OPERATOR'S MANUAL

MA1801/MA1801A

Dual Waveform Generator

Manual Revision 03/30/06
Manual Part Number: MAOM002
Instrument Part Number:
MA1801/MA1801A
Instrument Firmware 1.12

talon
INSTRUMENTS
An EADS North America Defense Company

CERTIFICATION

Talon Instruments certifies that this product met its published specifications at the time of shipment from the factory.

WARRANTY

Talon Instruments products are warranted against defects in materials and workmanship as follows:

(a) One year for the MA1801/MA1801A.

(b) Ninety days for cables.

During the warranty period, Talon Instruments will, at its option, either repair or replace products which prove to be defective.

For warranty service or repair, this product must be returned to the Talon Instruments factory. Buyer shall prepay shipping charges to the factory and Talon Instruments shall pay shipping charges to return the product to the Buyer. However, Buyer shall pay all shipping charges, duties, and taxes for products returned to Talon Instruments from another country.

Talon Instruments warrants that its software and firmware designated by Talon for use with its instruments will execute its programming instructions when properly installed on the instrument. Talon Instruments does not warrant that the operation of the instrument, or software, or firmware will be uninterrupted or error free.

LIMITATION OF WARRANTY

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by the Buyer, Buyer-supplied software or interfacing, unauthorized modification or misuse, operation outside of environmental specifications for the product, or improper site preparation or maintenance.

NO OTHER WARRANTY IS EXPRESSED OR IMPLIED. TALON INSTRUMENTS SPECIFICALLY DISCLAIMS THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

EXCLUSIVE REMEDIES

THE REMEDIES PROVIDED HEREIN ARE BUYER'S SOLE AND EXCLUSIVE REMEDIES. TALON INSTRUMENTS SHALL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER BASED ON CONTRACT, TORT, OR ANY OTHER LEGAL THEORY.

SAFETY FIRST

PROTECT YOURSELF AND THE EQUIPMENT.
Follow these precautions:

- Don't repair the module unless you are a qualified electronics technician and have instructions from Talon Instruments.
- Pay attention to the WARNING statements. They point out situations that can cause injury or death.
- Pay attention to the CAUTION statements. They point out situations that can cause equipment damage.
- Use ESD static control procedures when handling the MA1801/MA1801A or any of its modules.

Table of Contents

List of Tables

1 Introduction

1.1 General

The Model MA1801/MA1801A, hereafter referred to as MA1801A, is a single slot MA-Module that contains two 14-bit arbitrary waveform generator channels which can operate independently or in lock step. The maximum sample rate is 125 MS/s. All waveforms are output from data loaded into waveform memory.

Waveform memory (1M samples/channel) is provided so that multiple waveforms can be loaded at once. Also, memory paging allows one page of memory to be outputting while the other page is being re-loaded with new data.

Powerful sequencing allows waveform segments to be looped and output in any order. There are also provisions for jumping to different sequences based on a trigger input (with or without return).

Flexible triggering is also provided to start/stop/restart the waveform or for advancing to the next segment.

A sync pulse can be placed anywhere in the waveform. It also can be of varying width. During the sync pulse, an alternate waveform can replace the waveform currently being output.

Up to 4 adjacent MA1801A modules can be linked together (in a VXI carrier, for example) to provide 8 ARB channel outputs which are running synchronously.

1.2 Specifications

1.2.1 Module

ARB Channels 2
Memory per Channel
- Banks (Primary/Secondary) 2
- Size 1M

Signal Outputs (SigOut1A, SigOut2A, SigOut1B, SigOut2B) 4
- Configuration ..... Single Ended
- Connector ..... Multi Pin (J1-E, J1-F, J1-H, J1-J)
- Driver ..... 74LVC244 (LVTTL)
- Current ..... +/-24mA
- Termination ..... None
- Signal Source ..... Software Selectable

Trigger/Advance Inputs
- Front Panel Analog (TRIG1A, TRIG2A) 2
- Front Panel TTL (TRIG2A, TRIG2B) 2
- M-Module Bus (TRIGA, TRIGB) 2
- Internal Timer (TIMERA, TIMERB) 2
- Sequence Flag ..... 2 per channel
- Arb Marker ..... 2 per channel

Trigger/Advance Signal Control ..... Software Selectable
- Active High, Active Low, Rising Edge, Falling Edge

Trigger Range ..... DC to 20MHz
Trigger Delay ..... 0 to 65535 Sample Words
Trigger Response Delay ..... 7 Master Clocks + 20ns
Amplitude Modulation Input ..... 1 per channel
Master Clock Input 1
Master Clock Reference Input 1

Front Panel Analog Inputs (TRIG1A, TRIG1B)

Configuration ..... Single Ended
Connector ..... MCX (TA, TB)
Impedance ..... 10KΩ , ±5%
Range ..... 50mV pk-pk to ±10V
Threshold ..... –9.75V to +9.75V
- Resolution ..... 5mV (12 bits)
- Accuracy 100mV

Front Panel TTL Inputs (TRIG2A, TRIG2B)

Configuration ..... Single Ended
Connector ..... Multi Pin (J1-A, J1-B)
Receiver ..... 74LVC244 (TTL level tolerant)
Impedance 10KΩ

Amplitude Modulation Input

Connector ..... MCX (AM)
Impedance ..... 10KΩ , ±5%

Range .....~2V pk-pk for 90% modulation
AM Depth (at half gain) .....0% to 90%

Analog Outputs (1 per channel)

Configuration .....Single ended
Connector .....Front panel MCX (OA, OB)
Stand-by .....Normal or output off (relay disconnect)
Impedance .....50Ω, ±1%
Short-circuit Protection .....Any voltage between ±12V
Output Range
- No Load .....±10V
- 50 Ohm Load .....±5V

Gain control .....10:1 vernier
- Resolution .....12 bits
- Accuracy MA1801 .....1% of programmed value ± 50mV
- Accuracy MA1801A .....± 50mV

Offset
- No Load .....±10V
- 50 Ohm Load .....±5V
- Resolution.....5 mV (12-bits): 1.25mV in increment mode
- Accuracy MA1801 .....1% of programmed value ± 50 mV
- Accuracy MA1801A .....± 50 mV

Filter
- Control .....on/off
- Type .....7-pole elliptic
- Fco .....50 MHz

Internal Timers

Range .....20 µsec to 1.3107 sec
Resolution .....20 µsec

Master Clock

Internal
- Range .....1Hz to 125MHz
- Resolution.....~4 digits
- Accuracy .....50 ppm
- Jitter .....30 ps RMS
- External 10MHz Reference
  - Connector .....MCX (RC)
  - Impedance.....50Ω ±1%
  - Range .....50mV pk-pk to ±5 V
  - Feed through.....< 70dBc
  - Threshold .....Prog. –4.75V to +4.75V
    - Resolution.....2.5mV
    - Accuracy.....1 % of programmed value + 50mV

External
- Connector .....MCX (EC)
- Frequency .....1Hz to 100MHz
- Edge rate .....TBD V/ns min.
- Impedance .....50 Ohm ±1%
- Range .....50mV pk-pk to ±5 V
- Threshold.....Prog. –4.75V to +4.75V
  - Resolution 2.5mV
  - Accuracy .....1% of programmed value + 50mV

Sample Clock

Divided down from the master clock using a 1-32 bit binary divider.

Linked Operation

Channel B may be Linked to Channel A on the MA1801A. In Linked operation, Channel A provides the Sample Clock, Triggers and Gates to Channel B so that the two MA1801A modules will be running in lock step.

Master/Slave Operation

Up to 4 adjacent MA1801A Modules can be linked together on a single carrier using an available inter-module cable. Channel A on one module is designated as the Master. Any of the other channels can be slaves or operated independently.
The Master channel provides the Sample Clock, Triggers and Gates to the designated Slave channels so that they will all be running in lock step with the master Channel.

Software

The MA1801A is provided with VISA Plug&Play Instruments Driver and a Soft Front Panel.

1.2.2 Environmental

Temperature Range

Operating: .....0°C to 50°C (25°C +/- 10°C for specified operation)

Storage: .....-40°C to +71°C (RH not controlled.)

Altitude:

Operating: .....Sea level to 10,000 ft.
Storage: .....Sea level to 15,000 ft.
Relative Humidity (non-condensing)
0°C to +10°C: .....not controlled.
+11°C to +30°C: .....80 +/- 5% RH max.
+31°C to +40°C: .....75 +/- 5% RH max.
+41°C to +50°C: .....45 +/- 5% RH max.

1.2.3 Size

Dimensions: .....Single-wide MA-Module. 5.687”(144.5mm) long X 2.082”(52.9mm).
Weight: .....<0.113 kg (4 oz.).

1.2.4 Power

Total: < 10 Watts

Voltage	Peak Current	Dynamic Current
+5 Vdc	1.2 A	TBD
+12 Vdc	0.2 A	TBD
-12 Vdc	0.3 A	TBD

1.2.5 Cooling Requirements

For 10° C degree rise TBD l/s, TBD mm H20

1.2.6 Bus Compliance

The MA1801A module complies with the ANSI/VITA 12-1996 Specification for double wide M-Modules and the MA-Module trigger signal extension. The module also supports the optional IDENT and VXI-IDENT functions.

Module Type:	MA-Module
Addressing:	A08/A24
Data:	D16/D32
Interrupts	INTA & INTC
DMA	not supported
Triggers:	TRIGA and TRIGB
Identification	IDENT and VXI-IDENT
Manufacturer ID	F0F16
Model Number	00CC16
VXI Model Code	070916
Revision Level	000116

2 Preparation

2.1 Receiving Inspection

Check the shipment at the time of delivery and inspect each box for damage. Describe any box damage and list any shortages on the delivery invoice.

2.1.1 Unpacking Instructions

1. Unpack the box in a clean and dry environment. Save all the packing material in case the instrument must be returned for repair. The following is the packing list of the MA1801A.
   A. MA1801A MA-Module
   B. MA1801/MA1801A Operators Manual
   C. Front Panel J1 Mating Connector
   D. Mounting hardware (4-M3x6mm slotted fillister head screws)
   E. System Software CD
2. Verify that all the items on the packing list have been included. Call Talon's Customer Service representative (800-722-2528) if any items are missing.
3. Inspect the equipment carefully for any signs of mechanical damage regardless of the condition of the shipping boxes.
4. In the case of mechanical damage, call the shipper immediately and start the claim process.
5. Call Talon's Customer Service representative (800-722-2528) to inform them that the shipment arrived damaged. Please be prepared to provide a detailed damage report.

2.1.2 Handling Precautions

The MA1801A contains components that are sensitive to electrostatic discharge. When handling the module for any reason, do so at a static-controlled workstation, whenever possible. At a minimum, avoid work areas that are potential static sources, such as carpeted areas. Avoid unnecessary contact with the components on the module.

2.1.3 Returning Equipment

Follow these steps when you return equipment to Talon:

1. Call Talon Instruments (800-722-2528) for a Return Material Authorization number (RMA). The Talon Customer Service representative will ask for your name, telephone number, company name, equipment type, model number, serial number, and a description of your problem.
2. If at all possible, always use the original shipping container. Use a double-walled cardboard shipping container. Protect all sides, including the top and bottom, with shock absorbing material (minimum of 1 inch thick material) to prevent movement of the MA1801A within the container. Seal the shipping container with approved sealing tape. Mark "FRAGILE" on all sides, top, and bottom of the shipping container. If you use inadequate material, you'll be responsible for any shipping damage repair as carriers won't accept responsibility on incorrectly packed equipment.
3. Pack and ship the equipment to:

   Talon Instruments

   4 Goodyear
   Irvine, CA 92618

   Include the RMA number on the address label as well as the packing list.

2.1.4 Storage

The MA1801A should be stored in a clean, dry environment. In high humidity environments, protect the MA1801A from temperature variations that could cause internal condensation. The following environmental conditions apply to both shipping and storage:

Temperature	-40°C to +71°C
Relative Humidity	not controlled, non-condensing
Altitude	<40000 ft. (12192 m)

Vibration	<2g
Shock	<40g

2.2 Hardware Installation

All MA1801A modules must be installed into the carrier before the carrier is installed into the host system. MA1801A modules are installed by firmly pressing the connector on the MA1801A together with the connector on the carrier. Secure the MA1801A with mounting hardware provided as shown in figure 2-1.

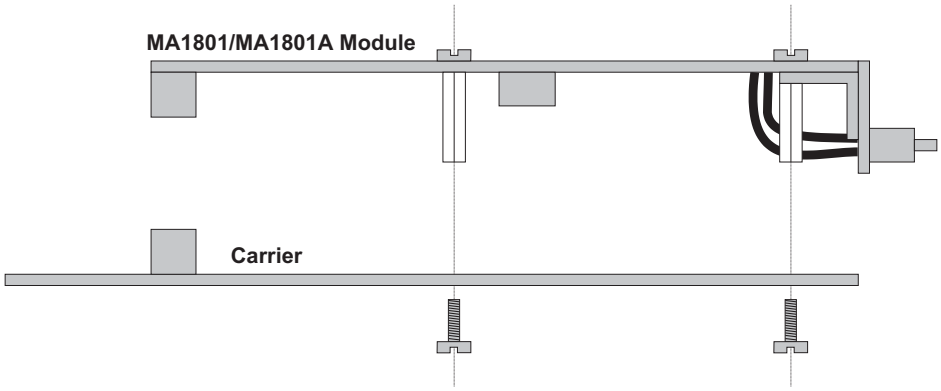

Figure 2-1 MA1801A Installation

2.2.1 Master/Slave Cable

If multiple MA1801A modules are installed and the application requires coupling channels on different modules then the master/slave cable must be installed prior to installing the MA1801As on the carrier.

The master/slave cable, figure 2-2, is a daisy chain cable that routes the signals necessary for intermodule coupling and includes a termination block on the far right hand side.

Figure 2-2 Master/Slave Cable (Part Number MA/304)

Remove unused connectors and cable from the left hand side of the master/slave cable. Install the master/slave cable into the J8 connector on each MA1801A, figure 2-3. Install the group of MA1801As onto the carrier board.

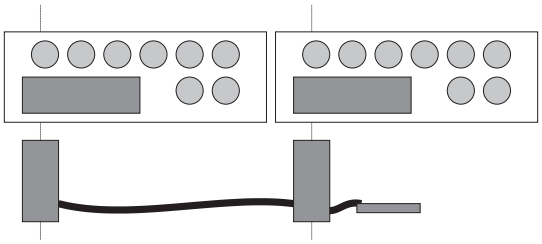

Figure 2-3 Master/Slave Cable Installation

CAUTION
Make sure to connect the master/slave cable into the J8 connector and not the J1 connector on the MA1801A.

2.3 Software Installation

The MA1801A is shipped with a VXI Plug&Play Instrument Driver. The software is included on the CD shipped with the MA1801A system. In addition to the software, the CD also contains the operators manual for the MA1801A in PDF format.

2.3.1 VXI Plug&Play Instrument Driver

The MA1801A Instrument Driver links the communication interface and an application development environment. It provides a higher level, more abstract view of the instrument. It also provides ADE-specific information that supports the capabilities of the ADE, such as a graphical representation. Some of the ADE that this Instrument Drivers supports are listed below:

- Borland Turbo C/C++
- Agilent Technologies Agilent VEE
- Microsoft Visual Basic
- Microsoft Visual C/C++
- National Instruments LabVIEW
- National Instruments LabWindows/CVI

Included with the Instrument Driver is the Soft Front Panel. The soft front panel is a graphical user interface for the MA1801A. It is used to verify communications and functionality when the MA1801A is first integrated into the system.

The MA1801A VXI Plug&Play Instrument Driver requires that VISA be installed. Contact your Slot 0 Resource Manager manufacturer.

To install MA1801A Plug&Play Instrument Driver:

1. Insert the MA1801A System CD into your computer's CD-ROM drive. The MA1801A System Installation menu will run automatically. If the installation menu, figure 2-4, does not appear, run FastMenu.exe from the FastMenu directory on the CD.

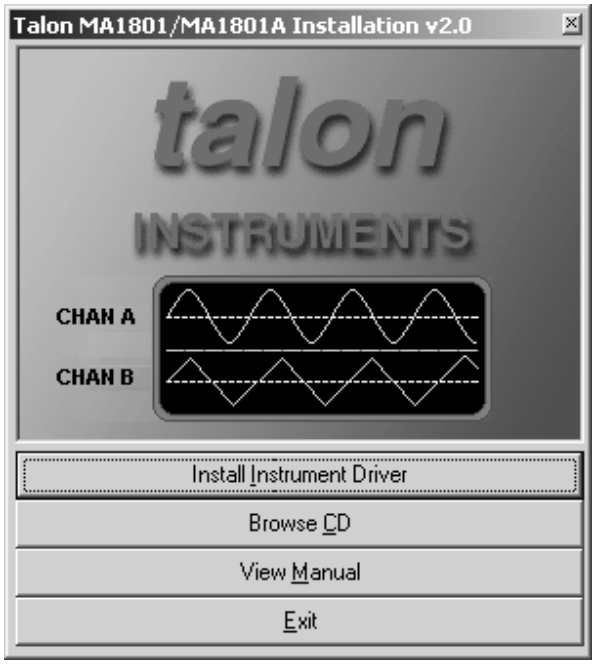

Figure 2-4 MA1801A System Installer

2. Press the “Install Instrument Driver” command button.
3. The installer will remove any versions that are currently loaded on the computer.
4. If a previous version was removed in step 3, then repeat step 2.
5. Follow the installer directions.
6. After the Instrument Driver is installed, the MA1801A Soft Front Panel will be launched.

The following files are installed from the CD:

- ANSI C source code for the Instrument Driver and Soft Front Panel, i.e., .c and .h files.
- MS Windows 32 bit DLL library, i.e., tama1801_32.dll and tama1801.def files.
- Microsoft 32 bit DLL import library, i.e., tama1801.lib file.
- Function panel file, i.e., tama1801.fp file.
- MS Visual Basic Function Declaration text file, i.e., tama1801.bas file.
- Windows help file, i.e., tama1801.hlp file.

Visit the Talon Instruments web site at “www.taloninst.com” and check for MA1801A Instrument Driver updates.

3 Functional Description

This section describes the MA1801A hardware block diagram, module connectors and identification/configuration registers. The full address map is provided in appendix A of this manual.

3.1 Hardware Block Diagrams

The MA1801A is a two channel arbitrary waveform generator. The MA1801A can output sample points at a rate up to 125M samples per second. Up to 1M sample points can be stored in static memory. Static memory is configured as two independent banks (primary and secondary) to allow dynamic update of the waveform memory. An intelligent sequencer and extensive triggering selections control the waveform memory to allow the user to create complex waveforms that includes looping, conditional branching and subroutines. The two channels can output independently or linked, in addition multiple MA1801As can be configured in a master/slave mode to provide more than two synchronized outputs.

3.1.1 Basic Block Diagram

Figure 3-1 illustrates the basic block diagram of the MA1801A.

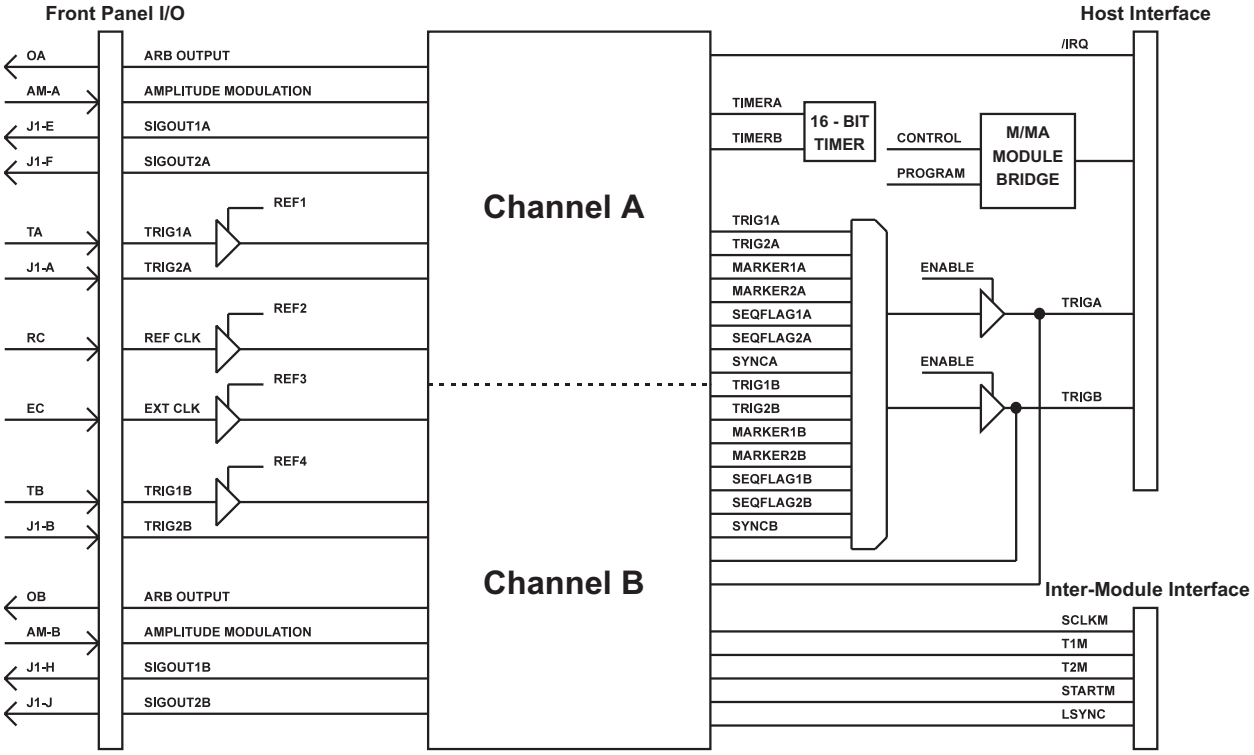

Figure 3-1 MA1801A Basic Block Diagram

3.1.1.1 Variable Voltage Inputs

The MA1801A has four variable voltage inputs, TRIG1A, TRIG1B, EXT CLK and CLK REF.

TRIG1A and TRIG1B can be used to trigger a start, stop, restart or advance for either channel of the MA1801A.

The EXT CLK can be selected as the sample clock for either channel.

The REF CLK can be used as the internal clock generator reference for either channel. The REF CLK input must be 10MHz.

3.1.1.2 Channel A and Channel B

Channel A and Channel B block diagrams are described in section 3.1.2.

3.1.1.3 16 - Bit Timer

The 16 - Bit counter allows the user to program two timers, TIMERA and TIMERB. TIMERA and TIMERB can be used as the start, stop, restart, advance or jump signal for either channel of the MA1801A.

NOTE

In revision 1.12 the timers are reset at the beginning of a sequence when selected as the stop signal or at the beginning of each sequence step if specified as an advance or jump signal. In revision 1.11 TIMERA is rest when channel A is started and TIMERB is reset when channel B is started.

3.1.1.4 TRIGA/TRIGB Outputs

The M-Module signals, TRIGA and TRIGB, can be programmed to output one of a number of signals for triggering other devices. TRIGA and TRIGB can be separately enabled.

3.1.1.5 Inter-Module Interface

The inter-module interface connects adjacent MA1801A modules together in a master/slave configuration. One channel is designated as the master and broadcasts the clock, trigger, start and sync signals to the designated slaves.

3.1.2 Channel Block Diagram

Figure 3-2 illustrates the channel block diagram for each channel.

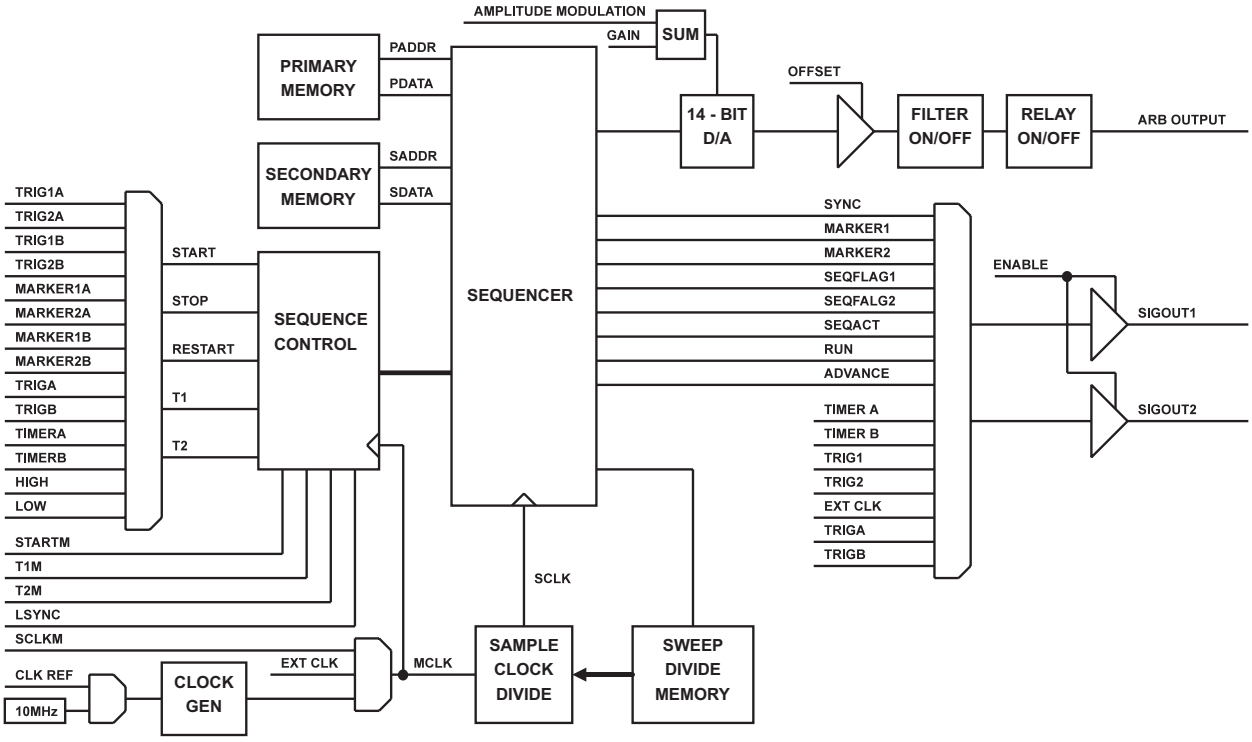

Figure 3-2 Channel Block Diagram

3.1.2.1 Primary and Secondary Memory

The primary and secondary memories contain the waveform data. Each memory is 32 bits by 256K. Each 32 bit memory location contains two waveform sample points. The waveform is formed by programming the sample points in these memories. While one bank is outputting waveform data the other bank can be programmed in a “ping-pong” fashion to output continually changing waveforms.

3.1.2.2 Sequence Control

The sequence control allows the user to program how the sequencer starts/stops and advances from one sequence step to another.

The START, STOP and RESTART signals control when the sequencer is active as described below:

START	The START signal is used to activate the sequencer and output the specified waveform. The START signal can only activate the sequence if the channel has been armed.
STOP	The STOP signal deactivates an active sequence waveform.
RESTART	The RESTART signal reactivates a previously stopped sequence waveform.

The T1 and T2 signals control when the sequencer advances from one step to another.

The user can select the source of the five control signals described above. In addition the user can program the active state of the control signal from one of the following selections:

- Active High
- Active Low
- Rising Edge
- Falling Edge

The master/slave and linked mode selections determine which set of signals are used to control the sequencer, the local set or the external set from the inter-module interface. If Channel A is programmed as the master channel then the inter-module signals are driven by channel A.

Manual control of the start, stop, restart and advance functions can be performed through register access.

The system delay of the sequence control signals is 7 master clocks plus 20ns. See the frequency divider section 3.1.2.4 for steps to minimize the system delay.

3.1.2.3 Clock Generator

An internal clock generator can be used to specify the sample clock. Every rising edge of the sample clock outputs a new waveform sample point.

The internal clock generator can be programmed from 1Hz to 125MHz.

The clock generators 10MHz reference clock can set to internal or external.

3.1.2.4 Sample Clock Divider

The sample clock divider is a binary clock divider which is used for setting the sample clock (SCLK) rate. The divider is a 32 bit value that is programmed one less than the desired divide value (SCDIV). The formula for sample rate is:

SCLK = MCLK ÷ SCDIV

For minimal system delay, master clock (MCLK) should be set as high as possible with the sample clock divider then used to set the desired sample clock rate.

3.1.2.5 Sweep Divide Memory

In sweep operations, the sample clock divider can be updated on the fly. An array of 256 divide values can be stored and cycled through for sweeping a waveform.

3.1.2.6 Sequencer

The sequencer is a bank of static RAM that defines 512 separate sequence steps. Each sequence step identifies a block of waveform memory to output (called a segment).

A sequence can be defined to output one or more segments. The desired number of words for each segment step can be specified (i.e. all of a given segment or some portion of it). Each segment can be looped and a jump sequence can be defined (i.e. the sequence that will be executed next if the jump condition is met). A return flag can also be set (i.e. jump with return). A jump can also occur immediately or at the end of the present segment. When a jump with return occurs, the sequence step jumped to will be output in SING mode with auto advance and then return to the step that was jumped from.

Each sequence step can be looped up to 64k times.

Two Sequence Flags can be programmed. Each one can be set HI or LOW on each sequence step.

The mode which controls advancing from one sequence step to the next can be of one of the following choices:

STEP	The sequence is advanced to the next step only when a valid trigger is received. The output of the MA1801A generates the current segment continuously until a trigger signal advances the sequence to the next step. If loops were specified, they are ignored in STEP mode.
SING	The MA1801A idles between steps until a valid trigger signal is sensed. After outputting the specified number of loops, the output level idles at a DC level equal to the last point of the waveform. The sequencer will step to the next step in the sequence when it receives its next valid trigger.
SING1	The MA1801A idles between steps until a valid trigger signal is sensed. After a trigger, the generator outputs one waveform loop. Then, the output level idles at a DC level equal to the last point of the last generated waveform. If loops were programmed, the sequencer repeats this segment each time a trigger is received, until the number of loops specified for this step has been reached. After reaching this number, the sequencer advances to the next step in the sequence.

A SYNC pulse can be defined to occur at any point during the output of the waveform. The reference starting point can be any one of the following:

- The beginning of the Sequence.
- The beginning of the designated Sequence Step.
- The beginning of the designated Sequence Step and on each loop of the Sequence Step.
- The beginning of the designated Sequence Step but only on the designated Loop Count.

The start trigger can be offset from 0-2M words. The width of the SYNC pulse can be from 1 to 4097 words.

During the period of the sync pulse, the data being presently output can be replaced by other data from the waveform memory.

A sequence can run continuous or for a burst from 1 to 1M times.

3.1.2.7 14 - Bit D/A

The gain reference level and buffered AM modulation inputs are SUMed, analog wise, and applied to this D/A as a reference input.

Note 1: The DAC output is not perfectly proportional to the reference input. Low gains may not be accurate and deep modulations will not be symmetrical.

Note 2: The bandwidth of the reference input is limited. Thus AM modulation is limited to ~100kHz for small modulations and perhaps down to 20kHz at 80% modulation.

3.1.2.8 Filter

The filter is a 7-pole elliptic filter with a frequency cutoff of 50 MHz. It is designed to improve the shape of sine waves above 10MHz. It will attenuate the amplitude of sine wave outputs ~10% at 10MHz and up to 25% at 25MHz.

3.1.2.9 Relay

The arbitrary waveform output can be relay isolated from the front panel.

3.1.2.10 SIGOUT1/SIGOUT2 Outputs

The MA1801A signals, SIGOUT1 and SIGOUT2 , can be programmed to output one of a number of signals for monitoring MA1801A status or triggering other devices. SIGOUT1 and SIGOUT2 for both channels are enabled together.

3.2 Module Connectors

Figure 3-3 illustrate the MA1801A PCB and front panel connectors.

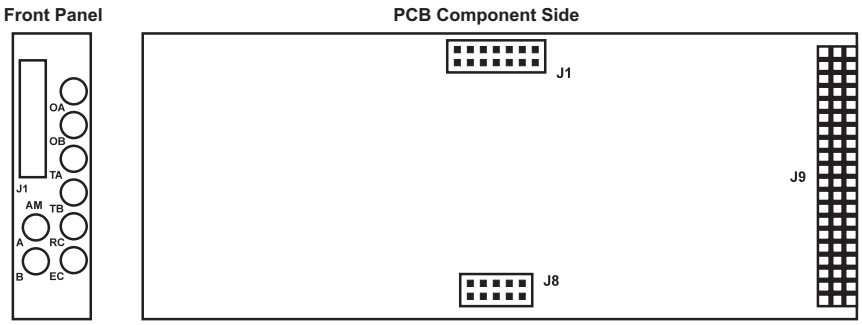

Figure 3-3 PCB and Front Panel Connectors

3.2.1 M/MA-Module Logic Bus Connector (PCB-J9)

The M/MA-Module logic bus connector contains signal and voltage connections for the M/MA-Module interface logic.

3.2.2 Inter Module Connector (PCB-J8)

The inter module connector is used to link adjacent MA1801A modules in a master slave mode.

3.2.3 FPGA Program Connector (PCB-J1)

The FPGA program connector is used to program the MA1801A FPGA serial PROM.

3.2.4 Coaxial Connectors (Front Panel MCX)

The front panel coax connectors provides various analog input and output signals. (See appendix B for pin assignments).

OA: Channel A waveform output.
OB: Channel B waveform output.
TA: Channel A trigger 1 input (TRIG1A).
TB: Channel B trigger 1 input (TRIG1B).
RC: Channel A & B shared reference clock input.
EC: Channel A & B shared external clock input.
AM A: Channel A amplitude modulation input.
AM B Channel B amplitude modulation input.

3.2.5 I/O Connector (Front Panel J1)

The front panel I/O connector provides various digital input and output signals. (See appendix B for pin assignments).

A: Channel A trigger 2 input (TRIG2A).
B: Channel B trigger 2 input (TRIG2B).
E: Channel A signal output 1 (SIGOUT1A).
F: Channel A signal output 2 (SIGOUT2A).
H: Channel B signal output 1 (SIGOUT1B).
J: Channel B signal output 2 (SIGOUT2B).
C,D,K: Signal ground.

3.3 Identification/Configuration Registers

3.3.1 M-Module PROM Registers

The MA1801A supports the identification function called IDENT. This IDENT function provides information about the module and is stored in sixteen word deep (32 byte) serial EEPROM. Access is accomplished with read/write operations on the last address in I/O space and the data is read one bit at a time.

The MA1801A also supports the VXI-IDENT function introduced by Hewlett-Packard. This function is not part of the approved ANSI/VITA 12-1996 standard. This extension to the M-module IDENT function increases the size of the EEPROM to at least 64 words (128 bytes) and includes VXI compatible ID and Device Type Registers. Details are shown in table 3-1

Word	Description	Value (hex)
0	Sync Code	5346
1	Module Number	00CC
2	Revision Number	0001
3	Module Characteristics	9E37
4-7	Reserved	0000
8-15	M-Module Specific	0000
16	VXI Sync Code	ACBA
17	VXI ID	CF0F
18	VXI Device Type	4709
19-31	Reserved	0000
32-63	M-Module Specific	0000

Table 3-1 M/MA Module EEPROM IDENT Words

Notes:

1) The Revision Number is the functional revision level of the module. It does not correspond to the hardware assembly level.
2) The Module Characteristics bit definitions are:

Bit(s)	Description
15	1 = burst access
14/13	00 (unused)
12	1 = needs ±12V
11	1 = needs +5V
10	1 = trigger outputs
9	1 = trigger inputs
8/7	00 = no DMA requestor
6/5	01 = interrupt type A (software EOI)
4/3	10 = 32-bit data
2/1	11 = 24-bit address
0	1 = memory access

3) The VXI Device Type word contains the following information:

Bit(s)	Description
15-12	4 = 524288 bytes of required memory
11-0	709_{16} = Talon specified model code

3.3.2 IO Registers

There are a variety of registers used to configure and control the MA1801A module. The registers are addressable within the I/O Space. An address map and details of the registers is provided in Appendix A.

4 Register Operation

The MA1801A register operation involves programming several control registers as well as the waveform memory. The entire MA1801A memory map is listed in appendix A.

The following sections describe typical arbitrary waveform generator functions and the register programming required for the MA1801A. Example code is listed in appendix C.

Two register access functions are used to read and write data from/to the MA180. These functions, peek16 and poke16, would be replaced by the users I/O library calls that will perform the similar function. The functions are described below:

Peek16:	Reads 16 bits of data from the specified address. ***Peek16 (int address, unsigned short *data)***
Poke16:	Writes 16 bits of data to the specified address. ***Poke16 (int address, unsigned short data)***

4.1 Initialize Clock/DAC Registers

The MA1801A clock and DAC registers must be initialized before values are written to them.

```
#define CLKGEN              0x40
#define GAIN_A              0x82
#define GAIN_B              0x86
#define OFFSET_A            0x8A
#define OFFSET_B            0x8E
#define TRIG1AREF           0x92
#define TRIG1BREF           0x96
#define RCLKREF             0x9A
#define ECLKREF             0x9E/*
     This function initializes the clock generator and DAC register
     sequence bits of the MA1801A module.
     Input:
         None
     Return Value:
         0
*/
int initModule(void)
{
     /* initialize clock generator sequence bits */
     Poke16(CLKGEN, 0x0F00);
     Poke16(CLKGEN + 2, 0xB0F0);
     Poke16(CLKGEN + 4, 0x0000);
     Poke16(CLKGEN + 6, 0x03E8);
     Poke16(CLKGEN + 8, 0x0700);
     Poke16(CLKGEN + 10, 0x2494);

     /* initialize channel A gain DAC rigister bits */
     Poke16(GAIN_A, 0x08C8);

     /* initialize channel A gain DAC rigister bits */
     Poke16(GAIN_B, 0x48C8);

     /* initialize channel A offset DAC rigister bits */
     Poke16(OFFSET_A, 0x87FF);

     /* initialize channel B offset DAC rigister bits */
     Poke16(OFFSET_B, 0xC7FF);

     /* initialize TRIG1A DAC rigister bits */
     Poke16(TRIG1AREF, 0x0940);

     /* initialize TRIG1B DAC rigister bits */
     Poke16(TRIG1BREF, 0x4940);

     /* initialize REFCLK DAC rigister bits */
     Poke16(RCLKREF, 0x8940);

     /* initialize EXTCLK DAC rigister bits */
     Poke16(ECLKREF, 0xC940);

     return 0;
}
```

4.2 Check the MA1801A Arm Status

Before programming the MA1801A, the channel should be disarmed to prevent the start trigger from generating an output before the new settings are written. Checking the MA1801A arm status is performed by reading the “Run Control Register”, section 2.1 of Appendix A. The run control register contains the arm and execution status for both channels.

The following timing diagram illustrates the arm and execution status bits for the triggered mode receiving a start trigger while armed and not armed.

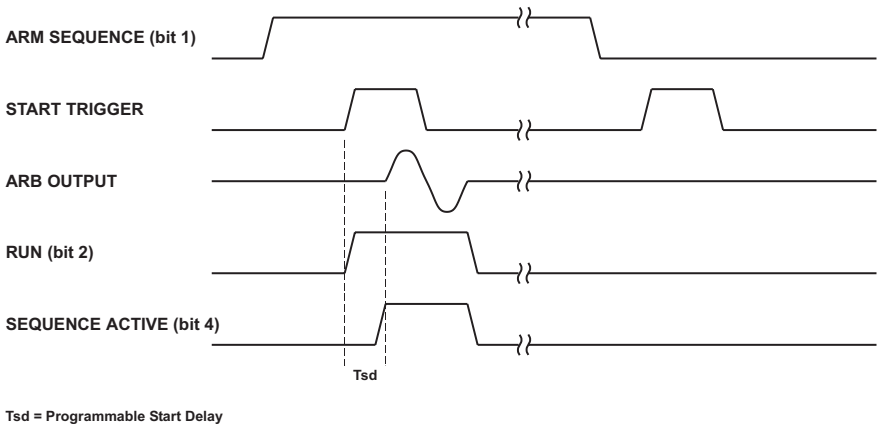

Figure 4-1 Status Bit Timing

The following code example returns the arm bit for the specified channel,

```
#define RUNCTRL     0x0
/*
     This function returns the status bits of the
     specified MA1801A channel.
     Input:
          channel: (0 = channel A, 1 = channel B)
     Return Value:
          Bit 0 = (0 = Not armed, 1 = Armed).
          Bit 1 = (0 = Stopped, 1 = Running).
          Bit 3 = (0 = Sequence not active, 1 = sequence active).
*/
int getStatus(short channel)
{
     unsigned short data;

     /* read run control register */
     Peek16(RUNCTRL, &data);

     /* shift the arm bit if channel is B */
     if (channel == CHANNEL_B)
          data >>= 8;

     /* return the status bits */
     return data & 0xB;
}
```

4.3 Stopping a Channel

There are three ways to stop a MA1801A channel from outputting;

1. Set the arm bit low in the run control register.
2. Set the reset sequence bit high in the run control register.
3. Isolate the output

Setting the arm bit low prevents the MA1801A from responding to the start trigger. Once the current waveform segment has completed the output will remain at the level of the last sample point.

Setting the reset sequence bit high forces the arm, run and sequence active bits low. The output will go to the programmed offset value.

The following code example will disarm or reset the specified channel based on the user selection,

```
#define RUNCTRL            0x0
#define STARTCTRL          0xA
/*
     This function stops the specified MA1801A channel.
     The channel can be stopped by disarming or resetting.
     Input:
          channel: (0 = channel A, 1 = channel B)
          type:    (0 = disarm, 1 = reset, 2 = isolate)
     Return Value:
          0
```

```
*/
int stopChannel(short channel, short type)
{
     unsigned short runctrl, startctrl;

     /* read run control register */
     Peek16(RUNCTRL, &runctrl);

     /* set the arm bit low if type == DISARM */
     if (type == DISARM)
          /* only disarm the specified channel */
          if (channel == CHANNEL_A)
               /* mask off channel A arm bit */
               runctrl &= 0xFFFE;
          else
               /* mask off channel B arm bit */
               runctrl &= 0xFEFF;
     else if (type == RESET || type == ISOLATE)
          if (type == ISOLATE) {
               /* read start control register */
               Peek16(STARTCTRL, &startctrl);

               /* isolate the output before resetting */
               if (channel == CHANNEL_A)
                    /* isolate channel A */
                    startctrl &= 0xFF3F;
               else
                    /* isolate channel B */
                    startctrl &= 0x3FFF;

               /* write start control register */
               Poke16(STARTCTRL, startctrl);
          }

          /* only reset the specified channel */
          if (channel == CHANNEL_A)
               /* set channel A reset bit */
               runctrl |= 0x8;
          else
               /* set channel B reset bit */
               runctrl |= 0x800;

     /* program the control register */
     Poke16(RUNCTRL, runctrl);

     return 0;
}
```

4.4 Setting the Amplitude/Offset

Output amplitude for each of the channels may be programmed separately within the range of 20 mV to 20 Vp-p into an open circuit, and 10 mV to 10 Vpp into 50Ω.

Offset for each of the channels may be programmed separately within the range of ±10V into an open circuit and ±5V into 50Ω.

Note that output peak voltage may never exceed ±10V into an open circuit and ±5V into 50Ω. Amplitude and offset may be set freely within a 10V window, as long as the following limits are kept:

$$| \text{offset} | + \text{Amplitude} \div 2 \leq 10$$

The amplitude of each output channel is programmed by setting the gain reference of the 14 bit DAC. The gain reference can be programmed from 0.01V to 10V with a resolution of 0.005V. The DAC output is not perfectly proportional to the reference input. Low gains may not be accurate and is therefore recommended that the gain reference be programmed greater than 1. Low amplitude waveforms can be achieved even with a gain greater than 1 by programming the sample memory to limit the output voltage. An external attenuator may also be used to generate low amplitude waveforms.

The following code example will program the gain and offset references,

```
#define RUNCTRL            0x0
#define GAIN_A             0x82
#define OFFSET_A           0x8A
/*
     This function programs the gain and offset
     of the specified MA1801A channel.
     Input:
          channel: (0 = channel A, 1 = channel B)
          gain:    (0.01 to 10.0)
          offset:  (-10.0 to 10.0)
     Return Value:
          0
*/
int setGainAndOffset(short channel, double gain, double offset)
{
```

```
    int gainAddr = GAIN_A, offsetAddr = OFFSET_A;
    unsigned short gainData = 0, offsetData = 0, runCtrl;

    /* update the addresses if channel B */
    if (channel == CHANNEL_B) {
        gainAddr += 4;
        gainData = 0x4000;
        offsetAddr += 4;
        offsetData = 0x4000;
    }

    /* calculate gain data */
    gainData += (gain * 200) + 2048;

    /* calculate offset data */
    offsetData += 0x8000 + (offset * 400) + 2048;

    /* program the gain register */
    Poke16(gainAddr, gainData);

    /* program the offset register */
    Poke16(offsetAddr, offsetData);

    /* update the reference DACs */
    /* read run control register */
    Peek16(RUNCTRL, &runCtrl);

    /* preserve arm bits and set the D/A update bit */
    runCtrl = (runCtrl & 0x101) | 0x8000;

    /* program the run control register */
    Poke16(RUNCTRL, runCtrl);

    /* query until D/A update complete */
    do {
        /* read run control register */
        Peek16(RUNCTRL, &runCtrl);
    } while ((runCtrl & 0x8000) == 0x8000);

    return 0;
}
```

4.5 Setting the Sample Clock

Data is clocked to the digital to analog converter (DAC) which, in turn, generates the waveform shape that was downloaded to the arbitrary waveform memory.

Users should be careful not to confuse waveform frequency with sample clock frequency. The sample clock frequency defines the frequency at which the generator clocks data points. The frequency of a given waveform is computed using the sample clock frequency and the number of data points. Use the following equation for computing the frequency of (one cycle of) a waveform:

Waveform Frequency = Sample Clock ÷ Number of data points

For example, using a sample clock of 100MHz with a 2000-point waveform will generate a 50kHz waveform at the output connector.

The sample clock can be set to the internal clock generator, or the external front panel input.

The following code example will program the sample clock source and frequency,

```
#define CLKCTRL             0x34
#define CLKGEN              0x40
/*
     This function programs the sample clock source
     and frequency of the specified MA1801A channel.
     Input:
          channel: (0 = channel A, 1 = channel B)
          source:  (0 = internal, 1 = external)
          freqHz:  (10 to 125,000,000)
     Return Value:
          0
*/
int setSampleClock(short channel, short source, double freqHz, short refSource)
{
     int div1, div2, fbdiv, cgdata;
     unsigned short data, clkctrl;
     double cgfreq;
     short coarseTune = 7;
     time_t t1, t2;

     /* read clock control register  */
     Peek16(CLKCTRL, &clkctrl);

     /* set the sample clock source */
     if (channel == CHANNEL_A) {
          /* set new value */
          clkctrl = (clkctrl & 0xFCFF) + (source << 8);
```

```
} else {
    /* set new value */
    clkctrl = (clkctrl & 0xF3FF) + (source << 10);
}

/* set internal clock generator */
if (freqHz == 0.0 || source == EXTERNAL) {
    /* get LSW of clock generator register 2 */
    Peek16(CLKGEN + 6, &data);

    /* disable the generator */
    data |= 0x2000;

    /* set LSW of clock generator register 2 */
    Poke16(CLKGEN + 6, data);
} else {
    /* claculate the control register divide count */
    for (cgfreq = freqHz, div2 = 0; cgfreq < 20000000.0; div2++)
        cgfreq *= 10;

    /* calculate the generator divide count and feedback */
    if (cgfreq < 5,000,000.00) {
        div1 = 0xB;  /* 40 */
        fbdiv = cgfreq * 0.002;
    } else if (cgfreq < 10000000.00) {
        div1 = 0xA;  /* 20 */
        fbdiv = cgfreq * 0.001;
    } else if (cgfreq < 20000000.00) {
        div1 = 0x9;      /* 10 */
        fbdiv = cgfreq * 0.0005;
    } else if (cgfreq < 50000000.00) {
        div1 = 0x2;      /* 4 */
        fbdiv = cgfreq * .0002;
    } else if (cgfreq < 100000000.00) {
        div1 = 0x1;      /* 2 */
        fbdiv = cgfreq * .0001;
    } else {
        div1 = 0;        /* 1 */
        fbdiv = cgfreq * .00005;
    }

    /* program new post divide value and reference enable */
    if (channel == CHANNEL_A)
        clkctrl = (clkctrl & 0xfff0) + div2;
    else
        clkctrl = (clkctrl & 0xff0f) + (div2 << 4);

    /* calculate clock generator register 2 */
    cgdata = ((fbdiv & 0xFF) << 24) + (div1 << 16) + (refSource << 12) + 0x3E8;

    /* write clock generator register 2 LSW */
    Poke16 (CLKGEN + 6, cgdata & 0xFFFF);

    /* write clock generator register 2 MSW */
    Poke16 (CLKGEN + 4, cgdata >> 16);

    /* calculate clock generator register 3 */
    cgdata = (coarseTune << 24) + (fbdiv >> 8) + (refSource * (7 << 17)) + 0x2480;

    /* write clock generator register 3 LSW */
    Poke16 (CLKGEN + 10, cgdata & 0xFFFF);

    /* write clock generator register 3 MSW */
    Poke16 (CLKGEN + 8, cgdata >> 16);

}

/* get LSW of clock generator register 1 */
Peek16(CLKGEN + 2, &data);

/* select the generator */
if (channel == CHANNEL_A)
    data &= 0xF7FF;
else
    data |= 0x800;

/* set LSW of clock generator register 1 */
Poke16(CLKGEN + 2, data);

/* write out clock control register  */
Poke16(CLKCTRL, clkctrl + 0x1000);

do {
    /* read clock control register and check bit 12  */
    Peek16 (CLKCTRL, &clkctrl);
} while (clkctrl & 0x1000);

/* delay for clock generator to settle */
t1 = clock();
do {
    t2 = clock();
} while ((t2 - t1) < 3);
```

```
        return 0;
}
```

4.6 Programming the Waveform Memory

The MA1801A generates arbitrary waveforms with 14 bits of vertical resolution. Any waveform it generates must first be downloaded to waveform memory.

The waveform memory is configured in two banks of 32-bit words. Each word represents two points on the horizontal (or time) waveform scale. Each word has a horizontal address that can range from 0 to 262142 (primary bank) or 262144 to 524284 (secondary bank) and a vertical address (or voltage level) that can range from 0 to 16383 (14 bits) stored in the upper and lower 16 (lower 16 bits is the first point). Using a high speed clocking circuit, the digital contents of the arbitrary waveform memory are extracted and routed to the Digital to Analog Converter (DAC). The DAC converts the digital data to an analog signal, and the output amplifier completes the task by amplifying or attenuating the signal at the output connector.

There is no need to use the complete memory bank every time a waveform is generated. The Waveform memory can be divided into smaller segments and different waveforms can be loaded into each segment. The various segments may then be loaded into a sequence table to generate long and complex waveforms. The sequence table can link up to 512 segments, while each segment can loop up to 65535 times. Note that sequence generators are separate for each channel. The separation exists also in master-slave mode.

The following code example will program the specified waveform memory with the contents of the data array.

```
#define MEMORY              0x2
#define EMADDR              0xC0
#define EMDATA              0xC4
/*
     This function programs the waveform memory
     of the specified MA1801A channel.
     Input:
          channel: (0 = channel A, 1 = channel B)
          memory:  (0 = primary, 1 = secondary)
          offset:  (0 - 0x3FFFF)
     Return Value:
          0
*/
int setWavformMemory(short channel, short memory, int offset, int samples, unsigned short data[])
{
     unsigned short membsy, memgrnt, memreg;
     unsigned short emaddr, *emdata;
     int error = 0;

     while (0x1) {
          /* read the memory register */
          Peek16(MEMORY, &memreg);

          /* update the memory variable to include the channel */
          memory += (channel * 2);

          /* check the grant bit to see if we have to request memory */
          memgrnt = 1 << (4 + memory);
          if ((memgrnt & memreg) == 0) {
               /* check the busy bit befor requesting memory */
               membsy = memgrnt << 4;
               if ((membsy & memreg)) {
                    error = ERROR_MEMREQ;
                    break;
               }
               /* request and select memory */
               memreg = (memory << 12) + (1 << memory);
               Poke16(MEMORY, memreg);

               /* read the memory register */
               Peek16(MEMORY, &memreg);

               /* did we get memory ? */
               if ((memgrnt & memreg) == 0) {
                    error = ERROR_MEMREQ;
                    break;
               }
          }

          /* write the extended memory address and page MSW */
          emaddr = (offset >> 16);
          Poke16(EMADDR, emaddr);

          /* write the extended memory address and page LSW */
          emaddr = offset;
```

```
            Poke16(EMADDR + 2, emaddr);

            /* write data samples */
            for (emdata = data; samples > 0; samples --)
                Poke16(EMDATA, *emdata++);

            /* release the memory */
            Poke16(MEMORY, memreg & 0xFFF0);

            break;
        }
        return error;
    }
```

4.7 Programming the Sequence Memory

The sequence generator is a very powerful tool that links waveform segments. The sequence logic is useful for generating long waveforms with looped sections. The looped waveform only has to be programmed once and the sequence logic will loop on this segment as many times as selected. When in sequenced mode, there is no loss of time between linked or looped segments. Each channel has its own dedicated sequence generator, even in master-slave mode. Sequence memory must be loaded to the generator before sequenced waveforms can be generated. The sequence memory contains 512 steps where each step specifies the following:

- Waveform address - This specifies the address in the waveform memory that will be output for specified step.
- Number of points - This specifies the number of points to output for this step.
- Loop Count - This specifies the loop count for this step.
- Mode - This specifies the way the instrument advances from the current step to the next step. Three modes are available:

  STEP — The sequence is advanced to the next step only when a valid trigger is received. The output of the MA1801A generates the current segment continuously until a trigger signal advances the sequence to the next step. If loops were specified, they are ignored in STEP mode.

  SING — The MA1801A idles between steps until a valid trigger signal is sensed. After outputting the specified number of loops, the output level idles at a DC level equal to the last point of the waveform. The sequencer will step to the next step in the sequence when it receives its next valid trigger.

  SING1 — The MA1801A idles between steps until a valid trigger signal is sensed. After a trigger, the generator outputs one waveform loop. Then, the output level idles at a DC level equal to the last point of the last generated waveform. If loops were programmed, the sequencer repeats this segment each time a trigger is received, until the number of loops specified for this step has been reached. After reaching this number, the sequencer advances to the next step in the sequence.

- Source - This specifies the advance trigger signal listed below:

  Signal1 low
  Signal1 high
  Signal1 falling
  Signal1 rising
  Signal2 low
  Signal2 high
  Signal2 falling
  Signal2 rising
  Primary not granted
  Secondary not granted
  Auto advance

- Jump Source - This specifies the jump trigger signal. When the advance trigger occurs and the jump trigger is true, the next sequence step will be set to the “Jump Sequence” instead of the next sequential step. One of the features of the MA1801A sequencer is the ability to specify a jump source. This feature allows the user to

perform “ping-pong” operation outputting data from one memory bank while updating the other. The jump source signal selections are listed below:

Signal1 low
Signal1 high
Signal1 falling
Signal1 rising
Signal2 low
Signal2 high
Signal2 falling
Signal2 rising
Primary granted
Primary not granted
Secondary granted
Secondary not granted
Timeout
Auto Jump

- Jump Sequence - This specifies the step number to jump to when the jump source is true.
- Flags - There are five sequence flags described below:

SeqFlag1	(Sequence Flag One) This flag can be set high or low and routed to the front panel SigOut signals and/or the backplane TRIGA/TRIGB signals.
SeqFlag2	(Sequence Flag Two) Sequence flag 2 can be set high or low and routed to the front panel SigOut signals and/or the backplane TRIGA/TRIGB signals.
SeqStop	(Sequence Stop) This flag indicates that the current step is the last step (1 - Stop after this step).
JumpRet	(Jump Return) This flag forces a jump step to return after the jump step is complete.
JumpImm	(Jump Immediate) This flag tells the sequencer to jump when the jump trigger is true even if there are loops remaining (1 - Jump immediately)

Each sequencer has two signals that are used for advancing and jumping. The sequence signals can be selected from a number of sources listed below:

1. Front Panel
   TRIG1A
   TRIG2A
   TRIG1B
   TRIG2B
2. M-Module Backplane
   TRIGA
   TRIGB
3. MA1801A Internal
   TIMERA
   TIMERB
   MARKER1A
   MARKER2A
   MARKER1B
   MARKER2B

The following code example will program the sequencer signals of the specified channel.

```
/*
     This function programs the sequence source signals
     of the specified MA1801A channel.
     Input:
        channel  (0 = channel A, 1 = channel B)
        signal1  (0 = TRIG1A, 1 = TRIG2A, 2 = TRIG1B,
                  3 = TRIG2B, 4 = MARKER1A, 5 = MARKER2A,
                  6 = MARKER1B, 7 = MARKER2B, 8 = TRIGA,
                  9 = TRIGB, 12 = TIMERA, 13 = TIMERB,
                  14 = High Level, 15 = Low Level)
        signal2  (0 = TRIG1A, 1 = TRIG2A, 2 = TRIG1B,
                  3 = TRIG2B, 4 = MARKER1A, 5 = MARKER2A,
```

```
                    6 = MARKER1B, 7 = MARKER2B, 8 = TRIGA,
                    9 = TRIGB, 12 = TIMERA, 13 = TIMERB,
                    14 = High Level, 15 = Low Level)
     Return Value:
          0
*/
int setSequenceSignal (short channel, short signal1, short signal2)
{
     unsigned short sequenceSig;

     /* get output signal register */
     Peek16(SEQSIG, &sequenceSig);

     /* program the signals of the selected channel */
     if (channel == CHANNEL_A)
          sequenceSig = (sequenceSig & 0xFF00) + (signal1 << 4) + signal1;
     else
          sequenceSig = (sequenceSig & 0xFF) + (signal1 << 12) + (signal1 << 8);

     /* write the output signal register */
     Poke16(SEQSIG, sequenceSig);

     return 0;
}
```

The following code example will program the sequence memory step of the specified channel.

```
#define MEMORY              0x2
#define EMADDR              0xC0
#define EMDATA              0xC4
/*
     This function programs the sequency memory
     of the specified MA1801A channel.
     Input:
          channel         (0 = channel A, 1 = channel B)
          step            (1 - 512)
          waveMemoryAddr  (0 - 0x7FFFC)
          wavePoints      (0 - 0xFFFFE)
          loop            (1 - 65535)
          mode            (1 = Step, 2 = Single, 3 = Single1)
          source          (0 = signal1 low, 1 = signal1 high, 2 = signal1 falling
                           3 = signal1 rising, 4 = signal2 low, 5 = signal2 high
                           6 = signal2 falling, 7 = signal2 rising, 9 = primary not granted
                           11 = secondary not granted, 14 = none, 15 = auto)
          jumpSource      (0 = signal1 low, 1 = signal1 high, 2 = signal1 falling
                           3 = signal1 rising, 4 = signal2 low, 5 = signal2 high
                           6 = signal2 falling, 7 = signal2 rising, 8 = primary granted,
                           9 = primary not granted, 10 = secondary granted,
                           11 = secondary not granted, 12 = timeout,
                           14 = unconditional, 15 = auto)
          jumpSequence    (1 - 512)
          flags           (Bit 0 = Sequence Flag 1, Bit 1 = Sequence Flag 2,
                           Bit 2 = Sequence Stop, Bit 3 = Jump Return,
                           Bit 4 = Jump Immediate)
     Return Value:
          0
*/
int setSequenceStep (short channel,
                     short step,
                     int waveMemoryAddr,
                     int wavePoints,
                     int loop,
                     short mode,
                     short source,
                     short jumpSource,
                     short jumpSequence,
                     short flags)
{
     int seqData[4] = {0,0,0,0};
     short i;
     int addr;

     if (channel == CHANNEL_A)
          /* select the sequence memory channel A */
          Poke16(MEMORY, 0x4000);
     else
          /* select the sequence memory channel B */
          Poke16(MEMORY, 0x5000);

     /* start filling seqData array with data */
     seqData[0] = (waveMemoryAddr / 4) & 0x7FFFF;
     seqData[1] = (wavePoints / 2) - 1;

     /* set the mode */
     switch (mode) {
          case SEQMODE_SINGLE:
               mode = 0;
               break;
          case SEQMODE_STEP:
               mode = 1;
               break;
          case SEQMODE_SINGLE1:
               mode = 2;
```

```
            break;
        default:
            mode = 0;
            break;
    }

    /* set the mode */
    seqData[1] += (mode << 28);

    /* set the source */
    seqData[1] += (source << 24);

    /* set the loop */
    seqData[2] += (loop - 1);

    /* set the jump source */
    seqData[1] += (jumpSource << 20);

    /* set the jump address */
    seqData[0] += ((jumpSequence - 1) << 19);

    /* set the flags */
    seqData[0] += ((flags & 0xF) << 28);
    seqData[1] += ((flags & 0x10) << 15);

    /* write the extended memory address and page MSW */
    addr = (step - 1) * 0x10;
    Poke16(EMADDR, addr >> 16);

    /* write the extended memory address and page LSW */
    Poke16(EMADDR + 2, addr & 0xFFFF);

    /* write sequence data */
    for (i = 0; i < 3; i++) {
        Poke16(EMDATA, seqData[i] >> 16);
        Poke16(EMDATA, seqData[i] & 0xFFFF);
    }

    return 0;
}
```

4.8 Setting the Internal Timer Period

The MA1801A has two internal timers that can be set to start, stop or advance waveforms (TIMERA and TIMERB). The internal timers are free-running generators, which are asynchronous with the main output and each other.

In firmware revision 1.11 and earlier, TIMERA is reset when channel A is started and TIMERB is reset when channel B is started.

In firmware revision 1.12 and later, if a timer is selected to stop a channel, it is reset when the channel is started. If a timer is selected as one of the sequence advance/jump signals, it is reset at the beginning of each sequence step.

The period of the internal trigger generators can be programmed from 20μs to 1.3107s. The default period is 100μs.

The following code example will program the specified timer.

```
#define TIMERB              0x38
#define TIMERA              0x3A
/*
    This function programs the trigger timer period
    of the specified MA1801A channel.
    Input:
        channel: (0 = channel A, 1 = channel B)
      triggerPeriod: (20e-6 - 1.3107)
    Return Value:
        0
*/
int setTimerPeriod (short channel, double triggerPeriod)
{
    unsigned short timer;
    int addr;

    /* convert trigger period */
    timer = (unsigned short) ((triggerPeriod / 20e-6) - 1);

    /* set the address */
    addr = (channel == CHANNEL_A)? TIMERA: TIMERB;

    /* write the burst control register */
    Poke16(addr, timer);

    return 0;
}
```

4.9 Setting the Operating Mode

Each of the MA1801A channels can be programmed to operate in one of four operating modes: continuous, triggered, gated, and burst. These modes are described below.

Continuous	In normal continuous mode, the selected waveform is generated continuously at the selected frequency, amplitude, and offset.
Triggered	In triggered mode, the MA1801Ais armed to generate one output waveform. The trigger circuit is sensitive to transitions on the selected trigger. Select between positive or negative transitions to trigger the instrument. The trigger level of the front panel TRIG1A and TRIG2A signals can be programmed to the desired threshold level. When triggered, the generator outputs one waveform cycle and remains idle at the last point of the waveform. The trigger signal can be selected from the front panel connector, the M-Module backplane TRIGA or TRIGB, or from an internal, programmable timing generator.
Gated	In gated mode, the MA1801A is armed to generate output waveforms as long as a gating signal is present. The gating signal can be selected from the same sources as the trigger signal. Unlike the triggered mode, the gated mode is level sensitive. When the gating signal is true, the waveform at the output connector will output continuously. When the gating signal goes false, the waveform at the output connector is first completed and the output goes to an idle state. The idle amplitude level, after the gating signal goes false, is the last point on the waveform.
Burst	The burst mode is an extension of the triggered mode where the MA1801A can be programmed to output a pre-determined number of waveforms (1 to 65535).

The trigger/gate signal can be selected from a number of sources listed below:

1. Front Panel
   TRIG1A
   TRIG2A
   TRIG1B
   TRIG2B
2. M-Module Backplane
   TRIGA
   TRIGB
3. MA1801A Internal
   TIMERA
   TIMERB
   MARKER1A
   MARKER2A
   MARKER1B
   MARKER2B

In addition to the trigger source, the trigger sensitivity can be select from the following:

1. Low level
2. High level
3. Falling edge
4. Rising edge

The trigger signal, whether it comes from the front panel, M-Module backplane or internal, has to pass through some electrical circuits. These circuits cause a small delay known as trigger response delay. The trigger response delay is 7 Master Clocks + 20ns. The trigger response delay can be minimized in the MA1801A by programming the master clock to at least 100MHz or higher and using the frequency divider to obtain the desired sample rate, see section 3.1.2.4. Trigger response delay is a factor that must be considered when applying a trigger signal. It defines how long it will take from a valid trigger edge to the moment that the output reacts.

These operating modes are identical for both channels. In master-slave mode, the operating modes are independent. The master channel provides the slave channels with a sample clock and start/stop signals. All channels will start generating waveforms when a valid trigger signal is received at the master channel.

Note that synchronization in master-slave mode does not require that all waveforms be equal in length. All waveforms will start at exactly the same time.

The following code example will program the operating mode selection and start trigger.

```
#define STARTCTRL          0xA
#define BURSTL00PB         0x2C
#define BURSTL00PA         0x2E
#define BURST              0x32
/*
     This function programs the start trigger source
     of the specified MA1801A channel.
     Input:
          channel: (0 = channel A, 1 = channel B)
          mode:  (0 = continuous, 1 = triggerd,
                  2 = gated, 3 = burst)
        startTrig: (0 = TRIG1A, 1 = TRIG2A, 2 = TRIG1B,
                    3 = TRIG2B, 4 = MARKER1A, 5 = MARKER2A,
                    6 = MARKER1B, 7 = MARKER2B, 8 = TRIGA,
                    9 = TRIGB, 12 = TIMERA, 13 = TIMERB,
                    14 = High Level, 15 = Low Level)
          slope:  (0 = low level, 1 = high level,
                   2 = falling edge, 3 = rising edge)
          burstCount: (1 - 65535)
     Return Value:
          0
*/
int setOperatingMode(short channel, short mode, short startTrig, short slope, int burstCount)
{
     unsigned short startctrl, burst;

     /* Read the start control register */
     Peek16(STARTCTRL, &startctrl);

     /* Read the burst control register */
     Peek16(BURST, &burst);

     /* set trigger based on new mode */
     if (mode == MODE_CONT) {
          /* set trigger low and start low if continuous mode */
          startTrig = 0xF;

          /* set burst continuous bit */
          burst |= (channel == CHANNEL_A)? 0x10: 0x1000;
     } else {
          /* adjust burst count based on mode */
          if (mode == MODE_BURST)
               burstCount--;
          else
               burstCount = 0;

          /* reset burst continuous bit */
          burst &= (channel == CHANNEL_A)? 0x1F00: 0x1F;

          /* set upper nibble burst count */
          burst |= (channel == CHANNEL_A)? ((burstCount & 0xF0000) >> 16): ((burstCount & 0xF0000) >> 8);

          /* add in trigger slope */
          startTrig += (slope << 4);

          /* adjust trigger to edge if not gate mode */
          if (mode != MODE_GATED)
               startTrig += 0x20;
     }

     /* write the burst control register */
     Poke16(BURST, burst);

     /* update and write the start control and burst count registers */
     if (channel == CHANNEL_A) {
          startctrl = (startctrl & 0xFFC0) + startTrig;
          Poke16(BURSTL00PA, burstCount);
     } else {
          startctrl = (startctrl & 0xC0FF) + (startTrig << 8);
          Poke16(BURSTL00PB, burstCount);
     }
     Poke16(STARTCTRL, startctrl);

     return 0;
}
```

4.10 Arming the Channel

Arming an MA1801A channel consists of programming the isolation relay to connect the output generator to the front panel and then setting the run register to allow the channel to respond to the selected start trigger.

In master-slave mode the slave channels should all be armed prior to the master channel to ensure proper synchronization.

In addition to the isolation relay, a 7-pole elliptic filter can be enabled for each channel.

The following code example will arm the specified channel and program the first step register.

```
#define RUNCTRL            0x0
#define STARTCTRL          0xA
#define FIRSTSEQB          0x3C
#define FIRSTSEQA          0x3E
/*
     This function connects and arms
     the specified MA1801A channel.
     Input:
          channel: (0 = channel A, 1 = channel B)
          outputMode:  (1 = Connect output without filter,
                        2 = Connect output with filter)
     Return Value:
          0
*/
int armChannel(short channel, short step, short outputMode)
{
     unsigned short runctrl, startctrl;

     /* get run register */
     Peek16(RUNCTRL, &runctrl);

     /* get start register */
     Peek16(STARTCTRL, &startctrl);

     /* update start and run control registers */
     if (channel == CHANNEL_A) {
          /* mask non selected channel and set arm bit of selected channel */
          runctrl = (runctrl & 0x100) + 0x1;

          /* set the output mode */
          startctrl = (startctrl & 0xFF3F) + (outputMode << 6);

          /* set the sequence step */
          Poke16(FIRSTSEQA, step - 1);
     } else {
          /* mask non selected channel and set arm bit of selected channel */
          runctrl = (runctrl & 1) + 0x100;

          /* set the output mode */
          startctrl = (startctrl & 0x3FFF) + (outputMode << 14);

          /* set the sequence step */
          Poke16(FIRSTSEQB, step - 1);
     }

     /* set start register */
     Poke16(STARTCTRL, startctrl);

     /* set run register */
     Poke16(RUNCTRL, runctrl);

     return 0;
}
```

4.11 Generating Manual Triggers

The MA1801A has five triggers described below:

START	The START signal is used to activate the sequencer and output the specified waveform. The START signal can only activate the sequence if the channel has been armed.
STOP	The STOP signal deactivates an active waveform.
RESTART	The RESTART signal reactivates a previously stopped waveform.
ADVANCE	The ADVANCE signal controls when the sequencer advances from one step to another.
JUMP	The JUMP signal controls whether the sequencer advances to the next sequential step or to the specified jump step.

The MA1801A allows all five triggers to be generated manually.

The following code example will generate a manual trigger to the specified channel.

```
#define RUNCTRL            0x0
/*
     This function generates a manual trigger to
     the specified MA1801A channel.
```

```
     Input:
          channel: (0 = channel A, 1 = channel B)
          trigger: (0 = start, 1 = stop, 2 = restart, 3 = advance, 4 = jump)
     Return Value:
          0
*/
int manualTrigger(short channel, short trigger)
{
     unsigned short runctrl;

     /* get run register */
     Peek16(RUNCTRL, &runctrl);

     switch (trigger) {
     case START_TRIG:
          /* set manual start if not running */
          if (channel == CHANNEL_A)
               if ((runctrl & 0xB) != 1)
                    break;
               else
                    runctrl = (runctrl & 0x101) + 4;
          else
               if ((runctrl & 0xB00) != 0x100)
                    break;
               else
                    runctrl = (runctrl & 0x101) + 0x400;

          /* set run register */
          Poke16(RUNCTRL, runctrl);

          break;
     case STOP_TRIG:
          /* set manual stop if running */
          if (channel == CHANNEL_A)
               if ((runctrl & 0x2) != 0x2)
                    break;
               else
                    runctrl = (runctrl & 0x101) + 0x2;
          else
               if ((runctrl & 0x200) != 0x200)
                    break;
               else
                    runctrl = (runctrl & 0x101) + 0x200;

          /* set run register */
          Poke16(RUNCTRL, runctrl);

          break;
     case RESTART_TRIG:
          /* set manual restart if stopped */
          if (channel == CHANNEL_A)
               if ((runctrl & 0xA) != 0x8)
                    break;
               else
                    runctrl = (runctrl & 0x101) + 0x2;
          else
               if ((runctrl & 0xA00) != 0x800)
                    break;
               else
                    runctrl = (runctrl & 0x101) + 0x200;

          /* set run register */
          Poke16(RUNCTRL, runctrl);

          break;
     case ADVANCE_TRIG:
          /* set manual advance */
          if (channel == CHANNEL_A)
               runctrl = (runctrl & 0x101) + 0x10;
          else
               runctrl = (runctrl & 0x0101) + 0x1000;

          /* set run register */
          Poke16(RUNCTRL, runctrl);

          break;
     case JUMP_TRIG:
          /* set manual jump */
          if (channel == CHANNEL_A)
               runctrl = (runctrl & 0x101) + 0x20;
          else
               runctrl = (runctrl & 0x101) + 0x2000;

          /* set run register */
          Poke16(RUNCTRL, runctrl);

          break;
     }

     return 0;
}
```

4.12 Assigning Stop/Restart Signals

Stop and restart signals are provided for each channel. The purpose of these signals is to allow intervention with the normal generation of output waveforms. Figure 4-2 below shows an example of how waveforms are affected using the stop and restart signals.

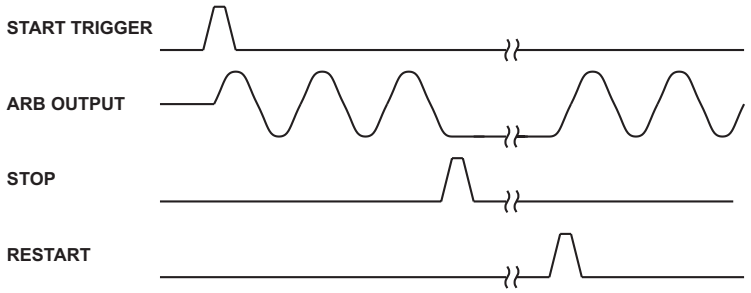

Figure 4-2 Stop/Restart Waveform

The stop/restart signals can be selected from a number of sources listed below:

1. Front Panel
   TRIG1A
   TRIG2A
   TRIG1B
   TRIG2B
2. M-Module Backplane
   TRIGA
   TRIGB
3. MA1801A Internal
   TIMERA
   TIMERB
   MARKER1A
   MARKER2A
   MARKER1B
   MARKER2B

In addition to the source, the stop/restart sensitivity can be selected from the following:

1. Low level
2. High level
3. Falling edge
4. Rising edge

In master-slave mode, transitions at the master channel will affect all channels that are slaved automatically. The stop and restart signals are only enabled during waveform output. However, the restart signal has no affect on the waveform before applying the stop signal. After the stop signal has stopped the waveform, the only signal that will resume waveform output is the restart signal. All other trigger signals during the stop interval are ignored.

The following code example will program the stop/restart signal of the specified channel.

```
/*
     This function sets the stop/start signals for
     the specified MA1801A channel.
     Input:
        channel:       (0 = channel A, 1 = channel B)
        stopSignal:    (0 = TRIG1A, 1 = TRIG2A, 2 = TRIG1B,
                        3 = TRIG2B, 4 = MARKER1A, 5 = MARKER2A,
                        6 = MARKER1B, 7 = MARKER2B, 8 = TRIGA,
                        9 = TRIGB, 12 = TIMERA, 13 = TIMERB,
                        14 = High Level, 15 = Low Level)
        stopSlope:     (0 = low level, 1 = high level,
                        2 = falling edge, 3 = rising edge)
        restartSignal: (0 = TRIG1A, 1 = TRIG2A, 2 = TRIG1B,
                        3 = TRIG2B, 4 = MARKER1A, 5 = MARKER2A,
                        6 = MARKER1B, 7 = MARKER2B, 8 = TRIGA,
                        9 = TRIGB, 12 = TIMERA, 13 = TIMERB,
```

```
                          14 = High Level, 15 = Low Level)
        restartSlope:   (0 = low level, 1 = high level,
                          2 = falling edge, 3 = rising edge)
    Return Value:
        0
*/
int setStopStartSignal (short channel, short stopSig, short stopSlope, short restartSig, short restartSlope)
{
    int addr;
    short stopStartReg;

    /* get address */
    if (channel == CHANNEL_A)
        addr = STOPRESTARTA;
    else
        addr = STOPRESTARTB;

    /* format register */
    stopStartReg = stopSig + (stopSlope << 4) + (restartSig << 8) + (restartSlope << 12);

    /* write the start/stop register */
    Poke16(addr, stopStartReg);

    return 0;
}
```

4.13 Setting the Output Signal Source

The MA1801A has two outputs per channel on the front panel as well as two M-Module backplane triggers.

The front panel outputs are called SIGOUT1A and SIGOUT2A for channel A and SIGOUT1B and SIGOUT2B for channel B. The M-Module backplane triggers are called TRIGA and TRIGB.

These signal are intended to be used for test verification, triggering other test equipment or as test signals.

The user can assign the following signals to SIGOUT1A or SIGOUT2A .

- Channel A Sync Signal
  SYNCA
- Channel A Waveform markers
  MARKER1A
  MARKER2A
- Channel A Sequence flags
  SEQFLAG1A
  SEQFLAG2A
- Internal Timers
  TIMERA
  TIMERB
- M-Module triggers
  TRIGA
  TRIGB
- Miscellaneous Signals
  TRIG1A After Input Buffer
  TRIG2A After Input Buffer
  EXTCLK After Input Buffer
  Sequence Active Flag
  Run Flag
  Advance Flag
  None, Set Low

The user can assign the following signals to SIGOUT1B or SIGOUT2B .

- Channel B Sync Signal
  SYNCB

- Channel B Waveform markers
  MARKER1B
  MARKER2B
- Channel A Sequence flags
  SEQFLAG1B
  SEQFLAG2B
- Internal Timers
  TIMERA
  TIMERB
- M-Module triggers
  TRIGA
  TRIGB
- Miscellaneous Signals
  TRIG1B After Input Buffer
  TRIG2B After Input Buffer
  EXTCLK After Input Buffer
  Sequence Active Flag
  Run Flag
  Advance Flag
  None, Set Low

The user can assign the following signals to TRIGA or TRIGB.

- Front panel trigger inputs
  TRIG1A
  TRIG2A
  TRIG1B
  TRIG2B
- Waveform markers
  MARKER1A
  MARKER2A
  MARKER1B
  MARKER2B
- Sequence flags
  SEQFLAG1A
  SEQFLAG2A
  SEQFLAG1B
  SEQFLAG2B
- Sync Signal
  SYNCA
  SYNCB
- None, Driver Disabled

The following code example will program the front panel output signals of the specified channel.

```
/*
     This function programs the front panel output signal
     of the specified MA1801A channel.
     Input:
        channel  (0 = channel A, 1 = channel B)
        sigOut1  (0 = SYNC, 1 = MARKER1, 2 = MARKER2,
                  3 = SEQFLAG1, 4 = SEQFLAG2, 5 = TIMERA,
                  6 = TIMERB, 7 = GTRIG1, 8 = GTRIG2,
                  9 = GEXTCLK, 10 = TRIGA, 11 = TRIGB,
                  12 = SEQACTIVE, 13 = RUN, 14 = ADVANCE,
                  15 = LOW)
        sigOut2  (0 = SYNC, 1 = MARKER1, 2 = MARKER2,
```

```
                    3 = SEQFLAG1, 4 = SEQFLAG2, 5 = TIMERA,
                    6 = TIMERB, 7 = GTRIG1, 8 = GTRIG2,
                    9 = GEXTCLK, 10 = TRIGA, 11 = TRIGB,
                    12 = SEQACTIVE, 13 = RUN, 14 = ADVANCE,
                    15 = LOW)
    Return Value:
         0
*/
int setFrontPanelOutput (short channel, short sigOut1, short sigOut2)
{
     unsigned short outputSig;

     /* get output signal register */
     Peek16(OUTPUTSIG, &outputSig);

     /* program the signals of the selected channel */
     if (channel == CHANNEL_A)
         outputSig = (outputSig & 0xFF00) + (sigOut2 << 4) + sigOut1;
     else
         outputSig = (outputSig & 0xFF) + (sigOut2 << 12) + (sigOut1 << 8);

     /* write the output signal register */
     Poke16(OUTPUTSIG, outputSig);

     return 0;
}
```

The following code example will program the M-Module backplane trigger signals.

```
/*
     This function programs the M-Module backplane output signal
     of the MA1801A module.
     Input:
        trigA  (0 = TRIG1A, 1 = TRIG2A, 2 = TRIG1B,
                3 = TRIG2B, 4 = MARKER1A, 5 = MARKER2A,
                6 = MARKER1B, 7 = MARKER2B, 8 = SEQFLAG1A,
                9 = SEQFLAG2A, 10 = SEQFLAG1B, 11 = SEQFLAG2B,
                12 = SYNCA, 13 = SYNCB, 14 = Not Used,
                15 = DISABLE)
        trigB  (0 = TRIG1A, 1 = TRIG2A, 2 = TRIG1B,
                3 = TRIG2B, 4 = MARKER1A, 5 = MARKER2A,
                6 = MARKER1B, 7 = MARKER2B, 8 = SEQFLAG1A,
                9 = SEQFLAG2A, 10 = SEQFLAG1B, 11 = SEQFLAG2B,
                12 = SYNCA, 13 = SYNCB, 14 = Not Used,
                15 = DISABLE)
    Return Value:
         0
*/
int setBackplaneOutput (short trigA, short trigB)
{
     unsigned short miscCtrl;

     /* get backplane output signal register */
     Peek16(MISCCTRL, &miscCtrl);

     /* program the signals */
     miscCtrl = (miscCtrl & 0xF00F) + (trigB2 << 8) + (trigA << 4);

     /* write the output signal register */
     Poke16(MISCCTRL, miscCtrl);

     return 0;
}
```

4.14 Programming Sync Pulse/Replace

Each channel of the MA1801A can generate a single TTL pulse for synchronizing other instruments (i.e., an oscilloscope) to the output waveform. The SYNC signal always appears at a fixed point relative to the waveform. The location of the signal along the waveform is programmable.

The validation event (source) for the sync pulse is selectable from a number of options described below:

Start Sequence	Generates a sync signal every time the waveform sequence is output, i.e. start trigger received.
Sequence Step	Generates a sync signal at the selected step.
Sequence Loop	Generates a sync signal at the selected step for every loop programmed at the specified step.
Single Loop	Generates a sync signal at the selected step and loop.

The sync pulse can be programed to occur at any offset relative to the selected source. The sync position offset can be set from 0 to 2097151 words after the sync source. Each word represents two waveform points.

The width of the sync pulse can be programmed to any width from 1 to 4095 words. Each word represents two waveform points.

The following code example will program the sync pulse of the specified channel.

```
/*
     This function program the sync signal of the specified
     MA1801A channel.
     Input:
         channel (0 = channel A, 1 = channel B)
         level   (0 = active high, 1 = active low)
         source  (0 = Start Sequence, 1 = Sequence Step,
                  2 = Sequence Loop, 3 = Single Loop)
         offset  (0 to 2097151 words)
         width   (1 to 4095 words)
        step    (1 to 512)
        loop    (1 to 65535)
     Return Value:
         0
*/
int setSyncOutput (short channel, short level, short source, int offset, short width, short step, int loop)
{
     int addr;
     unsigned short sync1;
     unsigned short sync2;
     unsigned short sync3;
     unsigned short sync4;

     /* set the address */
     addr = (channel == CHANNEL_A)? SYNCREGA: SYNCREGB;

     /* read the sync control 3 register to preserve replace bit */
     Peek16(addr + 4, &sync3);

     /* format the sync control registers */
     sync1 = ((step - 1) << 5) + ((offset >> 16) & 0x1F);
     sync2 = (offset & 0xFFFF);
     sync3 = (sync3 & 0x4000) + (level << 15) + (source << 12) + width;
     sync4 = loop - 1;

     /* program the sync controls */
     Poke16 (addr, sync1);
     Poke16 (addr + 2, sync2);
     Poke16 (addr + 4, sync3);
     Poke16 (addr + 6, sync4);

     return 0;
}
```

The MA1801A has a unique feature that allows the user to replace the current waveform data during a sync pulse with a different segment of waveform memory.

The following code example will program the sync pulse replace mode of the specified channel.

```
/*
     This function program the sync replace of the specified
     MA1801A channel.
     Input:
         channel        (0 = channel A, 1 = channel B)
         state          (0 = replase off, 1 = replace on)
         waveMemoryAddr (0 - 0x7FFFC)
     Return Value:
         0
*/
int setSyncReplace (short channel, short state, int waveMemoryAddr)
{
     int addr;
     unsigned short sync3;
     unsigned short sync5;
     unsigned short sync6;

     /* set the address */
     addr = (channel == CHANNEL_A)? SYNCREGA: SYNCREGB;

     /* read the sync control 3 register to program replace bit */
     Peek16(addr + 4, &sync3);

     /* format the sync control registers */
     sync3 = (sync3 & 0xBFFF) + (state << 14);
     sync5 = (waveMemoryAddr >> 16) & 0x7;
     sync6 = (waveMemoryAddr & 0xFFFF);

     /* program the sync controls */
     Poke16 (addr + 4, sync3);
     Poke16 (addr + 8, sync5);
     Poke16 (addr + 10, sync6);

     return 0;
}
```

4.15 Setting Channel Coupling

The MA1801A has two independent output channels. Each channel has its own sample clock generator, trigger inputs, sequence generator and control logic. There is no synchronization between channels unless the channel coupling is programmed. Channel coupling links one master channel with one or more slave channels. The generator is designed so that only channel A can become master. Channel B, when coupled to channel A, is designated as a slave channel. A slave channel receives its sample clock, triggers, sync and start/stop/restart inputs from the master channel. All other parameters can be programmed separately for the slave channel. On power-up, the internal reset routine sets all channels to operate independently.

Multiple MA1801As can be coupled through an inter-module cable. In this configuration channel A or channel B on the non-master MA1801A module can be linked as slave channels or run independently.

In master-slave mode, the following parameters are overridden by the master channel: sample clock, sequencer signals and start/stop/restart control. Waveform shape, sample clock divider, amplitude, filter, and output state remain unchanged.

The following code example will program the coupling mode of channel A and channel B for the specified module.

```
/*
     This function program the coupling mode of both
     MA1801A channels.
     Input:
          mode      (0 = internal, 1 = external)
          coupleA   (0 = couple mode off, 1 = couple mode on)
          coupleB   (0 = couple mode off, 1 = couple mode on)
     Return Value:
          0
*/
int setChannelCoupling (short mode, short coupleA, short coupleB)
{
     unsigned short miscCtrl;

     /* get coupling register */
     Peek16(MISCCTRL, &miscCtrl);

     /* reset the coupling bits */
     miscCtrl &= 0xFFF0;

     /* set the bits */
     if (mode == COUPLE_INT) {
          if (!coupleA == COUPLE_OFF && coupleB == COUPLE_ON)
               /* Set link B bit (A independent, B linked to A */
               miscCtrl |= 0x2;
          else if (coupleA == COUPLE_ON && coupleB == COUPLE_OFF)
               /* Set couple A and master bits (A master, B independent) */
               miscCtrl |= 0x5;
          else if (coupleA == COUPLE_ON && coupleB == COUPLE_ON)
               /* Set couple A, couple B and master bits (A master, B slave) */
               miscCtrl |= 0xD;
     } else if (mode == COUPLE_EXT) {
          if (coupleA == COUPLE_OFF && coupleB == COUPLE_ON)
               /* Set couple B bit (A independent, B slave) */
               miscCtrl |= 0x8;
          else if (coupleA == COUPLE_ON && coupleB == COUPLE_OFF)
               /* Set couple A bit (A slave, B independent) */
               miscCtrl |= 0x4;
          else if (coupleA == COUPLE_ON && coupleB == COUPLE_ON)
               /* Set couple A and couple B bits (A and B slave) */
               miscCtrl |= 0xC;
     }

     /* write the coupling register */
     Poke16 (MISCCTRL, miscCtrl);

     return 0;
}
```

4.16 Setting Sample Clock Divider

Each channel has a sample clock divider. The main usage of these dividers is in synchronized mode and sweeping. Note that the sample clock divider can accept integer values only from 1 through 4294967296. The dividers are active at all times. Upon power up, the sample clock dividers are set to 1. Changing the dividing ratio from 1 to another number modifies the sample clock from the value of the active sample clock to the sample clock divided by the clock divider value. For example, if channel B's active sample clock is 100 MHz and the clock divider is programmed to 25, the resultant sample clock frequency will be 4 MHz. Sample clock dividers affect the channel's sample clock regardless of whether the sample clock source is internal, external or coupled from the master channel.

The following code example will program the sample clock divider of the specified channel.

```
/*
     This function program the sample clock divider of the specified
     MA1801A channel.
     Input:
          channel        (0 = channel A, 1 = channel B)
          clockDivider   (1 to 4294967296)
     Return Value:
          0
*/
int setClockDivider (short channel, double clockDivider)
{
     unsigned int divide;
     int addr;

     /* convert to unsigned int 32 */
     divide = (unsigned int)clockDivider;

     /* get address */
     addr = (channel == CHANNEL_A)? SCDIVIDEA: SCDIVIDEB;

     /* write the register */
     Poke16 (addr, divide - 1);

     return 0;
}
```

4.17 Setting Start Trigger Delay

Channel synchronization controls the start trigger such that all channels start generating waveforms at the same instance, regardless of waveform shape and frequency settings. Due to variations in propagation delays, there is an unavoidable (but minimal) skew between the channels.

There are no internal provisions to correct skew between channels. However, skew can be corrected by adjusting the length of cable connected to the output connectors. One meter of 50Ω coax cable amounts to about ~4.85 ns of delay time. Use this general guideline to adjust the skew for your system. For example, if channel A lags channel B by 1.5 ns, delay the output of channel B by 1.5 ns. The output cable for channel B should be made roughly 0.31 meters longer than that of channel A to compensate for this skew.

While the initial skew between channels may be nulled only by using external cables, the MA1801A provides a register for programming an adjustable start trigger delay for each channel. The start trigger delay is adjusted in multiples of 2 sample clocks so that the higher the sample frequency, the better the start trigger delay resolution. For example, if the sample rate is 100 Khz, the resolution is 2 x (1 ÷ 100,000) = 20μs; for a sample rate of 125MHz, the resolution is 16ns.

The start trigger delay can be set from 0 to 65535 where each increment adds 2 sample clock periods to the start trigger delay.

The following code example will program the selected channels start trigger delay.

```
/*
     This function program the start trigger delay of the specified
     MA1801A channel.
     Input:
          channel       (0 = channel A, 1 = channel B)
          triggerDelay  (0 - 65535)
     Return Value:
          0
*/
int setTriggerDelay (short channel, int triggerDelay)
{
     int addr;
     unsigned short delay;

     /* set the address */
     addr = (channel == CHANNEL_A)? TRIGDLYA: TRIGDLYB;

     /* convert trigger delay from int to unsigned short */
     delay = (unsigned short) triggerDelay;

     /* write the threshold register */
     Poke16 (addr, delay);

     return 0;
}
```

4.18 Setting Input Thresholds

The MA1801A has four front panel inputs whose thresholds are programmable listed below.

1. TRIG1A (TA) -10.00V to +10.00V 5mV resolution
2. TRIG1B (TB) -10.00V to +10.00V 5mV resolution
3. EXTCLK (EC) -5.00V to +5.00V 2.5mV resolution
4. REFCLK (RC) -5.00V to +5.00V 2.5mV resolution

The following code example will program the selected inputs threshold.

```
/*
     This function program the input threshold of the specified
     MA1801A front panel signal.
     Input:
          signal    (0 = TRIG1A, 1 = TRIG1B, 2 = EXTCLK, 3 = REFCLK)
          threshold (-10.00 to +10.00)
     Return Value:
          0
*/
int setInputThreshold (short signal, double threshold)
{
     unsigned short runCtrl;
     short dacData = 0;
     int addr = 0;

     switch (signal) {
     case 0:   /* TRIG1A */
          addr = TRIG1AREF;
          dacData = 0x800 + (short)(threshold / 0.005);
          break;
     case 1:  /* TRIG1B */
          addr = TRIG1BREF;
          dacData = 0x4800 + (short)(threshold / 0.005);
          break;
     case 2:  /* EXTCLK */
          addr = ECLKREF;
          dacData = 0xC800 + (short)(threshold / 0.0025);
          break;
     case 3:  /* REFCLK */
          addr = RCLKREF;
          dacData = 0x8800 + (short)(threshold / 0.0025);
          break;
     default:
          break;
     }

     if (addr > 0) {
          /* write the dac register */
          Poke16(addr, dacData);


          /* update the reference DACs */
          /* read run control register */
          Peek16(RUNCTRL, &runCtrl);

          /* preserve arm bits and set the D/A update bit */
          runCtrl = (runCtrl & 0x101) | 0x8000;

          /* program the run control register */
          Poke16(RUNCTRL, runCtrl);

          /* query until D/A update complete */
          do {
               /* read run control register */
               Peek16(RUNCTRL, &runCtrl);
          } while ((runCtrl & 0x8000) == 0x8000);
     }

     return 0;
}
```

4.19 Generating Status/Event Interrupts

The following sections describes the MA1801A status and event registers as well as generating an event interrupt.

4.19.1 Status Register

The MA1801A status register allows the user to query the specified channels run status. The status is returned in a sixteen bit register where each bit indicates the following:

Bit 0	This bit indicates that the selected channel is waiting for the start trigger signal. (0 = not waiting, 1 = waiting)
Bit 1	This bit indicates that the selected channel is ready to accept the stop signal. (0 = not ready, 1 = ready)

Bit 2	This bit indicates that the selected channel is waiting for a restart signal. (0 = not waiting, 1 = waiting)
Bit 3	This bit indicates that the sequence step of the selected channel is in “single” or “single1” mode and waiting for the advance signal. (0 = not waiting, 1 = waiting)
Bit 4	This bit indicates that the sequence step of the selected channel is in “step” mode and waiting for the advance signal. (0 = not waiting, 1 = waiting)
Bit 5	This bit indicates that the current sequence step of the selected channel is enabled for jumps. (0 = not enabled, 1 = enabled)
Bit 6	This bit indicates that the current sequence step of the selected channel is enabled for jump with return. (0 = not enabled, 1 = enabled)
Bit 7	This bit indicates that the selected channels replace operation is in progress. (0 = no replace, 1 = replace)
Bit 8	This bit indicates that the selected channels sweep operation is in progress. (0 = no sweep, 1 = sweep)
Bit 9	This bit indicates that the MA1801A external clock input is less than 200Hz. (0 = no clock error, 1 = clock error)
Bit 10	This bit indicates that the selected channels primary memory is currently outputting a waveform. (0 = not busy, 1 = busy)
Bit 11	This bit indicates that the selected channels secondary memory is currently outputting a waveform. (0 = not busy, 1 = busy)
Bit 12	This bit indicates that the selected channels sequencer attempted to access the primary memory while it was not released. (0 = no error, 1 = error)
Bit 13	This bit indicates that the selected channels sequencer attempted to access the secondary memory while it was not released. (0 = no error, 1 = error)
Bit 14	This bit indicates that the coupling between channel A and channel B is out of sync. (0 = in sync, 1 = out of sync)
Bit 15	This bit indicates that the coupling between the master the and the slave channel(s) is out of sync. (0 = in sync, 1 = out of sync)

The following code example will return the status register of the specified channel.

```
/*
     This function returns the status register of the
     of the specified MA1801A channel.
     Input:
          channel: (0 = channel A, 1 = channel B)
     Output:
        status:
           Bit 0:  Waiting for Start; 0 = not waiting, 1 = waiting.
           Bit 1:  Stop Enabled; 0 = not enabled, 1 = enabled.
           Bit 2:  Waiting for restart; 0 = not waiting, 1 = waiting.
           Bit 3:  Waiting for advance SINGLE1 mode; 0 = not waiting,
                   1 = waiting.
           Bit 4:  Waitning for advance STEP mode; 0 = not waiting,
                   1 = waiting.
           Bit 5:  Jump enabled; 0 = disabled, 1 = enabled.
           Bit 6:  Gosub enabled; 0 = disabled, 1 = enabled.
           Bit 7:  Replace enabled; 0 = disabled, 1 = enabled.
           Bit 8:  Sweep; 0 = disabled, 1 = enabled.
           Bit 9:  Clock Slow; 0 = clock > 200Hz, 1 = clock < 200Hz.
           Bit 10: Primary Memory Status; 0 = not busy, 1 = busy.
           Bit 11: Secondary Memory Status; 0 = not busy, 1 = busy.
           Bit 12: Primary Error; 0 = no error, 1 = memory accessed
                   while not released.
           Bit 13: Secondary Error; 0 = no error, 1 = memory accessed
                   while not released.
           Bit 14: Link Error; 0 = no error, 1 = channel A/B out of
                   sync.
           Bit 15: Master/Slave error: 0 = no error, 1 = master/slave
                   chain out of sync.
     Return Value:
          0
*/
int getStatusRegister (short channel, unsigned short *status)
{
     int addr;

     /* set the address */
     addr = (channel == CHANNEL_A)? STATUSA: STATUSB;

     /* read the status register */
     Peek16(addr, status);

     return 0;
}
```

4.19.2 Event Register

The MA1801A event register allows the user to query the specified channels event status. The event register contains latched data and is cleared when read. The register is returned in a sixteen bit register where each bit indicates the following:

Bit 0	(Start) This bit indicates that the selected channel start occurred. (0 = no start, 1 = start)
Bit 1	(Stop) This bit indicates that the selected channel stop occurred. (0 = no stop, 1 = stop)
Bit 2	(Restart) This bit indicates that selected channel restart occurred. (0 = no restart, 1 = restart)
Bit 3	(Single Advance Occurred) This bit indicates that a sequence step of the selected channel “single” or “single1” mode received an advance. (0 = no advance, 1 = advance)
Bit 4	This bit indicates that a sequence step of the selected channel in “step” mode received an advance. (0 = no advance, 1 = advance)
Bit 5	This bit indicates that the selected channel jump occurred. (0 = no jump, 1 = jump)
Bit 6	This bit indicates that the selected channel jump with return occurred. (0 = no gosub, 1 = gosub)
Bit 7	This bit indicates that the selected channel replace operation occurred. (0 = no replace, 1 = replace)
Bit 8	This bit indicates that the selected channel sweep operation occurred. (0 = no sweep, 1 = sweep)
Bit 9	This bit indicates that the MA1801A external clock input was less than 200Hz. (0 = no clock error, 1 = clock error)
Bit 10	This bit indicates that the sequencer of the selected channel attempted to access the primary memory when it was not released. (0 = no error, 1 = error)
Bit 11	This bit indicates that the sequencer of the selected channel attempted to access the secondary memory when it was not released. (0 = no error, 1 = error)
Bit 12	This bit indicates that an advance timeout occurred on the selected channel. (0 = no timeout, 1 = timeout)
Bit 14	This bit indicates a sync error between channel A and B. (0 = no error, 1 = error)
Bit 15	This bit indicates a sync error in master/slave mode. (0 = no error, 1 = error)

The following code example will return the event register of the specified channel.

```
/*
     This function returns the event register of the
     of the specified MA1801A channel.  A one indicates
     the event occurred and is cleared after reading.
     Input:
         channel: (0 = channel A, 1 = channel B)
     Output:
        event:
           Bit 0:  Start Signal.
           Bit 1:  Stop Signal.
           Bit 2:  Restart Signal.
           Bit 3:  Single Advance Signal.
           Bit 4:  Step Advance Signal.
           Bit 5:  Jump Operation.
           Bit 6:  Gosub Operation.
           Bit 7:  Replace Operation.
           Bit 8:  Sweep Operation.
           Bit 9:  Clock Error.
           Bit 10: Primary Error.
           Bit 11: Secondary Error.
           Bit 12: Trigger Timeout.
           Bit 13: Not Used.
           Bit 14: Link Error.
           Bit 15: Master/Slave Error.
     Return Value:
         0
*/
int getEventRegister (short channel, unsigned short *event)
{
     int addr;

     /* set the address */
     addr = (channel == CHANNEL_A)? EVENTA: EVENTB;

     /* read the event register */
     Peek16(addr, event);

     return 0;
```

```
}
```

4.19.3 Event Enable (Interrupts)

The event enable register allows the user to generate an M-Module interrupt on any of the MA1801A event bits. Figure 4-3 illustrates the relationship between the event register, event enable and interrupts.

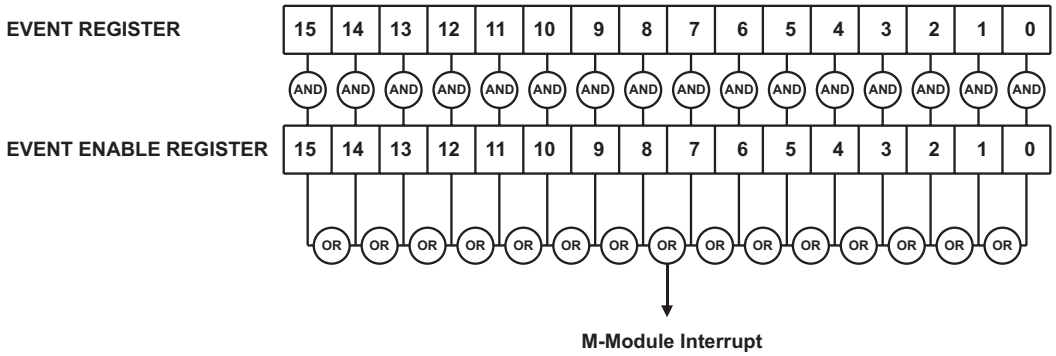

Figure 4-3 Event Enable Interrupts

The following code example will program the event enable register of the specified channel.

```
/*
     This function programs the event enable register of the
     of the specified MA1801A channel.  A one enables the
     event to generate an M-Module interrupt.
         channel: (0 = channel A, 1 = channel B)
     Output:
        enable:
           Bit 0:  Start Signal.
           Bit 1:  Stop Signal.
           Bit 2:  Restart Signal.
           Bit 3:  Single Advance Signal.
           Bit 4:  Step Advance Signal.
           Bit 5:  Jump Operation.
           Bit 6:  Gosub Operation.
           Bit 7:  Replace Operation.
           Bit 8:  Sweep Operation.
           Bit 9:  Clock Error.
           Bit 10: Primary Error.
           Bit 11: Secondary Error.
           Bit 12: Trigger Timeout.
           Bit 13: Not Used.
           Bit 14: Link Error.
           Bit 15: Master/Slave Error.
     Return Value:
         0
*/
int setEventEnable (short channel, unsigned short enable)
{
     int addr;

     /* set the address */
     addr = (channel == CHANNEL_A)? ENABLEA: ENABLEB;

     /* write the event enable register */
     Poke16(addr, enable);

     return 0;
}
```

4.19.4 Sequencer Address Status

The sequencer address status returns the current sequence step number and arbitrary memory address. These values can be used to determine the waveform position when a stop occurs.

The following code example will return the sequencer address status of the specified channel.

```
/*
     This function returns the current sequencer address
     status of the specified MA1801A channel.
     Input:
         channel: (0 = channel A, 1 = channel B)
     Output:
        sequenceStep
        memoryAddr
```

```
     Return Value:
          0
*/
int getSequencerAddrStatus (short channel, short *sequenceStep, int *memoryAddr)
{
     int addr;
     unsigned short datalsb;
     unsigned short datamsb;

     /* set the address */
     addr = (channel == CHANNEL_A)? SEQSTATUSA: SEQSTATUSB;

     /* read the sequence status registers */
     Peek16(addr, &datalsb);
     Peek16(addr + 2, &datamsb);

     /* format the output data */
     *sequenceStep = (datalsb >> 3) & 0x1FF;
     *memoryAddr = ((datalsb & 0x7) << 16) + datamsb;

     return 0;
}
```

4.20 Setting Trigger Timeout

The MA1801A timeout allows the user to program a timeout value from 20µs to 1.3107s in 20µs increments. The timeout signal indicates that the specified advance trigger did not occur within the specified time. The timeout signal is routed to the event register bit 12. It can also be selected as a jump source trigger in the sequence memory.

The following code example will program the trigger timeout of the specified channel.

```
/*
     This function programs the trigger timeout period
     of the specified MA1801A channel.
     Input:
         channel: (0 = channel A, 1 = channel B)
        triggerTimeout: (20e-6 - 1.3107)
     Return Value:
          0
*/
int setTriggerTimeout (short channel,  double triggerTimeout)
{
     unsigned short timer;
     int addr;

     /* convert trigger period */
     timer = (unsigned short) ((triggerTimeout / 20e-6) - 1);

     /* set the address */
     addr = (channel == CHANNEL_A)? TIMEOUTA: TIMEOUTB;

     /* write the burst control register */
     Poke16(addr, timer);

     return 0;
}
```

5 Register Operation Examples

This section includes typical arbitrary waveform generator examples utilizing the functions listed in section 4. All the examples are listed in appendix C.

Figure 5-1 illustrates a typical flowchart for programming the MA1801A.

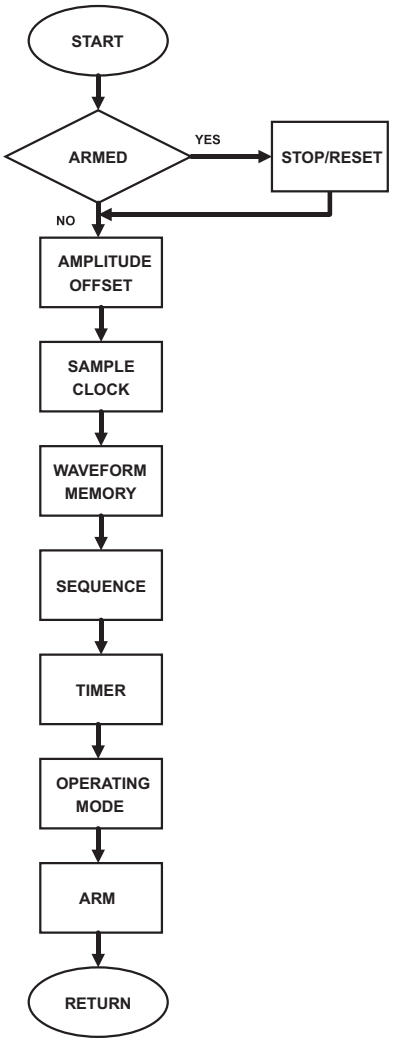

Figure 5-1 Example One Flow Chart

5.1 Example 1: Single Waveform Triggered

This example will program a waveform to output in triggered mode using the internal TIMERA as the trigger every 20μs.

The waveform is a 360 point sine^3 wave with an amplitude of 5V and an offset of 2.5V. The sample rate will be 75MHz giving a waveform frequency of:

Freq = SampleRate / Points
Freq = 75MHz / 360
Freq = 208.33KHz

5.1.1 Example 1 Main

The following code is the main body for example 1.

```
/*
     This function programs a single 360 point waveform on channel A
     and runs it in triggered mode using timerA as the trigger.

      sample clock: 75MHz
      waveform:     360 point sine^3
      amplitude:    5V
      offset:       2.5V
      mode:         triggered, TIMERA (20us)
*/
int example1 (void)
{
     int i, status;
     unsigned short array[360];
     double wData, pi = 3.1415926;

     /* get channel A status to check if armed */
     status = getStatus(CHANNEL_A);

     /* reset channel A if armed */
     if (status & 1)
         stopChannel(CHANNEL_A, RESET);

     /* set 5V amplitude and 2.5V offset */
     setGainAndOffset(CHANNEL_A, 5.0, 2.5);

     /* program the sample clock to 75MHz */
     setSampleClock(CHANNEL_A, INTERNAL, 75e6, INTERNAL);

     /* generat sine^3 wave */
     for (i = 0; i < 360; i++) {
         wData = sin (i * pi/180);
         array[i] = (unsigned short) ((8190.0 * wData * wData * wData)+8191);
     }

     /* upload the waveform to the MA1801A */
     setWavformMemory(CHANNEL_A, PRIMARY, 2048, 360, array);

     /* program the sequence to output the waveform */
     setSequenceStep (CHANNEL_A, 1, 2048, 360, 1, SEQMODE_SINGLE, SEQ_AUTO, SEQ_NONE, 1, 0x4);

     /* program TIMERA for a 20us period */
     setTimerPeriod (CHANNEL_A, 20e-6);

     /* set the operating mode and start trigger */
     setOperatingMode(CHANNEL_A, MODE_TRIG, SIG_INTA, SLOPE_HIGH, 2);

     /* disable start/stop by setting source to low and slope high */
     setStopStartSignal (CHANNEL_A, SIG_LOW, SLOPE_HIGH, SIG_LOW, SLOPE_HIGH);

     /* arm the channel */
     armChannel(CHANNEL_A, 1, CONNECT);

     return 0;
}
```

5.1.2 Example 1 Waveform

Example 1 will generate the waveform depicted below:

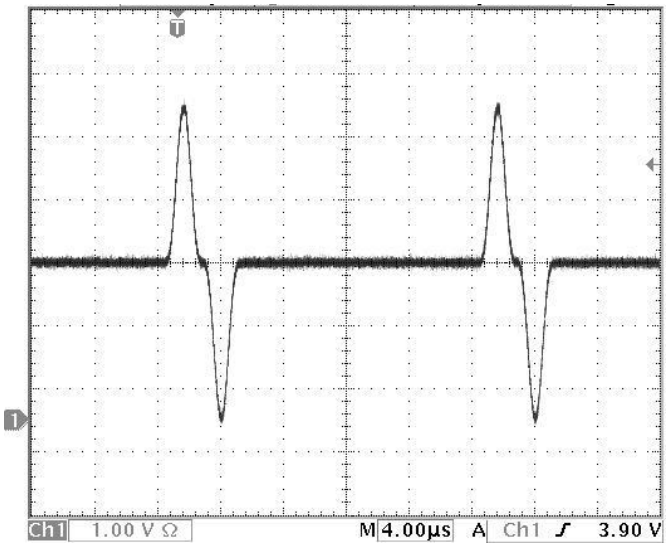

Figure 5-3 Example 1 Waveform

5.1.3 Example 1 Burst Mode Exercise

Change the “setOperatingMode” function call from:

setOperatingMode(CHANNEL_A, MODE_TRIG, START_INTA, SLOPE_HIGH, 1);

To:

setOperatingMode(CHANNEL_A, MODE_BURST, START_INTA, SLOPE_HIGH, 2);

Changing the operating mode from triggered to burst will result in the following waveform:

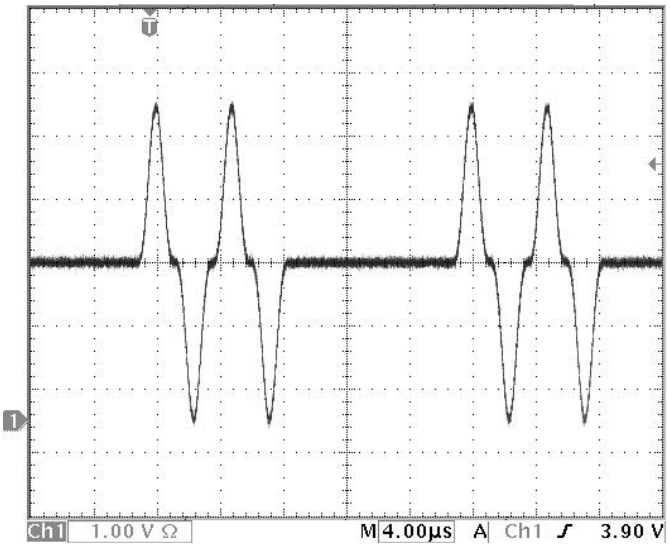

Figure 5-2 Example 1 Burst Mode

5.2 Example 2: Multiple Waveforms with Manual Trigger

This example will program two waveforms to output from a manual trigger.

Waveform 1 is a 720 point sine wave and waveform 2 is a 360 point triangle wave. The amplitude is 2V and offset 0V. The sample rate will be 125MHz giving a waveform 1 frequency of:

Freq = 125MHz / 720
Freq = 173.61KHz

And waveform 2 frequency of:

Freq = 125MHz / 360
Freq = 347.222KHz

5.2.1 Example 2 Main

The following code is the main body for example 2.

```
/*
     This function programs a two waveforms on channel A
     and runs them in trigger mode using manual as the
     trigger.

      sample clock: 125MHz
      waveform1:    720 point sine
      waveform2:    360 point triangle
      amplitude:    2V
      offset:       0V
      mode:         triggered, manual
*/
int example2 (void)
{
     int i, status;
     unsigned short array[720];
     double count, wData, pi = 3.1415926;

     /* get channel A status to check if armed */
     status = getStatus(CHANNEL_A);

     /* reset channel A if armed */
     if (status & 1)
         stopChannel(CHANNEL_A, RESET);

     /* set 2V amplitude and 0V offset */
     setGainAndOffset(CHANNEL_A, 2.0, 0.0);

     /* program the sample clock to 75MHz */
     setSampleClock(CHANNEL_A, INTERNAL, 125e6, INTERNAL);

     /* generat sine wave */
     for (i = 0, count = 0.0; count < 360.0; i++, count += 0.5) {
         wData = sin (count * pi/180);
         array[i] = (unsigned short) ((8190.0 * wData)+8191);
     }

     /* upload the waveform to the MA1801A */
     setWavformMemory(CHANNEL_A, PRIMARY, 2048, 720, array);

     /* generat triangle wave */
     for (i = 0; i <= 90; i++)
         array[i] = (unsigned short) (i * 8192.0 / 90.0) + 8191;
     for (i = 91; i <= 270; i++)
         array[i] = (unsigned short) ((180 - i) * 8192.0 / 90.0) + 8192;
     for (i = 271; i < 360; i++)
         array[i] = (unsigned short) ((i - 360) * 8192.0 / 90.0) + 8192;

     /* upload the waveform to the MA1801A */
     setWavformMemory(CHANNEL_A, PRIMARY, 5000, 360, array);

     /* program the sequence step 1 to output the waveform1 */
     setSequenceStep (CHANNEL_A, 1, 2048, 720, 1, SEQMODE_SINGLE, SEQ_AUTO, SEQ_NONE, 1, 0x0);

     /* program the sequence step 2 to output the waveform1 */
     setSequenceStep (CHANNEL_A, 2, 5000, 360, 1, SEQMODE_SINGLE, SEQ_AUTO, SEQ_NONE, 1, 0x4);

     /* set the operating mode and start trigger */
     setOperatingMode(CHANNEL_A, MODE_TRIG, SIG_LOW, SLOPE_HIGH, 1);

     /* disable start/stop by setting source to low and slope high */
     setStopStartSignal (CHANNEL_A, SIG_LOW, SLOPE_HIGH, SIG_LOW, SLOPE_HIGH);

     /* arm the channel */
     armChannel(CHANNEL_A, 1, CONNECT);

     /* manualy start the waveform */
     manualTrigger(CHANNEL_A, START_TRIG);

     return 0;
}
```

5.2.2 Example 2 Waveform

Example 2 will generate the waveform depicted below:

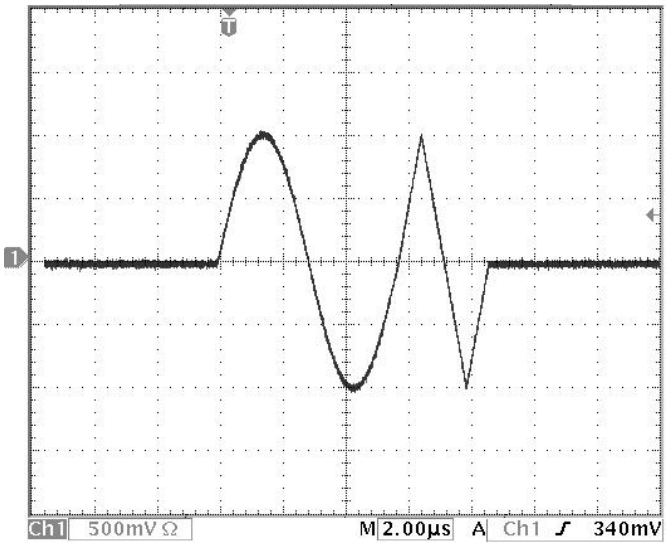

Figure 5-4 Example 2 Multiple Waveform

5.3 Example 3: Continuous Waveform with Stop/Start

This example will use the same waveforms from example 2 but will run them continuous. The stop/start signal will be set to internal TIMERA programmed to 20µs.

5.3.1 Example 3 Main

The following code is the main body for example 3.

```
/*
      This function programs a two waveforms on channel A
      and runs them in continuous mode.  Start/Stop set to the
      rising edge of timer A (20us).

        sample clock: 50MHz
        waveform1:    720 point sine
        waveform2:    360 point triangle
        amplitude:    2V
        offset:       0V
        mode:         continuous
        start/stop:    Rising edge TIMERA = 20us
*/
int example3 (void)
{
      int i, status;
      unsigned short array[720];
      double count, wData, pi = 3.1415926;

      /* get channel A status to check if armed */
      status = getStatus(CHANNEL_A);

      /* reset channel A if armed */
      if (status & 1)
          stopChannel(CHANNEL_A, RESET);

      /* set 2V amplitude and 0V offset */
      setGainAndOffset(CHANNEL_A, 2.0, 0.0);

      /* program the sample clock to 75MHz */
      setSampleClock(CHANNEL_A, INTERNAL, 125e6, INTERNAL);

      /* generat sine wave */
      for (i = 0, count = 0.0; count < 360.0; i++, count += 0.5) {
          wData = sin (count * pi/180);
          array[i] = (unsigned short) ((8190.0 * wData)+8191);
      }

      /* upload the waveform to the MA1801A */
      setWavformMemory(CHANNEL_A, PRIMARY, 2048, 720, array);
```

```
        /* generat triangle wave */
        for (i = 0; i <= 90; i++)
            array[i] = (unsigned short) (i * 8192.0 / 90.0) + 8191;
        for (i = 91; i <= 270; i++)
            array[i] = (unsigned short) ((180 - i) * 8192.0 / 90.0) + 8192;
        for (i = 271; i < 360; i++)
            array[i] = (unsigned short) ((i - 360) * 8192.0 / 90.0) + 8192;

        /* upload the waveform to the MA1801A */
        setWavformMemory(CHANNEL_A, PRIMARY, 5000, 360, array);

        /* program the sequence step 1 to output the waveform1 */
        setSequenceStep (CHANNEL_A, 1, 2048, 720, 1, SEQMODE_SINGLE, SEQ_AUTO, SEQ_NONE, 1, 0x0);

        /* program the sequence step 2 to output the waveform1 */
        setSequenceStep (CHANNEL_A, 2, 5000, 360, 1, SEQMODE_SINGLE, SEQ_AUTO, SEQ_NONE, 1, 0x4);

        /* set the operating mode and start trigger */
        setOperatingMode(CHANNEL_A, MODE_CONT, SIG_LOW, SLOPE_HIGH, 1);

        /* program TIMERA for a 20us period */
        setTimerPeriod (CHANNEL_A, 20e-6);

        /* set the start/stop source to rising edge timerA */
        setStopStartSignal (CHANNEL_A, SIG_INTA, SLOPE_RISE, SIG_INTA, SLOPE_RISE);

        /* arm the channel */
        armChannel(CHANNEL_A, 1, CONNECT);

        return 0;
}
```

5.3.2 Example 3 Waveform

Example 3 will generate the waveform similar to the one depicted below using single trigger on the oscilloscope:

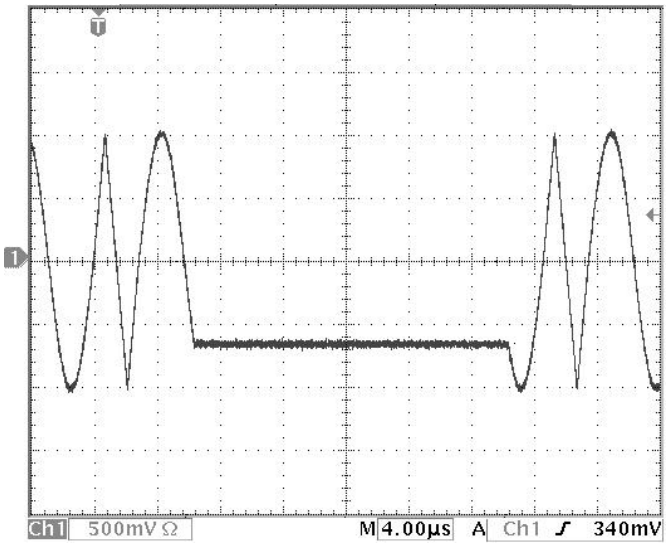

Figure 5-5 Example 3 Stop/Restart Waveform

5.4 Example 4: Sync Pulse Output

This example will use the same waveforms from example 2 but will run them continuous. The sync pulse will be programmed to output a 180 sample pulse at the beginning of sequence step 1.

5.4.1 Example 4 Main

The following code is the main body for example 4.

```
/*
      This function programs a two waveforms on channel A
      and runs them in continuous mode.  Sync pulse set to step
      mode and routed to sigout1a.

        sample clock: 50MHz
        waveform1:    720 point sine
```

```
        waveform2:    360 point triangle
        amplitude:    2V
        offset:       0V
        mode:         continuous
        sync pulse:   step 1, offset 0 for 90 words (180 samples)
*/
int example4 (void)
{
     int i, status;
     unsigned short array[720];
     double count, wData, pi = 3.1415926;

     /* get channel A status to check if armed */
     status = getStatus(CHANNEL_A);

     /* reset channel A if armed */
     if (status & 1)
         stopChannel(CHANNEL_A, RESET);

     /* set 2V amplitude and 0V offset */
     setGainAndOffset(CHANNEL_A, 2.0, 0.0);

     /* program the sample clock to 75MHz */
     setSampleClock(CHANNEL_A, INTERNAL, 125e6, INTERNAL);

     /* generat sine wave */
     for (i = 0, count = 0.0; count < 360.0; i++, count += 0.5) {
         wData = sin (count * pi/180);
         array[i] = (unsigned short) ((8190.0 * wData)+8191);
     }

     /* upload the waveform to the MA1801A */
     setWavformMemory(CHANNEL_A, PRIMARY, 2048, 720, array);

     /* generat triangle wave */
     for (i = 0; i <= 90; i++)
         array[i] = (unsigned short) (i * 8192.0 / 90.0) + 8191;
     for (i = 91; i <= 270; i++)
         array[i] = (unsigned short) ((180 - i) * 8192.0 / 90.0) + 8192;
     for (i = 271; i < 360; i++)
         array[i] = (unsigned short) ((i - 360) * 8192.0 / 90.0) + 8192;

     /* upload the waveform to the MA1801A */
     setWavformMemory(CHANNEL_A, PRIMARY, 5000, 360, array);

     /* program the sequence step 1 to output the waveform1 */
     setSequenceStep (CHANNEL_A, 1, 2048, 720, 1, SEQMODE_SINGLE, SEQ_AUTO, SEQ_NONE, 1, 0x0);

     /* program the sequence step 2 to output the waveform1 */
     setSequenceStep (CHANNEL_A, 2, 5000, 360, 1, SEQMODE_SINGLE, SEQ_AUTO, SEQ_NONE, 1, 0x4);

     /* set the operating mode and start trigger */
     setOperatingMode(CHANNEL_A, MODE_CONT, SIG_LOW, SLOPE_HIGH, 1);

     /* set the sync pulse to active low, step mode, offset 0, width 90, step 1 */
     setSyncOutput (CHANNEL_A, SYNC_LOW, SYNC_STEP, 0, 90, 1, 0);

     /* set the start/stop source to rising edge timerA */
     setFrontPanelOutput (CHANNEL_A, FPO_SYNC, FPO_LOW);

     /* arm the channel */
     armChannel(CHANNEL_A, 1, CONNECT);

     return 0;
}
```

5.4.2 Example 4 Waveform

Example 4 will generate the waveform similar to the one depicted below using single trigger on the oscilloscope:

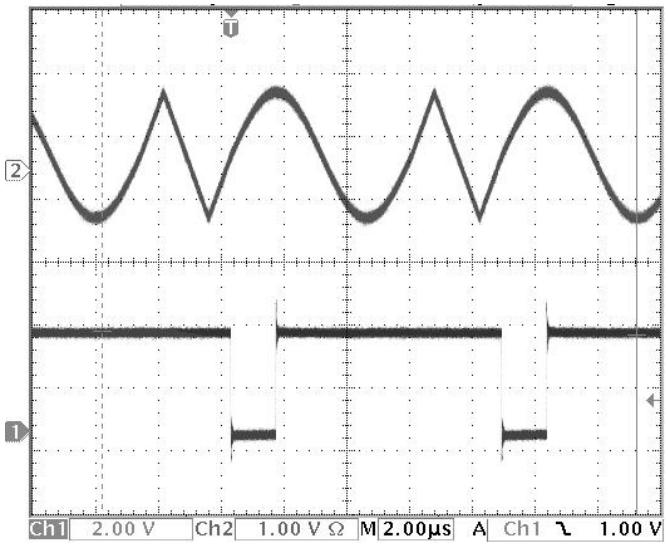

Figure 5-7 Example 4 Sync Waveform

5.4.3 Example 4 Sync Replace Exercise

Add the “setSyncReplace” function call after the “setSyncOutput” function:

```
setSyncReplace (CHANNEL_A, REPLACE_ON, 5000);
```

Enabling the sync replace will result in the following waveform:

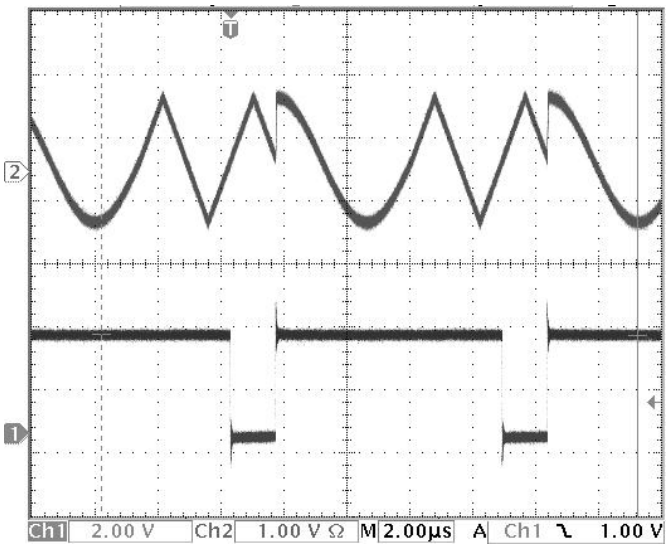

Figure 5-6 Example 4 Sync Replace Waveform

5.5 Example 5: Multi Channel Waveform

This example will program the waveform from example 1 on channel A and the waveform from example 2 on channel B. Channel A will output in triggered mode using the internal TIMERA as the trigger every 20µs. Channel B will output in triggered mode using a manual trigger.

5.5.1 Example 5 Main

The following code is the main body for example 5.

```
/*
     This function programs a single waveform on channel A
     and runs it in triggered mode using timerA as the trigger.
     Channel B is programed with two waveforms in triggerd mode
     using TIMERB as the trigger.

     Channel A
      sample clock: 75MHz
      waveform:     360 point sine^3
      amplitude:    5V
      offset:       2.5V
      mode:         triggered, TIMERA (20us)
    Channel B
      sample clock: 125MHz
      waveform1:    720 point sine
      waveform2:    360 point triangle
      amplitude:    2V
      offset:       0V
      mode:         triggered, manual
*/
int example5 (void)
{
     int i, status;
     unsigned short array[720];
     double count, wData, pi = 3.1415926;

     /* get channel A status to check if armed */
     status = getStatus(CHANNEL_A);

     /* reset channel A if armed */
     if (status & 1)
         stopChannel(CHANNEL_A, RESET);

     /* set 5V amplitude and 2.5V offset */
     setGainAndOffset(CHANNEL_A, 5.0, 2.5);

     /* program the sample clock to 75MHz */
     setSampleClock(CHANNEL_A, INTERNAL, 75e6, INTERNAL);

     /* generat sine^3 wave */
     for (i = 0; i < 360; i++) {
         wData = sin (i * pi/180);
         array[i] = (unsigned short) ((8190.0 * wData * wData * wData)+8191);
     }

     /* upload the waveform to the MA1801A */
     setWavformMemory(CHANNEL_A, PRIMARY, 2048, 360, array);

     /* program the sequence to output the waveform */
     setSequenceStep (CHANNEL_A, 1, 2048, 360, 1, SEQMODE_SINGLE, SEQ_AUTO, SEQ_NONE, 1, 0x4);

     /* program TIMERA for a 20us period */
     setTimerPeriod (CHANNEL_A, 20e-6);

     /* set the operating mode and start trigger */
     setOperatingMode(CHANNEL_A, MODE_TRIG, SIG_INTA, SLOPE_HIGH, 2);

     /* disable start/stop by setting source to low and slope high */
     setStopStartSignal (CHANNEL_A, SIG_LOW, SLOPE_HIGH, SIG_LOW, SLOPE_HIGH);

     /* disable the sync replace mode */
     setSyncReplace (CHANNEL_A, REPLACE_OFF, 5000);

     /* arm the channel */
     armChannel(CHANNEL_A, 1, CONNECT);

     /* get channel B status to check if armed */
     status = getStatus(CHANNEL_B);

     /* reset channel B if armed */
     if (status & 1)
         stopChannel(CHANNEL_B, RESET);

     /* set 2V amplitude and 0V offset */
     setGainAndOffset(CHANNEL_B, 2.0, 0.0);

     /* program the sample clock to 75MHz */
     setSampleClock(CHANNEL_B, INTERNAL, 125e6, INTERNAL);

     /* generat sine wave */
     for (i = 0, count = 0.0; count < 360.0; i++, count += 0.5) {
         wData = sin (count * pi/180);
         array[i] = (unsigned short) ((8190.0 * wData)+8191);
     }

     /* upload the waveform to the MA1801A */
     setWavformMemory(CHANNEL_B, PRIMARY, 0, 720, array);
```

```
        /* generat triangle wave */
        for (i = 0; i <= 90; i++)
            array[i] = (unsigned short) (i * 8192.0 / 90.0) + 8191;
        for (i = 91; i <= 270; i++)
            array[i] = (unsigned short) ((180 - i) * 8192.0 / 90.0) + 8192;
        for (i = 271; i < 360; i++)
            array[i] = (unsigned short) ((i - 360) * 8192.0 / 90.0) + 8192;

        /* upload the waveform to the MA1801A */
        setWavformMemory(CHANNEL_B, PRIMARY, 2000, 360, array);

        /* program the sequence step 1 to output the waveform1 */
        setSequenceStep (CHANNEL_B, 1, 0, 720, 1, SEQMODE_SINGLE, SEQ_AUTO, SEQ_NONE, 1, 0x0);

        /* program the sequence step 2 to output the waveform1 */
        setSequenceStep (CHANNEL_B, 2, 2000, 360, 1, SEQMODE_SINGLE, SEQ_AUTO, SEQ_NONE, 1, 0x4);

        /* set the operating mode and start trigger */
        setOperatingMode(CHANNEL_B, MODE_TRIG, SIG_LOW, SLOPE_HIGH, 1);

        /* disable start/stop by setting source to low and slope high */
        setStopStartSignal (CHANNEL_B, SIG_LOW, SLOPE_HIGH, SIG_LOW, SLOPE_HIGH);

        /* arm the channel */
        armChannel(CHANNEL_B, 1, CONNECT);

        /* manualy start the waveform */
        manualTrigger(CHANNEL_B, START_TRIG);

        return 0;
}
```

5.5.2 Example 5 Waveform

Example 5 will generate the waveform depicted below:

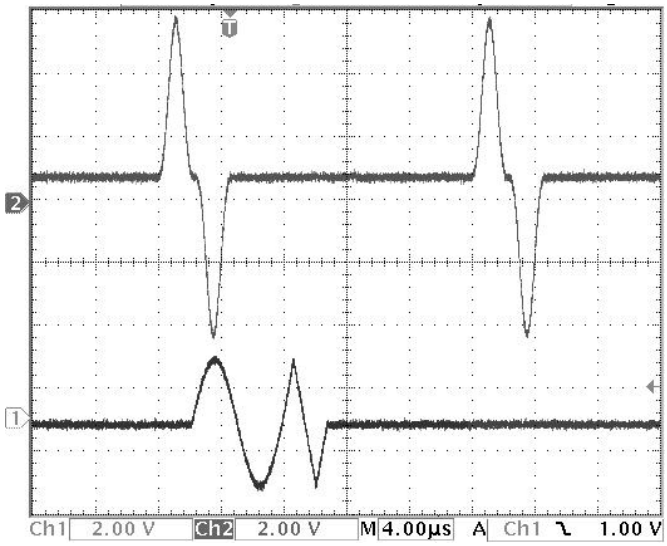

Figure 5-8 Example 5 Two Channel Waveform

5.6 Example 6 Channel Coupling

This example will use the same waveforms from example 5 but will couple channel A and B before running.

5.6.1 Example 6 Main

The following code is the main body for example 4

```
/*
      This function programs a single waveform on channel A
      and runs it in triggered mode using timerA as the trigger.
      Channel B is programed with two waveforms in triggerd mode
      using coupled to channel A.

      Channel A
       sample clock: 75MHz
```

```
      waveform:     360 point sine^3
      amplitude:    5V
      offset:       2.5V
      mode:         triggered, TIMERA (20us)
    Channel B
      sample clock: 125MHz
      waveform1:    720 point sine
      waveform2:    360 point triangle
      amplitude:    2V
      offset:       0V
      mode:         triggered, manual
*/
int example6 (void)
{
     int i, status;
     unsigned short array[720];
     double count, wData, pi = 3.1415926;

     /* get channel A status to check if armed */
     status = getStatus(CHANNEL_A);

     /* reset channel A if armed */
     if (status & 1)
         stopChannel(CHANNEL_A, RESET);

     /* set 5V amplitude and 2.5V offset */
     setGainAndOffset(CHANNEL_A, 5.0, 2.5);

     /* program the sample clock to 75MHz */
     setSampleClock(CHANNEL_A, INTERNAL, 75e6, INTERNAL);

     /* generat sine^3 wave */
     for (i = 0; i < 360; i++) {
         wData = sin (i * pi/180);
         array[i] = (unsigned short) ((8190.0 * wData * wData * wData)+8191);
     }

     /* upload the waveform to the MA1801A */
     setWavformMemory(CHANNEL_A, PRIMARY, 2048, 360, array);

     /* program the sequence to output the waveform */
     setSequenceStep (CHANNEL_A, 1, 2048, 360, 1, SEQMODE_SINGLE, SEQ_AUTO, SEQ_NONE, 1, 0x4);

     /* program TIMERA for a 20us period */
     setTimerPeriod (CHANNEL_A, 20e-6);

     /* set the operating mode and start trigger */
     setOperatingMode(CHANNEL_A, MODE_TRIG, SIG_INTA, SLOPE_HIGH, 2);

     /* disable start/stop by setting source to low and slope high */
     setStopStartSignal (CHANNEL_A, SIG_LOW, SLOPE_HIGH, SIG_LOW, SLOPE_HIGH);

     /* disable the sync replace mode */
     setSyncReplace (CHANNEL_A, REPLACE_OFF, 5000);

     /* get channel B status to check if armed */
     status = getStatus(CHANNEL_B);

     /* reset channel B if armed */
     if (status & 1)
         stopChannel(CHANNEL_B, RESET);

     /* set 2V amplitude and 0V offset */
     setGainAndOffset(CHANNEL_B, 2.0, 0.0);

     /* generate sine wave */
     for (i = 0, count = 0.0; count < 360.0; i++, count += 0.5) {
         wData = sin (count * pi/180);
         array[i] = (unsigned short) ((8190.0 * wData)+8191);
     }

     /* upload the waveform to the MA1801A */
     setWavformMemory(CHANNEL_B, PRIMARY, 0, 720, array);

     /* generat triangle wave */
     for (i = 0; i <= 90; i++)
         array[i] = (unsigned short) (i * 8192.0 / 90.0) + 8191;
     for (i = 91; i <= 270; i++)
         array[i] = (unsigned short) ((180 - i) * 8192.0 / 90.0) + 8192;
     for (i = 271; i < 360; i++)
         array[i] = (unsigned short) ((i - 360) * 8192.0 / 90.0) + 8192;

     /* upload the waveform to the MA1801A */
     setWavformMemory(CHANNEL_B, PRIMARY, 2000, 360, array);

     /* program the sequence step 1 to output the waveform1 */
     setSequenceStep (CHANNEL_B, 1, 0, 720, 1, SEQMODE_SINGLE, SEQ_AUTO, SEQ_NONE, 1, 0x0);

     /* program the sequence step 2 to output the waveform1 */
     setSequenceStep (CHANNEL_B, 2, 2000, 360, 1, SEQMODE_SINGLE, SEQ_AUTO, SEQ_NONE, 1, 0x4);

     /* set the operating mode and start trigger */
```

```
        setOperatingMode(CHANNEL_B, MODE_TRIG, SIG_LOW, SLOPE_HIGH, 1);

        /* couple channel B */
        setChannelCoupling (COUPLE_INT, COUPLE_OFF, COUPLE_ON);


        /* arm channel B */
        armChannel(CHANNEL_B, 1, CONNECT);

        /* arm channel A */
        armChannel(CHANNEL_A, 1, CONNECT);

        return 0;
}
```

5.6.2 Example 6 Waveform

Example 6 will generate the waveform depicted below:

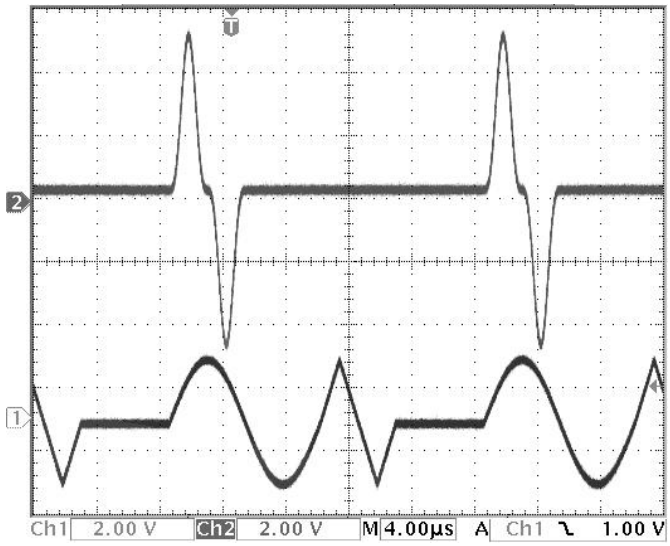

Figure 5-9 Example 6 Channel Coupling

5.6.3 Example 6 Clock Divider Exercise

Add the “setClockDivider” function call after the “setChannelCoupling” function:

```
setClockDivider(CHANNEL_A, 3);
```

Programming the clock divider to 3 for channel A will result in the following waveform:

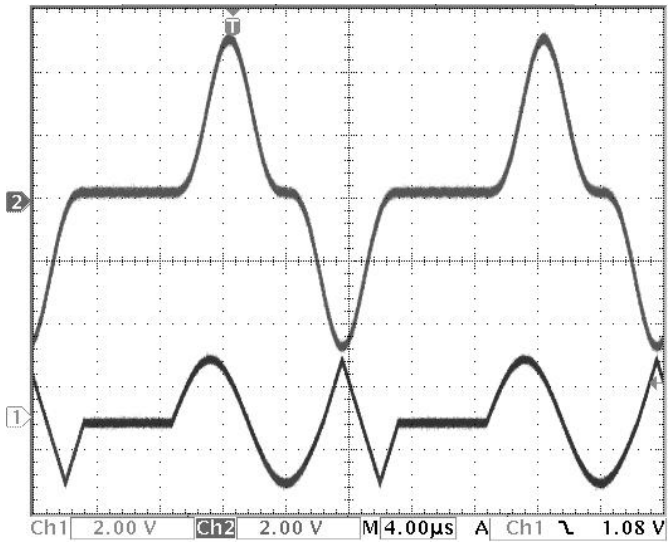

Figure 5-11 Example 6 Clock Divide Waveform

5.6.4 Example 6 Trigger Delay Exercise

Add the “setTriggerDelay” function call after the “setChannelCoupling” function:

```
setTriggerDelay(CHANNEL_B, 200);
```

Programming the trigger delay for channel B will result in the following waveform:

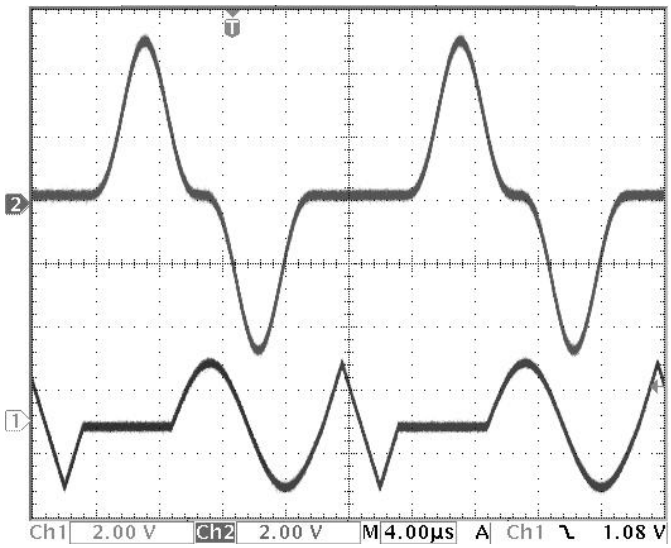

Figure 5-10 Example 6 Trigger Delay Waveform

Appendix A MA1801A Register Map

1 Introduction

Every M Module has 256 bytes of user defined I/O memory. MA Modules have an additional 16Mbytes of user defined memory space available.

The following sections describes the MA1801A registers.

2 I/O Memory

Table A-1 lists the I/O memory register map usage for the MA1801A.

Offset	Function
0h	Memory/Run Control
4h	Start Delay
8h	Start/Output Control
Ch	Timeout Control
10h	Stop/Restart Control
14h, 18h, 1Ch	Sync A Control
20h, 24h, 28h	Sync B Control
2ch, 30h	Sequence Control
34h	Signal/Clock Control
38h	Interval Timer
3Ch	First Sequence
40h, 44h, 48h	Clock Generator Control
80h	Gain Channel A
84h	Gain Channel B
88h	Offset Channel A
8Ch	Offset Channel B
90h	Trigger1A Reference Level
94h	Trigger1B Reference Level
98h	Reference Clock Reference Level
9Ch	External Clock Reference Level
C0h	Extended Memory Address/Page
C4h	Extended Memory Data
C8h	Frequency Divider A
CCh	Frequency Divider B
D0h	Status A
D4h	Event A
D8h	Event Enable A
DCh	Address Readback A
E0h	Status B
E4h	Event B
E8h	Event Enable B
ECh	Address Readback B
F0h	Sweep A
F4h	Sweep B
F8h	Reserved
FCh	MA1801A Version/IDPROM

Table A-1 I/O Memory Register Map

The following sections describes the registers listed above.

2.1 Memory/Run Control (0h)

The memory/run control register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DAC		RUN B						NU		RUN A						MEM SEL				MEM BSY				MEM GRNT				MEM REQ			

Field/Bit Definition:

MEM REQ (Memory Request) The memory request bits are used to access the arbitrary waveform memories. A one written at the bit position will request the associated memory. A zero will release the memory. Use the MEM GRNT bit to test if the memory has been granted. See note 1 for the bit position/memory mapping.

MEM GRNT (Memory Grant) The memory grant bits are used to indicate if the arbitrary waveform memories are available for programming. A one read at the bit position indicates the associated memory has been granted. A zero indicates that the memory has not been released. See note 1 for the bit position/memory mapping.

MEM BSY (Memory Busy) The memory busy bits are used to test if the arbitrary waveform memory is active. A one read at the bit position indicates that the associated memory is being used to generate a waveform. A zero indicates the memory is not being used. See note 1 for the bit position/memory mapping.

MEM SEL (Memory Select) The memory select bits are used to access the MA1801A memories.

Bit 15	Bit 14	Bit 13	Bit 12	Memory Select
0	0	0	0	Channel A Primary
0	0	0	1	Channel A Secondary
0	0	1	0	Channel B Primary
0	0	1	1	Channel B Secondary
0	1	0	0	Sequence Memory A
0	1	0	1	Sequence Memory B
0	1	1	0	Persistence data bank 1
0	1	1	1	Persistence data bank 2
1	0	0	0	Sweep Frequency Divide A
1	0	0	1	Sweep Frequency Divide B

RUN A The run A bits are used to control channel A execution, see note 2 and 3.

RUN B The run B bits are used to control channel B execution, see note 2 and 3.

DAC The DAC bits are used to control the D/A logic.

Bit#	Read Data	Write Action
30	Not used (0)	Reset DAC level to zero (0 = no action; 1 = reset)
31	DAC update status(0 = done, 1 = updating)	DAC update (0 = no action; 1 = update)

Notes:

1. MEM REQ, MEM GRNT, MEM BSY bit position/memory mapping.

Bit Position	Associated Memory
1	Channel A primary
2	Channel A secondary
3	Channel B primary
4	Channel B secondary

2. RUN A, RUN B bit position read/write description.

Bit Position	Read Data	Write Action
1	Arm Sequence (0 = not armed; 1 = armed)	
2	Run (0 = stopped, 1 = running)	Stop Sequence (0 = no action; 1 = manual stop)
3	Start/Restart (0 = no action; 1 = start/restart)	
4	Sequence Active (0 = not active, 1 = active)	Reset Sequence (0 = no action; 1 = reset)
5	Advance Manual Trigger (0 = no action ; 1 = trigger)	
6	Jump Manual Trigger (0 = no action ; 1 = trigger)	

3. Bit 2 (RUN) goes true with start/restart and false when stoped, sequence complete or reset. Bit 4 (Sequence Active) goes true after start, including start delay, and false when sequence complete or reset.

2.2 Start Delay (4h)

The start delay register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DELAY B																DELAY A															

Field/Bit Definition:

DELAY A — Start delay channel A. The start delay is triggered by the selected start source. Start delay is in word count intervals (word count is one long word of memory which generates 2 samples).

DELAY B — Start delay channel B. The start delay is triggered by the selected start source. Start delay is in word count intervals (word count is one long word of memory which generates 2 samples).

Notes:

None.

2.3 Start/Output Control (8h)

The start/output control register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SIGOUT2B				SIGOUT1B				SIGOUT2A				SIGOUT1A				OUTB		SC B		SS B				OUTA		SC A		SS A			

Field/Bit Definition:

SS A — (Start Signal Channel A) The start signal bits are used to select the source for the channel A start. See note 1 for the signal choices.

SC A — (Start Control Channel A) The start control bits are used to select the channel A start function. See note 2 for the start control choices.

OUTA — (Output Channel A) The output channel bits control the main front panel output for channel A. See note 3 for the output choices.

SS B — (Start Signal Channel B) The start signal bits are used to select the source for the channel B start. See note 1 for the signal choices.

SC B — (Start Control Channel B) The start control bits are used to select the channel B start function. See note 2 for the start control choices.

OUTB — (Output Channel B) The output channel bits control the main front panel output for channel B. See note 3 for the output control choices.

SIGOUT1A — (Signal Output One Channel A) These bits select the signal routed to the SigOut1A output. See note 4 for the signal choices.

SIGOUT2A — (Signal Output Two Channel A) These bits select the signal routed to the SigOut2A output. See note 4 for the signal choices.

SIGOUT1B — (Signal Output One Channel B) These bits select the signal routed to the SigOut1B output. See note 5 for the signal choices.

SIGOUT2B — (Signal Output Two Channel B) These bits select the signal routed to the SigOut2B output. See note 5 for the signal choices.

Notes:

1. Start Signal Select

Bit Code				Start Signal
0	0	0	0	TRIG1A
0	0	0	1	TRIG2A
0	0	1	0	TRIG1B
0	0	1	1	TRIG2B
0	1	0	0	Marker1A
0	1	0	1	Marker2A
0	1	1	0	Marker1B

0	1	1	1	Marker2B
1	0	0	0	TRIGA
1	0	0	1	TRIGB
1	0	1	0	NU
1	0	1	1	NU
1	1	0	0	INTERVAL TIMER A
1	1	0	1	INTERVAL TIMER B
1	1	1	0	HIGH
1	1	1	1	LOW

2. Start Control

Bit Code		Start Control
0	0	Low level starts channel
0	1	High level starts channel
1	0	Falling edge starts channel
1	1	Rising edge starts channel

3. Output channel control.

Bit Code		Output Channel Control
0	0	Output channel isolated
0	1	Output channel connected without filter.
1	0	Output channel connected with filter.
1	1	NU

4. Signal output selections channel A.

Bit Code				Signal Output
0	0	0	0	Sync A
0	0	0	1	Marker1A
0	0	1	0	Marker2A
0	0	1	1	SeqFlag1A
0	1	0	0	SeqFlag2A
0	1	0	1	INTERVAL TIMER A
0	1	1	0	INTERVAL TIMER B
0	1	1	1	Good Trig1A
1	0	0	0	Good Trig2A
1	0	0	1	GoodClkA
1	0	1	0	TRIGA
1	0	1	1	TRIGB
1	1	0	0	Seq Active
1	1	0	1	Run
1	1	1	0	Advance
1	1	1	1	Set Low

5. Signal output selections channel B.

Bit Code				Signal Output
0	0	0	0	Sync B
0	0	0	1	Marker1B
0	0	1	0	Marker2B
0	0	1	1	SeqFlag1B
0	1	0	0	SeqFlag2B
0	1	0	1	INTERVAL TIMER A
0	1	1	0	INTERVAL TIMER B
0	1	1	1	Good Trig1B
1	0	0	0	Good Trig2B
1	0	0	1	GoodClkB
1	0	1	0	TRIGA
1	0	1	1	TRIGB
1	1	0	0	Seq Active
1	1	0	1	Run
1	1	1	0	Advance
1	1	1	1	Set Low

2.4 Timeout Control (Ch)

The timeout control register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TIMEOUT B																TIMEOUT A															

Field/Bit Definition:

TIMEOUT A — The timeout A allows the user to jump to a sequence if an advance signal does not occur within the specified timeout. The timeout can be set from 20ms to 1.3107s in 20ms steps. The timeout is cleared each time a new waveform is started.

TIMEOUT B — The timeout B allows the user to jump to a sequence if an advance signal does not occur within the specified timeout. The timeout can be set from 20ms to 1.3107s in 20ms steps. The timeout is cleared each time a new waveform is started.

Notes:

None.

2.5 Stop/Restart Control (10h)

The stop/restart control register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NU	RESC B			RESS B				NU		STOC B		STOS B				NU		RESC A		RESS A				NU		STOC A		STOS A			

Field/Bit Definition:

STOS A — (Stop Signal Channel A) The stop signal bits are used to select the source for the channel A stop. See note 1 for the stop signal choices.

STOC A — (Stop Control Channel A) The stop control bits are used to select the channel A stop function. See note 2 for the stop control choices.

RESS A — (Restart Signal Channel A) The restart signal bits are used to select the source for the channel A restart. See note 1 for the restart signal choices.

RESC A — (Restart Control Channel A) The restart control bits are used to select the channel A restart function. See note 2 for the restart control choices.

STOS B — (Stop Signal Channel B) The stop signal bits are used to select the source for the channel B stop. See note 1 for the stop signal choices.

STOC B — (Stop Control Channel B) The stop control bits are used to select the channel B stop function. See note 2 for the stop control choices.

RESS B — (Restart Signal Channel B) The restart signal bits are used to select the source for the channel B restart. See note 1 for the restart signal choices.

RESC B — (Restart Control Channel B) The restart control bits are used to select the channel B restart function. See note 2 for the restart control choices.

Notes:

1. Stop/Restart Signal. Selecting TIMERA or TIMERB as the stop signal will reset the selected timer when the sequence is started.

Bit Code				Stop/Restart Signal
0	0	0	0	TRIG1A
0	0	0	1	TRIG2A
0	0	1	0	TRIG1B
0	0	1	1	TRIG2B
0	1	0	0	Marker1A
0	1	0	1	Marker2A
0	1	1	0	Marker1B
0	1	1	1	Marker2B
1	0	0	0	TRIGA
1	0	0	1	TRIGB
1	0	1	0	NU
1	0	1	1	NU

1	1	0	0	INTERVAL TIMER A
1	1	0	1	INTERVAL TIMER B
1	1	1	0	HIGH
1	1	1	1	LOW

2. Stop/Restart Control

Bit Code		Stop/Restart Control
0	0	Low level stops/restarts channel
0	1	High level stops/restarts channel
1	0	Falling edge stops/restarts channel
1	1	Rising edge stops/restarts channel

2.6 Sync A Control 1 (14h)

The sync A control 1 register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NU		SEQ STEP									WORD OFFSET																				

Field/Bit Definition:

WORD OFFSET (Word Offset Channel A) The word offset specifies where the sync pulse will occur. Each word contains two sample points.

SEQ STEP (Sequence Step Channel A) When “MODE” (control word 2) is set to step, this specifies the sequence step where the word offset will begin counting. Each word contains two sample points. Program the desired sequence step minus one, e.g., to program step 6, set the register to 5.

Notes:

None.

2.7 Sync A Control 2 (18h)

The sync A control 2 register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SL	RP	MODE		LENGTH												LOOP COUNT															

Field/Bit Definition:

LOOP COUNT (Loop Count Channel A) When “TYPE” is set to loop, the sync pulse will only occur at the specified loop count. Program the desired loop minus two, e.g., to program loop 1000, set the register to 998.

LENGTH (Length Channel A) This specifies the sync pulse length. The length is specified in words where each word contains two sample points. Each word contains two sample points. To program a length of 10 words (20 samples), set the register to 10.

MODE (Sync Mode Channel A) These bits control the sync mode.

Bit Code		Mode	Description
0	0	Start	Start the word offset from the beginning of the sequence.
0	1	Step	Start the word offset from the beginning of the specified step.
1	0	Loop	Start the word offset from the beginning of the specified step for every loop.
1	1	Single Loop	Start the word offset from the beginning of the specified step at the specified loop.

RP (Replace Channel A) This bit controls the sync replace function. (0 = sync only, 1 = replace)

SL (Sync Level Channel A) This bit sets the active sync level. (0 = high, 1 = low)

Notes:

None.

2.8 Sync A Control 3 (1Ch)

The sync A control 3 register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NU													REPLACE ADDR																	

Field/Bit Definition:

REPLACE ADDR (Replace Address Channel A) This field specifies the location of the waveform when replace is enabled (RP bit high in control register 2).

Notes:

None.

2.9 Sync B Control 1 (20h)

The sync B control 1 register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NU		SEQ STEP									WORD OFFSET																				

Field/Bit Definition:

WORD OFFSET (Word Offset Channel B) The word offset specifies where the sync pulse will occur. Each word contains two sample points.

SEQ STEP (Sequence Step Channel B) When “MODE” (control word 2) is set to step, this specifies the sequence step where the word offset will begin counting. Each word contains two sample points. Program the desired sequence step minus one, e.g., to program step 6, set the register to 5.

Notes:

None.

2.10 Sync B Control 2 (24h)

The sync B control 2 register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SL	RP	TYPE		LENGTH												LOOP COUNT															

Field/Bit Definition:

LOOP COUNT (Loop Count Channel B) When “TYPE” is set to loop, the sync pulse will only occur at the specified loop count. Program the desired loop minus two, e.g., to program loop 1000, set the register to 998.

LENGTH (Length Channel B) This specifies the sync pulse length. The length is specified in words where each word contains two sample points. Each word contains two sample points. To program a length of 10 words (20 samples), set the register to 10.

MODE (Sync Mode Channel B) These bits control the sync mode.

Bit Code		Mode	Description
0	0	Sequence	Start the word offset from the beginning of the sequence.
0	1	Step	Start the word offset from the beginning of the specified step.
1	0	Loop	Start the word offset from the beginning of the specified step for every loop.

1	1	Single Loop	Start the word offset from the beginning of the specified step at the specified loop.

RP (Replace Channel B) This bit controls the sync replace function. (0 = sync only, 1 = replace)

SL (Sync Level Channel B) This bit sets the active sync level. (0 = high, 1 = low)

Notes:

None.

2.11 Sync B Control 3 (28h)

The sync B control 3 register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NU													REPLACE ADDR																	

Field/Bit Definition:

REPLACE ADDR (Replace Address Channel B) This field specifies the location of the waveform when replace is enabled (RP bit high in control register 2).

Notes:

None.

2.12 Sequence Control 1 (2Ch)

The sequence control 1 register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BURST[15:0] B																BURST[15:0] A															

Field/Bit Definition:

BURST[15:0] A (Burst Channel A) Lower 16 bits of the channel A burst count. The upper four bits are programmed in the “Sequence Control 2” register.

BURST[15:0] B (Burst Channel B) Lower 16 bits of the channel B burst count. The upper four bits are programmed in the “Sequence Control 2” register.

Notes:

1. Program the burst count minus one, e.g., Programming the lower 16 bits to hex 423F and the upper 4 bits to hex F, sets the burst count to 1,000,000.

2.13 Sequence Control 2 (30h)

The sequence control 2 register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T2B SEL				T1B SEL				T2A SEL				T1A SEL				NU			CB	BURST[19:16] B				NU			CA	BURST[19:16] A			

Field/Bit Definition:

BURST[19:16] A (Burst Channel A) Upper 4 bits of the channel A burst count. The lower 16 bits are programmed in the “Sequence Control 1” register.

CA (Continuous Channel A) This bit enable/disable the continuous burst mode for channel A. (0 = disabled, 1 = enabled)

BURST[19:16] B (Burst Channel B) Upper 4 bits of the channel B burst count. The lower 16 bits are programmed in the “Sequence Control 1” register.

CB (Continuous Channel B) This bit enables/disables the continuous burst mode for channel B. (0 = disabled, 1 = enabled)

T1A SEL (Jump/Advance T1A Select) These bits select the source for the T1A signal. See note 1 for the signal choices.

T2A SEL (Jump/Advance T2A Select) These bits select the source for the T2A signal. See note 1 for the signal choices.

T1B SEL (Jump/Advance T1B Select) These bits select the source for the T1B signal. See note 1 for the signal choices.

T2B SEL (Jump/Advance T2B Select) These bits select the source for the T2B signal. See note 1 for the signal choices.

Notes:

1. Jump/Advance Signal. Selecting TIMERA or TIMERB will cause the selected timer to be reset at the beginning of each sequence step.

Bit Code				Jump/Advance Signal
0	0	0	0	TRIG1A
0	0	0	1	TRIG2A
0	0	1	0	TRIG1B
0	0	1	1	TRIG2B
0	1	0	0	Marker1A
0	1	0	1	Marker2A
0	1	1	0	Marker1B
0	1	1	1	Marker2B
1	0	0	0	TRIGA
1	0	0	1	TRIGB
1	0	1	0	NU
1	0	1	1	NU
1	1	0	0	INTERVAL TIMER A
1	1	0	1	INTERVAL TIMER B
1	1	1	0	HIGH
1	1	1	1	LOW

2.14 Signal/Clock Control (34h)

The signal/clock control register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NU			IU	CSB		CSA		ICDB				ICDA				NU	ES	EB	EA	TRIGB				TRIGA				CB	CA	LB	M

Field/Bit Definition:

M (Master Mode) This bit controls the master mode. (0 = disable master mo, 1 = enable master mode channel A).

LB (Link Channel B) This bit is used to link channel B to channel A. (0 = not linked, 1 = linked)

CA (Couple Channel A) This bit allows the user to assign channel A to the master/slave chain (coupled) or make it local. (0 = local, 1 = coupled)

CB (Couple Channel B) This bit allows the user to assign channel B to the master/slave chain (coupled) or make it local. (0 = local, 1 = coupled)

TRIGA (TRIGA Signal) These bits select the source for the TRIGA signal. See note 1 for the signal choices.

TRIGB (TRIGB Signal) These bits select the source for the TRIGB signal. See note 1 for the signal choices.

EA (Enable TRIGA Output) This bit enables/disables the TRIGA output. (0 = disabled, 1 = enabled)

EB (Enable TRIGB Output) This bit enables/disables the TRIGB output. (0 = disabled, 1 = enabled)

ES (Enable SigOut A1, A2, B1, B2) This bit enables/disables the SigOut outputs. (0 = disabled, 1 = enabled)

ICDA (Internal Clock Divider Channel A) These bits select the clock divide value for the channel A internal clock. See note 2 for the divide choices.

ICDB (Internal Clock Divider Channel B) These bits select the clock divide value for the channel B internal clock. See note 2 for the divide choices.

CSA (Clock Select Channel A)

Bit Code		Channel A Clock
0	0	Internal A
0	1	External
1	0	None
1	1	Internal B

CSB (Clock Select Channel B)

Bit Code		Channel A Clock
0	0	Internal B
0	1	External
1	0	None
1	1	Internal A

IU (Internal Clock Update) This bit sends the data in the “Clock Generator” registers to the specified channel. (0 = no action, 1 = update). This bit is reset to zero when the update process is complete.

Notes:

1. TRIGA/TRIGB Signal

Bit Code				Jump/Advance Signal
0	0	0	0	TRIG1A
0	0	0	1	TRIG2A
0	0	1	0	TRIG1B
0	0	1	1	TRIG2B
0	1	0	0	Marker1A
0	1	0	1	Marker2A
0	1	1	0	Marker1B
0	1	1	1	Marker2B
1	0	0	0	SeqFlag1A
1	0	0	1	SeqFlag2A
1	0	1	0	SeqFlag1B
1	0	1	1	SeqFlag2B
1	1	0	0	SYNCA
1	1	0	1	SYNCB
1	1	1	0	NU
1	1	1	1	None

2. Internal Clock Divide

Bit Code				Divide Value
0	0	0	0	1
0	0	0	1	10
0	0	1	0	100
0	0	1	1	1,000
0	1	0	0	10,000
0	1	0	1	100,000
0	1	1	0	1,000,000
0	1	1	1	10,000,000
1	0	0	0	100,000,000
1	0	0	1	1,000,000,000
1	0	1	0	10,000,000,000
1	0	1	1	100,000,000,000
1	1	0	0	1,000,000,000,000

2.15 Interval Timer (38h)

The interval timer register is illustrated below:

Bit #																												
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ITB																ITA															

Field/Bit Definition:

ITA (Interval Timer Channel A) The interval can be set from 20ms to 1.3107s in 20ms steps. (0 = 20ms, 1 = 40ms, etc...)

ITB (Interval Timer Channel B) The interval can be set from 20ms to 1.3107s in 20ms steps. (0 = 20ms, 1 = 40ms, etc...)

Notes:

1. The internal timer can be selected as any of the following:
   A) Start signal
   B) Stop signal, reset at the start of the sequence.
   C) Restart signal
   D) Jump signal, reset at the start of each sequence step
   E) Advance signal, reset at the start of each sequence step

2.16 First Sequence (3Ch)

The first sequence register is illustrated below:

Bit #																														
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NU					NTB		FSB									NU					NTA		FSA								

Field/Bit Definition:

FSA (First Sequence Channel A) This field sets the sequence address to execute for channel A.

NTA (Noise Type Channel A) This filed specifies the noise algorithm used when a noise waveform is output on channel A. See note 1 for the signal choices.

FSB (First Sequence Channel B) This field sets the sequence address to execute for channel B. See note 1 for the signal choices.

NTB (Noise Type Channel B) This filed specifies the noise algorithm used when a noise waveform is output on channel B.

Notes:

1. Noise Type Algorithm

Bit Code		Jump/Advance Signal
0	0	White
0	1	Gaussian-A
1	0	Gaussian-B
1	1	Pink

2.17 Clock Generator Control 1 (40h)

The clock generator control 1 register is illustrated below:

Bit #																														
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	1	0	1	1	CS	0	0	0	1	1	1	1	0	0	0	0

Field/Bit Definition:

CS (Channel Select) This bit select the clock generator to program. (0 = channel A, 1 = channel B)

Notes:

None.

2.18 Clock Generator Control 2 (44h)

The clock generator control 2 register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FEEDBACK[7:0]								0	0	0	0	PD1		PD0		0	0	PD	RS	0	0	1	1	1	1	1	0	1	0	0	0

Field/Bit Definition:

RS (Reference Select) This bit select the clock generator reference clock source. If the (1 = external, 0 = internal)

PD (Power Down) This bit enables/disables the clock generator. (1 = power down, 0 = normal)

PD0 (Post Divide Zero) This field programs the post divide count zero, see note 1.

Bit Code		Post Divide
0	0	1
0	1	2
1	0	4
1	1	8

PD1 (Post Divide One) This field programs the post divide count one, see note 1.

Bit Code		Divide
0	0	1
1	0	5

FEEDBACK[7:0] Lower eight bits of the feedback divide value, see note 1 of the “Clock Generator Control 3” register.

Notes:

1. The two post divide values are multiplied.
2. Frequency = FEEDBACK/1000 *20MHz (min = 80, max = 230MHz)

2.19 Clock Generator Control 3 (48h)

The clock generator control 3 register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	CTUNE				0	0	0	0	IR			0	0	0	1	0	0	1	0	0	1	0	FEEDBACK[13:8]					

Field/Bit Definition:

FEEDBACK[13:8] Upper six bits of the feedback divide value, see note 1.

IR (Internal Reference) This field enables/disables the internal reference clock. If the “RS” bit from the “Clock Generator Control 2” register is set to internal, then the internal reference clock must be enabled. (0 = enabled, 7 = disabled)

CTUNE (Course Tune) This field is set to factory default of 7.

Notes:

1. Frequency = FEEDBACK[13:0]/1000 *20MHz (min = 80MHz, max = 230MHz).

2.20 Gain Channel A (80h)

The gain channel A register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NU																0	0	0	0	GAIN											

Field/Bit Definition:

GAIN (Gain Channel A) 2050 = 0.01V, 2248 = 0.25V (1Vpp), 4048 = 2.5V (10Vpp). 1.25mV/step.

Notes:

None.

2.21 Gain Channel B (84h)

The gain channel B register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NU																0	1	0	0	GAIN											

Field/Bit Definition:

GAIN (Gain Channel B) 2050 = 0.01V, 2248 = 0.25V (1Vpp), 4048 = 2.5V (10Vpp). 1.25mV/step.

Notes:

None.

2.22 Offset Channel A (88h)

The offset channel A register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NU																1	0	0	0	OFFSET											

Field/Bit Definition:

OFFSET (Offset Channel A) 2048 = 0V, 48 = -10V, 4048 = 10V. 5mV/step.

Notes:

None.

2.23 Offset Channel B (8Ch)

The offset channel B register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NU																1	1	0	0	OFFSET											

Field/Bit Definition:

OFFSET (Offset Channel B) 2048 = 0V, 48 = -10V, 4048 = 10V. 5mV/step.

Notes:

None.

2.24 Trigger1A Reference Level (90h)

The trigger1A reference level register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NU																0	0	0	0	TG1A REF											

Field/Bit Definition:

TG1A REF (Trigger1A Reference) 2048 = 0V, 48 = -10V, 4048 = 10V. 5mV/step.

Notes:

None.

2.25 Trigger1B Reference Level (94h)

The trigger1B reference level register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NU																0	1	0	0	TG1B REF											

Field/Bit Definition:

TG1B REF (Trigger1B Reference) 2048 = 0V, 48 = -10V, 4048 = 10V. 5mV/step.

Notes:

None.

2.26 Reference Clock Reference Level (98h)

The reference clock reference level register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NU																1	0	0	0	RC REF											

Field/Bit Definition:

RC REF (Reference Clock Reference) 2048 = 0V, 48 = -5V, 4048 = 5V. 2.5mV/step.

Notes:

None.

2.27 External Clock Reference Level (9Ch)

The external clock reference level register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NU																1	1	0	0	EC REF											

Field/Bit Definition:

EC REF (External Clock Reference) 2048 = 0V, 48 = -5V, 4048 = 5V. 2.5mV/step.

Notes:

None.

2.28 Extended Memory Address / Page (C0h)

The extended memory address register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NU									S	PAGE				EMADDR																	

Field/Bit Definition:

EMADDR[17:0] (Extended Memory Address) This field specifies the address which data will be accessed via the extended memory data register. The address will automatically increment after each read/write of the extended memory data register (C4h).

PAGE (Page) This field is used for both the extended memory data register as well as waveform memory programming via extended memory access. Each page programs a 64K block of the waveform memory.

S (Swap MSW/LSW) Swap the word order for extended memory address register accesses. (0 = no swap, 1 = swap)

Notes:

1. Reading this register returns the current address and not the address written.
2. The following figure illustrates the 16 bit extended memory mapping with regards to the S bit and the EMADDR field.

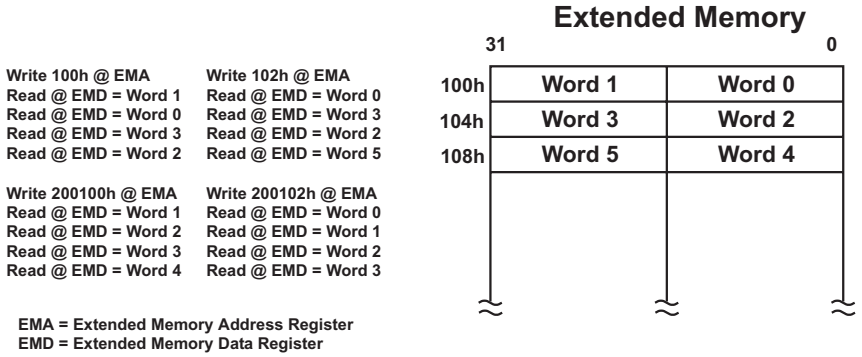

Figure A-1 16 Bit Extended Memory Mapping

2.29 Extended Memory Data (C4h)

The extended memory data register is illustrated below:

Bit #																														
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EMDATA																														

Field/Bit Definition:

EMDATA (Extended Memory Data) This register is used to read/write extended memory data. A 16 bit read/write to this register increments the extended memory address by 2 and address bit 0 is ignored. A 32 bit read/write to this register increments the extended memory address by 4 and address bit 1 and 0 are ignored.

Notes:

1. The “MEM SEL” field of the “Memory/Run Control Register” (0h) must be programmed to select the appropriate memory prior to accessing this register.

2.30 Sample Clock Divider A (C8h)

The sample clock divider A register is illustrated below:

Bit #																														
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SCDIVIDE																														

Field/Bit Definition:

SCDIVIDE (Sample Clock Divide Channel A) This register allows the user to minimize the trigger and sequence logic delay by allowing the user to specify a clock frequency that is a multiple of the sample clock. Program the divide count - 1.

Notes:

None.

2.31 Sample Clock Divider B (CCh)

The sample clock divider B register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SCDIVIDE																															

Field/Bit Definition:

SCDIVIDE (Sample Clock Divide Channel B) This register allows the user to minimize the trigger and sequence logic delay by allowing the user to specify a clock frequency that is a multiple of the sample clock. Program the divide count - 1.

Notes:

None.

2.32 Status A (D0h)

The status A register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NU																MSE	LE	SAE	PAE	SB	PB	CS	SW	RE	GE	JE	STA	SIA	WR	SE	WS

Field/Bit Definition:

WS (Waiting for Start) This bit indicates that channel A is waiting for the start signal. (0 = not waiting, 1 = waiting)

SE (Stop Enabled) This bit indicates that channel A is ready to accept the stop signal. (0 = not ready, 1 = ready)

WR (Waiting for Restart) This bit indicates that channel A is waiting for a restart signal. (0 = not waiting, 1 = waiting)

SIA (Single Advance) This bit indicates that a channel A sequence in “single” or “single1” mode is waiting for the advance signal. (0 = not waiting, 1 = waiting)

STA (Step Advance) This bit indicates that a channel A sequence in “step” mode is waiting for the advance signal. (0 = not waiting, 1 = waiting)

JE (Jump Enabled) This bit indicates that the current channel A segment is enabled for jumps. (0 = not enabled, 1 = enabled)

GE (Gosub Enabled) This bit indicates that the current channel A segment is enabled for jump with return. (0 = not enabled, 1 = enabled)

RE (Replace Enabled) This bit indicates that a channel A replace operation is in progress. (0 = no replace, 1 = replace)

SW (Sweep) This bit indicates that a channel A sweep operation is in progress. (0 = no sweep, 1 = sweep)

CS (Clock Slow) This bit indicates that the MA1801A external clock input is less than 200Hz. (0 = no clock error, 1 = clock error)

PB (Primary Busy) This bit indicates that the channel A primary memory is currently outputting a waveform. (0 = not busy, 1 = busy)

SB (Secondary Busy) This bit indicates that the channel A secondary memory is currently outputting a waveform. (0 = not busy, 1 = busy)

PAE (Primary Access Error) This bit indicates that channel A segment attempted to access the primary memory when it was not released. (0 = no error, 1 = error)

SAE (Secondary Access Error) This bit indicates that channel A segment attempted to access the secondary memory when it was not released. (0 = no error, 1 = error)

LE (Link Error) This bit indicates a sync error between channel A and B. (0 = no error, 1 = error)

MSE (Master/Slave Error) This bit indicates a sync error in master/slave mode. (0 = no error, 1 = error)

Notes:

None.

2.33 Event A (D4h)

The event A register is illustrated below:

Bit #																														
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NU																MSE	LE	NU	TMO	SAE	PAE	CE	SW	REP	GOSUB	JMP	STAO	SIAO	RSTR	STOP	STRT

Field/Bit Definition:

START (Start) This bit indicates that channel A start occurred. (0 = no start, 1 = start)

STOP (Stop) This bit indicates that channel A stop occurred. (0 = no stop, 1 = stop)

RSTRT (Restart) This bit indicates that channel A restart occurred. (0 = no restart, 1 = restart)

SIAO (Single Advance Occurred) This bit indicates that a channel A segment in “single” or “single1” mode received an advance. (0 = no advance, 1 = advance)

STAO (Step Advance Occurred) This bit indicates that a channel A segment in “step” mode received an advance. (0 = no advance, 1 = advance)

JMP (Jump) This bit indicates that a channel A jump occurred. (0 = no jump, 1 = jump)

GOSUB (Gosub) This bit indicates that a channel A jump with return occurred. (0 = no gosub, 1 = gosub)

REP (Replace) This bit indicates that a channel A replace operation occurred. (0 = no replace, 1 = replace)

SW (Sweep) This bit indicates that a channel A sweep operation occurred. (0 = no sweep, 1 = sweep)

CE (Clock Error) This bit indicates that the MA1801A external clock input was less than 200Hz. (0 = no clock error, 1 = clock error)

PAE (Primary Access Error) This bit indicates that channel A segment attempted to access the primary memory when it was not released. (0 = no error, 1 = error)

SAE (Secondary Access Error) This bit indicates that channel A segment attempted to access the secondary memory when it was not released. (0 = no error, 1 = error)

TMO (Timeout Occurred) This bit indicates that an advance timeout occurred. (0 = no timeout, 1 = timeout)

LE (Link Error) This bit indicates a sync error between channel A and B. (0 = no error, 1 = error)

MSE (Master/Slave Error) This bit indicates a sync error in master/slave mode. (0 = no error, 1 = error)

Notes:

1. All bit reset to zero after reading.

2.34 Event Enable A (D8h)

The event A register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NU																EVENT ENABLE															

Field/Bit Definition:

EVENT ENABLE — This field is used as a mask for interrupt generation. If the bit wise logical 'AND' of this register with the event A register is not zero, then an interrupt is generated.

Notes:

None.

2.35 Address Readback A (DCh)

The address readback A register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NU				SEQADDR									MADDR																		

Field/Bit Definition:

MADDR — (Memory Address) This field contains the current waveform memory address of channel A.

SEQADDR — (Sequence Address) This field contains the current sequence memory address of channel B.

Notes:

None.

2.36 Status B (E0h)

The status B register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NU																MSE	LE	SAE	PAE	SB	PB	CS	SW	RE	GE	JE	STA	SIA	WR	SE	WS

Field/Bit Definition:

WS — (Waiting for Start) This bit indicates that channel B is waiting for the start signal. (0 = not waiting, 1 = waiting)

SE — (Stop Enabled) This bit indicates that channel B is ready to accept the stop signal. (0 = not ready, 1 = ready)

WR — (Waiting for Restart) This bit indicates that channel B is waiting for a restart signal. (0 = not waiting, 1 = waiting)

SIA — (Single Advance) This bit indicates that a channel B sequence in "single" or "single1" mode is waiting for the advance signal. (0 = not waiting, 1 = waiting)

STA — (Step Advance) This bit indicates that a channel B sequence in "step" mode is waiting for the advance signal. (0 = not waiting, 1 = waiting)

JE — (Jump Enabled) This bit indicates that the current channel B segment is enabled for jumps. (0 = not enabled, 1 = enabled)

GE — (Gosub Enabled) This bit indicates that the current channel B segment is enabled for jump with return. (0 = not enabled, 1 = enabled)

RE — (Replace Enabled) This bit indicates that a channel B replace operation is in progress. (0 = no replace, 1 = replace)

SW — (Sweep) This bit indicates that a channel A sweep operation is in progress. (0 = no sweep, 1 = sweep)

CS	(Clock Slow) This bit indicates that the MA1801A external clock input is less than 200Hz. (0 = no clock error, 1 = clock error)
PB	(Primary Busy) This bit indicates that the channel B primary memory is currently outputting a waveform. (0 = not busy, 1 = busy)
SB	(Secondary Busy) This bit indicates that the channel B secondary memory is currently outputting a waveform. (0 = not busy, 1 = busy)
PAE	(Primary Access Error) This bit indicates that channel B segment attempted to access the primary memory when it was not released. (0 = no error, 1 = error)
SAE	(Secondary Access Error) This bit indicates that channel B segment attempted to access the secondary memory when it was not released. (0 = no error, 1 = error)
LE	(Link Error) This bit indicates a sync error between channel A and B. (0 = no error, 1 = error)
MSE	(Master/Slave Error) This bit indicates a sync error in master/slave mode. (0 = no error, 1 = error)

Notes:

None.

2.37 Event B (E4h)

The event B register is illustrated below:

Bit #																												
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NU																MSE	LE	NU	TMO	SAE	PAE	CE	SW	REP	GOSUB	JMP	STAO	SIAO	RSTR	STOP	STRT

Field/Bit Definition:

START	(Start) This bit indicates that channel B start occurred. (0 = no start, 1 = start)
STOP	(Stop) This bit indicates that channel B stop occurred. (0 = no stop, 1 = stop)
RSTRT	(Restart) This bit indicates that channel B restart occurred. (0 = no restart, 1 = restart)
SIAO	(Single Advance Occurred) This bit indicates that a channel B segment in “single” or “single1” mode received an advance. (0 = no advance, 1 = advance)
STAO	(Step Advance Occurred) This bit indicates that a channel B segment in “step” mode received an advance. (0 = no advance, 1 = advance)
JMP	(Jump) This bit indicates that a channel B jump occurred. (0 = no jump, 1 = jump)
GOSUB	(Gosub) This bit indicates that a channel B jump with return occurred. (0 = no gosub, 1 = gosub)
REP	(Replace) This bit indicates that a channel B replace operation occurred. (0 = no replace, 1 = replace)
SW	(Sweep) This bit indicates that a channel A sweep operation occurred. (0 = no sweep, 1 = sweep)
CE	(Clock Error) This bit indicates that the MA1801A external clock input was less than 200Hz. (0 = no clock error, 1 = clock error)
PAE	(Primary Access Error) This bit indicates that channel B segment attempted to access the primary memory when it was not released. (0 = no error, 1 = error)
SAE	(Secondary Access Error) This bit indicates that channel B segment attempted to access the secondary memory when it was not released. (0 = no error, 1 = error)
TMO	(Timeout Occurred) This bit indicates that an advance timeout occurred in channel B. (0 = no timeout, 1 = timeout)
LE	(Link Error) This bit indicates a sync error between channel A and B. (0 = no error, 1 = error)
MSE	(Master/Slave Error) This bit indicates a sync error in master/slave mode. (0 = no error, 1 = error)

Notes:

1. All bit reset to zero after reading.

2.38 Event Enable B (E8h)

The event enable B register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NU																EVENT ENABLE															

Field/Bit Definition:

EVENT ENABLE — This field is used as a mask for interrupt generation. If the bit wise logical 'AND' of this register with the event B register is not zero, then an interrupt is generated.

Notes:

None.

2.39 Address Readback B (ECh)

The address readback B register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NU				SEQADDR									MADDR																		

Field/Bit Definition:

MADDR — (Memory Address) This field contains the current waveform memory address of channel B.

SEQADDR — (Sequence Address) This field contains the current sequence memory address of channel B.

Notes:

None.

2.40 Sweep Register A (F0h)

The sweep register A register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NU								SWEEP STEPS								SWEEP TIMER															

Field/Bit Definition:

SWEEP TIMER — (Sweep Timer Channel A) The interval can be set from 20ms to 1.3107s in 20ms steps. (0 = 20ms, 1 = 40ms, etc...)

SWEEP STEPS — (Sweep Steps Channel A) This field contains the number of sweep steps programmed in the sweep frequency divide memory. Program the number of steps minus one.

Notes:

None.

2.41 Sweep Register B (F4h)

The sweep register B register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NU								SWEEP STEPS								SWEEP TIMER															

Field/Bit Definition:

SWEEP TIMER (Sweep Timer Channel B) The interval can be set from 20ms to 1.3107s in 20ms steps. (0 = 20ms, 1 = 40ms, etc...)

SWEEP STEPS (Sweep Steps Channel B) This field contains the number of sweep steps programmed in the sweep frequency divide memory. Program the number of steps minus one.

Notes:

None.

2.42 MA1801A Version/IDPROM (FCh)

The MA1801A version/IDPROM register is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VER								REV								ID PROM															

Field/Bit Definition:

IDPROM (MA1801A IDPROM)
REV (MA1801A FPGA Revision)
VER (MA1801A FPGA Version)

Notes:

None.

3 Extended Memory

The extended memory of the MA1801A is comprised of the following:

1. Waveform Memory
2. Sequence Memory
3. Persistence Memory

To access a specific memory you must use the “Memory/Run Control” register, see section 2.1.

The following sections describes each of these memories.

3.1 Waveform Memory

The waveform memory contains the data that defines a waveform. Each MA1801Achannel has a primary and secondary waveform memory bank, each containing 256K by 32 bit sample words.

The waveform memory is illustrated below:

Bit #																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M1	M2	DATA2														NU	MODE	DATA1													

Field/Bit Definition:

DATA1 (Waveform Data1) The first of two waveform sample points in normal mode. See note 1.

MODE (Waveform Mode) This bit determines how the memory contents will be evaluated.

Bit Level	Mode
0	Normal (output two sample points)
1	Noise (Generate preset noise algorithm)

DATA2 (Waveform Data2) The Second of two waveform sample points in normal mode. See note 1.

M2 (Marker 2 level) 0 = low, 1 = high.

M1 (Marker 1 level) 0 = low, 1 = high.

Notes:

1. The length is specified in waveform words where each word is two sample points.
2. Programming the waveform memory via extended memory requires the page register to be programmed (C0h)

3.2 Sequence Memory

The sequence memory contains the instructions that selects and outputs a waveform. Each MA1801Achannel has a sequence memory, each containing 512 steps. Each sequence step is defined by four 32 bit words.

The sequence memory is illustrated below:

Sequence Step Word 1
Bit #

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JR	LS	S2	S1	JSEQ									PS	WAVE ADDR																	

Sequence Step Word 2
Bit #

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LA	AV	AM		ACODE				JCODE				JI	WORD COUNT																		

Sequence Step Word 3
Bit #

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NU																LOOP COUNT															

Sequence Step Word 4
Bit #

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NU																															

Field/Bit Definition:

WAVE ADDR (Waveform Memory Address) This field identifies the waveform address for this sequence step.

PS (Primary/Secondary) This bit selects either the primary or secondary memory bank. (0 = primary, 1 = secondary).

JSEQ (Jump Sequence) This field specifies the sequence step destination when a jump occurs.

S1 (Sequence Flag 1 level) 0 = low, 1 = high.

S2 (Sequence Flag 2 level) 0 = low, 1 = high.

LS (Last Sequence) This bit indicates that this is the last sequence step. (0 = last sequence false, 1 = last sequence true).

JR (Jump Return) This bit indicates whether or not a jump will return to the next step or the current sequence (GOSUB) or the next step of the jump sequence (BRANCH). (0 = branch, 1 = gosub).

WORD COUNT (Word Count) This field specifies the number of waveform memory words to be output for this sequence step. (Program the count - 2)

JI (Jump Immediate) This bit controls whether a jump can occur before the specified number of loops have been performed. (0 = Jump at last loop, 1 = Jump after current loop).

JCODE (Jump Code) This field specifies the jump code for this sequence step.

Bit Code				Jump Code
0	0	0	0	Jump if T1 low
0	0	0	1	Jump if T1 high
0	0	1	0	Jump if T1 falling edge
0	0	1	1	Jump if T1 rising edge
0	1	0	0	Jump if T2 low
0	1	0	1	Jump if T2 high
0	1	1	0	Jump if T2 falling edge
0	1	1	1	Jump if T2 rising edge
1	0	0	0	Jump if primary memory granted
1	0	0	1	Jump if primary memory not granted
1	0	1	0	Jump if secondary memory granted
1	0	1	1	Jump if secondary memory not granted
1	1	0	0	Jump if timeout
1	1	1	0	Jump unconditionally
1	1	1	1	Jump disabled

ACODE (Advance Code) This field specifies the advance code for this sequence step.

Bit Code				Advance Code
0	0	0	0	Advance if T1 low
0	0	0	1	Advance if T1 high
0	0	1	0	Advance if T1 falling edge
0	0	1	1	Advance if T1 rising edge
0	1	0	0	Advance if T2 low
0	1	0	1	Advance if T2 high
0	1	1	0	Advance if T2 falling edge
0	1	1	1	Advance if T2 rising edge
1	0	0	1	Advance if primary memory not granted
1	0	1	1	Advance if secondary memory not granted
1	1	1	0	Advance disabled
1	1	1	1	Advance unconditionally

AM (Advance Mode) This field is used to select the advance mode.

Bit Code		Mode	Description
0	0	SINGLE	Output entire step (including loops) and then idle until advance condition true.
0	1	STEP	Output step continuously until advance condition true.
1	0	SINGLE1	Output a single loop and then idle until advance condition true.

AV (Alternate Advance) This bit enables the sweep interval timer as the advance signal. (0 - normal advance, 1 = sweep timer advance), see note 2.

LA (Loop Advance) This bit enables the sweep interval timer to increment the frequency divider address for this sequence step, see note 2.

Notes:

1. The length is specified in waveform words where each word is two sample points.
2. The sweep interval timer can be used to advance the sequence or increment the frequency divider address but not both in the same step.

3.3 Persistence Memory

The persistence memory is used to store the current MA1801A settings. The persistence memory is comprised of two 1K by 32 bit memories.

3.4 Sweep Frequency Divide Memory

The sweep frequency divide memory stores up to 256, 32 bit frequency divide values. These divide values are used for sweeping waveforms. Program the divide value minus one for each sweep step.

Appendix B Signal Description

1 MCX Coaxial Connectors

AOUT	(MCX-OA) Channel A waveform output.
BOUT	(MCX-OB) Channel B waveform output.
TRIG1A	(MCX-TA) Channel A trigger 1 input.
TRIG1B	(MCX-TB) Channel B trigger 1 input.
REFCLK	(MCX-RC) Channel A & B shared reference clock input.
EXTCLK	(MCX-EC) Channel A & B shared external clock input.
AMODA:	(MCX-A) Channel A amplitude modulation input.
AMODB:	(MCX-B) Channel B amplitude modulation input.

2 J1 Connector

TRIG2A	(J1-A) Channel A trigger 2 input.
TRIG2B	(J1-B) Channel B trigger 2 input.
SIGOUT1A	(J1-E) Channel A signal output 1.
SIGOUT2A	(J1-F) Channel A signal output 2.
SIGOUT1B	(J1-H) Channel B signal output 1.
SIGOUT2B	(J1-J) Channel B signal output 2.
GND	(J1-C, J1-D, J1-K) Signal ground.

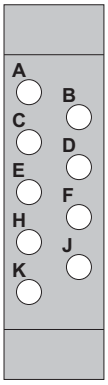

Figure B-1 Front Panel J1

Appendix C Example Code

```
#include <ansi_c.h>
/*
     This project contains the example code for the MA1801A Operators Manual.
     10/11/04
     RWA
*/


/* prototypes */
int Peek16( int address, unsigned short *data);
int Poke16( int address, unsigned short data);
int initInterface(void);
int closeInterface(void);
int initModule(void);
int example1a(void);
int example1b(void);
int example2(void);
int example3(void);
int example4(void);
int example5(void);
int example6(void);
int getStatus(short channel);
int stopChannel(short channel, short type);
int setGainAndOffset(short channel, double gain, double offset);
int setSampleClock(short channel, short source, double freqHz, short refSource);
int setWavformMemory(short channel, short memory, int offset, int samples, unsigned short data[]);
int setSequenceStep (short channel,
                     short step,
                     int waveMemoryAddr,
                     int wavePoints,
                     int loop,
                     short mode,
                     short source,
                     short jumpSource,
                     short jumpSequence,
                     short flags);
int setTimerPeriod (short channel, double triggerPeriod);
int setOperatingMode(short channel, short mode, short startTrig, short slope, int burstCount);
int armChannel(short channel, short step, short outputMode);
int setStopStartSignal (short channel, short stopSig, short stopSlope, short restartSig, short
restartSlope);
int  manualTrigger(short channel, short trigger);
int setSyncOutput (short channel, short level, short source, int offset, short width, short step, int loop);
int setSyncReplace (short channel, short state, int waveMemoryAddr);
int setFrontPanelOutput (short channel, short sigOut1, short sigOut2);
int setChannelCoupling (short mode, short coupleA, short coupleB);
int setClockDivider (short channel, double clockDivider);
int setTriggerDelay (short channel, int triggerDelay);

/* constants */
#define CHANNEL_A          0
#define CHANNEL_B          1
#define MODE_CONT          0
#define MODE_TRIG          1
#define MODE_GATED         2
#define MODE_BURST         3
#define DISARM             0
#define RESET              1
#define ISOLATE            2
#define CONNECT            1
#define FILTER             2
#define INTERNAL           0
#define EXTERNAL           1
#define SEQMODE_AUTO       0
#define SEQMODE_STEP       1
#define SEQMODE_SINGLE     2
#define SEQMODE_SINGLE1    3
#define PRIMARY            0
#define SECONDARY          1
#define SLOPE_LOW          0
#define SLOPE_HIGH         1
#define SLOPE_FALL         2
#define SLOPE_RISE         3
#define SIG_TRIG1A         0
#define SIG_TRIG2A         1
#define SIG_TRIG1B         2
#define SIG_TRIG2B         3
#define SIG_MRKR1A         4
#define SIG_MRKR2A         5
#define SIG_MRKR1B         6
#define SIG_MRKR2B         7
#define SIG_TRIGA          8
#define SIG_TRIGB          9
#define SIG_INTA           12
#define SIG_INTB           13
#define SIG_HIGH           14
#define SIG_LOW            15
```

```
#define SEQ_T1LOW          0
#define SEQ_T1HIGH         1
#define SEQ_T1FALL         2
#define SEQ_T1RISE         3
#define SEQ_T2LOW          4
#define SEQ_T2HIGH         5
#define SEQ_T2FALL         6
#define SEQ_T2RISE         7
#define SEQ_PGRNT          8
#define SEQ_PREL           9
#define SEQ_SGRNT          10
#define SEQ_SREL           11
#define SEQ_TOUT           12
#define SEQ_UNC            14
#define SEQ_MAN            14
#define SEQ_AUTO           15
#define SEQ_NONE           15
#define START_TRIG         0
#define STOP_TRIG          1
#define RESTART_TRIG       2
#define ADVANCE_TRIG       3
#define JUMP_TRIG          4
#define COUPLE_INT         0
#define COUPLE_EXT         1
#define COUPLE_OFF         0
#define COUPLE_ON          1
#define SYNC_HIGH          0
#define SYNC_LOW           1
#define SYNC_START         0
#define SYNC_STEP          1
#define SYNC_LOOP          2
#define SYNC_SLOOP         3
#define REPLACE_OFF        0
#define REPLACE_ON         1
#define FPO_SYNC           0
#define FPO_MRKR1          1
#define FPO_MRKR2          2
#define FPO_SEQFLAG1       3
#define FPO_SEQFLAG2       4
#define FPO_TIMERA         5
#define FPO_TIMERB         6
#define FPO_GTRIG1         7
#define FPO_GTRIG2         8
#define FPO_GEXTCLK        9
#define FPO_TRIGA          10
#define FPO_TRIGB          11
#define FPO_SEQACT         12
#define FPO_RUN            13
#define FPO_ADVANCE        14
#define FPO_LOW            15

/* register addresses */
#define RUNCTRL            0x0
#define MEMORY             0x2
#define TRIGDLYB           0x4
#define TRIGDLYA           0x6
#define OUTPUTSIG          0x8
#define STARTCTRL          0xA
#define TIMEOUTB           0xC
#define TIMEOUTA           0xE
#define STOPRESTARTB       0x10
#define STOPRESTARTA       0x12
#define SYNCREGA           0x14
#define SYNCREGB           0x20
#define BURSTLOOPB         0x2C
#define BURSTLOOPA         0x2E
#define SEQSIG             0x30
#define BURST              0x32
#define CLKCTRL            0x34
#define MISCCTRL           0x36
#define TIMERB             0x38
#define TIMERA             0x3A
#define FIRSTSEQB          0x3C
#define FIRSTSEQA          0x3E
#define CLKGEN             0x40
#define GAIN_A             0x82
#define GAIN_B             0x86
#define OFFSET_A           0x8A
#define OFFSET_B           0x8E
#define TRIG1AREF          0x92
#define TRIG1BREF          0x96
#define RCLKREF            0x9A
#define ECLKREF            0x9E
#define SCDIVIDEA          0xC8
#define SCDIVIDEB          0xCC
#define STATUSA            0xD2
#define EVENTA             0xD6
#define ENABLEA            0xDA
#define SEQSTATUSA         0xDC
#define STATUSB            0xE2
#define EVENTB             0xE6
#define ENABLEB            0xEA
```

```
#define SEQSTATUSB          0xEC
#define EMADDR              0xC0
#define EMDATA              0xC4

/* error code */
#define ERROR_MEMREQ  -1;

/* Example test */
int main (int argc, char *argv[])
{

     /* initialize VXI interface */
     if (initInterface()) {
         if (argc == 2)
             switch (*argv[1]) {
             case '1':
                 example1a();
                 example1b();
                 break;
             case '2':
                 example2();
                 break;
             case '3':
                 example3();
                 break;
             case '4':
                 example4();
                 break;
             case '5':
                 example5();
                 break;
             case '6':
                 example6();
                 break;
             }
         closeInterface();
     }


     return 0;
}

/*
     This function programs a single 360 point waveform on channel A
     and runs it in triggered mode using timerA as the trigger.

      sample clock: 75MHz
      waveform:     360 point sine^3
      amplitude:    5V
      offset:       2.5V
      mode:         triggered, TIMERA (20us)
*/
int example1a (void)
{
     int i, status;
     unsigned short array[360];
     double wData, pi = 3.1415926;

     /* get channel A status to check if armed */
     status = getStatus(CHANNEL_A);

     /* reset channel A if armed */
     if (status & 1)
         stopChannel(CHANNEL_A, RESET);

     /* set 5V amplitude and 2.5V offset */
     setGainAndOffset(CHANNEL_A, 5.0, 2.5);

     /* program the sample clock to 75MHz */
     setSampleClock(CHANNEL_A, INTERNAL, 75e6, INTERNAL);

     /* generat sine^3 wave */
     for (i = 0; i < 360; i++) {
         wData = sin (i * pi/180);
         array[i] = (unsigned short) ((8190.0 * wData * wData * wData)+8191);
     }

     /* upload the waveform to the MA1801A */
     setWavformMemory(CHANNEL_A, PRIMARY, 2048, 360, array);

     /* program the sequence to output the waveform */
     setSequenceStep (CHANNEL_A, 1, 2048, 360, 1, SEQMODE_SINGLE, SEQ_AUTO, SEQ_NONE, 1, 0x4);

     /* program TIMERA for a 20us period */
     setTimerPeriod (CHANNEL_A, 20e-6);

     /* set the operating mode and start trigger */
     setOperatingMode(CHANNEL_A, MODE_TRIG, SIG_INTA, SLOPE_HIGH, 2);

     /* disable start/stop by setting source to low and slope high */
     setStopStartSignal (CHANNEL_A, SIG_LOW, SLOPE_HIGH, SIG_LOW, SLOPE_HIGH);
```

```
     /* arm the channel */
     armChannel(CHANNEL_A, 1, CONNECT);

     return 0;
}

/*
     This function programs a single 360 point waveform on channel B
     and runs it in triggered mode using timerA as the trigger.

      sample clock: 125MHz
      waveform:     360 point sine
      amplitude:    6V
      offset:       -2V
      mode:         triggered, TIMERB (200us)
*/
int example1b (void)
{
     int i, status;
     unsigned short array[360];
     double wData, pi = 3.1415926;

     /* get channel A status to check if armed */
     status = getStatus(CHANNEL_B);

     /* reset channel A if armed */
     if (status & 1)
         stopChannel(CHANNEL_B, RESET);

     /* set 5V amplitude and 2.5V offset */
     setGainAndOffset(CHANNEL_B, 6.0, -2.0);

     /* program the sample clock to 75MHz */
     setSampleClock(CHANNEL_B, INTERNAL, 125e6, INTERNAL);

     /* generat sine wave */
     for (i = 0; i < 360; i++) {
         wData = sin (i * pi/180);
         array[i] = (unsigned short) ((8190.0 * wData)+8191);
     }

     /* upload the waveform to the MA1801A */
     setWavformMemory(CHANNEL_B, PRIMARY, 2048, 360, array);

     /* program the sequence to output the waveform */
     setSequenceStep (CHANNEL_B, 1, 2048, 360, 1, SEQMODE_SINGLE, SEQ_AUTO, SEQ_NONE, 1, 0x4);

     /* program TIMERB for a 20us period */
     setTimerPeriod (CHANNEL_B, 200e-6);

     /* set the operating mode and start trigger */
     setOperatingMode(CHANNEL_B, MODE_TRIG, SIG_INTB, SLOPE_HIGH, 2);

     /* disable start/stop by setting source to low and slope high */
     setStopStartSignal (CHANNEL_B, SIG_LOW, SLOPE_HIGH, SIG_LOW, SLOPE_HIGH);

     /* arm the channel */
     armChannel(CHANNEL_B, 1, CONNECT);

     return 0;
}

/*
     This function programs a two waveforms sequence on channel A
     and runs them in trigger mode using manual as the
     trigger.

      sample clock: 125MHz
      waveform1:    720 point sine
      waveform2:    360 point triangle
      amplitude:    2V
      offset:       0V
      mode:         triggered, manual
*/
int example2 (void)
{
     int i, status;
     unsigned short array[720];
     double count, wData, pi = 3.1415926;

     /* get channel A status to check if armed */
     status = getStatus(CHANNEL_A);

     /* reset channel A if armed */
     if (status & 1)
         stopChannel(CHANNEL_A, RESET);

     /* set 2V amplitude and 0V offset */
     setGainAndOffset(CHANNEL_A, 2.0, 0.0);

     /* program the sample clock to 75MHz */
     setSampleClock(CHANNEL_A, INTERNAL, 125e6, INTERNAL);
```

```
	/* generat sine wave */
	for (i = 0, count = 0.0; count < 360.0; i++, count += 0.5) {
	    wData = sin (count * pi/180);
	    array[i] = (unsigned short) ((8190.0 * wData)+8191);
	}

	/* upload the waveform to the MA1801A */
	setWavformMemory(CHANNEL_A, PRIMARY, 2048, 720, array);

	/* generat triangle wave */
	for (i = 0; i <= 90; i++)
	    array[i] = (unsigned short) (i * 8192.0 / 90.0) + 8191;
	for (i = 91; i <= 270; i++)
	    array[i] = (unsigned short) ((180 - i) * 8192.0 / 90.0) + 8192;
	for (i = 271; i < 360; i++)
	    array[i] = (unsigned short) ((i - 360) * 8192.0 / 90.0) + 8192;

	/* upload the waveform to the MA1801A */
	setWavformMemory(CHANNEL_A, PRIMARY, 5000, 360, array);

	/* program the sequence step 1 to output the waveform1 */
	setSequenceStep (CHANNEL_A, 1, 2048, 720, 1, SEQMODE_SINGLE, SEQ_AUTO, SEQ_NONE, 1, 0x0);

	/* program the sequence step 2 to output the waveform1 */
	setSequenceStep (CHANNEL_A, 2, 5000, 360, 1, SEQMODE_SINGLE, SEQ_AUTO, SEQ_NONE, 1, 0x4);

	/* set the operating mode and start trigger */
	setOperatingMode(CHANNEL_A, MODE_TRIG, SIG_LOW, SLOPE_HIGH, 1);

	/* disable start/stop by setting source to low and slope high */
	setStopStartSignal (CHANNEL_A, SIG_LOW, SLOPE_HIGH, SIG_LOW, SLOPE_HIGH);

	/* arm the channel */
	armChannel(CHANNEL_A, 1, CONNECT);

	/* manualy start the waveform */
	manualTrigger(CHANNEL_A, START_TRIG);

	return 0;
}

/*
	This function programs a two waveforms on channel A
	and runs them in continuous mode.  Start/Stop set to the
	rising edge of timer A (20us).

	  sample clock: 50MHz
	  waveform1:    720 point sine
	  waveform2:    360 point triangle
	  amplitude:    2V
	  offset:       0V
	  mode:         continuous
	  start/stop:    Rising edge TIMERA = 20us
*/
int example3 (void)
{
	int i, status;
	unsigned short array[720];
	double count, wData, pi = 3.1415926;

	/* get channel A status to check if armed */
	status = getStatus(CHANNEL_A);

	/* reset channel A if armed */
	if (status & 1)
	    stopChannel(CHANNEL_A, RESET);

	/* set 2V amplitude and 0V offset */
	setGainAndOffset(CHANNEL_A, 2.0, 0.0);

	/* program the sample clock to 75MHz */
	setSampleClock(CHANNEL_A, INTERNAL, 125e6, INTERNAL);

	/* generat sine wave */
	for (i = 0, count = 0.0; count < 360.0; i++, count += 0.5) {
	    wData = sin (count * pi/180);
	    array[i] = (unsigned short) ((8190.0 * wData)+8191);
	}

	/* upload the waveform to the MA1801A */
	setWavformMemory(CHANNEL_A, PRIMARY, 2048, 720, array);

	/* generat triangle wave */
	for (i = 0; i <= 90; i++)
	    array[i] = (unsigned short) (i * 8192.0 / 90.0) + 8191;
	for (i = 91; i <= 270; i++)
	    array[i] = (unsigned short) ((180 - i) * 8192.0 / 90.0) + 8192;
	for (i = 271; i < 360; i++)
	    array[i] = (unsigned short) ((i - 360) * 8192.0 / 90.0) + 8192;

	/* upload the waveform to the MA1801A */
```

```
    setWavformMemory(CHANNEL_A, PRIMARY, 5000, 360, array);

    /* program the sequence step 1 to output the waveform1 */
    setSequenceStep (CHANNEL_A, 1, 2048, 720, 1, SEQMODE_SINGLE, SEQ_AUTO, SEQ_NONE, 1, 0x0);

    /* program the sequence step 2 to output the waveform1 */
    setSequenceStep (CHANNEL_A, 2, 5000, 360, 1, SEQMODE_SINGLE, SEQ_AUTO, SEQ_NONE, 1, 0x4);

    /* set the operating mode and start trigger */
    setOperatingMode(CHANNEL_A, MODE_CONT, SIG_LOW, SLOPE_HIGH, 1);

    /* program TIMERA for a 20us period */
    setTimerPeriod (CHANNEL_A, 20e-6);

    /* set the start/stop source to rising edge timerA */
    setStopStartSignal (CHANNEL_A, SIG_INTA, SLOPE_RISE, SIG_INTA, SLOPE_RISE);

    /* arm the channel */
    armChannel(CHANNEL_A, 1, CONNECT);

    return 0;
}

/*
    This function programs a two waveforms on channel A
    and runs them in continuous mode.  Sync pulse set to step
    mode and routed to sigout1a.

      sample clock: 50MHz
      waveform1:    720 point sine
      waveform2:    360 point triangle
      amplitude:    2V
      offset:       0V
      mode:         continuous
      sync pulse:   step 1, offset 0 for 90 words (180 samples)
*/
int example4 (void)
{
    int i, status;
    unsigned short array[720];
    double count, wData, pi = 3.1415926;

    /* get channel A status to check if armed */
    status = getStatus(CHANNEL_A);

    /* reset channel A if armed */
    if (status & 1)
        stopChannel(CHANNEL_A, RESET);

    /* set 2V amplitude and 0V offset */
    setGainAndOffset(CHANNEL_A, 2.0, 0.0);

    /* program the sample clock to 75MHz */
    setSampleClock(CHANNEL_A, INTERNAL, 125e6, INTERNAL);

    /* generat sine wave */
    for (i = 0, count = 0.0; count < 360.0; i++, count += 0.5) {
        wData = sin (count * pi/180);
        array[i] = (unsigned short) ((8190.0 * wData)+8191);
    }

    /* upload the waveform to the MA1801A */
    setWavformMemory(CHANNEL_A, PRIMARY, 2048, 720, array);

    /* generat triangle wave */
    for (i = 0; i <= 90; i++)
        array[i] = (unsigned short) (i * 8192.0 / 90.0) + 8191;
    for (i = 91; i <= 270; i++)
        array[i] = (unsigned short) ((180 - i) * 8192.0 / 90.0) + 8192;
    for (i = 271; i < 360; i++)
        array[i] = (unsigned short) ((i - 360) * 8192.0 / 90.0) + 8192;

    /* upload the waveform to the MA1801A */
    setWavformMemory(CHANNEL_A, PRIMARY, 5000, 360, array);

    /* program the sequence step 1 to output the waveform1 */
    setSequenceStep (CHANNEL_A, 1, 2048, 720, 1, SEQMODE_SINGLE, SEQ_AUTO, SEQ_NONE, 1, 0x0);

    /* program the sequence step 2 to output the waveform1 */
    setSequenceStep (CHANNEL_A, 2, 5000, 360, 1, SEQMODE_SINGLE, SEQ_AUTO, SEQ_NONE, 1, 0x4);

    /* set the operating mode and start trigger */
    setOperatingMode(CHANNEL_A, MODE_CONT, SIG_LOW, SLOPE_HIGH, 1);

    /* set the sync pulse to active low, step mode, offset 0, width 90, step 1 */
    setSyncOutput (CHANNEL_A, SYNC_LOW, SYNC_STEP, 0, 90, 1, 0);

    /* enable the sync replace mode */
    setSyncReplace (CHANNEL_A, REPLACE_ON, 5000);

    /* set the start/stop source to rising edge timerA */
    setFrontPanelOutput (CHANNEL_A, FPO_SYNC, FPO_LOW);
```

```
     /* arm the channel */
     armChannel(CHANNEL_A, 1, CONNECT);

     return 0;
}

/*
     This function programs a single waveform on channel A
     and runs it in triggered mode using timerA as the trigger.
     Channel B is programed with two waveforms in triggerd mode
     using TIMERB as the trigger.

     Channel A
      sample clock: 75MHz
      waveform:     360 point sine^3
      amplitude:    5V
      offset:       2.5V
      mode:         triggered, TIMERA (20us)
    Channel B
      sample clock: 125MHz
      waveform1:    720 point sine
      waveform2:    360 point triangle
      amplitude:    2V
      offset:       0V
      mode:         triggered, manual
*/
int example5 (void)
{
     int i, status;
     unsigned short array[720];
     double count, wData, pi = 3.1415926;

     /* get channel A status to check if armed */
     status = getStatus(CHANNEL_A);

     /* reset channel A if armed */
     if (status & 1)
         stopChannel(CHANNEL_A, RESET);

     /* set 5V amplitude and 2.5V offset */
     setGainAndOffset(CHANNEL_A, 5.0, 2.5);

     /* program the sample clock to 75MHz */
     setSampleClock(CHANNEL_A, INTERNAL, 75e6, INTERNAL);

     /* generat sine^3 wave */
     for (i = 0; i < 360; i++) {
         wData = sin (i * pi/180);
         array[i] = (unsigned short) ((8190.0 * wData * wData * wData)+8191);
     }

     /* upload the waveform to the MA1801A */
     setWavformMemory(CHANNEL_A, PRIMARY, 2048, 360, array);

     /* program the sequence to output the waveform */
     setSequenceStep (CHANNEL_A, 1, 2048, 360, 1, SEQMODE_SINGLE, SEQ_AUTO, SEQ_NONE, 1, 0x4);

     /* program TIMERA for a 20us period */
     setTimerPeriod (CHANNEL_A, 20e-6);

     /* set the operating mode and start trigger */
     setOperatingMode(CHANNEL_A, MODE_TRIG, SIG_INTA, SLOPE_HIGH, 2);

     /* disable start/stop by setting source to low and slope high */
     setStopStartSignal (CHANNEL_A, SIG_LOW, SLOPE_HIGH, SIG_LOW, SLOPE_HIGH);

     /* disable the sync replace mode */
     setSyncReplace (CHANNEL_A, REPLACE_OFF, 5000);

     /* arm the channel */
     armChannel(CHANNEL_A, 1, CONNECT);

     /* get channel B status to check if armed */
     status = getStatus(CHANNEL_B);

     /* reset channel B if armed */
     if (status & 1)
         stopChannel(CHANNEL_B, RESET);

     /* set 2V amplitude and 0V offset */
     setGainAndOffset(CHANNEL_B, 2.0, 0.0);

     /* program the sample clock to 75MHz */
     setSampleClock(CHANNEL_B, INTERNAL, 125e6, INTERNAL);

     /* generat sine wave */
     for (i = 0, count = 0.0; count < 360.0; i++, count += 0.5) {
         wData = sin (count * pi/180);
         array[i] = (unsigned short) ((8190.0 * wData)+8191);
     }
```

```
    /* upload the waveform to the MA1801A */
    setWavformMemory(CHANNEL_B, PRIMARY, 0, 720, array);

    /* generat triangle wave */
    for (i = 0; i <= 90; i++)
        array[i] = (unsigned short) (i * 8192.0 / 90.0) + 8191;
    for (i = 91; i <= 270; i++)
        array[i] = (unsigned short) ((180 - i) * 8192.0 / 90.0) + 8192;
    for (i = 271; i < 360; i++)
        array[i] = (unsigned short) ((i - 360) * 8192.0 / 90.0) + 8192;

    /* upload the waveform to the MA1801A */
    setWavformMemory(CHANNEL_B, PRIMARY, 2000, 360, array);

    /* program the sequence step 1 to output the waveform1 */
    setSequenceStep (CHANNEL_B, 1, 0, 720, 1, SEQMODE_SINGLE, SEQ_AUTO, SEQ_NONE, 1, 0x0);

    /* program the sequence step 2 to output the waveform1 */
    setSequenceStep (CHANNEL_B, 2, 2000, 360, 1, SEQMODE_SINGLE, SEQ_AUTO, SEQ_NONE, 1, 0x4);

    /* set the operating mode and start trigger */
    setOperatingMode(CHANNEL_B, MODE_TRIG, SIG_LOW, SLOPE_HIGH, 1);

    /* disable start/stop by setting source to low and slope high */
    setStopStartSignal (CHANNEL_B, SIG_LOW, SLOPE_HIGH, SIG_LOW, SLOPE_HIGH);

    /* arm the channel */
    armChannel(CHANNEL_B, 1, CONNECT);

    /* manualy start the waveform */
    manualTrigger(CHANNEL_B, START_TRIG);

    return 0;
}

/*
    This function programs a single waveform on channel A
    and runs it in triggered mode using timerA as the trigger.
    Channel B is programed with two waveforms in triggerd mode
    using coupled to channel A.

    Channel A
     sample clock: 75MHz
     waveform:     360 point sine^3
     amplitude:    5V
     offset:       2.5V
     mode:         triggered, TIMERA (20us)
   Channel B
     sample clock: 125MHz
     waveform1:    720 point sine
     waveform2:    360 point triangle
     amplitude:    2V
     offset:       0V
     mode:         triggered, manual
*/
int example6 (void)
{
    int i, status;
    unsigned short array[720];
    double count, wData, pi = 3.1415926;

    /* get channel A status to check if armed */
    status = getStatus(CHANNEL_A);

    /* reset channel A if armed */
    if (status & 1)
        stopChannel(CHANNEL_A, RESET);

    /* set 5V amplitude and 2.5V offset */
    setGainAndOffset(CHANNEL_A, 5.0, 2.5);

    /* program the sample clock to 75MHz */
    setSampleClock(CHANNEL_A, INTERNAL, 75e6, INTERNAL);

    /* generat sine^3 wave */
    for (i = 0; i < 360; i++) {
        wData = sin (i * pi/180);
        array[i] = (unsigned short) ((8190.0 * wData * wData * wData)+8191);
    }

    /* upload the waveform to the MA1801A */
    setWavformMemory(CHANNEL_A, PRIMARY, 2048, 360, array);

    /* program the sequence to output the waveform */
    setSequenceStep (CHANNEL_A, 1, 2048, 360, 1, SEQMODE_SINGLE, SEQ_AUTO, SEQ_NONE, 1, 0x4);

    /* program TIMERA for a 20us period */
    setTimerPeriod (CHANNEL_A, 20e-6);

    /* set the operating mode and start trigger */
    setOperatingMode(CHANNEL_A, MODE_TRIG, SIG_INTA, SLOPE_HIGH, 2);
```

```
    /* disable start/stop by setting source to low and slope high */
    setStopStartSignal (CHANNEL_A, SIG_LOW, SLOPE_HIGH, SIG_LOW, SLOPE_HIGH);

    /* disable the sync replace mode */
    setSyncReplace (CHANNEL_A, REPLACE_OFF, 5000);

    /* get channel B status to check if armed */
    status = getStatus(CHANNEL_B);

    /* reset channel B if armed */
    if (status & 1)
        stopChannel(CHANNEL_B, RESET);

    /* set 2V amplitude and 0V offset */
    setGainAndOffset(CHANNEL_B, 2.0, 0.0);

    /* generate sine wave */
    for (i = 0, count = 0.0; count < 360.0; i++, count += 0.5) {
        wData = sin (count * pi/180);
        array[i] = (unsigned short) ((8190.0 * wData)+8191);
    }

    /* upload the waveform to the MA1801A */
    setWavformMemory(CHANNEL_B, PRIMARY, 0, 720, array);

    /* generat triangle wave */
    for (i = 0; i <= 90; i++)
        array[i] = (unsigned short) (i * 8192.0 / 90.0) + 8191;
    for (i = 91; i <= 270; i++)
        array[i] = (unsigned short) ((180 - i) * 8192.0 / 90.0) + 8192;
    for (i = 271; i < 360; i++)
        array[i] = (unsigned short) ((i - 360) * 8192.0 / 90.0) + 8192;

    /* upload the waveform to the MA1801A */
    setWavformMemory(CHANNEL_B, PRIMARY, 2000, 360, array);

    /* program the sequence step 1 to output the waveform1 */
    setSequenceStep (CHANNEL_B, 1, 0, 720, 1, SEQMODE_SINGLE, SEQ_AUTO, SEQ_NONE, 1, 0x0);

    /* program the sequence step 2 to output the waveform1 */
    setSequenceStep (CHANNEL_B, 2, 2000, 360, 1, SEQMODE_SINGLE, SEQ_AUTO, SEQ_NONE, 1, 0x4);

    /* set the operating mode and start trigger */
    setOperatingMode(CHANNEL_B, MODE_TRIG, SIG_LOW, SLOPE_HIGH, 1);

    /* couple channel B */
    setChannelCoupling (COUPLE_INT, COUPLE_OFF, COUPLE_ON);


    /* trigger delay exercize */
    setTriggerDelay (CHANNEL_B, 200);

    /* clock divider exercize */
    setClockDivider (CHANNEL_A, 3);

    /* arm channel B */
    armChannel(CHANNEL_B, 1, CONNECT);

    /* arm channel A */
    armChannel(CHANNEL_A, 1, CONNECT);

    return 0;
}

/*
    This function initializes the clock generator and DAC register
    sequence bits of the MA1801A module.
    Input:
        None
    Return Value:
        0
*/
int initModule(void)
{
    /* initialize clock generator sequence bits */
    Poke16(CLKGEN, 0x0F00);
    Poke16(CLKGEN + 2, 0xB0F0);
    Poke16(CLKGEN + 4, 0x0000);
    Poke16(CLKGEN + 6, 0x03E8);
    Poke16(CLKGEN + 8, 0x0700);
    Poke16(CLKGEN + 10, 0x2494);

    /* initialize channel A gain DAC rigister bits */
    Poke16(GAIN_A, 0x08C8);

    /* initialize channel A gain DAC rigister bits */
    Poke16(GAIN_B, 0x48C8);

    /* initialize channel A offset DAC rigister bits */
    Poke16(OFFSET_A, 0x87FF);

    /* initialize channel B offset DAC rigister bits */
```

```
     Poke16(OFFSET_B, 0xC7FF);

     /* initialize TRIG1A DAC rigister bits */
     Poke16(TRIG1AREF, 0x0940);

     /* initialize TRIG1B DAC rigister bits */
     Poke16(TRIG1BREF, 0x4940);

     /* initialize REFCLK DAC rigister bits */
     Poke16(RCLKREF, 0x8940);

     /* initialize EXTCLK DAC rigister bits */
     Poke16(ECLKREF, 0xC940);

     return 0;
}

#define RUNCTRL     0x0
/*
     This function returns the status bits of the
     specified MA1801A channel.
     Input:
          channel: (0 = channel A, 1 = channel B)
     Return Value:
          Bit 0 = (0 = Not armed, 1 = Armed).
          Bit 1 = (0 = Stopped, 1 = Running).
          Bit 3 = (0 = Sequence not active, 1 = sequence active).
*/
int getStatus(short channel)
{
     unsigned short data;

     /* read run control register */
     Peek16(RUNCTRL, &data);

     /* shift the arm bit if channel is B */
     if (channel == CHANNEL_B)
          data >>= 8;

     /* return the status bits */
     return data & 0xB;
}


#define RUNCTRL            0x0
#define STARTCTRL          0xA
/*
     This function stops the specified MA1801A channel.
     The channel can be stopped by disarming or resetting.
     Input:
          channel: (0 = channel A, 1 = channel B)
          type:    (0 = disarm, 1 = reset, 2 = isolate)
     Return Value:
          0
*/
int stopChannel(short channel, short type)
{
     unsigned short runctrl, startctrl;

     /* read run control register */
     Peek16(RUNCTRL, &runctrl);

     /* set the arm bit low if type == DISARM */
     if (type == DISARM)
          /* only disarm the specified channel */
          if (channel == CHANNEL_A)
               /* mask off channel A arm bit */
               runctrl &= 0xFFFE;
          else
               /* mask off channel B arm bit */
               runctrl &= 0xFEFF;
     else if (type == RESET || type == ISOLATE)
          if (type == ISOLATE) {
               /* read start control register */
               Peek16(STARTCTRL, &startctrl);

               /* isolate the output before resetting */
               if (channel == CHANNEL_A)
                    /* isolate channel A */
                    startctrl &= 0xFF3F;
               else
                    /* isolate channel B */
                    startctrl &= 0x3FFF;

               /* write start control register */
               Poke16(STARTCTRL, startctrl);
          }

          /* only reset the specified channel */
          if (channel == CHANNEL_A)
               /* set channel A reset bit */
               runctrl |= 0x8;
```

```
          else
               /* set channel B reset bit */
               runctrl |= 0x800;

     /* program the control register */
     Poke16(RUNCTRL, runctrl);

     return 0;
}


#define RUNCTRL            0x0
#define GAIN_A             0x82
#define OFFSET_A           0x8A
/*
     This function programs the gain and offset
     of the specified MA1801A channel.
     Input:
          channel: (0 = channel A, 1 = channel B)
          gain:    (0.01 to 10.0)
          offset:  (-10.0 to 10.0)
     Return Value:
          0
*/
int setGainAndOffset(short channel, double gain, double offset)
{
     int gainAddr = GAIN_A, offsetAddr = OFFSET_A;
     unsigned short gainData = 0, offsetData = 0, runCtrl;

     /* update the addresses if channel B */
     if (channel == CHANNEL_B) {
          gainAddr += 4;
          gainData = 0x4000;
          offsetAddr += 4;
          offsetData = 0x4000;
     }

     /* calculate gain data */
     gainData += (gain * 200) + 2048;

     /* calculate offset data */
     offsetData += 0x8000 + (offset * 400) + 2048;

     /* program the gain register */
     Poke16(gainAddr, gainData);

     /* program the offset register */
     Poke16(offsetAddr, offsetData);

     /* update the reference DACs */
     /* read run control register */
     Peek16(RUNCTRL, &runCtrl);

     /* preserve arm bits and set the D/A update bit */
     runCtrl = (runCtrl & 0x101) | 0x8000;

     /* program the run control register */
     Poke16(RUNCTRL, runCtrl);

     /* query until D/A update complete */
     do {
          /* read run control register */
          Peek16(RUNCTRL, &runCtrl);
     } while ((runCtrl & 0x8000) == 0x8000);

     return 0;
}

#define CLKCTRL            0x34
#define CLKGEN             0x40
/*
     This function programs the sample clock source
     and frequency of the specified MA1801A channel.
     Input:
          channel: (0 = channel A, 1 = channel B)
          source:  (0 = internal, 1 = external)
          freqHz:  (10 to 125,000,000)
     Return Value:
          0
*/
int setSampleClock(short channel, short source, double freqHz, short refSource)
{
     int div1, div2, fbdiv, cgdata;
     unsigned short data, clkctrl;
     double cgfreq;
     short coarseTune = 7;
     time_t t1, t2;

     /* read clock control register  */
     Peek16(CLKCTRL, &clkctrl);

     /* set the sample clock source */
```

```
if (channel == CHANNEL_A) {
    /* set new value */
    clkctrl = (clkctrl & 0xFCFF) + (source << 8);
} else {
    /* set new value */
    clkctrl = (clkctrl & 0xF3FF) + (source << 10);
}

/* set internal clock generator */
if (freqHz == 0.0 || source == EXTERNAL) {
    /* get LSW of clock generator register 2 */
    Peek16(CLKGEN + 6, &data);

    /* disable the generator */
    data |= 0x2000;

    /* set LSW of clock generator register 2 */
    Poke16(CLKGEN + 6, data);
} else {
    /* claculate the control register divide count */
    for (cgfreq = freqHz, div2 = 0; cgfreq < 20000000.0; div2++)
        cgfreq *= 10;

    /* calculate the generator divide count and feedback */
    if (cgfreq < 5,000,000.00) {
        div1 = 0xB;  /* 40 */
        fbdiv = cgfreq * 0.002;
    } else if (cgfreq < 10000000.00) {
        div1 = 0xA;  /* 20 */
        fbdiv = cgfreq * 0.001;
    } else if (cgfreq < 20000000.00) {
        div1 = 0x9;      /* 10 */
        fbdiv = cgfreq * 0.0005;
    } else if (cgfreq < 50000000.00) {
        div1 = 0x2;      /* 4 */
        fbdiv = cgfreq * .0002;
    } else if (cgfreq < 100000000.00) {
        div1 = 0x1;      /* 2 */
        fbdiv = cgfreq * .0001;
    } else {
        div1 = 0;        /* 1 */
        fbdiv = cgfreq * .00005;
    }

    /* program new post divide value and reference enable */
    if (channel == CHANNEL_A)
        clkctrl = (clkctrl & 0xfff0) + div2;
    else
        clkctrl = (clkctrl & 0xff0f) + (div2 << 4);

    /* calculate clock generator register 2 */
    cgdata = ((fbdiv & 0xFF) << 24) + (div1 << 16) + (refSource << 12) + 0x3E8;

    /* write clock generator register 2 LSW */
    Poke16 (CLKGEN + 6, cgdata & 0xFFFF);

    /* write clock generator register 2 MSW */
    Poke16 (CLKGEN + 4, cgdata >> 16);

    /* calculate clock generator register 3 */
    cgdata = (coarseTune << 24) + (fbdiv >> 8) + (refSource * (7 << 17)) + 0x2480;

    /* write clock generator register 3 LSW */
    Poke16 (CLKGEN + 10, cgdata & 0xFFFF);

    /* write clock generator register 3 MSW */
    Poke16 (CLKGEN + 8, cgdata >> 16);

}

/* get LSW of clock generator register 1 */
Peek16(CLKGEN + 2, &data);

/* select the generator */
if (channel == CHANNEL_A)
    data &= 0xF7FF;
else
    data |= 0x800;

/* set LSW of clock generator register 1 */
Poke16(CLKGEN + 2, data);

/* write out clock control register  */
Poke16(CLKCTRL, clkctrl + 0x1000);

do {
    /* read clock control register and check bit 12  */
    Peek16 (CLKCTRL, &clkctrl);
} while (clkctrl & 0x1000);

/* delay for clock generator to settle */
t1 = clock();
```

```
     do {
          t2 = clock();
     } while ((t2 - t1) < 3);

     return 0;
}

#define MEMORY              0x2
#define EMADDR              0xC0
#define EMDATA              0xC4
/*
     This function programs the waveform memory
     of the specified MA1801A channel.
     Input:
          channel: (0 = channel A, 1 = channel B)
          memory:  (0 = primary, 1 = secondary)
          offset:  (0 - 0x3FFFF)
     Return Value:
          0
*/
int setWavformMemory(short channel, short memory, int offset, int samples, unsigned short data[])
{
     unsigned short membsy, memgrnt, memreg;
     unsigned short emaddr, *emdata;
     int error = 0;

     while (0x1) {
          /* read the memory register */
          Peek16(MEMORY, &memreg);

          /* update the memory variable to include the channel */
          memory += (channel * 2);

          /* check the grant bit to see if we have to request memory */
          memgrnt = 1 << (4 + memory);
          if ((memgrnt & memreg) == 0) {
               /* check the busy bit befor requesting memory */
               membsy = memgrnt << 4;
               if ((membsy & memreg)) {
                    error = ERROR_MEMREQ;
                    break;
               }
               /* request and select memory */
               memreg = (memory << 12) + (1 << memory);
               Poke16(MEMORY, memreg);

               /* read the memory register */
               Peek16(MEMORY, &memreg);

               /* did we get memory ? */
               if ((memgrnt & memreg) == 0) {
                    error = ERROR_MEMREQ;
                    break;
               }
          }

          /* write the extended memory address and page MSW */
          emaddr = (offset >> 16);
          Poke16(EMADDR, emaddr);

          /* write the extended memory address and page LSW */
          emaddr = offset;
          Poke16(EMADDR + 2, emaddr);

          /* write data samples */
          for (emdata = data; samples > 0; samples --)
               Poke16(EMDATA, *emdata++);

          /* release the memory */
          Poke16(MEMORY, memreg & 0xFFF0);

          break;
     }
     return error;
}


#define MEMORY              0x2
#define EMADDR              0xC0
#define EMDATA              0xC4
/*
     This function programs the sequency memory
     of the specified MA1801A channel.
     Input:
          channel         (0 = channel A, 1 = channel B)
          step            (1 - 512)
          waveMemoryAddr  (0 - 0x7FFFC)
          wavePoints      (0 - 0xFFFFE)
          loop            (1 - 65535)
          mode            (1 = Step, 2 = Single, 3 = Single1)
          source          (0 = signal1 low, 1 = signal1 high, 2 = signal1 falling
                           3 = signal1 rising, 4 = signal2 low, 5 = signal2 high
```

```
                          6 = signal2 falling, 7 = signal2 rising, 9 = primary not granted
                          11 = secondary not granted, 14 = none, 15 = auto)
          jumpSource      (0 = signal1 low, 1 = signal1 high, 2 = signal1 falling
                          3 = signal1 rising, 4 = signal2 low, 5 = signal2 high
                          6 = signal2 falling, 7 = signal2 rising, 8 = primary granted,
                          9 = primary not granted, 10 = secondary granted,
                          11 = secondary not granted, 12 = timeout,
                          14 = unconditional, 15 = auto)
          jumpSequence    (1 - 512)
          flags           (Bit 0 = Sequence Flag 1, Bit 1 = Sequence Flag 2,
                          Bit 2 = Sequence Stop, Bit 3 = Jump Return,
                          Bit 4 = Jump Immediate)
     Return Value:
          0
*/
int setSequenceStep (short channel,
                     short step,
                     int waveMemoryAddr,
                     int wavePoints,
                     int loop,
                     short mode,
                     short source,
                     short jumpSource,
                     short jumpSequence,
                     short flags)
{
     int seqData[4] = {0,0,0,0};
     short i;
     int addr;

     if (channel == CHANNEL_A)
          /* select the sequence memory channel A */
          Poke16(MEMORY, 0x4000);
     else
          /* select the sequence memory channel B */
          Poke16(MEMORY, 0x5000);

     /* start filling seqData array with data */
     seqData[0] = (waveMemoryAddr / 4) & 0x7FFFF;
     seqData[1] = (wavePoints / 2) - 1;

     /* set the mode */
     switch (mode) {
          case SEQMODE_SINGLE:
               mode = 0;
               break;
          case SEQMODE_STEP:
               mode = 1;
               break;
          case SEQMODE_SINGLE1:
               mode = 2;
               break;
          default:
               mode = 0;
               break;
     }

     /* set the mode */
     seqData[1] += (mode << 28);

     /* set the source */
     seqData[1] += (source << 24);

     /* set the loop */
     seqData[2] += (loop - 1);

     /* set the jump source */
     seqData[1] += (jumpSource << 20);

     /* set the jump address */
     seqData[0] += ((jumpSequence - 1) << 19);

     /* set the flags */
     seqData[0] += ((flags & 0xF) << 28);
     seqData[1] += ((flags & 0x10) << 15);

     /* write the extended memory address and page MSW */
     addr = (step - 1) * 0x10;
     Poke16(EMADDR, addr >> 16);

     /* write the extended memory address and page LSW */
     Poke16(EMADDR + 2, addr & 0xFFFF);

     /* write sequence data */
     for (i = 0; i < 3; i++) {
          Poke16(EMDATA, seqData[i] >> 16);
          Poke16(EMDATA, seqData[i] & 0xFFFF);
     }

     return 0;
}
```

```
#define TIMERB              0x38
#define TIMERA              0x3A
/*
     This function programs the trigger timer period
     of the specified MA1801A channel.
     Input:
         channel: (0 = channel A, 1 = channel B)
      triggerPeriod: (20e-6 - 1.3107)
     Return Value:
         0
*/
int setTimerPeriod (short channel, double triggerPeriod)
{
     unsigned short timer;
     int addr;

     /* convert trigger period */
     timer = (unsigned short) ((triggerPeriod / 20e-6) - 1);

     /* set the address */
     addr = (channel == CHANNEL_A)? TIMERA: TIMERB;

     /* write the burst control register */
     Poke16(addr, timer);

     return 0;
}

#define STARTCTRL           0xA
#define BURSTLOOPB          0x2C
#define BURSTLOOPA          0x2E
#define BURST               0x32
/*
     This function programs the start trigger source
     of the specified MA1801A channel.
     Input:
         channel: (0 = channel A, 1 = channel B)
         mode:  (0 = continuous, 1 = triggerd,
                 2 = gated, 3 = burst)
      startTrig: (0 = TRIG1A, 1 = TRIG2A, 2 = TRIG1B,
                  3 = TRIG2B, 4 = MARKER1A, 5 = MARKER2A,
                  6 = MARKER1B, 7 = MARKER2B, 8 = TRIGA,
                  9 = TRIGB, 12 = TIMERA, 13 = TIMERB,
                  14 = High Level, 15 = Low Level)
         slope:  (0 = low level, 1 = high level,
                  2 = falling edge, 3 = rising edge)
         burstCount: (1 - 65535)
     Return Value:
         0
*/
int setOperatingMode(short channel, short mode, short startTrig, short slope, int burstCount)
{
     unsigned short startctrl, burst;

     /* Read the start control register */
     Peek16(STARTCTRL, &startctrl);

     /* Read the burst control register */
     Peek16(BURST, &burst);

     /* set trigger based on new mode */
     if (mode == MODE_CONT) {
         /* set trigger low and start low if continuous mode */
         startTrig = 0xF;

         /* set burst continuous bit */
         burst |= (channel == CHANNEL_A)? 0x10: 0x1000;
     } else {
         /* adjust burst count based on mode */
         if (mode == MODE_BURST)
             burstCount--;
         else
             burstCount = 0;

         /* reset burst continuous bit */
         burst &= (channel == CHANNEL_A)? 0x1F00: 0x1F;

         /* set upper nibble burst count */
         burst |= (channel == CHANNEL_A)? ((burstCount & 0xF0000) >> 16): ((burstCount & 0xF0000) >> 8);

         /* add in trigger slope */
         startTrig += (slope << 4);

         /* adjust trigger to edge if not gate mode */
         if (mode != MODE_GATED)
             startTrig += 0x20;
     }

     /* write the burst control register */
     Poke16(BURST, burst);

     /* update and write the start control and burst count registers */
```

```
     if (channel == CHANNEL_A) {
         startctrl = (startctrl & 0xFFC0) + startTrig;
         Poke16(BURSTLOOPA, burstCount);
     } else {
         startctrl = (startctrl & 0xC0FF) + (startTrig << 8);
         Poke16(BURSTLOOPB, burstCount);
     }
     Poke16(STARTCTRL, startctrl);

     return 0;
}

#define RUNCTRL            0x0
#define STARTCTRL          0xA
#define FIRSTSEQB          0x3C
#define FIRSTSEQA          0x3E
/*
     This function connects and arms
     the specified MA1801A channel.
     Input:
         channel: (0 = channel A, 1 = channel B)
         outputMode:  (1 = Connect output without filter,
                       2 = Connect output with filter)
     Return Value:
         0
*/
int armChannel(short channel, short step, short outputMode)
{
     unsigned short runctrl, startctrl;

     /* get run register */
     Peek16(RUNCTRL, &runctrl);

     /* get start register */
     Peek16(STARTCTRL, &startctrl);

     /* update start and run control registers */
     if (channel == CHANNEL_A) {
         /* mask non selected channel and set arm bit of selected channel */
         runctrl = (runctrl & 0x100) + 0x1;

         /* set the output mode */
         startctrl = (startctrl & 0xFF3F) + (outputMode << 6);

         /* set the sequence step */
         Poke16(FIRSTSEQA, step - 1);
     } else {
         /* mask non selected channel and set arm bit of selected channel */
         runctrl = (runctrl & 1) + 0x100;

         /* set the output mode */
         startctrl = (startctrl & 0x3FFF) + (outputMode << 14);

         /* set the sequence step */
         Poke16(FIRSTSEQB, step - 1);
     }

     /* set start register */
     Poke16(STARTCTRL, startctrl);

     /* set run register */
     Poke16(RUNCTRL, runctrl);

     return 0;
}
/*
     This function sets the stop/start signals for
     the specified MA1801A channel.
     Input:
        channel:       (0 = channel A, 1 = channel B)
        stopSignal:    (0 = TRIG1A, 1 = TRIG2A, 2 = TRIG1B,
                        3 = TRIG2B, 4 = MARKER1A, 5 = MARKER2A,
                        6 = MARKER1B, 7 = MARKER2B, 8 = TRIGA,
                        9 = TRIGB, 12 = TIMERA, 13 = TIMERB,
                        14 = High Level, 15 = Low Level)
        stopSlope:     (0 = low level, 1 = high level,
                        2 = falling edge, 3 = rising edge)
        restartSignal: (0 = TRIG1A, 1 = TRIG2A, 2 = TRIG1B,
                        3 = TRIG2B, 4 = MARKER1A, 5 = MARKER2A,
                        6 = MARKER1B, 7 = MARKER2B, 8 = TRIGA,
                        9 = TRIGB, 12 = TIMERA, 13 = TIMERB,
                        14 = High Level, 15 = Low Level)
        restartSlope:  (0 = low level, 1 = high level,
                        2 = falling edge, 3 = rising edge)
     Return Value:
         0
*/
int setStopStartSignal (short channel, short stopSig, short stopSlope, short restartSig, short restartSlope)
{
     int addr;
     short stopStartReg;
```

```
    /* get address */
    if (channel == CHANNEL_A)
        addr = STOPRESTARTA;
    else
        addr = STOPRESTARTB;

    /* format register */
    stopStartReg = stopSig + (stopSlope << 4) + (restartSig << 8) + (restartSlope << 12);

    /* write the start/stop register */
    Poke16(addr, stopStartReg);

    return 0;
}

/*
    This function generates a manual trigger to
    the specified MA1801A channel.
    Input:
        channel  (0 = channel A, 1 = channel B)
        trigger  (0 = start, 1 = stop, 2 = restart, 3 = advance, 4 = jump)
    Return Value:
        0
*/
int manualTrigger(short channel, short trigger)
{
    unsigned short runctrl;

    /* get run register */
    Peek16(RUNCTRL, &runctrl);

    switch (trigger) {
    case START_TRIG:
        /* set manual start if not running */
        if (channel == CHANNEL_A)
            if ((runctrl & 0xB) != 1)
                break;
            else
                runctrl = (runctrl & 0x101) + 4;
        else
            if ((runctrl & 0xB00) != 0x100)
                break;
            else
                runctrl = (runctrl & 0x101) + 0x400;

        /* set run register */
        Poke16(RUNCTRL, runctrl);

        break;
    case STOP_TRIG:
        /* set manual stop if running */
        if (channel == CHANNEL_A)
            if ((runctrl & 0x2) != 0x2)
                break;
            else
                runctrl = (runctrl & 0x101) + 0x2;
        else
            if ((runctrl & 0x200) != 0x200)
                break;
            else
                runctrl = (runctrl & 0x101) + 0x200;

        /* set run register */
        Poke16(RUNCTRL, runctrl);

        break;
    case RESTART_TRIG:
        /* set manual restart if stopped */
        if (channel == CHANNEL_A)
            if ((runctrl & 0xA) != 0x8)
                break;
            else
                runctrl = (runctrl & 0x101) + 0x4;
        else
            if ((runctrl & 0xA00) != 0x800)
                break;
            else
                runctrl = (runctrl & 0x101) + 0x400;

        /* set run register */
        Poke16(RUNCTRL, runctrl);

        break;
    case ADVANCE_TRIG:
        /* set manual advance */
        if (channel == CHANNEL_A)
            runctrl = (runctrl & 0x101) + 0x10;
        else
            runctrl = (runctrl & 0x101) + 0x1000;

        /* set run register */
        Poke16(RUNCTRL, runctrl);
```

```
        break;
    case JUMP_TRIG:
        /* set manual jump */
        if (channel == CHANNEL_A)
            runctrl = (runctrl & 0x101) + 0x20;
        else
            runctrl = (runctrl & 0x101) + 0x2000;

        /* set run register */
        Poke16(RUNCTRL, runctrl);

        break;
    }

    return 0;
}

/*
    This function programs the front panel output signal
    of the specified MA1801A channel.
    Input:
       channel  (0 = channel A, 1 = channel B)
       sigOut1  (0 = SYNC, 1 = MARKER1, 2 = MARKER2,
                 3 = SEQFLAG1, 4 = SEQFLAG2, 5 = TIMERA,
                 6 = TIMERB, 7 = GTRIG1, 8 = GTRIG2,
                 9 = GEXTCLK, 10 = TRIGA, 11 = TRIGB,
                 12 = SEQACTIVE, 13 = RUN, 14 = ADVANCE,
                 15 = LOW)
       sigOut2  (0 = SYNC, 1 = MARKER1, 2 = MARKER2,
                 3 = SEQFLAG1, 4 = SEQFLAG2, 5 = TIMERA,
                 6 = TIMERB, 7 = GTRIG1, 8 = GTRIG2,
                 9 = GEXTCLK, 10 = TRIGA, 11 = TRIGB,
                 12 = SEQACTIVE, 13 = RUN, 14 = ADVANCE,
                 15 = LOW)
    Return Value:
        0
*/
int setFrontPanelOutput (short channel, short sigOut1, short sigOut2)
{
    unsigned short outputSig;

    /* get output signal register */
    Peek16(OUTPUTSIG, &outputSig);

    /* program the signals of the selected channel */
    if (channel == CHANNEL_A)
        outputSig = (outputSig & 0xFF00) + (sigOut2 << 4) + sigOut1;
    else
        outputSig = (outputSig & 0xFF) + (sigOut2 << 12) + (sigOut1 << 8);

    /* write the output signal register */
    Poke16(OUTPUTSIG, outputSig);

    return 0;
}

/*
    This function programs the M-Module backplane output signal
    of the MA1801A module.
    Input:
       trigA  (0 = TRIG1A, 1 = TRIG2A, 2 = TRIG1B,
               3 = TRIG2B, 4 = MARKER1A, 5 = MARKER2A,
               6 = MARKER1B, 7 = MARKER2B, 8 = SEQFLAG1A,
               9 = SEQFLAG2A, 10 = SEQFLAG1B, 11 = SEQFLAG2B,
               12 = SYNCA, 13 = SYNCB, 14 = Not Used,
               15 = DISABLE)
       trigB  (0 = TRIG1A, 1 = TRIG2A, 2 = TRIG1B,
               3 = TRIG2B, 4 = MARKER1A, 5 = MARKER2A,
               6 = MARKER1B, 7 = MARKER2B, 8 = SEQFLAG1A,
               9 = SEQFLAG2A, 10 = SEQFLAG1B, 11 = SEQFLAG2B,
               12 = SYNCA, 13 = SYNCB, 14 = Not Used,
               15 = DISABLE)
    Return Value:
        0
*/
int setBackplaneOutput (short trigA, short trigB)
{
    unsigned short miscCtrl;

    /* get backplane output signal register */
    Peek16(MISCCTRL, &miscCtrl);

    /* program the signals */
    miscCtrl = (miscCtrl & 0xF00F) + (trigB << 8) + (trigA << 4);

    /* write the output signal register */
    Poke16(MISCCTRL, miscCtrl);

    return 0;
}
```

```
/*
     This function programs the trigger timeout period
     of the specified MA1801A channel.
     Input:
         channel         (0 = channel A, 1 = channel B)
        triggerTimeout  (20e-6 - 1.3107)
     Return Value:
         0
*/
int setTriggerTimeout (short channel,  double triggerTimeout)
{
     unsigned short timer;
     int addr;

     /* convert trigger timeout period */
     timer = (unsigned short) ((triggerTimeout / 20e-6) - 1);

     /* set the address */
     addr = (channel == CHANNEL_A)? TIMEOUTA: TIMEOUTB;

     /* write the trigger timeout register */
     Poke16(addr, timer);

     return 0;
}

/*
     This function returns the status register of the
     of the specified MA1801A channel.
     Input:
         channel  (0 = channel A, 1 = channel B)
     Output:
        status
           Bit 0  Waiting for Start; 0 = not waiting, 1 = waiting.
           Bit 1  Stop Enabled; 0 = not enabled, 1 = enabled.
           Bit 2  Waiting for restart; 0 = not waiting, 1 = waiting.
           Bit 3  Waiting for advance SINGLE1 mode; 0 = not waiting,
                  1 = waiting.
           Bit 4  Waitning for advance STEP mode; 0 = not waiting,
                  1 = waiting.
           Bit 5  Jump enabled; 0 = disabled, 1 = enabled.
           Bit 6  Gosub enabled; 0 = disabled, 1 = enabled.
           Bit 7  Replace enabled; 0 = disabled, 1 = enabled.
           Bit 8  Sweep; 0 = disabled, 1 = enabled.
           Bit 9  Clock Slow; 0 = clock > 200Hz, 1 = clock < 200Hz.
           Bit 10 Primary Memory Status; 0 = not busy, 1 = busy.
           Bit 11 Secondary Memory Status; 0 = not busy, 1 = busy.
           Bit 12 Primary Error; 0 = no error, 1 = memory accessed
                  while not released.
           Bit 13 Secondary Error; 0 = no error, 1 = memory accessed
                  while not released.
           Bit 14 Link Error; 0 = no error, 1 = channel A/B out of
                  sync.
           Bit 15 Master/Slave error: 0 = no error, 1 = master/slave
                  chain out of sync.
     Return Value:
         0
*/
int getStatusRegister (short channel, unsigned short *status)
{
     int addr;

     /* set the address */
     addr = (channel == CHANNEL_A)? STATUSA: STATUSB;

     /* read the status register */
     Peek16(addr, status);

     return 0;
}

/*
     This function returns the event register of the
     of the specified MA1801A channel.  A one indicates
     the event occured and is cleared after reading.
     Input:
         channel (0 = channel A, 1 = channel B)
     Output:
        event
           Bit 0  Start Signal.
           Bit 1  Stop Signal.
           Bit 2  Restart Signal.
           Bit 3  Single Advance Signal.
           Bit 4  Step Advance Signal.
           Bit 5  Jump Operation.
           Bit 6  Gosub Operation.
           Bit 7  Replace Operation.
           Bit 8  Sweep Operation.
           Bit 9  Clock Error.
           Bit 10 Primary Error.
           Bit 11 Secondary Error.
           Bit 12 Trigger Timeout.
```

```
            Bit 13 Not Used.
            Bit 14 Link Error.
            Bit 15 Master/Slave Error.
      Return Value:
          0
*/
int getEventRegister (short channel, unsigned short *event)
{
      int addr;

      /* set the address */
      addr = (channel == CHANNEL_A)? EVENTA: EVENTB;

      /* read the event register */
      Peek16(addr, event);

      return 0;
}

/*
      This function programs the event enable register of the
      of the specified MA1801A channel.  A one enables the
      event to generate an M-Module interrupt.
      Input:
          channel (0 = channel A, 1 = channel B)
      Output:
         enable
            Bit 0  Start Signal.
            Bit 1  Stop Signal.
            Bit 2  Restart Signal.
            Bit 3  Single Advance Signal.
            Bit 4  Step Advance Signal.
            Bit 5  Jump Operation.
            Bit 6  Gosub Operation.
            Bit 7  Replace Operation.
            Bit 8  Sweep Operation.
            Bit 9  Clock Error.
            Bit 10 Primary Error.
            Bit 11 Secondary Error.
            Bit 12 Trigger Timeout.
            Bit 13 Not Used.
            Bit 14 Link Error.
            Bit 15 Master/Slave Error.
      Return Value:
          0
*/
int setEventEnable (short channel, unsigned short enable)
{
      int addr;

      /* set the address */
      addr = (channel == CHANNEL_A)? ENABLEA: ENABLEB;

      /* write the event enable register */
      Poke16(addr, enable);

      return 0;
}

/*
      This function returns the current sequencer address
      status of the specified MA1801A channel.
      Input:
          channel (0 = channel A, 1 = channel B)
      Output:
         sequenceStep
         memoryAddr
      Return Value:
          0
*/
int getSequencerAddrStatus (short channel, short *sequenceStep, int *memoryAddr)
{
      int addr;
      unsigned short datalsb;
      unsigned short datamsb;

      /* set the address */
      addr = (channel == CHANNEL_A)? SEQSTATUSA: SEQSTATUSB;

      /* read the sequence status registers */
      Peek16(addr, &datalsb);
      Peek16(addr + 2, &datamsb);

      /* format the output data */
      *sequenceStep = (datalsb >> 3) & 0x1FF;
      *memoryAddr = ((datalsb & 0x7) << 16) + datamsb;

      return 0;
}


/*
```

```
     This function program the sync signal of the specified
     MA1801A channel.
     Input:
          channel (0 = channel A, 1 = channel B)
          level   (0 = active high, 1 = active low)
          source  (0 = Start Sequence, 1 = Sequence Step,
                   2 = Sequence Loop, 3 = Single Loop)
          offset  (0 to 2097151 words)
          width   (1 to 4095 words)
        step    (1 to 512)
        loop    (1 to 65535)
     Return Value:
          0
*/
int setSyncOutput (short channel, short level, short source, int offset, short width, short step, int loop)
{
     int addr;
     unsigned short sync1;
     unsigned short sync2;
     unsigned short sync3;
     unsigned short sync4;

     /* set the address */
     addr = (channel == CHANNEL_A)? SYNCREGA: SYNCREGB;

     /* read the sync control 3 register to preserve replace bit */
     Peek16(addr + 4, &sync3);

     /* format the sync control registers */
     sync1 = ((step - 1) << 5) + ((offset >> 16) & 0x1F);
     sync2 = (offset & 0xFFFF);
     sync3 = (sync3 & 0x4000) + (level << 15) + (source << 12) + width;
     sync4 = loop - 1;

     /* program the sync controls */
     Poke16 (addr, sync1);
     Poke16 (addr + 2, sync2);
     Poke16 (addr + 4, sync3);
     Poke16 (addr + 6, sync4);

     return 0;
}

/*
     This function program the sync replace of the specified
     MA1801A channel.
     Input:
          channel        (0 = channel A, 1 = channel B)
          state          (0 = replase off, 1 = replace on)
          waveMemoryAddr (0 - 0x7FFFC)
     Return Value:
          0
*/
int setSyncReplace (short channel, short state, int waveMemoryAddr)
{
     int addr;
     unsigned short sync3;
     unsigned short sync5;
     unsigned short sync6;

     /* set the address */
     addr = (channel == CHANNEL_A)? SYNCREGA: SYNCREGB;

     /* read the sync control 3 register to program replace bit */
     Peek16(addr + 4, &sync3);

     /* format the sync control registers */
     waveMemoryAddr /= 4;
     sync3 = (sync3 & 0xBFFF) + (state << 14);
     sync5 = (waveMemoryAddr >> 16) & 0x7;
     sync6 = (waveMemoryAddr & 0xFFFF);

     /* program the sync controls */
     Poke16 (addr + 4, sync3);
     Poke16 (addr + 8, sync5);
     Poke16 (addr + 10, sync6);

     return 0;
}

/*
     This function program the input threshold of the specified
     MA1801A front panel signal.
     Input:
          signal    (0 = TRIG1A, 1 = TRIG1B, 2 = EXTCLK, 3 = REFCLK)
          threshold (-10.00 to +10.00)
     Return Value:
          0
*/
int setInputThreshold (short signal, double threshold)
{
     unsigned short runCtrl;
```

```
    short dacData = 0;
    int addr = 0;

    switch (signal) {
    case 0:   /* TRIG1A */
        addr = TRIG1AREF;
        dacData = 0x800 + (short)(threshold / 0.005);
        break;
    case 1:  /* TRIG1B */
        addr = TRIG1BREF;
        dacData = 0x4800 + (short)(threshold / 0.005);
        break;
    case 2:  /* EXTCLK */
        addr = ECLKREF;
        dacData = 0xC800 + (short)(threshold / 0.0025);
        break;
    case 3:  /* REFCLK */
        addr = RCLKREF;
        dacData = 0x8800 + (short)(threshold / 0.0025);
        break;
    default:
        break;
    }

    if (addr > 0) {
        /* write the dac register */
        Poke16(addr, dacData);


        /* update the reference DACs */
        /* read run control register */
        Peek16(RUNCTRL, &runCtrl);

        /* preserve arm bits and set the D/A update bit */
        runCtrl = (runCtrl & 0x101) | 0x8000;

        /* program the run control register */
        Poke16(RUNCTRL, runCtrl);

        /* query until D/A update complete */
        do {
            /* read run control register */
            Peek16(RUNCTRL, &runCtrl);
        } while ((runCtrl & 0x8000) == 0x8000);
    }

    return 0;
}

/*
    This function program the sample clock divider of the specified
    MA1801A channel.
    Input:
        channel         (0 = channel A, 1 = channel B)
        clockDivider    (1 to 4294967296)
    Return Value:
        0
*/
int setClockDivider (short channel, double clockDivider)
{
    unsigned int divide;
    int addr;

    /* convert to unsigned int 32 */
    divide = (unsigned int)clockDivider - 1;

    /* get address */
    addr = (channel == CHANNEL_A)? SCDIVIDEA: SCDIVIDEB;

    /* write the upper 16 bits */
    Poke16 (addr, divide >> 16);

    /* write the lower 16 bits */
    Poke16 (addr + 2, divide & 0xFFFF);

    return 0;
}

/*
    This function program the coupling mode of both
    MA1801A channels.
    Input:
        mode      (0 = internal, 1 = external)
        coupleA   (0 = couple mode off, 1 = couple mode on)
        coupleB   (0 = couple mode off, 1 = couple mode on)
    Return Value:
        0
*/
int setChannelCoupling (short mode, short coupleA, short coupleB)
{
    unsigned short miscCtrl;
```

```
    /* get coupling register */
    Peek16(MISCCTRL, &miscCtrl);

    /* reset the coupling bits */
    miscCtrl &= 0xFFF0;

    /* set the bits */
    if (mode == COUPLE_INT) {
        if (coupleA == COUPLE_OFF && coupleB == COUPLE_ON)
            /* Set link B bit (A independant, B linked to A */
            miscCtrl |= 0x2;
        else if (coupleA == COUPLE_ON && coupleB == COUPLE_OFF)
            /* Set couple A and master bits (A master, B independant) */
            miscCtrl |= 0x5;
        else if (coupleA == COUPLE_ON && coupleB == COUPLE_ON)
            /* Set couple A, couple B and master bits (A master, B slave) */
            miscCtrl |= 0xD;
    } else if (mode == COUPLE_EXT) {
        if (coupleA == COUPLE_OFF && coupleB == COUPLE_ON)
            /* Set couple B bit (A independant, B slave) */
            miscCtrl |= 0x8;
        else if (coupleA == COUPLE_ON && coupleB == COUPLE_OFF)
            /* Set couple A bit (A slave, B independant) */
            miscCtrl |= 0x4;
        else if (coupleA == COUPLE_ON && coupleB == COUPLE_ON)
            /* Set couple A and couple B bits (A and B slave) */
            miscCtrl |= 0xC;
    }

    /* write the coupling register */
    Poke16 (MISCCTRL, miscCtrl);

    return 0;
}

/*
    This function program the start trigger delay of the specified
    MA1801A channel.
    Input:
        channel       (0 = channel A, 1 = channel B)
        triggerDelay  (0 - 65535)
    Return Value:
        0
*/
int setTriggerDelay (short channel, int triggerDelay)
{
    int addr;
    unsigned short delay;

    /* set the address */
    addr = (channel == CHANNEL_A)? TRIGDLYA: TRIGDLYB;

    /* convert trigger delay from int to unsigned short */
    delay = (unsigned short) triggerDelay;

    /* write the threshold register */
    Poke16 (addr, delay);

    return 0;
}
```