# STX104

Integrated Analog PC/104 Card with One Mega-Sample FIFO

Revision: January 1, 2005
Applies to Product Revision-G or higher

Apex Embedded Systems

# Reference Manual

# STX104 Reference Manual

Apex Embedded Systems
116 Owen Road
Monona, WI 53716
Phone 608.256.0767 • Fax 608.256.0765

# Table of Contents

# Support Policy

## General Support Policy.

We support all hardware products for a period of 3 months from time of delivery. See limited warranty terms.

## Recommended sequence in obtaining customer support

Review user manuals for additional information not found in demo software.

Go to our website at www.apexembedded.com/support.html.

Contact us via email at help@apexembedded.com. Technical support related inquiry answered typically within a 24-hour period.

## Need custom modifications?

Contact us for customization or modifications to our standard product. If you need large quantities we can generally save you money through optimizing card designs to meet your exact needs.

# Welcome

Dear Valued Customer:

Thank you for your interest in our products and services.

Apex Embedded Systems "Continuous improvement" policy utilizes customer feedback to improve existing products and create new product offerings based on needs of our customers.

Continued Success,

Apex Embedded Systems

**Caution**

A discharge of static electricity from your hands can seriously damage certain electrical components on any circuit board. Before handling any board, discharge static electricity from yourself by touching a grounded conductor such as your computer chassis (your computer must be turned off). Whenever you handle a board, hold it by the edges and avoid touching any board components or cable connectors.

Chapter
1

# Benefits and Features

*The STX104 provides high functionality while making life a little easier*

## Executive Summary

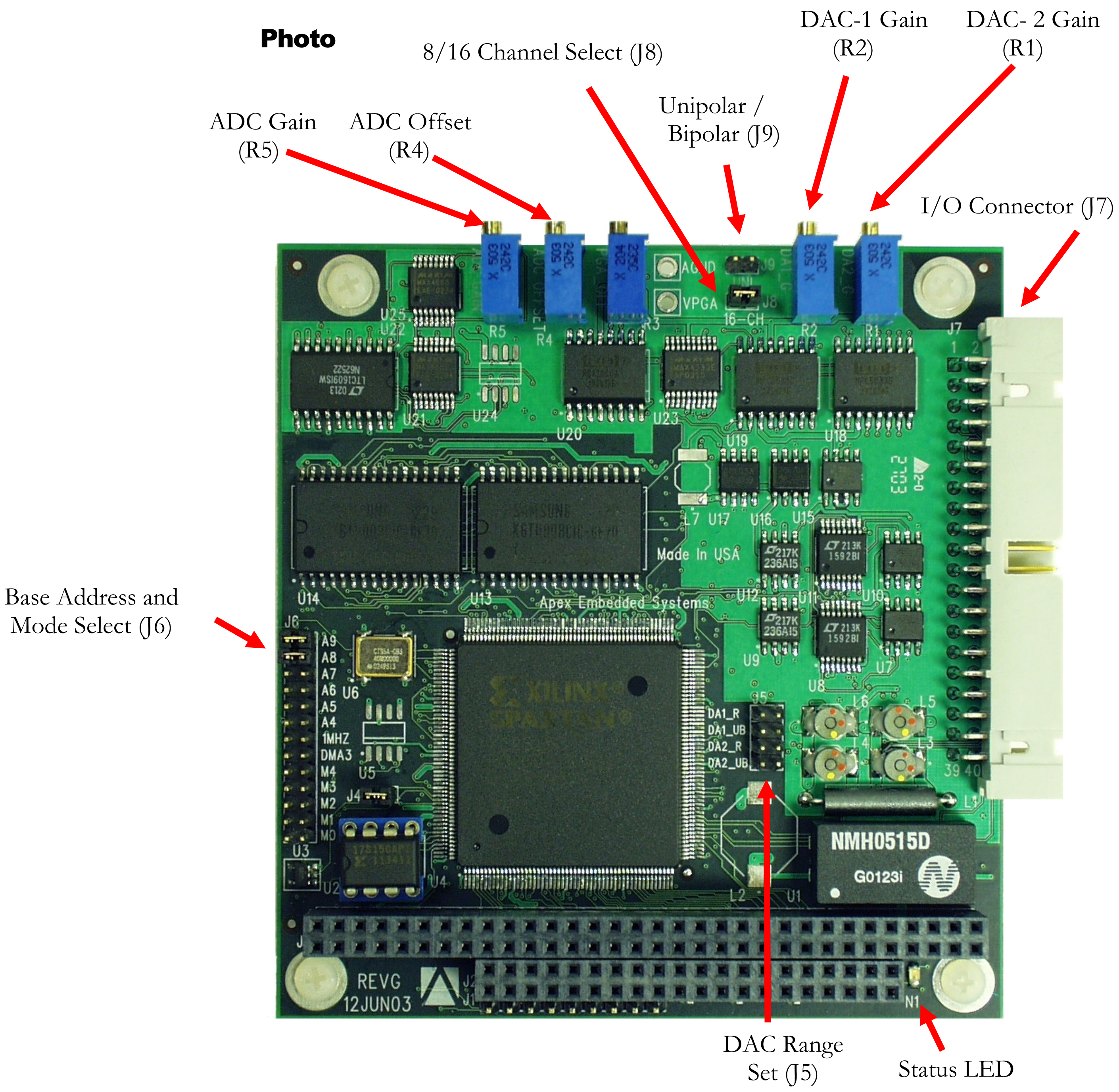
**What is the STX104?** The STX104 is a 16-channel 16-bit analog input and 2-channel 16-bit analog output card. The STX104 incorporates a large one mega-sample FIFO.

**What are the Benefits to using the STX104?** The STX104 has the following benefits:

- One mega-sample FIFO data storage provides continuous data streaming from ADC to CPU with reduction in interrupt overhead and relaxation of interrupt latencies.
- 200,000 Samples per second.
- 16-Sample moving average filters for data noise reduction
- Compatible with DAS16jr/16 and DAS1602 for using existing 3rd party drivers
- 16-bit data read (ADC data) operations double effective PC/104 bus bandwidth
- 16-bit data write (DAC data) operations reduce software overhead
- Use REP INSW (286 or higher CPU) to read-in blocks of ADC data from FIFO further increasing bandwidth and reducing complexity.
- Burst mode with only one interrupt generated per complete scan, thus reducing interrupt overhead and increasing effective throughput.
- One interrupt per 512-samples is possible in FIFO mode
- On-board LED to indicate that the STX104 is being addressed. By observing the LED you can quickly determine system activity.
- Polarized locking I/O connector. This eliminates board failures due to incorrect connector orientation.
- Extremely low DC drift

- External trigger input (DIN0) is deglitched preventing unwanted ADC triggers. Minimum valid pulse width required is 200nS.
- Industrial temperature range from -40°C to +85°C
- No tantalum capacitors or electrolytic capacitors used in the design
- Single +5V supply operation

## Photo



## ADC Data Gathering and Conversion Summary

**Single Conversion**

Compatibility Modes (M0 jumper):

| | |
|---|---|
| DAS16jr/16: | Polled, Interrupt or DMA |
| DAS1602: | Polled, Interrupt or DMA |

With FIFO Superset Enabled (M1 jumper):

| | |
|---|---|
| DAS16jr/16: | Polled, Interrupt or DMA |
| DAS1602: | Polled, Interrupt or DMA |

General Algorithm:

1. Wait for trigger
2. ADC Sample and convert
3. Increment multiplexer to next channel
4. Done

**Programmable Burst**

Compatibility Modes (M0 jumper):

| | |
|---|---|
| DAS16jr/16: | none |
| DAS1602: | Polled, Interrupt or DMA. FIFO status registers not available. Interrupt generated for every sample taken. |

With FIFO Superset Enabled (M1 jumper):

| | |
|---|---|
| DAS16jr/16: | none |
| DAS1602: | Polled, Interrupt or DMA. FIFO status registers Available for reading. Interrupt generated for each burst (scan) completion. |

General Algorithm:

1. Wait for trigger
2. Sample n-channels (set by BL[3:0]) at maximum sampling rate (10 uS)
3. Done

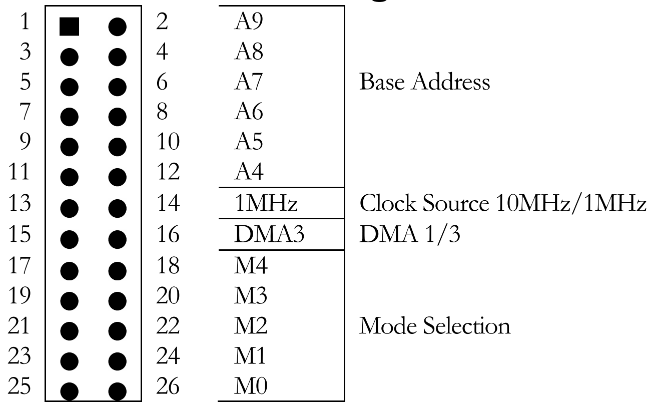
**Trigger Sources**

| TS1 | TS0 | Trigger Function |
|---|---|---|
| 0 | X | Software triggered ADC sampling only |
| 1 | 0 | Trigger on rising edge of DIN0 |
| 1 | 1 | Trigger on pacer clock output (CT_OUT2) |

# Chapter 2

# Hardware Configuration

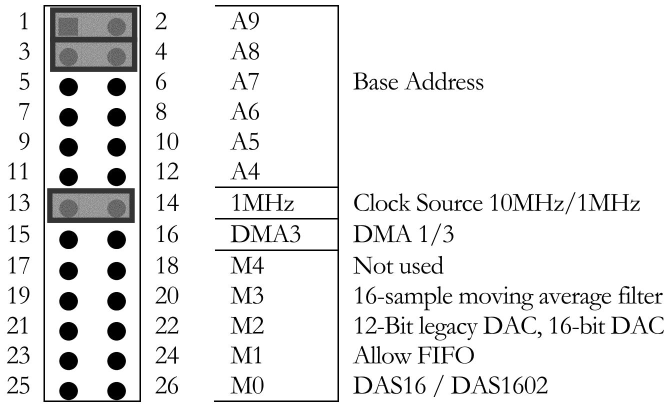
## Base Address and Configuration



| | | | | |
|---|---|---|---|---|
| 1 | ■ ● | 2 | A9 | Base Address |
| 3 | ● ● | 4 | A8 | |
| 5 | ● ● | 6 | A7 | |
| 7 | ● ● | 8 | A6 | |
| 9 | ● ● | 10 | A5 | |
| 11 | ● ● | 12 | A4 | |
| 13 | ● ● | 14 | 1MHz | Clock Source 10MHz/1MHz |
| 15 | ● ● | 16 | DMA3 | DMA 1/3 |
| 17 | ● ● | 18 | M4 | Mode Selection |
| 19 | ● ● | 20 | M3 | |
| 21 | ● ● | 22 | M2 | |
| 23 | ● ● | 24 | M1 | |
| 25 | ● ● | 26 | M0 | |

## Example Base Address

Base Address = 768 or 0x300 ( 11 0000 $0000_2$ )
Clock Source = 1 MHz
DMA = 1
Mode = DAS16jr/16 compatibility, no 16-bit transfers, no FIFO, 12-Bit DAC data.



| | | | | |
|---|---|---|---|---|
| 1 | ■ ● | 2 | A9 | Base Address |
| 3 | ● ● | 4 | A8 | |
| 5 | ● ● | 6 | A7 | |
| 7 | ● ● | 8 | A6 | |
| 9 | ● ● | 10 | A5 | |
| 11 | ● ● | 12 | A4 | |
| 13 | ● ● | 14 | 1MHz | Clock Source 10MHz/1MHz |
| 15 | ● ● | 16 | DMA3 | DMA 1/3 |
| 17 | ● ● | 18 | M4 | Not used |
| 19 | ● ● | 20 | M3 | 16-sample moving average filter |
| 21 | ● ● | 22 | M2 | 12-Bit legacy DAC, 16-bit DAC |
| 23 | ● ● | 24 | M1 | Allow FIFO |
| 25 | ● ● | 26 | M0 | DAS16 / DAS1602 |

## Base Address Table

| Base Address | A9 | A8 | A7 | A6 | A5 | A4 |
|---|---|---|---|---|---|---|
| 0x220 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0x240 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0x250 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0x260 | 1 | 0 | 0 | 1 | 1 | 0 |
| 0x280 | 1 | 0 | 1 | 0 | 0 | 0 |
| 0x290 | 1 | 0 | 1 | 0 | 0 | 1 |
| 0x2A0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0x2B0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 0x2C0 | 1 | 0 | 1 | 1 | 0 | 0 |
| 0x2D0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0x2E0 | 1 | 0 | 1 | 1 | 1 | 0 |
| 0x2F0 | 1 | 0 | 1 | 1 | 1 | 1 |
| 0x300* | 1 | 1 | 0 | 0 | 0 | 0 |
| 0x310 | 1 | 1 | 0 | 0 | 0 | 1 |
| 0x320 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0x330 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0x340 | 1 | 1 | 0 | 1 | 0 | 0 |
| 0x350 | 1 | 1 | 0 | 1 | 0 | 1 |
| 0x360 | 1 | 1 | 0 | 1 | 1 | 0 |
| 0x370 | 1 | 1 | 0 | 1 | 1 | 1 |
| 0x380 | 1 | 1 | 1 | 0 | 0 | 0 |
| 0x390 | 1 | 1 | 1 | 0 | 0 | 1 |
| 0x3A0 | 1 | 1 | 1 | 0 | 1 | 0 |
| 0x3B0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 0x3C0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0x3D0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 0x3E0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0x3F0 | 1 | 1 | 1 | 1 | 1 | 1 |

Note: 1 = Jumper installed, 0 = Jumper not installed.
* Factory Default

## Compatibility Selection and Extended Functions

| Mode | M4 | M3 | M2 | M1 | M0 |
|---|---|---|---|---|---|
| DAS16jr/16 + DAC02 Compatibility* | 0 | X | X | 0 | 0 |
| DAS1602 Compatibility w/1MegaSample FIFO (Single interrupt per sample during burst) | 0 | X | X | 0 | 1 |
| DAS1602 Compatibility and FIFO Status Registers Visible (Single interrupt per burst completion) | 0 | X | X | 1 | 1 |
| Enable FIFO Superset Functionality | 0 | X | X | 1 | X |
| 16-bit DAC Registers | 0 | X | 1 | X | X |
| 16-Sample moving average filter for all 8 or 16 ADC channels | 0 | 1 | X | X | X |

Note: 1 = Jumper installed, 0 = Jumper not installed, X = Don't care.
* Factory Default

The DAS1602 compatibility offers ADC sample bursting as well as ADC trigger blocking.

**Compatibility**

The STX104 is compatible with DAS16jr/16 and the DAS1602. The DAS16jr/16 mode supports 8-bit or 16-bit systems (XT or AT, respectively). The DAS1602 is intended for use with 16-bit systems only (AT). Jumper position M0 selects one of these compatibility modes.

**FIFO Superset Functionality**

Data FIFO superset functionality can be selected for either the DAS16jr/16 or the DAS1602 compatibility modes. The FIFO superset functionality is very similar to other cards on the market. Stuffing jumper position M1 enables FIFO superset functionality. Note that for the DAS1602 compatibility mode, the 1 mega-sample FIFO is enabled, however to maintain register compatibility the FIFO status registers are only visible once the M1 jumper is installed.

**16-Bit DAC Mode**

The 16-bit DAC mode allows the DAC registers to provide 12-bit legacy functionality as well as full 16-bit DAC functionality. Stuffing jumper M2 allows for full 16-bit DAC operation.

**12-Bit DAC Legacy Mode (M2 jumper not stuffed)**

**Base Address + 4** **D/A Channel-A Low Byte** **Read/Write**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| DA7 | DA6 | DA5 | DA4 | X | X | X | X |

**Base Address + 5** **D/A Channel-A High Byte** **Read/Write**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| DA15 | DA14 | DA13 | DA12 | DA11 | DA10 | DA9 | DA8 |

**16-Bit DAC Mode (M2 jumper stuffed)**

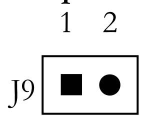
**Base Address + 4** **D/A Channel-A Low Byte** **Read/Write**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| DA7 | DA6 | DA5 | DA4 | DA3 | DA2 | DA1 | DA0 |

**Base Address + 5** **D/A Channel-A High Byte** **Read/Write**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| DA15 | DA14 | DA13 | DA12 | DA11 | DA10 | DA9 | DA8 |

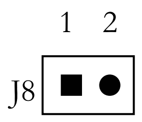
## Unipolar or Bipolar Analog Input



| Jumper | Function |
|---|---|
| 0* | Bipolar Analog Inputs ( +/- values accepted ) |
| 1 | Unipolar Analog Inputs ( + values only ) |

* Factory Default
Note: 1 = Jumper installed, 0 = Jumper not installed.

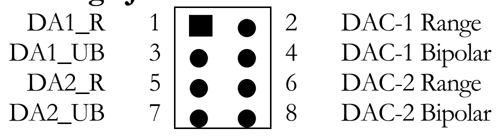
## Differential or Single Ended Analog Input



| Jumper | Function |
|---|---|
| 0* | Differential Inputs, 8-channels |
| 1 | Single-Ended Inputs, 16-channels |

* Factory Default
Note: 1 = Jumper installed, 0 = Jumper not installed.

## DAC Range Settings J5



**J5**

| DA1_UB | DA1_R | DAC-1 Range |
|---|---|---|
| 0 | 0 | 0 to +5 Volts * |
| 0 | 1 | 0 to +10 Volts |
| 1 | 0 | -5 to +5 Volts |
| 1 | 1 | -10 to +10 Volts |

* Factory Default
Note: 1 = Jumper installed, 0 = Jumper not installed.

| DA2_UB | DA2_R | DAC-2 Range |
|---|---|---|
| 0 | 0 | 0 to +5 Volts * |
| 0 | 1 | 0 to +10 Volts |
| 1 | 0 | -5 to +5 Volts |
| 1 | 1 | -10 to +10 Volts |

* Factory Default
Note: 1 = Jumper installed, 0 = Jumper not installed.

**16-Sample Moving Average Filter**

Installing jumper M3 enables the 16-sample moving average filter for all channels. The filter can be reset (or cleared) by writing to the Channel Register. The moving average filter is enabled for all channels and operates completely transparent to ADC acquisition modes. The ADC values read out will be the current sample plus the last fifteen samples summed together and divided by sixteen (average of sixteen ADC samples). It is important to recognize that after the filter is reset or at the beginning of data sampling that it may require at least 16-data samples per channel be taken until the data becomes current. In other words, there is an inherent 16-sample delay in the ADC data that is read out of the ADC Data Register.

The following example illustrates how you would take care of the first 16-samples in a continuous sampling data stream. Say you need to take 100 samples per second over four channels continuously using the moving average filter. The following outline below shows how you could pre-process the data.

1. Write to the Channel Register to sample from channel 0 to channel 3. This also clears the moving average filter. Set a software start-up flag to true to indicate that the first 16-sample times is a start-up period.
2. Set up triggering for 100 samples per second per channel. If you are using Burst mode, then the triggering rate must be 100Hz, otherwise the triggering rate must be 400Hz.
3. Wait for CNV to become zero. This takes approximately 6uS while the moving average filter is cleared.
4. Throw out the first 16-samples per channel. This can be accomplished by using the software start up flag to indicate that we are in a start up mode. Once 64-samples have been read (16-samples x 4-channels), the software start up flag is cleared and normal sampling operation resumes.
5. Now, sample and process data normally.

For high speed sampling where the one mega-sample FIFO is used, a similar software design philosophy could be used where the first 16-data sample sets read out of the FIFO are ignored. Alternatively, if data has been stored directly to a file, during file processing the first 16-sample sets could be ignored. The best location of taking care of the data sample lag time is very much dependent on your specific application and requirements.

## High Vibration Environments

Apex offers factory installed surface mount zero ohm resistors to replace all jumpers.

Chapter

3

# Register Set

## Register Set Summary

| Offset (Decimal) | Offset (Hex) | Write Register | Read Register |
|---|---|---|---|
| 0 | 0 | ADC Software Triggering | ADC LSB Data |
| 1 | 1 | None | ADC MSB Data |
| 2 | 2 | Channel Register *** | Channel Register |
| 3 | 3 | Digital Output | Digital inputs |
| 4 | 4 | DAC Channel-A LSB | None |
| 5 | 5 | DAC Channel-A MSB (Update Channel-A) | None |
| 6 | 6 | DAC Channel-B LSB | None |
| 7 | 7 | DAC Channel-B MSB (Update Channel-B) | None |
| 8 | 8 | Clear Interrupts Register | ADC Status Register |
| 9 | 9 | ADC Control Register | ADC Control Register |
| 10 | A | Pacer Clock Control Register | *FIFO Flags Register |
| 11 | B | ADC Configuration Register | ADC Configuration Register |
| 12 | C | 8254 Counter/Timer 0 Data | 8254 Counter/Timer 0 Data |
| 13 | D | 8254 Counter/Timer 1 Data | 8254 Counter/Timer 1 Data |
| 14 | E | 8254 Counter/Timer 2 Data | 8254 Counter/Timer 2 Data |
| 15 | F | 8254 Counter/Timer Configuration Register | *FIFO Data Status Register |
| 1028 | 404 | DAS1602 Conversion Disable Register ** | None |
| 1029 | 405 | DAS1602 Burst Mode Enable Register ** | None |
| 1030 | 406 | DAS1602 Function Enable Register ** | None |
| 1031 | 407 | None | DAS1602 Extended Status Register ** |

* Available in FIFO Superset mode only (jumper M1 installed)
** Available only in DAS1602 compatibility mode (jumper M0 installed)
*** Writing to this register will reset the acquisition controller, FIFO and moving average filter.

### Definition of Terms

| | | |
|---|---|---|
| ADC | = | Analog-to-Digital Converter |
| DAC | = | Digital-to-Analog Converter |
| FIFO | = | First-In-First-Out |
| LSB | = | Least Significant Byte |
| MSB | = | Most Significant Byte |
| Trigger | = | Trigger causes an ADC sample |
| Burst | = | Channels scanned as fast as possible per trigger |

## ADC Data Register

| Base Address + 0 | | | ADC LSB | | | | Read |
|---|---|---|---|---|---|---|---|
| **D7** | **D6** | **D5** | **D4** | **D3** | **D2** | **D1** | **D0** |
| AD07 | AD06 | AD05 | AD04 | AD03 | AD02 | AD01 | AD00 |

| Base Address + 1 | | | ADC MSB | | | | Read |
|---|---|---|---|---|---|---|---|
| **D7** | **D6** | **D5** | **D4** | **D3** | **D2** | **D1** | **D0** |
| AD15 | AD14 | AD13 | AD12 | AD11 | AD10 | AD09 | AD08 |

**Bit Definitions**

AD[15:0] = ADC data word.  AD0 is the least significant bit.  AD15 is the most significant data bit.

A reading of $0_{10}$ (0x0000) represents a negative full scale input and a value of $65535_{10}$ (0xFFFF) represents an input of positive full scale.

The two registers can both be read simultaneously by simply reading the data as a 16-bit I/O transaction (Examples: "in ax, dx" or "inpw(base_address+0)" ).

The ADC LSB register provides the least significant data byte and the ADC MSB register provides the most significant data byte.  Each of the two registers can be read in any order (when FIFO Superset is not enabled).  Writing any data to the ADC LSB register issues a software trigger.

Data bandwidth between the STX104 and CPU (PC/104 ISA bus) can be doubled by simply reading the ADC register as a 16-bit register.  Software examples are shown below.

When FIFO Superset is enabled (M1 jumper installed) the ADC data is read out of the FIFO as either a word or a byte at a time.  To read out the data correctly as a byte at a time, you must first read the LSB and then the MSB.  Once the MSB is read, the FIFO is considered to be read and FIFO is advanced to the next value to be read (if not empty).   You may continue to read the data until the FIFO empty flag is set.  It is possible to continuously sample analog inputs (continuous ADC sampling or triggering) and continuously read FIFO without any breaks in sampling.

It is the programmer's responsibility to be sure that data is read out in the correct sequence, as well as starting and stopping the acquisition as required.

**Software Examples**

Examples of how to read the ADC data:

8-Bit Read in C/C++:
```
unsigned ad_value[16];
...
adc_value(channel) = inportb(base_address+0) + ( inportb(base_addres+1) << 8 );
...
```

16-Bit Read in C/C++:
```
unsigned ad_value[16];
...
adc_value(channel) = inpw(base_address+0);
...
```

8-Bit Read and FIFO enabled in C/C++:
```
unsigned ad_value[16];
...
if ( fifo_not_empty() == true ) /* function to check fifo status */
{
 adc_value(channel) = inportb(base_address+0) + ( inportb(base_addres+1) << 8 );
}
...
```

16-Bit Read and FIFO enabled in C/C++:
```
unsigned ad_value[16];
...
if ( fifo_not_empty() == true ) /* function to check fifo status */
{
 adc_value(channel) = inpw(base_address+0);
}
...
```

**Using the REP INSW Instruction**

Using the REP INSW instruction provides a very high speed mechanism for reading out ADC data from the FIFO.  This instruction is available on 286 CPUs and above.  In fact, this is faster and easier to use than DMA; both in implementation and execution.  Using REP INSW alleviates video problems related with DMA as well as improved CPU timing management (i.e. DMA adds timing holes in the system that are not easily managed by simple real-time kernels).  The speed of any ISA bus I/O transaction is dependent on the CPU ISA bus speed which is usually set in the BIOS setup; use this setting with caution as it may affect performance of other cards and/or can cause data loss.  This is usually available on many CPU cards.

16-Bit Read with FIFO enabled in C/C++ using REP INSW:
```
typedef unsigned short WORD;

void insw(WORD port, void *buf, int count)
{
  _ES = FP_SEG(buf);   /* Segment of buf */
  _DI = FP_OFF(buf);   /* Offset of buf  */
  _CX = count;         /* Number to read */
  _DX = port;          /* Port           */
  asm   REP INSW;
}
```

```
Copyright (c) 1997-2005 by Apex Embedded Systems
Need something different?  Please let us know.


 void main()
 {
   unsigned *data[512];
   ...
   insw(0x300, data, 512);  /* assumes 512 samples in FIFO */
   ...
 }
```

## Voltage Input Conversion

| Input Range | Resolution | Input Voltage | | Binary Code | |
|---|---|---|---|---|---|
| | | -FS | +FS | -Full Scale | +Full Scale |
| ± 10V | 305uV | -10 | +10 | 0000 0000 0000 0000 | 1111 1111 1111 1111 |
| ± 5V | 153uV | -5 | +5 | 0000 0000 0000 0000 | 1111 1111 1111 1111 |
| ± 2.5V | 76uV | -2.5 | +2.5 | 0000 0000 0000 0000 | 1111 1111 1111 1111 |
| ± 1.25V | 38uV | -1.25 | +1.25 | 0000 0000 0000 0000 | 1111 1111 1111 1111 |
| 0-10V | 153uV | 0.00 | +10 | 0000 0000 0000 0000 | 1111 1111 1111 1111 |
| 0-5V | 76uV | 0.00 | +5 | 0000 0000 0000 0000 | 1111 1111 1111 1111 |
| 0-2.5V | 38uV | 0.00 | +2.5 | 0000 0000 0000 0000 | 1111 1111 1111 1111 |
| 0-1.25V | 19uV | 0.00 | +1.25 | 0000 0000 0000 0000 | 1111 1111 1111 1111 |

An example of general conversion in C/C++ is shown below illustrating how to set the gain and use it to determine the actual input voltage.

```
/****************************************************************************
   Apex Embedded Systems
   September 25, 2002
   STX104 Analog Input Demo
****************************************************************************/

#include <conio.h>
#include <stdio.h>
#include <dos.h>

/*****************************************************************************/
float adc_gain[8] = { 20.0/65536.0, 10.0/65536.0, 5.0/65536.0, 2.5/65536.0,
                      10.0/65536.0,  5.0/65536.0, 2.5/65536.0, 1.25/65536.0 };
float adc_offset[8] = { -10.0, -5.0, -2.5, -1.25, 0.0, 0.0, 0.0, 0.0 };

/*****************************************************************************/
float Adc_Voltage ( void )
{
        long sum;
        int index;
        float voltage;

        /* take several samples and average - especially useful for calibration */
        sum = 0;
        outportb(0x308, 0x00); /* reset INT bit */
        for ( index=0; index <256; index++ )
        {
                outportb(0x300, 0x00); /* start a sample */
                while ( (inportb(0x308) & 0x10) == 0x00 ); /* wait for sample */
                sum = sum + ((long)inpw(0x300));
                outportb(0x308, 0x00); /* reset INT bit */
        }
        sum = sum / 256;
        index = inportb( 0x30B ) & 0x07;
        voltage = ( (float) sum ) * adc_gain[index] + adc_offset[index];
        return voltage;
}
```

```
/**************************************************************************/
void main()
{
        int x,y;
        /* base address set to factory default at 0x300 */
        /* initialize STX104 */
        outportb(0x309, 0x00); /* no interrupts, no DMA and s/w trigger */
        outportb(0x302, 0x00); /* look at channel zero only */

        printf("0) +10.0V   input\n");
        printf("1)  +5.0V   input\n");
        printf("2)  +2.5V   input\n");
        printf("3)  +1.25V  input\n");
        printf("9) Quit this test\n");

        do {
                y = inportb( 0x30B ) & 0x07;
                if ( (y & 0x04) == 0x00 ) printf("Bipolar:  ");
                else printf("Unipolar: ");
                y = y & 0x03;
                printf(" Gain=");
                if ( y == 0 )      printf("10V,   ");
                else if ( y == 1 ) printf(" 5V,   ");
                else if ( y == 2 ) printf(" 2.5V, ");
                else if ( y == 3 ) printf(" 1.25V,");
                printf(" Voltage = %+10.6f        \r", Adc_Voltage());

                if ( kbhit() )
                {
                        y = getch();
                        /* set gain */
                        if ( y == '0' )      { x = 0;   outportb( 0x30B, x ); }
                        else if ( y == '1' ) { x = 1;   outportb( 0x30B, x ); }
                        else if ( y == '2' ) { x = 2;   outportb( 0x30B, x ); }
                        else if ( y == '3' ) { x = 3;   outportb( 0x30B, x ); }
                        else if ( y == '9' ) x = 9;
                }
        } while ( x != 9 );
        printf("\n\n");
}
```

## ADC Software Triggering ADC

**Base Address + 0** **Start ADC Conversion** **Write**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| X | X | X | X | X | X | X | X |

**Bit Definitions**

|  |  |  |
|---|---|---|
| X | = | Don't care |

Writing any data to this register will invoke a software trigger, only if it is selected in the ADC control register TS[1:0].

## Channel Register

| Base Address + 2 | | | Channel Scan | | | | Read/Write |
|---|---|---|---|---|---|---|---|
| **D7** | **D6** | **D5** | **D4** | **D3** | **D2** | **D1** | **D0** |
| LC3 | LC2 | LC1 | LC0 | FC3 | FC2 | FC1 | FC0 |

**Bit Definitions**

| | | |
|---|---|---|
| CH[3:0] | = | Current Channel to be sampled. See ADC Status Register. This value is sent to the analog multiplexers to set the current analog channel to be sampled. This value can be read in the ADC Status Register. |
| FC[3:0] | = | First Channel. This is the first analog channel that is sampled. When this value is written, it will also set the current-channel ( CH = FC ). |
| LC[3:0] | = | Last Channel. This is the last analog channel that is sampled. Once this channel has been sampled, the current-channel value will wrap to the first-channel value. |

The channel register contains *first-channel* and the *last-channel* in the scanning range of the analog multiplexer. When the channel register is written the *first-channel* value is also written to the *current-channel* which sets the analog channel to be sampled. Upon an ADC trigger, the ADC samples the *current-channel*, and the multiplexer is advanced to the next channel. The *current-channel* value will wrap from the *last-channel* to the to the *first-channel* once the *last-channel* as been sampled.

The *current-channel* value is incremented only when the ADC is sampled or triggered. For every ADC trigger, the *current-channel* is incremented. The *current-channel* is presented as CH[3:0] in the ADC Status Register.

If the STX104 is configured for differential input mode, the most significant bit of the *current-channel* (and therefore *first-* and *last-channel*) is ignored.

If the FIFO function is enabled, the user software must track the channel being read out of the FIFO. In other words, it is the software which must maintain synchronization.

If the *first-channel* is the same as the *last-channel*, then the analog channel (multiplexer) is not changed. This can be useful for sampling the same channel continuously.

It is recommended to have *first-channel* ≤ *last-channel* to prevent confusion of the channel sequencing as illustrated in examples (C) and (D) below.

**FIFO Reset**

Writing to the channel register will reset the FIFO when the FIFO Superset mode is enabled (jumper M1 is installed) and/or DAS1602 mode enabled (jumper M0 is installed).

**Acquisition Controller Reset**

Writing to the Channel Register resets the internal acquisition controller in all modes.

**Moving Average Filter Reset**

Writing to the Channel Register also resets the internal moving average filter in all modes when jumper M3 is installed. The CNV bit (see Status Register) will become active for approximately six microseconds while the moving average filter is reset.

**Examples**

Example channel sequencing (all in hexadecimal values):

| | | |
|---|---|---|
| a) | LC = D, FC = 3 | |
| | 16-Channel, Single Ended: | 3,4,5,6,7,8,9,A,B,C,D,3,4,5,... |
| | 8-Channel, Differential: | 3,4,5,3,4,5,... |
| b) | LC = 1, FC = 9 | |
| | 16-Channel, Single Ended: | 9,A,B,C,D,E,F,0,1,9,A,B,C,... |
| | 8-Channel, Differential: | 1,1,1,1,... |
| c) | LC = 6, FC = 5 | |
| | 16-Channel, Single Ended: | 5,6,5,6,... |
| | 8-Channel, Differential: | 5,6,5,6,... |
| d) | LC = 5, FC = 6 | |
| | 16-Channel, Single Ended: | 5,6,7,8,9,A,B,C,D,E,F,0,1,2,3,4,5,6,7,8,9,... |
| | 8-Channel, Differential: | 6,7,0,1,2,3,4,5,6,7,0,1,2,... |

**SS&H Output**

The Start Sample and Hold (SS&H) signal, at pin 14 of the I/O connector, is a TTL output used to drive the sample and holds line of external simultaneous sample and hold cards. The behavior of the SS&H output signal is related to the Channel Register. Writing any value to the Channel Register will bring the SS&H line active high. The SS&H line will go low, indicating a hold, when the *first_channel* sampling has completed (i.e. the input multiplexers now looking at the next channel). The SS&H line will return high when *last_channel* has been sampled and the multiplexers wrap back to the *first_channel*.

## Digital Input/Output Register

**Base Address + 3** **Digital Inputs** **Read**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| CM1 | CM0 | 0 | 0 | DIN3 | DIN2<br>GATE0 | DIN1 | DIN0<br>TRIG |

**Base Address + 3** **Digital Outputs** **Write**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| X | X | X | X | DOUT3 | DOUT2 | DOUT1 | DOUT0 |

**Bit Definitions**

DIN[3:0] = Digital Inputs.

DIN0 also functions as an ADC external trigger input. As a trigger input, it is deglitched by an internal match-filter and requires a minimum pulse width of no less than 200nS. See the Interrupt Summary section for further functionality.

DIN2 also functions as counter-zero gate input.

DOUT[3:0] = Digital Outputs. These bits are write only. Non-inverted digital output bits. You must maintain a copy in software for bit manipulation.

CM[1:0] = Compatibility mode. These bits are set to zero for DAS16jr/16 mode. These bits are set to one for DAS1602 mode compatibility.

X = Don't care

## ADC Status Register

The ADC status register provides ADC status information.

**Base Address + 8** **Status Register** **Read**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| CNV | UB | SD | INT | CH3 | CH2 | CH1 | CH0 |

**Bit Definitions**

CNV = ADC Conversion (and/or burst in DAS1602 mode) in progress. Writing to the Channel Register may cause the CNV bit to become active indicating that a STX104 internal reset is in progress (typically less than 1uS, and less than 10uS when moving average filter is enabled).
1 = ADC conversion, scan or acquisition reset in progress.
0 = ADC Idle (default)

UB = Unipolar / Bipolar ADC input mode setting (J9):
0 = bipolar.  Measure both negative and positive input voltages (J9 not stuffed)
1 = unipolar.  Measure only positive input values

SD = Single-ended / Differential ADC input mode setting (J8):
0 = Single-ended
1 = Differential (J8 not stuffed)

INT = Interrupt request status bit.
0 = No interrupt pending (default)
1 = Interrupt is pending; ADC trigger or burst conversion has completed

Interrupt Overflow

Note:  ADC conversions continue to occur on schedule (via selected trigger source) regardless of whether this bit is cleared.  If a new conversion occurs before this bit is cleared, an over-run condition has occurred.  Therefore, the programmer must ensure that the interrupt rate is not faster than the capacity of the CPU and software to respond.

If FIFO Superset is enabled (jumper M1 stuffed), then the only over-run that will occur is if the FIFO is full (FIFO Full flag is true or FF='1').  Thus, interrupt latency requirements are greatly relaxed.

If ADC interrupts are not enabled, this bit can still be used to determine when an ADC conversion has occurred when polling this bit.

CH[3:0] = Current ADC channel.  This is the channel currently selected on the board and is the channel that will be used for the next ADC conversion provided CNV=0 (unless a new value is written to the channel register).  The CH[3:0] will change shortly after an ADC trigger.

X = Don't care

## Clear Interrupt Register

| Base Address + 8 | | | Clear Interrupt | | | | Write |
|---|---|---|---|---|---|---|---|
| **D7** | **D6** | **D5** | **D4** | **D3** | **D2** | **D1** | **D0** |
| X | X | X | X | X | X | X | X |

**Bit Definitions**

| | | |
|---|---|---|
| X | = | Don't care |

Writing any data to this location clears any pending ADC interrupts, setting the INT bit to zero.

**FIFO Interrupt**

When the FIFO Superset mode is enabled (jumper M1 stuffed), writing any data to this location clears any pending FIFO interrupts setting the FF_INT bit to zero.  See FIFO Flags Register.

## ADC Control Register

Without FIFO Superset Enabled (jumper not installed at M1).

**Base Address + 9** **Control Register** **Read/Write**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| AIE | INT2 | INT1 | INT0 | X | DMA | TS1 | TS0 |

FIFO Superset Enabled (jumper installed at M1).

**Base Address + 9** **Control Register** **Read/Write**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| AIE | INT2 | INT1 | INT0 | FIE | DMA | TS1 | TS0 |

**Bit Definitions**

AIE = ADC Interrupt Enable.
    0 = Disable interrupt (default)
    1 = Enable interrupt

INT[2:0] = Interrupt select:
    000 = Not valid, interrupts disabled (default)
    001 = Not valid, interrupts disabled
    010 = IRQ2
    011 = IRQ3
    100 = IRQ4
    101 = IRQ5
    110 = IRQ6
    111 = IRQ7

FIE = FIFO Interrupt Enable (only when jumper M1 is installed):
    0 = Disable FIFO interrupt (default)
    1 = Enable FIFO interrupt

DMA = Direct Memory Access (DMA) enable:
    0 = Disable DMA transactions (default)
    1 = Enable DMA transactions

The DMA request lines are tri-stated until the DMA bit is enabled, thus allowing multiple DMA devices to share the selected DMA channel provided that they are not enabled at the same time.

TS1 = Trigger select or trigger enable:
    0 = Disable hardware trigger, software trigger only (default)
    1 = Enable hardware trigger (source selected with TS0 bit)

TS0 = Trigger select:
    0 = External trigger using DIN0.  Rising edge on DI0 generates ADC conversions (default)
    1 = Internal trigger using counter/timer.  Rising edge on the output of counter/timer two (8254) generate ADC conversions.

X = Don't Care

The control register is used to configure interrupts, select DMA mode and select ADC trigger source.

See Interrupt Summary for more information on interrupts.

DMA

We strongly encourage you to use the REP INSW instruction as it performs better than DMA and is substantially simpler to set up and use. Note that DMA performs I/O to memory transfers a byte at a time (DMA 1 and DMA 3 are byte wide transfers) while REP INSW performs I/O to memory transfers a word at a time. If you are designing a real-time system, you will have better timing control over your system by using REP INSW instruction over DMA. Since processor generated I/O cycles are faster than DMA generated cycles (typically 350ns versus 800ns), data transfer can take place faster than DMA.

When DMA is enabled, the 1 mega-sample FIFO is used internally.

In DMA mode, receiving a terminal count from the CPU will generate an interrupt, if the interrupt is enabled. If you are in DAS1602 mode (jumper M0 is installed), receiving a terminal count will set the Conversion Disable bit to false, thus disabling any additional ADC triggers (or sampling). Writing 0x00 to the Conversion Disable Register will allow ADC sampling to continue.

In order to use DMA, you must set up the computer's DMA controller and page registers before enabling DMA on the STX104 board.

The speed of any ISA bus I/O transaction is dependent on the CPU ISA bus speed which is usually set in the BIOS setup; use this setting with caution as it may affect performance of other cards and/or can cause data loss. This is usually available on many CPU cards.

## Interrupt Summary

One interrupt per sample when in burst mode

Without FIFO Superset Enabled (jumper not installed at M1).

| AIE | FIE | DMA | Interrupt Function |
|---|---|---|---|
| 0 | X | 1 | No interrupt generated.<br>Poll INT to determine when a DMA terminal count is received from the DMA controller to indicate completion of the DMA transfer. |
| 1 | X | 0 | Interrupt generated when an ADC conversion has completed.  Write to the Clear Interrupt Register to clear the interrupt.  In DAS1602 compatibility mode (jumper M0 installed), an interrupt is generated for each sample when in burst mode. |
| 1 | X | 1 | Interrupt generated when a DMA terminal count is received from the DMA controller to indicate completion of the DMA transfer. |

One interrupt per burst completion when in burst mode

FIFO Superset Enabled (jumper installed at M1).

| AIE | FIE | DMA | Interrupt Function |
|---|---|---|---|
| 0 | 0 | 0 | No interrupt generated.<br>Poll INT to determine when an ADC conversion has completed.<br>Poll INT_FF to determine when 512 additional samples have been queued to the FIFO. |
| 0 | X | 1 | No interrupt generated.<br>Poll INT to determine when a DMA terminal count is received from the DMA controller to indicate completion of the DMA transfer.<br>Poll INT_FF to determine when 512 additional samples have been queued to the FIFO. |
| 1 | X | 1 | Interrupt generated when a DMA terminal count is received from the DMA controller to indicate completion of the DMA transfer. |
| 0 | 1 | 0 | Interrupt generated when 512 additional samples are deposited in the FIFO.  Write to the Clear Interrupt Register to clear the interrupt.<br>Poll INT to determine when deglitched DIN0 rising-edge has occurred.  By connecting CT_OUT0 or CT_OUT2 to DIN0 you can create a polled timing function.  Any external input can produce a polled rising-edge.  If GCTRL bit is set then this can be used to detect the beginning of pacer clock gating (i.e. start of one or more samples).  Writing to the Clear Interrupt Register will also clear the INT bit. |
| 1 | 0 | 0 | Interrupt generated when an ADC conversion has completed.  Write to the Clear Interrupt Register to clear the interrupt.  In DAS1602 compatibility mode (jumper M0 installed), an interrupt is generated for each burst completion when in burst mode.<br>Poll INT_FF to determine when 512 additional samples have been queued to the FIFO. |
| 1 | 1 | 0 | Interrupt generated when 512 samples deposited in the FIFO.  Write to the Clear Interrupt Register to clear the interrupt.<br>Interrupt generated when deglitched DIN0 rising-edge has occurred.  By connecting CT_OUT0 or CT_OUT2 to DIN0 you can create an interrupt timing function.  Any external input can produce an interrupt.  If GCTRL bit is set then this can be used to detect the beginning of pacer clock gating (i.e. start of a group of samples).  Writing to the Clear Interrupt Register will also clear the INT bit. |

## Pacer Clock Control Register

DAS16jr/16 Compatibility Mode (M0 jumper not stuffed).

**Base Address + 10** **Pacer Clock Control** **Write**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| X | X | X | X | X | X | CT_SRC0 | GCTRL |

DAS1602 Compatibility Mode (M0 jumper stuffed).

**Base Address + 10** **Pacer Clock Control** **Write**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| BL3 | BL2 | BL1 | BL0 | X | X | CT_SRC0 | GCTRL |

**Bit Definitions**

BL[3:0] = Burst Length. Determines the number of conversions per trigger when in burst mode. One to sixteen samples (single-ended) or up to eight channels (differential) in a burst. When not in Burst mode, then these bits have no function. The burst length is only used in the DAS1602 compatibility mode.

If BL[3:0]=0x0, then 1-channel burst is performed. If BL[3:0]=0xF, then 16-channel burst is performed.

CT_SRC0 = Counter 0 Clock Source:
1 = Counter 0 Clock Source is a 100KHz on-board reference frequency. CT_SRC0 (J7.4) gates this signal. When this bit is high (default), the 100KHz signal runs, otherwise the 100KHz clock is stopped.
0 = Counter 0 Clock Source to Counter 0 is an inverted polarity copy of CT_CLK0 input. CT_CLK0 is connected to a 10K ohm pull-up resistor.

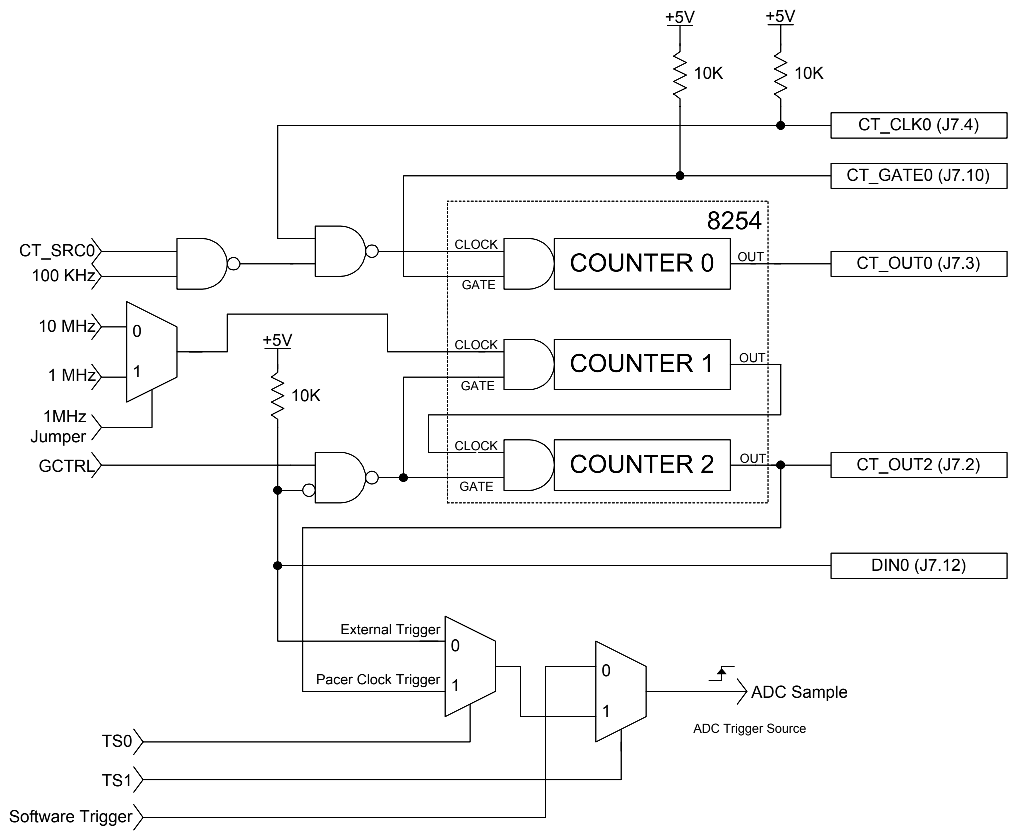
GCTRL = Counters 1 and 2 gate control:
0 = Counters 1 and 2 run freely with no gating.
1 = Counters 1 and 2 are gated by DIN0 (J7.12). DIN0 is connected to a 10K ohm pull-up resistor.

X = Don't care

Details on how to use the counter/timer control bits are shown in schematic form on the following page.

## ADC Clock and Triggering Summary

The drawing below summarizes the triggering and pacer clock mechanisms in the basic DAS16 mode.



| TS1 | TS0 | Trigger Function |
|---|---|---|
| 0 | X | Software triggered ADC sampling only |
| 1 | 0 | Trigger on rising edge of DIN0 |
| 1 | 1 | Trigger on pacer clock output (CT_OUT2) |

## FIFO Status Registers

The FIFO Status Registers are only available when FIFO Superset is Enabled (M1 jumper stuffed).

**Base Address + 10** **FIFO Flags Register** **Read**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| FF_INT | X | FF | FE | FBR11 | FBR10 | FBR9 | FBR8 |

**Base Address + 15** **FIFO Data Status Register** **Read**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| FBR7 | FBR6 | FBR5 | FBR4 | FBR3 | FBR2 | FBR1 | FBR0 |

**Bit Definitions**

| | | |
|---|---|---|
| FF_INT | = | FIFO Data Interrupt.  An interrupt will occur for 512 or more samples queued in the FIFO.  This bit can be polled and/or use to generate an interrupt to the CPU (setting FIE bit true in the ADC Control Register).<br>0 = No FIFO interrupt  (default at reset or power-up)<br>1 = FIFO interrupt active.  To Clear Interrupt Register, write any value at Clear Interrupt Register. |
| FF | = | FIFO Full Flag.<br>0 = FIFO not full  (default at reset or power-up)<br>1 = FIFO full/overflow has occurred. |
| FE | = | FIFO Empty Flag.<br>1 = Empty  (default at reset or power-up)<br>0 = FIFO not empty, data is present |
| FBR[11:0] | = | FIFO Data Blocks Remaining.  This provides a mechanism for measuring amount of data remaining within the FIFO.  Each block is 256 samples.  For example, if FBR=0x05 then at least 1,280 samples remain to be read out of the STX104 FIFO.  If FBR=0x00 and FE='0' then there are less than 256 samples remaining to be read out of the FIFO.  When FBR>0, use REP INSW to read in 256 samples at a time (1 block).  When FBR=0, use a software loop that monitors the FE bit while reading in the samples until the FIFO is empty. |
| X | = | Don't Care |

Note that writing to the Channel Register will reset the FIFO.

## ADC Configuration Register

Base Address + 11 — Analog Configuration — Write

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| X | X | X | X | X | X | G1 | G0 |

Base Address + 11 — Analog Configuration — Read

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| X | X | X | X | 0 | ADBU | G1 | G0 |

**Bit Definitions**

ADBU = ADC bipolar/unipolar:
- 0 = bipolar setting (jumper J9 not installed)
- 1 = unipolar setting

G[1:0] = ADC gain setting:
- 00 = gain of x1 (default)
- 01 = gain of x2
- 10 = gain of x4
- 11 = gain of x8

X = Don't care

Writing to this register sets the analog input gain for all 8/16 analog inputs. The ADBU bit is set by hardware. The current input gain is determined by reading this register.

**Gain summary**

| Input Range | Resolution | ADBU | G1 | G0 |
|---|---|---|---|---|
| ± 10V | 305uV | 0 | 0 | 0 |
| ± 5V | 153uV | 0 | 0 | 1 |
| ± 2.5V | 76uV | 0 | 1 | 0 |
| ± 1.25V | 38uV | 0 | 1 | 1 |
| 0-10V | 153uV | 1 | 0 | 0 |
| 0-5V | 76uV | 1 | 0 | 1 |
| 0-2.5V | 38uV | 1 | 1 | 0 |
| 0-1.25V | 19uV | 1 | 1 | 1 |

The ADC data range is $-32768_{10}$ to $32767_{10}$ for each of the input ranges listed below.

See the software example in the ADC Data Register section earlier in this chapter.

The gain setting is the ratio between the ADC full-scale range and the effective input signal range. For example, if the ADC full-scale range is 0-10V, a gain setting of 2 creates an input signal range of 0-5V, and gain setting of 4 creates an input range of 0-2.5V.

## 8254 Counter/Timer Registers

Data registers to/from the each of the three counter/timers.

**Base + 12** **8254 Counter/Timer 0 Data** **Read/Write**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| D7 | D7 | D5 | D4 | D3 | D2 | D1 | D0 |

**Base + 13** **8254 Counter/Timer 1 Data** **Read/Write**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| D7 | D7 | D5 | D4 | D3 | D2 | D1 | D0 |

**Base + 14** **8254 Counter/Timer 2 Data** **Read/Write**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| D7 | D7 | D5 | D4 | D3 | D2 | D1 | D0 |

Counter/Timer configuration register.

**Base + 15** **8254 Counter/Timer Mode Configuration Register** **Write**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| SC1 | SC0 | RW1 | RW0 | M2 | M1 | M0 | BCD |

**Bit Definitions**

BCD = Binary Coded Decimal (BCD) Counter (4 decades) if set to one, otherwise 16-bit binary counter.

M[2:0] = Counter/Timer Mode Select:
000 = Mode 0, Interrupt on terminal count
001 = Mode 1, Hardware retriggerable one-shot
X10 = Mode 2, Rate generator
X11 = Mode 3, Square wave generator
100 = Mode 4, Software triggered strobe
101 = Mode 5, Hardware triggered strobe (retriggerable)

RW[1:0] = Read/Write:
00 = Counter latch command
01 = Read/Write least significant byte only
10 = Read/Write most significant byte only
11 = Read/Write least significant byte first, then most significant byte.

SC[1:0] = Select Counter:
00 = Select Counter 0
01 = Select Counter 1
10 = Select Counter 2
11 = Read-Back Command (see read operations)

## DAC Data Registers

**Legacy Bit Alignment**

12-Bit DAC Legacy Mode (M2 jumper not stuffed).

**Base Address + 4** **DAC Channel-A Low Byte (LSB)** **Read/Write**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| DB7 | DB6 | DB5 | DB4 | X | X | X | X |

**Base Address + 5** **DAC Channel-A High Byte (MSB)** **Read/Write**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| DB15 | DB14 | DB13 | DB12 | DB11 | DB10 | DB9 | DA8 |

**Base Address + 6** **DAC Channel-B Low Byte (LSB)** **Read/Write**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| DB7 | DB6 | DB5 | DB4 | X | X | X | X |

**Base Address + 7** **DAC Channel-B High Byte (MSB)** **Read/Write**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| DB15 | DB14 | DB13 | DB12 | DB11 | DB10 | DB9 | DA8 |

**16-Bit DAC Mode**

DAC Integer Bit Alignment (M2 jumper stuffed).

**Base Address + 4** **DAC Channel-A Low Byte (LSB)** **Read/Write**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| DA7 | DA6 | DA5 | DA4 | DA3 | DA2 | DA1 | DA0 |

**Base Address + 5** **DAC Channel-A High Byte (MSB)** **Read/Write**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| DA15 | DA14 | DA13 | DA12 | DA11 | DA10 | DA9 | DA8 |

**Base Address + 6** **DAC Channel-B Low Byte (LSB)** **Read/Write**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| DB7 | DB6 | DB5 | DB4 | DB3 | DB2 | DB1 | DB0 |

**Base Address + 7** **DAC Channel-B High Byte (MSB)** **Read/Write**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| DA15 | DA14 | DA13 | DA12 | DB11 | DB10 | DB9 | DA8 |

**Bit Definitions**

| | | |
|---|---|---|
| DA[15:4] | = | DAC Channel-A data word in 12-Bit mode. DA4 is the least significant bit and DA15 is the most significant data bit. |
| DB[15:4] | = | DAC Channel-B data word in 12-Bit mode. DA4 is the least significant bit and DA15 is the most significant data bit. |
| DA[15:0] | = | DAC Channel-A data word in 16-Bit mode. DA0 is the least significant bit and DA15 is the most significant data bit. |
| DA[15:0] | = | DAC Channel-B data word in 16-Bit mode. DA0 is the least significant bit and DA15 is the most significant data bit. |

Updating DAC Outputs

Each channel is updated once the MSB is written.  Writing only the MSB will update the DAC channel output.  The results of changing jumper settings at J5 will only take affect after writing the MSB on the DAC output.  The two 8-bit DAC registers can be written simultaneously by writing the data as a 16-bit I/O transaction (Examples:  "out dx, ax" or  "outpw(base_address+4, dac_value)" ).

DAC outputs are available in either DAS16jr/16 or DAS1602 modes.  The DAC outputs are always enabled and available for use.  The DAC output bit alignments can be adjusted for either 12-bit legacy operation or full 16-bit DAC mode.

At power-up or reset the DAC outputs are at set to zero volts.

**Software Examples**

Examples of how to write to the DAC output register in 16-bit DAC mode.

8-Bit Writes in C/C++:
```
unsigned int dac_value;
...
outportb( base_address+4, dac_value & 0xFF );
outportb( base_address+5, dac_value >> 8 );
...
```

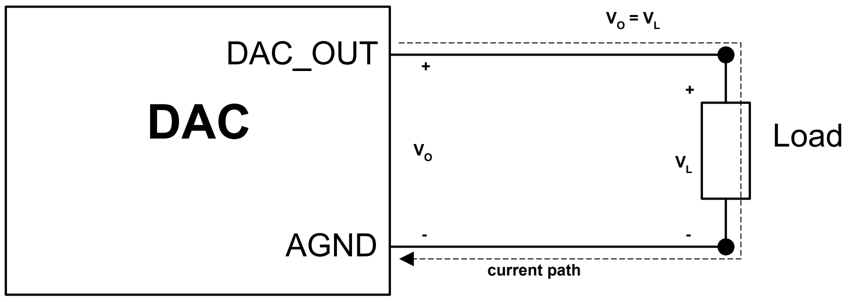
16-Bit Write in C/C++:
```
unsigned int dac_value;
...
outpw( base_address+4, dac_value );
...
```

**Output Voltage Conversion**

| Output Range | Resolution | Input Voltage | | Binary Code | |
|---|---|---|---|---|---|
| | | -FS | +FS | -Full Scale | +Full Scale |
| ± 10V | 305uV | -10 | +10 | 0000 0000 0000 0000 | 1111 1111 1111 1111 |
| ± 5V | 153uV | -5 | +5 | 0000 0000 0000 0000 | 1111 1111 1111 1111 |
| 0-10V | 153uV | 0.00 | +10 | 0000 0000 0000 0000 | 1111 1111 1111 1111 |
| 0-5V | 76uV | 0.00 | +5 | 0000 0000 0000 0000 | 1111 1111 1111 1111 |

**DAC Wiring**

The DAC output should be wired as shown below.



## DAS1602 Compatibility Configuration and Status Register

These registers are only available in DAS1602 Compatibility Mode (jumper M0 is stuffed).

**Base+1028** **Conversion Disable Register** **Write**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| CD7 | CD6 | CD5 | CD4 | CD3 | CD2 | CD1 | CD0 |

**Base+1029** **Burst Mode Enable Register** **Write**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| BME7 | BME6 | BME5 | BME4 | BME3 | BME2 | BME1 | BME0 |

**Base+1030** **Function Enable Register** **Write**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| FE7 | FE6 | FE5 | FE4 | FE3 | FE2 | FE1 | FE0 |

**Base+1031** **Extended Status Register** **Read**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| 0 | BME | FE | CD | 0 | 0 | WS | CLK |

**Bit Definitions**

FE[7:0] = DAS1602 Function Enable Register.  On power-up or reset the DAS1602 functions are disabled.  Writing 0x40 ($64_{10}$) to this register enables the DAS1602 functions.  Writing a 0x00 to this register disables DAS1602 functions.

CD[7:0] = Conversion Disable Register.  On power-up or reset the conversion triggers are enabled.  This register is only available if FE bit is true (DAS1602 Functions are enabled).  Writing a 0x00 to this register enables ADC triggering.  Writing 0x40 ($64_{10}$) to this register disables ADC triggering.  If the FIFO Superset jumper M1 is not installed, then when conversions are disabled, the FIFO is reset.  If the jumper M1 is installed, the FIFO is only reset by writing to the channel register.

BME[7:0] = Burst Mode Enable Register.  On power-up or reset the burst mode is disabled.  This register is only available if FE bit is true (DAS1602 Functions are enabled).  Writing a 0x00 to this register disables ADC burst mode.  Writing 0x40 ($64_{10}$) to this register enables ADC burst mode.

BME = Burst Mode Enabled:
- 0 = disabled (default)
- 1 = enabled

| | | |
|---|---|---|
| FE | = | DAS1602 Function Enabled:<br>0 = disabled, DAS16jr/16 functionality only (default)<br>1 = enabled |
| CD | = | Conversions Allowed:<br>0 = ADC Conversions Disabled<br>1 = ADC Conversions Allowed (default) |
| WS | = | Wait States.  This bit is always zero for no wait states. |
| CLK | = | Counter/Timer clock source:<br>0 = 1MHz selected (based on jumper 1MHz)<br>1 = 10MHz selected |

Note that when DMA mode is enabled, receiving a terminal count will disable ADC triggering (CD='0'). Writing 0x00 to the Conversion Disable Register will allow ADC triggering to occur.

Chapter

4

# Calibration

*Calibration made easy*

**DAC Calibration**

Perform the following procedure to adjust DAC channel one:

1. Select the desired output range by adjusting jumper settings at J5.
2. Install jumper M2 to select 16-bit DAC output.
3. Output full scale by writing 65535 (0xFFFF) to the DAC one output register.
4. Adjust potentiometer DA1_G (R2) until the desired full scale is reached. If you have selected 0-5V output, then adjust the output (J7 pin 17 to J7 pin 18) for 5.0000 volts.
5. Calibration is complete.

Perform the following procedure to adjust DAC channel two:

6. Select the desired output range by adjusting jumper settings at J5.
7. Install jumper M2 to select 16-bit DAC output.
8. Output full scale by writing 65535 (0xFFFF) to the DAC two output register.
9. Adjust potentiometer DA2_G (R1) until the desired full scale is reached. If you have selected 0-5V output, then adjust the output (J7 pin 16 to J7 pin 18) for 5.0000 volts.
10. Calibration is complete.

**ADC Calibration**

Perform the following procedure to adjust ADC input. We recommend averaging ADC values over at least 256 samples in order to provide the most accurate calibration possible, reference the Voltage Input Conversion section under the ADC Data Register description. This calibration uses the DAC outputs, therefore, the DAC outputs must be calibrated first. Note that the potentiometer PGA_OFF (R3) is factory calibrated and does not need adjustment.

1. Select the desired input mode by configuring J8 for differential or single-ended inputs.
2. Select the desired input range by configuring J9 for unipolar or bipolar inputs.
3. Set the desired gain by writing to the ADC Configuration Register (base_address + 11) to the desired ADC input gain. Note that the best overall calibration will be achieved when the ADC gain is set to x8.
4. Set the Channel Register (base_address + 2) to zero so that we are observing input channel zero.
5. Offset Adjustment.
   a. In the case of single-ended inputs (jumper J8 installed), short J7 pin 35 to J7 pin 37; effectively connecting the single-ended input to analog ground. In the case of differential inputs (jumper J8 not installed), short J7 pin 35 and pin 36 to J7 pin 37; effectively connecting the differential input to analog ground.
   b. While performing ADC conversions and reading the ADC value, adjust potentiometer ADC_OFFSET (R4) until the nearest zero value can be achieved.
6. Remove all jumpers from the I/O connector at J7.
7. Gain Adjustment.
   a. In the case of single-ended inputs (jumper J8 installed), connect J7 pin 35 to J7 pin 17; effectively connecting the single-ended input to the DAC output. In the case of single-ended inputs (jumper J8 not installed), connect J7 pin 36 to J7 pin 17 and pin 35 to J7 pin 37; effectively connecting the differential input to the DAC output. We are using the DAC output as it gives us the best analog signal to calibrate against (while in the field). You can alternatively, connect a known reference voltage that you have in your laboratory.
   b. Adjust the DAC output for ADC full-scale input minus two LSB. Measure this value using a calibrated voltmeter to the accuracy you require.
   c. While performing ADC conversions and reading the ADC value, adjust potentiometer ADC_GAIN (R5) until full scale minus two LSB is achieved. This can be achieved through averaging ADC values. Please reference ADC Conversion Register for software example on how to average data.
8. Remove all jumpers from the I/O connector at J7.
9. Repeat steps 5 through 8 until the ADC input is calibration to within the accuracy you need or +/- 2 LSB.

Chapter 5

# Connector Summary

## I/O Connector J7

| | | | |
|---|---|---|---|
| +5V POWER | 1 | 2 | CT_OUT2 |
| CT_OUT0 | 3 | 4 | CT_CLK0 |
| DOUT3 | 5 | 6 | DOUT2 |
| DOUT1 | 7 | 8 | DOUT0 |
| DIN3 | 9 | 10 | CT_GATE0 / DIN2 |
| DIN1 | 11 | 12 | TRIG / DIN0 |
| DGND | 13 | 14 | SSH |
| -5V | 15 | 16 | DAC_OUT_2 |
| DAC_OUT_1 | 17 | 18 | AGND |
| No Connection (NC) | 19 | 20 | AGND |
| CH7 LOW / CH15 | 21 | 22 | CH7 / CH7 HIGH |
| CH6 LOW / CH14 | 23 | 24 | CH6 / CH6 HIGH |
| CH5 LOW / CH13 | 25 | 26 | CH5 / CH5 HIGH |
| CH4 LOW / CH12 | 27 | 28 | CH4 / CH4 HIGH |
| CH3 LOW / CH11 | 29 | 30 | CH3 / CH3 HIGH |
| CH2 LOW / CH10 | 31 | 32 | CH2 / CH2 HIGH |
| CH1 LOW / CH9 | 33 | 34 | CH1 / CH1 HIGH |
| CH0 LOW / CH8 | 35 | 36 | CH0 / CH0 HIGH |
| AGND | 37 | 38 | No Connection (NC) |
| No Connection (NC) | 39 | 40 | No Connection (NC) |

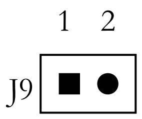
J7

| Name | Direction |
|---|---|
| CT_OUT0 | Output |
| CT_CLK0 | Input |
| CT_OUT2 | Output |
| TRIG/DIN0 | Input |
| DIN1 | Input |
| CT_GATE0/DIN2 | Input |
| DIN3 | Input |
| DOUT0 | Output |
| DOUT1 | Output |
| DOUT2 | Output |
| | |

| Name | Direction |
|---|---|
| DOUT3 | Output |
| DAC_OUT_1 | Output |
| DAC_OUT_2 | Output |
| CHn LOW /CHn | Input |
| CHn HIGH /CHn | Input |
| +5V POWER | Output |
| -5V | Output |
| CHn | Input |
| DGND | - |
| AGND | - |
| SSH* | Output |

* See Channel Register Description for more details

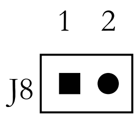
## Unipolar or Bipolar Analog Input J9



| Jumper | Function |
|---|---|
| 0* | Bipolar Analog Inputs ( +/- values accepted ) |
| 1 | Unipolar Analog Inputs ( + values only ) |

* Factory Default
Note: 1 = Jumper installed, 0 = Jumper not installed.

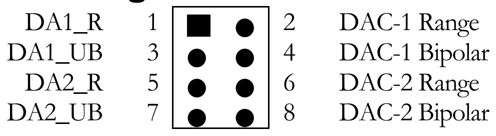
## Differential or Single Ended Analog Input J8



| Jumper | Function |
|---|---|
| 0* | Differential Inputs, 8-channels |
| 1 | Single-Ended Inputs, 16-channels |

* Factory Default
Note: 1 = Jumper installed, 0 = Jumper not installed.

## DAC Range Settings J5



J5

| DA1_UB | DA1_R | DAC-1 Range |
|---|---|---|
| 0 | 0 | 0 to +5 Volts * |
| 0 | 1 | 0 to +10 Volts |
| 1 | 0 | -5 to +5 Volts |
| 1 | 1 | -10 to +10 Volts |

* Factory Default
Note: 1 = Jumper installed, 0 = Jumper not installed.

| DA2_UB | DA2_R | DAC-2 Range |
|---|---|---|
| 0 | 0 | 0 to +5 Volts * |
| 0 | 1 | 0 to +10 Volts |
| 1 | 0 | -5 to +5 Volts |
| 1 | 1 | -10 to +10 Volts |

* Factory Default
Note: 1 = Jumper installed, 0 = Jumper not installed.

Chapter
6

# Specification

**Analog Inputs**

| | |
|---|---|
| ADC Resolution | 16-bits (1/65536 of full scale). |
| No missing codes guaranteed | |
| Number of Channels | 8 differential or 16 single-ended |
| Maximum Sampling Rate | 200,000 Samples Per Second (200 KSPS) |
| Input Type | True differential or single-ended. |
| Input Polarity | Unipolar/Bipolar jumper selectable. |
| Input Ranges | Bipolar: ±10V, ±5V, ±2.5V, ±1.25V<br>Unipolar: 0 to 10V, 0 to 5V, 0 to 2.5V 0 to 1.25V |
| Input Sensitivity | 19uV |
| Input Impedance | Differential: 20MΩ min. resistance in parallel with 47pF<br>Single-Ended: 20MΩ min. resistance in parallel with 27pF |
| Noise Characteristics | Gaussian behavior with maximum peak-to-peak internal noise of less than 1.5-LSB RMS over all input ranges and operating temperatures (1.2-LSB RMS typical). Jumper selectable 16-bit moving average filter drops noise to less than 1-LSB RMS over all input ranges and operating temperatures (0.6-LSB RMS typical). |
| DC Drift or Zero Drift | ±2ppm/°C |
| Input Bias Current | 50nA maximum |
| Absolute Maximum Input Voltage | ±35V |
| Common Mode Voltage Range | ±10V |
| Common Mode Rejection Ratio | 70dB at 60Hz |
| Integral Linearity Error | ±1.5 LSB (±3 LSB on 1.25V range) |
| Differential Linearity | ±1 LSB |
| Accuracy | 0.003% of reading ( or ±1 LSB) |
| Gain Drift | ±7ppm/°C |
| Trigger Sources | Programmable internal 32-bit timer, external source (DIN0/TRIG) or software polled. |
| Trigger Modes | Gated pacer, software polled. (Gate must be disabled by software after trigger event) |
| Data Transfer | From 1Mega-sample FIFO via interrupt, DMA or software |

**Analog Outputs**

| | |
|---|---|
| Resolution | 16-bits |
| Number of Channels | 2 DACs |
| Output Voltage Ranges | ±10V, ±5V, 0-10V, 0-5V. Maximum current to 5 milliamps per channel. Each channel independently configurable by jumpers. |
| Output Coupling | DC |
| Output Impedance | less than one ohm (0.7Ω typical), from 0 to ±5mA |

| | | |
|---|---|---|
| | Output short-circuit | ±35mA continuous |
| | Offset Error | less than 8 LSB |
| | Gain Error | Adjustable to 0 LSB by potentiometer |
| | Differential non-linearity | ±1 LSB maximum |
| | Settle Time | 10 microseconds |
| | Integral non-linearity | ±1 LSB maximum |
| | Monotonicity Guaranteed | |
| **Digital I/O** | Number of Inputs | 4 TTL compatible |
| | Input Voltage | Logic 0: 0.0V min, 0.8V max; Logic 1: 2.0V min, 5.5V max |
| | Input Current | ± 1uA max |
| | Number of Outputs | 4 TTL compatible |
| | Output Voltage | Logic 0: 0.0V min, 0.4V max; Logic 1: 2.4V min, 3.3V max |
| | Output Current | ±12mA per line max |
| **Counter/Timers** | A/D Pacer Timer | 32-bit down counter (2 82C54 counters cascaded) |
| | Clock Source | Jumper selectable 1 MHz or 10 MHz on-board clock source |
| | General Purpose | 16-bit down counter (1 82C54 counter) |
| | Interrupt/DMA Trigger | End of analog input conversion |
| **General** | Operating temperature range | -40 to 85°C |
| | Storage temperature range | -55 to 125°C |
| | Factory Calibration | Full NIST Traceable |
| | Humidity | 0 to 95% non-condensing |
| | Power Supply | +5 VDC ± 5% at less than 350mA |
| | Interface | PC/104 8- or 16-bit |

Chapter

7

# Application Notes

## Application Note S01: Continuous High Speed Sampling

An application requires 16-Channel Single-Ended inputs with bipolar +/-10 Volt input voltage range. Requirement is for 100KSPS combined on all sixteen channels, or 6.25KSPS per channel. The data will be queued up within the FIFO, and while data is being sampled continuously, the CPU will be reading ADC data in bursts of 65536 samples every 500mS on average. The starting and stopping of ADC sampling is to be under software control. The STX104 is to reside at base address 0x300.

**General Operation**

The GCTRL bit (Pacer Clock Control Register) will be used to start and stop the ADC sampling process. In effect, the GCTRL bit becomes the *gating* mechanism to enable or inhibit ADC sampling (triggering). Counter 1 and 2 will be set up as rate generators with a time period of 10uS. The trigger source will be set to internal trigger using the counter/timer (TS[1:0]="11"). Counter 0 output will be tied to DIN0 to provide a software time out mechanism by polling the INT bit in the status register; when the INT bit is set, 500mS has elapsed and it is time to read-out data.

**Hardware Configuration**

Hardware jumpers and physical wiring:

1. Set Base Address (0x300 is factory default)
2. Clock source set to 10MHz (no jumper installed at 1MHz)
3. DAS16jr/16 compatibility (no jumper installed at M0)
4. Allow FIFO superset (jumper installed at M1)
5. Bipolar analog inputs (no jumper installed at J9)
6. Single-Ended analog inputs (jumper installed a J8)
7. Analog inputs properly wired to the I/O connector J7.
8. CT_OUT0 is connected to DIN0

**Software Configuration**

The following pseudo code illustrates register configurations

1. Control Register set to 0x00
2. Write any value to Clear Interrupt Register to clear any pending interrupts

3. Pacer Clock Control Register set to 0x00
4. Analog Configuration Register set to 0x00
5. Setup counter/timer 1 to provide a 1uS time-out period by writing 0x74 to the counter/timer configuration register. Next, write 0x0A and 0x00 to counter/timer 1 data register.
6. Setup counter/timer 2 to provide a 10uS time-out period by writing 0xB4 to the counter/timer configuration register. Next, write 0x0A and 0x00 to counter/timer 2 data register.
7. Setup counter/timer 0 to provide a 500mS time-out period by writing 0x34 to the counter/timer configuration register. Next, write 0x50 and 0xC3 to counter/timer 0 data register.
8. Reset the FIFO and set the channel register. Write 0xF0 to the channel register. Wait for CNV bit to be cleared.
9. Reset the INT bit by  writing to the Clear Interrupt Register.
10. Control Register set to 0x03.
11. Wait for INT bit to be set. When set, clear it by writing to the status register. Next, set the Pacer Clock Control Register to 0x01 to begin sampling.
12. Wait for INT bit to be set. When set, clear it by writing to the status register. Next, perform ADC reads until the FIFO is empty (FE='1') or 65536 samples have been read-out.
13. Go to step #10.

```
/**************************************************************************/
void main()
{       /* base address set to factory default at 0x300 */
        /* initialize STX104 */
        outportb(0x309, 0x00); /* no interrupts, no DMA and s/w trigger */
        outportb(0x308, 0x00); /* clear any pending interrupts          */
        outportb(0x30A, 0x00); /* set pacer clock */
        outportb(0x30B, 0x00); /* Gain set to 10V */
        outportb(0x30F, 0x74); /* Set up Counter/Timer #1              */
        outportb(0x30D, 0x0A); /*  for 1uS output interval.            */
        outportb(0x30D, 0x00); /*  Uses a 10MHz clock.                 */
        outportb(0x30F, 0xB4); /* Setup Counter/Timer #2               */
        outportb(0x30E, 0x0A); /*  for 10uS output interval.           */
        outportb(0x30E, 0x00); /*  Uses a 1MHz clock pulse from CT1    */
        outportb(0x30F, 0x34); /* Set up Counter/Timer #0              */
        outportb(0x30C, 0x50); /*  for 500mS time out period.          */
        outportb(0x30C, 0xC3); /*  Uses a 10MHz clock.                 */
        outportb(0x308, 0x00); /* clear any pending interrupts          */
        outportb(0x309, 0x03); /* trigger adc using counter/timer       */
        /* sync up with our 500mS timer */
        while ( (inportb(0x308) & 0x10) == 0x00 ); /* wait for INT */
        outportb(0x308, 0x00); /* clear any pending interrupts    */
        outportb(0x30A, 0x01); /* set pacer clock */
        while (1)
        {       if ( (inportb(0x308) & 0x10) == 0x10 ) /* if 500mS has past  */
                {       outportb(0x308, 0x00); /* clear interrupt  */
                        Read_STX104_FIFO();     /* get the data  */
                }
                else
                {       /* do other things here */
                }
        }
```

## Application Note S02: Minimizing ISR Timing When Using REP INSW instruction and Illustrates 200,000 Samples per Second capability.

Using the REP INSW is an effective way to increase system throughput and have better control over system timing. When you get a FIFO interrupt from the STX104, you will want to keep your interrupt service routine (ISR) as fast as possible. Let your main-line code take care of actually reading out the FIFO. In this example the sample rate is set to 200,000 samples per second, so the FIFO interrupts will occur at 2.56mS intervals (or 390 interrupts per second). Simplified example code is shown below.

```
/***************************************************************************
   Apex Embedded Systems
   September 29, 2002
   STX104 REP INSW Example
***************************************************************************/

#include <conio.h>
#include <stdio.h>
#include <dos.h>

/***************************************************************************/
/* These are the port addresses of the 8259 Programmable
   Interrupt Controller (PIC).
*/
#define IMR             0x21   /* Interrupt Mask Register port */
#define ICR             0x20   /* Interrupt Control Port       */
/* An end of interrupt needs to be sent to the Control Port of
        the 8259 when a hardware interrupt ends. */
#define NSEOI           0x20   /* End Of Interrupt */

/***************************************************************************/
void interrupt (*old_isr_handler)(...);
int isr_count;
int prev_isr_count;
int mask;
int first_channel;
int last_channel;
int interrupt_number;
int base_address;
unsigned int ff_status;
int fbr, prev_fbr;
unsigned char ff_ff, ff_fe, ff_int;

/***************************************************************************/
void insw(WORD port, void *buf, int count)
 {
   _ES = FP_SEG(buf);   /* Segment of buf */
   _DI = FP_OFF(buf);   /* Offset of buf  */
   _CX = count;         /* Number to read */
   _DX = port;          /* Port           */
   asm   REP INSW;
 }

/***************************************************************************/
void interrupt STX104_Isr( ... )
{
        disable();
        isr_count++;  /* acquired at least two 256-sample FIFO blocks */
        /* clear interrupt */
        outportb(base_address+8, 0x00);
        outportb(ICR, NSEOI);
        enable();
}


/***************************************************************************/
void Install_Isr( int int_number )
{
  unsigned char mask;
  old_isr_handler = getvect( int_number+8 );
  setvect( int_number+8, Tracer_Isr );
  mask = 0x01 << int_number;
  outportb(IMR, inportb(IMR) & ~mask);  /* turn on PIC */
}

/***************************************************************************/
void Restore_Isr( int int_number )
{
   unsigned char mask;
   mask = 0x01 << int_number;
   outportb(IMR, inportb(IMR) | mask);  /* turn off PIC */
        setvect( int_number+8, old_isr_handler );
}

/***************************************************************************/
void Get_Fifo_Status( void )
{
   ff_status = ( ((unsigned int)inportb(base_address+10)) << 8 ) |
                 ((unsigned int)inportb(base_address+15));
   fbr = ff_status & 0x0fff;
   if ( (ff_status & 0x1000)!=0x0000 ) ff_fe = 1;
   else ff_fe = 0;
   if ( (ff_status & 0x2000)!=0x0000 ) ff_ff = 1;
   else ff_ff = 0;
   if ( (ff_status & 0x8000)!=0x0000 ) ff_int = 1;
   else ff_int = 0;
}

/***************************************************************************/
void Initialize( void )
{
        /* initialize variables */
           prev_isr_count = 0;  /* not currently used */
           base_address = 0x300;
           last_channel  = 15;  first_channel = 0;  interrupt_number = 3;
        /* Initialize STX104 */
           outportb(base_address+9,  0x00);
           outportb(base_address+3,  0x00);
           outportb(base_address+8,  0x00);
           outportb(base_address+10, 0x00);
           outportb(base_address+11, 0x00);
           outportb(base_address+2,  0x00);
           outportb(base_address+1030,0x40); /* function enable */
           outportb(base_address+1028,0x40); /* disable triggers */
           Install_Isr( interrupt_number );
        /* set counter/timer 1 & 2 to generate trigger every 5uS */
           outportb(0x30F,0x74); /* 10MHz clock source assummed */
           outportb(0x30D,0x0A);  /* 1uS output */
           outportb(0x30D,0x00);
           outportb(0x30F,0xB4);
           outportb(0x30E,0x05); /* 5*1uS output */
           outportb(0x30E,0x00);
           outportb(base_address+11,0x00); /* gain of 10V */
        /* channels = # + 1 */
           mask = (last_channel & 0x0F) - (first_channel & 0x0F);
           mask = mask << 4; /* burst amount */
           mask = mask | 0x00;
           outportb(base_address+10, mask);
           mask = ((last_channel & 0x0F)<<4) | (first_channel & 0x0F);
           outportb(base_address+2, mask); /* reset acquisition and fifo */
           isr_count = 0;  /* count indicates number of blocks gathered */
           /* the following CNV test has been added to test for STX104
              internal reset as a result of writing to Channel Register */
           /* wait for CNV to be false to make sure reset complete */
           while ( (inportb(base_address+8) & 0x80) == 0x80 ); /* wait */
           mask = 0x00 | ((interrupt_number & 0x07)<<4) | 0x08 | 0x03;
           outportb(base_address+9, mask);
           outportb(base_address+1029,0x40); /* enable bursting */
           outportb(base_address+1028,0x00); /* enable triggers */

}

/*****************************************************************************/
void Terminate()
{
        Restore_Isr( interrupt_number );
}

/*****************************************************************************/
void main(void)
{
        unsigned int data_buffer[32000];
        int i, wait;

           Initialize();

        /* wait for at least one block */
           wait = 0xff
           while ( wait == 0xff )
           {
                if (isr_count > 0 ) wait = 0x00;
           }

        /* Interrupts Off */
           Get_Fifo_Status();
           Outportb( base_address+9, ( inportb(base_address+9) & 0x70 ) );

        /* Read in n-blocks */
           Get_Fifo_Status();
           samples = fbr*256; /* read in n-blocks of data into the buffer */
           insw(base_address, &data_buffer, samples);

        /* Read in the remaining samples  */
           Get_Fifo_Status();
           i = samples;
           while ( ff_fe == 0 )
           {
              data_buffer[i] = inpw( base_address );
              i++;
              Get_Fifo_Status();
           }

           Terminate();
}
```

## Application Note S03:  Single Interrupt per Burst Operation

This application note illustrates how to implement burst sampling with one interrupt per burst operation when in DAS1602 compatibility mode and M1 jumper is not installed (FIFO jumper not installed).  In the software example shown below the setup is for ten burst operations per second with only one interrupt per burst operation.

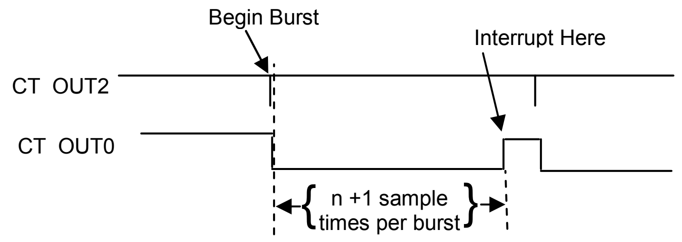
In order to maintain strict DAS1602 compatibility (M1 jumper not installed), the internal logic is designed such that during a burst operation one interrupt per sample is generated, so to obtain only one interrupt per burst operation, requires a bit different approach.

You will need to provide the following wiring at the I/O connector to generate only one interrupt per burst operation.  We need to run wires externally to connect the counter/timers and DIN0 appropriately as shown below:

CT_OUT2 (J7.2) → CT_GATE0 (J7.10)

CT_OUT0 (J7.3) → DIN0 (J7.12)

The timing diagram below illustrates what we are attempting to do.



Basically, CT_OUT2 generates a trigger to begin each burst operation.  CT_OUT0 will generate an interrupt at the end of the burst operation by driving DIN0.  In our example counter/timer 1&2 are configured as a rate generator to generate pulses ten times per second.  Since we want to sample 16-channels during one burst operation, we set the time out duration to 17-sample times (or about 170uS).  The added 10uS time interval provides additional guard time to ensure the burst has completed.

Counter/Timer 0 must be set up as hardware retriggerable one-shot (mode 1). Counter/Timer 1&2 must be set up as rate generators (mode 2) and run freely with no gating (GCTRL=1).  Counter/timer 0 will use the 100KHz on board oscillator (CT_SRC0=1).  The trigger source will be the counter/timers (TS1=1 and TS0=1). The interrupts will be enabled such that a rising edge on DIN0 will generate an interrupt (AIE=1, FIE=1 and DMA=0).

Example code is shown below.  It is assumed that a jumper is installed at postion M0 in order to enable DAS1602 compatibility, and jumper M1 installed to provide DIN0 interrupt generation.

```
/*****************************************************************************
   Apex Embedded Systems
   January 14, 2003
   STX104 Low Duty Cycle Sample Bursting


   Performs one burst ten times per second.  One interrupt is
   generated for each burst.
*****************************************************************************/
#include <conio.h>
#include <stdio.h>
#include <dos.h>
/*****************************************************************************/
/* These are the port addresses of the 8259 Programmable
   Interrupt Controller (PIC).
*/
#define IMR             0x21   /* Interrupt Mask Register port */
#define ICR             0x20   /* Interrupt Control Port       */
/* An end of interrupt needs to be sent to the Control Port of
        the 8259 when a hardware interrupt ends. */
#define NSEOI           0x20   /* End Of Interrupt */
/*****************************************************************************/
void interrupt (*old_isr_handler)(...);
int isr_count;
int prev_isr_count;
int mask;
int first_channel;
int last_channel;
int burst_count;
unsigned int data_buffer[16];
int interrupt_number;
int base_address;
unsigned int ff_status;
int fbr, prev_fbr;
unsigned char ff_ff, ff_fe, ff_int;
/*****************************************************************************/
void insw(WORD port, void *buf, int count)
{
   _ES = FP_SEG(buf);   /* Segment of buf */
   _DI = FP_OFF(buf);   /* Offset of buf  */
   _CX = count;         /* Number to read */
   _DX = port;          /* Port           */
   asm   REP INSW;
}
/*****************************************************************************/
void interrupt STX104_Isr( ... )
{
        disable();
        isr_count++;  /* useful for debugging */
        /* read samples to buffer */
        insw(base_address+0, &data_buffer, burst_count );
        /* clear interrupt */
        outportb(base_address+8, 0x00);
        outportb(ICR, NSEOI);
        enable();
}
/*****************************************************************************/
void Install_Isr( int int_number )
{
  unsigned char mask;
  old_isr_handler = getvect( int_number+8 );
  setvect( int_number+8, Tracer_Isr );
  mask = 0x01 << int_number;
  outportb(IMR, inportb(IMR) & ~mask);  /* turn on PIC */
}
/*****************************************************************************/
void Restore_Isr( int int_number )
{
   unsigned char mask;
   mask = 0x01 << int_number;
   outportb(IMR, inportb(IMR) | mask);  /* turn off PIC */
        setvect( int_number+8, old_isr_handler );
}
/*****************************************************************************/
void Initialize( void )
{
        /* initialize variables */
           prev_isr_count = 0;  /* not currently used */
           base_address = 0x300;
           last_channel  = 15;  first_channel = 0;  interrupt_number = 3;
           burst_count = last_channel - first_channel + 1;
        /* Initialize STX104 */
           outportb(base_address+3,  0x00);
           outportb(base_address+8,  0x00);
           outportb(base_address+9,  0x00);
           outportb(base_address+10, 0x00);
           outportb(base_address+11, 0x00);
           outportb(base_address+2,  0x00);
           outportb(base_address+1030,0x40); /* function enable */
           outportb(base_address+1028,0x40); /* disable triggers */
           Install_Isr( interrupt_number );
        /* set counter/timer 1 & 2 with pulse 10 times per second */
           outportb(0x30F,0x74); /* 10MHz clock source assummed */
           outportb(0x30D,0xE8);
           outportb(0x30D,0x03);
           outportb(0x30F,0xB4);
           outportb(0x30E,0xE8);
           outportb(0x30E,0x03);
        /* set counter/timer 0 to be hardware triggerable, 17uS one-shot */
           outportb(0x30F,0x32);
           outportb(0x30C,0x11);
           outportb(0x30C,0x00);
        /* gain of 10V */
           outportb(base_address+11,0x00);
        /* set burst counter, CT_SRC0=1, and GCTRL=0 */
           mask = (last_channel & 0x0F) - (first_channel & 0x0F);
           mask = mask << 4; /* burst amount */
           mask = mask | 0x02;
           outportb(base_address+10, mask);
        /* set channel register and reset acquisition controller and FIFO */
           mask = ((last_channel & 0x0F)<<4) | (first_channel & 0x0F);
           outportb(base_address+2, mask); /* reset acquisition and fifo */
           isr_count = 0;  /* count indicates number of blocks gathered */
           /* the following CNV test has been added to test for STX104
              internal reset as a result of writing to Channel Register */
           /* wait for CNV to be false to make sure reset complete */
           while ( (inportb(base_address+8) & 0x80) == 0x80 ); /* wait */
        /* set ADC control register */
           mask = 0x80 | ((interrupt_number & 0x07)<<4) | 0x08 | 0x03;
           outportb(base_address+9, mask);
           outportb(base_address+1029,0x40); /* enable bursting */
           outportb(base_address+1028,0x00); /* enable triggers */
}
/*****************************************************************************/
void Terminate()
{
        Restore_Isr( interrupt_number );
}
```

```
/**************************************************************************/
void main(void)
{
        unsigned int data_buffer[32000];
        int i, wait;

           Initialize();

        /*
                main loop - doing something useful here
        */

           Terminate();
}
```

## Application Note S04: Performing Raster or Line Scans.

This application note outlines two simple methods of implementing line or raster scan functionality using the STX104 board. In either case, one could use the DAC outputs to drive X and Y coordinates. This application note illustrates the use of the large FIFO and burst mode. Detailed FIFO readout is described in Application note S02.

Software triggered line scan sampling method:

1. STX104 hardware configuration: Install M0 and M1 configuration jumpers to enable DAS1602 bursting capability and extended FIFO functions, respectively.
2. Set up STX104 board to perform burst sampling and set for software triggering.
3. Initialize X and Y positions.
4. Begin line or raster scan by adjusting the X and Y positions as needed.
5. At a predetermined X-position issue a software trigger, this will start a sample burst (only one trigger per burst required when M1 jumper installed). Let the STX104 perform the burst sampling and deposit the samples into the FIFO.
6. Repeat step #5 until line scan is complete.
7. At the end of the line scan, read the entire FIFO contents from the STX104 to software. See Application note S02 for details.
8. Go to step #4.

Timer/Counter triggered line scan sampling method:

1. STX104 hardware configuration: Install M0 and M1 configuration jumpers to enable DAS1602 bursting and extended FIFO functions, respectively. Tie DIN0 to +5V.
2. Set up STX104 board to perform burst sampling using hardware triggering and keep gating to counter/timer off (GCTRL=1).
3. Initialize X and Y positions.
4. Begin line or raster scan by adjusting the X and Y positions as needed. In addition, load counter/timer 1 & 2 with appropriate sample timing (this is to keep the sample timing the same from line to line). Now set GCTRL=0 to begin sampling at the chosen trigger time intervals.
5. Adjust X position (and possibly Y position) as required while the STX104 performs the sample bursting and deposits the samples into the FIFO.
6. Repeat step #5 until line scan is complete.
7. At the end of the line scan, set GCTRL=1 and read the entire FIFO contents from the STX104 to software for further analysis.
8. Go to step #4.

Apex Embedded Systems
*PC-Based Solutions for Industrial and Portable Applications*

## Limited Warranty

Unless altered by written agreement, APEX EMBEDDED SYSTEMS (APEX) warrants to the original purchaser for a period of one year from the date of original purchase, that the products shall be free from defects in material and workmanship. APEX's obligation under this warranty is limited to replacing or repairing, at its option and its designated site, any products (except consumables) within the warranty period that are returned to APEX in the original shipping container(s) with an APEX RMA number referenced on the shipping documents.

This warranty will not apply to products that have been misused, abused, or altered. This warranty will not apply to prototypes of any kind, engineering services, software or products under pre-release status. Any returns must be supported by a Return Material Authorization (RMA) number issued by APEX. APEX reserves the right to refuse delivery of any shipment containing any shipping carton which does not have an RMA number displayed on the outside. Purchaser shall prepay transportation to APEX's location. If returned parts or products are repaired under the terms of this warranty, APEX will pay return transportation charges. Allow six (6) to eight (8) weeks for warranty repairs.

THE FOREGOING WARRANTY IS IN LIEU OF ALL WARRANTIES, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR THE PARTICULAR PURPOSE, AND OF ANY OTHER OBLIGATION ON THE PART OF APEX.

Warranty return address:

Apex Embedded Systems
Attn: Customer Service.    RMA#_________
116 Owen Road
Monona, WI 53716