

User Manual
for
CS210-12 (/14)

10 MSPS, 12-bit, PC/104, (ISA) Analog Input Module

CHASE SCIENTIFIC COMPANY

P.O. Box 1487
Langley, WA 98260

Tel: 360-221-8455
Fax: 360-221-8457

Email: techsupport@chase2000.com
Web: <http://www.chase2000.com>

CS210_manual_2030629.odt

© Copyright 1998-2003 by
Chase Scientific Company

This manual, the CS210 module, and the software drivers outlined in this document are copyrighted with all rights reserved. Under the copyright laws, the above mentioned may not be copied, in whole or in part, without the express written consent of Chase Scientific Company.

TABLE OF CONTENTS

1 GENERAL INFORMATION.....	4
1.1 INTRODUCTION.....	4
1.2 REFERENCES.....	4
1.3 DELIVERABLES.....	4
1.3.1 Software.....	4
1.3.2 Hardware.....	5
1.3.3 Checklist.....	5
1.4 PRODUCT SPECIFICATION.....	6
1.5 TECHNICAL SUPPORT / SOFTWARE UPDATES.....	7
1.6 WARRANTY.....	7
2 HARDWARE DESCRIPTION.....	9
2.1 INTRODUCTION.....	9
2.2 BLOCK DIAGRAM.....	9
2.3 BOARD DRAWING.....	10
2.4 BOARD I/O.....	11
2.4.1 Header Pinouts and Pin Descriptions.....	11
2.4.1.1 40-Pin Analog I/O.....	11
2.4.1.2 34-Pin Digital I/O.....	12
2.4.1.3 40-Pin Memory Expansion.....	12
2.4.2 Board Configuration.....	13
2.4.2.1 Base Address.....	13
2.4.2.2 IRQ Selection.....	14
2.4.3 I/O Map.....	14
2.5 REGISTER DEFINITIONS.....	15
2.6 ANALOG INPUT RANGES AND RESOLUTION.....	15
2.7 ANALOG OUTPUT RANGE AND RESOLUTION.....	16
2.8 DIGITAL I/O DESCRIPTION.....	16
3 THEORY OF OPERATION.....	17
3.1 INTRODUCTION.....	17
3.2 ACQUIRING ANALOG INPUT DATA.....	17
3.3 AUXILIARY INPUTS.....	17
4 SOFTWARE DRIVERS.....	17
4.1 INTRODUCTION.....	17
4.2 DRIVER INSTALLATION.....	18
4.2.1 DOS / Windows 3.1.....	18
4.2.2 Windows 95.....	18
4.2.3 Windows NT.....	18
4.3 FUNCTION CALLS.....	18
4.3.1 Function Declarations in Pascal.....	18
4.3.2 Function Declarations in C.....	19
4.3.3 Function Call Descriptions / Usage.....	19
4.3.3.1 Initialize ().....	19
4.3.3.2 Set_Frequency ().....	20
4.3.3.3 Select_Channel ().....	20
4.3.3.4 Select_Gain ().....	21
4.3.3.5 Set_Trigger ().....	21
4.3.3.6 Set_Memory_Size ().....	22
4.3.3.7 Set_Pretrigger ().....	22
4.3.3.8 Digitize ().....	23

4.3.3.9 Transfer_Data () [DOS Only].....	23
4.3.3.10 Read_Long ().....	24
4.3.3.11 Write_Port ().....	24
4.3.3.12 Read_Port ().....	25
4.3.3.13 Set_DAC ().....	25
4.3.3.14 Trigger_Done ().....	26
4.3.3.15 Board_Detected ().....	26
4.3.3.16 HW_IRQ ().....	27
4.3.3.17 Open_Driver () [Win95/NT Only].....	27
4.3.3.18 Close_Driver () [Win95/NT Only].....	27
4.3.3.19 Set_IRQ () [Win95/NT Only] { ADVANCE INFORMATION }.....	28
4.3.3.20 Is_IRQ_Set () [Win95/NT Only] { ADVANCE INFORMATION }.....	28
4.3.3.21 Destroy_IRQ () [Win95/NT Only] { ADVANCE INFORMATION }.....	29
4.3.3.22 Simulate_HW_Int () [Win95/NT Only] { ADVANCE INFORMATION }.....	29
4.4 PROGRAMMING EXAMPLES.....	29
4.4.1 DOS / Windows 3.1 Example Using Borland C++ 5.02.....	29
4.4.2 Windows 95/NT DLL Example using Borland C++ 5.02.....	32
4.5 ERROR RETURN CODE LIST.....	34
5 MISCELLANEOUS.....	35
5.1 CALIBRATION.....	35
5.2 MAINTENANCE.....	35

ILLUSTRATIONS / TABLES

FIGURE 1 – BLOCK DIAGRAM.....	9
FIGURE 2 – BOARD LAYOUT.....	10
FIGURE 3 - 40-PIN ANALOG I/O CONNECTOR.....	11
FIGURE 4 - 34-PIN DIGITAL I/O CONNECTOR.....	12
FIGURE 5 - 40-PIN MEMORY EXPANSION CONNECTOR.....	13
FIGURE 6 - PORT I/O.....	15

"CS210_man_V97_v1p5a.doc" was created on 29/06/2003 21:07:00 and last modified on 2/14/2010
02:52:49 PM

1 GENERAL INFORMATION

1.1 Introduction

The CS210-12 is a PC/104 (ISA) analog I/O module which features two (2) 12-bit, high speed differential/synchronous input channels, each capable of capturing data at 10 MSPS with a minimum bandwidth of 5 MHz.

The CS210 module comes configured with additional features including:

- (4) 10 MSPS multiplexed/differential analog input channels.
- (32) channels of CMOS/TTL compatible digital I/O (16 input/16 output)
- (4) unipolar voltage outputs with 12-bit resolution
- (1) external clock input
- (1) external trigger input

An extended temperature version is available to support -40°C to 85°C operation.

All analog I/O is available on a 40-pin header while all digital I/O is available on a separate 34-pin header. Each high speed channel has 1KB, 8KB or 32 KB per channel of software selectable onboard memory. Post/pre-triggering is selectable in 1 sample increments. Sources for the trigger include either one of the (2) high speed analog inputs using 1 of 256 levels of software selectable trigger thresholds, a software generated trigger, or an external TTL signal.

Timing is controlled by an onboard clock circuit referenced to a 10 MHz crystal oscillator and is divided down by a programmable 16-bit divider circuit. A software command or external TTL trigger can also be used to initiate the A/D conversion sequence. Once the A/D conversions are completed, data can be transferred from the module to main memory by direct I/O port reads or through an interrupt service routine. The CS210 stores up to 32 K samples for each channel.

1.2 References

“P996.1 Standard for Compact Embedded-PC Modules”; PC/104 Specification, Version 2.3, June 1996 by PC/104 Consortium.

“IEEE P996 draft standard”, 1987 by IEEE Standards Committee. This bus standard was the basis for the first PC/104 standard released in 1992.

1.3 Deliverables

1.3.1 Software

The CS210 comes with drivers for **MSDOS, Windows 3.xx, and Windows 95/NT**. Software comes on a single 3.5” diskette. Call Chase Scientific for the latest information on drivers for other operating system platforms.

MSDOS / Windows 3.xx software drivers include both Borland Turbo Pascal V6.0 and Borland Turbo C V4.02 source code.

Windows 95/NT software drivers are provided as a Dynamic Link Library (*.DLL) which is compatible with most 32-bit windows based development software including Microsoft C/C++, Borland C/C++, Borland Delphi, and Visual Basic. This DLL uses the “**cdecl**” calling convention which is default capatible with the compilers above and provides easy to use function calls to the system drivers “Vichw11.vxd” for Windows 95 and “Vichw11.sys” for Windows NT.

Actual Listing of files on Diskette:

```

----- D I R E C T O R I E S / F I L E S ----- 2030629
BASE_DIR
|
|
|  readme.txt                      // This file.
|  cs210dbg.zip                    // Debug program for AD210/CS210
(Win31/95/98)
|  CS210_manual_2030629.doc        // User manual for AD210/CS210 in MSWord 97.
|
|
|  16-Bit Versions for DOS_Win31
|  |
|  |  Bc4_16_DOS                   // Borland C++ V4.02 for DOS
|  |  Bc5_16_DOS                   // Borland C++ V5.02 for DOS
|  |
|  |
|  32-Bit Versions for Win95_98_NT
|  |
|  |  Bc5_32_Console_95NT_DLL      // Includes DLL, *.lib, *.h, *.exe example
|  |  Borland C++ Builder 3.0      // Includes DLL, *.lib, *.h, *.exe example
|  |
|  |  cdecl calling convention
|  |  |  Borland Lib File
|  |  |  MSVisual Lib File
|  |
|  |  Win95-NT Device Drivers      // Kernel Drivers
|  |  |
|  |  |  Vichw11.vxd               // Windows 95/98/ME Driver - copy this virtual
driver
|  |  |                          // to c:\<windows95>\system\
|  |  |
|  |  |  Vichw11.sys               // Windows NT/2000/XP Driver - copy this virtual
driver
|  |  |                          // to c:\<winNT>\system32\drivers\
|  |  |
|  |  |  Tvichw32.ini              // run "regini.exe tvichw32.ini" to install
|  |  |  regini.exe                // proper registry keys for NT only.
|
|
----- E N D -----

```

1.3.2 Hardware

The CS210 hardware consists of a single PC/104 compliant module. The module is shipped with a manual with complete hardware and software descriptions.

1.3.3 Checklist

Item #	Qty	Part Number	Description
1	1	CS210-SB-DF or CS210-SU-DF	Bipolar inputs analog/digital I/O PC/104 module Unipolar inputs analog/digital I/O PC/104 module
2	1	CS210-Drivers	3.5" diskette with DOS Pascal/C source drivers plus Dynamic Link Libraries for Windows 3.1/95/NT. Includes examples.
3	1	CS210-Manual	User manual for CS210 board and software drivers.

1.4 Product Specification

(all specifications are at 25C unless otherwise specified)

HIGH SPEED ANALOG INPUTS (CH1-2)

Number of inputs	(2) Synchronous, Differential
A/D Resolution	12-bit (1 part in 4096)
Input Bandwidth	5 MHz minimum @ 5V pk to pk (Av=1)
Input Ranges (software programmable)	+/- 0.5V, +/- 1.0V, +/- 2.5V, +/- 5.0V standard. [Optional unipolar version => 0-0.5V, 0-1.0V, 0-2.5V, 0-5V; use part number CS210-12-U instead of CS210-12-B]
Acquisition Time to Full-Scale Step	500 ns typical to 0.01% (1/2 LSB)
Input Bias Current	110 pA typical
Input Coupling	DC
Overvoltage Protection	+/- 50V
Nonlinearity	+/- 1 LSB max
Conversion Rate / channel	10 MHz maximum
Memory	1,1K, 8K, 32K bytes software selectable
Acquisition Mode	Single-shot initiated by software, external or internal triggering (see trigger sources)
Timebase	10 MHz, 5 MHz, 2.5 MHz, 1.25 MHz, etc. down to 152Hz (user programmable 16-bit divider)
Trigger	
Location	Post / Pretriggering selectable in 1 sample increments
Sources	ch1, ch2, external TTL signal
Slope	Positive, Negative
Coupling	DC

AUXILLARY, LOW SPEED ANALOG INPUTS (CH1-4)

Number of inputs	(4) Multiplexed (sequential), Differential
A/D Resolution	12 bits (1 part in 4096)
Input Bandwidth (Full Scale)	500 KHz minimum
Input Ranges (software programmable)	+/- 0.5V, +/- 1.0V, +/- 2.5V, +/- 5.0V standard. [Optional unipolar version => 0-0.5V, 0-1.0V, 0-2.5V, 0-5V; use part number CS210-12-U instead of CS210-12-B]
Acquisition Time to Full-Scale Step	1.5 microseconds maximum
Input Bias Current	25 pA typical
Input Coupling	DC
Overvoltage Protection	-35V, +55V
Nonlinearity	+/- 1 LSB max
Conversion Rate / Channel	10 MHz maximum
Conversion Trigger	Software trigger, internal clock, or external TTL signal
Pre/Posttrigger Capability	Same as high speed inputs.

ANALOG OUTPUTS

Number of Outputs	4 voltage outputs
-------------------	-------------------

Resolution	12 bits (1 part in 4096)
Fixed Output Ranges	0-5V unipolar [Optional 0-10V unipolar]
Settling Time	12 μ s max to +/- 0.01%
Accuracy	+/- 2.0 LSB max
Integral Nonlinearity	+/- 2 LSB max
Differential Nonlinearity	+/- 2 LSB max, guaranteed monotonic
Output Current	+/- 5 mA max per channel
Minimum Load	2K ohm
Update Method	Update can be performed independently or simultaneously on all 4 channels

DIGITAL I/O

Number of Input Lines	16, CMOS / TTL compatible
Number of Output Lines	16, CMOS / TTL compatible
Input Voltage	
Logic 0	-0.5V min, 0.8V max
Logic 1	2.0V min, 5.5V max
Output Voltage	
Logic 0	0.0V min, 0.4V max
Logic 1	2.5V min, VCC max
Output Current	+/- 3mA max per line
External Trigger	TTL / CMOS compatible, active high edge
Reset	All digital output data lines are set to logic low

GENERAL

Power Supply (Vcc)	+5VDC +/- 10%
Current Requirement	1.0A typical [both channels on at 10Ms/sec]
Operating Temperature	0 to 70 degrees C standard -40 to +85 degrees C extended
Operating Humidity	5 to 95% non-condensing
Size	PC/104: 3.55" x 3.775" ISA: Short AT Style Card (6.5" L x 4.5" T)
Data Bus	16-bits

1.5 Technical Support / Software Updates

For technical support:

Phone	(360) 221-8455
Fax	(360) 221-8457
Email	techsupport@chase2000.com
Mail	Chase Scientific Company P.O. Box 1487 Langley, WA 98260

For software updates:

Email	techsupport@chase2000.com
Web	http://www.chase2000.com

1.6 Warranty

Chase Scientific Company (hereafter called Chase Scientific) warrants to the original purchaser that its CS210-Sx-DF-A-5V, and the component parts thereof, will be free from defects in workmanship and materials for a period of ONE YEAR from the date of purchase.

Chase Scientific will, without charge, repair or replace at its option, defective or component parts upon delivery to Chase Scientific's service department within the warranty period accompanied by proof of purchase date in the form of a sales receipt.

EXCLUSIONS: This warranty does not apply in the event of misuse or abuse of the product or as a result of unauthorized alterations or repairs. It is void if the serial number is altered, defaced or removed.

Chase Scientific shall not be liable for any consequential damages, including without limitation damages resulting from loss of use. Some states do not allow limitation or incidental or consequential damages, so the above limitation or exclusion may not apply to you.

This warranty gives you specific rights. You may also have other rights that vary from state to state.

Chase Scientific warrants products sold only in the USA and Canada. In countries other than the USA, each distributor warrants the Chase Scientific products that it sells.

NOTICE: Chase Scientific reserves the right to make changes and/or improvements in the product(s) described in this manual at any time without notice.

2 HARDWARE DESCRIPTION

2.1 Introduction

The CS210 hardware consists of the following major functions:

- (2) 12-bit, 10 MSPS synchronous, differential, bipolar/unipolar A/D Inputs
- (4) 12-bit, 10 MSPS sequential, differential, bipolar/unipolar A/D Inputs
- (4) 12-bit, single ended, unipolar, D/A outputs
- (16) TTL Outputs, (16) TTL Inputs

2.2 Block Diagram

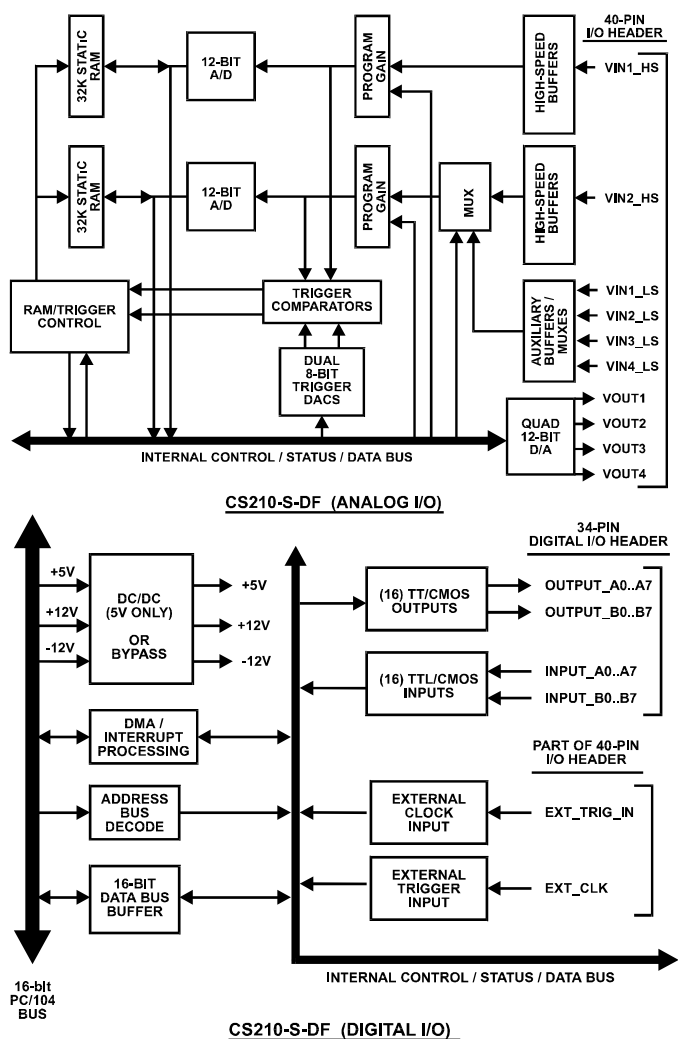


Figure 1 – Block Diagram

2.3 Board Drawing

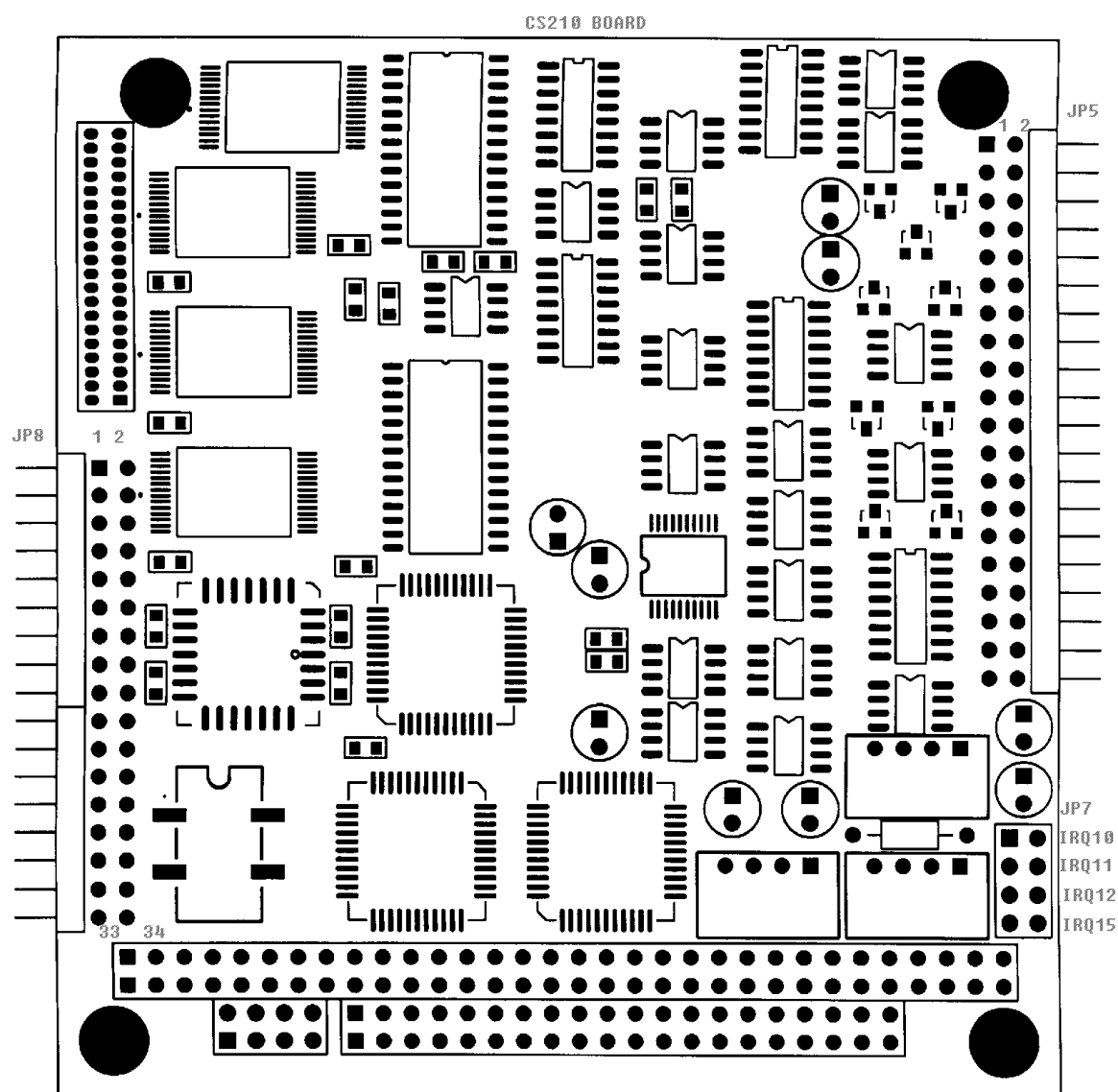


Figure 2 – Board Layout

2.4 Board I/O

2.4.1 Header Pinouts and Pin Descriptions

2.4.1.1 40-Pin Analog I/O

Connector JP5 (standard 100 mil dual right angle)

TOP VIEW			
AGND	1	2	AGND
AGND	3	4	AGND
n/c	5	6	n/c
+VIN1_HS	7	8	-VIN1_HS [SE=AGND]
n/c	9	10	n/c
+VIN2_HS	11	12	-VIN2_HS [SE=AGND]
n/c	13	14	n/c
n/c	15	16	n/c
AGND	17	18	AGND
AGND	19	20	AGND
n/c	21	22	n/c
+VIN1_LS	23	24	-VIN1_LS
+VIN2_LS	25	26	-VIN2_LS
+VIN3_LS	27	28	-VIN3_LS
+VIN4_LS	29	30	-VIN4_LS
n/c	31	32	n/c
DAC_OUT1	33	34	DAC_OUT2
DAC_OUT3	35	36	DAC_OUT4
AGND	37	38	AGND
EXT_CLK	39	40	EXT_TRIG_IN

Figure 3 - 40-Pin Analog I/O Connector

Signal Name	Definition
AGND	Analog Ground
+VIN1_HS, -VIN1_HS, +VIN2_HS, -VIN2_HS	(2) Channels of synchronous, differential, 10MHz, 12-bit, 5MHz bandwidth A/D inputs.
+VIN1_LS, -VIN1_LS, +VIN2_LS, -VIN2_LS, +VIN3_LS, -VIN3_LS, +VIN4_LS, -VIN4_LS	(4) Channels of sequential, differential, 10MHz, 12-bit, 500KHz bandwidth A/D inputs.
DAC_OUT1, DAC_OUT2, DAC_OUT3, DAC_OUT4	(4) Channels of single ended, 12-bit, D/A outputs.
EXT_CLK	External TTL clock input. Maximum frequency is 10MHz.
EXT_TRIG_IN	External TTL trigger input.

2.4.1.2 34-Pin Digital I/O

I/O CONNECTOR JP8/9 (standard 100 mil dual right angle)

TOP VIEW			
INPUT_A0	1	2	INPUT_B0
INPUT_A1	3	4	INPUT_B1
INPUT_A2	5	6	INPUT_B2
INPUT_A3	7	8	INPUT_B3
INPUT_A4	9	10	INPUT_B4
INPUT_A5	11	12	INPUT_B5
INPUT_A6	13	14	INPUT_B6
INPUT_A7	15	16	INPUT_B7
DGND	17	18	DGND
OUTPUT_A0	19	20	OUTPUT_B0
OUTPUT_A1	21	22	OUTPUT_B1
OUTPUT_A2	23	24	OUTPUT_B2
OUTPUT_A3	25	26	OUTPUT_B3
OUTPUT_A4	27	28	OUTPUT_B4
OUTPUT_A5	29	30	OUTPUT_B5
OUTPUT_A6	31	32	OUTPUT_B6
OUTPUT_A7	33	34	OUTPUT_B7

Figure 4 - 34-Pin Digital I/O Connector

Signal Name	Definition
INPUT_A0 – INPUT_A7	Digital input byte A.
INPUT_B0 – INPUT_B7	Digital input byte B.
DGND	Digital ground.
OUTPUT_A0 – OUTPUT_A7	Digital output byte A.
OUTPUT_B0 – OUTPUT_B7	Digital output byte B.

2.4.1.3 40-Pin Memory Expansion

Note: *The following connector pinout is for documentation purposes only. The user must not connect anything to this connector other than Chase Scientific compatible boards. For customization and timing information, contact factory.*

CONNECTOR JP10 (uses 50 mil by 100 mil dual connector)

TOP VIEW			
CH1_BIT_11_BUFF	1	2	CH2_BIT_11_BUFF
CH1_BIT_10_BUFF	3	4	CH2_BIT_10_BUFF
CH1_BIT_9_BUFF	5	6	CH2_BIT_9_BUFF
CH1_BIT_8_BUFF	7	8	CH2_BIT_8_BUFF
CH1_BIT_7_BUFF	9	10	CH2_BIT_7_BUFF
CH1_BIT_6_BUFF	11	12	CH2_BIT_6_BUFF
CH1_BIT_5_BUFF	13	14	CH2_BIT_5_BUFF
CH1_BIT_4_BUFF	15	16	CH2_BIT_4_BUFF
CH1_BIT_3_BUFF	17	18	CH2_BIT_3_BUFF

CH1_BIT_2_BUFF	19	20	CH2_BIT_2_BUFF
CH1_BIT_1_BUFF	21	22	CH2_BIT_1_BUFF
CH1_BIT_0_BUFF	23	24	CH2_BIT_0_BUFF
n/c	25	26	n/c
nEXT_RAM_CS	27	28	TRIG_EN_SYNC
MASTER_CLK	29	30	TRIG_DETECT
SEL_MEM_0	31	32	n/a
SEL_MEM_1	33	34	n/a
n/a	35	36	n/a
n/a	37	38	n/a
n/a	39	40	n/a

Figure 5 - 40-Pin Memory Expansion Connector

Signal Name	Definition
CH1_BIT_0_BUFF – CH1_BIT_11_BUFF	Buffered TTL output from high speed channel #1
CH2_BIT_0_BUFF – CH2_BIT_11_BUFF	Buffered TTL output from high speed channel #2
nEXT_RAM_CS	Output TTL low on this pin selects external RAM circuits.
MASTER_CLK	Output TTL A/D clock (rising edge synchronized with A/D's).
SEL_MEM_0, SEL_MEM_1	Output TTL Memory selection control logic.
TRIG_EN_SYNC	Output TTL synchronized trigger enable (active high).
TRIG_DETECT	Output TTL synchronized trigger detection (active high).

2.4.2 Board Configuration

2.4.2.1 Base Address

CS210 Rev. B :

The default Address is 0x240 (240 Hex or 576 Decimal). There is no jumper option for the user to modify the base address on this board revision. The base address is programmed at the factory during the in-circuit programming procedure. If you need an address other than the default, inform Chase Scientific in writing and the board will be shipped with the specified address. The board can also be reprogrammed at the factory with a one-day turn. The board will always be marked with the address value programmed.

CS210 Rev. C :

JP11

1	2	Shunt Position # 1
3	4	Shunt Position #2

Base Address		Header JP7 Position	
Hex	Decimal	Shunt #1	Shunt #2
240*	576*	Open	Open
2C0	704	Open	SHORT

300	768	SHORT	Open
340	832	SHORT	SHORT

*Default

2.4.2.2 IRQ Selection

(See description in software driver section on “HW_IRQ()” which can activate/deactivate interrupts)

JP7

1	2	Shunt Position # 1
3	4	Shunt Position #2
5	6	Shunt Position # 3
7	8	Shunt Position #4

IRQ Jumper Setting Table

IRQ #	Shunt #1	Shunt #2	Shunt #3	Shunt #4
IRQ 10	SHORT	Open	Open	Open
IRQ 11	Open	SHORT	Open	Open
IRQ 12	Open	Open	SHORT	Open
IRQ 15	Open	Open	Open	SHORT

2.4.3 I/O Map

The CS210 occupies 13 consecutive 8-bit locations in I/O space. For example, the default base address is 240 Hex. In this case the board occupies addresses 240 – 24C Hex.

NOTE: The following table provides a general overview of port operation. Due to hardware protocol complexity, the details on how to use these ports are beyond the scope of this manual. The software drivers should provide an effortless integration path for the typical user. However, if the need arises that is beyond what the software drivers can provide, please call the factory for technical support.

Base +	Bits	Write Function	Bits	Read Function
0	0 – 7	82C54 00 (16-bit clk divider)	0	Board Sense
			1	Trigger Done
			2 - 6	<i>not used</i>
			7	
1	0 – 7	82C54 01 (pretrigger value)	0 – 7	<i>not used</i>
2	0 – 7	82C54 10 (postrigger value)	0 – 7	<i>not used</i>
3	0 – 7	82C54 11 (control word)	0 – 7	<i>not used</i>
4	0	A0 ADG439	0 - 15	nREAD CH1 (A/D Buffer #1)
	1	A1 ADG439		
	2	DIN_MAX534, DIN_MAX525		
	3	SCLK_MAX534,		

		SCLK_MAX525		
	4	nCS_MAX525		
	5	nCS_MAX534		
	6	nEN_OUTPUT_CD		
	7	MUX_4CHAN (used by CS402)		
6	0	TRIG_EN	0 – 15	nREAD_CH2 (A/D Buffer #2)
	1	SEL_TRIG_0		
	2	SEL_TRIG_1		
	3	TRIG_POL		
	4	CH1_GAIN_0		
	5	CH1_GAIN_1		
	6	CH2_GAIN_0		
	7	CH2_GAIN_1		
8	0	SE_HS_0	0 – 7	nRD_PORT_A
	1	SE_HS_1		
	2	MUX_HS_0 (not used by CS210)		
	3	MUX_HS_1 (not used by CS210)		
	4	MUX_HS_2 (not used by CS210)		
	5	nRAM_CS		
	6	nCH1_OUT_EN		
	7	nCH2_OUT_EN		
9	0	SEL_MEM_0	0 – 7	nRD_PORT_B
	1	SEL_MEM_1		
	2	SEL_CLK_DIV		
	3	EXT_CLK_SEL		
	4	nEXT_RAM_CS		
	5	DMA_SEL		
	6	not used		
	7	DMA_EN		
10	0-7	OUTPUT_A0-A7	0 – 7	not used
11	0-7	OUTPUT_B0-B7		
12	0	SHUTDOWN (CS210 rev. C)	0-7	not used
	1-7	not used		

Figure 6 - Port I/O

2.5 Register Definitions

The table in Figure 6 2.4.3 provides a general overview of port operation. Due to hardware protocol complexity, the details on how to use these ports are beyond the scope of this manual. The software drivers should provide an effortless integration path for the user. However, if the need arises that is beyond what the software drivers can provide, please call the factory for technical support.

2.6 Analog Input Ranges and Resolution

All analog input offsets and gain are calibrated at the factory to be within +/- 2LSB at 25°C.

CS210-SB-DF-A-5V (bipolar standard)

GAIN	INPUT RANGE	RESOLUTION
------	-------------	------------

1	+/- 5.00 Volts	2.441 mV
2	+/- 2.50 Volts	1.221 mV
5	+/- 1.00 Volts	488 μ V
10	+/- 0.50 Volts	244 μ V

CS210-SU-DF-A-5V (unipolar input option)

GAIN	INPUT RANGE	RESOLUTION
1	0 - 5.00 Volts	1.221 mV
2	0 - 2.50 Volts	610 μ V
5	0 - 1.00 Volts	244 μ V
10	0 - 0.50 Volts	122 μ V

2.7 Analog Output Range and Resolution

The output range is 0-5V and has a 12-bit output resolution of 1.221 mV. The absolute accuracy and offset is adjusted at the factory to be within +/- 2 LSB at 25°C.

2.8 Digital I/O Description

The digital I/O is relatively straightforward. Their inputs and outputs are both CMOS and TTL compatible and can be accessed by reading and writing directly to the boards I/O ports (see 2.4.3) using the drivers provided (see section 4.xx for software driver documentation). The user can also access these ports without driver intervention since there is no underlying protocol required in this case.

Example code in C or Pascal (same for both):

```
Read_Port_Value = Read_Port(PortNum);    // Reads byte from PortNum = 1,2.

Write_Port(PortNum, Enable, PortValue);  // Writes PortValue to PortNum = 1,2;
                                         // Enable (low) = tristate.
```

3 THEORY OF OPERATION

3.1 Introduction

Because the PC/104 bus is much slower than the digitizing rate of the CS210, the data must first be digitized and stored on the CS210 and then transferred to system memory at the slower bus speed.

3.2 Acquiring Analog Input Data

Before capturing data, the user must first setup the correct digitizing rate, trigger input channel, trigger threshold (if applicable), analog input channel, gain, memory size, and pretrigger value.

After proper setup is performed, you're ready to digitize. When the digitize function is called, the A/D converter is activated and begins storing data into buffer RAM. The trigger detect circuit will initially be suppressed until the user programmed pretrigger time has elapsed. Then, after the pretrigger timer has completed and the trigger event has occurred, the A/D data will continue to be stored into the onboard buffer RAM until the postrigger timer has run out.

The buffer memory is circular in design and will actually wrap around on itself during acquisition while it is waiting for a trigger event. For example, if the memory were set to 1024 words and the pretrigger were set to 400 words, the digitizing clock will stop 624 words after the trigger event ($1024 - 400$). This creates a buffer with 400 words before the trigger event and 624 words after the trigger event. Anything that occurred more than 400 words before the trigger event is written over by the postrigger data.

Reading the buffer memories (1 for each high speed input) is done in a sequential manner and does not require any address information to be sent to the CS210. This allows data transfer to be very fast regardless whether you use memory map I/O or port I/O techniques. The only drawback is that if you only wanted a few words out towards the end of the buffer you would have to read through everything up to that point. This can usually be minimized by selecting the right post/pretriggering values.

3.3 Auxiliary Inputs

The (4) lower bandwidth analog inputs are multiplexed with the High Speed CH#1. So, while the two high-speed channels are actually synchronous (converted at the same time and stored into different buffers), the auxiliary inputs are multiplexed sequentially at different times. Note that the auxiliary inputs cannot be used while the high speed CH1 is digitizing. Also note that all the settings for CH1 apply to the auxiliary channels when they are selected.

4 SOFTWARE DRIVERS

4.1 Introduction

Our primary objective in designing software drivers is to get the user up and running as quickly as possible. While the details on individual function calls are listed in sections 4.2.xx, the programming examples in section 4.3.x will show you how to include them into your programs. Please note that the general structures of the

function calls are the same whether you are calling them under DOS, Windows 3.1, Windows 95, or Windows NT.

4.2 Driver Installation

4.2.1 DOS / Windows 3.1

Complete source code is provided in Turbo Pascal 6.0 and Turbo C 4.02. The only installation required to use these drivers is to simply insert them into your code. See the examples in section titled “DOS / Windows 3.1 Using Pascal / C Source”.

4.2.2 Windows 95

To use with Windows 95, copy the “vichw11.vxd” file to your Windows 95 system directory (e.g., C:\<WIN95>\SYSTEM). Then copy cs210drv32.dll to your current directory where your program is and/or to “\<windows95>\system” directory. See example code in section titled “Using Windows 95/NT DLL” on how to use this DLL in an actual program.

4.2.3 Windows NT

To use with Windows NT, copy the “vichw11.sys” file to your Windows NT drivers directory (e.g., C:\<WINNT>\SYSTEM32\DRIVERS) Then copy cs210drv32.dll to your current directory where your program is or to “\<winNT>\system32\ directory” directory. See example code in section titled “Using Windows 95/NT DLL” on how to use this DLL in an actual program.

4.3 Function Calls

4.3.1 Function Declarations in Pascal

Pascal Function Calls =>

```
function Initialize(Port: word): byte;
function Set_Frequency(Divider: word; Clk_Src: byte): byte;
function Select_Channel(SE_Num, Diff: byte): byte;
function Select_Gain(Chan, Gain: byte): byte;
function Set_Trigger(Chan, Polarity, Level: byte): byte;
function Set_Memory_Size(MemSize: byte): byte;
function Set_Pretrigger(Pretrig_Value: word): byte;

function Digitize(Transfer_Mode, Auto_Trigger: byte): byte;
function Transfer_Data(Buffer_Num: integer): byte;
function Read_Long(Buffer_Num: byte): word; {Debug Only}

function Write_Port(PortNum, Enable, PortValue : byte): byte;
function Read_Port(PortNum: byte): byte;
function Set_DAC(DAC_Num, DAC_Value, Latch_Out: word): byte;

function Trigger_Done: boolean;
function Board_Detected: boolean;

procedure HW_IRQ(Enable: integer); [Win95/NT only]
function Get_Var_Pointer(VarNum: byte) : Pointer; [Win95/NT only]
```

```
function Open_Driver: byte;           [Win95/NT only]
function Close_Driver: byte;         [Win95/NT only]
```

```
function Set_IRQ(IRQ_Num: byte; UserIntHandler: pointer): byte; [Win95/NT only]
function Is_IRQ_Set(IRQ_Num: byte): byte; [Win95/NT only]
function Destroy_IRQ(IRQ_Num: byte): byte; [Win95/NT only]
function Simulate_HW_Int(IRQ_Num: byte): byte; [Win95/NT only]
```

=> Advance information. These functions are not available as of date of printing.

4.3.2 Function Declarations in C

C Function Calls =>

```
unsigned char Initialize(unsigned int Port);
unsigned char Set_Frequency(unsigned int Divider, unsigned char Clk_Src);
unsigned char Select_Channel(unsigned char SE_Num, unsigned char Diff);
unsigned char Select_Gain(unsigned char Chan, unsigned char Gain);
unsigned char Set_Trigger(unsigned char Chan, unsigned char Polarity, \
                          unsigned char Level);
unsigned char Set_Memory_Size(unsigned char MemSize);
unsigned char Set_Pretrigger(unsigned int Pretrig_Value);

unsigned char Digitize(unsigned char Transfer_Mode, \
                      unsigned char Auto_Trigger);
unsigned char Transfer_Data(int Buffer_Num);
unsigned int Read_Long(unsigned char Buffer_Num); /* Debug Read from PCB */

unsigned char Write_Port(unsigned char PortNum, unsigned char Enable, \
                        unsigned char PortValue);
unsigned char Read_Port(unsigned char PortNum);
unsigned char Set_DAC(unsigned int DAC_Num, unsigned int DAC_Value, \
                     unsigned int Latch_Out);
unsigned char Trigger_Done();
unsigned char Board_Detected();

void HW_IRQ(short int Enable); [Win95/NT only]
far Get_Var_Pointer(unsigned char VarNum); [Win95/NT only]

unsigned char Open_Driver(); [Win95/NT only]
unsigned char Close_Driver(); [Win95/NT only]
```

```
unsigned char Set_IRQ(unsigned char IRQ_Num, \
                     void *UserIntHandler); [Win95/NT only]
unsigned char Is_IRQ_Set(unsigned char IRQ_Num); [Win95/NT only]
unsigned char Destroy_IRQ(unsigned char IRQ_Num); [Win95/NT only]
unsigned char Simulate_HW_Int(unsigned char IRQ_Num); [Win95/NT only]
```

=> Advance information. These functions are not available as of date of printing.

4.3.3 Function Call Descriptions / Usage

4.3.3.1 Initialize ()

Description

Initializes the CS210 to a known state using the Base Address passed to the software driver.

Declaration

```
function Initialize(Port: word): byte; // Pascal
unsigned char Initialize(unsigned int Port); // C
```

Remarks

Returns 0 if successful. “Port” is the Base port value that all I/O ports are offset from. See Figure 6 for overall port structure.

Examples

```
Initialize($240);    // Pascal
Initialize(0x240);   // C
```

4.3.3.2 Set_Frequency ()**Description**

Sets sampling clock rate.

Declaration

```
function Set_Frequency(Divider: word; Clk_Src: byte): byte; // Pascal
unsigned char Set_Frequency(unsigned int Divider, unsigned char Clk_Src); // C
```

Remarks

Returns 0 if successful.

Divider: 1 <= Divider <= 65535

Clk_Src:

0 = 10 MHz onboard clock source

1 = External clock source (see Figure 3 for input pin location on Analog I/O Connector)

Example

```
Set_Frequency(4,0);    // Set Update frequency to 2.5MHz using 10MHz
                        // onboard clock.
```

4.3.3.3 Select_Channel ()**Description**

Selects High or Low speed input channels on CH1. For the CS210, the only options are 1) select both high speed channel inputs or 2) swap one of the 4 low speed channel inputs with CH1 high speed channel.

Declaration

```
function Select_Channel(SE_Num,Diff: byte): byte; // Pascal
unsigned char Select_Channel(unsigned char SE_Num, unsigned char Diff); // C
```

Remarks

Returns 0 if successful.

SE_Num:

1 : Only available option if Diff = 1. This is normal High Speed, Dual Channel mode.

1 – 4 : This range is valid if Diff = 2. In this case SE_Num represents 1 of the 4 Low Speed channels.

Diff:

1: Differential High Speed mode.

2: Differential Low Speed Mode.

Note: Only values consistent with the above definitions are allowed.

Example

```
Select_Channel (1,1);    // Selects normal High Speed CH1 & CH2.
```

```
Select_Channel (3,2);    // Selects High Speed CH2 and Low Speed CH3 is
```

```
// swapped with High Speed CH1
```

4.3.3.4 Select_Gain ()

Description

Selects gain independently for each of the High Speed Channels. If a Low Speed channel is swapped with High Speed CH1 using “Select_Channel” function, then the Low Speed channel will use the same gain stage as High Speed channel CH1.

Declaration

```
function Select_Gain(Chan,Gain: byte): byte;           // Pascal
unsigned char Select_Gain(unsigned char Chan,unsigned char Gain); // C
```

Remarks

Returns 0 if successful.

Chan: 1,2 [channels 1 or 2]

Gain:

- 1 = Gain of 1, input range of 0 to 5.0V for CS210-SU
Gain of 1, input range of -5.0V to +5.0V for CS210-SB
- 2 = Gain of 2, input range of 0 to 2.5V for CS210-SU
Gain of 2, input range of -2.5V to +2.5V for CS210-SB
- 5 = Gain of 5, input range of 0 to 1.0V for CS210-SU
Gain of 5, input range of -1.0V to +1.0V for CS210-SB
- 10 = Gain of 10, input range of 0 to 0.5V for CS210-SU
Gain of 10, input range of -0.5V to +0.5V for CS210-SB

Example

```
Select_Gain(2,1); // Sets CH2 gain for +/- 5V range
```

4.3.3.5 Set_Trigger ()

Description

Selects inputs used for triggering. Options are CH1, CH2, or external TTL input (see figure 3 for pin location of “EXT_TRIG_IN”). Also sets trigger polarity, and trigger threshold level if an analog input is used.

Declaration

```
function Set_Trigger(Chan,Polarity,Level: byte): byte; // Pascal
unsigned char Set_Trigger(unsigned char Chan,unsigned char Polarity,\
                        unsigned char Level); // C
```

Remarks

Returns 0 if successful.

Chan:

- 1 = Selects High Speed Channel 1 as trigger channel (or low speed channel if swap mode – see section titled “Select_Channel”)
- 2 = Selects High Speed Channel 2 as trigger channel.
- 3 = Selects External TTL Trigger Input.

Polarity:

- 1 = positive edge causes trigger.
- 2 = negative edge causes trigger.

Gain of 2, input range of -2.5V to +2.5V for CS210-SB

Level: 0 <= Level <= 255.

Example

```
Set_Trigger(1,1,127);    // Selects CH1 as trigger input. Sets Polarity
                        // to positive edge triggered and sets trigger
                        // level to 0V (127) for a bipolar input.
```

4.3.3.6 Set_Memory_Size ()

Description

Selects onboard buffer memory size.

Declaration

```
function Set_Memory_Size(MemSize: byte): byte;           // Pascal
unsigned char Set_Memory_Size(unsigned char MemSize);    // C
```

Remarks

Returns 0 if successful.

MemSize:

- 1 = Sets buffer size to 1 Word for each of CH1 & CH2.
- 2 = Sets buffer size to 1K (1024) Word for each of CH1 & CH2.
- 3 = Sets buffer size to 8K (8192) Word for each of CH1 & CH2.
- 4 = Sets buffer size to 32K (32768) Word for each of CH1 & CH2.

Example

```
Set_Memory_Size(2);    // Selects 1K memory size for each of CH1, CH2 buffers
                        // on the CS210 PC/104 module
```

4.3.3.7 Set_Pretrigger ()

Description

Selects the number of words of data that will be available (after digitization) before the trigger event. To calculate the number of words of data that will be available after the trigger event simply subtract the Pretrigger value from the Total Memory size (i.e Mem_Size – Pretrig_Value).

Declaration

```
function Set_Pretrigger(Pretrig_Value: word): byte;       // Pascal
unsigned char Set_Pretrigger(unsigned int Pretrig_Value); // C
```

Remarks

Returns 0 if successful.

Pretrig_Value: 1 <= Pretrig_Value <= Memory_Size

Note: Postrigger value is (Memory_Size – Pretrig_Value)

Example

```
Set_Pretrigger(512);    // If Memory_Size was set to 1024, then the trigger
```

```
// point will be centered in memory. After digitizing
// there will be 512 data points (12-bit words)
// available before the trigger and 512 data points
// after the trigger.
```

4.3.3.8 Digitize ()

Description

Enables trigger circuitry and selects method of data transfer from board to system RAM.

Declaration

```
function Digitize(Transfer_Mode,Auto_Trigger: byte): byte;           // Pascal
unsigned char Digitize(unsigned char Transfer_Mode, \
                        unsigned char Auto_Trigger);                 // C
```

Remarks

Returns 0 if successful. Automatically enables board interrupt function if Transfer_Mode is set to 1 by making a call to HW_IRQ(1). You can manually reset it by “HW_IRQ(0)” when the interrupt has been serviced. “Transfer_Data” will also reset the interrupt when called.

Transfer_Mode:

- 0 = Disables hardware interrupt function.
- 1 = Enables hardware interrupt (jumper must be present for system detection)

Auto_Trigger:

- 0 = Programs board to wait for trigger event. Normal operation.
- 1 = Creates an artificial trigger to occur immediately after trigger circuitry is enabled.

Example

```
Digitize(0,1);              // Enables auto-trigger. If no interrupt handler is
                           // installed, user must check “Trigger_Done”, then
                           // transfer data manually using “Transfer_Data”.
```

4.3.3.9 Transfer_Data () [DOS Only]

Description

Transfers data from CS210 buffer RAM to variable “DataArray [0..1] [0..32767]”.

Declaration

```
function Transfer_Data(Buffer_Num: integer): byte;                 // Pascal
unsigned char Transfer_Data(int Buffer_Num);                         // C
```

Remarks

Returns 0 if successful. Hardware interrupt is also disabled by calling “HW_IRQ(0)”. If using DLL, then you must assign a variable with same structure as DataArray to the far pointer retrieved by calling “Get_Var_Pointer(1)”. See description of Get_Var_Pointer() for more information.

Buffer_Ram:

- 1 = Transfer CH1 (Buffer #1) to “DataArray [0] [0..32767]”.
- 2 = Transfer CH2 (Buffer #2) to “DataArray [1] [0..32767]”.
- 3 = Transfer CH1,2 (Buffer #1,2) to “DataArray [0..1] [0..32767]”.

Example

```
Transfer_Data(1);          // Transfers data from CH1 (buffer#1) to
```

```
// dataArray[0][0..32767].
```

4.3.3.10 Read_Long ()

Description

Reads individual data words. It is used for factory debug and is intended for in-house reference only. Extremely slow for single or multiple transfers. *Not recommended for user applications.*

Declaration

```
function Read_Long(Buffer_Num: byte): word;           // Pascal
unsigned int Read_Long(unsigned char Buffer_Num);     // C
```

Remarks

Buffer_Num:

- 1 = Returns 16-bit word from Buffer #1. Buffer address is auto-incremented.
- 2 = Returns 16-bit word from Buffer #2. Buffer address is auto-incremented.

Example

By repeating the same command 3 times, successive words can be retrieved from Buffer#1 board memory =>

```
var
  Port_Value : array[1..10] of word;           // Pascal example
  x : integer;

for x := 1 to 10 do
begin
  Port_Value[x] := Read_Long(1);
end;
```

4.3.3.11 Write_Port ()

Description

Writes user byte to output ports A or B. See Figure #4 for pinout information on user output ports.

Declaration

```
function Write_Port(PortNum, Enable, PortValue : byte): byte; // Pascal
unsigned char Write_Port(unsigned char PortNum, unsigned char Enable, \
                        unsigned char PortValue);             // C
```

Remarks

Returns 0 if successful.

PortNum:

- 1 = Selects user output port A
- 2 = Selects user output port B.

Enable:

- 0 = Output is tristated.
- 1 = Output is enabled.

PortValue:

- 0 <= PortValue <= 255.

Example

```
Write_Port(1,1,130);      // Writes output port A with byte value 130. Output
                          // is enabled.
```

4.3.3.12Read_Port ()

Description

Reads byte from input port A or B. See Figure #4 for pinout information on user output ports.

Declaration

```
function Read_Port(PortNum: byte): byte;      // Pascal
unsigned char Read_Port(unsigned char PortNum); // C
```

Remarks

Returns Port Value. See Figure 4 for pinout information on user input ports.

PortNum:

- 1 = Selects user input port A
- 2 = Selects user input port B.

Example

```
unsigned byte Port_Value;      // C Example

Port_Value = Read_Port(1);     // Reads input port A into variable
                              // "Port_Value".
```

4.3.3.13Set_DAC ()

Description

Updates 12-bit output DAC's independently or simultaneously. See Figure #3 for pinouts on output DACs.

Declaration

```
function Set_DAC(DAC_Num,DAC_Value,Latch_Out: word): byte;      // Pascal
unsigned char Set_DAC(unsigned int DAC_Num,unsigned int DAC_Value,\
                      unsigned int Latch_Out); // C
```

Remarks

Returns 0 if successful.

DAC_Num:

- 1 = Selects 12-bit DAC output #1.
- 2 = Selects 12-bit DAC output #2.
- 3 = Selects 12-bit DAC output #3.
- 4 = Selects 12-bit DAC output #4.

DAC_Value:

- 0 <= DAC_Value <= 4095.

Latch_Out:

- 0 = DAC_Value is stored in register, but DAC output voltage is not updated.
- 1 = All (4) DAC's are updated with last respective register values.
- 2 = All (4) DAC's are updated with current DAC_Value.

Example

```

Set_DAC(1,2010,1);           // Stores DAC_Value into register of DAC#1,
                              // then updates output voltage of all DAC's.
                              //
                              // If the only change since the last "all DAC
                              // update" was to DAC#1, then DAC#1 is the
                              // output voltage that will change.

```

4.3.3.14 Trigger_Done ()**Description**

Returns TRUE (or "1" for C) if trigger has occurred after Digitize () function has been called. Used to determine if buffered data (on CS210 card) is ready for reading/transferring.

Declaration

```

function Trigger_Done: boolean;           // Pascal
unsigned char Trigger_Done();             // C

```

Remarks

Returns 1 if TRUE, otherwise returns 0. If Trigger_Done returns TRUE, then the hardware interrupt (if enabled) is disabled by calling HW_IRQ(0).

Example

```

if (Trigger_Done) then Transfer_Data(1);           // Pascal Example

if (Trigger_Done > 0)                               // C Example
{
    Transfer_Data(1);
}

```

4.3.3.15 Board_Detected ()**Description**

Returns TRUE (1) if CS210 board is present on the PC/104 bus.

Declaration

```

function Board_Detected: boolean; // Pascal
unsigned char Board_Detected();    // C

```

Remarks

Returns 1 if TRUE, otherwise returns 0. Function **Initialize()** must be called with correct base port address before calling **Board_Detected**. This function is generally only used by **Initialize()** to verify board is present during startup.

Example

```
Initialize($240);  
if (Board_Detected) then Digitize(0,1);           // Pascal Example  
  
Initialize(0x240);  
if (Board_Detected) then Digitize(0,1);           // C Example
```

4.3.3.16HW_IRQ ()**Description**

Enables and disables the hardware interrupt circuitry on the CS210 module.

Declaration

```
procedure HW_IRQ(Enable: integer);           // Pascal  
void HW_IRQ(short int Enable);               // C
```

Remarks

Normally, interrupts are enabled when Digitize is called and disabled when Transfer_Data() is called. Note that the interrupt jumper must be properly in place for interrupts to occur.

Example

```
HW_IRQ(0);           // Resets (disables) hardware interrupt on CS210 module.
```

4.3.3.17Open_Driver () **[Win95/NT Only]****Description**

Loads and initializes Windows 95 (vichw32.vxd) or NT (vichw32.sys) software driver depending on which operation system is detected. This function must be used successfully before any other calls are executed.

Declaration

```
function Open_Driver: byte;                   // Pascal  
unsigned char Open_Driver(void);              // C
```

Remarks

Returns 0 if successful. . User must load interface program “cs210drv.dll” before executing this function. See section “Using Windows 95/NT DLL” for example on importing a DLL using Delphi and Visual C++ programs.

Example

```
Open_Driver;
```

4.3.3.18Close_Driver () **[Win95/NT Only]****Description**

Closes software driver and removes it from memory.

Declaration

```
function Close_Driver: byte;                   // Pascal  
unsigned char Close_Driver(void);              // C
```

Remarks

Returns 0 if successful. .

Example

```
Open_Driver();
```

4.3.3.19 Set_IRQ () [Win95/NT Only] { ADVANCE INFORMATION }

Description

Specifies which hardware interrupt (IRQ10,11,12,15) is to be unmasked for processing. Also passes pointer to UserInterruptHandler.

Declaration

```
function Set_IRQ(IRQ_Num: byte; UserIntHandler: pointer): byte;    // Pascal
unsigned char Set_IRQ(unsigned char IRQ_Num, void *UserIntHandler); // C
```

Remarks

Returns 0 if successful.

IRQ_Num:

10,11,12,15 are only valid IRQ numbers. Any other value will be treated as **nil**.

UserIntHandler:

Pointer to user interrupt handler function or procedure.

Example

```
Set_IRQ(10,@Transfer_Data_Handler);    // Unmasks hardware interrupt 10 and
                                         // points to interrupt handler routine
                                         // Transfer_Data_Handler.
```

4.3.3.20 Is_IRQ_Set () [Win95/NT Only] { ADVANCE INFORMATION }

Description

Determines if hardware IRQ_Num is set.

Declaration

```
function Is_IRQ_Set(IRQ_Num: byte): byte;    // Pascal
unsigned char Is_IRQ_Set(unsigned char IRQ_Num); // C
```

Remarks

Returns 1 if TRUE.

IRQ_Num:

10,11,12,15 are only valid IRQ numbers. Any other value will be treated as **nil**.

Example

```
if (Is_IRQ_Set(10) != 1)
{
    Set_IRQ(10,@Transfer_Data_Handler); ; // Sets interrupt if not already set.
}
```

4.3.3.21 Destroy_IRQ () [Win95/NT Only] { ADVANCE INFORMATION }**Description**

Disables hardware interrupt specified by user.

Declaration

```
function Destroy_IRQ(IRQ_Num: byte): byte;           // Pascal
unsigned char Destroy_IRQ(unsigned char IRQ_Num);    // C
```

Remarks

Returns 0 if successful. Sets interrupt vector to **nil**.

IRQ_Num:

10,11,12,15 are only valid IRQ numbers. Any other value will be treated as **nil**.

Example

```
Destroy_IRQ(10);    // Masked interrupt IRQ10.
```

4.3.3.22 Simulate_HW_Int () [Win95/NT Only] { ADVANCE INFORMATION }**Description**

Simulates a hardware interrupt. Used for debugging purposes.

Declaration

```
function Simulate_HW_Int(IRQ_Num: byte): byte;       // Pascal
unsigned char Simulate_HW_Int(unsigned char IRQ_Num); // C
```

Remarks

Returns 1 if successful.

IRQ_Num:

10,11,12,15 are only valid IRQ numbers. Any other value will be treated as **nil**.

Example

```
Simulate_HW_Int(10);    // Simulates hardware interrupt IRQ10.
```

4.4 Programming Examples**4.4.1 DOS / Windows 3.1 Example Using Borland C++ 5.02**

```
//
// CS210 Sample File - Calls "cs210drv.c"
// =====
//
// 16-Bit DOS/Win31 Version
//
// Filename: cs210tst.c
//
// Web site: http://www.chase2000.com
// Email:    support@chase2000.com
//
// (c) Chase Scientific 1998
```

```

//

#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <alloc.h>
#include <math.h>

#include "cs210drv.c" // Actual driver routines for CS210

/*****

main ()
{
    unsigned long x;
    int i,j;
    BYTE TempReadByte;
    FILE *fout1, *fout2;

// Allocate memory for dynamic variable dataArray. See cs210drv.c
// for declaration of dataArray.
    if ((dataArray = (Array) calloc(2,sizeof(WORD *))) == NULL)
    {
        printf( "Not Enough Memory\n" ) ;
        exit(1) ;
    }
    for (x = 0; x < 2; x++)
    {
        if ( (dataArray[x] = (WORD *) calloc(32767,sizeof(WORD))) == NULL )
        {
            printf( "Not Enough Memory\n" ) ;
            exit(1);
        }
        else
            if (x == 1) printf ("calloc Successful\n" ) ;
    }

    fout1 = fopen("data1.dat", "w") ;
    fout2 = fopen("data2.dat", "w") ;

    Initialize(0x240); // Initialize base port to 240 Hex

    if (Board_Detected())
        printf("CS210 Board Detected. \n") ;
    else
    {
        printf("CS210 Board NOT Detected. \n") ;
        exit(1) ;
    }

    printf("Setting up paramters ... \n") ;

    Set_Frequency(4,0); // Set update clock rate to 2.5MHz using 10MHz
                        // onboard clock.

    Select_Channel (1,1); // Selects normal High Speed CH1 & CH2
    Select_Gain(1,1); // Sets CH1 gain to +/- 5V range
    Select_Gain(2,1); // Sets CH2 gain to +/- 5V range

    Set_Trigger(1,1,128); // Selects CH1 as trigger input. Sets polarity
                        // to positive edge
    triggered and sets trigger

```

```

// level to 0V (128)
for bipolar input.

    Set_Memory_Size(4);        // "4" Selects 32K memory size for each of CH1, CH2
                                // buffers on the
CS210 PC/104 module.

    Set_Pretrigger(512);       // Centers trigger point in memory. There will be
                                // 512 12-bit data
points before the trigger
                                // event and 512 data
points after trigger event.

printf("Start Digitize. \n") ;
Digitize(0,1);                // 2nd Parameter "1" Enables auto-trigger.

printf("Waiting for PostTriggering to Complete ... \n") ;
while (!Trigger_Done());
printf("Digitize Complete! \n\n") ;

printf("Transfer Data from Board to System Memory. \n") ;
Transfer_Data(3);              // Manually transfers data from CH1 & CH2 buffer
                                // data to dataArray[0][].

Write_Port(1,1,130);          // Write output port A with byte value 130. Output
                                // is enabled.

TempReadByte = Read_Port(1);   // Reads user input port 1 into TempReadByte

Set_DAC(1,0,2);               // Updates all DACs with zeros

printf("Writing CH1, CH2 Data from Memory to Files Data1.dat, Data2.dat \n") ;
for ( i = 0; i < Memory_Size; i++ ) {
    fprintf( fout1, "%d\n", dataArray[0][i] ) ;
    fprintf( fout2, "%d\n", dataArray[1][i] ) ;
}
printf("Writing Complete. \n\n") ;

for (i = 0; i < 2; i++)        // Release memory for dynamic variable dataArray
    free(dataArray[i]);
free(dataArray);

fclose( fout1 ) ;
fclose( fout2 ) ;

printf("");
char buffer[83];
char *p;
printf("Hit return key to exit");
p = cgets(buffer);

return (0);
}

```

4.4.2 Windows 95/NT DLL Example using Borland C++ 5.02

```

//
// CS210 Example File for Borland C++ 5.02
// =====
//
// 32-Bit Win95/NT Version for "cs210drv32.dll"
//
// Filename: cs210tst_dll.cpp
//
// Web site: http://www.chase2000.com
// Email: support@chase2000.com
//
// (c) Chase Scientific 1998
//

#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <alloc.h>
#include <math.h>

#include "cs210drv.h" // Header file for cs210 DLL
#pragma link "cs210drv32.lib"

/*****

main ()
{
    unsigned long x;
    int i,j;
    BYTE TempReadByte;
    FILE *fout1, *fout2;

    typedef WORD **Array;
    Array DataArray;

    WORD Memory_Size = 32767;

    // Allocate memory for dynamic variable DataArray. See cs210drv.c
    // for declaration of DataArray.
    if ((DataArray = (Array) calloc(2,sizeof(WORD *))) == NULL)
    {
        printf( "Not Enough Memory\n" ) ;
        exit(1) ;
    }
    for (x = 0; x < 2; x++)
    {
        if ( (DataArray[x] = (WORD *) calloc(32767,sizeof(WORD))) == NULL )
        {
            printf( "Not Enough Memory\n" ) ;
            exit(1);
        }
        else
            if (x == 1) printf ("calloc Successful\n" ) ;
    }

    fout1 = fopen("data1.dat", "w") ;
    fout2 = fopen("data2.dat", "w") ;

    Open_Driver();

    CS210_Initialize(0x240); // Initialize base port to 240 Hex

```

```

if (Board_Detected())
    printf("CS210 Board Detected. \n") ;
else
{
    printf("CS210 Board NOT Detected. \n") ;
    exit(1) ;
}

printf("Setting up paramters ... \n") ;

Set_Frequency(4,0);          // Set update clock rate to 2.5MHz using 10MHz
                               // onboard clock.

Select_Channel (1,1);        // Selects normal High Speed CH1 & CH2
Select_Gain(1,1);            // Sets CH1 gain to 1V/V, +/- 5V range
Select_Gain(2,1);            // Sets CH2 gain to 1V/V, +/- 5V range

Set_Trigger(1,1,128);        // Selects CH1 as trigger input. Sets polarity
                               // to positive edge triggered and sets trigger
                               // level to 0V (128) for bipolar input.

Set_Memory_Size(4);          // "4" Selects 32K memory size for each of CH1, CH2
                               // buffers on the CS210 PC/104 module.

Set_Pretrigger(512);         // Centers trigger point in memory. There will be
                               // 512 12-bit data points before the trigger
                               // event and 512 data points after trigger event.

printf("Start Digitize. \n") ;
Digitize(0,1);               // 2nd Parameter "1" Enables auto-trigger.

printf("Waiting for PostTriggering to Complete ... \n") ;
while (!Trigger_Done());
printf("Digitize Complete! \n\n") ;

printf("Transfer Data from Board to System Memory. \n") ;
Transfer_Data(1,DataArray[0]);
Transfer_Data(2,DataArray[1]);

Write_Port(1,1,130);         // Write output port A with byte value 130. Output
                               // is enabled.

TempReadByte = Read_Port(1);  // Reads user input port 1 into TempReadByte

Set_DAC(1,0,2);              // Updates all DACs with zeros

printf("Writing CH1, CH2 Data from Memory to Files Data1.dat, Data2.dat \n") ;
for ( i = 0; i <= Memory_Size; i++ ) {
    fprintf( fout1, "%d\n", DataArray[0][i] ) ;
    fprintf( fout2, "%d\n", DataArray[1][i] ) ;
}
printf("Writing Complete. \n\n") ;

for (i = 0; i < 2; i++) // Release memory for dynamic variable DataArray
    free(DataArray[i]);
free(DataArray);

fclose( fout1 ) ;
fclose( fout2 ) ;

```

```

    printf("");
    char buffer[83];
    char *p;
    printf("Hit return key to exit");
    p = cgets(buffer);

Close_Driver();

    return (0);

}

//
// CS210 Header File for 32-Bit DLL
// =====
//
// 32-Bit Win95/NT Version for "cs210drv32.dll"
//
// Filename: cs210drv.h
//
// Web site: http://www.chase2000.com
// Email:    support@chase2000.com
//
// (c) Chase Scientific 1998
//

#ifndef cs210drvH
#define cs210drvH

typedef unsigned short WORD;
typedef unsigned char BYTE;
#define IMPORT extern "C" __stdcall

IMPORT BYTE CS210_Initialize(WORD Port);
IMPORT BYTE Set_Frequency(WORD Divider, BYTE Clk_Src);
IMPORT BYTE Select_Channel(BYTE SE_Num, BYTE Diff);
IMPORT BYTE Select_Gain(BYTE Chan, BYTE Gain);
IMPORT BYTE Set_Trigger(BYTE Chan, BYTE Polarity, BYTE Level);
IMPORT BYTE Set_Memory_Size(BYTE MemSize); /*1,2,3,4: 1,1K,8K,32K*/
IMPORT BYTE Set_Pretrigger(WORD Pretrig_Value);
IMPORT BYTE Digitize(BYTE Transfer_Mode, BYTE Auto_Trigger);
IMPORT BYTE Transfer_Data(int Buffer_Num, void *UserPtr);
IMPORT WORD Read_Long(BYTE Buffer_Num); /* Debug Read from PCB */

IMPORT BYTE Write_Port(BYTE PortNum, BYTE Enable, BYTE PortValue);
IMPORT BYTE Read_Port(BYTE PortNum);
IMPORT BYTE Set_DAC(WORD DAC_Num, WORD DAC_Value, WORD Latch_Out);
IMPORT BYTE Trigger_Done(void);
IMPORT BYTE Board_Detected(void); /*0 = not detected, 1= detected*/
IMPORT void HW_IRQ(int Enable);

IMPORT void Set_8Bit_DAC(BYTE DAC_Num, BYTE Value);

IMPORT BYTE Open_Driver(void);
IMPORT BYTE Close_Driver(void);

#endif

```

4.5 Error Return Code List

Code	Constant Name	Description
------	---------------	-------------

0	Successful	Normal operation. Successful function execution.
1	Par Out Of Range	Parameter passed to function is out of range.
2	Board Not Detected	Unable to detect presence of CS210 module on bus.

5 MISCELLANEOUS

5.1 Calibration

The CS210 has no user feature to calibrate for input and output gain accuracy or offsets. The gains and offsets are calibrated at the factory to be within 2 LSB at 25°C and nominal voltages.

5.2 Maintenance

No maintenance is required. However, a yearly calibration is recommended if the user desires to maintain the CS210 modules' specified accuracy. Call factory for maintenance and/or extended warranty information.

Trademarks:

MS-DOS, Windows 3.1, Windows 95, and Windows NT are registered trademarks of Microsoft Corporation.