AN13112

How to Use .mac File to Initialize Device Connected to FlexSPI on i.MX RT

Rev. 0 — 30 August 2021 Application Note

Contents

1 Introduction

The i.MX RT series MCU is a crossover product from NXP. It includes a FlexSPI controller which supports various vendor devices, such as serial NOR Flash, HyperBus devices.

For debugging convenience, download the code directly to the on-chip RAM or external RAM to run. When debugging the code in the external RAM, initialize the external RAM before the code is downloaded into the RAM. IAR is realized through the `.mac` file.

This application note uses IAR as the debugging environment. It describes how to use the `.mac` file to initialize the devices connected to the i.MX RT FlexSPI controller, such as HyperRAM, Flash, and so on.

The hardware is based on RT1060-EVK board and the software is based on SDK 2.7.0.

2 Overview

In the embedded development, the program and read speed of the Flash chip is slow. However, the code may be modified frequently when debugging, so the read and program speed of the code has a great influence on the development speed. Therefore, for debugging, the code is downloaded directly into the on-chip RAM or external RAM to run. The debugging speed can be improved.

When using the on-chip RAM for debugging, set the ROM and RAM addresses in the `icf` file of IAR to the addresses of the on-chip RAM. If the on-chip RAM space is not enough to use, use the external RAM. When debugging code in external RAM, initialize the external RAM before the code is downloaded. In IAR, the initialization of external RAM is done through the `.mac` file.

This application note describes how to use the `.mac` file to initialize the external RAM connected to FlexSPI controller. It includes the following steps:

1. PIN initialization
2. Clock initialization
3. FlexSPI initialization and device initialization

Now, FlexSPI device is initialized successfully.

3 Example

This section uses the `hello_world` demo as an example to explain how to modify the `.mac` file of the SDK to initialize the HyperRAM device via FlexSPI. To initialize other devices, such as Flash, imitate HyperRAM initialization code and modify relevant register values according to the parameters of the device used.

1. Enter the *SDK_2.7.0_EVK-MIMXRT1060\boards\evkmimxrt1060\demo_apps\hello_world\iar* path, copy `evkmimxrt1060.mac`, and rename it to `evkmimxrt1060_hyperram_init.mac`.
2. Open the `evkmimxrt1060_hyperram_init.mac` file and add initialization functions to initialize the PIN, clock, and device, as shown in Figure 1.

```
208 execUserPreload()
209 {
210   _load_dcdc_trim();
211   _pin_init();
212   _clock_init();
213   _hyperram_Init();
214 }
215
216 execUserReset()
217 {
218   _load_dcdc_trim();
219   _pin_init();
220   _clock_init();
221   _hyperram_Init();
222 }
```

Figure 1. Adding initialization functions

3.1 PIN initialization

Initialize pins connected between FlexSPI and external device on hardware. Generally, there are pins such as data pin, clock pin, DQS pin. Initialize all pins according to the hardware connection. Figure 2 shows the detailed pin initialization code.

```
_pin_init()
{
  // Set Pin Mux for HyperRAM
  __writeMemory32(0x00000011, 0x401F81D4, "Memory"); // FLEXSPIB_DATA03
  __writeMemory32(0x00000011, 0x401F81D8, "Memory"); // FLEXSPIB_DATA02
  __writeMemory32(0x00000011, 0x401F81DC, "Memory"); // FLEXSPIB_DATA01
  __writeMemory32(0x00000011, 0x401F81E0, "Memory"); // FLEXSPIB_DATA00
  __writeMemory32(0x00000011, 0x401F81E4, "Memory"); // FLEXSPIB_SCLK
  __writeMemory32(0x00000011, 0x401F81E8, "Memory"); // FLEXSPIA_DQS
  __writeMemory32(0x00000011, 0x401F81EC, "Memory"); // FLEXSPIA_SS0_B
  __writeMemory32(0x00000011, 0x401F81F0, "Memory"); // FLEXSPIA_SCLK
  __writeMemory32(0x00000011, 0x401F81F4, "Memory"); // FLEXSPIA_DATA00
  __writeMemory32(0x00000011, 0x401F81F8, "Memory"); // FLEXSPIA_DATA01
  __writeMemory32(0x00000011, 0x401F81FC, "Memory"); // FLEXSPIA_DATA02
  __writeMemory32(0x00000011, 0x401F8200, "Memory"); // FLEXSPIA_DATA03

  // Set Pin Config for HyperRAM
  __writeMemory32(0x000110F9, 0x401F83C4, "Memory"); // FLEXSPIB_DATA03
  __writeMemory32(0x000110F9, 0x401F83C8, "Memory"); // FLEXSPIB_DATA02
  __writeMemory32(0x000110F9, 0x401F83CC, "Memory"); // FLEXSPIB_DATA01
  __writeMemory32(0x000110F9, 0x401F83D0, "Memory"); // FLEXSPIB_DATA00
  __writeMemory32(0x000110F9, 0x401F83D4, "Memory"); // FLEXSPIB_SCLK
  __writeMemory32(0x000110F9, 0x401F83D8, "Memory"); // FLEXSPIA_DQS
  __writeMemory32(0x000110F9, 0x401F83DC, "Memory"); // FLEXSPIA_SS0_B
  __writeMemory32(0x000110F9, 0x401F83E0, "Memory"); // FLEXSPIA_SCLK
  __writeMemory32(0x000110F9, 0x401F83E4, "Memory"); // FLEXSPIA_DATA00
  __writeMemory32(0x000110F9, 0x401F83E8, "Memory"); // FLEXSPIA_DATA01
  __writeMemory32(0x000110F9, 0x401F83EC, "Memory"); // FLEXSPIA_DATA02
  __writeMemory32(0x000110F9, 0x401F83F0, "Memory"); // FLEXSPIA_DATA03
}
```

Figure 2. PIN initialization

3.2 Clock initialization

Clock initialization is to configure the clock for FlexSPI controller. Configure the FlexSPI clock according to the clock frequency supported by the HyperRAM device and the maximum clock frequency supported by the FlexSPI controller. Figure 3 shows the detailed clock initialization code.

```
_clock_init()
{
  __var reg;
  // Enable all clocks
  __writeMemory32(0xffffffff, 0x400FC068, "Memory");
  __writeMemory32(0xffffffff, 0x400FC06C, "Memory");
  __writeMemory32(0xffffffff, 0x400FC070, "Memory");
  __writeMemory32(0xffffffff, 0x400FC074, "Memory");
  __writeMemory32(0xffffffff, 0x400FC078, "Memory");
  __writeMemory32(0xffffffff, 0x400FC07C, "Memory");
  __writeMemory32(0xffffffff, 0x400FC080, "Memory");

  // Config PLL for FlexSPI
  // PERCLK_PODF: 1 divide by 2
  __writeMemory32(0x04900001, 0x400FC01C, "Memory");
  // Enable SYS PLL but keep it bypassed.
  __writeMemory32(0x00012001, 0x400D8030, "Memory");
  do
  {
    reg = __readMemory32(0x400D8030, "Memory");
  }while((reg & 0x80000000) == 0);
  // Disable bypass of SYS PLL
  __writeMemory32(0x00002001, 0x400D8030, "Memory");

  // PFD2_FRAC: 29, PLL2 PFD2=528*18/PFD2_FRAC=327
  // Ungate SYS PLL PFD2
  __writeMemory32(0x001D0000, 0x400D8100, "Memory");

  __writeMemory32(0x44100001, 0x400FC01C, "Memory");//divide by 1

  __message "clock init done\n";
}
```

Figure 3. Clock initialization

3.3 FlexSPI and device initialization

To initialize the FlexSPI controller, perform the following steps:

1. Reset FlexSPI before configuration.
2. Set MCR0[MDIS] to 0x1 (make sure the FlexSPI controller is in stop mode).
3. Configure FlexSPI module control registers: MCR0, MCR1, MCR2.

NOTE

Do not modify MCR0[MDIS].

4. If AHB command is used, configure the AHB bus control register (AHBCR) and the AHB RX buffer control register (AHBRXBUFxCR0).
5. Configure the IP RX/TX FIFO control registers: IPRXFCR, IPTXFCR.

To initialize the device, perform the following steps:

1. According to the connected HyperRAM device parameters, configure the Flash control registers: FLSHxCR0, FLSHxCR1, FLSHxCR2.
2. According to the selected clock source, configure the DLL control register (DLLxCR).
3. According to whether the write mask is enabled or not, configure the flash control register FLSHxCR4.
4. Set MCR0[MDIS] to 0x0 (exit stop mode).
5. To unlock LUT, configure LUTKEY and LUTCR register.
6. According to the HyperRAM device parameters, update the LUT table (for AHB command or IP command access) to set the timing of HyperRAM data read and write, register read, and write operations.
7. To lock LUT, configure LUTKEY and LUTCR register.
8. To reset the FlexSPI controller, set MCR0[SWRESET] to 0x1.

Figure 4 shows the detail FlexSPI and device initialization code.

```
_hyperram_Init()
{
  // Config FlexSPI Registers
  __writeMemory32(0xFFFF80C0, 0x402A8000, "Memory"); // MCR0
  _Flexspi_reset();
  __writeMemory32(0xFFFF3032, 0x402A8000, "Memory"); // MCR0
  __writeMemory32(0xFFFFFFFF, 0x402A8004, "Memory"); // MCR1
  __writeMemory32(0x200801F7, 0x402A8008, "Memory"); // MCR2
  __writeMemory32(0x00000078, 0x402A800C, "Memory"); // AHBCR
  __writeMemory32(0x80000020, 0x402A8020, "Memory"); // AHBRXBUFCR00
  __writeMemory32(0x80000020, 0x402A8024, "Memory"); // AHBRXBUFCR01
  __writeMemory32(0x80000020, 0x402A8028, "Memory"); // AHBRXBUFCR02
  __writeMemory32(0x80000020, 0x402A802C, "Memory"); // AHBRXBUFCR03
  __writeMemory32(0x00000000, 0x402A80B8, "Memory"); // IPRXFCR
  __writeMemory32(0x00000000, 0x402A80BC, "Memory"); // IPTXFCR
  // Config HyperRAM
  __writeMemory32(0x00004000, 0x402A8060, "Memory"); // FLASHA1CR0
  __writeMemory32(0x00021C63, 0x402A8070, "Memory"); // FLASHA1CR1
  __writeMemory32(0x00000100, 0x402A8080, "Memory"); // FLASHA1CR2
  __writeMemory32(0x00001D00, 0x402A80C0, "Memory"); // DLLCR
  __writeMemory32(0x00000000, 0x402A8094, "Memory"); // FLASHA1CR4
  __writeMemory32(0x00000004, 0x402A8094, "Memory"); // FLASHA1CR4
  __writeMemory32(0xFFFF3030, 0x402A8000, "Memory"); // MCR0
  __writeMemory32(0x5AF05AF0, 0x402A8018, "Memory"); // LUTKEY
  __writeMemory32(0x00000002, 0x402A801C, "Memory"); // LUTCR
  __writeMemory32(0x8B1887A0, 0x402A8200, "Memory"); // LUT[0]
  __writeMemory32(0xB7078F10, 0x402A8204, "Memory"); // LUT[1]
  __writeMemory32(0x0000A704, 0x402A8208, "Memory"); // LUT[2]
  __writeMemory32(0x00000000, 0x402A820C, "Memory"); // LUT[3]
  __writeMemory32(0x8B188720, 0x402A8210, "Memory"); // LUT[4]
  __writeMemory32(0xB7078F10, 0x402A8214, "Memory"); // LUT[5]
  __writeMemory32(0x0000A304, 0x402A8218, "Memory"); // LUT[6]
  __writeMemory32(0x00000000, 0x402A821C, "Memory"); // LUT[7]
  __writeMemory32(0x8B1887E0, 0x402A8220, "Memory"); // LUT[8]
  __writeMemory32(0xB7078F10, 0x402A8224, "Memory"); // LUT[9]
  __writeMemory32(0x0000A704, 0x402A8228, "Memory"); // LUT[10]
  __writeMemory32(0x00000000, 0x402A822C, "Memory"); // LUT[11]
  __writeMemory32(0x8B188760, 0x402A8230, "Memory"); // LUT[12]
  __writeMemory32(0xB7078F10, 0x402A8234, "Memory"); // LUT[13]
  __writeMemory32(0x0000A304, 0x402A8238, "Memory"); // LUT[14]
  __writeMemory32(0x00000000, 0x402A823C, "Memory"); // LUT[15]
  __writeMemory32(0x5AF05AF0, 0x402A8018, "Memory"); // LUTKEY
  __writeMemory32(0x00000001, 0x402A801C, "Memory"); // LUTCR
  _Flexspi_reset();
  __message "HyperRAM init done\n";
}
```

```
_Flexspi_reset()
{
  __var reg;
   reg = __readMemory32(0x402A8000, "Memory");
  __writeMemory32((reg | 0x1), 0x402A8000, "Memory");
  do
  {
    reg = __readMemory32(0x402A8000, "Memory");
  }while((reg & 0x1) != 0);
}
```

Figure 4. FlexSPI and device initialization

4 References

1. *i.MX RT1060 Processor Reference Manual* (document IMXRT1060RM)

5 Revision history

Table 1.

Revision number	Date	Description
0	30 August 2021	Initial release

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Limited warranty and liability — Information in this document is provided solely to enable system and software implementers to use NXP 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. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP 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 NXP 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. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

Right to make changes - NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer’s applications and products. Customer’s responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer’s applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 30 August 2021

Document identifier: AN13112

arm