Application Note AN210

Running Two Applications on a TCP/IP Development Board

Introduction

To run a Download Manager (DLM) and a Downloaded Program (DLP) on the same target, or any other situation where two separate, co-resident programs are desired, a little cleverness is necessary. This application note describes a solution that can be used on a Rabbit based TCP/IP Development Board with the standard configuration of two 256K flash EPROMs and one 128K or 512K RAM.

The Dynamic C compiler and libraries usually handle the details of the Memory Management and Memory Interface Units (MMU and MIU) so that most Dynamic C users don't have to know them, but anyone writing their own download manager is advised to understand the MMU/MIU. For a detailed description of how memory addressing works on the Rabbit CPU, see Application Note 202, Rabbit Memory Management In a Nutshell.

This application note can serve as the starting point for the creation of a DLM that operates over a modem. However, the details of communication and transmission of the DLP are left to future application notes and the user. Here, the 2nd program is simply brought into the first program as constant data at compile time using Dynamic C's **`#ximport`** compiler directive, but the mechanics of loading a second co-resident program into the second flash (the one on CS2) and switching execution to it are well illustrated.

Files needed

This example was created and tested under Dynamic C 7.03, and should be run under that version. Subsequent versions of Dynamic C will be released with examples similar to this one. The following five files are needed and are in the zip file that accompanies this application note. Extracting the files to the Dynamic C root directory will automatically place the files in the correct locations:

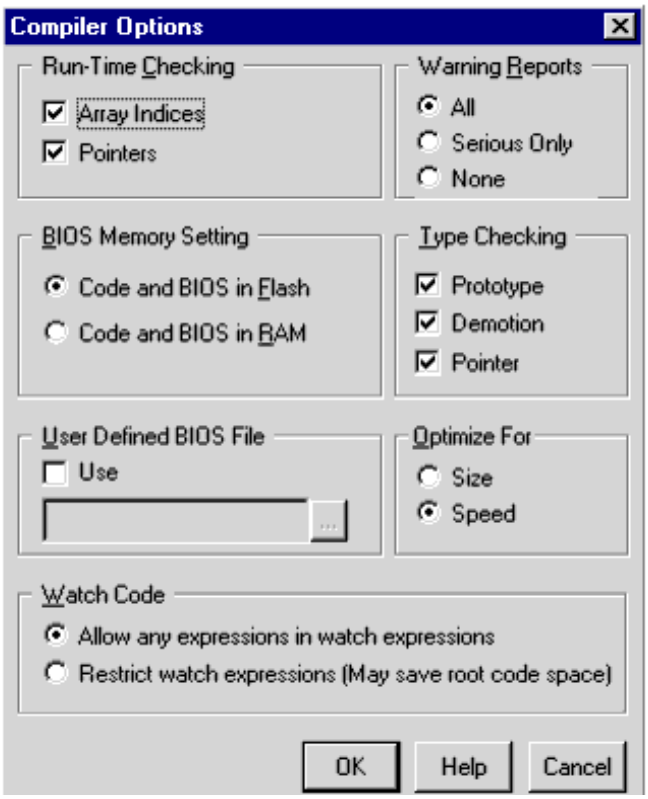
1. **\BIOS\DLMBIOS.C** - BIOS file to compile **DLM.C**
2. **\BIOS\DLPBIOS.C** - BIOS file to compile **DLP.C**
3. **LIB\BIOSLIB\FLASHWR.LIB** - This is the standard flash driver library, modified to move the hardcoded flash transfer buffer location for both **DLP.C** and **DLM.C**. This hardcoding will go away after DC version 7.03, so that **FLASHWR.LIB** will work without modification for analogous examples.
4. **\DLM.C** - Sample primary program to run the primary flash EPROM (the one on CS0).
5. **\DLP.C** - Sample secondary program, to be loaded to the second flash by **DLM.C**. This sample just sends the character 'Z' out continuously at 19200 baud over serial port C, which is labeled as TX/RX/GND on the header block of the TCP/IP development board.

How it works

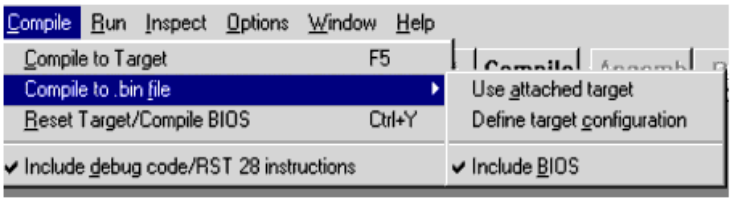
- The **#ximport "DLP.BIN"** directive brings in the program for the second flash into DLM as constant data.
- A function that writes to the MIU bank register that determines how memory quadrant 0 (physical addresses 0-1FFFF) is mapped is copied into an array. It is copied there because it not possible to change the mapping of the primary flash while code is being run from it.
- The secondary program is moved from the primary flash to the secondary flash using a function provided for writing to the second flash.
- The function that remaps quadrant 0 to the second flash is called in RAM.
- The function does not return, but instead calls physical address 0 which is now where DLPBIOS and DLP are.
- DLPBIOS runs the chip and board initialization, determines that the board is in run mode, and runs the DLP.

Compiling DLP.C

With the TCP/IP development board hooked up, use Dynamic C's **File** menu to open **DLP.C**. Open the **Compiler** options dialog by using the **Options | Compiler** menu.



Check **Use** under **User Defined BIOS File**, then click the button on the edit box and find and select **/BIOS/DLPBIOS.C** using the File Dialog box that appears. Now use the Compile menu to compile the **DLP.C** to a BIN file. Use the **Define target configuration** choice on the secondary menu. Make sure the **Include BIOS** option is checked. Choose the configuration for the TCP/IP development board you are using. If you are using a 512K RAM TCP/IP board and the that configuration does not appear on the target list, then click the **Specify Board Parameters** button and choose the appropriate configuration.



When compilation finishes there should be a 9 kilobyte file called **DLP.BIN** in the same directory as **DLP.C**.

Compiling DLM.C

Now open **DLM.C**. Change the user-defined BIOS to **DLMBIOS.C**. Use the **Compile** menu or F5 to compile **DLM.C** the target. Hit F9 to run it. If you look at the TCP/IP board's serial output on an oscilloscope or terminal, you should see a continuous stream of the letter Z being transmitted.

BIOS Changes for Compiling Two Co-resident Programs

DLMBIOS.C and **DLPBIOS.C** are modifications of the standard BIOS, **RABBITBIOS.C**. Two necessary changes were made in the BIOS source code to accomplish the goal of compiling two co-resident programs. The first change divides the RAM between **DLM.C** and **DLP.C** and the second change remaps memory quadrant 0. In addition, **DLPBIOS.C** defines the macro **DLPBIOS** so that the flash driver will correctly locate the flash transfer buffer in RAM.

Dividing the RAM

The following macros near the top of the BIOS source code configure the memory sizes. The macros on the right are internally defined by Dynamic C during the cold boot process to match the actual memory sizes in 4K byte units.

```
#define RAM_SIZE     _RAM_SIZE_
#define FLASH_SIZE   _FLASH_SIZE_
```

As of version 7.03, Dynamic C only detects the primary 256K flash on CS0 at start up, so **_FLASH_SIZE_** is set to 0x40 and doesn't need to be changed. We want to split the RAM between the two programs, so both **DLMBIOS.C** and **DLPBIOS.C** divide the RAM size in half:

```
#define RAM_SIZE    _RAM_SIZE_/2
```

A macro called **RAM_START**, which defines physical address where RAM starts in 4 kilobyte units, is normally set to 0x80 to start RAM at 80000h. For the DLP, it is set 0x90 to start RAM at 90000h: the DLP uses the top 64K of the RAM, and the DLM uses the bottom 64K.

```
#define RAM_START 0x90
```

Remapping Quadrant 0

The following two lines in **DLPBIOS.C** where the mapping of quadrant 0 is set up:

```
ld  a, FLASH_WSTATES | 0x00 | (MB0CR_INVRT_A18<<4) | (MB0CR_INVRT_A19<<5)
...
ld  a, FLASH_WSTATES | 0x00 | (MB0CR_INVRT_A18<<4) | (MB0CR_INVRT_A19<<5)
```

are changed to:

```
ld  a, FLASH_WSTATES | 0x02 | (MB0CR_INVRT_A18<<4) | (MB0CR_INVRT_A19<<5)
...
ld  a, FLASH_WSTATES | 0x02 | (MB0CR_INVRT_A18<<4) | (MB0CR_INVRT_A19<<5)
```

to change from CS0 to CS2.

Skipping Debug Mode

This change is not really necessary. Its purpose is to fool DLPBIOS into thinking the programming cable is not attached and run the DLP unconditionally rather enter debug mode so that DLP can start running without disconnecting the cable and recycling power.

This line of code in **DLPBIOS.C**:

```
jr  nz,RunMode
```

is changed to:

```
jp  RunMode
```

Code Listing of DLM.C

```
#ximport "dlp.bin" dlp
#define CS2 2
#define WE0 0


char  Chip_SW_Func [30];   // RAM space for chip switch function
void  CopyChipSWtoRAM();
void  SwitchChip(int CSOEWE);
root  int WriteFlash2(unsigned long flashDst, void* rootSrc, int len);
int   LoadProgram2ndFlash(unsigned long xfile);
//*****************************************************************
root main(){
    CopyChipSWtoRAM();        // Put flash switching function in RAM

    if(LoadProgram2ndFlash(dlp))
        printf("Write Flash failed with error code %d",retval);
    else
        SwitchChip(CS2|WE0);   // Map the 2nd flash to 00000 and go there
}
#asm

//*********************************************************
// This function never runs, a copy of it RAM is run
//  register hl contains the integer parameter


_SwitchFlashChip::
    ipset 3                    // turn off interrupts
    ld   a,(MB0CRShadow)       // get shadow reg
    and  0xf8                  // mask out CS/WE/OE bits
    or   L                     // set to new CS,WE,OE which is in L
    ioi  ld (MB0CR),a          // load MIU bank register
    lcall  0xf2:0xE000         // call physical addr. 000000
_EndSwitchFlash::
#endasm

//*****************************************************************
root SwitchChip(int CSOEWE){
#asm
      call Chip_SW_Func   // call RAM copy of _SwitchFlashChip
#endasm
}

//*****************************************************************
void  CopyChipSWtoRAM(){
    // put flash switching function in RAM
    memcpy(Chip_SW_Func, _SwitchFlashChip,
     ((int)((unsigned)_EndSwitchFlash - (unsigned)_SwitchFlashChip)+1));
}
```

```
//*********************************************************************
int LoadProgram2ndFlash(unsigned long xfile){
   // Returns 0 if successful
   // xfile is the value given by the #import directive


    char  sectorBuf   [0x200];
    long length, fileOffset,flashOffset;
    int retval,Size;

    fileOffset = xfile+4;    // offset in xmem where file data begins
    flashOffset = 0xc0000ul; // physical address off quadrant 3
    xmem2root(&length, xfile, sizeof(long)); // get the length of the DLP


    // write it to the 2nd flash one sector at a time

    while(length > 0) {
        // Size is sector size OR file length MOD sector size
        if (length >= _FlashInfo.sectorSize)
            Size = _FlashInfo.sectorSize;
        else
            Size = (int)length;
        xmem2root (sectorBuf, fileOffset, Size); // copy sector to RAM
        if(retval = WriteFlash2(flashOffset,  sectorBuf, Size))
            break;

        fileOffset  += Size;
        flashOffset += Size;
        length      -= Size;
    }
    return retval;
}

//*********************************************************************
root int WriteFlash2(unsigned long flashDst, void* rootSrc, int len) {}

// NOT SHOWN HERE, see zip file. Function to write second
// flash, returns 0 if successful
```