Qualcomm Technologies International, Ltd.

Qualcomm Kalimba DSP Simulator

User Guide

80-CT416-1 Rev. AK

October 23, 2017

Revision history

Revision	Date	Description
1	FEB 2010	Initial release. Alternative document number CS-00127831-UG.
2	MAR 2010	Editorial updates.
3	MAR 2010	Editorial updates.
4	MAR 2011	Editorial updates.
5	JUL 2011	Removed references to `kalsim_args.txt` from the Port Scripts example. Updated to latest CSR™ style.
6	JAN 2012	Updated to latest CSR style.
7	APR 2014	Updated to latest CSR style.
8	APR 2016	Editorial updates and updates to reflect Kalsim External/2016a
9	SEP 2016	Updated to conform to QTI standards; no technical content was changed in this document revision.
AK	OCT 2017	Added to the Content Management System. DRN updated to use Agile number. No technical content was changed in this document revision.

Contents

Tables

1 Kalsim - overview

Kalsim is an instruction set simulator that runs on a host PC to simulate the operation of a Kalimba DSP.

The key features of Kalsim include:

- Complete bit-exact simulation of the Kalimba instruction set.
- Debugging interface operation is the same as real hardware, giving complete interoperability with standard Kalimba debugging tools (xIDE, Matlab, Python).
- Kalimba peripheral simulation including MMU ports, flash file system, interrupt controller, and timers.
- High performance host to Kalimba I/O.
- Reproducibility of execution. This includes run-to-run identical timing of interrupts and streams, even while debugging tools are attached.
- Detection of unmapped memory accesses.
- Support for CSR8670 (Kalimba architecture 3) and CSR8675 (Kalimba architecture 5).
- Instruction tracing.

1.1 Kalsim variants

Kalsim currently supports two variants of the Kalimba architecture present within the Qualcomm® BlueCore™ range of chips. Kalsim only accepts executable `.kap` files for the correct simulator variant and rejects other ones.

Architecture	Supported IC	Kalsim Variant
3	CSR8670	`kalsim_csr8670`
5	CSR8675	`kalsim_csr8675`

Kalsim comes in an optimized variant for each supported chip variant. However, user options are identical between the different supported BlueCore variants.

1.2 What does Kalsim simulate

The simulated Kalimba DSP Core is closely modeled on the real hardware. It decodes the instructions, and performs the intended operations with bit-exact accuracy and similar timing as the real hardware.

DSP memory

The simulated Kalimba DSP has the same quantities and bit widths of RAM as the real hardware. DM memory is modeled as a 24-bit memory. Memory-mapped peripherals are at the same location as in real hardware. The PM memory is modeled as a 32-bit memory, which is word addressable on architecture 3 and byte addressable on architecture 5.

Timers

The timers are simulated with an accuracy of a simulated clock cycle. This may however not result in timer-triggered events occurring at exactly the correct cycle, see Interrupt controller and timer timing.

Interrupt controller

The simulated interrupt controller models the real hardware. Interrupt priorities and nested interrupts are supported.

Debug interface

Kalsim has a debugging interface that models the SPI interface. This allows applications to be debugged in a fashion that is as close as possible to real hardware.

MMU port streams

The MMU Interface on real hardware consists of a set of virtual read and write ports that can stream data to other parts of the chip. Kalsim can connect the simulated DSP to files as input and output streams. This allows applications such as an MP3 decoder to be simulated, with an input MP3 file on the host system being streamed in one port and the resulting stereo sound output to two files as raw PCM samples.

Port scripts can be used to specify the rate of data flow into and out of the simulated MMU buffers. Different sources and sinks have different data flow characteristics. For example, the DAC and ADC ports give a constant flow of data at the audio sample rate, in small chunks of just a few 16-bit samples at a time. In contrast, packets to be sent over a Bluetooth connection may vary in flow rate and occur in variable-sized chunks that are large in comparison to those from the ADC/DAC.

1.3 Kalsim stall cycle modeling

Certain combinations of instructions can cause a Kalimba DSP processor to stall for one cycle or more, to fulfill its dataflow scheduling requirements. Kalsim can model these stall cycles to a high level of detail. Some interactions in the hardware are challenging to simulate without degrading the simulation speed, so it is impractical to simulate them. An example would be the stalls introduced in the CSR8675 around branching. To maximize execution speed, stall cycle modeling is not enabled by default. However it can be enabled using the command line options described in Kalsim command line parameters.

1.4 Differences between Kalsim and real hardware

The following differences exist between Kalsim and the real hardware:

- Memory mapped register timing
- MMU behavior

- Interrupt controller and timer timing
- Memory mapped register operation
- VM behavior

Memory mapped register timing

Memory mapped registers take effect immediately in Kalsim, unlike in the real DSP hardware where it may take a few cycles for certain registers to take effect. This should have no noticeable impact on most applications.

MMU behavior

In real hardware, the transfer rate and typical block size vary depending upon the application and the destination of the data. For example, the MMU in real hardware consumes port output audio data at a constant rate, while port data destined for other purposes (for example, Bluetooth transfers) is transferred in blocks at regular intervals.

By default, Kalsim processes all MMU buffers every 6400 simulated clock cycles, copying in/out as much data as possible. This has been found to be a reasonable simulation of average MMU behavior.

Some applications need finer control, which can be achieved using port scripts. For example, some audio applications use the MMU buffer fill amount to determine whether an audio file has completed decoding. To support these applications, use a port script, see Chapter .

Interrupt controller and timer timing

There is a difference in the rate with which interrupt sources are monitored. This means that interrupts occur at slightly different timings compared to using real hardware. The interrupt controller (and the rest of the firmware) is simulated every X DSP cycles, where X is 6400 by default and can be changed by the command line options, see Kalsim command line parameters. This default period is equivalent to an 80 µs interrupt resolution on an 80 MIPS Kalimba.

Unsupported memory mapped registers

Kalsim simulates a subset of the memory mapped registers that exist on the Kalimba DSP. Writing to and reading from memory mapped registers that have not been simulated (for example, PIOs) results in a warning.

VM behavior

Kalsim does not simulate the VM present on BlueCore devices. Instead, it simulates some common VM behaviors such as sending the `MESSAGE_GO` message upon receipt of the `MESSAGE_KALIMBA_READY` message, and sending messages when DSP ports are connected.

1.5 Extra features provided by Kalsim

A number of extra features are available in Kalsim that do not exist in real hardware. These are largely to support debugging and logging, and include:

- Unmapped memory access warnings
- Instruction tracing

Some DSP algorithms may read from unmapped memory locations, or write to unmapped memory. Such bugs can be difficult to trace on real hardware. When no debugger is attached Kalsim shows a warning when an application attempts to write to or read from unmapped memory, so that the fault in the DSP algorithm can be detected. With the debugger attached, Kalsim breaks at a location of the unmapped memory read.

Kalsim can also be used to produce a trace of the program counter during execution. This can be useful to profile the frequently used parts of a Kalimba application or to get a better understanding of what is causing a bug.

2 Running Kalsim

To run Kalsim on the command line, type:

```
kalsim_{chip} [option… ] executable
```

The simulator accepts `.kap` files like an actual Kalimba DSP on a BlueCore chip. Each variant of Kalsim is specifically tailored for a given chip and uses a separate executable. Kalsim version-checks the `.kap` file to ensure it matches the architecture and the chip. For example:

```
kalsim_csr8670 --infile 0 sin16.in my_first_dsp_app_kalimba.kap
```

Most arguments start with two dash characters followed by switch name and optional parameters. Some commonly used arguments feature only a single dash followed by a letter as a shortcut.

To list the all the options present in Kalsim, type:

```
kalsim_{chip} --help
```

2.1 Kalsim command line parameters

Table 2-1 describes the command-line parameters. In Table 2-1, square brackets, `[ ]`, are used for optional arguments and angled brackets, `< >`, are used for mandatory arguments.

Table 2-1 Command line parameters

Option	Description
`--clock-speed [X]`	Run Kalsim at a different clock speed than the actual chip. Clock speed is given in MHz.
`-d, --debug`	Wait for a debugger to attach before running.
`--dmset <addr> <val>`	Sets a DM location after download (and after bootloader completes.) Decimal and Hex (`0x123`) values are supported.
`-e, --exit-on-stack-fault`	Prints a message and exits if stack underflow or overflow.
`-f, --logfile [file base name]`	Outputs statistics to a logfile (default: `logfile.txt`). This option is used to enable stall-cycle modeling. (2)
`--statistics`	Enable stall cycle model and generate more detailed cycle information(2)
`-h, --help`	Print a description of command-line parameters.
`-i, --infile <port> [S] <filename> [s=port_script] [r=rate]`	File to feed into a Kalimba MMU read port. See below for a description of the `S, rate, and port_script` options.
`--ignore-errors`	If there is an error, ignore it and continue execution.
`-m, --memcheck`	Terminate simulator with error if bad memory read/write detected.

Table 2-1 Command line parameters (cont.)

Option	Description
`-n, --non-interactive`	Disable exiting on a '**q**' keypress (for invoking Kalsim inside a script).
`-o, --outfile <port> [S] <filename> [s=port_script] [r=rate]`	File to write to from a Kalimba MMU write port. See (1) below for a description of the `S, rate, and port_script` options.
`--pmset <addr> <val>`	Sets a PM location after download (and after bootloader completes). Decimal and Hex (`0x123`) values are supported.
`-q, --quiet`	Do not print banner and progress information.
`--raw-instruction-trace [file]`	Produces an uncompressed program counter trace in the file specified.
`-s, --streamtimeout [X]`	Terminate simulator when no stream writes have occurred for `X` million cycles. `X` defaults to 32 (0.5 sec of virtual `KALIMBA_ARCH 2` time).
`-t, --terminate-at-pc <X>`	Terminate simulator when PC equals `X` for an extended period.
`--timer-update-period <X>`	Number of Kalimba cycles between timer updates (that is, checking for interrupts). 10 <= `X` <= 10000. Defaults to 6400
`--version`	Print chip target and memory map information.

NOTE (1) Input and output port specifiers have an optional `S` parameter. This selects a software based stream, rather than the default hardware based stream. In general, use software streams for everything other than PCM audio. So, for example a stereo MP3 decoder would use `S` on the input port but not on the two output ports.

The optional `s=<port_script>` parameter assigns a port script to the port, see Chapter . The optional `r=<bits/s>` arguments is used to describe hardware stream streaming at a fixed rate, like an ADC or a DAC.

(2) Stall cycle information by Kalsim is not guaranteed to be accurate.

2.2 Kalsim errors, warnings, and announcements

Kalsim can generate errors and warnings while executing. Errors cause Kalsim to terminate while warnings are just raised and the simulation continues. When the debugger is attached, errors and unmapped memory warnings cause Kalsim to break at the point where this error or warning was raised. Errors do not cause Kalsim to terminate when the debugger is attached.

An example of a Kalsim error:

```
kalsim: error: The architecture of the KAP file (3) does not match
               the Kalimba architecture of Kalsim (5). The KAP file is
               for a csr8670, while this is a csr8675 Kalsim.
```

Warnings generated by Kalsim look as follows:

```
kalsim: code execution warning: Write access attempted of unmapped
                                memory location: 0x40b3. The value
                                attempted to be written was: 0x0
                                PC:0x11 Instructions executed:16592
```

Normally Kalsim displays a few messages during startup and sometimes during execution. As an example:

```
kalsim: Loading my_first_dsp_app_kalimba.kap.
```

This terminal output can be disabled with the `--quiet` command line argument.

Port scripts can also generate messages. These are prefixed with the name of the actual port script generating the messages. These messages cannot be silenced by the `--quiet` command line argument.

3 Kalsim tutorial

This section is intended to give an overview of how Kalsim can be used during development. It does not cover all options in depth. To get a complete picture of the options consult the help on the command line or see Kalsim port scripts.

3.1 Running a Kalsim simulation

In this example “my first dsp app” is used as an example application. Kalsim accepts and generates raw PCM samples in the MMU port simulation.

This example requires one input file named `sin16.in` which should contain 2 seconds of a sine wave sampled at 44.1 kHz and 16-bit resolution. This example assumes that “my first dsp app” has been compiled for a CSR8670 chip.

In the command line type:

```
kalsim_csr8670 my_first_dsp_app_kalimba.kap -i 0 sin16.in r=705600 -i 1
sin16.in r=705600  -o 0 output_a.raw r=705600 -o 1 output_b.raw r=705600
```

As the ADC and the DAC are being simulated the sample rate for each of the streams needs to be specified, this is done with the `r=` argument, which uses bit/s as a unit, 44100 samples/s * 16 bits/ sample = 705,600 bits/s in the example. It is also important to match the `.kap` file to the architecture since CSR8670 `.kap` files are incompatible with CSR8675 Kalsim and vice versa.

For more detailed control of how stream port scripts can be used, see Chapter . If the sample rate is omitted from the input streams “my first dsp app” is not able to stream data since it looks at data wrapping inside of the buffers to start operating. The simulation can be terminated by pressing `q` in the terminal running Kalsim at any time during the simulation.

The resulting output files, `output_left.raw` and `output_right.raw`, can be examined and played back with an application which can import the raw output files produced by Kalsim.

NOTE Audacity is able to import the raw files produced by Kalsim and is available freely at http://www.audacityteam.org.

The output files produced by Kalsim in this example are mono 44.1 kHz 16-bit little endian, just like the input file provided.

The produced output starts with 1024 samples of silence. This silence is caused by the simulated processing time of samples passing through the buffers and the simulated Kalimba before reaching the output file. The output streams start streaming at the beginning of the simulation, so this small delay is visible when simulating with a fixed sample rate. After the file has finished streaming, Kalsim continues generating output with the stale data in the MMU buffers until Kalsim is terminated. This is visible in the output because after the 2 seconds of input, the stale 512 samples are repeated continuously.

The streams can also be specified without the sample rate however this means that the MMU buffer levels determine how much data the Kalimba consumes. This is the equivalent of having the MCU generate the data, as in USB or A2DP streams.

3.2 Debugging an application running on kalsim

Kalsim provides extensive debugging support. It is often easier to debug problems in Kalsim because a simulation is repeatable, even with the debugger attached.

The `-d` option enables debugger support in Kalsim. However to attach xIDE to it the debug transport must be changed. Debugging with Kalsim can only be done in a workspace that only contains a Kalimba project. The workspace `my_first_dsp_app_kalsim.xiw` demonstrates the “my first dsp app” project in a suitable form.

The debug transport can be changed by selecting **Transport** in the **Debug** menu and selecting **Kalsim** in the **Transport** window. If the Kalsim transport option is not available, start Kalsim in debugger mode. The option should then appear in the **Transport** window.

When running in Debug mode Kalsim remains halted until xIDE attaches and issues the **run** command. To attach the debugger to Kalsim, select **attach** in the **Debug** menu. Debugging an instance of Kalsim should be identical to real hardware.

3.3 Automated regression testing with Kalsim

Normally Kalsim runs in Interactive mode, which means that the user can terminate the simulation cleanly by pressing `q` at any time during the simulation. However, due to the way this termination mechanism works, this is not always desirable during regression testing.

Kalsim can be run in Non-interactive mode using:

```
kalsim_{chip} -n <other parameters>
```

3.4 Kalsim instruction tracing

Kalsim provides an instruction trace capability. The raw instruction trace generates a binary trace of all the program counter addresses the simulator passed through in the order in which it passed through them.

The format is every 4 bytes stores a 32-bit little endian integer that represents a program counter address. The instruction trace is enabled by `--raw-instruction-trace`, and requires that a file name is specified.

Because the traces are uncompressed these files result in large files if Kalsim is run for a long time.

4 Kalsim port scripts

Port scripts allow arbitrary data flows to be simulated. A port script consists of a sequence of instructions in a text file. A port script can be attached to any input or output port. It is possible to use multiple port scripts in a simulation.

4.1 Kalsim syntax guide

The instructions supported in port scripts are described below. Port scripts support single-line comments that begin with `//`.

stream

Allows a set amount of data to flow into/out of a port. The rate of transfer can be specified. Data can be specified to arrive in blocks (sections of a specific size). This instruction is useful for simulating data sources and data sinks that are controlled by the MCU, such as SCO data and Bluetooth data. This instruction only enforces a maximum data rate.

Syntax:

```
stream <amount> <rate> [block size]
```

Table 4-1describes the `stream` parameters.

Table 4-1 stream parameters

Parameter	Description
<amount>	Quantity of data to transfer in this instruction (in bits)
<rate>	Average transfer rate (in bits per second)
[block size]	Size of indivisible data units (optional, in bits)

strict_stream

Forces a set amount of data to flow into/out of a port. The rate of transfer can be specified. Data can be specified to arrive in blocks (sections of a specific size). This instruction is useful for simulating data sources and data sinks that are hardware-controlled, such as DACs and ADCs. This instruction enforces a constant data rate. On a write port, data is repeated if the DSP application does not provide enough data. On a read port, input data is lost if the DSP does not consume data quickly enough.

Syntax:

```
strict_stream <amount> <rate> [block size] [extra buffer]
```

Table 4-2 describes the `strict_stream` parameters.

Table 4-2 strict_stream parameters

Parameter	Description
<amount>	Quantity of data to transfer in this instruction (in bits)
<rate>	Average transfer rate (in bits per second)
`[block size]`	Size of indivisible data units (optional, in bits)
`[extra buffer]`	Used to simulate extra MCU buffering of input/output data (optional, in bits)

echo

Prints a line of text. Used to provide debugging information.

Syntax:

```
echo <text>
```

Table 4-3 describes the `echo` parameter.

Table 4-3 echo parameters

Parameter	Description
<text>	Any text to be written to `stdout` (Kalsim output window)

time

Prints the current simulation time in microseconds. Used to provide debugging information.

Syntax:

```
time
```

wait

Waits for *n* microseconds. It is advisable to reduce the timer update period from its default 6400 cycles (=100 µs at 64 MHz) if accurate timing is required. This can be done using Kalsim's command-line options.

Syntax:

```
wait <microseconds>
```

Table 4-4 describes the `wait` parameter.

Table 4-4 wait parameters

Parameter	Description
<microseconds>	Number of microseconds to delay

4.2 Kalsim port script example

This example demonstrates the use of a port script to test the response of an AAC decoder application to an input file that is streamed at the wrong speeds to the Kalimba. When the input audio file is streamed too fast, the application cannot read in the data fast enough and the audio skips (jumps). When the input file is streamed too slowly, the decoder application runs out of audio to decode and the output audio wraps (repeats).

Command line:

```
kalsim_{chip} aac_decoder.kap --infile 0 S al05_44k1.adts
s=port_script1.kps         --outfile 0 music_left.dat s=port_script2.kps --
outfile 1 music_right.dat s=port_script3.kps
```

Contents of `port_script1.kps`:

```
echo First port script
echo About to wait 100 us:
wait 100
// Start putting some data in
// Syntax: stream <amount> <rate in bps> [block size] [extra buffer]
echo About to start streaming experiment
strict_stream 100000 132120 1000 40000
echo Just streamed audio file too fast – audio will skip
stream 100000 5000 1000
echo Now streamed the low-data-rate part
// Stream remainder of file
stream 1000000 10000
```

Contents of `port_script2.kps` and `port_script3.kps`:

```
wait 100000
echo A data rate of 705600 bps is equivalent to 44.1k 16-bit samples per
second
strict_stream 0x1000000 705600 16
```

Document references

Document	Reference
Qualcomm Kalimba DSP Assembler User Guide	80-CT425-1/CS-00212259-UG
Qualcomm Kalimba Architecture 3 DSP User Guide	80-CE519-1/CS-00202067-UG
Qualcomm Kalimba Architecture 5 Assembler Quick Reference	CS-00216639-UG

Terms and definitions

Term	Definition
AAC	Advanced Audio Coding
ADC	Analog to Digital Converter
Codec	Coder decoder
DAC	Digital to Analogue Converter
DM	Data Memory
DSP	Digital Signal Processor
I/O	Input/Output
ISS	Instruction Set Simulator
Kalimba	An open platform DSP coprocessor, enabling support of enhanced audio applications, such as echo and noise suppression, and file compression/decompression
LPT	Line Printer Terminal
MCU	Micro Controller Unit
MIPS	Million Instructions Per Second
MMU	Memory Management Unit
MP3	MPEG-1 audio layer 3
PC	Personal Computer
PCM	Pulse Code Modulation
PIO	Parallel Input/Output
PM	Program Memory
QTIL	Qualcomm Technologies International, Ltd.
RAM	Random Access Memory
SCO	Synchronous Connection-Oriented
SPI	Serial Peripheral Interface
USB	Universal Serial Bus
VM	Virtual Machine
xIDE	CSR’s Integrated Development Environment for BlueCore applications