Altera Avalon Mailbox (simple) Core 41

2016.12.19

UG-01085 Subscribe Send Feedback

Core Overview

In a multiprocessor design, each processor may be dedicated to perform a specific task. Communication between processors becomes crucial if the tasks of each individual processor are interdependent. Communication between processors may involve data passing or task control coordination to accomplish certain functions.

The Altera Avalon Mailbox (simple) component provides the medium of communication between processors. It provides a "message" passing location between the sending processor and receiving processor. The receiving processor is notified upon a message arrival. The notification can be in the form of an interrupt issuing to the receiving processor or it can continue pooling for new messages by the receiving processor.

If more than two processors require "message" passing, multiple Mailboxes can be instantiated between the two processors. Each Altera Avalon Mailbox corresponds to one direction message passing.

Functional Description

Altera Avalon Mailbox (simple) provides two 32-bit registers for message passing between processors, Command register (0x0) and Pointer register (0x1). The message sender processor and message receiver processor have individual Avalon Memory Mapped (Avalon-MM) interfaces to a Mailbox component. A write to the command register by the sender processor indicates a pending message in the Mailbox and an interrupt will be issued to the receiver processor. Upon retrieval of the message by the receiver processor via a read transaction, the message is consumed, Mailbox is empty. The status register (0x2) is used to indicate if the Mailbox is full or empty.

The Mailbox Avalon-MM interface which receives messages, or identified as sender interface, will back pressure the sender if there is message pending in the Mailbox. This will ensure every single message passed into the Mailbox is not overwritten. Upon message arrival, the receiving processor will then receive a level interrupt by the Mailbox. The interrupt will hold high until the single message is retrieved from the Mailbox via the Avalon-MM interface of receiving processor.

In addition, the Interrupt Masking Register (0x3) is writable by the Avalon-MM interface to mask its dedicated interrupt output. For example, receiver interface will be able to set the mask bit to mask off the message pending interrupt generated by Mailbox. Meanwhile, sender interface will be able to set the mask bit to mask off the message space interrupt output.

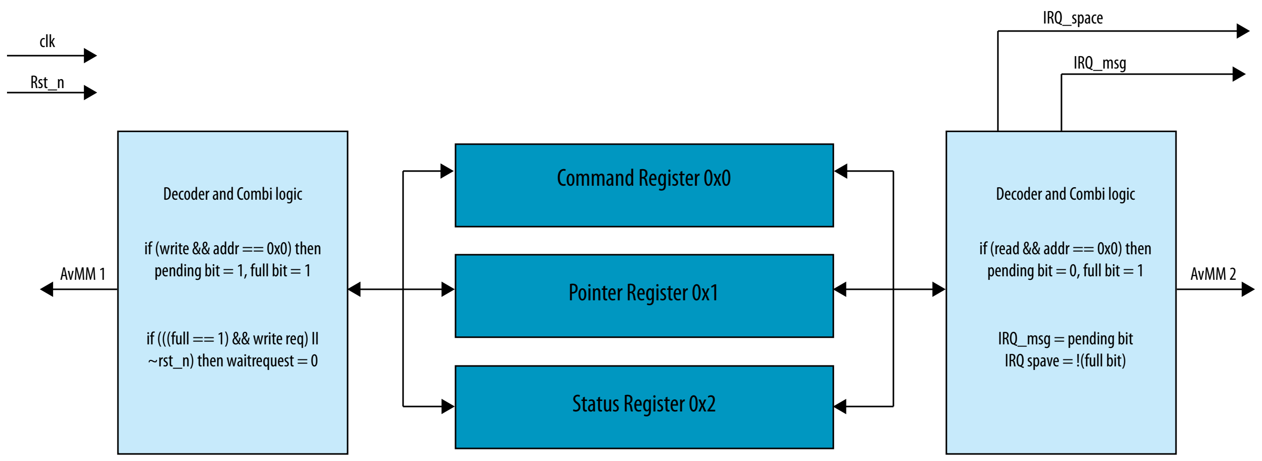
© 2016 Intel Corporation. All rights reserved. Intel, the Intel logo, Altera, Arria, Cyclone, Enpirion, MAX, NIOS, Quartus and Stratix words and logos are trademarks of Intel Corporation in the US and/or other countries. Other marks and brands may be claimed as the property of others. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

ALTERA now part of Intel

101 Innovation Drive, San Jose, CA 95134

Figure 41-1: Altera Avalon Mailbox (simple) Block Diagram



The Mailbox is clocked with single source. Both of the Avalon-MM Slave interfaces have its individual function to set and clear the Full bit and Message Pending bit. The Avalon-MM Slave of the sender processor will only set the status bits, while the Avalon-MM Slave of the receiver processor only clears the status bit.

An interrupt is derived from the Status register bits. It will remain high until the message in the Mailbox is read.

Message Sending and Retrieval Process

The following are steps needed to send and receive messages through the Altera Avalon Mailbox (simple) component:

1. A process or master that intends to send a message will write to the Mailbox's Pointer register at address offset 0x1, then only to the Command register at address offset 0x0. Writing to the Command register indicates the completion of a message passing into the Mailbox.
2. When there is a message pending in the Mailbox, a level interrupt signal is issued to the processor that should receive the message. Optionally, the receiver processor may choose to poll the Status register at address offset 0x2 to determine if any message has arrived, if the interrupt signal is not used.
3. The process or master that needs to receive the message reads the Mailbox's Pointer register and then the Command register through the connected Avalon-MM interface. Upon reading of Command register, the message is considered delivered, and the Mailbox is empty.

Registers of Component

The following table illustrates the Mailbox registers map that is observed by each processor from its Avalon-MM interfaces.

Table 41-1: Mailbox Register Map

Word Address Offset	Register/ Queue Name	Attribute
`0x0`	Command register	R/W for sender, RO for receiver
`0x1`	Pointer register	R/W for sender, RO for receiver
`0x2`	Status register	RO

Word Address Offset	Register/ Queue Name	Attribute
`0x3`	Interrupt Masking register	Read (R) for both sender and receiver. Sender can only write to Message Space Interrupt Mask bit, Receiver can only write to Message Pending Interrupt bit.

Command Register

The Command register is a 32-bit register which contains a user defined command to be passed between processors. This register is read-writeable via Avalon-MM Slave (sender). However it is only readable by the Avalon-MM Slave (receiver) interface.

Pointer Register

Instead of passing huge data via the Mailbox, a Pointer register is introduced. The Pointer register contains the 32-bit address to the payload of the message. A payload could be the raw data to be passed to the receiving processor for further processing. However, a message could contain zero payload or data for processing. A write to the Pointer may not be necessary for a message passing.

This register is read-writeable via Avalon-MM Slave (sender). However it is only readable by Avalon-MM Slave (receiver) interface.

Status Register

The Status register presents the full or empty status of the Mailbox. As the Mailbox can only contain one message at a time, the full bit status also indicates if there is message pending in the Mailbox. This register is read only by both Avalon-MM Slave interfaces.

Table 41-2: status Register Field

Bit Fields													
31	...		...		...		...		...		2	1	0
Reserved												Mailbox full	Message pending

Table 41-3: Mailbox status Register Descriptions

Filed Name	Description	Reset Value
Message pending	Value ‘0’ indicates the Mailbox has no message. Value ‘1’ indicates the Mailbox has message pending for retrieval.	0
Mailbox full	Value ‘1’ indicates the Mailbox is full. Value ‘0’ indicates Mailbox has space for incoming message.	0
Reserved	-	0

Interrupt Masking Register

The Interrupt Masking Register provides a masking bit to the Message Pending Interrupt and Message Space Interrupt. This register is accessible by both the sender and receiver of the Avalon-MM Slave interface. However, the editable bit is only applicable for its corresponded interrupt. This means the sender Avalon-MM Slave can only modify the masking bit of Message Space Interrupt, whereas receiver Avalon-MM Slave can only modify the masking bit of Message Pending Interrupt. Read access of the whole register is available to both Avalon-MM Slave Interfaces.

Table 41-4: Interrupt Masking Register Field

Bit Fields													
31	...		...		...		...		...		2	1	0
Reserved												Message space mask	Message pending mask

Table 41-5: Interrupt Masking Register Descriptions

Filed Name	Description	Reset Value
Message pending mask	Value '0' to mask off the Message Pending Interrupt output. Value '1' enable Message Pending Interrupt upon triggered.	0
Mailbox space mask	Value '0' to mask off the Message Space Interrupt output. Value '1' enable Message Space Interrupt upon triggered.	0
Reserved	-	0

Interface

Component Interface

Altera Avalon Mailbox (simple) component consists of two Avalon-MM Slave interfaces, one dedicated for each processor. The Mailbox also provides active high level interrupt output, which is served as message arrival notification to the receiving processor. Optionally, a secondary IRQ is created as notification to the message sender indicating if Mailbox is available for incoming message.

Altera Avalon Mailbox (simple) has only one clock domain with one associated reset interface. Requirement of different clock domains between two processors is handled through the Qsys fabric. The following table describes the interfaces behavior of the component.

Table 41-6: Component Interface Behavior

Interface Port	Description	Details
Avalon MM Slave (sender)	Avalon-MM Slave interface for processor of message sender.	This interface apply wait request signal for back pressuring the Avalon-MM Master if Mailbox is already full.
Avalon MM Slave (receiver)	Avalon-MM Slave interface for processor of message receiver.	This interface only has read capability with readWaitTime=1.
Clock	Clock input of component.	It supports maximum frequency up to 400MHz on CycloneIV and 600MHz in StratixIV devices.
Reset_n	Active LOW reset input/s.	Support asynchronous reset assertion. De-assertion of reset has to be synchronized to the input clock.
IRQ_msg	Message Pending Interrupt output to processor of message receiver upon message arrival. The signal will remain high until the message is retrieved.	Interrupt assertion and deassertion is synchronized to input clock.
IRQ_space	Message Space Interrupt output processor of message sender whenever Mailbox has space for incoming message. The signal will assert high as long as the Mailbox is yet full.	Interrupt assertion and deassertion is synchronized to input clock. The connection of this interrupt port to the top level is depends on configuration parameter of `MSG_SPACE_NOTIFY`.

Component Parameterization

Table 41-7: Altera Avalon Mailbox (simple) TCL Component Configuration Parameters

Parameter Name	Description	Default Value	Allowable Range
`MSG_SPACE_NOTIFY`	Boolean 'true' will enable interrupt output to message sending processor for indicating available space for incoming message	0	0, 1
`MSG_ARRIVAL_NOTIFY`	Boolean 'true' will enable interrupt output to message receiver processor for indicating a message is pending for retrieval.	1	0, 1

HAL Driver

This section describes the HAL driver for Altera Avalon Mailbox (simple) soft IP core. Altera Avalon Mailbox (simple) component provides a medium of communication between processors. It provides a message passing path between the sending processor and receiving processor. The receiver processor is notified through an interrupt upon message arrival or the driver will poll the status register if in polling mode. Altera Avalon Mailbox (simple) provides three 32-bit registers for message passing between processors, Command (0x0), Pointer (0x4), and Status register (0x8).

The driver code is located at:

```
/acds/main/ip/altera_avalon_mailbox/hal/src/altera_avalon_mailbox_simple.c

/acds/main/ip/altera_avalon_mailbox/hal/inc/altera_avalon_mailbox_simple.h

/acds/main/ip/altera_avalon_mailbox/inc/altera_avalon_mailbox_simple_regs.h

/acds/main/ip/altera_avalon_mailbox/altera_avalon_mailbox_simple_sw.tcl
```

Feature Description

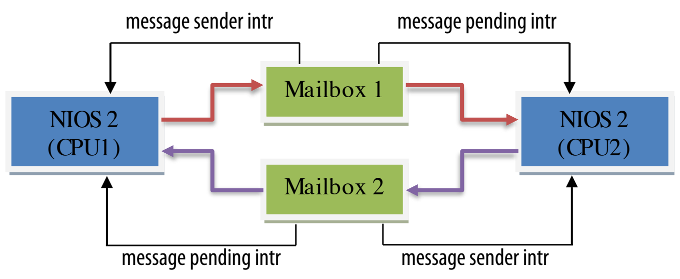
The Mailbox driver message delivery depends on how the QSYS design of the sender processor, receiver processor and Mailbox are interconnected. The Mailbox driver provides the features to send message to target processor and retrieve message for the receiver processor. The driver include an interrupt service routine when interrupt mode is used.

Configuration

Interrupt Mode

The figure below is an example of a design using the Altera Avalon Mailbox (simple) in interrupt mode. The sender CPU(1) will initiate a transfer of the message to the receiver CPU(2) by writing the command data to the Command register through Mailbox 1. The Command register will send a message pending interrupt to the receiver. The message pending interrupt is connected to the receiver CPU(2)'s IRQ to notify that a message has arrived. Once the Command register in Mailbox 1 is read, the message pending interrupt is cleared and the message is processed. On the sender CPU(1) side, once the message is read, a message sender interrupt will be flagged signaling that Maibox 1 is free to transmit another message.

Figure 41-2: Example of a Bi-Directional Altera Avalon Mailbox System Using Interrupt Mode



Polling Mode

In the case of polling mode, you will always check on the Mailbox Status register if a message has arrived or free to send. Driver API functions include a timeout parameter, which allows you to specify whether a read or send operation must be completed within a certain period of time.

Driver Implementation

An opened Mailbox instance will register a sender/receiver interrupt service routine (ISR), if interrupts are supported with sender/receiver callbacks. When a Mailbox interrupt is disabled, an ISR will not register and polling mode will need to be used. You must close the Mailbox driver when it is unused.

Table 41-8: Mailbox APIs

Function Name	Description
altera_avalon_mailbox_send	Send message to Mailbox
altera_avalon_mailbox_status	Query current state of Mailbox
altera_avalon_mailbox_retrieve_poll	Read from Mailbox pointer register to retrieve messages
altera_avalon_mailbox_open	Claims a handle to a Mailbox, enabling all the other functions to access the Mailbox core
altera_avalon_mailbox_close	Close the handle to a Mailbox

Table 41-9: altera_avalon_mailbox_open

Prototype:	altera_avalon_mailbox_dev* altera_avalon_mailbox_open (const char* name, altera_mailbox_tx_cb tx_callback, altera_mailbox_rx_cb rx_callback)
Include:	<altera_avalon_mailbox_simple.h>
Parameters:	Name — The Mailbox device name to open. tx_callback – User to provide callback function to notify when a sending message is completed. rx_callback – User to provide callback function to notify when a receive a message.
Returns:	Pointer to mailbox
Description:	altera_avalon_mailbox_open() find and register the Mailbox device pointer. This function also registers the interrupt handler and user callback function for a interrupt enabled Mailbox.

Table 41-10: altera_avalon_mailbox_close

Prototype:	void altera_avalon_mailbox_close (altera_avalon_mailbox_dev* dev);
Include:	<altera_avalon_mailbox_simple.h>
Parameters:	dev—The Mailbox to close.
Returns:	Null
Description:	alt_avalon_mailbox_close() closes the mailbox de-registering interrupt handler and callback functions and masking Mailbox interrupt.

Table 41-11: altera_avalon_mailbox_send

Prototype:	int altera_avalon_mailbox_send (altera_avalon_mailbox_dev* dev, void* message, int timeout, EventType event)
Include:	<altera_avalon_mailbox_simple.h>
Parameters:	*message – Pointer to message command and pointer structure. Timeout – Specifies number of loops before sending a message. Give a '0' value to wait until the message is transferred. EventType – set 'POLL' or 'ISR'.
Returns:	Return 0 on success and 1 for fail.
Description:	altera_avalon_mailbox_send () sends a message to the mailbox. This is a blocking function when the sender interrupt is disabled. This function is in non-blocking when interrupt is enabled.

Table 41-12: altera_avalon_mailbox_retrieve_poll

Prototype:	int altera_avalon_mailbox_retrieve_poll (altera_avalon_mailbox_dev* dev,alt_u32* msg_ptr, alt_u32 timeout)
Include:	<altera_avalon_mailbox_simple.h>
Parameters:	dev - The Mailbox device to read message from. timeout – Specifies number loops before sending a message. Give a '0' value to wait until a message is retrieved. msg_ptr – A pointer to an array of two Dwords which are for the command and message pointer. This pointer will be populated with a receive message if successful or NULL if error.
Returns:	Return pointer to message and command. Return 'NULL' in messages if timeout. This is a blocking function.
Description:	altera_avalon_mailbox_retrieve_poll () reads a message pointer and command to Mailbox structure from the Mailbox and notifies through callback.

Table 41-13: altera_avalon_mailbox_status

Prototype:	alt_u32 altera_avalon_mailbox_status (altera_avalon_mailbox_dev* dev)
Include:	<altera_avalon_mailbox_simple.h>
Parameters:	dev -The Mailbox device to read status from

Returns:	For a receiving Mailbox: - 0 for no message pending - 1 for message pending For a sending Mailbox: - 0 for Mailbox empty (ready to send) - 1 for Mailbox full (not ready to send)
Description:	Indicates to sender Mailbox it is full or empty for transfer. Indicates to receiver Mailbox has a message pending or not.

Example 41-1: Device structure

```
// Callback routine type definition
typedef void(*altera_mailbox_rx_cb)(void *message);
typedefvoid (*altera_mailbox_tx_cb)(void *message,int status);

typedef enum mbox_type { MBOX_TX = 0,MBOX_RX } MboxType;
typedef enum event_type { ISR = 0, POLL } EventType;

typedef struct altera_avalon_mailbox_dev
{
    alt_dev                 dev;                  /* Device linke-list entry
*/
    alt_u32                 base;                 /* Base address of Mailbox
*/
    alt_u32                 mailbox_irq;          /* Mailbox IRQ */
    alt_u32                 mailbox_intr_ctrl_id; /* Mailbox IRQ ID */
    altera_mailbox_tx_cb    tx_cb;                /* Callback routine
pointer */
    altera_mailbox_rx_cb    rx_cb;                /* Callback routine
pointer */
    MboxType                mbox_type;            /* Mailbox direction */
    alt_u32*                mbox_msg;             /* a pointer to message
array to be
                                                * received or sent */
    alt_u8                  lock;                 /* Token to indicate
mbox_msg already taken */
    ALT_SEM                 (write_lock)          /* Semaphore used to
control access to the
                                                * write in multi-threaded mode */
} altera_avalon_mailbox_dev;
```

Driver Examples

The figure below demonstrates writing to a Mailbox. For this example, assume that the hardware system has two processors communicating via Mailboxes. The system includes two Mailbox cores, which provides two-way communication between the processors.

Example 41-2: Sender Processor Using Mailbox to Send a Message.

```
#include <stdio.h>
#include "altera_avalon_mailbox_simple.h"
#include "altera_avalon_mailbox_simple_regs.h"
#include "system.h"

/* example callback function from users*/
void tx_cb (void* report, int status) {
    if (!status) {
       printf (“Transfer done”);
    } else {
       printf (“error in transfer”);
    }

int main_sender()
{
alt_u32 message[2] = {0x00001111, 0xaa55aa55};
int timeout     = 50000;
alt_u32 status;
alt_avalon_mailbox_simple_dev* mailbox_sender;

/* Open mailbox on sender processor */
mailbox_sender = alt_avalon_mailbox_open("/dev/mailbox_simple_0", tx_cb,
NULL);

    if (!mailbox_sender){
        printf ("FAIL: Unable to open mailbox_simple");
        return 1;
        }

    /* Send a message to the other processor using interrupt */
    altera_avalon_mailbox_send (mailbox_sender, message, 0, ISR);

    /* Using polling method to send a message, with infinite timeout */
        timeout = 0;
        status = altera_avalon_mailbox_send (mailbox_sender, message,
timeout, POLL);

        if (status) {
            printf (“error in transfer”);
        } else {
            printf (“Transfer done”);
        }

    /* Closing mailbox device and de-registering interrupt handler and
callback */
    altera_avalon_mailbox_close (mailbox_sender);
    return 0;
    }
```

Example 41-3: Receiver Processor Waiting for Message.

```
#include <stdio.h>
#include "altera_avalon_mailbox_simple.h"
#include "altera_avalon_mailbox_simple_regs.h"
#include "system.h"

void rx_cb (void* message) {
    /* Get message read from mailbox */
    alt_u32* data = alt_u32* message;
    if (message!= NULL) {
        printf (“Message received”);
    } else {
      printf (“Incomplete receive”);
    }

int main_receiver()
{
alt_u32* message[2];
int timeout     = 50000;
alt_avalon_mailbox_simple_dev* mailbox_rcv;

    /* This example is running on receiver processor */
    mailbox_rcv = alt_avalon_mailbox_open("/dev/mailbox_simple_1", NULL,
rx_cb);
    if (!mailbox_rcv){
        printf ("FAIL: Unable to open mailbox_simple");
        return 1;
    }

    /* For interrupt disable system */
    altera_avalon_mailbox_retrieve_poll (mailbox_rcv,message, timeout)
    if (message == NULL) {
        printf (“Receive Error”);
    } else {
      printf (“Message received with Command 0x%x and Message 0x%x\n”,
message[0], message[1]);
    }

altera_avalon_mailbox_close (mailbox_rcv);
return 0;
}
```

Document Revision History

Table 41-14: Altera Avalon Mailbox (simple) Core Revision History

Date	Version	Changes
November 2015	2015.11.06	Added HAL Driver section.
June 2015	2015.06.12	Initial release.