TB024

Downloading HEX Files to External FLASH Memory Using PIC17CXXX PICmicro® Microcontrollers

Author: *Rodger Richey*
Microchip Technology Inc.

INTRODUCTION

The PIC17CXXX devices have the capability to interface external FLASH memory into the 64K x 16 program memory space. Coupled with this feature is the ability to read and write to the entire program memory of the device. Using one of the standard serial interfaces on the PICmicro (USART, SPI, I^2C™), a complete hex file can be downloaded into the external FLASH memory by a bootloader program. The PIC17CXXX family consists of seven devices as shown in Table 1

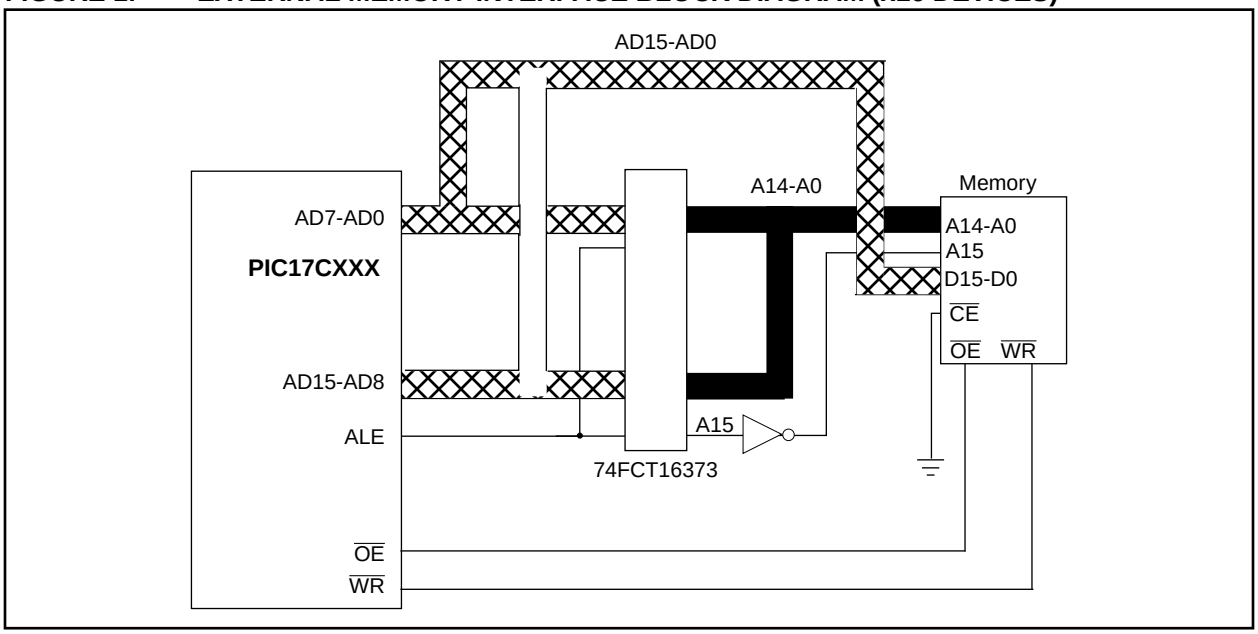
TABLE 1 FEATURES LIST

Features	PIC17C42A	PIC17C43	PIC17C44	PIC17C756A	PIC17C762	PIC17C766
Max Freq for Ops	33 MHz	33 MHz	33 MHz	33 MHz	33 MHz	33 MHz
Op Voltage Range	2.5V - 6.0V	2.5V - 6.0V	2.5V - 6.0V	3.0V - 5.5V	3.0V - 5.5V	3.0V - 5.5V
Prog Memory x16	2K	4K	8K	16K	8K	16K
Data Memory (bytes)	232	454	454	902	678	902
Hardware Multiplier	Yes	Yes	Yes	Yes	Yes	Yes
Timers	4	4	4	4	4	4
Capture Inputs	2	2	2	4	4	4
PWM Outputs	2	2	2	3	3	3
USART/SCI	1	1	1	2	2	2
A/D Channels	-	-	-	12	16	16
Power-on Reset	Yes	Yes	Yes	Yes	Yes	Yes
Brown-out Reset	-	-	-	Yes	Yes	Yes
ICSP	-	-	-	Yes	Yes	Yes
Watchdog Timer	Yes	Yes	Yes	Yes	Yes	Yes
Interrupt Sources	11	11	11	18	18	18
I/O pins	33	33	33	50	66	66

FLASH SELECTION

The first decision is what FLASH memory to use in the circuit. This document will focus on the Am29F100 from AMD. This device has a selectable memory/interface size: 128K x 8 or 64K x 16. The 16-bit interface is chosen because the PIC17CXXX devices have 16-bit wide program memory. The address line A15 may need to be inverted depending on the PICmicro internal OTP memory size and the FLASH memory selected. The AMD device needs to access address locations 2AAAh and 5555h for program and erase operations. For PICmicro microcontrollers with 8K or less program memory, no inversion is necessary, but is required for the 16K and larger devices. The address location 2AAAh in the FLASH memory is mapped on top of internal program memory, which takes precedence. Any access to 2AAAh will be to the internal OTP memory and not to the external FLASH memory. The inversion is transparent to the designer except that program or erase operations will use address locations AAAAh and D555h instead due to the inversion. The Technical Brief (TB027), Simplifying External Memory Connections of PIC17CXXX PICmicro mirocontrollers, covers the memory mapping and circuit connection considerations in more detail. Figure 1 shows a block diagram for the external memory connections.

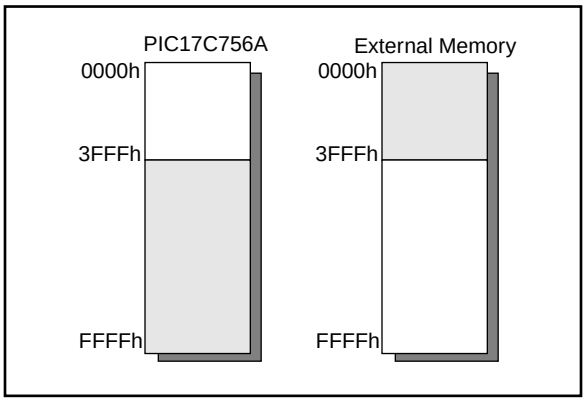
FIGURE 1: EXTERNAL MEMORY INTERFACE BLOCK DIAGRAM (x16 DEVICES)



MICROCONTROLLER CONFIGURATION

The microcontroller has several operating modes. The first being **Microcontroller** mode, which uses only the internal OTP program memory. In this mode all I/O pins function as I/O pins. The second mode is **Microprocessor** mode, which uses only external memory. In this mode, 19 of the I/O pins function as the external memory interface (3 for control, 16 for address/data). The final mode is **Extended Microcontroller** mode, which uses internal OTP program memory. The remainder of 64K is external to the device. This mode must be used to program the external FLASH memory and the bootloader routine must reside in the OTP memory. Refer to the PIC17C7XX data sheet (DS30289) or the PIC17C4X data sheet (DS30412) for more information about processor modes. Figure 2 shows the memory map configuration for extended microcontroller mode for the PIC17C756A.

FIGURE 2: PIC17C756A IN EXTENDED MICROCONTROLLER MODE



HEX FILE FORMAT

The HEX file to be programmed into program memory will be read into the microcontroller using one of its standard interface modules: USART, SPI, or I^2C. The formats supported by the Microchip development tools are the Intel Hex Format (INHX8M), Intel Split Hex Format (INHX8S), and the Intel Hex 32 Format (INHX32). The format required by the PIC17CXXX devices is the INHX32 due to the 64K of address space. Please refer to Appendix A in the MPASM User's Guide (DS33014) for more information about HEX file formats. The INHX32 format supports 32-bit addresses using a linear address record. The basic format of the INHX32 hex file is:

:BBAAAATTHHHH...HHHHCC

Each data record begins with a 9 character prefix and always ends with a 2 character checksum. All records begin with a '**:**' regardless of the format. The individual elements are described below.

- **BB** - is a two digit hexadecimal byte count representing the number of data bytes that will appear on the line. Divide this number by two to get the number of words per line.
- **AAAA** - is a four digit hexadecimal address representing the starting address of the data record. Format is high byte first followed by low byte. The address is doubled because this format only supports 8-bits (to find the real PICmicro address, simply divide the value **AAAA** by 2).
- **TT** - is a two digit record type that will be '00' for data records, '01' for end of file records and '04' for extended address record.
- **HHHH** - is a four digit hexadecimal data word. Format is low byte followed by high byte. There will be **BB**/2 data words following **TT.**
- **CC** - is a two digit hexadecimal checksum that is the two's complement of the sum of all the preceding bytes in the line record.

The HEX file is composed of ASCII characters 0 thorough 9 and A to F and the end of each line has a carriage return and linefeed. The downloader code in the PICmicro must convert the ASCII characters to binary numbers for use in programming.

PICmicro CODE

The code for the PIC17CXXX devices was written using the MPLAB-C17 C compiler. A demo version of the MPLAB-C17 C compiler is available off the Microchip website, *www.microchip.com*. This code uses USART2 on the PIC17C756A as the interface to the PC. In addition to USART2, two I/O pins are used to implement hardware handshaking with the PC host. Handshaking must be used because the program time of the FLASH memory prevents the PC from simply streaming the data down to the PICmicro microcontroller. The PICmicro microcontroller itself does not have enough RAM to buffer the incoming data while the FLASH is programming. Listing 1 shows the C code. Figure 3 shows a flowchart for the downloader code.

In this particular example, the hardware USART2 is used to download hex files from the PC host. Hardware handshaking is used to communicate with the PC. The function `DataRdyU2` properly asserts the handshake signals to the PC to receive one byte of data.

Two other functions not listed read in a byte (`Hex8in`) or a word (`Hex16in`) and return the binary value of the ASCII characters read. `Hex8in` reads two characters and converts them to an 8-bit value. `Hex16in` reads in 4 characters and converts them to binary. The format for `Hex16in` is high byte then low byte.

LISTING 1: HEX DOWNLOAD CODE WRITTEN FOR MPLAB™-C17

```
void EraseFlash(void)
{
    rom int *EFp;                        // FLASH requires following sequence to
    unsigned int dataEF;                 // initiate a write

    EFp = (rom int *)0xd555;             // Setup pointer to D555h
    *EFp = 0xaaaa;                       // Write data AAAAh
    EFp = (rom int *)0xaaaa;
    *EFp = 0x5555;
    EFp = (rom int *)0xd555;
    *EFp = 0x8080;
    EFp = (rom int *)0xd555;
    *EFp = 0xaaaa;
    EFp = (rom int *)0xaaaa;
    *EFp = 0x5555;
    EFp = (rom int *)0xd555;
    *EFp = 0x1010;
    EFp = (rom int *)0x8000;
    do                                    // Wait for FLASH to erase
    {
        dataEF = *EFp;
        if(dataEF & 0x0020)
       Nop();
        Nop();
    } while(!(dataEF&0x0080));
    return;
}
void ProgPreamble(void)
{                                         // FLASH requires a preamble before each
    rom int *PPp;                         // word that is programmed

    PPp = (rom int *)0xd555;              // Setup pointer to D555h
    *PPp = 0xaaaa;                        // Write data AAAAh
    PPp = (rom int *)0xaaaa;
    *PPp = 0x5555;
    PPp = (rom int *)0xd555;
    *PPp = 0xa0a0;
    return;
}
char DownloadHex(void)
{
    unsigned char ByteCount,RecType,Checksum,FChecksum;
    unsigned char DHi,Errors;
    unsigned char bytes;
    unsigned int AddrL,AddrH;
    unsigned int HexData;
    unsigned char temp;
    char str[5];
    rom int *DHp;

    EraseFlash();                          // Erase FLASH
    AddrH = 0;                             // Make high address word 0
    while(1)                               //
    {                                      // Wait for a :
        while(1)
        {
            while(!DataRdyU2());
            if(RCREG2 == ':')
                break;
        }
        Errors = 0;                        // Preset errors to 0
        ByteCount = Hex8in();              // Read in ByteCount and store in Checksum
        Checksum = ByteCount;
        AddrL = Hex16in();                 // Read in low word of address and add
        Checksum += (unsigned char)AddrL;  // to Checksum
```

```
        Checksum += ((unsigned char)(AddrL>>8));
        RecType = Hex8in();                                 // Read in RecordType and add to Checksum
        Checksum += RecType;
        if(RecType == 0x00)                                 // Data record
        {
            if(AddrH)                                       // Assemble 16-bit word address
                DHp = (rom int *)((AddrL>>1)+0x8000);       // from AddrH and AddrL
            else
                DHp = (rom int *)(AddrL>>1);
            bytes = ByteCount>>1;                           // get number of words in record
            for(DHi=0;DHi<bytes;DHi++)                      // loop for number of words
            {
                 temp = Hex8in();                           // Read in word of data and
                HexData = (unsigned int)Hex8in();           // add to Checksum
                Checksum += temp;
                Checksum += (unsigned char)HexData;
                 HexData <<= 8;
                 HexData |= (unsigned int)temp;
                 if(DHp > (rom int *)0x3fff)                // If address in not in OTP
                 {                                          // then program
                    while(1)
                    {
                    ProgPreamble();                         // Program preamble
                    *DHp = HexData;                         // write cycle
                         while((HexData&0x0080) != (*DHp&0x0080))  // Wait for program cycle
                         Nop();                             // to terminate
                        if(*DHp == HexData)                 // Make sure data was programmed
                        break;                              // If not try to reprogram
                     }
                }
                DHp++;                                      // Increment address pointer
            }
            FChecksum = Hex8in();                           // Read in LineChecksum
            if(FChecksum != (~Checksum + 1))                // Compare to calculated
                Errors = 1;                                 // If not equal, increment errors
        }
        else if(RecType == 0x04)                            // Extended address record
        {
            AddrH = Hex16in();                              // Read in 16-bits of address
            Checksum += (unsigned char)AddrH;               // and add to Checksum
            Checksum += ((unsigned char)(AddrH>>8));
            FChecksum = Hex8in();                           // Read in Line Checksum
            if(FChecksum != (~Checksum + 1))                // Compare to calculated
                Errors = 1;                                 // If not equal, increment errors
        }
        else if(RecType == 0x01)                            // End of file record
        {
            FChecksum = Hex8in();                           // Read in LineChecksum
            if(FChecksum != (~Checksum + 1))                // Compare to calculated
                Errors = 1;                                 // If not equal, increment errors
            break;
        }
    }
    return Errors;                                          // Return number of errors
}
```

FIGURE 3: FLOWCHART

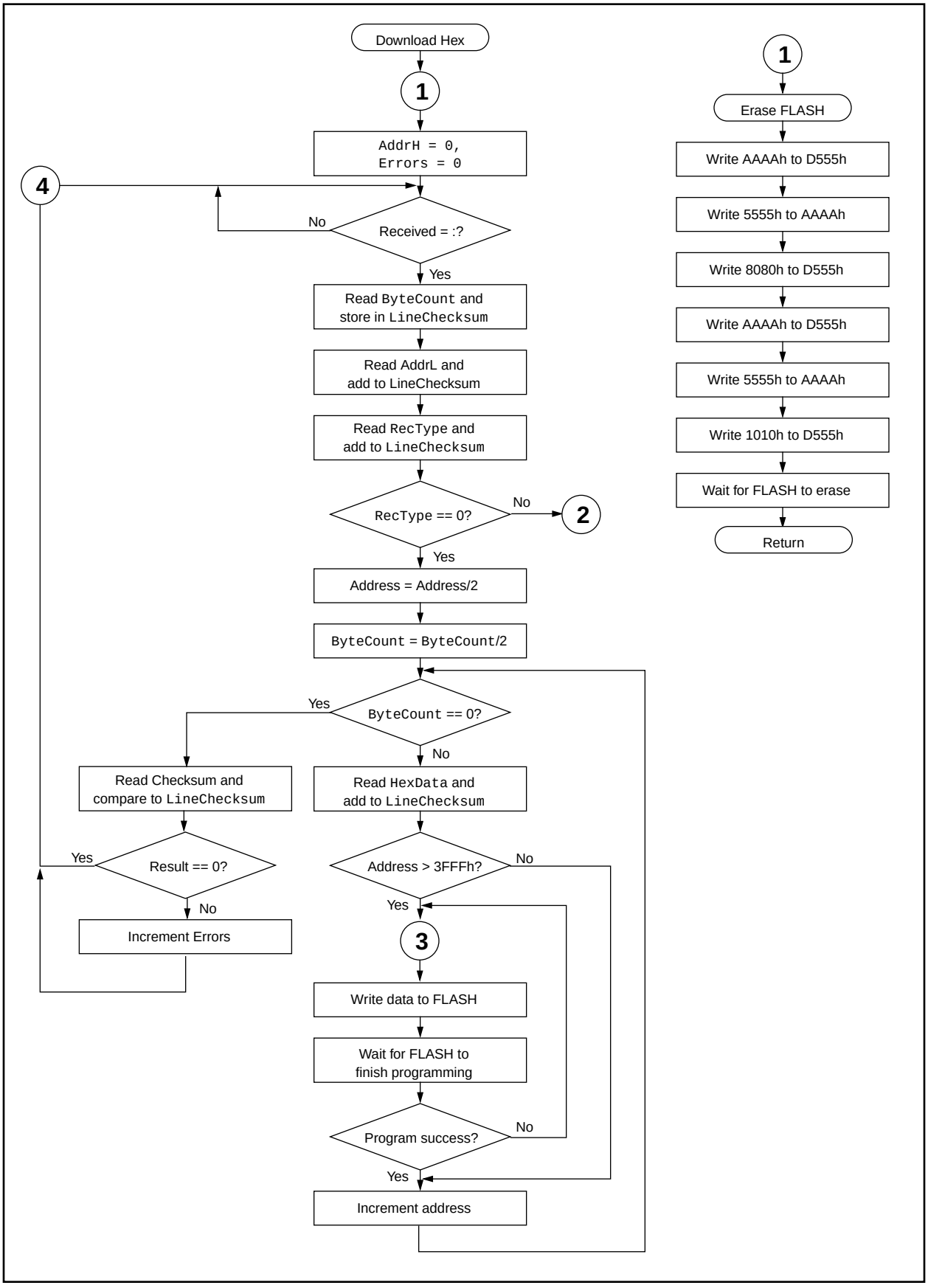


FIGURE 3 : FLOWCHART CONT'D