EFINIX®

AN 038: Programming with an MCU and the JTAG Interface

AN038-v1.0
November 2021
www.efinixinc.com

Contents

Introduction

When designing your system, you may have a microcontroller or microprocessor (MCU) that controls a number of devices in a JTAG chain. To streamline your design, you would like to take advantage of the JTAG chain to program your Trion or Titanium FPGA. This scenario is a type of *passive configuration* in which the FPGA receives the configuration clock and data from an external active module.

Unlike SPI passive mode, this method only requires the 4-pin JTAG interface plus a few extra pins, depending on the FPGA you are targeting. The configuration data is transferred via JTAG, which is serial, so the configuration time can be slow. Therefore, you do not want to use this method when you need the FPGA to wake up "instantly." This application note descibes how to use a MCU to configure an FPGA via the JTAG interface.

Note: Before attempting the method described in this document, you should have a solid knowledge of JTAG commands and the JTAG TAP. This method is for expert users.

JTAG Commands for Programming

With this programming method, the MCU is acting as a JTAG programmer. It needs to send commands via the JTAG interface. The Efinity® software can export a bitstream file to serial vector format (**.svf**), which contains JTAG commands and bitstream data. Typically you would use the **.svf** to program the FPGA with a JTAG SVF player. However, in this context, the **.svf** provides a template for the JTAG commands the MCU needs to use for programming.

The following code snippet shows an example of a Trion T35 bitstream file converted to **.svf**.

1. The first 5 lines are preamble.
2. The `FREQUENCY` line sets the JTAG clock frequency, in this case it is the maximum.
3. The next 4 lines relate to the position of the device in the JTAG chain. This **.svf** assumes that the T35 is the only device in the chain.
4. The `SIR` and `SDR` lines after the `Check idcode` comment specify the `IDCODE` instruction and `IDCODE` data. You should always check the `IDCODE` before programming.
5. The `SIR` after the `Enter programming mode` comment issues the `PROGRAM` instruction.
6. The following `SDR` commands send the bitstream data.
7. At the end of the bitstream data, it flushes zeroes for 1,000 `TCK` cycles.
8. Finally, the lines after the `Enter user mode` comment send the `ENTERUSER` instruction and send 1,000 `TCK` clock cycles in the `IDLE` or `SHIFT_DR` state.

Figure 1: Example T35 .svf

```
TRST OFF;
ENDIR IDLE;
ENDDR IDLE;
STATE RESET;
STATE IDLE;
FREQUENCY 6E6 HZ;
TIR 0;
HIR 0;
TDR 0;
HDR 0;
//
```

```
// Check idcode
SIR 4 TDI (3);
SDR 32 TDI (00000000) TDO (00240A79) MASK (ffffffff);
//
// Enter programming mode
SIR 4 TDI (4);
//
// Begin bitstream
//
ENDDR IDLE;
HDR 0;
TDR 0;
SDR 3000 TDI (518CB8065C633E011718CF8045C633E00088100000000...);
SDR 3000 TDI (000000000000000000000000000000000000000000000...);
SDR 3000 TDI (000000000000000000000000000000000000000000000...);
...
SDR 616 TDI (4028140A05028140A05028140A05028140A05028140A04...);
// Extra clock ticks in SDR state
SDR 3000 TDI (000000000000000000000000000000000000000000000...);
//
// Enter user mode
SIR 4 TDI (7);
RUNTEST 100 TCK;
//
```

Note: For detailed information on **.svf** files, refer to the Serial Vector Format Specification by ASSET InterTech, Inc.

Converting a Bitstream

The Efinity® software generates a bitstream as a **.bit** file for JTAG programming. You can use this file for programming, or you can convert it to raw binary (**.bin**) format. The **.bin** has a slightly smaller size than the **.bit**.

When you are sending the bitstream using JTAG commands though, you need to convert it to little endian and then send it in one or more `PROGRAM` commands. The example **.svf** discussed previously breaks the bitstream into many separate commands. However, the number of commands you need depends on the capabilities of the hardware you are using.

Connect the FPGA and MCU

You connect the MCU to the FPGA using the 4 JTAG pins `TCK`, `TMS`, `TDI`, and `TDO`. Additionally, you need to connect a few other pins for managing configuration as described in the following table.

Table 1: Pins to Connect

Pins	FPGA	Guideline
TCK, TMS, TDI, and TDO	All	
CSI	Packages that have a CSI pin.	Pull high during configuration. For packages that do not have a CSI pin, this signal is held high in the package.
CRESET_N	All	Pull high during configuration.
TEST_N	All	Pull high during configuration.
CDONE	All	Optional. Use to monitor configuration done status.
SS	T4 and T8 all packages T13 all packages T20 W80, F169, and F256	Connect as shown in **Trion Schematic and Programming** on page 5

The following sections show connection schematics and the programming sequence for Trion and Titanium FPGAs.

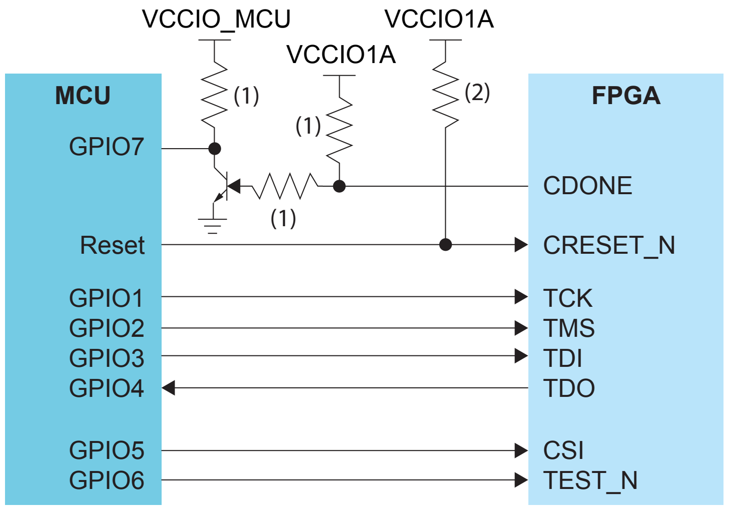
Trion Schematic and Programming

The connection schematic and programming sequence differs slightly for different Trion FPGAs. Follow the instructions for your FPGA/package combination.

For JTAG chains, you configure one FPGA at a time in any order. The FPGAs not being configured are in `BYPASS` mode (not user mode).

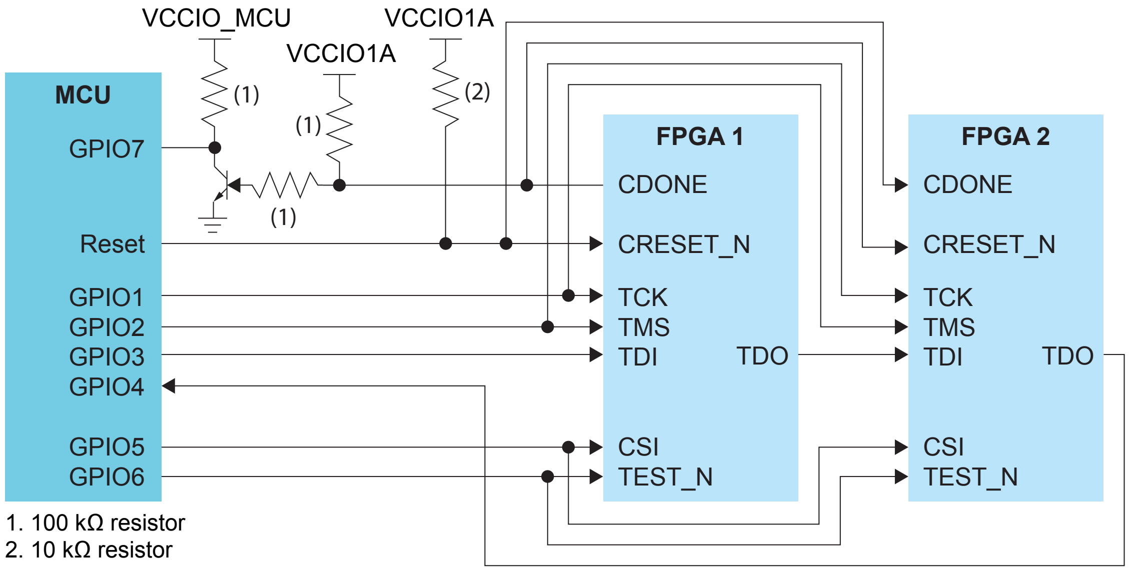
T20 (F324, and F400) and T35, T55, T85, and T120 (All Packages)

Figure 2: Single FPGA



1. 100 kΩ resistor
2. 10 kΩ resistor

Figure 3: Multiple FPGAs



1. 100 kΩ resistor
2. 10 kΩ resistor

If you want to monitor the `CDONE` of each FPGA separately instead of the system status, connect each `CDONE` to a separate GPIO on the MCU.

The configuration sequence is:

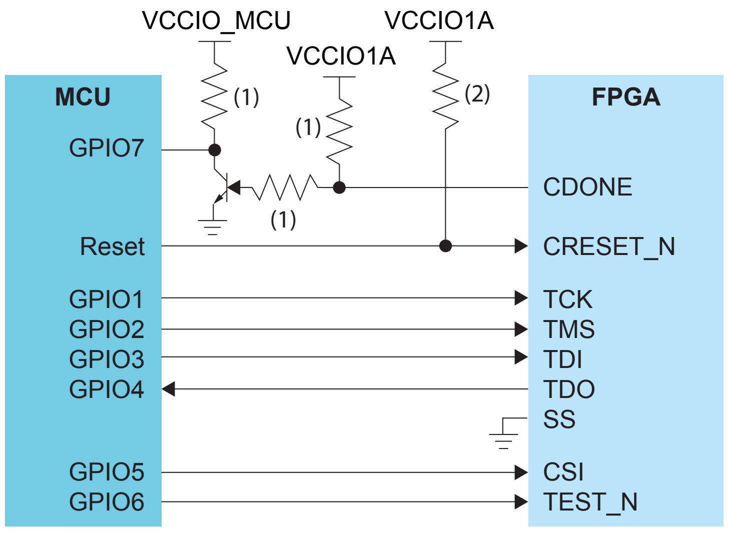
1. For a chain, put all devices except the one you are configuring into `BYPASS` mode.
2. Perform an `IDCODE` check for the FPGA you want to configure.
3. Send the `PROGRAM` command and the bitstream data as well as the zeros for flushing.
4. Send the `ENTERUSER` command.
5. Send 1,000 TCK cycles in the `IDLE` or `SHIFT_DR` state.
6. The FPGA's `CDONE` pin goes high.
7. Repeat steps 2 - 6 for any other FPGAs in the chain.

When you finish configuring all FPGAs, all of their `CDONE` pins should be high.

T4 (All), T8 (All), T13 (All) and T20 (W80, F169, and F256)

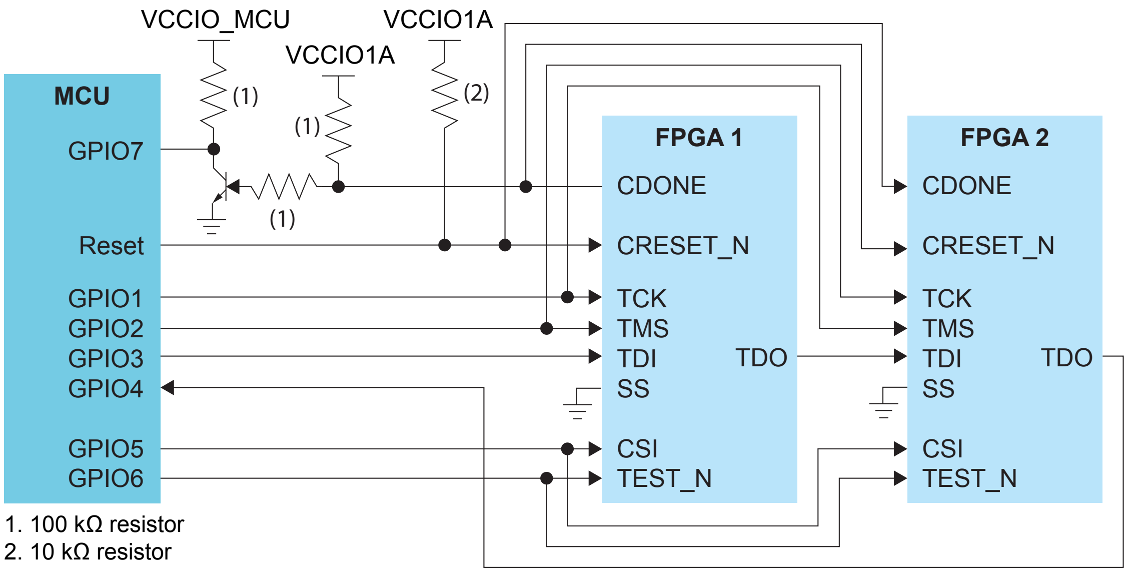
These FPGAs require you to connect the SS pin to ground and have some additional configuration sequence requirements.

Figure 4: Single FPGA



1. 100 kΩ resistor
2. 10 kΩ resistor

Figure 5: Multiple FPGAs



1. 100 kΩ resistor
2. 10 kΩ resistor

If you want to monitor the CDONE of each FPGA separately instead of the system status, connect each CDONE to a separate GPIO on the MCU.

The configuration sequence is:

1. For a chain, put all devices except the one you are configuring into BYPASS mode.
2. Pulse CRESET_N (1 to 0 to 1) at the beginning of configuration. The FPGA(s) go into passive mode. You can probe the FPGA's SCK pin to confirm that it is tri-stated.
3. Perform an IDCODE check for the FPGA you want to configure.
4. Send the PROGRAM command and the bitstream data as well as the zeros for flushing. The FPGA is in the JTAG SHIFT_DR state while the whole bitstream is sent, including

flushing zeros at the end. Do not cycle to `PAUSE_DR` or `IDLE` in the middle of transferring the bitstream or zeros.

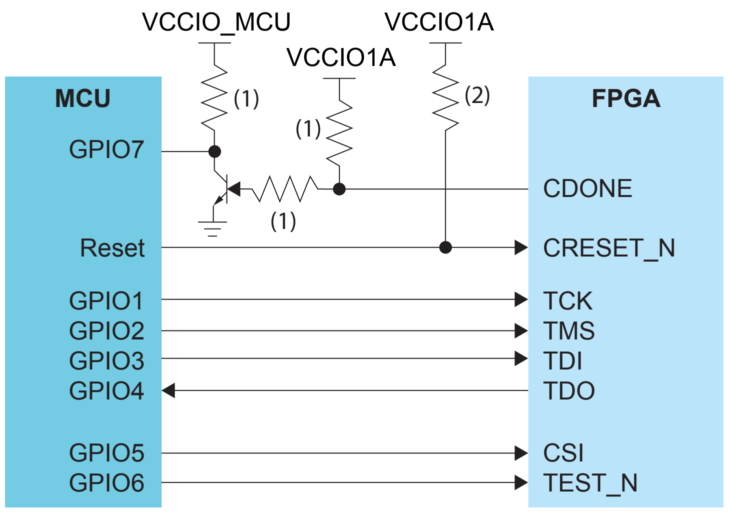
5. Send the `ENTERUSER` command.
6. Send 1,000 TCK cycles in the `IDLE` or `SHIFT_DR` state.
7. The FPGA's `CDONE` pin goes high.
8. Repeat steps 2 - 6 for any other FPGAs in the chain.

When you finish configuring all FPGAs, all of their `CDONE` pins should be high.

Titanium Schematic and Programming

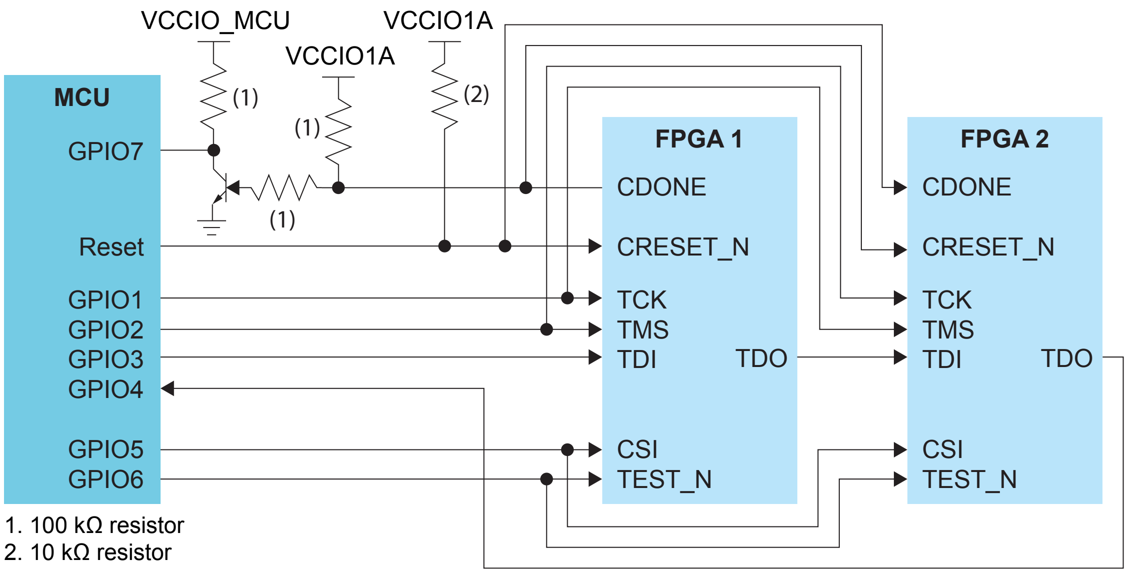
For JTAG chains, you configure one FPGA at a time in any order. The FPGAs not being configured are in `BYPASS` mode (not user mode).

Figure 6: Single FPGA



1. 100 kΩ resistor
2. 10 kΩ resistor

Figure 7: Multiple FPGAs



1. 100 kΩ resistor
2. 10 kΩ resistor

If you want to monitor the CDONE of each FPGA separately instead of the system status, connect each CDONE to a separate GPIO on the MCU.

The configuration sequence is:

1. For a chain, put all devices except the one you are configuring into BYPASS mode.
2. Perform an IDCODE check for the FPGA you want to configure.
3. Send **two** PROGRAM commands and the bitstream data as well as the zeros for flushing.
4. Send the ENTERUSER command.
5. Send 1,000 TCK cycles in the IDLE or SHIFT_DR state.
6. The FPGA's CDONE pin goes high.
7. Repeat steps 2 - 6 for any other FPGAs in the chain.

When you finish configuring all FPGAs, all of their CDONE pins should be high.

Appendix

This appendix provides reference information.

JTAG Device IDs

Table 2: Titanium JTAG Device IDs

FPGA	Package	JTAG Device ID
Ti35	All	0x10660A79
Ti60ES	All	0x00360A79
Ti60	All	0x10661A79

Table 3: Trion JTAG Device IDs

FPGA	Package	JTAG Device ID
T4, T8	BGA49, BGA81	0x0
T8	QFP144	0x00210A79
T13	All	0x00210A79
T20	WLCSP80, QFP144, BGA169, BGA256	0x00210A79
T20	BGA324, BGA400	0x00240A79
T35	All	0x00240A79
T55, T85, T120	All	0x00220A79

Supported JTAG Instructions

The following table shows the JTAG instructions Titanium FPGAs support.

Table 4: Supported JTAG Instructions (Titanium)

Instruction	Binary Code [4:0]	Description
BYPASS	11111	Enables BYPASS.
DEVICE_STATUS	01100	Lets you read the device configuration status.
EFUSE_PREWRITE	11000	Loads user data for fuse operations.
EFUSE_USER_WRITE	11010	Blows fuses as defined in EFUSE_PREWRITE.
EFUSE_WRITE_STATUS	11011	Returns status of EFUSE_USER_WRITE operation.
ENTERUSER	00111	Changes the FPGA into user mode.
EXTEST	00000	Enables the boundary-scan EXTEST operation.
IDCODE	00011	Enables shifting out the IDCODE.
INTEST	00001	Enables the boundary-scan INTEST operation.
JTAG_USER1	01000	Connects the JTAG User TAP 1.
JTAG_USER2	01001	Connects the JTAG User TAP 2.
JTAG_USER3	01010	Connects the JTAG User TAP 3.
JTAG_USER4	01011	Connects the JTAG User TAP 4.
PROGRAM	00100	JTAG configuration.
SAMPLE/PRELOAD	00010	Enables the boundary-scan SAMPLE/PRELOAD operation.
USERCODE	01101	Use this instruction to program a 32-bit signature into the FPGA during programming.

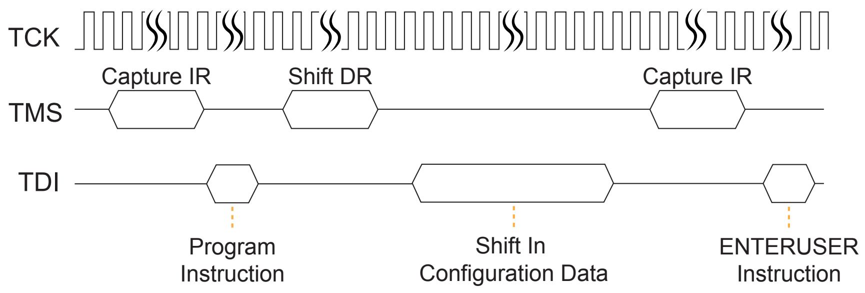
The following table shows the JTAG instructions Trion FPGAs support.

Table 5: Supported JTAG Instructions (Trion)

Instruction	Binary Code [3:0]	Description
SAMPLE/PRELOAD	0010	Enables the boundary-scan SAMPLE/PRELOAD operation
EXTEST	0000	Enables the boundary-scan EXTEST operation
BYPASS	1111	Enables BYPASS
IDCODE	0011	Enables shifting out the IDCODE
PROGRAM	0100	JTAG configuration
ENTERUSER	0111	Changes the FPGA into user mode.

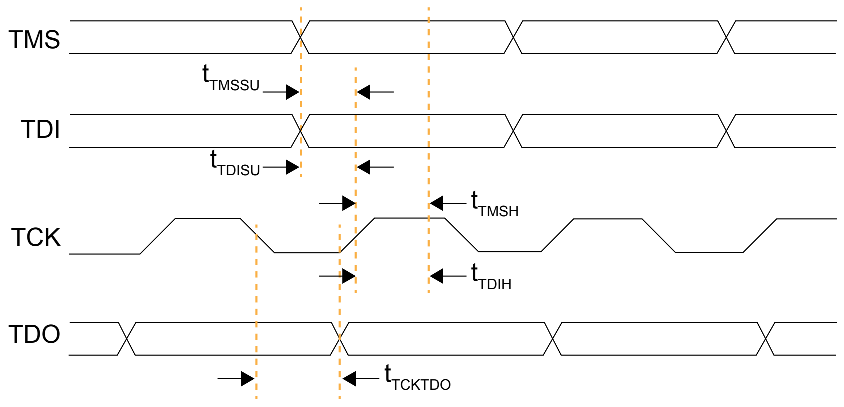
JTAG Waveforms and Timing

Figure 8: JTAG Programming Waveform



When configuration ends, the JTAG host issues the ENTERUSER instruction to the FPGA. After `CDONE` goes high and the FPGA receives the ENTERUSER instruction, the FPGA waits for t_{USER} to elapse, and then it goes into user mode.

Figure 9: Boundary-Scan Timing Waveform



Note: The FPGA may go into user mode before t_{USER} has elapsed. Therefore, you should keep the system interface with the FPGA in reset until t_{USER} has elapsed.

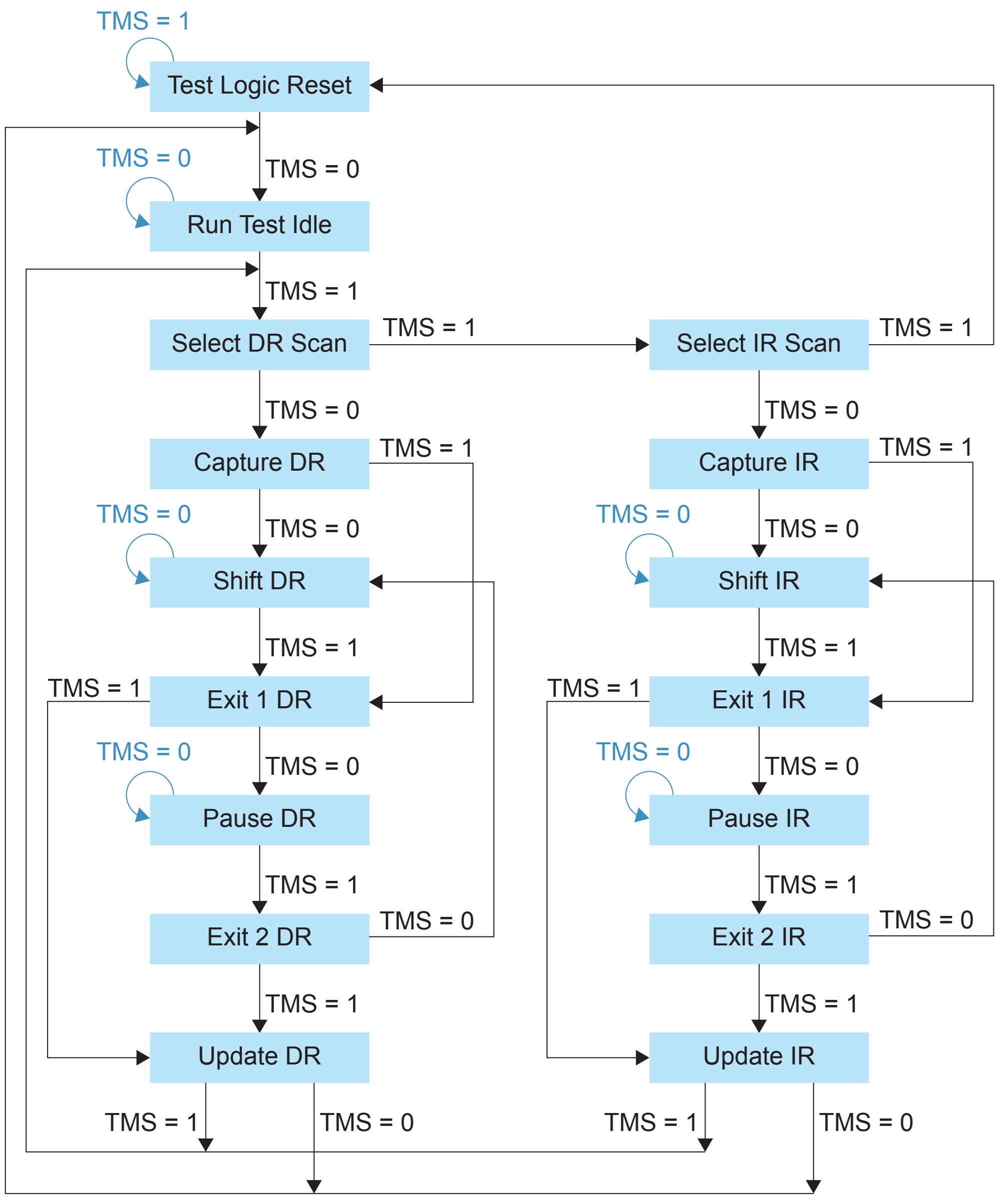
Follow these guidelines for the JTAG signals:

- Because the `TCK` and `TMS` signals connect devices in the JTAG chain, they must have good signal quality.
- `TCK` should transition monotonically at the receiving devices and should be terminated correctly. Poor `TCK` quality can limit the maximum frequency you can use for configuration.
- Buffer `TMS` and `TCK` so they have sufficient drive strength at all receiving devices.
- Ensure that the logic high voltage is compatible with all devices in the JTAG chain.
- If your chain contains devices from different vendors, you might need to drive optional JTAG signals, such as `TRST` and enables.

JTAG TAP Flow Chart

The following figure shows the standard JTAG TAP flow chart for your reference.

Figure 10: JTAG TAP Flow Chart



Revision History

Table 6: Revision History

Date	Version	Description
November 2021	1.0	Initial release.