# Si4010 NVM Programming Utility User's Guide

## 1. Purpose

This document is a user guide for the Si4010 NVM Programming Utility. The purpose of the utility is to prepare and burn user code and data into the non-volatile memory (NVM) of supported devices. This is a one time programmable NVM. However, the full NVM does not need to be programmed at one time. Some parts of the NVM can be programmed in one step, and more of the NVM can be programmed at a later time.

## 2. Overview

The Si4010 has 8 kB of NVM which is used to store the trim and user code. However, the code is not executed from NVM, but from 4.5 kB of RAM on the chip. Upon boot, a boot routine copies the user code stored in the NVM to the RAM and runs it from there.

For the boot routine to understand the user code, say in the form of IntelHEX (or Verilog MEM) input file formats, it has to be processed and "composed" into data structures the boot routine will understand. Therefore, the code has to be passed through a composer program to prepare the boot data structures, which will be burnt into the NVM.

The other task of the compose process is to create specialized code to burn the "composed" data structures into NVM. The output of the compose process is an NVM burn file (*.nbf default extension) which contains all the information needed to program the device.

The Si4010 NVM Programming Utility (Si4010_NVM_Burner.exe) is a graphic "shell" around the NVM composer command line executable (gui_composer.exe). The Si4010 NVM Programming Utility is also capable of writing the internal EEPROM (MTP) or burning the composed NVM burn file to the connected device. The following sections will provide details on both the GUI and the underlying command line executable.

# 3. NVM Programming Utility

## 3.1. Operation Flow

The NVM GUI flow is as follows:

1. Select the **USB adapter**.
2. Press the **Connect** button t**o connect t**o the part.

Either:

1. Select input IntelHEX or Verilog MEM input files for the **Boot** or **Application (Overlay)** section.
2. If Overlays are used, specify NVM start addresses in the **Overlay** section. For a trial run to determine the sizes of the overlays, keep the addresses as 0.
3. Specify NVM start **addresses** in the **Boot** section. This is rarely, if ever, needed. Keep all the address values as they are – first line as 0xE180 (for example) address is *filled automatically*, subsequent addresses as 0x0. The automatic fill of the first item is coming from the information stored in the **Factory** part of the NVM, value **wBoot_NvmUserBeg** variable.
4. Specify the new output **NVM Burn File**. Choose **Overwrite** if desired. The GUI will check the file existence before running the composer.
5. Press the **Compose** button and observe the results. If there are no errors or conflicts, one can proceed to **Burn**. If there are errors, see the **Compose Log**, change the inputs, and hit **Compose** again until there are no errors. Then go to the burn steps below.

Or:

**Load** existing, previously generated, **NVM Burn File**. The **Compose Map** gets filled in from the file. Then go to the burn steps below. In both cases:

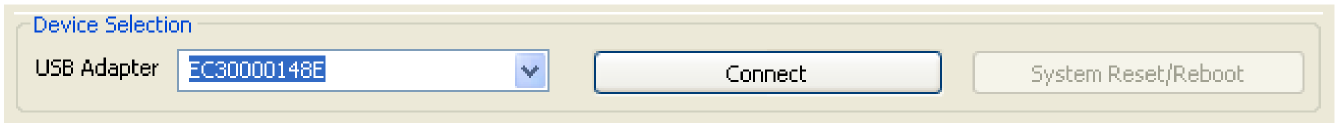
1. Make sure the 6.5 V is connected to the GPIO[0] of the part.
2. Press the **Burn** button to the burn the part. Observe the results.

## 3.2. GUI Description

The GUI consists of a common device selection area and a 4 tab control which are described in the following sections.

### 3.2.1. Device Selection

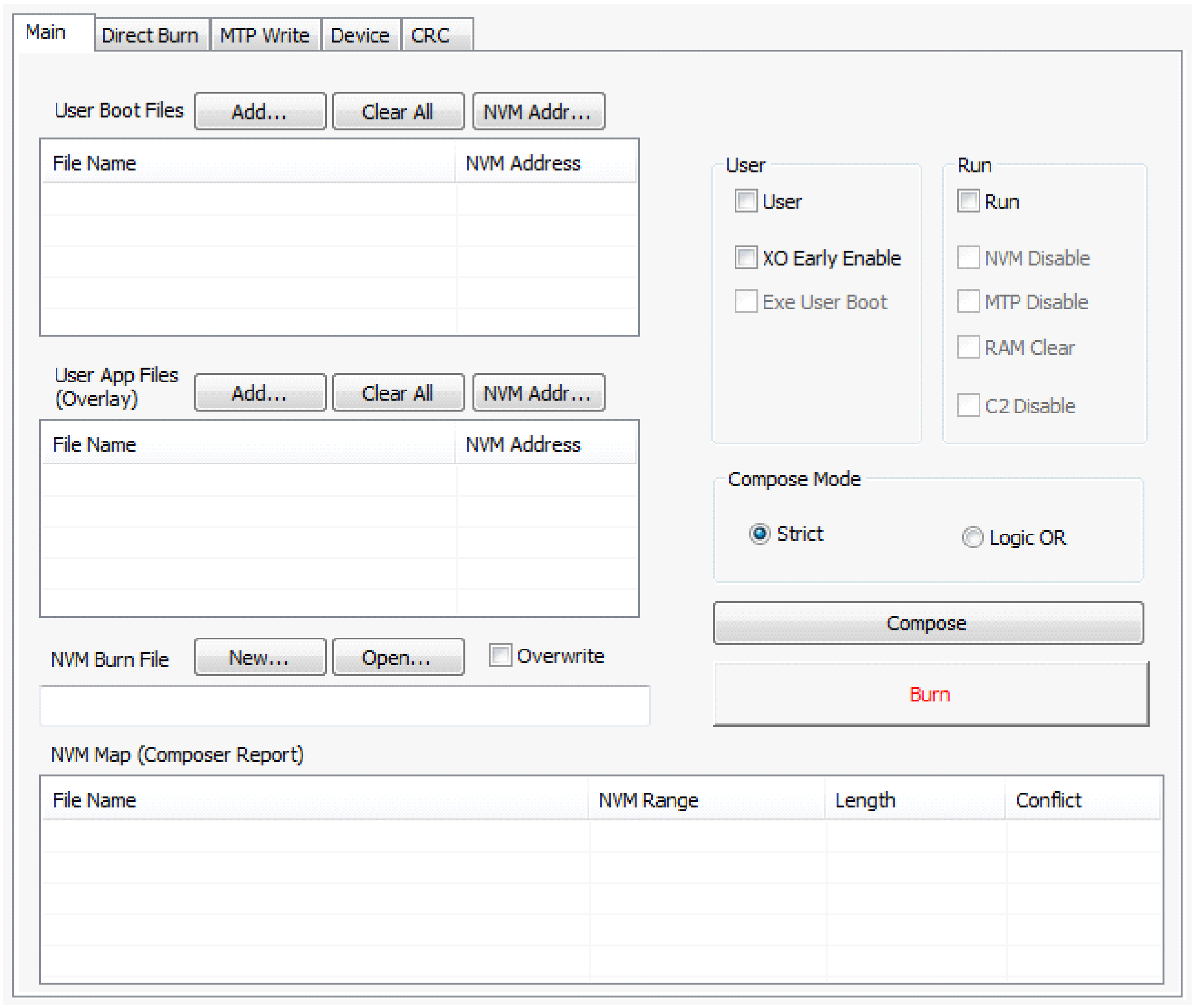
The NVM GUI is using the C2 debugging chain to talk to the device through a USB or ToolStick Debug Adapter. The device selection is common to the Main, Direct Burn and Device tabs.



**USB Adapter**: Select the adapter from the available list.

- **Connect**: When pressed, the C2 debug chain connects to the part. The part is not running any code after the connection. When successfully **connected** to the part, the Connect button text should change to **Disconnect**.
- **System Reset/Reboot**: The button is inactive (shaded) if the device is not connected to the GUI. When pressed, the system reset is invoked and the part reboots. It is equivalent to using the **Reset** button in the Silicon Labs IDE. After the reset the part does not execute any code unless it was burnt as a **User** part (with the **Exe User Boot** bit set) or a **Run** part.
- **Disconnect**: The button should change state to **Connect**. After it is disconnected the device behaves as if the power to the device was cycled.

### 3.2.2. Main Tab

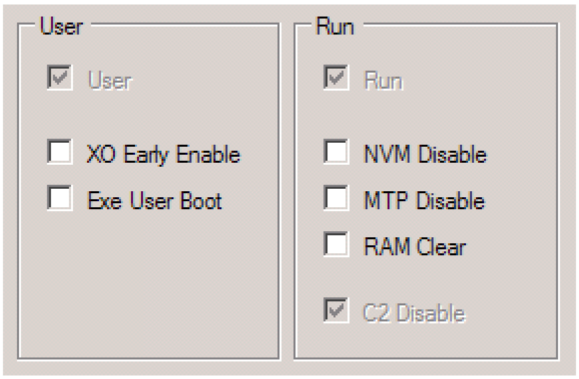


Compose Mode: [✓ Strict] Logic OR

#### 3.2.2.1. Information Retrieved from the Device

After the GUI is connected to the device and the Device Information is displayed, the other parts of the GUI get their values based on the values retrieved from the device. The check box values affected by this are User, Run, and C2 Disable.

However, if the part is programmed as Run, it is no longer possible to connect to it from the burner GUI.


User: [✓ User], XO Early Enable, Exe User Boot

Run: [✓ Run], NVM Disable, MTP Disable, RAM Clear, [✓ C2 Disable]

Note that it is not possible to retrieve the previously programmed information related to the remaining 5 check boxes. For example, if the Exe User Boot bit was previously burnt, and then you reconnect to the device, then the check box will stay clear and not reflect the current status of the device.

#### 3.2.2.2. User and Run Flags

The User and Run flag boxes set the behavior of the chip. Each check box corresponds to a single bit in the control registers and their corresponding values in the NVM. Once the bit is set then it cannot be cleared.

The known values of the User, Run, and C2 Disable bits are read during the Connect or System Reset events. Once the bit is set in the chip it cannot be changed, so the check box is marked and disabled as shown above.

The behavior of the other boxes depends on the User and Run check box setting. If the box is set then it does not matter if it is set by the user checking it, or by the System Reset settings already programmed into the chip.

All 5 unchecked boxes have corresponding bits in the PROT3_CTRL register described in the Si4010 specification document.

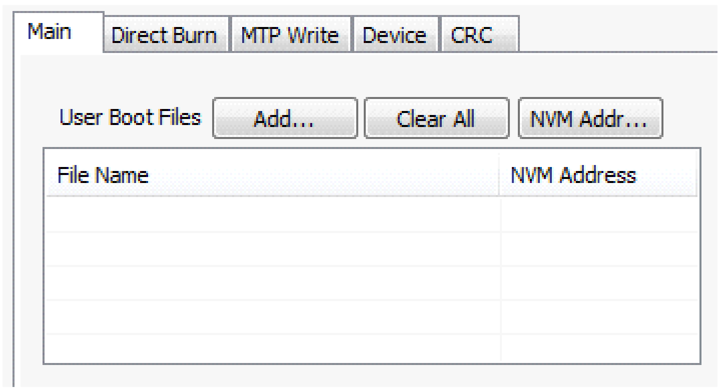
#### 3.2.2.3. User Boot Input Files

The input file type is determined by file extension. There are two input file types, IntelHEX (.hex extension) and Verilog MEM file (.mem extension).

The User Boot input files determine the user specified load which is loaded to the chip at boot time if the chip state is User or Run. The User Boot load is the only user specified NVM content the boot routine will be aware of.

There is a possibility of more than one file. If there is more than one file they are ordered in the box. The file order matters and they will be processed in the order listed. All the files listed in this box will be loaded by boot routine at boot time.

The input file box is shown below for reference:



When the GUI starts this box is empty. The user entered values (files and addresses) in the box will persist in between Connect/Disconnect sessions with different devices.

- **Add**: Opens a file selection dialog box with *.hex and *.mem masks as default.
- **Clear All**: Clears all the content of the box, including all the file names and addresses. There is a right click context menu **Clear** which allows removal of a single file.
- **NVM Addr**: Modifies **NVM Address** for the selected file line. This is the address at which the files converted **Block** will start in NVM. The first file address is not editable and is retrieved from the device and filled automatically. If the address value is set to 0 then the converted blocks will be stored in the NVM back to back without any gaps.

Each file in this box will be converted into a separate single **Block**.

The user can choose to have gaps in between the converted blocks. That is useful if some converted blocks need to be placed at specific NVM addresses and possibly used as an overlay at chip run time. The composer will automatically introduce a gap in the NVM if the blocks are not back to back. The boot routine will still boot all the blocks in this box, even if there are gaps in between blocks. However, once the gap is introduced in between blocks in this **User Boot** window, that NVM space is not usable any more.

The list box items are:

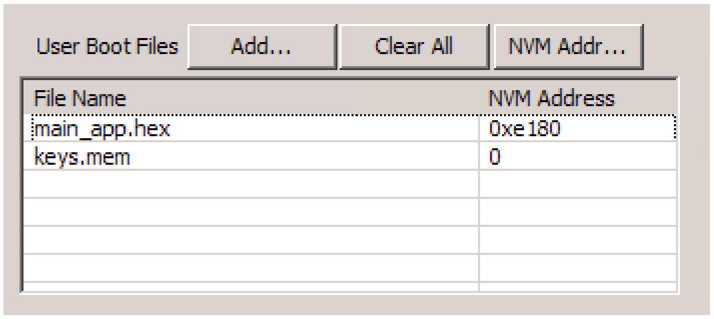
- **File Name**: User chosen IntelHEX or Verilog MEM file. The name displayed is without the file path to simplify viewing.
- **NVM Address**: The right column in the box. Displays the NVM starting address of the converted file Block in the NVM. The address is in hexadecimal notation.
- A single file can be removed from the list by right clicking on it and selecting "clear."

### 3.2.2.4. Input File Box Behavior

When the box is empty (there are no files), the first file added will get an automatic, uneditable, NVM address taken from the value of the variable wBoot_NvmUserBeg read from the device after system reset (see above).

The subsequent files get a default NVM address value as 0. That value is editable, but the special value 0 has its own meaning. If kept there it means that the subsequent files are composed into the NVM back to back without any gaps. The NVM address of the converted file is determined by the NVM address where the previous file fit into the NVM.

Example of the input box with two files:



| File Name | NVM Address |
|---|---|
| main_app.hex | 0xe180 |
| keys.mem | 0 |

### 3.2.2.5. User Application Files for Overlay

Very similar input file box to the **User Boot** above with almost identical behavior. The **User App** input files determine the user specified load which is loaded to the chip by the user at runtime using **bNvm_LoadBlock()** or **bNvm_CopyBlock()** API functions.

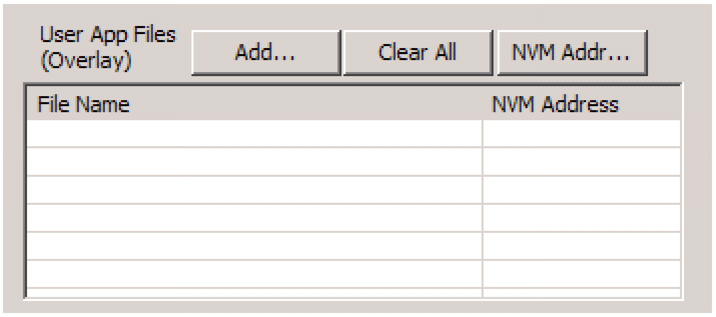
The boot routine does not know about these files.

It is a user responsibility to place the converted files in the NVM at non-conflicting addresses. Therefore, valid NVM addresses are required for these files.

Each of the files listed is converted into a single Block. The specified NVM Address for each file must match the NVM address provided as an input argument to the bNvm_LoadBlock() function. The API function then reads the composed blocks starting from the specified NVM address.

For example, if there are 3 overlay functions, there will be 3 files listed, one for each overlay.

The input file box is shown below for reference:



When the NVM GUI starts this box is empty. The user entered values (files and addresses) in the box will persist in between **Connect/Disconnect** sessions with different devices.

- **Add**: Opens a file using file dialog box.
- **Clear All**: Clear all the content of the box, all the file names and addresses.
- **NVM Addr**: Modify NVM Address for the selected file line. All the files listed have editable NVM addresses. The valid address is required to be provided by the user!

The list box items:

- **File Name**: User chosen IntelHEX or Verilog MEM file. The name displayed is without the file path to simplify the viewing.
- **NVM Address**: NVM starting address of the converted file in the NVM. The address is in hexadecimal notation.
- A single file can be removed from the list by right clicking on it and selecting “clear”.

#### 3.2.2.6. Input File Box Behavior and Overlay

When the file is added to the list the NVM address is set to 0 for each file. That is not a valid address. The valid address has to be in the region of 0xE010 to 0xFFB0. But the user can keep the invalid address there for a trial Compose run. Here is why:

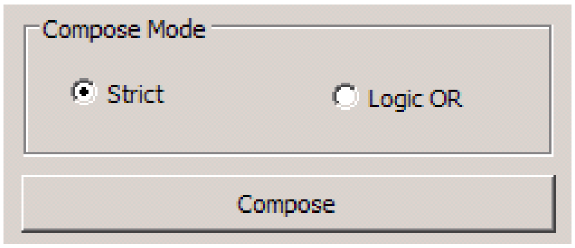
Each overlay is one file as entered. The size of the file is known by the user, but what is not known is how much space the converted file (composed Block) is going to take in the NVM. To determine this, the user must use a two-step compose process.

1. Keep all the **NVM Address** fields as 0x0 or some other invalid address. Press **Compose**. The composer will run and report the lengths of each composed region, indicating how much space each file is going to take in the NVM. The compose process reports a failure because of the invalid addresses, but that is expected.
2. Take the length information from the first **Compose** run and edit the **NVM Address** fields of the input files to reflect the actual NVM sizes. Then the **Compose** button can be hit again and then the correct compose run is invoked, which should end up in valid, burnable output.

It is up to the user to pick a proper, non-overlapping, **NVM Address** for each **User App** file.

#### 3.2.2.7. Compose Mode and Compose Button

The composer generates the burning algorithm data. It is required to specify the burning algorithm behavior before composing. The **Compose Mode** selection controls the algorithm. See discussion about the mode in the previous sections.



**Compose**: When the button is pressed, the NVM GUI checks whether the **NVM Burn File** is specified. The user must specify the output file. The compose process starts. The GUI collects the composer log output and displays it in the **Device** tab in the **Compose Output** window.

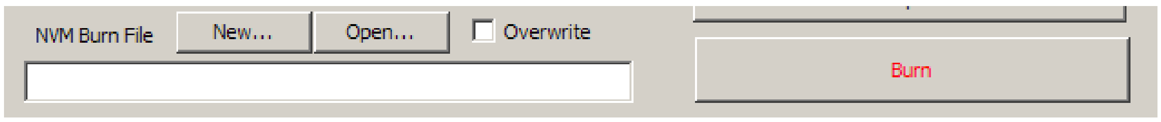
After the composer finishes, the GUI displays the **Compose Map** section from the just generated file into the **NVM Map**.

The **NVM Burn File** will always be recreated even if there is a composer run failure.

### 3.2.2.8. NVM Burn File and Burn Button

The NVM Burn File box serves two purposes:

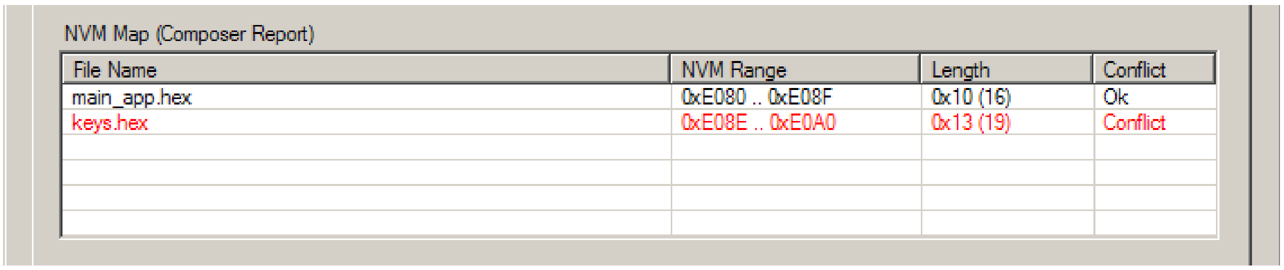
1. To enter the new output file for the next composer run. The entered value persists in between subsequent composer runs. It is not necessary to enter a new file for each composer run. It is possible just to type the file name directly without invoking the file input box.
2. To open the existing **NVM Burn File**, generated and saved during previous NVM GUI session. Then there would not be a **Compose** step and the selected file will be used for burning.



- **New**: Enter a new **NVM Burn File** file name for the subsequent compose run.
- **Open**: Open the existing **NVM Burn File**. The existing **NVM Burn File** is read into the GUI, and the [**Compose Map**] section from that file is displayed in the **NVM Map** list box.
- **Overwrite**: Check box to overwrite the specified **NVM Burn File** during the next composer run.
- **Burn**: Burn the data into the device as specified in the currently opened **NVM Burn File**. The user must make sure that the 6.5 V is supplied to GPIO0 of the chip.

### 3.2.2.9. NVM Map

After the composer is run by pressing **Compose** button, or after the user **Open**-ed the existing **NVM Burn File** manually, the N**VM Burn File** is opened, and the items from the [**Compose Map**] are listed in the **NVM Map** box.

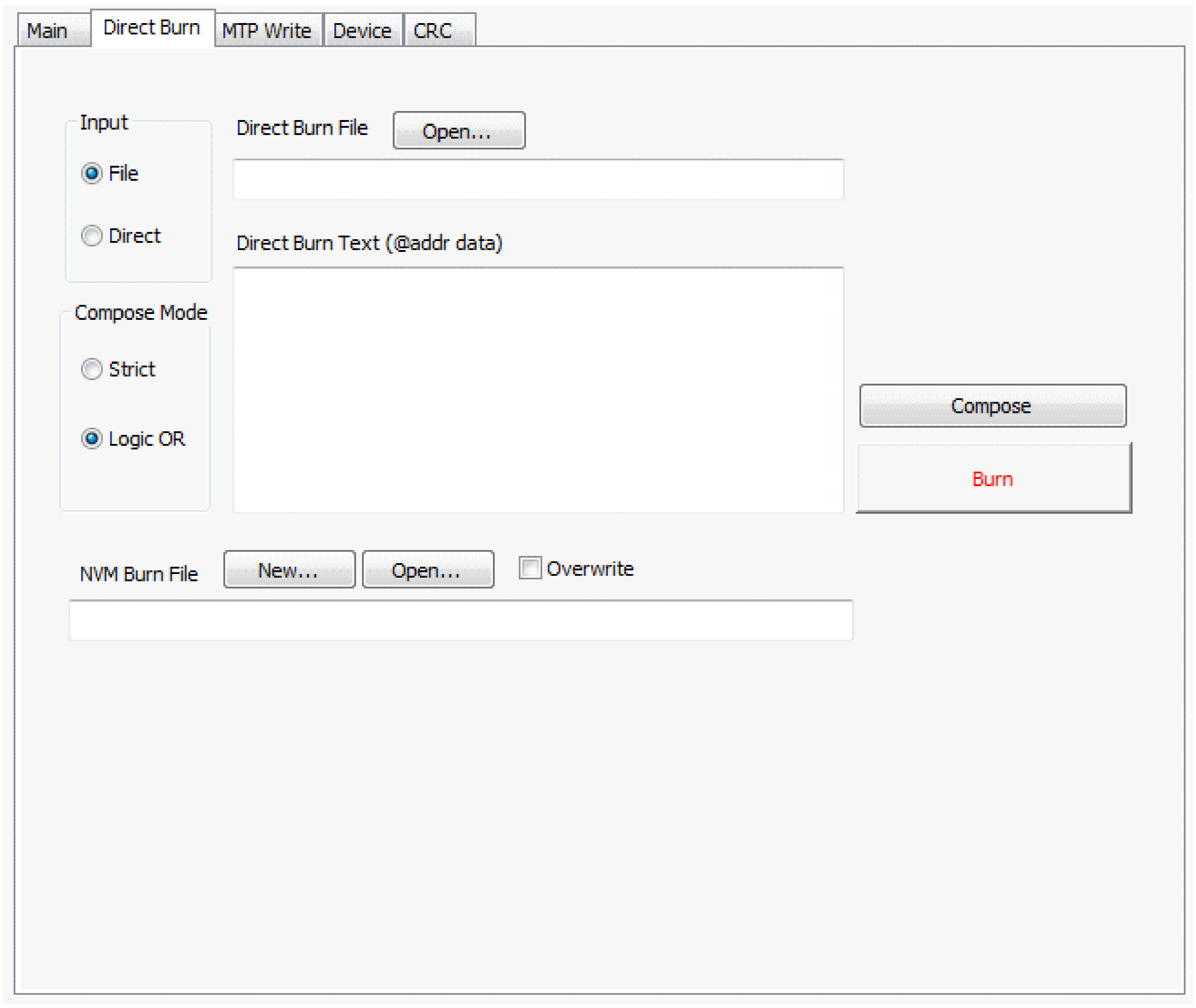


NVM Map (Composer Report)

| File Name | NVM Range | Length | Conflict |
|---|---|---|---|
| main_app.hex | 0xE080 .. 0xE08F | 0x10 (16) | Ok |
| keys.hex | 0xE08E .. 0xE0A0 | 0x13 (19) | Conflict |
| | | | |
| | | | |
| | | | |
| | | | |

If the last column (the **Conflict** column) says either **Conflict** or **Error** then the line is highlighted in red. It is not possible to burn the **NVM Burn File** which includes any conflict or error.

### 3.2.3. Direct Burn Tab



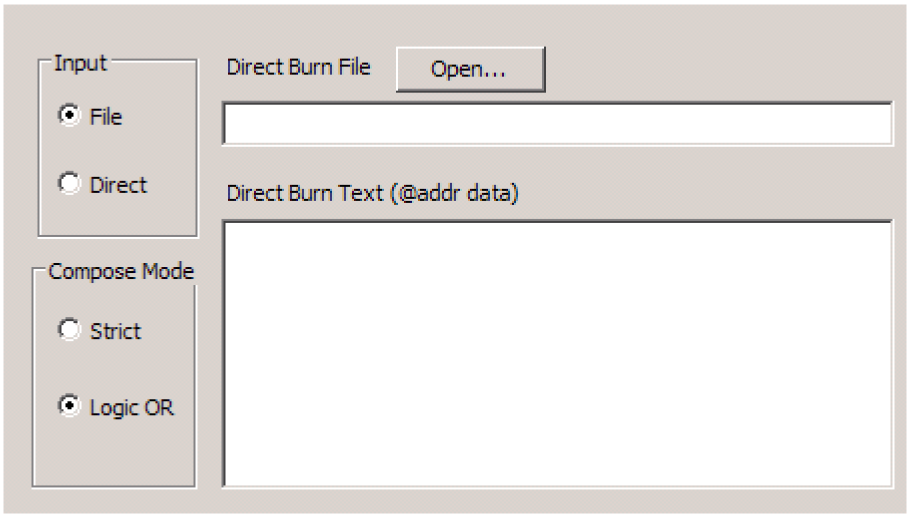
The **Direct Burn** tab describes the second (advanced) mode of the NVM GUI composer. In this mode the composer takes either a direct string of NVM addresses and data in the Verilog MEM file fashion, or the Verilog MEM like file with direct NVM addresses and data specified. The format uses hex numbers without a 0x prefix and is described below.

The input addresses and data are the actual direct NVM address and data. This mode will be used only for advanced purposes.

#### 3.2.3.1. Data Input

- **Input**: The data input radio button selects either input from a Verilog MEM **File** (no IntelHEX is allowed in this mode) or from a **Direct** text directly pasted into the input text box.
- **Direct Burn File**: Verilog MEM file (***.mem**) file path and name. It is possible to paste the name of the file directly to the box.
- **Direct Burn Text**: The input text must be in the format of **@addr data data** with 4 hexadecimal digits describing address and 2 digits describing the data values. For example:

```
@0003 15 A4 3E
7E 56 @0015 89 F5 CD
89
AB
@10F4
DF C7 A4
```



Input: [✓ File] Direct

Compose Mode: Strict [✓ Logic OR]

#### 3.2.3.2. Compose Mode and Compose Button

The buttons behave similarly as in the **Main** tab. Pressing the **Compose** button just reflects the input entered on the currently selected **Direct Burn** tab.

- **Compose Mode**: Exactly the same meaning and behavior as in the **Main** tab. However, there is no connection between this **Compose Mode** box and the one on the **Main** tab. They are independent. Their default values are the opposite of each other - for Direct Burn we want **Logic OR** to be the default.
- **Compose**: Same behavior as in the **Main** tab, but the data passed to NVM composer is collected and put together based on the inputs in the current **Direct** tab.
  Note that the NVM composer writes the following single line into the output file in the **[Compose Map]** section:
  [Compose Map]

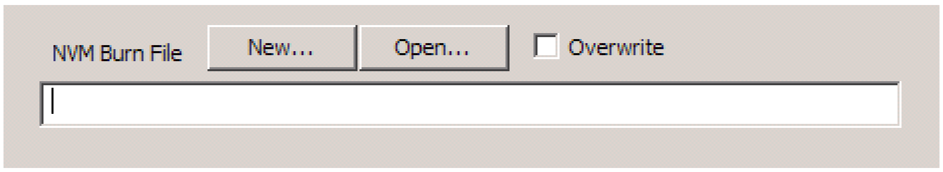
```
DirectBurn -- -- -- -- Ok
```

Otherwise, the output file is identical to the regular one.

### 3.2.3.3. NVM Burn File and Burn Button

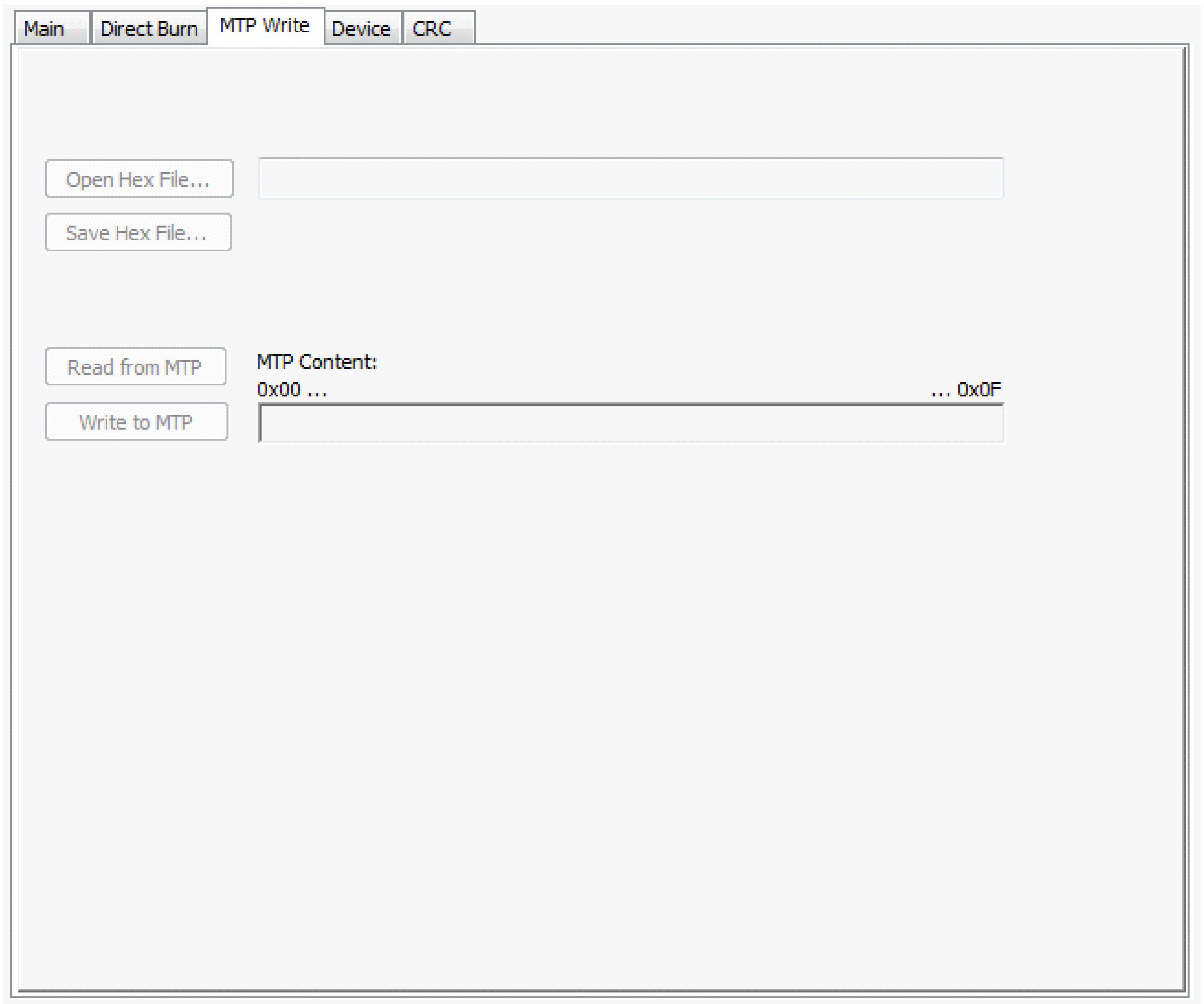
It has the exact same behavior as on the Main tab. The user can select the output file for the composer in the NVM Burn File box.

The **Burn** button invokes the exact same sequence of events as in the **Main** tab, but it takes the **NVM Burn File** from the local **Direct Burn** tab as shown below.



The **NVM Burn File** is local to each tab and not a common value, “seen through” from each tab as a common value.

### 3.2.4. MTP Write



The GUI is capable to read or write the content of the internal EEPROM (MTP) of the connected device. When connecting to the device, the utility reads the MTP memory content of the chip and displays it in the MTP content box. The controls on the MTP Write tab are only enabled if the utility is connected to a device.

#### 3.2.4.1. Open Hex File

The Open Hex File button is used to select the Intel hex file that contains the data bytes the user wants to modify in the MTP memory. When a hex file is selected the MTP data is extracted from the file and displayed in the MTP content box. At addresses where the hex file does not contain data, the original data read from the MTP memory will be kept.

**Note:** The current release does not support Verilog MEM files as an input for writing the MTP. However, the user can use the hexext.bat tool to create the IntelHEX file from a Verilog MEM file. For more information on the hexext.bat tool, refer to “AN674: Si4010 NVM Burning Tools and Flows”.

#### 3.2.4.2. Save Hex File

When the Save Hex File button is pressed the selected hex file is updated with the current content of the MTP content box and saved.

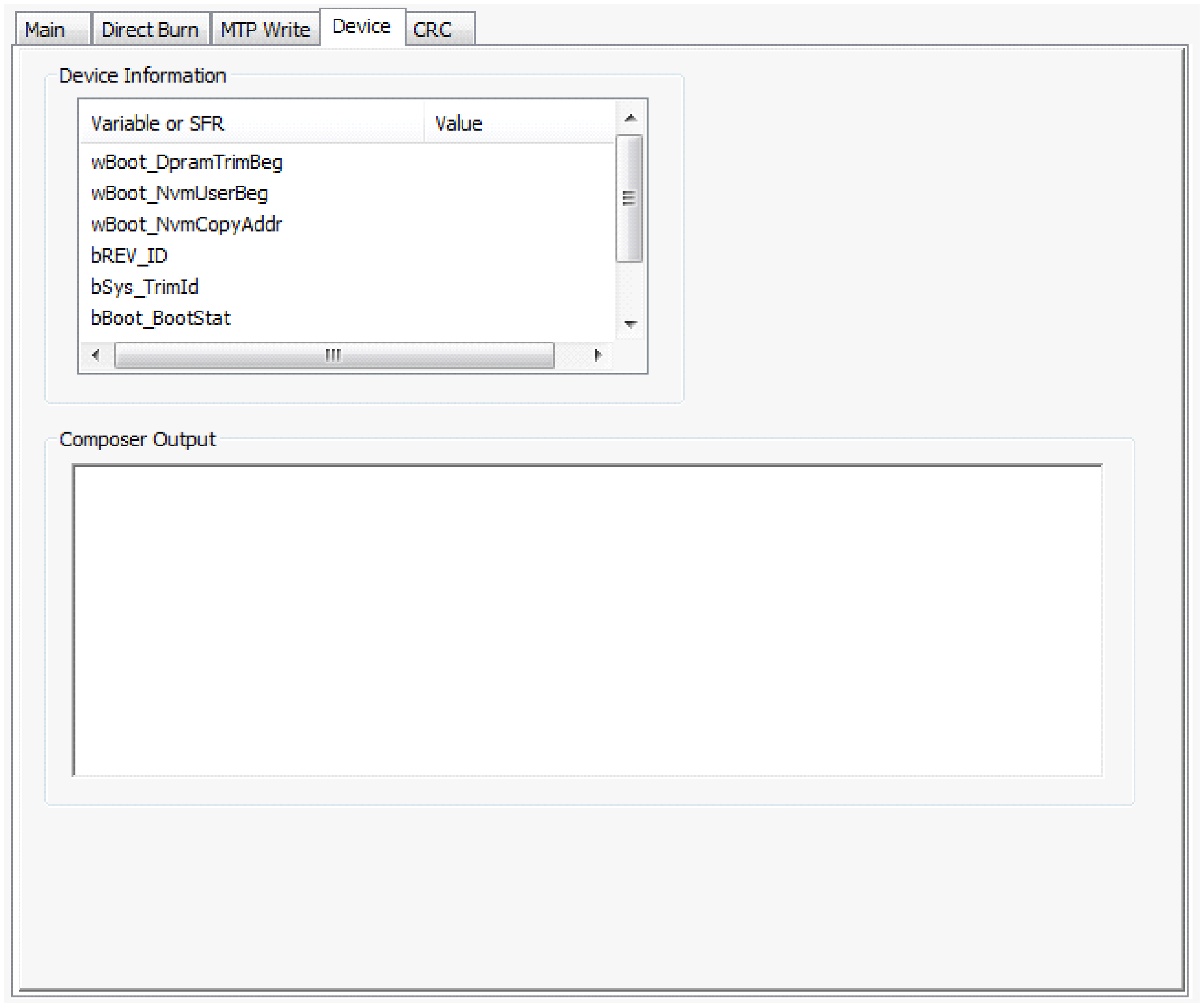
#### 3.2.4.3. Read from MTP

The Read from MTP button will read all the 32 bytes of the MTP memory and display it in the MTP content box.

#### 3.2.4.4. Write to MTP

When the Write to MTP button is pressed, the current content of the MTP content box is written in the MTP memory of the device.

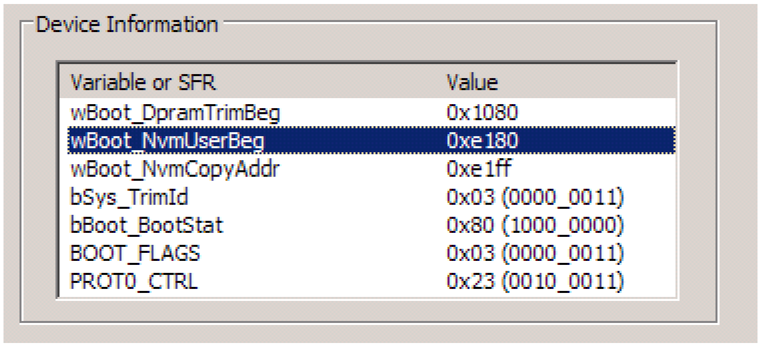
### 3.2.5. Device Tab



| Variable or SFR | Value |
|---|---|
| wBoot_DpramTrimBeg | |
| wBoot_NvmUserBeg | |
| wBoot_NvmCopyAddr | |
| bREV_ID | |
| bSys_TrimId | |
| bBoot_BootStat | |

### 3.2.5.1. Device Information

The Device Information box has a list of Variable or SFR values. The values are refreshed after each Connect or more specifically, after each System Reset. When the device is disconnected the Value column is cleared.


Device Information

| Variable or SFR | Value |
|---|---|
| wBoot_DpramTrimBeg | 0x1080 |
| wBoot_NvmUserBeg | 0xe180 |
| wBoot_NvmCopyAddr | 0xe1ff |
| bSys_TrimId | 0x03 (0000_0011) |
| bBoot_BootStat | 0x80 (1000_0000) |
| BOOT_FLAGS | 0x03 (0000_0011) |
| PROT0_CTRL | 0x23 (0010_0011) |

The presented values are critical values about the status of the part. They also convey information about the beginning addresses of RAM and NVM regions available to the user.

### 3.2.5.2. Composer Output

The C**omposer Output** box collects the composer log whenever the **Compose** button is pressed on the **Main** or **Direct Burn** tabs.

### 3.2.6. CRC Tab

This tab has two user buttons. The “Run” button outputs CRC calculation results to the “CRC Calculation Output” text box. The “Clear” button clears the text box.

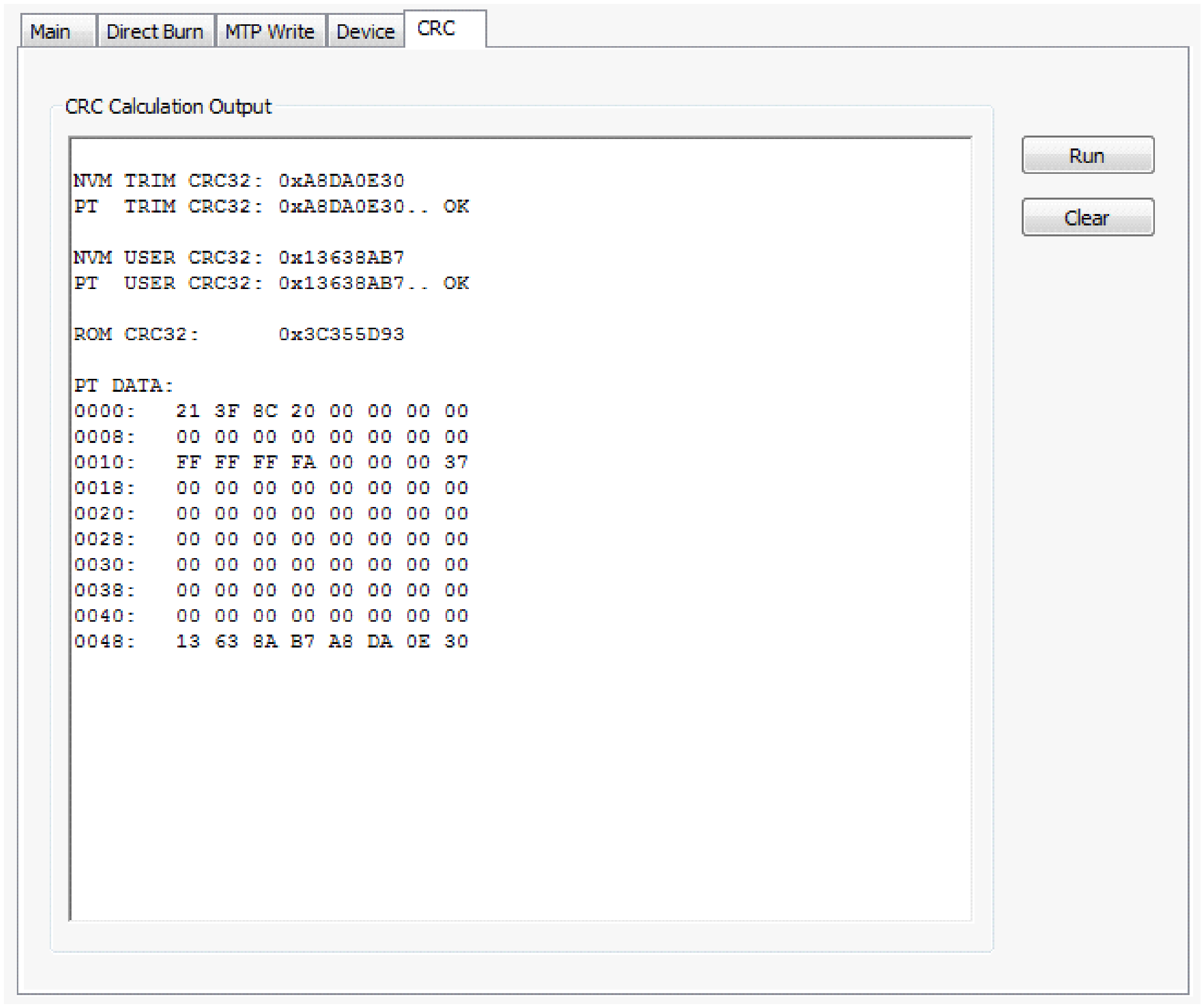
The CRC calculation compares expected and calculated CRC values. For more information on CRC burn flow, refer to “AN674: Si4010 NVM Burning Tools and Flows”.

The following CRC values are listed:

- NVM TRIM CRC32: Just calculated CRC over the Factory region of NVM.
- PT TRIM CRC32: Expected CRC over the Factory region of NVM. This value is factory programmed to the device by Silicon Labs. Result of the CRC comparison can be either “OK” or “DIFF”.
- NVM USER CRC32: Just calculated CRC over the User region of NVM. It is all zero if no CRC.
- PT USER CRC32: Expected CRC over the User region of NVM. This value is programmed to the device by the user when using the CRC burn flow. Result of the CRC comparison can be either “OK” or “DIFF”.

**Note:** PT USER CRC32 cannot be written by the NVM Programming Utility. See AN674.

- ROM CRC32 (0x3C355D93): Just calculated CRC over ROM. The value 0x3C355D93 corresponds to revC die currently in production. This value would change only if there is a ROM respin.



```
NVM TRIM CRC32: 0xA8DA0E30
PT  TRIM CRC32: 0xA8DA0E30.. OK

NVM USER CRC32: 0x13638AB7
PT  USER CRC32: 0x13638AB7.. OK

ROM CRC32:      0x3C355D93

PT DATA:
0000:   21 3F 8C 20 00 00 00 00
0008:   00 00 00 00 00 00 00 00
0010:   FF FF FF FA 00 00 00 37
0018:   00 00 00 00 00 00 00 00
0020:   00 00 00 00 00 00 00 00
0028:   00 00 00 00 00 00 00 00
0030:   00 00 00 00 00 00 00 00
0038:   00 00 00 00 00 00 00 00
0040:   00 00 00 00 00 00 00 00
0048:   13 63 8A B7 A8 DA 0E 30
```

# 4. NVM Composer

## 4.1. Operation Modes

The NVM composer works in two mutually exclusive modes, which have corresponding tabs on the GUI:

- **Main (regular) mode**: This mode is a regular composer mode and the most common mode of operation for a user. It takes user input files, IntelHEX or Verilog MEM files, and other flags and NVM addresses, and generates the proper NVM burn file (.nbf). The addresses in the input files are the addresses in the RAM where the user wants the data to be after the boot or user copy. In other words, the user supplies input files with data locations where those should be after boot or after overlay load. The composer will decide how that information maps into the NVM.
- **Direct burn mode**: This is advanced mode. It takes either a direct string of NVM address and data in Verilog MEM format, or a Verilog MEM file with direct NVM address and data specified. The format uses hex numbers without a 0x prefix. The input addresses and data are the actual direct NVM addresses and data. This mode will be used only for advanced purposes, like burning a part specific key after the whole load was programmed into the NVM.

## 4.2. Compose Mode for Burn

When an NVM bit is not burnt (programmed), it has logic value 0. The programmed bit has logic value 1.

Before composing, the user has to decide how the burn is going to be done. There are two burn modes which the composer can generate:

- **Strict**: In this mode, the existing value of each NVM bit is checked immediately before it is programmed. If 0 is to be programmed to the existing bit value of 1, the burner program returns error at runtime as “error on conflict” since it is not possible to program 0 into the existing bit of 1.
- **Logic OR**: There will be no bit conflict error during programming. The resulting new value of the NVM bit is the logic OR between the current value of the NVM bit and the new (to be programmed) value of the bit. If the bit is already programmed as 1 and the user desires to program it to 0, nothing is done to the bit and there is no error. This mode allows to program additional bits in already existing bytes without a need to know the current value of other bits in the byte.

The User must decide before composing which mode to use, **Strict** or **Logic OR**.

## 4.3. Output File Format

The composer generates a single output **NVM burn file** with default ***.nbf** extension. The file consists of several sections and contains all the information needed to burn the data into NVM. The file can also be loaded to the NVM GUI at a later time and used for NVM burning without running the composer again. That is beneficial in the situations when the user wants to burn the same data into many chips.

The NVM burn file from the composer is then loaded back into the NVM GUI and parsed.The **[Compose Map]** is displayed for user information. During the **Burn** process the content of other sections of the file are used to actually program the NVM.

## 4.4. Input File Formats and Extensions

**Important**: The composer requires there be no spaces in any paths or file names.

The composer understands two types of input files, distinguished by required file extensions:

- **IntelHEX**: Required extension ***.hex**.
  It is a standard, 16-bit address IntelHEX file.
- **Verilog MEM file**: Required extension ***.mem**
  This file has Verilog memory hex file format. The exception from the format is that the comment allowed in the file starts with "**//**" and whatever follows is comment to the end of the line. Comments delimited by compound C characters "**/* … */**" are not allowed.
  The file format is hexadecimal **@addr** destination address followed by one or more byte values **byte byte byte** on a single or multiple line with address being incremented with each byte until the next **@addr** is encountered. The hex values do not use **0x** prefix nor any other prefix. The address and byte data are separated by white spaces or end of lines. For example (the letter case does not matter, capital hex symbols used in the example by choice):

```
@0003 15 A4 3E
7E 56 @0015 89 F5 CD
89
AB
@10F4
DF C7 A4
```

## NOTES:

## Simplicity Studio

One-click access to MCU tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

*www.silabs.com/simplicity*

**MCU Portfolio**
*www.silabs.com/mcu*

**SW/HW**
*www.silabs.com/simplicity*

**Quality**
*www.silabs.com/quality*

**Support and Community**
*community.silabs.com*

**Silicon Laboratories Inc.**
**400 West Cesar Chavez**
**Austin, TX 78701**
**USA**

**http://www.silabs.com**