Diamond Technologies Inc.

DT7000
Communication I/O Gateway
Programming Guide
Version 1.3

6 Clock Tower Place
Suite 100
Maynard, MA 01754
USA

Tel: (866) 837-1931
Tel: (978) 461-1140
FAX: (978) 461-1146

http://www.diamondt.com/

Liability

Diamond Technologies Inc. shall not be liable for technical or editorial errors or omissions contained herein, nor for incidental or consequential damages resulting from the use of this material. Those responsible for the use of this device must ensure that all necessary steps have been taken to verify that the applications meet all performance and safety requirements including any applicable laws, regulations, codes, and standards.

There are many applications of this product. The examples and illustrations in this document are included solely for illustrative purposes. Because of the many variables and requirements associated with any particular implementation, Diamond Technologies Inc. cannot assume responsibility for actual use based on these examples and illustrations.

Diamond Technologies Inc., reserves the right to modify our products in line with our policy of continuous product development. The information in this document is subject to change without notice and should not be considered as a commitment by Diamond Technologies Inc.

Intellectual Property Rights

Diamond Technologies Inc. has intellectual property rights relating to technology embodied in the product described in this document. These intellectual property rights may include patents and pending patent applications in the US and other countries.

Diamond Technologies Inc. and the Diamond Technologies logo are trademarks of Diamond Technologies Inc. All other trademarks are the property of their respective holders.

I Revision History

Version	Date	Description
0.1	09/14/10	Preliminary Version
0.2	08/17/11	Added Detail Content
1.0	08/19/11	Added Diagnostics Interface sections
1.1	11/15/11	Added detail to samples
1.2	01/23/12	Updated for updates in DT7000 main firmware
1.3	07/01/13	Updated Cover pages

ii Reference Documents

Table of Contents

1.0 Introduction

This document describes application programming for Diamond Technologies **DT7000 Communications I/O Gateway**. The DT7000 module provides a gateway between industrial networks, and physical I/O devices, including serial digital and analog I/O. Using this programmability an application program can be installed in the gateway to perform intermediate processing of serial data, I/O data and fieldbus data. This allows the gateway to act as an embedded logic controller either standalone or in conjunction with the fieldbus network.

2.0 DT7000 Programming

The DT7000 module defines data in an input array and an output array. Physical inputs such as digital and analog inputs, and received serial data are mapped into the input array. Physical outputs such as digital and analog outputs and serial data to be transmitted are mapped from the output array. The standard DT7000 firmware connects the input and output arrays to the installed fieldbus allowing the fieldbus network to read the physical inputs and write the physical outputs.

When an application program is installed the DT7000 firmware separates the fieldbus from the input and output arrays, and installs the application in between. The application is able to read the physical inputs connected to the gateway and read the fieldbus output data sent by the master over the network. The application is also able to write values to the physical outputs and write values to the fieldbus input data to be read by the master over the network. The application can further perform logic or process the data as required. Figure 1 shows the architecture of the DT7000 firmware with a user application installed.

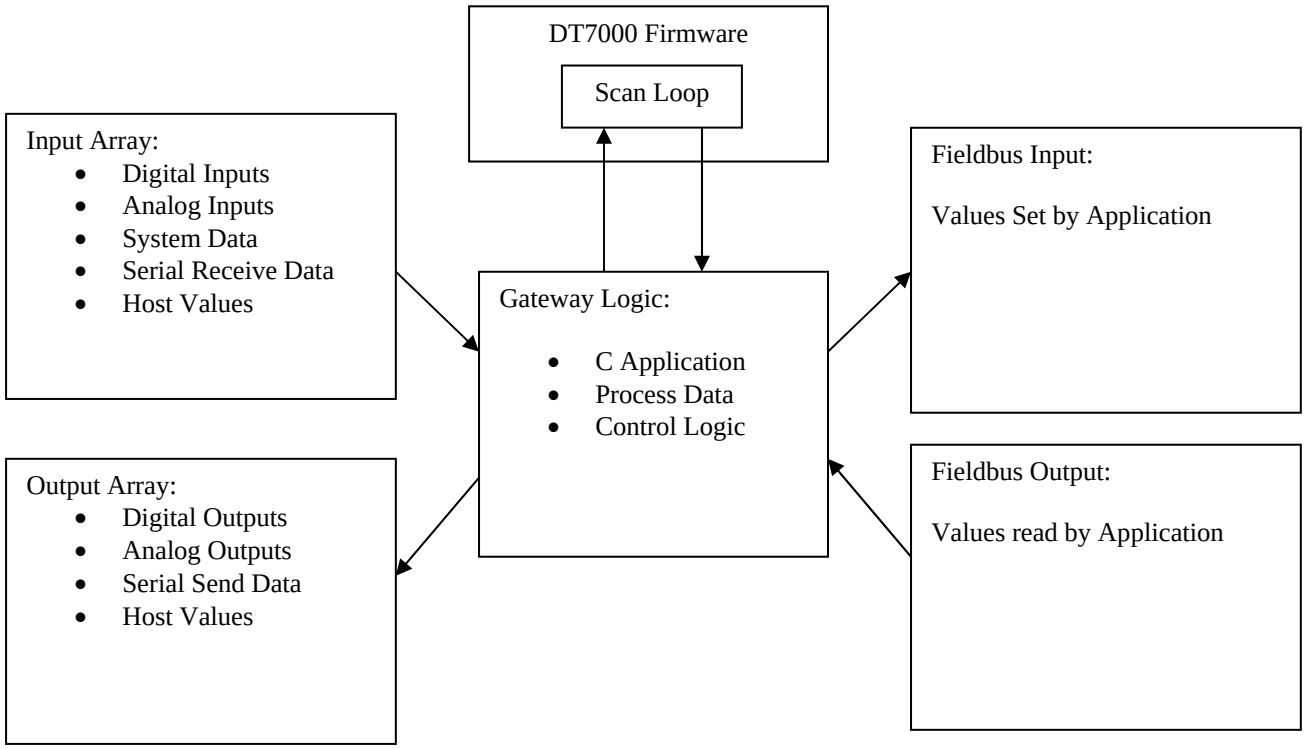


Figure 1 User Application Block diagram

2.1 User Program Architecture

The user application program for the DT7000 is created in C using standard development tools. The user application program is created to run as an integral part of the standard DT7000 firmware. All the standard DT7000 functionality, including interfacing to I/O, serial channels and the industrial network, along with other functionality such as timing, configuration management, and resource allocation is handled by the DT7000 firmware. The user program is thus simplified to the logic and data manipulation required by the application. The user program consists of the following components.

Code:

GatewayLogic – Routine. The user program consists of a single routine which is called each time through the scan loop. This routine is installed at a pre-defined location in the DT7000 program memory.

Data:

DT_InData – A data structure is defined which matches the physical input devices configured in the DT7000. The user code can then access these input devices through this structure.

DT_OutData – A data structure is defined which matches the physical output devices configured in the DT7000. The user code can write to these output devices through this structure.

FB_InData – A data structure is defined which matches the desired fieldbus input data to be read by the master. The user code can then write the data to the fieldbus master through this structure.

FB_OutData – A data structure is defined which matches the desired fieldbus output data to be written by the master. The user code can then access the data set by the fieldbus master through this structure.

DTSysDat – A data structure is defined by the DT7000 which includes system information and control values. This structure provides an interface between the user program and the DT7000 for these system functions.

Since the user application consists of a single routine called as part of the scan loop, the code must be developed accordingly. The **GatewayLogic** routine must execute and return in an efficient manner. Execution time of this routine will directly add to the scan time. The routine should never wait for events in line. A state machine can be created to wait for events. Timing can be accomplished within the state machine using the 0.1 msec timer available in the **DT_SysDat** structure, or the msec timer or sec timer which can be configured into the **DT_InData** structure. Examples of using state machines to wait for events with timing are in the sample applications.

A header file **DT7000Gateway.h** is included in the user program. This provides location information for the code and data structures listed above. The code and data structures must be located at the proper locations to integrate with the DT7000 firmware. Refer to the sample applications to locate the code and data structures.

The header file also provides a definition of the **DTSysDat** structure. Refer to this header file for a description of the components available through this structure. The sample applications demonstrate the use of some of these components.

The diagnostic port can be used to display information to debug the application. It can also be used as a user interface if required by the user application. The sample applications demonstrate the use of the diagnostic interface.

The user application can store configuration information to non-volatile flash memory if required. The sample applications have examples of storing configuration information to flash.

2.2 Development Tools

The following development tools are recommended for creating the user program.

MPLAB – IDE	Microchip	http://www.microchip.com/
PCD - C Compiler, Command Line	CCS	http://www.ccsinfo.com/
PICKit 3 – Programmer/Debugger (Optional)	Microchip	http://www.microchip.com/

Note the PICKit 3 Programmer/Debugger is optional. Code created in MPLAB and compiled with the PCD compiler can be loaded into the DT7000 directly through the diagnostic port without the need for a programmer. The PICKit 3 is used for source level debugging from within MPLAB. Most user programs can be simply debugged with diagnostic outputs sent to the diagnostic port during execution eliminating the need for a debugger. The PICKit3 can also be use to load the DT7000 main firmware. It is possible to write application code which corrupts the main firmware, or configuration. In this case the main firmware must be reloaded with a programmer to recover the module.

2.2 Program Download

Once created the user program can be downloaded to the DT7000 through the Diagnostic Interface. A diagnostic cable (CAB-DT7-DIAG) is provided as part of the DT7000 starter kit or can be purchased separately from Diamond Technologies. The cable allows for a connection to the DT7000 diagnostic interface using a terminal program such as Windows HyperTerminal or the open source software Tera Term (www.logmett.com).

The diagnostic interface by default is accessed through Port D of the DT7000. The default communications parameters in the DT7000 are as follows:

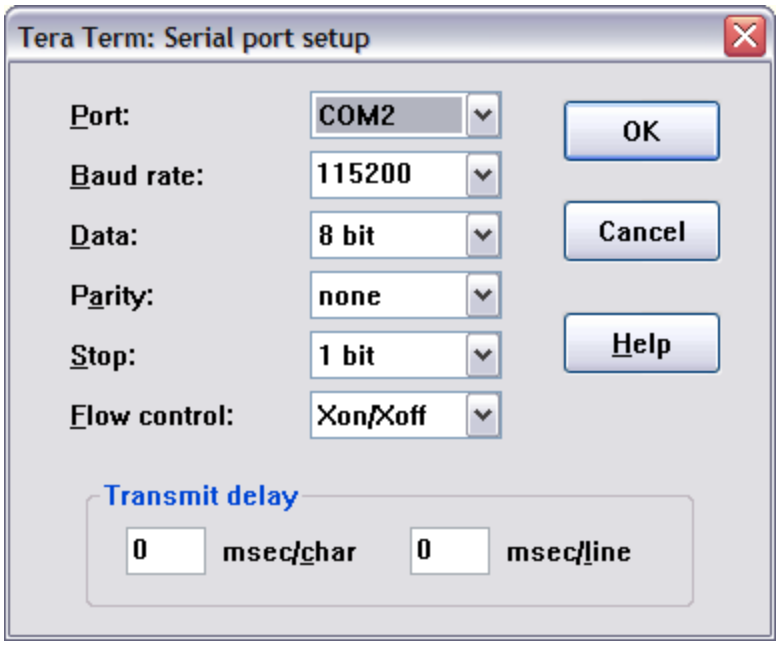
Default Communication Parameters (Port D)	
Baud	115200
Data Bits	8
Parity	None
Stop Bits	1

The diagnostic interface allows for various configuration and diagnostic functions including downloading user application code. For more information regarding the diagnostic interface refer to the **DT7000 Users Guide**

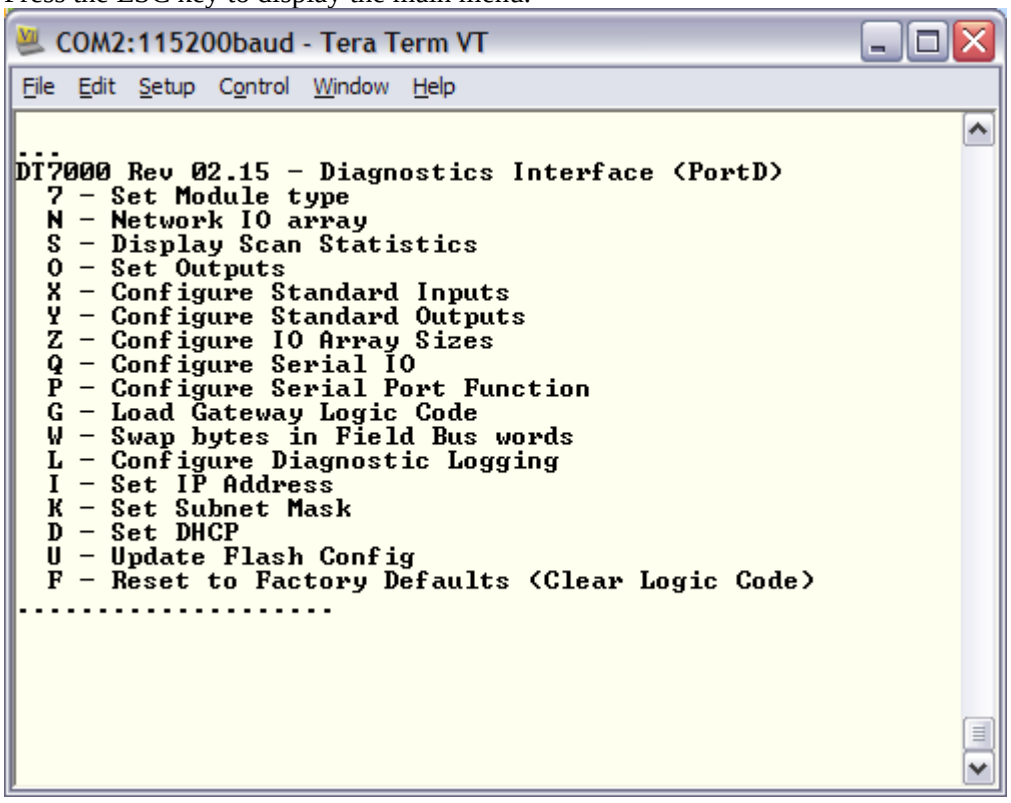
To download the user application program through the diagnostic interface using Tera Term perform the following steps.

1. Connect the diagnostic cable to port D of the DT7000 and to an available port on your PC.

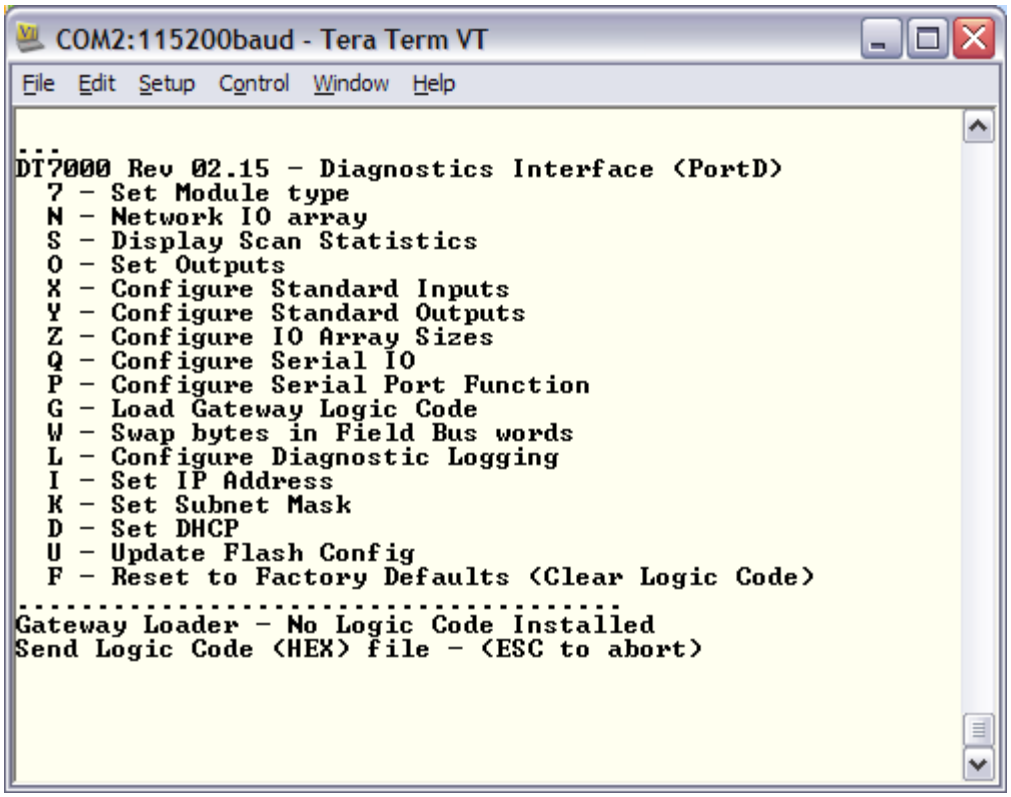
2. Start Tera Term and set the appropriate communications settings.



3. Press the ESC key to display the main menu.

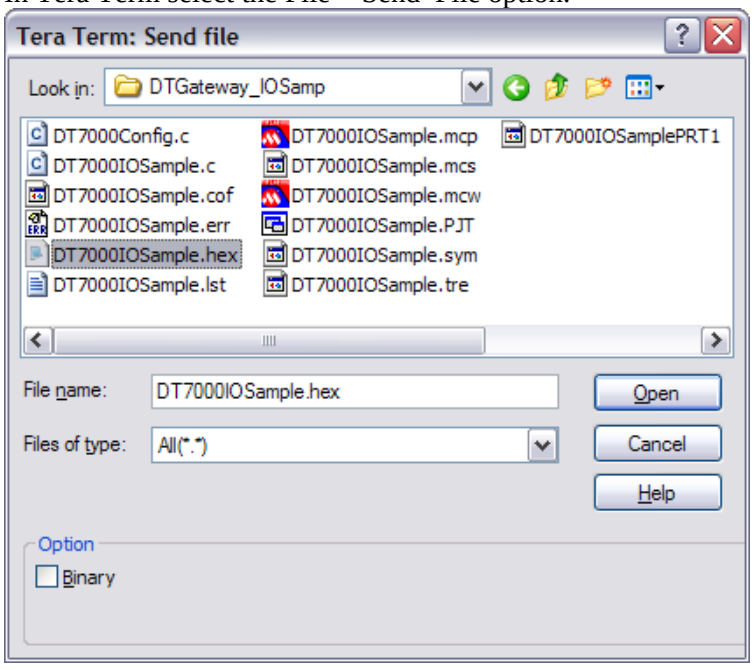


4. Press “G” to select “Load Gateway Logic Code”

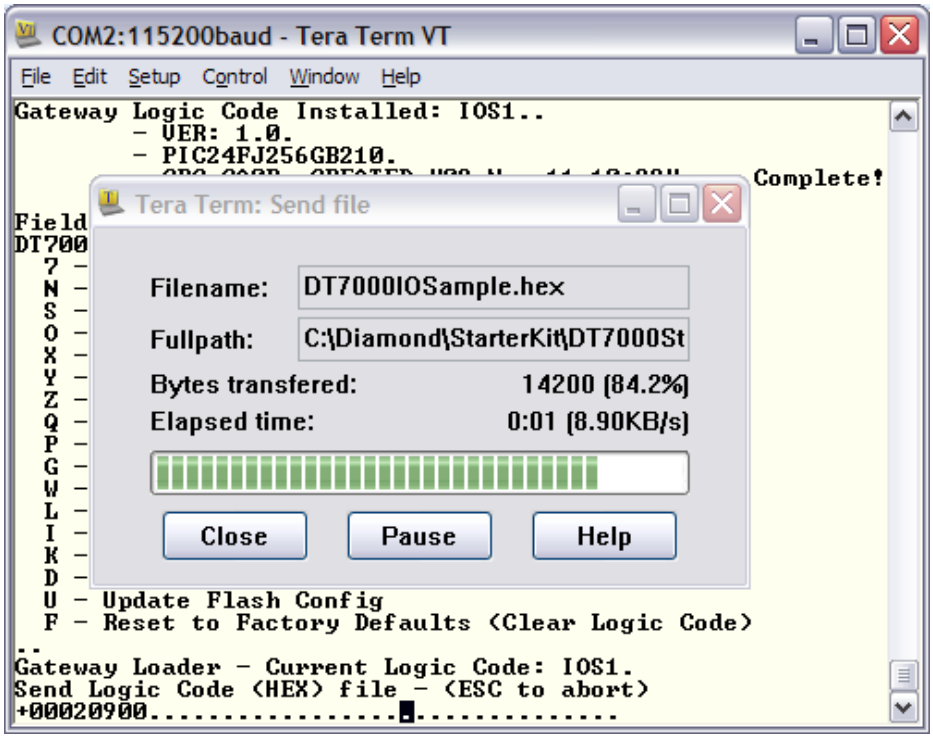


If a user application is currently installed the version will be displayed. ESC can be pressed to abort the load.

5. In Tera Term select the File > Send File option.

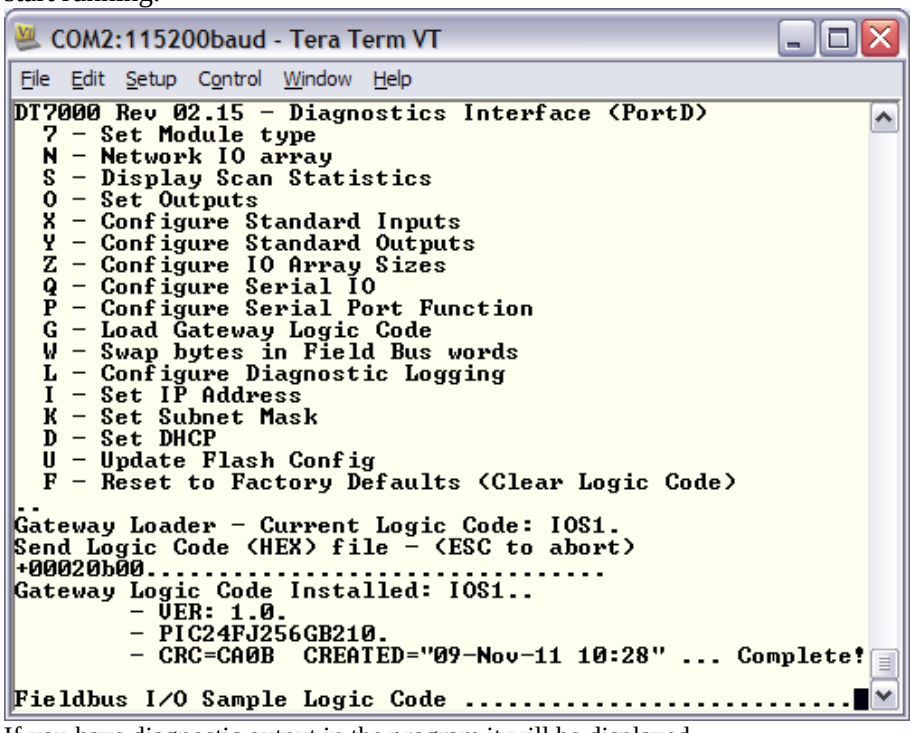


6. Browse to the .hex file in your project folder and click “Open”.



The DT7000 will display the address of application code as it is being downloaded to the DT7000.

7. Once completed the diagnostic interface will indicated a successful download, and the application will start running.



If you have diagnostic output in the program it will be displayed.

3.0 Sample Applications

Two sample user programs are provided. These programs demonstrate most of the functionality available to a DT7000 user application, and can be used as a starting point for program development. These programs are distributed as MPLAB projects. The source code for the program can be viewed, modified, and compiled using MPLAB and the CCS compiler, and loaded into the DT7000 through the diagnostic port.

3.1 I/O Sample

The first sample program demonstrates the use of the digital and analog I/O along with the fieldbus and some simple processing. In this sample an analog input is read. The 10 bit value is converted to a floating point value indicating actual voltage (0-10 Volts), and this floating point value is made available on the fieldbus. In addition a limit value can be set via the fieldbus. If the read voltage is greater then the limit value, output 0 is set, other wise output 0 is cleared. In addition the digital and analog I/O is made accessible from the fieldbus.

This application first demonstrates putting DT7000 I/O data on the fieldbus.

A file **DTConfig.C** is included which defines the DT7000 configuration for this application. In this file, the **DT_InData**, and **DT_OutData** structures are defined to represent the physical devices utilized on the DT7000 in this application. The user application can read the various inputs into the DT7000 through the **DT_InData** values. For instance **DT_InData.AnalogIn[2]** will contain the current value of analog input 2. The user application can write the physical outputs by assigning values to the **DT_OutData** values. For instance the assignment **DT_OutData.DigOuts = 0x05** will turn on digital outputs 0 and 3.

```
struct
{
        BYTE    DigIns;         // Low byte of word 0 is digital INS
        BYTE    Reserved;       // High byte of word 0 is not used
        WORD    AnalogIn[8];    // All 8 analog inputs enabled. First is used.
        WORD    SecTimer;       // Second Timer
        BYTE    FirmMin;        // Minor firmware version
        BYTE    FirmMaj;        // Major firmware version
} DT_InData;

struct
{
        BYTE    DigOuts;        // Low byte of word 0 is digital OUTS
        BYTE    Reserved;       // High byte of word 0 is not used
        WORD    AnalogOut[4];   // Four words for analog outputs
} DT_OutData;
```

Next the **FB_InData** and **FB_OutData** structures are defined to represent the data we want on the fieldbus. These structures are defined in the order and with the data types desired for the fieldbus representation of the data. The user application will provide data to the host through the **FB_InData** values and will get data from the host through the **FB_OutData** values.

```
struct
{
        BYTE    DigIns;                 // Low byte of word 0 is digital INS
        BYTE    FBDigOuts;              // High byte of word 0 - Install copy of digital outs for FB to read back
        WORD    SecTimer;               // Second Timer
        BYTE    FirmMin;        // Minor firmware version
        BYTE    FirmMaj;        // Major firmware version
        WORD    AnalogIn;
```

```
        float   Voltage;        // Floating point voltage read on analog in 1
} FB_InData;

struct
{
        BYTE    DigOuts;        // Low byte of word 0 is digital OUTs (bit 0 set by logic, not set by fieldbus)
        BYTE    Reserved;               // High byte of word 0 is not used
        WORD    AnalogOut;
        float   VoltageLimit;
} FB_OutData;
```

The structures must be located to specific memory locations. This allows both the DT7000 main firmware and the user application to access these structures.

```
#locate DT_InData = DTPDINSTART
#locate DT_OutData = DTPDOUTSTART
#locate FB_InData = FBPDINSTART
#locate FB_OutData = FBPDOUTSTART
```

The structures also must match the set configuration of the DT7000 in terms of the enabled I/O devices, and sizes. The configuration is set with the **GatewayConfig** routine. This routine performs the following steps.

1. Checks the CONFIG bit ((FlConfig.Debug_Log & 0x80) == 0) to determine if the board needs to be configured for this application.
2. Sets the CONFIG bit, so the board is now marked as configured for next power up.
3. Sets the data size of the arrays to match the above defined structures.
4. Enables the desired physical input and output devices by setting StdInEnbMask, and StdOutEnbMask.
5. Sets the Port A and Port C functions to be available (by default they are MODBUS and Generic Serial).
6. Sets the DTSysDat.FlUpdate value which causes this configuration to be saved to flash,
7. Prints a message on the diagnostic port indicating the board has been configured.

Based on the requirements of the application additional configuration can be done in this routine.

The main **GatewayLogic** routine first checks the **DTSysDat.FirstPass** variable. This variable will be TRUE the first time the **GatewayLogic** routine is called. When this is TRUE the **GatewayConfig** routine is called and other global variables are initialized.

Note that variables must be explicitly initialized in the code when the **FirstPass** variable is TRUE as opposed to assigning an initial value as part of the declaration. A declaration with an initial value

```
BYTE SetOut0 = 0;       //This will not work!!!
```

will cause the compiler to create initialization code prior to the call to main. When creating user application code this initialization code will not be loaded or executed, since only the **GatewayLogic** routine is loaded.

The **GatewayLogic** routine next copies some of the physical inputs to the fieldbus and likewise copies fieldbus values to the physical outputs. Since the GatewayLogic routine is executed on every scan loop, the assignment

```
FB_InData.AnalogIn = DT_InData.AnalogIn[0];
```

has the effect of providing a continuously updating analog value on the fieldbus inputs. Looking at the input data with a fieldbus master, this value will always show the current analog value. Likewise the assignment

```
DT_OutData.DigOuts = (FB_OutData.DigOuts&0xFE) + SetOut0;
```

has the effect of tying the physical outputs to a fieldbus output value on the master. From the master, any time it changes this value the digital outputs will change accordingly. The exception is bit 0, which in our application

will be set when analog input 0 exceeds a limit value. The logic in the assignment excludes bit 0 in the fieldbus output and adds the **SetOut0** variable in its place. The **SetOut0** variable will be set when the limit is exceeded.

```
if (FB_InData.Voltage >= FB_OutData.VoltageLimit) SetOut0 = 1;
else SetOut0 = 0;
```

The GatewayLogic routine demonstrates the creation of a processed value to be put on the fieldbus.

```
RdAnalogIn = DT_InData.AnalogIn[0];          // Freeze a value of analog in to use and display below
FB_InData.Voltage = (RdAnalogIn * 10.0)/1024; // 10 Volt full scale
```

Since **FB_InData.Voltage** is defined as a float, and since the floating point constant 10.0 is used, this assignment converts the 10 bit analog input to a 32 bit floating point value corresponding to the actual voltage from 0 to 10 volts.

Finally the sample demonstrates using the diagnostic port to display debug and diagnostic data. At the bottom of the GatewayLogic routine, it looks for a **'9'** character to be pressed on the diagnostic port. If this is seen it displays the last measured voltage as a raw 10 bit value, as the 32 bit hex representation of the floating point value and as the converted floating point value corresponding to voltage. It also displays the voltage limit value as the 32 bit hex representation, and as the floating point value. From the master, depending on how you are able to represent the value, you may only see the 32 bit hex representation of the value, or you may be able to see the actual floating point value.

Note there is a limitation in user applications which does not allow displaying floating point values with the %f format code. Therefore the floating point values must be converted to integers before displaying as shown.

3.2 Barcode Sample

The second sample program demonstrates the use of the serial I/O along with some digital I/O in a simple control application. In this sample a barcode reader is connected to port B. The barcode reader reads UPC Barcodes and sends the codes in a specific format over the serial channel. The sample program receives the serial string and extracts the UPC code. It compares the code to 3 possible match codes, and sets the appropriate output (0, 1, or 2) if the code matches one of the stored match codes. It further keeps a count of the total codes read, and the number matching each match code. The counts, and the last read code are made available on the fieldbus. Additionally the match codes can be taught via a command over the fieldbus, or via a command through the diagnostic port. Additional diagnostics are available through the diagnostic port.

The application is meant to be used with a serial barcode reader. However if a barcode reader is not available, a simple PC application is provided which can simulate the barcode reader. This application (BarcodeSimulator.exe) allows you to select from a number of UPC codes, and will send these codes out the PC serial port to the DT7000 in the same format as a barcode reader.

This application also includes a DTConfig.C file to configure the DT7000. In this application Port B is configured to use the generic serial driver which allows the user application to access it through the DT_InData and DT_OutData structures. The DT_InData and DT_OutData structures are defined to include the Port B data.

```
struct
{
        WORD    SecTimer;               // Second Timer
        BYTE    FirmMin;                // Minor firmware version
        BYTE    FirmMaj;                // Major firmware version
        // Port B Generic Serial
```

```
        serStat   PortBStat;              // Status byte
        BYTE      PortBRxLen;             // Receive length
        BYTE      PortBRxBuff[20];        // Rcv Data buffer
} DT_InData;

struct
{
        BYTE      DigOuts;                // Low byte of word 0 is digital OUTS
        BYTE      Reserved;               // High byte of word 0 is not used
        // Port B Generic Serial
        serCont   PortBCont;              // Control byte
        BYTE      PortBTxLen;             // Transmit length
        BYTE      PortBTxBuff[20];        // Data buffer
} DT_OutData;
```

The routine looks for data to be received on the serial port. Since the serial data is buffered by the DT7000 it waits until an entire bar code string (STX, 12 character UPC code, CR, LF), is in the buffer before processing.

```
if (DT_InData.PortBStat.RxDat != DT_OutData.PortBCont.RxAck)     // Serial Data available
{
        if ((DT_InData.PortBRxLen) >= 15)                 //// Data is STX UPC CR,LF. If not whole string, ignore
        {
```

Once the serial data is processed, the RxAck bit is set to be the same as the RxDat bit. This indicates to the serial driver that this data has been processed and more data can be put in the serial buffer.

```
DT_OutData.PortBCont.RxAck = DT_InData.PortBStat.RxDat;          // Toggle Toggle bit indicating data read
```

In run mode the routine simply compares the data in the Port B serial buffer to the stored match codes. If it matches one of the codes, the corresponding output is set and the counter incremented. Regardless the total count is incremented, and the code is put into the fieldbus data. The code is also displayed on the diagnostic port.

The application demonstrates storing configuration data in flash. The three match codes can be taught and stored into flash, so they will be remembered after a power cycle. To do this a structure containing the data to be stored in flash is defined, and located to the beginning of the application data area.

```
// The following structure will be written to flash to store config data.
struct
{
        BYTE      StrUPCCode1[12];  // UPC Code to verify for out 1
        BYTE      StrUPCCode2[12];  // UPC Code to verify for out 2
        BYTE      StrUPCCode3[12];  // UPC Code to verify for out 3
}FL_CfgDat;

// This must be located at the start of user ram.
#locate FL_CfgDat = APRAMSTART
```

When new match codes are taught the values in this structure are updated with the new codes. Then to store the structure to flash the DTSysDat.WrFlCfgSze variable is updated with the size of the data to be written to flash.

```
DTSysDat.WrFlCfgSze = sizeof(FL_CfgDat);
```

This triggers the DT7000 main firmware to store this structure to flash. On startup (on FirstPass) the application initializes the structure with the values stored in flash. Setting the DTSysDat,RdFlCfgSze variable with the size of the data to read, triggers the DT7000 main firmware to update the structure with the stored values from flash.

```
DTSysDat.RdFlCfgSze = sizeof(FL_CfgDat);          // Read match codes from flash
```

The user code configuration area is limited to 756 bytes. The size of the Fl_CfgDat structure must be kept less the 756 bytes.

Flash memory has a limited endurance. The flash memory in the DT7000 is guaranteed for a minimum of 10000 write cycles. Care should be taken to write the flash only as needed to update configuration data. Writing the flash continuously during normal operation will exceed the life expectancy very quickly and render the module unusable.

Appendix A: DT7000 Memory Map

The DT7000 memory map shows the entire memory utilization of the DT7000 and is provided for reference. The DT7000Gateway.h file defines all the locations and sizes for user applications. Using the samples as a template will create user applications compatible with the DT7000 firmware.

Program Memory		
00000 . 001FF	Interrupt Vectors	
00200 . 1E3FF	Main DT7000 Code	61696 Instructions
1E400 . 1E7FF	DT7000 Configuration	768 Bytes
1E800 . 1FFFF	BootLoader	3072 Instructions
20000 . 2A3FF	User Application Code	20992 Instructions
2A400 . 2A7FF	User Configuration + User ID	768 Bytes
2A800 . 2ABFF	Reserved +PIC Configuration Words	

Data Memory		
0000 . 07FF	Special Function Registers	
0800 . 08FF	Reserved for ICD	
0900 .	Main DT7000 Data	

2CFF		
2D00 . 2FFF	Configuration Structure	
3000 . 3AFF	Application data Structure	
3B00 . 3Bff	DT_SysDat Structure	
3C00 . 3CFF	FB_InData	
3D00 . 3DFF	FB_OutData	
3E00 . 3EFF	DT_InData	
3F00 . 3FFF	DT_OutData	
4000 . 5FFF	User Code Data	
6000 . 7FFF	Reserved	
8000 . FFFF	EDS Window Reserved	