 0 );            // eeprom_data := LSB
    teds_addr += eeprom_data;
    // teds_addr has addr of ptr to 0th/1st teds block

    // below CCS went awry, so split expression
    teds_addr += blocknum;   // teds_addr := addr of ptr to desired blk
    teds_addr += blocknum;   // teds_addr := addr of ptr to desired blk

    ee_read( teds_addr+1 );  // MSB
    teds_addr2 = eeprom_data;
    teds_addr2 <<= 8;

    ee_read( teds_addr );    // LSB
    teds_addr2 += eeprom_data;

    // teds_addr2 is now addr of desired block

    // send the block of teds.
```

```
    // serve bytes until NCAP quits.
    // there are ~4 length bytes to start, and 2 checksum bytes at the end.
    // the length bytes don't count themselves but do count the checksum bytes.
    // the checksum accounts for the length bytes but not the cksum bytes.
    // i.e. we send start byte, some length bytes, and then N additional bytes,
    // where N is the value held in the length bytes

    for( ; ; )
    {
        ee_read( teds_addr2++ );
        // check for NIOE deassertion, now or earlier this frame
        if ( frame_aborted() )
          break;
        wr_1451_byte( eeprom_data );
    }
}



////////////////////////////////////////////////////////////////
//  TEDS Reading functions (Global)
//     limited to 65536 bytes
////////////////////////////////////////////////////////////////

// read our prolog/catalog/directory at start of eeprom;
// init. teds_format_vsn, teds_blks_per_chnl, num_channels

void process_teds_prolog( void )
{
    // get supported chnl count from teds, to validate chnl_addr's later
    ee_read( 2 );                       // fixed addr for chnl count
    num_channels = eeprom_data;
    if ( num_channels > MAX_NUM_CHANNELS || (!num_channels) )
        num_channels = MAX_NUM_CHANNELS; // sanity check in case e^2 absent e.g.

    // get other metadata from teds - image format version, etc.
    // note: if addr of ptr block > 4 ; this prolog includes a vsn_no.
    // (before 970425, the hpl prolog lacked this)
    ee_read( 0 );            // low byte of addr of ptr blk
    teds_format_vsn = 1;     // 1 here denotes assumption that E^2 holds a vsn_no
    teds_blks_per_chnl = 3;  // until 970425 -- chnl,chnl_id,cal
    if ( eeprom_data <= 4 )
    {
        ee_read( 1 );        // hi byte of addr of ptr blk
        if ( eeprom_data == 0 )
        {
            // old-ish teds image; has no vsn_no; call it 0
            teds_format_vsn = 0;
        }
    }
    if ( teds_format_vsn )
    {
        ee_read( 4 );        // fixed addr for image format version
        teds_format_vsn = eeprom_data; // needn't keep around probably
        ee_read( 6 );        // should base addr on teds_format_vsn
        teds_blks_per_chnl = eeprom_data;
    }

    AFTER_PROCESS_TEDS_PROLOG
}


// we're limited to 65KB TEDS below (EEPROMs for NSP3 are smaller)

void rd_meta_teds( void )
{
```



```
    rd_teds_block( 0 );
}


void rd_meta_id_teds( void )
{
    rd_teds_block( 1 );
}


void rd_full_teds_eeprom( void )
{
    // note: we cease reading bytes when EEPROM capacity exhausted
    teds_addr2 = 0;
    while ( teds_addr2 < EEPROM_BYTES )
    {
        ee_read( teds_addr2++ );
        // check for NIOE deassertion, now or earlier this frame
        if ( frame_aborted() )
          break;
        wr_1451_byte( eeprom_data );
    }
}


////////////////////////////////////////////////////////////////
// TEDS Reading functions (channel-specific)
//  limited to 65536 bytes
////////////////////////////////////////////////////////////////

void rd_channel_teds( void )
{
    rd_teds_block( 2+ ( chnl_addr-1 ) * teds_blks_per_chnl );
}


void rd_channel_id_teds( void )
{
    rd_teds_block( 3+ ( chnl_addr-1 ) * teds_blks_per_chnl );
}


void rd_calibration_teds( void )
{
    rd_teds_block( 4 + ( chnl_addr-1 ) * teds_blks_per_chnl );
}


void rd_calibration_id_teds( void )
{
    if ( teds_blks_per_chnl >= 4 )
        rd_teds_block( 5 + ( chnl_addr-1 ) * teds_blks_per_chnl );
}


void wr_calibration_id_teds( void )
{
    // FIXME: not implemented at this time
    post_inval_cmd( );
}


////////////////////////////////////////////////////////////////
// TEDS Writing functions (global)
////////////////////////////////////////////////////////////////
```

```
void wr_full_teds_eeprom( void )
{
    // we expect host to wait “Write Cal TEDS Time” between bytes
    // note: we cease reading and burning bytes when EEPROM capacity exhausted
    // it’s assumed that 65536 is max teds block size

    // Check write_protect.
    // This macro should call post_inval_cmd( ) and return if the TEDS is
    // write protected
    CHECK_WRITE_PROTECT

    ee_wr_enable( EE_EWEN_OP );        // write-enable the eeprom

    // accept and copy teds bytes to e^2
    // we maybe should confirm checksum and size - no need,
    // ncap will read back and check

    teds_addr2 = 0;
    while ( teds_addr2 < EEPROM_BYTES )
    {
        if ( frame_aborted() )
          break;
        eeprom_data = rd_1451_byte();  // try to get next byte
        if ( aborted_1451_io )
          break ;                     // don’t write if no byte given
        ee_write( teds_addr2++ );      // this can take a while, e.g. 10ms
    }

    ee_wr_enable( EE_EWDS_OP );        // write-disable the eeprom

    // in case ted_format_vsn or num_channels etc. is affected
    process_teds_prolog();   // init e.g. teds_format_vsn, teds_blks_per_chnl
}


////////////////////////////////////////////////////////////////////
// TEDS Writing functions (channel-specific)
////////////////////////////////////////////////////////////////////

void wr_calibration_teds( void )
{
    // we expect host to wait “Write Cal TEDS Time” between bytes
    // note: we cease reading and burning bytes when EEPROM capacity exhausted
    // it’s assumed that 65536 is max teds block size

    // write cal teds is not to be conditioned on jumper
    // // check write_protect jumper - if A4 is down, jumper is in, abort

    if ( !RA4 )
    {
      post_inval_cmd();
      return ;
    }

    // teds address calculating could be shared better between rd/wr...

    // get addr of ptr to 0th block
    ee_read( 1 ); // eeprom_data := MSB
    teds_addr = eeprom_data;
    teds_addr <<= 8;
    ee_read( 0 ); // eeprom_data := LSB
    teds_addr += eeprom_data;
    // teds_addr has addr of ptr to 0th/1st teds block

    // below CCS went awry, so split expr.
```



```
    teds_addr += 4; // teds_addr := addr of ptr to desired blk
    teds_addr += 4; // teds_addr := addr of ptr to desired blk

    ee_read( teds_addr+1 ); // MSB
    teds_addr2 = eeprom_data;
    teds_addr2 <<= 8;

    ee_read( teds_addr ); // LSB
    teds_addr2 += eeprom_data;

    // teds_addr2 is now addr of desired block

    // proceed to accept and copy teds bytes to e^2
    // there are ~4 length bytes to start, and 2 checksum bytes at the end.

    ee_wr_enable( EE_EWEN_OP ); // write-enable the eeprom
    while ( teds_addr2 < EEPROM_BYTES )
    {
        if ( frame_aborted() )
          break ;
        eeprom_data = rd_1451_byte();  // try to get next byte
        if ( aborted_1451_io )
          break;                       // don't write if no byte sent
        ee_write( teds_addr2++ );      // this can take a while, e.g. 10ms
    }
    ee_wr_enable( EE_EWDS_OP );        // write-disable the eeprom
}
```

```
/*********************************************************************
*                                                                    *
*   Filename:       stimteds.h                                       *
*   Date:           06/27/2000                                       *
*   Revision:       1.4                                              *
*                                                                    *
*   Tools:          Hi-Tech PIC C Compiler V7.85                     *
*                   MPLAB 5.00.00                                    *
*                                                                    *
*                                                                    *
*   Description:    Variable/Function references and PICmicro pin    *
*                   definitions for communicating with TEDS.         *
*                                                                    *
*********************************************************************/



extern uchar  frame_aborted( void );

extern unsigned int  teds_addr;
extern unsigned int  teds_addr2;

extern uchar num_channels;

extern uchar  teds_format_vsn;
extern uchar  teds_blks_per_chnl;
extern uchar  chnl_addr;


extern uchar wr_1451_byte( uchar abyte );
extern uchar  rd_1451_byte( void );

extern uchar frame_aborted( void );
extern uchar  aborted_1451_io;
extern void  post_inval_cmd( void );



// EEPROM related constants - pins; size

#ifndef EE_CS_PIN
#define EE_CS_PIN  RB4      /* 93C86 pin 1 - chip select           */
#endif

#ifndef EE_SK_PIN
#define EE_SK_PIN  RB1      /*       pin 2 - clock to eeprom       */
#endif

#ifndef EE_DI_PIN
#define EE_DI_PIN  RB2      /*       pin 3 - data into eeprom      */
#endif

#ifndef EE_DO_PIN
#define EE_DO_PIN  RB3      /*       pin 4 - data out of eeprom    */
#endif



// Define this if the STIM has write-protection circuitry for the TEDS
#ifndef CHECK_WRITE_PROTECT
#define CHECK_WRITE_PROTECT
#endif

// Define this if you want to be called after the TEDS prolog is processed
#ifndef AFTER_PROCESS_TEDS_PROLOG
#define AFTER_PROCESS_TEDS_PROLOG
```



```
#endif


#define EE_READ_OP      0b110     /* start_bit included in each of these */
#define EE_ERASE_OP     0b111
#define EE_WRITE_OP     0b101     /* implicitly erases too */
#define LEN_OF_BYTEWISE_EE_OPCODE 3
#define EE_ADDR_LEN     11        /* ORG pin=0 i.e. “by 8” EEPROM orgnzn */


// next 4 distinguished by address’s 2 msbits i.e. last 2 bits here
#define EE_EWEN_OP      0b10011   /* start_bit included in each of these */
#define EE_EWDS_OP      0b10000
#define EE_ERAL_OP      0b10010
#define EE_WRAL_OP      0b10001



/
```

```
/*********************************************************************
*                                                                    *
*          The PICmicro as an IEEE 1451.2 Compatible Smart           *
*               Transducer Interface Module (STIM)                   *
*                                                                    *
*********************************************************************
*                                                                    *
*   Filename:       stimtemp.c                                       *
*   Date:           06/27/2000                                       *
*   Revision:       1.8                                              *
*                                                                    *
*   Contributor:    Richard L. Fischer                               *
*   Company:        Microchip Technology Inc.                        *
*                                                                    *
*   Contributor:    Jeff Burch                                       *
*   Company:        Agilent Technologies                             *
*                                                                    *
*                                                                    *
*   Tools:          Hi-Tech PIC C Compiler V7.85                     *
*                   MPLAB 5.00.00                                    *
*                                                                    *
*********************************************************************
*                                                                    *
*   Files required:                                                  *
*                                                                    *
*                   stimtemp.c                                       *
*                                                                    *
*                                                                    *
*                   pic.h        (Hi-Tech file)                      *
*                   common.h                                         *
*                   stimtemp.h                                       *
*                                                                    *
*********************************************************************
*                                                                    *
*   Notes:                                                           *
*                                                                    *
*   Description:     Template for developing new STIM firmware       *
*                                                                    *
*********************************************************************/

       #include <pic.h>               // processor if/def file
       #include “common.h”
       #include “stimtemp.h”



// This template illustrates recommended STIM coding conventions. This code is
// fully runable and implements a two channel loop-back STIM. Channel 1 is a
// 16 bit actuator. Channel 2 is a 16 bit sensor. When the device is triggered,
// the cached actuator value is copied to the sensor data buffer.


// === Step 1:Define how many channels are implemented on this STIM ===

// Note: this value is used to allocate memory for interrupt masks and status registers
// The STIM may have less channels than this as specified in the TEDS. See the
// global variable num_channels;
// #define MAX_NUM_CHANNELS 2; modify this define statement in the common.h file


// === Step 2:  Generate the STIM Version string with the ‘generateVersion’ tool ===

// STIM Version string
// Generated by generateVersion STIM_Template_1.0
const uchar vsn_string[ 4 + 0x13 ] = {
        0x00, 0x00, 0x00, 0x13,
```



```
        'S', 'T', 'I', 'M', '_', 'T', 'e', 'm', 'p', 'l', 'a', 't', 'e', '_', '1', '.', '0',
        0xfa, 0x26,
};


// === Step 4:  Define the direction and initial values for each PIC port ===

// Port A:  A5  inputNIOE
//          A4  n/c
//          A3  n/c
//          A2  n/c
//          A1  n/c
//          A0  n/c
#define TRIS_A_VAL      0b00111111
#define INIT_A_VAL      0b00000000

// Port B:  B7  n/c
//          B6  n/c
//          B5  n/c
//          B4  outputEEPCSinitial value = 0
//          B3  inputEEPDOUT
//          B2  outputEEPDIninitial value = 0
//          B1  outputEEPDCLKinitial value = 0
//          B0  n/c
#define TRIS_B_VAL      0b11101001
#define INIT_B_VAL      0b00000000

// Port C:  C7  outputNACKinitial value = 1
//          C6  outputNINTinitial value = 1
//          C5  outputDOUTinitial value = 0
//          C4  inputDIN
//          C3  inputDCLK
//          C2  inputNTRIG
//          C1  n/c
//          C0  n/c
#define TRIS_C_VAL      0b00011111
#define INIT_C_VAL      0b11000000


// === Step 5:  Define any macros to override default behaviors ===

// None in this case...
// #define NO_AUX_STATUS_REGISTERS; modify this define statement in the common.h file


// === Step 6:  Declare these required functions ===

void transducer_initialize(void);
void transducer_trigger(void);
void transducer_read(void);
void transducer_write(void);


// === Step 7:  Implement the required functions ===

// This example supports a two channel STIM. Channel 1 is an actuator.
// Its value is 'looped back' to the Channel 2 sensor
unsigned int channel1Val;           // uint 16 data model
unsigned int channel2Val;           // uint 16 data model


// --- Initialize the transducer ---
void transducer_initialize(void)
{
    // This is called after the tri-state registers have been configured.
    // Nothing to do in this example
```

```
}

// ----- Perform the trigger and ack operation ----
void transducer_trigger( void )
{
// here is where we need to act on some transducer, whether it be an ADC or something else

    // Perform trigger operation. In this example,
    // we loop back the actuator to the sensor
    channel2Val = channel1Val;

    standard_trigger( );          // Call the generic trigger processing function
}


// ----- Read the transducer --------
void transducer_read( void )
{
    // The NCAP is reading data from the STIM. Use the wr_1451_byte( )
    // function to send data out the 1451.2 interface
    switch ( chnl_addr )
    {
      case 0:                            // Channel 0 - Read global transducer data
       wr_1451_byte(( uchar )( channel1Val>>8 ));// high byte
       wr_1451_byte(( uchar )( channel1Val ));// low byte
       wr_1451_byte(( uchar )( channel2Val>>8 ));// high byte
       wr_1451_byte(( uchar )( channel2Val ));// low byte
      break;

      case 1:                           // Read channel 1 transducer data
         wr_1451_byte(( uchar )( channel1Val>>8 ));// high byte
         wr_1451_byte(( uchar )( channel1Val ));    // low byte
      break;

      case 2:                           // Read channel 2 transducer data
         wr_1451_byte(( uchar )( channel2Val>>8 )); // high byte
         wr_1451_byte(( uchar )( channel2Val ));    // low byte
      break;

      default:                          // Illegal command
         post_inval_cmd();              // function in stimdot2.c file
      break;                            // purpose: set STIM invalid command bit
    }
}


// ----- Write the transducer --------
void transducer_write( void )
{
    unsigned int dummy;

    // The NCAP is writing data to the STIM. Use the rd_1451_byte( )
    // function to read data from the 1451.2 interface
    switch ( chnl_addr )
    {
      case 0:                   // Channel 0 - Write global transducer data
       channel1Val = rd_1451_byte() << 8;// high byte, actuator write
       channel1Val |= rd_1451_byte();  // low byte
       dummy = rd_1451_byte() << 8;      // high byte, discard the sensor write
       dummy |= rd_1451_byte();          // low byte
      break;

      case 1:                             // Channel 1 write (actuator)
       channel1Val = rd_1451_byte() << 8; // high byte
       channel1Val |= rd_1451_byte();     // low byte
      break;
```



```
      default:                              // Illegal command
        post_inval_cmd();                 // function in stimdot2.c file
      break;                                // purpose: set STIM invalid command bit
    }

    // Note: we cache the value but don't transfer it to the loop back
    // sensor until we get the trigger.
}
```

```
/*********************************************************************
*                                                                    *
*   Filename:       stimtemp.h                                       *
*   Date:           06/27/2000                                       *
*   Revision:       1.4                                              *
*                                                                    *
*   Tools:          Hi-Tech PIC C Compiler V7.85                     *
*                   MPLAB 5.00.00                                    *
*                                                                    *
*   Notes:                                                           *
*                                                                   *
*   Description:     Variable and function references for the        *
*                    template.c file                                 *
*                                                                    *
*********************************************************************/


/* Reference Linkage */

extern void  standard_trigger( void );
extern void  post_inval_cmd( void );
extern uchar wr_1451_byte( uchar abyte );
extern uchar rd_1451_byte( void );

extern uchar chnl_addr;
```



```
/*********************************************************************
*                                                                    *
*          The PICmicro as an IEEE 1451.2 Compatible Smart           *
*               Transducer Interface Module (STIM)                   *
*                                                                    *
*********************************************************************
*                                                                    *
*   Filename:       stimcmd.c                                        *
*   Date:           06/27/2000                                       *
*   Revision:       1.2                                              *
*                                                                    *
*   Contributor:    Richard L. Fischer                               *
*   Company:        Microchip Technology Inc.                        *
*                                                                    *
*   Contributor:    Jeff Burch                                       *
*   Company:        Agilent Technologies                             *
*                                                                    *
*                                                                    *
*   Tools:          Hi-Tech PIC C Compiler V7.85                     *
*                   MPLAB 5.00.00                                    *
*                                                                    *
*********************************************************************
*                                                                    *
*   Files required:                                                  *
*                                                                    *
*                   stimcmd.c                                        *
*                                                                    *
*                   common.h                                         *
*                   stimdot2.h                                       *
*                   stimcmd.h                                        *
*                                                                    *
*********************************************************************
*                                                                    *
*   Notes:                                                           *
*                                                                    *
*   Description:   This file is used to process the functional       *
*                  and channel address commands issued by the NCAP   *
*                  over the 1451.2 interface. The functions          *
*                  called are located in the stimtemp.c,             *
*                  stimteds.c and stimdot2.c files.                  *
*                                                                    *
*********************************************************************/

        #include “common.h”
        #include “stimdot2.h”          //
        #include “stimcmd.h”           //


// Process the command

void process_valid_cmd( void )
{
    switch ( func_addr )                    // evaluate functional address
    {
      // This group of function codes are valid for both channel 0 and real channels

        case READ_TRANSDUCER_DATA:          // Read address 128
          transducer_read( );               // function in stimtemp.c
          break;

        case WRITE_TRANSDUCER_DATA:         // Write address 0
          transducer_write( );              // function in stimtemp.c
          break;
//->
        case WRITE_CONTROL_CMD:             // Write address 1
          wr_ctrl_cmd();                    // function in stimdot2.c
```

```
  break;

case READ_STATUS:                   // Read address 130
  rd_status();                      // function in stimdot2.c
  break;

case READ_AUX_STATUS:               // Read address 132
  rd_aux_status();                  // function in stimdot2.c
  break;

case READ_INT_MASK:                 // Read address 133
  rd_int_mask();                    // function in stimdot2.c
  break;

case WRITE_INT_MASK:                // Write address 5
  wr_int_mask();                    // function in stimdot2.c
  break;

case READ_AUX_INT_MASK:             // Read address 134
  rd_aux_int_mask();                // function in stimdot2.c
  break;

case WRITE_AUX_INT_MASK:            // Write address 6
  wr_aux_int_mask();                // function in stimdot2.c
  break;

default:

if ( chnl_addr )                    // Channel 1-255 Function
{
  // The following group of function codes are for real channels only
  switch ( func_addr )
  {
    case READ_TEDS:                 // Read address 160
      rd_channel_teds( );           // function in stimteds.c
      break;

    case READ_ID_TEDS:              // Read address 161
      rd_channel_id_teds( );        // function in stimteds.c
      break;

    case WRITE_CALIBRATION_TEDS:    // Write address 64
      wr_calibration_teds( );       // function in stimteds.c
      break;

    case READ_CALIBRATION_TEDS:     // Read address 192
      rd_calibration_teds( );       // function in stimteds.c
      break;

    case READ_CALIBRATION_ID_TEDS:  // Read address 193
      rd_calibration_id_teds( );    // function in stimteds.c
      break;

    case WRITE_CALIBRATION_ID_TEDS: // Write address 65
      wr_calibration_id_teds( );    // function in stimteds.c
      break;

    default:
    // Allow user to provide additional functions
    DEFAULT_1451DOT2_FUNCTION
    break;
  }
}

else                                // else Global Channel 0
{
```



```
        // The following group of function codes are for channel 0 only
        switch ( func_addr )
        {
          case READ_TRIGGERED_CHANNEL_ADDR:// Read address 131
              rd_trig_chnl_addr();         // function in stimdot2.c
              break;

          case WRITE_TRIGGERED_CHANNEL_ADDR:// Write address 3
              wr_trig_chnl_addr();          // function in stimdot2.c
              break;

          case READ_FW_VERSION:           // Read address 135
              rd_fw_version();            // function in stimdot2.c
              break;

          case READ_TEDS:                 // Read address 160
              rd_meta_teds();             // function in stimteds.c
              break;

          case READ_ID_TEDS:              // Read address 161
              rd_meta_id_teds();          // function in stimteds.c
              break;

          case READ_FULL_TEDS_EEPROM:     // Read address 240
              rd_full_teds_eeprom();      // function in stimteds.c
              break;

          case WRITE_FULL_TEDS_EEPROM:    // Read address 32
              wr_full_teds_eeprom();      // function in stimteds.c
              break;

          default:
              // Allow user to provide additional functions
              DEFAULT_1451DOT2_FUNCTION   //
              break;
        }
      }
    }


    AFTER_1451DOT2_FRAME

    while ( !frame_aborted() )   // Wait until the NCAP raises NIOE
    {
      // If the NCAP is still trying to read data, just clock out 0.
      // We stay here until the NCAP is finished and raises NIOE

      wr_1451_byte(0); // provides ‘done’ handshake
    }
}
```

```
/*****************************************************************
*                                                                *
*   Filename:       stimcmd.h                                    *
*   Date:           06/27/2000                                   *
*                                                                *
*   Tools:          Hi-Tech PIC C Compiler V7.85                 *
*                   MPLAB 5.00.00                                *
*                                                                *
*   Notes:                                                       *
*                                                                *
*   Description:     Reference linkage for variables and functions.  *
*                                                                *
*****************************************************************/



/* The following declarations are reference linkage type */

extern uchar  func_addr;                // reference linkage
extern uchar  chnl_addr;                // reference linkage


/* Reference linkage to stimteds.c file */
//extern void  process_teds_prolog();      // init e.g. teds_format_vsn, teds_blks_per_chnl
extern void  rd_meta_teds( void );
extern void  rd_meta_id_teds( void );
extern void  rd_channel_teds( void );
extern void  rd_channel_id_teds( void );
extern void  rd_calibration_teds( void );
extern void  rd_calibration_id_teds( void );

// Extension Read(s)
extern void   rd_fw_version( void );
extern void   rd_full_teds_eeprom( void );

// TEDS Write function(s)
extern void   wr_calibration_teds( void );
extern void   wr_calibration_id_teds( void );
extern void   wr_full_teds_eeprom( void );



//extern void   transducer_initialize( void );
//extern void   transducer_trigger( void );
extern void   transducer_read( void );
extern void   transducer_write( void );


/* 1451 Byte I/O Functions */
extern uchar  wr_1451_byte( uchar abyte );
//extern uchar  rd_1451_byte( void );
//extern void   frame_complete( void );
//extern uchar  frame_aborted( void );     // tests NIOE and aborted_1451_io; sets latter


// 1451 Transducer, Global Functions
extern void   rd_status( void );
extern void   rd_aux_status( void );
extern void   rd_trig_chnl_addr( void );
extern void   wr_trig_chnl_addr( void );
extern void   rd_int_mask( void );
extern void   wr_int_mask( void );
extern void   rd_aux_int_mask( void );
extern void   wr_aux_int_mask( void );
extern void   wr_ctrl_cmd( void );
extern void   post_inval_cmd( void );
```



```
//extern void   update_nint( void );
//extern void   standard_trigger( void );
//extern void   standard_reset( void );
//extern void   init_mask_registers( void );
//extern void   init_int_registers( void );


#ifndef AFTER_1451DOT2_FRAME
#define AFTER_1451DOT2_FRAME
#endif
```

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-786-7200 Fax: 480-786-7277
Technical Support: 480-786-7627
Web Address: http://www.microchip.com

Atlanta
Microchip Technology Inc.
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770-640-0034 Fax: 770-640-0307

Boston
Microchip Technology Inc.
2 LAN Drive, Suite 120
Westford, MA 01886
Tel: 508-480-9990 Fax: 508-480-8575

Chicago
Microchip Technology Inc.
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas
Microchip Technology Inc.
4570 Westgrove Drive, Suite 160
Addison, TX 75001
Tel: 972-818-7423 Fax: 972-818-2924

Dayton
Microchip Technology Inc.
Two Prestige Place, Suite 150
Miamisburg, OH 45342
Tel: 937-291-1654 Fax: 937-291-9175

Detroit
Microchip Technology Inc.
Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250 Fax: 248-538-2260

Los Angeles
Microchip Technology Inc.
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888 Fax: 949-263-1338

New York
Microchip Technology Inc.
150 Motor Parkway, Suite 202
Hauppauge, NY 11788
Tel: 631-273-5305 Fax: 631-273-5335

San Jose
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7955

AMERICAS (continued)

Toronto
Microchip Technology Inc.
5925 Airport Road, Suite 200
Mississauga, Ontario L4V 1W1, Canada
Tel: 905-405-6279 Fax: 905-405-6253

ASIA/PACIFIC

China - Beijing
Microchip Technology, Beijing
Unit 915, 6 Chaoyangmen Bei Dajie
Dong Erhuan Road, Dongcheng District
New China Hong Kong Manhattan Building
Beijing, 100027, P.R.C.
Tel: 86-10-85282100 Fax: 86-10-85282104

China - Shanghai
Microchip Technology
Unit B701, Far East International Plaza,
No. 317, Xianxia Road
Shanghai, 200051, P.R.C.
Tel: 86-21-6275-5700 Fax: 86-21-6275-5060

Hong Kong
Microchip Asia Pacific
Unit 2101, Tower 2
Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2-401-1200 Fax: 852-2-401-3431

India
Microchip Technology Inc.
India Liaison Office
No. 6, Legacy, Convent Road
Bangalore, 560 025, India
Tel: 91-80-229-0061 Fax: 91-80-229-0062

Japan
Microchip Technology Intl. Inc.
Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa, 222-0033, Japan
Tel: 81-45-471- 6166 Fax: 81-45-471-6122

Korea
Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea
Tel: 82-2-554-7200 Fax: 82-2-558-5934

ASIA/PACIFIC (continued)

Singapore
Microchip Technology Singapore Pte Ltd.
200 Middle Road
#07-02 Prime Centre
Singapore, 188980
Tel: 65-334-8870 Fax: 65-334-8850

Taiwan
Microchip Technology Taiwan
10F-1C 207
Tung Hua North Road
Taipei, Taiwan
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

Denmark
Microchip Technology Denmark ApS
Regus Business Centre
Lautrup hoj 1-3
Ballerup DK-2750 Denmark
Tel: 45 4420 9895 Fax: 45 4420 9910

France
Arizona Microchip Technology SARL
Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - ler Etage
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

Germany
Arizona Microchip Technology GmbH
Gustav-Heinemann-Ring 125
D-81739 München, Germany
Tel: 49-89-627-144 0 Fax: 49-89-627-144-44

Italy
Arizona Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-039-65791-1 Fax: 39-039-6899883

United Kingdom
Arizona Microchip Technology Ltd.
505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44 118 921 5858 Fax: 44-118 921-5835

05/16/00

DNV Certification, Inc. USA — ANSI-RAB QMS ACCREDITED — DNV — DNV MSC The Netherlands Accredited by the RvA — ISO 9001 / QS-9000 REGISTERED FIRM

Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999. The Company's quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs and microperipheral products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001 certified.