AN 904: Intel® MAX® 10 Hitless Update Implementation Guidelines

Subscribe

Send Feedback

AN-904 | 2020.02.24

Latest document on the web: PDF | HTML

Contents

1. Intel® MAX® 10 Hitless Update Implementation Guidelines.. 3
1.1. Introduction..3
1.2. Types of Intel MAX 10 Design Update Flow..3
1.3. Stages of Intel MAX 10 Hitless Update Flow..4
1.4. Intel MAX 10 Hitless Update Requirement and Limitation...5
1.5. Hitless Update Through Intel Quartus Prime Programmer.. 5
1.5.1. Creating IPS File...6
1.5.2. Executing IPS File... 6
1.6. Hitless Update using Intel Jam STAPL Byte-Code Player.. 6
1.6.1. Generating .jam/.jbc File.. 7
1.6.2. Interpreting Jam File Information using Jam Player...................................... 7
1.7. Hitless Update Using Self-Developed Algorithm..8
1.8. Boundary-Scan Pattern Configuration Bit Location... 9
1.9. JTAG Instructions...10
1.10. Sample of Hitless Update Algorithm in Jam Format... 11
1.11. Document Revision History for AN 904: Intel MAX 10 Hitless Update Implementation Guidelines.. 12

1. Intel® MAX® 10 Hitless Update Implementation Guidelines

1.1. Introduction

Intel® MAX® 10 devices offer the hitless update feature, which provides you the capability and flexibility to control the state of the I/O pins during the internal flash image update and reconfiguration of a Intel MAX 10 device. All of the I/O pins can remain stable without any disruption throughout the hitless update process. This feature also allows the Intel MAX 10 device to behave as a system controller when monitoring and controlling critical signals without interruption.

These guidelines help you implement the hitless update through Intel Quartus® Prime Programmer or the Intel Jam™ Standard Test and Programming Language (STAPL) Byte-Code Player. These guidelines also provide the details of the hitless update flow to enable you to develop your own source code to implement the hitless update through your microprocessor or host controller.

1.2. Types of Intel MAX 10 Design Update Flow

The Intel MAX 10 FPGA provides various flexibility to perform updates on the Intel MAX 10 design. It enables you to select the correct configuration method that suits the needs of your system.

The following are the available configuration or programming methods:

- In-system programming: You can program the internal flash including the configuration flash memory (CFM) and user flash memory (UFM) of the Intel MAX 10 devices with In-System Programming (ISP) through industry standard IEEE 1149.1 JTAG interface. ISP offers the capability to program, erase, and verify the CFM and UFM. The JTAG circuitry and ISP instructions for Intel MAX 10 devices comply to the IEEE-1532-2002 programming specification. The device can be reconfigured to the new image from the updated internal flash through power cycle or pulse `nCONFIG` signal. The device is out of user mode during the entire ISP process and all I/O pins remain tri-stated.
- Real-time programming: The real-time ISP feature updates the internal flash with a new design image while the device is still operating in user mode. During the internal flash programming, the device continues to operate using the existing design. After the new design image programming process completes, the device does not reset. The new design image update takes effect in the next reconfiguration cycle.
- Remote system updates: With the remote system upgrade feature, enhancements and bug fixes for FPGA devices can be done remotely by updating the internal flash through the On-Chip Flash Intel FPGA IP core and perform reconfiguration from remote site through the Dual Configuration Intel FPGA IP core.

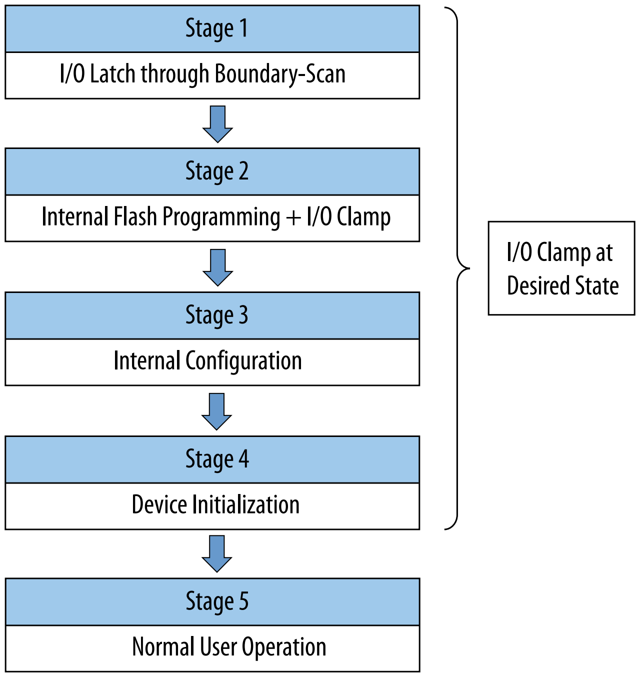
However, all the internal flash programming methods require you to perform reconfiguration to get the new design running on the device. During the reconfiguration process, all the I/O pins can be either tri-stated or weak pull-up, depending on Intel Quartus Prime settings. Consequently, your system may be impacted if the downstream device relies on the status of the I/O pins of the Intel MAX 10 devices.

1.3. Stages of Intel MAX 10 Hitless Update Flow

At a high level, the Intel MAX 10 hitless update flow may be categorized into five stages:

- Stage 1: I/O latch through boundary-scan. The I/O state is a setup based on a real time I/O state sampling or by shifting in predefined boundary-scan data.
- Stage 2: Internal flash programming and I/O clamp. The Intel MAX 10 internal flash (CFM and UFM) is programmed and updated while the design is still running, or the I/O pins clamped at a user-defined state. For example, you can store critical design registers or finite state machine (FSM) values and desired I/O pin states into the UFM before performing Stage 2.
- Stage 3: Internal configuration. The I/Os remain in the desired state while the configuration takes place from the internal flash into CRAM.
- Stage 4: Device initialization. The I/Os are released after a period of time that you define, after the internal configuration is completed. During the defined period, you can unload the I/O state data, register or FSM value that previously stored in the User Flash Memory, force the user design logic into a correct state to output the same desired I/O value as the clamping state, to ensure no disruption to the system.
- Stage 5: Normal user operation.

Figure 1. Stages of Intel MAX 10 Hitless Update



1.4. Intel MAX 10 Hitless Update Requirement and Limitation

The Intel MAX 10 hitless update must be performed through external JTAG pins. You should use either the Intel FPGA Download Cable connected to JTAG header or an external host or microprocessor to control the JTAG state machine when running the hitless update flow.

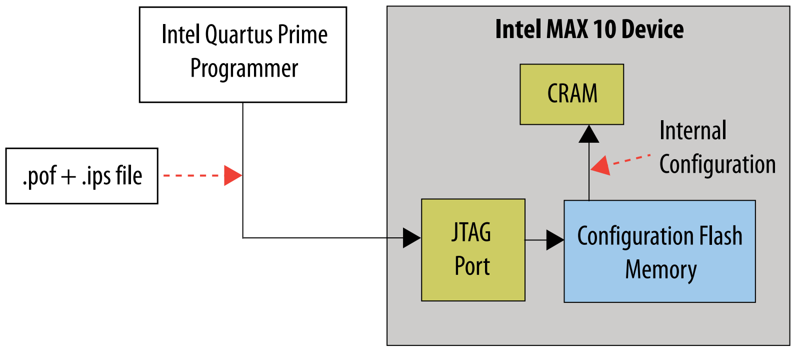
The states of all the output pins are clamped at a user-defined state and no input signal can be read during the clamping period. If you need to process some input signals during the clamping period (from internal configuration complete to clamp release), you can use the global clock pins as input pins in your design. The global clock pins can route the input signals into the core fabric even though the I/Os have not been released.

1.5. Hitless Update Through Intel Quartus Prime Programmer

The Intel Quartus Prime Programmer does the following functions:

- Reads a user-specified input `.ips` file to hold each I/O to the desired state
- Performs real-time programming operation to write the configuration bitstream into the internal flash of the Intel MAX 10 FPGA
- Performs I/O state clamping control and initiates the configuration process.

Figure 2. Implementing Hitless Update using Intel Quartus Prime Programmer



To implement the hitless update through the Intel Quartus Prime programmer, you must create and execute a pin state information (`.ips`) file.

Note: You can use the created `.ips` file to program the device with any design, provided that it targets the same device and package. You must use the `.ips` file together with a POF file.

1.5.1. Creating IPS File

To create an `.ips` file, perform the following steps:

1. Click **Programmer** on the toolbar, or on the **Tools** menu, click **Programmer** to open the **Programmer**.
2. Click **Add File** in the programmer to add the programming file (POF, Jam, or JBC).
3. Click on the programming file (the entire row will be highlighted) and on the **Edit** menu, click **ISP CLAMP State Editor**.
4. Specify the states of the pins in your design in the **ISP Clamp State Editor**. By default, all pins are set to **tri-state**.
5. Click **Save** to save IPS file after making the modifications.

1.5.2. Executing IPS File

To execute ISP Clamp, perform the following steps:

1. In the Intel Quartus Prime **Programmer**, select the `.pof` necessary to program to the device.
2. Select the `.pof`, right click and select **Add IPS File** and turn-on **ISP CLAMP**.

   Note: You can change the start-up delay of the I/O Clamp after configuration. To do this, select **Tools ➤ Options**, turn on the **Overwrite MAX10 configuration start up delay when using IO Clamp in Programmer** option, and change the delay value accordingly.
3. Select the `.pof` in the **Program/Configure** column.

   Note: For third party programming, you can generate the `.jam` or `.jbc` file from the `.pof` file with `.ips` file.
4. After you configure the required settings, click **Start** to start programming.

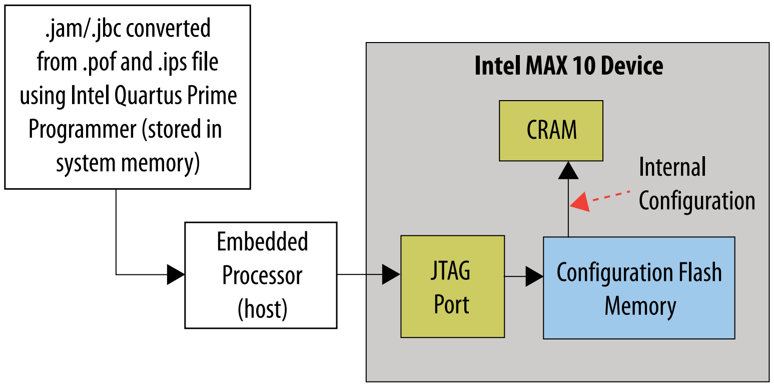
   Note: To enable the real-time ISP mode, turn on **Enable real-time ISP to allow background programming**.

1.6. Hitless Update using Intel Jam STAPL Byte-Code Player

The Jam Standard Test and Programming Language (STAPL) meets the necessary system requirements of in-system programming via an embedded processor, as it offers small file sizes, ease of use, and platform independence.

Using the Jam Standard Test and Programming Language (STAPL) for in-system programming via an embedded processor permits convenient in-field upgrades, easy design prototyping, and fast programming.

Figure 3. Implementing Hitless Update using Jam STAPL for In-System Programming via Embedded Processor


.jam/.jbc converted from .pof and .ips file using Intel Quartus Prime Programmer (stored in system memory)
Embedded Processor (host)
Intel MAX 10 Device
CRAM
Internal Configuration
JTAG Port
Configuration Flash Memory

The Jam or JBC player approach is critical for the application that does not allow the use of the Intel Quartus Prime Programmer. From the Intel Quartus Prime Programmer, you can convert the `.pof` together with the `.ips` file described in Hitless Update Through Intel Quartus Prime Programmer on page 5 to a `.jam` or `.jbc` file, so that the same hitless upgrade mechanism or algorithm from the Intel Quartus Prime Programmer can be integrated into the `.jam` or `.jbc` file. The Jam or JBC player let you run the hitless upgrade JTAG algorithm based on the input `.jam` or `.jbc` file.

To implement the hitless update through Intel Jam STAPL Byte-Code Player, you have to:

1. Generate the `.jam` or `.jbc` file. The `.jam` or `.jbc` file contains all information required to program the in-system programmability (ISP)-capable device(s), including the programming algorithm and the hitless update mechanism if you include the `.ips` file during file conversion.
2. Interpret JAM file information using JAM Player runs on the embedded processor.

1.6.1. Generating .jam/.jbc File

To generate the `.jam` or `.jbc` file, perform the following steps:

1. Add both `.pof` and `.ips` files in the Intel Quartus Prime Programmer, as described in the Hitless Update Through Intel Quartus Prime Programmer on page 5.
2. On the Intel Quartus Prime Programmer menu, select **File ➤ Create/Update ➤ Create Jam, SVF, or ISC File**.
3. In the **File Format** list, select the format you want to generate.

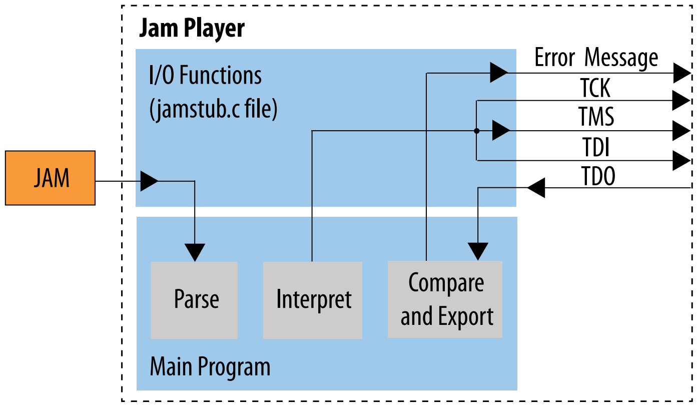
   Note: The generated file name does not indicate whether it was converted from a `.sof` or a `.pof` file. You should rename the generated file to avoid future confusion.

1.6.2. Interpreting Jam File Information using Jam Player

The Jam Player is a C program that parses the Jam file, interprets each Jam STAPL instruction, and reads and writes data to and from the JTAG chain.

The variables processed by the Jam Player depend on the initialization list variables present at the time of execution. Because each application has unique requirements, you can easily modify the Jam Player source code.

Figure 4. Jam Player Source Code Structure



Note: The JTAG I/O pins are TCK, TMS, TDI, and TDO.

The main program performs all of the basic functions of the Jam Player without modification. You must only modify the I/O functions. They are contained in the `jamstub.c` file, as shown in the figure above, and include functions which specify addresses to I/O pins, delay routines, operating system-specific functions, and routines for file I/O pins. You can customize these functions by simply editing the `jamstub.c` file then compile the source code for use on any platform.

The Jam Player resides permanently in system memory, where it interprets the commands given in the Jam file and generates a binary data stream for device programming. This structure confines all upgrades to the Jam file, and allows the Jam Player to adapt to any system architecture.

Related Information

- JAM Player Download Page
- Using JAM Player via an Embedded Processor

1.7. Hitless Update Using Self-Developed Algorithm

You can create your own hitless update driver or algorithm if you choose not to use the Intel Quartus Prime Programmer and the Jam player.

You can implement the hitless update in the Intel MAX 10 devices through JTAG Instructions by sampling the existing I/O states, editing the nSTATUS and CONF_DONE I/O bits, and loading it back to the boundary-scan cells. This allows the configuration flow can be well controlled and the I/O state can be retained across the entire configuration flow at the same time.

To implement the hitless update flow, perform the following steps:

Note: Ensure that the Intel MAX 10 device is in user mode.

1. Shift out the existing or desired I/O state from the boundary scan through the SAMPLE/PRELOAD JTAG instruction and edit the boundary-scan data (Pattern A). Shift in and update the boundary-scan register with pattern A through SAMPLE/PRELOAD.

   Note: Pattern A is equal to the pattern sampled in user mode, with exception that the `CONF_DONE OE` bit is set to 0 and the output bit is set to 0, as well as the `NSTATUS OE` bit is set to 0 and the output bit is set to 0 in order to drive the `NSTATUS` and `CONF_DONE` pins to 0.

2. Enter ISP mode with the ISP_ENABLE_CLAMP instruction, and stay in the RUNTEST IDLE state for at least 10 TCK pulses.
3. Perform an internal flash read or write operation through the `ISP_PROGRAM` and `ISP_READ` instructions.

   Note: Contact Intel Support to obtain the Programming Specification.

4. Disable ISP mode with the ISP_DISABLE instruction, and stay in the RUNTEST IDLE state for at least 10 TCK pulses.
5. Exit ISP mode with the EXTEST instruction, and stay in the RUNTEST IDLE state for at least 10 TCK pulses.
6. Shift in and update the boundary-scan register with Pattern B through SAMPLE/PRELOAD without issuing any instruction.

   Note: Pattern B is equal to pattern A, with exception that the `CONF_DONE OE` bit is set to 0 and the output bit is set to 0, as well as the `NSTATUS OE` bit is set to 1 in order to pull up the `NSTATUS` pin externally to 1.

7. Wait for device initialization and internal configuration (Refer to the *Internal Configuration Time for Intel MAX 10 Devices (Uncompressed .rbf)* and *Internal Configuration Time for Intel MAX 10 Devices (Compressed .rbf)* tables in the *Intel MAX 10 FPGA Device Datasheet* for internal configuration time).
8. Shift in and update boundary-scan register with Pattern C through SAMPLE/PRELOAD without issuing any instruction.

   Note: Pattern C is equal to pattern A, such that the `CONF_DONE OE` bit is set to 1, as well as the `NSTATUS OE` bit is set to 1 in order to pull up the `CONF_DONE` and `NSTATUS` pins externally to 1.

9. Wait for start-up (Refer to the *Internal Configuration Timing Parameter for Intel MAX 10 Devices* table in the *Intel MAX 10 FPGA Device Datasheet* for start-up time). Device enters user mode. The design core is running now but I/O state is still clamped.
10. Disable EXTEST with JTAG TAP RESET to release the I/O clamp. You can insert any amount of delay before issuing the EXTEST instruction to release the clamp.

Related Information

Intel MAX 10 FPGA Device Datasheet

1.8. Boundary-Scan Pattern Configuration Bit Location

The length of the boundary-scan pattern shifted out from the boundary-scan cell varies between different device densities. Hence, the position of configuration bits, such as nSTATUS and `CONF_DONE`, is different. In the boundary-scan data, each pin has three control bits to control the settings of the I/O pin:

- Input bit
- OE bit
- Output bit

Updating the configuration bits state within the boundary-scan data is a critical step to ensure that the entire hitless update process is works flawlessly. You must know the configuration bit position within the boundary-scan data based on the device used.

Table 1. Configuration Bit Position Based on Intel MAX 10 Device Densities

Device	Configuration Bit Position	Total Boundary-Scan Chain Bits
10M02	nSTATUS: Input bit (bit 21), OE bit (bit 22), Output bit (bit 23) CONF_DONE: Input bit (bit 12), OE bit (bit 13), Output bit (bit 14)	492
10M04	nSTATUS: Input bit (bit 21), OE bit (bit 22), Output bit (bit 23) CONF_DONE: Input bit (bit 12), OE bit (bit 13), Output bit (bit 14)	756
10M08	nSTATUS: Input bit (bit 21), OE bit (bit 22), Output bit (bit 23) CONF_DONE: Input bit (bit 12), OE bit (bit 13), Output bit (bit 14)	756
10M16	nSTATUS: Input bit (bit 21), OE bit (bit 22), Output bit (bit 23) CONF_DONE: Input bit (bit 12), OE bit (bit 13), Output bit (bit 14)	960
10M25	nSTATUS: Input bit (bit 21), OE bit (bit 22), Output bit (bit 23) CONF_DONE: Input bit (bit 12), OE bit (bit 13), Output bit (bit 14)	1140
10M40	nSTATUS: Input bit (bit 21), OE bit (bit 22), Output bit (bit 23) CONF_DONE: Input bit (bit 12), OE bit (bit 13), Output bit (bit 14)	1500
10M50	nSTATUS: Input bit (bit 21), OE bit (bit 22), Output bit (bit 23) CONF_DONE: Input bit (bit 12), OE bit (bit 13), Output bit (bit 14)	1500

1.9. JTAG Instructions

Table 2. JTAG Instructions

Instruction Name	Instruction Binary	Description
SAMPLE/ PRELOAD	00 0000 0101	• Permits an initial data pattern to be an output at the device pins. • Allows you to capture and examine a snapshot of signals at the device pins if the device is operating in normal mode.
EXTEST	00 0000 1111	• Forces test pattern at the output pins and capture the test results at the input pins. • Allows you to test the external circuitry and board-level interconnects.
BYPASS	11 1111 1111	• Places the 1-bit bypass register between the TDI and TDO pins. • Allows the BST data to pass synchronously through target devices to adjacent devices during normal device operation.
ISP_ENABLE_CLAMP	10 0011 0011	Enables ISP mode and forces all I/Os to follow the content of the JTAG boundary-scan register.
ISP_DISABLE	10 0000 0001	Disables ISP mode.

1.10. Sample of Hitless Update Algorithm in Jam Format

The following example shows the Jam Standard Test and Programming Language (STAPL) Format File (`.jam`) used to execute the hitless update algorithm.

The example below is based on an Intel MAX 10 10M50 device.

```
ACTION DOWNLOAD = FORCE_REFRESH RECOMMENDED;

DATA DEVICE_DATA;
BOOLEAN sampled[1500]; ‘1500 is the total boundary scan chain bit for 10M50 
device
BOOLEAN nstatus_0_confdone_0[1500];
BOOLEAN nstatus_1_confdone_0[1500];
BOOLEAN nstatus_1_confdone_1[1500];
ENDDATA;

PROCEDURE FORCE_REFRESH USES DEVICE_DATA;
'sample all pins state
IRSCAN     10, $005;
WAIT IDLE, 10 CYCLES, 1 USEC, IDLE;
DRSCAN 1500, $0, CAPTURE sampled[1499..0];
WAIT IDLE, 10 CYCLES, 25 USEC, IDLE;

'MAX10 10M50 boundary scan cell for nSTATUS: Input bit(bit 21) OE bit(bit 22) 
Output bit (bit 23)
'CONF_DONE: Input bit(bit 12) OE bit(bit 13) Output bit(bit 14)
'Setting nStatus=0, CONF_DONE=0 ... (Pattern A)
nstatus_0_confdone_0[1499..0] = sampled[1499..0];
nstatus_0_confdone_0[12] = 0;
nstatus_0_confdone_0[13] = 0;
nstatus_0_confdone_0[14] = 0;
nstatus_0_confdone_0[21] = 0;
nstatus_0_confdone_0[22] = 0;
nstatus_0_confdone_0[23] = 0;

'Setting nStatus=1, CONF_DONE=0 ... (Pattern B)
nstatus_1_confdone_0[1499..0] = sampled[1499..0];
nstatus_1_confdone_0[12] = 0;
nstatus_1_confdone_0[13] = 0;
nstatus_1_confdone_0[14] = 0;

'Keep nStatus=1, CONF_DONE=1 ... (Pattern C)
nstatus_1_confdone_1[1499..0] = sampled[1499..0];

'Preload sampled pin state with nStatus=0 and CONF_DONE=0 to boundary scan 
registers
IRSCAN     10, $005;
WAIT IDLE, 10 CYCLES, 1 USEC, IDLE;
DRSCAN 1500, nstatus_0_confdone_0[1499..0];
WAIT IDLE, 10 CYCLES, 25 USEC, IDLE;

'ISP_ENABLE_CLAMP
IRSCAN    10, $233;
WAIT     IDLE, 1000 CYCLES;

'''''User can perform In System Programming here

'ISP_DISABLE
IRSCAN     10, $201;
WAIT     IDLE, 1000 CYCLES;

'EXTEST
IRSCAN    10, $00F;
WAIT     IDLE, 1000 CYCLES;

'Preload sampled pin state with nStatus=1 and CONF_DONE=0 to boundary scan 
registers
DRSCAN 1500, nstatus_1_confdone_0[1499..0];
```

```
WAIT IDLE, 10 CYCLES, 25 USEC, IDLE;
WAIT     IDLE, 1000000 USEC;

'Preload sampled pin state with nStatus=1 and CONF_DONE=1 to boundary scan
registers
DRSCAN 1500, nstatus_1_confdone_1[1499..0];
WAIT IDLE, 10 CYCLES, 25 USEC, IDLE;
WAIT     IDLE, 4000000 USEC;              'Device already in user mode. User
can change any number of delay here before release the I/O

'TAP reset
STATE     RESET;                           'I/O release from clamp
ENDPROC;
```

1.11. Document Revision History for AN 904: Intel MAX 10 Hitless Update Implementation Guidelines

Document Version	Changes
2020.02.24	Restructured the document.
2020.02.20	Initial release.