Sequencer GUI Tool User Guide

Getting started with the Sequencer GUI tool

Introduction

The Sequencer GUI tool provides a graphical user interface for creating and editing sensor sequencer timing programs. This document describes the Sequencer tool provided by the DevSuite team**.**

Supported Devices (also installed in C:\Aptina Imaging Sequencer\seq_data)

- AF0130, AGS67M
- AR0145, AR0147
- AR0225, AR0231, AR0233-REV2, AR0233. AR0235, AR0246
- AR0323, AR0341, AR0423, AR0521, AR052XXX, AR0544
- AR0820, AR0822, AR0823, AR0830
- AR1212, AR1335, AR1337, AR1630, AR2020
- ARX383, ARX3A0
- AS0149
- ATXR11, Sirius, TC0323, VAYU, XC2D, XC3F, XG11-VAYU, XG11,
- XGS12M2, XGS12M, XGS16M, XGS45M, XGS5M
- XH11, XJ11, XK11, XP11

Installing the Sequencer Tool

The tool is included with MigMate but run Sequencer as Administrator to be able to connect it to MigMate.

For those not using MigMate, a separate installer is also available on the H: drive in the DevSuite folder.
H:\DevSuite\Sequencer
Get the file called DevSuite_SVN_*nnnnn*_sequencer.zip and run the self-installing exe file inside it.

Creating a New Program from Scratch

To create a new program click New. Select the type of sensor device, and select the first sequence to create. Depending on the type of device you may be prompted to select which signals will be used in the new sequence. You can add or remove signals at any time.

Loading an Existing Program

If you have previously saved a program with the File -> Save menu, it created a .seq file. Load it with menu File -> Open or Recent Files.

For many device types you can load an existing sequencer program from a hex file or an opcode (.i) file. Use menu File -> Import From Hex or Import From Opcode File. Importing isn't supported for every sensor type. For AR0823, AR1212 or similar with an auxiliary sequencer, you will need to load opcode files for the main sequencer and auxiliary sequencer separately. See the section below titled *Editing an Opcode Assembly File*.

Connecting to a Real Sensor Device

The tool can connect to a real sensor to upload the current sequencer program to the device, or get the sequencer program from the device. Depending on the type of device, the Sequencer tool may connect to a MigMate board through the MigMate application, or to a Demo board using the DevSuite ApBase library.

To connect through MigMate, run MigMate and start the project. In the Sequencer application select menu Connect to MigMate Camera. To connect to a Demo camera, plug in the camera, and in the Sequencer application select menu Connect to Demo Camera. You do not need to run DevWare before connecting to a Demo camera, but you may run DevWare simultaneously with Sequencer if desired.

After than you can read or write the sequencer program to the camera using the Camera menu or the buttons. Reading a sequencer program out of the device is supported on most sensor types, but not all.

Editing in the Timing Window

To insert edge transitions into the timing diagram click the right mouse button at the location. Once an edge is added, it can be dragged left or right with the mouse. To drag multiple edges together, hold down the Ctrl key and select the edges, then release the Ctrl key and drag any of the selected edges; they will all move together. Right-click on an edge to remove it or change its value.

To adjust a wait period (move all edges) use Shift + Left-Click to remove a clock cycle, or Shift + Right-Click to insert a clock cycle at the clicked location; or right-click on the clock cycle ruler.

Use the Add Sequence menu to add more sequences to the program as needed. Each sequence appears as a tab in the main window.

Use the Save button to save your work. The sequencer program will be saved into a .seq file which can be loaded back into the tool at a later time using the File Open menu.

Depending on the type of sensor device, you can create a DevWare INI file, MigMate VB, or shared settings Python file containing the commands to load the sequencer program into the sensor device. Note that the Sequencer tool cannot read these files; to save your work use the Save button.

Editing an Opcode Assembly File

For AR0233, AR0323, AR0823, Sirius, and all AR0147 class sensors, the tool can import from an opcode assembly file (.i file). Select Import from Opcode File on the File menu. Once imported, the file contents are loaded into a tab on the right-hand Code panel called Code Editor. The code can be edited, and the changes will be reflected in the sequence timing tabs when the cursor moves to a new line, either from arrow keys, or Enter. When you change the opcode file, any changes made in the timing window tabs will be lost.

To save the changes to the opcode file, use the File menu in the Code Editor tab.

Instructions with syntax errors will be highlighted in red. The full syntax error text will appear in the bottom portion of the Code Editor.

If you hover the mouse over a multi-assign instruction in Code Editor, a tooltip will show all of the signals and their values for that instruction.

If you select menu View -> Cursor then a white vertical line in the timing diagram will indicate the clock cycle corresponding to the instruction where the text cursor is. The status line at the bottom of the Code Editor also shows the clock cycle of the instruction the text cursor is on.

You can export a synthesized opcode with menu File -> Export to Opcode File. This will be a new file created from the timing diagram, not related to your .i source file.

GUI Features

Sequence Tab Window

Adjust the horizontal scale with Zoom In and Zoom Out menus, or Ctrl + Mouse Wheel.

The Time Scale menu will add time in nanoseconds to the time axis. The clock frequency can be set by a menu item, or in some cases read from the real device when connected.

The Occupied Clock Cycles feature visually indicates which clock cycles have an edge transition or instruction somewhere in the sequence.

Sequencer Code Window

The Sequencer Code window shows the sequencer program in various formats, including DevWare INI file commands, MigMate VB code, or shared settings Python code, as appropriate for the type of sensor device. If the program was imported from an opcode assembly file (.i file) then that file will be loaded here in an editor tab. The Warnings tab shows compiler error messages. The text updates automatically as the sequence edges are changed. The window is initially docked to the right-hand edge of the main window, but can be undocked.

Signals

Many of the Signal menu commands operate on the signals you have first selected in the sequence window. Select signals by clicking on the signal names. You can select multiple signals using the Ctrl or Shift keys similar to selecting files in a folder window.

Simulate Running a Sequence (AR0233, AR0823, AR1212)

The tool can simulate running the Init sequences, or any Main sequence, showing complete timing after expanding sub-sequences in-line in a new tab. For AR1212, the window is split, and the top half shows the primary sequencer and the bottom half shows the auxiliary sequencer.

Use menu Sequence Runs -> Add Run to create a simulated sequence run tab. Use menu Sequence Runs -> Sensor Operating Mode to set the sensor register values to be simulated. There is one Sensor Operating Mode for each Run tab, and the tool supports one Run tab for each for the Init sequence and each Main sequence. The signals on a Run tab cannot be edited. Any changes made to other sequence tabs will be immediately reflected in the run.

The Run tabs and their corresponding sensor operating mode parameters are also saved in the .seq file when the sequence program is saved.

Comparing Timing Sequences

The compare features lets you compare the timings you are editing to a previously saved file. Save the reference timings in a .seq file using the File -> Save menu or toolbar button. To compare the active timings to the previously saved file, use menu Compare -> Open Sequencer File to Compare With, and select the file. Or select from recently used files.

The names of signals with a difference will be highlighted in pink, and the locations in the timing diagram where differences occur will be marked with red ticks. You can continue to edit the current timings. The reference file is not editable.

Sequencer GUI Files

For each supported device there is a .sdf (sequencer definition) file in the seq_data directory that comes with the Sequencer GUI application. The .sdf file lists all of the sequencer inputs and outputs used by the device. When

you start a new sequencer program, the application reads the sdf file for the device to populate the GUI. These files are normally provided by the DevSuite team with the application, but you can create your own in a text editor.

When you have created a sequencer program and save it, the Sequencer GUI creates a .seq (sequencer program) file. This file contains all of the information from the original sdf file, plus your sequencer timing data. To resume working on a sequencer program you just need to open the .seq file using the File Open menu.

The .sdf and .seq files are XML files, and are documented below.

Feature Requests and Reporting Bugs

Use the same JIRA system as DevSuite for creating issues. For Component, select Sequencer.

https://jira.onsemi.com/secure/CreateIssue!default.jspa?pid=15601

Appendix A: SDF File Format

On occasion the user may need to modify a sequencer definition file. The Sequencer application does not have facilities for creating or editing these files, so the file format is described here. The sdf files are XML files that can be edited in any text editor. The file looks something like this:

```
<?xml version="1.0"?>
<program sensor="AR0123" clockmhz="100" class="AR0123" aka="C11A AR0123AT" ramsize="512">
  <resource type="signal" number="0">row_reset</resource>
  <resource type="signal" number="1">row_select</resource>
  … and so on for all of the signals …
  <group>ref_ctrl<resources>ref_ctrl_0 ref_ctrl_1</resources>
    <enum value="0" name="2.0 to 3.3V (Shutter)" short="Sh"></enum>
    … and so on for all of the values to be named …
  </group>
  … and so on for all of the signal groups …
  <sequencename type="main" number="0">Main Linear Mode</sequencename>
  … and so on for all of the sequence names …
</program>
```

That is what is in common for most devices. There are more possible tags and attributes depending on the sensor type. See below.

Program Tag

The XML root element is a **program** tag. The **sensor** attribute will appear on the UI as the sensor name, and can be any user-chosen value. The **class** attribute defines which sequencer compiler to use, and it must one of the values given in the table below.

Sensor	Class
XC2D, XC3F, XG11, XH11, XJ11, XK11, XP11	XC2D
VAYU, XG11-VAYU	VAYU
AR0233, AR0246, AR0323, TC0232, AR0423, AR0820, AR0822, Sirius	AR0233
AR1212	AR1212
AR0147, AS0149, AR0231, AR0225, AR052XXX	AR0147
ARX3A0, AR0521, AR1335, AR1337, AR1630	AR1335
ARX383, AR0145, AR0235	ARX383
AR0544, AR0830, AR2020	AR0830
AR0341, AR0823	AR0823
AF0130	AF0130
XGS5M, XGS11M, XGS16M, XGS45M, AGS67M	AGS

The **aka** attribute is a space-separated list of MigMate FabIDs and part numbers as they appear in Demo kit xsdat files. The **aka** attribute is used to select the correct sdf file when connecting to MigMate or a Demo kit.

If there are different .sdf files for different versions of the same sensor, then use the **rev** attribute in the program tag to associate the file to a particular sensor revision. For example rev="1" if the .sdf file is only for rev 1. If the file is used by more than one sensor version, list all versions; rev="2 3".

The attribute **ramsize** is used by most classes to indicate the amount of RAM, in bytes, allocated to sequencer instructions. For class XC2D and VAYU **ramsize** is the maximum number of timing instances for one sequence. The **clockmhz** attribute is optional and used only for displaying the Time Scale.

The AR0233, AR0823 and AR1212 classes of devices support the special attributes **usesetclear** and **usewaitfields**. **usesetclear**="false" suppresses the generation of SET and CLEAR opcodes. **usewaitfields**="false" suppresses the use of the wait fields in opcodes having them, and all waits will be done with WAIT opcodes.

The AR0823 and AR1212 class supports the attribute **auxramsize** to specify the amount of RAM, in bytes, for the auxiliary sequencer.

The XC2D class of devices supports the special attributes **bitwidth**, **maxrows** and **maxglobalrows**. The **bitwidth** attribute specifies how many signals are available, and affects how many registers are programmed in the output scripts. Normally it would be "32", "48", "64", or “80”. The **maxrows** and **maxglobalrows** attributes specify how many rolling shutter and global shutter sequence memories are available. **Maxrows** is normally "4", and **maxglobalrows** is normally "0" or "1".

The ARX383 class supports the special attributes **maxwait** and **rowtime**. The **maxwait** attribute limits the maximum number of clock cycles to use in a single wait instruction. This is to cover a bug in first version of the sequencer where waits longer than 256 cycles don’t work correctly, even though the instruction was supposed to support up to 4096 cycles. The **rowtime** attribute specifies the sensor row time in microseconds. This is used for the time scale on the Global timing tab.

Seqblock Tags (AGS, XGS)

For AGS and XGS series sensors, the **program** tag contains three or four **seqblock** tags describing each sequencer block. For all other sensor types, the **seqblock** tag is not used.

The **seqblock** tag has the attributes **name**, **ramaddr**, **ramsize**, **bitwidth**, **bytewidth**, **step**, and **regstep**. Inside the **seqblock** tag there are some number of **state** tags, and a **regbases** tag. The **state** tag has the attributes **type**, **state**, **regaddr**, and optionally **bitwidth**. Inside the **state** tag is a name. The **regbases** tag contains a list of register addresses for the base address register of each of the sequences supported by the block. Example:

```
<seqblock name="FSM" ramaddr="0x4000" ramsize="0x600" bitwidth="48" bytewidth="6"
          step="0 5" regstep="0x381E">
  <state type="default" state="0x0000000070CAEE23" regaddr="0x3834">Default</state>
  <state type="freeze" state="0x000007FFFFFFFFFF" regaddr="0x383A"
         seqnumbers="0 5 6 7 8">SFOT Freeze</state>
  <state type="freeze" state="0x000007FFFFFFFFFF" regaddr="0x3840"
         seqnumbers="1 2 3 4">EFOT Freeze</state>
  <state type="freeze" state="0x000007FFFFFFFFFF" regaddr="0x3846"
         seqnumbers="9">DSFOT Freeze</state>
  <state type="freeze" state="0x000007FFFFFFFFFF" regaddr="0x384C"
         seqnumbers="10">TSFOT Freeze</state>
  <state type="resolve" state="0x0000000008400000" regaddr="0x3824"
         bitwidth="28">Resolve</state>
  <regbases>0x3852 0x3854 0x3856 0x3858 0x385A 0x385C 0x385E 0x3860
            0x3862 0x3864 0x3866</regbases>
</seqblock>
```

The name of a block must be either “FSM”, “LSM”, “CLSM”, or “ALSM” as these names have some special meaning in the tool. The **ramsize** is given in bytes. The **bytewidth** attribute is optional, and is calculated from the bitwidth if absent. The **state** tags hold the default, freeze and resolve bits. The state type attribute must be either “default”, “freeze”, or “resolve”. The resolve state can have a smaller bitwidth than the block’s instruction width when not all bits participate in merging.

Resources Tags

The **program** tag contains one or more **resource** tags describing each programmable signal or special instruction of the sequencer. Each **resource** tag creates a row on the timing diagram window where an edge or instruction can be inserted with the mouse. For each one there is at least a resource type, a number, and the name in the following format:

<resource type="*resource_type*" number="*resource_number*">*signal_name*</resource>

The accepted values for *resource_type* and *resource_number* depend on the device class.

Class	Resource Type	Resource Number	Optional Attributes (comments)
XC2D, VAYU	“signal”	0 … 79	latch, latchsetup, latchwidth
AR0233	“analogsignal”	0 … 143	
	“digitalsignal”	0 … 31	
	“pulsesignal”	0 … 95	
	“pause”	0 … 7	start, delay
	“compare”	0 … 31	
	“wait”	1 … 4	(4 is conditional wait)
AR0823	“analogsignal”	0 … 287	
	“auxanalogsignal”	0 … 287	
	“digitalsignal”	0 … 103	
	“auxdigitalsignal”	0 … 103	
	“pulsesignal”	0 … 239	
	“pseudostart”	0 … 15	
	“pseudopause”	0	
	“pseudofinish”	0	
	“auxjump”	0 ,… 15	
AR1212	“analogsignal”	0 … 143	
	“auxanalogsignal”	0 … 143	
	“digitalsignal”	0 … 31	
	“pulsesignal”	0 … 95	
	“pause”	0 … 7	start, delay
	“compare”	0 … 31	
	“wait”	1 … 4	(4 is conditional wait)
AR0147	“signal”	0 … 47	
	“opcode”	0 … 255	
	“opcodeoperand”	0 … 255	
	“wait”	1 … 4	
AR1335	“signal”	0 … 51	
	“opcode”	0x71 … 0x73	
	“pause”	0 … 3	
AR0830	“signal”	0 … 99	
	“opcode”	0xE1 … 0xE3	
	“pause”	0 … 3	
ARX383	“signal”	0 … 99	
	“opcode”	0xE1 … 0xE3	
	“pause”	0 … 3	
	“pulseregister”	0x0000 … 0xFFFE	(number is the register

Class	Resource Type	Resource Number	Optional Attributes (comments)
			address)
AF0130	“signal”	0 … 95	for Sequencer 1
	“auxsignal”	0 … 95	for Sequencer 2
	“pause”	0 … 3	
AGS	“signal”	0 … 63	seqblock (required)

The “opcode” resource type declares an instruction opcode that can be inserted into the program timeline on the GUI, but otherwise has no meaning for the GUI. Likewise “opcodeoperand” declares an instruction opcode that needs a parameter. The opcode itself goes in the resource **number** attribue. The “wait” resource type is for special-purpose wait delay instructions.

The “pulseregister” resource type declares a register that holds the rising edge time and the falling edge time for a single pulse on a signal. The register address goes in the **number** attribute. This is used by the ARX383 class on the Global timing tab.

For class XC2D the Sequencer supports latched signals, where there is an enable signal and a latch signal that control a single final output signal. In this case use the number of the enable signal is in the **number** attribute, and use the number of the latch signal in the **latch** attribute. (Except that the latch attribute is ignored for global shutter and global readout sequences.) For example, if row_enable is signal 6 and row_latch is signal 0, use:

```
<resource type="signal" number="6" latch="0">row</resource>
```

Defined this way only the “row” signal appears on the GUI, and the application will automatically set the enable signal and pulse the latch signal as needed. The enable setup time and latch pulse width default to one clock cycle each, but can be set to other values for a signal using the **latchsetup** and **latchwidth** attributes. For example:

```
<resource type="signal" number="6" latch="0" latchsetup="3" latchwidth="2">row</resource>
```

For class AGS, each signal needs a seqblock attribute naming the sequencer block of the signal.

```
<resource type="signal" seqblock="fsm" number="0">fsm_seq_rs_cp_lo_clk_en</resource>
```

For classes supporting Pause instructions, the wait time for a Pause instruction depends on other signals. To show an accurate wait time on the GUI, use the **start** and **delay** attributes with “pause” declarations. The GUI will show the actual wait time for the Pause instruction as a bar on the time line. In the start attribute put the signal name that starts the timer that the Pause will wait for. Put the delay until the pause will finish in the **delay** attribute. For example:

```
<resource type="pause" number="1" start="adc_movm_all1 adc_movm_all2"
delay="76">wait_move_all_1_2</resource>
```

If there is more than one possible start signal, list all names separated by spaces. The Pause wait time will be calculated starting from the rising edge of the start signal most recently preceding the Pause instruction in the same sequence. If the wait delay can’t be calculated then the Pause instruction will occupy one clock cycle on the GUI, and a vertical zig-zag line will be drawn to indicate an unknown number of missing cycles.

Output signals can be declared as low-true with the **inverted** attribute. For example:

```
<resource type="analogsignal" number="32" inverted="true">rampbuf_sh</resource>
```

The color of the trace on the GUI for a signal can be set with the **color** attribute. The format is the same as the 6-digit HTML color codes—a “#” sign and six hex digits representing red-green-blue. For example:

```
<resource type="signal" number="16" color="#44CCFF">clamp</resource>
```

Plural Signal Assignments (AF0130)

The AF0130 can assign more than one number per named signal. Further, some assignments are hardcoded, and other assignments can be programmed through registers. In the **resource** tag, list the hardcoded numbers for a signal in the **number** attribute. For example:

```
<resource type="signal" number="3 26 36">sel0</resource>
```

For each signal, also give the corresponding register address that controls the register-programmed number assignments for that signal with the **register** attribute. For example:

```
<resource type="signal" number="0" register="0x3B40">rst1</resource>
```

Optionally, any initial register-assigned numbers can be defined with an **assign** attribute. For example:

```
<resource type="signal" number="61" register="0x3B7A" assign="84">bstr_boost_vtxhi</resource>
```

Multiple numbers may be listed in the **assign** attribute. The tool will calculate the register value automatically. It's possible that a signal may have no hardcoded number, only register-programmed numbers. In that case use an empty **number** attribute with an **assign** attribute with a number assignment. The tool requires that all signals have at least one number assigned at all times. For example:

```
<resource type="signal" number="" assign="45" register="0x3BBC">cds_add_trg</resource>
```

Each number can only be associated with one signal at a time.

Each signal number that's available for register-based reassignment should be listed with an **assignableresource** tag. The tag needs **type**, **number**, **mask** and **value** attributes to define how the register value for a signal number assignment shall be calculated. For example:

```
<assignableresource type="signal" number="10" mask="0x0003" value="0x0001">
</assignableresource>
```

Group Tags

The **group** tag defines a set of signals that form a single multi-bit value. Group tags are optional. The group appears on the GUI on a single line. For example three address signals that form a single 3-bit address value:

```
  <group>ADDR<resources>addr0 addr1 addr2</resources></group>
```

The signals in the group are listed in a **resources** tag within the **group** tag. The signals are in order from LSB to MSB. The signals can be indicated by name or by reference number. Signals referenced by number don't need to have **resource** tags. Reference numbers for type "signal" are D0, D1, etc.; for type "analogsignal" are Da0, Da1, etc.; for type "digitalsignal" are Dd0, Dd1, etc.; and for type "auxanalogsignal" are AuxDa0, AuxDa1, etc.

You can specify unused bit positions with a 0 (zero) in the signal list. This only affects the value shown on the GUI, it doesn't affect the sequencer code generation.

A **group** tag can optionally have any number of **enum** tags. The **enum** tags give text names to group values. When the value of a group corresponds to one of the **enum** tags, the GUI shows the name instead of the numeric value. The optional **short** attribute defines a second, shorter name that will be used by the GUI when the long name doesn't fit in the diagram.

```
  <group>seq_clg_ref<resources>seq_clg_ref_0 seq_clg_ref_1</resources>
    <enum value="0" name="Shutter" short="Sh"></enum>
    <enum value="1" name="Integrating" short="Int"></enum>
    <enum value="2" name="Read" short="Rd"></enum>
  </group>
```

Sequencename Tags

The **sequencename** tag is optional, and provides default names for sequences to show on the GUI. (The sequence names can be changed by the user with the Sequence Names dialog.) The sequencename tag can have **type** and **number** attributes like described for sequence tags below. The name text goes inside the tag. For example:

```
<sequencename type="main" number="0">Main Linear</sequencename>
```

Note that changing the sdf file will not affect any existing seq files (saved sequencer programs). The seq files have the same format as the sdf files, but also include the sequence timing data.

Appendix B: SEQ File Format

Sequence programs created by the tool are saved in .seq files. The format is documented here for interoperability purposes. Hand-editing seq files is possible, but not recommended. The seq file format is a superset of the sdf file. The seq file contains all of the information from the device's sdf file, plus the sequence programs, and possibly some more **resource** tags.

The XML root element is a **program** tag. The program tag contains one or more **resource** tags, and one or more **sequence** tags representing each sequence. Each sequence tag contains zero or more **trace** tags representing each signal or type of instruction used by the sequence. Each trace tag contains zero or more **edge** tags representing each signal state transition or other special instruction in the program. In summary, the hierarchy is as follows:

<program>
    <seqblock>
    <resource>
    <group>
        <resources>
    <sequencename>
    <sequence>
        <trace>
            <edge>
    <run>

The seq file contains all of the **seqblock**, **resource, group** and **sequencename** tags from the sdf file.

Depending on the type of device, the seq file may also contain label, wait, jump and terminate resource tags. These create lines on the GUI for inserting the corresponding instructions into the program. You can also put these in the sdf file if you want to give them descriptive names for the GUI, but they are optional in the sdf file. Example:

```
<resource type="jump" number="0">Jump0</resource>
```

Class	Resource Type	Resource Number
AR0233	"jump"	0 … 31
	"wait"	1 … 4
	"terminate"	n/a
AR0823, AR1212	"jump"	0 … 31
	"auxjump"	0 … 31
	"wait"	1 … 4
	"terminate"	n/a

AR0147	“label”	0 … 15
	“jumplabel”	0 … 15
	“terminate”	n/a
AR1335, ARX383, AR0830	“jumpaddr”	0 … 1
	“terminate”	n/a

For each sequence there is a **sequence** tag. The sequence tag occurs only directly inside the program tag. The sequence tag has **type** and **number** attributes. Example:

```
<sequence type="main" number="0">  …  </sequence>
```

The allowed values for type and number depend on the sensor device class.

Class	Sequence Type	Sequence Number
XC2D	“shutter”	0 … 3
	“readout”	0 … 3
	“globalshutter”	0
	“globalreadout”	0
AR0233	“init”	0 … 15
	“main”	0 … 7
	“row”	0 … 31
AR0823	“init”	0 … 15
	“main”	0 … 7
	“row”	0 … 31
	“auxinit”	0 … 7
	“auxmain”	0 … 15
	“auxrow”	0 … 31
	“sfot”	0 … 15
	“efot”	0 … 15
AR1212	“init”	0 … 15
	“main”	0 … 7
	“row”	0 … 31
	“auxinit”	0 … 7
	“auxmain”	0 … 15
	“auxrow”	0 … 31
AR0147	“main”	0
AR1335, AR0830	“main”	0 … 7
ARX383	“main”	0 … 7
	“sfot”	0
AF0130	“main”	0 … 15
	“auxmain”	0 … 15

Inside each **sequence** tag there are zero or more **trace** tags. Each trace tag corresponds to a line on the GUI where the user can insert edge transitions or other special instructions. Inside the trace tag is the name of a resource and any corresponding edges. The trace tag has no attributes.

Inside each **trace** tag are zero or more **edge** tags corresponding to instructions in the sequence program that operate on the signals or generate special instructions. An edge tag has a **type** attribute and an optional **value** attribute. Inside the trace tag is the clock cycle relative to the beginning of the sequence where the signal change or instruction is to occur. Example of a trace tag with two edge tags:

```
<trace>adc_az1b<edge type="set">85</edge><edge type="clear">482</edge></trace>
```

The allowed values for the edge type depend on the resource type of the trace, and further on the instruction set of the sequencer.

Resource Type	Edge Type	Edge Value
“signal”, “auxsignal”, “analogsignal” , “auxanalogsignal”, “digitalsignal”, “pulseregister”	“set”	n/a
	“clear”	n/a
	“toggle”	n/a
“pulsesignal”	“pulse”	n/a
	“condpulse”	n/a
“pause”	“pause”	n/a
“wait”	“wait”	Clock cycles
“compare”	“compare”	Compare value
“label”	“label”	Label number
“jump”, “auxjump”	“jump”	Row sequence number
	“condjump”	Row sequence number
“jumplabel”	“jump”	Label number
“jumpaddr”	“jump”	Clock cycle
“opcode”	“opcode”	n/a
“opcodeoperand”	“opcode”	Operand value
“terminate”	“terminate”	n/a

For AR0233 or AR1212 there may be **run** tags. These correspond to main sequence simulation run tabs created by the Add Run menu. The run tag has a **sequence** attribute indicating which sequence it is for. AR1212 will also have an **auxtype** attribute to indicate which type of sequence to run on the auxiliary sequencer. Auxtype can be “auxmain”, “sfot” or “efot”. Inside the run are the name of the run, and tags for each of the variables. For example:

```
<run sequence="Main0">Run Main0<operation_mode>1</operation_mode>
  <t1_ofl_en>0</t1_ofl_en>
  … and so on for the rest of the variables …
</run>
```

The variable names and allowed values are as follows:

Class	Variable	Allowed Values
AR0233	operation_mode	0 … 3
	t1_ofl_en	0 … 1
	t2_ofl_en	0 … 1
	lfm_linear_mode_seq_en	0 … 1
	lfm_hdr_mode_seq_en	0 … 1
	run6_seq_en	0 … 1
	run7_seq_en	0 … 1
	num_exp_max	0 … 3
	lfm_mode	0 … 1
	lfm_fim_mode	0 … 1
	t1_e1_e2_e3_sel	0 … 3
	t2_e1_e2_e3_sel	0 … 3
	t1_depth_sense_en	0 … 1

	ocl_t1_donut	0 … 1
	lfm_2b_sel	0 … 1
	col_gain_t1	0 … 7
	col_gain_t2	0 … 7
	col_gain_t3	0 … 7
	col_gain_t4	0 … 7
	col_gain_t1_e1	0 … 7
	col_gain_t1_e2	0 … 7
	col_gain_t1_e3	0 … 7
	ana_coarse_gain_thresh	0 … 7
	wait_delay_exp	0 … 3
	seq_lp_mode	0 … 1
	line_length_pck	0 … 65535
AR1212	t1_ofl_en	0 … 1
	t2_ofl_en	0 … 1
	non_pipeline_mode	0 … 1
	run6_seq_en	0 … 1
	run7_seq_en	0 … 1
	num_exp_max	0 … 3
	t1_e1_e2_e3_sel	0 … 3
	t2_e1_e2_e3_sel	0 … 3
	sfot_dummy_seq_switch_en	0 … 1
	efot_dummy_seq_switch_en	0 … 1
	local_sg_fd_reset_en	0 … 1
	local_sg_fd_reset_always_en	0 … 1
	global_sg_fd_reset_en	0 … 1
	global_sg_fd_reset_always_en	0 … 1
	final_partial_tx_en	0 … 1
	sh1_aux_seq_en	0 … 1
	sh1_delay	0 … 32767
	sh2_aux_seq_en	0 … 1
	sh2_delay	0 … 32767
	nr_sfot_rows	0 ... 16
	nr_efot_rows	0 … 16
	integ_time	0 … 65535
	sg_fd_reset_row_count	0 …
	seq_lp_mode	0 … 1
	line_length_pck	0 … 65535
	slpck	0 … 65535