Implementing Secure Remote Updates using Triscend E5 Configurable System-on-Chip Devices

January, 2000, v1.00 Application Note (AN-02)

Abstract:

Remotely updating embedded equipment in the field via a secure and reliable communications path reduces maintenance costs and product introduction risks. This capability also provides an additional revenue stream through cost-effective product upgrades. The Triscend E5 Configurable System on Chip (CSoC) device is particularly attractive in downloadable applications because it contains both programmable hardware and an industry standard microcontroller. Modern cryptography provides powerful tools to prevent accidental or malicious installation of incorrect updates via download. This application note provides example code that implements secure updates for the Triscend E5.

Introduction

This application note describes a simple but highly secure programmable download technique for Triscend's 8-bit E5 CSoC family. The working software supplied implements the technique on the Triscend E5 Development Board.

The ability to download new software and programmable hardware releases to equipment already deployed in the field provides many advantages for embedded systems manufacturers:

1. Perform product upgrades or correct serious product design errors easily in the field, without recalling the product.
2. Introduce the product to the market quickly with a stable, reduced feature set. Update the product as new features stabilize or as competitive products appear. This is of particular value where the product needs to implement a standardized function (*e.g.* a communications protocol) but some details of the standard may still be subject to change.
3. Customize products for various markets or customers after they are shipped.
4. Obtain an additional revenue stream from software/hardware upgrades. Introduce new features and deliver them in a cost-effective, timely fashion.

Sending a field technician to replace boot PROMs is expensive. Updating a system remotely drastically reduces upgrade costs. Undeniably, the ability to economically update equipment in the field is attractive. However, it is essential that the system be updated securely and reliably. An incorrect update may damage the embedded system or the equipment it controls. With a conventional non-downloadable embedded system, a malicious individual requires physical access to the embedded system in order to change its software. Consequently, it is very difficult to attack more than a small number of systems. A downloadable system potentially allows an attacker to access every product shipped by the manufacturer. Fortunately, modern cryptographic technology can 'harden' the download process to the point where it no longer provides an easy target to hackers.

Secure Download Architecture

Implementing secure download on an 8-bit microcontroller with limited external FLASH memory is a challenging task. Every byte of memory or processor cycle used by the download code is one less byte or cycle available to the 'real' function of the embedded system. A 'lightweight' download technique is essential. Unfortunately, this rules out a solution based on established Internet standards such as Secure Sockets Layer (SSL).

Figure 1 shows a simple architecture for secure download to remote, embedded equipment. The security of this scheme comes from a shared secret key that is known only to the equipment manufacturer and is embedded in the equipment at the time of manufacture.

The three main tasks of the secure download application are to:

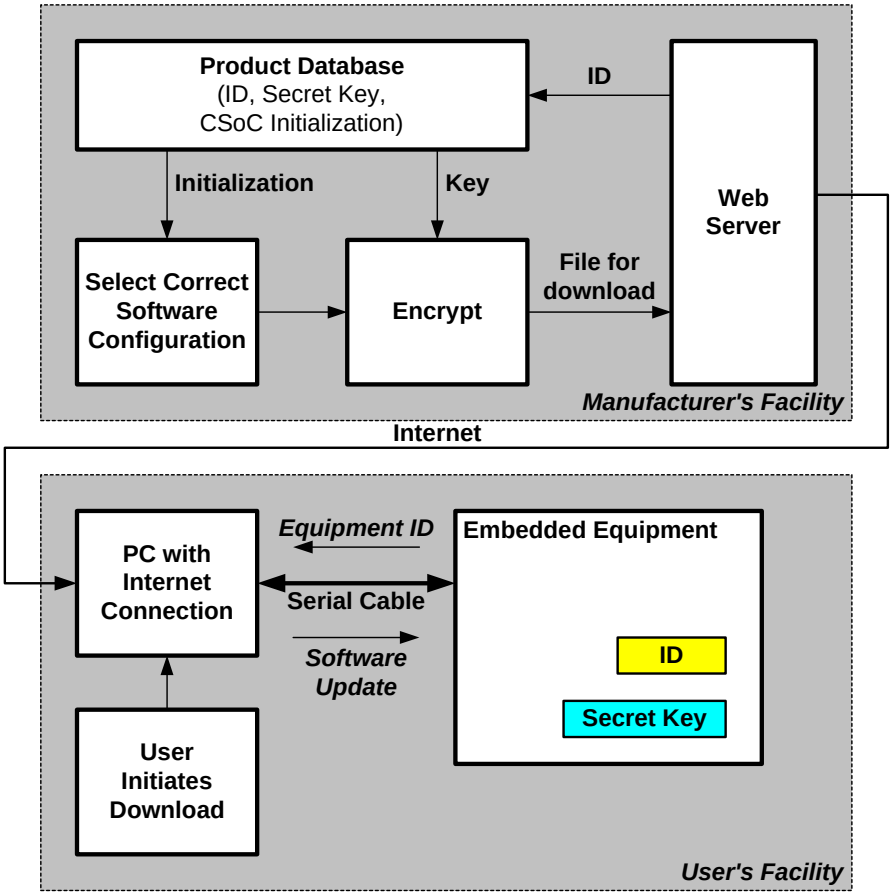


Figure 1. A potential software update architecture leveraging the world-wide web infrastructure. Simple approaches are possible using traditional phone lines or UARTs.

- Manage the non-volatile Flash memory that stores the currently operating application and the new image being downloaded.
- Implement a cryptographic protocol that prevents unauthorized parties from downloading to the equipment. The ability to prevent third parties from copying the image being downloaded to the device is another desirable feature.
- Implement a reliable communications mechanism to transfer 100K byte files without error between the PC and the Development Board.

Managing the Flash Memory

The Triscend E5 development board contains an AMD29LV002BB 256K-byte Flash memory chip [3] and a 512K byte SRAM device. The SRAM device is not used in this application.

The size of the initialization data file for the CSoC device varies according to the size of the user's microcontroller code, but is normally more than 64K bytes and less than 100K bytes. Much of this data initializes the E5's Configurable System Logic (CSL) matrix. Therefore, it is possible to store, at most, two complete CSoC images in the development board's 256K byte Flash memory chip.

The validity of a downloaded Flash image cannot be determined until the entire image is available. There are 40K bytes of on-chip SRAM memory on the CSoC device, though this is not enough to buffer a complete download image. There is an external SRAM device on the development board, but the goal of this application note is to demonstrate stand-alone secure updates. Consequently, the external SRAM is not used. In this application, the download image must be immediately stored to Flash as it is downloaded. Although the current CSL configuration is loaded into and operating from on-chip RAM memory, its image in

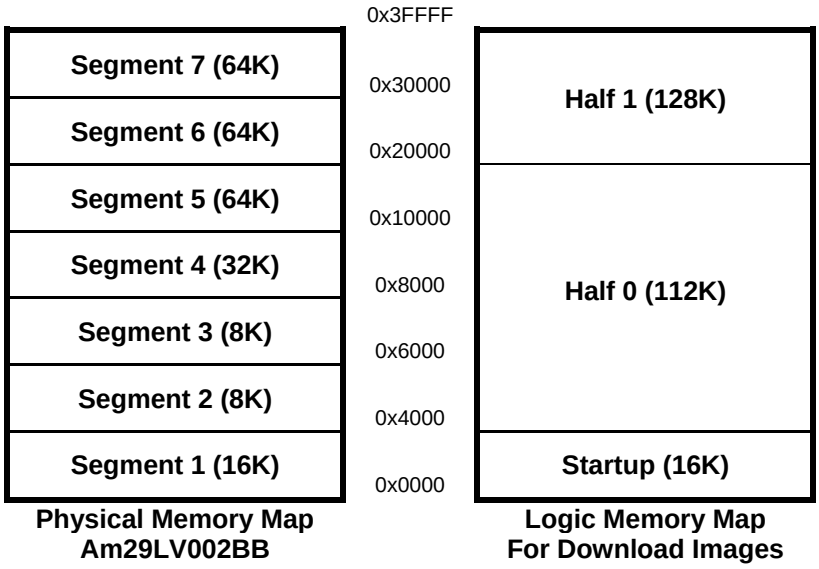


Figure 2. Subsequent updates are stored in either Half 0 or Half 1 in the Flash memory. The various Flash segments, with varying sizes, complicate the update software little.

Flash memory must not be overwritten by the new image being downloaded. If the download does not complete successfully or is rejected by the security software, the current image must remain intact in non-volatile memory so that it is still operational after a system reboot.

A FLASH memory has the ability to electrically erase a region of the device. The AM29LV002BB memory on the Triscend E5 Development board is composed of seven such 'segments' each of which can be individually erased.

The properties of Flash memory complicate the implementation of software download:

1. It is not possible to overwrite a single byte of Flash memory. Instead, an entire segment, containing as much as 64K bytes of data, must be erased.
2. Erasing a segment and writing a byte of memory require that the Flash device execute a programming algorithm. While this algorithm is running, the Flash memory is not available for use by the CSoC. This implies that the CSoC cannot fetch instructions from the Flash memory while programming. Therefore, code that controls the Flash must execute from the E5's on-chip RAM.
3. The segments within the Flash memory are of various arbitrary sizes.

The currently operating CSoC configuration must not be disturbed until the newly downloaded configuration is verified. To guarantee this, two regions capable of holding an entire CSoC configuration are provided in the development board Flash memory, as shown in Figure 2. These regions (Half 0 and Half 1) alternately store the download image. When the present CSoC configuration is stored in Half 0, a new download image is saved into Half 1 and *vice versa*. A third small region at the beginning of the memory (Startup) holds a simple bootstrap loader, discussed later.

Typical E5 Initialization Process

Before discussing how to implement updates, an overview of the typical E5 initialization process is helpful. A Triscend E5 CSoC device is initialized by applying power or by asserting the reset pin, RST-. During initialization, the 8032 locates and starts executing a primary initialization routine located in the ROM embedded within the CSoC device. This ROM is only used for primary initialization and is not visible to the user application. The purpose of the primary initialization routine is to locate and load the user's initialization data, called secondary initialization. Based on the logic levels applied to the VSYS-, SLAVE-, and A31/PMOD pins, the primary initialization routine determines where the secondary initialization routine resides--either in external byte-wide memory, external serial memory, or internal SRAM. This application note only considers the case where secondary initialization code is located in external Flash.

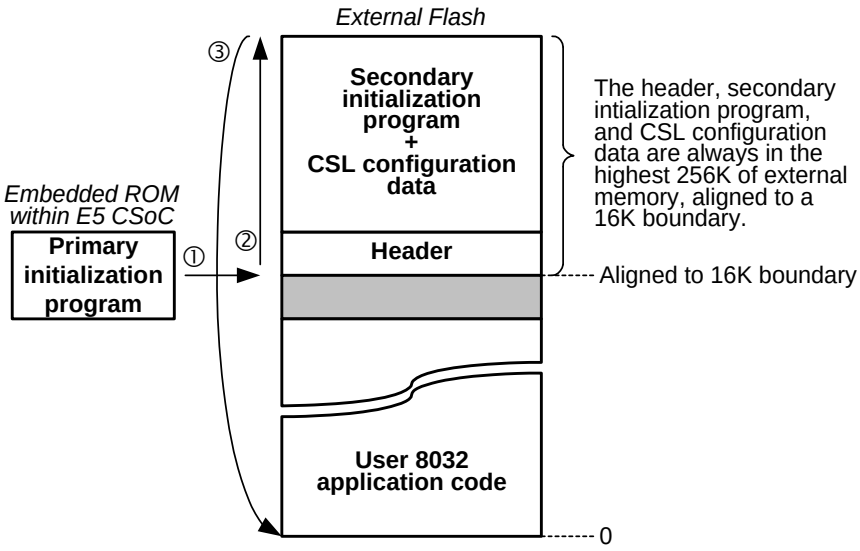


Figure 3. In a typical application, only a single initialization image is loaded from external Flash.

The primary initialization routine performs the steps outlined in Figure 3.

1. The 8032 begins executing the primary initialization code stored in the embedded ROM. The routine searches the last 256K bytes of external memory to locate the four-byte 'header' 0x8025A5A9, which is aligned on a 16K-byte boundary. On the E5 development board, there are only 256K bytes of external memory and therefore only 16 possible locations to search. The header marks the beginning of a valid secondary initialization routine. After locating a valid header, a code mapper is programmed so that the 8032 can start executing the secondary initialization route.
2. The secondary initialization routine then programs the CSL matrix from the bitstream information created by FastChip.
3. After programming the CSL matrix, the secondary initialization program modifies the code mapper to point to location 0x80_0000, which is the start of external memory. The secondary initialization routine then causes the 8032 to jump to logical address 0x0000, where the user's 8032 application program resides. The code mapper redirects this logical address to physical address 0x80_0000. Physical address 0x80_0000 then points to the external Flash. Because only the lower 18 address lines connect to the Flash, physical address 0x80_0000 points to the lowest location within the Flash memory, 0x0_0000.

WHICH ADDRESS WHERE?

Discussing addresses during initialization can be somewhat confusing because there are several frames of reference for address spaces, plus redirection through the internal code and data mappers, as shown in Figure 4. The 8032 processor itself has 64K byte address spaces for external data (XDATA) and program code (CODE). These 16-bit addresses are referred to as 'logical' addresses.

The on-chip Configurable System Interconnect (CSI) bus provides a 32-bit address space into which various resources are mapped. These 32-bit addresses are referred to as 'physical' addresses.

External memory appears between 0x80_0000 and 0xFF_FFFF on the CSI bus. So, in the example above, external memory address 0 (the start of the Flash chip) is accessed through on-chip CSI bus address 0x80_0000 which is mapped by a CSoC code mapper onto 8032 code space address 0x0000.

Modifying the Initialization Process for Remote Updates

Supporting remote Flash updates requires some modifications to the typical E5 initialization process. Because the CSoC always jumps to application code at 0x0_0000 in external memory,

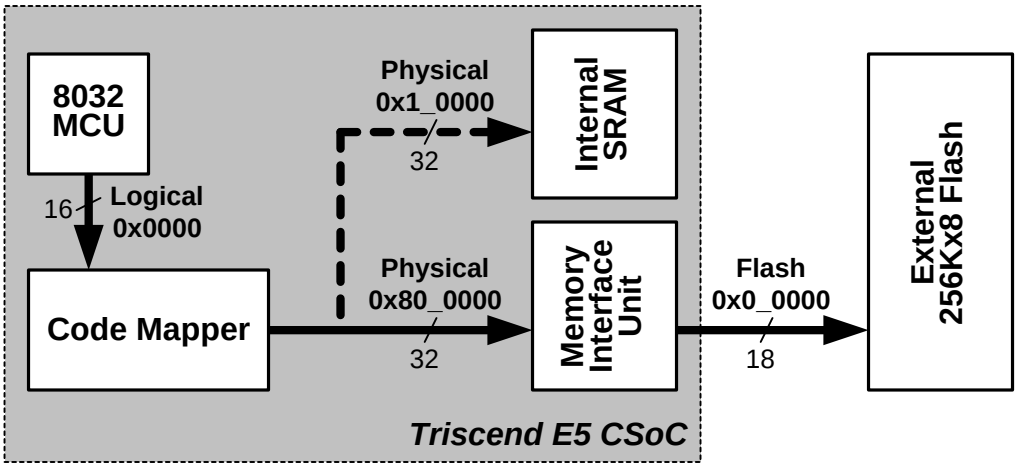


Figure 4. The 8032's 16-bit logic address is re-mapped to a 32-bit physical address space.

valid code must reside at that location regardless of whether the updated configuration image is loaded from Half 0 or Half 1. The code at 0x_0000 must not erased at any point in the download process. If the CSoC were powered off or re-initialized while this segment of Flash memory was erased, the CSoC could no longer boot correctly when power was restored. Therefore, the first segment of Flash memory is assigned to a special startup program, which is not altered by download. The Startup program modifies the typical initialization process, as shown in Figure 5.

The Startup routine performs various functions:

- Identifies the valid initialization image, and determines whether the image appears in Half 0 or Half 1.
- Locates the user's 8032 application code, which starts at the first location in the valid half of memory.
- Copies the 8032 application code to the E5's internal SRAM memory. As discussed earlier, any code that modifies Flash memory

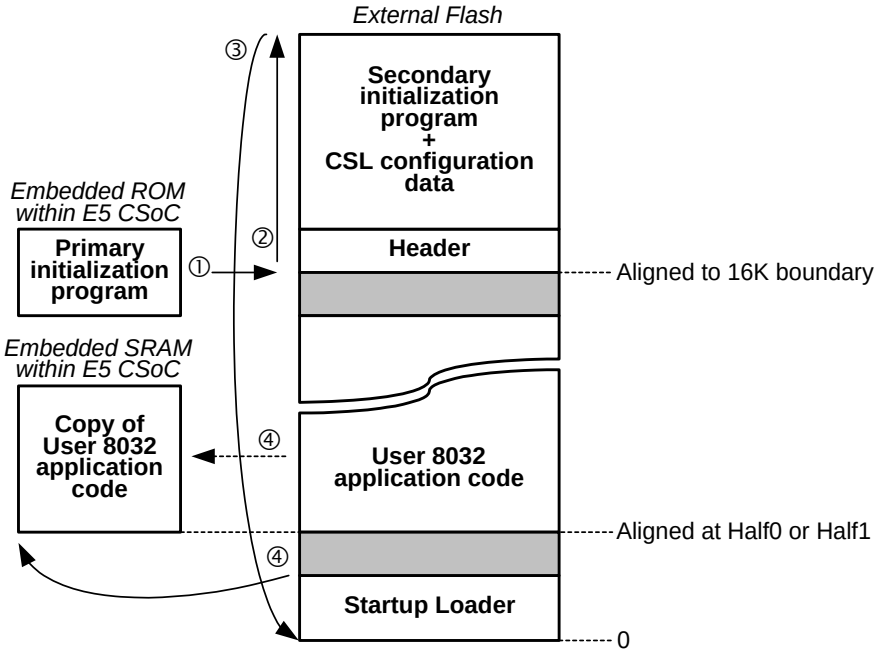


Figure 5. A modified initialization process supports secure Flash updates.

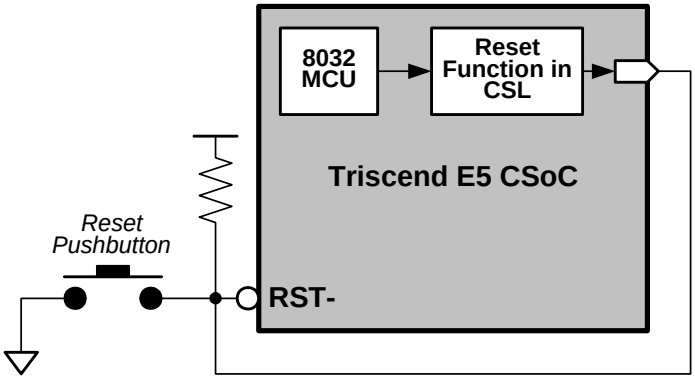


Figure 6. A Triscend E5 CSoC can reset itself using CSL/PIO circuitry.

needs to operate from on chip RAM. Because the internal SRAM space is limited, only those functions required during Flash updating need to be copied into SRAM.

- Sets up a code mapper so that the 8032 can execute code from internal SRAM.
- Jumps to internal SRAM and starts executing the 8032 application code.

The download application activates a new download image after it is verified cryptographically. The application then erases the Flash segments containing the currently operating image. This operation deletes the header from the corresponding half of Flash memory so that only the newly downloaded image has a valid header. If the downloaded image does not verify correctly, it is immediately erased. Consequently, that half of memory will not contain a valid header.

A potential security issue exists by writing the downloaded data to Flash as it arrives. A problem arises if the embedded system was reset or lost power after writing the header to Flash, but before the cryptographic software had verified the complete download. The loader program might detect the header in an incomplete or spurious image and activate it. To prevent this, the software reserves the position of the magic marker in the downloaded image but does not actually write it to Flash until after the image is cryptographically checked. All other data in the image is immediately written to the Flash. The software also checks that there is exactly one header in the downloaded image.

Activating the Downloaded Image

For most applications, the downloaded software would activate automatically after installation. However, the updated image contains CSL programming information, which is only loaded after a full hardware reset. A reset signal from 8032 software or CSL is not sufficient because an application reset only resets the 8032 and causes the 8032 to execute the 'new' software. Unfortunately, an application reset does not force the CSoC device to reload the CSL with the new configuration. If the new software depends on features in the new CSL configuration, which is not present in the currently loaded CSL configuration, then this mismatch could cause the system to fail.

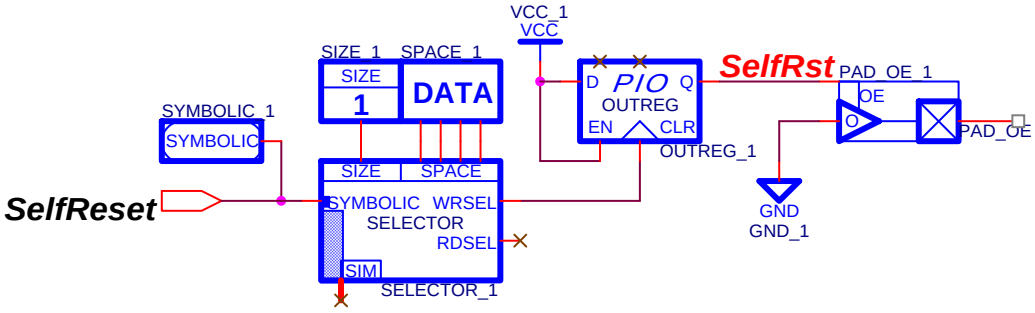


Figure 7. Self reset circuit using a Selector and one PIO pin, drawn in OrCAD Capture.

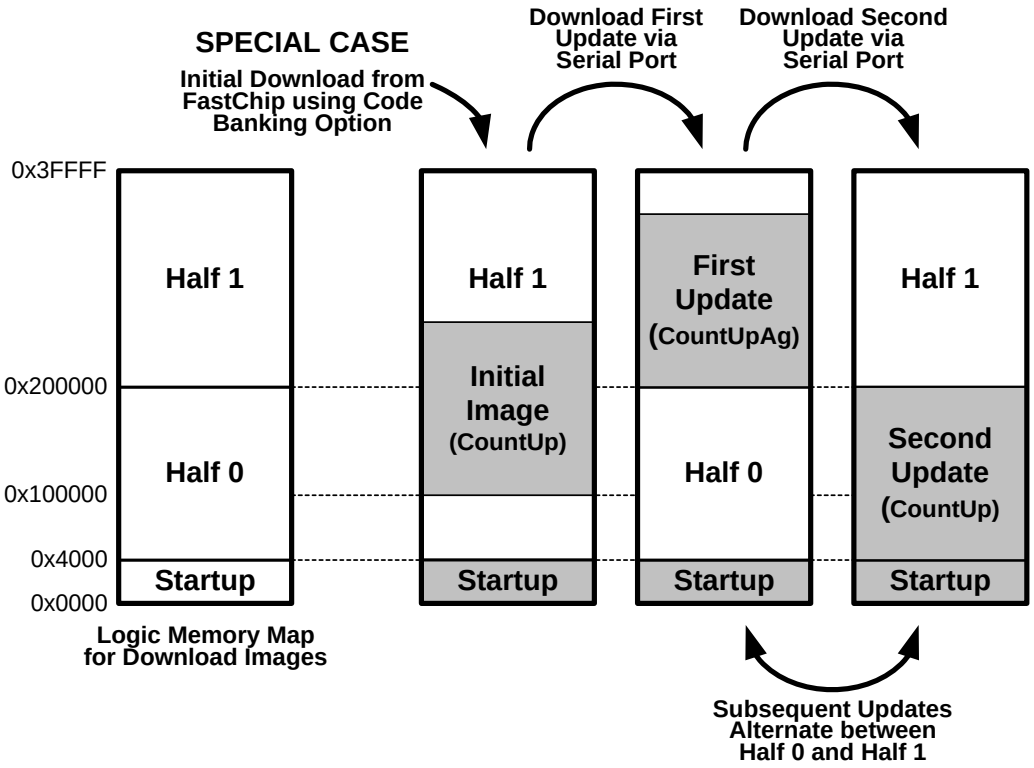


Figure 8. The Initial Download from FastChip is a special case. Subsequent updates are stored at either Half 0 or Half 1 in Flash memory.

The only sources for a full hardware reset on the E5 CSoC are a power-on-reset, the external reset pin on the E5 (RST-), or a JTAG reset. If the downloaded image must be activated without intervention, then the CSL design must trigger a hardware reset by connecting a PIO output to the E5's RST- reset pin via a trace on the printed circuit board, as shown in Figure 6. The circuitry to cause a self-reset condition can be as simple as that shown in Figure 7, which requires a single Selector and a single PIO pin. Normally, the PIO output is floating or high-impedance. This is so that other devices can connect to the E5's RST- pin without contention. When the 8032 writes to the symbolic address "SelfReset," the Selector's WRSEL output is asserted, causing the PIO output flip-flop to load a logic High. This, in turn, enables the output, which drives a logic Low on the pin. This logic Low forces a device-wide reset. When the E5 begins to re-initialize, the PIO output is forced to high-impedance, de-asserting the RST-. The output flip-flop is also loaded with zero, clearing its output and disabling the PIO.

On the development board, however, there is no trace from any of the PIO pins to the RST- pins. Therefore, after successfully downloading an image, press the reset button on the development board or cycle the power to the board to activate the new image.

Initial Configuration

The initial configuration is downloaded to Flash memory using FastChip and the E5's JTAG interface. This configuration contains two independent 8032 programs, the Startup program and the user application, as well as the CSL configuration information. FastChip loads multiple .Hex files as 8032 code segments using the Bank Switching option available in the FastChip Download menu. One complication is that FastChip locates these files on 64K boundaries. The first .Hex file is loaded at address 0x00000 and the second at address 0x10000 (the next 64K byte boundary). For this application, however, it would be better if the second .Hex were loaded at 0x04000, which is the beginning of Half 0 in the Flash memory map. A large CSoC image starting at 0x10000 could overlap the beginning of Half 1 of the memory. Unfortunately, the FastChip 1999 provides no means to more precisely locate the .Hex file in Flash.

There are two ways of addressing this problem:

1. A custom program could process the .Hex file and move the configuration to 0x04000 prior to download.
2. The Startup program can recognize user code starting at 0x10000 as a special case and call the correct address.

This second and simpler approach is used in the demonstration application. In operation, the download function deletes a portion of the initial configuration already stored in Flash. This happens as the download function clears Half 1 of the memory, preparing to store the downloaded image. Fortunately, this does prevent the initial configuration code from operating because it is copied into the CSoC's on-chip RAM. However, it means that at least one software update must take place before the CSoC is ever reset or before the product is shipped to a customer. Otherwise, there would not be a complete image in the Flash memory. Consequently, the initial configuration serves merely as a temporary 'bootstrap' until the first download. After a successful first download, the Flash is configured correctly for secure updates.

Figure 8 shows graphically how the initial configuration and subsequent updates are loaded into Flash.

An attractive side effect of this technique is that all products could have the same initial configuration loaded during board assembly. Then, using software download, each product is personalized with a unique secret cryptographic key before shipping them to customers. The initial configuration key used during board assembly does not have to be secret because it will be changed by the first software download, which occurs before product shipment.

Cryptographic Security

The primary purpose of the software download cryptographic security is to prevent people, other than the equipment manufacturer, who do not have physical access to the embedded system, from changing its software configuration. A secondary purpose is to protect the software from being copied in transit between the manufacturer and the equipment user.

If someone has physical access to the equipment, that person could potentially determine the secret key by accessing the Flash memory in the equipment that holds its program. However, if they have this level of access, they could also simply reprogram the Flash memory to install any software they liked. Determining the secret key for one piece of equipment gives no information about the key for other units. Therefore, the fact that the key could be determined by opening the box and reading out the Flash memory does not constitute a serious additional vulnerability compared with non-downloadable systems.

If securing the system against attacks from people with physical access is an issue, then tamper-proofing techniques must be used. The Triscend CSoC 'secure' battery backed mode might be of benefit in this context. Tamper proofing is outside the scope of this application note.

The shared secret key scheme has the property that only the manufacturer can create valid download files for the embedded system. Unlike the secure-applet technology used on web browsers, it is not possible to download third party software securely. Secure applet protocols require public-key cryptography and are considered too complex to implement on an 8-bit microcontroller.

The shared-secret key scheme implemented here uses the Data Encryption Standard (DES) which is a public domain, standardized algorithm. Triple DES is used in Cipher Block Chaining (CBC) mode as the underlying cipher. This provides extremely strong cryptographic protection. Recently it was reported that single DES was successfully 'cracked' using a large number of machines on the Internet to search the key space. Triple DES uses a 168-bit key compared with single DES's 56-bit key. The best known attack on Triple DES requires 2^{56} times more computation than cracking single-DES. Triple-DES is widely used for high-security applications.

While Triple DES provides excellent security there are two reasons why it may be desired to 'fall back' to single (56 bit) DES.

1. If the product is exported outside the United States, using 56 bit single DES will make it easier to obtain export clearance.
2. The decryption time is the limiting factor on the speed of software download. Single DES will run approximately three times faster than triple-DES.

Although single DES has been cracked, the amount of effort required to do so is large and single DES may provide more than adequate security for many applications.

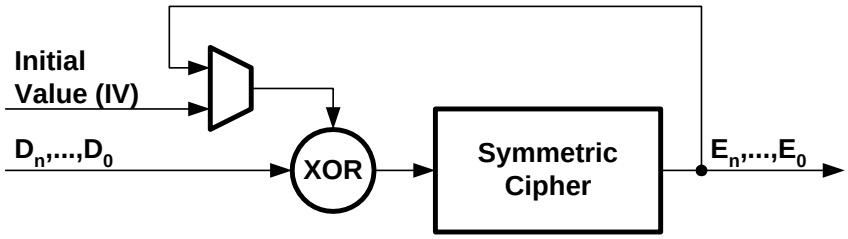


Figure 9. Cipher Block Chaining (CBC) Mode.

The Cipher Block Chaining mode of DES, shown in Figure 9, has two important properties in this application:

1. The ciphertext for a 64 bit block in the file depends on the corresponding plaintext block and all previous plaintext blocks and a randomly chosen Initial Value (IV). This ensures that a bitstream file encrypted on two separate occasions will result in completely different ciphertext.
2. The last ciphertext block can be encrypted again and used as a Message Authentication Code (MAC). If this block decrypts to the expected value, then the entire message has been transmitted correctly. Since computing the MAC involves knowledge of the secret key, an attacker cannot compute a valid MAC for his own download file.

This description of the cryptographic software is relatively brief because the topic is covered elsewhere. Reference [1] is an excellent practical source on cryptographic algorithms.

Key Management

In order for the shared secret key scheme to work correctly, the same secret key used to encrypt the download image is also stored in the embedded equipment. Each unit of embedded equipment should have a unique secret key.

The demonstration software simply encodes the keys as constant arrays in the cryptographic code, which is adequate for evaluation purposes. However, in a real deployment, mechanisms would ensure that downloads are encrypted with the correct key and would customize the downloaded binaries to contain the correct secret key. These key-management utility programs are beyond the scope of this application note.

Communications Protocol

Download speed, while important, is not critical because software updates occur relatively infrequently. There are, however, several requirements for the serial communications protocol between the PC and the embedded system.

1. It must be able to transfer large (~100K byte) files reliably. Unless some kind of error correction is provided, almost all downloads are likely to fail.
2. It must supply data to the decryption software in the correct order. There is insufficient memory available within the E5 CSoC to buffer a complete download image prior to decryption using Cipher Block Chaining (CBC). With CBC, one 64-bit word of cipher text cannot be decrypted until the previous word has been decrypted. This constraint rules out 'pure' datagram protocols such as UDP [2].
3. The application software must be able to prevent new data from arriving while it decrypts the previous packet. Decryption time is the bottleneck on download speed.
4. The communications protocol must have a small code footprint because it shares the limited on chip RAM with the user application, the cryptographic code and the flash management code.

Because of the constraints on code size, the communications interface avoids a standard TCP/IP protocol stack. Instead, the application uses an ad-hoc packet-based mechanism that works effectively over a serial cable between a host PC and the development board. The communications code is separate from the remainder of the application code. If required, the communications code could be replaced with a TCP/IP stack.

The download code uses a packet-based mechanism to transmit the image. Packets are binary data framed using the mechanism defined

for Serial Line IP (SLIP) [2]. The packets contain a length count, a checksum and data. If the checksum is incorrect, a Negative Acknowledge (NAK) character is sent back to the PC and the packet is re-transmitted. If the checksum is correct, the packet is decrypted and stored in Flash memory. After decryption is complete, an acknowledge character (ACK) is sent enabling transfer of the next packet. If the message authentication code (MAC) on an entire download image is incorrect, then a Cancel character (CAN) is sent indicating that the software will not be activated. Otherwise, the last packet is acknowledged with End of Text (ETX) indicating that the download will be activated. After receiving a correct download image, the download application is disabled but the 'user' task continues to execute.

The Software

Application Note Design Files and Software Source Code

The various files for this application note are available from the Triscend web site as a compressed archive file.

`www.triscend.com/app_notes/an02.zip`

CountUp	FastChip project
CountUp.c	Keil 'C' program for design
CountUpAgain	FastChip project
CountUpAgain.c	Keil 'C' program for design
Download	Keil 'C' project
download.c	Top-level C file
datagram.c	Serial comm. protocol
des.c	DES decryption
decrypt.c	Converts decrypted data into CSoC image
install.c	Saves downloaded data to Flash and transfers control to new image
nvdata.c	Low-level Flash access
serial.c	Interrupt-controlled UART interface
Startup	Keil 'C' project
Startup.c	Top-level
Nvdata.c	Low-level Flash access
Jump2Rom.a51	Low-level code to activate on-chip RAM
des_pc	Microsoft C++ project
download.c	Top-level C++ file
readhex.c	Reads Intel Hex file and stores as large array
encrypt.c	Application level data communications and DES cipher block chaining (CBC) mode
des.c	Single and triple DES
datagram.c	Serial comm. protocol
Debug	Contains compiled executable and debugged designs for download

Limitations and Restrictions

This application design and associated software is intended as a demonstration only. Triscend Corporation does not make any representation or warranty regarding this design, the associated software, or any item based on this design. Triscend disclaims all express and implied warranties, including but not limited to the implied fitness of this design for a particular purpose and freedom from infringement.

Without limiting the generality of the foregoing, Triscend does not make any warranty of any kind that any item developed based on this design, or any portion of it, will not infringe any copyright, patent, trade secret or other intellectual property right of any person or entity in any country. It is the responsibility of the user to seek licenses for such intellectual property rights were applicable. Neither Triscend nor its agents shall be liable for any damages arising out of or concerning the use of the design, including liability for lost profit, business interruption, or any other damages whatsoever.

PC Software

The PC software has two main functions. It encrypts the downloaded file and transfers it over the serial interface to the development board. In an actual deployment, encryption is performed at the manufacturer's facility and only the download program would run on the user's PC. However, during development it is convenient to package the functions together in a single program.

The software is supplied as a Microsoft C++ project and has been tested under Windows NT 4.0, service pack 5 (SP5). It uses the COM2: port on the host PC. If this port is not available, the source code must be edited and recompiled.

Bootstrap Loader (Startup)

The bootstrap loader is stored in the first segment of the FLASH memory using JTAG download from FastChip. The loader program locates the

user application resident in Flash memory and loads it into CSoC on-chip SRAM. It then passes control to the user application, now resident in on-chip RAM.

The Bootstrap loader is supplied as a Keil C project.

Download Code (Download)

The download code is supplied as a library, which must be linked to the user's application code. The download code continuously monitors the 8032 serial interface and downloads any file that appears.

The download code is supplied as a Keil C project and uses the RTX-51 tiny real-time operating system bundled with Keil C. The download code operates as a task under RTX-51. If the user application has stringent real time constraints and must remain active during image download, then use RTX-51 tiny to limit the amount of processing cycles available to the download function. The download code is only activated when characters are sent to the on-chip UART, so it normally will not steal processor cycles from the user application.

The download application is too large to compile with the restricted evaluation version of Keil C supplied on the Triscend FastChip CD-ROM, which is limited to 2K bytes.

The Demonstration Programs

To illustrate the concepts of remotely updating a product via download, two variations of a simple application are provided.

1. **CountUp** — uses the timer function of RTX-51 and a seven-segment decoder design built in CSL logic to display a count value on the development board LED displays.
2. **CountUpAgain** — Implements a counter like the CountUp design, but increments the display by a four-bit number entered using the top four switches on DIP switch S1. This design has additional CSL components to interface to the DIP switch and slightly different 8032 code from CountUp.

Correctly swapping between these designs requires that both the CSL logic and the 8032 code be updated.

These demonstration designs are provided as FastChip 1999 projects and should be copied to your FastChip projects directory. The corresponding Keil C projects are also within the FastChip project directory.

Running the Demonstration

The Demonstration

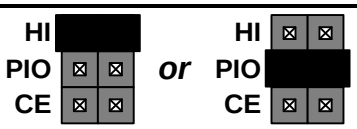
As described earlier, the demonstration uses two simple applications that run on the development board:

1. The first application, **CountUp**, simply counts continuously on the seven segment displays.
2. The second application, **CountUpAgain**, counts on the seven segment displays but increments the count by a 4-bit number entered on the DIP switch S1, switches 1 through 4.

The demonstration has four stages:

1. Install the initial image containing the **Startup** loader and the **CountUp** application to the on-board Flash via JTAG using FastChip's Download function.
2. Replace the **CountUp** application with **CountUpAgain** using secure software download.
3. Replace **CountUpAgain** with **CountUp** using secure software download.
4. Change the key used for encryption on the PC and verify that the downloaded image is rejected.

Table 1. Triscend E5 Development Board jumper settings for Flash download.

JP2 SRAM	JP4 Flash	JP14 Reset	JP17 Serial PROM
			

Setting up the Development board

The Triscend development board is described in detail in reference [5]. A quick checklist is provided:

1. The board power supply must be set up according to the instructions in [5]. Positions A31, VSYS and SLAVEB of DIP switch S2 must be in the open (HI) position. Set jumper per Table 1, JP4 to CE, enabling Flash memory. Set jumpers JP2 and JP17 to disable SRAM and serial memory respectively.
2. Use the FastChip I/O editor to route from the UART RxD signal to pin 145 and the TxD signal to pin 147, as shown in Figure 10. This step is already performed in the example projects.
3. Set clock selector jumper, JP16, to X2, selecting the 25 MHz oscillator chip. If the oscillator chip frequency is not 25 MHz, modify timer values in the example software appropriately. In FastChip, configure the 'Clocks' dedicated resource to select 'Clock Input to XTAL/BCLK', as shown in Figure 11. This step is already performed in the example projects.
4. In FastChip, configure the 'MIU' dedicated resource so that the Memory Interface Unit addresses 256K bytes of external memory. This step is already performed in the example projects.
5. Connect a 9-pin DB9 male-to-female serial cable between the PC serial port COM2 and the development board. The board interface looks like a Modem (DCE) so a normal modem cable, not a null-modem cable, is required. Most PC modem cables have a DB-9 connector on the PC end and a DB-25 at the Modem end. Using this type of cable also requires a DB25-to-DB9 converter plug. The development board does not support hardware CTS/RTS handshaking, so if you make up cables by hand only, the receive/transmit data and power/ground wires need to be connected.
6. The PC software is set up to use COM2. If you wish to use a different communications port, then modify the source file (datagram.c) and recompile.

The JTAG download interface is required to download the initial software configuration.

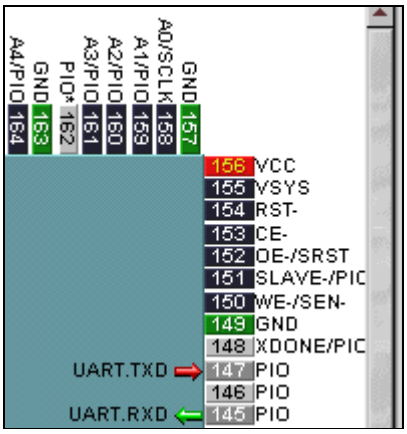


Figure 10. Place the UART TxD signal on pin 147. Place the RxD signal on pin 145.

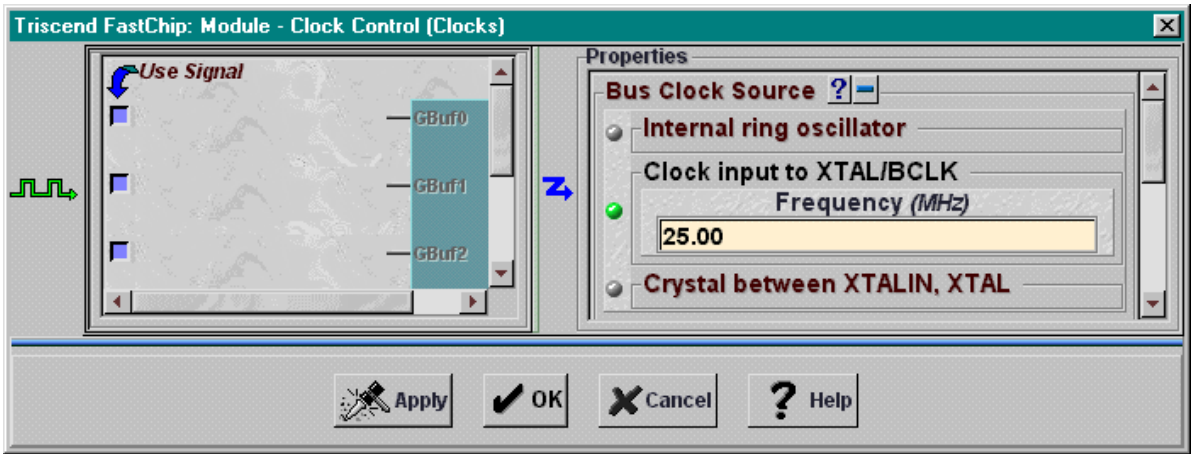


Figure 11. Configure the Bus Clock Source for BCLK and set the frequency to 25.00 MHz.

Installing the Software

The two CSoC designs must be installed in the Projects directory under the FastChip installation path. Simply copy the directories across.

Creating the Initial Image

The initial image is downloaded to the development board Flash memory through the JTAG cable. It contains both the startup code and the application code. It is created as follows:

1. Open the **CountUp** project in FastChip.
2. Using the FastChip **Download** function, select '**Direct Program**', '**Flash Memory**' and '**AM29LV002BB-120**' as the part name. Select '**Use Bank Switching**'. Load Bank 0 with the '**Startup.hex**' file from the Startup project. Load Bank 1 with the '**CountUp.hex**' file created by Keil C by building the **CountUp** project in the current directory. These options are shown in Figure 12.
3. Click **OK** to program the on-board Flash EPROM via JTAG. The process might take a few minutes while FastChip erases and then programs the Flash.

After downloading the files, you will see the two seven-segment LEDs count continuously.

Downloading the First Update

Create a new hex file to download to and update the development board

1. Open the **CountUpAgain** project in FastChip.
2. On the **Download** menu, deselect '**Use Bank Switching**'. Select the hex file '**CountAg.hex**' created by building the Keil C project in the current directory. Select '**Parallel EPROM**' as the memory device, '**AM27C020-200**' as the part name, and '**Generate Hex File**'. Save the resulting file as '**CountUpAgainImage.hex**'. These options are shown in Figure 13.
3. Click **OK** to generate a hex file containing the full 8032 code and CSL configuration.

To download the new update to the board:

1. Run the program des_pc.exe (built from the Microsoft C project) in the CountUpAgain directory.

   ```
   des_pc CountUpAgainImage.hex
   ```

2. When the PC reports a successful download, reset the development board by pressing the reset button. This starts up the updated hardware and software.

Try various values of the top four switches of DIP switch S1 and see the counter increment by various amounts. Initially, the DIP switch may be set to 0 so the counter may not increment until you change its value.

Downloading the Second Update

Create a complete image binary in Project **CountUp** as you did for project **CountUpAgain**, the first update. Do not enable bank switching as you did when creating the initial image. Use **CountUp.hex** as the application code file. Save the result in **CountUpImage.hex**. The options are shown in Figure 14.

Download in the same way and after reset the board when complete. The two LED displays increment by one, regardless of DIP switch S1.

Cryptographic Verification

To prove that the cryptographic verification is working, change the key used by the PC to encrypt the downloaded update image. The key is contained in the routine des3_ede in file DES.c. Change one of the keys keyA, keyB or keyC, then rebuild pc_des.

Note that KeyA==KeyB==KeyC at present, which causes the triple DES routine to implement single DES encryption. The download program on the development board is also set for single-DES encryption at present to reduce download time.

Try downloading again and you will see that the download is rejected.

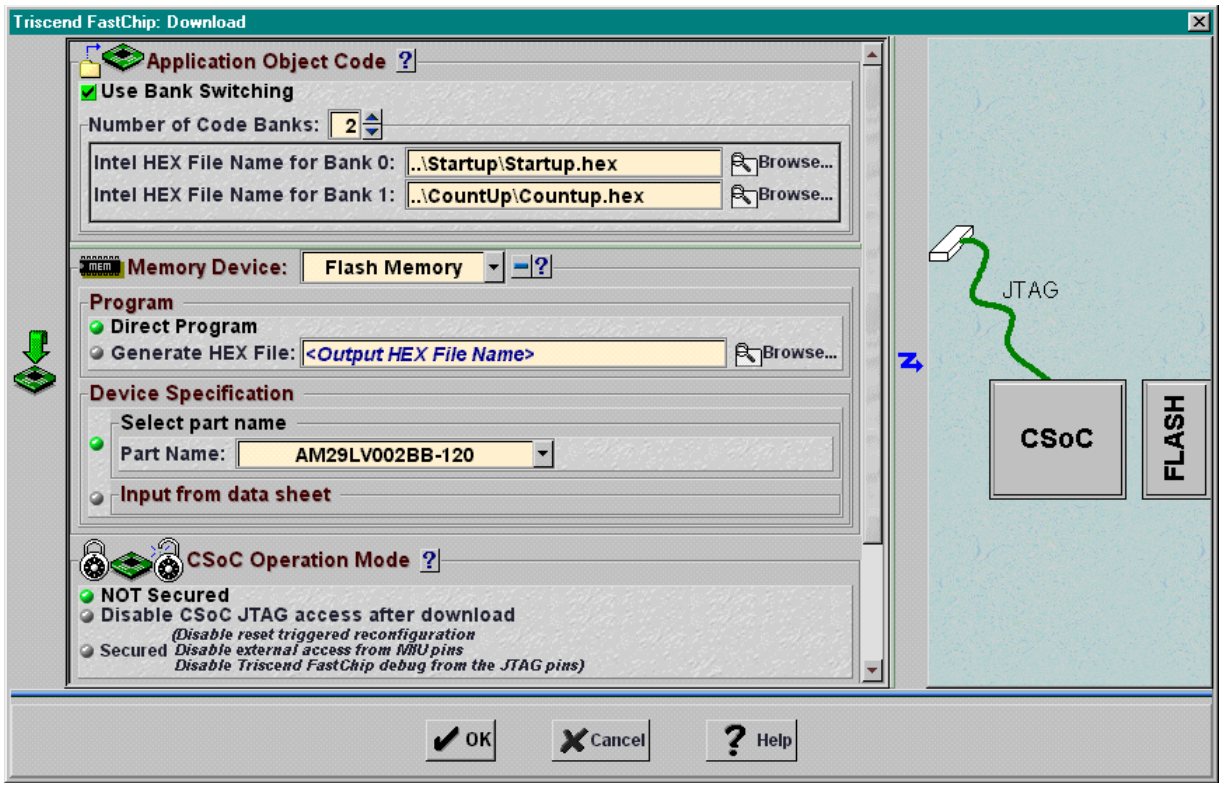


Figure 12. Download Settings for Initial Image.

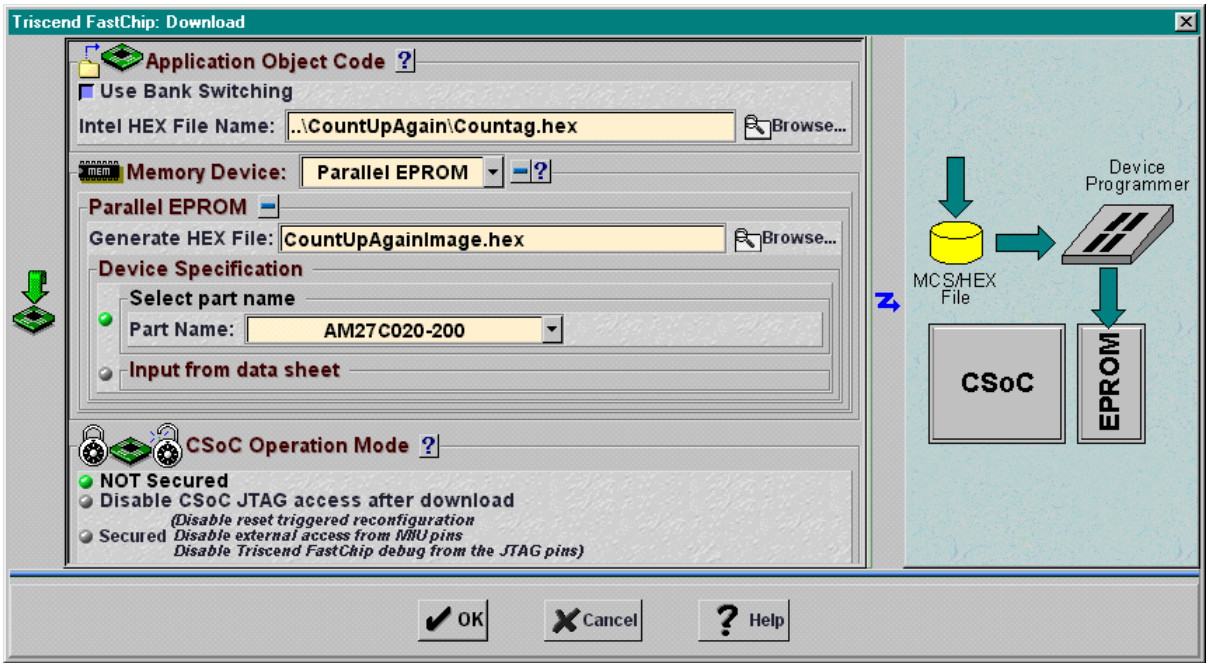


Figure 13. Download Settings for First Update Image.

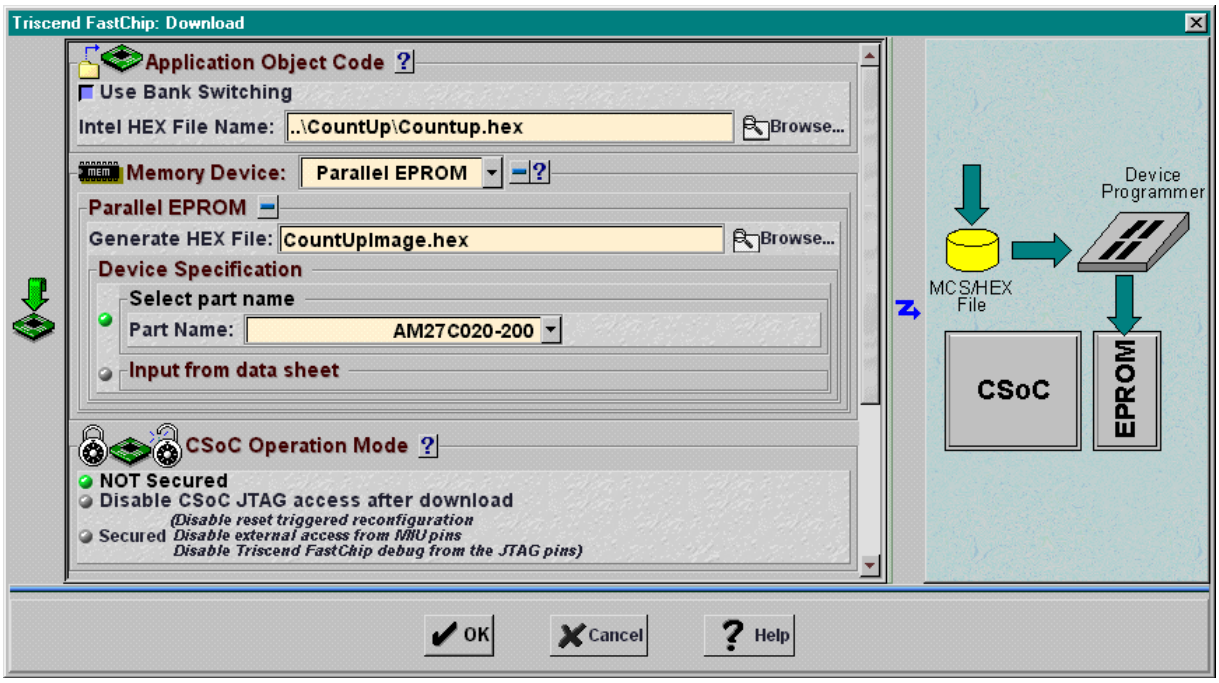


Figure 14. Download Settings for Second Update Image.

Summary

This application note demonstrates a highly secure remote update capability using a Triscend E5 Configurable System-on-Chip (CSoC) device. The supplied demonstration code can be extended in many ways, but it provides the basic components to develop a field-upgradable product without sacrificing security.

Literature

Textbooks

1. **Applied Cryptography**, second edition, Bruce Schneier. Wiley. ISBN 0-471-12845-7.

2. **TCP/IP Illustrated, Volume 1. The Protocols**, W. Richard Stevens. Addison Wesley. ISBN 0-201-63346-9.

Datasheets

1. "Am29LV002B 2 Megabit (256Kx8-bit) CMOS 3.0 Volt-only Boot Sector Flash Memory", Publication 21520, Rev: C. January 1999. AMD Corporation www.amd.com.

Triscend Literature

1. "Triscend E5 Configurable Processor Family," product description www.triscend.com/products/dse5csoc.pdf.

2. "Triscend E5 Development Board User's Guide" www.triscend.com/products/Dev_Board_Guide.pdf.

Acknowledgements

Triscend thanks Algotronix, Ltd. for demonstrating the feasibility of these concepts and for creating the application code.

Algotronix, Ltd.
P.O. Box 23116
Edinburgh EH8 8JQ
Scotland

Tel: +44 131 556 9242
Fax: +44 131 556 9247
Web: www.algotronix.com

Revision History

Revision	Date	Comment
1.00	6-JAN-2000	Initial release.

Triscend Corporation
301 N. Whisman Rd.
Mountain View, CA 940403-3969

Tel: 1-650-968-8668 x166
Fax: 1-650-934-9393

E-mail: **contact_us@triscend.com**
Web: **www.triscend.com**

AN02

Implementing Secure Remote Updates using Triscend E5 Configurable System-on-Chip Devices

— Table of Contents —

Triscend