Booting off Flash Larger than 4MB in IDT Integrated Communications Processors

Application Note AN-507

Notes

By Brandon Wong

Overview

Although this document uses the RC32438 Data Sheet and User Reference Manual as examples, the information applies equally to the RC32434, RC32435, RC32365, RC32355, and RC32336 devices.

General Purpose I/O Pin Functions

Table 1 displays the alternate GPIO functions. For additional information, see Chapter 6 of the 79RC32438 User Reference Manual at IDT.com.

GPIO Pins	Alternate Function Pin Name	Alternate Function Description	Alternate Function Pin Type
20	MADDR[22]	Memory and Peripheral Bus Address	Output
21	MADDR[23]	Memory and Peripheral Bus Address	Output
22	MADDR[24]	Memory and Peripheral Bus Address	Output
23	MADDR[25]	Memory and Peripheral Bus Address	Output

Table 1 General Purpose I/O Pin Alternate Function

All of the GPIO pins are configured to be an INPUT on power-on (GPIOCFG Register = 0x0000_0000). The GPIO[23:0] pins have internal pull-ups, but these may not be strong enough. The GPIO shared Address Lines are by default GPIO Inputs, pull-ups or pull-downs may be necessary to access the proper location in flash upon boot up. The IDT processors boot from Device 0 or 0xBFC0_0000.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
B				F				C				0			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0				0				0				0			

For example if a 8MB Boot Flash device is used, a couple of options can be presented:

1. Use a Pull-Up on A22 / GPIO20 line, so the boot code will reside in the upper 4MB of Flash.
2. Use a Pull-Down on A22 / GPIO20 line, so the boot code will reside in the lower 4MB of Flash.

Note: A25:A23 / GPIO23:21 should have pull-ups in both of these situations.

Each of these options has advantages and disadvantages.

Notes

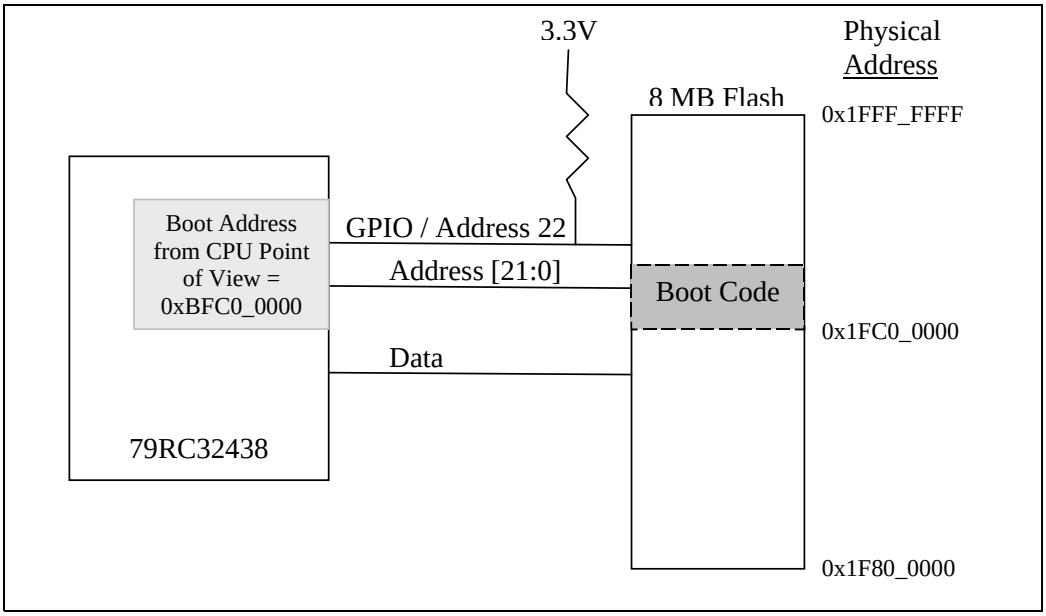

Figure 1 8MB Flash with Pull-up on Address 22

Pull-up on A22

On power-on or reset, the reset vector is 0xBFC0_0000. Once the GPIO is configured to be a MADDR22, the Program Counter will continue to execute in the 0xBFC0_0000 region.

Advantage:

No software changes need to be made to account for the address during execution. The software provided by IDT (Linux, IDT/boot, IDT/sim) would not need to be modified to jump to a different address, and the makefile and/or linker script would not need to be modified to reflect this.

Disadvantage:

The FLASH will not be a contiguous block. It will be broken up by the boot code, because the boot code will reside in the upper 4 MB of FLASH. The lower 4MB of FLASH and any region after the boot code will be untouched. If Linux or any other OS would like to utilize FLASH, then it must know these boundary regions. This may or may not be a real issue depending on how the 2 regions are used.

Notes

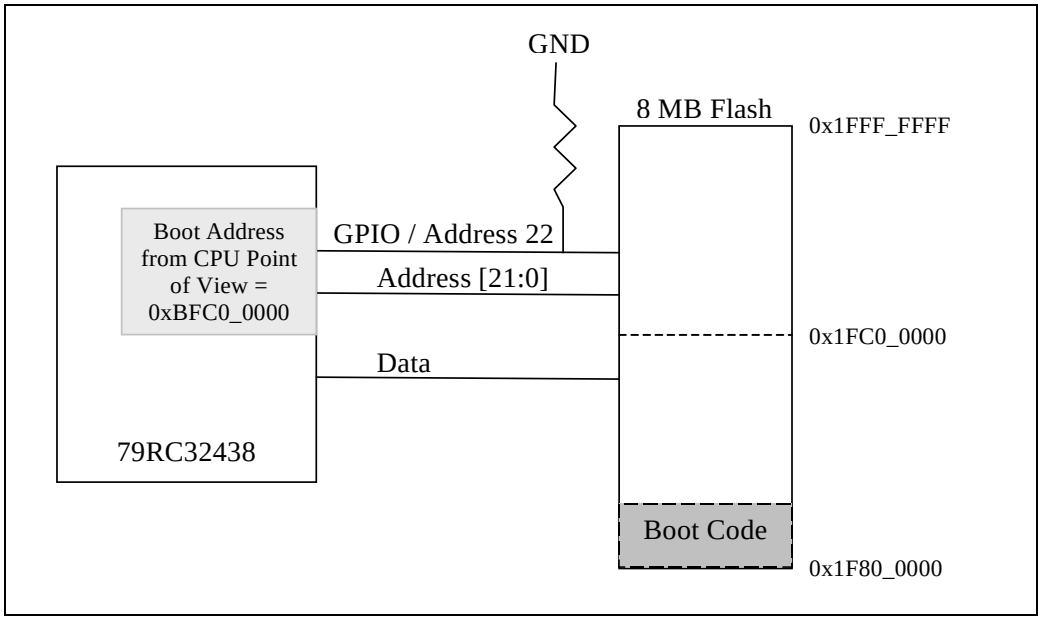

Figure 2 8MB Flash with Pull-down on Address 22

Pull-down on A22

CPU will think the reset vector is 0xBFC0_0000, but the actual physical address is 0xBF80_0000 due to the pull-down on A22. Thus the Program Counter will need to be changed to the 0xBF8x_xxxx region before the GPIO/Address pins are configured from GPIO Inputs to Address lines. Otherwise A22 will be driven HIGH after the GPIO configuration, and the CPU will begin executing from the 0xBFCx_xxxx region.

Advantage:

One contiguous block of un-used FLASH is provided, since the boot initialization code resides at the beginning of FLASH (0xBFC0_0000).

Disadvantage:

Method 1) Software must perform a jump to the 0xBF8x_xxxx region, before the GPIO Input line is changed to A22. It can just be a simple jump in the reset vector table to the initialization function.

Method 2) Change the linker script, so that the boot code resides at 0xBF8x_xxxx region.

This will change the Program Counter to 0xBF8x_xxxx, but the FLASH will still be executing the same code region at this point. This is due to the fact that A22 is still a GPIO Input and is pulled-down. Once the GPIO line is changed to A22, the code will continue to execute as expected in the 0xBF8x_xxxx. The makefile and/or linker script may need to be modified to reflect the address change.

> Note: If the jump to the 0xBF8x_xxxx is not performed, the Program Counter will still be executing code in the 0xBFCx_xxxx region after the GPIO Line is changed to A22. This would cause a problem, because the boot code region lays at 0xBF8x_xxxx region and not the 0xBFCx_xxxx region.

Example of Method 1)

```
bfc00000:       j       bf800408 <__start>
bfc00004:       nop
```

Example of Method 2 using IDT/sim)

File = "Sim\MAKE\EB434LE\romscrip"

The following pictures show the contents of the linker script file. The blue circles show the original content, while the red circles show what changed.

Notes

```
OUTPUT_FORMAT("elf32-bigmips", "elf32-bigmips",
        "elf32-littlemips")
OUTPUT_ARCH(mips)
ENTRY(start)
 SEARCH_DIR(/usr/local/mips64orion-idt-elf/lib);
/* Do we need any of these for elf?
    __DYNAMIC = 0;    */
_DYNAMIC_LINK = 0;
SECTIONS
{
  /* Read-only sections, merged into text segment: */
  . = 0xBFC00000;
  .interp       : { *(.interp)   }
  .reginfo        (NOLOAD) : { *(.reginfo) }
  .dynamic        : { *(.dynamic) }
  .dynstr         : { *(.dynstr)   }
  .dynsym         : { *(.dynsym)   }
  .hash           : { *(.hash)   }
  .rel.text       :
    { *(.rel.text) *(.rel.gnu.linkonce.t*) }
  .rela.text      :
    { *(.rela.text) *(.rela.gnu.linkonce.t*) }
  .rel.data       :
    { *(.rel.data) *(.rel.gnu.linkonce.d*) }
  .rela.data      :
    { *(.rela.data) *(.rela.gnu.linkonce.d*) }
  .rel.rodata     :
    { *(.rel.rodata) *(.rel.gnu.linkonce.r*) }
  .rela.rodata    :
    { *(.rela.rodata) *(.rela.gnu.linkonce.r*) }
  .rel.got        : { *(.rel.got)    }
  .rela.got       : { *(.rela.got)   }
  .rel.ctors      : { *(.rel.ctors)  }
  .rela.ctors     : { *(.rela.ctors) }
  .rel.dtors      : { *(.rel.dtors)  }
  .rela.dtors     : { *(.rela.dtors) }
  .rel.init       : { *(.rel.init) }
  .rela.init      : { *(.rela.init)  }
  .rel.fini       : { *(.rel.fini) }
  .rela.fini      : { *(.rela.fini)  }
  .rel.bss        : { *(.rel.bss)    }
  .rela.bss       : { *(.rela.bss)   }
  .rel.plt        : { *(.rel.plt)    }
  .rela.plt       : { *(.rela.plt)   }
  .init           : { *(.init) } =0
  .text 0xBFC00000 :
  {
    _ftext = . ;
    *(.text)
    *(.stub)
```

Notes

```
OUTPUT_FORMAT("elf32-bigmips", "elf32-bigmips",
         "elf32-littlemips")
OUTPUT_ARCH(mips)
ENTRY(start)
 SEARCH_DIR(/usr/local/mips64orion-idt-elf/lib);
/* Do we need any of these for elf?
    __DYNAMIC = 0;    */
_DYNAMIC_LINK = 0;
SECTIONS
{
  /* Read-only sections, merged into text segment: */
  . = 0xBF800000;
  .interp     : { *(.interp)  }
  .reginfo        (NOLOAD) : { *(.reginfo) }
  .dynamic        : { *(.dynamic) }
  .dynstr         : { *(.dynstr)   }
  .dynsym         : { *(.dynsym)   }
  .hash           : { *(.hash)   }
  .rel.text       :
    { *(.rel.text) *(.rel.gnu.linkonce.t*) }
  .rela.text      :
    { *(.rela.text) *(.rela.gnu.linkonce.t*) }
  .rel.data       :
    { *(.rel.data) *(.rel.gnu.linkonce.d*) }
  .rela.data      :
    { *(.rela.data) *(.rela.gnu.linkonce.d*) }
  .rel.rodata     :
    { *(.rel.rodata) *(.rel.gnu.linkonce.r*) }
  .rela.rodata    :
    { *(.rela.rodata) *(.rela.gnu.linkonce.r*) }
  .rel.got        : { *(.rel.got)    }
  .rela.got       : { *(.rela.got)   }
  .rel.ctors      : { *(.rel.ctors)  }
  .rela.ctors     : { *(.rela.ctors) }
  .rel.dtors      : { *(.rel.dtors)  }
  .rela.dtors     : { *(.rela.dtors) }
  .rel.init       : { *(.rel.init) }
  .rela.init      : { *(.rela.init)  }
  .rel.fini       : { *(.rel.fini) }
  .rela.fini      : { *(.rela.fini)  }
  .rel.bss        : { *(.rel.bss)    }
  .rela.bss       : { *(.rela.bss)   }
  .rel.plt        : { *(.rel.plt)    }
  .rela.plt       : { *(.rela.plt)   }
  .init           : { *(.init) } =0
  .text 0xBF800000 :
  {
    _ftext = . ;
    *(.text)
    *(.stub)
```

IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers skilled in the art designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only for development of an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising out of your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Rev.1.0 Mar 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.