Xilinx Standalone Library Documentation

XilMailbox Library v1.2

UG1367 (v2020.2) December 16, 2020

XILINX®

Table of Contents

XilMailbox Library API Reference

The XilMailbox library provides the top-level hooks for sending or receiving an inter-processor interrupt (IPI) message using the Zynq® UltraScale+™ MPSoC and Versal ACAP IPI hardware.This library supports Zynq Ultrascale+ MPSoC and Versal platforms. For more details on the IPI interrupts, see the Zynq UltraScale+ MPSoC Technical Reference Manual (UG1085).

The XilMailbox library supports the following features:

- Triggering an IPI to a remote agent.
- Sending an IPI message to a remote agent.
- Callbacks for error and recv IPI events.
- Reading an IPI message.

The following is a list of software initalization events for a given IPI channel:

- IPI Initialization using XMailbox_Initalize() function. This step initializes a library instance for the given IPI channel.
- XMailbox_Send() function triggers an IPI to a remote agent.
- XMailbox_SendData() function sends an IPI message to a remote agent. Message type should be either XILMBOX_MSG_TYPE_REQ (OR) XILMBOX_MSG_TYPE_RESP.
- XMailbox_Recv() function reads an IPI message from a specified source agent. Message type should be either XILMBOX_MSG_TYPE_REQ (OR) XILMBOX_MSG_TYPE_RESP.
- XMailbox_SetCallBack() using this function user can register call backs for recv and error events.

Table 1: **Quick Function Reference**

Type	Name	Arguments
u32	XMailbox_Send	`XMailbox` * InstancePtr u32 RemoteId u8 Is_Blocking

Table 1: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
u32	XMailbox_SendData	XMailbox * InstancePtr u32 RemoteId void * BufferPtr u32 MsgLen u8 BufferType u8 Is_Blocking
u32	XMailbox_Recv	XMailbox * InstancePtr u32 SourceId void * BufferPtr u32 MsgLen u8 BufferType
s32	XMailbox_SetCallBack	XMailbox * InstancePtr XMailbox_Handler HandlerType CallBackFunc CallBackRef
u32	XMailbox_Initialize	XMailbox * InstancePtr u8 DeviceId
u32	XIpiPs_Init	XMailbox * InstancePtr u8 DeviceId
u32	XIpiPs_Send	XMailbox * InstancePtr u8 Is_Blocking
u32	XIpiPs_SendData	XMailbox * InstancePtr void * MsgBufferPtr u32 MsgLen u8 BufferType u8 Is_Blocking
u32	XIpiPs_PollforDone	XMailbox * InstancePtr
u32	XIpiPs_RecvData	XMailbox * InstancePtr void * MsgBufferPtr u32 MsgLen u8 BufferType
XStatus	XIpiPs_RegisterIrq	void

Table 1: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
void	XIpiPs_ErrorIntrHandler	void
void	XIpiPs_IntrHandler	void

Functions

XMailbox_Send

This function triggers an IPI to a destination CPU.

Prototype

```
u32 XMailbox_Send(XMailbox *InstancePtr, u32 RemoteId, u8 Is_Blocking);
```

Parameters

The following table lists the `XMailbox_Send` function arguments.

Table 2: **XMailbox_Send Arguments**

Type	Name	Description
`XMailbox` *	InstancePtr	Pointer to the `XMailbox` instance
u32	RemoteId	Mask of the CPU to which IPI is to be triggered
u8	Is_Blocking	If set, triggers the notification in blocking mode

Returns

- XST_SUCCESS if successful
- XST_FAILURE if unsuccessful

XMailbox_SendData

This function sends an IPI message to a destination CPU.

Prototype

```
u32 XMailbox_SendData(XMailbox *InstancePtr, u32 RemoteId, void *BufferPtr,
u32 MsgLen, u8 BufferType, u8 Is_Blocking);
```

Parameters

The following table lists the `XMailbox_SendData` function arguments.

Table 3: **XMailbox_SendData Arguments**

Type	Name	Description
`XMailbox` *	InstancePtr	Pointer to the `XMailbox` instance
u32	RemoteId	Mask of the CPU to which IPI is to be triggered
void *	BufferPtr	Pointer to buffer which contains the message to be sent
u32	MsgLen	Length of the buffer/message
u8	BufferType	Type of buffer (XILMBOX_MSG_TYPE_REQ (OR) XILMBOX_MSG_TYPE_RESP)
u8	Is_Blocking	If set, triggers the notification in blocking mode

Returns

- XST_SUCCESS if successful
- XST_FAILURE if unsuccessful

XMailbox_Recv

This function reads an IPI message.

Prototype

```
u32 XMailbox_Recv(XMailbox *InstancePtr, u32 SourceId, void *BufferPtr, u32
MsgLen, u8 BufferType);
```

Parameters

The following table lists the `XMailbox_Recv` function arguments.

Table 4: **XMailbox_Recv Arguments**

Type	Name	Description
`XMailbox` *	InstancePtr	Pointer to the `XMailbox` instance
u32	SourceId	Mask for the CPU which has sent the message
void *	BufferPtr	Pointer to buffer to which the read message needs to be stored
u32	MsgLen	Length of the buffer/message
u8	BufferType	Type of buffer (XILMBOX_MSG_TYPE_REQ or XILMBOX_MSG_TYPE_RESP)

Returns

- XST_SUCCESS if successful

- XST_FAILURE if unsuccessful

XMailbox_SetCallBack

This routine installs an asynchronous callback function for the given HandlerType.

Prototype

```
s32 XMailbox_SetCallBack(XMailbox *InstancePtr, XMailbox_Handler
HandlerType, void *CallBackFuncPtr, void *CallBackRefPtr);
```

Parameters

The following table lists the `XMailbox_SetCallBack` function arguments.

Table 5: **XMailbox_SetCallBack Arguments**

Type	Name	Description
`XMailbox` *	InstancePtr	Pointer to the `XMailbox` instance.
`XMailbox_Handler`	HandlerType	Specifies which callback is to be attached.
Commented parameter CallBackFunc does not exist in function XMailbox_SetCallBack.	CallBackFunc	Address of the callback function.
Commented parameter CallBackRef does not exist in function XMailbox_SetCallBack.	CallBackRef	User data item that is passed to the callback function when it is invoked.

Returns

- XST_SUCCESS when handler is installed.
- XST_INVALID_PARAM when HandlerType is invalid.

XMailbox_Initialize

This function initializes the `XMailbox` instance.

Prototype

```
u32 XMailbox_Initialize(XMailbox *InstancePtr, u8 DeviceId);
```

Parameters

The following table lists the `XMailbox_Initialize` function arguments.

Table 6: **XMailbox_Initialize Arguments**

Type	Name	Description
XMailbox *	InstancePtr	Pointer to the instance to be worked on
u8	DeviceId	IPI instance to be worked on

Returns

XST_SUCCESS if initialization was successful XST_FAILURE in case of failure

XIpiPs_Init

This function initializes the Zynq UltraScale+ MPSoC Mailbox instance.

Prototype

```
u32 XIpiPs_Init(XMailbox *InstancePtr, u8 DeviceId);
```

Parameters

The following table lists the `XIpiPs_Init` function arguments.

Table 7: **XIpiPs_Init Arguments**

Type	Name	Description
XMailbox *	InstancePtr	Pointer to the instance to be worked on
u8	DeviceId	IPI instance to be worked on

Returns

XST_SUCCESS if initialization was successful XST_FAILURE in case of failure

XIpiPs_Send

This function triggers an IPI to a destination CPU.

Prototype

```
u32 XIpiPs_Send(XMailbox *InstancePtr, u8 Is_Blocking);
```

Parameters

The following table lists the `XIpiPs_Send` function arguments.

Table 8: **XIpiPs_Send Arguments**

Type	Name	Description
XMailbox *	InstancePtr	Pointer to the XMailbox instance.
u8	Is_Blocking	If set, triggers the notification in blocking mode

Returns

XST_SUCCESS in case of success XST_FAILURE in case of failure

XIpiPs_SendData

This function sends an IPI message to a destnation CPU.

Prototype

```
u32 XIpiPs_SendData(XMailbox *InstancePtr, void *MsgBufferPtr, u32 MsgLen,
u8 BufferType, u8 Is_Blocking);
```

Parameters

The following table lists the `XIpiPs_SendData` function arguments.

Table 9: **XIpiPs_SendData Arguments**

Type	Name	Description
XMailbox *	InstancePtr	Pointer to the XMailbox instance
void *	MsgBufferPtr	Pointer to buffer which contains the message to be sent
u32	MsgLen	Length of the buffer/message
u8	BufferType	Type of buffer
u8	Is_Blocking	If set, triggers the notification in blocking mode

Returns

XST_SUCCESS in case of success XST_FAILURE in case of failure

XIpiPs_PollforDone

This function polls for an acknowledgement using the Observation Register.

Prototype

```
u32 XIpiPs_PollforDone(XMailbox *InstancePtr);
```

Parameters

The following table lists the `XIpiPs_PollforDone` function arguments.

Table 10: **XIpiPs_PollforDone Arguments**

Type	Name	Description
`XMailbox` *	InstancePtr	Pointer to the `XMailbox` instance

Returns

XST_SUCCESS in case of success XST_FAILURE in case of failure

XIpiPs_RecvData

This function reads an IPI message.

Prototype

```
u32 XIpiPs_RecvData(XMailbox *InstancePtr, void *MsgBufferPtr, u32 MsgLen,
u8 BufferType);
```

Parameters

The following table lists the `XIpiPs_RecvData` function arguments.

Table 11: **XIpiPs_RecvData Arguments**

Type	Name	Description
`XMailbox` *	InstancePtr	Pointer to the `XMailbox` instance
void *	MsgBufferPtr	Pointer to buffer to which the read message needs to be stored
u32	MsgLen	Length of the buffer/message
u8	BufferType	Type of buffer

Returns

- XST_SUCCESS if successful
- XST_FAILURE if unsuccessful

XIpiPs_RegisterIrq

Prototype

```
XStatus XIpiPs_RegisterIrq(XScuGic *IntcInstancePtr, XMailbox *InstancePtr,
u32 IpiIntrId);
```

XIpiPs_ErrorIntrHandler

Prototype

```
void XIpiPs_ErrorIntrHandler(void *XMailboxPtr);
```

XIpiPs_IntrHandler

Prototype

```
void XIpiPs_IntrHandler(void *XMailboxPtr);
```

Enumerations

Enumeration XMailbox_Handler

Contains XMAILBOX Handler Types.

Table 12: **Enumeration XMailbox_Handler Values**

Value	Description
XMAILBOX_RECV_HANDLER	For Recv Handler.
XMAILBOX_ERROR_HANDLER	For Error Handler.

Data Structure Index

The following is a list of data structures:

XMailbox

Holds the function pointers for the operations that can be performed.

Declaration

```
typedef struct
{
  u32(* XMbox_IPI_Send)(struct XMboxTag *InstancePtr, u8 Is_Blocking),
  u32(* XMbox_IPI_SendData)(struct XMboxTag *InstancePtr, void *BufferPtr,
u32 MsgLen, u8 BufferType, u8 Is_Blocking),
  u32(* XMbox_IPI_Recv)(struct XMboxTag *InstancePtr, void *BufferPtr, u32
MsgLen, u8 BufferType),
  XMailbox_RecvHandler RecvHandler,
  XMailbox_ErrorHandler ErrorHandler,
  void * ErrorRefPtr,
  void * RecvRefPtr,
  XMailbox_Agent Agent
} XMailbox;
```

Table 13: **Structure XMailbox member description**

Member	Description
XMbox_IPI_Send	Triggers an IPI to a destination CPU.
XMbox_IPI_SendData	Sends an IPI message to a destination CPU.
XMbox_IPI_Recv	Reads an IPI message.
RecvHandler	Callback for rx IPI event.
ErrorHandler	Callback for error event.
ErrorRefPtr	To be passed to the error interrupt callback.
RecvRefPtr	To be passed to the receive interrupt callback.
Agent	Used to store IPI Channel information.

XILINX

Appendix A

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator (DocNav) provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the Vivado® IDE, select **Help → Documentation and Tutorials**.
- On Windows, select **Start → All Programs → Xilinx Design Tools → DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In DocNav, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

Note: For more information on DocNav, see the Documentation Navigator page on the Xilinx website.

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

Copyright