Qualcomm Technologies International, Ltd.

Classic vs Native VM vs Assisted Native VM

Application Note

80-CT403-1 Rev. AJ

October 26, 2017

Revision history

Revision	Date	Description
1	OCT 2008	Original publication of this document. Alternative document number CS-00122636-AN.
2	JUN 2009	Section on selecting the execution mode in xIDE added.
3	JUL 2010	Minor editorial updates.
4	AUG 2011	Assisted Native VM added. Republished in new CSR™ style.
5	JAN 2012	Updated to latest CSR style
6	APR 2014	Updated to latest CSR style
7	MAY 2015	Removed Table 2-2
8	SEP 2016	Updated to conform to QTI standards; no technical content was changed in this document revision.
AJ	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

Figures

1 Virtual machine types

Classic VM

The Classic VM isolates user code in a memory sandbox. The application code is interpreted at runtime and cannot interfere with the basic Bluetooth operation of the chip. This allows the use of prequalified Bluetooth stack firmware.

The issues with Classic VM are:

- Slow execution speed because instructions are interpreted at run time
- 16-bit program counter limiting programs to 64 K words or, if it is less, the size of available flash

NOTE (1) A Large VM was implemented in to allow applications larger than the 64 K word limit, but it was slower than Classic VM. Its distribution was limited to specific applications and only some Unified Firmware v22 releases support it. There is no support for Large VM in Unified Firmware releases after v22 and Qualcomm® BlueLab™ v3.6.

(2) Any code that compiles and runs on Classic VM should also compile and run on Native VM or Assisted Native VM as there are no restrictions or changes to the programming rules.

Native VM

The BlueLab v4.0 release, and QTIL application SDKs that were based on it, introduced Native VM as an alternative to the Classic VM. From Unified Firmware v23d onward Qualcomm® BlueCore™ Firmware supports Native VM execution.

A user application built for Native VM executes instructions directly on the Bluetooth MCU processor. Extra machine instructions, added during compilation, vet memory access so the user application is still effectively executing in a memory sandbox. The application is unable to interfere with the basic Bluetooth operations.

A Native VM application executes instructions at the speed of the processor and faster than a Classic VM application. In addition, the flash size of BlueCore limits the size of the Native VM application, so applications larger than 64 K are possible. However, because of the memory vetting instructions added during the compilation, the code size increases. This negatively affects the speed of execution for code that performs memory access for code variables.

Figure 1-1 Example showing use of stack, global and, pointer variables

```
Uinit8 global_var;
Void foo()
{
   uint8 stack_var;
   uint8 *pointer_var
}
```

Table 1-1 shows the approximate speed and code size increases for different variable types for Native VM applications.

Table 1-1 Memory speed and size increases for Native VM compared to Classic VM

	Stack Variables	Global Variables	Pointer Variables
Speed Increase	~100x	~30x	<5x
Code Size Increase	0	2x	4x

NOTE (1) The extra memory checking is added where a pointer is dereferenced. Passing a pointer variable as a parameter has no additional cost.

(2) Any code that compiles and runs on Classic VM should also compile and run on Native as there are no restrictions or changes to the programming rules.

Assisted Native VM

The CSR 8670 device added Assisted Native VM as an improved version of the Native VM. This is supported on the CSR8670 device with Firmware from 26c onwards.

As with the Native VM, the user application executes instructions directly on the Bluetooth MCU processor. In this case, the extra machine instructions are not added during compilation to vet the memory accesses. There is hardware assistance on the CSR8670 device that vets the memory accesses so the user application is still effectively executing in a memory sandbox. The application continues to be unable to interfere with the basic Bluetooth Operation.

An Assisted Native VM application executes instructions at the speed of the processor at the same speed as the Native VM and therefore faster than a Classic VM application. Applications are again only limited by the flash size of the BlueCore so applications larger than 64 K are possible. As the memory vetting instructions are executed in hardware, there is no code size increase due to the additional memory vetting instructions. The additional size of pointers compared to Classic VM means that there is a small size increase compared with Classic VM.

For the CSR8670, the Assisted Native VM firmware build is provided alongside the Classic VM build. Native VM builds are not provided.

NOTE Any code that compiles and runs on Classic VM should also compile and run on Assisted Native VM as there are no restrictions or changes to the programming rules.

Observations on Classic and Native VM operation

An investigation into the differences between a complete product-ready mono headset Bluetooth application built for both Classic and Native VM performed at QTIL produced the following observations:

- Native code and stack size increased by ~70%.
- Speed of Native execution measured from power on to SCO audio presence was ~31% faster.
- Assisted Native code increased by ~10% and stack size increased by ~70%

A QTIL headset development board using a BlueCore5-Multimedia and a CSR8670 were used for this investigation. The performance testing was not exhaustive.

These statistics may not be comparable for other BlueCore hardware platform designs or applications.

2 Native VM and Assisted Native VM issues

Native VM

Breakpoints and single-stepping do not work when debugging Native VM applications on BlueCore5 and, BlueCore6. All other debug functionality still works, such as:

- Print output
- Message tracing
- BlueStack tracing
- Panic capture
- Postmortem inspection of variables and memory

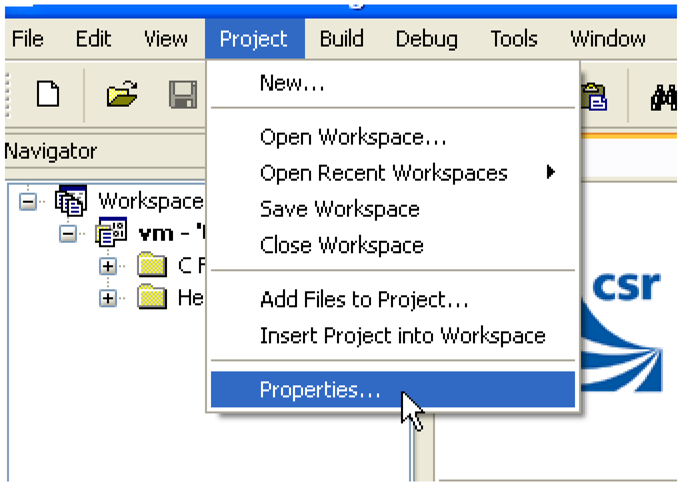
Assisted Native VM

All debug functionality with Assisted Native VM and the CSR8670 work as expected. This includes the four VM Breakpoints and the ability to single-step.

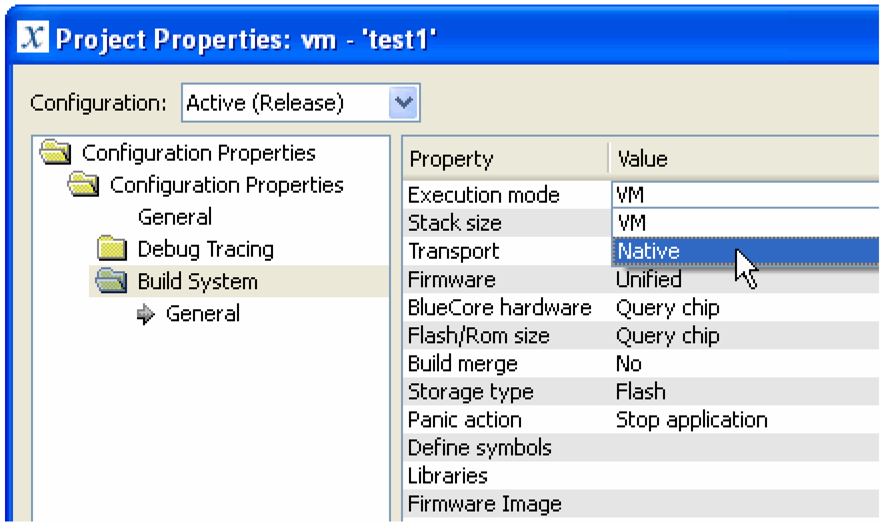
3 Selection of VM execution mode in xIDE

To set the execution mode of an application in xIDE:

1. Select **Properties...** from the **Project** menu on the xIDE menu bar:



A **Project Properties** window appears:



Property	Value
Execution mode	VM
Stack size	VM
Transport	Native
Firmware	Unified
BlueCore hardware	Query chip
Flash/Rom size	Query chip
Build merge	No
Storage type	Flash
Panic action	Stop application
Define symbols	
Libraries	
Firmware Image	

2. Select the **Build System** folder from the **Configuration Properties** tree so that the **Build System** properties are displayed in the right-hand pane.
3. Double-click on the **Execution mode** field and select the required option from the drop-down list.

4 Deciding to use Native VM execution mode

Consider the following points when deciding to use Native VM:

- Native VM only speeds up the on-chip application code. It does not affect the speed of firmware execution or Bluetooth radio throughput.
- Native VM speeds up VM-intensive tasks such as:
  - Parsing button press tables
  - AT command processing HFP/AGHFP libraries)
  - Complex user interface interactions
- Native VM increases the code and stack size of the application, depending on the type of memory access used for the variables of that application.
- The size of available flash is the only limit to Native VM application size.

Breakpoints and single-stepping for debugging do not work on BlueCore5 and BlueCore6. All other debug functionality does work.

Terms and definitions

Term	Definition
AGHFP	Audio Gateway Hands Free Profile
AT	Terminal control commands
BlueCore	Group term for the range of QTIL Bluetooth wireless technology ICs
Bluetooth	Set of technologies providing audio and data transfer over short-range radio connections
HFP	Hands-Free Profile
IC	Integrated Circuit
MCS	Modulation and Coding Scheme
MCU	Micro-Controller Unit
QTIL	Qualcomm Technologies International, Ltd.
SCO	Synchronous Connection-Oriented
SDK	Software Development Kit
VM	Virtual Machine
xIDE	The QTIL Integrated Development Environment for BlueCore applications