Application Note

Flash Development Toolkit

Application Note (Applications)

User Program Mode (H8/3694F)

Renesas Electronics
www.renesas-electoronics.com

Rev.1.00 2006.06

Renesas Flash Development Toolkit

Application Note (Applications)

User Program Mode (H8/3694F)

Revision 1.0

Renesas Technology Corp.

Contents

1. Introduction

This application note describes the following items with respect to the use of the Flash Development Toolkit and the use of the user program mode (user mode) of the H8/3694F (H8/300H Tiny Series) using the Flash Development Toolkit:

(1) Boot mode (programming the user area)

(2) User program mode (user mode)

Read the explanation of these items to understand differences between the boot mode and the user program mode and understand the user program mode.

This application note describes the above items using a sample program created by referencing a boot mode control program. This sample program programs and erases on-chip flash memory. To program or erase flash memory in the user program mode, refer to this sample program.

2. H8/3694F (H8/300H Tiny Series)

2.1 Flash Memory Configuration

The flash memory version of the H8/3694F incorporates 32-Kbyte flash memory. In addition, it has an area for containing a flash memory programming and erasing control program. This application note calls the area containing the control program the boot area and flash memory the user area. The flash memory configuration is shown in Table 2-1.

Table 2-1 Flash Memory Configuration

Area	Type	Size	Blocks
Boot area	Control program	–	–
User area	Flash memory	32 Kbytes	5 blocks Four 1-Kbyte blocks One 28-Kbyte block

2.2 Programming Modes

The following two modes are available to program and erase flash memory: The boot mode which enables on-board programming/erasing operations and the programmer mode which enables programming/erasing operations using a PROM programmer. In addition to the above modes, the user program mode enables on-board programming/erasing operations. When the H8/3694F is started from the reset state, it enters a mode depending on the input levels of the TEST and $\overline{\text{NMI}}$ pins and port as listed in Table 2-2. The input level of each pin must be set at least 4 states before the reset state is cleared.

When the LSI enters the boot mode, the boot program incorporated into the LSI starts up. The boot program transfers the programming control program from the externally connected Flash Development Toolkit to the on-chip RAM via SCI3, erases the entire flash memory, then executes the programming control program. The boot mode is available for initial programming in the on-board state and forced return when data cannot be programmed or erased in the user program mode.

In the user program mode, any desired block can be erased and programmed by causing a branch to a user-provided programming/erasing program.

For details, refer to the Hardware Manual.

Table 2-2 Programming Mode Selection

LSI Status After the Reset State Is Cleared		TEST	$\overline{\text{NMI}}$	P85	PB0	PB1	PB2
On-board programming modes	User program mode	0	1	X	X	X	X
	Boot mode	0	0	1	X	X	X
Programmer mode		1	X	X	0	0	0

Note: 1. X: Don't care

2.3 On-Board Programming Modes

There are two on-board programming modes: The boot mode and the user boot mode. On-board programming modes are listed in Table 2-3.

Table 2-3 On-Board Programming Modes

Item	Boot Mode	User Program Mode
Function	This mode is a program mode that uses an on-chip SCI interface. The user area can be programmed. This mode can automatically adjust the bit rate between the host and this LSI. The entire user area is erased first.	The user area can be programmed by using a desired interface.
Control program	Boot area (On-chip boot program)	User area (User-created user program)
Programming/erasing enable area	User area	User area
All erasure	✓ (Automatic)	✓
Block division erasure	✓*1	✓
Programming data transfer	From the host via the SCI	From a desired device via RAM
Reset start	On-chip boot program storage area (Boot area)	User area
Transition to the user program mode	Changing mode setting and reset	Changing the FLSHE bit setting

Note: 1. All-erasure is performed. After that, the specified block can be erased.

The entire user area is erased in the boot mode. Then, the user area can be programmed by commands. However, the contents of the area cannot be read until the entire erasing is done.

3. Functions of the Flash Development Toolkit

The Flash Development Toolkit is an on-board flash programming tool for Renesas F-ZTAT microcomputers, which offers a sophisticated and easy-to-use graphical user interface.

When it is used with Renesas High-performance Embedded Workshop (HEW), it allows users who develop embedded application software using Renesas F-ZTAT microcomputers to use an integrated environment.

The Flash Development Toolkit can also be used as an editor for S-record and hexadecimal files.

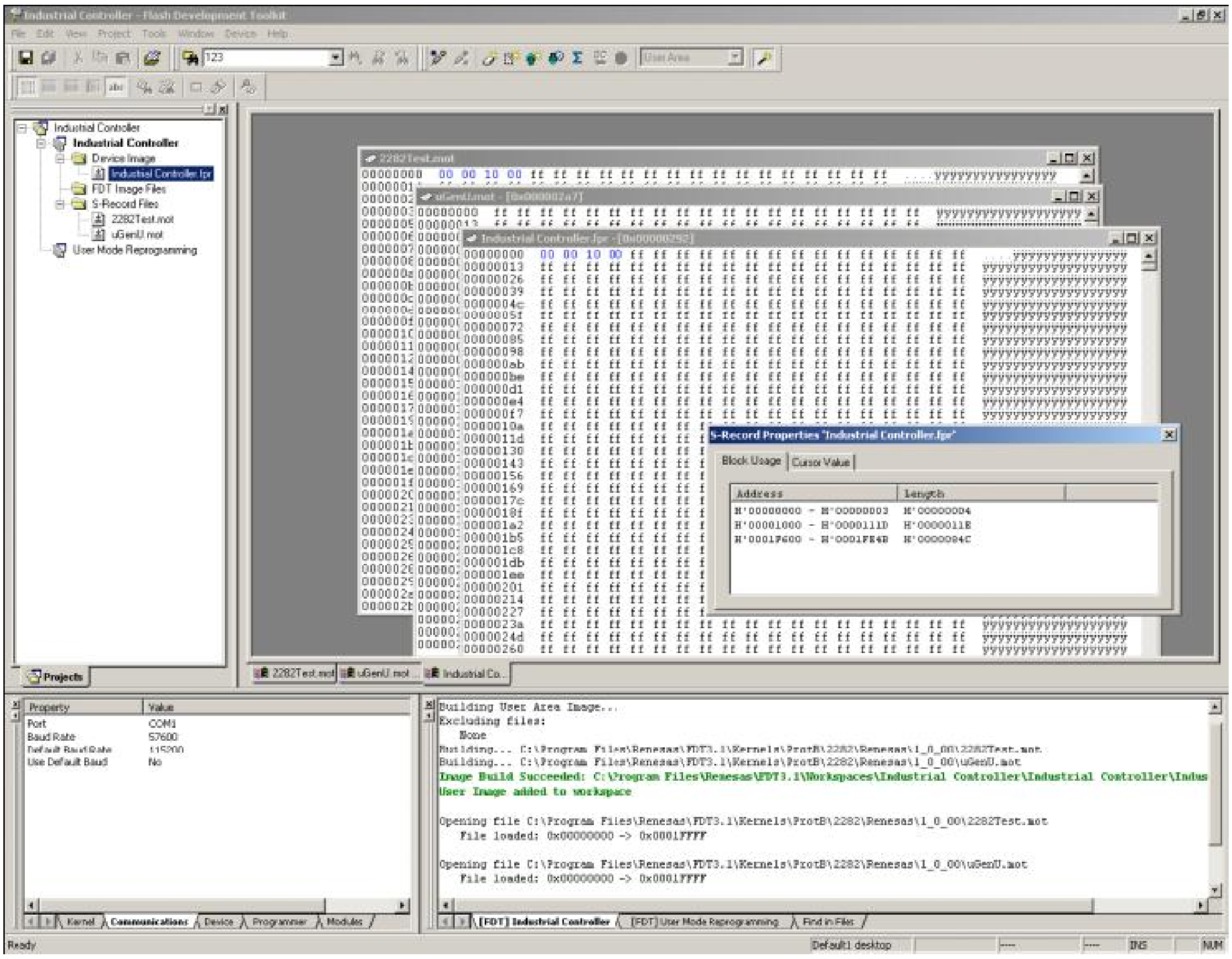
Note: F-ZTAT (Flexible-Zero Turn Around Time) is a trademark of Renesas Technology Corp.

3.1 Main Functions

- Connecting a device: Connects a device to the interface of the Flash Development Toolkit.
- Disconnecting the device: Disconnects the device from the interface of the Flash Development Toolkit.
- Erasing blocks: Opens the "Erase Block" dialog to erase all or individual blocks in flash memory on the device.
- Checking the blank status: Checks whether the flash section on the target device is blank.
- Uploading data: Uploads data from the target device.
- Downloading a target file: Downloads an active file using the hexadecimal editor.
- Returning a checksum: Returns a checksum of data in flash memory.
- Specifying a flash area: Sets a flash area in which non-programming (such as uploading and blank check) operations are to be performed.
- The Flash Development Toolkit is available in the simple interface mode and basic simple interface mode to facilitate the usability of the kit.

For details, refer to Renesas Flash Development Toolkit 3.4 User's Manual.

The graphical user interface screen of the Flash Development Toolkit is shown in Figure 3-1.



Address	Length
H'00000000 - H'00000003	H'00000004
H'00001000 - H'0000111D	H'0000011E
H'0001F600 - H'0001FE4B	H'0000084C

Property	Value
Port	COM1
Baud Rate	57600
Default Baud Rate	115200
Use Default Baud	No

```
Building User Area Image...
Excluding files:
   None
Building... C:\Program Files\Renesas\FDT3.1\Kernels\ProtB\2282\Renesas\1_0_00\2282Test.mot
Building... C:\Program Files\Renesas\FDT3.1\Kernels\ProtB\2282\Renesas\1_0_00\uGenU.mot
Image Build Succeeded: C:\Program Files\Renesas\FDT3.1\Workspaces\Industrial Controller\Industrial Controller\Indus
User Image added to workspace

Opening file C:\Program Files\Renesas\FDT3.1\Kernels\ProtB\2282\Renesas\1_0_00\2282Test.mot
   File loaded: 0x00000000 -> 0x0001FFFF

Opening file C:\Program Files\Renesas\FDT3.1\Kernels\ProtB\2282\Renesas\1_0_00\uGenU.mot
   File loaded: 0x00000000 -> 0x0001FFFF
```

Figure 3-1 Graphical User Interface of the Flash Development Toolkit

4. Operating the Flash Development Toolkit

4.1 Connecting the Adapter Board

On-board programming adapter board for F-ZTAT* microcomputers HS0008EAUF1H (called the adapter board hereafter), which is connected between a host computer and user system, has a function which can write a user application program in flash memory built into an F-ZTAT microcomputer on the user system (on-board) and erase it from the flash memory using the Flash Development Toolkit.

The adapter board connection is shown in Figure 4-1.

Note: F-ZTAT (Flexible-Zero Turn Around Time) is a trademark of Renesas Technology Corp.

Note: FDM (flash development module) is a former name of the adapter board.

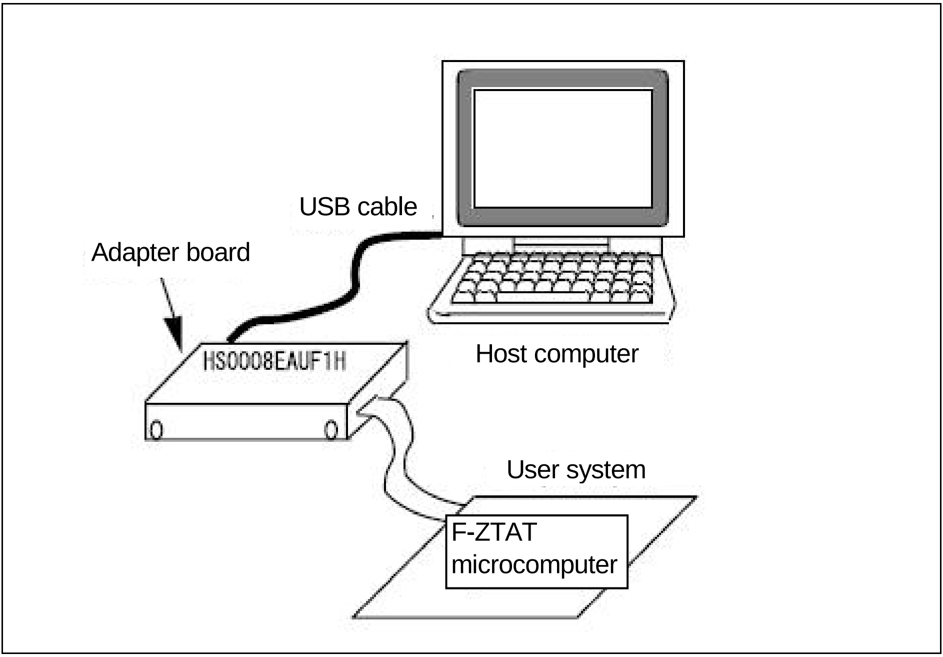


Figure 4-1 Connecting the Adapter Board

The pin numbers and corresponding signals of the user system interface cable used for connecting the adapter board and user system are listed in the following table.

Table 4-1 Pin Numbers and Corresponding Signals of the HS0008EAUF1H User System Interface Cable

No.	Signal Name	No	Signal Name
1	$\overline{\text{RES}}$	2	GND
3	FWx	4	GND
5	MD0	6	GND
7	MD1	8	GND
9	MD2 (IO0)	10	GND
11	MD3 (IO1)	12	GND
13	MD4 (IO2)	14	GND
15	RXD (TXD on the user system side)	16	GND*1
17	TXD (RXD on the user system side)	18	VIN (Vcc or PVcc)*2
19	SCK (NC)	20	VIN (PVcc)*2

Notes: 1. Be sure to connect pin No. 16 to GND to detect that the user system is connected properly.

2. For a device with Vcc and PVcc, be sure to supply Vcc or PVcc (pin No. 18) and PVcc (pin No. 20) to the VIN pins of the user interface connector, respectively. To use a device under condition Vcc = PVcc or when PVcc is not present in the device, be sure to supply Vcc to both VIN pins Vcc or PVcc (pin No. 18) and PVcc (pin No. 20).

Connecting the Adapter Board

An example of connecting the H8/3694F and Renesas adapter board (HS0008EAUF1H) is shown in Figure 4-2. The pull-up and pull-down resistor values shown are only examples. Evaluate the microcomputer to determine the actual values on the user system.

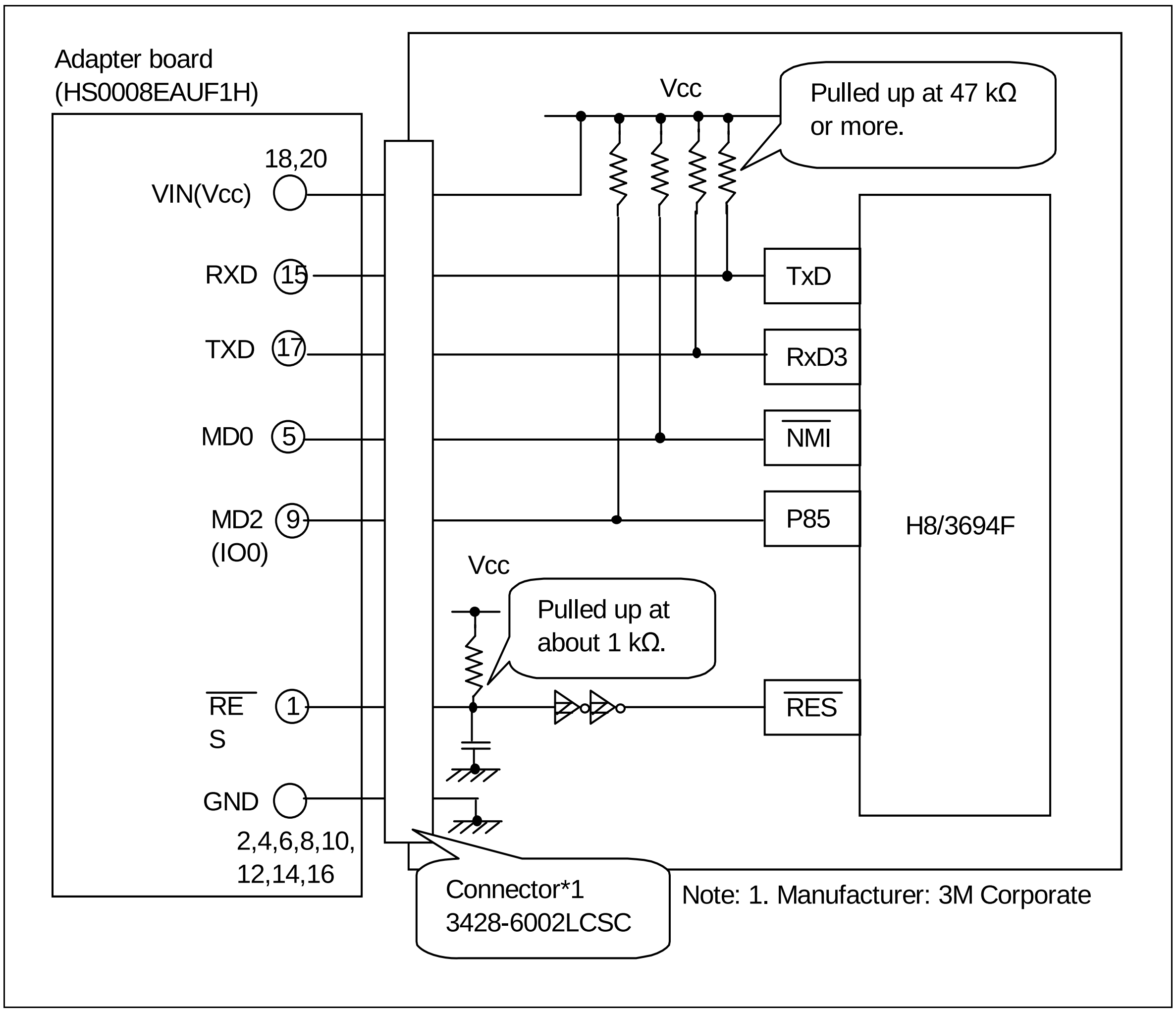


Figure 4-2 Example of Connecting the H8/3694F and Adapter Board

4.1.1 Setting Pins on the Adapter Board

An example of setting pins for the boot mode when the H8/3694F user system and Renesas adapter board (HS0008EAUF1H) is shown in Table 4-2.

Table 4-2 Example of Setting Pins on the H8/3694F and Adapter Board (for the Boot Mode)

Pin No.	Pin on the Adapter Board	Pin on the Device	Input/Output	Output Level
1	$\overline{\text{RES}}$	$\overline{\text{RES}}$	Output (default)	Adapter board
3	FWx	NC	NC	-
5	MD0	$\overline{\text{NMI}}$	Output	Low (0)
7	MD1	NC	NC	-
9	MD2 (IO0)	P85	Output	High (1)
11	MD3 (IO1)	NC	NC	-
13	MD4 (IO2)	NC	NC	-
15	RXD	TXD	Input (default)	Adapter board
17	TXD	RXD	Output (default)	Adapter board
19	SCK (NC)	NC	NC (default)	-

Note: NC: Means no connection.

Table 4-3 Programming Mode Selection

LSI Status After the Reset State Is Canceled		TEST	$\overline{\text{NMI}}$	P85	PB0	PB1	PB2
		−	MD0	MD2(IO0)	−	−	−
On-Board Programming Modes	User Program Mode	0	1	X	X	X	X
	Boot Mode	0	0	1	X	X	X
Programmer mode		1	X	X	0	0	0

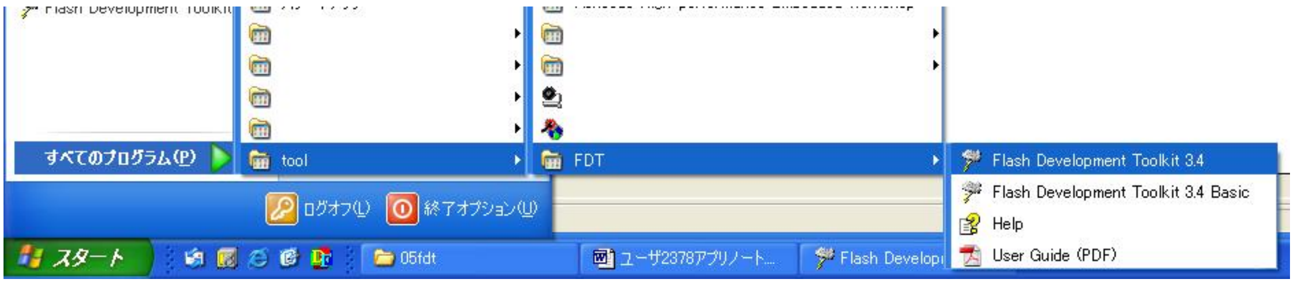
Note: X: Don't care

4.2 Setting the Flash Development Toolkit

Set the Flash Development Toolkit first to write a program in flash memory.

4.2.1 Starting the Flash Development Toolkit

From the "All Programs" menu, select "Flash Development Toolkit 3.4."

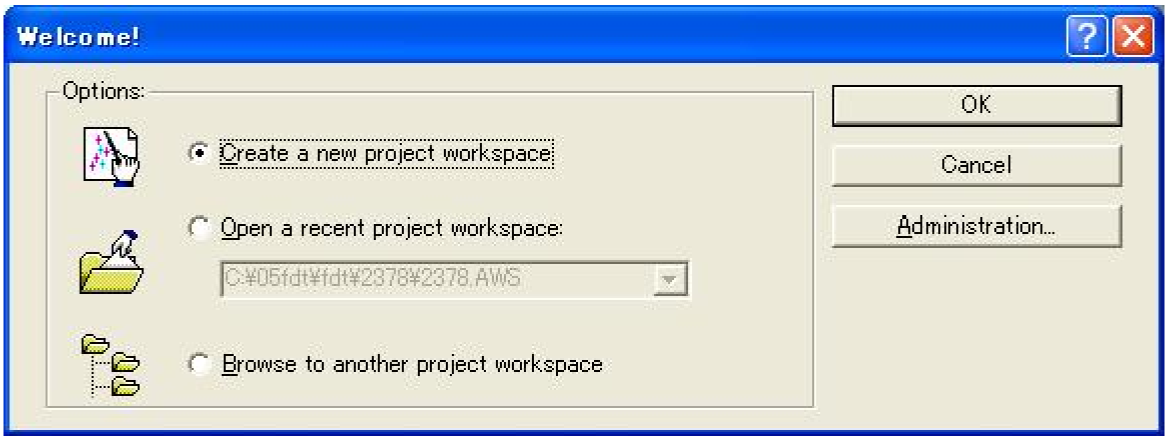


4.2.2 Selecting an Option

The "Welcome!" screen of the Flash Development Toolkit appears.

Select "Create a new project workspace."

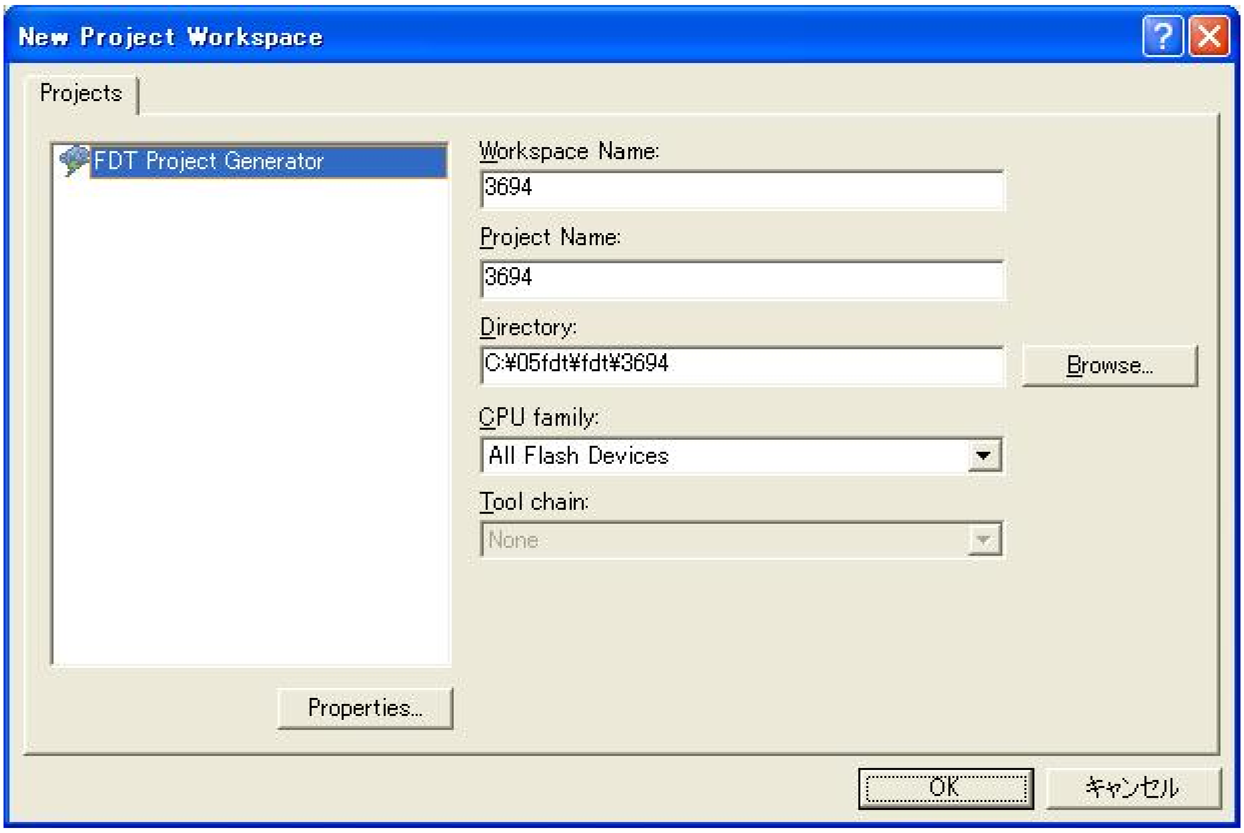
When the Flash Development Toolkit is started up for the second and subsequent times, the previously selected device and port information are retained. Select "Open a recent project workspace."


Welcome!
Options:
Create a new project workspace
Open a recent project workspace:
C:¥05fdt¥fdt¥2378¥2378.AWS
Browse to another project workspace
OK
Cancel
Administration...

When you have selected an option, click "OK."

4.2.3 Setting a New Project Workspace

Set a new project workspace. Use "Browse..." and select a directory, and specify the device name in "Workspace Name." Specify a project name if required. In this example, specify the same name in "Workspace Name:" and "Project Name:."



New Project Workspace

Projects

FDT Project Generator

Workspace Name: 3694

Project Name: 3694

Directory: C:¥05fdt¥fdt¥3694

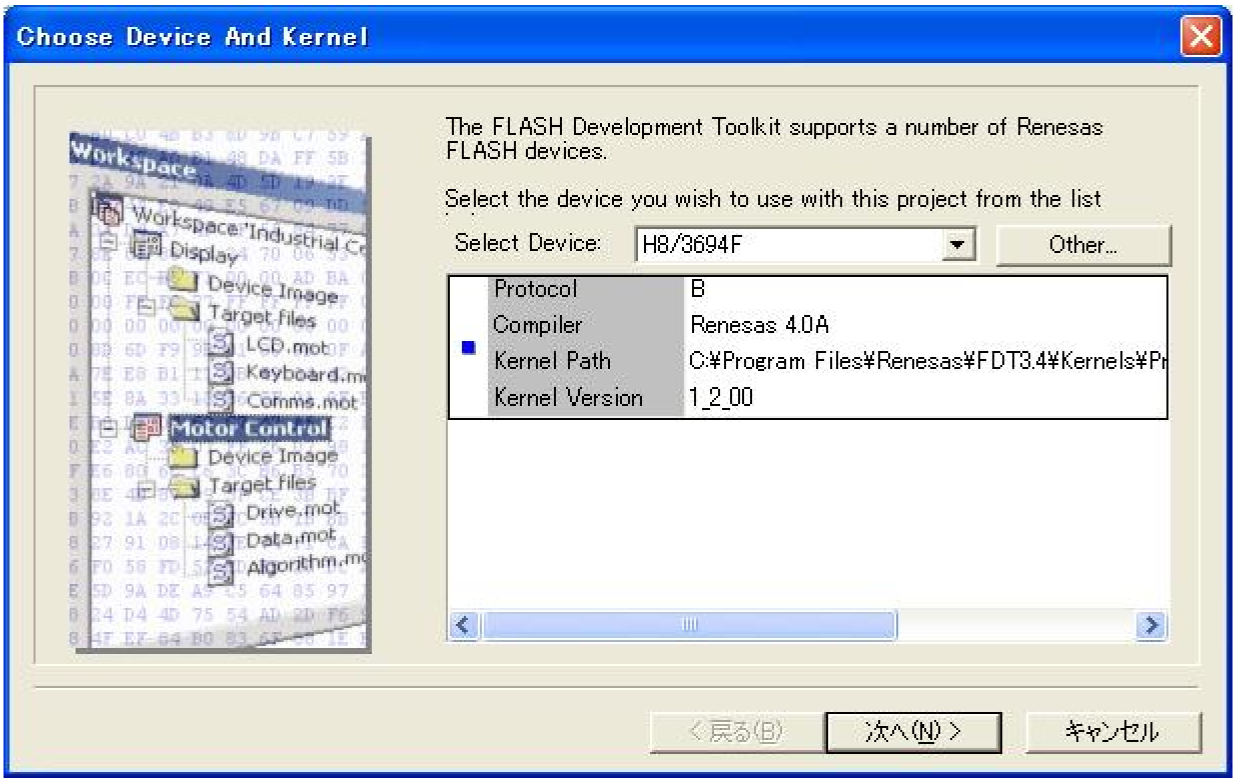
CPU family: All Flash Devices

Tool chain: None

When you have set the project workspace, click "OK."

4.2.4 Selecting the Device and Kernel

Select the target device from the pull-down menu. In this example, select H8/3694F.



The FLASH Development Toolkit supports a number of Renesas FLASH devices.

Select the device you wish to use with this project from the list

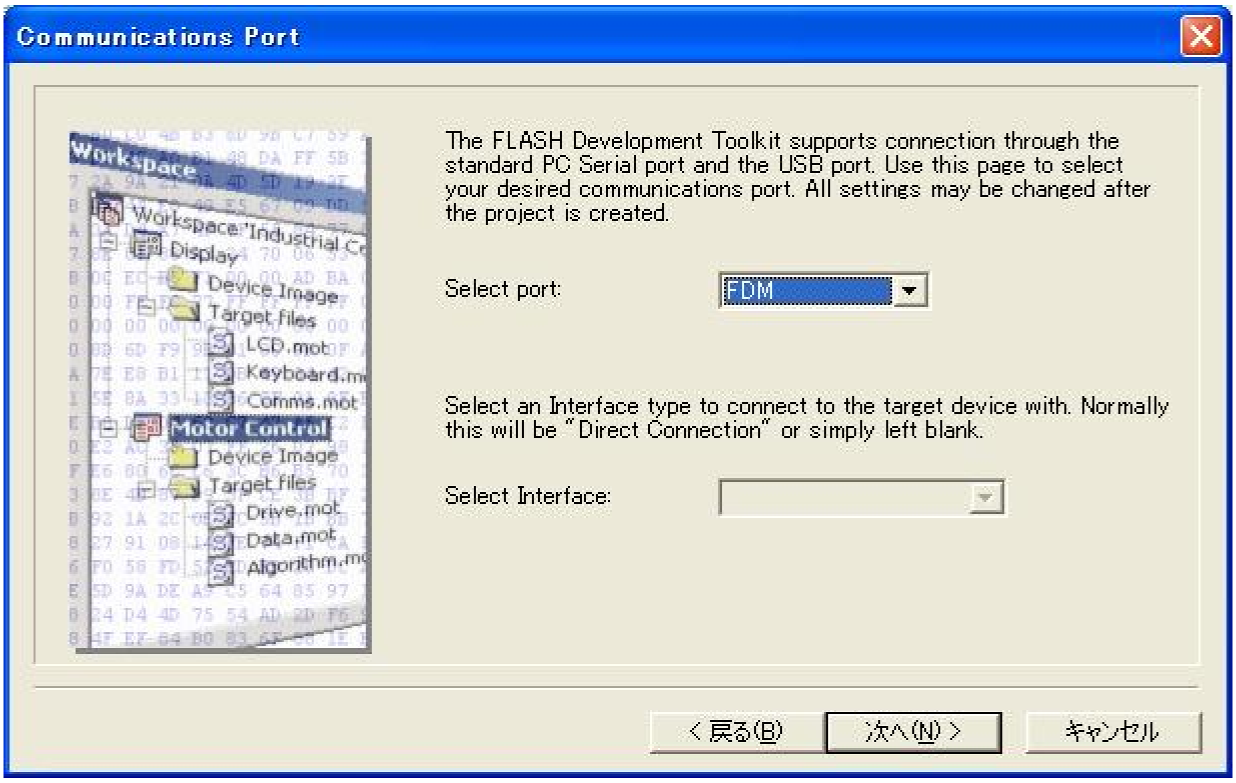
Select Device: H8/3694F

Protocol	B
Compiler	Renesas 4.0A
Kernel Path	C:¥Program Files¥Renesas¥FDT3.4¥Kernels¥Pr
Kernel Version	1_2_00

When you have set the device, click "Next."

4.2.5 Selecting a Communications Port

Select the adapter board (FDM) from the pull-down menu.



The FLASH Development Toolkit supports connection through the standard PC Serial port and the USB port. Use this page to select your desired communications port. All settings may be changed after the project is created.

Select port: FDM

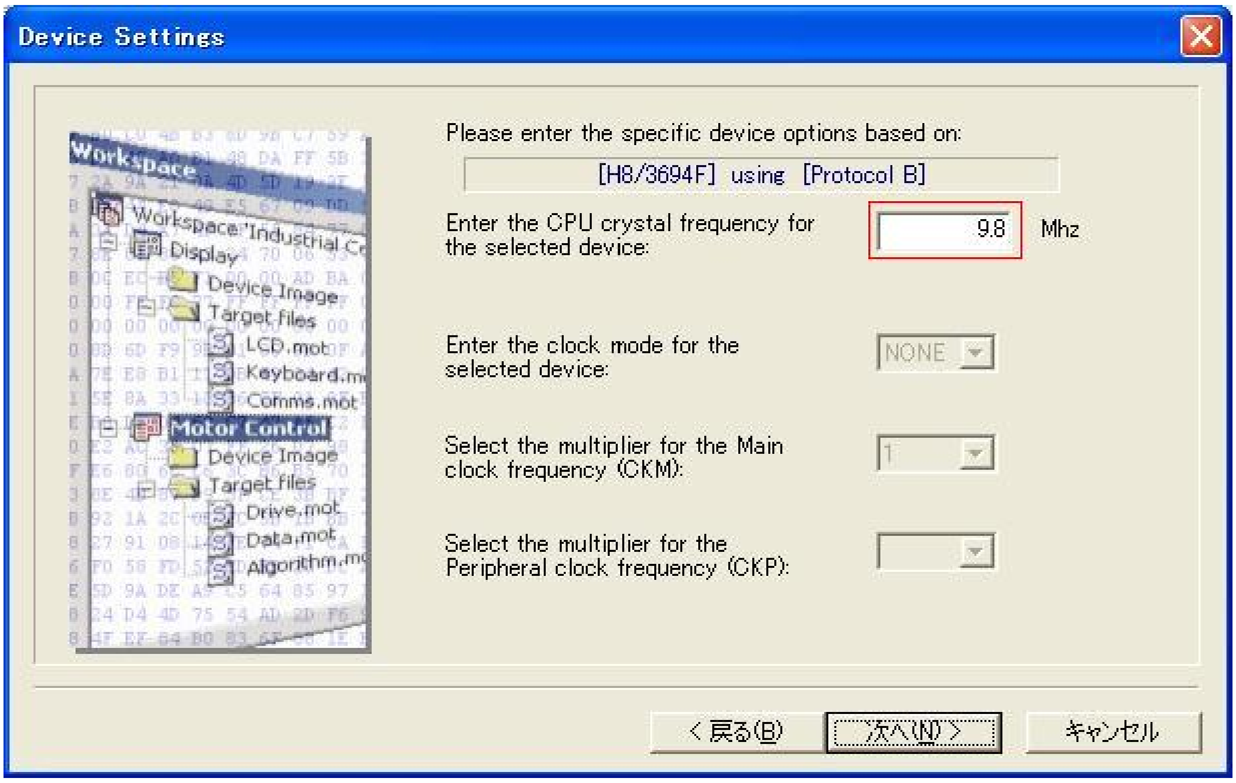
Select an Interface type to connect to the target device with. Normally this will be "Direct Connection" or simply left blank.

Select Interface:

When you have selected the communications port, click "Next."

4.2.6 Device Settings (Setting the Input Clock)

In the first column enter the frequency of the clock used for the board in MHz. For example, enter 9.8 (MHz).



Device Settings

Please enter the specific device options based on:

[H8/3694F] using [Protocol B]

Enter the CPU crystal frequency for the selected device: 9.8 Mhz

Enter the clock mode for the selected device: NONE

Select the multiplier for the Main clock frequency (CKM): 1

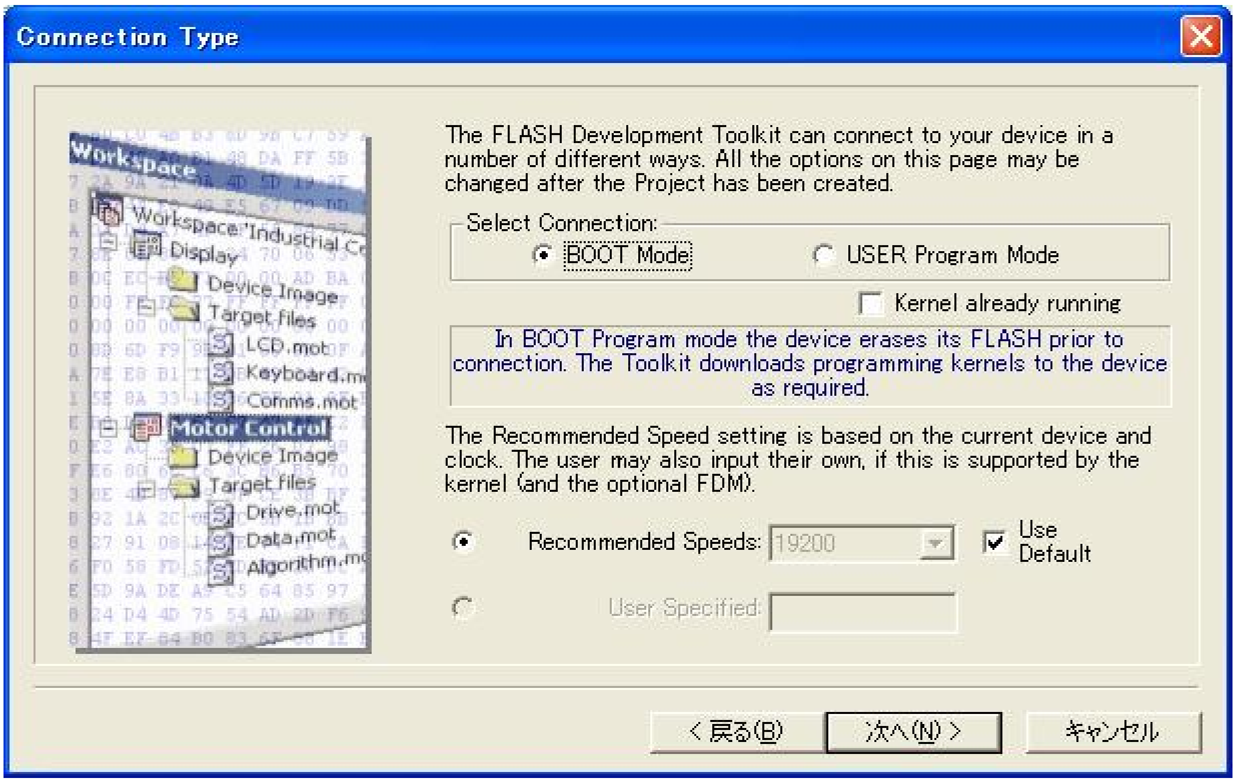
Select the multiplier for the Peripheral clock frequency (CKP):

When you have entered the value, click "Next."

The input clock is the frequency of the clock directly input to the microcomputer. Enter the frequency of the crystal or ceramic resonator connected to the user system with three significant digits. The input clock differs from the operating frequency (PLL output).

4.2.7 Selecting the Connection Type (Communication Speed)

Set the baud rate. For example, select "Use Default."



Connection Type

The FLASH Development Toolkit can connect to your device in a number of different ways. All the options on this page may be changed after the Project has been created.

Select Connection:

[✓ BOOT Mode] USER Program Mode

☐ Kernel already running

In BOOT Program mode the device erases its FLASH prior to connection. The Toolkit downloads programming kernels to the device as required.

The Recommended Speed setting is based on the current device and clock. The user may also input their own, if this is supported by the kernel (and the optional FDM).

[✓ Recommended Speeds: 19200] ☑ Use Default

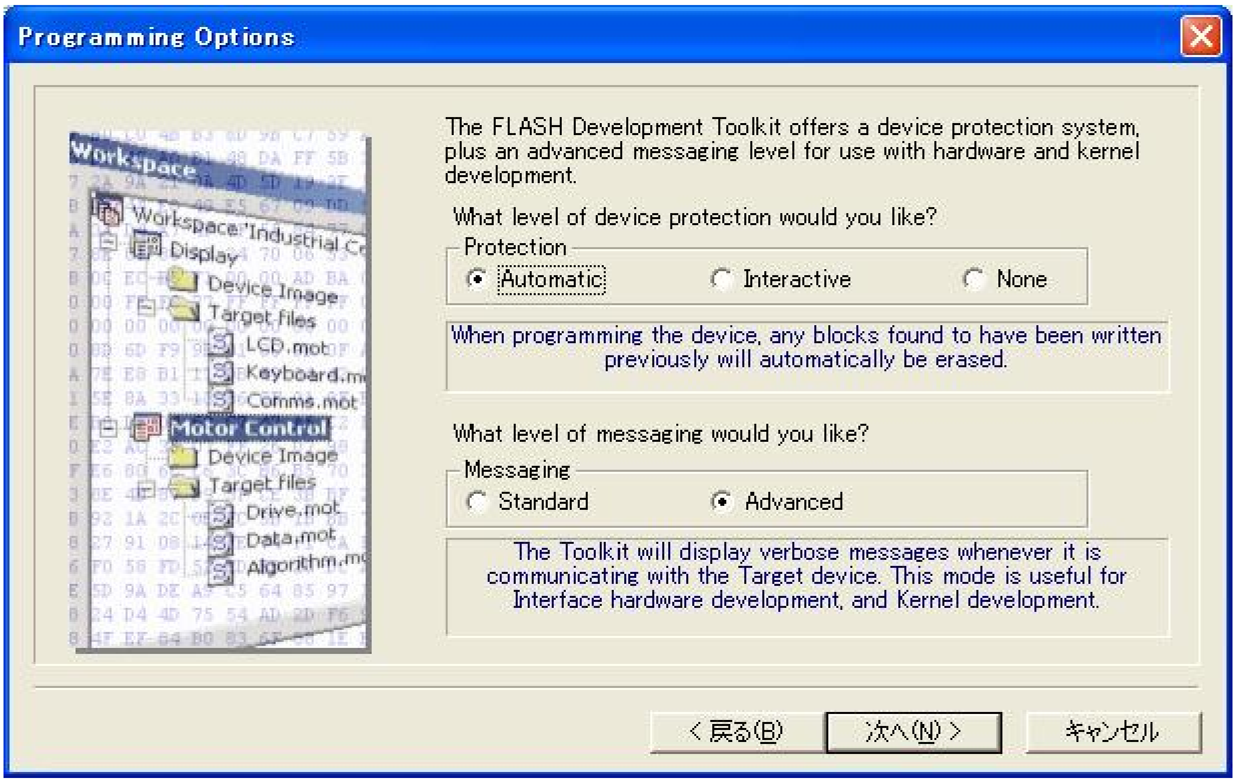
User Specified:

< 戻る(B) 次へ(N) > キャンセル

When you have selected the baud rate, click "Next."

4.2.8 Selecting Programming Options (Protection Level and Messaging Level)

Select the protection level and messaging level. For example, select "Automatic" for "Protection" and "Advanced" for "Messaging."



The FLASH Development Toolkit offers a device protection system, plus an advanced messaging level for use with hardware and kernel development.

What level of device protection would you like?

Protection: [✓ Automatic] Interactive None

When programming the device, any blocks found to have been written previously will automatically be erased.

What level of messaging would you like?

Messaging: Standard [✓ Advanced]

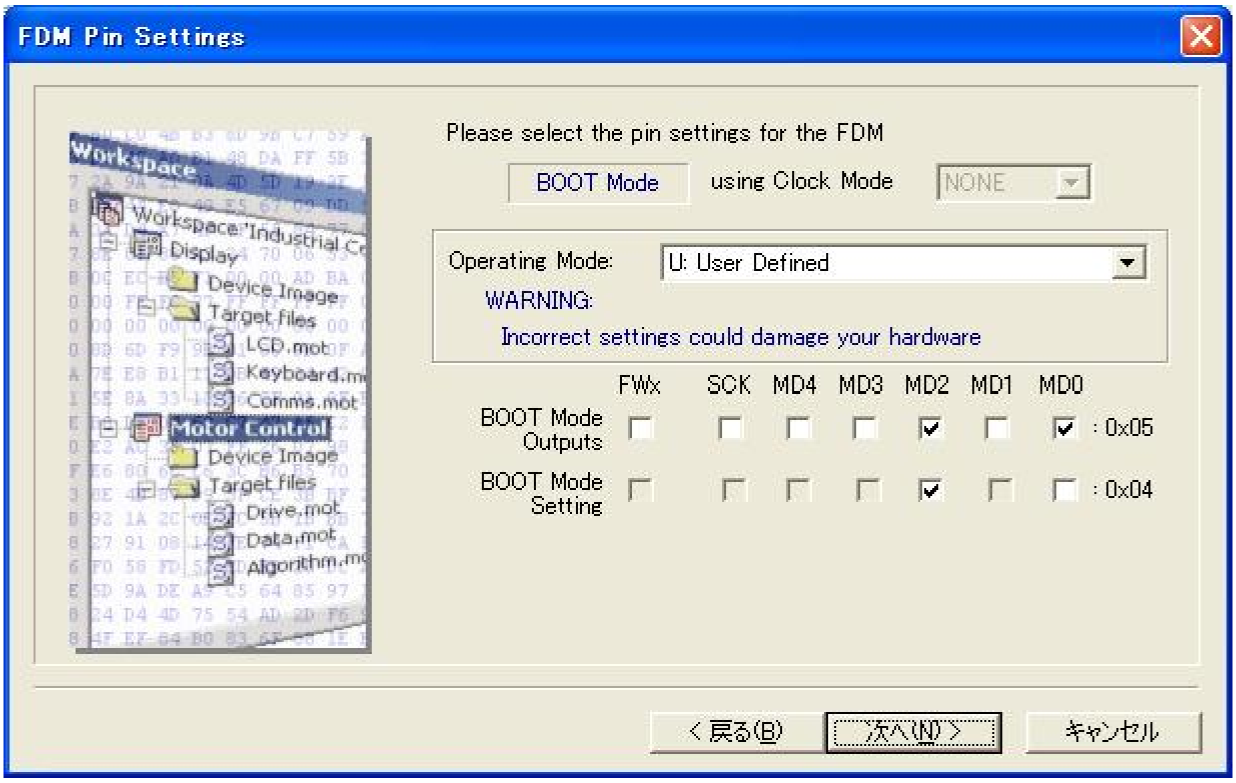
The Toolkit will display verbose messages whenever it is communicating with the Target device. This mode is useful for Interface hardware development, and Kernel development.

When you have selected programming options, click "Next."

4.2.9 Adapter Board Pin Settings

Set the pins on the adapter board (FDM) for the boot mode.

In the H8/3694F boot mode, set the output of P85 to high (1) and that of $\overline{\text{NMI}}$ to low (0). On the H8/3694F user system, MD2 (IO0) is connected to P85 and MD0 is connected to $\overline{\text{NMI}}$. For this reason, set the output of MD2 (IO0) to high (1) and that of MD0 to low (0). No FWE pin setting is required because no FWE pin is given.



Please select the pin settings for the FDM

BOOT Mode using Clock Mode NONE

Operating Mode: U: User Defined

WARNING:
Incorrect settings could damage your hardware

	FWx	SCK	MD4	MD3	MD2	MD1	MD0	
BOOT Mode Outputs					✓		✓	: 0x05
BOOT Mode Setting					✓			: 0x04

When you have set the pins, click "Next."

An example of connecting the H8/3694F and Renesas adapter board (HS0008EAUF1H) is shown in Figure 4-3. The pull-up and pull-down resistor values shown are only examples. Evaluate the microcomputer to determine the actual values on the user system.

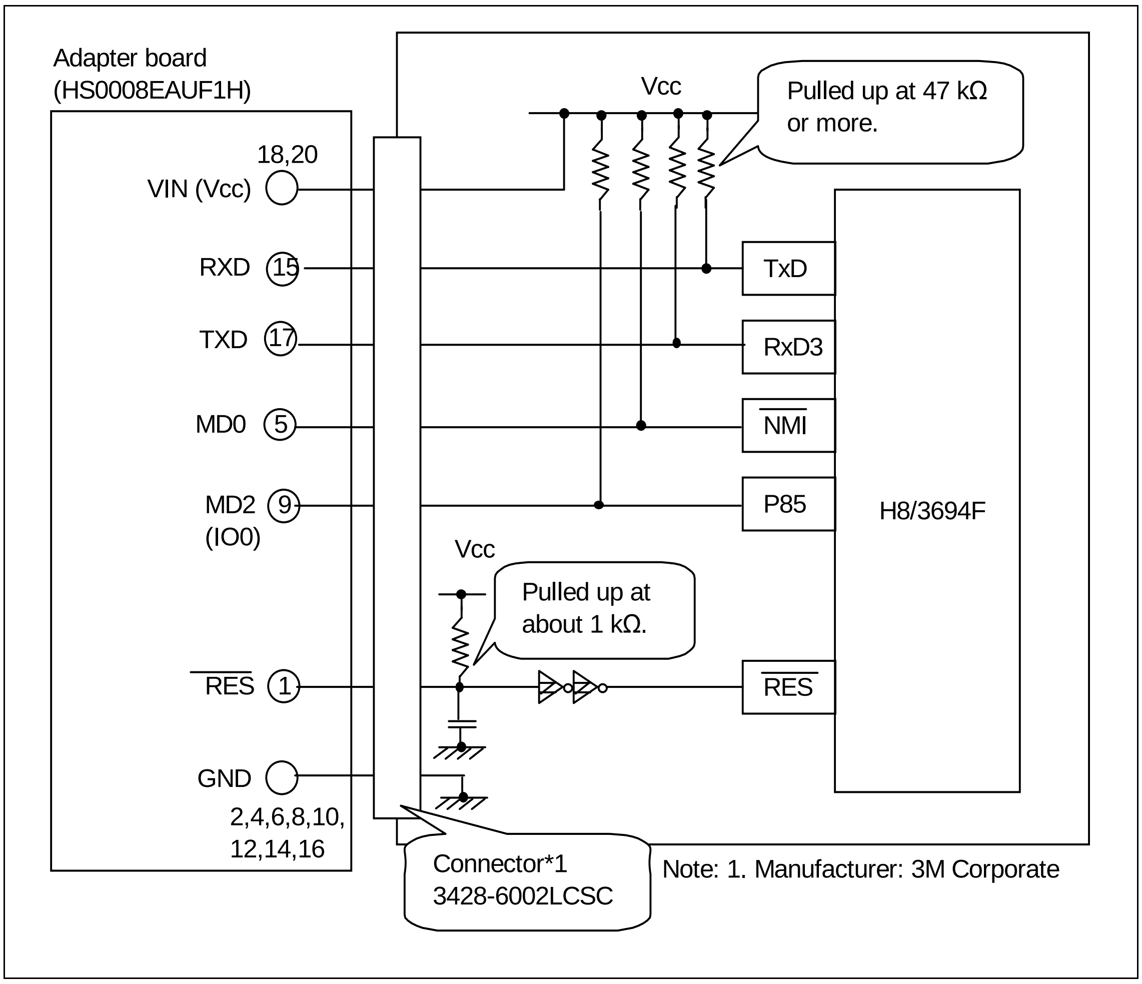


Figure 4-3 Example of Connecting the H8/3694F and Adapter Board

An example of setting pins for the boot mode when the H8/3694F user system and Renesas adapter board (HS0008EAUF1H) is shown in Table 4-4.

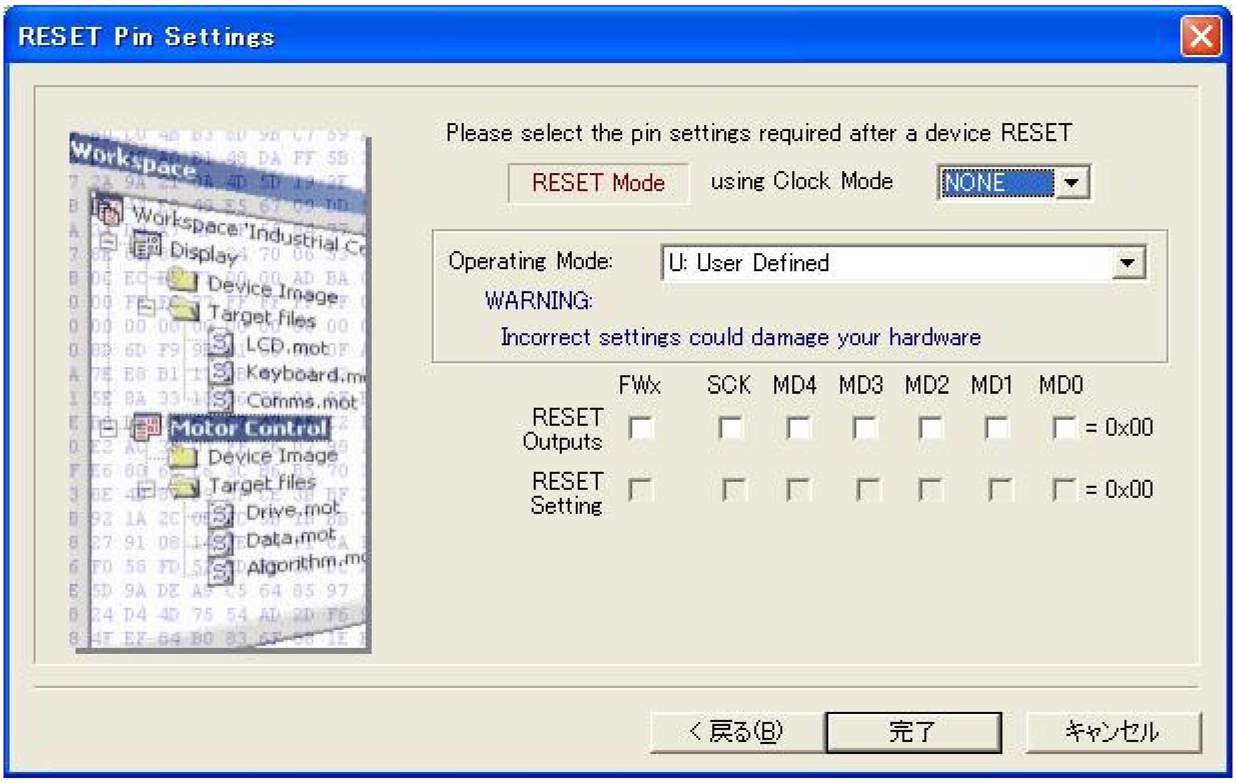
Table 4-4 Example of Setting Pins on the H8/3694F and Adapter Board (for the Boot Mode)

Pin No.	Pin on the Adapter Board	Pin on the Device	Input/Output	Output Level
1	$\overline{\text{RES}}$	$\overline{\text{RES}}$	Output (default)	Adapter board
3	FWx	NC	NC	-
5	MD0	$\overline{\text{NMI}}$	Output	Low (0)
7	MD1	NC	NC	-
9	MD2 (IO0)	P85	Output	High (1)
11	MD3 (IO1)	NC	NC	-
13	MD4 (IO2)	NC	NC	-
15	RXD	TXD	Input (default)	Adapter board
17	TXD	RXD	Output (default)	Adapter board
19	SCK (NC)	NC	NC (default)	-

Note: NC: Means no connection.

4.2.10 Reset Mode Pin Settings

Set pins on the adapter board for restarting the device in the reset mode. These settings are not required for this procedure.



Please select the pin settings required after a device RESET

RESET Mode using Clock Mode NONE

Operating Mode: U: User Defined

WARNING:

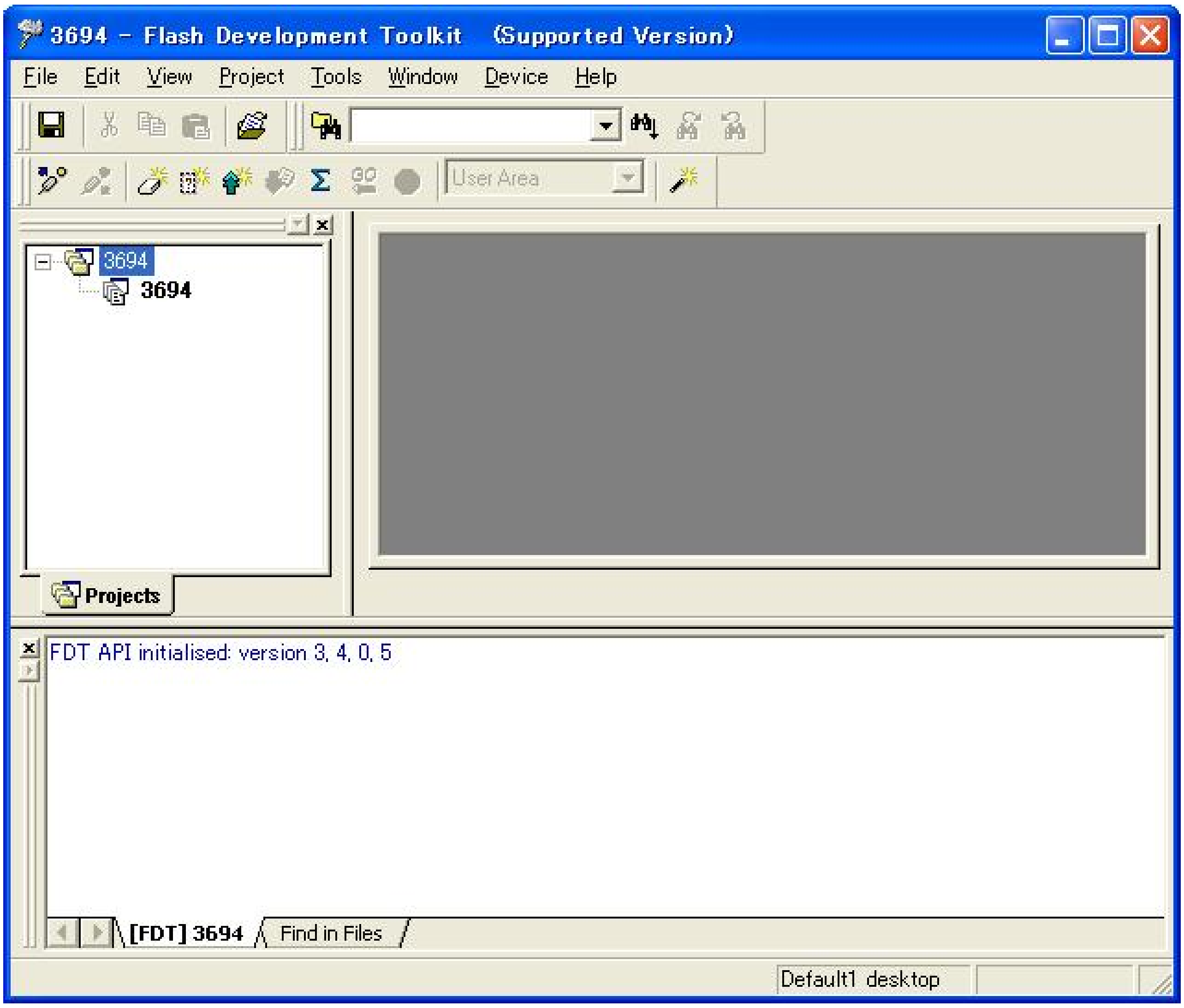
Incorrect settings could damage your hardware

	FWx	SCK	MD4	MD3	MD2	MD1	MD0	
RESET Outputs	☐	☐	☐	☐	☐	☐	☐	= 0x00
RESET Setting	☐	☐	☐	☐	☐	☐	☐	= 0x00

When you have set the items, click "Finish."

4.2.11 Completion of Setting

The H8/3694F board has been set to the Flash Development Toolkit in the boot mode.

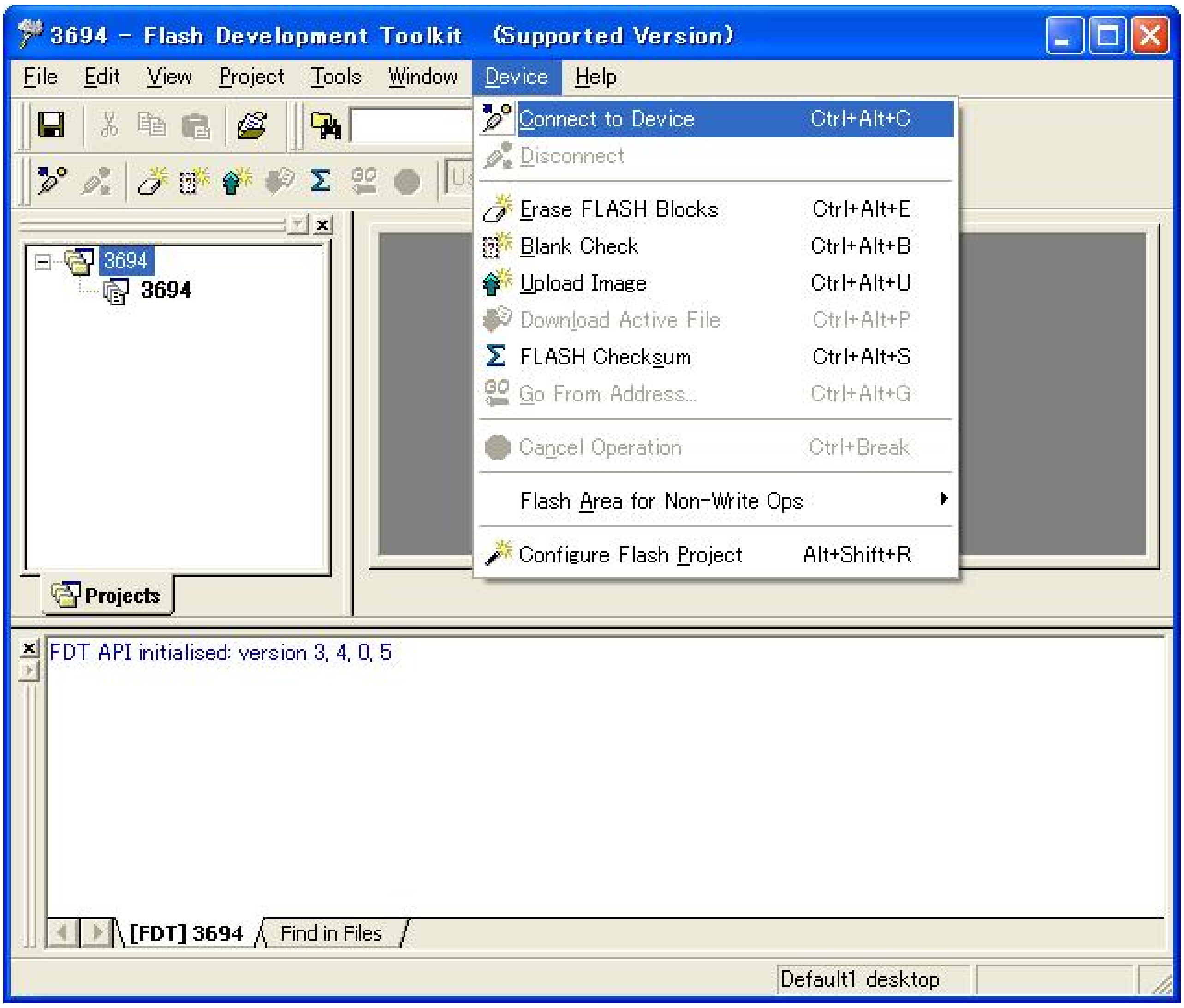

3694 - Flash Development Toolkit (Supported Version)
File Edit View Project Tools Window Device Help
User Area
3694
3694
Projects
[FDT] 3694
Find in Files
Default1 desktop

FDT API initialised: version 3, 4, 0, 5

4.2.12 Connecting the Device

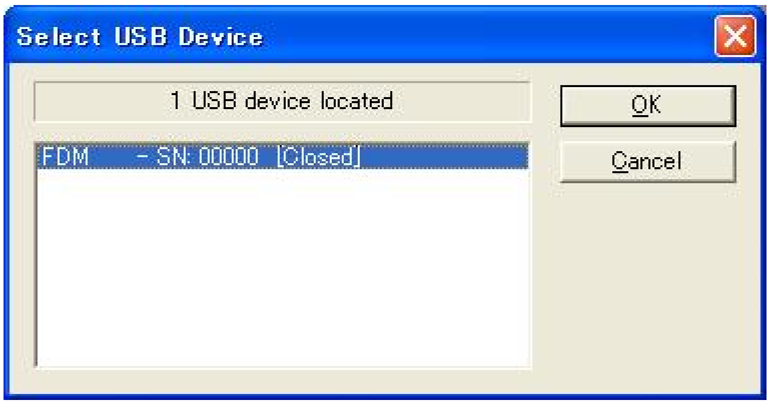
Connect the adapter board (FDM) to a PC and the H8/3694F board to the adapter board and turn on the power.

After the completion of the connection, click "Device" to open the pull-down menu and click "Connect to Device."


3694 - Flash Development Toolkit (Supported Version)
File
Edit
View
Project
Tools
Window
Device
Help
Connect to Device Ctrl+Alt+C
Disconnect
Erase FLASH Blocks Ctrl+Alt+E
Blank Check Ctrl+Alt+B
Upload Image Ctrl+Alt+U
Download Active File Ctrl+Alt+P
FLASH Checksum Ctrl+Alt+S
Go From Address... Ctrl+Alt+G
Cancel Operation Ctrl+Break
Flash Area for Non-Write Ops
Configure Flash Project Alt+Shift+R
3694
3694
Projects
[FDT] 3694
Find in Files
Default1 desktop

```
FDT API initialised: version 3, 4, 0, 5
```

Select the adapter board (FDM).

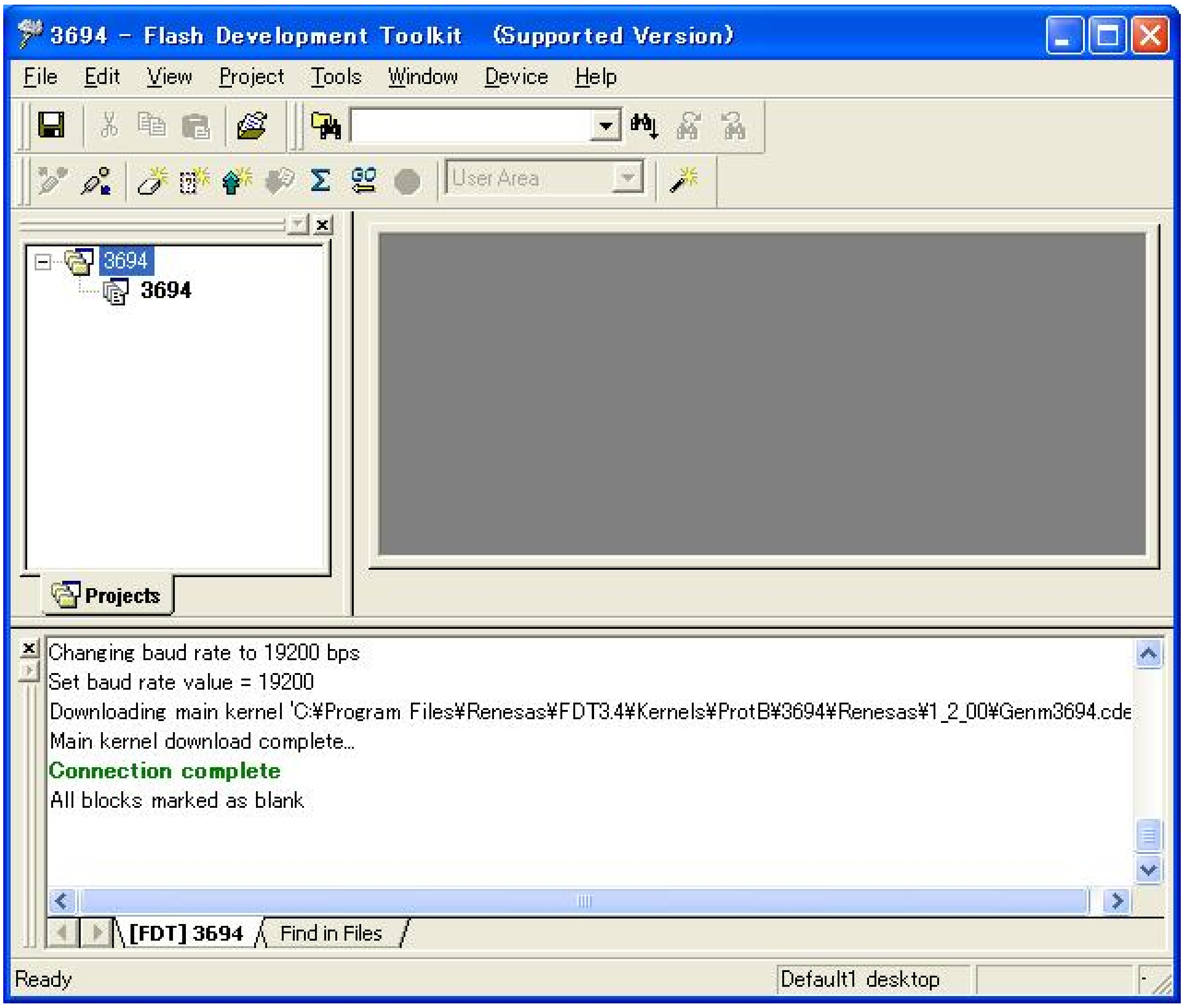


When you have selected the device, click "OK."

4.2.13 Completion of Connection

The H8/3694F board has been connected to the Flash Development Toolkit in the boot mode.

At this time, the contents of the user area have been erased.


3694 - Flash Development Toolkit (Supported Version)
File Edit View Project Tools Window Device Help
User Area
3694
3694
Projects
[FDT] 3694
Find in Files
Ready
Default1 desktop

```
Changing baud rate to 19200 bps
Set baud rate value = 19200
Downloading main kernel 'C:¥Program Files¥Renesas¥FDT3.4¥Kernels¥ProtB¥3694¥Renesas¥1_2_00¥Genm3694.cde
Main kernel download complete...
Connection complete
All blocks marked as blank
```

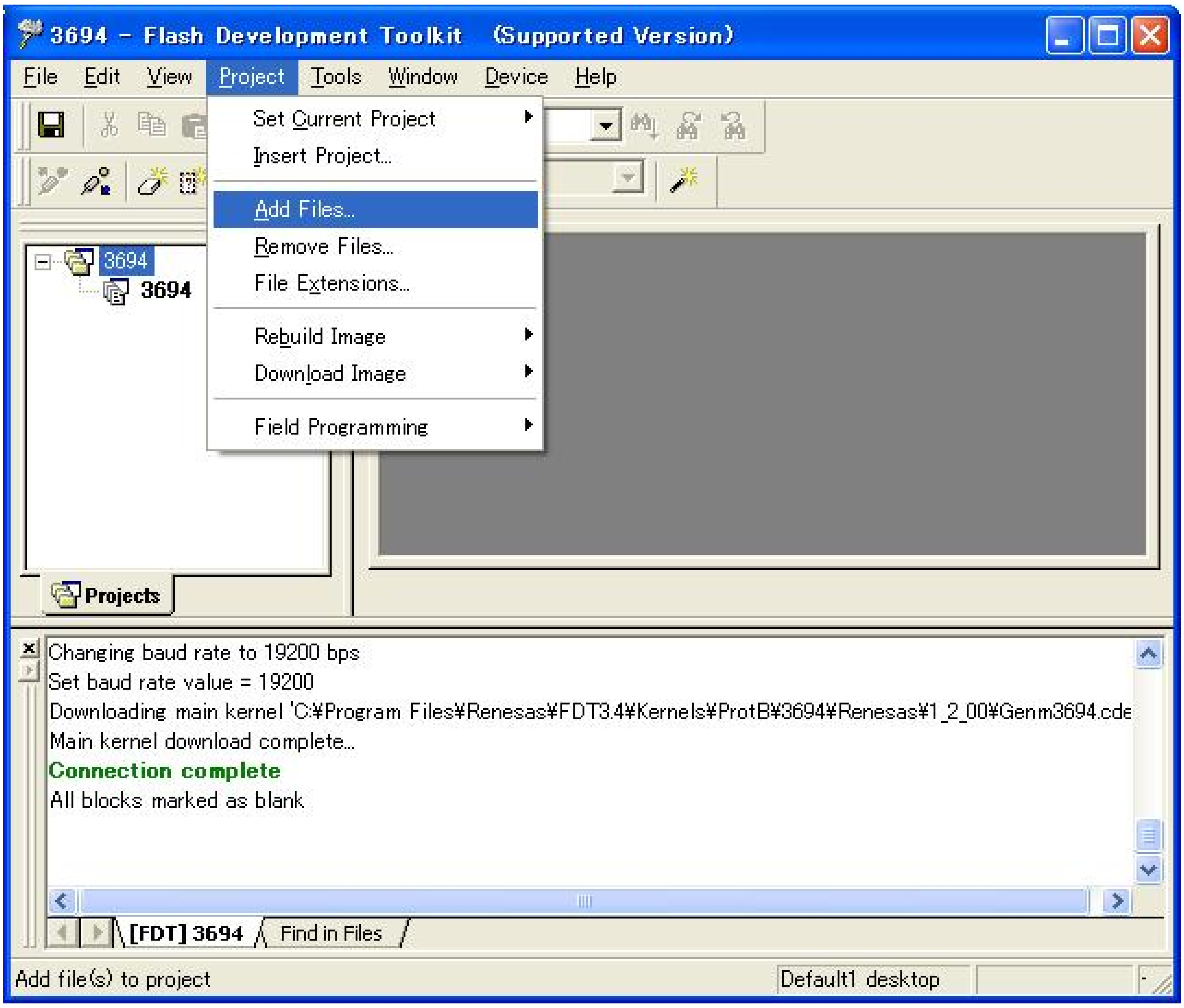
4.3 Boot Mode (Programming the User Area)

Write a program in the user area in the boot mode. The program to be written is the sample test program supplied with the Flash Development Toolkit (files 3694Test.mot and uGenU.mot (S-type files)). The bit rate in this program must be modified according to the frequency. For how to modify the bit rate, refer to section 7.1.1, Bit Rate Setting (GenTest.h). The program is contained in the Renesas\FDT3.4\Kernels\ProtB folder for the Flash Development Toolkit. The following is the full pathname of the Flash Development Toolkit programs when they are installed in the Program Files folder:

C:\Program Files\Renesas\FDT3.4\Kernels\ProtB\3694\Renesas\1_2_00

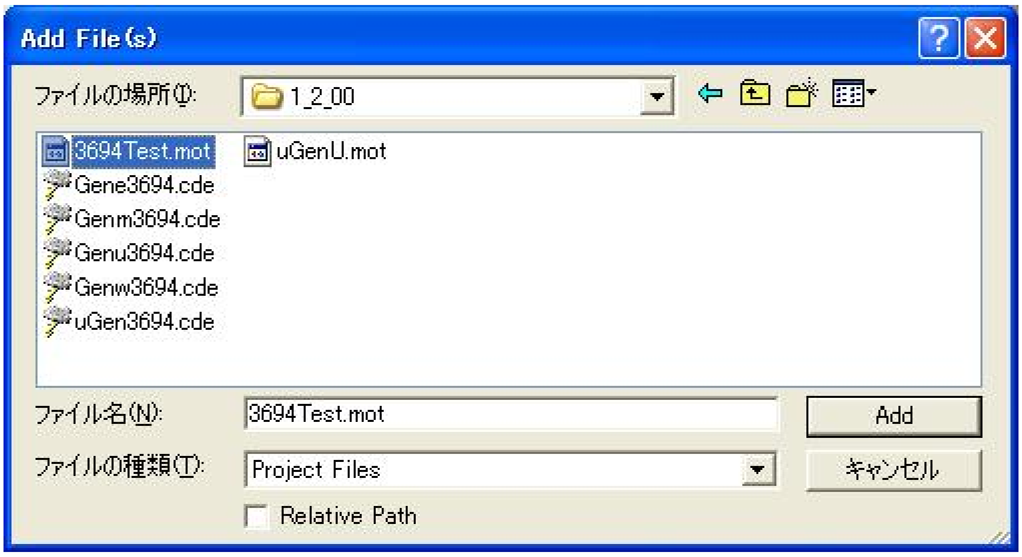
4.3.1 Selecting Files

To select files to be programmed, select "Add Files..." from the "Project" pull-down menu.



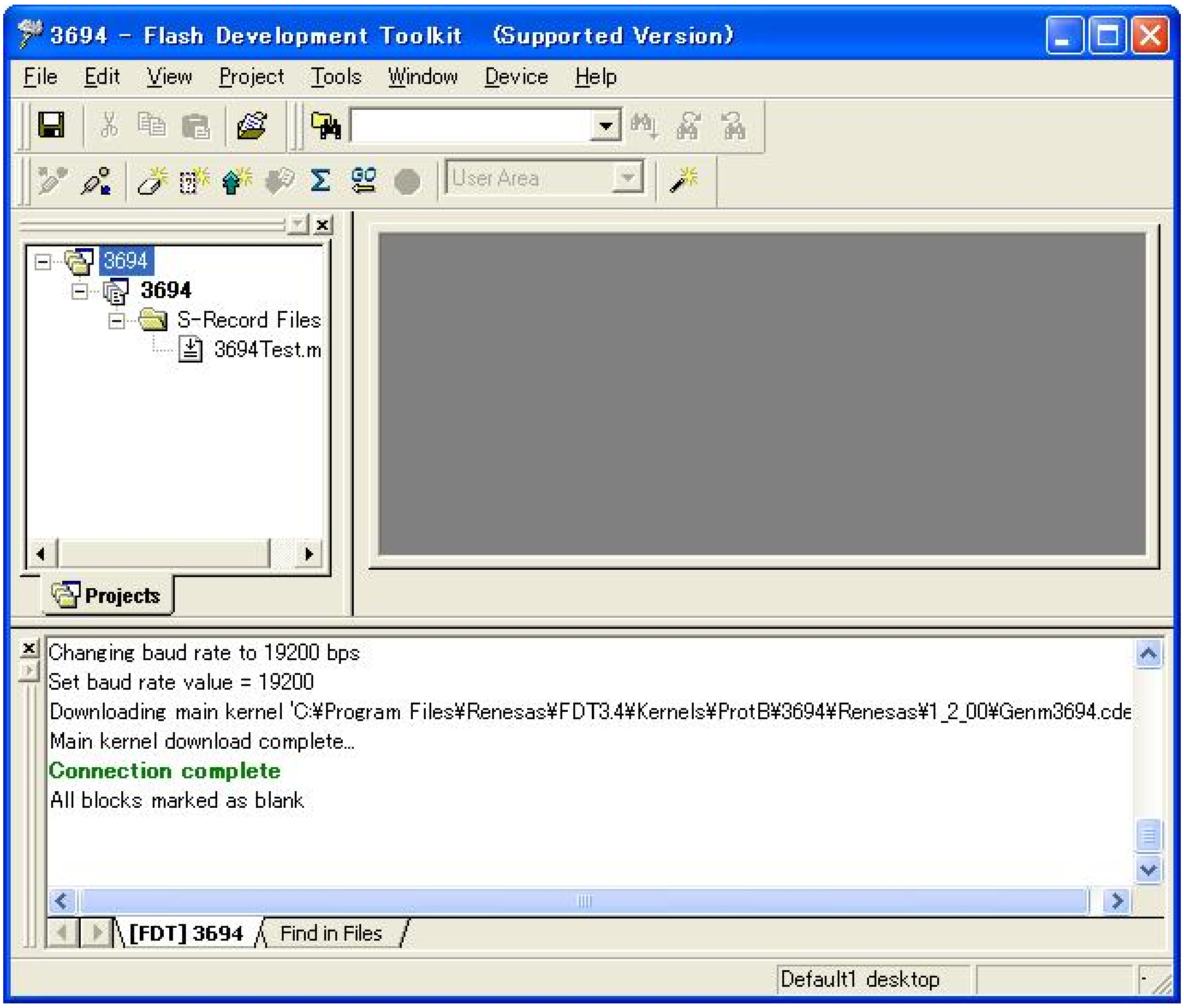
```
Changing baud rate to 19200 bps
Set baud rate value = 19200
Downloading main kernel 'C:¥Program Files¥Renesas¥FDT3.4¥Kernels¥ProtB¥3694¥Renesas¥1_2_00¥Genm3694.cde
Main kernel download complete...
Connection complete
All blocks marked as blank
```

In the "Add Files" dialog box, add file 3694Test.mot.



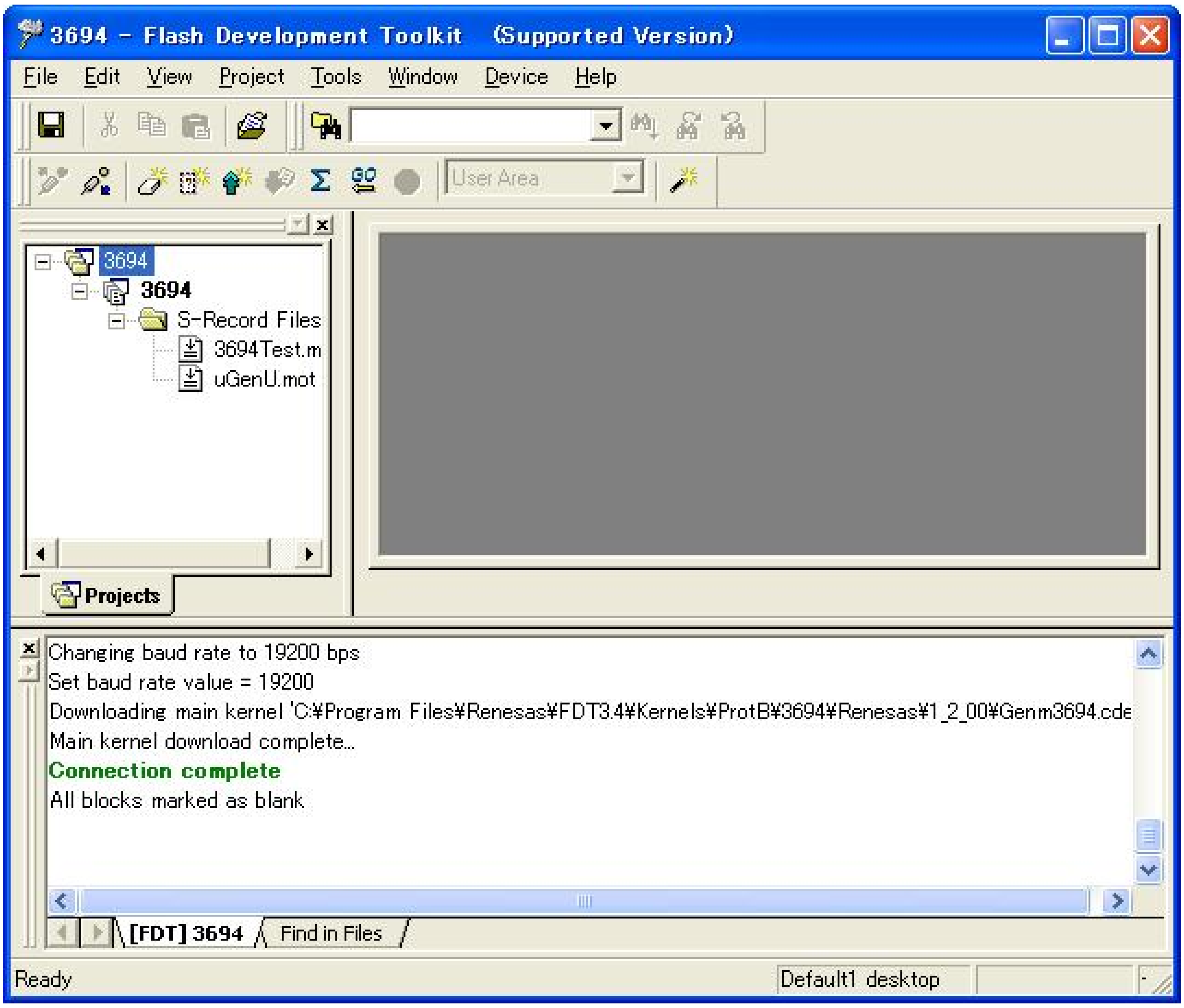
When you have selected the file, click "Add."

File 3694Test.mot is now added to the project.


3694 - Flash Development Toolkit (Supported Version)
File Edit View Project Tools Window Device Help
User Area
3694
3694
S-Record Files
3694Test.m
Projects
[FDT] 3694
Find in Files
Default1 desktop

```
Changing baud rate to 19200 bps
Set baud rate value = 19200
Downloading main kernel 'C:¥Program Files¥Renesas¥FDT3.4¥Kernels¥ProtB¥3694¥Renesas¥1_2_00¥Genm3694.cde
Main kernel download complete...
Connection complete
All blocks marked as blank
```

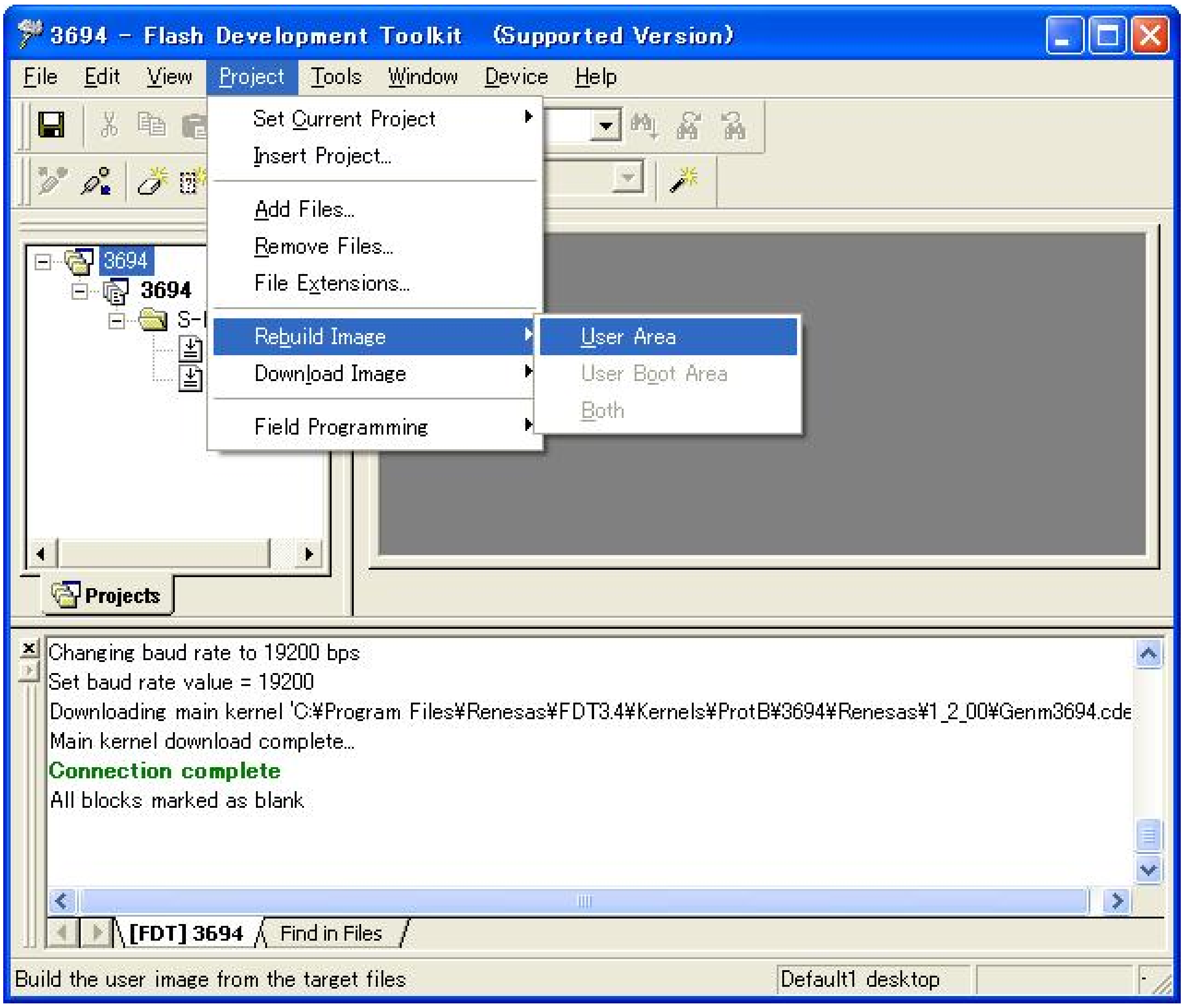
In the same way, add uGenU.mot.



```
Changing baud rate to 19200 bps
Set baud rate value = 19200
Downloading main kernel 'C:¥Program Files¥Renesas¥FDT3.4¥Kernels¥ProtB¥3694¥Renesas¥1_2_00¥Genm3694.cde
Main kernel download complete...
Connection complete
All blocks marked as blank
```

4.3.2 Building the Image

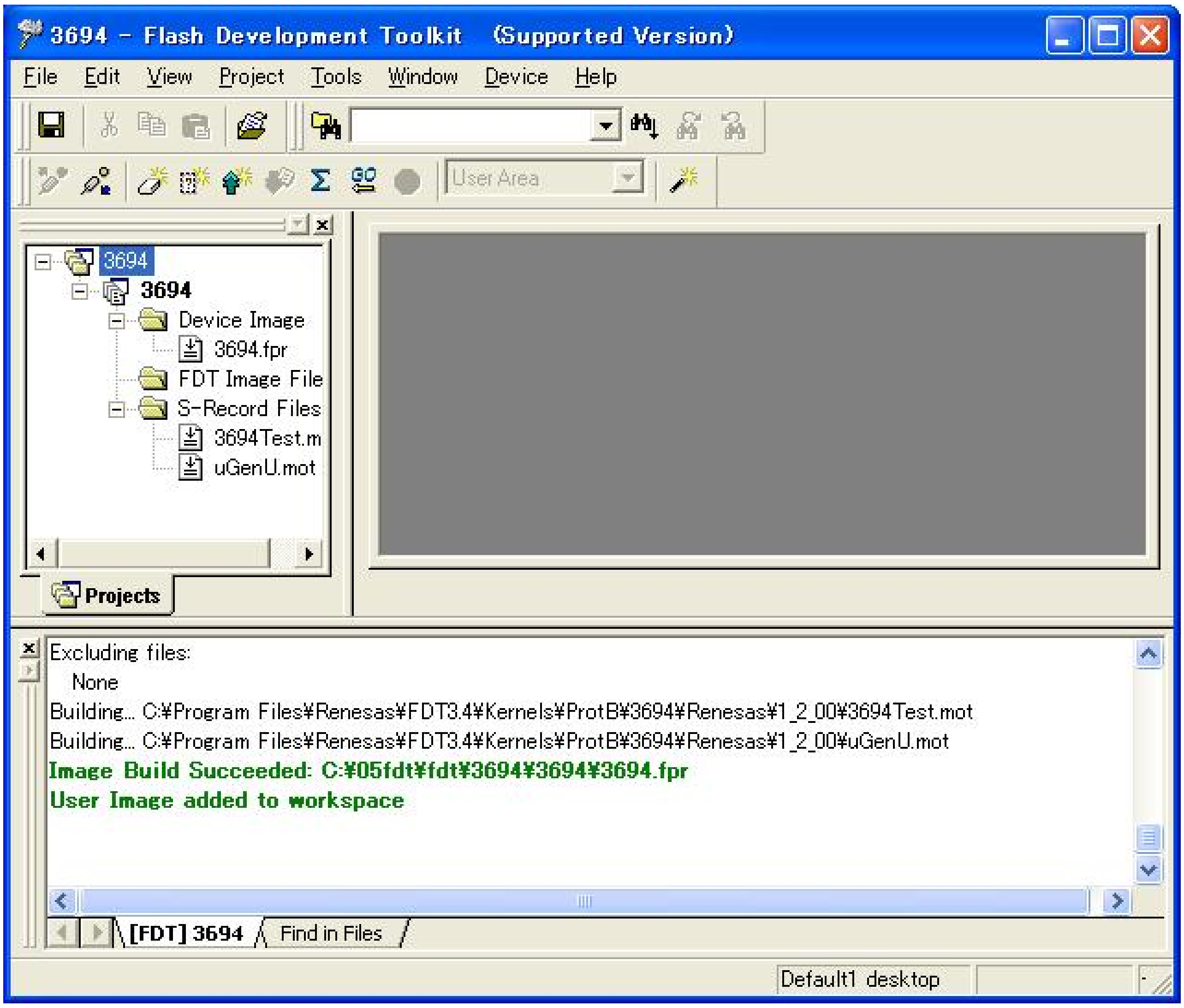
Build the user area device image because more than one file is to be programmed. From the "Project" pull-down menu, select "Rebuild Image" then "User Area."



```
Changing baud rate to 19200 bps
Set baud rate value = 19200
Downloading main kernel 'C:¥Program Files¥Renesas¥FDT3.4¥Kernels¥ProtB¥3694¥Renesas¥1_2_00¥Genm3694.cde
Main kernel download complete...
Connection complete
All blocks marked as blank
```

Build the user image from the target files

Image file 3694.fpr is created.

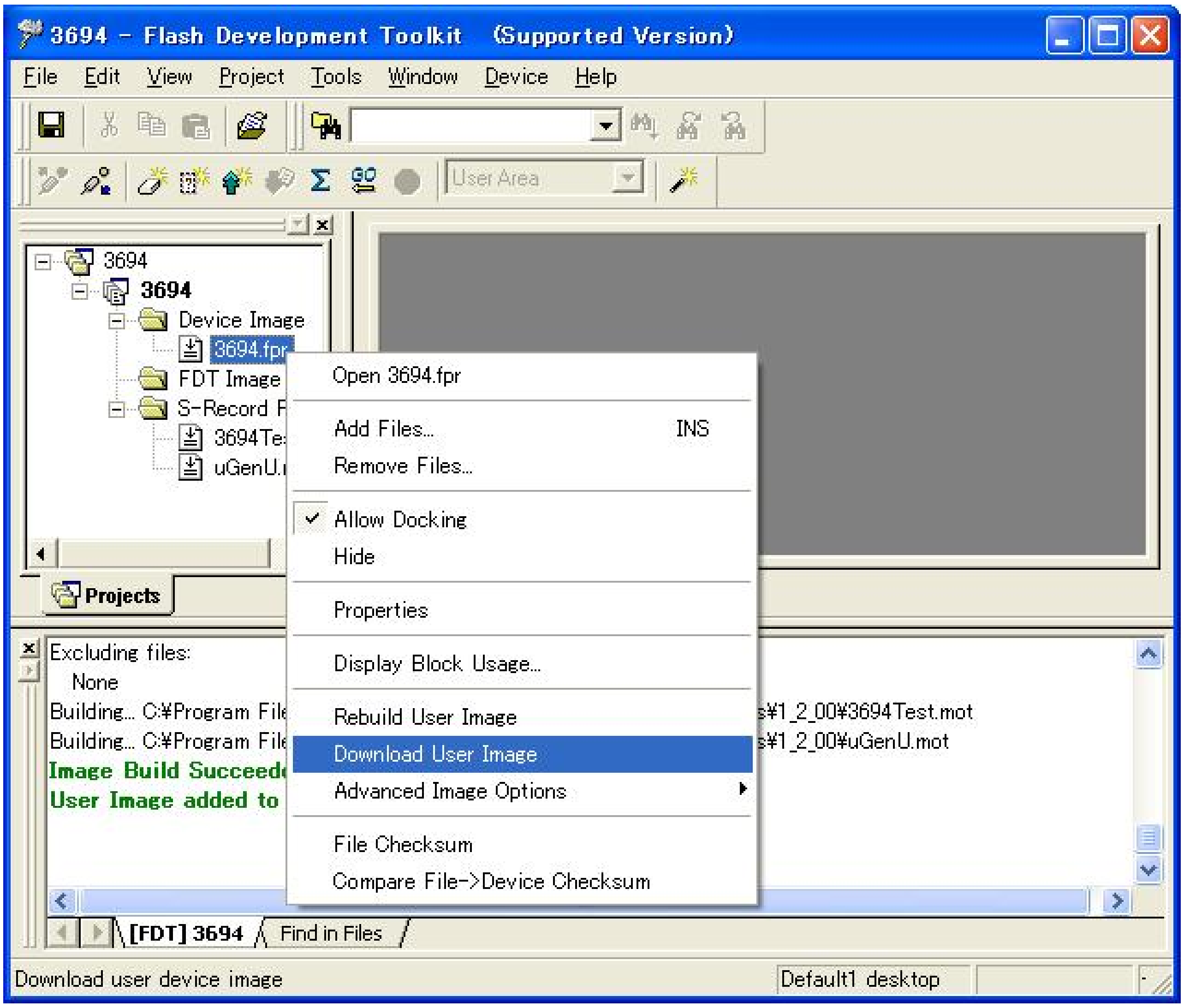


```
Excluding files:
  None
Building... C:¥Program Files¥Renesas¥FDT3.4¥Kernels¥ProtB¥3694¥Renesas¥1_2_00¥3694Test.mot
Building... C:¥Program Files¥Renesas¥FDT3.4¥Kernels¥ProtB¥3694¥Renesas¥1_2_00¥uGenU.mot
Image Build Succeeded: C:¥05fdt¥fdt¥3694¥3694¥3694.fpr
User Image added to workspace
```

4.3.3 Programming

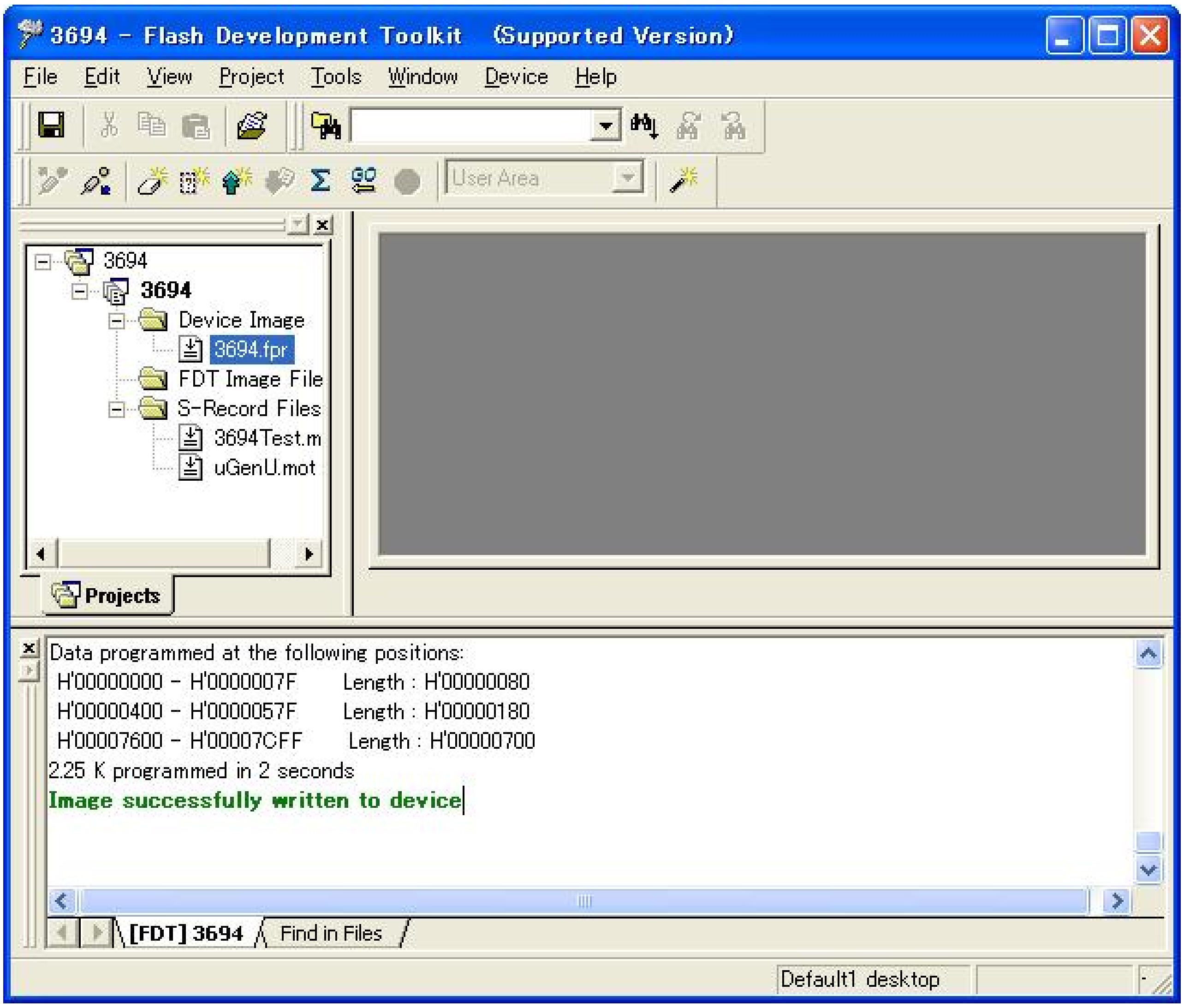
Program the user area.

Click the right mouse button on file 3694.fpr to display the pop-up menu. Click "Download User Image" to download file 3694.fpr to the user area.



```
Excluding files:
  None
Building... C:¥Program File                              s¥1_2_00¥3694Test.mot
Building... C:¥Program File                              s¥1_2_00¥uGenU.mot
Image Build Succeede
User Image added to
```

You can check that the program has been downloaded to the user area.

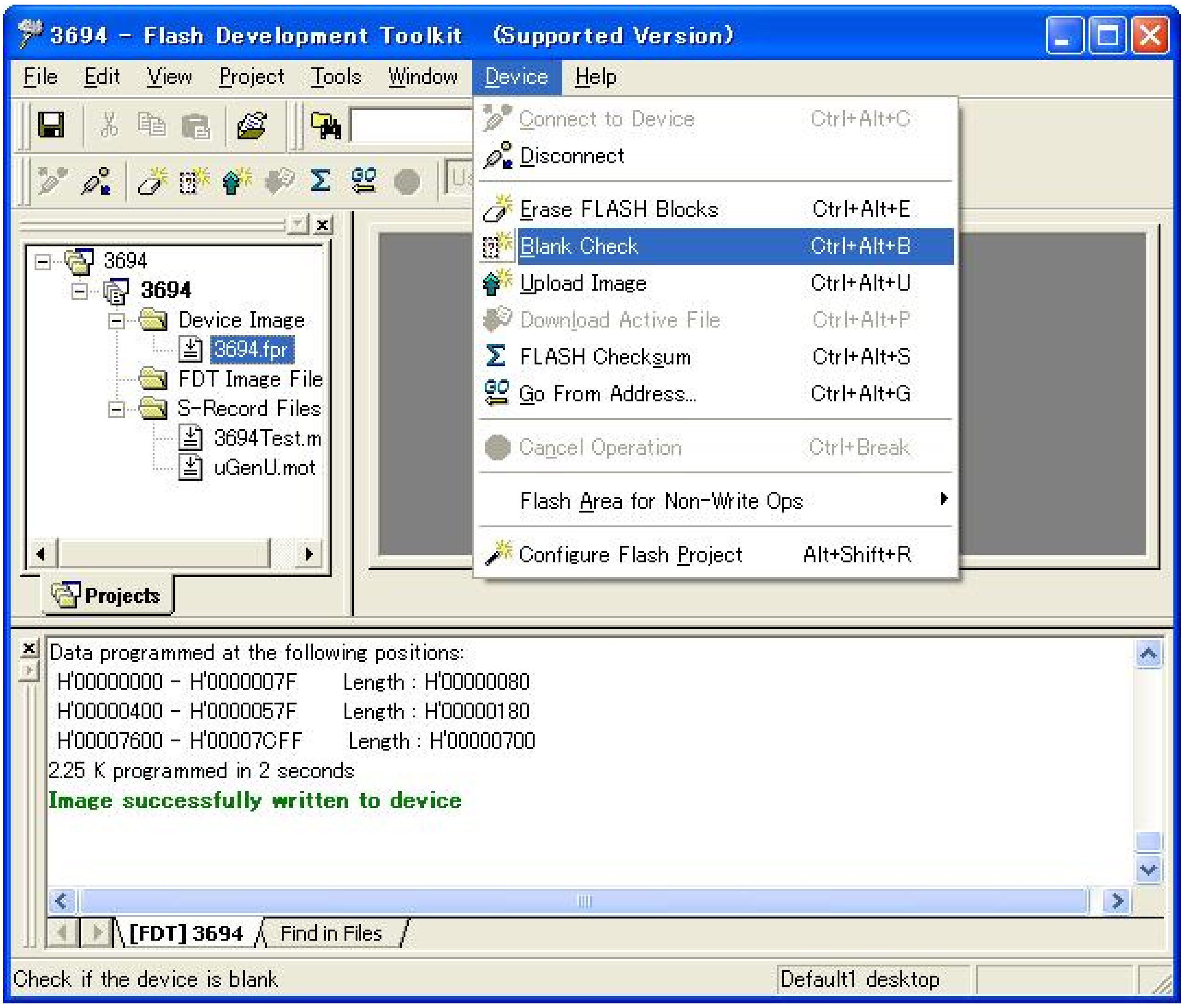


```
Data programmed at the following positions:
 H'00000000 - H'0000007F      Length : H'00000080
 H'00000400 - H'0000057F      Length : H'00000180
 H'00007600 - H'00007CFF      Length : H'00000700
2.25 K programmed in 2 seconds
Image successfully written to device
```

4.3.4 Blank Check

To confirm that the user boot area has been programmed, perform a blank check.

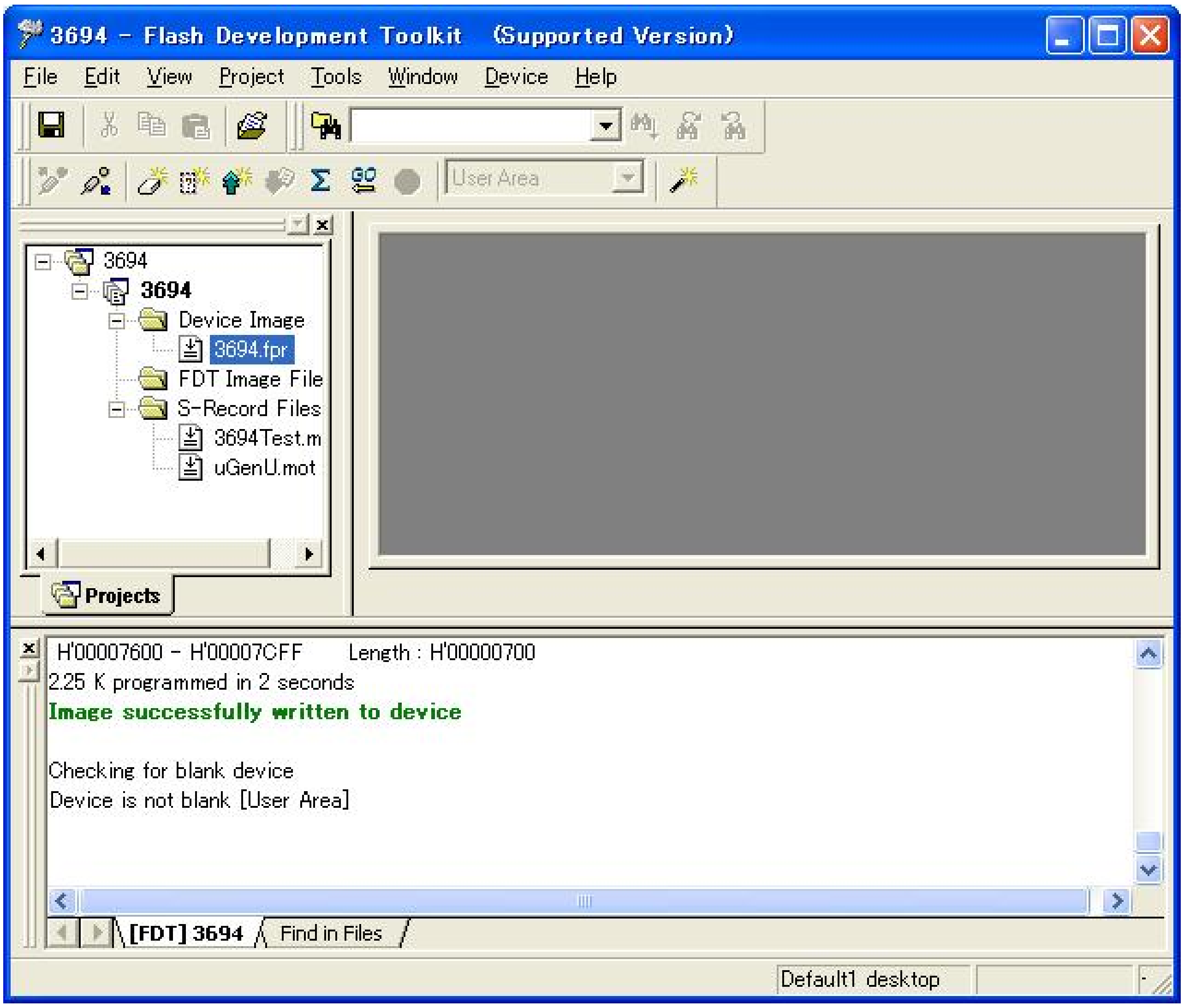
Click "Device" to open the pull-down menu and click "Blank Check."


3694 - Flash Development Toolkit (Supported Version)
File Edit View Project Tools Window Device Help
Connect to Device Ctrl+Alt+C
Disconnect
Erase FLASH Blocks Ctrl+Alt+E
Blank Check Ctrl+Alt+B
Upload Image Ctrl+Alt+U
Download Active File Ctrl+Alt+P
FLASH Checksum Ctrl+Alt+S
Go From Address... Ctrl+Alt+G
Cancel Operation Ctrl+Break
Flash Area for Non-Write Ops
Configure Flash Project Alt+Shift+R
3694
3694
Device Image
3694.fpr
FDT Image File
S-Record Files
3694Test.m
uGenU.mot
Projects
[FDT] 3694
Find in Files
Check if the device is blank
Default1 desktop

```
Data programmed at the following positions:
 H'00000000 - H'0000007F      Length : H'00000080
 H'00000400 - H'0000057F      Length : H'00000180
 H'00007600 - H'00007CFF      Length : H'00000700
2.25 K programmed in 2 seconds
Image successfully written to device
```

The result of the blank check for the selected area is displayed.

The user area is not blank.



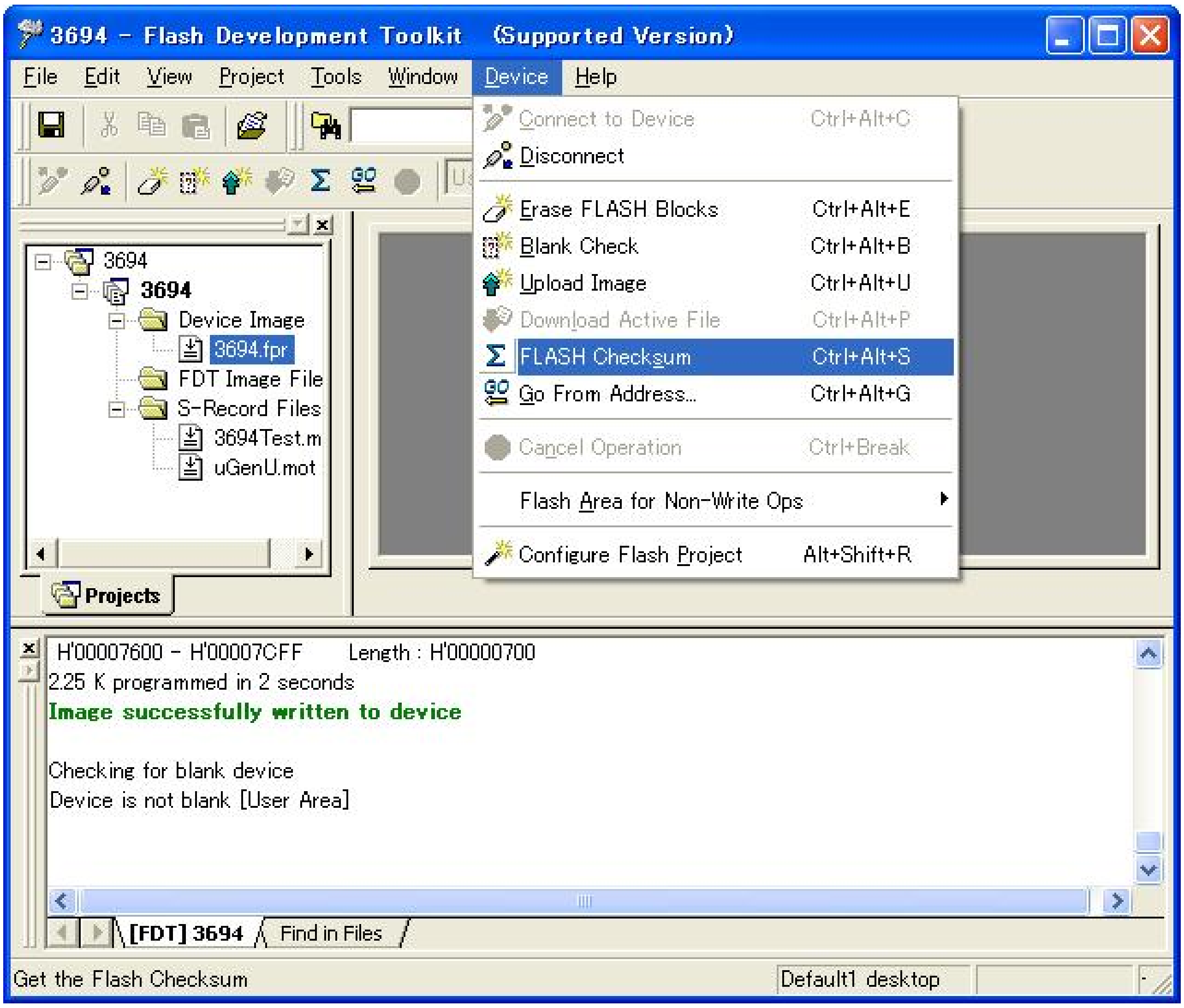
```
 H'00007600 - H'00007CFF      Length : H'00000700
2.25 K programmed in 2 seconds
Image successfully written to device

Checking for blank device
Device is not blank [User Area]
```

4.3.5 Checksum

To confirm that the user boot area has been programmed, display a checksum.

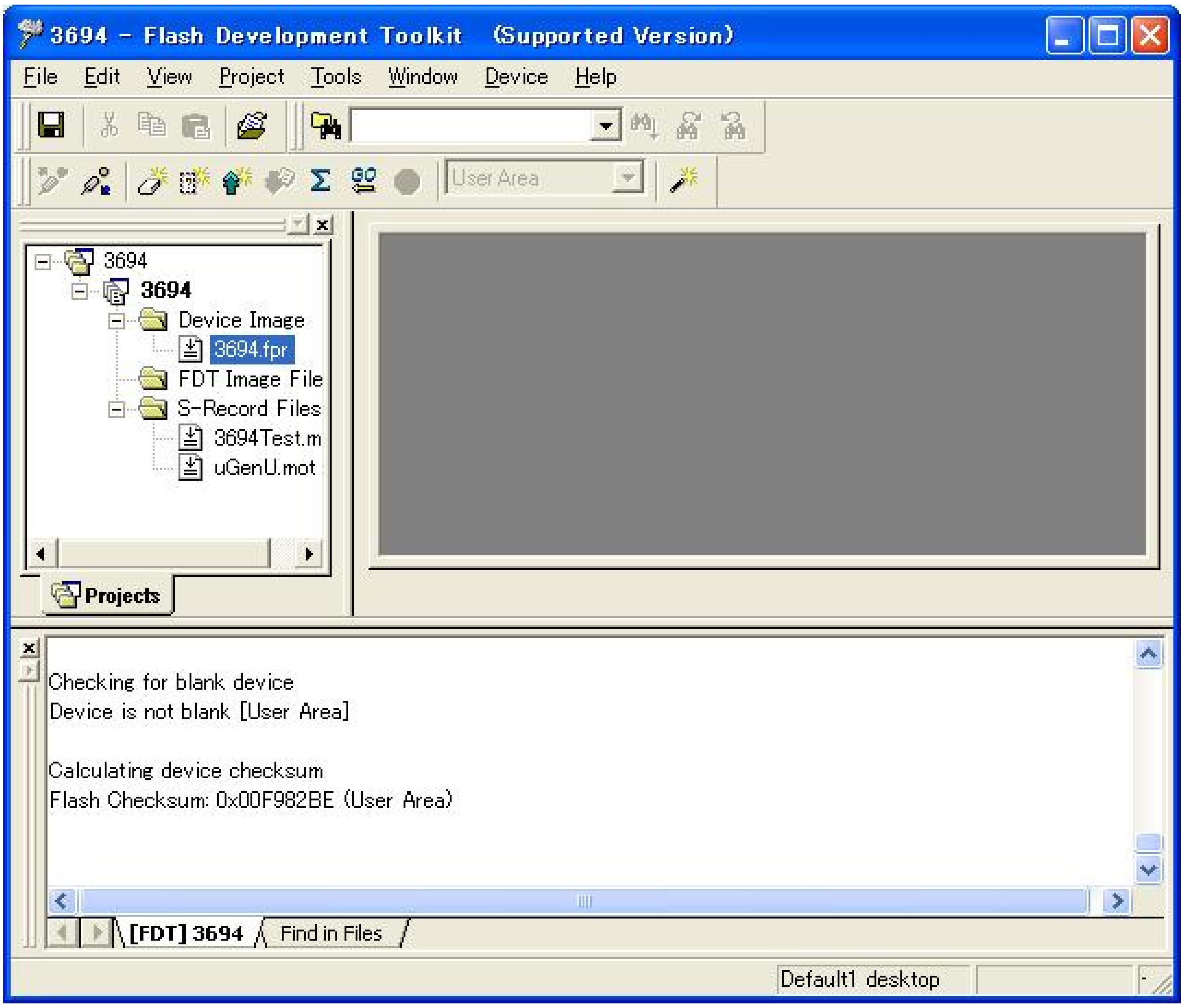
Click "Device" to open the pull-down menu and click "FLASH Checksum."


3694 - Flash Development Toolkit (Supported Version)
File Edit View Project Tools Window Device Help
Connect to Device Ctrl+Alt+C
Disconnect
Erase FLASH Blocks Ctrl+Alt+E
Blank Check Ctrl+Alt+B
Upload Image Ctrl+Alt+U
Download Active File Ctrl+Alt+P
FLASH Checksum Ctrl+Alt+S
Go From Address... Ctrl+Alt+G
Cancel Operation Ctrl+Break
Flash Area for Non-Write Ops
Configure Flash Project Alt+Shift+R
3694
3694
Device Image
3694.fpr
FDT Image File
S-Record Files
3694Test.m
uGenU.mot
Projects
[FDT] 3694
Find in Files
Get the Flash Checksum
Default1 desktop

```
H'00007600 - H'00007CFF      Length : H'00000700
2.25 K programmed in 2 seconds
Image successfully written to device

Checking for blank device
Device is not blank [User Area]
```

The result of the checksum calculation is displayed.



```
Checking for blank device
Device is not blank [User Area]

Calculating device checksum
Flash Checksum: 0x00F982BE (User Area)
```

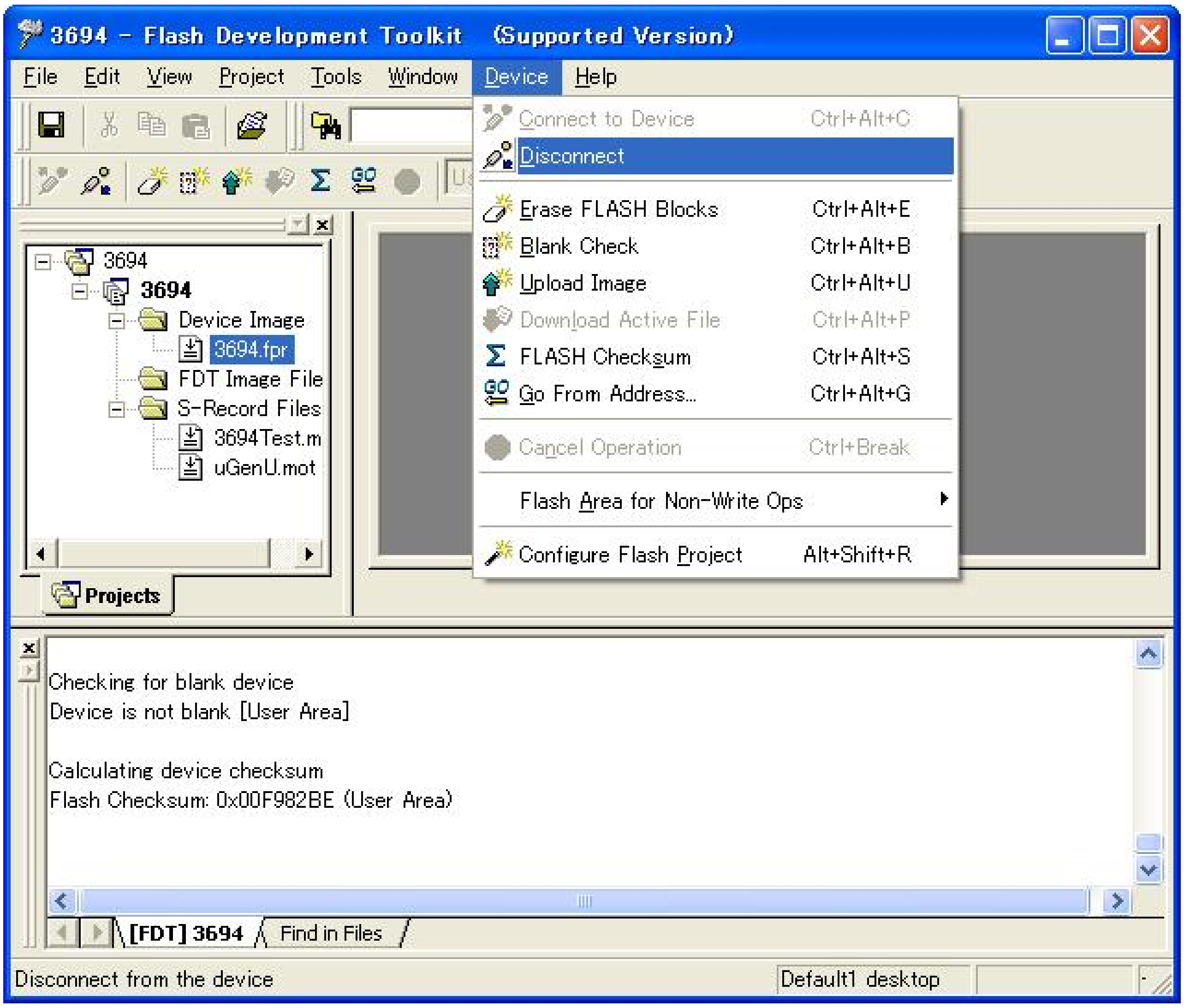
When the user area is blank, the following value is displayed as the result:

```
Calculating device checksum
Flash Checksum: 0x00FF0000 (User Area)
```

4.3.6 Disconnecting the Device

After the completion of programming, disconnect the device.

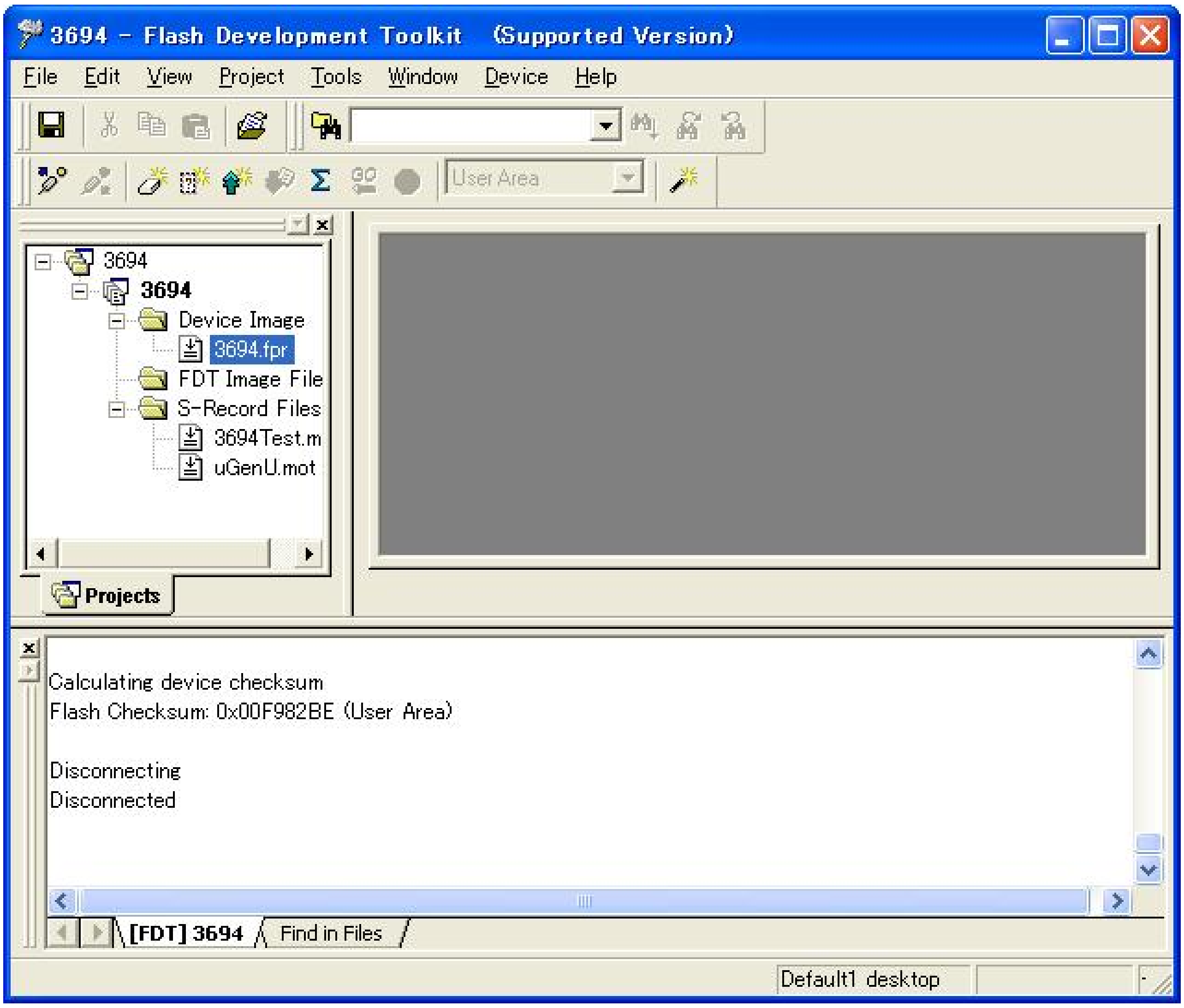
Click "Device" to open the pull-down menu and click "Disconnect."


3694 - Flash Development Toolkit (Supported Version)
File Edit View Project Tools Window Device Help
Connect to Device Ctrl+Alt+C
Disconnect
Erase FLASH Blocks Ctrl+Alt+E
Blank Check Ctrl+Alt+B
Upload Image Ctrl+Alt+U
Download Active File Ctrl+Alt+P
FLASH Checksum Ctrl+Alt+S
Go From Address... Ctrl+Alt+G
Cancel Operation Ctrl+Break
Flash Area for Non-Write Ops
Configure Flash Project Alt+Shift+R
3694
3694
Device Image
3694.fpr
FDT Image File
S-Record Files
3694Test.m
uGenU.mot
Projects
[FDT] 3694
Find in Files
Disconnect from the device
Default1 desktop

```
Checking for blank device
Device is not blank [User Area]

Calculating device checksum
Flash Checksum: 0x00F982BE (User Area)
```

The device is disconnected.



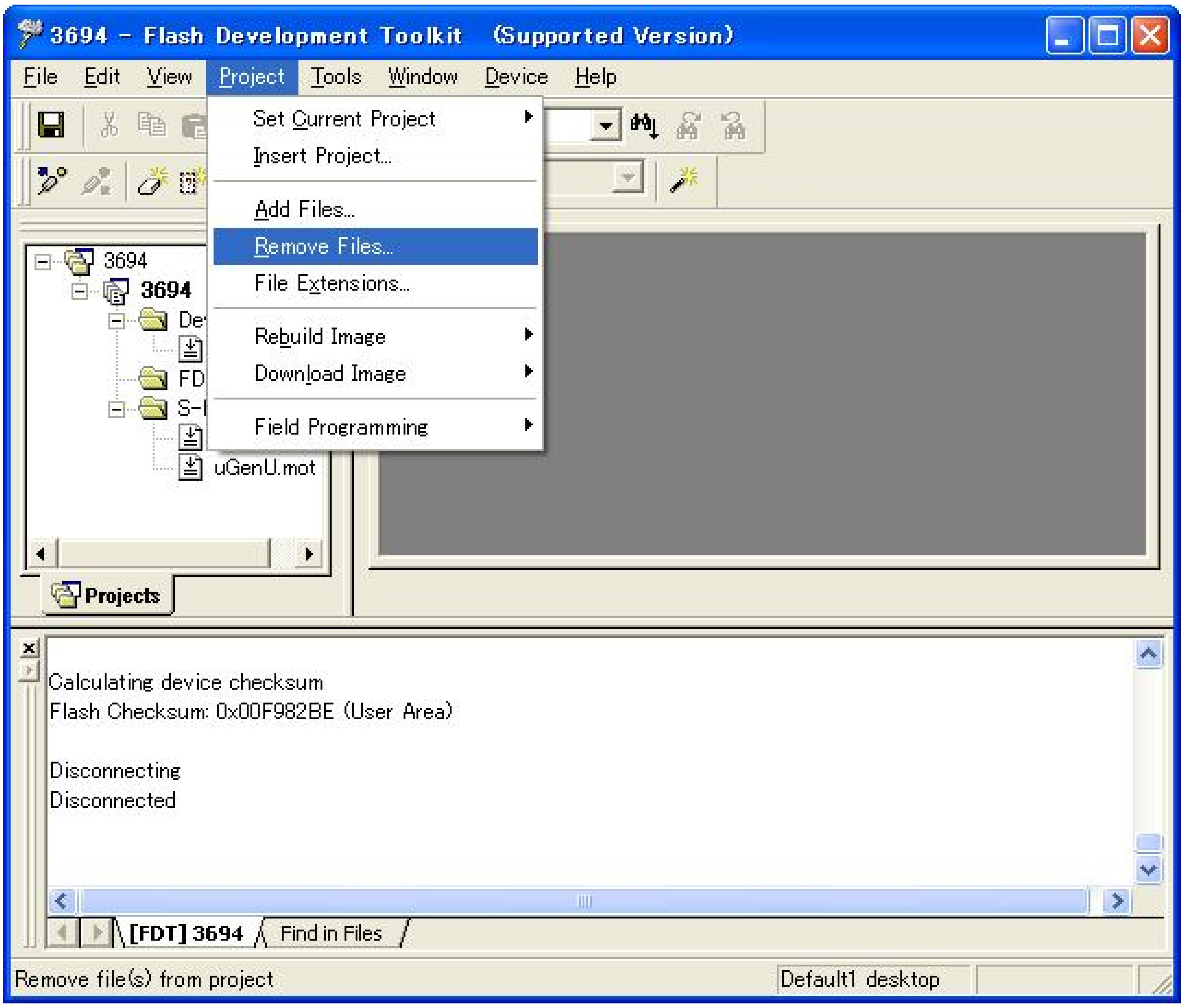
```
Calculating device checksum
Flash Checksum: 0x00F982BE (User Area)

Disconnecting
Disconnected
```

4.3.7 Removing Files

Remove files.

Click "Project" to open the pull-down menu and click "Remove Files...."

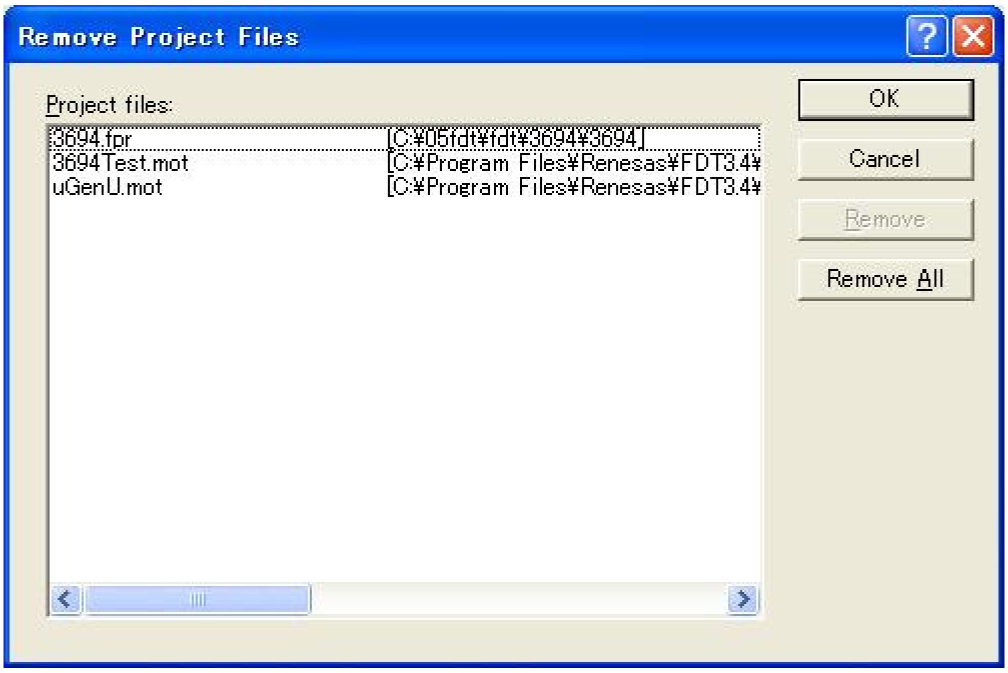


```
Calculating device checksum
Flash Checksum: 0x00F982BE (User Area)

Disconnecting
Disconnected
```

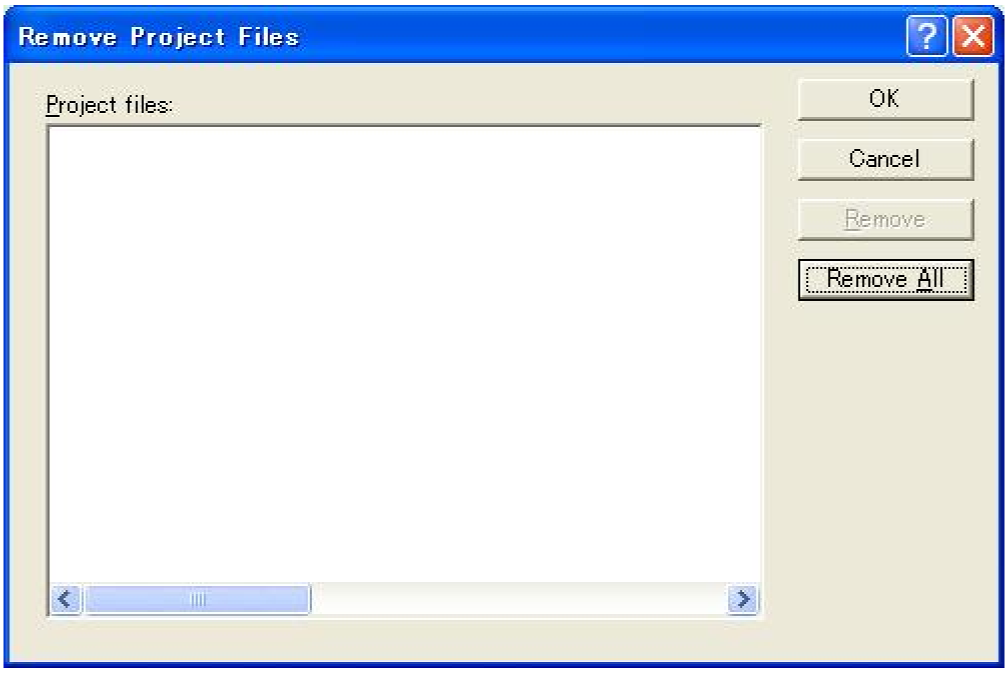
Remove file(s) from project

The project files are desplayed.



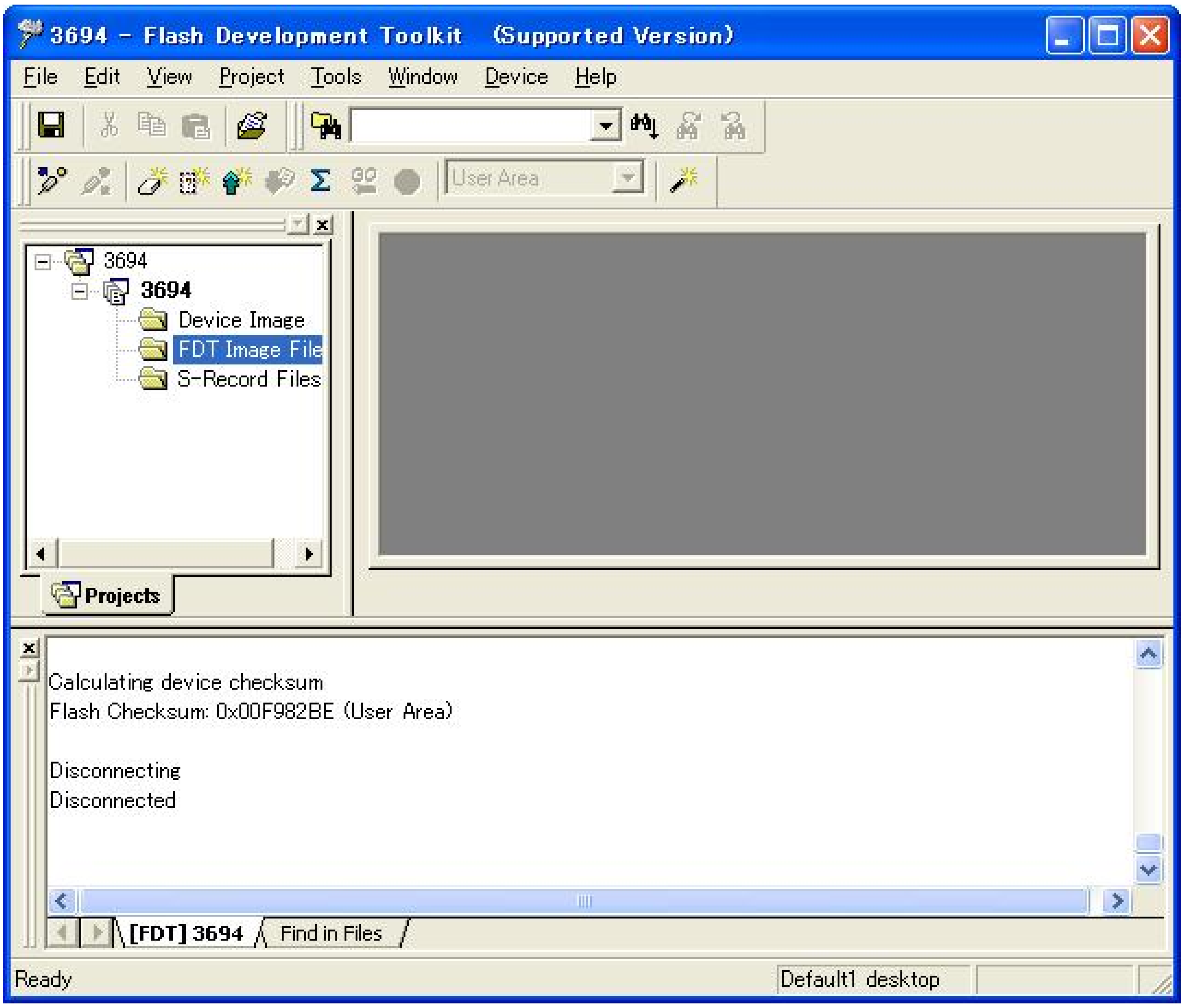
```
3694.fpr          [C:¥05fdt¥fdt¥3694¥3694]
3694Test.mot      [C:¥Program Files¥Renesas¥FDT3.4¥
uGenU.mot         [C:¥Program Files¥Renesas¥FDT3.4¥
```

Click "Remove All."



Click "OK."

The files are removed.



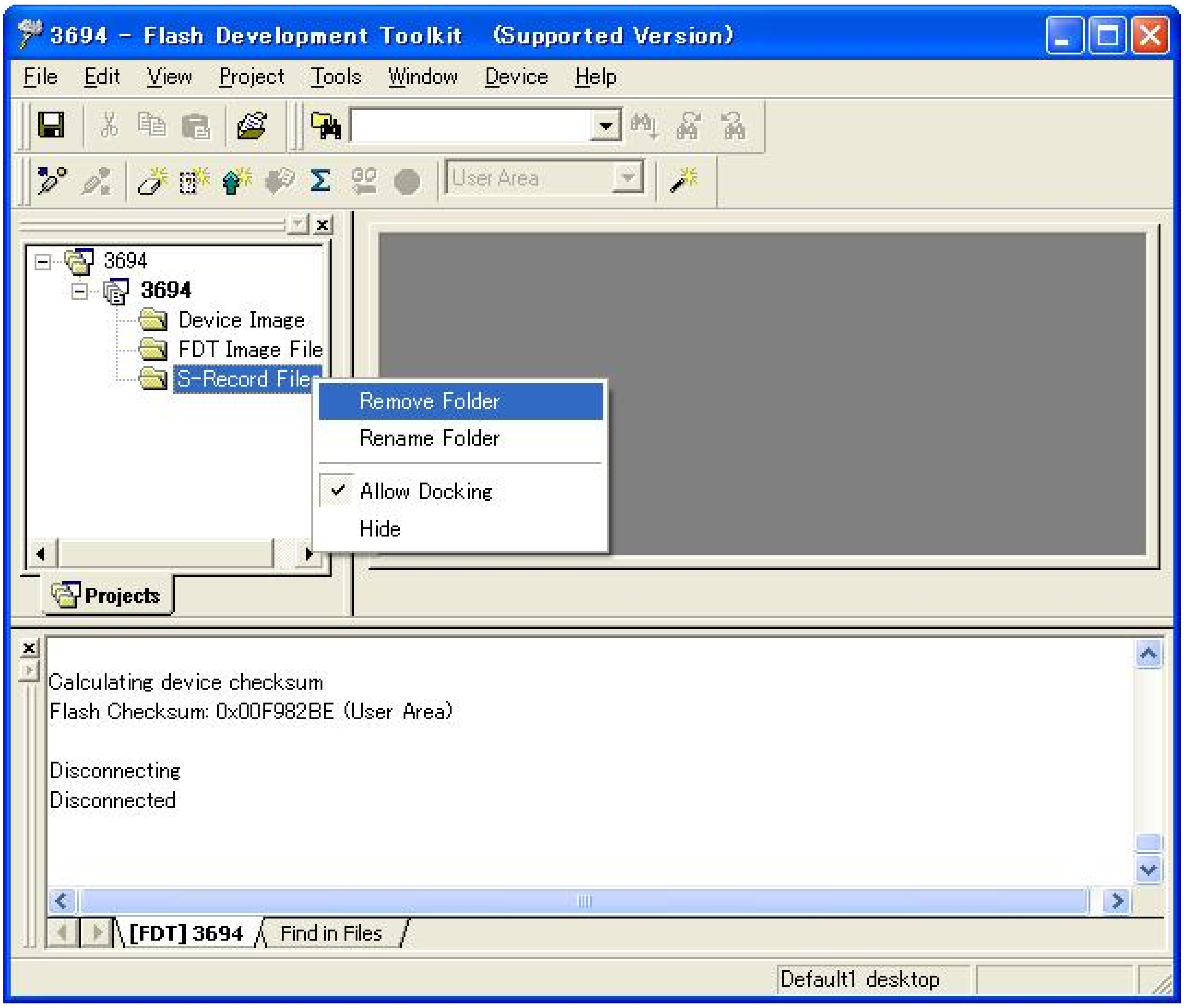
```
Calculating device checksum
Flash Checksum: 0x00F982BE (User Area)

Disconnecting
Disconnected
```

4.3.8 Removing Folders

Remove folders.

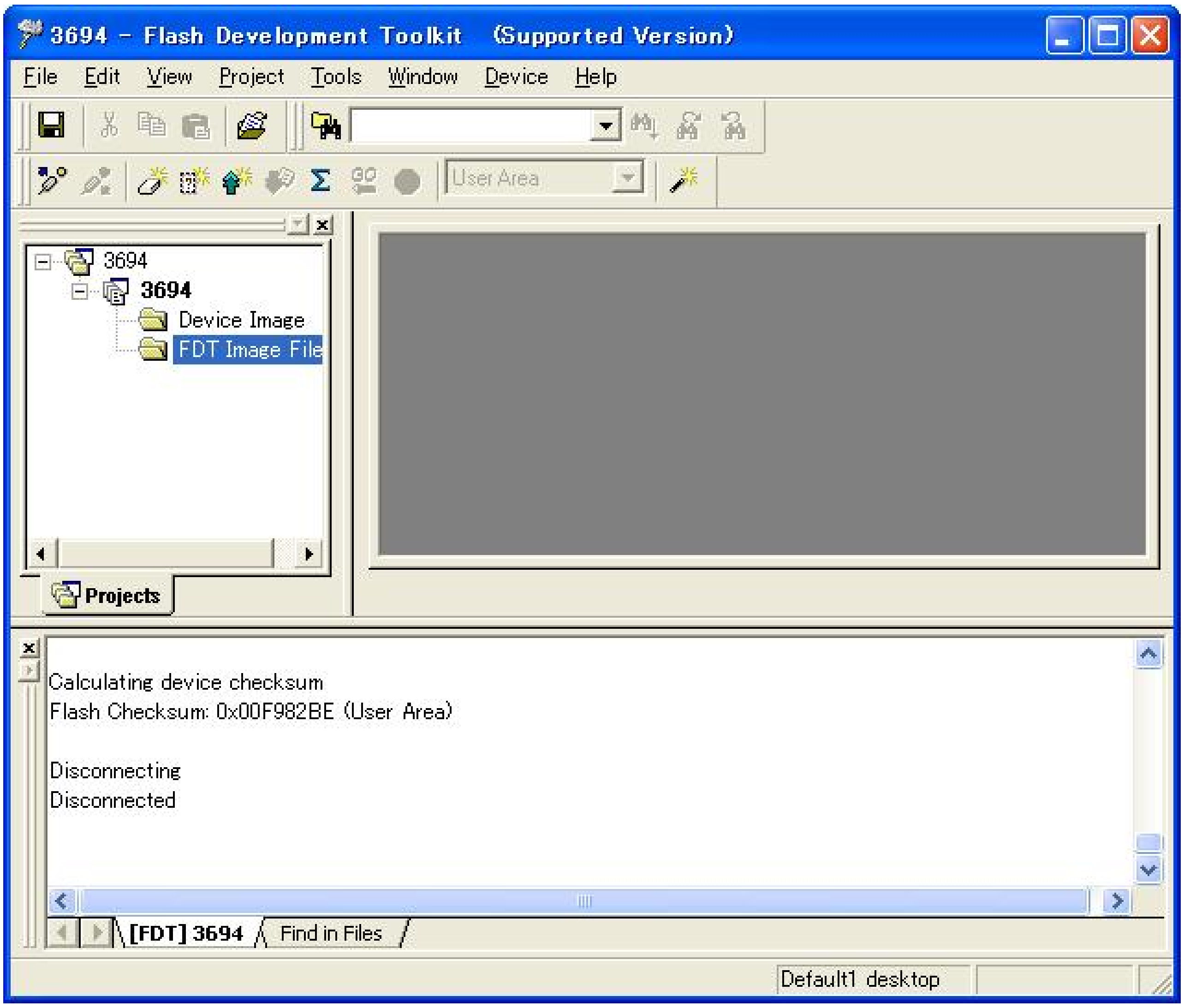
Click the right mouse button on a folder to display the pop-up menu and click "Remove Folder."



```
Calculating device checksum
Flash Checksum: 0x00F982BE (User Area)

Disconnecting
Disconnected
```

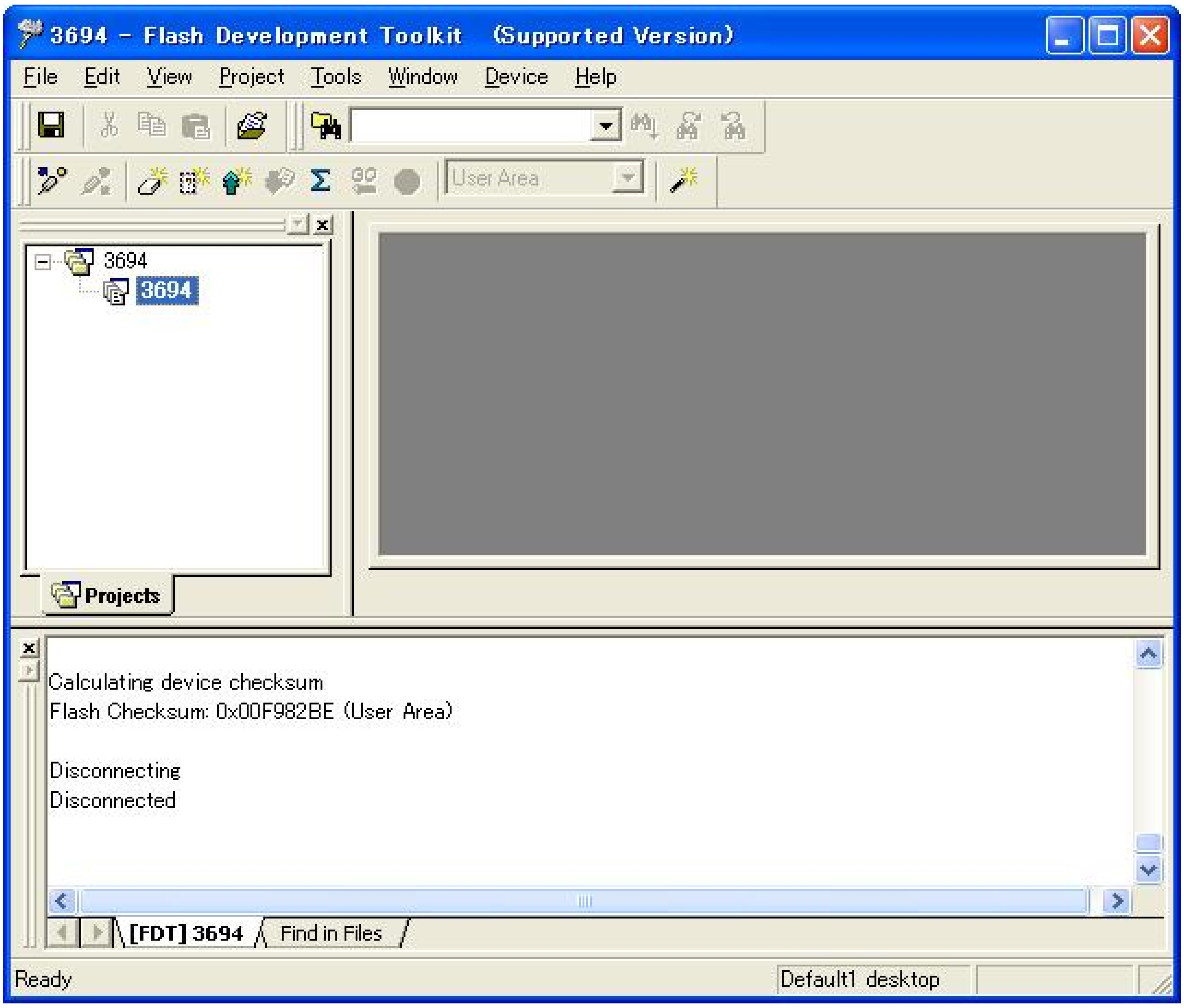
The folder is removed.


3694 - Flash Development Toolkit (Supported Version)
File Edit View Project Tools Window Device Help
User Area
3694
3694
Device Image
FDT Image File
Projects
[FDT] 3694
Find in Files
Default1 desktop

```
Calculating device checksum
Flash Checksum: 0x00F982BE (User Area)

Disconnecting
Disconnected
```

In the same way, remove the Device Image and FDT Image Files folders.


3694 - Flash Development Toolkit (Supported Version)
File Edit View Project Tools Window Device Help
User Area
3694
3694
Projects
[FDT] 3694
Find in Files
Ready
Default1 desktop

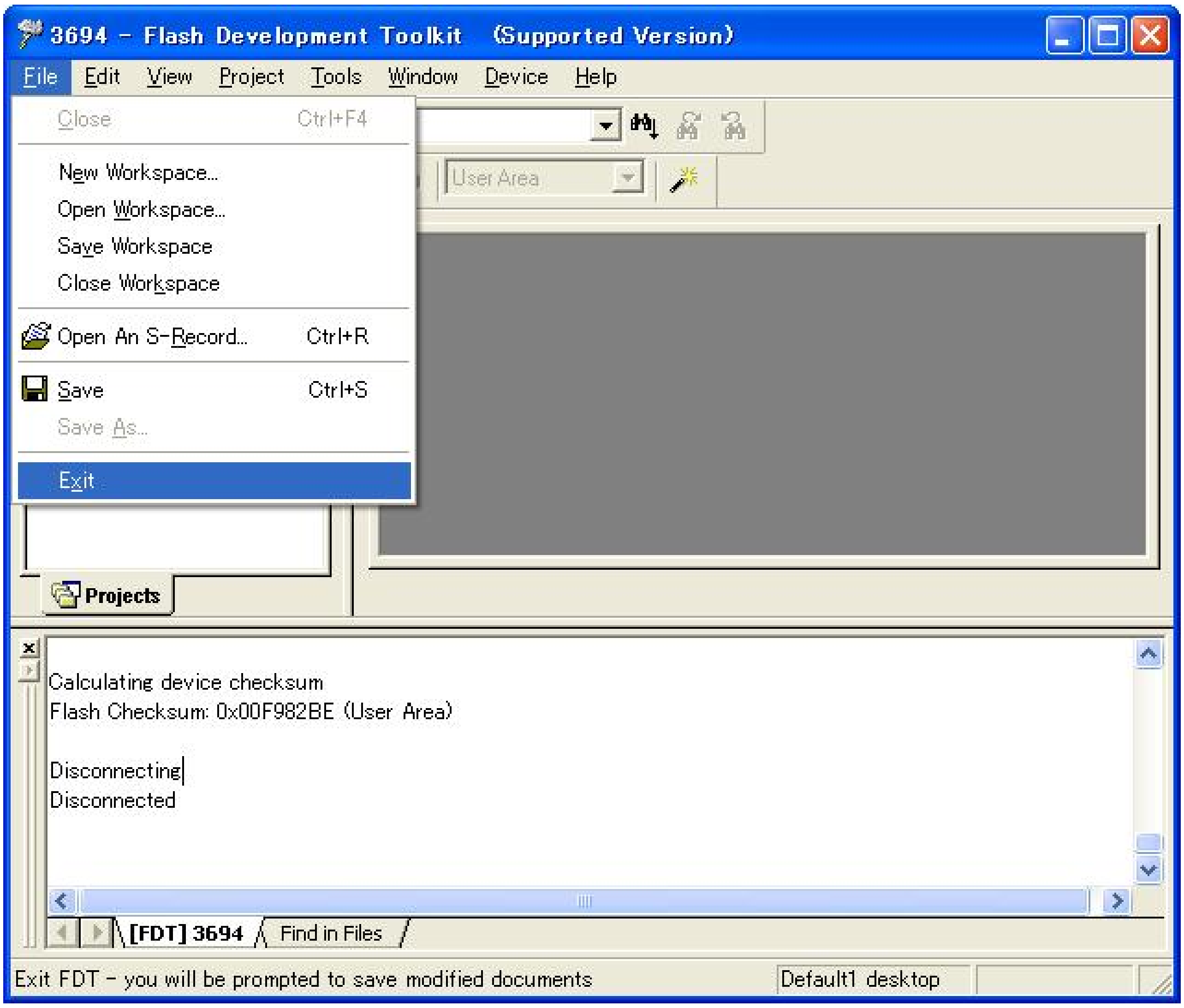
```
Calculating device checksum
Flash Checksum: 0x00F982BE (User Area)

Disconnecting
Disconnected
```

4.3.9 Exiting

Save the work folder and exit the Flash Development Toolkit.

Click "File" to open the pull-down menu and click "Exit."



```
Calculating device checksum
Flash Checksum: 0x00F982BE (User Area)

Disconnecting
Disconnected
```

Exit FDT - you will be prompted to save modified documents

The Flash Development Toolkit terminates.

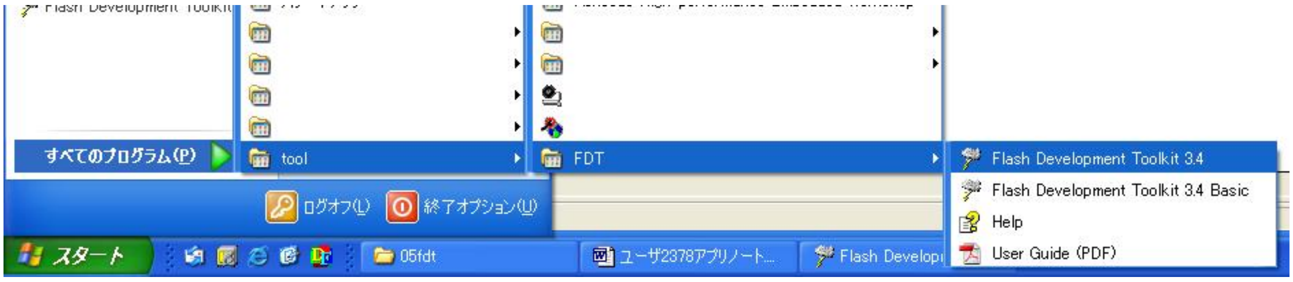
The work file space of the Flash Development Toolkit is saved as file 3694.AWS.

4.4 User Program Mode

In the user program mode, the user area can be programmed or erased.

4.4.1 Starting the Flash Development Toolkit

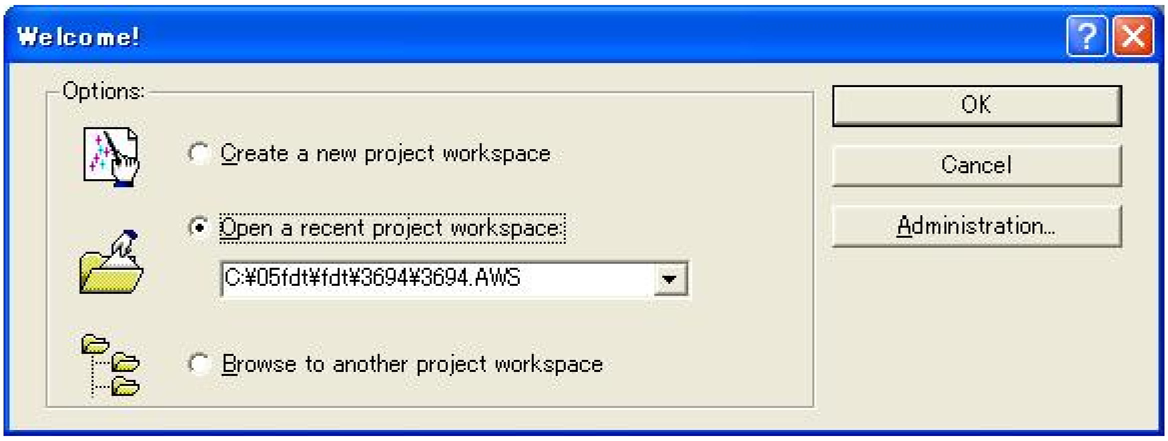
From the "All Programs" menu, select "Flash Development Toolkit 3.4."



4.4.2 Selecting an Option

The "Welcome!" screen of the Flash Development Toolkit appears.

Select "Open a recent project workspace" and project workspace file 3694.AWS.


Welcome!

Options:

Create a new project workspace

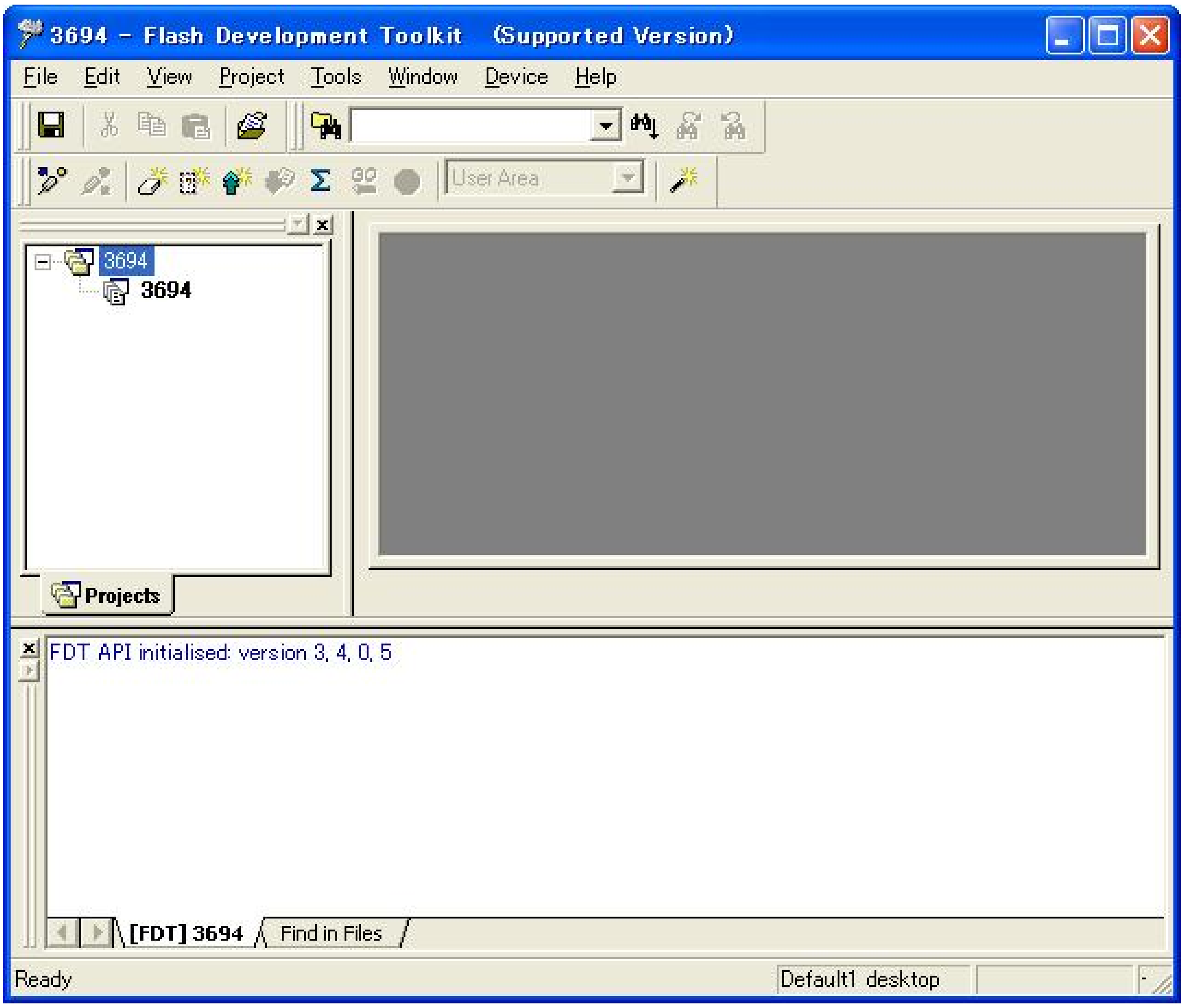
[✓ Open a recent project workspace:]

C:¥05fdt¥fdt¥3694¥3694.AWS

Browse to another project workspace

When you have selected an option, click "OK."

Project 3694 is displayed.



FDT API initialised: version 3, 4, 0, 5

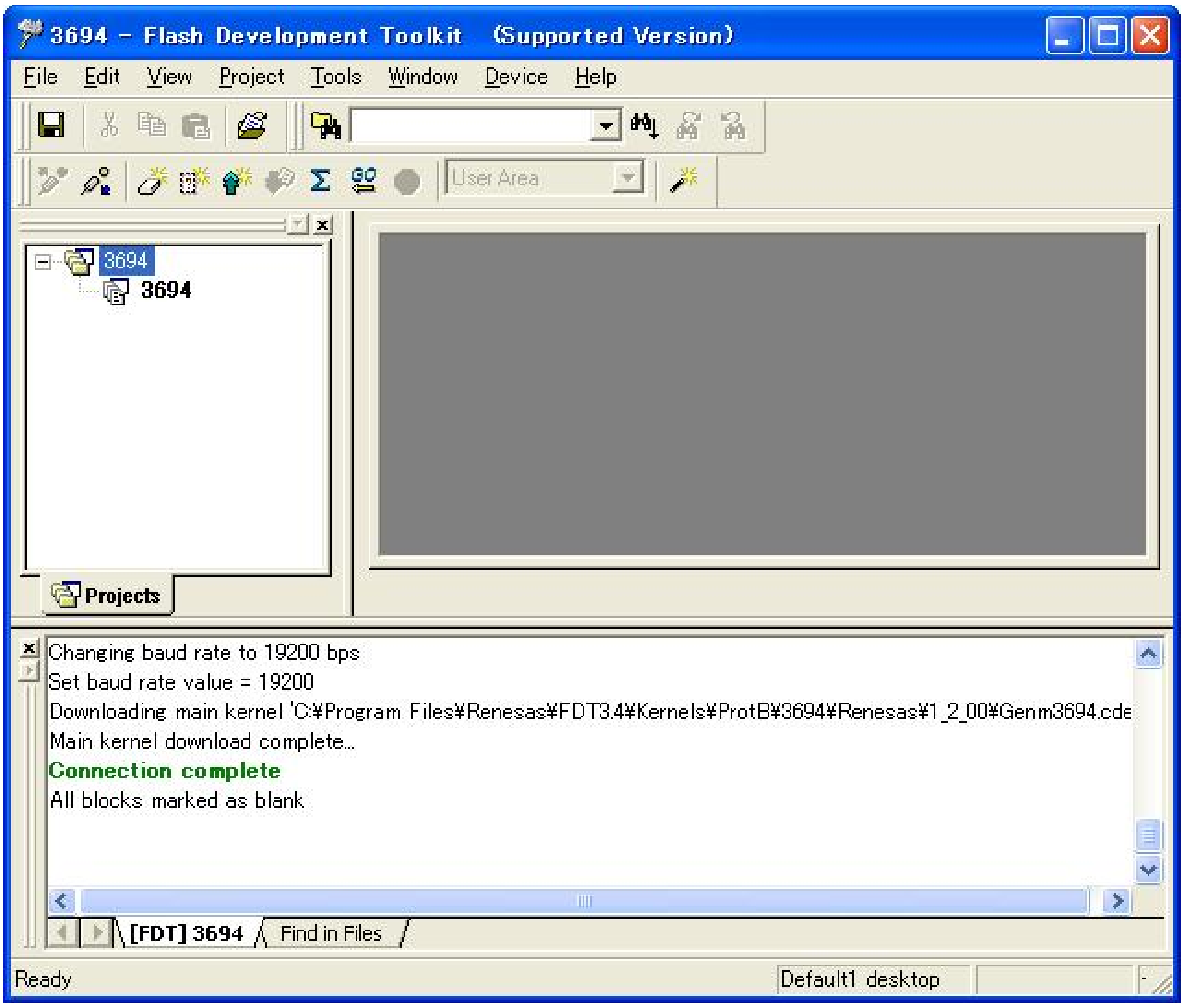
The Flash Development Toolkit can also be activated by directly opening (or double-clicking on) project workspace file 3694.AWS.

4.4.3 Connecting the Device

Connect the adapter board (FDM) to a PC and the H8/3694F board to the adapter board and turn on the power.

After the completion of the connection, click "Device" to open the pull-down menu and click "Connect to Device."

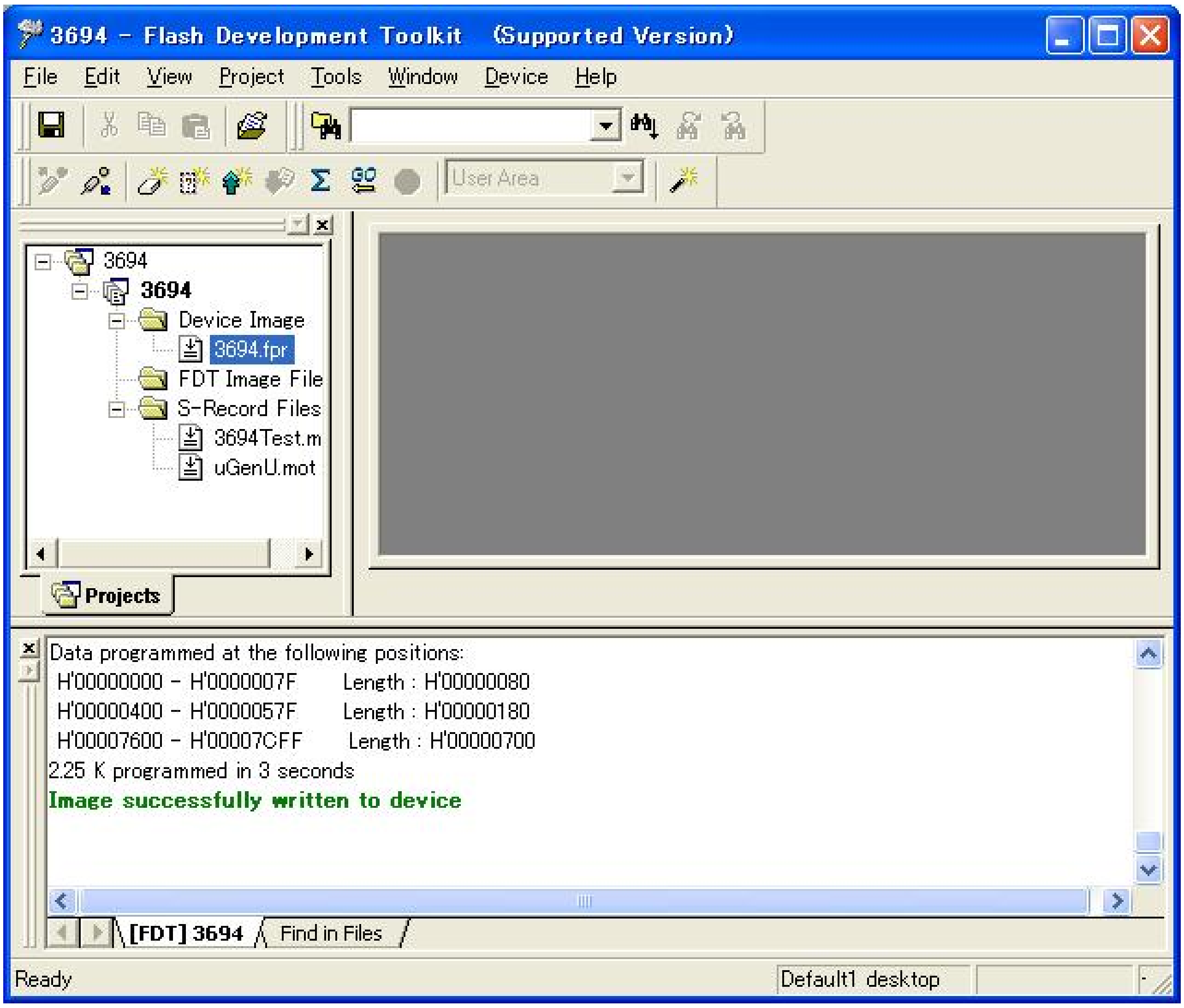
The device is connected.


3694 - Flash Development Toolkit (Supported Version)
File Edit View Project Tools Window Device Help
User Area
3694
3694
Projects
[FDT] 3694
Find in Files
Ready
Default1 desktop

```
Changing baud rate to 19200 bps
Set baud rate value = 19200
Downloading main kernel 'C:¥Program Files¥Renesas¥FDT3.4¥Kernels¥ProtB¥3694¥Renesas¥1_2_00¥Genm3694.cde
Main kernel download complete...
Connection complete
All blocks marked as blank
```

4.4.4 Writing a Program in the User Area

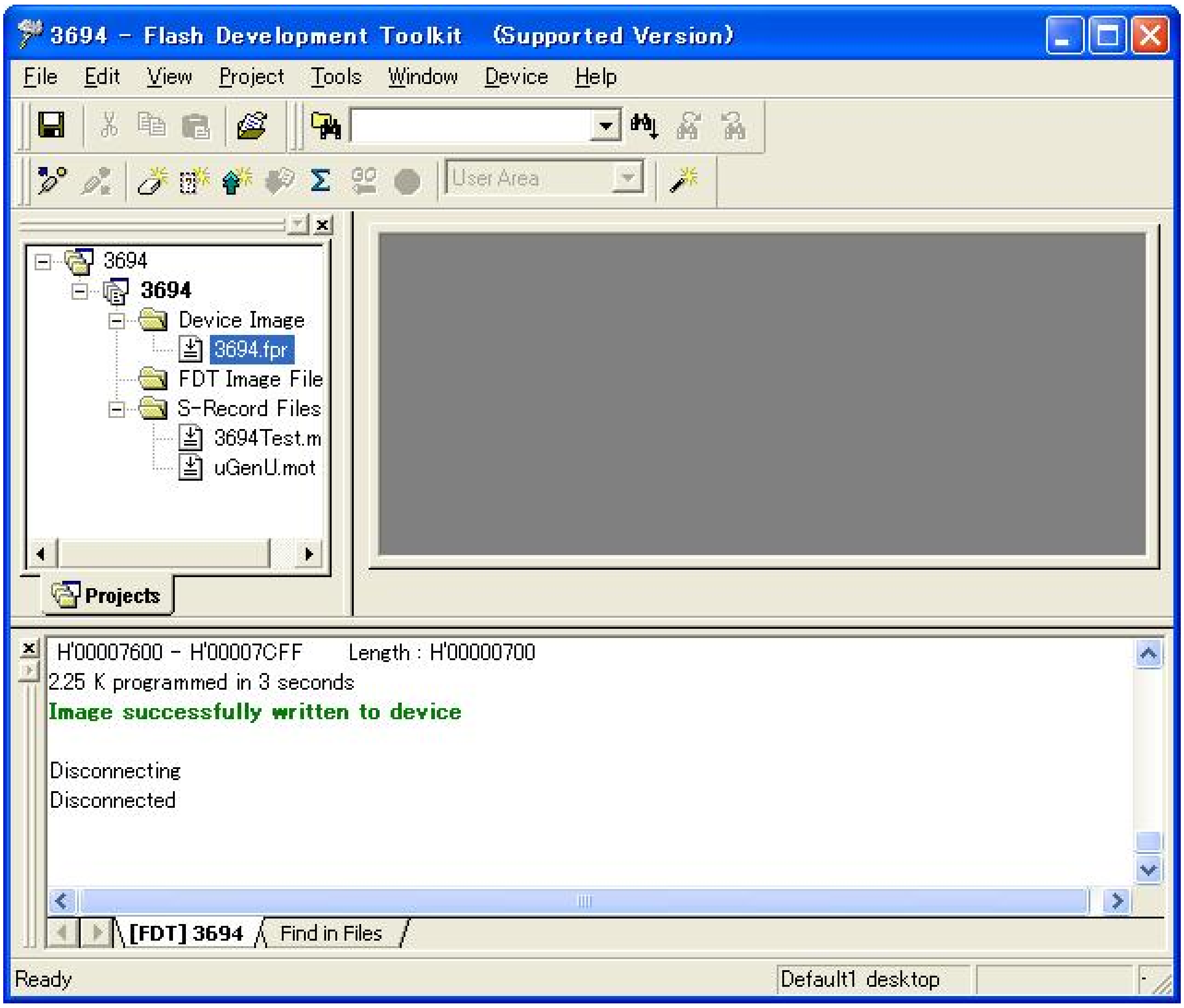
Add files 3694Test.mot and uGenU.mot and build the image to create file 3694.fpr. Then, download file 3694.fpr to write the program in the user area.



```
Data programmed at the following positions:
 H'00000000 - H'0000007F      Length : H'00000080
 H'00000400 - H'0000057F      Length : H'00000180
 H'00007600 - H'00007CFF      Length : H'00000700
2.25 K programmed in 3 seconds
Image successfully written to device
```

4.4.5 Disconnecting the Device

Click "Device" to open the pull-down menu and click "Disconnect."

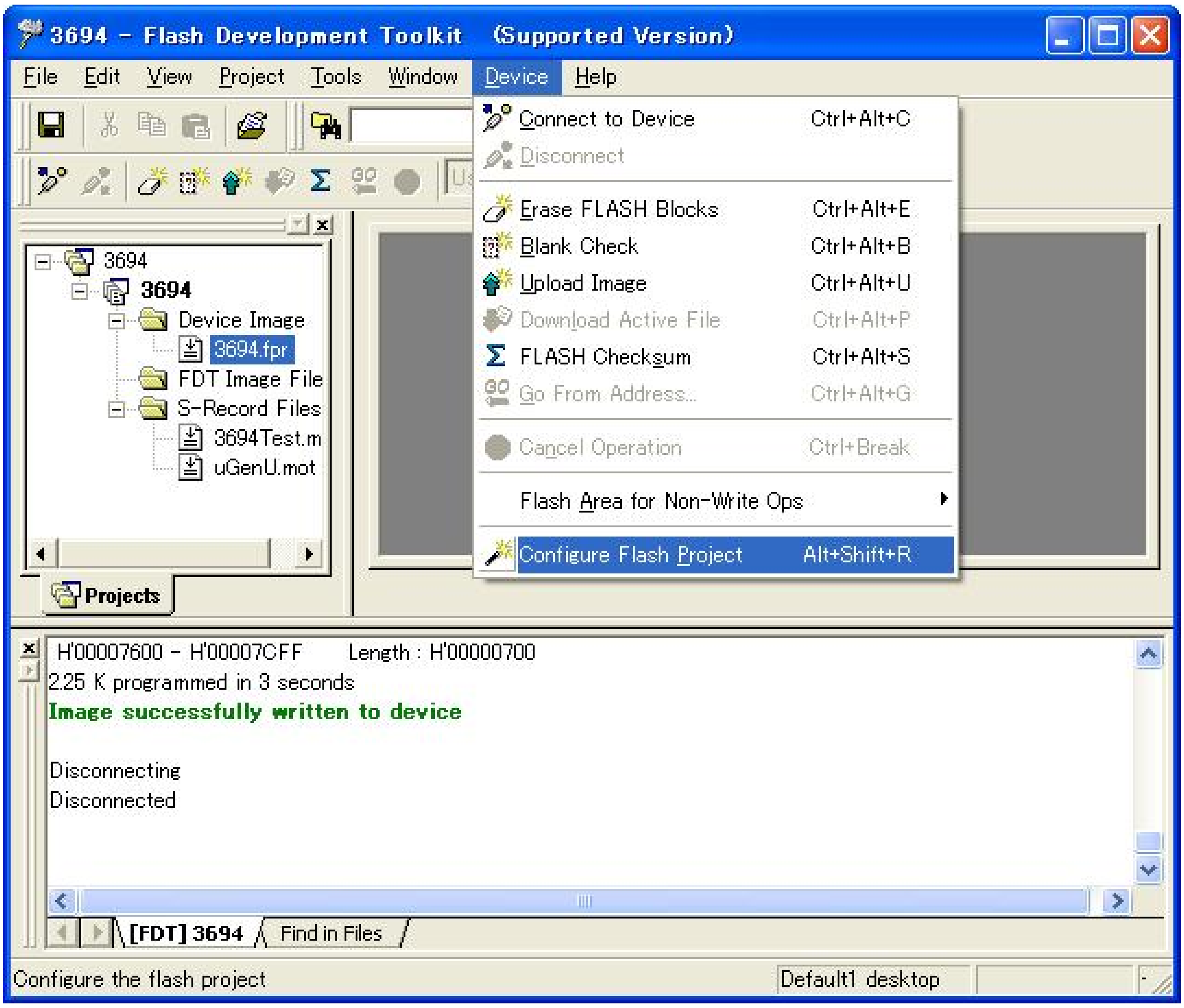


```
 H'00007600 - H'00007CFF      Length : H'00000700
2.25 K programmed in 3 seconds
Image successfully written to device

Disconnecting
Disconnected
```

4.4.6 Configuring the Project

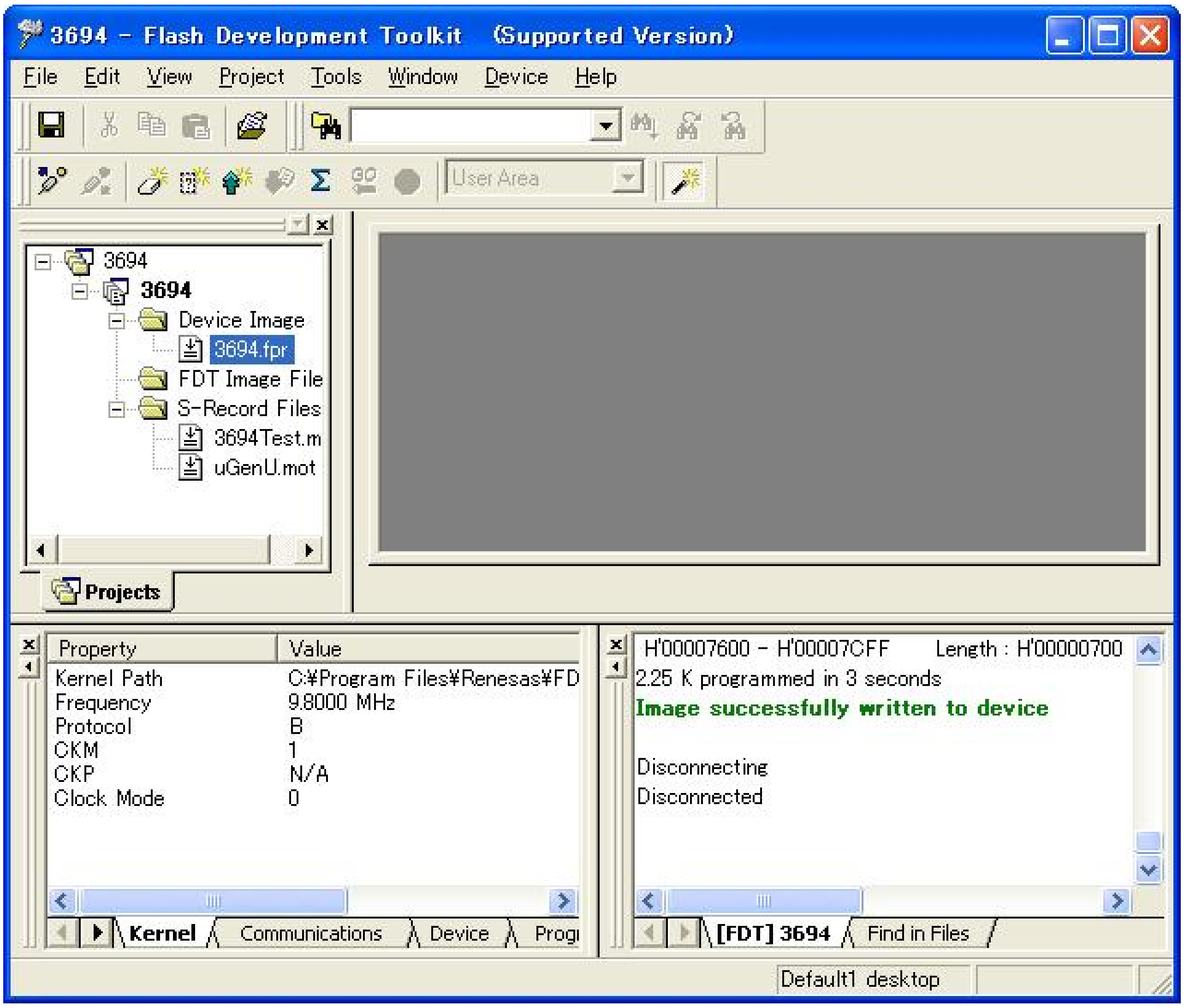
Click "Device" to open the pull-down menu and click "Configure Flash Project."


3694 - Flash Development Toolkit (Supported Version)
File
Edit
View
Project
Tools
Window
Device
Help
Connect to Device Ctrl+Alt+C
Disconnect
Erase FLASH Blocks Ctrl+Alt+E
Blank Check Ctrl+Alt+B
Upload Image Ctrl+Alt+U
Download Active File Ctrl+Alt+P
FLASH Checksum Ctrl+Alt+S
Go From Address... Ctrl+Alt+G
Cancel Operation Ctrl+Break
Flash Area for Non-Write Ops
Configure Flash Project Alt+Shift+R
3694
3694
Device Image
3694.fpr
FDT Image File
S-Record Files
3694Test.m
uGenU.mot
Projects
[FDT] 3694
Find in Files
Configure the flash project
Default1 desktop

```
H'00007600 - H'00007CFF     Length : H'00000700
2.25 K programmed in 3 seconds
Image successfully written to device

Disconnecting
Disconnected
```

The configure project window appears.



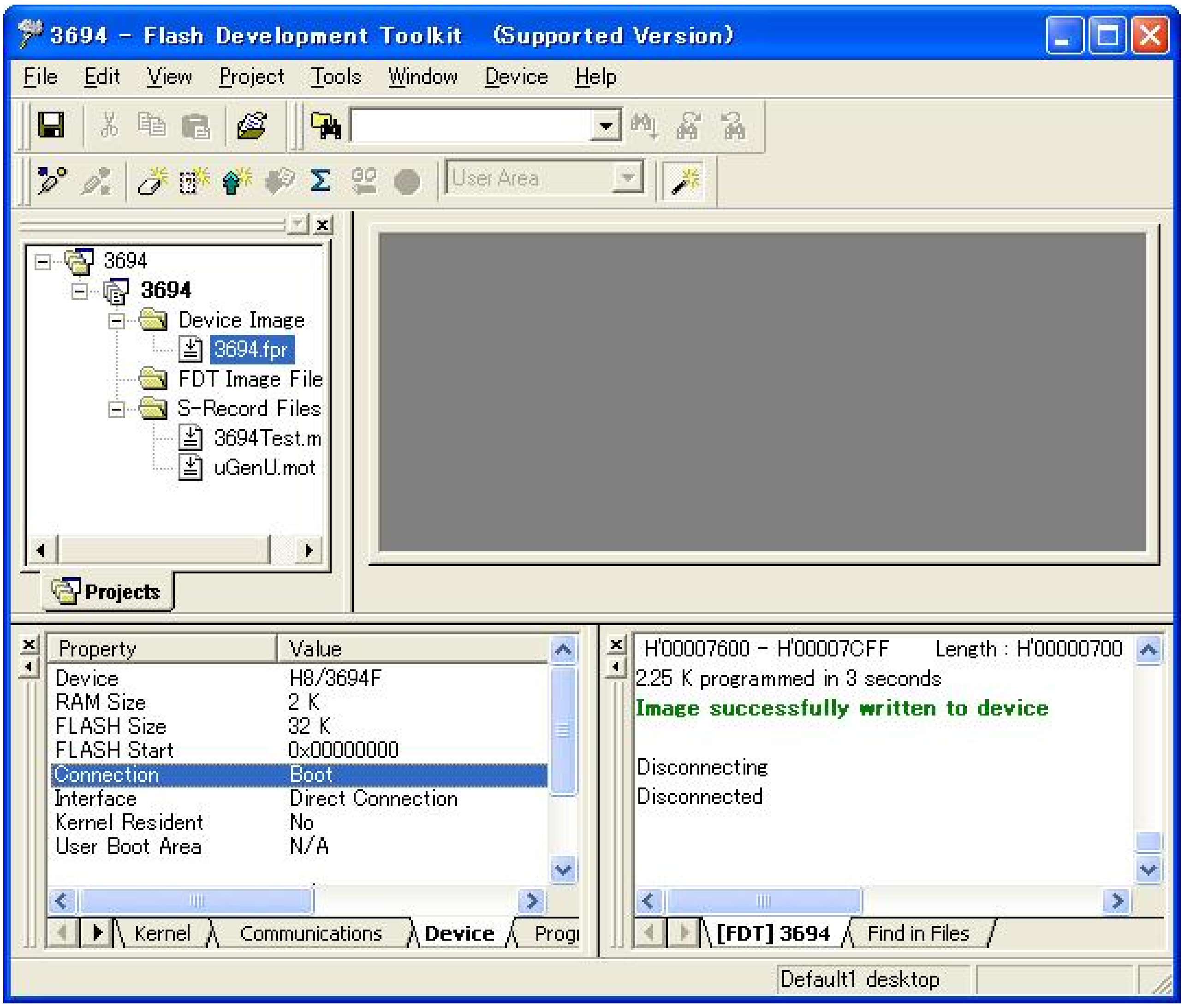
Property	Value
Kernel Path	C:¥Program Files¥Renesas¥FD
Frequency	9.8000 MHz
Protocol	B
CKM	1
CKP	N/A
Clock Mode	0

```
H'00007600 - H'00007CFF      Length : H'00000700
2.25 K programmed in 3 seconds
Image successfully written to device

Disconnecting
Disconnected
```

4.4.7 Setting User Program Mode

Select the "Device" tab in the configure project window and double-click "Connection" and "Boot."



Property	Value
Device	H8/3694F
RAM Size	2 K
FLASH Size	32 K
FLASH Start	0x00000000
Connection	Boot
Interface	Direct Connection
Kernel Resident	No
User Boot Area	N/A

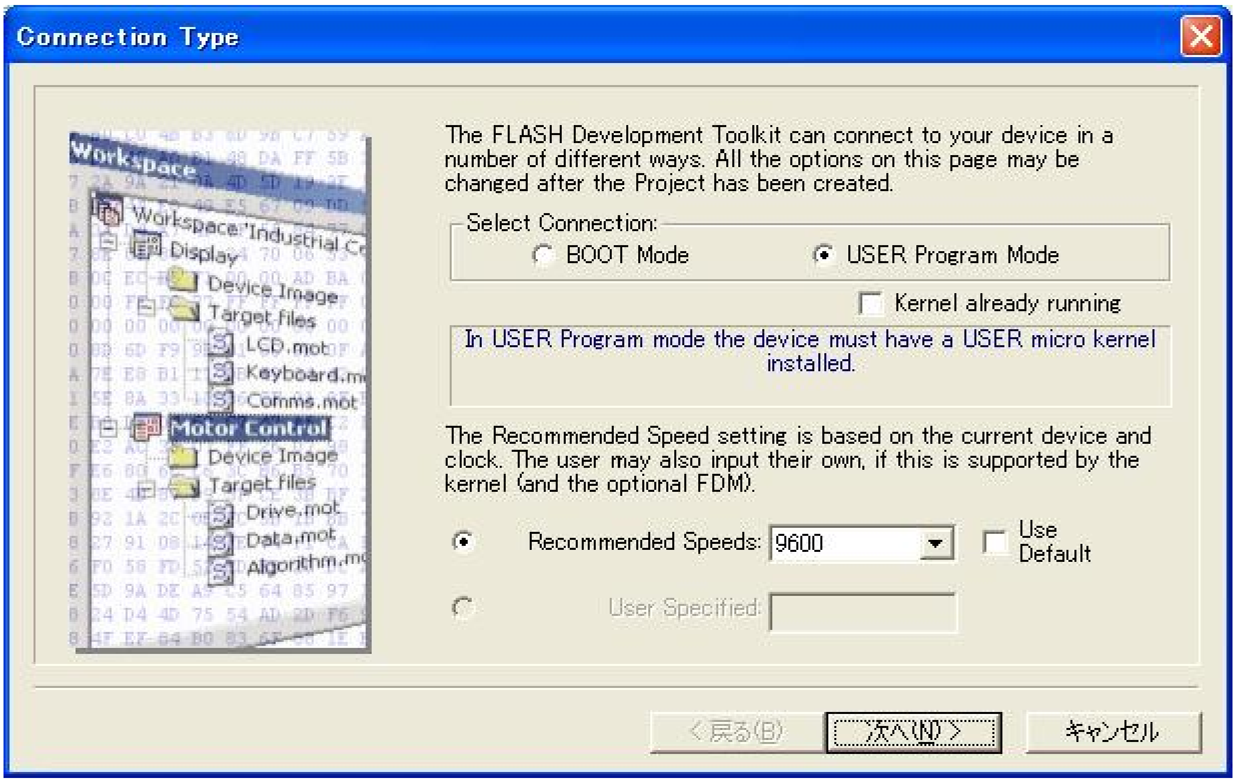
```
H'00007600 - H'00007CFF      Length : H'00000700
2.25 K programmed in 3 seconds
Image successfully written to device

Disconnecting
Disconnected
```

Set the connection type.

Select "USER Program Mode" in "Select Connection:."

Set the baud rate to 9600 bps.



Connection Type

The FLASH Development Toolkit can connect to your device in a number of different ways. All the options on this page may be changed after the Project has been created.

Select Connection: BOOT Mode [✓ USER Program Mode]

Kernel already running

In USER Program mode the device must have a USER micro kernel installed.

The Recommended Speed setting is based on the current device and clock. The user may also input their own, if this is supported by the kernel (and the optional FDM).

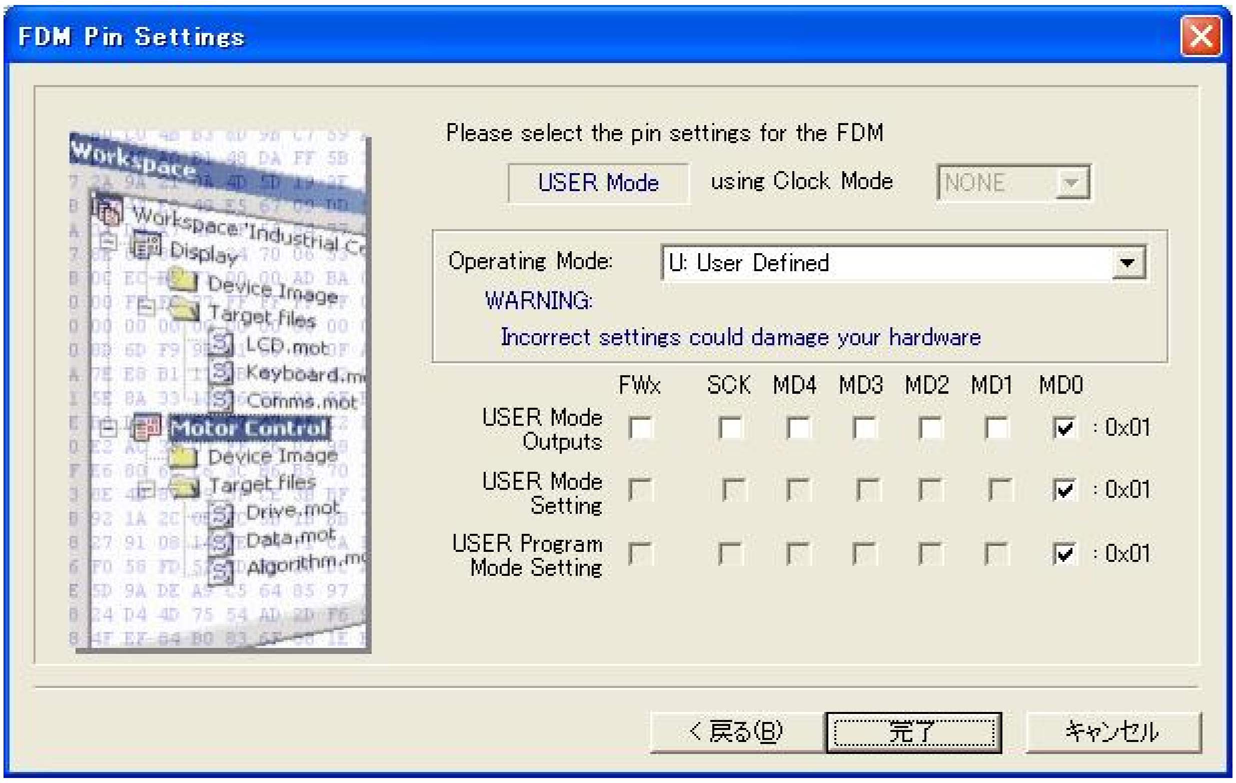
[✓ Recommended Speeds: 9600] Use Default

User Specified:

When you have selected the baud rate, click "Next."

Set the pins on the adapter board (FDM) for the user program mode.

In the H8/3694F user program mode, set the output of $\overline{\text{NMI}}$ to high (1). On the H8/3694F user system, MD0 is connected to $\overline{\text{NMI}}$. For this reason, set the output of MD0 to high (1). No FWE pin setting is required because no FWE pin is given.



Operating Mode: U: User Defined

WARNING:
Incorrect settings could damage your hardware

	FWx	SCK	MD4	MD3	MD2	MD1	MD0	
USER Mode Outputs							✓	: 0x01
USER Mode Setting							✓	: 0x01
USER Program Mode Setting							✓	: 0x01

When you have set the pin, click "Finish."

An example of connecting the H8/3694F and Renesas adapter board (HS0008EAUF1H) is shown in Figure 4-4. The pull-up and pull-down resistor values shown are only examples. Evaluate the microcomputer to determine the actual values on the user system.

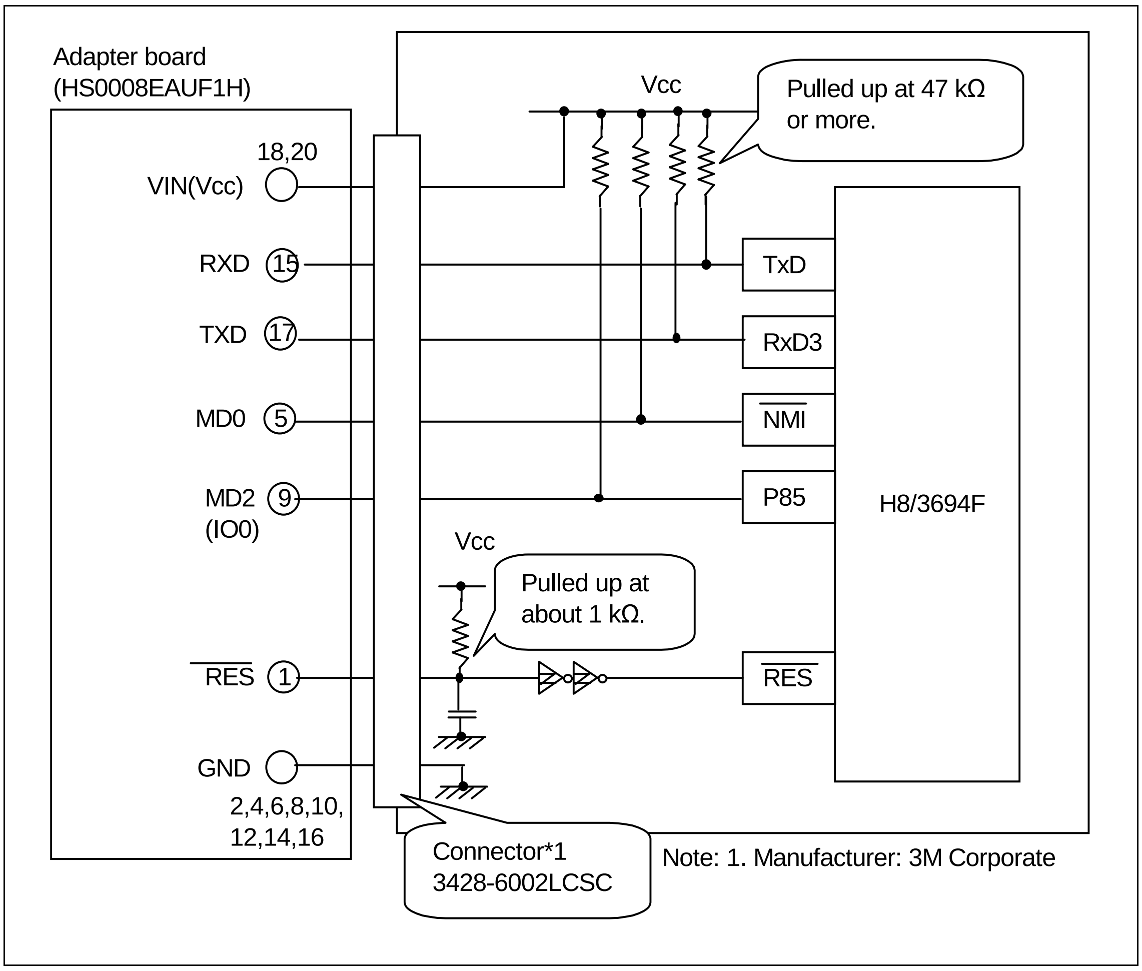


Figure 4-4 Example of Connecting the H8/3694F and Adapter Board

An example of setting pins for the user program mode when the H8/3694F user system and Renesas adapter board (HS0008EAUF1H) is shown in Table 4-5.

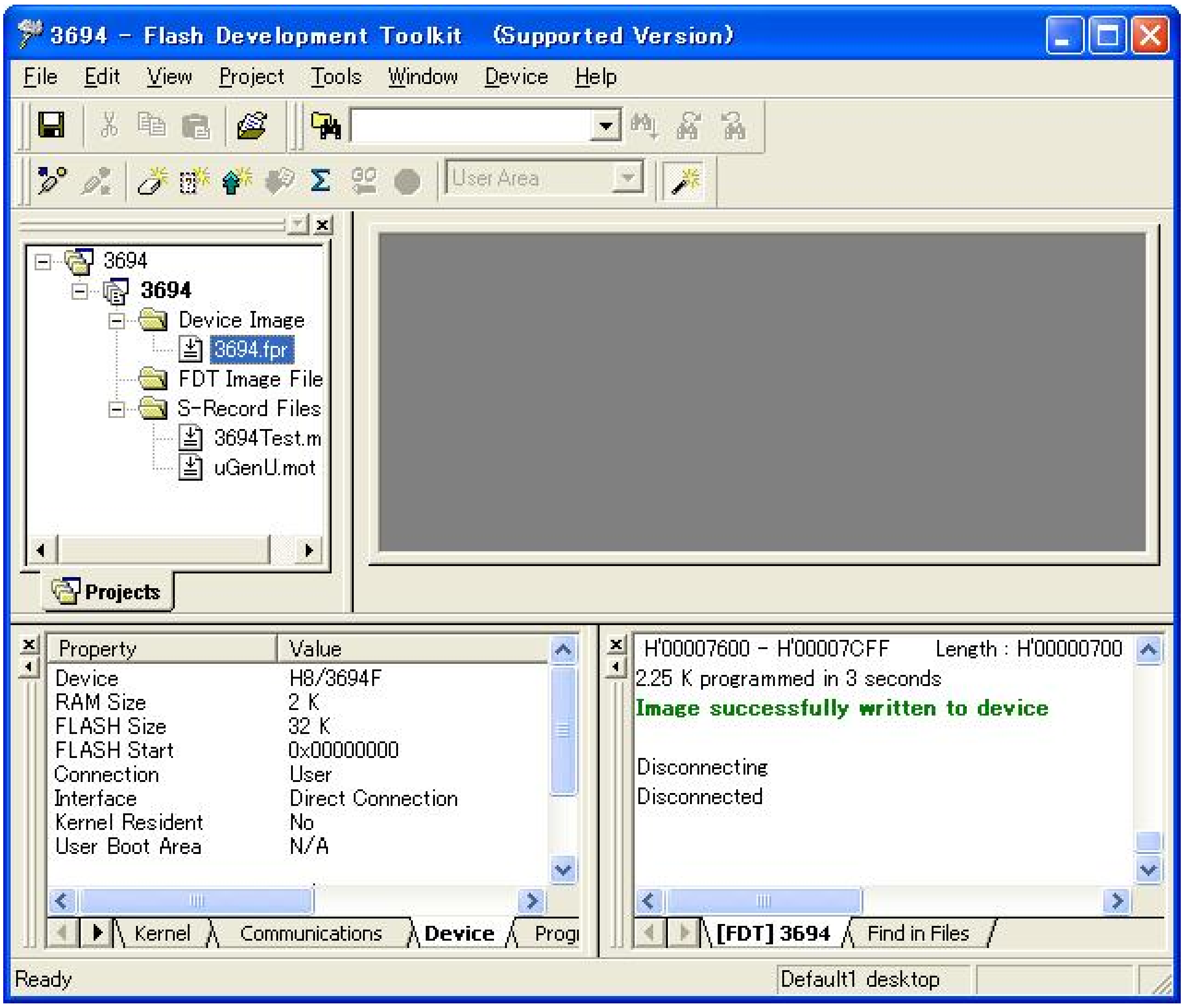
Table 4-5 Example of Setting Pins on the H8/3694F and Adapter Board (for the User Program Mode)

Pin No.	Pin on the Adapter Board	Pin on the Device	Input/Output	Output Level
1	$\overline{\text{RES}}$	$\overline{\text{RES}}$	Output (default)	Adapter board
3	FWx	NC	NC	-
5	MD0	$\overline{\text{NMI}}$	Output	High (1)
7	MD1	NC	NC	-
9	MD2 (IO0)	P85	Input	-
11	MD3 (IO1)	NC	NC	-
13	MD4 (IO2)	NC	NC	-
15	RXD	TXD	Input (default)	Adapter board
17	TXD	RXD	Output (default)	Adapter board
19	SCK (NC)	NC	NC (default)	-

Note: NC: Means no connection.

4.4.8 Completion of Setting

The user program mode has been set.



Property	Value
Device	H8/3694F
RAM Size	2 K
FLASH Size	32 K
FLASH Start	0x00000000
Connection	User
Interface	Direct Connection
Kernel Resident	No
User Boot Area	N/A

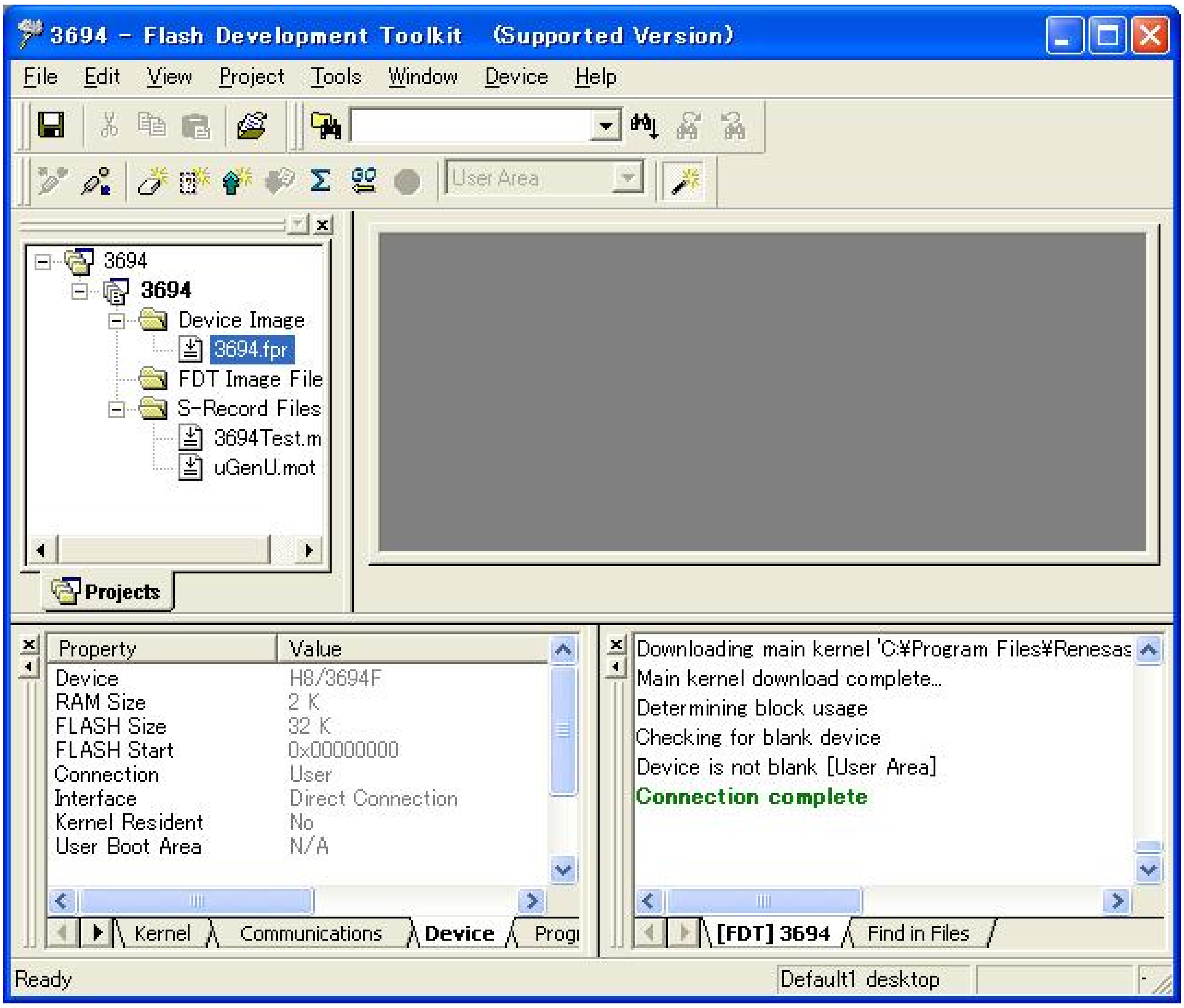
```
H'00007600 – H'00007CFF    Length : H'00000700
2.25 K programmed in 3 seconds
Image successfully written to device

Disconnecting
Disconnected
```

4.4.9 Connecting the Device

After the completion of the setting, click "Device" to open the pull-down menu and click "Connect to Device."

The connection in the user program mode is completed.

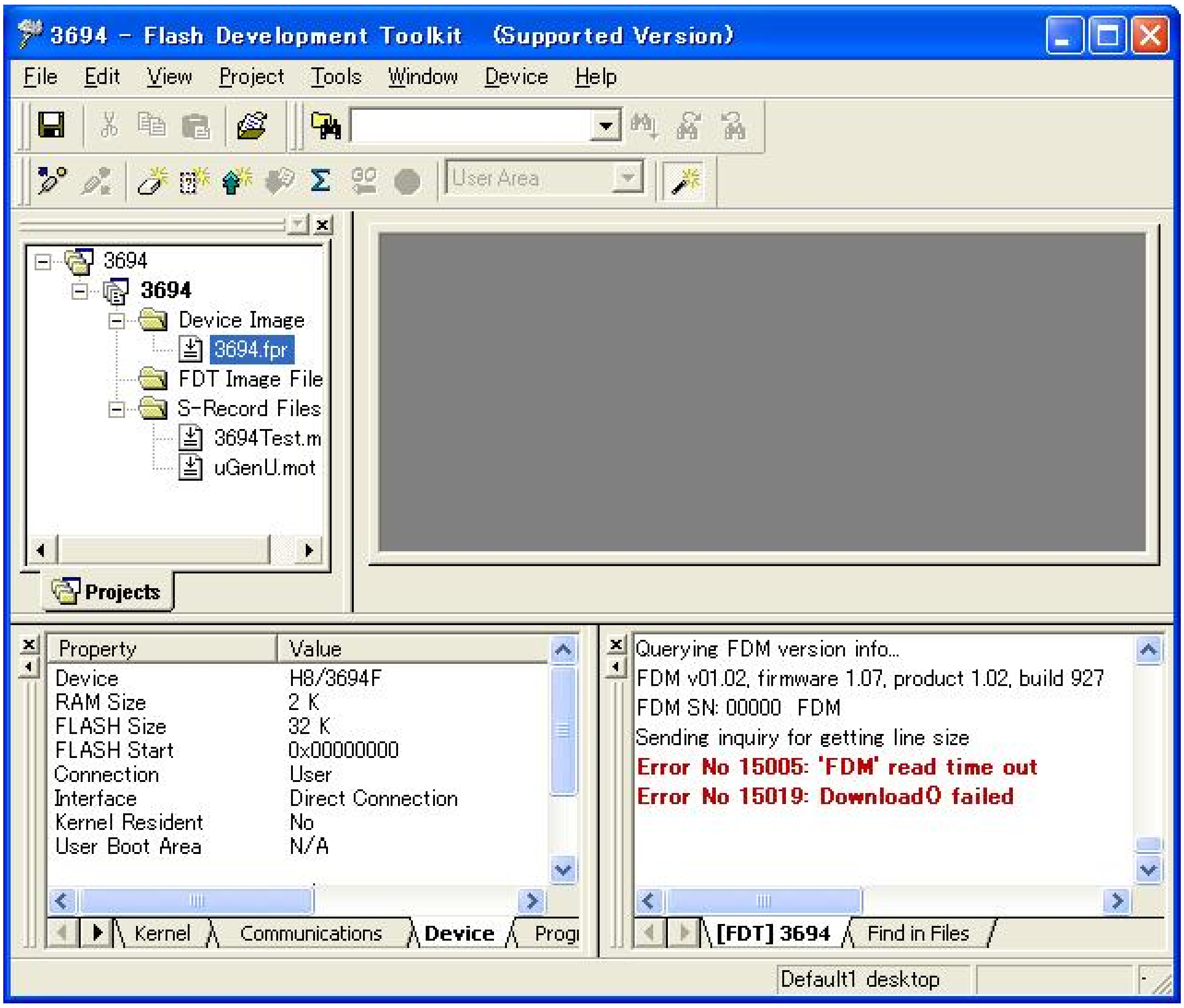


Property	Value
Device	H8/3694F
RAM Size	2 K
FLASH Size	32 K
FLASH Start	0x00000000
Connection	User
Interface	Direct Connection
Kernel Resident	No
User Boot Area	N/A

```
Downloading main kernel 'C:¥Program Files¥Renesas
Main kernel download complete...
Determining block usage
Checking for blank device
Device is not blank [User Area]
Connection complete
```

4.4.10 Timeout

A timeout error may occur during an attempt to connect the device.



Property	Value
Device	H8/3694F
RAM Size	2 K
FLASH Size	32 K
FLASH Start	0x00000000
Connection	User
Interface	Direct Connection
Kernel Resident	No
User Boot Area	N/A

```
Querying FDM version info...
FDM v01.02, firmware 1.07, product 1.02, build 927
FDM SN: 00000  FDM
Sending inquiry for getting line size
Error No 15005: 'FDM' read time out
Error No 15019: Download() failed
```

There are several possible causes. Either of the following operations may not be performed. Check them.

(1) Modify the bit rate in the sample program to 9600 bps.

For how to modify the bit rate, refer to section 7.1.1, Bit Rate Setting (GenTest.h).

(2) Connect the serial input to the I/O bus (26P) (J2).

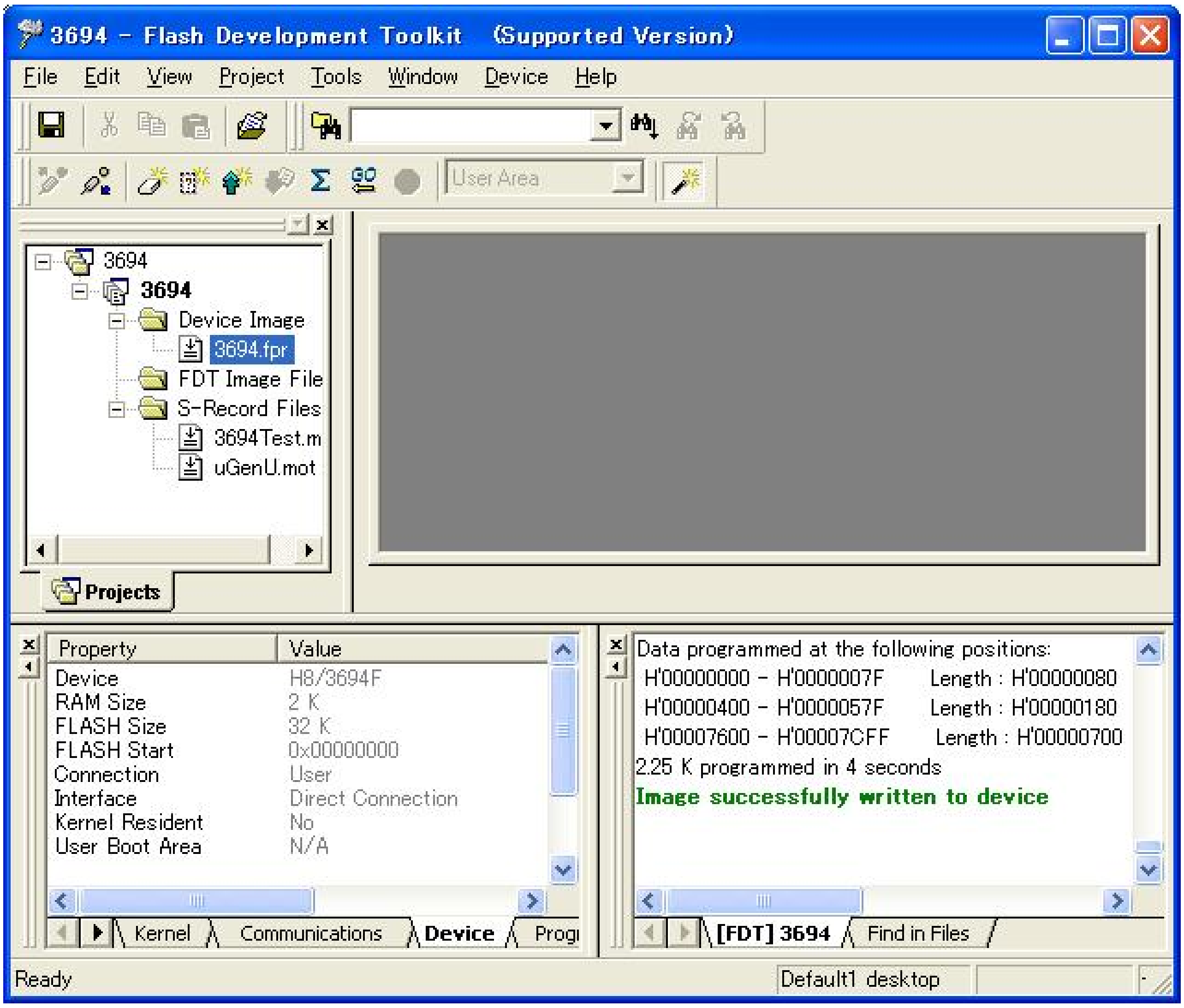
For details on connection to the I/O bus, refer to section **エラー! 参照元が見つかりません。**, **エラー! 参照元が見つかりません。**.

4.4.11 Programming

Write a program in the user area in the user program mode.

Click the right mouse button on file 3694.fpr to display the pop-up menu. Click "Download User Image" to download file 3694.fpr to the user area.

You can check that the program has been downloaded to the user area.



Property	Value
Device	H8/3694F
RAM Size	2 K
FLASH Size	32 K
FLASH Start	0x00000000
Connection	User
Interface	Direct Connection
Kernel Resident	No
User Boot Area	N/A

```
Data programmed at the following positions:
 H'00000000 - H'0000007F      Length : H'00000080
 H'00000400 - H'0000057F      Length : H'00000180
 H'00007600 - H'00007CFF      Length : H'00000700
2.25 K programmed in 4 seconds
Image successfully written to device
```

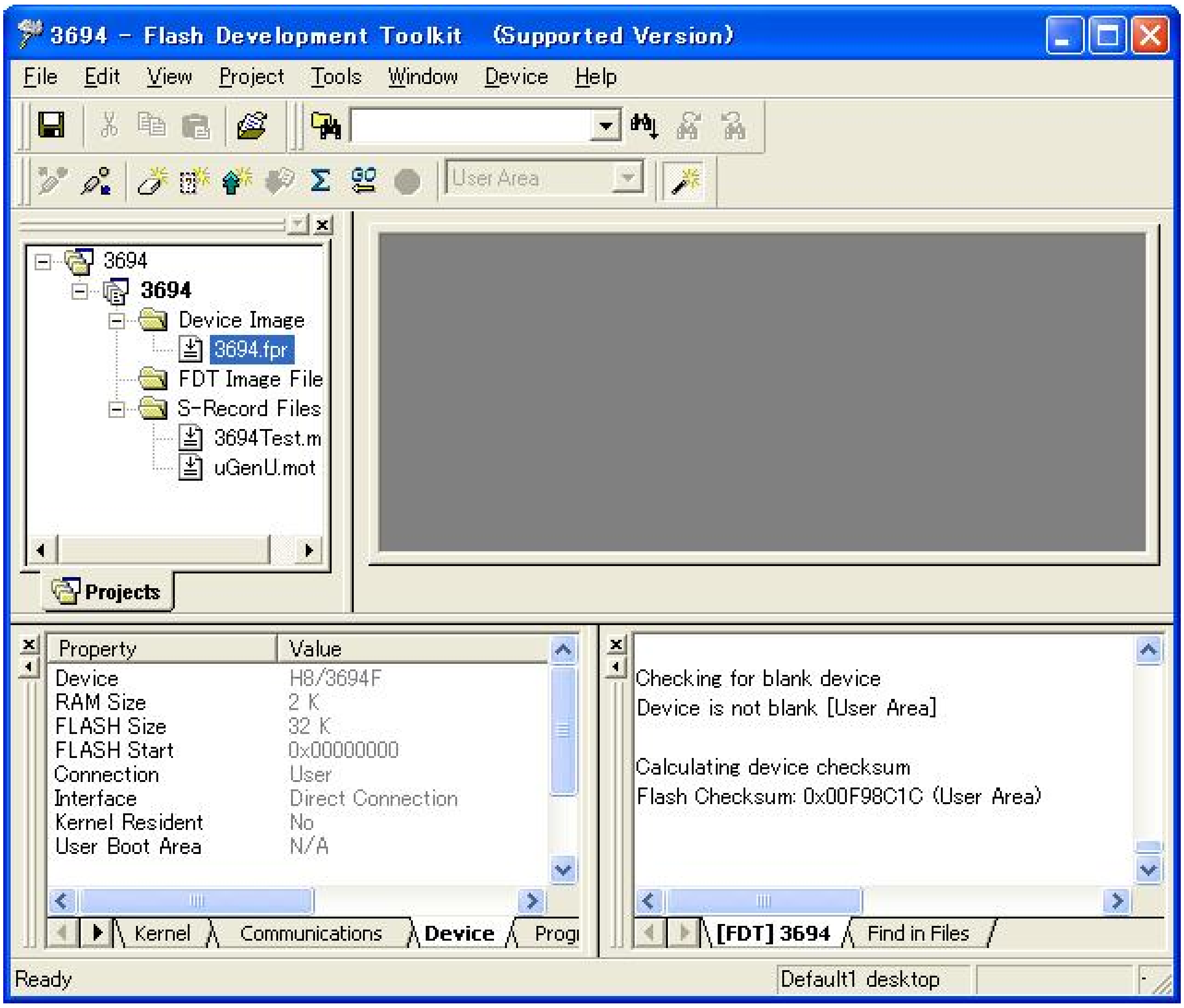
4.4.12 Blank Check and Checksum

To confirm that the user area has been programmed, perform a blank check and calculate a checksum.

Click "Device" to open the pull-down menu and click "Blank Check."

Click "Device" to open the pull-down menu and click "FLASH Checksum."

The results of the blank check and checksum calculation are displayed.



Property	Value
Device	H8/3694F
RAM Size	2 K
FLASH Size	32 K
FLASH Start	0x00000000
Connection	User
Interface	Direct Connection
Kernel Resident	No
User Boot Area	N/A

```
Checking for blank device
Device is not blank [User Area]

Calculating device checksum
Flash Checksum: 0x00F98C1C (User Area)
```

5. Flash Development Toolkit Processing

The Flash Development Toolkit can be connected in either of the following two modes: the boot mode and user program mode. In both modes, the continuation of the execution from a previous session (direct connection to the main kernel) can be specified. The connection modes of the Flash Development Toolkit are listed in Table 5-1. Usually, use new connection processing. The hexadecimal codes are the command codes of the Flash Development Toolkit.

Table 5-1 Connection Modes of the Flash Development Toolkit

Mode	Normal Processing	Continuation of the Execution from a Previous Session
Boot mode	Baud rate adjustment Micro kernel transfer H'27 (Programming unit inquiry) H'10 (Device selection) H'11 (Clock mode selection) H'3F (New baud rate setting) Main kernel transfer	H'27 (Programming unit inquiry) H'4F (Status request) H'4D (User area blank check)
User program mode	H'27 (Programming unit inquiry) H'10 (Device selection) H'11 (Clock mode selection) H'3F (New baud rate setting) Main kernel transfer	H'27 (Programming unit inquiry) H'4F (Status request) H'4D (User area blank check)

6. Sample Program

This section describes the sample program in the user program mode of the H8/3694F.

6.1 Program Configuration

The configuration of the sample program is shown in Table 6-1.

Table 6-1 Program Configuration

No.	Program	Function	Location and Startup
1	Main processing module	Causes a branch to the micro kernel.	Stored in ROM in the boot mode in advance. Initiated by a reset.
2	Micro kernel	Processes inquiry and selection commands. Transfers the transmit and receive modules to RAM. Receives the main kernel and stores it in RAM. Causes a branch to the main kernel.	Stored in ROM in the boot mode in advance. Branches from the main processing module.
3	Main kernel	Processes programming, erasing, and check commands. Receives the programming or erasing program and stores it in RAM. Calls the programming or erasing program.	Received by the micro kernel and stored in RAM. Branches from the micro kernel.
4	Programming kernel	Programs flash memory.	Received by the main kernel and stored in RAM. Called from the main kernel.
5	Erasing kernel	Erases flash memory.	Received by the main kernel and stored in RAM. Called from the main kernel.

6.2 File Configuration

The program files are contained in the C:\Program Files\Renesas\FDT3.4\Kernels\ProtB\3694\Renesas\1_2_00 folder. The file configuration of each program module is shown below. These program modules are provided as a sample of a program in the user program mode that is to be created uniquely by the user.

6.2.1 Main Processing Module

Table 6-2 File Configuration of the Main Processing Module

No.	File Name	Description	Type
1	3694Test.mot	Load module of the main processing module in the user program mode	S-type format
2	bTest.bat	Batch file of the main processing module in the user program mode	MS-DOS batch file
3	l3694t.xcl	Link subfile of the main processing module in the user program mode	Linkage editor command file
4	Strt3694.src	Stack initial setting source file	Assembly language source file
5	GenTest.c	Main source file of the main processing module in the user program mode	C source file
6	GenTest.h	GenTest.c function prototype declaration	C header file
7	io3694.h	SCI and port register definition	C header file
8	KDevice.h	Device-specific information (such as kernel location definition)	C header file
9	KStruct.h	Structure definition and other information	C header file
10	KTypes.h	Type definition and other information	C header file

6.2.2 Micro Kernel

Table 6-3 File Configuration of the Micro Kernel

No.	File Name	Description	File Type
1	uGenU.mot	Load module of the micro kernel in the user program mode	S-type format
2	ub3694u.bat	Batch file of the micro kernel in the user program mode	MS-DOS batch file
3	ul3694u.xcl	Link subfile of the micro kernel in the user program mode	Linkage editor command file
4	uGenu.c	Main source file of the micro kernel in the user program mode	C source file
5	CmdFunc.c	Command function source file	C source file (common)
6	CmdFunc.h	CmdFunc.c function prototype declaration	C header file
7	Commands.h	Command ID definition	C header file
8	DeviceInfo.h	Device-specific information (inquiry/response data)	C header file
9	Extern.h	External reference definition for functions and variables	C header file
10	uGenu.h	uGenu.c function prototype declaration	C header file
11	io3694.h	SCI and port register definition	C header file
12	KDevice.h	Device-specific information (such as kernel location definition)	C header file
13	KStruct.h	Structure definition and other information	C header file
14	KTypes.h	Type definition and other information	C header file
15	H8runtime.lib	Runtime library	Required for re-creating the kernel. (This file is not provided. Use a library configuration tool to create the file.) Note: Refer to BuildAll.bat.

6.2.3 Main Kernel

Table 6-4 File Configuration of the Main Kernel

No.	File Name	Description	File Type
1	Genu3694.cde	Load module of the main kernel in the user program mode	Binary format
2	b3694u.bat	Batch file of the main kernel in the user program mode	MS-DOS batch file
3	l3694u.xcl	Link subfile of the main kernel in the user program mode	Linkage editor command file
4	FDTUMain.c	Source file of the main kernel in the user program mode	C source file
5	CopyFunc.c	Source file of the copy function of the main kernel in the user program mode	C source file
6	CmdFunc.c	Command function source file	C source file (common)
7	CmdFunc.h	CmdFunc.c function prototype declaration	C header file
8	Commands.h	Command ID definition	C header file
9	DeviceInfo.h	Device-specific information (inquiry/response data)	C header file
10	Extern.h	External reference definition for functions and variables	C header file
11	FDTBMain.h	FDTBMain.c function prototype declaration	C header file
12	io3694.h	SCI and port register definition	C header file
13	KDevice.h	Device-specific information (such as kernel location definition)	C header file
14	KStruct.h	Structure definition and other information	C header file
15	KTypes.h	Type definition and other information	C header file

6.2.4 Programming Kernel

Table 6-5 File Configuration of Programming Kernel Main

No.	File Name	Description	File Type
1	Genw3694.cde	Load module of the programming kernel	Binary format
2	b3694w.bat	Batch file of the programming kernel	MS-DOS batch file
3	l3694w.xcl	Link subfile of the programming kernel	Linkage editor command file
4	F3694w.src	Source file of the programming kernel	Assembly language source file
5	FDTWrite.c	Main source file of the programming kernel	C source file
6	WriteTime.c	Source file of programming wait time calculation	C source file
7	Commands.h	Command ID definition	C header file
8	F3694asm.inc	Flash memory register definition and other information	Assembly language include file
9	io3694.h	SCI and port register definition	C header file
10	KAlg.h	Programming/erasing function definition	C header file
11	KDevice.h	Device-specific information (such as kernel location definition)	C header file
12	KStruct.h	Structure definition and other information	C header file
13	KTypes.h	Type definition and other information	C header file
14	H8runtime.lib	Runtime library	Required for re-creating the kernel. (This file is not provided. Use a library configuration tool to create the file.) Note: Refer to BuildAll.bat.

6.2.5 Erasing Kernel

Table 6-6 File Configuration of the Erasing Kernel

No.	File Name	Description	File Type
1	Gene3694.cde	Load module of the erasing kernel	Binary format
2	b3694e.bat	Batch file of the erasing kernel	MS-DOS batch file
3	l3694e.xcl	Link subfile of the erasing kernel	Linkage editor command file
4	F3694e.src	Source file of the erasing kernel	Assembler source file
5	FDTErase.c	Source file of the erasing kernel	C source file
6	EraseTime.c	Source file of erasing wait time calculation	C source file
7	Commands.h	Command ID definition	C header file
8	F3694asm.inc	Flash memory register definition and other information	Assembler include file
9	io3694.h	SCI and port register definition	C header file
10	KAlg.h	Programming/erasing function definition	C header file
11	KDevice.h	Device-specific information (such as kernel location definition)	C header file
12	KStruct.h	Structure definition and other information	C header file
13	KTypes.h	Type definition and other information	C header file
14	H8runtime.lib	Runtime library	Required for re-creating the kernel. (This file is not provided. Use a library configuration tool to create the file.) Note: Refer to BuildAll.bat.

6.3 Relationships between Program Modules and Files

The relationships between program modules and files are given in Table 6-7.

Table 6-7 Relationships between Program Modules and Files

Program Name	Batch File	Source Files	Header Files	Subcommand File	Library File	Output File
Main processing module (User program mode)	bTest.bat	GenTest.c strt3694.src	GenTest.h io3694.h KDevice.h KStruct.h KTypes.h	l3694t.xcl	–	3694Test.mot
Micro kernel (User program mode)	ub3694u.bat	Ugenu.c CmdFunc.c	Ugenu.h CmdFunc.h Commands.h DeviceInfo.h Extern.h io3694.h KDevice.h KStruct.h KTypes.h	ul3694u.xcl	H8runtime.lib	uGenU.mot
Main kernel (User program mode)	b3694u.bat	FDTUMain.c CopyFunc.c CmdFunc.c	FDTUMain.h CmdFunc.h Commands.h DeviceInfo.h Extern.h io3694.h KDevice.h KStruct.h KTypes.h	l3694u.xcl	–	Genu3694.cde
Programming kernel	b3694w.bat	FDTWrite.c WriteTime.c F3694w.src	F3694asm.inc Commands.h io3694.h KAlg.h KDevice.h KStruct.h KTypes.h	l3694w.xcl	H8runtime.lib	Genw3694.cde
Erasing kernel	b3694e.bat	FDTErase.c EraseTime.c F3694e.src	F3694asm.inc Commands.h io3694.h KAlg.h KDevice.h KStruct.h KTypes.h	l3694e.xcl	H8runtime.lib	Gene3694.cde
Micro kernel (Boot mode)	ub3694.bat	Ugen.c CmdFunc.c strt3694.src	Ugen.h CmdFunc.h	ul3694.xcl	–	uGen3694.cde
Main kernel (Boot mode)	b3694m.bat	FDTBMain.c CmdFunc.c	FDTBMain.h CmdFunc.h	l3694m.xcl	–	Genm3694.cde
All build batch file	BuildAll.bat	–		–	H8runtime.lib	–

Note: Boot mode program modules and all build batch files are included.

6.4 Build Operation

Build operation is not required when the provided program is used. When re-creation is required due to such as the use of a different operating frequency, build operation is required.

Executing build operation deletes all generated files. Create a copy, then execute build operation because a current file may be required.

6.4.1 SET Command

Before executing build operation, set the environment. Insert the following commands in the set.bat batch file to execute set before build operation:

```
set PATH=%PATH%;C:\Hew3\Tools\Renesas\H8\6_1_0\Bin
set CH38=C:\Hew3\Tools\Renesas\H8\6_1_0\Include
set CH38TMP=C:\Hew3\Tools\Renesas\H8\6_1_0\Ctemp
```

6.4.2 Library File

A library file is required for executing build operation. No library file is provided, so use a library configuration tool to create the file. For the command, refer to BuildAll.bat. Use the following command to create a library file. Executing BuildAll allows all programs including a library file to be created.

```
REM -- LIBRARY COMPILE --
"%CH38%\..\bin\lbg38" -output=H8runtime.lib -head=runtime -cpu=300HN
```

6.4.3 Output Files

Open the "Command Prompt" window by clicking "Accessories," then "Command Prompt." On the window, execute each batch file to create the relevant output file.

Table 6-8 Batch Files and Output Files

No.	Program	Batch File	Output File	Output File Type
1	Main processing module	bTest.bat	3694Test.mot	S-type file
2	Micro kernel	ub3694u.bat	uGenU.mot	S-type file
3	Main kernel	b3694u.bat	Genu3694.cde	Binary load module file
4	Programming kernel	b3694w.bat	Genw3694.cde	Binary load module file
5	Erasing kernel	b3694e.bat	Gene3694.cde	Binary load module file
6	Library	BuildAll.bat	H8runtime.lib	Library file

6.5 Modules

The modules are listed in Table 6-9.

Table 6-9 Modules

Program	File	Module	Function
Main processing module	Strt3694.src	startup	Start
	GenTest.c	main	Main processing
		WDTStop	Watchdog timer stop
		InitSCI	SCI initial setting
		Get	Reception
		Put	Transmission
		JumpCopy	Branch to copy
Micro kernel	Ugenu.c	StartFDTUserKernel	Start FDT
		PrepareFDTUserKernel	Prepare FDT
		PrepareRAM	Prepare RAM
		CmdFunc	Command function
	CmdFunc.c	ReferFunc	Reference function
		SelectDevice	Device selection
		SelectClockMode	Clock mode selection
		SetNewBaudRate	New baud rate setting
		SendSciBreak	Break transmission
		RequestBootPrgSts	Program status
		SendAck	ACK transmission
		GetCmdData	Command read
Main kernel	FDTUMain.c	Kernelmain	Main kernel
		ProcessCommand	Command processing
	CopyFunc.c	CopyFunction	Copy function
	CmdFunc.c	RequestBootPrgSts	Program status
		SumcheckUserArea	User area checksum
		SendAck	ACK transmission
		CheckBlank	Blank check
		ReadMemory	Memory read
		GetCmdData	Command read
Programming kernel	FDTWrite.c	WriteFLASH	Flash programming
		RequestBootPrgSts	Program status
		SendAck	ACK transmission
		GetWriteData	Programming data reception
	WriteTime.c	WriteWaitTime	Programming wait time
	F3694w.src	flash_write	Data programming
		CalCount	Time calculation
Erasing kernel	FDTErase.c	EraseFLASH	Flash erasing
		RequestBootPrgSts	Program status
		SendAck	ACK transmission
		GetEraseData	Erase data reception
	EraseTime.c	EraseWaitTime	Erasing wait time
	F3694e.src	block_erase	Block erasing
		CalCount	Time calculation

6.6 Module Hierarchical Structure

The module hierarchical structure is shown in Figure 6-1.

```
VECT (0x0000)   Reset vector
  │—startup (0x400) (Strt3694.src)   Start (Main processing module)
      │—main (GenTest.c)   Main processing
            │—WDTStop   Watchdog timer stop
            │—InitSCI   SCI initial setting
            │—JumpCopy   Branch to copy
                │—CopyFDT (0x7600)   Micro kernel copy
                    │—StartFDTUserKernel (uGenu.c)   Start micro kernel (Micro kernel)
                        │—PrepareFDTUserKernel   Prepare micro kernel
                        │     │—CmdFunc   Command function
                        │     │     │—Get   Reception
                        │     │     │—SendAck   ACK transmission
                        │     │     │     │—Put   Transmission
                        │     │     │—ReferFunc   Reference function
                        │     │     │     │—Put   Transmission
                        │     │     │—GetCmdData   Command read
                        │     │     │     │—Get   Reception
                        │     │     │     │—ErrorCode   Error code macro
                        │     │     │     │—Put   Transmission
                        │     │     │—SelectDevice   Device selection
                        │     │     │     │—SendAck   ACK transmission
                        │     │     │     │—ErrorCode   Error code macro
                        │     │     │     │—Put   Transmission
                        │     │     │—SelectClockMode   Clock mode selection
                        │     │     │     │—ErrorCode   Error code macro
                        │     │     │     │—Put   Transmission
                        │     │     │     │—SendAck   ACK transmission
                        │     │     │—SetNewBaudRate   New bit rate selection
                        │     │     │     │—ErrorCode   Error code macro
                        │     │     │     │—Put   Transmission
                        │     │     │     │—SendAck   ACK transmission
                        │     │     │     │—SendSciBreak   Break transmission
                        │     │     │     │     │—Get   Reception
                        │     │     │     │     │—SendAck   ACK transmission
                        │     │     │     │—Get   Reception
                        │     │     │—RequestBootPrgSts   Program status
                        │     │     │     │—Put   Transmission
                        │     │     │—Put   Transmission
                        │     │—PrepareRAM   Prepare RAM
                        │     │—Get   Reception
                        │     │—ErrorCode   Error code macro
                        │     │—Put   Transmission
                        │     │—SendAck   ACK transmission
                        │—RAMStartAddress (0xF780)   RAM start address
                    (To be continued)
```

Figure 6-1 Module Hierarchical Structure (1)

```
(Continued)
 │—RAMStartAddress (0xF780)    RAM start address
     │—Kernelmain (FDTUMain.c)    Main kernel
         │—ProcessCommand    Command processing
         │    │—Get    Reception
         │    │—RequestBootPrgSts    Program status
         │    │    │—Put    Transmission
         │    │—SumcheckUserArea    User area checksum
         │    │    │—Put    Transmission
         │    │—SendAck    ACK transmission
         │    │—GetCmdData    Command read
         │    │    │—Get    Reception
         │    │    │—ErrorCode    Error code macro
         │    │    │—Put    Transmission
         │    │—ReadMemory    Memory read
         │    │    │—ErrorCode    Error code macro
         │    │    │—Put    Transmission
         │    │—CheckBlank    Blank check
         │    │    │—ErrorCode    Error code macro
         │    │    │—Put    Transmission
         │    │    │—SendAck    ACK transmission
         │    │—Put    Transmission
         │—CopyFunction (CopyFunc.c)    Copy function
              │—Get    Reception
              │—ErrorCode    Error code macro
              │—Put    Transmission
              │—SendAck    ACK transmission
              │—FLASHFunc (0xFB10)    Flash function
             (To be continued)
```

Figure 6-2 Module Hierarchical Structure (2)

```
(To be continued)
  │—FLASHFunc (0xFB10)   Flash function
      │—EraseFLASH (FDTErase.c)   Flash erasing (Erasing kernel)
      │   │—EraseWaitTime   Erasing wait time
      │   │   │—CalCount (F3694e.src)   Time calculation
      │   │—Get   Reception
      │   │—RequestBootPrgSts   Program status
      │   │—GetEraseData   Erase data reception
      │   │   │—Get   Reception
      │   │   │—ErrorCode   Error code macro
      │   │   │—Put   Transmission
      │   │—block_erase (F3694e.src)   Block erasing
      │   │—Put   Transmission
      │   │—SendAck   ACK transmission
      │—WriteFLASH (FDTWrite.c)   Flash programming (Programming kernel)
          │—WriteWaitTime   Programming wait time
          │   │—CalCount (F3694w.src)   Time calculation
          │—Get   Reception
          │—RequestBootPrgSts   Program status
          │—GetWriteData   Programming data reception
          │   │—Get   Reception
          │   │—ErrorCode   Error code macro
          │   │—Put   Transmission
          │—flash_write (F3694w.src)   Data programming
          │—Put   Transmission
          │—SendAck   ACK transmission
```

Figure 6-3 Module Hierarchical Structure (3)

6.7 Program Processing Flow

The processing flow of the sample program is shown in Figure 6-4.

In the user program mode, bit rate adjustment and user area erase processing, which are performed during boot operation, are not performed. Accordingly, the program and data written in flash memory can be saved.

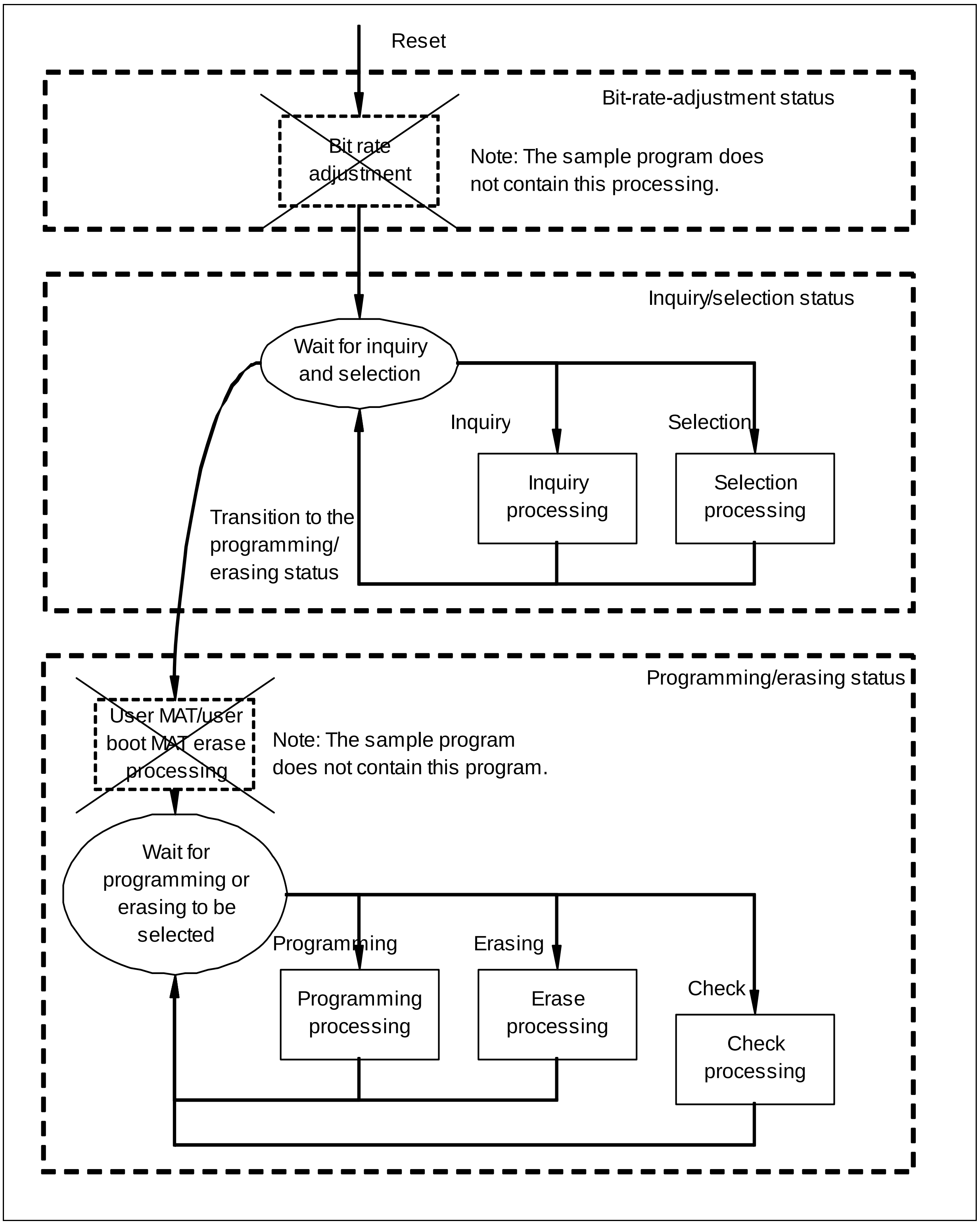
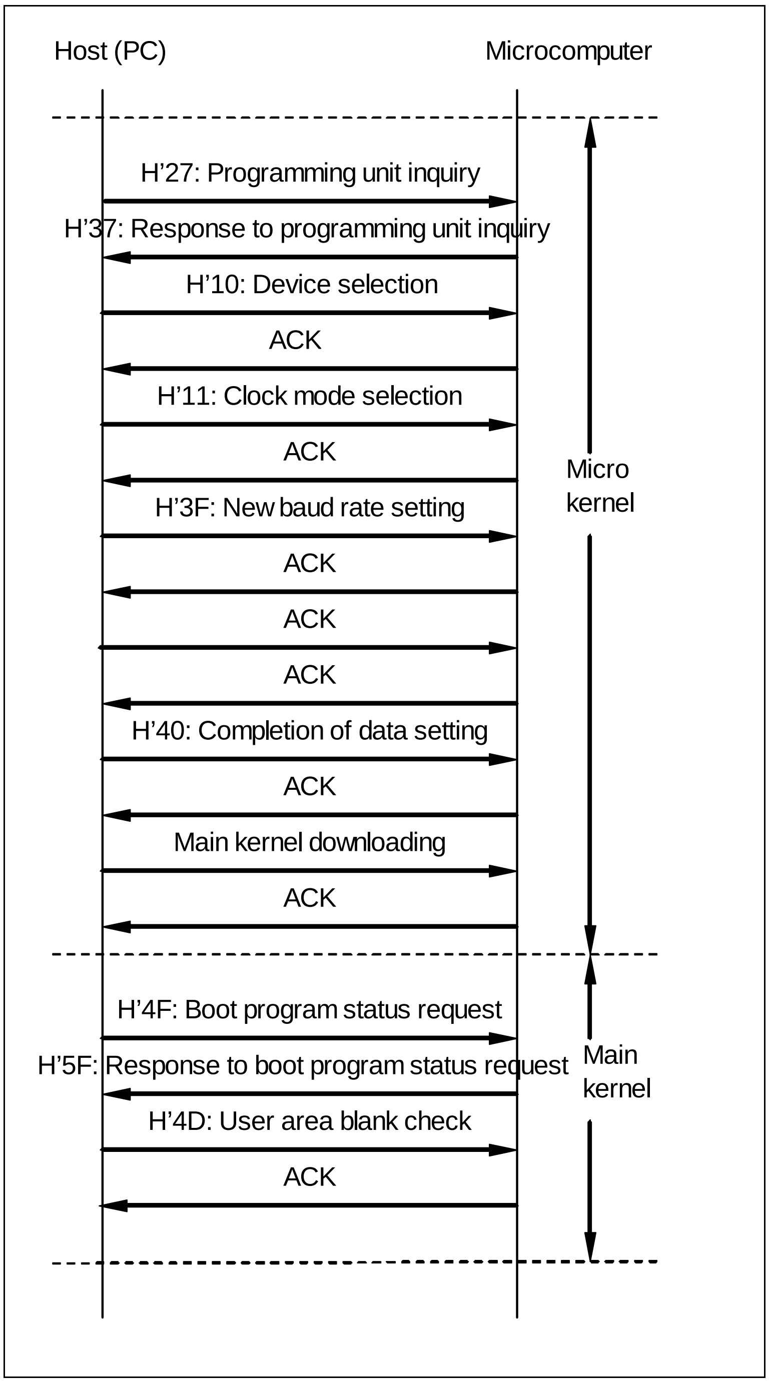

Reset
Bit-rate-adjustment status
Bit rate adjustment
Note: The sample program does not contain this processing.
Inquiry/selection status
Wait for inquiry and selection
Inquiry
Selection
Inquiry processing
Selection processing
Transition to the programming/ erasing status
Programming/erasing status
User MAT/user boot MAT erase processing
Note: The sample program does not contain this program.
Wait for programming or erasing to be selected
Programming
Erasing
Check
Programming processing
Erase processing
Check processing

Figure 6-4 Program Processing Flow

6.8 Command Sequence in the User Program Mode

The sequence of the commands between the Flash Development Toolkit and microcomputer when a device is connected, when flash memory is programmed, and when flash memory is erased in the user program mode is shown in Figure 6-5, Figure 6-6, and Figure 6-7.



Host (PC)
Microcomputer
H’27: Programming unit inquiry
H’37: Response to programming unit inquiry
H’10: Device selection
ACK
H’11: Clock mode selection
ACK
H’3F: New baud rate setting
ACK
ACK
ACK
H’40: Completion of data setting
ACK
Main kernel downloading
ACK
Micro kernel
H’4F: Boot program status request
H’5F: Response to boot program status request
H’4D: User area blank check
ACK
Main kernel

Figure 6-5 When a Device Is Connected

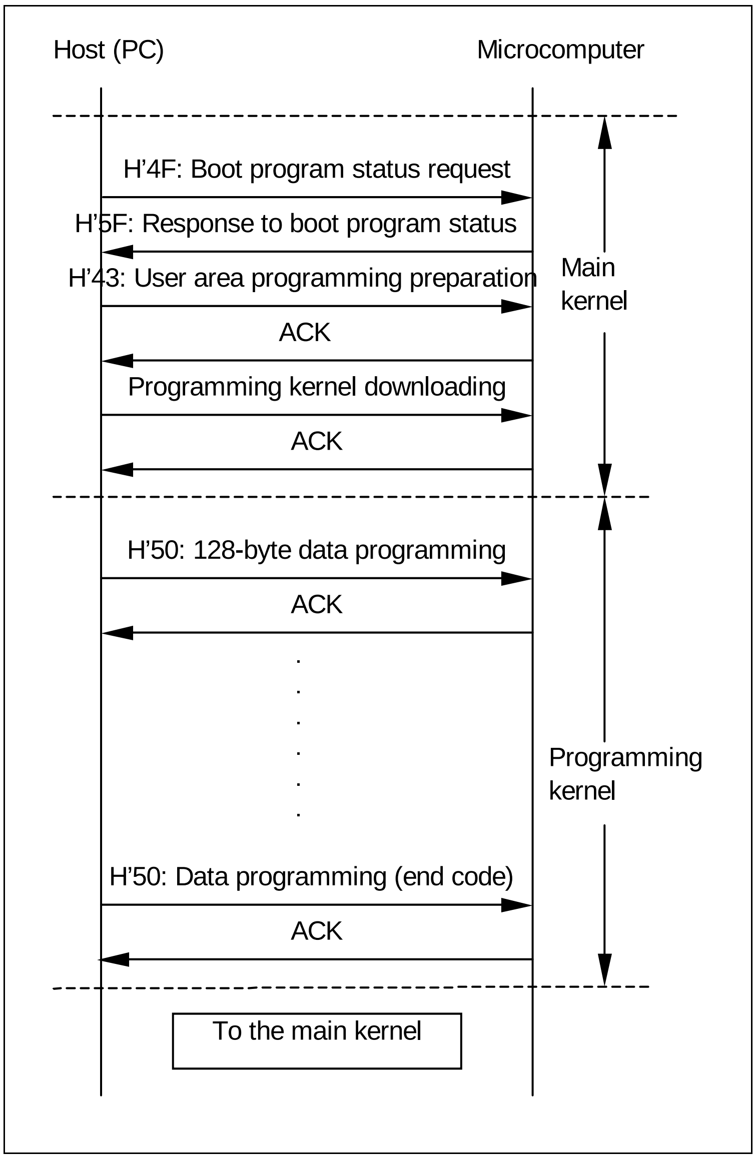
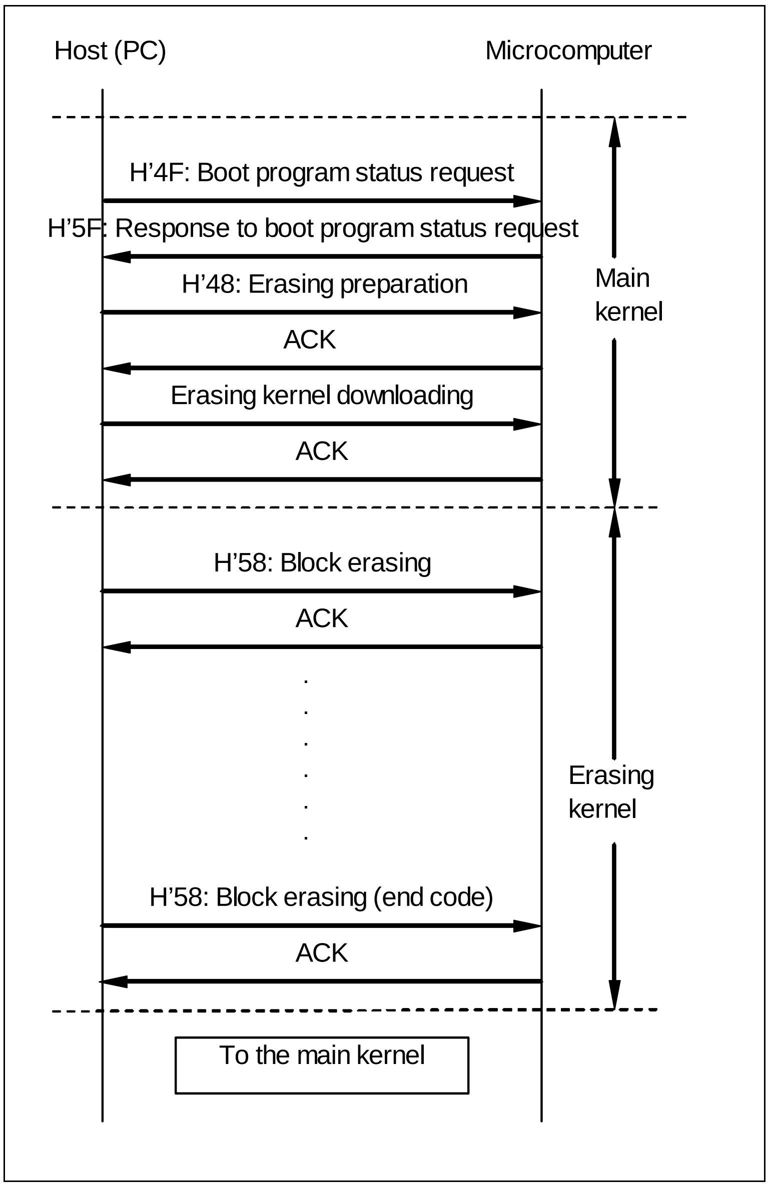

Host (PC)
Microcomputer
H’4F: Boot program status request
H’5F: Response to boot program status
H’43: User area programming preparation
ACK
Programming kernel downloading
ACK
Main kernel
H’50: 128-byte data programming
ACK
H’50: Data programming (end code)
ACK
Programming kernel
To the main kernel

Figure 6-6 When Flash Memory Is Programmed



Host (PC)
Microcomputer
H’4F: Boot program status request
H’5F: Response to boot program status request
H’48: Erasing preparation
ACK
Erasing kernel downloading
ACK
Main kernel
H’58: Block erasing
ACK
Erasing kernel
H’58: Block erasing (end code)
ACK
To the main kernel

Figure 6-7 When Flash Memory Is Erased

6.9 Program Sequence

This section describes the program sequence of the sample program. An outline of the program sequence is given in Table 6-10.

Table 6-10 Program Sequence

No.	Sequence	Processing
1	Preparation	• To use the Flash Development Toolkit in the user program mode, program the main processing module and micro kernel in flash memory in advance. • When the entire flash memory can be erased with no problems, they can be programmed in the boot mode of using the Flash Development Toolkit.
2	Main processing module	• The reset vector causes a branch to the main processing module by a power-on reset. • Performs initial setting for the stack pointer. • Performs initial setting for the SCI. • Pushes the addresses and sizes of the SCI interface functions (Put and Get) on the stack. (Passes arguments to the micro kernel.) • Causes a branch to the micro kernel.
3	Micro kernel	• Receives and responds to the device specification inquiry/selection commands. • After receiving the data setting completion command, receives the main kernel and stores it in RAM. • Stores the SCI interface functions (Put and Get) in RAM. • Causes a branch to the main kernel.
4	Main kernel	• Receives and responds to the programming, erasing, and check commands. • After receiving the programming/erasing preparation command, receives the programming or erasing kernel and stores it in RAM. · Calls the programming or erasing kernel.
5	Programming kernel	• Receives programming data and the programming destination address. • Programs flash memory. • Receives programming end data, then returns control to the main kernel.
6	Erasing kernel	• Receives the erase block number. • Erases the block in flash memory. • Receives erasing end data, then returns control to the main kernel.

6.9.1 Preparation

The flow of preparation is shown below:

(1) To use the Flash Development Toolkit in the user program mode, program the main processing module and micro kernel in flash memory in advance.
When the entire flash memory can be erased with no problems, they can be programmed using the boot mode of the Flash Development Toolkit. Alternatively, they can be programmed in the programmer mode using a PROM programmer.

(2) After programming the main processing module and micro kernel in the flash memory, set the pins on the microcomputer to the user program mode, perform a reset, and initiate the user program mode.

6.9.2 Main Processing Module

The flow of the main processing module is shown below. The main processing module runs in ROM.

(1) The reset vector causes a branch to start (startup).

(2) Start (startup) sets the stack pointer and calls main processing (main).

(3) Main processing (main) calls watchdog timer stop (WDTStop) and SCI initial setting (InitSCI) and causes a branch to branch to copy (JumpCopy).
Watchdog timer stop (WDTStop) stops the watchdog timer and SCI initial setting (InitSCI) sets the SCI bit rate.

(4) Micro kernel copy (CopyFDT) sets the addresses and sizes of the SCI interface functions (Get and Put) in the variable area and branches to start micro kernel (StartFDTUserKernel) via micro kernel copy (CopyFDT).

6.9.3 Micro Kernel

The flow of the micro kernel is shown below. The micro kernel runs in ROM.

(1) Command function (CmdFunc) processes each command, responds to each inquiry, and sets selection.

(2) Reference function (ReferFunc) and program status (RequestBootPrgSts) respond to each inquiry that corresponds to one of the following commands:
Supported device inquiry
Clock mode inquiry
Multiplication ratio inquiry
Operating frequency inquiry
User area information inquiry
Erase block information inquiry
Programming unit inquiry
Boot program status inquiry

(3) A selection setting command is set using one of the following modules:
Device selection (SelectDevice)
Clock mode selection (SelectClockMode)
New bit rate selection (SetNewBaudRate)

(4) The command for a transition to the programming/erasing status ends command processing and calls prepare RAM (PrepareRAM).

(5) Prepare RAM (PrepareRAM) receives the main kernel (Kernelmain) and stores it in RAM and transfers the SCI interface functions (Get and Put) to RAM.

(6) Control branches to the main kernel (Kernelmain) transferred to RAM.

6.9.4 Main Kernel

The flow of the main kernel is shown below. The main kernel runs in RAM.

(1) Command processing (ProcessCommand) processes commands. The following commands are to be processed:
Memory read (ReadMemory)
User area sum check (SumcheckUserArea)
User area blank check (CheckBlank)
Boot program status inquiry (RequestBootPrgSts)

(2) When command processing receives the user area programming selection command or erasing selection command, it calls copy function (CopyFunction).

(3) Copy function (CopyFunction) receives the programming kernel (WriteFLASH) or erasing kernel (EraseFLASH) corresponding to the command, stores it in RAM, and calls the programming kernel (WriteFLASH) or erasing kernel (EraseFLASH).

6.9.5 Programming Kernel

The flow of the programming kernel is shown below. The programming kernel runs in RAM.

(1) Flash programming (WriteFLASH) calculates the wait time using programming wait time (WriteWaitTime). Then, it receives a command.

(2) When the received command is the boot program status inquiry command, flash programming calls program status (RequestBootPrgSts).

(3) When the received command is the 128-byte programming command, programming data reception (GetWriteData) receives programming data.

(4) When the received programming data is not programming end (the address data is H'FFFFFFFF), data programming (flash_write) programs flash memory.

(5) When the received programming data is programming end, flash programming terminates programming and returns control to the main kernel.

6.9.6 Erasing Kernel

The flow of the erasing kernel is shown below. The erasing kernel runs in RAM.

(1) Flash erasing (EraseFLASH) calculates the wait time using erasing wait time (EraseWaitTime).
Then, it receives a command.

(2) When the received command is the boot program status inquiry command, flash erasing calls program status (RequestBootPrgSts).

(3) When the received command is the block erasing command, erase data reception (GetEraseData) receives the erase block number.

(4) When the received erase block number is not erasing end (H'FF), block erasing (block_erase) erases a block in flash memory.

(5) When the received erase block number is erasing end, flash erasing terminates erasing and returns control to the main kernel.

6.10 Memory Map

The memory map corresponding to the program sequence of the sample program is shown in Table 6-11.

Table 6-11 Program Sequence and Memory Map

ROM/ RAM	Address	Sequence				
		Main Processing Module	Micro Kernel	Main Kernel	Programming Kernel	Erasing Kernel
ROM	H’0000 -	Reset vector			Programming enable area	Erasing enable area
	H’0400 -	Main processing module				
	H’7600 -	Micro kernel	Micro kernel			
RAM	H’F780 -		Main kernel Put, Get	Main kernel Put, Get	Main kernel Put, Get	Main kernel Put, Get
	H’FB08 -		Variable			
	H’FB10 -			Programming kernel or erasing kernel	Programming kernel	Erasing kernel
	H’FF10 -	Global variable	Global variable	Global variable	Global variable	Global variable
	- H’FF7F	Stack	Stack	Stack	Stack	Stack

7. Source Files of the Sample Program

This section describes the functions and processing of the following main source files of the sample program.

7.1 Header Files

This sample program uses the following header files.

7.1.1 Bit Rate Setting (GenTest.h)

A bit rate is set.

```
                                                        /* 20MHz 9600bps */
//#define MA_BRR_SCI                        0x40            /* Bit rate register channel 3 */
                                                        /* 9.8MHz 9600bps */
#define MA_BRR_SCI                          0x1f            /* Bit rate register channel 3 */
```

In the user program mode, a device is connected at 9600 bps. For this reason, the bit rate register (BRR) in the SCI module must be set according to the operating frequency. In this example, the operating frequency is 9.8 MHz. To set 9600 bps, MA_BRR_SCI is set to 31 (0x1f). The relationships between operating frequencies and BRR register settings are shown in Table 7-1.

Table 7-1 Operating Frequencies and BRR Register Settings (When the Bit Rate Is 9600 (bit/s))

Operating Frequency ϕ (MHz)	BRR Setting	Error (%)
8	25	0.16
9.8304	31	0.00
10	32	−1.36
12	38	0.16
12.288	39	0.00
14	45	−0.93
14.7456	47	0.00
16	51	0.16
17.2032	55	0.00
18	58	−0.69
19.6608	63	0.00
20	64	0.16

The MA_BRR_SCI value is set according to the operating frequency of the board and perform a build with a batch file or HEW to create an S-type file program.

7.1.2 I/O Register Definition (io3694.h)

The registers and bits related to the SCI module and WDT are defined.

```
/**************************************************************************/
/*    H8/3694F,36014F,36024F,36064F Internal I/O Include File             */
/**************************************************************************/

/**************************************************************************/
/*                SCI                                                     */
/*--------------------------------------------------------------------*/
/*                          CHANNEL 3                                     */
/**************************************************************************/
#define SCI_SMR          (*(volatile unsigned char *)0xFFA8)
#define SCI_BRR          (*(volatile unsigned char *)0xFFA9)
#define SCI_SCR3         (*(volatile unsigned char *)0xFFAA)
   #define TE                       (unsigned char)0x20
   #define RE                       (unsigned char)0x10
   #define TE_RE                    (unsigned char)(TE | RE)
#define SCI_TDR          (*(volatile unsigned char *)0xFFAB)
#define SCI_SSR          (*(volatile unsigned char *)0xFFAC)
   #define TDRE                     (unsigned char)0x80
   #define RDRF                     (unsigned char)0x40
   #define ERR_CLR                  (unsigned char)0xC7
   #define TEND                     (unsigned char)0x04
#define SCI_RDR          (*(volatile unsigned char *)0xFFAD)

/**************************************************************************/
/*                I/O Port                                                */
/*--------------------------------------------------------------------*/
/*                          Port 1          (in use : TXD)                */
/**************************************************************************/
#define PMR1                (*(volatile unsigned char *)0xFFE0)
   #define TXD                      (unsigned char)0x02

/**************************************************************************/
/*                I/O Port                                                */
/*--------------------------------------------------------------------*/
/*                          Port 2          (in use : P22/TXD at SCI Break)  */
/**************************************************************************/
#define PCR2                (*(volatile unsigned char *)0xFFE5)
   #define PCR22                    (unsigned char)0x04
```

```
#define PDR2                (*(volatile unsigned char *)0xFF5D)
   #define P22                      (unsigned char)0x04

/************************************************************************/
/*                  WDT                                                 */
/*--------------------------------------------------------------------*/
/*                                                                      */
/************************************************************************/
#define TCSRWD              (*(volatile unsigned char *)0xFFC0)
```

7.1.3 Macro Definition (KAlg.h)

Labels used in the program are defined. ERASE_END is used to determine the end of erasing by the block erasing command. WRITE_END is used to determine the end of programming by the 128-byte programming command.

```
/* D E F I N E S */
#define LOOP_END                                  1
#define bufSize                                   0x80
#define BLOCK_NO_ERROR                            0xE1
#define ERASE_END                                 0xFF
#define WRITE_END                                 0xFFFFFFFF
#define ADDRESS_ERROR                             0xF1
#define ADDRESS_BOUNDARY_ERROR                    0xF2
```

7.2 Main Processing Module (Strt3694.src and GenTest.c)

7.2.1 Module Hierarchical Structure

The module hierarchical structure of the main processing module is shown in Figure 7-1.

```
VECT (0x0000)   Reset vector
  │—startup (0x400) (Strt3694.src)   Start (Main processing module)
      │—main (GenTest.c)   Main processing
          │—WDTStop   Watchdog timer stop
          │—InitSCI   SCI initial setting
          │—JumpCopy   Branch to copy
              │—CopyFDT (0x7600)   Micro kernel copy
                  │—StartFDTUserKernel (uGenu.c)   Start micro kernel (Micro kernel)
```

Figure 7-1 Module Hierarchical Structure of the Main Processing Module

7.2.2 Reset Vector (GenTest.c and GenTest.h)

Reset vector H'400 is set in the CVECT section.

(1) GenTest.c

```
/*Declare the vector table*/
#pragma section VECT
const WORD RESET_VECTOR = (DWORD)RESET_JMP_ADDRESS;
#pragma section
```

(2) GenTest.h

```
/*
   This value specifies the address to where to program
   will JMP on startup. This value should be the link address
   for the associated asm file.
*/
#define RESET_JMP_ADDRESS          0x400
```

7.2.3 Stack (Strt3694.src)

The stack pointer is set to H'FF80.

```
          MOV.L     #H'FF80, ER7
```

7.2.4 Main Processing (main)

```
        WDTStop();
        InitSCI();
        JumpCopy();
```

The watchdog timer is stopped, SCI initial setting is performed, and control branches to the micro kernel.

7.2.5 Branch to Copy (JumpCopy)

（１） JumpCopy

```
        /* Create Function Pointer & assign address to it */
        FuncPtr CopyFDT = (FuncPtr)USER_KERNEL_LINK_ADDRESS;
                                                        /*This is where the linker has put the code*/
        /* Store structure elements */
        ParamFDT.GetFuncPtr = (GetPtr)Get;
        ParamFDT.PutFuncPtr = (PutPtr)Put;
        ParamFDT.PutSize = (WORD)((DWORD)Dummy - (DWORD)Put);
        ParamFDT.GetSize = (WORD)((DWORD)Put - (DWORD)Get);

        ParamFDT.RAMStartAddress = RAM_START_ADDRESS;

        /* Jump to CopyFDT */
        (*CopyFDT)((paramFDT *)&ParamFDT);
```

（２） GenTest.h

```
/*
  These defines relate to the USER kernel.
  In order to call the user kernel we must know the address it was
  linked at.
*/
#define USER_KERNEL_LINK_ADDRESS        0x7600
```

The addresses and sizes of the SCI interface functions (Get and Put) are set in ParamFDT and control branches to CopyFDT. The address of CopyFDT is H'7600, which is indicated by USER_KERNEL_LINK_ADDRESS. Start micro kernel (StartFDTUserKernel) of the micro kernel is programmed at H'7600.

7.3 Micro Kernel (uGenu.c and CmdFunc.c)

7.3.1 Module Hierarchical Structure

The module hierarchical structure of the micro kernel is shown in Figure 7-2.

```
|—StartFDTUserKernel (uGenu.c)    Start micro kernel (Micro kernel)
    |—PrepareFDTUserKernel    Prepare micro kernel
    |   |—CmdFunc    Command function
    |   |   |—ReferFunc    Reference function
    |   |   |—GetCmdData    Command read
    |   |   |—SelectDevice    Device selection
    |   |   |—SelectClockMode    Clock mode selection
    |   |   |—SetNewBaudRate    New bit rate selection
    |   |   |—RequestBootPrgSts    Program status
    |   |—PrepareRAM    Prepare RAM
    |—RAMStartAddress (0xF780)    RAM start address
        |—Kernelmain (FDTUMain.c)    Main kernel
```

Figure 7-2　Module Hierarchical Structure of the Micro Kernel

7.3.2 Start Micro Kernel (StartFDTUserKernel)

（１）StartFDTUserKernel

```
    PrepareFDTUserKernel(parameters);

    /* Pass execution to the main kernel */
    (*((FuncPtr)parameters->RAMStartAddress))(parameters);
```

（２）GenTest.h

```
/*Use these defines to specify the range of RAM FDT can use*/
#define RAM_START_ADDRESS          0xF780
```

Prepare micro kernel is called and the module stored at RAMStartAddress in RAM is called. Prepare micro kernel stores the main kernel in the area starting at RAMStartAddress in RAM. RAMStartAddress is set to H'F780.

7.3.3 Prepare Micro Kernel (PrepareFDTUserKernel)

```
while(1){
        /* Command Function */
        CmdFunc(parameters->PutFuncPtr, parameters->GetFuncPtr);
        /*Prepare RAM */
        if(!PrepareRAM(parameters, &ParamFDT)){
                break;
        }
}
```

Command function is called. When command function terminates (on receiving the command for a transition to the programming/erasing status), prepare RAM is called. When prepare RAM terminates (the main kernel has been received and stored normally), prepare micro kernel terminates.

7.3.4 Command Function (CmdFunc and CmdFunc.c)

The structure of command function is shown below.

```
while(1){
        /* Acquisition of a command ID */
        add_sum = Get(&commandID, 1);
        switch(commandID)
        {
                case finishDataSet:
                        return;
                case supportDevice:
                        ReferFunc(commandID, deviceData, sizeof(deviceData), Put);
                        break;
                case selectDevice:
                                SelectDevice(cmdBuf.bdata, Put);
                        break;
                case referClockMode:
                        ReferFunc(commandID, clockModeData, sizeof(clockModeData), Put);
                        break;
                case selectClockMode:
                                SelectClockMode(cmdBuf.bdata, Put);
                        break;
                case referRatio:
                        ReferFunc(commandID, ratioData, sizeof(ratioData), Put);
                        break;
                case setNewBaudRate:
                        SetNewBaudRate(commandID, (BaudRate *)cmdBuf.bdata, Put, Get);
                        break;
                case referUserRomInfo:
                        ReferFunc(commandID, usrRomData, sizeof(usrRomData), Put);
                        break;
                case referEraseBlockInfo:
                        ReferFunc(commandID, eraseBlkData, sizeof(eraseBlkData), Put);
                        break;
                case referWriteSystem:
                        ReferFunc(commandID, writeSysData, sizeof(writeSysData), Put);
                        break;
                case referFrequency:
                        ReferFunc(commandID, frequencyData, sizeof(frequencyData), Put);
                        break;
                case referWriteSize:
                        ReferFunc(commandID, writeSizeData, sizeof(writeSizeData), Put);
                        break;
                case requestBootPrgSts:
                        RequestBootPrgSts(Put);
                        break;
                default:
                        cBuff[0] = COMMAND_ERROR;
                        cBuff[1] = commandID;
                        Put(cBuff, 2);
                        break;
        }
}
```

When command function receives the command for a transition to the programming/erasing status, it terminates processing. When command function receives another command, it processes the command, then enters the command reception wait state.

The processing module for each command is contained in CmdFunc.c. CmdFunc.c contains both the command processing modules for the micro kernel, as well as those for the main kernel. #ifdef is used to determine whether a command processing module is for the micro kernel or main kernel.

These command processing modules are used not only in the user program mode, but also in the boot mode. In the user program mode, the SCI interface functions (Get and Put) have arguments, but in the boot mode, they have no arguments.

7.3.5 Prepare RAM (PrepareRAM)

```
kernelPos = (BYTE *)parameters->RAMStartAddress;
/* Receive size of User Kernel module from host */
add_sum = Get((BYTE *)&kernelSize, sizeof(kernelSize));
/* Download kernel to beginning of allowable RAM */
add_sum+= Get(kernelPos, kernelSize);
/* Adjust start position of RAM */
parameters->RAMStartAddress += kernelSize;

/*
        Copy the Get and Put functions into memory, first Get() then Put()
*/
pSrc = (BYTE *)parameters->GetFuncPtr;
pDest = (BYTE *)parameters->RAMStartAddress;
/* Now perform the copy */
for(i = 0; i < parameters->GetSize; i++, pSrc++, pDest++)
{
        *pDest = *pSrc;
}

parameters->RAMStartAddress += parameters->GetSize;
pSrc = (BYTE *)parameters->PutFuncPtr;
pDest = (BYTE *)parameters->RAMStartAddress;
/* Now perform the copy */
for(i = 0; i < parameters->PutSize; i++, pSrc++, pDest++)
{
        *pDest = *pSrc;
}
```

Prepare RAM performs the following processing:

(1) Receives the main kernel and stores it in RAM.

(2) Copies the Get function into RAM.

(3) Copies the Put function into RAM.

Prepare RAM sets the starting address of each function and stores it in RAM. Prepare RAM also stores the size of each function in RAM. The functions are executed in RAM to erase or program flash memory.

7.4 Main Kernel (FDTUMain.c, CmdFunc.c, and CopyFunc.c)

7.4.1 Module Hierarchical Structure

The module hierarchical structure of the main kernel is shown in Figure 7-3.

```
│—Kernelmain (FDTUMain.c)   Main kernel
    │—ProcessCommand   Command processing
    │   │—RequestBootPrgSts   Program status
    │   │—SumcheckUserArea   User area checksum
    │   │—SendAck   ACK transmission
    │   │—GetCmdData   Command read
    │   │—ReadMemory   Memory read
    │   │—CheckBlank   Blank check
    │—CopyFunction (CopyFunc.c)   Copy function
        │—FLASHFunc (0xFB10)   Flash function
```

Figure 7-3 Module Hierarchical Structure of the Main Kernel

7.4.2 Main Kernel (Kernelmain)

```
/* Main control processing loop */
while (1)
{
        if(ProcessCommand(&commandID, parameters))
        {
                CopyFunction(commandID, parameters);
        }
}
```

The main kernel executes command processing (ProcessCommand) repeatedly. Command processing receives and processes commands. Copy function (CopyFunction) is called only when the user area programming selection or erasing selection command is received. It receives the erasing or programming kernel corresponding to the command and stores it in RAM. The erasing kernel erases data and the programming kernel programs data. When programming or erasing terminates, command processing is called again.

7.4.3 Command Processing (ProcessCommand)

The structure of command processing is shown below:

```
        /* Acquisition of a command ID */
        add_sum = Get(commandID, 1);
        switch(*commandID)
        {
                case requestBootPrgSts:
                        RequestBootPrgSts(Put);
                        break;
                case sumcheckUserArea:
                        SumcheckUserArea(Put);
                        break;
                case prepareErase:
                case prepareUserAreaWrite:
                        return(TRUE);
                case readMemory:
                                ReadMemory(cmdBuf.bdata, Put);
                        break;
                case checkBlank:
                        CheckBlank(Put);
                        break;
                default:
                        cBuff[0] = COMMAND_ERROR;
                        cBuff[1] = *commandID;
                        Put(cBuff, 2);
                        break;
        }
        return(FALSE);
```

When command processing receives the erasing or programming selection command, it returns TRUE. Other commands are processed by the relevant command processing modules and command processing returns FALSE.

The command processing modules are contained in CmdFunc.c.

7.4.4 Copy Function (CopyFunction)

(1) CopyFunction

```
BYTE *funcAddress = (BYTE *)FUNC_START, add_sum;
FLASHFuncPtr FLASHFunc = (FLASHFuncPtr)FUNC_START;

/* Acquire size of function to be downloaded */
add_sum = Get((BYTE *)&size, sizeof(size));
/* Download function to RAM address received */
add_sum+= Get(funcAddress, size);

/* Pass execution to the FLASH function */
(*FLASHFunc)(Put, Get);
```

(2) KDevice.h

```
#define FUNC_START          0xFB10          /* Write/Erase function start position */
#define WRITE_DATA          0xFE90          /* write-data area start position */
```

Copy function receives the erasing or programming kernel and stores it in a RAM area starting at H'FB10 indicated by FUNC_START. After storing the kernel, copy function calls FUNC_START to execute erasing or programming. Erasing or programming is determined according to the command received by command processing (ProcessCommand).

7.5 Erasing Kernel (FDTErase.c, EraseTime.c, and F3694e.src)

7.5.1 Module Hierarchical Structure

The module hierarchical structure of the erasing kernel is shown in Figure 7-4.

```
|—FLASHFunc (0xFB10)   Flash function
    |—EraseFLASH (FDTErase.c)   Flash erasing (Erasing kernel)
    |   |—EraseWaitTime (EraseTime.c)   Erasing wait time
    |   |   |—CalCount (F3694e.src)   Time calculation
    |   |—RequestBootPrgSts   Program status
    |   |—GetEraseData   Erase data reception
    |   |—block_erase (F3694e.src)   Block erasing
```

Figure 7-4 Module Hierarchical Structure of the Erasing Kernel

7.5.2 Flash Erasing (EraseFLASH)

The structure of flash erasing is shown below:

```
/* Waiting time calculation of erase processing */
EraseWaitTime();

do{
        /* Acquisition of a command ID */
        add_sum = Get(&commandID, 1);
        /* Is it a demand of boot status command? */
        if (commandID == requestBootPrgSts){
                RequestBootPrgSts(Put);
        }else{
                /* Acquisition of command data    */
                if(GetEraseData(&blk_no, add_sum, Put, Get)){
                        return;
                }
                if (blk_no != ERASE_END){
                        /* Erase start */
                        rsts = block_erase(blk_no);
                        if(rsts){
                                if (rsts == BLOCK_NO_ERROR){
                                        cBuff[1] = ERASE_BLOCK_NO_ERROR;
                                }else{
                                        cBuff[1] = ERASE_ERROR;
                                }
                                return;
                        }
                }else{
                        end_flg = LOOP_END;
                }
        }
}while(!end_flg);
```

Flash erasing (EraseFLASH) calculates the erasing wait time using erasing wait time (EraseWaitTime).
Then, it receives a command. When the command is program status inquiry, flash erasing responds with the status using program status (RequestBootPrgSts).
When the command is block erasing and the data is not erasing end, flash erasing specifies the block number and calls block erasing (block_erase).
When the data is erasing end, flash erasing terminates erasing and returns control to the main kernel.

7.5.3 Erasing Wait Time (EraseWaitTime and CalCount)

(1) EraseWaitTime

```
        SWES_W = CalCount(1)+1;
        SWEC_W = ESUS_W = CalCount(100)+1;
        ESUC_W = EC_W = CalCount(10)+1;
        ES_W = CalCount(10000);
        EVS_W = CalCount(20)+1;
        EVC_W = CalCount(4)+1;
        DLCH_W = CalCount(2)+1;
```

(2) CalCount

```
FREQ:   .EQU            H'FF10          ; Frequency(Global data) import from "KDevice.h"
LCNT:   .EQU            D'600           ; 1µs loop counter
; _CalCount     .EQU    $
                SUB.W       E0,E0
                MOV.W       @FREQ,R1            ;frequency
                MULXU.W     R1,ER0
                MOV.W       #LCNT,R1
                DIVXU.W     R1,ER0
                RTS
```

Erasing wait time (EraseWaitTime) calculates the wait time (µs) after each bit of the relevant register is set to 1 or cleared to 0 when erasing is executed. Time calculation (CalCount) calculates the wait time with the number of instructions based on the given frequency, assuming that one instruction requires 6 cycles. The frequency is the operating frequency calculated based on the value given by new bit rate selection. The frequency in 10 kHz is stored at H'FF10.
To erase flash memory using a dedicated interface without using the Flash Development Toolkit, create a program, referring to the user manual for how to set the operating frequency and how to calculate the erasing wait time.

Examples of calculated erasing wait time values are listed in Table 7-2.

Table 7-2 Examples of Erasing Wait Time Values (Operating Frequency: 20 MHz)

Erasing Wait Time	Variable	Time (μs)	Software Loop Count
After the SWE bit is set	SWES_W	1	4
After the SWE bit is cleared	SWEC_W	100	334
After the ESU bit is set	ESUS_W	100	334
After the ESU bit is cleared	ESUC_W	10	34
After the E bit is set	ES_W	10000 (10 ms)	33333
After the E bit is cleared	EC_W	10	34
After the EV bit is set	EVS_W	20	67
After the EV bit is cleared	EVC_W	4	14
After dummy data is programmed	DLCH_W	2	7

7.6 Programming Kernel (FDTWrite.c, WriteTime.c, and F3694w.src)

7.6.1 Module Hierarchical Structure

The module hierarchical structure of the programming kernel is shown in Figure 7-5.

```
│—FLASHFunc (0xFB10)    Flash function
    │—WriteFLASH (FDTWrite.c)    Flash memory programming (Programming kernel)
           │—WriteWaitTime (WriteTime.c)    Programming wait time
           │    │—CalCount (F3694w.src)    Time calculation
           │—RequestBootPrgSts    Program status
           │—GetWriteData    Programming data reception
           │—flash_write (F3694w.src)    Data programming
```

Figure 7-5　Module Hierarchical Structure of the Programming Kernel

7.6.2 Flash Memory Programming (WriteFLASH)

The structure of flash memory programming is shown below:

```
/* Waiting time calculation of write processing */
WriteWaitTime();

do{
        /* Acquisition of a command ID */
        add_sum = Get(&commandID, 1);
        /* Is it a demand of boot status command? */
        if (commandID == requestBootPrgSts){
                RequestBootPrgSts(Put);
        }else{
                /* Acquisition of command data    */
                if(GetWriteData(&pAddress, add_sum, Put, Get)){
                        return;
                }
                if (pAddress != WRITE_END){
                        /* Write-in start */
                        rsts = flash_write((BYTE *)WRITE_DATA, (BYTE *)pAddress);
                        if(rsts)
                        {
                                if (rsts == ADDRESS_ERROR ||
                                        rsts == ADDRESS_BOUNDARY_ERROR)
                                {
                                        cBuff[1] = WRITE_ADDRESS_ERROR;
                                }else{
                                        cBuff[1] = WRITE_ERROR;
                                    }
                                    return;
                              }
                        }else{
                              end_flg = LOOP_END;
                        }
                  }
            }while(!end_flg);
```

Flash memory programming (WriteFLASH) calculates the programming wait time using programming wait time (WriteWaitTime).

Then, it receives a command. When the command is program status inquiry, flash memory programming responds with the status using program status (RequestBootPrgSts).

When the command is 128-byte programming and the data is not programming end, flash memory erasing specifies the programming address and programming data and calls data programming (flash_write).

When the data is programming end, flash memory programming terminates programming and returns control to the main kernel.

7.6.3 Programming Wait Time (WriteWaitTime and CalCount)

(1) WriteWaitTime

```
      P10S_W = CalCount(10);
      P30S_W = CalCount(30);
      PSUS_W = CalCount(50);
      SWEC_W = PSUS_W * 2;
      P200S_W = SWEC_W * 2;
      PSUC_W = PC_W = PVS_W = DLCH_W = CalCount(5);
```

(2) CalCount

```
FREQ:    .EQU          H'FF10          ; Frequency(Global data) import from "KDevice.h"
LCNT:    .EQU          D'600           ; 1µs loop counter
_CalCount      .EQU    $
               SUB.W       E0,E0
               MOV.W       @FREQ:16,R1          ;frequency
               MULXU.W     R1,ER0
               MOV.W       #LCNT,R1
               DIVXU.W     R1,ER0
               RTS
```

Programming wait time (WriteWaitTime) calculates the wait time (µs) after each bit of the relevant register is set to 1 or cleared to 0 when programming is executed. Time calculation (CalCount) calculates the wait time with the number of instructions based on the given frequency, assuming that one instruction requires 6 cycles. The frequency is the operating frequency calculated based on the value given by new bit rate selection. The frequency in 10 kHz is stored at H'FF10.

To program flash memory using a dedicated interface without using the Flash Development Toolkit, create a program, referring to the user manual for how to set the operating frequency and how to calculate the programming wait time.

Examples of calculated programming wait time values are listed in Table 7-3.

Table 7-3 Examples of Programming Wait Time Values (Operating Frequency: 20 MHz)

Programming Wait Time	Variable	Time (μs)	Software Loop Count
After the PSU bit is set	PSUS_W	50	166
After the PSU bit is cleared	PSUC_W	5	16
Programming time for additional programming	P10S_W	10	33
Programming time for programming after the P bit is set (Programming count: 1 to 6)	P30S_W	30	100
Programming time for programming (Programming count: 7 to 1000)	P200S_W	200	666
After the SWE bit is cleared	SWEC_W	100	34
After the P bit is cleared	PC_W	5	16
After the PV bit is set	PVS_W	5	16
After dummy data is programmed	DLCH_W	5	16

8. Using Programming/Erasing Kernels (Supplied Programs)

You can use the flash memory programming/erasing logical modules by connecting them to your own developed program without the Flash Development Toolkit interface section (main processing module and micro kernel). This section describes the logical modules which are part of the programming/erasing kernels.

8.1 Programming

8.1.1 Used Files

File Name	Description	Language
F3694w.src	Source file of 128-byte programming	Assembly language
F3694asm.inc	Header file common to 128-byte programming and block erasing	Assembly language

8.1.2 Module Specifications

Name	128-byte programming
Type	unsigned char flash_write(unsigned char *data, unsigned char *adr)
Function	Programs 128-byte data.
Arguments	data: Programming data start address adr: Programming destination address
Return Value	Processing result Normal termination: H'00 Maximum programming count error: H'01 Programmed data error: H'02 FWE error: H’D1 FLER error: H’D2 Programming address error: H'F1 128-byte boundary address error: H'F2
Input	Programming wait time
Processing	Executes programming in 128-byte units. For details, refer to program/program-verify flowchart in the hardware manual.
Note	To use the module, set the programming wait time (software loop count) in the global variable area in advance.

8.2 Erasing

8.2.1 Used Files

File Name	Description	Language
F3694e.src	Source file of block erasing	Assembly language
F3694asm.inc	Header file common to 128-byte programming and block erasing	Assembly language

8.2.2 Module Specifications

Name	Block erasing
Type	unsigned char block_erase (unsigned char blk_no)
Function	Erases a block.
Argument	blk_no: Block number
Return Value	Processing result Normal termination: H'00 Erasing error: H’01 FWE error: H’D1 FLER error: H’D2 Block number error: H'E1 Maximum erasing count error: H’E2
Input	Erasing wait time
Processing	Executes erasing of each block. For details, refer to erase/erase-verify flowchart in the hardware manual.
Note	To use the module, set the erasing wait time (software loop count) in the global variable area in advance.

Flash Development Toolkit Application Note (Applications)
User Program Mode (H8/3694F)

Publication Date:	Jun. 28, 2006	Rev.1.00
Published by:	Sales Strategic Planning Div. Renesas Technology Corp.	
Edited by:	Microcomputer Tool Development Department Renesas Solutions Corp.	

Flash Development Toolkit
Application Note (Applications)

Renesas Electronics Corporation
1753, Shimonumabe, Nakahara-ku, Kawasaki-shi, Kanagawa 211-8668 Japan

REJ06J0004-0100