# Lab – Keystone I Boot Lab

Applications team – Ran Katzur

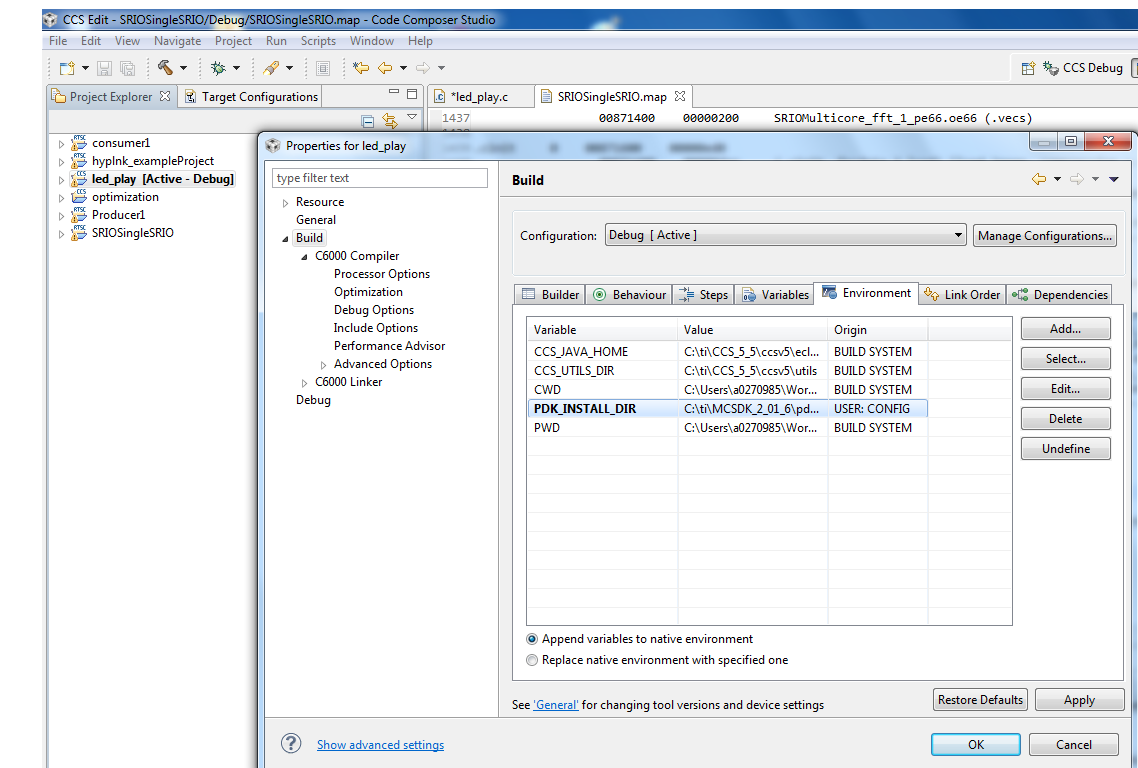
## Purpose

The purpose of this lab is to demonstrate all the steps that are needed to boot an allocation from SPI flash memory.

The source code for this Lab, a source code and a linker command file will be given by the instructor. Appendix A has a copy of source code and linker command files.

## Task 1: Observe the source file and build the application. Verify that it works correctly

1. Open CCS Editor and import the led\_play project. The location will be provided by the instructor
2. Right click on the project and Open Project properties. Under Build or C/C++ Build (may vary based on version of CCS), set the PDK\_INSTALL\_DIR variable in the Environment.



1. Right click on the project in the Editor view and Rebuild the project in your CCS environment.
2. Load and run the led\_play.out file on core 0 of the EVMC6678 to ensure that the rebuilt example runs on your EVM. Observe the LED blinks.

### Task 2: Building the btbl file – boot table format

The RBL expects the image flashed on the SPI flash to be in Boot Table Format. The led\_play example application Code has to be first converted into a Boot Table Format, using the hex6x utility present in CCS installation folder. (…\ccsv5\tools\compiler\c6000\_7.4.2\bin) (Or a different version of the compiler). The hex6x utility expects an rmd file in which you provide path to the application binary and a format in which the boot table is expected. The documentation for hex6x utility is provided in the TMS320C6000 Assembly Language Tools documentation that is part of the compiler documentation. The hex6x utility reads the sections in the application binary and creates a flat binary in boot Table format that allows the ROM to interpret and load the sections of the application binary. The RMD file contains, few of the following information:-

1. The Application.out file that has to be flashed.
2. –a for the output hex format in ASCII
3. –e the entry point for the address, i.e. \_c\_init00
4. Output file that contains the application.out in boottable format.
5. Memory sections with the MEM and ROW WIDTH
6. Create a new directory c:\temp
7. Copy the out file from the project to the temp directory that you just created. Note, you can copy the out file from the debug directory of the project
8. Copy hex6x from the bin directory (…\ccsv5\tools\compiler\c6000\_7.4.2\bin) to the temp directory
9. Open a cmd window and cd it to the temp directory
10. Create the rmd file led\_play.rmd using notepad or any other editor as follows:

***led\_play.out***

***-a***

***-boot***

***-e \_c\_int00***

***ROMS***

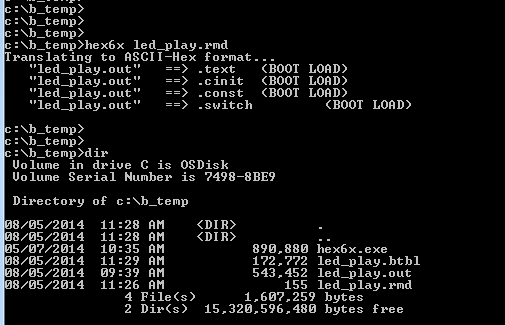
***{***

***ROM1: org = 0x0C000000, length = 0x100000, memwidth = 32, romwidth = 32***

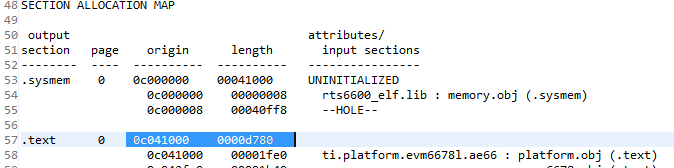
***files = {led\_play.btbl}***

***}***

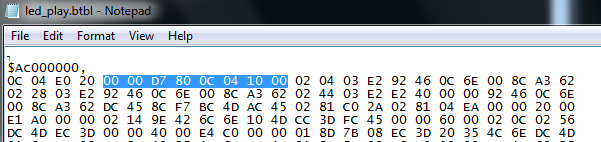
1. Run hex6x with led\_play.rmd “hex6x led\_play.rmd”
2. The following is a screen shot of the hex6x run:



1. Do dir and notice that the file led\_play.btbl was generated
2. Note that if you look at the led\_play.map map file, and find the text section, you can see the definition of the section and the length in the btbl file. From the map file:



And from the file led\_play.btbl:



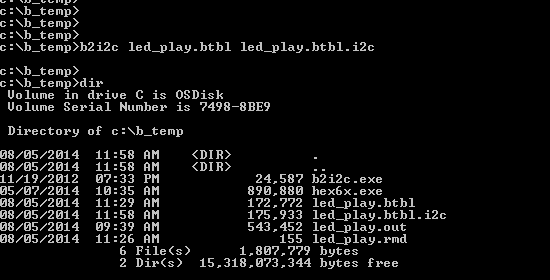
### Task 3: Convert to i2c/SPI format

From the generated output in previous step which is in the boot table format convert it into the i2c/spi format by passing through the b2i2c.

The byte-aligned boot table is then divided into 0x80 byte blocks and appended with length and checksum to adhere to the format required by the RBL, this is generated by passing through the b2i2c utility.

The b2i2c utility is part of the MCSDK installation and present in the following folder. mcsdk\_2\_01\_XX\_YY\tools\boot\_loader\ibl\src\util\btoccs

1. Copy the b2i2c.exe utility from the release (directory …\MCSDK\_2\_01\_XX\mcsdk\_2\_01\_XX\_YY\tools\boot\_loader\ibl\src\util\btoccs into the temp directory
2. Run b2i2c, specify the input and output file name “b2i2c led\_play.btbl led\_play.btbl.i2c”
3. The screen shot of the run is given below
4. Do dir and see that the i2c format file led\_play.btbl.i2c was generated



### Task 4: Convert to CCS downloaded format

Next the i2c formatted file need to be converted into CCS acceptable .dat format using b2ccs utility present in the mscdk\tools\boot\_loader\ibl\src\util\btoccs.

1. Copy the b2ccs.exe utility from the release (directory …\MCSDK\_2\_01\_XX\mcsdk\_2\_01\_XX\_YY\tools\boot\_loader\ibl\src\util\btoccs into the temp directory
2. Run b2ccs, specify the input and output file name “b2ccs led\_play.btbl.i2c led\_play.i2c.ccs”
3. The screen shot of the run is given below
4. Do dir and see that the i2c format file led\_play.i2c.ccs was generated



### Task 5: Adding Boot parameter Table

An updated boot parameter table is read from the SPI before the actual boot starts. To combine together the boot parameter table and the boot table in the ccs format romparse.exe is used. A \*.map file contains the name of the boot table and the values for the boot parameter table. The following shows a standard boot parameter map file:

section {

boot\_mode = 50

param\_index = 0

options = 1

core\_freq\_mhz = 1000

exe\_file = "led\_play.i2c.ccs"

next\_dev\_addr\_ext = 0x0

sw\_pll\_prediv = 5

sw\_pll\_mult = 32

sw\_pll\_postdiv = 2

sw\_pll\_flags = 1

addr\_width = 24

n\_pins = 4

csel = 0

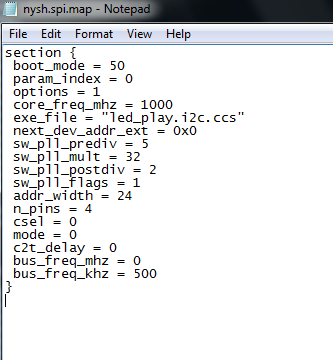
mode = 0

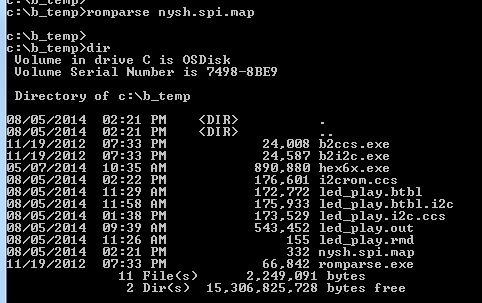
c2t\_delay = 0

bus\_freq\_mhz = 0

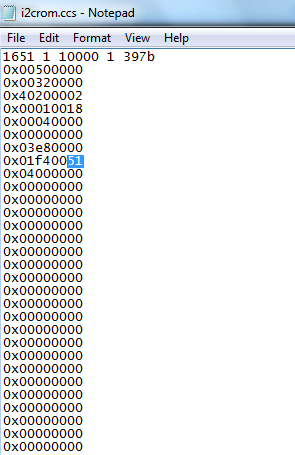
bus\_freq\_khz = 500

}

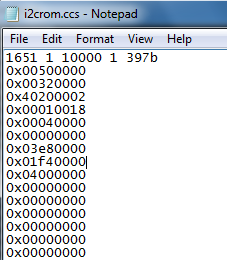
1. Copy romparse.exe utility from \MCSDK\_2\_XX\_YY\mcsdk\_2\_01\_XX\_YY\tools\boot\_loader\ibl\src\util\romparse to the temp directory
2. Create a map file. You can copy and paste the above file into nysh.spi.map (nysh stands for Keystone I first two families, Nyquist and Shannon). A screen shot is given below
3. Run romparse with the map file as a parameter “romparse nysh.spi.map”
4. Note that the program romparse hardcoded the name of the output file (i2crom.ccs) and the i2c address into the parameter table that is appended to the boot table.
5. The screen shot of the run is given below
6. Do dir and see that the i2c format file i2crom.ccs was generated



1. The program romparse was developed to work with EEPROM connected via i2c. SPI boot protocol is the same as i2c, except that EEPROM is connected to page 0x51 of the i2c while SPI boot starts from 0. The 0x51 is hard-written into the output file i2crom.ccs. The user must change this value into 00. This is done by the following:
   1. Open the file i2crom.ccs with an editor (I use notepad in the screen shots below)



* 1. Change the 51 to 0. The same file will be looked like the following:

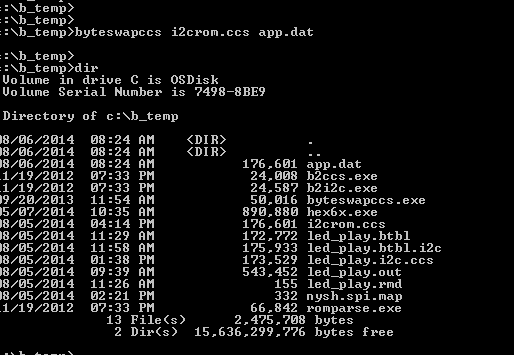


* 1. Save the modified file

### Task 5: Big Endian format

The program led\_play was built as little endian. The EVM is running as little endian as well, but the RBL always works as big endian. The program byteswapccs swaps the bytes for big endian RBL. The source for byteswapccs.c is given in the Appendix. An executable was built and will be given to the students.

1. Create an EXE from byteswapccs.c (This is already done)
2. Run byteswapccs.exe with the input file as the i2crom.ccs generated at previous step and output as the app.dat that will be flashed to the NOR
3. The screen shot of the run is given below
4. Do dir and see that the big endian file app.dat was generated



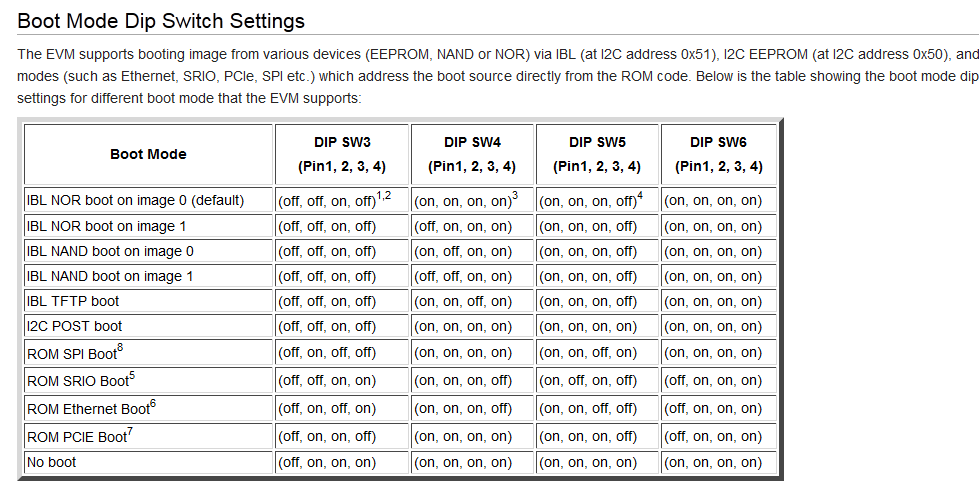
### Task 5: Flash the EVM SPI flash

1. Configuring the EVM for CCS NOR flashing

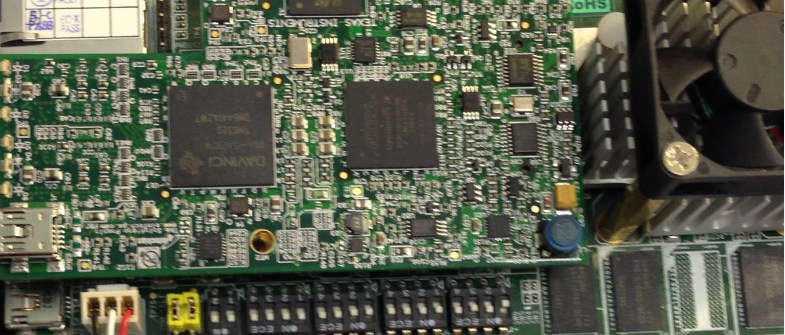
Flashing the EVM SPI flash is done using CCS connected to the EVM. The EVM is in no-boot (or sleep) mode. The EVM mode is determined by the setting of four switches on the board, Sw3, SW4, SW5 and SW6. The switches control the following:

* SW3 DSP Boot mode, DSP Configuration
* SW4 DSP boot Configuration
* SW5 DSP boot Configuration
* SW6 DSP boot Configuration, PLL setting, PCIe mode Selection

The following table is taken from <http://processors.wiki.ti.com/index.php/TMDXEVM6678L_EVM_Hardware_Setup> describes the various mode setting of the EVM:



The location of the switched on the EVM for the non-boot case is given by the following pictures.



And a close-up of the switches



1. The CCS NOR writer is part of the release at location MCSDK\_2\_XX\_YY\mcsdk\_2\_01\_XX\_YY\tools\writer\nor\evmc6678l . The README.txt file (in the \nor\docs directory) gives instructions how to flash the NOR memory. The way the flash data was developed, some minor modifications to the README.txt file are needed. The following is the updated instructions:

Steps to program the NOR:

1. Be sure to set the boot mode dip switch to no boot/EMIF16 boot mode on the EVM.

2. Copy app.dat file to writer\nor\evmc66xxl\bin directory

3. Change the file\_name to app.dat and start\_addr to 0 in writer\nor\evmc66xxl\bin\norwriter\_input.txt if necessary. See the screen shot below for the norwrite\_input.txt file

4. Open CCSv5 and launch the evmc66xx emulator target configuration and connect to core 0.

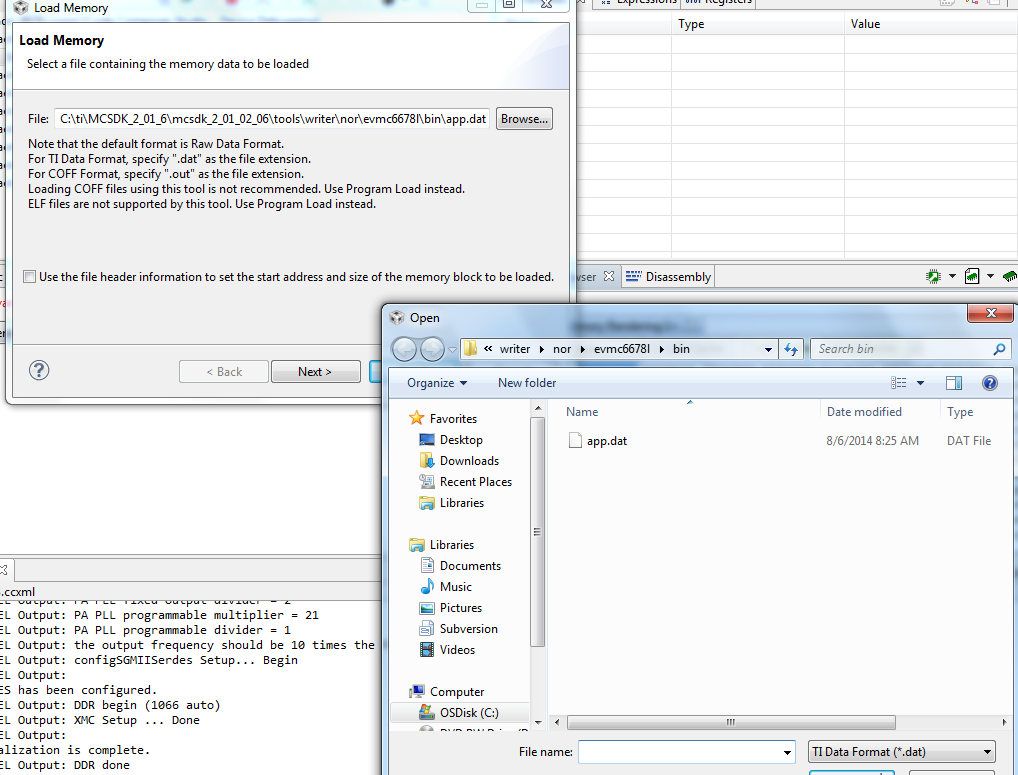
5. Load the program writer\nor\evmc66xxl\bin\norwriter\_evm66xxl.out to CCS, be sure evmc66xxl.gel is used in CCS and DDR is initialized. Ignore the red comment that says that it does not find the main() C source.

6. Open the Memory view (in CCSv5, view->Memory Browser), and view the memory address 0x80000000.

7. Load app.dat to 0x80000000:

\* In CCSv5, right click mouse in memory window, select "load memory".

\* Browse and select writer\nor\evmc66xxl\bin\app.dat (TI data format), click "next" . See the following screen shot



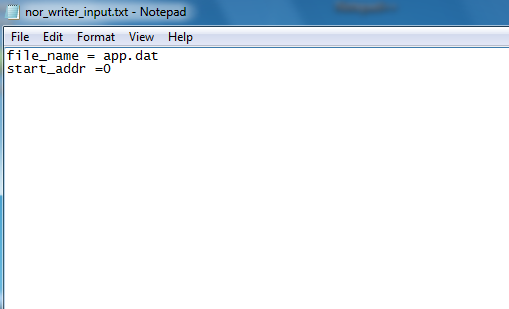
\* Set the Start Address to "0x80000000", enter the size of the file. If you check the line “use the file header information to set the start address and size of memory block to be loaded, it will load the file size automatically, see the picture below. Click "finish"



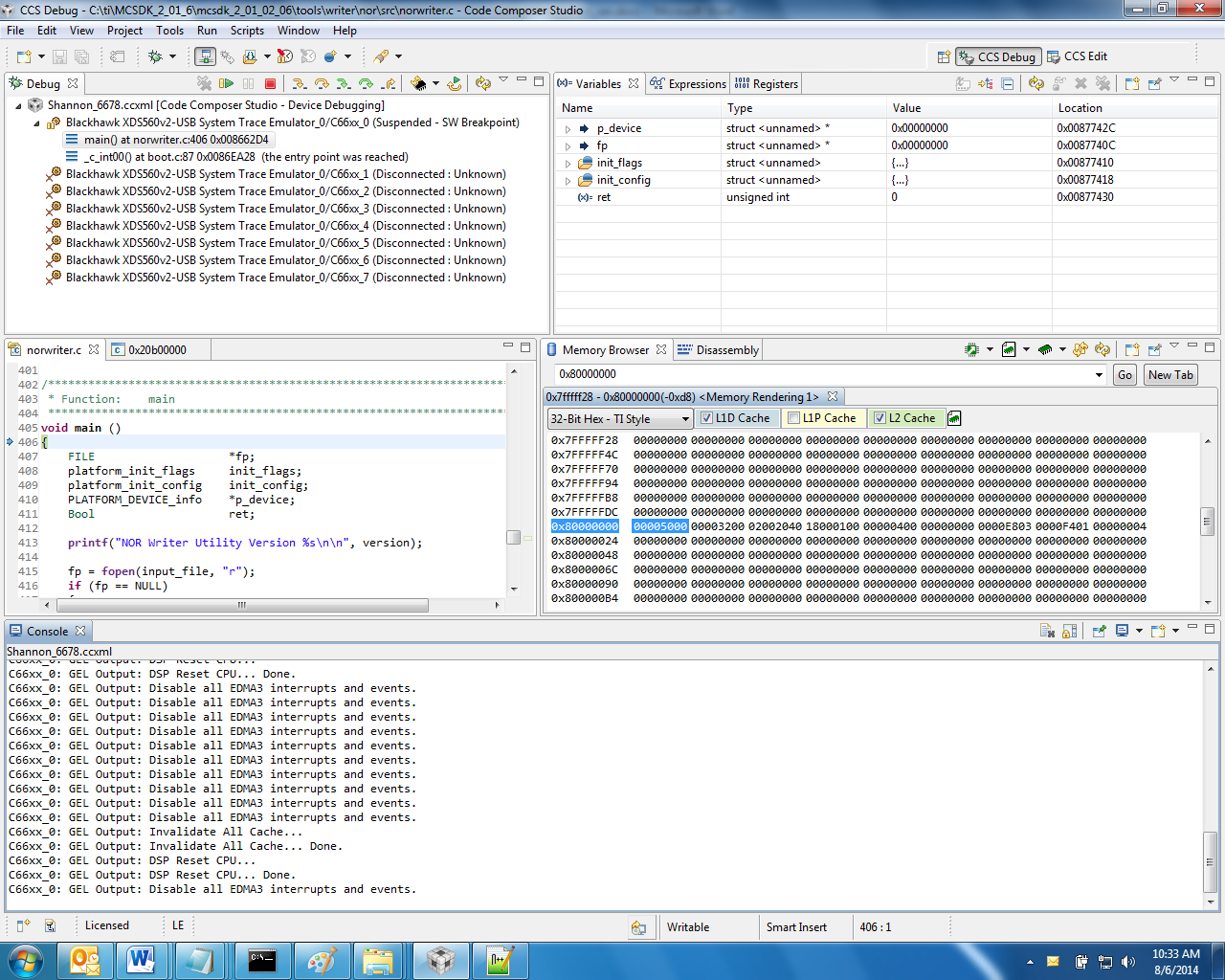
8. After the app.dat file is loaded into the memory, run the program (in CCSv5, press F8), it will start to program the NOR.

9. When programming is completed, the console will print "NOR programming completed successfully", if there is any error, the console will show the error message.

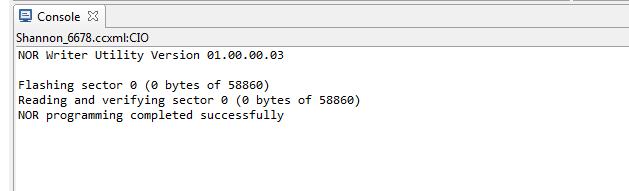
1. The following screen shots shows the nor\_writer\_input.txt file



1. The following screen shoots shows the CCS after loading the memory and the norwriter program

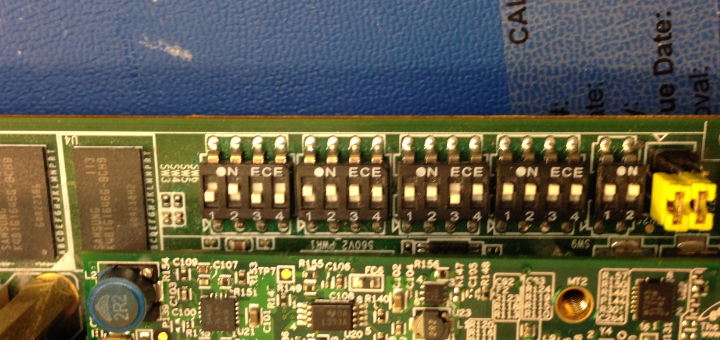


1. The next screen shoot shows the console after running the norwriter program



### Task 6: Boot from NOR SPI

1. Power off the EVM, change the EVM switched according to the Boot Mode Dip Switch Setting from above - ROM SPI BOOT off, on, off, off on,on,on,on on,on,off,on on,on,on,on . A screen shot of the dip switch is given below



1. Power up the EVM. The LED will blink. The last screen shot shows the blinking LED



# Appendix A

/\*

\* led\_play.c

\*

\* Created on: Aug 1, 2014

\* Author: a0272049

\*/

**#include** <cerrno>

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <string.h>

**#include** "ti\platform\platform.h"

**#include** "ti\platform\resource\_mgr.h"

/\* OSAL functions for Platform Library \*/

uint8\_t \***Osal\_platformMalloc** (uint32\_t num\_bytes, uint32\_t alignment)

{

**return** **malloc**(num\_bytes);

}

**void** **Osal\_platformFree** (uint8\_t \*dataPtr, uint32\_t num\_bytes)

{

/\* Free up the memory \*/

**if** (dataPtr)

{

**free**(dataPtr);

}

}

**void** **Osal\_platformSpiCsEnter**(**void**)

{

/\* Get the hardware semaphore.

\*

\* Acquire Multi core CPPI synchronization lock

\*/

**while** ((CSL\_semAcquireDirect (PLATFORM\_SPI\_HW\_SEM)) == 0);

**return**;

}

**void** **Osal\_platformSpiCsExit** (**void**)

{

/\* Release the hardware semaphore

\*

\* Release multi-core lock.

\*/

CSL\_semReleaseSemaphore (PLATFORM\_SPI\_HW\_SEM);

**return**;

}

**void** **main**(**void**) {

platform\_init\_flags init\_flags;

platform\_init\_config init\_config;

platform\_info p\_info;

uint32\_t led\_no = 0;

**char** message[] = "\r\nHello World.....\r\n";

uint32\_t length = **strlen**((**char** \*)message);

uint32\_t i;

/\* Initialize platform with default values \*/

**memset**(&init\_flags, 0x01, **sizeof**(platform\_init\_flags));

**memset**(&init\_config, 0, **sizeof**(platform\_init\_config));

**if** (platform\_init(&init\_flags, &init\_config) != Platform\_EOK) {

**return**;

}

platform\_uart\_init();

platform\_uart\_set\_baudrate(115200);

platform\_get\_info(&p\_info);

/\* Write to the UART \*/

**for** (i = 0; i < length; i++) {

**if** (platform\_uart\_write(message[i]) != Platform\_EOK) {

**return**;

}

}

/\* Play forever \*/

**while**(1) {

platform\_led(led\_no, PLATFORM\_LED\_ON, PLATFORM\_USER\_LED\_CLASS);

platform\_delay(30000);

platform\_led(led\_no, PLATFORM\_LED\_OFF, PLATFORM\_USER\_LED\_CLASS);

led\_no = (++led\_no) % p\_info.led[PLATFORM\_USER\_LED\_CLASS].count;

}

}

/\*led\_play

Created on: Aug 1, 2014

Author: a0272049

\*/

-c

-heap 0x41000

-stack 0xa000

/\* Memory Map \*/

MEMORY

{

L1PSRAM (RWX) : org = 0x0E00000, len = 0x7FFF

L1DSRAM (RWX) : org = 0x0F00000, len = 0x7FFF

L2SRAM (RWX) : org = 0x0800000, len = 0x080000

MSMCSRAM (RWX) : org = 0xc000000, len = 0x200000

DDR3 (RWX) : org = 0x80000000,len = 0x10000000

}

SECTIONS

{

.csl\_vect > MSMCSRAM

.text > MSMCSRAM

GROUP (NEAR\_DP)

{

.neardata

.rodata

.bss

} load > MSMCSRAM

.stack > MSMCSRAM

.cinit > MSMCSRAM

.cio > MSMCSRAM

.const > MSMCSRAM

.data > MSMCSRAM

.switch > MSMCSRAM

.sysmem > MSMCSRAM

.far > MSMCSRAM

.testMem > MSMCSRAM

.fardata > MSMCSRAM

platform\_lib > MSMCSRAM

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* FILE NAME: byteswapccs.c

\*

\* DESCRIPTION: A CCS file is read in, the data is byte swapped, and a CCS file is written out

\*

\* usage: byteswapccs infile outfile

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <malloc.h>

int main (int argc, char \*argv[])

{

FILE \*fin, \*fout;

unsigned int v, b0, b1, b2, b3;

int a, b, c, d, n;

int i;

char iline[132];

if (argc != 3) {

fprintf (stderr, "usage: %s infile outfile\n", argv[0]);

return (-1);

}

fin = fopen (argv[1], "r");

if (fin == NULL) {

fprintf (stderr, "%s: Could not open input file %s\n", argv[1]);

return (-1);

}

printf (stderr, "%s: input file %s\n", argv[1]);

fout = fopen (argv[2], "w");

if (fout == NULL) {

fprintf (stderr, "%s: Could not open output file %s\n", argv[2]);

fclose (fin);

return (-1);

}

printf (stderr, "%s: output file %s\n", argv[2]);

/\* Read the CCS data file header, write it out unchanged \*/

fgets (iline, 132, fin);

sscanf (iline, "%x %x %x %x %x", &a, &b, &c, &d, &n);

fputs (iline, fout);

printf ( " header -> %x %x %x %x %x", &a, &b, &c, &d, &n);

for (i = 0; i < n; i++) {

fgets (iline, 132, fin);

sscanf (&iline[2], "%x", &v);

b0 = (v >> 24) & 0xff;

b1 = (v >> 16) & 0xff;

b2 = (v >> 8) & 0xff;

b3 = (v >> 0) & 0xff;

v = (b3 << 24) | (b2 << 16) | (b1 <<8) | b0;

fprintf (fout, "0x%08x\n", v);

printf (fout, "0x%08x\n", v);

}

fclose (fout);

fclose (fin);

return (0);

}