Product Security Features for Altera Devices

2017.03.29

AN-680 Subscribe Send Feedback

The Altera® Arria® II GX, Arria V, Cyclone® III LS, Cyclone V, and Stratix® V device families include user mode anti-tamper features. These device families include a configuration engine incorporating management features to detect changes in the design configuration, allowing you to control the response. The configuration engine is a hard IP block that manages the SRAM download to an Altera FPGA device.

For more information on Altera device security features, refer to the *Arria II Device Handbook*, *Arria V Device Handbook*, *Cyclone III Device Handbook*, *Cyclone V Device Handbook*, and *Stratix V Device Handbook*.

Related Information

- **Arria II Device Handbook**
- **Arria V Device Handbook**
- **Cyclone III Device Handbook**
- **Cyclone V Device Handbook**
- **Stratix V Device Handbook**

Anti-Tamper Features

Altera provides two broad types of anti-tamper features; passive protection mechanisms and user mode anti-tamper features.

Passive Protection

Passive protection includes inherent security features that protect the integrity of configuration data.

Passive protection includes support for configuration file encryption and the physical layout of the chip, minimizing susceptibility to side channel attacks. These features require limited interaction with the FPGA design and are available in the device design or are managed during the configuration process.

User Mode Anti-Tamper Features

User mode anti-tamper features include the set of functions available to the FPGA device core routing fabric from the configuration engine of the device.

The anti-tamper features are a set of status and reporting functions and a control interface to the configuration engine. A user-designed controller monitors the integrity of the device and its associated configuration data with status and reporting functions. The controller issues commands to the configuration engine to clear critical program information when the controller detects events out of normal operating conditions.

Device Configuration Engine

The device configuration engine is a hardened block that manages the device configuration download and provides a set of functions related to the configuration image.

The specific functions from the configuration block to the FPGA device logic differs for each device family, but the general architecture of each device family is similar. The device configuration engine includes three feature classes:

- Control interface to the configuration engine (JTAG)
- Status and reporting functions
- User functions

Control Interface to the Configuration Engine (JTAG)

The JTAG port acts as the control interface to the configuration engine. In the Arria II GX, Arria V, Cyclone III LS, Cyclone V, and Stratix V device families, the JTAG port is accessible internally from the FPGA routing fabric.

The JTAG test access port (TAP) controller has a private instruction set to allow management of the configuration process. The controller includes functions such as the loading and clearing of volatile key variables, reconfiguration of the device, and certain configuration mode setting options. This JTAG port acts as the serial management interface to the configuration engine. Accessibility of JTAG resources from the core allows integration of all configuration monitor services within the FPGA device design to enhance security.

The JTAG port makes the system vulnerable if sensitive information can be read out of a device. Altera devices do not have a read back path for either the Advanced Encryption Standard (AES) key material or the configuration data. No physical path from the AES key registers or the configuration RAM array route back out to the test data out (TDO) pin on the JTAG port. For anti-tamper enhanced device families, an optional JTAG secure mode restricts the instruction set from the external JTAG port.

Arria V, Cyclone III LS, Cyclone V, and Stratix V device families disconnect the external JTAG port completely when the internal JTAG port is active.

Status and Reporting Functions

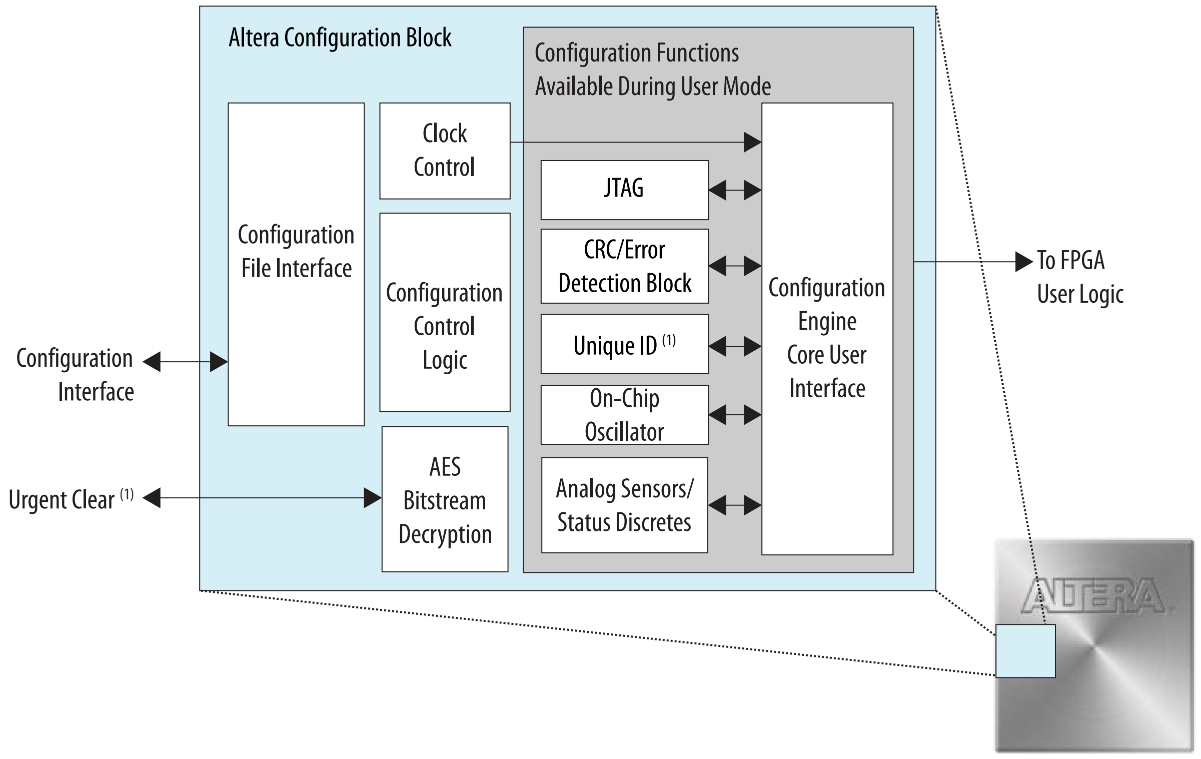
The status and reporting functions allow the system design to monitor the device. An example of a status and reporting function is the cyclic redundancy check (CRC) circuit; this circuit is available in most device families that monitor for single event upset (SEU) faults in the configuration RAM.

Most status and reporting functions are primitives that are instantiated and accessed by the FPGA design.

User Functions

Hard wired user functions are meant to enhance security at the application layer. An example of a user function is the internal oscillator routed from the configuration engine to provide an uninterruptible clock supply to your design.

Figure 1: Configuration Engine Anti-Tamper Features


Altera Configuration Block
Configuration Functions Available During User Mode
Clock Control
JTAG
CRC/Error Detection Block
Unique ID (1)
On-Chip Oscillator
Analog Sensors/ Status Discretes
Configuration File Interface
Configuration Control Logic
AES Bitstream Decryption
Configuration Engine Core User Interface
To FPGA User Logic
Configuration Interface
Urgent Clear (1)
ALTERA

(1)

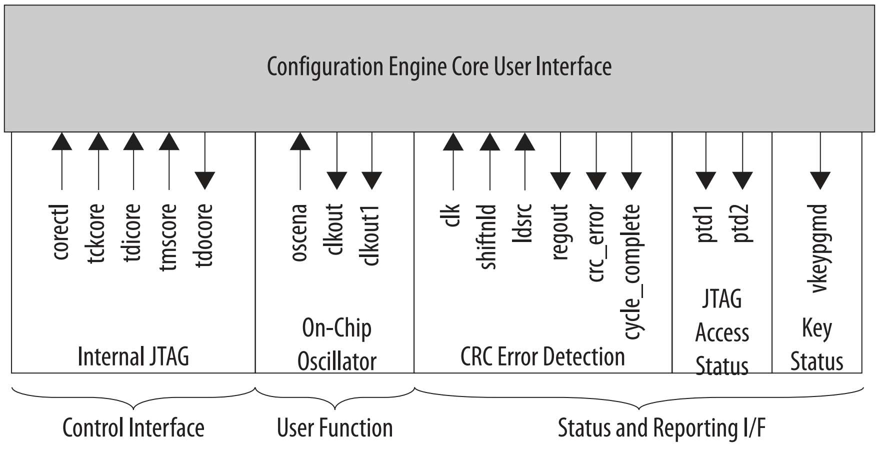
Cyclone III LS

The Cyclone III LS device family supports the following anti-tamper features:

- AES-256 encrypted configuration files
- Internally accessible JTAG interface, with an instruction set for AES key management and status reporting
- JTAG anti-tamper prevention
- On-chip oscillator
- CRC error detection

[1] Only available in 28 nm device families.

Figure 2: Cyclone III LS Configuration Engine: Core User Interface



Cyclone III LS JTAG Block

The Cyclone III LS device family JTAG block differs from the Cyclone III device family in the following ways:

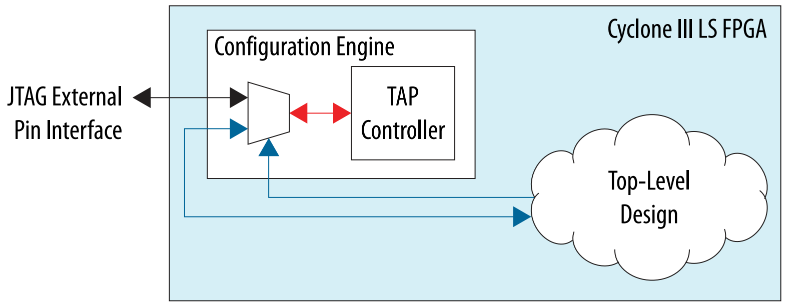
- JTAG core access; the TAP controller is driven internally from within the FPGA
- The external JTAG port powers up with JTAG secure mode, where only the mandatory JTAG instruction set is available
- An expanded JTAG instruction set with instructions to manage the AES key and JTAG secure mode

Cyclone III LS Internal JTAG Block Access

Cyclone III LS devices allow you to optionally drive the JTAG TAP controller from within the FPGA. A single port interface, either the internal JTAG interface or the external JTAG pin interface, is active at a given time. When the internal JTAG port is active, Cyclone III LS devices tri-state the external JTAG interface and break chain continuity for the rest of the JTAG chain.

Note: External JTAG pins connect to the JTAG TAP controller by dedicated routing, with no connection points to and from the core routing. Sampling of JTAG input pins from the core logic is not possible while the internal port is active. Similarly, it is not possible to drive the TDO output pin directly from the core.

Figure 3: JTAG TAP Controller Access, Cyclone III LS



The internal JTAG port is accessed by instantiating the JTAG atom. For the Quartus II software to recognize the atom, the following Quartus settings file (**.qsf**) assignment must be set:

```
set_global_assignment -name INI_VARS “sgn_enable_ciiigl_advance_feature=on”
```

Refer to the JTAG chapter in the *Cyclone III Device Handbook* for the JTAG port interface timing characteristics.

Note: If you are instantiating the JTAG atom for internal controller access, all functions requiring the use of the JTAG port must be removed from your design; this includes all on-chip debugging tools and anything relying on the JTAG UART from SOPC Builder or Qsys. Instantiating this function and another resource that uses the external JTAG port causes a resource conflict resulting in a compilation error.

You can assert the `corectl` signal high to enable the internal port. Only a single JTAG interface, either internal or external, is active at a given time.

Example 1: JTAG Controller Block (Internal Access) Component Declaration

```
component cycloneiiils_jtag
    generic (
        lpm_hint       :    string := "UNUSED";
        lpm_type       :    string := "cycloneiiils_jtag"    );
    port(
        clkdruser      :    out std_logic;
        corectl        :    in std_logic := '0';
        runidleuser    :    out std_logic;
        shiftuser      :    out std_logic;
        tck               :    in std_logic := '0';
        tckcore        :    in std_logic := '0';
        tckutap        :    out std_logic;
        tdi               :    in std_logic := '0';
        tdicore        :    in std_logic := '0';
        tdiutap        :    out std_logic;
        tdo               :    out std_logic;
        tdocore        :    out std_logic;
        tdouser        :    in std_logic := '0';
        tdoutap        :    in std_logic := '0';
        tms               :    in std_logic := '0';
        tmscore        :    in std_logic := '0';
        tmsutap        :    out std_logic;
        updateuser     :    out std_logic;
        usr1user       :    out std_logic
    );
end component;
```

Related Information

Cyclone III Device Handbook

Cyclone III LS JTAG Atom Port and Parameter Description

Table 1: Cyclone III LS JTAG Atom Port and Parameter Description

Port/Parameter	Description
	Input Ports

Port/Parameter	Description
`Corectl`	This signal enables the JTAG port from the configuration engine to the FPGA device core. The signal is active high. The device resets when the external port is enabled.
`Tckcore`	The core JTAG test clock (TCK) signal.
`Tdicore`	The core JTAG test data in (TDI) signal.
`Tmscore`	The core test mode select (TMS) signal.
`Tdi`	The external JTAG TDI pin.
`tck`	The external JTAG TCK pin.
`Tms`	The external JTAG TMS pin.
`Tdouser`	This signal is not supported. Typically, it supports on-chip scan chains targeted from the `USER1` and `USER0` instructions. On-chip debugging applications use this signal; `USER1` and `USER0` scan chains are not supported when using the internal JTAG port.
Output Ports	
`Tdo`	The external port TDO pin.
`Tdocore`	The core TDO signal.
`Runidleuser`	These signals are not supported. Typically, they support on-chip scan chains targeted from the `USER1` and `USER0` instructions. On-chip debugging applications use these signals; `USER1` and `USER0` scan chains are not supported when using the internal JTAG port.
`shiftuser`	
`Tckutap`	
`tdiutap`	
`tmsutap`	
`Updateuser`	
`usr1user`	

Cyclone III LS JTAG Secure Mode

The Cyclone III LS device powers up in JTAG secure mode. When there is a power-on reset, the device restricts the full JTAG instruction set from the external interface, except the three mandatory JTAG instructions (`SAMPLE`/`PRELOAD`, `BYPASS`, `EXTEST`) and a private Altera JTAG instruction, `FACTORY`. When the `FACTORY` command executes successfully, it enables a full JTAG instruction set. For an additional layer of configuration data protection, successful execution of the `FACTORY` instruction clears the battery-backed AES key and forces a reset of the device. Cyclone III LS devices actively clear the configuration data upon a reset.

The configuration engine only accepts the FACTORY command before any configuration data is loaded.

The configuration cycle includes the following five stages:

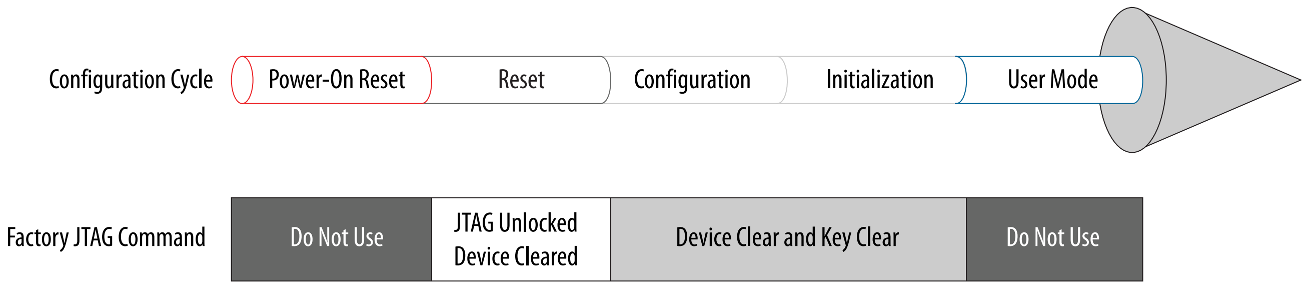
- **Power-on reset** - The device powers up and waits for all voltage rails to reach their nominal values. All I/Os on the device are tri-stated. The volatile key is tied into the power-on reset circuitry.
- **Reset** - This is the initialization phase before the configuration data is actively loaded. The device actively clears all configuration memory and embedded RAM on the device.
- **Configuration** - Configuration data transfers to the device over the dedicated configuration pins.
- **Initialization** - This is the device configuration engine processing phase before the device is released into user mode.
- **User Mode** - This is the completion of the configuration load process. The configuration loaded into the design becomes active.

To successfully enable the full JTAG instruction set from the external port, the FACTORY command must be issued before the onset of the configuration stage, or before the start header byte (0x6a) is sent to the device. Configuration onset is delayed by holding the external nStatus pin low or by withholding configuration data to the device. The configuration RAM contents and the battery-backed AES key are actively cleared when the FACTORY command is issued after the configuration stage begins. The JTAG secure mode remains enabled.

Note: The FACTORY command has no affect when issued during user mode or reset.

The following figure illustrates the behavior of the FACTORY command when issued during the various stages of the configuration cycle:

Figure 4: FACTORY Command Execution Behavior



The ptd1 and ptd2 signals from the JTAG port protection status block monitor JTAG secure mode status from within the core.

Refer to *Cyclone III LS JTAG Instruction Set* for further details on enabling the full JTAG instruction set.

Related Information

Cyclone III LS JTAG Instruction Set on page 7

Cyclone III LS JTAG Instruction Set

The Cyclone III LS device powers up in JTAG secure mode with three mandatory JTAG instructions (SAMPLE/PRELOAD, BYPASS, EXTEST) and a private Altera JTAG instruction, FACTORY. FACTORY enables the full JTAG instruction set of the device.

Enabling the Cyclone III LS Expanded JTAG Instruction Set

The following procedure enables the expanded JTAG instruction set:

1. Power up all supplies needed to configure the device, including `VccBAT`.
2. Hold `nSTATUS` low or withhold a configuration clock or programming file in the external memory to prevent the device from loading configuration data.
3. Load the `FACTORY` JTAG instruction.
4. Wait in Run-Test/Idle (RTI) state of the JTAG TAP controller, continuing to clock the TCK.
5. Release `nStatus` if it is held low.
6. Wait for the `nStatus` pin to go high.

Cyclone III LS JTAG Instruction Set and Target Registers

The following table lists the Cyclone III LS JTAG instructions with relevance to AES key management or anti-tamper features.

Table 2: Cyclone III LS JTAG Instruction Set and Target Registers

JTAG Instruction	Instruction Binary	Target Register
`FACTORY`	10 1000 0001	Bypass register
`KEY_CLR_VREG`	00 0010 1001	Bypass register
`KEY_PROG_VOL`	01 1010 1101	Security key scan chain[6]
`KEY_VERIFY`	00 0001 0011	Key verification scan register
`PULSE_NCONFIG`	00 0000 0001	Bypass register
`RESET_AES`	10 1000 0010	Bypass register

Refer to the *Cyclone III LS Device Handbook* for information on other JTAG instructions.

Related Information

Cyclone III Device Handbook

FACTORY

The Cyclone III LS device powers up with a limited JTAG instruction set. To open the full JTAG instruction set, you must issue the FACTORY instruction after a power-on reset and before the configuration cycle begins. If `FACTORY` issues successfully, the `ptd1` and `ptd2` signals from the JTAG port protection status block deasserts.

Refer to *Enabling the Cyclone III LS Expanded JTAG Instruction Set* for the procedure to unlock the JTAG instruction set.

Issuing the `FACTORY` instruction always clears the volatile key, resets the configuration AES decryption engine, and resets the device.

If the FACTORY instruction issues after any configuration frames are loaded, the full JTAG instruction set is not available. However, FACTORY can still clear key material and re-initialize the device.

Refer to *Cyclone III LS JTAG Secure Mode* for more information on the effect of FACTORY on the configuration time line.

Related Information

- **Enabling the Cyclone III LS Expanded JTAG Instruction Set** on page 8
- **Cyclone III LS JTAG Secure Mode** on page 6

KEY_CLR_VREG

The KEY_CLR_VREG instruction clears all registers related to the volatile key registers and deasserts the vkeypgmd bit from the volatile key verify block.

KEY_PROG_VOL

The KEY_PROG_VOL instruction programs the battery-backed AES key. Two 256 bit sequences are required to generate the 256 bit security key. The length of the key entered into the device is 512 bits. The RTI state requires an additional 400 TCK cycles after the 512 bits of key material are shifted into the key registers. Do not issue additional JTAG instructions before 400 TCK clock cycles have elapsed.

The vkeypgmd signal from the volatile key verify block asserts high after the key is successfully programmed into the key registers. You must reconfigure the device once KEY_PROG_VOL executes successfully.

Refer to *Using the Design Security Feature in Cyclone III LS Devices* for more information about the generation of encrypted files for Cyclone III LS.

Related Information

Using the Design Security Feature in Cyclone III LS Devices

KEY_VERIFY

You can read out the register contents of the key verify register with the KEY_VERIFY instruction. The key verify register asserts when the battery-backed key RAM is successfully programmed with a key value.

PULSE_NCONFIG

The PULSE_NCONFIG instruction resets the device and clears the embedded memory.

RESET_AES

The RESET_AES instruction resets the programming file decryption engine. RESET_AES is only accessible from the internal JTAG interface.

Cyclone III LS Volatile Key Verify Block Status

The volatile key verify block status bit from the configuration engine indicates whether or not a valid key is programmed into the key registers. When this bit is high, the configuration engine enables programming file decryption. This status bit can be queried with the KEY_VERIFY JTAG instruction.

Example 2: Volatile Key Block Status Component Declaration

```
component cycloneiiils_volatilekeyblock
    generic (
        lpm_hint    :    string := "UNUSED";
```

```
            lpm_type    :    string := "cycloneiiils_volatilekeyblock"    );
        port(
            vkeypgmd    :    out std_logic
        );
    end component;
```

Cyclone III LS Volatile Key Verify Block Status Port and Parameter Description

Table 3: Volatile Key Block State Port and Parameter Description

Port/Parameter	Description
Output Ports	
`Vkeypgmd`	This signal indicates that a valid key has been programmed into the key registers. The signal is active high. The configuration engine enables programming file decryption when this bit is enabled.

Cyclone III LS JTAG Port Protection Status

The JTAG port protection status block provides a set of redundant status signals to the FPGA device core. These signals indicate the state of the external JTAG port.

Example 3: Protection Status Block Component Declaration

```
    component cycloneiiils_testaccessblock
        generic (
            lpm_hint    :    string := "UNUSED";
            lpm_type    :    string := "cycloneiiils_testaccessblock"    );
        port(
            ptd1           :    out std_logic;
            ptd2           :    out std_logic
        );
    end component;
```

Cyclone III LS JTAG Port Protection Status Port and Parameter Description

Table 4: JTAG Port Protection Status Port and Parameter Description

Port/Parameter	Description
Output Ports	
`ptd1`	This signal indicates that the JTAG instruction set is limited to `SAMPLE`, `PRELOAD`, `EXTEST` and `FACTORY`. The signal is active high. The device always powers up with this signal asserted high.

Port/Parameter	Description
`ptd2`	This is a redundant signal; it has the same functionality as `ptd1`.

Cyclone III LS On-Chip Oscillator

The on-chip oscillator is the internal chip oscillator from the configuration engine used during active configuration modes. This clock output does not pass through the clock divisor. Nominal frequency of the output is 60 MHz, but output frequency varies from 40 MHz to 80 MHz due to variations in process, voltage, and temperature.

Example 4: On-Chip Oscillator Component Declaration

```
component cycloneiiils_oscillator
    generic (
        lpm_hint                :    string := "UNUSED";
        lpm_type                :    string :=
"cycloneiiils_oscillator"    );
    port(
        clkout                  :    out std_logic;
        clkout1                 :    out std_logic;
        observableoutputport    :    out std_logic;
        oscena                  :    in std_logic := '0'
    );
end component;
```

Cyclone III LS On-Chip Oscillator Port and Parameter Description

Table 5: On-Chip Oscillator Port and Parameter Description

Port/Parameter	Description
Input Ports	
`Oscena`	This is the user enable signal for the core to turn off `OSC_CORE` output. The oscillator still runs internally. This signal is active high.
Output Ports	
`clkout`	The internal oscillator output signal (40 to 80 MHz range).
`clkout2`	This is a redundant signal; it has the same functionality as `clkout`.
`Observableoutputport`	This is a reserved output signal; you should leave the port open.

Cyclone III LS CRC Error Detection

The CRC error detection circuit continuously scans the configuration logic to check for any errors in the configuration RAM. You can enable this function in the Quartus II software under the **Device and Pin Options** dialog box. The primitive opens up additional ports to read the 32 bit CRC signature from the configuration engine.

Example 5: CRC Error Detection Component Declaration

```
component cycloneiiils_crcblock
    generic (
        lpm_hint                    :    string := "UNUSED";
        lpm_type                    :    string := "cycloneiiils_crcblock";
        oscillator_divider    :    natural := 1    );
    port(
        clk                            :    in std_logic := '0';
        crcerror                    :    out std_logic;
        cyclecomplete            :    out std_logic;
        ldsrc                          :    in std_logic := '0';
        regout                      :    out std_logic;
        shiftnld                    :    in std_logic := '0'
    );
end component;
```

Refer to the *Cyclone III LS Device Handbook* and *Error Detection and Recovery Using CRC in Altera FPGA Devices* for further information on the CRC error detection engine.

Related Information

- Cyclone III Device Handbook
- Error Detection and Recovery Using CRC in Altera FPGA Devices

Cyclone III LS CRC Error Detection Port and Parameter Description

Table 6: CRC Error Detection Port and Parameter Description

Port/Parameter	Description
	Parameters
`oscillator_divider`	This signal is the clock divider for the internal oscillator feeding the CRC circuit. The valid values are from 1 to 256.
	Input Ports
`clk`	This signal designates the clock input of the cell. The cell operates in relation to the rising edge of the clock; both the loading of the data into the cell or data out of the cell occurs on the rising edge. This is a required port.
`Ldsrc`	This signal is an input to the error detection block. If `ldsrc`=0, the pre-computed CRC register loads into the 32 bit shift register at the rising edge of `clk` when `shiftnld`=0. If `ldsrc`=1, the signature register (result of the CRC calculation) loads into the shift register at the rising edge of `clk` when `shiftnld`=0. This port is ignored when `shiftnld`=1. This is a required port.

Port/Parameter	Description
`shiftnld`	This signal is an input to the error detection block. If `shiftnld=1`, the data shifts from the internal shift register to the `regout` at each rising edge of `clk`. If `shiftnld=0`, the shift register parallel loads either the pre-calculated CRC value or the update register contents depending on the `ldsrc` port input. This is a required port.
Output Ports	
`Crcerror`	This signal is the output of the cell that is synchronized to the internal oscillator of the device (60MHz, nominal) and not to the `clk` port. It asserts automatically high if the error block detects that an SRAM bit has flipped and the internal CRC computation shows a difference in the pre-computed value. This signal must be connected to an output pin or a bidirectional pin. When `Crcerror` connects to an output pin, you can only monitor the `CRC_ERROR` pin (the core cannot access this output). If core logic uses the `CRC_ERROR` signal to read error detection logic, this signal must be connected to a `BIDIR` pin. The signal is fed to the core indirectly by feeding a `BIDIR` pin that has its output enable port connected to VCC.
`regout`	This signal is the output of the error detection shift register synchronized to the `clk` port, to be read by core logic. It shifts one bit each cycle; you should clock the `clk` signal at 31 cycles to read out the 32 bits of the shift register. The values at the `regout` port are an inversion of the actual values.
`cyclecomplete`	This signal asserts high for one clock cycle every time a CRC detection cycle has completed. This signal is synchronous to the internal oscillator of the device.

Arria II GX

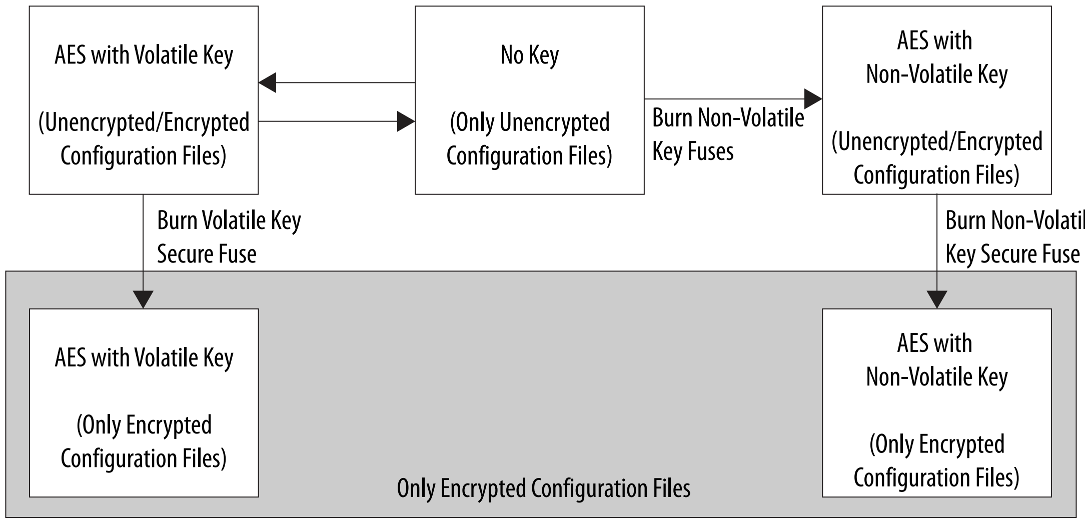
The Arria II GX device family supports the following anti-tamper features:

- AES-256 encrypted configuration files
- Internally accessible JTAG interface, with an instruction set for AES key management and status reporting
- JTAG anti-tamper prevention
- CRC error detection

Arria II GX devices support the use of two different types of keys for configuration file encryption: a non-volatile One Time Programmable (OTP) PolySilicon Fuse based key, and a battery-backed volatile key. Only a single key is present on the device at any given time. Once the non-volatile fuses are set, the volatile keys are no longer used.

An additional set of option bits (set by a fuse) locks the device into accepting only an encrypted configuration. Two fuses are available, one for the non-volatile key and one for the battery-backed volatile key. Both fuses are set by a set of JTAG instructions.

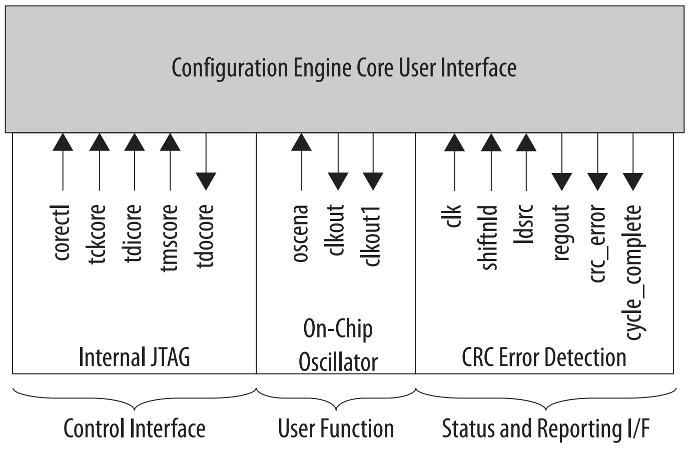
Figure 5: Arria II GX Encryption Modes – Sequence and Restrictions


AES with Volatile Key
(Unencrypted/Encrypted Configuration Files)
No Key
(Only Unencrypted Configuration Files)
Burn Non-Volatile Key Fuses
AES with Non-Volatile Key
(Unencrypted/Encrypted Configuration Files)
Burn Volatile Key Secure Fuse
Burn Non-Volatile Key Secure Fuse
AES with Volatile Key
(Only Encrypted Configuration Files)
Only Encrypted Configuration Files
AES with Non-Volatile Key
(Only Encrypted Configuration Files)

The JTAG interface manages all key functions, including programmable polyfuses. The JTAG instruction set for Arria II GX devices allows for key management functions and commands to query the status of the decryption block. Two JTAG ports are available: one from the external pin interface, and one internal port accessible from the FPGA device routing fabric.

Refer to *Arria II GX JTAG Instruction Set and Target Registers* for further details on enabling the full JTAG instruction set, including key management functions and instructions to program polyfuses for Arria II GX encryption modes.

Figure 6: Arria II GX Configuration Engine-Core User Interface



Refer to the *Arria II Device Handbook* and *Using the Design Security Features in Altera FPGAs* for more information on the design security feature in Arria II devices.

Related Information

- **Arria II GX JTAG Instruction Set and Target Registers** on page 18

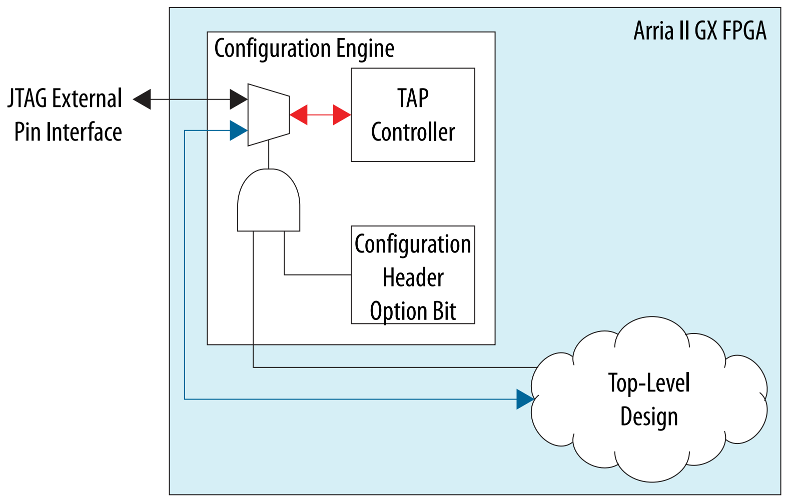
- Arria II Device Handbook
- Using the Design Security Features in Altera FPGAs

Arria II GX JTAG Block

Arria II devices allow you to optionally drive the JTAG TAP controller from within the FPGA. A single port interface, either internal or external, is active at a given time. Arria II GX devices maintain JTAG chain continuity when the internal JTAG port is active. The TDI pin routes directly to the TDO pin and controllers are not able to detect the Arria II device in the chain.

Note: External JTAG pins connect to the JTAG TAP controller by dedicated routing, with no connection points to and from the core routing. Sampling of JTAG input pins from the core logic is not possible while the internal port is active. Similarly, it is not possible to drive the TDO output pin directly from the core.

Figure 7: JTAG TAP Controller Access, Arria II GX



Arria II GX JTAG Secure Mode

Arria II GX devices support JTAG secure mode. The following JTAG instructions are active when JTAG secure mode is enabled:

- `BYPASS`
- Volatile key management instructions: `KEY_CLR_VREG`; `KEY_PROG_VOL`; and `KEY_VERIFY`
- Device reset instruction: `PULSE_NCONFIG`

You can enable the JTAG anti-tamper mode by setting a polyfuse; enabling this mode is not reversible. The JTAG anti-tamper polyfuse is set through the JTAG instruction `KEY_DISABLE_JTAG`. Setting the fuse limits the instruction set from both the external and internal JTAG interfaces.

Refer to the *Arria II GX JTAG Instruction Set* for further information on the instruction set.

Related Information

Arria II GX JTAG Instruction Set and Target Registers on page 18

Arria II GX Internal JTAG Block Access

You can access the internal JTAG port through instantiation of the JTAG primitive. The internal port is active when the JTAG internal access option bit is set and the FPGA device design asserts `corectl`. The Quartus II software does not support the full atom by default. To enable support for the internal JTAG port, the following **.qsf** assignment must be set:

```
set_global_assignment -name INI_VARS "sgn_enable_ciiigl_advance_feature=on"
```

Note: Cyclone III LS devices use the same **.qsf** assignment to allow access to the internal JTAG port.

You can set the `sgn_enable_ciiigl_advance_feature` **.qsf** assignment to enable compilation support for the JTAG instantiation template provided below. The assignment prompts the Quartus II software to set the JTAG configuration header option bit during the assembler stage. If you are converting the programming file to a different file format (for example, a **.rbf** or a **.pof** file), you should convert in a project directory that has the `sgn_enable_ciiigl_advance_feature` assignment set.

Refer to the JTAG chapter in the *Arria II Device Handbook* for the JTAG port interface timing characteristics.

Note: If you are instantiating the JTAG atom for internal controller access, all functions requiring the use of the JTAG port must be removed from your design; this includes all on-chip debugging tools and anything relying on the JTAG UART from SOPC Builder or Qsys. Instantiating this function and another resource that uses the external JTAG port causes a resource conflict that results in a compilation error.

Example 6: JTAG Controller Block (Internal Access) Component Declaration

```
component arriaii_jtag
    generic (
        lpm_hint       :    string := "UNUSED";
        lpm_type       :    string := " arriaii_jtag"    );
    port(
        clkdruser      :    out std_logic;
        corectl        :    in std_logic := '0';
        runidleuser    :    out std_logic;
        shiftuser      :    out std_logic;
        tck               :    in std_logic := '0';
        tckcore        :    in std_logic := '0';
        tckutap        :    out std_logic;
        tdi               :    in std_logic := '0';
        tdicore        :    in std_logic := '0';
        tdiutap        :    out std_logic;
        tdo               :    out std_logic;
        tdocore        :    out std_logic;
        tdouser        :    in std_logic := '0';
        tdoutap        :    in std_logic := '0';
        tms               :    in std_logic := '0';
        tmscore        :    in std_logic := '0';
        tmsutap        :    out std_logic;
        updateuser     :    out std_logic;
        usr1user       :    out std_logic
    );
end component;
```

Related Information

Arria II Device Handbook

Arria II GX JTAG Atom Port and Parameter Description

Table 7: Arria II GX JTAG Atom Port and Parameter Description

Port/Parameter	Description
Input Ports	
`Corectl`	This signal enables the JTAG port from the configuration engine to the FPGA device core. The signal is active high. The device resets when the external port is enabled.
`Tckcore`	The core TCK signal.
`Tdicore`	The core TDI signal.
`Tmscore`	The core TMS signal.
`Tdi`	The external JTAG TDI pin.
`tck`	The external JTAG TCK pin.
`Tms`	The external JTAG TMS pin.
`Tdouser`	This signal is not supported. Typically, it supports on-chip scan chains targeted from the `USER1` and `USER0` instructions. On-chip debugging applications use this signal; `USER1` and `USER0` scan chains are not supported when using the internal JTAG port.
Output Ports	
`Tdo`	The external port TDO pin.
`Tdocore`	The core TDO signal.
`Runidleuser`	These signals are not supported. Typically, they support on-chip scan chains targeted from the `USER1` and `USER0` instructions. On-chip debugging applications use these signals; `USER1` and `USER0` scan chains are not supported when using the internal JTAG port.
`shiftuser`	
`Tckutap`	
`tdiutap`	
`tmsutap`	
`Updateuser`	
`usr1user`	

Arria II GX JTAG Instruction Set and Target Registers

The following table describes the Arria II GX JTAG instructions with relevance to AES key management or anti-tamper features.

Table 8: Arria II GX JTAG Instruction Set and Target Registers

JTAG Instruction	Instruction Binary	Target Register
`KEY_CLR_VREG`	00 0010 1001	Bypass register
`KEY_DISABLE_JTAG`	00 0010 1010	Bypass register
`KEY_PROG_OTP`	11 0001 0000	AES key scan chain
`KEY_PROG_OTP_EN`	11 0001 0111	Bypass register
`KEY_PROG_VOL`	00 0010 1001	AES key scan chain
`KEY_SECURE_OTP`	00 1110 0101	Bypass register
`KEY_VERIFY`	00 0001 0011	Key verification scan registers
`PULSE_NCONFIG`	00 0000 0001	Bypass register
`VOL_KEY_SECURE`	00 1110 0110	Bypass register

Refer to the*Arria II GX Device Handbook* for information on other JTAG instructions.

Related Information

Arria II Device Handbook

KEY_CLR_VREG

The `KEY_CLR_VREG` instruction clears all registers related to the volatile key registers and deasserts the `vkeypgmd` bit from the volatile key verify block.

KEY_DISABLE_JTAG

The `KEY_DISABLE_JTAG` instruction burns the JTAG secure fuse.

Once the fuse is set, the JTAG state machine accepts only the following instructions :

- `BYPASS`
- `KEY_CLR_VREG`
- `KEY_PROG_VOL`
- `KEY_VERIFY`
- `PULSE_NCONFIG`

Note: This restriction applies to internal and external JTAG ports.

After issuing the `KEY_DISABLE_JTAG` instruction, the controller cycles the JTAG state machine to the RTI state and waits in this state for 10 µs. After a minimum of 10 µs in RTI, the controller cycles the JTAG state machine to test logic reset. You must reconfigure the device for the JTAG secure fuse to take effect.

KEY_PROG_OTP

The `KEY_PROG_OTP` instruction programs the non-volatile key fuses. You must issue the `KEY_PROG_OTP_EN` instruction prior to the execution of `KEY_PROG_OTP`.

Note: The `KEY_PROG_OTP` instruction is not supported from the internal JTAG interface.

Refer to *Using the Design Security Features in Altera FPGAs* for more information on programming the non-volatile key.

Related Information

Using the Design Security Features in Altera FPGAs

KEY_PROG_OTP_EN

You must use the `KEY_PROG_OTP_EN` instruction prior to the execution of the `KEY_PROG_OTP` instruction.

Note: The `KEY_PROG_OTP_EN` instruction is not supported from the internal JTAG interface.

Refer to *Using the Design Security Features in Altera FPGAs* for more information on programming the non-volatile key.

Related Information

Using the Design Security Features in Altera FPGAs

KEY_PROG_VOL

The `KEY_PROG_VOL` instruction programs the battery-backed AES key. Two 256 bit sequences are required to generate the 256 bit security key. The length of the key entered into the device is 512 bits. The RTI state requires an additional 400 TCK cycles after the 512 bits of key material are shifted into the key registers. No additional JTAG instructions are issued before 400 TCK clock cycles have elapsed. The `KEY_PROG_VOL` instruction is executable from the internal JTAG interface.

After the key is successfully programmed into the key registers, the `vkeypgmd` status bit from the key verify scan chain asserts high. You can access the key verify scan chain with the `KEY_VERIFY`JTAG instruction. The volatile key is active upon the next reconfiguration of the device.

KEY_SECURE_OTP

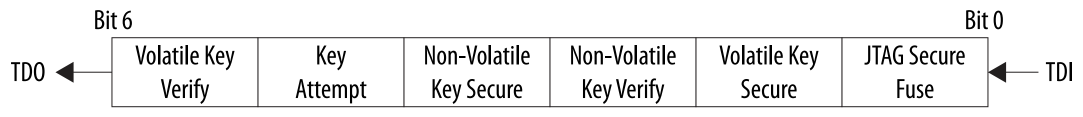
The `KEY_SECURE_OTP` instruction sets the non-volatile key secure fuse. Once set, the configuration engine allows only configuration files encrypted with the non-volatile key. Setting this fuse disables factory test modes. The `KEY_SECURE_OTP` instruction only issues if a non-volatile key is present.

After issuing the `VOL_KEY_SECURE` instruction, the controller cycles the JTAG state machine to the RTI state and waits in this state for 10 µs. After a minimum of 10 µs in RTI, the controller cycles the JTAG state machine to test logic reset. You must reconfigure the device for the non-volatile key secure fuse to take effect.

KEY_VERIFY

The `KEY_VERIFY` instruction targets the 6 bit key verify register. The following figure shows the register format:

Figure 8: Key Verify Register for Arria II GX Devices



KEY_VERIFY Register Status Bits

The following table describes the KEY_VERIFY status bits; all bits are active high.

Table 9: Key Verify Register Status Bits

Status Bit	Associated JTAG Instruction	Description
JTAG secure fuse	KEY_DISABLE_JTAG	This bit indicates that JTAG secure fuse is set. The instruction set is limited to the following instructions: • BYPASS • KEY_CLR_VREG • KEY_PROG_VOL • KEY_VERIFY • PULSE_NCONFIG Instruction set restrictions apply to both the internal and external JTAG ports.
Key attempt	KEY_PROG_OTP	This bit indicates if KEY_PROG_OTP has been issued. The bit asserts high whether or not the non-volatile key is programmed successfully. The configuration engine does not accept the KEY_PROG_OTP command when this bit is asserted.
Non-volatile key secure fuse	KEY_SECURE_OTP	This bit indicates if the non-volatile key secure fuse is set. When set, the non-volatile key secure fuse locks the device into only accepting configuration images encrypted with the non-volatile key. When KEY_SECURE_OTP or VOL_KEY_SECURE are set, the factory test mode is disabled.
Non-volatile key verify	KEY_PROG_OTP	This bit indicates if a non-volatile AES key is set.
Volatile key secure fuse	VOL_KEY_SECURE	This bit indicates if the volatile key secure fuse is set. When set, the volatile key secure fuse locks the device into only accepting configuration images encrypted with the volatile key. When KEY_SECURE_OTP or VOL_KEY_SECURE are set, the factory test mode is disabled.

Status Bit	Associated JTAG Instruction	Description
Volatile key verify	`KEY_PROG_VOL`	This bit indicates if an AES key is present in battery-backed RAM.

PULSE_NCONFIG

The `PULSE_NCONFIG` instruction resets the device and clears the embedded memory.

VOL_KEY_SECURE

The `VOL_KEY_SECURE` instruction sets the volatile key secure fuse. Once set, the configuration engine allows only configuration files encrypted with the volatile key. Setting this fuse disables factory test modes. The `VOL_KEY_SECURE` instruction is only issued if a volatile AES key is present.

After issuing the `VOL_KEY_SECURE` instruction, the controller cycles the JTAG state machine to the RTI state and waits in this state for 10 µs. After a minimum of 10 µs in RTI, the controller cycles the JTAG state machine to test logic reset. The device must be reconfigured for the volatile key secure fuse to take effect.

Arria II GX CRC Error Detection

The CRC error detection circuit continuously scans the configuration logic to check for any errors in the configuration RAM. You can enable this function in the Quartus II software under the **Device and Pin Options** dialog box. The primitive opens up additional ports to read the 32 bit CRC signature from the configuration engine.

Example 7: CRC Error Detection Component Declaration

```
component arriaii crcblock
    generic (
        lpm_hint             :    string := "UNUSED";
        lpm_type             :    string := "stratixv_crcblock";
        oscillator_divider    :    natural := 256    );
    port(
        clk                    :    in std_logic := '0';
        crcerror             :    out std_logic;
        regout               :    out std_logic;
        shiftnld             :    in std_logic := '0'
    );
end component;
```

Refer to the *Arria II Device Handbook* and *Test Methodology of Error Detection and Recovery using CRC in Altera FPGA Devices* for further information on the CRC error detection engine.

Related Information

- **Arria II Device Handbook**
- **Test Methodology of Error Detection and Recovery using CRC in Altera FPGA Devices**

Arria II GX CRC Error Detection Port and Parameter Description

Table 10: CRC Error Detection Port and Parameter Description

Port/Parameter	Description
Parameters	
`oscillator_divider`	This signal is the clock divider for the internal oscillator feeding the CRC circuit. The valid values are from 1 to 256.
Input Ports	
`clk`	This signal designates the clock input of the cell. The cell operates in relation to the rising edge of the clock; both the loading of the data into the cell or data out of the cell occurs on the rising edge. This is a required port.
`shiftnld`	This signal is an input to the error detection block. If `shiftnld`=1, the user shift register shifts from the internal shift register to the `regout` at each rising edge of `clk`. If `shiftnld`=0, the shift register parallel loads the contents of the user update register. This is a required port. This input triggers clock enable for the user update register to deassert after two `EDCLK` cycles. After driving the `SHIFTNLD` signal low, you must wait at least two `EDCLK` cycles before clocking the `ED_CLK` signal.
Output Ports	
`Crcerror`	This signal is the output of the cell synchronized to the internal oscillator of the device (60MHz, nominal) and not to the `clk` port. It asserts automatically high if the error block detects that an SRAM bit has flipped and the internal CRC computation shows a difference in the pre-computed value. This signal must be connected to an output pin or a bidirectional pin. When `Crcerror` is connected to an output pin, you can only monitor the `CRC_ERROR` pin (the core cannot access this output). If core logic uses the `CRC_ERROR` signal to read error detection logic, this signal must be connected to a `BIDIR` pin. The signal is fed to the core indirectly by feeding a `BIDIR` pin that has its output enable port connected to VCC.
`regout`	This signal is the output of the error detection shift register synchronized to the `clk` port; it is read by core logic. It shifts one bit each cycle.

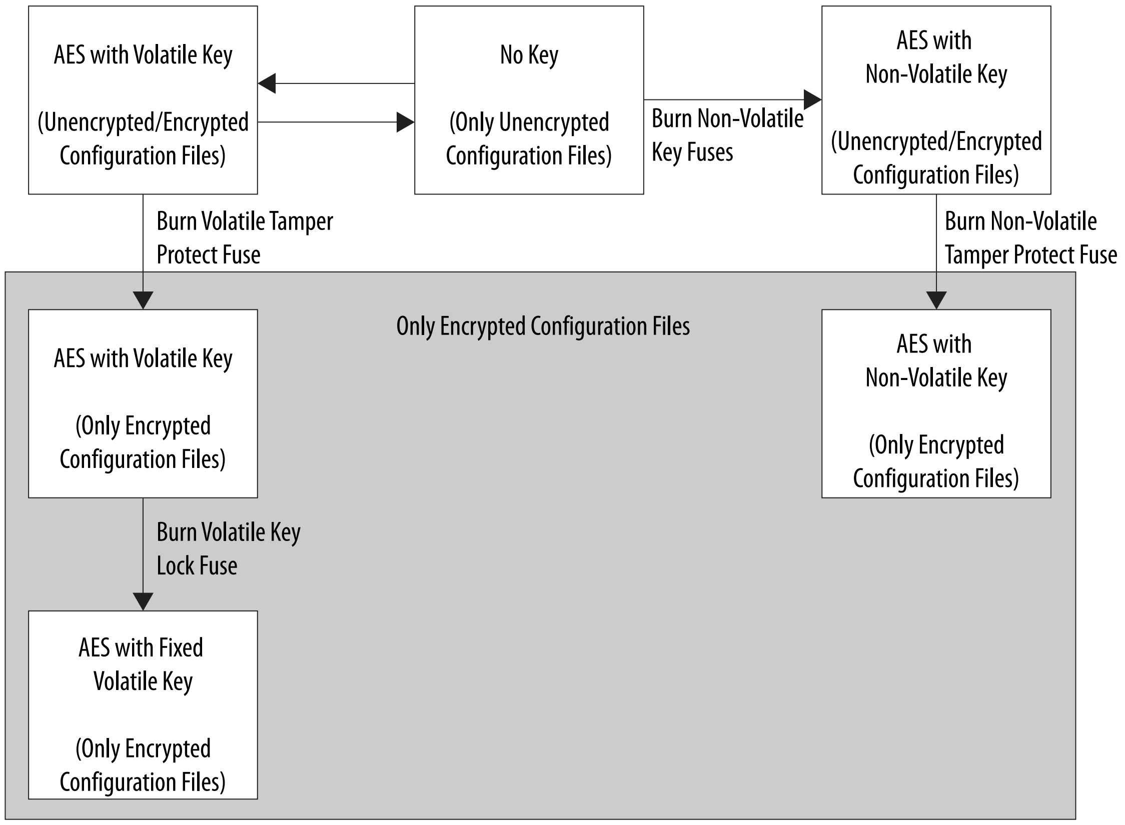
Arria V, Cyclone V, and Stratix V

The 28 nm device families support the following anti-tamper features:

- AES-256 encrypted configuration files
- Internally accessible JTAG interface, with an instruction set for AES key management and status reporting
- JTAG port prevention
- CRC error detection
- Built in configuration RAM error correction
- Temperature sensing diode (TSD) (Arria V and Stratix V devices only)
- Volatile key urgent clear pin
- Unique device ID

28 nm devices support the use of two different types of keys for configuration file encryption: a non-volatile polyfuse based key, and a battery-backed volatile key. Only a single key is present on the device at any given time. There are three option bits, each set by a set of polyfuses, that allow various modes of operation for decryption; two of the bits are tamper protect fuses and the other bit is a volatile key lock fuse. The tamper protect fuses lock the device into accepting only encrypted files with non-volatile keys or volatile keys. The volatile key lock fuse disables re-programmability of the volatile key. The following figure shows the operational modes for configuration file decryption in 28nm devices.

Figure 9: 28 nm Device Encryption Modes - Sequence and Restrictions


AES with Volatile Key
(Unencrypted/Encrypted Configuration Files)
No Key
(Only Unencrypted Configuration Files)
Burn Non-Volatile Key Fuses
AES with Non-Volatile Key
(Unencrypted/Encrypted Configuration Files)
Burn Volatile Tamper Protect Fuse
Burn Non-Volatile Tamper Protect Fuse
Only Encrypted Configuration Files
AES with Volatile Key
(Only Encrypted Configuration Files)
AES with Non-Volatile Key
(Only Encrypted Configuration Files)
Burn Volatile Key Lock Fuse
AES with Fixed Volatile Key
(Only Encrypted Configuration Files)

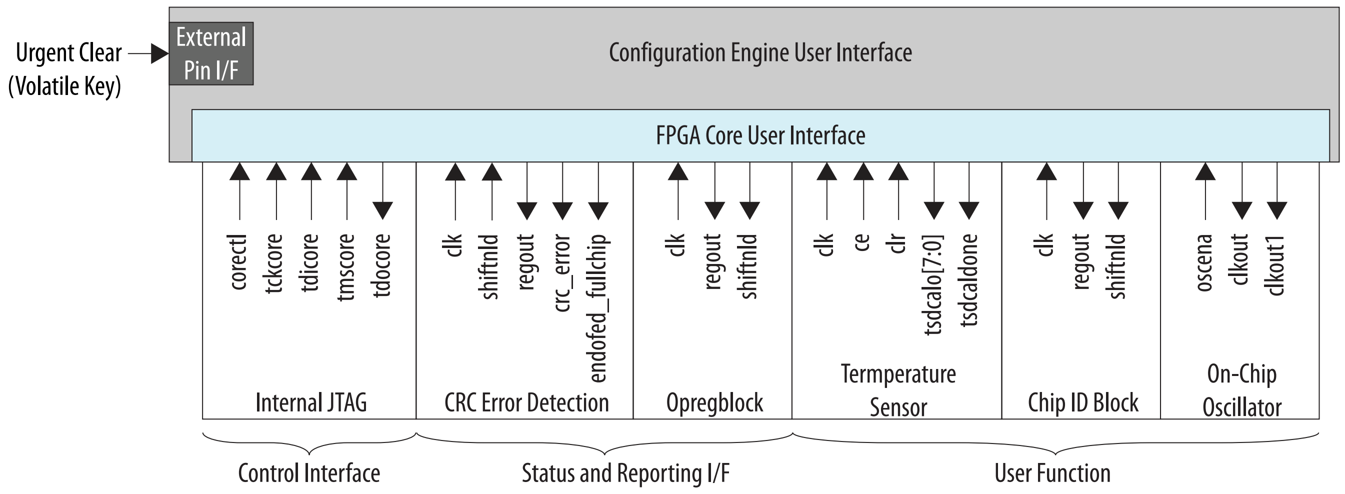
Refer to the *Arria V, Cyclone V, and Stratix V JTAG Instruction Set* for further details on the JTAG instructions for setting the polyfuse values.

Note: Programming a non-volatile key prevents the use of a battery-backed volatile key.

Caution: Use extreme caution when setting the volatile tamper protect fuse or the volatile key lock fuse through their respective JTAG instructions. Either fuse can limit the ability to program a new key. The volatile tamper protect fuse locks the device to encrypted configuration files and restricts the instruction set available from the external JTAG interface. The volatile key lock fuse restricts the ability to reprogram the volatile key. If key material is lost when the device is in a state where new key material cannot be reprogrammed, the device could be rendered unusable.

Refer to *Arria V, Cyclone V, and Stratix V JTAG Port Protection* for further details on JTAG port protection.

Figure 10: Configuration Engine User Interface for 28nm Device Families



Refer to the *Arria V Device Handbook*, *Cyclone V Device Handbook*, *Stratix V Device Handbook*, and *Using the Design Security Features in Altera FPGAs* for more information on the design security feature in Arria II devices.

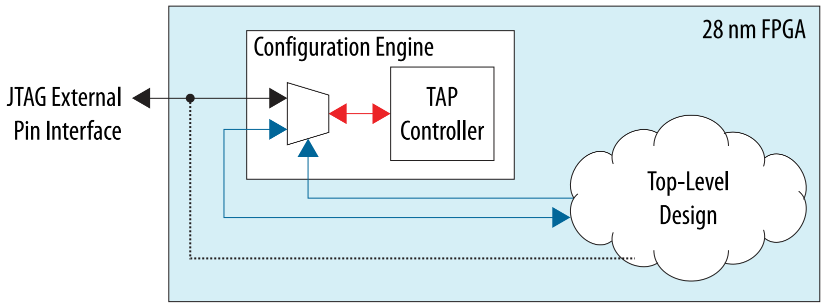
Related Information

- **Arria V, Cyclone V, and Stratix V JTAG Instruction Set and Target Registers** on page 29
- **Arria V, Cyclone V, and Stratix V JTAG Port Protection** on page 25
- **Arria V Device Handbook**
- **Cyclone V Device Handbook**
- **Stratix V Device Handbook**
- **Using the Design Security Features in Altera FPGAs**

Arria V, Cyclone V, and Stratix V JTAG Block

The system design drives the JTAG TAP controller completely from within the FPGA device. A single port interface, either internal or external, is active at a given time. The external pin interface connects to the JTAG TAP controller through dedicated routing, but there are routing stubs from the route that branch off into the core routing fabric. The system design can sample the dedicated input pins (`TCK`, `TDI`, `TMS`, or `TRST`) to drive the dedicated output port; the system design can choose to maintain the JTAG chain continuity by routing the signal from the TDI pin to the TDO pin.

Figure 11: JTAG TAP Controller Access, 28 nm Devices



Arria V, Cyclone V, and Stratix V JTAG Port Protection

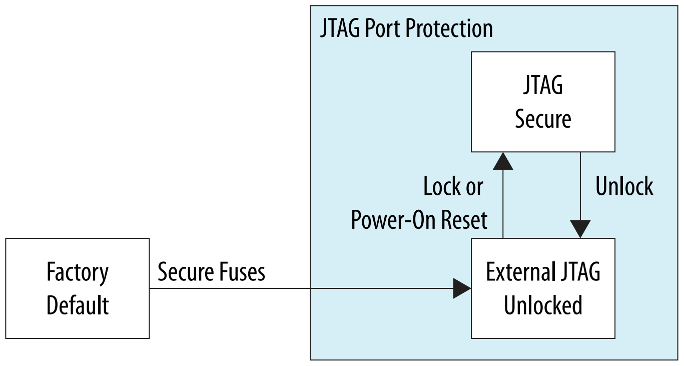
Arria V, Cyclone V, and Stratix V devices support JTAG secure mode. The JTAG instruction set includes the following instructions from the external JTAG port:

- `BYPASS`
- `SAMPLE/PRELOAD`
- `EXTEST`
- `IDCODE`

For Stratix V and Cyclone V devices, setting either of the `KEY_SECURE` fuses enables JTAG port protection on the external interface. Once enabled, external port behavior is controlled by two JTAG instructions, `UNLOCK` and `LOCK`. The `UNLOCK` and `LOCK` commands must be issued by the internal JTAG port interface; these instructions are not available over the external pin interface. A successful `UNLOCK` instruction allows the full JTAG instruction set from the external port interface. Conversely, a successful `LOCK` command re-enables JTAG port protection. A power-on reset (POR) also re-enables JTAG port protection. The JTAG secure mode does not restrict the instruction set from the internal port interface.

Refer to the *Arria V, Cyclone V, and Stratix V JTAG Instruction Set* for further information on the instruction set.

Figure 12: JTAG Port Protection Functional Model for 28 nm Device Families



Related Information

Arria V, Cyclone V, and Stratix V JTAG Instruction Set and Target Registers on page 29

Arria V, Cyclone V, and Stratix V Internal JTAG Block Access

You can access the internal JTAG port through instantiation of the JTAG primitive. The internal port is active when the JTAG internal access option bit is set and the FPGA device design asserts `corectl`. The Quartus II software does not support the full atom by default. To enable compilation support of the full JTAG atom for the appropriate device family, the following **.qsf** assignments must be set:

```
set_global_assignment -name INI_VARS "<JTAG_QSF_Assigment>=on"
```

Device Family	.qsf Assignment for JTAG Compilation Support
Arria V	`sgn_enable_arriav_jtag_core_access`
Cyclone V	`sgn_enable_cyclonev_jtag_core_access`
Stratix V	`sgn_enable_stratixv_jtag_core_access`

Set the JTAG **.qsf** assignment to enable compilation support for the JTAG instantiation template provided in *Arria II GX Internal JTAG Block Access*. If you are converting the programming file to a different file format (for example, a **.rbf** or a **.pof** file), you should convert in a project directory that has the appropriate **.qsf** assignment set.

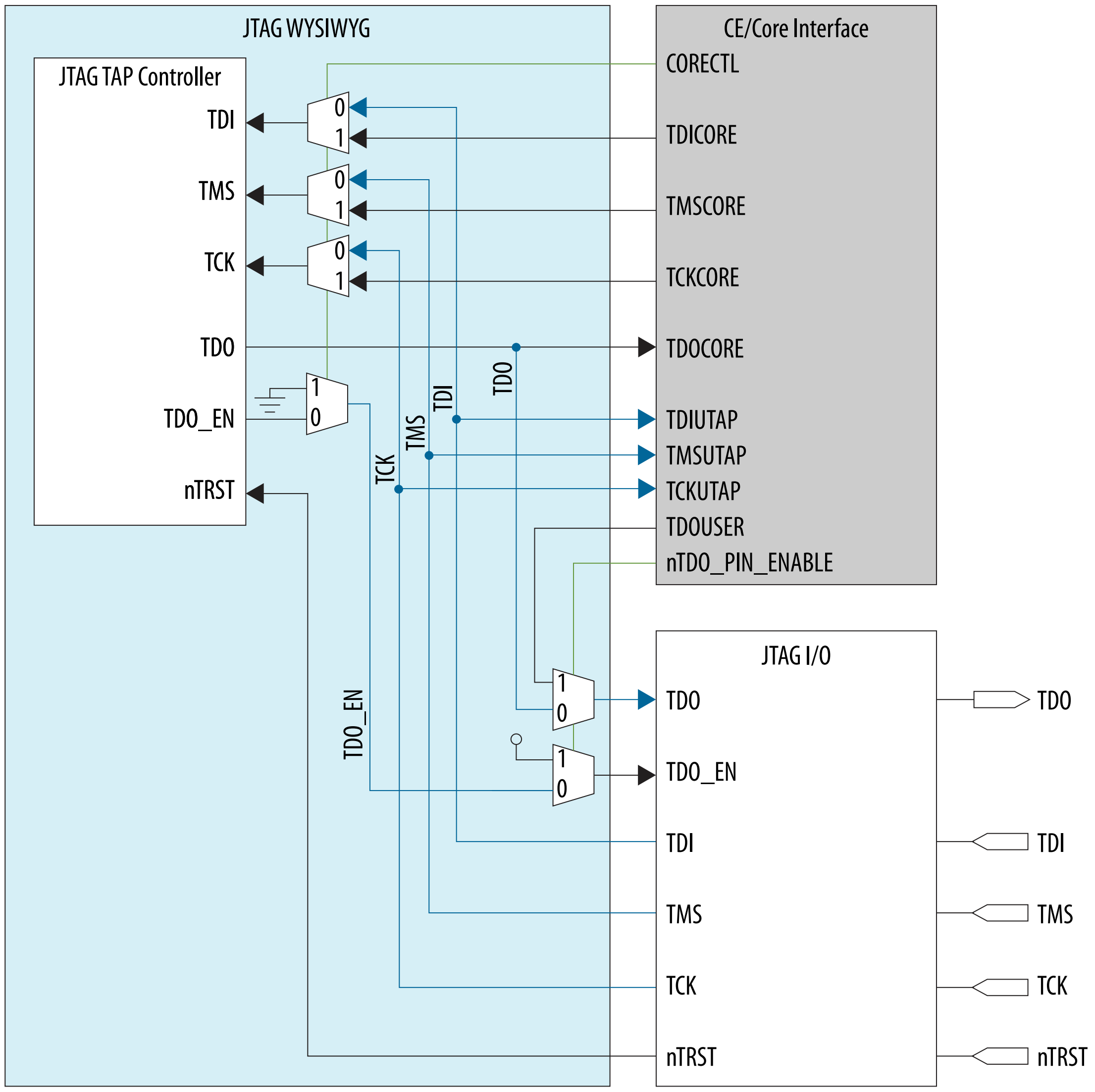
Refer to the JTAG chapters in the *Arria V Device Handbook*, *Cyclone V Device Handbook*, and *Stratix V Device Handbook* for the JTAG port interface timing characteristics.

Note: If you are instantiating the JTAG atom for internal controller access, all functions requiring the use of the JTAG port must be removed from your design; this includes all on-chip debugging tools and anything relying on the JTAG UART from SOPC Builder or Qsys. Instantiating this function and another resource that uses the external JTAG port causes a resource conflict resulting in a compilation error.

Example 8: JTAG Controller Block (Internal Access) Component Declaration

```
component <device_name>_jtag
    generic (
        lpm_hint            :   string := "UNUSED";
        lpm_type            :   string := " <device_name>_jtag");
    port(
        clkdruser           :   out std_logic;
        ntrst            :   in  std_logic;
        corectl             :   in std_logic := '0';
        runidleuser      :   out std_logic;
        shiftuser           :   out std_logic;
        tck                    :   in std_logic := '0';
        tckcore             :   in std_logic := '0';
        tckutap             :   out std_logic;
        tdi                    :   in std_logic := '0';
        tdicore             :   in std_logic := '0';
        tdiutap             :   out std_logic;
        tdo                    :   out std_logic;
        tdocore             :   out std_logic;
        tdouser             :   in std_logic := '0';
        tdoutap             :   in std_logic := '0';
        tms                    :   in std_logic := '0';
        tmscore             :   in std_logic := '0';
        tmsutap             :   out std_logic;
        updateuser       :   out std_logic;
        usr1user            :   out std_logic;
        ntdopinena       :   in std_logic := '1';
    );
end component;
```

Figure 13: 28 nm JTAG Atom Primitive



Related Information

- Arria II GX Internal JTAG Block Access on page 16
- Arria V Device Handbook
- Cyclone V Device Handbook
- Stratix V Device Handbook

Arria V, Cyclone V, and Stratix V JTAG Atom Port and Parameter Description

Table 11: Arria V, Cyclone V, and Stratix V JTAG Atom Port and Parameter Description

Port/Parameter	Description
Input Ports	
`Corectl`	This signal enables the JTAG port from the configuration engine to the FPGA device core. The signal is active high. The device resets when the external port is enabled.
`nTRST`	The external JTAG test reset (TRST) pin.
`Tckcore`	The core TCK signal.
`Tdicore`	The core TDI signal.
`Tmscore`	The core TMS signal.
`Tdi`	The external JTAG TDI pin.
`tck`	The external JTAG TCK pin.
`Tms`	The external JTAG TMS pin.
`Tdouser`	This port is directly connected to the `TDO` output pin. `ntdopinena`, when asserted low, selects `tdouser` as the active signal path.
Output Ports	
`Tdo`	The external port TDO pin.
`Tdocore`	The core TDO signal.
`Runidleuser`	These signals are not supported. Typically, they support on-chip scan chains targeted from the `USER1` and `USER0` instructions. On-chip debugging applications use these signals; `USER1` and `USER0` scan chains are not supported when using the internal JTAG port.
`shiftuser`	
`Tckutap`	The routing stub from the `TCK` pin to the core logic. It samples the external `TCK` pin. This routing stub is active if the external interface is disconnected from the JTAG TAP controller.
`tdiutap`	The routing stub from `TDI` pin to the core logic. It samples the external `TDI` pin. This routing stub is active if the external interface is disconnected from the JTAG TAP controller.
`tmsutap`	The routing stub from `TMS` pin to the core logic. It samples the external `TMS` pin. This routing stub is active if the external interface is disconnected from the JTAG TAP controller.

Port/Parameter	Description
Updateuser	These signals are not supported. Typically, they support on-chip scan chains targeted from the USER1 and USER0 instructions. On-chip debugging applications use these signals; USER1 and USER0 scan chains are not supported when using the internal JTAG port.
usr1user	

Arria V, Cyclone V, and Stratix V JTAG Instruction Set and Target Registers

The following table describes the Arria V, Cyclone V, and Stratix V JTAG instructions with relevance to AES key management or anti-tamper features. The external JTAG interface issues all of the following JTAG instructions, except LOCK and UNLOCK.

Table 12: 28 nm JTAG Instruction Set and Target Registers

JTAG Instruction	Instruction Binary	Target Register	Accessible from Core
IP_RECONFIG	11 0011 1101	Bypass register	Yes
KEY_CLR_VREG	00 0010 1001	Clears Volatile Key Register	Yes
KEY_PROG_VOL	01 1010 1101	AES key scan chain	Yes
KEY_SECURE_OTP	00 1110 0101	Bypass register	Yes
KEY_VERIFY	00 0001 0011	Key verification scan registers	Yes
LOCK	01 1111 0000	Bypass register	Yes
KEY_PROG_OTP_EN	11 0001 0111	Bypass register	No
KEY_PROG_OTP	11 0001 0000	AES key scan chain	No
TEST_DISABLE	00 0010 1010	Bypass register	No
UNLOCK	11 0011 0001	Bypass register	Yes
VOL_KEY_LOCK	01 0100 0001	Bypass register	Yes
VOL_KEY_LOCK_EN	10 1011 1001	Bypass register	Yes
VOL_KEY_SECURE	00 1110 0110	Bypass register	Yes
VOL_KEY_ZERO	00 0011 0011	AES key scan chain	Yes

Refer to the *Arria V Device Handbook*, *Cyclone V Device Handbook*, and *Stratix V Device Handbook* for information on other JTAG instructions. Instructions not explicitly defined in this document should be treated as Reserved.

Related Information

- Arria V Device Handbook
- Cyclone V Device Handbook
- Stratix V Device Handbook

IP_RECONFIG

The `IP_RECONFIG` instruction triggers a reconfiguration of the device. To successfully execute a reconfiguration, `IP_RECONFIG` must be followed by a different JTAG instruction, such as `BYPASS` or `IDCODE`.

KEY_CLR_VREG

The `KEY_CLR_VREG` instruction clears all registers related to the volatile key registers and deasserts the `vkeypgmd` bit from the volatile key verify block.

KEY_PROG_OTP

The `KEY_PROG_OTP` instruction programs the non-volatile key fuses. You must use the `KEY_PROG_OTP_EN` command prior to issuing the `KEY_PROG_OTP` instruction. The internal JTAG interface does not support `KEY_PROG_OTP`.

Refer to *Using the Design Security Features Altera FPGAs* for instructions on programming the non-volatile key fuses.

Related Information

Using the Design Security Features Altera FPGAs

KEY_PROG_OTP_EN

You must use the `KEY_PROG_OTP_EN` instruction prior to issuing the `KEY_PROG_OTP` command. The internal JTAG interface does not support `KEY_PROG_OTP_EN`.

Refer to *Using the Design Security Features Altera FPGAs* for instructions on programming the non-volatile key fuses.

Related Information

Using the Design Security Features Altera FPGAs

KEY_PROG_VOL

The `KEY_PROG_VOL` instruction programs the battery-backed AES key. The user provided key is 256 bits in length. 28 nm devices require the 256 bit user provided final encryption key and a hard-coded key.

For more information related to programming the volatile key from within the device core please contact Altera.

KEY_SECURE_OTP

The KEY_SECURE_OTP instruction sets the non-volatile tamper protect fuse. The set fuse enables the following:

Note: For more information related to programming the volatile key from within the device core please contact Altera for additional technical support.

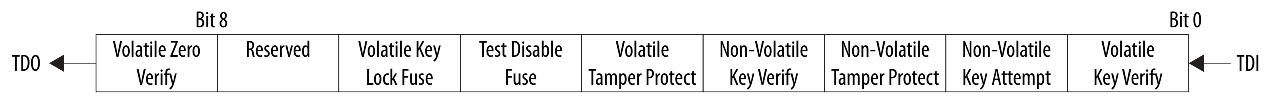
- Encryption of all configuration files with the non-volatile AES key
- Enables JTAG secure mode

The KEY_SECURE_OTP instruction only issues when a non-volatile key is present. After issuing the KEY_SECURE_OTP instruction, the controller cycles the JTAG state machine to the RTI state and waits in this state for 10 µs. After a minimum of 10 µs in RTI, the controller cycles the JTAG state machine to test logic reset. The device must be reconfigured for the non-volatile key secure fuse to take effect.

KEY_VERIFY

The KEY_VERIFY instruction targets the 6 bit key verify register. The following figure shows the register format:

Figure 14: Key Verify Register for 28 nm Devices



KEY_VERIFY Register Status Bits

The following table describes the KEY_VERIFY status bits; all bits are active high.

Table 13: Key Verify Register Status Bits

Status Bit	Associated JTAG Instruction	Description
Non-volatile key attempt	KEY_PROG_OTP	This bit indicates that execution of the KEY_PROG_OTP instruction was attempted.
Non-volatile key verify	KEY_PROG_OTP	This bit indicates successful execution of the KEY_PROG_OTP instruction.
Non-volatile tamper protect	KEY_SECURE_OTP	This bit indicates if the non-volatile tamper protect fuse is set. When set, the non-volatile tamper protect fuse locks the device into only accepting configuration images encrypted with the non-volatile key.
Test disable fuse	TEST_DISABLE	This bit indicates if the test disable fuse is set. The test disable fuse disables all device manufacturing test modes.

Status Bit	Associated JTAG Instruction	Description
Volatile clear verify	`VOL_KEY_CLEAR`	This bit indicates successful execution of the `VOL_KEY_CLEAR` instruction. All key material and AES registers are initialized to known values. **Note:** The `VOL_KEY_CLEAR` instruction also sets the volatile key verify bit.
Volatile key lock fuse	`VOL_KEY_LOCK`	This bit indicates if the volatile key lock fuse is set. Once set, the configuration engine does not support reprogramming of the battery-backed AES key.
Volatile key verify	`KEY_PROG_VOL` , `VOL_KEY_CLEAR`	This bit indicates if an AES key is present in battery-backed RAM. Successful execution of `KEY_PROG_VOL` or `VOL_KEY_CLEAR` sets the volatile key verify bit.
Volatile tamper protect fuse	`VOL_KEY_SECURE`	This bit indicates if the volatile tamper protect fuse is set. When set, the volatile tamper protect fuse locks the device into only accepting configuration images encrypted with the volatile key.

LOCK

The `LOCK` instruction re-enables JTAG secure mode. If any of the tamper protect fuses are set, issuing `LOCK` limits the JTAG instruction set available from the external port interface. Only the following instructions are allowed from the external port:

- `BYPASS`
- `SAMPLE/PRELOAD`
- `EXTEST`
- `IDCODE`

The `LOCK` instruction only issues from the core JTAG interface.

Refer to *Arria V, Cyclone V, and Stratix V JTAG Port Protection* for information on the functional model of the JTAG secure mode.

Related Information

Arria V, Cyclone V, and Stratix V JTAG Port Protection on page 25

TEST_DISABLE

The `TEST_DISABLE` instruction disables the manufacturer's test mode; Altera uses a built-in self test (BIST) for device qualification. The core JTAG interface cannot issue `TEST_DISABLE`.

After issuing the `TEST_DISABLE` instruction, the controller cycles the JTAG state machine to the RTI state and waits in this state for 10 µs. After a minimum of 10 µs in RTI, the controller cycles the JTAG state machine to test logic reset. The device must be reconfigured for `TEST_DISABLE` to take effect.

UNLOCK

The `UNLOCK` instruction enables the full instruction set from the external JTAG port, when JTAG secure mode is enabled.

Arria V, Cyclone V, and Stratix V JTAG Port Protection describes the functional model for the JTAG secure mode; the `UNLOCK` instruction only issues from the core JTAG interface.

Related Information

Arria V, Cyclone V, and Stratix V JTAG Port Protection on page 25

VOL_KEY_LOCK

The `VOL_KEY_LOCK` instruction sets the volatile key lock fuse bit. When set, the volatile key lock fuse bit prevents any further programming of the volatile key. This fuse can only be set when the volatile tamper protect fuse is set.

Note: There is a potential of rendering the device unusable when the volatile key lock fuse bit is set. If the volatile key is lost for any reason, the device does not accept a new volatile key. A new configuration cycle is not allowed because only encrypted images are accepted.

You must first issue the `VOL_KEY_LOCK_EN` instruction to successfully set `VOL_KEY_LOCK`.

After issuing the `VOL_KEY_LOCK` instruction, the controller cycles the JTAG state machine to the RTI state and waits in this state for 10 µs. After a minimum of 10 µs in RTI, the controller cycles the JTAG state machine to test logic reset. The device must be reconfigured for the volatile key lock fuse to take effect.

VOL_KEY_LOCK_EN

The `VOL_KEY_LOCK_EN` instruction enables the `VOL_KEY_LOCK` instruction. You must issue `VOL_KEY_LOCK_EN` prior to the `VOL_KEY_LOCK` instruction to prevent accidental setting of the volatile key lock fuse bit.

VOL_KEY_SECURE

The `VOL_KEY_SECURE` instruction sets the volatile tamper protect fuse. When set, the volatile tamper protect enables the following:

- Encryption of all configuration files with the non-volatile AES key
- JTAG secure mode

You can only issue the `VOL_KEY_SECURE` instruction when a volatile AES key is present.

After issuing the `VOL_KEY_SECURE` instruction, the controller cycles the JTAG state machine to the RTI state and waits in this state for 10 µs. After a minimum of 10 µs in RTI, the controller cycles the JTAG state machine to test logic reset. You must reconfigure the device for the volatile key secure fuse to take effect.

Note: Loss of the volatile key when the volatile tamper protect fuse is set renders the device unusable. When set, the volatile tamper protect fuse enables JTAG secure mode and locks the device into accepting only configuration files encrypted with a volatile key. JTAG secure mode disables key programming instructions from the external JTAG port. You can only reprogram key material through a configuration image with access to the internal JTAG port. The configuration image directly programs in new key material using the JTAG instructions for programming the key, or unlocks the external JTAG port so that an external loader can provide key material. If the volatile key is lost for any reason, the device does not accept a new volatile key. A new configuration cycle is not allowed; only encrypted images are accepted.

VOL_KEY_ZERO

The `VOL_KEY_ZERO` instruction is similar to the `KEY_PROG_VOL` instruction, except that it sets the battery-backed AES key to zero and verifies that both the battery-backed AES key and the AES block round keys are properly cleared.

Note: Setting the `VOL_KEY_LOCK` fuse disables the `VOL_KEY_ZERO` instruction.

Setting VOL_KEY_ZERO

For information related to clearing the volatile key from within the device core please contact Altera.

Arria V, Cyclone V, and Stratix V On-Chip Oscillator

The on-chip oscillator is the internal chip oscillator from the configuration engine used during active configuration modes. This clock output does not pass through the clock divisor. Nominal frequency of the output is 80 MHz, but output frequency varies from 50 MHz to 100 MHz due to variations in process, voltage, and temperature.

Example 9: On-Chip Oscillator Component Declaration

```
component stratixv_oscillator
    generic (
        lpm_hint    :    string := "UNUSED";
        lpm_type    :    string := " stratixv_oscillator"    );
    port(
        clkout      :    out std_logic;
        clkout1     :    out std_logic;
        oscena      :    in std_logic := '0'
    );
end component;
```

Arria V, Cyclone V, and Stratix V On-Chip Oscillator Port and Parameter Description

Table 14: On-Chip Oscillator Port and Parameter Description

Port/Parameter	Description
Input Ports	
`Oscena`	This is the user enable signal for the core to turn off `OSC_CORE` output. The oscillator still runs internally. This signal is active high.
Output Ports	
`clkout`	The internal oscillator output signal (50 to 100 MHz range).
`clkout2`	This is a redundant signal; it has the same functionality as `clkout`.

Arria V, Cyclone V, and Stratix V CRC Error Detection

The CRC error detection circuit continuously scans the configuration logic to check for any errors in the configuration RAM. You can enable this function in the Quartus II software under the **Device and Pin**

Options dialog box. The primitive opens up additional ports to read the 32 bit CRC signature from the configuration engine.

Example 10: CRC Error Detection Component Declaration

```
component <device_name>_crcblock
    generic (
        lpm_hint        :    string := "UNUSED";
        lpm_type        :    string := "stratixv_crcblock";
        oscillator_divider    :    natural := 256    );
    port(
        clk                :    in std_logic := '0';
        crcerror        :    out std_logic;
        regout          :    out std_logic;
        shiftnld        :    in std_logic := '0'
    );
end component;
```

Refer to *Test Methodology of Error Detection and Recovery using CRC in Altera FPGA Devices* for further information on the CRC error detection engine.

Related Information

Test Methodology of Error Detection and Recovery using CRC in Altera FPGA Devices

Arria V, Cyclone V, and Stratix V CRC Error Detection Port and Parameter Description

Table 15: CRC Error Detection Port and Parameter Description

Port/Parameter	Description
	Parameters
`oscillator_divider`	This signal is the clock divider for the internal oscillator feeding the CRC circuit. The valid values are from 1 to 256.
	Input Ports
`clk`	This signal designates the clock input of the cell. The cell operates in relation to the rising edge of the clock; both the loading of the data into the cell or data out of the cell occurs on the rising edge. This is a required port.
`shiftnld`	This signal is an input to the error detection block. If `shiftnld`=1, the user shift register shifts from the internal shift register to the `regout` at each rising edge of `clk`. If `shiftnld`=0, the shift register parallel loads the contents of the user update register. This is a required port. This input triggers clock enable for the user update register to deassert after two `EDCLK` cycles. After driving the `SHIFTNLD` signal low, you must wait at least two `EDCLK` cycles before clocking the `ED_CLK` signal.

Port/Parameter	Description
Output Ports	
`Crcerror`	This signal is the output of the cell that is synchronized to the internal oscillator of the device (60MHz, nominal) and not to the `clk` port. It asserts automatically high if the error block detects that an SRAM bit has flipped and the internal CRC computation shows a difference in the pre-computed value. This signal must be connected to an output pin or a bidirectional pin. When `Crcerror` is connected to an output pin, you can only monitor the `CRC_ERROR` pin (the core cannot access this output). If core logic uses the `CRC_ERROR` signal to read error detection logic, this signal must be connected to a `BIDIR` pin. The signal is fed to the core indirectly by feeding a `BIDIR` pin that has its output enable port connected to VCC.
`regout`	This signal is the output of the error detection shift register synchronized to the `clk` port; it is read by core logic. It shifts one bit each cycle; you should clock the `clk` signal 31 cycles to read out the 32 bits of the shift register.

Arria V, Cyclone V, and Stratix V Option Register Block Interface

The FPGA configuration file contains header information that determines various operational modes of the FPGA. The option register is 440 bits wide. The shift register is a read-only interface that allows an application to monitor the integrity of the option register bits. The option register is read out from a 440 bit circular shift register by a three pin serial interface.

Example 11: Option Register Block Component Declaration

```
component <device_name>_opregblock
    generic (
        lpm_type    :    string := "stratixv_opregblock"    );
    port(
        clk            :    in std_logic := '0';
        regout      :    out std_logic;
        shiftnld    :    in std_logic := '0'
    );
end component;
```

Arria V, Cyclone V, and Stratix V Option Register Block Interface

Table 16: Option Register Block Port Interface Description

Port/Parameter	Description
Input Ports	

Port/Parameter	Description
`clk`	This signal designates the clock input of the cell. The cell operates in relation to the rising edge of the clock.
`shiftnld`	This signal is an input into the option register block. If `shiftnld=1`, the user shift register shifts data to the `regout` at each rising edge of `clk`. If `shiftnld=0`, the 440 bit shift register resets to the initial value.
Output Ports	
`regout`	This signal is the output of the option bit shift register synchronized to the `clk` port; it is read by core logic. It shifts one bit each cycle.

Arria V, Cyclone V, and Stratix V Unique Chip ID Block

The chip ID block has a 64 bit unique ID per die.

The chip ID block has a 64 bit unique ID per die. The unique chip ID is read out from a 90 bit circular shift register by a three pin serial interface. The initial 64 bits contain the unique ID value. The last 26 bits are a concatenation of various fuse bits set during the manufacturing flow; these bits have Altera reserved values.

The Unique Chip ID register is implemented as a barrel shift register. The Chip ID function is accessible through instantiation of the ALTCHIP_ID Megafunction. Refer to the ALTCHIP_ID Megafunction User Guide for detailed information on using the Unique chip ID.

Related Information

ALTCHIP_ID Megafunction

Temperature Sensing Diode

The TSD is a PN junction diode that detects the die temperature. An internal analog-to-digital converter (ADC) circuit is accessible to the core routing fabric. The Temperature Sensor (ALTTEMP_SENSE) Megafunction is available to monitor the temperature from the core.

Refer to the *Temperature Sensor (ALTTEMP_SENSE) Megafunction User Guide* for detailed information on using the TSD for Arria II GX and Stratix V devices.

Related Information

Temperature Sensor (ALTTEMP_SENSE) Megafunction User Guide

Document Revision History

Date	Changes
March 2017	Corrected links to device handbooks.
January 2015	Updated the Cyclone V device `.qsf` assignment in *Arria V, Cyclone V, and Stratix V Internal JTAG Block Access.*

Date	Changes
December 2013	Initial release.

Related Information

Arria V, Cyclone V, and Stratix V Internal JTAG Block Access on page 26