## Contents

1. Mailbox Avalon® ST Client Intel FPGA IP Overview ................................................................. 3
   1.1. Device Family Support ........................................................................................................... 4
       1.1.1. Parameters .................................................................................................................. 5
       1.1.2. Interfaces .................................................................................................................... 5
   1.2. Command and Response Header ......................................................................................... 7
   1.3. Supported Commands .......................................................................................................... 8
   1.4. Error Codes .......................................................................................................................... 12
   1.5. Document Revision History for the Mailbox Avalon ST Client Intel FPGA IP User Guide .. 13
1. Mailbox Avalon® ST Client Intel FPGA IP Overview

The Mailbox Avalon® ST Client Intel® FPGA IP provides a communication channel between your custom logic and the secure device manager (SDM). You can use the Mailbox Avalon ST Client IP to send command packets and receive response packets from SDM peripheral modules. The Mailbox Avalon ST Client IP defines functions that the SDM runs.

Your custom logic can use this communication channel to receive information and access flash memory from the following peripheral modules:

- The Chip ID
- The Temperature Sensor
- The Voltage Sensor
- Quad serial peripheral interface (SPI) flash memory

The following figure shows an application in which the Avalon ST Client Intel FPGA IP reads the Chip ID.
1.1. Device Family Support

The following lists the device support level definitions for Intel FPGA IPs:

- **Advance support** — The IP is available for simulation and compilation for this device family. Timing models include initial engineering estimates of delays based on early post-layout information. The timing models are subject to change as silicon testing improves the correlation between the actual silicon and the timing models. You can use this IP for system architecture and resource utilization studies, simulation, pinout, system latency assessments, basic timing assessments (pipeline budgeting), and I/O transfer strategy (data-path width, burst depth, I/O standards tradeoffs).

- **Preliminary support** — The IP is verified with preliminary timing models for this device family. The IP meets all functional requirements, but might still be undergoing timing analysis for the device family. It can be used in production designs with caution.

- **Final support** — The IP is verified with final timing models for this device family. The IP meets all functional and timing requirements for the device family and can be used in production designs.

<table>
<thead>
<tr>
<th>Device Family</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Agilex™</td>
<td>Advance</td>
</tr>
</tbody>
</table>

*Note:* Intel does not provide simulation modes for the Mailbox Avalon ST Client Intel FPGA IP.

**Related Information**

Mailbox Avalon ST Client Release Notes
1.1.1. Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable status interface</td>
<td>On</td>
<td>When you enable this interface, the Mailbox Avalon ST Intel FPGA Client IP includes the command_status_invalid signal. When command_status_invalid asserts, you must reset the IP.</td>
</tr>
<tr>
<td></td>
<td>Off</td>
<td></td>
</tr>
</tbody>
</table>

1.1.2. Interfaces

The following figure illustrates the Mailbox Avalon ST Client IP interfaces:

**Figure 3. Mailbox AvalonST Client FPGA IP Interfaces**

![Mailbox Avalon ST Client FPGA IP Interfaces](image)

For more information about Avalon ST interfaces, refer to the *Avalon Interface Specifications*.

**Related Information**

Avalon Interface Specifications

1.1.2.1. Clock and Reset Interfaces

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_clk</td>
<td>Input</td>
<td>This is the clock for the Avalon ST interfaces. The maximum frequency in 250 MHz.</td>
</tr>
<tr>
<td>in_reset</td>
<td>Input</td>
<td>This is an active high reset. Assert in_reset to reset the Mailbox Avalon ST Client Intel FPGA IP. When the in_reset signal asserts, the SDM must flush any pending activity from the Mailbox Avalon ST Client Intel FPGA IP. The SDM continues to process commands from other clients.</td>
</tr>
</tbody>
</table>
1.1.2.2. Command Interface

Use the Avalon Streaming (Avalon ST) interface to send commands to the SDM.

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>command_ready</td>
<td>Output</td>
<td>The Mailbox Avalon ST Client Intel FPGA IP asserts command_ready when it is ready to receive commands from the application. The ready_latency is 0 cycles. The Mailbox Avalon ST Client can accept command_data[31:0] in the same cycle that command_ready asserts.</td>
</tr>
<tr>
<td>command_valid</td>
<td>Input</td>
<td>The command_valid signal asserts to indicate that command_data is valid.</td>
</tr>
<tr>
<td>command_data[31:0]</td>
<td>Input</td>
<td>The command_data bus drives commands to the SDM. Refer to Table 3 on page 8 for definitions of the commands.</td>
</tr>
<tr>
<td>command_startofpacket</td>
<td>Input</td>
<td>The command_startofpacket asserts in the first cycle of a command packet.</td>
</tr>
<tr>
<td>command_endofpacket</td>
<td>Input</td>
<td>The command_endofpacket asserts in the last cycle of a command packet.</td>
</tr>
</tbody>
</table>

Figure 4. Timing for Avalon ST Command Packet

1.1.2.3. Response Interface

The SDM Avalon ST Client IP sends responses to your application using the response interface.

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>response_ready</td>
<td>Input</td>
<td>Application logic can assert the response_ready signal whenever it is able to receive a response.</td>
</tr>
<tr>
<td>response_valid</td>
<td>Output</td>
<td>The SDM asserts response_valid to indicate that response_data is valid.</td>
</tr>
<tr>
<td>response_data[31:0]</td>
<td>Output</td>
<td>The SDM drives response_data to provide the requested information. The first word of the response is a header that identifies the command that the SDM is providing. Refer to Table 3 on page 8 for definitions of the commands.</td>
</tr>
<tr>
<td>response_startofpacket</td>
<td>Output</td>
<td>The response_startofpacket asserts in the first cycle of a response packet.</td>
</tr>
<tr>
<td>response_endofpacket</td>
<td>Output</td>
<td>The response_endofpacket asserts in the last cycle of a response packet.</td>
</tr>
</tbody>
</table>
1.1.2.4. Command Status Interface

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>command_status_invalid</td>
<td>Output</td>
<td>The command_status_invalid asserts to indicate an error. This signal typically asserts to indicate that the length of the command specified in the command header does not match the length of the command sent. When command_status_invalid asserts, your application logic must assert in_reset to restart the Mailbox Avalon ST Client Intel FPGA IP.</td>
</tr>
</tbody>
</table>

Figure 6. Reset After command_status_invalid Asserts

1.2. Command and Response Header

The first word of the command and response packets is a header that provides basic information about the command or response.

Figure 7. Command and Response Header Format

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RESERVED</td>
</tr>
<tr>
<td>30</td>
<td>ID</td>
</tr>
<tr>
<td>29</td>
<td>0</td>
</tr>
<tr>
<td>28</td>
<td>LENGTH</td>
</tr>
<tr>
<td>27</td>
<td>0</td>
</tr>
<tr>
<td>26</td>
<td>COMMAND / ERROR CODE</td>
</tr>
</tbody>
</table>

Note:

The following table describes the fields of the header.

Table 2. Command and Response Header Format

<table>
<thead>
<tr>
<th>Header</th>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>[27:24]</td>
<td>The command ID. Application logic creates this ID which is present in both the command and response header. Use this ID to match the command to the command response.</td>
</tr>
<tr>
<td>0</td>
<td>[23]</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>
### 1.3. Supported Commands

#### Table 3. Command Code List and Description

<table>
<thead>
<tr>
<th>Command</th>
<th>Code (Hex)</th>
<th>Command Data Length</th>
<th>Response Length</th>
<th>Description and Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOOP</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Sends an OK status response.</td>
</tr>
<tr>
<td>GET_IDCODE</td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>The response contains one argument which is the JTAG IDCODE for the device.</td>
</tr>
<tr>
<td>GET_CHIPID</td>
<td>12</td>
<td>0</td>
<td>2</td>
<td>The response contains 64-bit CHIPID value with the least significant word first.</td>
</tr>
<tr>
<td>GET_USERCODE</td>
<td>13</td>
<td>0</td>
<td>1</td>
<td>The response contains one argument which is the 32-bit JTAG USERCODE that the configuration bitstream writes to the device.</td>
</tr>
<tr>
<td>GET_VOLTAGE</td>
<td>18</td>
<td>1</td>
<td>1</td>
<td>The GET_VOLTAGE command has a single argument which is a bitmask specifying the channels to read. Bit 0 specifies channel 0, bit 1 specifies channel 1, and so on. The response includes a one-word argument for each bit set in the bitmask. The voltage returned is an unsigned fixed-point number with 16 bits below the binary point. For example, a voltage of 0.75V returns 0x0000C000. (1)</td>
</tr>
<tr>
<td>GET_TEMPERATURE</td>
<td>19</td>
<td>1</td>
<td>1</td>
<td>The GET_TEMPERATURE command has a single argument which is a channel bitmask indicating which temperature sensors to read. This argument is optional. If omitted, the command only reads channel 0. The response contains one word for each channel temperature requested. The temperature returned is a signed fixed value with 8 bits below the binary point. For example, a temperature of 10°C returns 0x00000A00. A of temperature -1.5°C returns 0xFFFFFE80. (1)</td>
</tr>
</tbody>
</table>

(1) Refer to Intel Agilex Power Management User Guide for more information about temperature sensor channels and locations.
For Intel Agilex devices, the channels return the temperatures for the following locations:
- Channel 0: Samples the temperature from the core fabric.
- Channel 1: Samples the temperature from the left transceiver tile.
- Channel 4: Samples the temperature from the right transceiver tile.

If the channel bitmask specifies an invalid channel number, the command returns an error code which is any value in the range 0x8000000 - 0x80000FF.

Information about the local temperature sensor channels is preliminary for Intel Agilex devices.

<table>
<thead>
<tr>
<th>Command</th>
<th>Code (Hex)</th>
<th>Command Data Length</th>
<th>Response Length</th>
<th>Description and Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSU_IMAGE_UPDATE</td>
<td>5C</td>
<td>2</td>
<td>0</td>
<td>Triggers reconfiguration from the data source which can be either the factory or an application image. This command takes an optional 64-bit argument that specifies the reconfiguration data address in the flash.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Bit [63:32]: Reserved (write as 0).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Bit [31:0]: The start address of an application image.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Returns a non-zero response if the device is already processing a configuration command.</td>
</tr>
<tr>
<td>CONFIG_STATUS</td>
<td>4</td>
<td>0</td>
<td>6</td>
<td>Reports the status of the last reconfiguration. You can use this command to check the configuration status during and after configuration. The response contains the following information:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Word</th>
<th>Summary</th>
<th>Description</th>
</tr>
</thead>
</table>
| 0    | State                      | Describes the most recent configuration related error. Returns 0 when there are no configuration errors.  
The error field has 2 fields:  
• Upper 16 bits: Major error code.  
• Lower 16 bits: Minor error code.  
Refer to , Error Codes for the CONFIG_STATUS and RSU_STATUS for more information. |
| 1    | Version                    | The version of the RSU data structure.                                      |
| 2    | Pin status                 | • Bit [31]: Current nSTATUS output value (active low)  
• Bit [30]: Detected nCONFIG input value (active low)  
• Bit [29:3]: Reserved  
• Bit [2:0]: The MSEL value at power up |
| 3    | Soft function status       | Contains the value of each of the soft functions, even if you have not assigned the function to an SDM pin.  
• Bit [31:6]: Reserved  
• Bit [5]: HPS_WARMRESET  
• Bit [4]: HPS_COLDRESET  
• Bit [3]: SEU_ERROR  
• Bit [2]: CVP_DONE  
• Bit [1]: INIT_DONE  
• Bit [0]: CONF_DONE  

continued...
<table>
<thead>
<tr>
<th>Command</th>
<th>Code (Hex)</th>
<th>Command Data Length</th>
<th>Response Length</th>
<th>Description and Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>Error location</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Contains the error location. Returns 0 if there are no errors.</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td>Error details</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Contains the error details. Returns 0 if there are no errors.</td>
</tr>
<tr>
<td>RSU_STATUS</td>
<td>5B</td>
<td>0</td>
<td>9</td>
<td>Reports the current remote system upgrade status. You can use this command to check the configuration status during configuration and after it has completed. This command returns the following responses:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0-1 Current image</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Flash offset of the currently running application image.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2-3 Failing image</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Flash offset of the highest priority failing application image. If multiple images are available in flash memory, stores the value of the first image that failed. A value of all 1s indicates no failing images. If there are no failing images, the remainder of the remaining words of the status information do not store valid information.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4 State</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Failure code of the failing image. The error field has two parts:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Upper 16 bits: Major error code.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Lower 16 bits: Minor error code.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Returns 0 for no failures. Refer to , Error Codes for the CONFIG_STATUS and RSU_STATUS for more information.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5 Version</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The version of the RSU software.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6 Error location</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Stores the error location of the failing image. Returns 0 for no errors.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7 Error details</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Stores the error details for the failing image. Returns 0 if there are no errors.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8 Current image retry counter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Count of the number of retries that have been attempted for the current image. The counter is 0 initially. The counter is set to 1 after the first retry, then 2 after a second retry. Specify the maximum number of retries in your Intel Quartus® Prime Settings File (.qsf). The command is: set_global_assignment -name RSU_MAX_RETRY_COUNT 3. Valid values for the MAX_RETRY counter are 1-3. The actual number of available retries is MAX_RETRY - 1.</td>
</tr>
<tr>
<td>QSPI_OPEN</td>
<td>32</td>
<td>0</td>
<td>1</td>
<td>Requests exclusive access to the quad SPI. The SDM accepts the request if the quad SPI is not in use and the SDM is not configuring the device. Returns OK if the SDM grants access. Returns the ALT_SDM_MBOX_RESP_DEVICE_BUSY when the quad SPI flash is busy.</td>
</tr>
<tr>
<td>Command</td>
<td>Code (Hex)</td>
<td>Command Data Length</td>
<td>Response Length</td>
<td>Description and Response</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------</td>
<td>---------------------</td>
<td>-----------------</td>
<td>------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>QSPI_CLOSE</td>
<td>33</td>
<td>0</td>
<td>1</td>
<td>Closes the exclusive access to the quad SPI interface.</td>
</tr>
</tbody>
</table>
| QSPI_SET_CS     | 34         | 1                   | 1               | Specifies one of the attached quad SPI devices via the chip select lines. Takes a one-word argument as described below:  
  - Bits[31:28]: Flash device to select. The value 4'b0000 selects the flash that corresponds to nCSO[0]. nCSO[0] is the only signal that the FPGA can use to access the quad SPI flash device. The HPS can use nCSO[3:1] to access HPS data.  
  - Bits[27:0]: Reserved (write as 0). The HPS can use nCSO[3:1] to access 3 additional quad SPI devices.  
  This command is optional for the AS x4 configuration scheme. Is required for all other configuration schemes. |
| QSPI_READ       | 3A         | 2                   | N               | Reads the attached quad SPI device. The maximum read size is 4 kilobytes (KB). Takes two arguments:  
  - The quad SPI flash address (one word). The address must be word aligned. The device returns the 0x1 error code for non-aligned addresses.  
  - Number of words to read (one word).  
  When successful returns OK followed by the read data from the quad SPI device. A failure response returns an error code.  
  For a partially successful read, QSPI_READ may erroneously return the OK status.  
  Note: You cannot run the QSPI_READ command while device configuration is in progress. |
| QSPI_WRITE      | 39         | 2+N                 | 0               | Writes data to the quad SPI device. Takes three arguments:  
  - The flash address offset (one word). The write address must be word aligned. The device returns error code 0x3FF for non-aligned addresses.  
  - The number of words to write (one word).  
  - The data to be written (one or more words).  
  A successful write returns the OK response code.  
  To prepare memory for writes, Intel recommends using the QSPI_ERASE command before issuing this command.  
  Note: You cannot run the QSPI_WRITE command while device configuration is in progress. |
| QSPI_ERASE      | 38         | 2                   | 0               | Erases a sector of the quad SPI device. Takes two arguments:  
  - The flash address offset to start the erase (one word). The address must be the start address of a sector within the flash memory; consequently, the address must be 64 KB aligned. Returns an error for non-64 KB aligned addresses.  
  - The number of words to erase specified in multiples of 0x4000 words.  
  A successful erase returns the OK response code. |
| QSPI_READ_DEV ICE_REG | 35 | 2 | N | Reads registers from the quad SPI device. The maximum read is 8 bytes. Takes two arguments:  
  - The opcode for the read command.  
  - The number of bytes to read. |

Note: The SDM grants exclusive access to the client using this mailbox. Other clients cannot access the quad SPI until the active client relinquishes access using the QSPI_CLOSE command.
A successful read returns the OK response code followed by the data read from the device. Pads data that is not a multiple of 4 bytes to the next word boundary.

QSPI_WRITE_DEVICE_REG

<table>
<thead>
<tr>
<th>Command Code (Hex)</th>
<th>Command Data Length</th>
<th>Response Length</th>
<th>Description and Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>2+N</td>
<td>0</td>
<td>Writes to registers of the quad SPI. The maximum write is 8 bytes. Takes three arguments: • The opcode for the write command. • The number of bytes to write. • The data to write. To perform a sector erase or sub-sector erase, you must specify the serial flash address in most significant byte (MSB) to least significant byte (LSB) order as the following example illustrates. To erase a sector of a Micron 2 gigabit (Gb) flash at address 0x04FF0000 using the QSPI_WRITE_DEVICE_REG command, write the flash address in MSB to LSB order as shown here: Header: 0x00003036 Opcode: 0x000000DC Number of bytes to write: 0x00000004 Flash address: 0x0000FF04 A successful write returns the OK response code. This command pads data that is not a multiple of 4 bytes to the next word boundary.</td>
</tr>
</tbody>
</table>

QSPI_SEND_DEVICE_OP

<table>
<thead>
<tr>
<th>Command Code (Hex)</th>
<th>Command Data Length</th>
<th>Response Length</th>
<th>Description and Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>1</td>
<td>0</td>
<td>Sends a command opcode to the quad SPI. Takes one argument: • The opcode to send the quad SPI device. A successful command returns the OK response code.</td>
</tr>
</tbody>
</table>

Related Information
Intel Agilex Power Management User Guide

1.4. Error Codes

The response packet header includes the error code when the command fails.

<table>
<thead>
<tr>
<th>Value (Hex)</th>
<th>Error Code Response</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OK</td>
<td>Indicates that the command completed successfully. A command may erroneously return the OK status if a command, is partially successful.</td>
</tr>
<tr>
<td>1</td>
<td>INVALID_COMMAND</td>
<td>Indicates that the command is incorrectly formatted.</td>
</tr>
<tr>
<td>2</td>
<td>UNKNOWN_BR</td>
<td>Indicates that the command code is not understood.</td>
</tr>
<tr>
<td>3</td>
<td>UNKNOWN</td>
<td>Indicates that the currently loaded firmware cannot decode the command code.</td>
</tr>
<tr>
<td>4</td>
<td>INVALID_COMMAND_PARAMETERS</td>
<td>The length or indirect setting in header is not valid. Or the command data is invalid.</td>
</tr>
<tr>
<td>5</td>
<td>COMMAND_INVALID_ON_SOURCE</td>
<td>Command is from a source for which it is not enabled.</td>
</tr>
<tr>
<td>6</td>
<td>CLIENT_ID_NO_MATCH</td>
<td>Indicates that the Client ID requesting quad SPI or SD MMC access does not have exclusive access.</td>
</tr>
<tr>
<td>7</td>
<td>INVALID_ADDRESS</td>
<td>The address is invalid. This error indicates one of the following conditions:</td>
</tr>
</tbody>
</table>

continued...
<table>
<thead>
<tr>
<th>Value (Hex)</th>
<th>Error Code Response</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>TIMEOUT</td>
<td>The command timed out.</td>
</tr>
<tr>
<td>9</td>
<td>HW_NOT_READY</td>
<td>The hardware is not ready. Can indicate either an initialization or configuration problem.</td>
</tr>
<tr>
<td>100</td>
<td>NOT_CONFIGURED</td>
<td>Indicates that the device is not configured.</td>
</tr>
<tr>
<td>1FF</td>
<td>ALT_SDM_MBOX_RESP_DEVICE_BUSY</td>
<td>Indicates that the device is busy.</td>
</tr>
<tr>
<td>2FF</td>
<td>ALT_SDM_MBOX_RESP_NO_VALID_RESP_AVAILABLE</td>
<td>Indicates that there is no valid response available.</td>
</tr>
<tr>
<td>3FF</td>
<td>ALT_SDM_MBOX_RESP_ERROR</td>
<td>General Error.</td>
</tr>
</tbody>
</table>

1.5. Document Revision History for the Mailbox Avalon ST Client Intel FPGA IP User Guide

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Intel Quartus Prime Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019.09.30</td>
<td>19.3</td>
<td>Initial release.</td>
</tr>
</tbody>
</table>