How To Adapt a CodeWarrior ARMv8 Stationery "Hello World C Project" to Download to, And Run From, OCRAM

Introduction

CodeWarrior projects provide an excellent test bed for developing and evaluating embedded target code. Standard CodeWarrior projects typically initialize and use target DDR to host the downloaded project code, but there are some cases in which target evaluation or testing must be performed but target DDR is not available. This Tech Note describes a method for downloading a CodeWarrior ARMv8 project to the target's On-Chip RAM, or OCRAM, and running the target code in OCRAM, without needing any other target memory.

Limitations

OCRAM in NXP's LS1012A , LS208xA, LS204xA, LS102xA and LS104xA processors is 128K (0x20000) bytes, which is large enough to hold some embedded C or C++ code while still allowing for standard uses of RAM such as variables, stack, and heap. But care must be taken to ensure that test code intended to be hosted by and run from only OCRAM not only fits within this space during downloads from the CodeWarrior ARMv8 debugger, but also has enough available space to manage the target code's run-time RAM requirements such as variables, stack and heap. Recursive functions are especially vulnerable to stack overflows in this limited RAM space.

The limited capacity of OCRAM on QorIQ LS processors prevents the use of large support libraries such as stdio.h, which prevents the use of convenient functions such as printf(). The user can still monitor data and variable values in the debugger's Registers, Variables, and Memory views.

Debugging should be limited to use only one core due to the stack and heap space needed for each core.

Trace data can be collected while debugging in OCRAM. Be sure to select the On-Chip buffer in CodeWarrior ARMv8's *Trace Commander* view. Refer to the "ARMv8_Tracing_and_Analysis" manual for details.

Required Changes

Three general changes need to be made to any CodeWarrior ARMv8 C stationery project to enable it to run from only OCRAM:

1. Source code:
    a. delete all instances of "**#include** <stdio.h>" and **printf()**

    b. delete references to **___DDR_ADDRESS**
2. Linker control file: redefine the mapping of target code
3. Enable the use of OCRAM to execute code (LS20xxA projects only)

Source Code Changes:

"**#include** <stdio.h>" is included in three source files of every "Hello World C" project:

- src\exceptions\exceptions.c
- src\gic\gic.c
- src\main.c

Delete "**#include** <stdio.h>", and all references to **printf()**, from each of these three files and save the changes.

Next, it is recommended (though not required) to edit src\start.S, to delete the use of the Linker Control File variable **___DDR_ADDRESS**. This variable is used during the initialization of the target's MMU. Leaving this variable, and this portion of the initialization code, in place will not affect the operation of the target code, but doing so makes for bad code. If the reference to **___DDR_ADDRESS** is retained in start.S, make sure it is defined in the Linker Control File. If it is deleted, then also delete the block of code that uses it, in the function **_init_tables**,

Linker Control File Changes:

The Linker Control File will also need editing. This section describes what linker definitions are required.

In the CodeWarrior project, expand the **Linker_Files** folder and edit the file **aarch64elf.x**.

OCRAM on LS20xxA processors starts at address 0x1800_0000, so for an LS20xxA target only, change the definition of

PROVIDE (___OCRAM_ADDRESS = 0x10000000);

to

PROVIDE (___OCRAM_ADDRESS = 0x18000000);

NOTE: PROVIDE (___OCRAM_ADDRESS = 0x10000000); is valid for LS1043A (and derivatives) and LS1012A (and derivatives.)

Next, change

```
PROVIDE (___START_RAM_ADDRESS = ___DDR_ADDRESS + ___CORE_NUMBER * ___MEMORY_SIZE);
```

to

```
PROVIDE (___START_RAM_ADDRESS = ___OCRAM_ADDRESS + ___CORE_NUMBER * ___OCRAM_SIZE);
```

Next, change

```
PROVIDE (___STACK_AND_HEAP_SIZE = 0x80000);
```

to

```
PROVIDE (___STACK_AND_HEAP_SIZE = 0x2000);
```

NOTE: The Linker variable ___STACK_SIZE_PER_CORE is not used, so does not need to be reduced in defined size.

Finally, change

```
PROVIDE(___END_RAM_ADDRESS_EXPECTED = ___START_RAM_ADDRESS + ___MEMORY_SIZE);
```

to

```
PROVIDE(___END_RAM_ADDRESS_EXPECTED = ___START_RAM_ADDRESS + ___OCRAM_SIZE);
```

The CodeWarrior build tools will warn you with a "Not enough memory" error displayed in the Console view when building the CodeWarrior project, when your code size grows too large to fit in the available OCRAM space. Should this happen, the solution is one or both of the following:

- Reduce code size
- Reduce ___STACK_AND_HEAP_SIZE value

A convenient way to determine how much total RAM your code requires is to examine the contents of either the .map or .lst file after building your project. You will find one or both of these files in your project's **Debug** folder, but they may not be generated if any errors are generated. If you do not get either a .map or .lst file, edit the Linker Control File and temporarily make ___OCRAM_SIZE large enough to build without errors.

Open the .map or .lst file using your favorite text editor and examine it for your project's use of target memory. The generated file will tell you the calculated value for how much RAM was required for the build. For example a .map file will look something like this:

```
0x00000000180bd5b0  PROVIDE (___END_RAM_ADDRESS, .)
0x0000000018020000  PROVIDE (___END_RAM_ADDRESS_EXPECTED, (___START_RAM_ADDRESS + ___OCRAM_SIZE))
```

A .lst file will have the same information but the two symbols, ___END_RAM_ADDRESS and ___END_RAM_ADDRESS_EXPECTED, are listed separately in the symbol table. Each is listed with its calculated value so the same information is available as in the .map file.

The upper limit of OCRAM is 0x18020000. If the address calculated for ___END_RAM_ADDRESS is greater than 0x18020000, you will get the "Not enough memory" error. Determine how much RAM space above 0x18020000 is being requested by your code, then determine if a smaller ___STACK_AND_HEAP_SIZE size will be enough (while still being large enough to support your executable code), or if paring down the size of your code is required. Some iterations may be required to get your code down to a manageable size.

Enable the use of OCRAM to execute code (LS20xxA projects only):

For LS20xxA projects, edit the Target Initialization File in the project's Target Connection Configuration, to add these lines just before "end" at the end of the file:

```
# Enable "honor_ewa_en" bit in "SA Auxiliary Control register" of the CCN-504.
CCSR_LE_M(0x04080500, 0x000008d7)
```

Recommended (but not required) changes

Debugging more than one core in OCRAM would take some very careful allocation of OCRAM resources for each core's Stack and Heap spaces. In most cases, limiting the project to use only one core would be best, so the phrase + ___CORE_NUMBER * ___OCRAM_SIZE would not needed, and could be safely deleted.

Delete references to DDR in the aarch64elf.x Linker Control file:

```
PROVIDE (___DDR_ADDRESS = 0x80000000);

PROVIDE (___MEMORY_SIZE = 0x1000000);
```

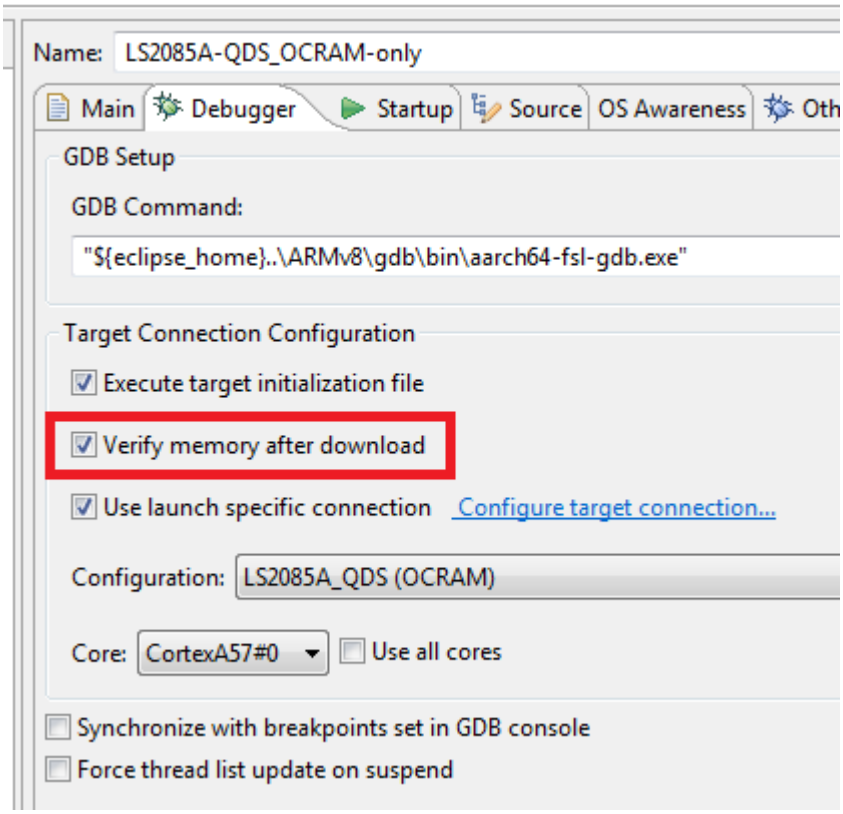
Delete the call to DDR initialization near the end of the Target Initialization file:

- LS1043-RDB, LS1023A-RDB:
    - Init_DDRC()

- LS2085A-RDB, LS2045A-RDB:
    - DDR_freq, DPDDR_freq = Detect_DDR_Freq()
    - Init_DDRC(DDR_freq)
    - Init_DPDDRC(DPDDR_freq)

- LS2080A-RDB, LS2040A-RDB:
    - DDR_freq = Detect_DDR_Freq()
    - Init_DDRC(DDR_freq)

Enable the *Verify memory after download* option in the Debugger settings:



Debugging

Debugging the OCRAM-hosted application proceeds exactly the same as for DDR-hosted code. Configure the Target Connection definition as described in Section 5.1 "Target Connection configurator

overview" of the ARMv8_Targeting_Manual, then start the debugger according to Section 2.5.1 "Debugging Bareboard project" in this same manual.

Conclusion

Debugging a CodeWarrior ARMv8 project usually requires target DDR. Boards that don't have DDR available, or not enough DDR, can still be debugged if the guidelines discussed in this Tech Note are followed, to limit the size of the application code to fit within the always-available OCRAM in the target LS processor. Once a project's application code is reduced enough to squeeze into the limited OCRAM space, the full power of the CodeWarrior debugger becomes available.