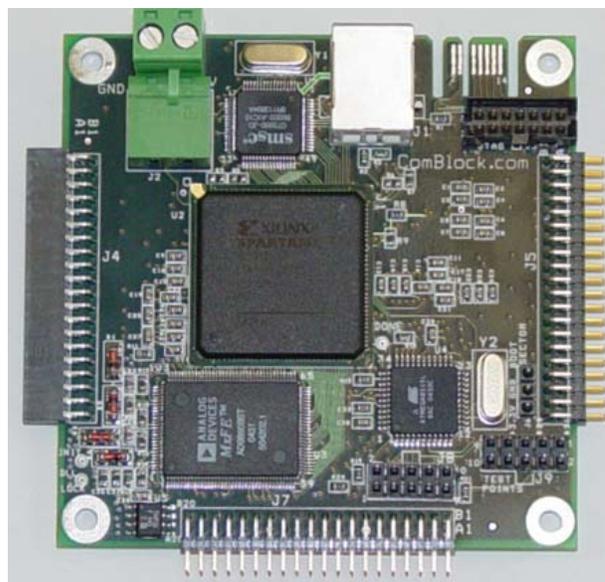


### Key Features

- Channel emulator with analog input and output.
- Typical applications:
  - Power line channel emulator
  - Wireless channel emulator.
- Sampling rate: 64 Msamples/s.
- Multi-path:
  - Up to 40 paths
  - each path is modeled as a delay (0 to 1023 samples) and a 18-bit precision amplitude scaling coefficient.
- Additive signal from optional arbitrary waveform generator.
-  **ComScope** –enabled: key internal signals can be captured in real-time and displayed on host computer.
- Connectorized 3”x 3” module for ease of prototyping. Standard 40 pin 2mm dual row connectors (left, right, bottom). Single 5V supply with reverse voltage and overvoltage protection. Interfaces with 3.3V LVTTTL logic.

For the latest data sheet, please refer to the **ComBlock** web site: [www.comblock.com/download/com1232.pdf](http://www.comblock.com/download/com1232.pdf). These specifications are subject to change without notice.

For an up-to-date list of **ComBlock** modules, please refer to [www.comblock.com/product\\_list.htm](http://www.comblock.com/product_list.htm).

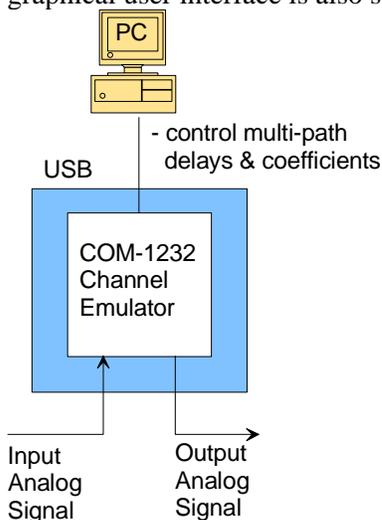


COM-1232

### Typical Configurations

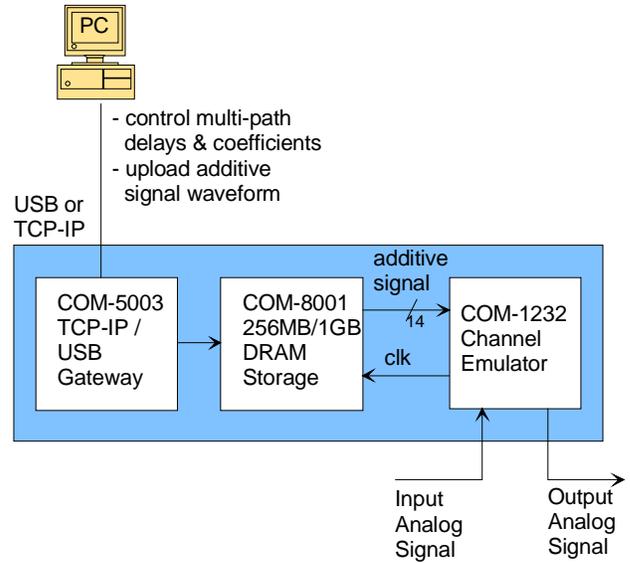
#### Stand-Alone

The channel emulator 40 delays and 40 coefficients are programmed in real-time over a USB connection from a host computer. Best real-time performances are obtained when the parameters are set by custom software programs (see [template](#)). A graphical user interface is also supplied.

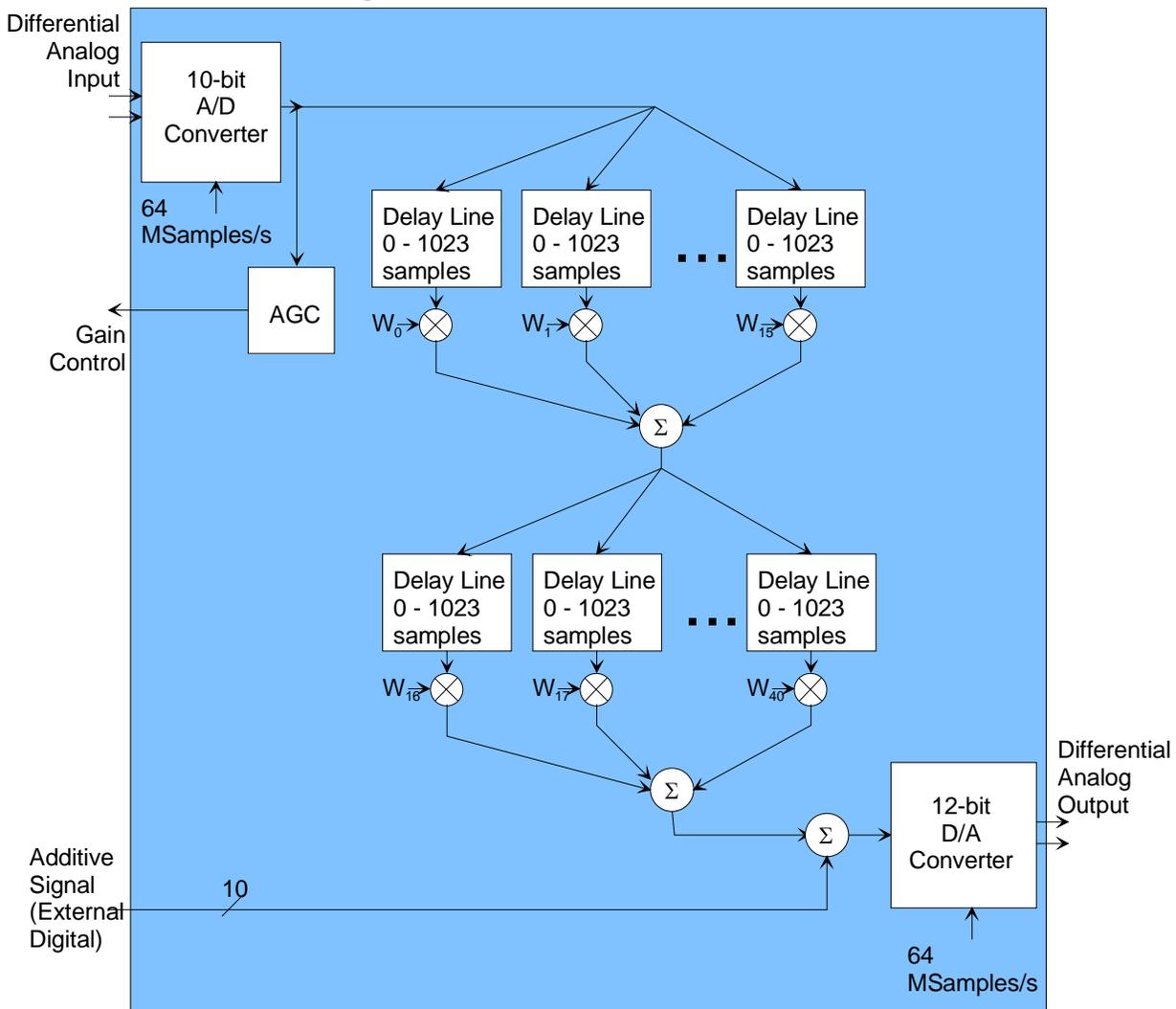


## Multi-Path & Additive Waveform

A more complex channel emulator can be assembled using the [COM-8001](#) arbitrary waveform generator. Large files representing 10-bit precision analog sampled signals can be uploaded through the COM-5003 to the COM-8001 SDRAM memory, then played back at the selected speed (up to 64 Msamples/s). The resulting signal is added to the COM-1232 output.



## Functional Block Diagram



## Electrical Interface

Analog Input Interfaces (J7)	Definition
RX_P / RX_N	Differential analog inputs. (_P for +, _N for -). 200 Ohm input impedance. 2Vpp differential (1Vpp on each RX_P and RX_N signal) for full scale 10-bit ADC conversion. Common-mode voltage is approximately 2.3V. It is recommended that the input be AC coupled.
RX_AGC1	Output. When this channel emulator is connected directly to an analog receiver, it generates an analog 0 – 3.3V signal to control the analog gain prior to A/D conversion. The purpose is to use the maximum dynamic range while preventing saturation at the A/D converter. 0 is the maximum gain, +3.3V is the minimum gain. Pin J7/A6.
External Digital Input (J4)	Definition
DATA_P1_IN[9:0] DATA_P2_IN[9:0]	Real input signal to be added to the emulator output. For speed reasons, two input samples are supplied in parallel. 10-bit precision, unsigned format. Bit 9 is the most significant bit. The data source is expected to send DATA_Px_IN at the falling edge of CLK_IN while this module will sample it at the rising edge.
CLK_IN	Input synchronous clock. Read the two samples DATA_P1_IN and DATA_P2_IN at the rising edge of CLK_IN. Maximum frequency 40 MHz (i.e. 80 MSamples/s).
TRIGGER_IN	Input pulse indicating the first sample from a COM-8001 arbitrary waveform generator. Useful as trigger to ComScope.

SAMPLE_CLK_REQ_OUT	Flow control output. Requests samples from an external data source.
Analog Output Interfaces (J7)	Definition
TX_P / TX_N	Differential outputs. (_P for +, _N for -). Full range 2Vpp differential (1Vpp on each TX_P and TX_N signal). Common mode voltage is approximately 1V. Output impedance 100 Ohm.

Monitoring & Control	Definition
USB 2.0	Type B receptacle. This interface is used only for monitoring and control. Use USB 2.0 approved cable for connection to a host computer. Maximum recommended cable length is 3'.
Power Interface	4.75 – 5.25VDC. Terminal block. Power consumption is approximately proportional to the symbol clock rate ( $f_{\text{symbol\_clk}}$ ). The maximum power consumption is 650mA.

**Important: Digital I/O signals are 0-3.3V LVTTTL. Inputs are NOT 5V tolerant!**

## Configuration

Complete assemblies can be monitored and controlled centrally over a single USB, or, when available through adjacent ComBlocks, LAN/TCP-IP, asynchronous serial, or CardBus connection.

Users can access the full set of configuration features by specifying 8-bit control registers as listed below. In general, the control registers are to be set manually through the ComBlock Control Center or by software using the ComBlock API (see [www.comblock.com/download/M&C\\_reference.pdf](http://www.comblock.com/download/M&C_reference.pdf)). In the specific case of the COM-1232, the multi-paths delays and amplitude scaling coefficients are expected to be controlled dynamically through a user-developed custom application program. A [C-language code template](#) is provided to help developers in this task.

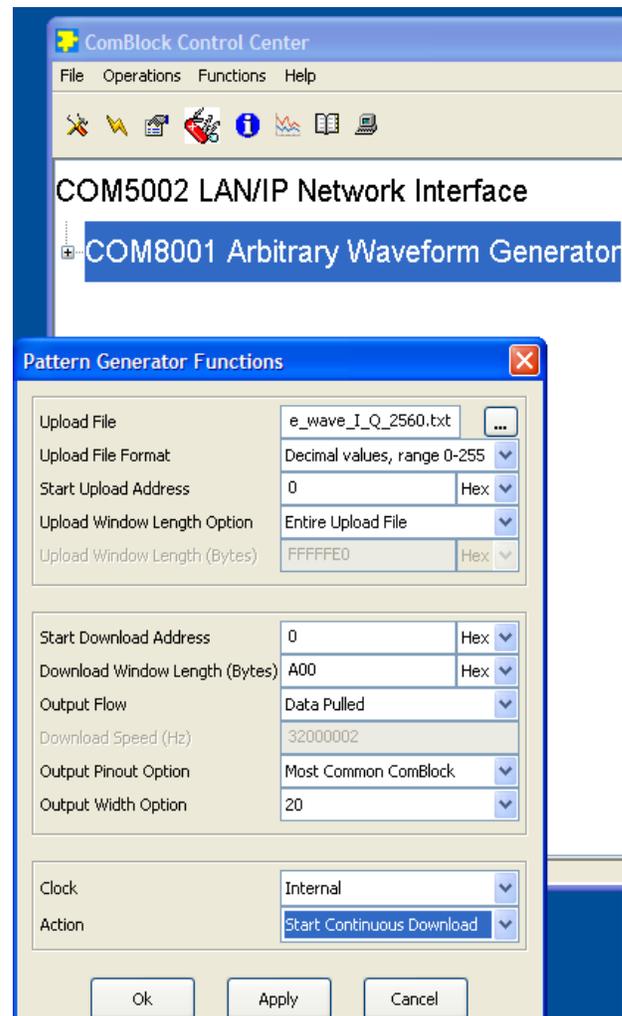
Control registers REG0 through REG3 are read/write. Their contents is stored in non-volatile

memory. Control registers REG4 through REG173 are write-only and stored in volatile memory.

Undefined control registers or register bits are for backward software compatibility and/or future use. They are ignored in the current firmware version.

Inputs	
Parameters	Configuration
Rx ADC gain	The analog signal prior to the built-in A/D converter can be amplified by steps of about 1 dB. This 5-bit unsigned integer controls the variable gain between 0 and 20 dB. REG0 = bits 4-0
<a href="#">AGC1</a> response time	The front-end AGC1 response time is user controlled. The RX_AGC1 analog gain control signal is updated as follows 0 = every sample, 1 = every 2 samples, 2 = every 4 samples, 3 = every 8 samples, etc.... 20 = every 1 million samples. Valid range 0 to 20. REG1 bits 4-0
AGC1 enabled	Enable or disable the automatic gain control for an external analog receiver. 0 = fixed at a preset level (see REG2) 1 = enabled REG1 bit 7
Fixed gain	Gain settings for an external analog receiver. This setting is used when the AGC1 must be disabled (for example during receiver level measurements). Unsigned 8-bit number. REG2 bits 7-0.
External additive input	Enable (1) or disable (0) the external additive input signal. REG3 bit 0
Channel Emulator	
Parameters	Configuration
Coefficient $W_i$	Signed (2's complement) 18-bit precision coefficient. 40 coefficients are referred to by their index $i$ in the range 0 to 39.  The amplitude scaling coefficient $W_i$ are expressed as a numerical value in 1.17 <a href="#">fractional binary format</a> , signed (meaning one sign bit and 17 bits following the decimal point). Unit gain is 0x1FFFF.  Unlike the other ComBlock configuration registers, the coefficients values are not stored in non-volatile memory. API users should therefore use the "SRT" Set Register Temporary command. At power up, the coefficients are zero.

	REG <sub>10+4*i</sub> = $W_i$ (7:0) REG <sub>11+4*i</sub> = $W_i$ (15:8) REG <sub>12+4*i</sub> (1:0) = $W_i$ (17:16)
Delay $D_i$	Delay expressed as number of samples. Valid range 0 – 1023 samples. Index $i$ is in the range 0 to 39. Unlike the other ComBlock configuration registers, the delays values are not stored in non-volatile memory. API users should therefore use the "SRT" Set Register Temporary command. At power up, the delays are zero. REG <sub>12+4*i</sub> (7:2) = $D_i$ (5:0) REG <sub>13+4*i</sub> (3:0) = $D_i$ (9:6)



*Graphical user interface example for the 500x-8001-1232 assembly*

## ComScope Monitoring

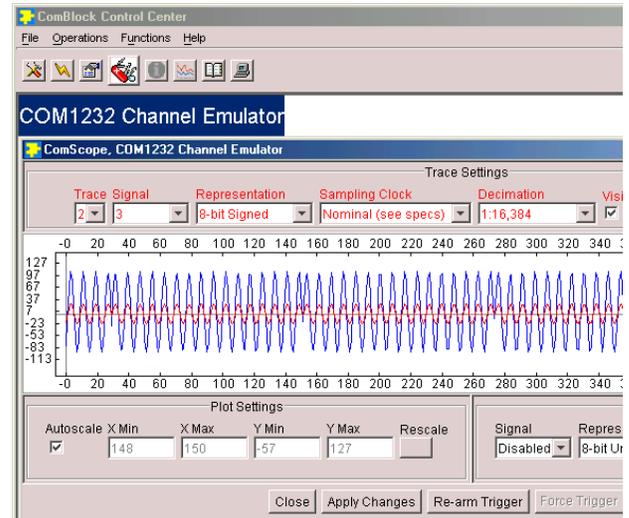
Key internal signals can be captured in real-time and displayed on a host computer using the ComScope feature of the ComBlock Control Center. The COM-1232 signal traces and trigger are defined as follows:

Trace 1 signals	Format	Nominal sampling rate	Buffer length (samples)
1: Input signal from ADC	8-bit signed (8MSB/10)	$f_{clk}$	512
2: Input signal from left digital connector	8-bit signed (8MSB/10)	$f_{clk}$	512
Trace 2 signals	Format	Nominal sampling rate	Capture length (samples)
1: Output signal prior to DAC	8-bit signed (8MSB/12)	$f_{clk}$	512
2: front-end AGC RX_AGC1	8-bit unsigned (8MSB/10)	Decimated sampling rate. See <a href="#">AGC1</a>	512
3: Intermediate multi-path signal (after 16 paths)	8-bit signed (8MSB/18)	$f_{clk}$	512
Trigger Signal	Format		
1: Start of arbitrary waveform sequence (TRIGGER_IN)	Binary		

Signals sampling rates can be changed under software control by adjusting the decimation factor and/or selecting the  $f_{clk}$  processing clock as real-time sampling clock.

In particular, selecting the  $f_{clk}$  processing clock as real-time sampling clock allows one to have the same time-scale for all signals.

The ComScope user manual is available at [www.comblock.com/download/comscope.pdf](http://www.comblock.com/download/comscope.pdf).



*ComScope Window Sample*

## Digital Test Points

Test points are provided for easy access by an oscilloscope probe.

Digital Test Point	Definition
TP1	Overflow detected while summing the first 16 paths.
TP2	Overflow detected while summing the last 24 paths.
TP3	Overflow detected while adding the external signal to the multi-path output..
TP4	Future use
TP5	Future use
TP6	Future use
TP7	Future use
TP8	Future use
TP9	Future use
TP10	Future use
DONE	'1' indicates proper FPGA configuration.
INITB	Reference clock $f_{clk} / 8 = 8$ MHz.

## Operation

### Additive External Signal

An external signal can optionally be added to the channel output prior to digital-to-analog conversion (see [block diagram](#)). This external signal is real, sampled at 64 Msamples/s with 10-bit precision in unsigned format. It is received exclusively through the left J4 input connector.

The most flexible manner for generating such additive signal is to use the COM-8001 arbitrary waveform generator. Several other ComBlock modules, in particular most ComBlock modulators, are also compatible with this interface.

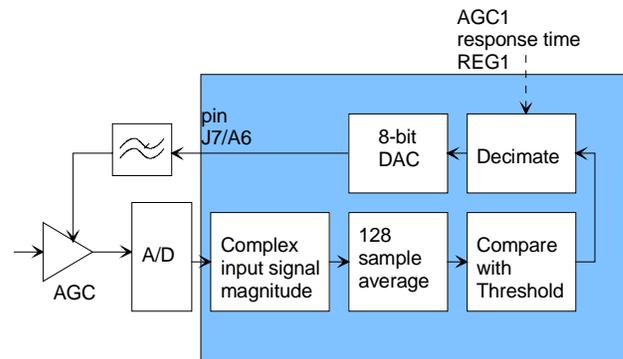
### AGC1

Prior to being routed to the channel emulator, the input signal is subject to automatic gain control.

The purpose of this AGC is to prevent saturation at the input signal A/D converters while making full use of the A/D converters dynamic range. The AGC can be enabled or set at a fixed gain under software control.

The principle of operations is outlined below:

- (a) The magnitude of the complex input samples is computed and continuously averaged over 128 samples.
- (b) The average magnitude is compared with a target magnitude threshold and the AGC gain is adjusted accordingly. Users can control the rate at which the gain control value is updated (to prevent instabilities, depending on the gain control slope and linearity at the RF front-end). See control register REG1.
- (c) An 8-bit D/A converter generates the analog gain control signal RX\_AG1 for use by the external variable gain amplifiers.



**AGC1 principle**

### Fractional Representation

Throughout this document, key signals are described in fractional binary format denoted by  $x.y$ . The total number of bits is  $x+y$ . The number of bits representing the numerical value below the decimal point is  $y$ .  $x$  denotes the number of bits representing the numerical value above the decimal point, including one bit for the sign in the case of signed values.

Examples:

Format	Fractional representation	Decimal Equivalent
1.17 signed	0.1000000000000000	0.5
1.17 signed	1.1000000000000000	-0.5

### Recovery

This module is protected against corruption by an invalid FPGA configuration file (during firmware upgrade for example) or an invalid user configuration. To recover from such occurrence, connect the BOOT pin to the nearby ground pin using a jumper and power-up the module. Remove the jumper after 3 seconds. The module will be automatically configured with a boot configuration which restores communications. This boot file is un-erasable. Once this is done, the user can safely restore the user configuration and/or re-load a valid FPGA configuration file into flash memory using the ComBlock Control Center.

## Programming Template #1

A C-language template for programming the multi-paths delays and amplitude scaling coefficients is shown below. The connection between the host computer and the ComBlock assembly is assumed to be over TCP/IP/LAN.

```
/* -----
template for configuring the COM-1232 40 paths parameters
delay (0-1023 samples) and 18-bit amplitude scaling coefficient.

The assembly comprises three ComBlocks:
COM-5003 LAN/TCP-IP interface
COM-8001 Arbitrary waveform generator
COM-1232 Channel emulator

Operation:
Connect the COM-5003 (running option -B) to the LAN.
Power up the assembly then run this program.
The ComBlock Control Center cannot be used at the same time as
both programs use the same IP port on the ComBlock.
-----*/

#include <stdio.h>
#include "winsock2.h"
#define MAX_MESSAGE_LENGTH 273
short send_command(SOCKET, char*);

void main() {

    char sendbuf[MAX_MESSAGE_LENGTH];
    long W0, D0;
    unsigned char a;
    long total_bytes_sent = 0;

    // Initialize Winsock.
    WSADATA wsaData;
    int iResult = WSASStartup( MAKEWORD(2,2), &wsaData );
    if ( iResult != NO_ERROR )
        printf("Error at WSASStartup()\n");

    // Create a socket.
    SOCKET m_socket;
    m_socket = socket( AF_INET, SOCK_STREAM, IPPROTO_TCP );

    if ( m_socket == INVALID_SOCKET ) {
        printf( "Error at socket(): %ld\n", WSAGetLastError() );
        WSACleanup();
        return;
    }

    // Connect to a server.
    sockaddr_in clientService;

    clientService.sin_family = AF_INET;

    // INSERT THE COM-5003 IP ADDRESS BELOW
    clientService.sin_addr.s_addr = inet_addr( "172.16.1.130" );
    // port 1028 is reserved for ComBlock monitoring and control.
    clientService.sin_port = htons(1028);

    if ( connect( m_socket, (SOCKADDR*)&clientService, sizeof(clientService) ) == SOCKET_ERROR ) {
        printf( "Failed to connect.\n" );
        WSACleanup();
        return;
    }

    // Assign ComBlock IDs after power up.
    sprintf(sendbuf, "@000SAC001\r\n");
    send_command(m_socket, sendbuf);
    // assign ID 001 to the first module
}
```

```

printf(sendbuf, "@001MFW9\r\n");
send_command(m_socket, sendbuf);
    // forward messages to all ports

printf(sendbuf, "@000SAC002\r\n");
send_command(m_socket, sendbuf);
    // assign ID 002 to the first module

printf(sendbuf, "@002MFW9\r\n");
send_command(m_socket, sendbuf);
    // forward messages to all ports

printf(sendbuf, "@000SAC003\r\n");
send_command(m_socket, sendbuf);
    // assign ID 003 to the first module

// Set Path amplitude scaling coefficient W0 and delay D0
W0 = 0x000102; // 18-bit signed. format 1.17
D0 = 512;      // Delay
a = (unsigned char)(W0 & 0x000000FF);
printf(sendbuf, "@003SRT10%02X\r\n", a);
send_command(m_socket, sendbuf);

a = (unsigned char)((W0>>8) & 0x000000FF);
printf(sendbuf, "@003SRT11%02X\r\n", a);
send_command(m_socket, sendbuf);

a = (unsigned char)((W0>>16) & 0x00000003);
a |= (unsigned char)((D0 << 2) * 0x000000FC);
printf(sendbuf, "@003SRT12%02X\r\n", a);
send_command(m_socket, sendbuf);

a = (unsigned char)((D0 >> 6) * 0x000000FF);
printf(sendbuf, "@003SRT13%02X\r\n", a);
send_command(m_socket, sendbuf);

// close socket
closesocket(m_socket);

return;
}

/* send command actually comprises three transactions:
1) sending the original command
2) sending a dummy query
3) waiting for the dummy query's response
This is an indirect way to ensure that the rate at which commands are sent are neither
too slow nor too fast.
*/
short send_command(SOCKET m_socket, char* command){
    short message_length;
    char dummy_query[MAX_MESSAGE_LENGTH];
    char dummy_reply[MAX_MESSAGE_LENGTH];
    short total_bytes_sent = 0;
    short total_bytes_received = 0;

    // send command. Make sure all bytes are sent.
    message_length = strlen(command);
    total_bytes_sent = 0;
    while(total_bytes_sent != message_length){
        total_bytes_sent += send( m_socket, &command[total_bytes_sent], message_length-
total_bytes_sent, 0 );
    }
    Sleep(5);    // wait 5 ms before next command

    // dummy command
    sprintf(dummy_query, "@001GRG00\r\n");
    // send command. Make sure all bytes are sent.
    message_length = strlen(dummy_query);
    total_bytes_sent = 0;
    while(total_bytes_sent != message_length){
        total bytes sent += send( m socket, &dummy query[total bytes sent], message length-

```

```

total_bytes_sent, 0 );
    }

    // wait for dummy command's response
    // response is 11-byte long.
    total_bytes_received = 0;
    while(total_bytes_received < 11){
        total_bytes_received += recv( m_socket, &dummy_reply[total_bytes_received], 11-
total_bytes_received, 0 );
    }

    return strlen(dummy_reply);
}

```

Note: More information about Winsock programming can be found at [http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winsock/winsock/finished\\_server\\_and\\_client\\_code.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winsock/winsock/finished_server_and_client_code.asp) . Be sure to include a reference to the Winsock2 library (WS2\_32.lib) in the project release and/or debug settings.

## Programming Template #2

A C-language template for generating binary files to load into the arbitrary waveform generator is shown below. The main objective of this template code is to describe how to format (i.e. pack) 10-bit precision unsigned samples into a byte-oriented binary file.

```

// com8001.cpp : Generates a (large) binary file for testing the COM-8001
// arbitrary waveform generator. The file consists of a real sinewave.
// Samples are unsigned, 10-bit precision, for interface
// with the COM-1232 channel emulator module.
// Change f (sinewave frequency), ilmax (number of samples),
// then re-compile.

#include <math.h>
#include <stdio.h>

// Declare a look-up table to "flip bitwise" 10-bit values (for example,
// 0101100001 will be become 1000011010 after a bitwise flip).
const unsigned short FLIP_VALUES_10_BITS [] =
{0,512,256,768,128,640,384,896,64,576,320,832,192,704,448,960,32,544,288,800,160,
672,416,928,96,608,352,864,224,736,480,992,16,528,272,784,144,656,400,912,80,592,
336,848,208,720,464,976,48,560,304,816,176,688,432,944,112,624,368,880,240,752,496,
1008,8,520,264,776,136,648,392,904,72,584,328,840,200,712,456,968,40,552,296,808,
168,680,424,936,104,616,360,872,232,744,488,1000,24,536,280,792,152,664,408,920,
88,600,344,856,216,728,472,984,56,568,312,824,184,696,440,952,120,632,376,888,248,
760,504,1016,4,516,260,772,132,644,388,900,68,580,324,836,196,708,452,964,36,548,
292,804,164,676,420,932,100,612,356,868,228,740,484,996,20,532,276,788,148,660,
404,916,84,596,340,852,212,724,468,980,52,564,308,820,180,692,436,948,116,628,372,
884,244,756,500,1012,12,524,268,780,140,652,396,908,76,588,332,844,204,716,460,972,
44,556,300,812,172,684,428,940,108,620,364,876,236,748,492,1004,28,540,284,796,
156,668,412,924,92,604,348,860,220,732,476,988,60,572,316,828,188,700,444,956,124,
636,380,892,252,764,508,1020,2,514,258,770,130,642,386,898,66,578,322,834,194,706,
450,962,34,546,290,802,162,674,418,930,98,610,354,866,226,738,482,994,18,530,274,
786,146,658,402,914,82,594,338,850,210,722,466,978,50,562,306,818,178,690,434,946,
114,626,370,882,242,754,498,1010,10,522,266,778,138,650,394,906,74,586,330,842,202,
714,458,970,42,554,298,810,170,682,426,938,106,618,362,874,234,746,490,1002,26,538,
282,794,154,666,410,922,90,602,346,858,218,730,474,986,58,570,314,826,186,698,442,
954,122,634,378,890,250,762,506,1018,6,518,262,774,134,646,390,902,70,582,326,838,
198,710,454,966,38,550,294,806,166,678,422,934,102,614,358,870,230,742,486,998,22,
534,278,790,150,662,406,918,86,598,342,854,214,726,470,982,54,566,310,822,182,694,
438,950,118,630,374,886,246,758,502,1014,14,526,270,782,142,654,398,910,78,590,334,
846,206,718,462,974,46,558,302,814,174,686,430,942,110,622,366,878,238,750,494,

```

1006,30,542,286,798,158,670,414,926,94,606,350,862,222,734,478,990,62,574,318,830,  
190,702,446,958,126,638,382,894,254,766,510,1022,1,513,257,769,129,641,385,897,65,  
577,321,833,193,705,449,961,33,545,289,801,161,673,417,929,97,609,353,865,225,737,  
481,993,17,529,273,785,145,657,401,913,81,593,337,849,209,721,465,977,49,561,305,  
817,177,689,433,945,113,625,369,881,241,753,497,1009,9,521,265,777,137,649,393,905,  
73,585,329,841,201,713,457,969,41,553,297,809,169,681,425,937,105,617,361,873,233,  
745,489,1001,25,537,281,793,153,665,409,921,89,601,345,857,217,729,473,985,57,569,  
313,825,185,697,441,953,121,633,377,889,249,761,505,1017,5,517,261,773,133,645,389,  
901,69,581,325,837,197,709,453,965,37,549,293,805,165,677,421,933,101,613,357,869,  
229,741,485,997,21,533,277,789,149,661,405,917,85,597,341,853,213,725,469,981,53,  
565,309,821,181,693,437,949,117,629,373,885,245,757,501,1013,13,525,269,781,141,  
653,397,909,77,589,333,845,205,717,461,973,45,557,301,813,173,685,429,941,109,621,  
365,877,237,749,493,1005,29,541,285,797,157,669,413,925,93,605,349,861,221,733,477,  
989,61,573,317,829,189,701,445,957,125,637,381,893,253,765,509,1021,3,515,259,771,  
131,643,387,899,67,579,323,835,195,707,451,963,35,547,291,803,163,675,419,931,99,  
611,355,867,227,739,483,995,19,531,275,787,147,659,403,915,83,595,339,851,211,723,  
467,979,51,563,307,819,179,691,435,947,115,627,371,883,243,755,499,1011,11,523,267,  
779,139,651,395,907,75,587,331,843,203,715,459,971,43,555,299,811,171,683,427,939,  
107,619,363,875,235,747,491,1003,27,539,283,795,155,667,411,923,91,603,347,859,219,  
731,475,987,59,571,315,827,187,699,443,955,123,635,379,891,251,763,507,1019,7,519,  
263,775,135,647,391,903,71,583,327,839,199,711,455,967,39,551,295,807,167,679,423,  
935,103,615,359,871,231,743,487,999,23,535,279,791,151,663,407,919,87,599,343,855,  
215,727,471,983,55,567,311,823,183,695,439,951,119,631,375,887,247,759,503,1015,15,  
527,271,783,143,655,399,911,79,591,335,847,207,719,463,975,47,559,303,815,175,687,  
431,943,111,623,367,879,239,751,495,1007,31,543,287,799,159,671,415,927,95,607,351,  
863,223,735,479,991,63,575,319,831,191,703,447,959,127,639,383,895,255,767,511,1023};

```
int main(int argc, char* argv[])
{
    long il,ilmax;
    double f,pi,fs;
    unsigned short rsample1, rsample2, rsample3, rsample4;
    unsigned short rsample1_inv, rsample2_inv, rsample3_inv, rsample4_inv;
    FILE *stream;
    unsigned char fivebytes[5];
    short numwritten;

    ilmax = 32*100000;           // size of file in sample size. Always a multiple of 32
    pi = 4.0 * atan(1.0);      // define pi
    fs = 64000000.0;          // sampling rate in Hz: 64 MSamples/s
    f = 5000000.;             // sinewave frequency in Hz

    printf("Generating test file for the COM-8001 -> COM-1232\n");

    if((stream = fopen("binary_file.bin","w+b")) == NULL)
        printf("the file 'binary_file.dat' was not opened\n");

    // Waveform generation

    for(il = 0; il<ilmax; il += 4){

        // generate four 10-bit real samples and format (pack) into 5 bytes before
        // storing into a file

        rsample1 = (unsigned short)(sin(2.0*pi*f*il/fs) * 511.0 + 511.5);
        rsample2 = (unsigned short)(sin(2.0*pi*f*(il+1)/fs) * 511.0 + 511.5);
        rsample3 = (unsigned short)(sin(2.0*pi*f*(il+2)/fs) * 511.0 + 511.5);
        rsample4 = (unsigned short)(sin(2.0*pi*f*(il+3)/fs) * 511.0 + 511.5);
```

```

// range 0.5 - 1023.5 (10-bit) unsigned

// invert 10-bits MSB <-> LSB
rsample1_inv = FLIP_VALUES_10_BITS[rsample1];
rsample2_inv = FLIP_VALUES_10_BITS[rsample2];
rsample3_inv = FLIP_VALUES_10_BITS[rsample3];
rsample4_inv = FLIP_VALUES_10_BITS[rsample4];

fivebytes[0] = (unsigned char)((rsample2_inv >> 2) & 0x00FF);
fivebytes[1] = (unsigned char)((rsample2_inv << 6) & 0x00C0)
| (unsigned char)((rsample1_inv >> 4) & 0x003F);
fivebytes[2] = (unsigned char)((rsample1_inv << 4) & 0x00F0)
| (unsigned char)((rsample4_inv >> 6) & 0x000F) ;
fivebytes[3] = (unsigned char)((rsample4_inv << 2) & 0x00FC)
| (unsigned char)((rsample3_inv >> 8) & 0x0003);
fivebytes[4] = (unsigned char)((rsample3_inv) & 0x00FF);

numwritten = fwrite(&fivebytes,1,5,stream);

}

fclose(stream);
// Format conversion

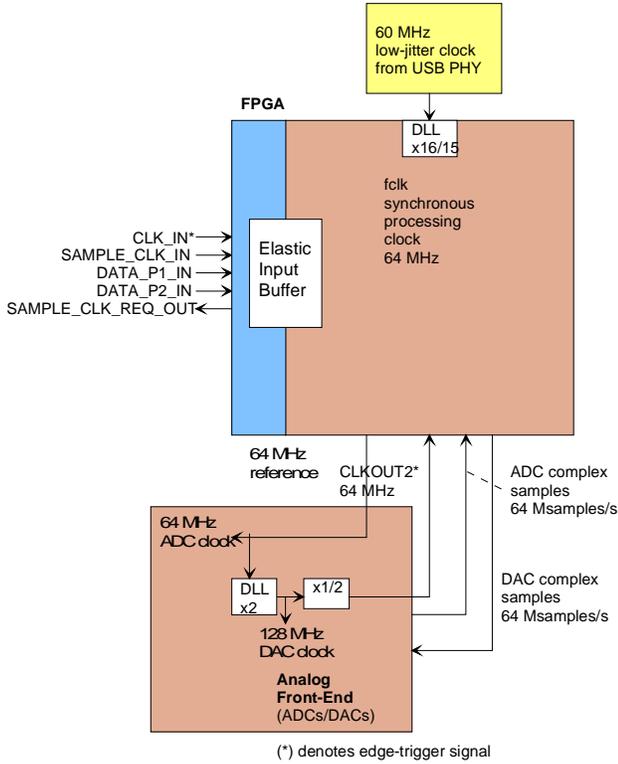
return 0;
}

```

# Timing

## Clocks

The clock distribution scheme embodied in the COM-1232 is illustrated below.



### Baseline clock architecture

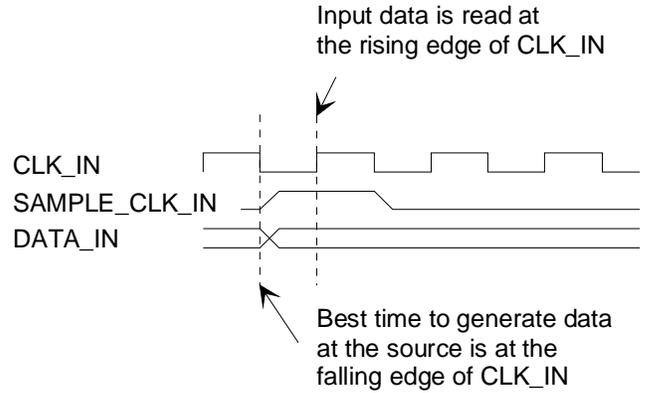
**Yellow = 60 MHz reference clock**

**Blue = 40 MHz arbitrary waveform gen. clock**

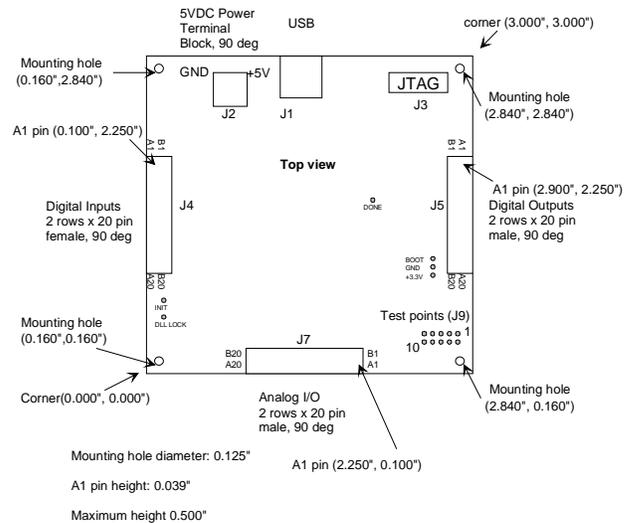
**Brown =  $f_{clk}$  64 MHz processing zone**

The core signal processing performed within the FPGA is synchronous with the processing clock  $f_{clk}$ . In order to minimize clock jitter, the processing clock is derived from a 60 MHz reference clock with low-jitter.  $f_{clk}$  is used for internal processing, for generating the ADC and DAC sampling clocks and for the external input additive signal.

# Input

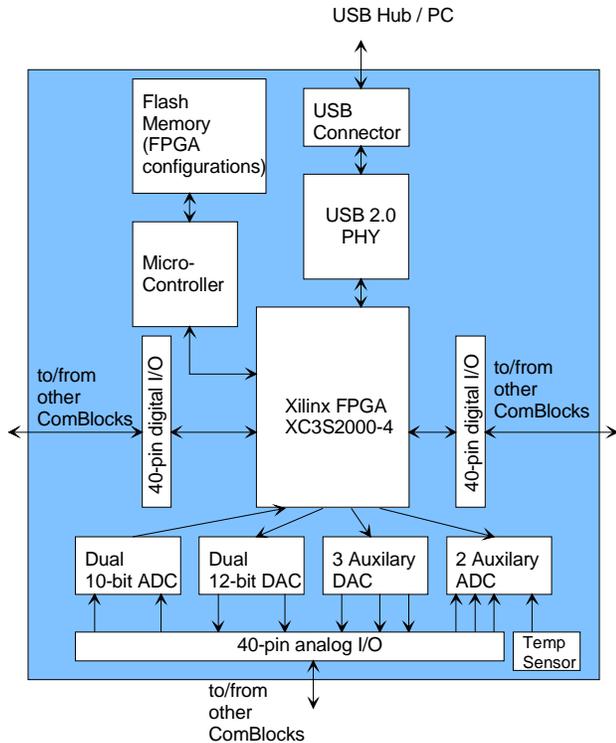


# Mechanical Interface



COM-1232

## Schematics



**Hardware Block Diagram**

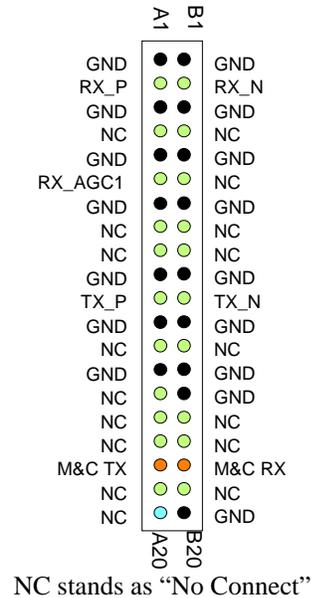
The board schematics are available on the ComBlock CD-ROM supplied with the module and on-line at [http://www.comblock.com/download/com\\_1200schematics.zip](http://www.comblock.com/download/com_1200schematics.zip)

## Pinout

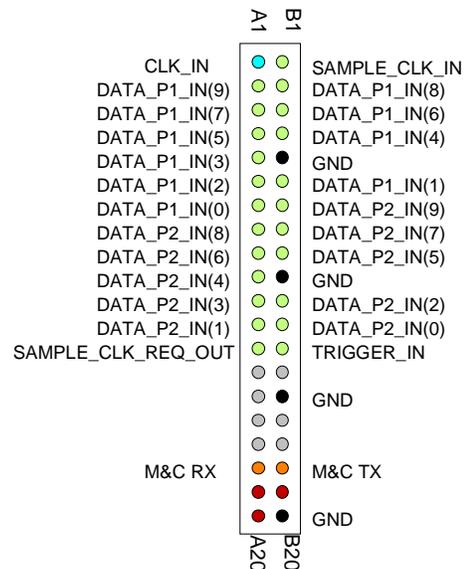
### USB

USB type B receptacle, as the COM-1232 is a USB device.

## Analog I/O Connector J7



## Input Connector J4



## I/O Compatibility List

(not an exhaustive list)

Input
<a href="#">COM-8001</a> Arbitrary Waveform Generator

## Configuration Management

This specification is to be used in conjunction with VHDL software revision 1.

## ***ComBlock Ordering Information***

COM-1232 Channel Emulator

MSS • 18221 Flower Hill Way #A •  
Gaithersburg, Maryland 20879 • U.S.A.  
Telephone: (240) 631-1111  
Facsimile: (240) 631-1676  
E-mail: [sales@comblock.com](mailto:sales@comblock.com)