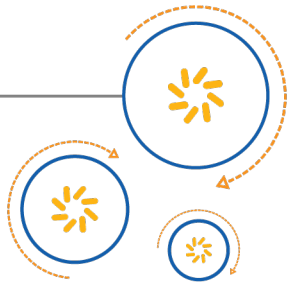
Qualcomm Technologies International, Ltd.



Qualcomm cVc License Key Testing in Production

Application Note

80-CE517-1 Rev. AA

October 18, 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	SEP 2011	Original publication of this document. Alternative document number CS-00126248-AN.
2	JAN 2012	Added GAIA license testing
3	APR 2012	Updated to latest CSR™ style
4	MAY 2013	Replaced incorrect patch in ROM device GAIA example
5	DEC 2013	Updated production test settings for ADK 3.0 and to new CSR style
6	MAY 2014	Add note that no licence is required for GAIA on Flash parts from ADK 3.0 onward.
7	MAY 2015	Removed references to GAIA as testing no longer required with latest ADK.
8	SEP 2016	Updated to conform to QTI standards; no technical content was changed in this document revision.
AA	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 VM application code

1.1 ADK 2.0 and earlier releases

1.1.1 headset_debug.h

To enable license key testing in the Headset code, the symbol `CVC_PRODTEST` must be defined.

This is done in `headset_debug.h` header by replacing:

```
#define CVC_PRODTESTx
```

with:

```
#define CVC_PRODTEST
```

1.1.2 main.c

In the `main` function, the boot mode is checked on start up. Boot mode 4 is used for license key testing.

NOTE Set PSKEY_INITIAL_BOOTMODE to `0004` using **PSTools** to start up in Boot mode 4.

1.1.3 headset_dut.c

cVc and aptX

The function `cvcProductionTestEnter in headset_dut.c` loads the correct Qualcomm® Kalimba™ image. This is determined by the configuration of the Headset. See cVc PS Key configuration for more detail.

The function `cvcProductionTestKalimbaMessage` handles the messages received from Kalimba and sends the necessary messages to configure the Kalimba when the `.kap` file has loaded correctly.

It exits the VM application when a `CVC_SECPASSED_MSG or CVC_SECFAILED_MSG` is received from Kalimba.

1.2 ADK 2.5 and later releases

1.2.1 sink_debug.h

The macro `CVC_PRODTEST` needs to be defined in the `sink_debug.h` header.

1.2.2 main.c

The PSKEY_INITIAL_BOOTMODE must be set to `0004` using **PSTool**.

1.2.3 sink_dut.c

cVc and aptX

The function `cvcProductionTestEnter` in `sink_dut.c` loads the correct Kalimba image. This is determined by the configuration of the Headset PS Keys, see cVc PS Key configuration for more details.

The function `cvcProductionTestKalimbaMessage` handles the messages received from the Kalimba and sends the necessary messages to configure the Kalimba when the `.kap` file has loaded correctly. This function needs to be modified in order for the aptX license check to be performed.

In ADKs 2.5 and 2.5.1, replace the code:

```
/*MESSAGE_SET_SAMPLE_RATE, rate, mismatch, clock_mismatch*/
KalimbaSendMessage(0x7050,0xac44, 0, 0, 0);
```

with:

```
/*MESSAGE_SET_DAC_SAMPLE_RATE, rate, mismatch, clock_mismatch*/
KalimbaSendMessage(0x1070,0x113A, 0, 0, 0);
/*MESSAGE_SET_CODEC_SAMPLE_RATE, rate, mismatch, clock_mismatch*/
KalimbaSendMessage(0x1071,0x113A, 0, 0, 0);
```

NOTE This code change is contained in ADK 3.0 and subsequent releases.

The second argument of both `KalimbaSendMessage` calls is equal to the sample rate divided by 10.

This setting is necessary in order to use a 16-bit variable to handle sample rates such as 96000 Hz.

The value sent to Kalimba is multiplied by 10 in the codec DSP code then restored back to the correct sample rate value.

The function `cvcProductionTestKalimbaMessage` exits the VM application when a `CVC_SECPASSED_MSG or CVC_SECFAILED_MSG` is received from the Kalimba.

2 Firmware

Unified 23g or later firmware is required to support production line license checking.

3 Production test code

3.1 Qualcomm BlueSuite

To access the required TestEngine function `bccmdGetVmStatus` use Qualcomm® BlueSuite™ v2.2 or later . This function allows the tester to read the status of the VM and exit code (when the VM application has terminated).

3.2 bccmdGetVmStatus

Table 3-1 and Table 3-2 show the values returned from `bccmdGetVmStatus`. The values in the tables are defined in `TestEngine.h`.

Table 3-1 Values for returned status parameter

Value	Status	Description
0	`VM_STATUS_BOOT`	Qualcomm® BlueCore™ not initialized
1	`VM_STATUS_FAIL`	Failed to initialize
2	`VM_STATUS_RUN`	Running
3	`VM_STATUS_PANIC`	A panic occurred
4	`VM_STATUS_EXIT`	Normal termination

Table 3-2 Values for returned exit code parameter

Value	Exit Code	Description
1	`CVC_PRODTEST_PASS`	cVc is enabled and the cVc license key is correct
2	`CVC_PRODTEST_FAIL`	The license key is incorrect for this Bluetooth address, license key, and cVc configuration
3	`CVC_PRODTEST_NO_CHECK`	cVc is not enabled on this device so no test has been carried out
4	`CVC_PRODTEST_FILE_NOT_FOUND`	The cVc file for this configuration has not been found in the file system

3.3 cVc example code

```
// Perform CVC license check
static const uint16 PSKEY_INITIAL_BOOTMODE = 0x3cd;
static const uint16 BOOT_MODE_CVC_TEST = 4;  // boot mode for CVC license
```

```
key testing
static const uint16 BOOT_MODE_NORMAL = 1;    // normal boot mode
uint32 dutHandle = openTestEngineSpi(1, 0, SPI_LPT);
if(dutHandle == 0)
{
  cout << "Failed to connect to device under test" << endl;
  return;
}
// Set boot mode for CVC
uint16 bootMode = BOOT_MODE_CVC_TEST;
if(psWrite(dutHandle, PSKEY_INITIAL_BOOTMODE, PS_STORES_I, 1, &bootMode) !
= TE_OK)
{
  cout << "Failed to set boot mode" << endl;
  closeTestEngine(dutHandle);
  return;
}
if(bccmdSetWarmReset(dutHandle, 0) != TE_OK)
{
  cout << "Cold reset failed" << endl;
  closeTestEngine(dutHandle);
  return;
}
// CVC license key check
uint16 status;
uint16 exitCode;
// LOOP_DELAY_MS * MAX_ATTEMPTS give approx timeout in ms
static const uint16 MAX_ATTEMPTS = 100;
static const uint32 LOOP_DELAY_MS = 100;
uint16 attempts(0);
do
{
  if (bccmdGetVmStatus(dutHandle, &status, &exitCode) != TE_OK)
  {
    cout << "bccmdGetVmStatus failed" << endl;
    closeTestEngine(dutHandle);
    return;
  }
  Sleep(LOOP_DELAY_MS);
  ++attempts;
}while((status == VM_STATUS_BOOT || status == VM_STATUS_RUN) && (attempts
< MAX_ATTEMPTS));
// Check returned status and exit code
if ((status == VM_STATUS_EXIT) && (exitCode == CVC_PRODTEST_PASS))
{
  cout << "CVC Production Test PASS" << endl;
```

```
}
else
{
  cout << "CVC Production Test FAIL" << endl;
}
// Set boot mode back to normal
bootMode = BOOT_MODE_NORMAL;
if(psWrite(dutHandle, PSKEY_INITIAL_BOOTMODE, PS_STORES_I, 1, &bootMode) !
= TE_OK)
{
  cout << "Failed to set boot mode" << endl;
  closeTestEngine(dutHandle);
  return;
}
closeTestEngine(dutHandle);
```

3.4 aptX example code

```
// Perform aptX license check
static const uint16 FEATURE_MAP_LEN = 6;
static const uint16 PSKEY_INITIAL_BOOTMODE = 0x3cd;
static const uint16 PSKEY_CONFIG_FEATURE_BLOCK = 0x290;
static const uint16 FEATURE_MAP_APTX_ENABLED[FEATURE_MAP_LEN] = {0x1332,
0x8c01, 0x11e7, 0x20a1, 0x3e40, 0xa51a};
static const uint16 BOOT_MODE_APTX_TEST = 4;  // boot mode for aptX
license key testing
static const uint16 BOOT_MODE_NORMAL = 1;    // normal boot mode
uint32 dutHandle = openTestEngineSpi(1, 0, SPI_LPT);
if(dutHandle == 0)
{
  cout << "Failed to connect to device under test" << endl;
  return;
}
// Read current feature map
uint16 featureMap[FEATURE_MAP_LEN];
if(psRead(dutHandle, PSKEY_CONFIG_FEATURE_BLOCK, PS_STORES_I,
FEATURE_MAP_LEN, &featureMap) != TE_
{
  cout << "Failed to read current feature map" << endl;
  closeTestEngine(dutHandle);
  return;
}
// Set feature map with aptX enabled
if(psWrite(dutHandle, PSKEY_CONFIG_FEATURE_BLOCK, PS_STORES_I,
FEATURE_MAP_LEN, &FEATURE_MAP_APTX_ENABLED) != TE_OK)
{
```

```
  cout "Failed to write aptX feature map" << endl;
  closeTestEngine(dutHandle);
  return;
}
// Set boot mode for aptX license check
uint16 bootMode = BOOT_MODE_APTX_TEST;
if(psWrite(dutHandle, PSKEY_INITIAL_BOOTMODE, PS_STORES_I, 1, &bootMode) !
= TE_OK)
{
  cout << "Failed to set boot mode" << endl;
  closeTestEngine(dutHandle);
  return;
}
if(bccmdSetWarmReset(dutHandle, 0) != TE_OK)
{
  cout << "Cold reset failed" << endl;
  closeTestEngine(dutHandle);
  return;
}
// aptX license key check
uint16 status;
uint16 exitCode;
// LOOP_DELAY_MS * MAX_ATTEMPTS give approx timeout in ms
static const uint16 MAX_ATTEMPTS = 100;
static const uint32 LOOP_DELAY_MS = 100;
uint16 attempts(0);
do
{
  if (bccmdGetVmStatus(dutHandle, &status, &exitCode) != TE_OK)
  {
    cout << "bccmdGetVmStatus failed" << endl;
    closeTestEngine(dutHandle);
    return;
  }
  Sleep(LOOP_DELAY_MS);
  ++attempts;
}
while((status == VM_STATUS_BOOT || status == VM_STATUS_RUN) && (attempts
< MAX_ATTEMPTS));
// Check returned status and exit code
// aptX checking uses same values for exitCode as CVC testing
if ((status == VM_STATUS_EXIT) && (exitCode == CVC_PRODTEST_PASS))
{
  cout << "aptX Production Test PASS" << endl;
}
else
```

```
{
  cout << "aptX Production Test FAIL" << endl;
}
// Set boot mode back to normal
bootMode = BOOT_MODE_NORMAL;
if(psWrite(dutHandle, PSKEY_INITIAL_BOOTMODE, PS_STORES_I, 1, &bootMode) !
= TE_OK)
{
  cout << "Failed to set boot mode" << endl;
  closeTestEngine(dutHandle);
  return;
}
// Return feature map to original value
if(psWrite(dutHandle, PSKEY_CONFIG_FEATURE_BLOCK, PS_STORES_I,
FEATURE_MAP_LEN, &featureMap) != TE_OK)
{
  cout << "Failed to restore feature map" << endl;
  closeTestEngine(dutHandle);
  return;
}
closeTestEngine(dutHandle);
```

3.5 cVc PS Key configuration

To carry out testing, a standard set of cVc-enabled PS Keys are required on the headset. Table 3-3 shows the PS Keys that are critical to the correct operation of cVc on a headset.

Table 3-3 PS Keys for cVc license configuration

PS Key	Name	Description
0x0001	PSKEY_BDADDR	Bluetooth address, which must correspond with the cVc license key.
0x0290	PSKEY_USR6	Feature map. Word 4 Bits[11:8] of Word 4, configure the audio plug-in. For example, if this key is set to: 1332 8c01 11e7 21a1 3e40 a41a The second digit in the 4th Word configures the plug-in and the type of cVc used.
0x2288	PSKEY_DSP48	The cVc license key is stored here.

3.6 aptX PS Key configuration

Table 3-4 shows the PS Keys for aptX configuration. For testing an aptX license key, the audio plug-in must be set to zero and the aptX codec enabled in PSKEY_USR6.

Table 3-4 PS Keys for aptX license configuration

PS Key	Name	Description
`0x0001`	PSKEY_BDADDR	Bluetooth address, this must correspond with the atpX license key
`0x0290`	PSKEY_USR6	Feature map, the audio plug-in is configured by word 4, Bits [11:8] and should be set to zero. The `A2dpOptionalCodecsEnabled` bitfield should be configured so that the aptX codec is enabled: `1332 8c01 11e7 20a1 3e40 a51a`
`0x226c`	PSKEY_DSP20	The aptX license key is stored in this PS Key

3.7 Testing cVc and aptX on the same device

To perform multiple license key tests, work through the tests described in this document in sequence using the correct PS Key configuration for each individual test.

Terms and definitions

Term	Definition
aptX	Proprietary high quality audio codec
BlueCore	Group term for the range of QTIL Bluetooth wireless technology ICs
BlueSuite	BlueCore family of software utilities for Bluetooth evaluation and development
cVc	Clear Voice Capture
IC	Integrated Circuit
Kalimba	An open platform DSP co-processor, enabling support of enhanced audio applications, such as echo and noise suppression, and file compression/decompression
PS Key	Persistent Store Key
QTIL	Qualcomm Technologies International, Ltd.
ROM	Read Only Memory
VM	Virtual Machine