Analog I/O Wildcard User Guide

Version 1.1
September, 2002

Analog I/O Wildcard User Guide

The Analog I/O Wildcard provides eight 12-bit digital to analog (DAC) outputs and eight 16-bit analog to digital (A/D) inputs. This tiny 2" by 2.5" module is a member of the Wildcard™ series that connects to the QED Board or PanelTouch Controller host via the Wildcard Carrier Board, or connects directly to the EtherSmart™ Controller.

This document describes the capabilities of the Analog I/O Wildcard, tells how to configure the hardware, and presents an overview of the driver software. A glossary of the software functions and complete hardware schematics are also included.

Analog I/O Wildcard Hardware

The Analog I/O Wildcard comprises a Wildcard bus header, field header, digital logic circuitry, an octal 12-bit digital to analog converter (DAC), an octal 16-bit analog to digital converter (A/D), and a 4.096 volt reference. The 4.096 reference voltage varies less than 100 microvolts per degree Celsius change in temperature. Jumpers enable module address selection and reference voltage selection among 5V, 4.096V, the DAC reference voltages (1.024 or 2.048 V), or an external reference voltage. The Wildcard bus header interfaces to the host processor (QED Board, Panel-Touch Controller, or EtherSmart Controller), and the field header brings out the analog I/O signals for the reference, DAC, and A/D. Specifications are summarized in Table 1-1.

Table 1-1 Analog I/O Wildcard Specifications

Analog Inputs	
Input Channels	8 unipolar single-ended, or 4 unipolar differential
Resolution	16-bits (0 – 65,535 counts)
Input Filtering	Land patterns are provided for optional input RC filters
Input Voltage Range	+IN: -0.2 V to 5.2 V -IN: -0.2 V to 1.25 V
Full Scale Differential Voltage	Jumper selectable full scale reference: 1.024 V, 2.048 V, 4.096 V, 5.0 V, or external reference.
Excitation	Jumper selectable output excitation reference is provided of: 1.024 V, 2.048 V, 4.096 V, or 5.0 V.
NonLinearity	Integral: ± 8 LSB max, ± 3 LSB typ; Differential: ± 1 LSB typ
Noise and Accuracy	20 μV rms effective input noise; 14.4 bits effective resolution
Sample Rate	Up to 17k samples per second

Analog Outputs	
Output Channels	8
Resolution	12-bits (0 – 4095 counts)
Output Filtering	Land patterns are provided for optional output RC filters
Full Scale Voltage	Jumper selectable: 2.048 V, 4.096 V, or 2x external reference; 4.6 V max.
Settling Time	1 μsec typically, slew rate is typically 10V/μsec
Load Impedance	Capable of driving 2 kΩ minimum resistance, 100 pF maximum capacitance, see data sheet for load regulation.
NonLinearity	Integral: ± 2 LSB typ Differential: ± 0.5 LSB typ
Update Rate	Up to 15k samples per second

Connecting To the Wildcard Carrier Board

To connect the Analog I/O Wildcard to the Wildcard Carrier Board, follow these simple steps:

1. Connect the Wildcard Carrier Board (also known as the "Module Carrier Board") to the QED Board as outlined in the "Wildcard Carrier Board User Guide".

2. With the power off, connect the 24-pin Module Bus on the Analog I/O Wildcard to Module Port 0 or Module Port 1 on the Wildcard Carrier Board. The corner mounting holes on the module should line up with the standoffs on the Wildcard Carrier Board. The module ports are labeled on the silkscreen of the Wildcard Carrier Board. Note that the Analog I/O Wildcard headers are configured to allow direct stacking onto the Wildcard Carrier Board, even if other modules are also installed. Moreover, the latest version of the Wildcard Carrier Board is designed to directly stack onto the QED Board. Do not use ribbon cables to connect the Analog I/O Wildcard to the Wildcard Carrier Board. Use of ribbon cables on the Analog I/O Wildcard's field header is fine.

   CAUTION: The Wildcard Carrier Board does not have keyed connectors. Be sure to insert the module so that all pins are connected. The Wildcard Carrier Board and the Analog I/O Wildcard can be permanently damaged if the connection is done incorrectly.

Selecting the Module Address

Once you have connected the Analog I/O Wildcard to the Wildcard Carrier Board, you must set the address of the module using jumper shunts across J1 and J2.

The Module Select Jumpers, labeled J1 and J2, select a 2-bit code that sets a unique address on the module port of the Wildcard Carrier Board. Each module port on the Wildcard Carrier Board accommodates up to 4 modules. Module Port 0 provides access to modules 0-3 while Module Port 1 provides access to modules 4-7. Two modules on the same port cannot have the same address (jumper settings). Table 1-2 shows the possible jumper settings and the corresponding addresses.

Table 1-2 Jumper Settings and Associated Addresses

Module Port	Module Address	Installed Jumper Shunts
0	0	None
	1	J1
	2	J2
	3	J1 and J2
1	4	None
	5	J1
	6	J2
	7	J1 and J2

Selecting the Reference Voltage

The remaining jumpers on the Analog I/O Wildcard select the reference voltage for both the 12-bit DAC and the 16-bit A/D. These jumpers, referred to as the voltage reference selection jumpers, are labeled J3, J4, J5, and J6 and named “DAC Ref”, “4V Ref”, “5V Ref”, and “Ref In/Out” respectively.

In most applications the DAC uses its own internal reference, and the jumpers J3, J4, and J5 select one of 3 reference voltages for the A/D converter (the DAC reference output, the 4.096V onboard reference, or the 5V reference, respectively). If installed, jumper J6 connects the A/D reference to pin 4 (VREF_PIN) on the Field Header.

The default configuration of the reference voltage selection jumpers is J4 and J6 installed. This connects the temperature-stable 4.096 reference voltage to the A/D reference input pin providing a 0 to 4.096 input range for each channel of the A/D. The default configuration also connects the 4.096 reference voltage to the reference pin on the Field Header (pin 4) for external connections. Using the default configuration requires that you initialize the DAC to generate its own reference voltage internally by passing the `INT_1V_DAC12` or `INT_2V_DAC12` parameters to the `Init_Analog_IO` function. This will allow the DAC to output voltages in the range of 0 to 2.048 volts or 0 to 4.096 volts respectively, as the maximum DAC output is twice the DAC reference voltage. The following sections describe the function of each jumper and list the valid jumper configuration options. CAUTION: Not

all jumper configurations are valid and certain invalid configurations may damage the module.

DAC Reference Jumper

The DAC reference jumper, labeled J3, and named "DAC Ref" on the silkscreen of the Analog I/O Wildcard, connects the DAC reference pin to the A/D reference input pin. The DAC reference voltage can be generated internally, or supplied externally by installing jumper J6. The DAC reference voltage is configured with the function `Init_Analog_IO` and the constants `INT_1V_DAC12, INT_2V_DAC12`, or `EXT_DAC12.` Table 1-3 summaries the configuration options available for the DAC reference voltage with a jumper shunt installed across J3.

Table 1-3 DAC reference voltage configuration options with jumper J3 installed

Init_Analog_IO Option	DAC Ref Pin State	DAC Reference Pin Voltage	A/D Reference Voltage	DAC Output Range	A/D Input Range
INT_1V_DAC12	Output	1.024 volts	1.024 volts	0 to 2.048 volts	0 to 1.024 volts
INT_2V_DAC12	Output	2.048 volts	2.048 volts	0 to 4.096 volts	0 to 2.048 volts
EXT_DAC12	Input	Voltage on pin 4 of Field Header with J6 installed	Voltage on pin 4 of Field Header with J6 installed	0 to 2x voltage on pin 4 of Field Header with J6 installed	0 to voltage on pin 4 of Field Header with J6 installed

4.096 Volt Reference Jumper

The 4.096 Reference Jumper, labeled J4, and named "4V Ref" on the silkscreen of the Analog I/O Wildcard, connects the onboard 4.096 voltage reference to the reference input pin on the A/D. This configures the input range of the A/D as 0 to 4.096 volts. With this option, the DAC must be configured to generate its own reference voltage (using the `INT_1V_DAC12` or `INT_2V_DAC12` options with the `Init_Analog_IO` function).

5.0 Volt Reference Jumper

The 5.0 Reference Jumper, labeled J5, and named "5V Ref" on the silkscreen of the Analog I/O Wildcard, connects the onboard 5 volt analog supply to the reference input pin on the A/D. This configures the input range of the A/D as 0 to 5.0 volts. With this option, the DAC must be configured to generate its own reference voltage (using the `INT_1V_DAC12` or `INT_2V_DAC12` options with the `Init_Analog_IO` function).

Reference In/Out Jumper

The Reference In/Out Jumper, labeled J6, connects the reference input pin of the A/D to the reference pin on the Field Header (pin 4). Install this jumper if the reference voltage is desired on the Field Header or an external reference voltage is required to drive the A/D or the DAC.

Table 1-4 shows the valid jumper configurations for selecting the reference voltage. **Jumper configurations that are not shown in the table may damage the Analog I/O Wildcard!**

Table 1-4 Valid Voltage Reference Jumper Configurations

Installed Jumpers	Description
J4 & J6	Default Configuration. Connects the 4.096 internal reference voltage to the A/D reference input pin and to the reference pin on the Field Header. The DAC must be configured to generate an internal reference of 1.024V or 2.048V. A/D input range: 0 to 4.096V. DAC output range: 0 to 2.048V or 0 to 4.096V.
J5 & J6	Connects the 5.0 analog supply voltage to the A/D reference input pin and to the reference pin on the Field Header. The DAC must be configured to generate an internal reference of 1.024V or 2.048V. A/D input range: 0 to 5.0V. DAC output range: 0 to 2.048V or 0 to 4.096V.
J3 & J6	Connects the DAC reference pin to the A/D reference input pin and to the reference pin on the Field Header. If the DAC is configured to generate an internal reference of 2.048V, 2.048V appears on pin 4 of the Field Header and the A/D reference input pin. A/D input range: 0 to 2.048V. DAC output range: 0 to 4.096V. If the DAC is configured to generate an internal reference of 1.024V, 1.024V appears on pin 4 of the Field Header and the A/D reference input pin. A/D input range: 0 to 1.024V. DAC output range: 0 to 2.048V. If the DAC is configured to use an external reference, the reference voltage that is supplied to pin 4 of the Field Header becomes the reference for both the A/D and the DAC. A/D input range: 0 to Vpin4. DAC output range: 0 to 2*Vpin4.
J3	Connects the DAC reference pin to the A/D reference input pin. The DAC must be configured to generate an internal reference of 1.024V or 2.048V. The reference pin on Field Header is unconnected. A/D input range: 0 to 1.024V or 0 to 2.048V. DAC output range: 0 to 2.048V or 0 to 4.096V.
J4	Connects the 4.096 internal reference voltage to the A/D reference input pin. The reference pin on Field Header is unconnected. The DAC must be configured to generate an internal reference of 1.024V or 2.048V. A/D input range: 0 to 4.096v. DAC output range: 0 to 2.048V or 0 to 4.096V.
J5	Connects the 5.0 analog supply voltage to the A/D reference input pin. The reference pin on Field Header is unconnected. The DAC must be configured to generate an internal reference of 1.024V or 2.048V. A/D input range: 0 to 5.0v. DAC output range: 0 to 2.048V or 0 to 4.096V.

Analog I/O Wildcard Field Header

The analog inputs and outputs are brought out to a 24-pin dual row header on the Analog I/O Wildcard as shown in Table 1-5.

Table 1-5 Analog I/O Wildcard Field Header

Signal	Pins	Signal
GND	1 – 2	+5V
VAN	3 – 4	REF
ADCGND	5 – 6	ADCGND
AD16_CH7	7 – 8	AD16_CH6
AD16_CH5	9 – 10	AD16_CH4
AD16_CH3	11 – 12	AD16_CH2
AD16_CH1	13 – 14	AD16_CH0
DACGND	15 – 16	DACGND
DAC12_CH7	17 – 18	DAC12_CH6
DAC12_CH5	19 – 20	DAC12_CH4
DAC12_CH3	21 – 22	DAC12_CH2
DAC12_CH1	23 – 24	DAC12_CH0

To connect your transducer signals or control inputs to the Field Bus (H3 on the Analog I/O Wildcard) use a ribbon cable or the Screw Terminal Module that brings out the signals to screw terminal blocks. Shielding the connecting wires is highly recommended for optimal performance.

Software

A package of pre-coded device driver functions is provided to make it easy to use the Analog I/O Wildcard. This code is available as a pre-compiled “kernel extension” library to C and Forth programmers.

Overview of the Software Device Driver Functions

The Analog I/O Wildcard driver code makes it easy to initialize the A/D and DAC, acquire 16-bit samples from the A/D, and write 12-bit values to the DAC. The following sections

describe the functions that initialize the A/D and DAC, read from the A/D inputs, and write to the DAC outputs.

Most of the functions accept as input parameters the channel number and the Analog I/O Wildcard Number (0 through 7). Be sure the module number passed to the software functions correspond to the hardware jumper settings as described in Table 1-2 above.

Initializing the Analog I/O Software Drivers

Use `Init_Analog_IO` to initialize the software drivers for the DAC and A/D, set the reference voltage of the DAC, and output 0 volts to all DAC channels. `Init_Analog_IO` must be called before attempting to read a value from the A/D or write a value to the DAC. The constants `INT_2V_DAC12`, `INT_1V_DAC12`, and `EXT_DAC12`, specify one of the three different reference voltage options for the DAC when passed to `Init_Analog_IO`. The following section provides more information about the DAC reference voltage options.

Using the DAC Drivers

The Analog I/O Wildcard has eight 12-bit DAC outputs. Each DAC accepts a number between 0 and 4095 that we'll designate as N, and outputs a voltage given by Equation 1-1.

$$Vout = 2 * Vref * (N / 4096) \qquad \text{Eqn. 1-1}$$

There are three different options for Vref:

1. The DAC's internally generated 2.048 volts, selected by passing the constant `INT_2V_DAC12` to `Init_Analog_IO`. This is the default option and provides an output range for each DAC channel of 0 to 4.096 volts.
2. The DAC's internally generated 1.024 volts, selected by passing the constant `INT_1V_DAC12` to `Init_Analog_IO`. This provides an output range for each DAC channel of 0 to 2.048 volts.
3. An externally generated voltage applied to the reference pin (pin 4) of the Field Header with jumpers J3 and J6 installed, selected by passing the constant `EXT_DAC12` to `Init_Analog_IO`. The maximum voltage of the external reference voltage is 5 volts. However, voltages above 2.048 volts will result in degraded performance of the DAC. Also, the DAC's maximum specified output voltage is 4.6 volts, corresponding to an external reference of 2.3 V.

The constants that are associated with the DAC output channels are `DAC12_CH0`, `DAC12_CH1`, `DAC12_CH2`, `DAC12_CH3`, `DAC12_CH4`, `DAC12_CH5`, `DAC12_CH6`, and `DAC12_CH7`. To output a voltage on channel DAC 0 (pin 24 on the Field Header), use the function `To_DAC` as shown in Listings 1-1 and 1-2.

Listing 1-1 C Code Listing for outputing 2.000 votls to Channel 0 on Module 0.

```
#include <\mosaic\allqed.h>                    // include all qed utilities
#include "library.c"                           // be sure to include kernel ext

#define ANALOG_MODULE0 0                      // define current module
void main ( void )
{
  Init_Analog_IO(INT_2V_DAC12,ANALOG_MODULE0); // init DAC to use 2.048 int ref
  To_DAC12( 2000, DAC12_CH0, ANALOG_MODULE0);  // output 2.000 volts to ch 0
}
```

Listing 1-2 Forth Code Listing for outputing 2.048 volts to Channel 0 on Module 0

```
DECIMAL                                    \ set base to decimal
0 CONSTANT ANALOG_MODULE0                  \ define current module
INT_2V_DAC12 ANALOG_MODULE0 Init_Analog_IO \ init DAC to use 2.048 internal ref
2000 DAC12_CH0 ANALOG_MODULE0 To_DAC12     \ output 2.000 volts to channel 0
```

Another useful function, named `To_All_DACs`, simultaneously outputs a single 12-bit value to all DAC channels on a specified module. `To_All_DACs` uses the primitives `Delay_Update_DAC12` and `Update_DAC12` to simultaneously output the specified value to all channels. Listings 1-3 and 1-4 demonstrate how to use the `Delay_Update_DAC12` and `Update_DAC12` functions to simultaneously write a stair-step pattern to all eight output channels of a DAC.

Listing 1-3 C Code Listing for using Delay_Update_DAC12 and Update_DAC12.

```
#include <\mosaic\allqed.h>                   // include all qed utilities
#include "library.c"                          // be sure to include kernel ext
#define ANALOG_MODULE0 0                      // define current module
void main ( void )
{
  Init_Analog_IO(INT_2V_DAC12,ANALOG_MODULE0); // init DAC to use 2.048 int ref
  Delay_Update_DAC12( ANALOG_MODULE0 );        // delay DAC update
  To_DAC12( 0500, DAC12_CH0, ANALOG_MODULE0);  // output 0.5 volts to ch 0
  To_DAC12( 1000, DAC12_CH1, ANALOG_MODULE0);  // output 1.0 volts to ch 0
  To_DAC12( 1500, DAC12_CH2, ANALOG_MODULE0);  // output 1.5 volts to ch 0
  To_DAC12( 2000, DAC12_CH3, ANALOG_MODULE0);  // output 2.0 volts to ch 0
  To_DAC12( 2500, DAC12_CH4, ANALOG_MODULE0);  // output 2.5 volts to ch 0
  To_DAC12( 3000, DAC12_CH5, ANALOG_MODULE0);  // output 3.0 volts to ch 0
  To_DAC12( 3500, DAC12_CH6, ANALOG_MODULE0);  // output 3.5 volts to ch 0
  To_DAC12( 4000, DAC12_CH7, ANALOG_MODULE0);  // output 4.0 volts to ch 0
  Update_DAC12( ANALOG_MODULE0 );             // simultaneously update all dacs
}
```

Listing 1-4 Forth Code Listing for using Delay_Update_DAC12 and Update_DAC12.

```
DECIMAL                                    \ set base to decimal
```

```
0 CONSTANT ANALOG_MODULE0                    \ define current module
: STAIR_STEP ( -- )
  INT_2V_DAC12 ANALOG_MODULE0 Init_Analog_IO \ init DAC to use 2.048 int ref
  ANALOG_MODULE0 Delay_Update_DAC12          \ delay DAC update
  0500 DAC12_CH0 ANALOG_MODULE0 To_DAC12     \ output 0.5 volts to channel 0
  1000 DAC12_CH1 ANALOG_MODULE0 To_DAC12     \ output 1.0 volts to channel 1
  1500 DAC12_CH2 ANALOG_MODULE0 To_DAC12     \ output 1.5 volts to channel 2
  2000 DAC12_CH3 ANALOG_MODULE0 To_DAC12     \ output 2.0 volts to channel 3
  2500 DAC12_CH4 ANALOG_MODULE0 To_DAC12     \ output 2.5 volts to channel 4
  3000 DAC12_CH5 ANALOG_MODULE0 To_DAC12     \ output 3.0 volts to channel 5
  3500 DAC12_CH6 ANALOG_MODULE0 To_DAC12     \ output 3.5 volts to channel 6
  4000 DAC12_CH7 ANALOG_MODULE0 To_DAC12     \ output 4.0 volts to channel 7
  ANALOG_MODULE0 Update_DAC12                \ simultaneously update all dacs
;
```

Using the A/D Drivers

The eight input lines of the Analog I/O Wildcard can be configured as either eight 16-bit unipolar single-ended input channels or four 16-bit unipolar-differential input channels. In single-ended configuration, the convertor can digitize only positive ground-referenced voltages. Each differential channel pair can also only convert a positive differential voltage, but the channel pair can be reconfigured to swap the polarity of the inputs so that a negative difference voltage can also be read. Consequently the four differential inputs can each be reconfigured so that there are a total of eight differential input combinations.

The A/D converts the positive voltage difference between a greater, “positive” input (+IN) and a lesser, “negative” input (–IN) into a digital number in the range 0 to 65536, with 0 corresponding to +IN = –IN and 65535 corresponding to +IN – –IN = Vref. If the voltage on the –IN pin is greater than that of the +IN pin, the conversion result is zero.

In single-ended mode the –IN input is connected to ground and eight input channels of +IN are provided for reading positive voltages referenced to ground. In differential mode, the +IN and –IN are assigned to different input channels, and the positive voltage difference, +IN minus –IN, is converted. The –IN input must be kept within the range –0.2 V to +1.25 V, and should not be greater than the +IN input. The +IN input must be kept within the range –0.2 V to +5.2 V and produces meaningful results for values from the –IN input value up to the –IN input value plus the reference voltage. Because both the +IN and –IN input voltage ranged extend well below ground (to –0.2V), voltage differences near or slightly below ground can be read. Further, the +IN can range up to 5.2V.

The sixteen different input options are itemized in Table 1-6.

Table 1-6 Analog Input Connection Options

Associated Constant	Positive Input (+IN)	Negative Input (–IN)	Type
AD16_CH0	AD16_CH0 (pin 14)	ADCGND (pin 5,6)	Single Ended
AD16_CH1	AD16_CH1 (pin 13)	ADCGND (pin 5,6)	Single Ended
AD16_CH2	AD16_CH2 (pin 12)	ADCGND (pin 5,6)	Single Ended
AD16_CH3	AD16_CH3 (pin 11)	ADCGND (pin 5,6)	Single Ended
AD16_CH4	AD16_CH4 (pin 10)	ADCGND (pin 5,6)	Single Ended
AD16_CH5	AD16_CH5 (pin 9)	ADCGND (pin 5,6)	Single Ended
AD16_CH6	AD16_CH6 (pin 8)	ADCGND (pin 5,6)	Single Ended
AD16_CH7	AD16_CH7 (pin 7)	ADCGND (pin 5,6)	Single Ended
AD16_CH0_CH1	AD16_CH0 (pin 14)	AD16_CH1 (pin 13)	Differential
AD16_CH1_CH0	AD16_CH1 (pin 13)	AD16_CH0 (pin 14)	Differential
AD16_CH2_CH3	AD16_CH2 (pin 12)	AD16_CH3 (pin 11)	Differential
AD16_CH3_CH2	AD16_CH3 (pin 11)	AD16_CH2 (pin 12)	Differential
AD16_CH4_CH5	AD16_CH4 (pin 10)	AD16_CH5 (pin 9)	Differential
AD16_CH5_CH4	AD16_CH5 (pin 9)	AD16_CH4 (pin 10)	Differential
AD16_CH6_CH7	AD16_CH6 (pin 8)	AD16_CH7 (pin 7)	Differential
AD16_CH7_CH6	AD16_CH7 (pin 7)	AD16_CH6 (pin 8)	Differential

To read a voltage from channel 2 (pin 12 on the Field Header) on module 1 with a single-ended conversion, use the function `AD16_Sample` as shown in the example code Listings 1-5 and 1-6.

Listing 1-5 C Code Listing for reading A/D Channel 2 on Module 1.

```
#include <\mosaic\allqed.h>                    // include all qed utilities
#include "library.c"                           // be sure to include kernel ext

#define ANALOG_MODULE1 1                       // define current module
void main ( void )
{
  uint ad16_result;
  Init_Analog_IO(INT_2V_DAC12,ANALOG_MODULE1); // init DAC to use 2.048 int ref
  ad16_result = AD16_Sample( AD16_CH2, ANALOG_MODULE1 ); // read ch 2 on mod 1
  printf("AD Result = %u\n",ad16_result);      // print out A/D counts
}
```

Listing 1-6 Forth Code Listing for reading A/D Channel 2 on Module 1.

```
DECIMAL                                    \ set base to decimal
1 CONSTANT ANALOG_MODULE1                  \ define current module
INT_2V_DAC12 ANALOG_MODULE1 Init_Analog_IO \ init DAC to use 2.048 internal ref
AD16_CH2 ANALOG_MODULE1 AD16_Sample        \ read sample from ch 2 on mod 1
U.                                         \ print out A/D counts
```

To convert the 16-bit result returned from `AD16_Sample` into a voltage, use Equation 1-2.

Input Voltage = +IN – -IN = (Count / 65536) * Vref **Eqn. 1-2**

Vref in Equation 1-2 is the reference voltage selected using the voltage reference selection jumpers which were described in the section entitled "Selecting The Reference Voltage". With the default voltage reference selection jumper configuration (jumpers J4 and J6 installed), Vref is 4.096 volts.

Components and transducers with high output impedances connected to the analog inputs will introduce errors in the analog to digital converter. Table 1-7 shows the maximum source resistance at various sampling rates. Larger source resistances may cause conversion errors of more than one least significant bit (LSB).

The function `AD16_Multiple` rapidly obtains a specified number of samples from an A/D channel and stores the results as sequential 2-byte values in memory starting at the specified extended address. If the specified extended address is in common RAM, the fastest sampling frequency is approximately 17 kHz (corresponding to 57.5 microseconds per sample). If the specified extended address is in paged memory, the fastest sampling frequency is approximately 12 kHz (corresponding to 82.5 microseconds per sample). The timing parameter specifies the timing of the samples, with 0 representing the fastest sampling rate, and 65,535 representing the slowest sampling rate. See the glossary entry for more information.

Table 1-7 Maximum Source Resistance at Various Sampling Rates

Sample Rate (samples per second)	Maximum Source Resistance
< 1K	60 Ohms
1 K	57 Ohms
2 K	50 Ohms
5 K	40 Ohms
10 K	30 Ohms
20 K	20 Ohms

Installing the Analog I/O Wildcard Driver Software

The Analog I/O Wildcard device driver software is provided as a pre-coded modular runtime library, known as a “kernel extension” because it enhances the on-board kernel's capabilities. The library functions are accessible from C and Forth.

Mosaic Industries can provide you with a web site link that will enable you to create a packaged kernel extension that has drivers for all of the hardware that you have on your system. In this way the software drivers are customized to your needs, and you can generate whatever combination of drivers you need. Make sure to specify the Analog I/O Wildcard Drivers in the list of kernel extensions you want to generate, and download the resulting “packages.zip” file to your hard drive.

For convenience, a separate pre-generated kernel extension for the Analog I/O Wildcard is available from Mosaic Industries on the Demo and Drivers media (diskette or CD). Look in the Drivers directory, in the subdirectory corresponding to your hardware (QED, PanelTouch, or EtherSmart), in the ANIO_Module folder.

The kernel extension is shipped as a “zipped” file named “packages.zip”. Unzipping it (using, for example, winzip or pkzip) extracts the following files:

- readme.txt - Provides summary documentation about the library.
- install.txt - The installation file, to be loaded to COLD-started QED Board.
- library.4th - Forth name headers and utilities; prepend to Forth programs.
- library.c - C callers for all functions in library; #include in C code.
- library.h - C prototypes for all functions; #include in extra C files.

Library.c and library.h are only needed if you are programming in C. Library.4th is only needed if you are programming in Forth. The uses of all of these files are explained below.

We recommend that you move the relevant files to the same directory that contains your application source code.

To use the kernel extension, the runtime kernel extension code contained in the install.txt file must first be loaded into the flash memory of the QED Board. Start the QED Terminal software with the QED board connected to the serial port and turned on. If you have not yet tested your QED board and terminal software, please refer to the documentation provided with the QED Terminal software. Once you can hit enter and see the 'ok' prompt returned in the terminal window, type

```
COLD
```

to ensure that the board is ready to accept the kernel extension install file. Use the "Send File" menu item of the terminal to download the install.txt to the QED Board or Panel-Touch Controller.

Now, type

```
COLD
```

again and the kernel has been extended! Once install.txt has been loaded, it does not need to be reloaded each time you revise your source code.

Using the Driver Code with C

Move the library.c and library.h files into the same directory as your other C source code files. After loading the install.txt file as described above, use the following directive in your source code file:

```
#include "library.c"
```

This file contains calling primitives that implement the functions in the kernel extension package. The library.c file automatically includes the library.h header file. If you have a project with multiple source code files, you should only include library.c once, but use the directive

```
#include "library.h"
```

in every additional source file that references the Analog I/O functions.

Note that all of the functions in the kernel extension are of the _forth type. While they are fully callable from C, there are two important restrictions. First, _forth functions may not be called as part of a parameter list of another _forth function. Second, _forth functions may not be called from within an interrupt service routine unless the instructions found in the file named

\fabius\qedcode\forthirq.c

are followed.

NOTE: If your compiler was purchased before June 2002, you must update the files, qlink.bat and qmlink.bat in your /fabius/bin directory on your installation before using the kernel extension. You can download a zip file of new versions at

http://www.mosaic-industries.com/Download/new_qlink.zip

The two new files should be placed in c:\Fabius\bin. This upgrade only has to be done once for a given installation of the C compiler.

Using the Driver Code with Forth

After loading the install.txt file and typing COLD, use the terminal to send the "library.4th" file to the QED Board. Library.4th sets up a reasonable memory map and then defines the constants, structures, and name headers used by the Analog I/O Wildcard kernel extension. Library.4th leaves the memory map in the download map.

After library.4th has been loaded, the board is ready to receive your high level source code files. Be sure that your software doesn't initialize the memory management variables DP, VP, or NP, as this could cause memory conflicts. If you wish to change the memory map, edit the memory map commands at the top of the library.4th file itself. The definitions in library.4th share memory with your Forth code, and are therefore vulnerable to corruption due to a crash while testing. If you have problems after reloading your code, try typing COLD, and reload everything starting with library.4th. It is very unlikely that the kernel extension runtime code itself (install.txt) can become corrupted since it is stored in flash on a page that is not typically accessed by code downloads.

We recommend that your source code file begin with the sequence:

```
WHICH.MAP 0=
IFTRUE 4 PAGE.TO.RAM  \ if in standard.map...
       5 PAGE.TO.RAM
       6 PAGE.TO.RAM
      DOWNLOAD.MAP
ENDIFTRUE
```

This moves all pre-loaded flash contents to RAM if the QED Board is in the standard (flash-based) memory map, and then establishes the download (RAM-based) memory map. At the end of this sequence the QED Board is in the download map, ready to receive additional code.

We recommend that your source code file end with the sequence:

```
4 PAGE.TO.FLASH
5 PAGE.TO.FLASH
6 PAGE.TO.FLASH
STANDARD.MAP
SAVE
```

This copies all loaded code from RAM to flash, and sets up the standard (flash-based) memory map with code located in pages 4, 5 and 6. The SAVE command means that you can often recover from a crash and continue working by typing RESTORE as long as flash pages 4, 5 and 6 haven't been rewritten with any bad data.

Glossary

This glossary defines important constants and functions from the Analog I/O Wildcard driver code.

Overview of Glossary Notation

The main glossary entries presented in this document are listed in case-insensitive alphabetical order (the underscore character comes at the end of the alphabet). The keyword name of each entry is in **bold** typeface. Each function is listed with both a C-style declaration and a Forth-style stack comment declaration as described below. The "C:" and "4th:" tags at the start of the glossary entry distinguish the two declaration styles.

The Forth language is case-insensitive, so Forth programmers are free to use capital or lower case letters when typing keyword names in their program. Because C is case sensitive, C programmers must type the keywords exactly as shown in the glossary. The case conventions are as follows:

- Function names begin with a capital letter, and every letter after an underscore is capitalized. Other letters are lower case, except for capitalized acronyms such as "DAC".
- Constant names and C macros use capital letters.
- Variable names use lower case letters.

Each glossary entry starts with C-style and Forth-style declarations, and presents a description of the function. Here is a sample glossary entry:

C: void **To_DAC12** (int value, int channel_num, int module_num)
4th: **To_DAC12** (value\channel_num\module_num --)

Writes the specified 12-bit value to the specified channel of the 12-bit DAC on the specified module. The eight valid module numbers are 0 to 7 while the channel number is specified with one of the following constants `DAC12_CH0`, `DAC12_CH1`, `DAC12_CH2`, `DAC12_CH3`, `DAC12_CH4`, `DAC12_CH5`, `DAC12_CH6`, and `DAC12_CH7`. The 12-bit value is clamped to the range of 0 to 4095 but no error checking is performed on the channel number or the module number. `Init_Analog_IO` must be called before calling `To_DAC12` to initialize the DAC's reference voltage. Unlike the routines for the 8-bit DAC and 12-bit A/D, a resource variable is not needed for the 12-bit DAC and the 16-bit A/D in multitasking systems. `To_DAC12` executes in approximately 37 microseconds. See also `Init_Analog_IO`.

The C declaration specifies the return data type before the function name, and lists the comma-delimited input parameters between parentheses, showing the type and a descriptive name for each.

The Forth declaration contains a "stack picture" between parentheses; this is recognized as a comment in a Forth program. The items to the left of the double-dash (--) are input parameters, and the item to the right of the double-dash is the output parameter. Forth is stack-based, and the first item shown is lowest on the stack. The backslash (\ ) character is read as "under" to indicate the relative positions of the input parameters on the stack. In the Forth declaration the parameter names and their data types are combined. All unspecified parameters are 16-bit integers. Forth promotes all characters to integer type.

The presence of both C and Forth declarations is helpful: the C syntax shows the types of the parameters, and the Forth declaration provides a descriptive name of the output parameter.

Glossary Quick Reference

Configuration Function

void **Init_Analog_IO** (int reference_option, int module_num)

Constants

AD16_CH0	**AD16_CH0_CH1**
AD16_CH1	**AD16_CH1_CH0**
AD16_CH2	**AD16_CH2_CH3**
AD16_CH3	**AD16_CH3_CH2**
AD16_CH4	**AD16_CH4_CH5**
AD16_CH5	**AD16_CH5_CH4**
AD16_CH6	**AD16_CH6_CH7**
AD16_CH7	**AD16_CH7_CH6**

DAC12_CH0	**DAC12_CH4**
DAC12_CH1	**DAC12_CH5**
DAC12_CH2	**DAC12_CH6**
DAC12_CH3	**DAC12_CH7**

INT_1V_DAC12	**INT_2V_DAC12**
EXT_DAC12	

A/D Routines

void **AD16_Multiple** (xaddr buffer, uint timing, uint samples, int channel, int module)
uint **AD16_Sample** (int channel_num, int module_num)

DAC Routines

void **To_DAC12** (int value, int channel_num, int module_num)
void **To_All_DACs** (int value, int module_num)
void **Delay_Update_DAC12** (int module_num)
void **Update_DAC12** (int module_num)

Glossary Entries

C: **AD16_CH0**
4th: **AD16_CH0** (-- n)
A constant (= 0x04) that, when passed as a parameter to `AD16_Sample` or `AD16_Multiple`, configures the 16-bit A/D to perform single-ended conversions on Channel 0, pin 14, on the Analog I/O Field Header. See also `AD16_Sample` and `AD16_Multiple`.

C: **AD16_CH0_CH1**
4th: **AD16_CH0_CH1** (-- n)
A constant (= 0x00) that, when passed as a parameter to `AD16_Sample` or `AD16_Multiple`, configures the 16-bit A/D to perform differential conversions between Channel 0 and Channel 1 (pins 14 and 13) on the Analog I/O Field Header. See also `AD16_Sample` and `AD16_Multiple`.

C: **AD16_CH1**
4th: **AD16_CH1** (-- n)
A constant (= 0x44) that, when passed as a parameter to `AD16_Sample` or `AD16_Multiple`, configures the 16-bit A/D to perform single-ended conversions on Channel 1, pin 13, on the Analog I/O Field Header. See also `AD16_Sample` and `AD16_Multiple`.

C: **AD16_CH1_CH0**
4th: **AD16_CH1_CH0** (-- n)
A constant (= 0x40) that, when passed as a parameter to `AD16_Sample` or `AD16_Multiple`, configures the 16-bit A/D to perform differential conversions between Channel 1 and Channel 0 (pins 13 and 14) on the Analog I/O Field Header. See also `AD16_Sample` and `AD16_Multiple`.

C: **AD16_CH2**
4th: **AD16_CH2** (-- n)
A constant (= 0x14) that, when passed as a parameter to `AD16_Sample` or `AD16_Multiple`, configures the 16-bit A/D to perform single-ended conversions on Channel 2, pin 12, on the Analog I/O Field Header. See also `AD16_Sample` and `AD16_Multiple`.

C: **AD16_CH2_CH3**
4th: **AD16_CH2_CH3** (-- n)
A constant (= 0x10) that, when passed as a parameter to `AD16_Sample` or `AD16_Multiple`, configures the 16-bit A/D to perform differential conversions between Channel 2 and Channel 3 (pins 12 and 11) on the Analog I/O Field Header. See also `AD16_Sample` and `AD16_Multiple`.

C: **AD16_CH3**
4th: **AD16_CH3** (-- n)
A constant (= 0x54) that, when passed as a parameter to `AD16_Sample` or `AD16_Multiple`, configures the 16-bit A/D to perform single-ended conversions on Channel 3, pin 11, on the Analog I/O Field Header. See also `AD16_Sample` and `AD16_Multiple`.

C: **AD16_CH3_CH2**
4th: **AD16_CH3_CH2** (-- n)
A constant (= 0x50) that, when passed as a parameter to `AD16_Sample` or `AD16_Multiple`, configures the 16-bit A/D to perform differential conversions between Channel 3 and Channel 2 (pins 11 and 12) on the Analog I/O Field Header. See also `AD16_Sample` and `AD16_Multiple`.

C: **AD16_CH4**
4th: **AD16_CH4** (-- n)
A constant (= 0x24) that, when passed as a parameter to `AD16_Sample` or `AD16_Multiple`, configures the 16-bit A/D to perform single-ended conversions on Channel 4, pin 10, on the Analog I/O Field Header. See also `AD16_Sample` and `AD16_Multiple`.

C: **AD16_CH4_CH5**
4th: **AD16_CH4_CH5** (-- n)
A constant (= 0x20) that, when passed as a parameter to `AD16_Sample` or `AD16_Multiple`, configures the 16-bit A/D to perform differential conversions between Channel 4 and Channel 5 (pins 10 and 9) on the Analog I/O Field Header. See also `AD16_Sample` and `AD16_Multiple`.

C: **AD16_CH5**
4th: **AD16_CH5** (-- n)
A constant (= 0x64) that, when passed as a parameter to `AD16_Sample` or `AD16_Multiple`, configures the 16-bit A/D to perform single-ended conversions on Channel 5, pin 9, on the Analog I/O Field Header. See also `AD16_Sample` and `AD16_Multiple`.

C: **AD16_CH5_CH4**

4th: **AD16_CH5_CH4** (-- n)
A constant (= 0x60) that, when passed as a parameter to AD16_Sample or AD16_Multiple, configures the 16-bit A/D to perform differential conversions between Channel 5 and Channel 4 (pins 9 and 10) on the Analog I/O Field Header. See also AD16_Sample and AD16_Multiple.

C: **AD16_CH6**
4th: **AD16_CH6** (-- n)
A constant (= 0x34) that, when passed as a parameter to AD16_Sample or AD16_Multiple, configures the 16-bit A/D to perform single-ended conversions on Channel 6, pin 8, on the Analog I/O Field Header. See also AD16_Sample and AD16_Multiple.

C: **AD16_CH6_CH7**
4th: **AD16_CH6_CH7** (-- n)
A constant (= 0x30) that, when passed as a parameter to AD16_Sample or AD16_Multiple, configures the 16-bit A/D to perform differential conversions between Channel 6 and Channel 7 (pins 8 and 7) on the Analog I/O Field Header. See also AD16_Sample and AD16_Multiple.

C: **AD16_CH7**
4th: **AD16_CH7** (-- n)
A constant (= 0x74) that, when passed as a parameter to AD16_Sample or AD16_Multiple, configures the 16-bit A/D to perform single-ended conversions on Channel 7, pin 7, on the Analog I/O Field Header. See also AD16_Sample and AD16_Multiple.

C: **AD16_CH7_CH6**
4th: **AD16_CH7_CH6** (-- n)
A constant (= 0x70) that, when passed as a parameter to AD16_Sample or AD16_Multiple, configures the 16-bit A/D to perform differential conversions between Channel 7 and Channel 6 (pins 7 and 8) on the Analog I/O Field Header. See also AD16_Sample and AD16_Multiple.

C: void **AD16_Multiple** (xaddr buffer, uint timing, uint samples, int channel, int module)
4th: **AD16_Multiple** (xaddr\timing\num_samples\channel_num\module_num --)
Acquires num_samples from the specified channel of the 16-bit A/D on the specified module and stores the samples as sequential unsigned 16-bit values starting at the specified buffer address. The timing parameter specifies the timing of the samples. The eight valid module numbers are 0 to 7. For single-ended conversions, the channel is selected using one of the following constants: AD16_CH0, AD16_CH1, AD16_CH2, AD16_CH3, AD16_CH4, AD16_CH5, AD16_CH6, and AD16_CH7. Single-ended sampling means that the input voltage of the specified channel is referenced to ADCGND (pins 5 or 6 on the Field Header).

For differential conversions, the channel is selected using one of the following constants: AD16_CH0_CH1, AD16_CH1_CH0, AD16_CH2_CH3, AD16_CH3_CH2, AD16_CH4_CH5, AD16_CH5_CH4, AD16_CH6_CH7, and AD16_CH7_CH6. Differential sampling means that the voltage of the second specified channel is subtracted from the voltage of the first specified channel and the resulting voltage is digitized by the A/D. Be sure that the second specified channel's voltage does not exceed 1.25 volts and the second specified channel's voltage does not exceed the first specified channel's voltage (i.e. the 16-bit A/D can only operate in unipolar mode). Init_Analog_IO must be called before calling AD16_Multiple. Unlike the routines for the 8-bit DAC and 12-bit A/D, a resource variable is not required for the 12-bit DAC and the 16-bit A/D in multitasking systems. If the specified buffer is in common memory, the first sample is taken after 32.5 microseconds and subsequent samples are taken every (57.5+2.5*timing parameter) microseconds. If the specified buffer is in paged memory, the first sample is taken after 32.5 microseconds and subsequent samples are taken every (82.5+2.5*timing parameter) microseconds. If the buffer crosses a page boundary, the sampling interval increases by approximately 4 microseconds for the sample stored after the page boundary was crossed. Of course, the operation of interrupts (including timesliced multitasking) will affect these sampling times. Disables interrupts for 27 microseconds per sample. See also AD16_Sample and Init_Analog_IO.

C: uint **AD16_Sample** (int channel_num, int module_num)
4th: **AD16_Sample** (channel_num\module_num -- 16-bit_result)
Returns a single 16-bit sample from the specified channel of the 16-bit A/D on the specified module. The eight valid module numbers are 0 to 7. For single-ended conversions, the channel is selected using one of the following constants: AD16_CH0, AD16_CH1, AD16_CH2, AD16_CH3, AD16_CH4, AD16_CH5, AD16_CH6, and AD16_CH7. Single-ended sampling means that the input voltage of the specified channel is referenced to ADCGND (pins 5 or 6 on the Field Header). For differential conversions, the channel is selected using one of the following constants: AD16_CH0_CH1, AD16_CH1_CH0, AD16_CH2_CH3, AD16_CH3_CH2, AD16_CH4_CH5, AD16_CH5_CH4, AD16_CH6_CH7, and AD16_CH7_CH6. Differential sampling means that the voltage of the second specified channel is subtracted from the voltage of the first specified channel and the resulting voltage is digitized by the A/D. Be sure that the second specified channel's voltage does not exceed 1.25 volts and the second specified channel's voltage does not exceed the first specified channel's voltage (i.e. the 16-bit A/D can only operate in unipolar mode). Init_Analog_IO must be called before calling AD16_Sample. Unlike the routines for the 8-bit DAC and 12-bit A/D, a resource variable is not required for the 12-bit DAC or the 16-bit A/D in multitasking systems. AD16_Sample executes in 52.25 microseconds and disables interrupts for 27 microseconds. See also AD16_Multiple and Init_Analog_IO.

C: **DAC12_CH0**
4th: **DAC12_CH0** (-- n)

A constant (= 0x00) that, when passed as a parameter to the To_DAC12 function, configures the 12-bit DAC to output a voltage on Channel 0, pin 24, of the Analog I/O Field Header. See also To_DAC12.

C: **DAC12_CH1**
4th: **DAC12_CH1** (-- n)
A constant (= 0x10) that, when passed as a parameter to the To_DAC12 function, configures the 12-bit DAC to output a voltage on Channel 1, pin 23, of the Analog I/O Field Header. See also To_DAC12.

C: **DAC12_CH2**
4th: **DAC12_CH2** (-- n)
A constant (= 0x20) that, when passed as a parameter to the To_DAC12 function, configures the 12-bit DAC to output a voltage on Channel 2, pin 22, of the Analog I/O Field Header. See also To_DAC12.

C: **DAC12_CH3**
4th: **DAC12_CH3** (-- n)
A constant (= 0x30) that, when passed as a parameter to the To_DAC12 function, configures the 12-bit DAC to output a voltage on Channel 3, pin 21, of the Analog I/O Field Header. See also To_DAC12.

C: **DAC12_CH4**
4th: **DAC12_CH4** (-- n)
A constant (= 0x40) that, when passed as a parameter to the To_DAC12 function, configures the 12-bit DAC to output a voltage on Channel 4, pin 20, of the Analog I/O Field Header. See also To_DAC12.

C: **DAC12_CH5**
4th: **DAC12_CH5** (-- n)
A constant (= 0x50) that, when passed as a parameter to the To_DAC12 function, configures the 12-bit DAC to output a voltage on Channel 5, pin 19, of the Analog I/O Field Header. See also To_DAC12.

C: **DAC12_CH6**
4th: **DAC12_CH6** (-- n)
A constant (= 0x60) that, when passed as a parameter to the To_DAC12 function, configures the 12-bit DAC to output a voltage on Channel 6, pin 18, of the Analog I/O Field Header. See also To_DAC12.

C: **DAC12_CH7**
4th: **DAC12_CH7** (-- n)

A constant (= 0x70) that, when passed as a parameter to the To_DAC12 function, configures the 12-bit DAC to output a voltage on Channel 7, pin 17, of the Analog I/O Field Header. See also To_DAC12.

C: void **Delay_Update_DAC12** (int module_num)
4th: **Delay_Update_DAC12** (module_num --)
Configures the 12-bit DAC to accept 12-bit values for each DAC channel but to delay outputting the voltage to the corresponding pin of the DAC until Update_DAC12 is called. This option is disabled by default and is typically used to simultaneously set the output voltages of all 12-bit DAC channels. See also Update_DAC12 and To_All_DACs.

C: **EXT_DAC12**
4th: **EXT_DAC12** (-- n)
A constant (= 0x00) that, when passed as a parameter to the Init_Analog_IO function, configures the 12-bit DAC to use the voltage on the reference pin of the Analog I/O Field Header, pin 4 as its reference if jumper J6 is installed. See also Init_Analog_IO.

C: void **Init_Analog_IO** (int reference_option, int module_num)
4th: **Init_Analog_IO** (reference_option\module_num --)
Initializes the software drivers for the 12-bit DAC and the 16-bit A/D, sets the reference voltage of the DAC to the specified option on the specified module, and sets all 8 DAC channels to 0 volts. The reference option is selected using one of the following constants: INT_2V_DAC12, INT_1V_DAC12, and EXT_DAC12. The eight valid module numbers are 0 to 7. See also INT_2V_DAC12, INT_1V_DAC12, and EXT_DAC12.

C: **INT_1V_DAC12**
4th: **INT_1V_DAC12** (-- n)
A constant (= 0x04) that, when passed as a parameter to the Init_Analog_IO function, configures the 12-bit DAC to use its own internally generated 1.024 volt as its reference. With this option, the DAC output range is from 0 to 2.048 volts. See also Init_Analog_IO.

C: **INT_2V_DAC12**
4th: **INT_2V_DAC12** (-- n)
A constant (= 0x06) that, when passed as a parameter to the Init_Analog_IO function, configures the 12-bit DAC to use its own internally generated 2.048 volt as its reference. With this option, the DAC output range is 0 to 4.096 volts. See also Init_Analog_IO.

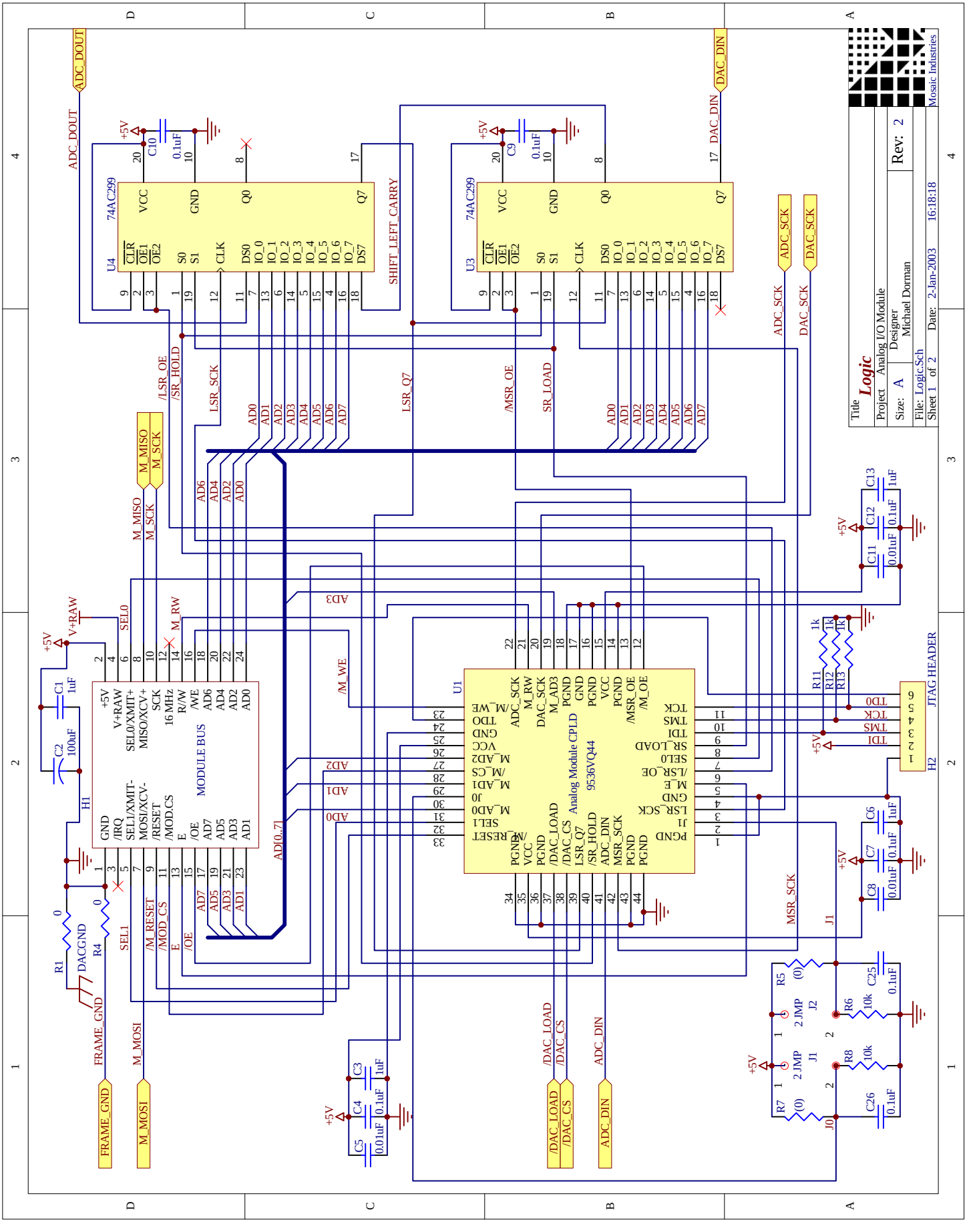
C: void **To_All_DACs** (int value, int module_num)
4th: **To_All_DACs** (value\module_num --)
Writes the specified 12-bit value simultaneously to all channels of the 12-bit DAC on the specified module. To_All_DACs uses the functions Delay_Update_DAC12 and Update_DAC12 to output all voltages to all channels at the same time. The eight valid

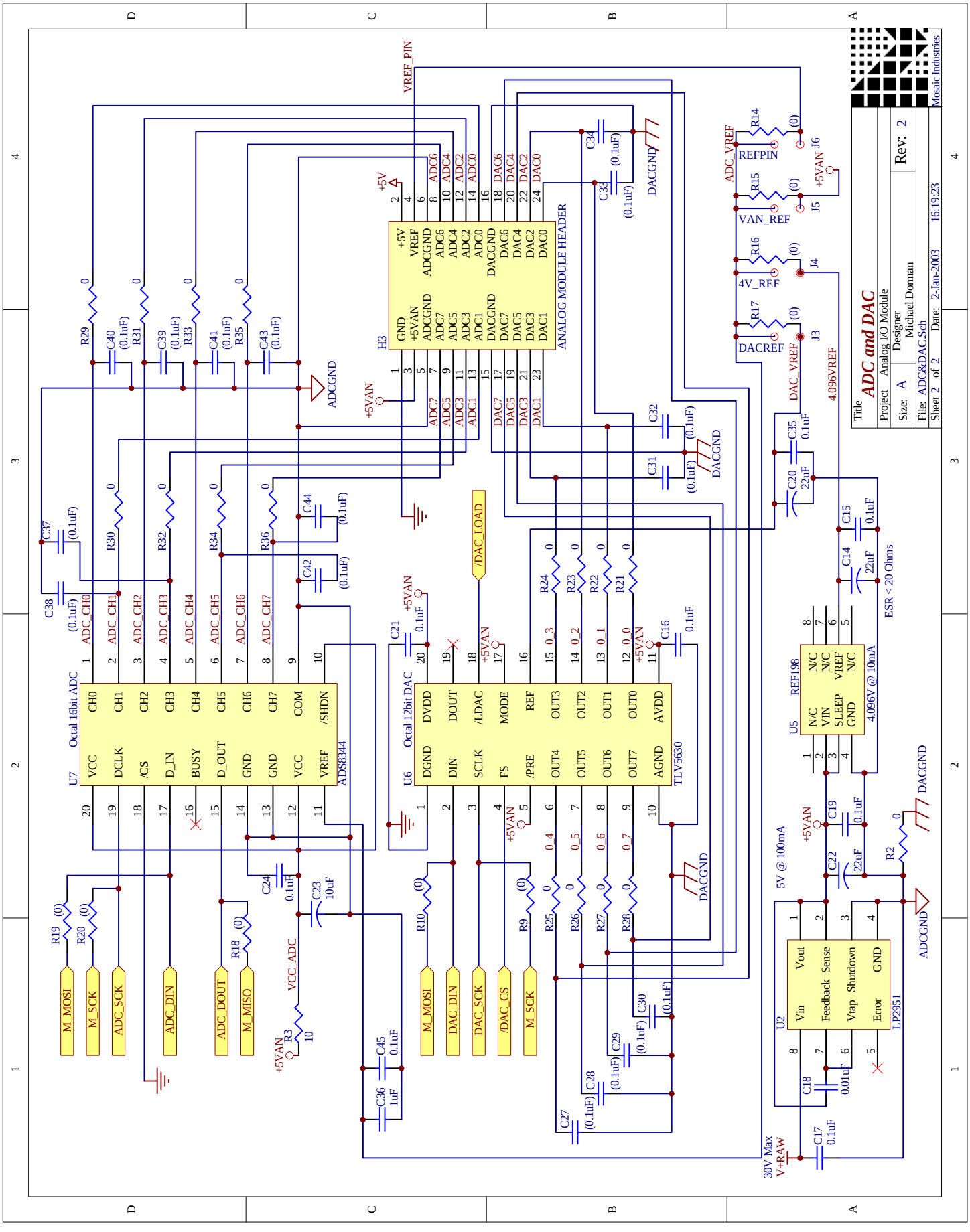
module numbers are 0 to 7. The 12-bit value is clamped to the range of 0 to 4095 but no error checking is performed on the module number. `Init_Analog_IO` must be called before this routine initialize the reference voltage of the DAC. See also `Init_Analog_IO`, `Delay_Update_DAC12`, and `Update_DAC12`.

C: void **To_DAC12** (int value, int channel_num, int module_num)
4th: **To_DAC12** (value\channel_num\module_num --)
Writes the specified 12-bit value to the specified channel of the 12-bit DAC on the specified module. The eight valid module numbers are 0 to 7 while the channel number is specified with one of the constants `DAC12_CH0`, `DAC12_CH1`, `DAC12_CH2`, `DAC12_CH3`, `DAC12_CH4`, `DAC12_CH5`, `DAC12_CH6`, and `DAC12_CH7`. The 12-bit value is clamped to the range of 0 to 4095 but no error checking is performed on the channel number or the module number. `Init_Analog_IO` must be called before calling `To_DAC12` to initialize the DAC's reference voltage. Unlike the routines for the 8-bit DAC and 12-bit A/D, a resource variable is not needed for the 12-bit DAC and the 16-bit A/D in multitasking systems. `To_DAC12` executes in approximately 37 microseconds and disables interrupts for 16.5 microseconds. See also `Init_Analog_IO`.

C: void **Update_DAC12** (int module_num)
4th: **Update_DAC12** (module_num --)
Configures the 12-bit DAC to immediately output a voltage on each DAC channel corresponding to the last 12-bit value written to the channel. This option is enabled by default and is typically used after a call to the `Delay_Update_DAC12` routine. See also `Delay_Update_DAC12`.

Hardware Schematics


Title Logic
Project Analog I/O Module
Size: A
Designer Michael Dorman
Rev: 2
File: Logic.Sch
Date: 2-Jan-2003
Sheet 1 of 2
16:18:18
Mosaic Industries
ADC_DOUT
DAC_DIN
74AC299
SHIFT_LEFT_CARRY
ADC_SCK
DAC_SCK
LSR_SCK
/LSR_OE
/SR_HOLD
LSR_Q7
/MSR_OE
SR_LOAD
M_MISO
M_SCK
M_RW
/M_WE
SEL0
SEL1
/M_RESET
/MOD_CS
E
/OE
AD0
AD1
AD2
AD3
AD4
AD5
AD6
AD7
AD[0..7]
MODULE BUS
Analog Module CPLD
9536VQ44
JTAG HEADER
/DAC_LOAD
/DAC_CS
ADC_DIN
MSR_SCK
FRAME_GND
M_MOSI
DACGND
V+RAW
+5V


ADC and DAC
Project Analog I/O Module
Designer Michael Dorman
Rev: 2
Size: A
File: ADC&DAC.Sch
Sheet 2 of 2
Date: 2-Jan-2003
16:19:23
Mosaic Industries
ANALOG MODULE HEADER
Octal 16bit ADC
ADS8344
Octal 12bit DAC
TLV5630
REF198
LP2951
VREF_PIN
ADCGND
DACGND
+5VAN
+5V
4.096VREF
ESR < 20 Ohms
5V @ 100mA
30V Max
V+RAW
4.096V @ 10mA
REFPIN
VAN_REF
4V_REF
DACREF
ADC_VREF
DAC_VREF
/DAC_LOAD
M_MOSI
M_SCK
ADC_SCK
ADC_DIN
ADC_DOUT
M_MISO
VCC_ADC
DAC_DIN
DAC_SCK
/DAC_CS