EUROPEAN LABORATORY FOR PARTICLE PHYSICS
CERN — BE DEPARTMENT

CERN-BE-Note-2009-035 BI

Automatic Test Card – LINAC4 SPL

M. Aqil

Abstract

In order to fully characterize an analogue to digital converter its linearity, distortion, cross talk etc. must be measured. This can be done by supplying known test signals to each of its inputs and analysing the digital values read from the device.

Geneva, Switzerland
07/12/2009

Table of Contents

Chapter 01
Introduction

In order to fully characterize an analogue to digital converter its linearity, distortion, cross talk etc. must be measured. This can be done by supplying known test signals to each of its inputs and analysing the digital values read from the device.

1.1 ADC card

For the readout of secondary emission grids (SEMGrids), used for transverse profile measurements, several tens of parallel analogue to digital conversion channels are needed. For the SEMGrids in Linac-4 the profiles must in addition be time-resolved, which requires sampling of the individual channels at a rate of some 250 kHz. In order to get a cost effective solution a VME based multichannel ADC card featuring 36 parallel ADC channels has been designed. The ADCs are configured and read out with the help of an FPGA which can be programmed over the VME bus. In addition memory extern to the FPGA is available. FPGA registers are accessible through the VME bus or through an AVR microcontroller via a serial protocol which allows programming the card through any of the two access channels. (reference to Greg's hardware manual)

1.2 Test Card

Since some 100 ADC cards are required to fulfil the demand for SEMgrids, DC current transformers and maybe other types of measurement equipment an efficient test procedure is required allowing quick verification of the card's functionality and giving access to its function through easily usable testfunctions. For this reason a programmable test card has been designed supplying test signals as well as all necessary timing signals to the ADC card. Through a multiplexing scheme either a pure sine signal, a DC signal of programmable signal level or ground can be fed to any of the ADC channels. The test card is controlled through an AVR microcontroller, which is accessed by a PC through a serial protocol. (reference to Greg's hardware manual)

1.3 Scope of the project

The scope of the project was the implementation of a serial protocol for the AVR microcontroller on ADC test card. This protocol must give access to functions controlling the signal generator, the DC voltage generator and the multiplexer on the test card as well as the creation of trigger signals to the ADC.

In addition a GUI based program written in MatLab on a PC was needed to give access to these test functions as well as readout of the ADC.

This report describes the AVR Microcontroller code on the test card and the GUI based PC driver used to analyze the properties of each ADC channel by exposing it to various signals through the Test Card.

This document contains a user manual for the operation of Test Card but also a description of all code to be used for future enhancements.

The report is divided into the following chapters:

1. Introduction
2. Hardware Description
3. Command/Data Communication Protocol
4. Test Card Programming
5. PC Side Programming
6. User Manual
7. Appendices

For people interested in the operation of the system only, without detailed knowledge of its implementation, reading chapter 6 is sufficient.

Hardware description explains hardware design and functionality of the Test card.

….

….

Chapter 02
Hardware Description

2.1 Hardware Description of the Test Card

Figure 1 shows a block diagram of the ADC test card built around an AVR (ATmega8L) microcontroller:

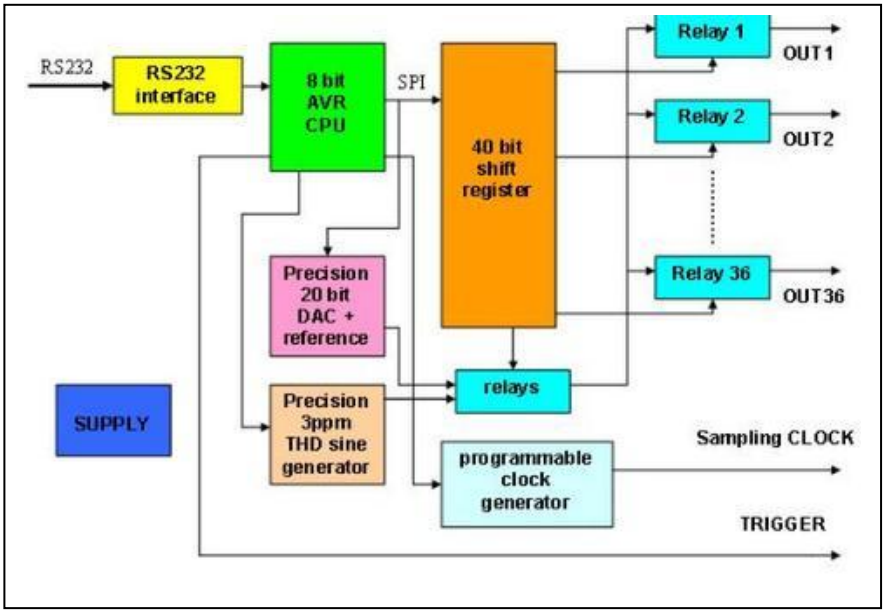


Figure 1: Block Diagram of the Test Card [Reference]

The microcontroller has the responsibility of controlling all the components according to serially received user commands (see next chapter for details).

There are 36 channel relays connected to the AVR through a 40 bit shift register via the SPI port. NC (normally closed) contacts of each relay are connected to ground and NO (normally open) contacts are connected to the output of the signal-relay used to switch between sine wave and DC voltage. Sine wave is connected to NC contact of the signal-relay and DC voltage is connected to the NO contact.

The negative signal of the signal-relay is connected to NO contact of the signal polarity-relay. The NC contact of the signal polarity-relay is connected to ground for asymmetric signal selection. The sine wave generator is always active. It's frequency can be switched between Freq1 (NC contact) and Freq2 (NO contact) by a Generator-range-relay.

To generate the DC voltage a 20-bit DAC is attached to the AVR through the SPI port. The DAC is controlled in 20 bit straight binary mode. It takes 20 bit data in the form of left adjusted 3 bytes (24 bits).

The ADC sampling clock can also be provided from the test card. For this a 1MHz crystal module is employed, the output of which is passed to an 8-bit (actually dual 4-bit) binary counter. Each divide by 2 output of the counter is connected to a multiplexer. The multiplexer is controlled by the AVR through 3 binary connections Freq_Sel 0, 1 and 2 to select a particular frequency as ADC sampling clock.

The AVR's port-pin addresses for the above mentioned devices are tabulated below:

Table 1: Test Card's AVR MCU Pin Connections

<table>
<tr><th>AVR MCU Pin</th><th>Active Level</th><th>Connection Name</th><th colspan="4">Description</th></tr>
<tr><td>PB4</td><td></td><td>MISO</td><td colspan="4">SPI data line, input, controlled by SPI interface</td></tr>
<tr><td>PB3</td><td></td><td>MOSI</td><td colspan="4">SPI data line, output, controlled by SPI interface</td></tr>
<tr><td>PB2</td><td>High</td><td>LATCH_relays</td><td colspan="4">Latches data sent to the shift registers. Toggle this pin</td></tr>
<tr><td>PB1</td><td>Low</td><td>DAC_SIO_En</td><td colspan="4">DAC serial data input enable – low enables DAC write</td></tr>
<tr><td>PB0</td><td>Low</td><td>DAC_CSn</td><td colspan="4">DAC chip select – must be low during SPI transfer to the DAC</td></tr>
<tr><td rowspan="4">PD7
PD6
PD5</td><td rowspan="4"></td><td rowspan="4">FREQ_SEL_2..0</td><td colspan="4">Selects ADC sampling clock</td></tr>
<tr><td>PD7</td><td>PD6</td><td>PD5</td><td>Frequency</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>500 KHz</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>250 KHz</td></tr>
</table>

			0	1	0	125 KHz
			0	1	1	62.5 KHz
			1	0	0	31.25 KHz
			1	0	1	15.625 KHz
			1	1	0	7.8125 KHz
			1	1	1	3.90625 KHz
PC4	High	GEN_range	Selects sine wave generator high frequency.			
PC3	High	SRC_SW0	Selects signal source. High: DAC, Low: Generator			
PC2	High	SRC_SW1	Selects symmetrical/asymmetrical signal source. High: asymetrical			
PC1	High	TRIG_1	Trigger 1 output for ADC card. Toggle to trigger measurements			
PC0	High	TRIG_2	Trigger 2 output for ADC card. Toggle to trigger measurements			
PD4		DIG_IO1	Digital input/output 1, general purpose			
PD3		DIG_IO2	Digital input/output 2, general purpose			

See Appendix_____ for the detailed schematic.

All these options are selectable by using particular commands, see next chapters for the communication protocol and C-code algorithm.

Chapter 03
Command/Data Communication Protocol of the Test Card

Although a Matlab GUI based driver software has been written to allow selecting any available option of the test card the communication protocol is presented here. Its understanding permits modification or a re-write of the MatLab code. See Appendix ---- for the full commands chart.

The Serial Baud Rate is fixed to **19200 bits per seconds**.

The communication protol is divided into three types of commands,

1. Hello command (for initialisation)
2. SET commands
3. Read commands

3.1 Hello Command

Hello command is used to initialize the serial port. This command consists of only 1 byte 'H'.

Command	Discription	Execution Response	Discription
'H'	Say Hello to initialize the Serial port	'H'	Received Hi

In response the Card will return 'H' (Hi). This communication ensures that serial connection is correctly set up.

3.2 SET commands

Set commands are used to set/change the value of any option of the test card. This command is further divided into three types on the basis of the bytes following the command.

3.1.1 Set-commands following one bytes

There is only one set command that follows one byte,

Command	Sub-Command	Discription	Execution Response		Discription
'S'	'd'	Initialize all to default values	'E'	'd'	Executed Default Command

This command initializes all the options of the test card to default values. The default values are as follows:

Option	Default Values
Signal Source Polarity	Symmetric
ADC Clock Frequency	500 KHz
Sine wave Frequency	Freq1
Digital I/O 0 & 1	1 (as Input)
Signal other than GND	SineWave
Trigger Output 0 & 1	0 (as Output)
DAC O/P	0 V
Mux Channels	All are GND

In response of this, the controller initializes all its variables and test card options according to the above mentioned default values and sends back ‘E’ and ‘d’ to confirm that the default values are set.

3.1.2 Set-commands following two bytes

Most of the set commands follow two bytes, as seen from the underneath table,

Command	Sub-Command	Parametric Byte	Discription	Execution Response		Discription
'S'	'A'		Signal Source Polarity	'E'	'A'	Executed Signal Polarity Command
		'0'	Symmetric			
		'1'	Asymmetric			
'S'	'C'		ADC Clock Frequency	'E'	'C'	Executed ADC Sample Clock Command
		'0'	500 KHz			
		'1'	250 KHz			
		'2'	125 KHz			
		'3'	62.5 KHz			
		'4'	31.25 KHz			
		'5'	15.625 KHz			
		'6'	7.8125 KHz			
		'7'	3.90625 KHz			
'S'	'G'		Sine wave Frequency	'E'	'G'	Executed Sinewave Frequency
		'0'	Freq1			
		'1'	Freq2			
'S'	'I'		Dig_I_O_1 | Dig_I_O_2	'E'	'I'	Executed Digital I/O Command
		'0'	0 | 0			
		'1'	0 | 1			
		'2'	1 | 0			
		'3'	1 | 1			
'S'	'S'		Signal other than GND	'E'	'S'	Executed Signal (other than GND) Select Command
		'0'	SineWave			
		'1'	DC Voltage			
'S'	'T'		Trig_O_1 | Trig_O_2	'E'	'T'	Executed Trigger O/P Command
		'0'	0 | 0			
		'1'	0 | 1			
		'2'	1 | 0			
		'3'	1 | 1			

The table shows how one can set any option by sending Set-command ‘S’ following a sub set-command and a defined byte. The received execution response confirms the desired selection.

3.1.3 Set-commands following four bytes

The DC voltage can be produced/changed through DAC write command as follows:

Command	Sub-Command	DAC Bytes	Discription	Execution Response		Discription
'S'	'D'	3 Binary Bytes (MS-Byte First) (Left adjusted 20 bits of DAC value)	Set the binary formatted DAC value to generate required DC voltage	'E'	'D'	Executed DAC Write Command

The test card employs a DAC1220 which produces 0V to 5V by taking 20 bits serially formatted as straight binary or 2's complement. To have a bipolar output of -5V to 5V, operational amplifiers have been used, see circuit in Appendix ____.

The transfer function of the DAC along with the OpAmps (up to the output connectors) has been measured. The calibration equation, so found, is as follows,

$$DAC = round[\,(1.0656 * Voltage + 5.3047) * 10^5]$$

Equation 1

Once the desired voltages are known one has to find the binary equivalent DAC value by using above equation. The DAC value is sent to the test card after shifting le obtained binary value 4 times left. The execution response of the test card will be received to inform that the desired voltage has been produced.

3.1.4 Set-commands following six bytes

All channel outputs of the test card are controlled by the AVR controller through MUX latch-relays . Ground is connected to all channels by default. The selected signal (Sine wave or DC Voltage), other than ground, can be sourced at a channel by energising the particular latch-relay. The following command sends the setting to all 36 channels,

Command	Sub-Command	MUX-Latch Bytes	Discription	Execution Response		Discription
'S'	'M'	5 Binary Bytes (MS-Byte First) (Right adjusted 36 bits of MUX-latch relays logics)	Set relay logics of 36 output Channels 0 to apply GND and 1 to apply selected signal	'E'	'M'	Executed MUX-Relays Command

The five binary bytes should have the correct binary formatted information to connect the channels to ground (if '0') or to connect to the selected source (if '1'). Each channel information should be placed at the desired bit location as shown in above figure (i.e. channel 0 at bit 0 and channel 35 at bit 35, and the most significant 4 bits are don't care).

Character 'E' and 'M' will be received after the successful execution of this command.

3.3 READ commands

ead command is used to find the current configuration of a particular card option. Read commands are as follows,

Request Command

Command	Sub-Command	Discription
'R'	'A'	Signal Polarity Info
'R'	'C'	ADC Clock Info
'R'	'G'	Generator Info
'R'	'I'	Digital I/O Info
'R'	'S'	Signal Source Info
'R'	'T'	Trigger O/P Info
'R'	'D'	DAC Info
'R'	'M'	MUX-Latch-Relay Info

'R' : Read ...

Execution Response

Catch	Sub-Command	Following Byte(s)	Discription
'C'	'A'	'0'/'1'	'1' = Asynchronous, '0' = Synchronous
'C'	'C'	'0' --- '7'	'0' = 500KHz, '1' = 250KHz , ..., '7' = 1.953125 KHz
'C'	'G'	'0'/'1'	'0' = freq1, '1' = freq2
'C'	'I'	'0' --- '3'	(Dig_I_O_2 = bit0 and Dig_I_O_1 = bit1)
'C'	'S'	'0'/'1'	'0' = SineWave, '1' = DC Voltage
'C'	'T'	'0' --- '3'	(Trig_2 = bit0 and Trig_1 = bit1)
'C'	'D'	3 Binary Bytes (MS-Byte First)	Left adjusted 20 bits of DAC value (use inverse DAC equation for voltage)
'C'	'M'	5 Binary Bytes (MS-Byte First)	Relay logics of 36 output Channels (right adjusted); '0' = GND, '1' = Selected signal

'C' : Catch...

All the read/request commands consist of 2 bytes. In response of a read command, the user will receive a catch frame of the requested option. The catch frames for each options are tabulated above. The format of the read commands are similar to the corresponding set commands except the first byte, 'C' instead of 'S'. To find the current DC voltages, the DAC bytes should be shifted right 4 times and then passed to the following equation (inverse of Equation 1).

$$Voltage = 9.3842 * 10^{-6} * \text{DAC} - 4.978$$

Equation 2

3.4 Error Messages

As discussed above, there are three valid command bytes ('H', 'S' & 'R'), their respective sub-command bytes and valid range of parameters (following bytes). Parametric following bytes of 'D' and 'M' sub-commands are not limited in values.

Error codes are used to inform the user in case of detecting an invalid command/sub-command or parametric byte. The following table describes the error codes,

Error Code	Discription
'X'	Invalid Command or Sub-Command
'P'	Invalid Parameter

The test card discarts the requested command in case of receiving an invalid command or invalid sub-command and notifies it by sending invalid command/sub-command byte 'X'. In case of receiving invalid parametric byte (following byte) the requested command is discarted and notified as invalid parametric byte 'P'. There is no parametric error for the DAC and MUX commands.

Chapter 04
Test Card Programming

The AVR Microcontroller of the test card is programmed in C language by using the environment WinAVR 2007. This chapter discusses the code by means of an abstract algorithm of the code, the C-code is attached in Appendix ____. The soft version of the program can be found in the attached CD.

First note that the WinAVR compiler's "Make file" must be edited to define the current hardware as:

1. 'MCU = atmega8' at line number 44, define of AVR controller
2. F_CPU = 8000000 at line number 53, define AVR crystal-clock's speed
3. TARGET = Serial_ProtocolEC at line number 61, define target project
4. SRC = $(TARGET).c Initialize.c at line number 65, define other C-file for linking

The whole program consits of seven functions spread over two files "Initialize.c" and "Serial_Protocol_EC.c". The following algorithm describes these functions in sequence of their arrival during the execution.

4.1 AVR Code (Algorithm)

As mentioned above, the code is divided into seven functions, so the algorithm for seven functions is as follows

4.1.1 AVR code's Main function

Step 00: Define Functions excluding Receive interrupt function

Step 01: Define variables to be be used for command and data manipulation

Step 02: Start of main function

Step 03: Set baud rate variable to 25 to produce 19200 baud rate with 8MHz Crystal

Step 04: Call "Initialze" function with input argument baud rate

Step 05: Call the "Init_Variables" to set all variables to default, mostly 0

Step 06: Enable global interrupt

Step 07: Call “Initialize_DAC” function again to avoid port overloads

Step 08: Infinitely stay here and execute serial ports receive interrupt as arrived.

4.1.2 AVR code’s Initialize function

This function sets port directions depending on connections, configures serial & SPI ports, initializes all MUX-relays to 0 and calls “Initialize_DAC” function.

Step 00: Define dirrection of B0(DAC_CSn), B1(DAC_SDI_En), B2(LATCH_relay), B3(MOSI), B5(SCK) as output

Step 01: Define direction of C0(trig_out2), C1(trig_out1), C2(SRC_SW1), C3(SRC_SW1), C4(GEN_range) as output

Step 02: Define direction of D3(Dig I/O 2), D4(Dig I/O 1), D5--D7(Freq_sel 0--2) as output

Step 03: Deactivate (set to low) DAC_CSn and DAC_SDIO_En initially

Step 04: Set all pins of Port C to zero to set default values

Step 05: Set Port D3(Dig I/O 2) and D4(Dig I/O 1)

Step 06: Clear Port D5 to D7 for lowest ADC clock generation

Step 07: Initialize serial port in 8-bit mode with one stop bit and baud rate as passed. Enable transmission and interrupt based reception.

Step 08: Initialize SPI (serial peripheral Interface) as master with 0.5MHz (Xtal/16) clock

Step 09: Set MUX-relays, attached to SPI, to zero (send five zero bytes with PB2 active)

Step 10: Call Initialize_DAC fuunction

4.1.3 AVR code’s Initialize_DAC function

This function initializes the DAC (DAC1220E) connected to the SPI port.

One thing to be noted: the selected DAC samples data on the rising edge. Therefore the SPI clock phase should be switched (CPHA=1) to sample the data at trailling edge of ACK. This ensures a valid logic level at the following rising edge.

Step 00: Set CPHA to 1 for DAC communication, as discussed above

Step 01: Activate (make low) the DAC select bits, DAC_CSn and DAC_SDI_En

Step 02: Wait for 20 µsec

Step 03: Load SPDR with DAC command writing byte (0x24)

Step 04: Wait for SPIF bit in SPSR to set

Step 05: Wait for 20 µsec

Step 06: Load SPDR with DAC's 1st command byte (0x02)

Step 07: Wait for SPIF bit in SPSR to set

Step 08: Load SPDR with DAC's 2nd command byte (0xF0)

Step 09: Wait for SPIF bit in SPSR to set

Step 10: Wait for 20 µsec

Step 11: Deactivate (make high) the DAC select bits, DAC_CSn and DAC_SDI_En

Step 12: Wait for 2 msec [to write three data bytes to make output voltage to zero]

Step 13: Activate (make low) the DAC select bits, DAC_CSn and DAC_SDI_En

Step 14: Wait for 10 µsec

Step 15: Load SPDR with DAC's data write writing byte (0x40)

Step 16: Wait for SPIF bit in SPSR to set

Step 17: Wait for 10 µsec

Step 18: Load SPDR with 0x00 (because of straight binary mode)

Step 19: Wait for SPIF bit in SPSR to set

Step 20: Repeat above 2 steps 2 times

Step 21: Wait for 10 µsec

Step 22: Deactivate (make high) the DAC select bits, DAC_CSn and DAC_SDI_En

Step 23: Set CPHA to 0 for MUX relays

4.1.4 AVR code's Init_Variables function

To save the current configuration at AVR controller level corresponding variables are made. This function initializes these variables to default values as,

Step 00: Signal polarity to Symetric

Step 01: ADC sample clock to highest (500KHz)

Step 02: Sinewave frequency to Lowest

Step 03: Sinewave as the selected source, other than ground

Step 04: Both Digital I/Os as 1

Step 05: Both Trigger O/Ps as 0

Step 06: Three bytes DAC array to 0

Step 07: Five bytes MUX array to 0

Step 08: Serial reception frame and length to 0

4.1.5 AVR code's Serial Port Receive Interrupt function

This interrupt based function is designed to obey the user commands promptly. A frame variable is used to receive different formatted commands, see communication protocol above for a list of commands with full format. Command execution is done as the full command is received and a command execution response is transmitted back to the user.

Step 00: Save the received byte in a serial temporary variable STemp

Step 01: If frame = 0, follow the steps otherwise go to step # 06

Step 02: If STemp = 'H' (Hello command) then Call Transmit function to send back 'H' and return from serial interrupt routine

Step 03: If STemp = 'S' (Set command) then make frame = 1 for bytes to follow and return from serial interrupt routine

Step 04: If STemp = 'R' (Read command) then make frame = 3 for bytes to follow and return from serial interrupt routine

Step 05: Otherwise call Transmit function to send 'X' as command/sub-command error byte and return from serial interrupt routine

Step 06: If frame = 1, follow the steps otherwise go to step # 13

Step 07: Save the received byte as command (set's sub-command)

Step 08: If command = 'd' (default configuration command) then

a. Call Initialize function with variable BaudRate to initialize hardware to default
b. Call Init_Variables function to initialize all variables to default as well
c. Transmit Execution response by sending characters 'Ed'
d. Make frame = 0 for next command's service
e. Return from serial interrupt routine

Step 09: In all the following six cases make frame = 2 and F_Length = 1 to receive one parametric byte and return from serial interrupt routine

i. Command = 'A' (signal polarity Asymetric/Symetric command)
ii. Command = 'C' (ADC clock command)
iii. Command = 'G' (Generator range command for sinewave's frequency)
iv. Command = 'I' (Digital I/O command)
v. Command = 'S' (Signal source, other than ground, selection command)

vi. Command = ‘T’ (Trigger O/P command)

Step 10: If command = ‘D’, make frame = 2 and F_Length = 3 to receive three DAC parameter bytes and return from serial interrupt routine

Step 11: If command = ‘M’, make frame = 2 and F_Length = 5 to receive five MUX-relays parameter bytes and return from serial interrupt routine

Step 12: Otherwise call Transmit function to send ‘X’ as command/sub-command error byte, Make frame = 0 for next command’s service and return from serial interrupt routine

Step 13: If frame = 2, follow the steps otherwise go to step # 24

Step 14: Decrease F_Length by one

Step 15: If F_Length = 0 make frame = 0

Step 16: If command = ‘A’ then

a. If STmp = ‘0’ or ‘1’ then follow next bullets otherwise go to Step 16 (i)
b. Save STemp in “Asym” variable
c. Shift STemp left 2 times to allign value to desired port pin
d. Mask all bits of STemp to 0 except Bit-2
e. Clr PortC2 (Asymetric control pin)
f. OR'ed PortC to the masked STemp byte
g. Transmit Execution response by sending characters ‘EA’
h. Return from serial interrupt routine
i. Call Transmit function to send ‘P’ as parameter error byte, Make frame = 0 for next command’s service and return from serial interrupt routine

Step 17: If command = ‘C’ then

a. If STmp = ‘0’ to ‘7’ then follow next bullets otherwise go to Step 17 (i)
b. Save STemp in “Clk” variable
c. Shift STemp left 5 times to allign value to desired port pin
d. Mask all bits of STemp to 0 except Bits 5 to 7
e. Clr PortD5 to PortD7 (Clock frequency control pins)
f. OR'ed PortD to the masked STemp byte
g. Transmit Execution response by sending characters ‘EC’
h. Return from serial interrupt routine

i. Call Transmit function to send ‘P’ as parameter error byte, Make frame = 0 for next command’s service and return from serial interrupt routine

Step 18: If command = ‘G’ then

a. If STmp = ‘0’ or ‘1’ then follow next bullets otherwise go to Step 18 (i)
b. Save STemp in “Generator” variable
c. Shift STemp left 4 times to allign value to desired port pin
d. Mask all bits of STemp to 0 except Bit-4
e. Clr PortC4 (Sinewave frequency generator's control pin)
f. OR'ed PortC to the masked STemp byte
g. Transmit Execution response by sending characters ‘EG’
h. Return from serial interrupt routine
i. Call Transmit function to send ‘P’ as parameter error byte, Make frame = 0 for next command’s service and return from serial interrupt routine

Step 19: If command = ‘I’ then

a. If STmp = ‘0’ to ‘3’ then follow next bullets otherwise go to Step 19 (i)
b. Save STemp in “Digital_I_O” variable
c. Shift STemp left 3 times to allign value to desired port pin
d. Mask all bits of STemp to 0 except Bits 3 to 4
e. Clr PortD3 and PortD4 (Digital I/O 2 and 1 control pins)
f. OR'ed PortD to the masked STemp byte
g. Transmit Execution response by sending characters ‘EI’
h. Return from serial interrupt routine
i. Call Transmit function to send ‘P’ as parameter error byte, Make frame = 0 for next command’s service and return from serial interrupt routine

Step 20: If command = ‘S’ then

a. If STmp = ‘0’ or ‘1’ then follow next bullets otherwise go to Step 20 (i)
b. Save STemp in “Signal” variable
c. Shift STemp left 3 times to allign value to desired port pin
d. Mask all bits of STemp to 0 except Bit-3
e. Clr PortC3 (Signal-source control pin)
f. OR'ed PortC to the masked STemp byte

g. Transmit Execution response by sending characters ‘ES’
h. Return from serial interrupt routine
i. Call Transmit function to send ‘P’ as parameter error byte, Make frame = 0 for next command’s service and return from serial interrupt routine

Step 21: If command = ‘T’ then

a. If STmp = ‘0’ to ‘3’ then follow next bullets otherwise go to Step 21 (h)
b. Save STemp in “Trigger” variable
c. Mask all bits of STemp to 0 except Bits 0 to 1
d. Clr PortC0 and PortC1 (Trigger O/P 2 and 1 control pins)
e. OR'ed PortC to the masked STemp byte
f. Transmit Execution response by sending characters ‘ET’
g. Return from serial interrupt routine
h. Call Transmit function to send ‘P’ as parameter error byte, Make frame = 0 for next command’s service and return from serial interrupt routine

Step 22: If command = ‘D’ then

a. Save STemp in DAC array variable at index F_Length
b. If F_Length = 0 then follow the bullets otherwise return from the serial interrupt routine
c. Set CPHA to 1 for DAC communication, as discussed above
d. Activate (make low) the DAC select bits, DAC_CSn and DAC_SDI_En
e. Wait for 20 µsec
f. Load SPDR with DAC’s data write writing byte (0x40)
g. Wait for SPIF bit in SPSR to set
h. Wait for 10 µsec
i. Call SPI_Write function with variables DAC and number of bytes 3 to send three DAC bytes to SPI port
j. Wait for 10 µsec
k. Deactivate (make high) the DAC select bits, DAC_CSn and DAC_SDI_En
l. Set CPHA to 0 for MUX relays
m. Transmit Execution response by sending characters ‘ED’
n. Return from serial interrupt routine

Step 23: If command = ‘M’ then

a. Save STemp in MUX array variable at index F_Length
b. If F_Length = 0 then follow the bullets otherwise return from the serial interrupt routine
c. Activate (make high) the MUX latch select bit
d. Call SPI_Write function with parameters MUX and number of bytes 5 to send five MUX relay bytes to SPI port
e. Deactivate (make low) the MUX latch select bit
f. Transmit Execution response by sending characters ‘EM’
g. Return from serial interrupt routine

Step 24: (frame must be 3) follow the steps to response the read command

a. Make frame = 0 for next command’s service
b. If STemp = ‘A’, Call transmit function three times to send ‘C’ (catch command), ‘A’ (Asymetric sub-command) and 1 byte’s Asym variable respectively and return from serial interrupt routine
c. If STemp = ‘C’, Call transmit function three times to send ‘C’ (catch command), ‘C’ (ADC Clock sub-command) and 1 byte’s Clk variable respectively and return from serial interrupt routine
d. If STemp = ‘G’, Call transmit function three times to send ‘C’ (catch command), ‘G’ (sinewave frequency generator range sub-command) and 1 byte’s Generator variable respectively and return from serial interrupt routine
e. If STemp = ‘I’, Call transmit function three times to send ‘C’ (catch command), ‘I’ (Digital I/O sub-command) and 1 byte’s Digital_I_O variable respectively and return from serial interrupt routine
f. If STemp = ‘S’, Call transmit function three times to send ‘C’ (catch command), ‘S’ (Signal source sub-command) and 1 byte’s Signal variable respectively and return from serial interrupt routine
g. If STemp = ‘T’, Call transmit function three times to send ‘C’ (catch command), ‘T’ (Trigger O/P sub-command) and 1 byte’s Trigger variable respectively and return from serial interrupt routine

h. If STemp = ‘D’, Call transmit function five times to send ‘C’ (catch command), ‘D’ (DAC sub-command) and 3 byte’s of DAC array variable (MSB first) respectively and return from serial interrupt routine

i. If STemp = ‘M’, Call transmit function seven times to send ‘C’ (catch command), ‘M’ (MUX relay sub-command) and 5 byte’s of MUX array variable (MSB first) respectively and return from serial interrupt routine

j. Otherwise call Transmit function to send ‘X’ as command/sub-command error byte, Make frame = 0 for next command’s service and return from serial interrupt routine

4.1.6 AVR code’s Serial Port Transmit function

This function is designed to send one byte without overwrite to the user through the serial port. This function has an input argument containing the byte to be sent.

Step 00: Copy passed variable to Trans variable

Step 01: Wait for USART’s data register empty bit to set

Step 02: Copy Trans to USART’s data register to send

Step 03: Return from the function

4.1.7 AVR code’s SPI write function

This function is designed to write a number of bytes of an indexed variable to the SPI port. This function has two input arguments containing the array variable and the number of bytes to be sent (starting from MSB).

Step 00: Copy passed variables to local variables ptr (indexed variable) and length (number of bytes)

Step 01: Make a local counter variable

Step 02: counter = length

Step 03: SPDR = ptr[counter] that is write variable’s indexe byte (starting from MSB) to SPI’s data register to send

Step 04: Wait for SPI’s writing interrupt flag bit to set

Step 05: Decrease counter variable by one

Step 06: Go to Step 03 if counter is greater than 0

Step 07: Return from the function

Chapter 05
PC Side Programming

PC side programming is done using MatLab. MatLab coding is similar to the C language. To facilitate the user, this terminal is based on GUI (Graphical User Interface) . The current version of the software is manually operated. Manually in the sense, that the user will select all the operations/options to be operated on manually selected channel(s) of the ADC card. An automatic test scheme is in progress which itself internally uses the manual test files to automatically apply test scheme on the channels of the ADC card.

MatLab based driver software is consists of two types of files,

1. Driver software's figure files
2. Driver software's M-code files

5.1 Driver software's figure files

GUI Window file made through MatLab is called GUI Figure file. For a GUI window, one has to design a figure file having all the panels, buttons popup menus etc. MatLab will generate an M-code file of the figure file having I/O functions of all the employed options. Now the user has the responsibility to write the appropriate code as desired. So a Matlab Gui's figure file is useless without an m-file. There are six Figure files to operate the software manually.

S. No.	File Name	Discription
01	Main	Starting GUI window
02	Hardware	Hardware setting window for the Test Card
03	Signal_response	Test card's operation window
04	Hardware1	Hardware setting window for the ADC Card
05	Config_ADC_Card	ADC card's operation window
06	ADC_Data	Efficient data plotting window for 36 channel mode

The highlighted one's are for the ADC card. The defination of these GUI windows can be seen in section 6.1 along with an example to understand easily.

5.2 Driver software's M-code files

The MatLab code file is called M-code file. Every GUI figure file has also an m-code file.
In designing and implementing this softwrae piority is given to facilitation for the operating user and ease of future upgrade for a programmer.
Many complex and big tasks are further performed through separate files. So the total m-files of the driver software are as follows,

S. No.	File Name	Discription
01	Main	MatLab code of main window
02	Hardware	MatLab code of hardware window
03	Signal_response	MatLab code of Signal_response window
04	Hardware1	MatLab code of hardware1 window
05	Config_ADC_Card	MatLab code of Config_ADC_Card window
06	ADC_Data	MatLab code of ADC_Data window
07	Read_Set_Current_Configuration	Read current configs of Test card
08	Set_Set_Default_Configuration	Set default configs of the test card
09	Set_MUX_Channels	Set channels at GUI (called by file 06)
10	Voltage_2_DAC_Conect_bytes.m	Convert Voltage to DAC bytes
11	DAC_Conect_bytes_2_Voltage	Convert DAC bytes to voltage
12	set_all_channels_2_GND	Set all channels at GUI to GND
13	set_all_channels_2_Source_Selected	Set all channels at GUI to selected source
14	Find_ADC_and_Ch_numbers	Find ADC and ADC channel for the selected channel
15	Find_Ch_4m_ADC_and_ch_no	Find the channel from from ADC and ADC channel
16	Read_Dis_Internal_Frequency	Read internal frequency of ADC card

17	**Read_Dis_Number_of_Measurements**	**Read number of measurements and display**
18	**Read_Display_Status**	**Read status register and display**
19	**Read_Set_Current_Configuration_ADC**	**Set default configs of the ADC card**
20	**Read_Display_ADC_Data**	**Read, display ADC data and save into MS-XL files**

The highlighted one's are for the ADC card.

The software reads the ADC data and separates it according to channels. The data received is in ASCII-hex format. So it is first converted to decimal then signed decimal (by taking additive inverse of 2's complement of the values greater than 32767). Then the corresponding voltages can be found by using the ADC calibration equation,

$$Data\_36\_Channels = (0.1526 * Data\_36\_Channels + 0.0509) * 1.0e-003$$

Equation 3

The program shows the voltage plot of single or 36 channels depending on the acquisition mode. Channel voltage output(s) is/are also saved in the current directory as sheet 2 of the file named "Single_Channel.xls" or "36_Channels.xls" depending on the acquisition mode. Sheet 1 of the corresponding files consist of the as it is received ascii-hex formatted data from ADC card.

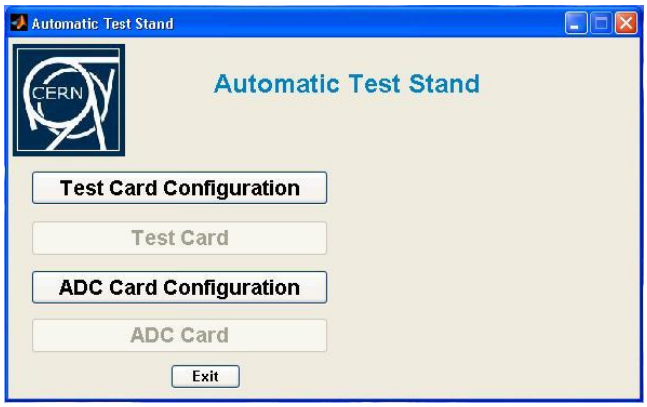
Chapter 06

User Manual

This chapter discusses how to use the driver software of ADC-test and ADC cards to test the functionality of the ADC card. This chapter flows the executing sequence of the software using an example.

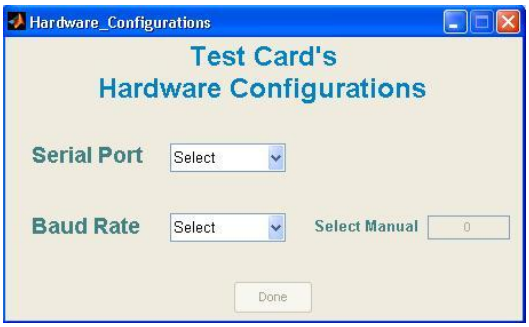
6.1 Software Execution (GUI Windows)

Main.exe is the executable file to start the software if a deployed stand alone package is available. To run the application in the MatLab environment, if MatLab is installed, set the code directory as current directory and just type "main" on MatLab's command window to start the application. The starting window will be

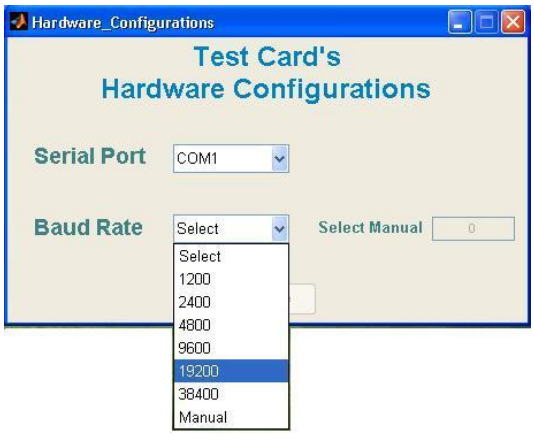


The enabled options are forcing the user to configure the hardware of the cards and check the connection first.

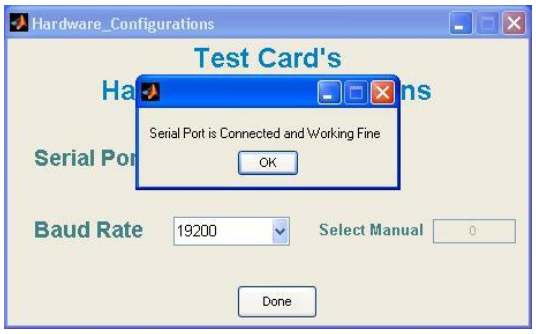
The *Test Card Configuration* option is opened as



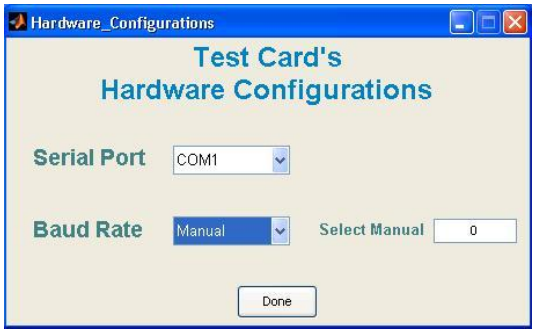
Set the COM port to which the Test card is connected. There are a number of standard baud rates available to be selected.



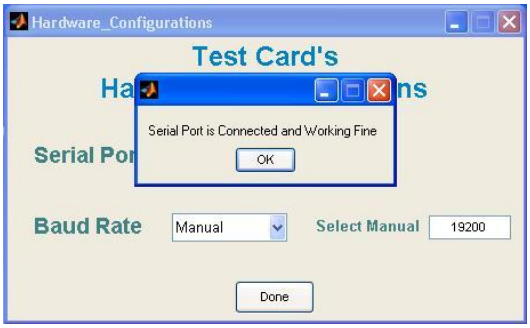
Test card's baud rate is fixed to *19200 bits per seconds*. Once the user selects the baud rate, the software sends the Hello command, see section 3.1, to check the connection. If a valid Hi byte is received it displays this good new as



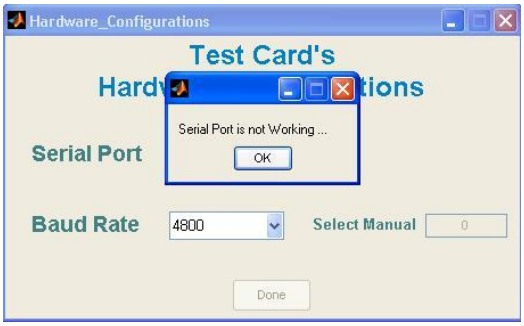
For a different baud rate, if changed through test card's AVR program, manual option enables the text window as



Now the user has to type the correct baud rate and press enter. The result will be similar if the serial communication is done correctly.

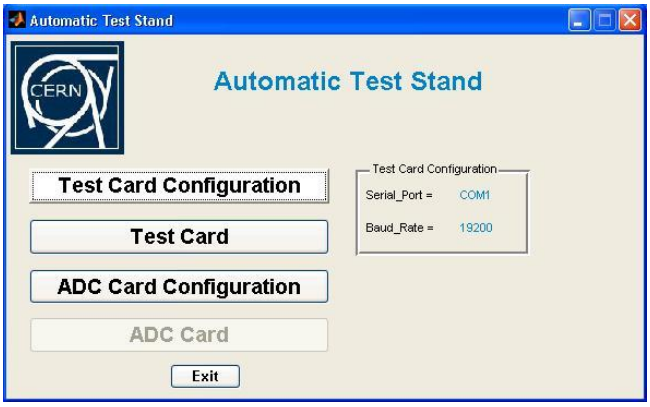


If the software was unable to receive the valid response of the Hello command, it will show an error message of this form,



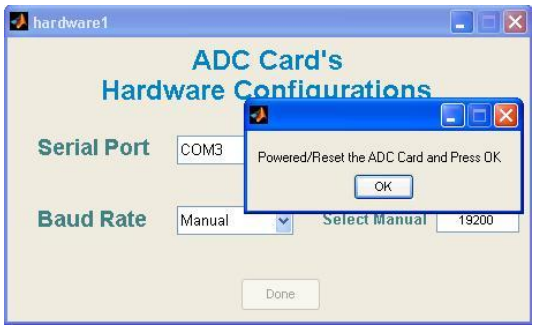
To get rid of this error, the user has to check the serial com-port, connections, baud rate, circuit power etc and select the baud rate again for a new attempt. In this case the communication baud rate was not matched.

Once hardware connection is done, the main window becomes,

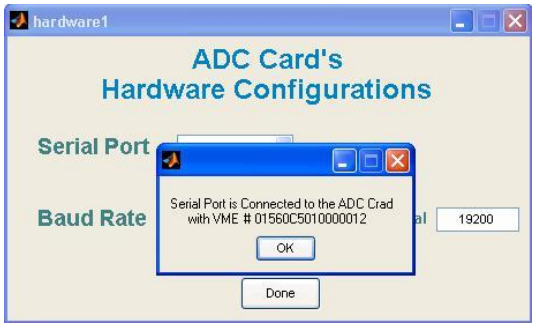


Test card operation window is enabled to be used. Now one can go to this or can select the configuration of the ADC card. But the proper data acquisition will only be done after the *ADC card configuration*.

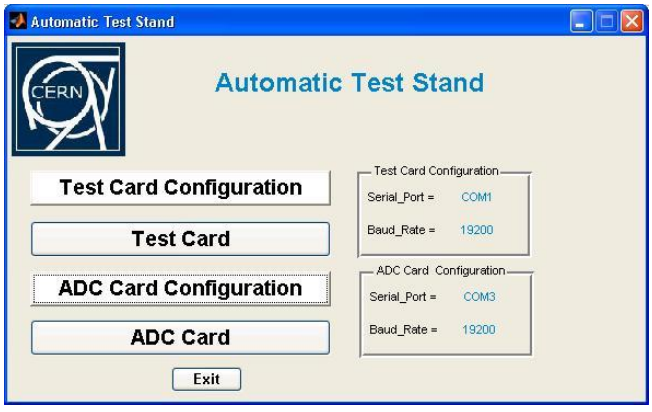
The ADC card's hardware configuration window looks similar to that of the test card. Once the user select the correct serial com-port and baud rate (ADC card has baud rate *19200 bps*) the message arrives,



It means if the ADC card is not powered yet then power it otherwise just press the reset button on the ADC card. This will enable the card to send the card introduction including VME (Virtual Machine Environment) number and effectively the software will match the programmed VME number to test the connection. The failed connection gives the same error message as in the case of test card connection (see above) and successful connection results

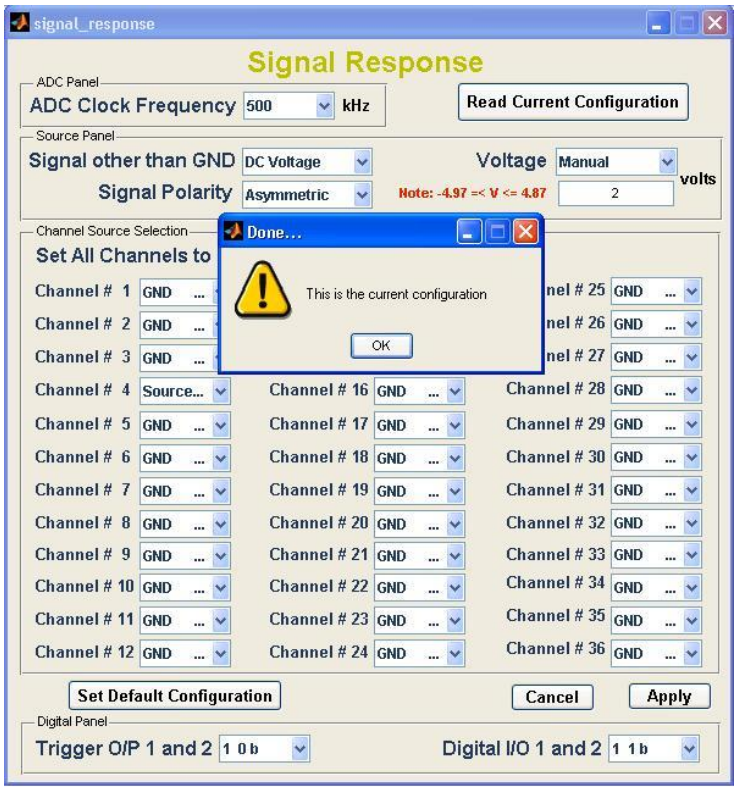


After the successful hardware connections of both the cards, the main window looks like,



The brief hardware configurations for both the cards is shown. Now the software is ready to operate on both cards properly.

Once the user presses the *Test Card* operation window button it opens a new window and scans all the currrent state of the test card,



Now the user can select the required options as,

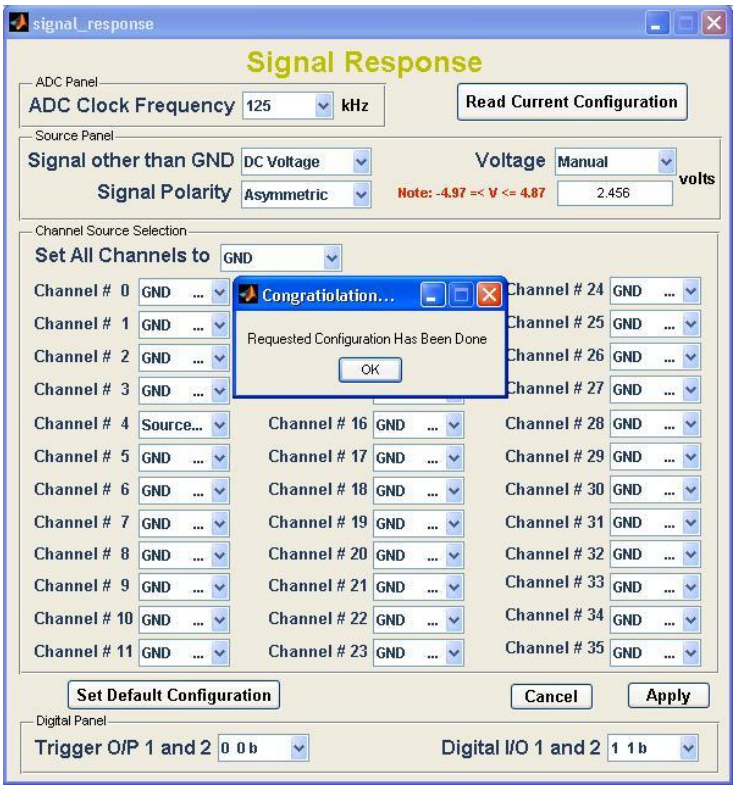
1. Sampling frequency for ADC.
2. Any one of the two available signals, other than Ground.
    i. Sine wave
    ii. DC Voltage
3. Selected signal's property
    i. One of the two frequencies of the sine wave, if selected 2(i).
    ii. Any voltage listed or typed manually within limits, if selected 2(ii).
4. Signal polarity (Asymetric or symetric).
5. Now all the channels can be connected to the selected source (sine wave or DC voltage) or ground by using the "Set All Channels to" option.
6. All channels are individually set to the required electrical signal (ground or the selected source) by using the corresponding "Channel #" option.
7. The currently set configuration can be read at any time by using the refered button.
8. Also the default settings can be applied at any time by using the corresponding button.

Lets take an example to set a required confguration.

For this example we want to select,

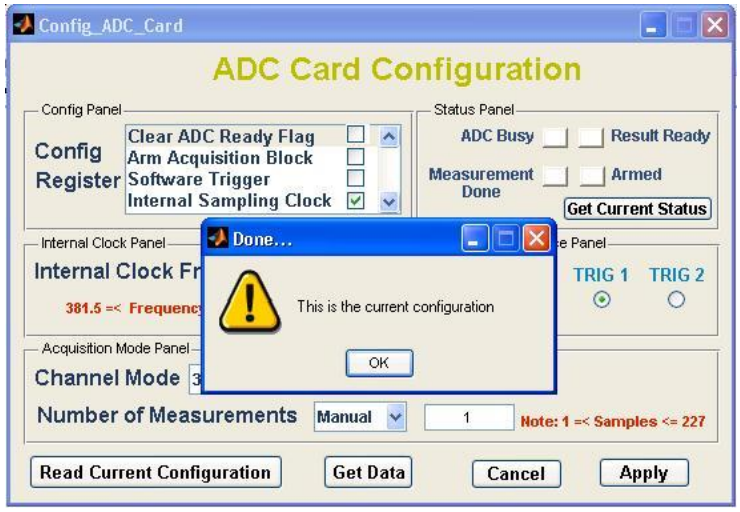
a. ADC frequency 125 KHz.
b. DC voltage signal other than Ground.
c. DC voltage is typed as 2.456V manually but within limits since the selection 2 is DC voltage.
d. Signal polarity as Asymetric
e. Set all channels to ground.
f. Channel 4 (counting starts from 0) is connected to the selected source (DC voltage here).

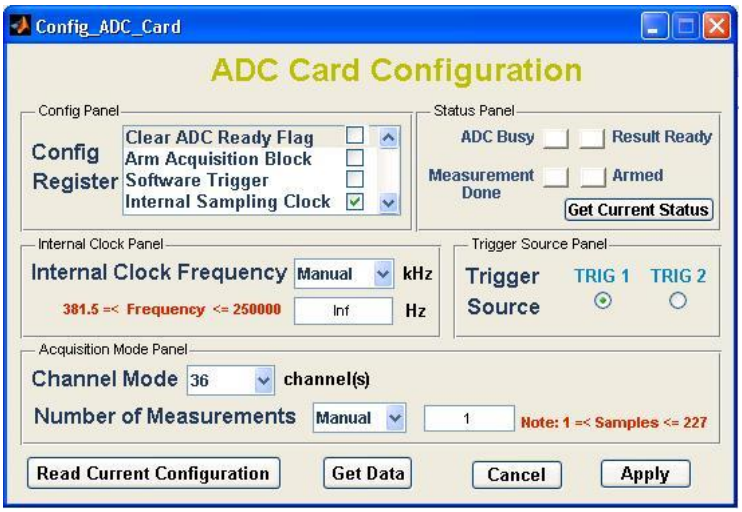
Apply Button will send this new seting to the test card and successful communication will be like this



Now the channels are showing the electrical signals at the output connectors, connected to the ADC card. Note that the Trigger outputs and Digital I/Os are sent as selected, they do not wait for the Apply button to be pressed.

It’s time to initialize the ADC card in single channel acquisition mode. To do so the main window’s *ADC card* button will open the window and displays the current setting of the ADC card after scanning as





Don't be afraid of infinite frequency. It is because the timer clock register of ADC card has zero value which according to the formula (Sample Timer = $25x10^{06}$ / Internal Frequency) makes it divide by zero.

Now the user can select the desired configuration for the required acquisition. The following features are available,

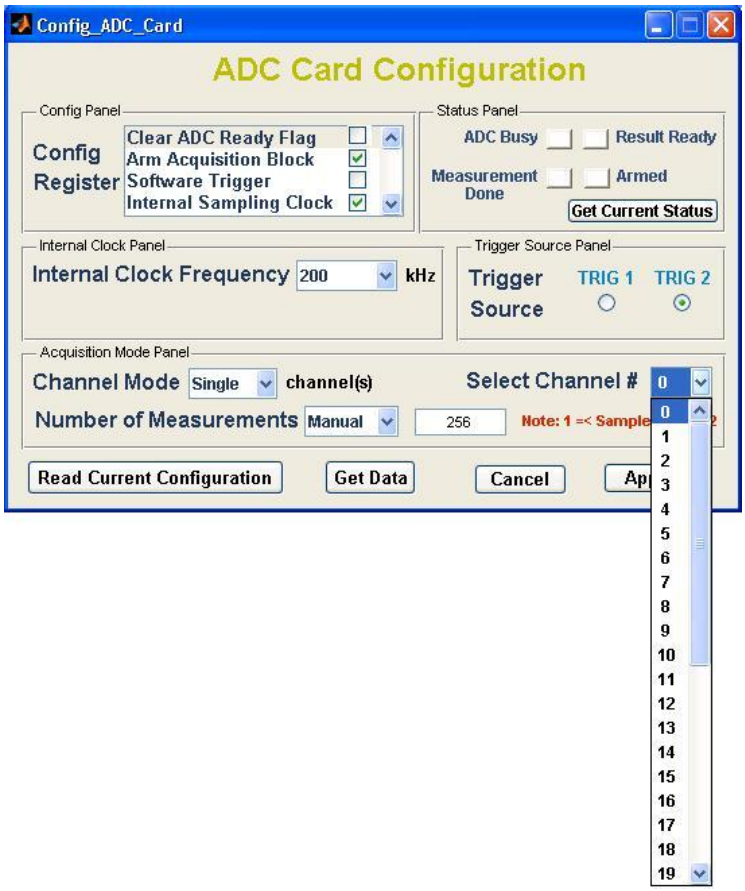
1. Clear ADC flag
2. Arm Acquisition Block
3. Software Triggering
4. Internal ADC Sampling Clock
5. Acquisition Mode
   i. 36 channels
   ii. Single channel (then select the desired channel(s))

6. Number of Measurements
7. Get current status
8. Currently set configuration can be read at any time by using the referred button.

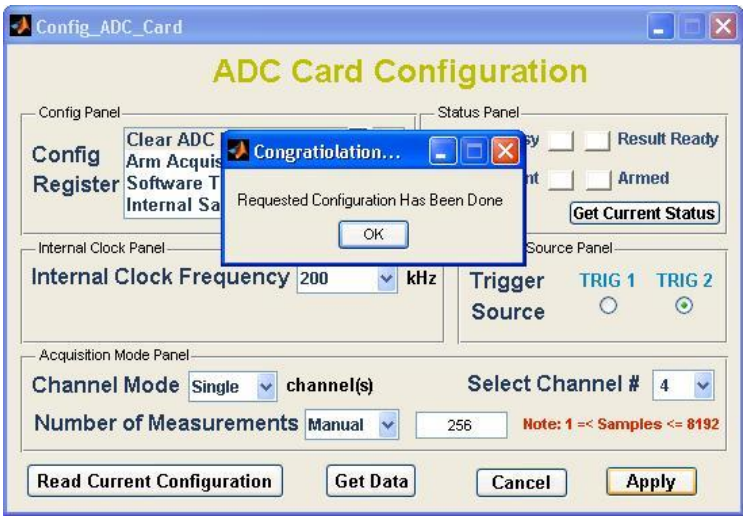
6.1.1 Single Channel Acquisition

Lets sample channel number 4 only, 256 times by the using internal sample clock at 200ksps.

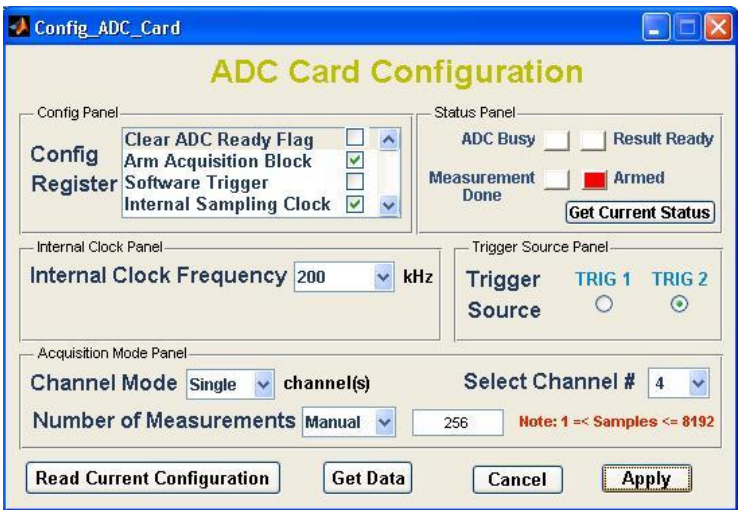
For this, arm the acquisition block, desired internal sampling frequency 200Ksps is selected as shown (the external frequency, selected through test Card will be discarded). Single channel mode is useful to sample a single channel (channel 4 here). One can select other channel or all channels as can be seen from the "Select Channel #" popup menu.



After all setting the selections the *Apply* button will send these configurations to Card. The successful configuration response will be



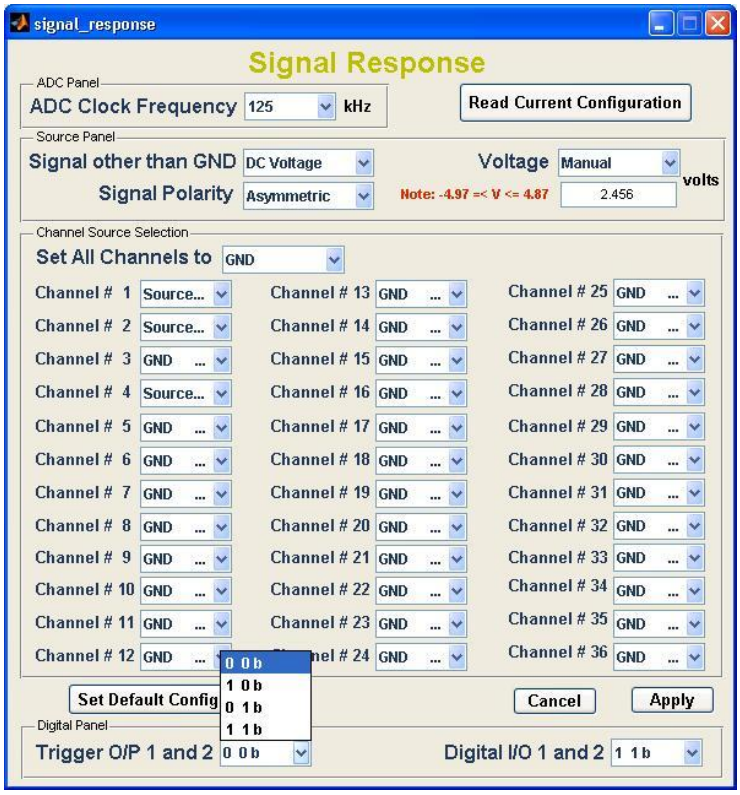
Card's status will be displayed after pressing the Ok button, as



Now the required 256 samples of channel number 4 with sample frequency 200 ksps is just a trigger away. The card can be triggered by software or hardware.

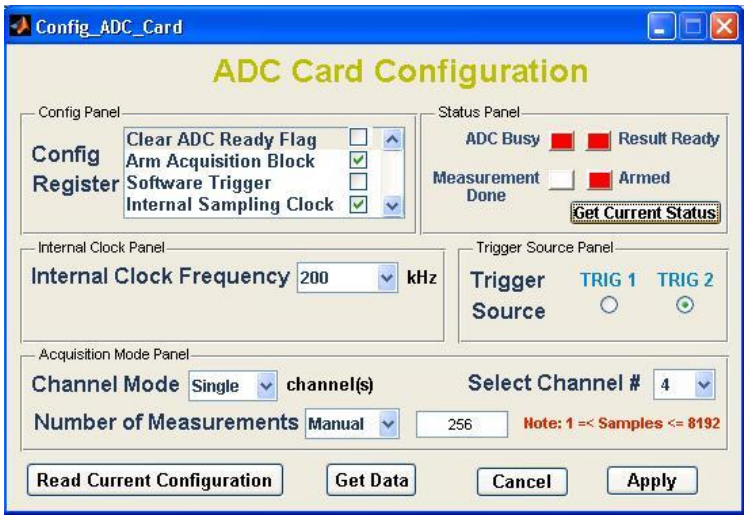
1. Software triggering requires checking of the corresponding check box in ADC Card's and pressing Apply button.
2. Hardware Triggering requires a high level at the selected Trigger source pin (Trig1 or Trig2). In our case we can do this by using "Triger O/P 1 & 2" option of the test Card's window.

Let's trigger by hardware as

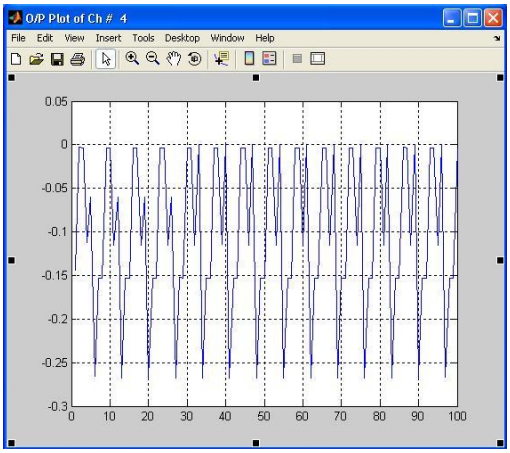


Once you select the proper logic of the Trigger it will be sent.

The ADC card's current status can be seen by pressing the "Get Current Status" button. Once the required sampling is done, one will get



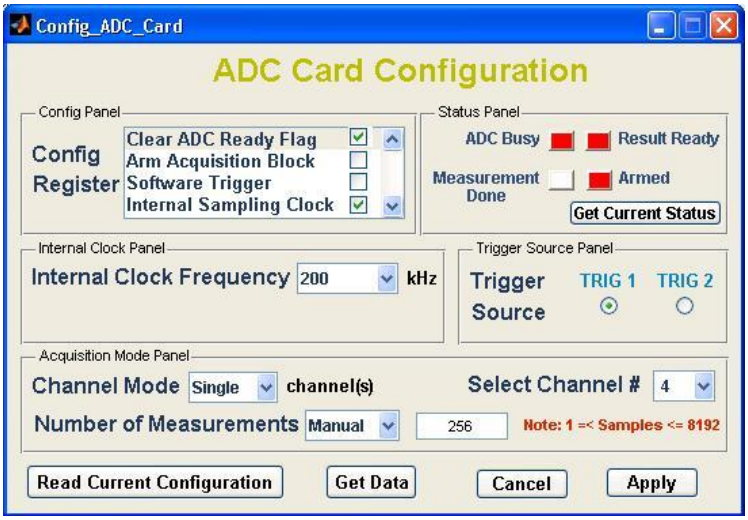
Result ready shows that the data is ready to be read from the ADC card's buffers. The "Get Data" button will let the user read, display (output and FFT of the output) plot and save the received (raw and calibrated) data in a file (Single_Channel.Xls). The output plot will look like this



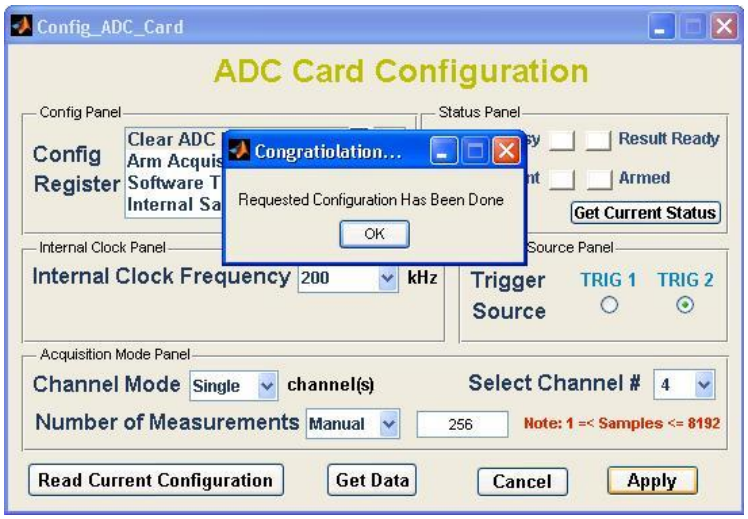
FFT is not suitable for this case because of too few number of samples. For a good FFT plot, the maximum number of samples (8192 here) is recommended. In the presence of 36 channel mode the time taking "All" channel sampling in "Single" channel mode is introduced just because of this fact.

6.1.1.1 Resetting of ADC Card for Next Mode/Channel

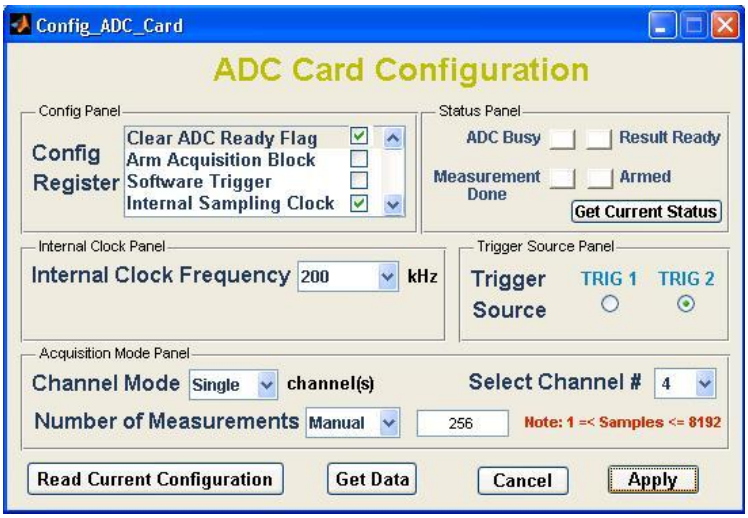
Next mode/channel acquisition requires the reset (clear flag and arm acquisition) of the ADC card. This can be done by checking the "Clear ADC Ready Flag" and unchecking of "Arm Acquisition Block" check boxes as



When applied,



Pressing ok will let you see the current status



Hence the card is ready to be configures in any other mode. Remember to clear the chek box of "Clear ADC ReadyFlag" during the next acquisition settings.

6.1.2 All 36 channels Acquisition

All the 36 channels can be sampled in two ways,

1. Using "36 channel" mode of the ADC Card
2. Selecting "All" channels as the selected channel in "Single Channel" mode

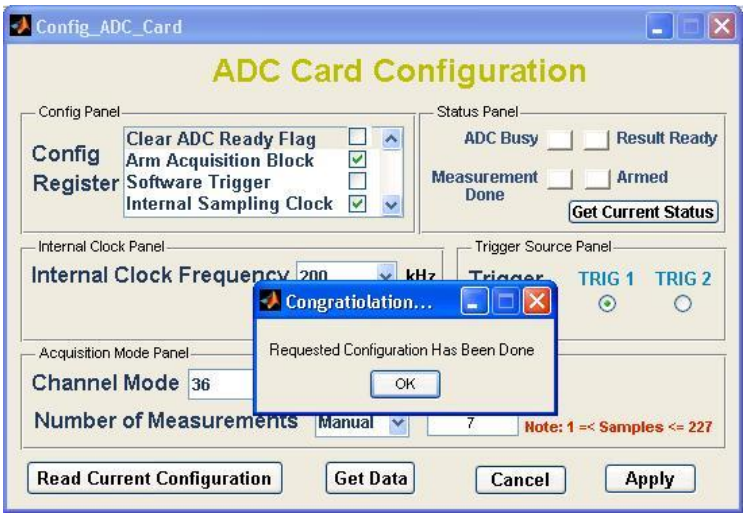
The difference between these two methods is the number of measurements of each channel. The 36 channel mode allows only 227 (8192/36) samples of each channel. Where as the single channel mode allows 8192 samples of the selected channel. So by means of software, all channels can be sampled one by one in single channel mode to effectively acquire 8192 samples of each channels.

If only a dc level is of interest then the 36 channel mode is suitable but in case of certain tests where large number of samples are necessary, like FFT, "All" channel selection of the single channel mode is suitable.

6.1.2.1 Using "36 channel" mode of the ADC Card

As in the case of single channel mode, the same settings can be selected for 36 channel mode but number of measurements is limited.

Let's take 7 samples of all 36 channels with internal clock frequency of 200 ksps and external trigger 2. One can set these options and get the following window after applying.



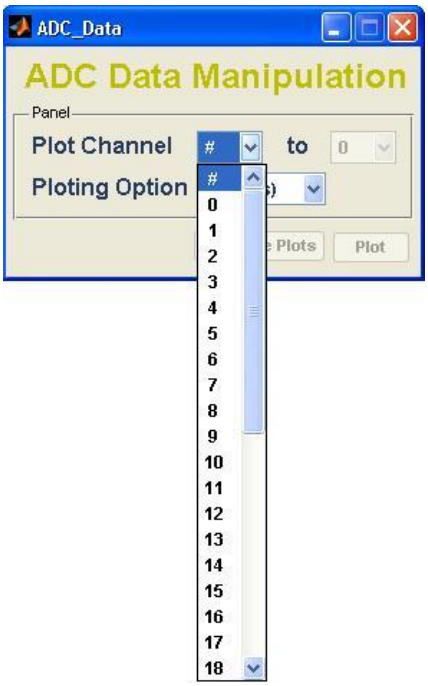
Remember that the test card should be set accordingly to apply the desired signals on the channels before triggering.

Now that the card is initialized in this mode we are just a trigger away from sampling. Triggering can be done as we did in single channel mode above, see Figure _____. Verify the sampling status before getting the data by using the "Get Current Status" button and wait for the similar status as of Figure _____ above.

After verification the "Get Data" button will get, calibrate and save the data and open a new window of following shape for plotting.

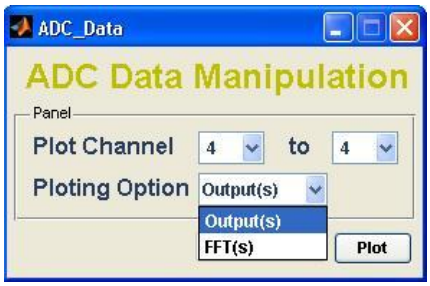
This window is made to plot the output and/or its Fourier transform calculated through a FFT algorithm of a single or up to four above signals on a single plot or on separate plots. Output and/or FFT output of all channels can be plotted on separate plots (4 figures having 9 plots each). Let's see how it can be done;

One has to select the start channel number from '0' to '35' or 'All'.

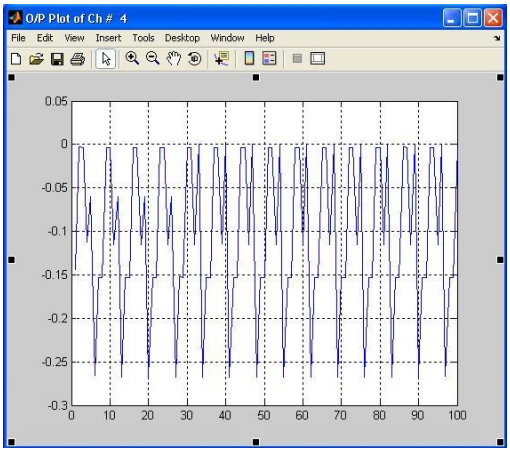


Let's select channel # 4 as the start channel,

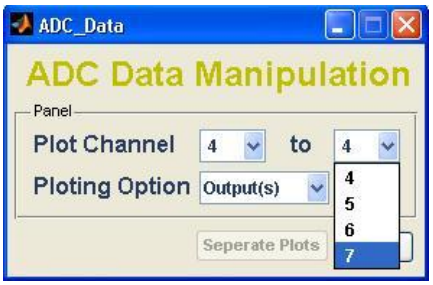
As can be seen that the plotting end channel is selected same as well. If single channel, channel 4 here, is required to be plotted then its ok. Now the plotting option can be switched, as shown below, if FFT plot is required.

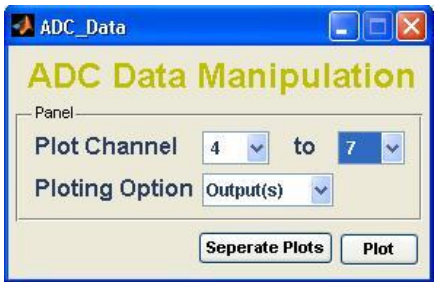


Plot button will be pressed to get the channel # 4's plot, output plot at this time, as

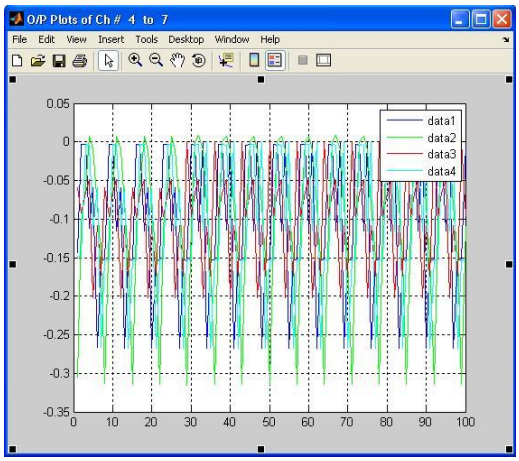


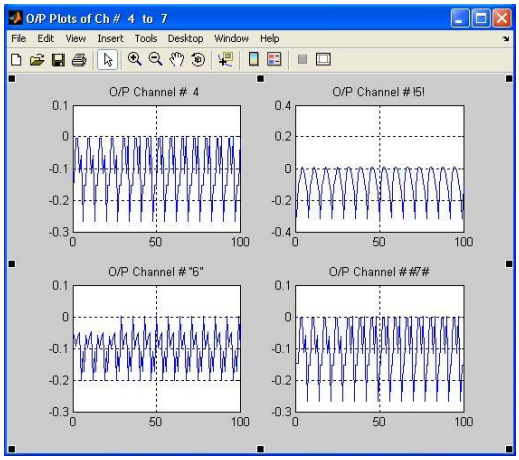
If comparision with near channels is required then select one of the four subsequent end channels,



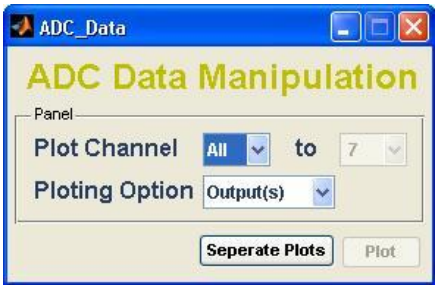


And the outputs and/or FFT-outputs of channel # 4 to 7 can be plotted on a single figure by using "Plot" button and/or seperately by using "Separate Plots" button. The two figures will be like these,

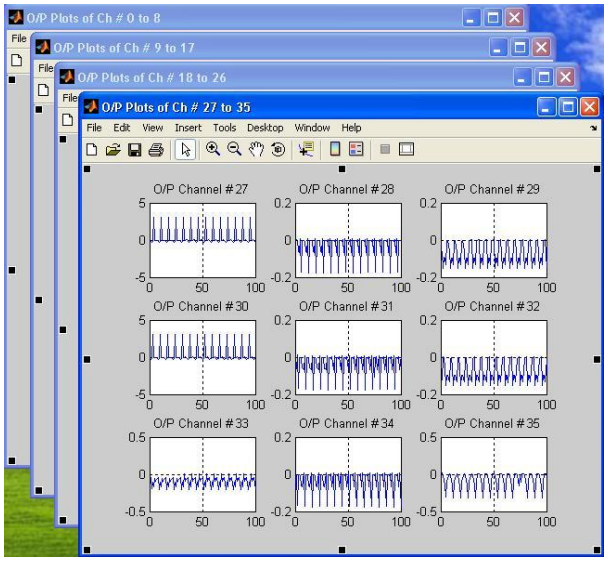




All channels can be plotted by using the option ‘All’ in the start channel popup menue as



And the plotting option can be switched to FFT if output’s FFT is required. The plot (separate in all channel case) of outputs of all 36 channls will be like,



Hope this is easy to be understand and feasible for a user to work with.