

Introduction

MAX[®] II devices support the real-time in-system programmability (ISP) feature that allows you to program the device while it is still in operation. This feature enables you to perform in-field updates to the MAX II device at any time without affecting system operation.

MAX II devices also allow you to control when a new design should replace the existing design and begin functioning. The new design replaces the existing design only when there is either a power cycle to the device (powering down and powering up again) or with the execution of certain ISP instructions to start the SRAM download process when real-time ISP has completed. The flash to SRAM download process clears all device registers of any previously stored values and by default tri-states the I/O pins during the process.

This application note, together with design examples, describes how to force the SRAM download through the execution of ISP instructions, freeze the I/O pins during this process, and restore the register data states before the new design takes over.



Refer to the *Real-Time ISP & ISP Clamp for MAX II Devices* chapter in the *MAX II Device Handbook* for more information on the MAX II real-time ISP feature.

Forcing SRAM Download

To force an SRAM download without a power cycle, you must put the device in programming mode (either ISP or ISP Clamp mode), and then immediately exit programming mode and return to user mode. The SRAM download starts when the device exits (non real-time ISP) programming mode. If you execute a real-time ISP update, the new design is only loaded into the SRAM after the Joint Test Action Group (JTAG) commands execute for ISP or ISP Clamp entry and exit. The time required for the SRAM download is the same as that specified for the t_{CONFIG} power-up timing specification, from 200 to 450 μs depending on device density (refer to the *DC & Switching Characteristics* chapter in the *MAX II Device Handbook*).

You must use the JTAG interface to shift in the appropriate instructions to bring the device into programming mode or back to user mode. Using the JTAG interface also allows you to control the Test Access Port (TAP) controller state machine. The JTAG interface requires the use of the JTAG pins (TCK, TMS, TDI, and TDO) for communication with the device. The

easiest way to force the SRAM download is with a special set of Jam Standard Test and Programming Language (STAPL) commands that execute the short instruction sequence needed. This application note explains how you can use a Jam STAPL file to do this.



For more information on the MAX II JTAG interface and the TAP controller state machine, refer to the *IEEE 1149.1(JTAG) Boundary-Scan Testing for MAX II Devices* chapter in the *MAX II Device Handbook*.

I/O Clamping During the SRAM Download

The MAX II ISP Clamp feature allows you to clamp the I/O pins to specific states when the device is in ISP mode. You can either clamp the pins to high, low, or tri-state, or sample the state of the pins before the device enters ISP and then clamp the pins to those sampled states when the device is in ISP mode.

Using the ISP Clamp feature to sample and clamp the I/O pin state as the device enters ISP mode provides a glitch-free transition between user mode and ISP. You can use this same process to control the I/O states during the forced SRAM download procedure. The `ISP_ENABLE_CLAMP` instruction, followed by the `ISP_DISABLE` instruction, clamps the I/O pins to the values in the output register of the Boundary Scan Register. This same sequence forces the flash to SRAM download, or refresh. At the very completion of SRAM download, the outputs are still clamped while input pins begin to function. Using this procedure, you have a way to not only control I/O states during the SRAM download process, but also a way to create signals to set register values within the device to a non-power-up state. This procedure is explained in the subsequent design examples.



During configuration output pins default to the following settings: maximum drive strength, slow slew rate, and weak pull-up resistor enabled. These settings are in effect from the time the `ISP_ENABLE_CLAMP` instruction is issued until t_{CONFIG} amount of time after the `ISP_DISABLE` instruction is issued. If your design uses minimum drive strength and/or a fast slew-rate setting on output pins, the output drive characteristics will differ during the SRAM download process. Also, be aware that tri-stated pins become tri-stated with weak pull-up resistors during the SRAM download process.

Capturing & Retaining the Register Data

By default, the SRAM download process clears all the registers in the device. If you need to retain register data, you can store it just before the SRAM download starts, and then reload it back when the SRAM download is complete.

Design Update with UFM Storage Example

The MAX II user flash memory (UFM) allows you to store the register data before the SRAM download starts. The SRAM download process does not clear the UFM during SRAM download. After the SRAM download, you can read the data back from the UFM and reload the registers.

Instead of UFM storage, you can also use the MAX II ISP Clamp feature to retain output register data. ISP Clamp releases the input pins and enables the design to function, before releasing the output pins. You can use the input pins to sample the clamp state of the output pins and reload the registers before releasing the output pins. This application note explains how you can retain the register data with the two methods.

This SRAM download example demonstrates how you can control the I/O pin state and retain the register data with the UFM block. The design example consists of two parts:

- The Jam file
- The Quartus® II design example

You can modify the Jam file and design or create a new design based on this to suit your own requirements.

The Jam File

The Jam file contains the ISP and JTAG instructions that are shifted into the device to force the device into certain modes: user mode, ISP Clamp mode, or Real-Time ISP mode. The file also controls the TAP controller state machine.

The Quartus II software programmer cannot execute this custom Jam file. To execute the Jam file, you need the Jam STAPL player to communicate with the device through the JTAG interface. You can download the Jam STAPL player from www.altera.com.

To execute the Jam file, do the following:

- At the directory of your Jam file, type the following at the command-line prompt:

```
jam -adownload download.jam
```

Where `jam` is the Jam player executable, `download` is the Jam STAPL action, and `download.jam` is the Jam file that contains the instructions.

Table 1 describes the ISP and JTAG instructions in the Jam file.

Table 1. ISP & JTAG Instructions		
Instruction Name	Instruction Code	Description
SAMPLE_PRELOAD	00 0000 0101	Captures the current state of the I/O pins so that the device can clamp the pins to their respective states when it enters ISP Clamp mode.
ISP_ENABLE_CLAMP	10 0011 0011	Forces the device to enter ISP Clamp mode.
ISP_DISABLE	10 0000 0001	Forces the device to exit ISP or ISP Clamp to start the SRAM download.
RT_ISP_ENABLE	01 1001 1001	Forces the device to enter real-time ISP mode and asserts the UFM real-time ISP busy (<code>rtpbusy</code>) signal as a trigger signal to store the register data in the UFM. This signal also acts as the indicator for the design to reload the data into the registers.
RT_ISP_DISABLE	01 0110 0110	Forces the device back to user mode from real-time ISP mode.

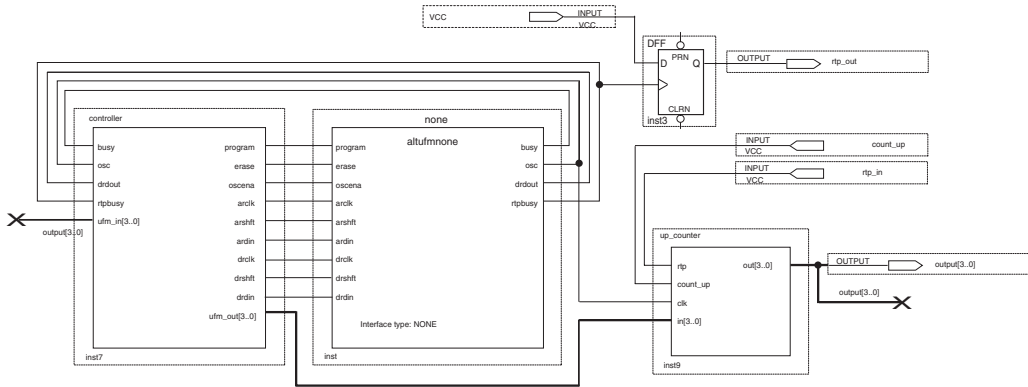
The Quartus II Design Example

The Quartus II design example consists of several modules:

- Controller
- `altufm_none` megafunction
- 4-bit up counter
- D flip-flop

These modules should be part of the new update design and the existing design. Figure 1 shows the block diagram of the Quartus II design.

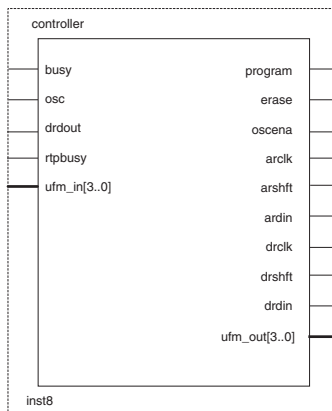
Figure 1. Design Example



Controller

The controller handles all interface operations with the UFM block for the read, write, and erase operations. Once the controller detects the `rtpbusy` signal, it captures the data from the 4-bit up-counter and stores the data in the UFM before the SRAM download operation starts. The 4-bit up-counter is an example piece of synchronous logic whose register values are captured and restored. The controller automatically retrieves the data back from the UFM for the counter after the SRAM download. [Figure 2](#) shows the controller module.

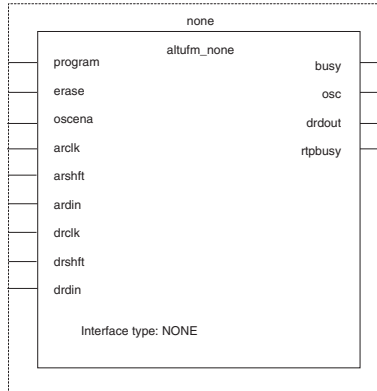
Figure 2. Controller Module



altufm_none Megafunction

The `altufm_none` megafunction (Figure 3) instantiates the MAX II UFM and interfaces the user logic and UFM block.

Figure 3. *altufm_none* Megafunction

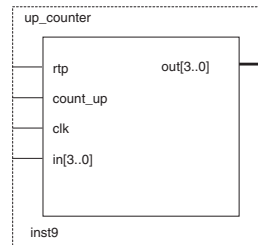


Refer to the *Using User Flash Memory in MAX II Devices* chapter in the *MAX II Device Handbook* for more information on the `altufm_none` megafunction.

4-Bit Up-Counter

The design uses a 4-bit up-counter to show how the SRAM download process can work without interrupting the function of the counter or without losing data. In your actual design you might have other registers that require data retention during the SRAM download process.

By default, the counter powers up to all 0's. The counter value increments by one with every rising edge to the `count_up` port. The `rtp` input pin connects to the output pin of the D flip-flop externally off-chip. The output pin of the D flip-flop indicates when the counter should load the data from the UFM through the `in[3..0]` port. Do not route the output signal from the D flip-flop to the `rtp` port of the counter internally. Figure 4 shows the 4-bit up-counter.

Figure 4. 4-Bit Up-Counter

D Flip-Flop

The design uses a D flip-flop to capture the `rtpbusy` signal from the `altufm_none` megafunction. An input pin drives to the D input of the D flip-flop. This pin is tied to the V_{CCIO} supply of the device externally. The D flip-flop clock input is connected to the `rtpbusy` signal of the `altufm_none` megafunction internally. The Q output of the flip-flop drives to an output pin that connects to the input pin of the `rtp` port of the 4-bit up-counter externally. This signal indicates when the counter should be loaded with the data from the UFM.

The `rtp` signal acts as an indicator of whether a normal power cycle has occurred (logic 0) or a real-time ISP update has occurred (logic 1). This allows you to control whether you have a power-up reset state on registers or a preset/non-power-up state on registers.

Design Update Procedure

This section explains the Jam file SRAM download procedure until the release of the counter output pins.



The actual real-time ISP should be completed earlier before you start the SRAM download process. The counter output pins are glitch-free during the entire process.

1. The device detects the trigger signal and stores the register data into the UFM.

The Jam file issues the `RT_ISP_ENABLE` instruction and puts the device in real-time ISP mode. The `rtpbusy` signal from the `altufm_none` megafunction triggers the controller to erase one of the UFM sectors before storing the data of the registers into one of the addresses in that UFM sector. You can specify the UFM address in the controller module. The design only erases the UFM sector of that address.

The `rtpbusy` signal also causes the output pin from the Q output of the D flip-flop to go high. The device clamps this output pin high until the SRAM download process completes and releases the pin together with other output pins. This signal serves as an indicator to the 4-bit up-counter that the counter must read the data from the controller module after the SRAM download, before reloading the registers.

2. The device exits real-time ISP mode.

The Jam file issues the `RT_ISP_DISABLE` instruction.

3. The device samples the state of the I/O pins.

The Jam file issues the `SAMPLE_PRELOAD` instruction to capture the states of the I/O pins (make sure that the pins do not change state during and after the process of capturing the pin state).

4. The device enters ISP Clamp mode.

The Jam file issues the `ISP_ENABLE_CLAMP` instruction and the device goes into ISP Clamp mode. Entering the ISP Clamp mode clamps the I/O pins to the states captured during the execution of the `SAMPLE_PRELOAD` instruction.

5. The device exits ISP Clamp mode.

The Jam file issues the `ISP_DISABLE` instruction and the device exits ISP Clamp mode. Exiting the ISP Clamp mode forces the SRAM download and clears all the registers in the device. The new design starts to function after the SRAM download, but all the output pins are still in the clamped states. The controller module reads back the data from the UFM. The counter detects the Q output signal from the D flip-flop and loads the registers with the data read out from the UFM by the controller.

6. The device releases output pins.

The Jam file forces the TAP controller state machine to the reset state and the device releases all output pins. This is a glitch-free transition and you can resume the counter operation from the value the counter had before the SRAM download.

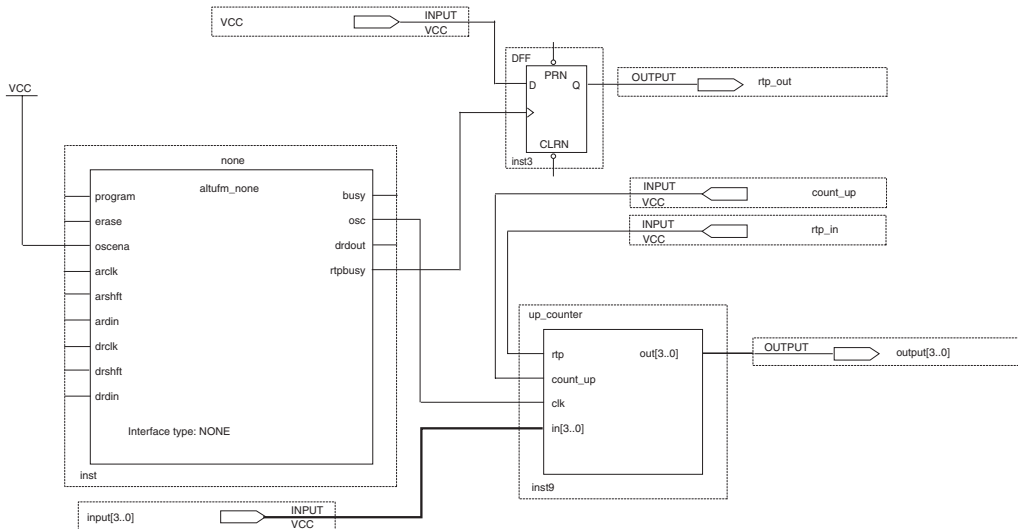
Non-UFM Design Update Alternatives

If you do not want to use the UFM to store register data before the SRAM download process, you can easily modify the existing design to do so. This section discusses two non-UFM design update alternatives.

Retaining Register Data with ISP Clamp & Input Pins

Instead of using the UFM to store the register data, you use the MAX II ISP Clamp feature to retain the register data. The user design in the logic array and input pins start to function as soon as the device exits ISP Clamp mode. However, the device still clamps the output pins until the TAP controller state machine reaches the reset state. This allows you to drive clamped output pins back into the device input pins. In essence, you are using clamped output pins as storage for register data that drove those output pins. Figure 5 shows the design schematic.


Figure 5. Design for Retaining Register Values without Using UFM



This design does not use the controller module because it does not require writing to or reading from the UFM, and therefore consumes less LE resources. You must still instantiate the `altufm_none` megafunction to access the `rtpbusy` signal as well as the `osc` clock. Connect the `oscena` port to `VCC` internally to always enable the `osc` clock.

As in the previous design shown in [Figure 1 on page 5](#), the D flip-flop output pin connects to the input pin for the `rtp` port of the 4-bit up-counter module. The D flip-flop input pin connects to the V_{CCIO} supply of the device externally. The clock input is connected to the `rtpbusy` signal from the `altufm_none` megafunction internally.

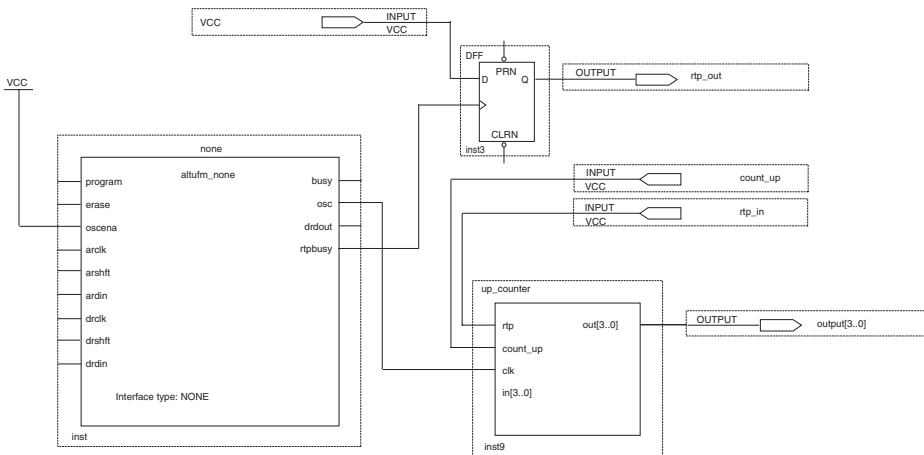
This design uses four additional input pins. The counter `in[3..0]` port connects to four input pins, and these pins connect to the output pins of the `out[3..0]` port externally. Do not route the output signals back to the `in[3..0]` port internally. This design might not be suitable if you need to store the data for a large number of registers because it consumes many I/O pins.

 You do not need to modify the Jam file for this design.

Loading Registers with Specified Values After SRAM Download

The design shown in [Figure 6](#) loads the registers with the preset values each time after an SRAM download. The design needs only minimal modification to the counter module for implementation. [Figure 6](#) shows the schematic for this design.

Figure 6. Design for Loading Registers with Specific Values



The preset values can be specified in the counter module. You can modify the Verilog code of the counter module by specifying the values in the `.v` file and then recompiling the design. The counter does not need the input port `in[3..0]`.

You can enter the preset values in the following part of the Verilog file for the counter module.

```
always @ (posedge clk)
begin
    if (count <= 999)
        begin
            count = count + 1;
            if (count == 1000 && rtp == 0)
                out = 4'b0; //this is the default
                    //register data when
                    //powered up
            else if (count == 1000 && rtp == 1)
                out = 4'b1111; //this is the preset data
                    //to be loaded into the
                    //registers after each
                    //SRAM download
        end
end
```



You do not need to modify the Jam file for this design.

Conclusion

This design example provides a solution that allows you to force the SRAM download after real-time ISP without the need for a power cycle. This design also provides a solution that allows you to retain the data of the registers in the device after going through the SRAM download process, either by using the UFM or input pins to retain the data. Together with the ISP Clamp feature, you have a glitch-free SRAM download process for the MAX II device.



101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
www.altera.com
Applications Hotline:
(800) 800-EPLD
Literature Services:
literature@altera.com

Copyright © 2006 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.



I.S. EN ISO 9001