



## AN012 - C8051F0xx Bootloader Considerations and Example

### Relevant Devices

This application note applies to the following devices:

C8051F000, C8051F001, C8051F002, C8051F005, C8051F006, C8051F010, C8051F011, C8051F012, C8051F015, C8051F016, and C8051F017.

### Introduction

This document describes the considerations and use of bootstrap loaders with the C8051F0xx family of devices. A bootloader provides in-system reprogrammability of program memory (FLASH) upon reset of a device or upon a received command. Considerations when implementing bootloader in the C8051F0xx family of devices is discussed and an example bootloader is provided.

### Bootloader Operation

A bootloader will download code from a specified source (host) upon reset of a device. The bootloader will receive a *bootload enable* signal upon reset, and configure the device to receive the code as raw data to download to a memory location (FLASH memory in the case of C8051F0xx devices). After successful download, the bootloader then redirects execution to the new code. Bootloaders in the C8051F0xx family of devices can take many forms, but most will follow the same basic sequence upon enabling the bootloader:

1. Configure the peripherals and input/output port pins for downloading data (e.g., SPI, SMBus, UART, etc.)
2. Erase memory for receiving the download.

3. Send a READY signal to the host indicating that it is ready to receive data.
4. Receive the download and store in memory. (This may include error control or a transmission protocol.)
5. Jump to the proper entry point of the downloaded code and begin executing the code.

### Hardware Considerations

The bootstrap loader requires a communication connection between the host and a C8051F0xx communication peripheral, and a means to signal the device to initiate the bootload sequence.

### *Pinout and The Digital Crossbar*

The C8051F0xx utilizes a Digital Crossbar to assign digital peripherals to port pins for external interface. (Please see application note “AN01 - Configuring the Port I/O Crossbar Decoder”.) The crossbar allows use of any combination of digital peripherals, but the user must take into consideration that software can change the device pinout.

In most cases, the bootloader will use the same pinout as the end application.



## Bootload Enable

Upon reset or other condition that requires in-system reprogramming, the device must have an input to signal the bootload sequence to begin. This can be done by reading a general purpose I/O port pin as a *bootload enable* signal. Once the pinout is decided for an application, a safe choice can be made for what pin should be used for a bootload enable signal. This way, the host or other hardware can signal the C8051F0xx to commence the load sequence. Alternatively, a bootload signal may be generated in software.

In the example provided in this application note, port pin P1.4 is the *bootload enable* input signal and is sampled at the end of hardware reset. A bootstrap load is initiated when port pin P1.4 is held low. Note that the reset default state of the port pins is high (logic '1'), so the input signals at GPIO pins after a hardware reset should be logic '0' to signal a bootload operation.

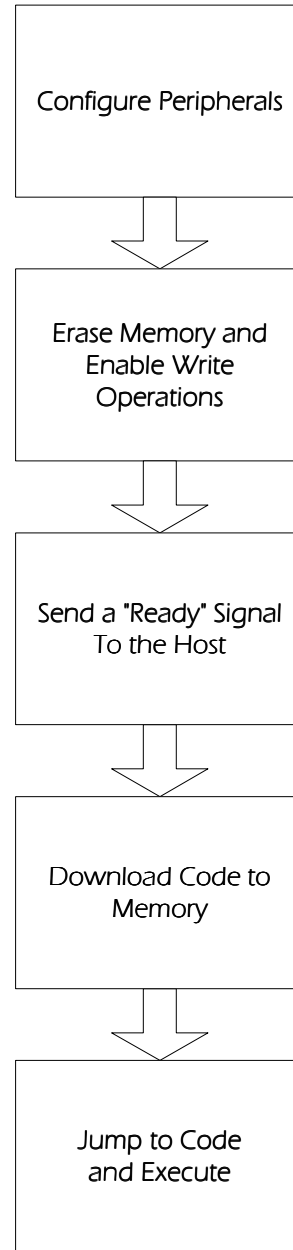


Figure 1. Bootload Flow Diagram

## Software Considerations

The software for bootloaders perform the basic functions in the program flow in Figure 1 above. When enabled, the bootloader must get the device ready to receive data. First, the bootloader will configure necessary communi-



cation peripherals. The bootloader program must then erase and write-enable memory for the download. To establish a communication link when the bootloader is enabled, the bootloader may establish the bit rate at which the host will send data via autobaud detection. Otherwise, the host and device may use a predetermined baud rate. Once the device is ready to receive data, the device should signal the host. The host will then send the data, which may be preceded by information about the download (such as the number of bytes the host will send).

The provided example is a simple bootloader program with no error control and downloads data in a contiguous memory space. A more sophisticated bootloader may use a transmission protocol and program non-contiguous memory addresses across multiple sectors.

### **Autobaud Detection**

A bootloader may utilize autobaud detection to determine the host transmission rate. For example, the host may send a training byte (e.g., 0x55) to be used by the C8051F0xx to determine the UART baud rate. Alternatively, the host may send a byte at a predetermined baud rate that establishes the transmission rate for the download. Thus, the device can configure UART and Timers to operate at the proper baud rate before receiving the data. In the example provided, autobaud detection is not used. Instead, a predetermined value of 115.2 kbytes per second and a system clock rate of 18.432 MHz is assumed.

### **Writing To FLASH Memory**

Code is written to FLASH during the bootloader sequence. There are special considerations when writing or erasing code from FLASH memory. Please see application note: "AN009 Writing to FLASH from Application Code". In

general, a bootloader will erase one or more 512-byte pages of FLASH. Once the FLASH is write enabled, the target is ready to receive the download. Note that both the write enable (PSWE) and erase enable (PSEE) bits must be set for erase operations. Once erase operations are enabled, writing to any byte in a FLASH memory page will erase the entire 512-byte page.

One constraint for using FLASH is the speed at which write operations take place. A write operation takes a maximum of 40  $\mu$ s per byte to complete. Thus, the host may not transmit at a rate higher than 25 kbytes per second, or 250 kbps (in UART 8-N-1 mode). This must be taken into consideration when using FLASH, as some peripherals can transmit faster than this rate.

The example bootloader in this application note uses the UART. The fastest common UART bit rate is 115.2 kbps, or 11.52 kbytes per second. This is well below the transmission speed threshold for writing to FLASH.

Note that the CPU core is stalled during FLASH erase and write operations. Peripherals such as the Timers and the UART continue to operate normally. Any interrupts which occur while the CPU is stalled will be held and serviced upon the completion of the write/erase operation.

### **Example Bootstrap Loader**

The example provided is a primitive bootloader with no transmission protocol and will write a maximum of 512-bytes (one page of FLASH memory) in a contiguous address space. The bootloader in this example performs system clock and port pin configurations in function *Main()*, and tests the port pin P1.4



for a logic '0'. If P1.4 is low, the program jumps to the *bootload()* function that configures the FLASH (erase and write enable) and UART. This function then sends a byte to signal the device is ready for download. The first two bytes received are loaded as a variable *NumBytes* that is the number of bytes to be downloaded. The data is then downloaded to FLASH memory, and lastly the bootloader vectors to the appropriate address to execute the downloaded code.

### Configuration

The example bootloader uses the UART and Timer 1 for 115.2 kbyte baud rate operation. The system clock is based on an external 18.432 MHz crystal. Configurations in this example are as follows:

1. **Configure the system clock source:** The OSCXCN and OSCICN special function registers (SFR) are set for using an external 18.432 MHz crystal as the system clock source. The program waits for the External Crystal Valid (XTLVLD) flag to be set before switching to the external clock source.
2. **Configure the I/O Port Pins:** This bootloader uses the UART, and assumes the SPI and SMBus are not used by the end application. XBR0 is configured to assign the UART signals to port pins. Peripheral *outputs* should be configured as push-pull by setting the corresponding bits in the PRT0CF register to '1'. The other port pins are left in their open-drain default setting. Lastly, the crossbar is enabled by setting XBARE (XBR2.6) to '1'.

In the *boot\_load()* function, the following are configured:

3. **Configure Timer 1 for baud rate generation:** For baud rate generation, the

CKCON, TH1, TMOD, TCON, and PCON registers must be set.

4. **Configure the UART:** Configure SCON for Mode 1 (8-bit, variable rate) and enable the UART.
5. **Configure FLASH prescale:** To use FLASH memory, the prescaler must be set based upon the system clock frequency by configuring the FLSCCL register.

This bootloader uses P1.4 as a bootload enable signal. Thus, the program tests P1.4 to test if it is a logic '0'. If P1.4 is sampled in the low state, the program jumps to execute the function *boot\_load()*. After the aforementioned configurations, *boot\_load()* executes the following:

1. **Erase FLASH sector that contains the address where the downloaded code will be stored:** FLASH erase is accomplished by setting the PSWE and PSEE bits in the PSCTL register. Once a byte is written to any address in the FLASH page, the entire page is erased. After the erase, the PSEE and PSWE bits are cleared to disable write and erase operations. The download address is defined as a constant *DWNLD\_SECTOR* at the beginning of the code.
2. **Indicate device is in bootloader mode and ready to receive data:** In this example, we transmit the byte 0x5A to the host to indicate that the bootloader is ready to receive data.
3. **Indicate number of bytes to receive for download:** In this example, the first two received bytes indicate the number of bytes that will be downloaded. Because we are using a maximum of one FLASH page, the number of bytes must be less than or equal to 512 bytes.
4. **Download the code to FLASH:** The PSWE bit is set to enable write operations



and the function uses a *for* loop to write the data to consecutive bytes in memory. Once the loop counts the number of bytes specified in the first two bytes (in the previous step), the download is complete and the function exits the *for* loop. After the download, clear the PSWE = '0' and FLSCL = 0x8F to disable write and erase operations.

5. **Execute downloaded code:** The last step is to execute the downloaded code. This example assumes the entry point of the downloaded code is located at *DWNLD\_SECTOR*, which also contains the first byte downloaded. In C code, a function pointer (*\*boot*) is used to redirect the program counter to the downloaded code.

The bootstrap loader example software is provided on the following page. Please note this is a simple example. There are many different techniques that may be employed to implement bootloaders in the C8051F0xx family of devices.



## Example Source Code

```

/*****
Copyright 2001, CYGNAL INTEGRATED PRODUCTS, INC.

FILE NAME   : bootloader.c
TARGET MCU  : C8051F0xx
DESCRIPTION : bootloader routine and test driver.
AUTHOR     : RS

NOTES:
(1) This program was written for the Keil C51 v6.01 compiler and linker.
*****/

// Compiler configuration

#include <c8051F000.h>                // Include C8051F000 register defs

#define LOGIC_LOW 0x00
#define LOGIC_HIGH 0x01
#define DWNLD_SECTOR 0x1000          // Starting address of download
#define timer_delay_const (-18432000/1000) // Value to load Timer 1 for 1 ms delay

// Function prototypes
void boot_load ( void );

// Variables
sbit BLE_PIN = P1^4;                // Port pin used as /BLE signal

////////////////////////////////////
// Main program code
////////////////////////////////////

void main (void)
{

                                // Disable watchdog timer

    WDTCN = 0xDE;
    WDTCN = 0xAD;

                                // Configure system clock source.
                                // Select external crystal (18.432MHz)
                                // as clock source and set freq control
                                // bits.

    OSCXCN = 0x66;

// Delay 1 ms prior to checking XTLVLD.
    CKCON = 0x10;                // T1 use sysclk undivided
    TCON = (TCON & 0x3F);        // Stop Timer 1 and clear TF1
    TMOD = ((TMOD & 0x0F) | 0x10); // Configure T1 in 16-bit mode
    TH1 = (timer_delay_const >> 8); // Load delay constant
    TL1 = timer_delay_const;

    TR1 = 1;                     // Start timer 1
    while (!TF1);                // Wait for 1 ms
    TR = 0;                       // Stop timer
    TF1 = 0;                      // Clear overflow flag
}

```



## AN012 - C8051F0xx Bootloader Considerations and Example

```
while (!(OSCXCN & 0x80));           // Wait for Crystal valid flag (oscilla-
OSCICN = 0x08;                     // -tor running and stable.
                                   // Switch to external clock.

// Configure the port I/O
XBR0 = 0x04;                       // Enable UART
XBR2 = 0x40;                       // Enable the XBar
PRT0CF = 0x01;                    // set P0.0(TX) to push-pull
if ( BLE_PIN == LOGIC_LOW )       // Check state of /BLE signal (P1.4)
{
    boot_load();                  // If /BLE is true, jump to bootload()
}

while(1);                          // Spin forever (if we don't get /BLE)
}

////////////////////////////////////
//
// boot_load()
//
// The boot_load() routine configures timer 1 for baud rate generation,
// erases the sector of Flash used to store the downloaded code and then
// sends back 0x5A to indicate ready to receive the download.
// It then waits for input to the UART. The first two bytes received are
// interpreted as the number of bytes to download to flash (0x00 - 0x200) sent MSB
// first. The following bytes are written to contiguous addresses in Flash starting
// at address DWNLD_SECTOR. Once the download is complete, a jump is
// performed to execute the code just downloaded whose entry point is assumed to be
// DWNLD_SECTOR. No error detection is performed on the downloaded code before being
// written or executed.
//
////////////////////////////////////
void boot_load ( void )
{
    void (*boot)( void);           // Function pointer used to jump to the
                                   // downloaded code.

    int i, NumBytes;
    char xdata *address;          // 'address' is declared as an XDATA
                                   // pointer in order to write to FLASH.

    address = DWNLD_SECTOR;       // Pointer to beginning of download.

    // Configure Timer1 for baud rate generation.
    TCON = (TCON & 0x3F);         // Stop Timer 1 and clear TF1
    CKCON|= 0x10;
    TH1 = (-10);
    TMOD = ((TMOD & 0x0F) | 0x20); // Place Timer1 in 8-bit autoreload mode
    TR = 1;                      // Start Timer1

    // Configure the UART (8-N-1)
    PCON = 0x80;                 // Set SMOD to double baud rate for
                                   // 115.2KHz
    SCON = 0x50;                 // Put UART in 8-N-1 variable rate mode
```



## AN012 - C8051F0xx Bootloader Considerations and Example

```

// (Mode 1) and enable UART.

// Set FLASH prescaler for 18.432MHz clock,
FLSCL = ((FLSCL & 0xF0) | 0x09);

// Erase FLASH sector that contains address DWNLD_SECTOR
PSCTL = 0x03; // enable writes (PSWE) and sector erase
// (PSEE)
*address = 0x00; // dummy write to sector to initiate
// erase
PSCTL = 0x00; // disable writes and sector erase

// Indicate we are in bootloader mode and ready to receive download.
SBUF = 0x5A; // transmit byte to indicate bootloader
// is ready.
while (!TI); // Wait to complete transmit
TI = 0; // clear TX complete indicator

// Next 2 characters read are number of bytes to download (0x00 to 0x200).
// First character read is MSB, next is LSB.
while (!RI); // Wait for receive
i = SBUF; // Store first (high) byte
RI = 0; // Clear RX complete indicator
while (!RI); // Wait for second byte
NumBytes = (i << 8) | SBUF; // Store low byte
RI = 0; // Clear RX complete indicator
NumBytes = (NumBytes <= 0x200) ? NumBytes : 0x200; // Restrict to 512 bytes max.

// Download the code to flash.
PSCTL = 0x01; // enable FLASH writes and redirect MOVX
// to XRAM.
for( i = 0; i < NumBytes; i++)
{
    while (!RI); // Wait to receive next byte
    *address++ = SBUF; // write to flash
    RI = 0; // Clear RX complete indicator
}
PSCTL = 0x00; // disable writes and erases and
// redirect MOVX to XRAM.
FLSCL |= 0x0F; // disable writes and erases

// Execute the code just downloaded.
boot = DWNLD_SECTOR; // put address of first byte in boot
(*boot)(); // vector to execute code starting at
// DWNLD_SECTOR.

// We should never get here, but just in case.
// All your base are belong to us.
return;
}
```