SMH Technologies
UNIVERSAL PRODUCTION
IN–SYSTEM PROGRAMMING

03/05/2011 Samuele Stefanoni

DC10357

# AN00149 From compiler to FlashRunner for Texas Instrument™ TMS320™ family devices

FlashRunner is a Universal In-System Programmer, which uses the principles of In-Circuit Programming to program Texas Instrument™ TMS320™ microcontrollers.

This Application Note assumes that you are familiar with both FlashRunner and the main features of the TMS320™ family. Full documentation about these topics is available in the FlashRunner user's manual and in device-specific datasheets.

## 1. Introduction

This Application Note illustrates how to produce a correct binary from the developing environment to be used with FlashRunner for In-Circuit Programming. The validity of this Application Note is limited to Texas Instrument™ TMS320™ family.

## 2. Code Composer Studio

Firmware for Texas Instrument™ devices has to be compiled with proprietary environment Code Composer Studio™. In this AN we will describe how to set up Code Composer Studio™ version 3.3. Projects are compiled in proprietary *.out* format which doesn't work with external programming tools.

Texas Instrument™ released a tool with Code Composer Studio™ environment which translates *.out* format to standard format. This tool is *hex2000* and can be found in "CCS_root"/C2000/cgtools/bin path. All the information regarding how to deal with *hex2000* can be found in *"TMS320C28x Assembly Language Tools v5.0.0"* .

## 3. Programming

TMS320™ family works in *big endian* way and each 32 bit address contains 16 bit word data, so data packet has to be in this format:

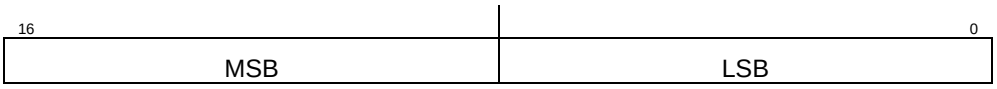


*Fig. 1: data frame format*

When starts programming session, FlashRunner reads each byte from the binary, from left to right, and make itself the inversion to form a word.

| Offset | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00000000 | 46 | 53 | 54 | 42 | 49 | 4E | 30 | 30 | 33 | 54 | 45 | 53 | 54 | 20 | 20 | 20 |
| 00000010 | 20 | 46 | 52 | 42 | 30 | 30 | 30 | 24 | 6A | DF | 0D | 9C | 3E | EF | 8B | 00 |

*Fig. 2: example binary*

This means that if you have the binary in Fig. 2 as input file for FlashRunner, and you want to program the target with this data, it will be sent to the device as:

**00h: 53 46**

**01h: 42 54**

**02h: 4E 49**

# 4. Setting *hex2000*

To be sure not programming wrong data you should bay attention on how you convert from *.out* to a format compatible with FlashRunner. As FlashRunner works with binary, the best way is to set up *hex2000* to produce directly a binary file.

First you can set up Code Composer Studio™ to launch directly *hex2000* every time you build the project by right clicking on the project name under the "Project" folder, and choosing "Build Options…" from the menu. Click on the "General" tab to add a new "Build Command" in "Final build steps:" and finally insert this line: *hex2000 ".\to_bin.cmd"*.

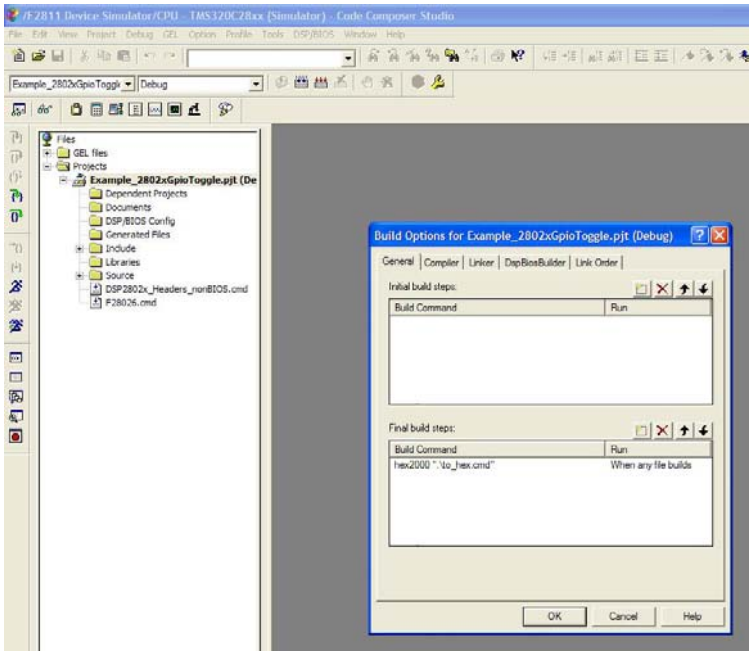


*Fig. 3: Code Composer settings*

This command will launch *hex2000* which it will take as options the content of *to_bin.cmd*, located in the root of your project. The content of the *to_bin.cmd* is the final step for a correct binary. Here the content:

```
".\debug\Example_2802xGpioToggle.out"
-map Example_2802xGpioToggle_map.map
-image
-fill FFFFh
-b
-memwidth 8
```

```
-romwidth 8


ROMS
{
  KER_2802:   org = 0x7E8000, len = 0x7FF0
                                  files =
{Example_2802xGpioToggle.bin}
}
```

The parameter –image will produce an exact image of what will be written in flash, taking care of the holes to be filled. –fill is used together with –image command and it will set up how to fill the holes. –b will translate *.out* to *.bin*. –memwidth 8 and -romwidth 8 will assure that the binary will be in byte format so that FlashRunner will take up first the LSB and then the MSB.

In the ROMS directive we'll set org and len parameters: the first takes in account at which address of the source should the binary start (we've choosen the beginning of the flash memory to decrease the size of the binary and speed up programming) and the second the amount of source data to put in the binary.

Last step is to convert the obtained binary in FRB format using Control Panel.

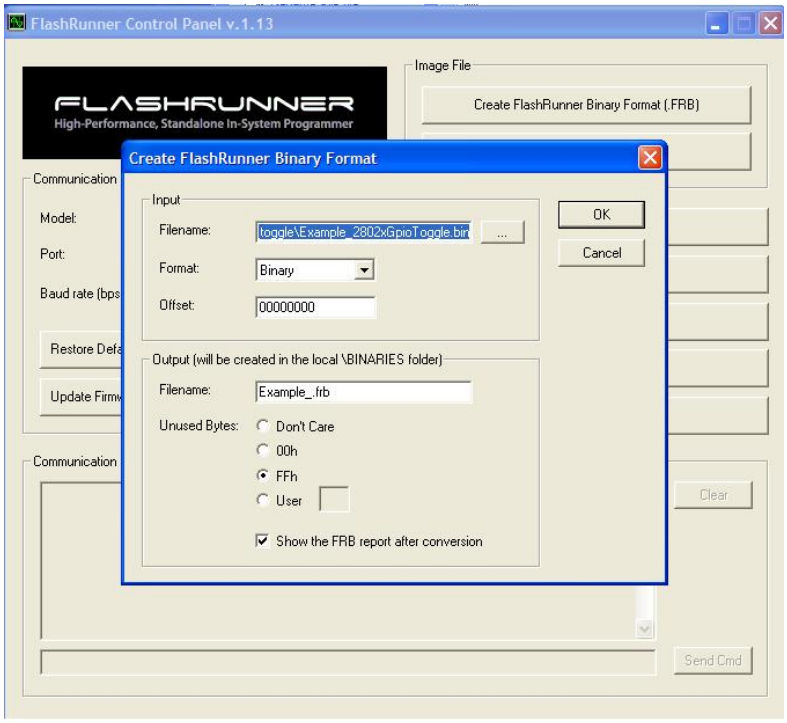


*Fig. 4: from .bin to .frb conversion*

# 5. References

SPRU513C: TMS320C28x Assembly Language Tools v5.0.0, *October 2007*