

# Configuring DDR in U-Boot using QCVS

## 1. Introduction

This document describes how to configure the double data rate (DDR) memory in U-Boot, running on the NXP QorIQ platform, using the DDR tool included in QorIQ Configuration and Validation Suite (QCVS).

This document explains:

- Purpose of the QCVS DDR tool
- How to get optimized DDR initialization code for a custom board
- How to port U-Boot to the custom board

### Contents

1. Introduction.....	1
2. Preliminary background.....	2
3. Porting U-Boot to a custom board .....	2
4. Programming U-Boot using flash programmer	7



## 2. Preliminary background

The QCVS DDR tool provides a graphical user interface (GUI) that helps in creating and validating DDR configurations and getting an optimized DDR initialization code as output.

The DDR tool has two components, DDR configuration tool and DDR validation tool. The DDR configuration tool allows you to create a configuration for the DDR component, and the DDR validation tool allows you to validate the DDR configuration using various validation scenarios.

For more information about the QCVS DDR tool, see [QCVS DDR Tool User Guide](#).

## 3. Porting U-Boot to a custom board

Porting U-Boot to a custom board involves the following major actions:

- [Add new custom board definition to U-Boot](#)
- [Get optimized DDR initialization code for custom board](#)
- [Add DDR initialization code to U-Boot](#)
- [Rebuild U-Boot](#)

### 3.1. Add new custom board definition to U-Boot

---

**NOTE** Steps in this section are based on porting U-Boot to a custom board for the LS2080ARDB board.

---

To add new custom board definition to U-Boot, you need:

- U-Boot package for an NXP reference design board
- Custom board U-Boot needs to be ported to

Perform the following steps to port U-Boot to a custom board using an NXP SoC:

1. Create a new board family and board name under the `/board` directory from the U-Boot source layout:

```
mkdir -p /board/<custom_board_family>/<custom_board_name>
```

2. Copy the files and directories of a similar board to the new directory you just created:

```
cp /board/freescale/ls2080ardb  
/board/<custom_board_family>/<custom_board_name>
```

3. Edit the `/board/<custom_board_family>/<custom_board_name>/Kconfig` file with the custom board information:

```

if TARGET_<CUSTOM_BOARD_NAME>

config SYS_BOARD
    default "<custom_board_name>"

config SYS_VENDOR
    default "<custom_board_family>"

config SYS_SOC
    default "fsl-lsch3"

config SYS_CONFIG_NAME
    default "<custom_board_name>"

endif

```

4. Add information about the custom board in the /arch/arm/Kconfig file from U-Boot sources:

```

config TARGET_<CUSTOM_BOARD_NAME>
    bool "Support <custom_board_name>"
    select ARM64
    select ARMV8_MULTIENTRY
    select SUPPORT_SPL
    help
        Support for Custom <custom_board_name> platform.

source "board/<custom_board_family>/<custom_board_name>/Kconfig"

```

5. Create a new configuration file, /configs/<custom\_board\_name>\_defconfig:

```

CONFIG_ARM=y
CONFIG_TARGET_<CUSTOM_BOARD_NAME>=y
# CONFIG_SYS_MALLOC_F is not set
CONFIG_DM_SPI=y
CONFIG_DM_SPI_FLASH=y
CONFIG_DEFAULT_DEVICE_TREE="<CUSTOM_DEVICE_TREE>"
CONFIG_SYS_EXTRA_OPTIONS="SYS_FSL_DDR4, LS2080A"
# CONFIG_CMD_SETEXPR is not set
CONFIG_OF_CONTROL=y
CONFIG_DM=y
CONFIG_NETDEVICES=y
CONFIG_E1000=y
CONFIG_FSL_DSPI=y

```

### 3.2. Get optimized DDR initialization code for custom board

After adding the new board definition to U-Boot sources, the next step is to get a valid DDR configuration. This may be tricky especially in the following scenarios:

- The new board does not feature the same DDR implementation as the reference design board, though it has the same DDR modules
- The new board does not have DDR DIMM modules, rather it has discrete DDR implementation

The DDR controller can work with a wide variety of DDR modules. Therefore, the DDR initialization code needs to deal with all possible variations for DDR modules, in terms of number of ranks, size per rank, and a number of timing settings.

Getting the optimized DDR initialization code involves the following complex tasks:

- Program DDR
- Validate DDR programming
- Get a working and optimized DDR configuration, that is, DDR initialization code

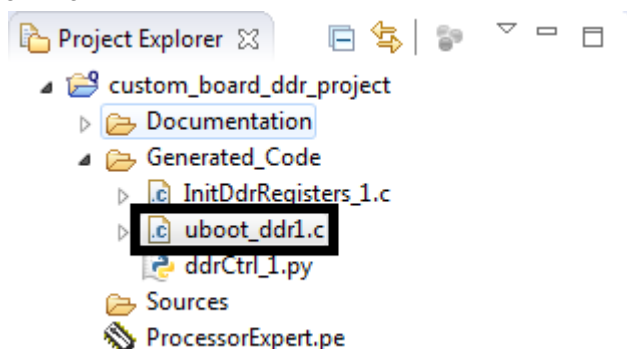
The QCVS DDR tool helps you perform the above complex tasks with less manual effort using an easy-to-use GUI. See [QCVS DDR Tool User Guide](#) for more information on how to perform these steps.

### 3.3. Add DDR initialization code to U-Boot

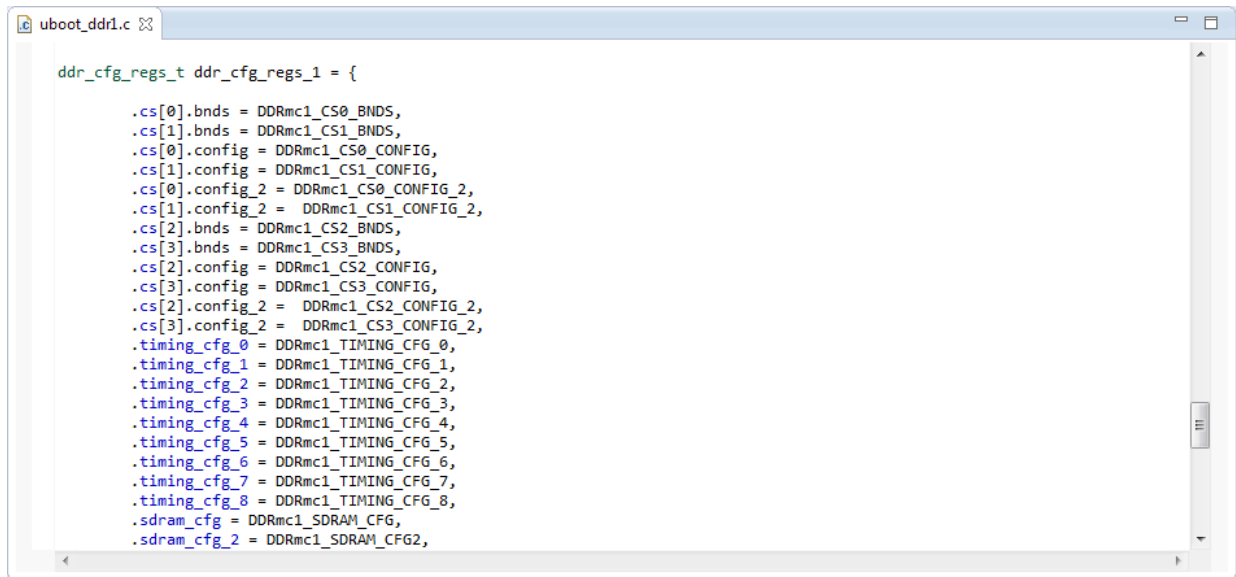
Perform the following steps to add DDR initialization code to U-Boot sources:

1. Open the QCVS project in CodeWarrior/Eclipse IDE.
2. Expand the **Generated\_Code** node in **Project Explorer** and open the generated file containing DDR initialization code (the file will be used in U-Boot integration), as shown in the figure below.

**Figure 1. Generated DDR initialization file**



The figure below shows the code of the DDR initialization file.

**Figure 2. DDR initialization code**


```

uboot_ddr1.c

ddr_cfg_regs_t ddr_cfg_regs_1 = {

    .cs[0].bnds = DDRmc1_CS0_BNDS,
    .cs[1].bnds = DDRmc1_CS1_BNDS,
    .cs[0].config = DDRmc1_CS0_CONFIG,
    .cs[1].config = DDRmc1_CS1_CONFIG,
    .cs[0].config_2 = DDRmc1_CS0_CONFIG_2,
    .cs[1].config_2 = DDRmc1_CS1_CONFIG_2,
    .cs[2].bnds = DDRmc1_CS2_BNDS,
    .cs[3].bnds = DDRmc1_CS3_BNDS,
    .cs[2].config = DDRmc1_CS2_CONFIG,
    .cs[3].config = DDRmc1_CS3_CONFIG,
    .cs[2].config_2 = DDRmc1_CS2_CONFIG_2,
    .cs[3].config_2 = DDRmc1_CS3_CONFIG_2,
    .timing_cfg_0 = DDRmc1_TIMING_CFG_0,
    .timing_cfg_1 = DDRmc1_TIMING_CFG_1,
    .timing_cfg_2 = DDRmc1_TIMING_CFG_2,
    .timing_cfg_3 = DDRmc1_TIMING_CFG_3,
    .timing_cfg_4 = DDRmc1_TIMING_CFG_4,
    .timing_cfg_5 = DDRmc1_TIMING_CFG_5,
    .timing_cfg_6 = DDRmc1_TIMING_CFG_6,
    .timing_cfg_7 = DDRmc1_TIMING_CFG_7,
    .timing_cfg_8 = DDRmc1_TIMING_CFG_8,
    .sdram_cfg = DDRmc1_SDRAM_CFG,
    .sdram_cfg_2 = DDRmc1_SDRAM_CFG2,
}

```

The DDR initialization file obtained from QCVS is a C source code file containing the following elements:

- A data structure definition and an instance of data structure, which stores the DDR configuration register values
  - Defines, which store the values for all the DDR configuration registers required for initialization
  - A macro definition for endianness byte swap. By default, the endianness used matches endianness of the platform.
3. Edit the /board/<custom\_board\_family>/<custom\_board\_name>/ddr.c file with the initialization code generated by the QCVS DDR tool.

**Figure 3. Edited U-Boot DDR file**

```

1  /*
2   * Copyright 2015 Freescale Semiconductor, Inc.
3   *
4   * SPDX-License-Identifier: GPL-2.0+
5   */
6
7  #include <common.h>
8  #include <fsl_ddr.h>
9  #include <fsl_ddr_dimm_params.h>
10 #include "ddr.h"
11
12 DECLARE_GLOBAL_DATA_PTR;
13
14 /* DDR Controller 1 configuration global structures */
15 ddr_cfg_regs_t ddr_cfg_regs_1 = {
16
17     .cs[0].bnds = DDRmc1_CS0_BNDS,
18     .cs[1].bnds = DDRmc1_CS1_BNDS,
19     .cs[0].config = DDRmc1_CS0_CONFIG,
20     .cs[1].config = DDRmc1_CS1_CONFIG,
21     .cs[0].config_2 = DDRmc1_CS0_CONFIG_2,
22     .cs[1].config_2 = DDRmc1_CS1_CONFIG_2,
23     .cs[2].bnds = DDRmc1_CS2_BNDS,
24     .cs[3].bnds = DDRmc1_CS3_BNDS,
25     .cs[2].config = DDRmc1_CS2_CONFIG,
26     .cs[3].config = DDRmc1_CS3_CONFIG,
27     .cs[2].config_2 = DDRmc1_CS2_CONFIG_2,
28     .cs[3].config_2 = DDRmc1_CS3_CONFIG_2,
29     .timing_cfg_0 = DDRmc1_TIMING_CFG_0,
30     .timing_cfg_1 = DDRmc1_TIMING_CFG_1,
31     .timing_cfg_2 = DDRmc1_TIMING_CFG_2,
32     .timing_cfg_3 = DDRmc1_TIMING_CFG_3,
33     .timing_cfg_4 = DDRmc1_TIMING_CFG_4,
34     .timing_cfg_5 = DDRmc1_TIMING_CFG_5,

```

---

**NOTE** Check the endianness used by U-Boot. If required, you can change the endianness using SWAT(x).

---

If changing the DDR code from U-Boot (to match the code generated by DDR tool) is difficult, then you can use an adaptation layer to adapt the DDR tool – generated code with the U-Boot DDR code.

## 3.4. Rebuild U-Boot

Perform the following steps to rebuild U-Boot:

1. Set the ARCH and CROSS\_COMPILE environment variables:

```

$ export ARCH=arm
$ export CROSS_COMPILE=/<path_to_toolchain>/aarch64-fsl-
linux/aarch64-fsl-linux-

```

2. Configure the board:

```

$ make <cutom_board_name>_defconfig

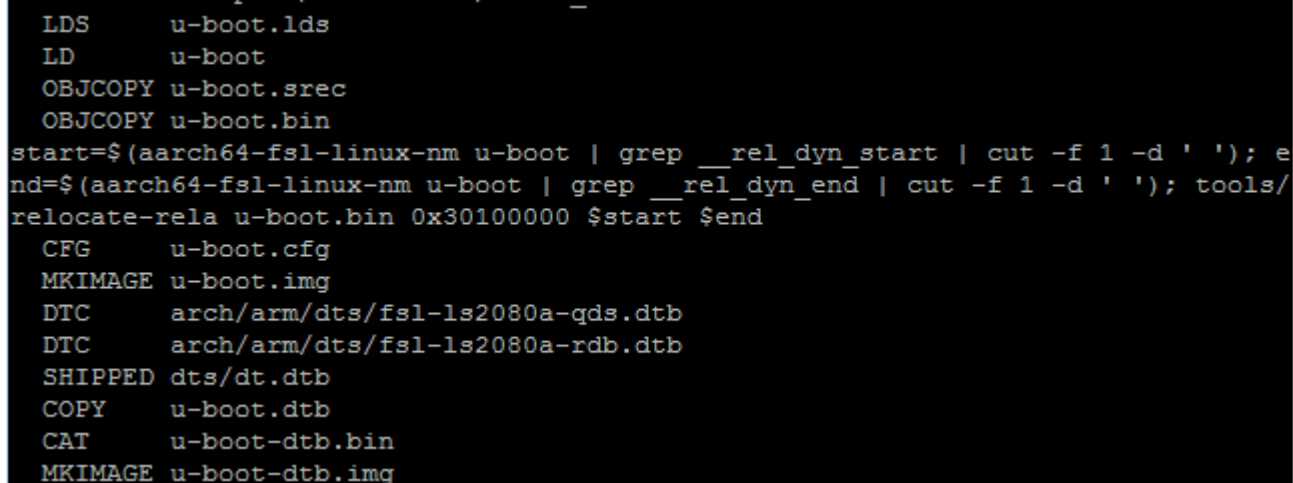
```

### 3. Build the U-Boot:

```
$ make
```

The figure below shows the output of the U-Boot console, after building the U-Boot.

**Figure 4. U-Boot console output**

A screenshot of a terminal window showing the output of the 'make' command for U-Boot. The output lists various steps: LDS u-boot.lds, LD u-boot, OBJCOPY u-boot.srec, OBJCOPY u-boot.bin, a complex command to find relocation addresses and use 'relocate-rela', CFG u-boot.cfg, MKIMAGE u-boot.img, DTC arch/arm/dts/fsl-ls2080a-qds.dtb, DTC arch/arm/dts/fsl-ls2080a-rdb.dtb, SHIPPED dts/dt.dtb, COPY u-boot.dtb, CAT u-boot-dtb.bin, and MKIMAGE u-boot-dtb.img.

```
LDS      u-boot.lds
LD       u-boot
OBJCOPY  u-boot.srec
OBJCOPY  u-boot.bin
start=$(aarch64-fsl-linux-nm u-boot | grep __rel_dyn_start | cut -f 1 -d ' '); e
nd=$(aarch64-fsl-linux-nm u-boot | grep __rel_dyn_end | cut -f 1 -d ' '); tools/
relocate-rela u-boot.bin 0x30100000 $start $end
CFG      u-boot.cfg
MKIMAGE  u-boot.img
DTC      arch/arm/dts/fsl-ls2080a-qds.dtb
DTC      arch/arm/dts/fsl-ls2080a-rdb.dtb
SHIPPED  dts/dt.dtb
COPY     u-boot.dtb
CAT      u-boot-dtb.bin
MKIMAGE  u-boot-dtb.img
```

## 4. Programming U-Boot using flash programmer

After building U-Boot, you can program it using CodeWarrior flash programmer. See CodeWarrior Targeting Manual for instructions on how to use flash programmer.

**How to Reach Us:**

Home Page:

[nxp.com](http://nxp.com)

E-mail:

[nxp.com/support](mailto:support@nxp.com)

Information in this document is provided solely to enable system and software implementers to use Freescale products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. Freescale reserves the right to make changes without further notice to any products herein.

Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. Freescale does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address: [nxp.com/SalesTermsandConditions](http://nxp.com/SalesTermsandConditions).

Freescale, the Freescale logo, CodeWarrior, QorIQ, and Processor Expert are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. All other product or service names are the property of their respective owners.

© 2016 Freescale Semiconductor, Inc.

