Qualcomm Technologies International, Ltd.

KalAsm 2 to KalAsm 3

User Guide

80-CT433-1 Rev. AL

October 27, 2017

Qualcomm Technologies International, Ltd. (formerly known as Cambridge Silicon Radio Limited) is a company registered in England and
Wales with a registered office at: Churchill House, Cambridge Business Park, Cowley Road, Cambridge, CB4 0WZ, United Kingdom.
Registered Number: 3665875 | VAT number: GB787433096

Revision history

Revision	Date	Description
1	FEB 2010	Initial release. Alternative document number CS-00128869-UG.
2	MAR 2010	Corrected minor typos and document references
3	MAR 2010	Editorial updates
4	JAN 2011	Updated porting advice to reflect additional KalAsm 3 features and improvements
5	MAR 2011	Introduction of KalSim
6	AUG 2011	Updated to new company template. Qualcomm® BlueLab™ references removed
7	AUG 2011	Correction to table 2.1.
8	JAN 2012	Updated to latest CSR™ style.
9	APR 2014	Updated to latest CSR style
10	SEP 2016	Updated to conform to QTI standards; no technical content was changed in this document revision.
AL	OCT 2017	Added to the Content Management System. DRN updated to use the Agile number. No technical content was changed in this document revision.

Contents

Tables

1 KalAsm 2/KalAsm 3 toolchain comparison

Table 1-1 shows the differences in the Kalimba toolchain between KalAsm 2 and KalAsm 3.

Table 1-1 Toolchain changes

KalASm 2 tool name	KalAsm 3 tool name	Tool function
KalAsm 2	kas	Assembles Kalimba code
	klink	Links Kalimba applications
	kobjdump	Provides disassembly from Kalimba linked executable files
	kmapper	Produces a memory map description of a Kalimba linked executable file
kalpac	elf2kap	Creates `.kap` files from Kalimba linked executable files

The new toolchain uses different intermediate file formats to the previous toolchain. Library files now have the extension `.a` instead of `.klib`, and object files have the extension `.o` instead of `.klo`.

KalAsm 3 supports:

- QTIL products containing newer revisions of the Kalimba DSP.
- Linkscripts to specify memory layout, which offer greater control and flexibility than the previous toolchain.
- Improvements in expression handling.
- Faster application building than previous toolchain.
- Improved syntax checking of DSP code.

2 KalAsm 3 porting guide - overview

The syntax changes between KalAsm 2 and KalAsm 3 are minor. The porting actions required to ensure KalAsm 3 compatibility are described below. The porting actions described allow the source to be assembled using both KalAsm 2 and KalAsm 3.

For further information refer to:

- *The KalAsm 3 User Guide* that describes the KalAsm 3 DSP toolchain
- *Qualcomm Kalimba DSP Simulator User Guide* that describes Kalsim, the Kalimba DSP simulator

2.1 Audit pre-processor directives for KalAsm porting

KalAsm 3 uses a standard C pre-processor, whereas KalAsm 2 used a custom pre-processor with slightly different syntax and behavior. There are some minor changes to the use of pre-processor directives.Pre-processor directives must start with a #. The complete list is as follows:

- `include`
- `define undef`.

  NOTE For example, change all instances of `.define` should be changed to `#define` for the new toolchain.

- `if ifdef ifndef else elseif/elsif/elif endif`
- When using `#ifdef`, do not use brackets around the definedsymbol. i.e. change `#ifdef (FOO)` to `#ifdef FOO` or `#if defined(FOO)`.
- `error warning`
- `.message / #message` is no longer available. QTIL recommends that you use the `#warning` directive.

2.2 Prepare linkscript for flash for KalAsm porting

This section describes how flash data is handled in KalAsm 2 and KalAsm 3.

The flash segment `DMCONST_WINDOWED16` is included in the default linker script. When data is placed with `.VAR/DMCONST_WINDOWED16`, then this flash segment is used.

For example, change:

```
.VAR/MYFLASHSEGMENT $my.variable;
```

to:

```
.VAR/DMCONST_WINDOWED16 $my.variable;
```

To maintain KalAsm 2 and KalAsm 3 compatibility place the following lines around any lines containing the pseudo-ops `.DEFGROUP` or `.DEFSEGMENT`:

```
#ifndef KALASM3
…
#endif
```

The KalAsm 2 toolchain used the pseudo-ops `.DEFGROUP` and `.DEFSEGMENT` to define flash groups and segments. For example:

```
#ifndef KALASM3
// -- Define a new flash segment to use called 'FLASHDATA' --
//        Name       Start  End    Width      Tag    Relocatable?
.DEFGROUP FlashGroup 0x0000 0xFFFF 16 myflash.mydata RELOCATABLE;
//          Name      CIRCULAR?   Link Order   Group list
.DEFSEGMENT FLASHDATA             1            FlashDataGroup;
// This address gets filled in by firmware upon a KalimbaLoad call from
the VM
.VAR/DM1 $myflash.mydata.address;
#endif
```

These pseudo-ops are no longer needed with the KalAsm 3 toolchain; instead, this function is moved to linkscripts. It is not necessary to explicitly create a named variable to hold the flash data address (in this example `.VAR/DM1 $myflash.mydata.address`). KalAsm 3 creates a suitably named variable automatically.

If further flash segments are needed, then create an application linkscript (in the example below, name `myflash.mydata` to match the name of the `$myflash.mydata.address` variable):

```
Overlay myflash.mydata DMFLASH 16;
Segment FLASHDATA 10 myflash.mydata;
```

Place the two lines above in a file called `<appname>.link` in the project directory, where *`<appname>`* is the application name.

2.3 Checking assembler warnings when porting to KalAsm 3

KalAsm 3 generates warnings when it encounters some instructions. Check these warnings to ensure that code assembles as expected. For example, the following statement is ambiguous:

```
M[r1 – 1 + 3] = Null;
```

It fits the syntax rule:

```
M[<register> - <expression>] = <register>;
```

Therefore KalAsm 3 assembles this as:

```
M[r1 – 4] = Null;
```

It prints a warning that the instruction is ambiguous. To resolve the warning, include brackets in the expression as appropriate:

```
M[r1 – (1 + 3)] = Null;
```

or

```
M[r1 – (1 – 3)] = Null;
```

Similarly, change:

```
M[$symbol + r1 - 1] = Null;
```

to:

```
M[r1 + ($symbol - 1)] = Null;
```

2.4 Checking any assembler errors when porting to KalAsm 3

KalAsm 3 has stricter rules on instruction syntax is stricter in some rare cases, for example when `Carry` and `Borrow` are used:

The shortcut for `r0 = Null + Null + Carry`:

```
r0 = Carry;
```

Must be written as:

```
r0 = Null + Carry;
```

An assembler error is issued if you use the unsupported form `r0 = Carry`.

2.5 Verifing expressions used in .CONST when porting to KalAsm 3

KalAsm 3 evaluates expressions in `.CONST` statements in a more consistent manner than KalAsm 2. Unusual expressions should be checked for consistency.

Comparing the following `.CONST` expressions between KalAsm 2 and KalAsm 3:

```
.CONST FOO 1.0;
r0 = FOO + 3;
```

KalAsm 2 interpretes this as `0x7fffff + 0x3 = 0x800002`.

KalAsm 3 casts 3 to a float (because it recognizes that FOO is a float) and reports an error (because 1.0 + 3 = 4.0, which overflows a fractional type):

```
.CONST BAR 1.0;
r0 = BAR - 1;
```

This is interpreted by KalAsm 2 as `0x7fffff – 0x1 = 0x7ffffe`.

KalAsm 3 casts 1 to a float (1.0) and interprets the statement as 1.0 – 1.0 = 0, and therefore the result is different between the assemblers.

KalAsm 3 supports various functions inside expressions. These functions (in particular `round()`, `int()`, `ceil()`, `floor()`, and `trunc()`) may help to get expressions working correctly. See the *KalAsm 3 User Guide* for a complete list of supported functions.

2.6 Checking .VAR initializations when porting to KalAsm 3

KalAsm 2 included a shortcut to include initialization data from a file:

```
.VAR $data[] = "inputdata.dat";
```

The name in quotes was interpreted as a filename to include data from. This shortcut does not exist in KalAsm 3.

A mixed-assembler (KalAsm 2 and KalAsm 3) source equivalent of the above statement is:

```
.VAR $data[] =
              #include "inputdata.dat"
              ;
```

As with KalAsm 2, you can move the semicolon into the included file `inputdata.dat` if necessary.

3 Mixed source KalAsm 2 and KalAsm 3 syntax

This topic describes syntax considerations when using the same source for KalAsm 2 and KalAsm 3. Most syntax is highly compatible between the two assemblers.

Expressions syntax compatibility

To keep an application to be compatible with KalAsm 2, extra checks are required.

Integer constants of the following syntax behaves identically in KalAsm 2 and KalAsm 3:

```
.CONST $VALUE 123;
```

However, the following constant behaves differently in KalAsm 2 and KalAsm 3:

```
.CONST $VALUE 0.9;
```

KalAsm 2 evaluates `.CONST` expressions in 24-bit representation. The result in KalAsm 2 depends on whether the number was an integer or a fractional number. For example, Kalasm2 interpretes 1 as an integer `0x000001`, yet 1.0 as a fractional expressed as `0x7fffff` as follows.

If `$VALUE` is used in an expression as follows:

```
.define $VALUE 1.0
r0 = 0.6 * $VALUE (frac);
```

Then KalAsm 2 would have interpreted this as 0.6 * 1.0 = 0.6.

By contrast, the expression:

```
.CONST $VALUE 1.0;
r0 = 0.6 * $VALUE (frac);
```

KalAsm 2 interpretes this as `0.6 * 0x7fffff = 5033164.2`, which is then saturated to 1.0, the maximum fractional value.

KalAsm 3 introduces a new, more consistent expression evaluation system that does not suffer from this problem. This improves the readability of Kalimba assembly code, and makes it more intuitive. It also allows greater flexibility and accuracy in expressions.

Therefore, if KalAsm 2 code is ported to KalAsm 3 (and required to work as mixed-source), take care to verify that the code is assembled as expected, when using complex expressions involving fractional constants.

One method is to compare the generated code from KalAsm 2 and KalAsm 3. You can quickly identify any problems by comparing the disassembly at instructions that use `.CONST` values created from complex expressions. If you see a difference, you can replace `.CONST` with `#define` (pre-processor definitions) for the affected expression. This avoids using the `.CONST` expression evaluator, and therefore reduces the number of possible differences between the KalAsm 2-produced and KalAsm 3-produced code.

When comparing disassembly, note that KalAsm 2 and KalAsm 3 may choose differing binary encodings for some instructions, or choose different instructions that result in the same operation. For example, to load a constant into a register (for example. `r2 = 56`), KalAsm 2 uses the `OR` instructions:

```
r2 = Null OR 56;
```

Whereas KalAsm 3 uses the `+` instruction:

```
r2 = 56 + Null;
```

The two instructions are equivalent, so this substitution is safe. Ignore this difference when comparing disassembly between KalAsm 2 and KalAsm 3.

Document references

Document	Reference
KalAsm 3 User Guide	80-CT425-1/CS-00212259-UG
Qualcomm Kalimba DSP Simulator User Guide	80-CT416-1/CS-00127831-UG

Terms and definitions

Term	Definition
DSP	Digital Signal Processor
QTIL	Qualcomm Technologies International, Ltd