Customizable Flash Programmer User Guide

Online Version
Send Feedback

UG-20198

683271
2023.04.28

Contents

1. Customizable Flash Programmer Overview

The Customizable Flash Programmer is a tool that programs the FPGA configuration bitstream (Raw Programming Data File (`.rpd`)) into the third-party flash devices. The Customizable Flash Programmer tool is a System Console tool that uses the soft design (Programmer Image) to program the flash device.

1.1. Device Family Support

The Customizable Flash Programmer tool supports the following devices:

- Cyclone® IV
- Cyclone V
- Intel® Cyclone 10 LP
- Intel Cyclone 10 GX
- Arria® II
- Arria V
- Intel Arria 10
- Stratix® IV
- Stratix V

This tool does not support the Intel Stratix 10 devices because the Intel Stratix 10 devices have a different configuration architecture.

1.2. Software Support

The Intel Quartus® Prime Programmer tool has limited support for third-party flash programming. The support is as follows:

- Version 18.0 and earlier—Support only the Intel configuration devices.
- Version 18.1—Limited support for third-party flash programming.

For third-party flash programming that is not supported by the Intel Quartus Prime Programmer, use the Customizable Flash Programmer tool.

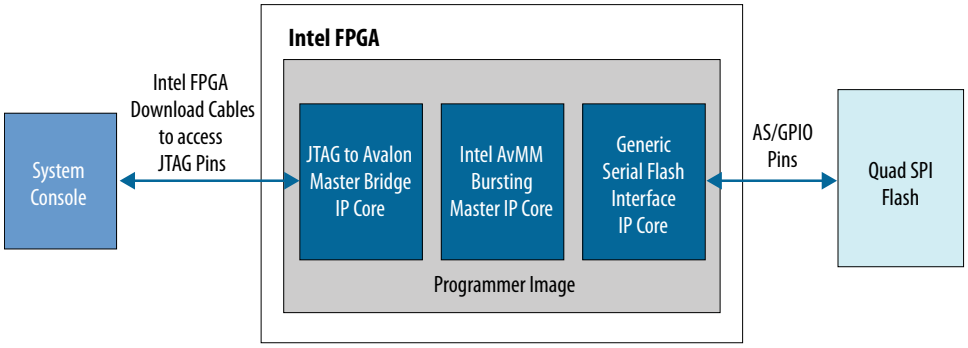
The Programmer Image provided in the Design Store is developed using the Intel Quartus Prime Standard Edition software version 18.0. You need to compile the Programmer Image using the Intel Quartus Prime version 18.0 or later. Intel recommends using the Intel Quartus Prime Standard Edition software because the design example is developed using the Standard Edition.

Related Information

Configuration Devices
Lists the supported third-party flash devices.

2. Customizable Flash Programmer

Figure 1. Customizable Flash Programmer Block Diagram



You can download the following files for the Customizable Flash Programmer from the Design Store:

- Programmer Image—The soft logic to be programmed into the FPGA
- System Console TCL scripts (`customizable_programmer.tcl`)—The script to run the programmer

Related Information

- Customizable Flash Programmer Files for Intel Arria 10 Devices
  Provides the Programmer Image and System Console TCL scripts for the Customizable Flash Programmer tool.
- Customizable Flash Programmer Files for Cyclone V Devices
  Provides the Programmer Image and System Console TCL scripts for the Customizable Flash Programmer tool.

2.1. Programmer Image

The Programmer Image is a simple Intel Quartus Prime project that consists of the following IP cores:

- JTAG to Avalon® Master Bridge Intel FPGA IP core
- Intel AvMM Bursting Master IP core (Custom IP)
- Generic Serial Flash Interface Intel FPGA IP core

JTAG to Avalon Master Bridge IP Core

This IP core allows you to send Avalon Memory-Mapped (Avalon-MM) instruction from the System Console to access the Avalon-MM registers in the design. You can find this standard IP core in the Intel Quartus Prime IP Catalogue.

Intel AvMM Bursting Master IP core

This IP core is a custom IP core developed to trigger burst write to or burst read from the Generic Serial Flash Interface IP core.

Generic Serial Flash Interface IP Core

This IP core is only available in the Intel Quartus Prime Standard Edition and Intel Quartus Prime Pro Edition software version 18.0 and later. You can instantiate this IP core from the Platform Designer IP Catalogue. This IP core is flexible to execute any flash command.

Related Information

Generic Serial Flash Interface Intel FPGA IP Core User Guide

2.2. System Console TCL Script

The TCL script is running in the System Console environment. The script reads the `.rpd` file and sends the file to the Programmer Image via JTAG port to program the quad serial peripheral interface (quad SPI) flash.

This script accesses the Avalon-MM registers in the Programmer Image. This script has many sub-functions such as read flash device ID, read status register, and write status register.

Related Information

- Register Map, Generic Serial Flash Interface Intel FPGA IP Core User Guide
  Provides more information on the Avalon-MM register map.
- Analyzing and Debugging Designs with System Console, Intel Quartus Prime Pro Edition User Guide: Debug Tools
  Provides more information on the System Console.

2.3. Limitations and Restrictions

There are some known limitations and restrictions to use the Customizable Flash Programmer tool.

- Intel Arria 10 and Intel Cyclone 10 GX devices only support 4-byte addressing and 10 dummy clock cycles for single and quad fast read commands. For workaround, generate the `.rpd` file using the EPCQL devices and program the `.rpd` file to the flash using the System Console TCL script.
- The third-party flash must support the following commands:

Table 1. Required Commands for Third-Party Flash

Command	Opcode
Normal Read	0x03
Single IO Fast Read	0x0B
Quad IO Fast Read	0xEB
Read Status Register	0x05

3. Implementing the Customizable Flash Programmer

To implement the Customizable Flash Programmer, follow these steps:

1. Generate the Raw Programming Data File (`.rpd`) from the design that you want to program into the flash device.
   a. Understand quad SPI flash byte-addressing.
   b. Understand dummy clock cycles for Single and Quad IO Fast Read in your flash device.
   c. Generate the `.rpd` file using the Convert Programming File tool.
2. Compile the Programmer Image project and generate the SRAM Object File (`.sof`).
   a. Download the project from the Design Store.
   b. Compile the Programmer Image project.
      i. Select the target FPGA.
      ii. Assign the clock pin accordingly if needed. Certain FPGAs, such as Stratix IV, require external clock.
3. Customize the TCL script for the flash device selected.

   Certain flash registers may need to be programmed. For example: setting power up 4-byte addressing mode, dummy clock cycles, and quad enable.
4. Program the flash using the System Console.
   a. Configure the Programmer Image `.sof` file into FPGA using the Intel Quartus Prime Programmer.
   b. Execute the TCL script and program the flash using the System Console.

3.1. Generating the Raw Programming Data File (.rpd) from User Design

The `.rpd` file is the configuration bitstream to be programmed into the quad SPI flash for active serial (AS) configuration. The FPGA reads the configuration bitstream from quad SPI after power-on-reset (POR).

Note: Generate the `.rpd` file from the your design. Do not generate the `.rpd` file from the Programmer Image.

3.1.1. Understanding Quad SPI Flash Byte-Addressing

The flash devices usually support one or both of the following byte-addressing modes:

- 3-byte addressing
- 4-byte addressing

Note: Refer to the third-party quad SPI flash datasheet for the byte-addressing modes supported for your flash devices.

The flash device reads either 24-bit (3-byte) address or 32-bit (4-byte) address before the flash device starts taking data to write to the flash memory, or output the data if the flash device receives a read command.

Figure 2. Reading Configuration Data from Flash with 3-Byte and 4-Byte Addressing

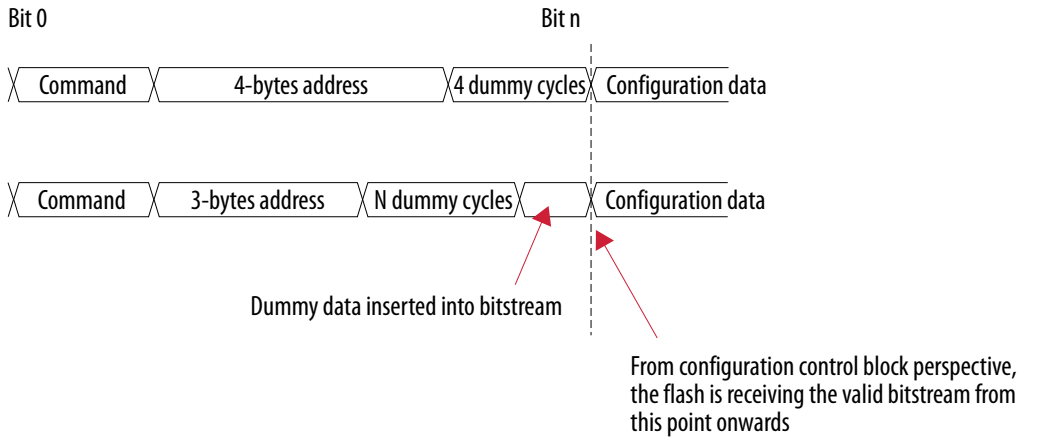


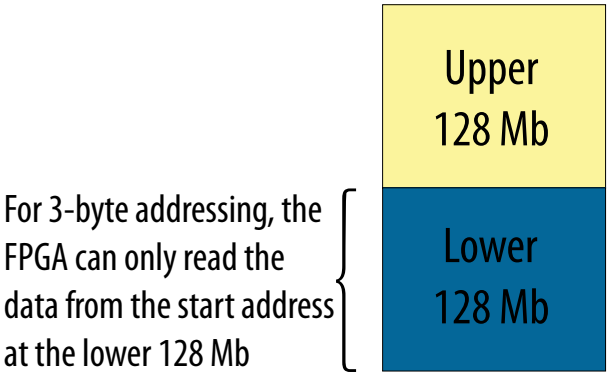
Table 2. Byte-Addressing Requirement for Intel FPGAs

FPGA Devices	Required Power Up Byte-Addressing of the Flash Devices
Legacy device older than 28nm devices, Intel Cyclone 10 LP	3-byte addressing
Cyclone V, Arria V, Stratix V	• 3-byte addressing (limited memory address access) • 4-byte addressing
Intel Arria 10, Intel Cyclone 10 GX	4-byte addressing

For example, if your flash device does not support power up 4-byte addressing mode, you cannot use your flash device for the Intel Arria 10 and Intel Cyclone 10 GX configuration.

Flash devices with density more than 128 megabits (Mb) require 4-byte address to access the memory space higher than 128 Mb. For flash devices that do not support non-volatile 4-byte addressing setting, the FPGA is unable to read the configuration image that has the start address beyond 128 Mb and unable to store image beyond 128Mb for Remote System Update application.

Figure 3. Memory Space of a 256 Mb Flash



3.1.2. Understanding Dummy Clock Cycles for Single and Quad IO Fast Read

Table 3. Flash Commands for Configuration Control Block

Command	Opcode	Dummy Clock Cycles Dependency
Normal Read	0x03	No
Single IO Fast Read	0x0B	Yes
Quad IO Fast Read	0xEB	Yes

Third-party quad SPI flash devices may have fixed or configurable dummy clock cycles. Determine the number of dummy clock cycles for the 0x0B and 0xEB commands from the third-party quad SPI flash datasheet for you to generate the `.rpd` file for your design.

3.1.3. Generating the `.rpd` File Using the Convert Programming File Tool

3.1.3.1. Generating `.rpd` File for Intel Arria 10 and Intel Cyclone 10 GX Devices

To generate the `.rpd` file for Intel Arria 10 and Intel Cyclone 10 GX devices using the Convert Programming File tool, follow these steps:

1. Specify the programming file type: **POF**
2. Select the configuration device.
   - If the flash density is 256 Mb, select EPCQL256.
   - If the flash density is 512 Mb, select EPCQL512.
   - If the flash density is 1024 Mb, select EPCQL1024.
3. Select mode: **Active Serial x1** or **Active Serial x4**.
4. For Advanced Options, check **Advanced... ➤ Disable EPCS/EPCQ ID check**.
5. Check **Create config data RPD (Generate output_file_auto.rpd)**.
6. Under **Input files to convert**, select the `.sof` file for your design.

Intel recommends enabling compression for the `.sof` file to reduce the `.rpd` file size.

7. Click **Generate**.

The generated <*output_file*>`_auto.rpd` file is the configuration bitstream to be programmed into the quad SPI flash using the Customizable Flash Programmer. The generated Programmer Object File (`.pof`) can be ignored.

3.1.3.2. Generating `.rpd` File for Other FPGA Devices

To generate the `.rpd` file for other FPGA devices, follow these steps:

1. Create `quartus.ini` in any of the following folders:
   - Quartus project folder
   - For Windows: `%QUARTUS_ROOTDIR%\bin64`
   - For Linux: `$QUARTUS_ROOTDIR/linux64`
2. Fill in the `quartus.ini` directive according to the flash byte-addressing mode, regardless of the flash density.

Quad SPI Flash Byte-Addressing Mode	`quartus.ini` Directive
3-byte	PGM_ALLOW_QSPI128=ON
4-byte	PGM_ALLOW_QSPI512=ON

3. Set the following settings in the Convert Programming File tool.
   a. Specify the programming file type: **POF**
   b. Select the configuration device.
      - Select QSPI128 for 3-byte addressing.
      - Select QSPI512 for 4-byte addressing.
   c. Select mode: **Active Serial x1** or **Active Serial x4**.
   d. For Advanced Options:
      i. Check **Advanced... ➤ Disable EPCS/EPCQ ID check**.
      ii. Fill up the dummy clock for quad SPI flash single IO and quad IO modes. Refer to the third-party quad SPI flash datasheet for this information.
   e. Check **Create config data RPD (Generate output_file_auto.rpd)**.
   f. Under **Input files to convert**, select the `.sof` file for your design.

      Intel recommends enabling compression for the `.sof` file to reduce the `.rpd` file size.
   g. Click **Generate**.

The generated <*output_file*>`_auto.rpd` file is the configuration bitstream to be programmed into the quad SPI flash using the Customizable Flash Programmer. The generated `.pof` file can be ignored.

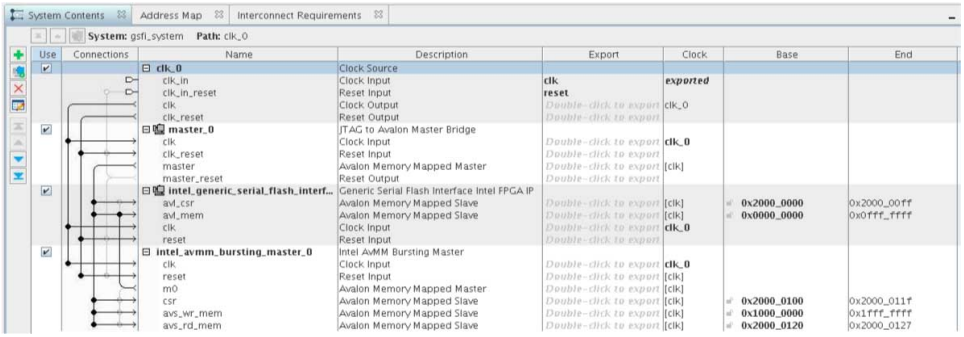
3.2. Compiling the Programmer Image Project and Generating the .sof File

For you to send command from the System Console to the FPGA soft logic, configure the FPGA using the Programmer Image. The Generic Serial Flash Interface IP core then generates quad SPI command to access the quad SPI flash via active serial (AS) pins or general-purpose I/O (GPIO) pins.

The Programmer Image project consists of only three components:

- JTAG to Avalon Master Bridge Intel FPGA IP core
- Intel AvMM Bursting Master IP core (Custom IP)
- Generic Serial Flash Interface Intel FPGA IP core

Figure 4. Programmer Image Project



3.2.1. Configuring the IP Cores Setting

3.2.1.1. Configuring JTAG to Avalon Master Bridge IP Core

The JTAG to Avalon Master Bridge IP core allows you to send Avalon-MM instruction from the System Console to access the Avalon-MM registers in the design. You can find this standard IP core in the Intel Quartus Prime IP Catalogue. You may instantiate this IP core without changing the default settings of this IP core.

3.2.1.2. Configuring Intel AvMM Bursting Master IP Core

The Intel AvMM Bursting Master IP core is a custom IP core developed to trigger burst write to or burst read from the Generic Serial Flash Interface IP core.

The following files are located in the Intel Quartus Prime project folder:

- `intel_avmm_bursting_master_hw.tcl`
- `intel_avmm_bursting_master.sv`
- `altera_avalon_sc_fifo_export_fill_lel.sv`

Ensure the three files are located in the project folder correctly. Instantiate this IP core in the Platform Designer environment. You can find the IP core in the IP Catalog: **Project ➤ Peripherals ➤ Debug and Performance ➤ Intel AvMM Bursting Master**.

Table 4. Recommended Settings for Intel AvMM Bursting Master IP Core

Properties	Setting	Description
General Master Properties		
Master Address Width	28 – 256M Bytes	Support up to 2 Gb flash density.
Master Data Width	—	This setting is not applicable.
byteenable Width	—	This setting is not applicable.
Master is Pipelined?	—	This setting is not applicable.
Burst Specific Properties		
burstcount Width	7 – 64 Words	Page Write usually takes up to 256 bytes or 64 words per transaction., Certain flash devices may accept more bytes per Page Write transaction.
Master Users Bursts?	BURST	Set to burst mode.

3.2.1.3. Configuring Generic Serial Flash Interface IP Core

The Generic Serial Flash Interface IP core is only available in the Intel Quartus Prime Standard Edition and Intel Quartus Prime Pro Edition software version 18.0 and later. You can instantiate this IP core from the Platform Designer IP Catalogue.

Intel recommends setting the `Device Density (Mb)` to **2048** even if you are using a lower device density. This setting may support any density lower than 2048 Mb.

Note: If the value for the `Device Density (Mb)` is changed, the base address of your Avalon-MM register maps deviates from the default base address defined in the TCL script. If you need to change to other device density, you need to match the base address defined in the TCL script to the Platform Designer Avalon-MM base address.

You can set the `Number of Chip Selects used` up to **3**. Only Intel Arria 10 and Intel Cyclone 10 GX support cascaded flash. If you set this value to 3 for other FPGAs, ensure the first chip is always selected during flash access.

3.2.2. Assigning External Clock Source for the IP Cores

The reference design utilizes the internal oscillator as the clock source. If you want to use an external clock as the source, you need to assign the external clock pin in Pin Assignment Editor accordingly.

For certain devices such as Stratix IV, you cannot use the internal oscillator as the clock source. You must use an external clock as the source and you need to assign the external clock pin accordingly.

3.2.3. Generating the Programmer Image `.sof` File from the Reference Project

To generate the `.sof` file, follow these steps:

1. Open the reference project using the Intel Quartus Prime software.

The reference project is created using the Intel Quartus Prime Standard Edition software version 18.0. Intel recommends opening this project using the same version of the Intel Quartus Prime software.

2. Select the target device based on the device used on the board.
3. Configure the IP core setting in Platform Designer. This step is optional.
4. Regenerate HDL in the Platform Designer.
5. Assign the clock pin if you use an external clock as the clock source in your design.
6. Recompile the project to generate the `.sof` file.

The generated `.sof` file is used to configure the FPGA. You may run the TCL script in the System Console to program the `.rpd` file into the quad SPI flash for FPGA configuration.

3.3. Customizing the TCL Script

The Customizable Flash Programmer uses the System Console framework to send commands to the FPGA that instantiate JTAG to Avalon Master Bridge IP core.

You can program the `.rpd` file to any third-party quad SPI flash using the TCL script. You can customize the TCL script to perform any quad SPI flash command, such as to program the flash registers to set the 4-byte addressing, dummy clock cycles, and quad enable (QE) bit.

Some example functions are created in the TCL script. You can use the TCL script to access the Generic Serial Flash Interface IP core Avalon-MM registers to carry out certain flash transactions, such as read Status Register, sector erase, and read Device ID. If your design uses the Generic Serial Flash Interface IP core, you can use this script as a reference to enable your design to access the quad SPI flash.

3.3.1. TCL Script Structures

The TCL script is designed to be extensible. The script is consisting of the following:

- Main script (`customizable_programmer.tcl`)—This script contains the global variables, common functions, and user settings. You can execute this script using the System Console to program the `.rpd` file to the flash.
- Extended script—This script stores the third-party flash commands. You may duplicate the sample script for other third-party flash devices. For example, programming NVCR for Micron flash and storing all Micron flash-specific functions using the `micon_mt25q.tcl` script. The `cypress_s25fs_s.tcl` script stores S25FS-S flash-specific functions.

Some examples of calling flash-specific functions are as follows:

- Source the extended script (micron_mt25q.tcl) in the main script (`customizable_programmer.tcl`).

Figure 5. Example of Sourcing an Extended Script in the Main Script

```
# source the flash specific instruction, use 1 at a time only
source micron_mt25q.tcl
#source cypress.tcl
```

Figure 6. Example of Flash-Specific Commands in the Extended Script

```
namespace eval micron_mt25q {

    proc clear_flag_status_register {} {
        global mp flash_cmd_ctrl flash_cmd_write_data_0
        set_flash_cmd_setting 0x00000050
        master_write_32 $mp $flash_cmd_ctrl 0x1
    }
```

- Call the `clear_flag_status_register` function in the main script using the following command.

```
micron_mt25q::clear_flag_status_register
```

3.3.2. TCL Script User Setting

You may use any text editor to edit the TCL script under the User Setting section.

Table 5. TCL Script User Setting

Setting	Options	Description
`chip_select`	0, 1, 2	Specify the access to which flash attached to the FPGA. Only applicable to Intel Arria 10 and Intel Cyclone 10 GX. For other FPGA devices, always set to 0.
`byte_addressing`	0, 1	Select the flash byte-addressing mode. • 0 = 3-byte addressing • 1 = 4-byte addressing
`dummy_clock`	N	Specify the number of dummy clock cycles for the flash commands. Refer to the third-party quad SPI flash datasheet.
`baud_rate`	1 – 16	Specify the IP clock divisor for the quad SPI clock. • 1 = div2 • 2 = div4 • 3 = div6 • 16 = div32 For example, if `baud_rate` = 1, the IP clock is 100 MHz, the quad SPI clock is 50 MHz.
`cs_assert_delay`	N	Specify the delay clock cycles before asserting nCS to initiate the flash command.

continued...

Setting	Options	Description
`read_opcode`	flash read opcode	Enter the flash read opcode. Refer to the third-party quad SPI flash datasheet for flash read opcode.
`write_opcode`	flash write opcode	Enter the flash write opcode. Refer to the third-party quad SPI flash datasheet for flash write opcode.
`polling_opcode`	0x05	Applicable to all flash devices except Micron flash device.
	0x70	Only applicable to Micron flash device.
`verify_during_programming`	0, 1	Verify the written data on the fly. Intel recommends setting `verify_during_programming` = 1 for first time programming to ensure the script works. For subsequent programming, set `verify_during_programming` = 0 to reduce programming time.
`burst_auto`	0, 1	Set `burst_auto` = 1 to improve programming time. Set `burst_auto` = 0 if burst auto mode is unstable.
`source <extended_script>.tcl`	*<Extended TCL script file>*	Source for extended TCL script. For example, to source for Micron-specific flash commands in an extended TCL scripts, use this command: `source micron_mt25q.tcl`.

3.3.3. Customizing the Programming Flow

The standard programming flow is as follows:

1. Read the `.rpd` file.
2. Write the `.rpd` file to the flash with either `write_rpd_data_auto` or `write_rpd_data` function depending on the `burst_auto` setting.

The `write_rpd_data_auto` function sends >256 bytes continuously to the Intel AvMM Bursting Master IP core. The Intel AvMM Bursting Master IP core handles the incoming bitstream automatically and performs quad SPI Page Write transactions with 256 bytes per transaction until the bitstream is fully programmed into the flash.

The `write_rpd_data` function sends exactly 256 bytes to the Intel AvMM Bursting Master IP core. This function triggers the Intel AvMM Bursting Master IP core to perform quad SPI Page Write of 256 bytes.

Figure 7. Programming Flow of the TCL Script

```
proc programming_flow { rpd_file {offset 0}} {
    global burst_auto
    read_rpd_file $rpd_file
    if {$burst_auto} {
        write_rpd_data_auto $rpd_file $offset
    } else {
        write_rpd_data $rpd_file $offset
    }
}
```

You can customize the programming flow by editing the TCL script.

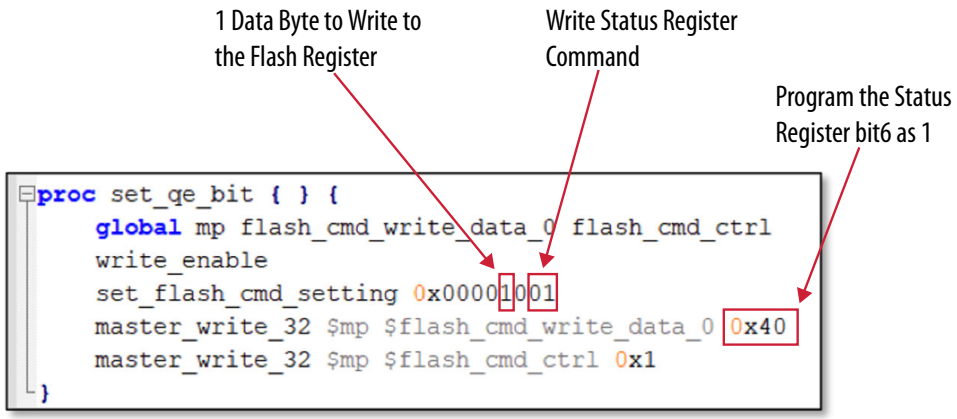
Figure 8. Example of a Customized Programming Flow

```
proc programming_flow { rpd_file {offset 0}} {
    global burst_auto
    read_rpd_file $rpd_file

    micron_mt25q::write_nonvolatile_config_register 0xafee

    if {$burst_auto} {
        write_rpd_data_auto $rpd_file $offset
    } else {
        write_rpd_data $rpd_file $offset
    }
}
```

3.3.4. Creating Custom Functions

You may need to create the custom functions to carry out certain flash transactions.

Figure 9. Example of Custom Function to Program Macronix Flash Quad Enable (QE) Bit to Enable Quad IO



Related Information

Register Map, Generic Serial Flash Interface Intel FPGA IP Core User Guide
Provides more details about the register map.

3.4. Programming the Flash Using the System Console

You can program the `.rpd` file using the following methods:

- Launch the System Console and execute the commands in the TCL Console window.
- Execute the system-console commands in the Nios® II Command Shell.

3.4.1. Executing Commands in the System Console

Ensure the Programmer Image `.sof` file is configured into the FPGA to execute the System Console command correctly.

To execute the commands in the System Console to program the `.rpd` file, follow these steps:

1. Launch the Nios II Command Shell.
2. Change the directory to your working folder where you save the TCL script and `.rpd` file. For example: `cd C:\My_Projects`.
3. Launch the System Console by executing this command: `system-console`.
4. At the TCL Console window, load the TCL script by executing this command: `source customizable_programmer.tcl`.
5. Program the `.rpd` file by executing this command: `programming_flow <your_rpd_file>`.

In System Console environment, you can execute many other commands besides programming the `.rpd` file. You may execute other functions in the TCL script directly from the TCL Console window. For example, you can execute `read_memory_id` to read the flash device ID, or `read_status_reg` to read the Status Register of the flash. Any function written in the TCL script, including the custom functions in extended script, can be executed from the TCL Console window.

An example to run a custom function in the custom script is as follows:

```
micron_mt25q::read_flag_status_register
```

3.4.2. Executing the System-Console Commands in the Nios II Command Shell

To execute the system-console commands to program the `.rpd` file, follow these steps:

1. Launch the Nios II Command Shell.
2. Change the directory to your working folder where you save the TCL script and `.rpd` file. For example: `cd C:\My_Projects`.
3. Program the `.rpd` file using this command: `system-console --script=customizable_programmer.tcl <your_rpd_file>`.

4. Document Revision History for the Customizable Flash Programmer User Guide

Document Version	Changes
2023.04.28	Updated `polling_opcode` row in the table in the *TCL Script User Setting* section to replace 0x07 with 0x70.
2018.11.28	Initial release.