U-Boot SPL: DDR interactive mode

Stable: 23/03/2021 - 10:22 / Revision: 10/02/2021 - 10:00

A quality version of this page, approved on *23 March 2021*, was based off this revision.

In U-Boot, the U-Boot SPL can provide a debug console for DDR configuration.

It is the embedded part used by the STM32CubeMX DDR tuning tools.

Contents

1 DDR interactive mode

This mode is defined under the compilation flag CONFIG_STM32MP1_DDR_INTERACTIVE.

When this configuration flag is activated, there are two ways to enter interactive mode:

1. **load SPL and key press the character 'd'** (for DDR) in the console.
Polling is only done during 200 ms of the boot sequence.
It is therefore difficult to do manually and you should keep the key "d" pressed during the SPL startup until the **DDR>** prompt
2. **compile to force DDR_INTERACTIVE mode** by adding **DDR_INTERACTIVE=1** in the make option
the SPL then automatically enters DDR mode:

```
PC $> DEVICE_TREE=<device tree name> DDR_INTERACTIVE=1 make
```

When the DDR tool console is available, the following output is displayed after the SPL information banner:

```
0:DDR_RESET
DDR>
```

Interactive mode uses the same UART as the SPL console, by default the STMicroelectronics boards use UART4 with Bps=115200 Bit=8 Parity=None Stop=1

1.1 DDR steps

DDR interactive mode uses 5 steps to initialize the DDR controller and the PHY (including the "Mode Register" 0-3 sent to DDR during initialization) with parameters found in the device tree:

step	name	Description	DDR CTL state	DDR PHY state	DDR CTL registers	DDR PHY registers
0	DDR_RESET	Initial state Clock initialized **'param'** initialized with Device Tree	Reset	Reset	Reset Values	Reset Values
1	DDR_CTRL_INIT_DONE	Controller init done	Clocked	Reset	Init done with **'param.ctl'** CTL register can be updated (command edit)	Reset Values
	DDR		Running			Init done with **'param.phy'**

step	name	Description	DDR CTL state	DDR PHY state	DDR CTL registers	DDR PHY registers
2	PHY_INIT_DONE	DDR PHY init done	Waiting PHY	Clocked	-	PHY register can be updated (command edit)
3	DDR_READY	DDR ready and ports are open test can be executed	Running	Running	-	-
4	RUN	Continue execution (U-Boot load)	Running	Running	-	-

1.2 commands available

with :

- <type> = ctl, phy (for all DDR controller or PHY registers)

or one category: static, timing, map, perf, cal, dyn

- <reg> = name of the register (dx0dqstr for example)

You can execute the commands:

- help: displays help
- info: manages the DDR information (step, name, size and speed)
  - info: displays the DDR information
  - info <param> <val>: changes the <param> with <value> (<param> = "step", "name", "size" or "speed")
- freq: manages the DDR PHY frequency
  - freq: displays the DDR frequency
  - freq <frequency>: changes the frequency (in kHz), allowed only in step 0

```
DDR>freq
DDRPHY = 40000 kHz
DDR> freq 523000
DDRPHY = 522999 kHz
```

- param: manages the input parameters (read from the device tree)
  - param [type|reg]: prints input parameters
  - param <reg> <val>: edits parameters in step 0
- print [type|reg]: dump register
- edit: modifies register in the DDR controller and DDR PHY

need to be done at the correct step, AFTER the initialization from the input parameters (values from device tree)

  - step 1 => CTRL register update allowed
  - step 2 => PHY register update allowed

    - **step 3** => DDR ready (only for dynamic register for controller and PHY)
  - edit [**type**|**reg**]: modifies value register in DDR controller or PHY
- step: manages the step
  - step: lists the available steps
  - step [n]: goes to the step <n>

  you can reset and restart init sequence with "step 0"

```
0:DDR_RESET
1:DDR_CTRL_INIT_DONE
2:DDR_PHY_INIT_DONE
3:DDR_READY
4:RUN
```

- next: goes to the next step
- go: continues the U-Boot SPL execution (load u-boot, same than "step 4")
- reset: reboots the machine
- test: test management : see next chapter
- tuning: executes the tuning command, today only one sub command
  - tuning <n>: executing tuning procedure <n>
  - tuning help: displays the help message, lists available tuning procedures

```
0:Read DQS gating:software read DQS Gating
1:Bit de-skew:
2:Eye Training:or DQS training
3:Display registers:
4:Bist config:[nbErr] [seed]
```

The "param" command is a simple way to test the modified settings, as it modifies the input parameters (**'param'** read from the device tree). It is recommanded to execute this command at **step 0**. The modified values are applied at the correct #DDR steps.

The "print" and "edit" commands directly access the CTRL and PHY registers, so the values can be overidden by the input parameters when the driver executes the intialization steps. These commands are used for detailed debug of the DDR initialization.

1.3 Examples

```
DDR>info
step = 3 : DDR_READY
name = DDR3-1066/888 bin G 2x4Gb 533MHz v1.40
size = 0x40000000
speed = 533000 kHz
```

```
DDR>param cal
==phy.cal==
dx0dllcr= 0x40000000
dx0dqtr= 0xffffffff
dx0dqstr= 0x3db02000
dx1dllcr= 0x40000000
dx1dqtr= 0xffffffff
dx1dqstr= 0x3db02000
dx2dllcr= 0x40000000
```

```
dx2dqtr= 0xffffffff
dx2dqstr= 0x3db02000
dx3dllcr= 0x40000000
dx3dqtr= 0xffffffff
dx3dqstr= 0x3db02000
```

```
DDR>print dx0dqstr
dx0dqstr= 0x3db03001
```

```
DDR>print mstr
mstr= 0x00040401
```

1.4 DDR test

To execute the DDR test you need to be in the 'DDR ready' step (step 3).

To generate the list of tests, execute the 'test' command without any parameters:

```
DDR>step 3
```

```
DDR>test
test:25
0:All:
1:Simple DataBus:[addr]
2:DataBusWalking0:[loop] [addr]
3:DataBusWalking1:[loop] [addr]
4:AddressBus:[size] [addr]
5:MemDevice:[size] [addr]
6:SimultaneousSwitchingOutput:[size] [addr]
7:Noise:[pattern] [addr]
8:NoiseBurst:[size] [pattern] [addr]
9:Random:[size] [loop] [addr]
10:FrequencySelectivePattern :[size]
11:BlockSequential:[size]
12:Checkerboard:[size]
13:BitSpread:[size]
14:BitFlip:[size]
15:WalkingOnes:[size]
16:WalkingZeroes:[size]
17:infinite read:[addr]
18:infinite write:[addr]
```

Each test can be executed individually, and is identified by its number:

```
DDR>test 1
execute 1:Simple DataBus
Result: Pass [address 0xc0000000]
```

All tests can be executed with test 0:

```
DDR>test 0
execute 0:All
execute 1:Simple DataBus
```

```
result 1:Simple DataBus = Passed

execute 2:DataBusWalking0
running 100 loops at 0xc0000000
result 2:DataBusWalking0 = Passed

execute 3:DataBusWalking1
running 100 loops at 0xc0000000
result 3:DataBusWalking1 = Passed
....
check buffer.
pattern = 00000001.
check buffer.
result 15:WalkingOnes = Passed

Result: Pass [0/16 test failed]
```

2 Source code organization

- **driver = drivers/ram/stm32mp1/**
  - ddr initialization : stm32mp1_ddr.c
  - test : stm32mp1_tests.c
  - tuning code: stm32mp1_tuning.c
- device tree files for **DDR setting: arch/arm/dts/stm32mp15-*dtsi** , for example:
  - stm32mp15-ddr3-2x4Gb-1066-binG.dtsi
  - stm32mp15-ddr3-1x4Gb-1066-binG.dtsi

These files provide a list of defines:

```
#define DDR_<NAME>   <VALUE>
```

and include stm32mp15-ddr.dtsi to provide each register value for the DDR controller and PHY.

These DDR files are included in the board files stm32mp157c_<board>-uboot.dts needed to build the device tree with the required DDR configuration nodes.

3 Load the SPL into embedded RAM

To execute the SPL in DDR interactive mode, and to tune the DDR settings, the SPL needs to be loaded into embedded RAM:

- manually (to be repeated after each reset):
  - #with STM32CubeProgrammer (#USB or #UART)
  - #with GDB
- automatically by ROM code after each reset:
  - #Boot from SD card

3.1 with STM32CubeProgrammer

You can use the CLI command to directly load the SPL into the embedded RAM for peripheral boot on USB or UART.

In this case the load with STM32Programmer needs to be repeated after each reset.

Warning

The manual connection is difficult in parallel of STM32Programmer: it is recommended to activate the compilation flag DDR_INTERACTIVE to force the interactive mode.

The next example uses the Linux cli, but you can also use ST32CubeProgrammer for Windows (.bat file) with the same parameters.

3.1.1 USB

- Compile SPL with DDR_INTERACTIVE=1
- Select USB_Boot on the board

(for example STM32MP157x-EV1_-_hardware_description#Boot_related_switches).

- Connect USB to USB HS, reset the board and execute the STM32CubeProgrammer CLI command :

```
PC $> STM32_Programmer.sh -c port=usb1 -w spl/u-boot-spl.stm32 0x01 --start 0x01
```

3.1.2 UART

- Compile SPL with DDR_INTERACTIVE=1
- Select UART_Boot on the board

(for example STM32MP157x-EV1_-_hardware_description#Boot_related_switches).

- Connect board UART to the PC, reset the board and the STM32CubeProgrammer CLI command (with the correct tty interface : ttyS0 / ttyUSB0):

```
PC $> STM32_Programmer.sh -c port=/dev/ttyS0 br=115200 -w spl/u-boot-spl.stm32 0x01 --
start 0x01
```

3.2 with GDB

- Compile SPL with DDR_INTERACTIVE=1

- Select "Engineering boot" = "Forced USB boot for flashing" on the board

(for example STM32MP157x-EV1_-_hardware_description#Boot_related_switches).

- After GDB connection, reset and attach to the target, execute the commands:

```
(gdb) file u-boot-spl.elf
(gdb) load
(gdb) set $dtb =  __bss_end
(gdb) restore spl/u-boot-spl.dtb binary $dtb
```

Warning: SPL DTB is not included in the ELF file by default; you need to load it manually with GDB at the correct location.

3.3 Boot from SD card

You can directly update SPL on the SD card used, and let the Boot ROM code load the binary for the next boot form SD card.

In this case, activate the DDR mode by pressing the 'd' key continously (DDR_INTERACTIVE=1 is not mandatory).

3.3.1 Writing the SPL to SDMMC on Linux PC

See How to manually update bootloaders to update partition 1 and 2 with partition <n> = /dev/mmcblk0p<n> or /dev/sdb<n> :

```
PC $> dd if=u-boot-spl.stm32 of=<dev>1 conv=fdatasync
PC $> dd if=u-boot-spl.stm32 of=<dev>2 conv=fdatasync
```

3.3.2 Writing SPL in SDMMC with STM32CubeProgrammer

Warning: STM32CubeProgrammer uses the DDR for U-Boot execution, so this method cannot be used during debug of the DDR settings and execution of the unitary (DDR) test..

To boot from SDCARD, the generated files must be written in 2 partitions :

- 1: spl/u-boot-spl.stm32
- 4: u-boot.img

The Linux command is:

```
PC $> STM32_Programmer.sh -c port=usb1  -w FlashLayout.tsv
```

With FlashLayout.tsv following STM32CubeProgrammer_flashlayout format, for example:

```
#Option        Id        Name        Type        Device        Offset
Binary
P        0x01        fsbl1        Binary        SDMMC1        0x00004400        spl/u-
boot-spl.stm32
PE        0x02        fsbl2        Binary        SDMMC1        0x00044400        none
P        0x03        ssbl        Binary        SDMMC1        0x00084400        u-boot.img.
bin
```

You can also use programmer for Windows (.bat file) with the same parameters.

Das U-Boot -- the Universal Boot Loader (see U-Boot_overview)

Universal Asynchronous Receiver/Transmitter