TECHNICAL NOTE

TN206

How Dynamic C Cold-Boots a Rabbit Target

This document describes how Dynamic C versions prior to DC 7.10 bootstrap a Rabbit CPU-based target. Dynamic C assumes that target controller boards using the Rabbit CPU have no pre-installed firmware. It takes advantage of the Rabbit's bootstrap mode, which allows memory and I/O writes to take place over the programming port.

The source code for initial and secondary loaders is provided (please download the accompanying ZIP file), along with instructions on how to compile them. These files are compatible with Dynamic C 7.05 only. Previous versions use a slower baud rate to load the secondary loader, and additional changes are expected in later versions.

This information is provided as an example of how the Rabbit's programming port can be used to bootstrap a Rabbit-based board. Simpler methods of loading software are possible, and will be discussed later.

Click here to download the sample code for this document.

Overview

On start up, Dynamic C uses the PC's DTR line on the serial port to assert the Rabbit RESET line to put the processor in the cold-boot mode. Refer to the Designer's Handbook for the Rabbit chip (e.g., the *Rabbit 2000 Microprocessor Designer's Handbook)* for further details on the cold-boot mode.

Next, Dynamic C uses a four-stage process to load a user program:

1. Load the initial loader, `COLDLOAD.BIN`, via "triplets" sent at 2400 bps from the PC to a target in bootstrap mode.

2. Run `COLDLOAD.BIN` to load the secondary loader, pilot BIOS, at 57,600 bps (19,200 bps before Dynamic C 7.05).

3. Run the pilot BIOS, `PILOT.BIN`, and load the BIOS (as Dynamic C compiles it).

4. Run the BIOS and load the user program at 115,200 bps (after Dynamic C compiles the user program to a file).

Compiling the Initial Loader

The source code for the cold loader is in COLDLOAD.C in the BOOTSTRAP.ZIP file.

To compile it, open it in Dynamic C and use the "Compile to .bin File" command on the Compile menu and uncheck the "Include BIOS" option. You cannot compile this file directly to the target and debug it, because you will overwrite the debug kernel and target communications while you are using them, and lose communication with the target. "Compile to .bin File" will create a file named COLDLOAD.BIN in the same directory where COLDLOAD.C is located.

COLDLOAD.C contains special compiler directives that locate the compiled code to 0x0000 on top of the BIOS code, which is much bigger than the COLDLOAD code. Dynamic C does not load the whole file to the target, it only loads until the triplet 80h,024h,080h is encountered. See Loading Algorithm Details below.

To try the new cold loader, copy COLDLOAD.BIN to the BIOS directory. Make a backup copy of the existing COLDLOAD.BIN to be safe. The new cold loader will be used when you restart Dynamic C or hit
<Ctrl+Y>.

Compiling the Secondary Loader

The source code for the secondary loader is in PILOT.C in the BOOTSTRAP.ZIP file.

To compile and test the secondary loader, do the same steps used for the initial loader. Be sure to make a backup copy of PILOT.BIN in case something goes wrong. This file is larger than needed also. The initial loader loads a fixed number of bytes for the pilot BIOS, then begins executing it.

Loading Algorithm Details

The programming cable keeps the SMODE pins pulled high so that on reset or power up, the Rabbit board is in bootstrap mode waiting for triplets. When Dynamic C starts, the following sequence of events takes place:

1. The serial port is opened with the DTR line high, closed, then reopened with the DTR line low at 2400 baud. This pulses the reset line on the target low (the programming cable inverts the DTR line) and prepares the PC to send triplets.

2. A group of db triplets (defined in the file COLDLOAD.BIN) consisting of 2 address bytes and a data byte are sent to the target. The first few bytes sent are sent to I/O addresses to set up the MMU and MIU and do system initialization. The MMU is set up so that RAM is mapped to 0x00000, and flash is mapped 0x80000.

3. The remaining triplets place a small program at memory location 0x00000. The last triplet sent is 0x80,0x24,0x80. This tells the CPU to ignore the SMODE pins and start running code at address 0x00000.

4. The PC now bumps the baud rate on the serial port to 57,600 bps (19,200 bps before Dynamic C 7.05), and the initial loader program does the following:
    - The crystal speed is measured to determine what divisor is needed to set a baud rate of 57,600 or 19,200. The divisor is stored at address 0x4002 for later use by the BIOS, and the programming port is set up to be a 57,600 (or 19,200 prior to Dynamic C 7.05) baud serial port.
    - The program enters a loop where it receives a fixed number of bytes that are the secondary loader program (`PILOT.BIN` sent by the PC). It writes those bytes to memory location 0x4100. After all of the bytes are received, the program execution jumps to 0x4100.

5. The secondary loader does a wraparound test to determine how much RAM is available, and reads the flash ID. This information is available to Dynamic C upon request.

6. The secondary loader now enters a finite state machine (FSM) that implements the Dynamic C/Target communications protocol. Dynamic C compiles the core of the regular BIOS and sends it to the target at address 0x00000 which is still mapped to RAM. Note that this requires the BIOS core to be 0x4000 or less in size. The source code for the BIOS is in `\BIOS\RABBITBIOS.C`

7. The FSM checks memory location 0x4001 (previously set to zero) after receiving each byte. When the compilation and loading to RAM of the BIOS is complete, Dynamic C signals the target to run the BIOS by sending a one to 0x4001.

8. The BIOS runs some initialization code, including setting up the serial port for 115200 bps, setting up serial interrupts and starting a new FSM.

9. The BIOS copies itself to flash at 0x80000, and switches the mapping of flash and RAM so that RAM is at 0x80000 and flash is at 0x00000. As soon as this remapping is done, the BIOS's execution of instructions begins happening in flash.

10. When the user compiles a program to the target, it is first written to a file, then the file is loaded to the target using the BIOS's FSM. The file is used as an intermediate step because fix-ups are done after the compilation and would cause extra wear on the flash if done straight to the flash.

11. When the program is fully loaded, Dynamic C sets a breakpoint at the beginning of `main()` and runs the program up to the breakpoint. Dynamic C is now is debug mode.

Alternatives for Loading Without Dynamic C

The BIOS code released with Dynamic C versions 6.50 and later comes with support for "cloning." Cloning is a way for an already programmed Rabbit board to copy its flash contents into another Rabbit board via a special cable connecting both programming ports. Cloning is described in the Designer's Handbook for the Rabbit chip and Technical Note 207, *Rabbit Cloning Board*.

The Rabbit Field Utility (`RFU.EXE`) is a 32-bit Windows application for performing field updates of programs using the algorithm described in this document. This program loads BIN files generated by Dynamic C into Rabbit-based boards.

The RabbitLink board is a robust solution for remote downloading over a network that uses the algorithm described in this document to load the target it is connected to.

A sample serial downloader for boards using two flashes is provided with Dynamic C starting with version 7.05.

Methods of loading programs from a PC or other device involving fewer steps can be implemented by users if they wish.

A one-stage way of loading an application to a Rabbit-based board would be to use the bootstrap mode to load the application program to RAM and run out of RAM. The effective speed of loading a program this way is 80 bytes per second since each byte requires 2 additional address bytes, and 2400 bps translates to about 240 bytes per second. It is not possible to write to flash this way unless the flash write requires no special write algorithm.