SEQUENCER MODULE

WARNING: The Sequencer module has been deprecated since ~2015.
The Tiger Programmable Logic Card can replace most of the functionality.

Overview

Old documentation that you can download in PDF form:
ASI MS2000 SEQUENCER MODULE

The SEQUENCER module consists of several hardware signal outputs, both TTL signals (TTL#) and Analog Voltage Outputs (AVO#). Incremental stage motion can be directly controlled in a sequence as well (STG#). A TTL trigger INPUT and various RS232 serial and manual control inputs are also supported. Each hardware output is configurable using a serial command. Logic connecting the inputs and outputs is implemented using a programmable BLOCK structure. BLOCKs, by themselves, do not produce output. Instead they provide the structure for building the event sequence required for a task. BLOCKs and OUTPUTs can have START, REPEAT, STEP, and RESET condition codes. They also can have DELAYS and a number of REPETITIONS associated with them, as well as completion action codes.

Conditions

Condition codes connect events with logical progressions of the BLOCKS and OUTPUTS. Not all condition codes are applicable in every situation. The table below lists the valid **CONDITION CODES** with an asterisk indicating which codes are available for the logic commands.

CONDITION Description	CODE	BLK START	BLK REPEAT	TTL START	TTL STOP	AVO, STG, or LST STEP	AVO, or STG RESET
NEVER	0	*	*	*	*	*	*
TTL_TRIGGER_RECEIVED	1	*	*	*	*	*	*
ARM_COMMAND_RCVD	2	*	*	*	*	*	*
AT_BUTTON_PRESSED	3	*	*	*	*	*	*
XYZ_STAGE_NOT_BUSY	4	*	*	*	*	*	*
BLOCK_#n_DELAY_COMPLETE	5	*	*	*	*	*	*
BLOCK_#n_COMPLETE	6	*	*	*	*	*	*
BLOCK_#n_REPEAT	7	*	*	*	*	*	*
BLOCK_#n_REPEAT_or_START	8	*	*	*	*	*	*
BLOCK_#n_DELAY_COMPLETE_or_START	9	*	*	*	*	*	*

CONDITION Description	CODE	BLK START	BLK REPEAT	TTL START	TTL STOP	AVO, STG, or LST STEP	AVO, or STG RESET
BLOCK_#n_REPEAT_or_COMPLETE	10	*	*	*	NA	NA	NA
BLOCK_#n_REPETION_#m	11	*	NA	*	NA	NA	NA
ALWAYS	12	*	*	NA	NA	NA	NA
ARRAY_MOVE_DONE ARRAY_M0DULE	13	*	*	*	*	*	*

Unused BLOCKS and OUTPUTS are set up by default with all codes set to NEVER .

The condition codes number 5 through 11 refer to the state change of a particular block. When using these codes you must also supply the block number in question, even if it is the same as the block you are defining.

Actions

Action codes connect BLOCK completion with MS-2000 controller actions.

Description	Firmware Module Used	Code	BLK END
ACTION_IDLE	none	0	*
ACTION_RING_BUFFER_NEXT_POSITION	RING_BUFFER	1	*
ACTION_INITIATE_AUTOFOCUS	AFOCUS	2	*
ACTION_ARRAY_NEXT_POSITION	ARRAY_MODULE	3	*
ACTION_ARRAY_START	ARRAY_MODULE	4	*
ACTION_SEND _WHERE	none	5	*
ACTION_SEND_TIMESTAMP*	none	6	*
ACTION_SEND_STATESTAMP*	none	7	*

*Presently not Implemented

Actions let the sequencer direct certain MS2000 controller functions. For example, RING_BUFFER or ARRAY_MODULE firmware can be used to set up a series of XY target locations. These positions can be traversed using ACTION_MOVE_NEXT_POSITION or ACTION_ARRAY_NEXT_POSITION codes at the end of a BLOCK.

BLOCKS

The BLOCK state logic structure is shown in the illustration below.

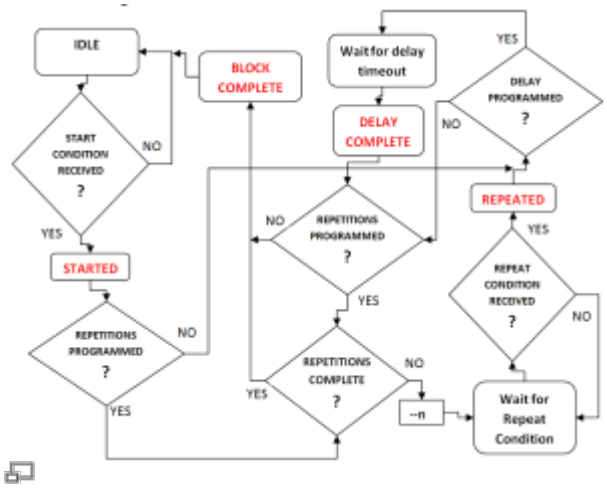

Figure 1: BLOCK logic structure.

BLOCKs are defined with the BLK command with syntax and parameter values: Both delay times and repetitions are handled with the block logic. Delays are processed and repetitions are incremented. BLOCKS start when an event matching the START condition code occurs. If repetitions are programmed, the block waits for a REPEAT condition to continue. If delays are programmed, the delays time out before the block continues to check for the next REPEAT condition. Once all repetitions are complete, the block is COMPLETE and returns to the IDLE state. If no repetitions are programmed, any programmed delay is timed out before the block COMPLETEs. In Figure 1, transient conditions that can correspond to the CONDITION CODES are shown in **RED**.

Repetitions are processed before the delays are timed out.

BLOCKS without repetitions programmed can be used as delay generators.

BLOCKS without delays programmed can be used looping counters.

BLOCKS with both delays and repetitions can be used as timed sequence generators.

BLOCK SEQUENCING

The transitions points in the BLOCK sequence progression can be used as conditions to start, repeat, stop, step, or reset other sequencer components. Blocks spend most of their time either IDLE, waiting for a REPEAT condition, or timing out a programmed DELAY.

Figure 2: Basic BLOCK sequence execution.

Transition states in one block can be used to place another block in a transition state. There is the potential that several transitions states must be quickly processed in one time step. Up to six simultaneous transition states may be processed before a recursion error is flagged.

INTERACTIVE CONTROL

The SEQUENCER is programmed via the serial RS232 or USB port on the controller using the commands described in the sections below. There are several methods to start and/or stop a programmed sequence. For interactive control, the controller @ button provides a convenient master start button. You can use the condition code AT_BUTTON_PRESSED to start a “master” block. Once the sequencer is actively running pressing the @ button will stop the sequencer and abort the running sequence. All stage motion is halted, and blocks with ALWAYS start conditions are place in the IDLE state.

Systems that include the ARRAY_MODULE for XY motion can use the @ button “long press” to start the XY array scan.

BLK (BLOCK) command syntax

Blocks are defined with BLK keyword followed immediately by the integer block number. A space delimits the block identifier, (e.g. BLK1), from a comma delimited set of parameters. Sending just the block identifier without a parameter list is the query command. The present parameter settings are returned. If spaces are used instead of integers when specifying the parameter list, the parameters corresponding to the spaces will remain unchanged. If only a partial list of parameters is provided, only those parameters will be changed.

`BLKn START, strt_blck #, rep #, REPEAT, rpt_blck #, # of reps, delay time, END`

n 1 to 6 completes the BLOCK identifier.

Comma delimited parameter list:	
START	BLK START condition code.
Start Block #	specifies the block number when START=5 to 11
Rep #	specifies the repetition number when START=11
REPEAT	BLK REPEAT condition code.
Repeat Block #	specifies the block number when REPEAT = 5 to 10
# of repetitions	any integer <64k
Delay time	in milliseconds, less than 64k milliseconds
END	BLK END action code.

BLOCKS with repeats must include both a valid BLK REPEAT condition code and a non-zero number of repetitions.

EXAMPLES:

`BLK1 2,0,0,0,1,20,0,0` Here BLOCK 1 Starts on the ARM_COMMAND and Repeats when a TTL trigger is received 20 times.

BLK2 `7,1,0,0,0,0,50,0` BLOCK2 Starts every time BLOCK 1 Repeats. There is a 50ms delay and the block runs just once.

BLK3 `3,0,0,5,1,10,35,0` BLOCK3 starts with the @ button, has 10 repetitions that repeat when the programmed 35ms delay times out.

BLK2 The query returns the current parameters for BLK2.

TTL (TTL_OUTPUT) command syntax

TTL outputs are defined with TTL keyword followed immediately by the integer output number. A space delimits the TTL output identifier, (e.g. TTL1), from a comma delimited set of parameters. Sending just the TTL output identifier without a parameter list is the query command. The present parameter settings are returned. If spaces are used instead of integers when specifying the parameter list, the parameters corresponding to the spaces will remain unchanged. If only a partial list of parameters is provided, only those parameters will be changed.

`TTLn START, start_block #, rep #, STOP, stop_block #, width, polarity`
n 1 to 5 completes the TTL output identifier.

Comma delimited parameter list:	
START	TTL START condition code.
Start Block #	specifies the block number when START=5 to 11
Rep #	specifies the repetition number when START=11
STOP	TTL STOP condition code.
Stop Block #	specifies the block number when STOP=5 to 10
Pulse Width	output pulse width in milliseconds (<64k); zero will cause the output toggle rather than time a pulse.
Polarity	1 for normally low positive pulse; -1 for normally high low-going pulse

When no STOP condition is supplied, the Pulse Width parameter determines the duration of the output pulse.

EXAMPLES:

TTL1 `6,2,0,0,0,30,1` TTL output 1 starts when block 2 completes, the pulse width is 30ms and the pulse is active HIGH.

TTL2 `11,1,5,6,1,0,-1` TTL output 2 starts on the 5th repetition of Block 1, last until Block 1 completes, and is active LOW.

AVO (ANALOG_VOLTAGE_OUT) command syntax

Analog voltage outputs are defined with AVO keyword followed immediately by the integer output number. A space delimits the AVO identifier, (e.g. AVO1), from a comma delimited set of parameters. Sending just the AVO identifier without a parameter list is the query command. The present parameter settings are returned. If spaces are used instead of integers when specifying the parameter list, the parameters corresponding to the spaces will remain unchanged. If only a partial list of parameters is provided, only those parameters will be changed.

`AVOn STEP, step block #, rep #, V0, dV, RESET, reset block #`

n 1 to 2 completes the analog output identifier

Comma delimited parameter list:	
STEP	AVO STEP condition code.
Step Block #	specifies the block number when STEP=5 to 11
Rep #	specifies the repetition number when STEP=11
RESET	AVO RESET condition code
Reset Block #	specifies the block number when RESET=5 to 10
V0	Start Waveform voltage in millivolts (<10000)
dV	Step size in millivolts (-10000 to 10000)

The AVO channels supply 0 to 10V DC and are controlled with a 12 bit DAC on the microprocessor. Hence, these channels have a resolution of about 2.5 mV. The output buffer can source current up to 25mA.

EXAMPLES:

`AVO1 7,1,0,6,1,5000,-100` Analog Output 1 steps each time Block1 repeats. Starts at 5000mV and steps - 100mV on each step. The voltage resets back to 5000mV when Block1 completes.

STG (STAGE) command syntax

Incremental stage movements are defined with STG keyword followed immediately by the integer axis index. A space delimits the STG identifier, (e.g. STG2), from a comma delimited set of parameters. Sending just the STG identifier without a parameter list is the query command. The present parameter settings are returned. If spaces are used instead of integers when specifying the parameter list, the parameters corresponding to the spaces will remain unchanged. If only a partial list of parameters is provided, only those parameters will be changed. The STG command has the same parameter list as the AVO command. This command provides direct access to the MS2000 controller for a doing Z-series, or similar stepped-move operations.

`STGn STEP, step block #, rep #, P0, dP, RESET, reset block #` n 1 to 4 The stage identifier index maps to the axes supported by the controller. Typically, the mapping is 1→X, 2→Y, 3→Z, 4→F. When the query command is issued, this axis index will be replaced by the axis name character in the returned string.

Comma delimited parameter list:	
STEP	STG STEP condition code.

Comma delimited parameter list:	
Step Block #	specifies the block number when STEP=5 to 11
Rep #	specifies the repetition number when STEP=11
RESET	STG RESET condition code
Reset Block #	specifies the block number when RESET=5 to 10
P0	Start position (units are 0.1µm)*
dP	Step size (units are 0.1µm)

Stage channels can be either motorized stage axes or the DAC-controlled piezo axis.

*If a Start Position is not specified or is 0.0, the RESET condition will move the stage back to the original position before the first step was commanded.

EXAMPLES:

STG3 7,1,0,6,1,-100,10 STG3 (Z) steps each time Block1 repeats. Starts at -10 µm and steps 1000 nm on each step. The position resets back to -10 µm when Block1 completes.

LST (LIST) command syntax

The LST function adds flexibility to the other commands. LISTS are used to provide repetition or step-specific values for AVOs, BLK delays (and STGs maybe eventually). A typical applications would be where specific analog control voltages are needed for each loop cycle rather than the uniform steps provided with the AVO command, or when variable exposure times are required for different wavelengths. LSTn STEP, Step Block #, VAR, m, value_1, value_2, value_3, …, value_m n 1 to 4 Up to four lists may be defined.

Comma delimited parameter list:	
STEP	STG STEP condition code.
Step Block #	specifies the block number when STEP=5 to 10
VAR	Variable Code – see table.
m	number of values. Must be no more than 10.
Value_1	first value,
Value_2	second value, all values have integer range ±32k
Etc…	etc… up to the maximum number of values.

The LST command will set the AVO voltage when the step happens. An AVO RESET condition will set the voltage back to the voltage defined in the AVO command and set the LIST value pointer back to the first value in the list.

LISTS can be defined for the variables defined in the table below.

VAR code	**Variable**
0	None
1	AVO1 Output Voltage

VAR code	Variable
2	AVO2 Output Voltage
3	BLK1 Delay Time
4	BLK2 Delay Time
5	BLK3 Delay Time
6	BLK4 Delay Time
7	BLK5 Delay Time
8	BLK6 Delay Time

(This table might expand as necessary.)

EXAMPLES:

`LST1 7,1,1,3,500,3000,4500` LST1 steps through the values on BLK1 repeat, these are AVO1 Output voltage values, there are three of them, 0.5V, 3V, and 4.5V

Auxiliary Commands

In addition to the commands that set up BLOCKS, OUTPUTS, and STAGE moves, there are several MS2000 controller commands that can be used with the SEQUENCER.

ARM, SAVESET, and TTL.

Command:ARM

Shortcut	ARM
Format	ARM [X] [Y=log] [Z]
Remembered	Using SS Z

Without arguments, this command establishes the **ARM_COMMAND_RCVD** event. Typically it is used to start the SEQUENCER via a single serial command.

ARM X re-initializes the SEQUENCER. This command is useful if the RESET conditions are changed or if the sequencer is hung in a non-IDLE state.

ARM Y=1 turns on or ARM Y=0 turns off the serial log file creation.

ARM Z forces a “sequence stop” condition. The sequencer is re-initialized; all blocks with ALWAYS start conditions are placed in the IDLE state, and all stage motion is halted.

End of Block Actions to control MS2000 functions

When the `RING_BUFFER` or `ARRAY_MODULE` firmware modules are included in the firmware build, then `ACTION_RING_BUFFER_NEXT_POSITION` or `ACTION_ARRAY_NEXT_POSITION` will advance

the stage to the next position defined by those firmware modules. See the Programming Manual for details of the RING BUFFER commands RBMODE and LOAD. See the Array Scanning Manual section for details involving the ARRAY MODULE functionality.

If hardware video AUTOFOCUS is included in the controller, then ACTION_INITIATE_AUTOFOCUS will command the controller do the autofocus operation. See the ASI Video Autofocus Manual for details.

Time and stage position information can be requested using ACTION_SEND_TIMESTAMP or ACTION_SEND_WHERE. Data is sent on the serial port.

Programming Tasks

There are several ways to accomplish sequencing of repetitive tasks. Each piece of hardware has its own characteristic time to complete its task, and needs to be started at the correct time with respect to other hardware components. Some devices lend themselves to being told when to do something, and then reliably perform the task in a fixed amount of time. Other devices may have a large uncertainty in the time the task requires. This can lead to several different methods of most efficiently programming a task. We will consider several examples.

Simple Tasks

Go-forever timing block:

BLK1 12,0,0,0,0,0,100,0 This block runs forever; completes and starts every 100ms. Use ARM X after you enter the block to start it running.

Continuous Pulse Generator:

TTL1 8,1,0,0,0,25,1 Use the previously defined BLK1 for the repetition rate and set the pulse width to 25ms when defining the TTL channel output.

Go-once timing block:

BLK2 3,0,0,0,0,0,100,0 This block runs once when you press the @ button and completes after 100ms.

Single Pulse Generator:

TTL2 3,0,0,0,0,25,1 Makes a 25ms pulse on TTL2 when you press the @ button.

Sequencer Timing as Master

In this example, the sequencer will command camera acquisitions, filter changes, and Z-stage moves using fixed delays. This is the simplest method of programming, and is a good place to start.

Let us assume we have a camera that requires 40ms to expose an image and read out the data, and that the camera can be triggered with an external signal. We also have a piezo-Z stage that moves quickly and we want to define a sequence that will automate acquisition of a 10-frame Z-series.

We will set up a BLOCK that will loop 10 times with a fixed delay to allow time for the camera to take an image and transfer data.

`BLK1 3,0,0,5,1,10,40,0` Start on @ button, repeat when delay complete, 10 repetitions, 40ms delay between repeats.

`STG3 5,1,0,6,1,-50,10` Step on BLK1 DELAY COMPLETE, Reset on BLK1 COMPLETE , reset position is -5µm, step is 1µm.

`TTL1 7,1,0,0,0,10,1` Send 10ms trigger pulse on BLK1 repeat for camera trigger.

These settings will generate the camera trigger just as the stage is commanded to move to a new position. With a single delay it is not possible to control both the relative position of the camera trigger and the delay between Z-moves. If the camera can be edge triggered, then the falling edge of the TTL1 signal would make a nice trigger point, and the pulse width becomes the delay between starting the move and starting the exposure. Inverting the polarity of the pulses can generate a positive edge at the correct time. If the camera requires a level trigger, then we can add a BLOCK just for this purpose.

`BLK2 7,1,0,0,0,0,15,0` A 15ms delay generator triggered by BLK1 repeat.

`TTL1 6,2,0,0,0,10,1` Send 10ms trigger pulse 15ms after BLK1 repeat.

Now let’s add a block to control a wavelength switcher. We will assume that we need to supply a TTL pulse to move the switcher to the next position, and that there are three wavelengths programmed. We will include a 150ms delay for the filter wheel.

`BLK3 3,0,0,6,1,2,150,0` Start on @ button, repeat twice when BLK1 (Z-series) completes, include 150ms delay.

`TTL2 6,1,0,0,0,10,1` Send 10ms trigger pulse when BLK1 (Z-series) completes.

`BLK1 9,3(,0,5,1,10,40,0)` Change BLK1 start condition to when BLK3 Starts or Delay is complete.

Even though BLK3 only has 2 repetitions, we get a total of three filter change pulses and Z-series because we changed the BLK1 start condition to be when BLK3 starts as well as when it repeats. When doing this interactively, only the first two parameters need to be typed in; the rest will remain unchanged. In this way, it is possible to setup the inner sequence structure and test it with the @ button trigger. When that block is working as desired, you can set up another control block with a repeat condition that can be used to start the first (inner) sequence.

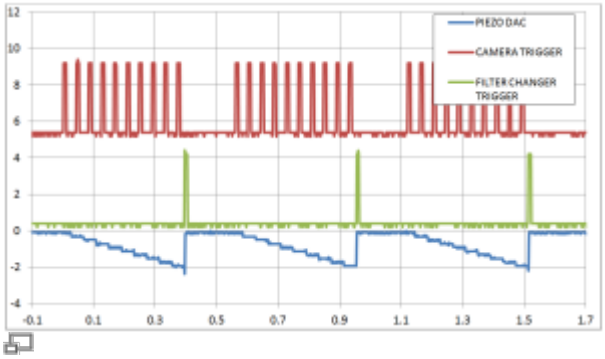

Figure 3: Oscillogram generated by the Sequencer Timing as Master example.

Camera as Master

In the previous example, the sequencer provided all of the delay intervals and timing information for all of the tasks. Now, let us assume that the camera will control the movement of the Z-stage. This might happen if the camera had some exposure control so that sometimes the camera will be taking longer exposures than at other times.

`BLK1 9,3,0,1,0,10,0,0` Start on BLK3 as before, repeat on TTL Trigger, 10 repetitions.

We need to change the stage step condition (since we no longer have the delay) to the Repeat:

`STG3 7,1` Let's change TTL1 to monitor when we get the external trigger.

`TTL1 7,1,0,0,0,10,1` Send 10ms pulse when BLK1 repeats

Perhaps we need to send a TTL enable signal to the camera to start it taking pictures. Let's add a TTL output definition to accomplish this. Without the enable pulse, the camera won't know to hold off gathering pictures when the filter wheel is turning. TTL3 8,1,0,6,1,0,1 The pulse goes high when Block 1 STARTs and goes low again when Block 1 is COMPLETE.

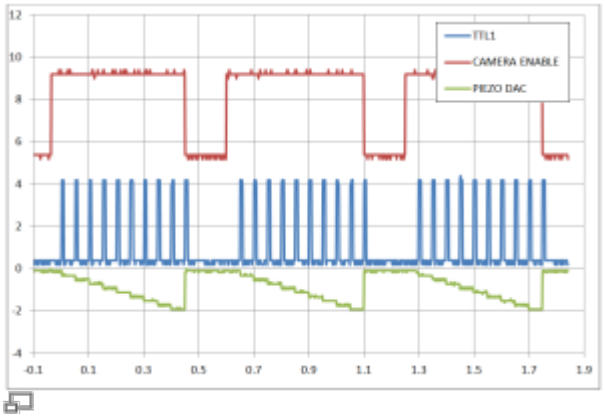

Figure 4: Oscillogram generated by Camera as Master example.

Incorporating XY Stage Positions

Simple XY moves can be programmed using the STGn sequencer command, but considerably more XY control is possible by including the ARRAY_MODULE firmware package. The following example will assume that the ARRAY_MODULE is present in the firmware. Furthermore, we will assume that the XY positions have been configured using the ARRAY commands and that all the sequencer needs to do is

to coordinate the movement to the array locations with everything else. The example is an extension of the Sequencer Timing as Master above. Colors are used in the notation below to illustrate dependency.

`BLK1 5,2,0,5,1,10,40,0` Z-axis control block for 10 steps with 40ms delay.

`BLK2 8,3,0,0,0,0,150,0` Delay generator for filter changer timing.

`BLK3 13,0,0,6,1,2,0,0` Master start/repeat block; being started when XY move is done.

`BLK4 11,3,2,6,1,1,0,3` Special Timing block to generate the move to next array position. Note the END ACTION happens same time as TTL3 pulse definition below.

`BLK5 7,1,0,0,0,0,15,0` Delay for the camera trigger.

`TTL1 6,5,0,0,0,10,1` Camera Trigger.

`TTL2 8,2,0,0,0,20,1` Filter changer trigger.

`TTL3 6,4,0,0,0,30,1` End of sequence / next position trigger.

`STG3 5,1,0,6,1,-50,10` Z-stage motion configuration.

`LST1 8,2,1,3,2000,4000,8000` Use of the LST function to generate arbitrary voltages.

`AVO1 0,0,0,6,4,5000,0` The LST statement above controls AVO1 except for the RESET voltage, set here.

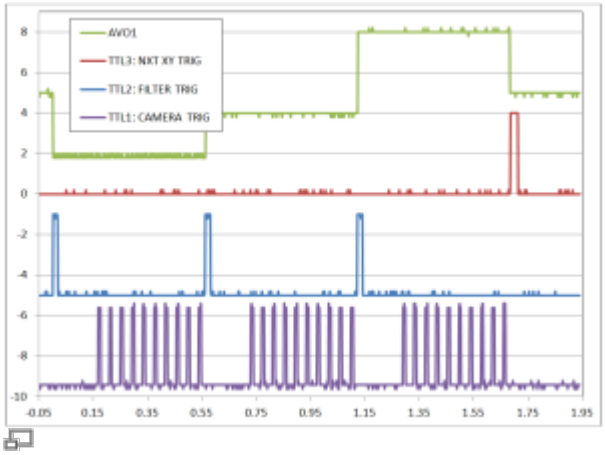

Figure 5: Timing for one position of XY ARRAY example.

To start this sequence, the master timing loop is waiting for the ARRAY_MOVE_DONE condition. This is condition is satisfied whenever an array move is complete, or once the ARRAY task starts, the stage is in position at the first XY location. The ARRAY task can be started with either the command RM X=0, or with a long-press of the @ button. The timing to request the controller to move to the next XY position is established with BLK4. The block starts on the second (last) repetition of BLK3 – roughly at the beginning of the last series of Z-moves – and completes when the Z-moves and camera triggers are done. The waveform on TTL3 shows the timing, and the END ACTION for BLK4 specifies the ACTION_ARRAY_NEXT_POSITION.

Debugging Sequence Programming

As with any flexible programming system, it is easy to generate programs that don't do what you think they should. A common problem is for a BLOCK to be waiting for a repeat or start condition that never materializes. The bottom line of the LCD display shows the states of the BLOCKS as a series of letters near the middle of the line (starting from the fifth character).

```
HRR IRDIII 00:10:45
```

The states for the BLOCKS are listed in order of block number from left to right. The letters correspond to the block state according to the list in Figure 3.

BLK and TTL State Codes	
Code	**State Description**
I	IDLE (Waiting for START Condition)
R	WAITING_FOR_REPEAT_CONDITION
D	WAITING_FOR_DELAY_TIMEOUT
s	START – transient state
r	REPEAT – transient state
c	COMPLETE – transient state
A	TTL Active
T	TTL – timing output pulse

Using this display you can determine if blocks are hung up or not performing as desired.

Sequence Log

A log of sequence events can be turned on using the serial command **ARM Y=1**. The log data is sent over the serial connection, and can be quite voluminous, so using a fast baud rate and responsive client program is helpful. The event log includes a time stamp, event name, and the present status of all of the blocks and TTL outputs. The time stamp is reset to zero when the first BLOCK is started. A partial log output is shown below for the example in the Programming Tasks section.

```
 T: 40274 EXT   TRIG    BLKS:IIIIII   TTLS:IIIII Ready
T: 40285 AT    PRESS   BLKS:IIIIII   TTLS:IIIII Ready
T:     0 BLK 1 START   BLKS:ssIIII   TTLS:sIIII Ready
T:     0 BLK 2 START   BLKS:RsIIII   TTLS:sIIII Ready
T:     0 TTL 1 START   BLKS:RRIIII   TTLS:sIIII Ready
T:   480 EXT   TRIG    BLKS:RRIIII   TTLS:AIIII Ready
T:   480 BLK 1 REPET   BLKS:rRIIII   TTLS:AsIII Ready
T:   480 TTL 2 START   BLKS:RRIIII   TTLS:AsIII Ready
T:   971 EXT   TRIG    BLKS:RRIIII   TTLS:AIIII Ready
T:   971 BLK 1 REPET   BLKS:rRIIII   TTLS:AsIII Ready
T:   971 TTL 2 START   BLKS:RRIIII   TTLS:AsIII Ready
T:  1462 EXT   TRIG    BLKS:RRIIII   TTLS:AIIII Ready
T:  1463 BLK 1 REPET   BLKS:rRIIII   TTLS:AsIII Ready
T:  1463 TTL 2 START   BLKS:RRIIII   TTLS:AsIII Ready
```

```
T:   1953 EXT   TRIG    BLKS:RRIIII   TTLS:AIIII Ready
T:   1954 BLK 1 REPET   BLKS:rRIIII   TTLS:AsIII Ready
T:   1954 TTL 2 START   BLKS:RRIIII   TTLS:AsIII Ready
```

Most external events, block START and REPEAT events, and TTL START events are given entries in the log. You will note that several events can happen at a single time step. In the sample log, when the @ button was pressed, the time stamp was reset to zero. BLOCKS 1 and 2 were started, as was the TTL1 pulse. The first external trigger pulse was received at 480ms, causing the BLK1 repeat and the start of the TTL2 output pulse, etc… Block states and the TTL output states are shown, as well as the status of the external trigger input.

WARNING: writing to the log file can take a considerable amount of processor time and will impact the timing for sequences where operations are happening faster than the log can be written. Use only for debug purposes, and increase delays appropriately as required.

Troubleshooting Techniques

- Build simple test examples.
- Set up TTL outputs that you can monitor on an oscilloscope. If you don’t have an oscilloscope, you can use test lights on the TTL channels.
- Slow everything down and use the log file.
- Substitute the @ Button Press for External Trigger conditions so you can step through your sequence.

Sequencer Timing Performance

The SEQUENCER module shares the MS2000's processor with the controller's other functions. The average task loop time with the SEQUENCER module is about 200µs, maximum time is 500µs when the controller is idle, and can reach about 1ms when the controller is busy with serial commands or axis moves. Hence, typical processing delays will be <0.5ms, but could extend to more than a millisecond occasionally. Timing delays are counted using a 1 millisecond timer interrupt, but subsequent processing of the delay triggered events can have the additional timing uncertainty of 0 – 1 ms described above. If precise absolute timing is required, be aware that these errors will accumulate, and can cause overall absolute timing uncertainty after many sequencer steps.

deprecated, sequencer, ms2000