Using NUC1262 CRC to Verify Code Integrity

Example Code Introduction for 32-bit NuMicro® Family

Information

Application	This example code uses NUC1262 CRC for self-verification of code integrity.
BSP Version	NUC1262_Series_BSP_CMSIS_V3.00.001
Hardware	NuMaker-NUC1262SE V1.0

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1 Overview

This example code uses NUC1262 CRC to perform self-verification of code integrity. Ensuring the integrity of the execution code is one of the security conditions that the terminal product should have. This can avoid abnormal results and crashes caused by incorrect code during the execution of the product, or data theft due to code tampering.

This example code does not require additional external hardware modules. Users need to download the SRecord toolkit (http://srecord.sourceforge.net/) and use *srec_cat.exe* to calculate the CRC32 checksum of the original code, generate a new binary file containing the original code and its CRC32 checksum. Then convert this binary file into HEX format. This HEX file can be programmed into the chip through Keil IDE directly.

1.1 Principle

Since this example code does not require additional external hardware modules, the following section will focus on the steps to verify the CRC32 checksum of the original code, and instructions on how to use *srec_cat.exe* to generate a file containing the CRC32 checksum in this software project.

The memory map of the original application code and its CRC32 checksum are shown in Figure 1-1.

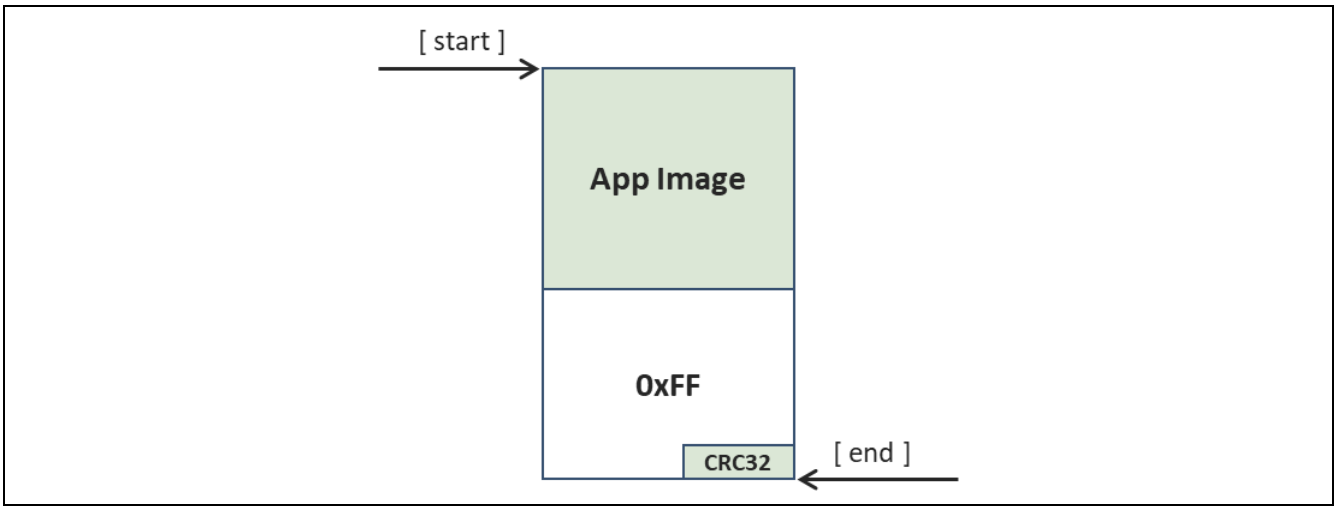

Figure 1-1 Memory Map of Application and CRC32

Figure 1-2 shows the steps to verify the CRC32 checksum of the original code.

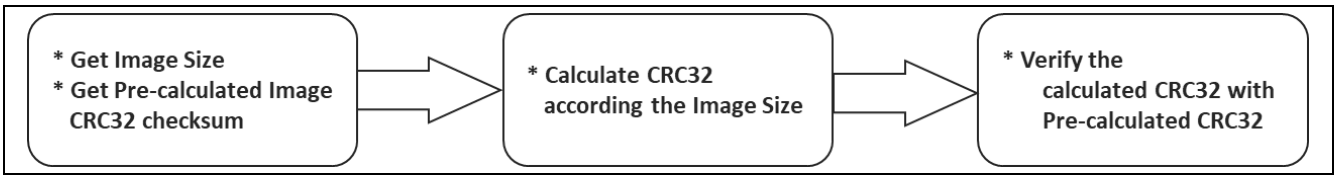

Figure 1-2 Get, Calculate and Verify CRC32 Checksum

[Get Image Size]:

The number of bytes of the original code can be calculated from the length of ER_RO and ER_RW (Refer to *NUC1262_Verify_Image_CRC32.map* file).

[Get Pre-calculated Image CRC32 checksum]:

The checksum of the original code can be read through the pre-defined address in *main.c*, IMAGE_CRC32_ADDRESS.

[Calculate CRC32 according the Image Size]:

Calculate the CRC32 checksum based on the number of bytes of the original code.

[Verify the calculated CRC32 with Pre-calculated CRC32]:

Check whether the calculated CRC32 is the same as the pre-calculated CRC32.

This example code uses *GenCRC.bat* to execute related commands of *srec_cat.exe*. The content of *GenCRC.bat* and the configuration in the project are shown in Figure 1-3.

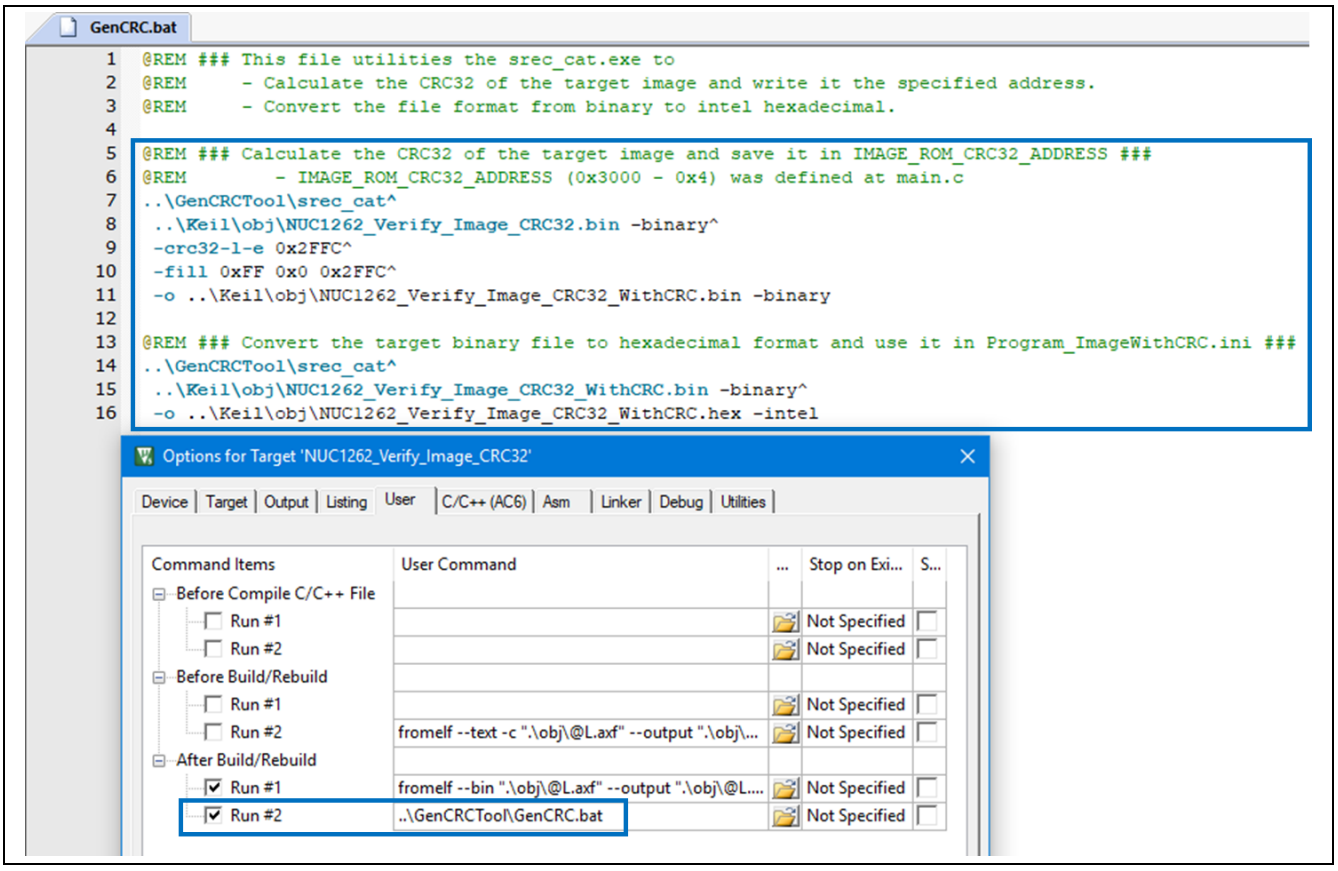

Figure 1-3 GenCRC.bat Content and Configuration

After compiling the project and generating the binary format file of the original code, the first srec_cat command in GenCRC.bat will be executed. This command will first calculate the CRC32 checksum of the original code, and assign the checksum to the specified address (for example, 0x2FFC), and then generate a new binary file containing the original code and its CRC32 checksum. Then, the reserved area after the original code and between the stored checksum will be assigned as all 0xFF. The second srec_cat command is to convert the new binary file into an intel HEX format file. The purpose is to directly program the original code and its checksum to the specified location by Keil IDE. The file used to implement Keil IDE programming is *ProgramTargetImageWithCRC.ini*, and its content and configuration in the project are shown in Figure 1-4.

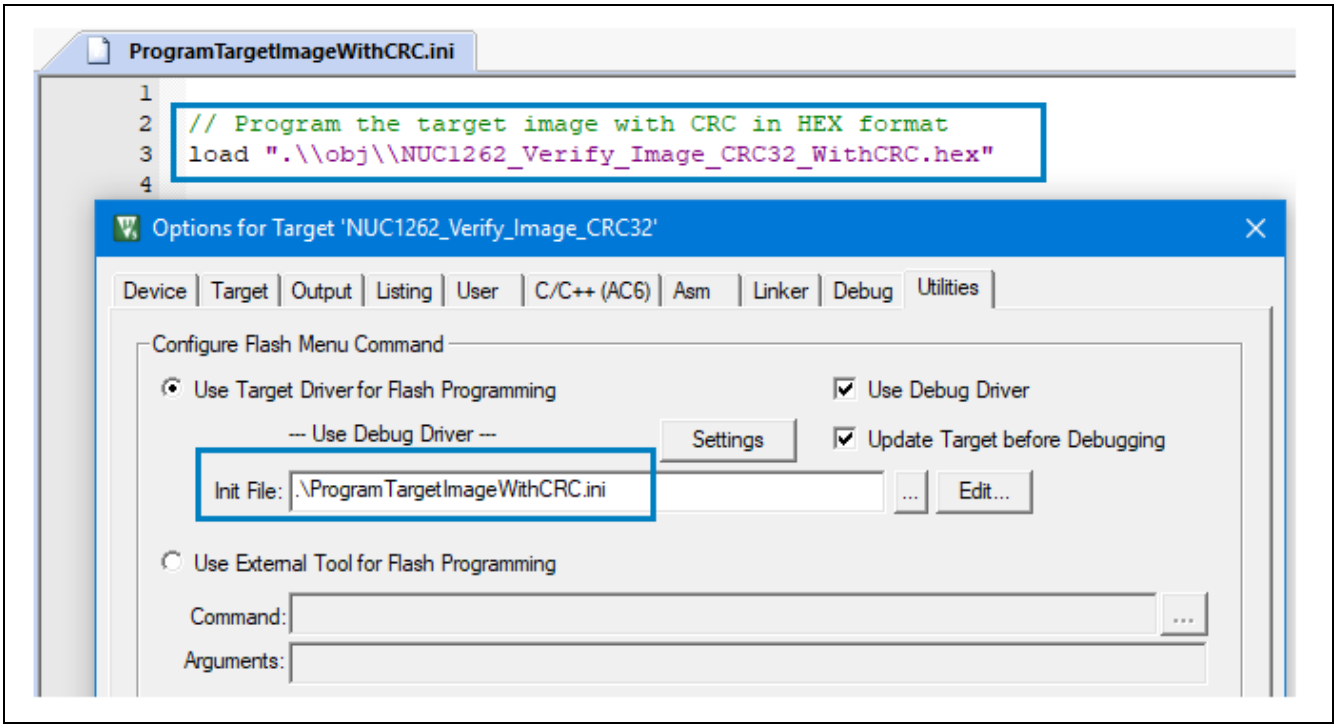

Figure 1-4 ProgramTargetImageWithCRC.ini Content and Configuration

1.2 Demo Result

The compilation and programming results of this example code are shown in Figure 1-5, which shows the information of executing the srec_cat command twice in *GenCRC.bat* and programming the HEX file specified in *ProgramTargetImageWithCRC.ini*.

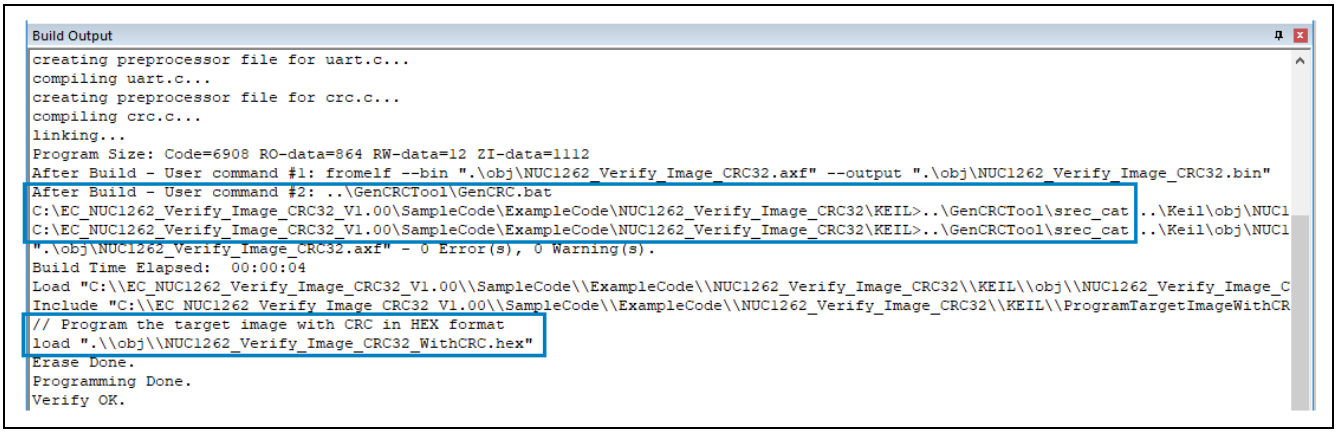

Figure 1-5 Compilation and Programming Results

As shown in Figure 1-6, the number of original code data bytes 0x1e68 and the CRC32 checksum 0x129c3e3f can be found in the *NUC1262_Verify_Image_CRC32_WithCRC.bin*.

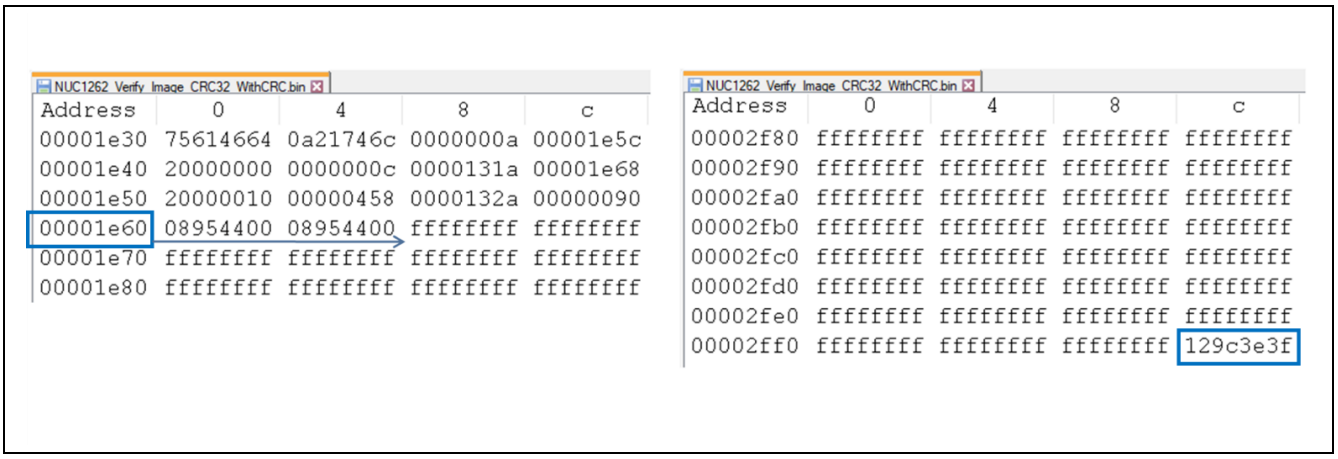

Figure 1-6 Code Length and CRC32 Checksum

The execution results will be printed out by the PC UART of the original code byte count, CRC32 checksum and the calculated checksum according to the original code byte count, as shown in Figure 1-7.

```
[ Perform CRC32 calculation to verify image integrity ]
    # CPU @ 72000000 Hz.
    # The target Image ROM size is 0x1e68 (7784).
    # The pre-calculated CRC32 value 0x129c3e3f is stored in 0x00002ffc.

Start to perform HW CRC to claculate the Image ROM CRC32 checksum......
        - CRC32 checksum: 0x129c3e3f

[Compare Image ROM checksum ... PASS]
```

Figure 1-7 Execution Results on PC UART

2 Code Description

A function to calculate the number of bytes of the original code. The code is located in *main.c*.

```
/**
  * @brief      Get Image Size
  * @param[in]  None
  * @return     Image ROM size (bytes)
  * @details    This function gets the target image size according to the Image RO and RW size.
  */
static uint32_t GetImageSize(void)
{
    extern uint32_t Image$$ER_RO$$Length; // Execution Region ER_RO can be found in NUC1262_Verify_Image_CRC32.map
    extern uint32_t Image$$ER_RW$$Length; // Execution Region ER_RW can be found in NUC1262_Verify_Image_CRC32.map
    uint32_t u32ROSize, u32RWSize;

    u32ROSize = (uint32_t)&Image$$ER_RO$$Length;
    u32RWSize = (uint32_t)&Image$$ER_RW$$Length;

    //printf("\tRO: %d, RW: %d.\n", u32ROSize, u32RWSize);

    return (u32ROSize + u32RWSize);
}
```

After initializing the system, the original code CRC32 checksum can be obtained through the IMAGE_CRC32_ADDRESS address, and then compared with the checksum calculated by the CRC module. The code is located in *main.c*.

```
    ……
 #define IMAGE_CRC32_ADDRESS         (0x3000 - 0x4) // 0x2FFC, the flash address to store pre-calculated CRC32
    ……
int main(void)
{
    volatile uint32_t addr;
    uint32_t u32ImageROMSize;
    uint32_t u32PreChks, u32CalChks;

    /* Unlock protected registers */
```

```
    SYS_UnlockReg();

    /* Init System, peripheral clock and multi-function I/O */
    SYS_Init();

    /* Init UART0 for printf */
    UART0_Init();

    /* Get the target image size and pre-calculated CRC32 checksum */
    u32ImageROMSize = GetImageSize();
    u32PreChks      = inpw(IMAGE_CRC32_ADDRESS);

    printf("\n\n[ Perform CRC32 calculation to verify image integrity ]\n");
    printf("    # CPU @ %d Hz.\n", SystemCoreClock);
    printf("    # The target Image ROM size is 0x%x (%d).\n", u32ImageROMSize,
u32ImageROMSize);
    printf("    # The pre-calculated CRC32 value 0x%08x is stored in 0x%08x.\n",
u32PreChks, IMAGE_CRC32_ADDRESS);
    printf("\n");



    printf("Start to perform HW CRC to claculate the Image ROM CRC32 checksum......\n");

    /* Configure CRC controller for CRC32 calculation */
    CRC_Open(CRC_32, (CRC_WDATA_RVS | CRC_CHECKSUM_RVS | CRC_CHECKSUM_COM), 0xFFFFFFFF,
CRC_CPU_WDATA_32);

    /* Calculate the Image ROM CRC32 checksum */
    for(addr=0; addr<u32ImageROMSize; addr+=4)
    {
        CRC_WRITE_DATA(inpw(addr));
    }
    u32CalChks = CRC_GetChecksum();
    printf("\t- CRC32 checksum: 0x%08x\n", u32CalChks);

    if(u32PreChks == u32CalChks)
    {
        printf("\n[Compare Image ROM checksum ... PASS]\n");
    }
    else
    {
        printf("\n[Compare Image ROM checksum ... WRONG]\n");
```

```
        printf("\tPre-calculated checksum:  0x%08x\n", u32PreChks);
        printf("\tCalculate CRC32 checksum: 0x%08x\n", u32CalChks);
    }

    /* Disable CRC function */
    CRC->CTL &= ~CRC_CTL_CRCEN_Msk;

    while(1) {}
}
```

3 Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - NUC1262_Series_BSP_CMSIS_V3.00.001
- IDE version
  - Keil uVersion 5.28

3.2 Hardware Requirements

- Circuit components
  - NuMaker-NUC1262SE V1.0
- Pin Connect
  - Connect the UART0 TX (PB.13) pin to the PC UART RX to show the execution results of this example code.

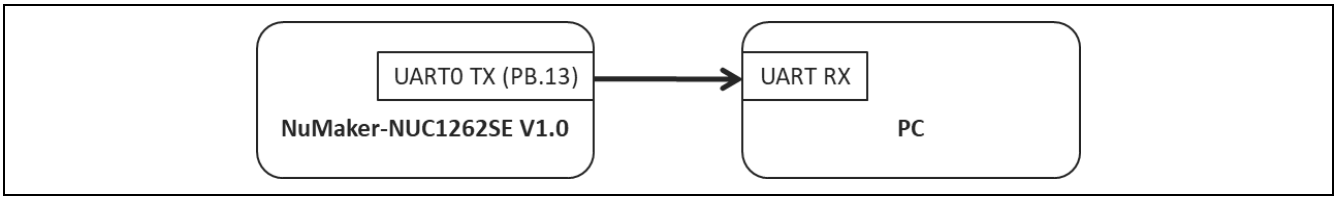

Figure 3-1 Pin Connect

4 Directory Information

The directory structure is shown below.

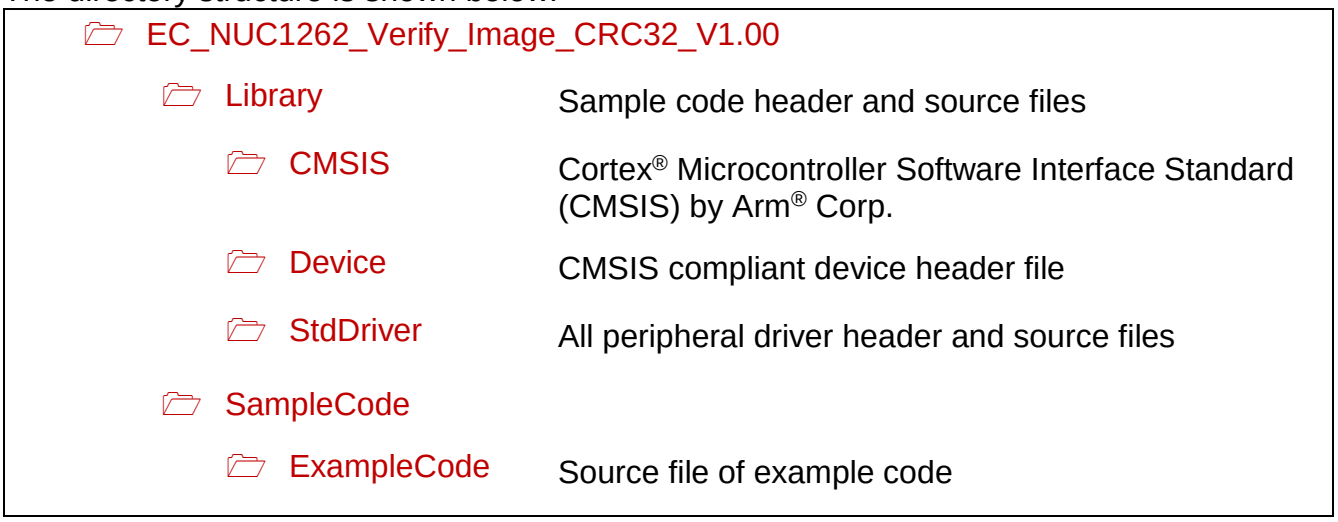

Figure 4-1 Directory Structure

5 Example Code Execution

1. Browse the sample code folder as described in the Directory Information section and double-click NUC1262_Verify_Image_CRC32.uvprojx.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6 Revision History

Date	Revision	Description
2021.09.24	1.00	1. Initially issued.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.