PV2Way Communications I/O Development Guide
OpenCORE 2.0, rev. 2
Mar 27, 2010

Table of Contents

1. Introduction....4
2. PVCommsIONode....4
2.1. Two MIO Components / Two Ports....4
2.2. One MIO Component / One Port....5
2.3. Two MIO Components / One Port....6
2.4. One MIO Component / Two Ports....7
3. Implementing Baseband Comm IO components....7
3.1. PvmiMIOControl Interface....7
3.2. PvmiMediaTransfer and Data Transfer Models....7
3.3. PvmiCapabilityAndConfig Interface....8
4. FAQ....9

List of Figures

Figure 1: Two MIO Components / Two Ports....4
Figure 2: One MIO Component / One Port....5
Figure 3: Two MIO Components / One Port....6
Figure 4: One MIO Component / Two Ports....7

1. Introduction

This document establishes guidelines for developing baseband communications I/O components to work with the PV2Way engine's PVCommsIONode. The PVCommsIONode serves to abstract the details (media transfer model, unidirectional/bidirectional operation, etc.) of device specific baseband communications from the PV2Way engine. A knowledge of the interfaces detailed in the reference documents is implied.

2. PVCommsIONode

The PVCommsIONode is designed to abstract device specific baseband communication details from the rest of the PV2Way Engine, and allow flexibility for both baseband side and PV2Way Engine side sending and receiving of data. Below are the following use case scenarios for the PVCommsIONode:

2.1. Two MIO Components / Two Ports

Scenario: Two unidirectional baseband components, two unidirectional ports

The PVCommsIONode establishes a MediaDataTransfer session with each baseband component. Data flowing from the baseband input component is relayed to the output port, and data flowing from the input port is relayed to the baseband media output component.

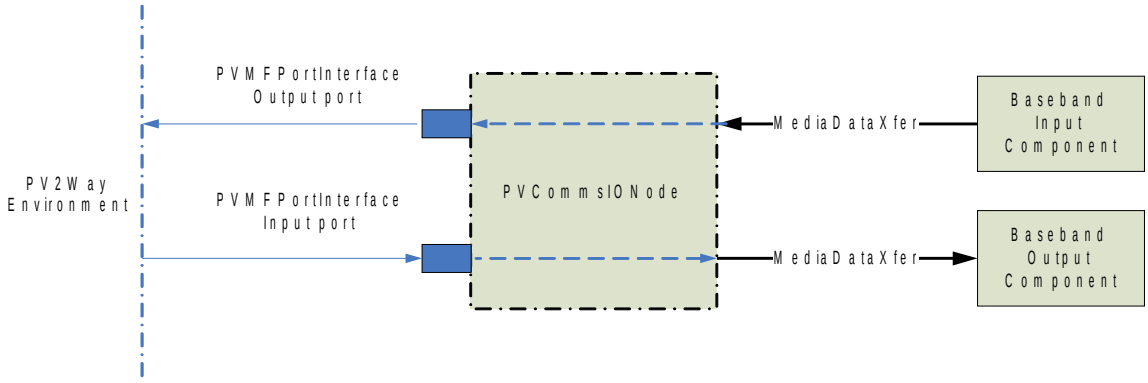

Figure 1: Two MIO Components / Two Ports

2.2. One MIO Component / One Port

Scenario: One bidirectional baseband component, one bidirectional port

The PVCommsIONode establishes a MediaDataTransfer session with a bidirectional baseband component. Data flowing from the baseband component is relayed and send out over the port interface, and data flowing in from the port interface is relayed to the baseband component.

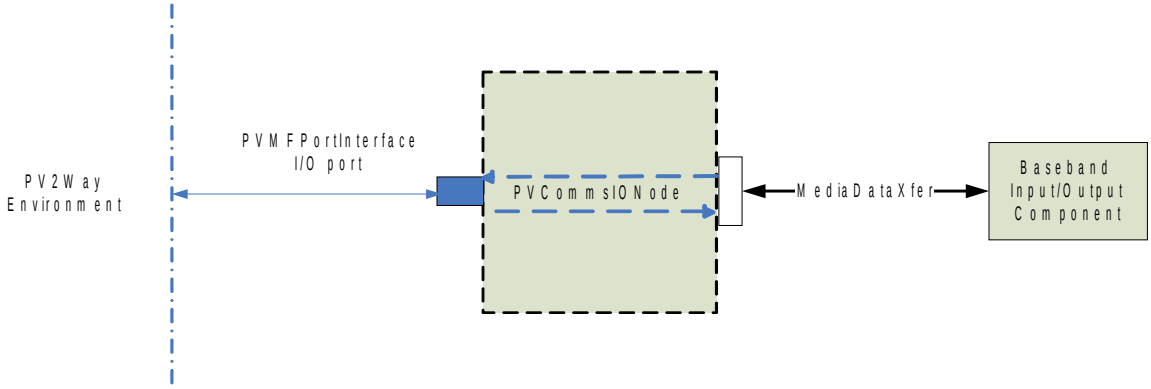

Figure 2: One MIO Component / One Port

2.3. Two MIO Components / One Port

Scenario: Two unidirectional baseband components, one bidirectional port

The PVCommsIONode establishes a MediaDataTransfer session with each baseband component. Data flowing from the baseband input component is relayed to the port, and sent out over the port interface, and data flowing in from the port interface is relayed to the baseband media output component.

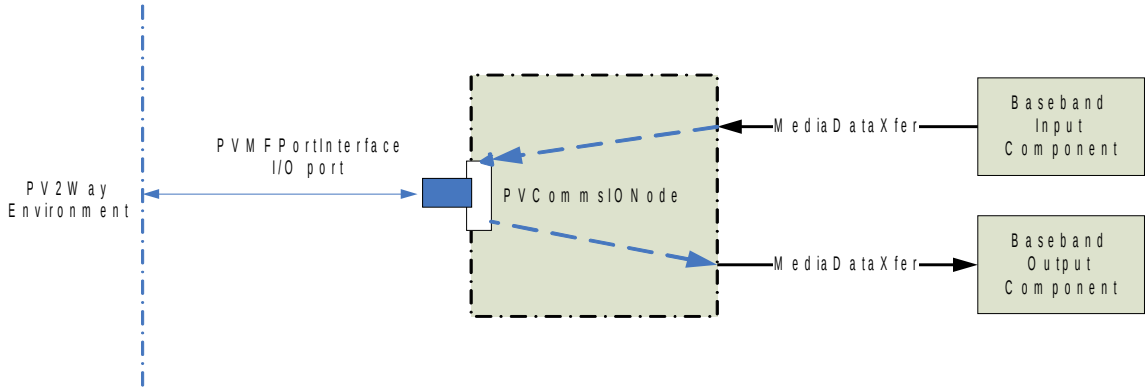

Figure 3: Two MIO Components / One Port

2.4. One MIO Component / Two Ports

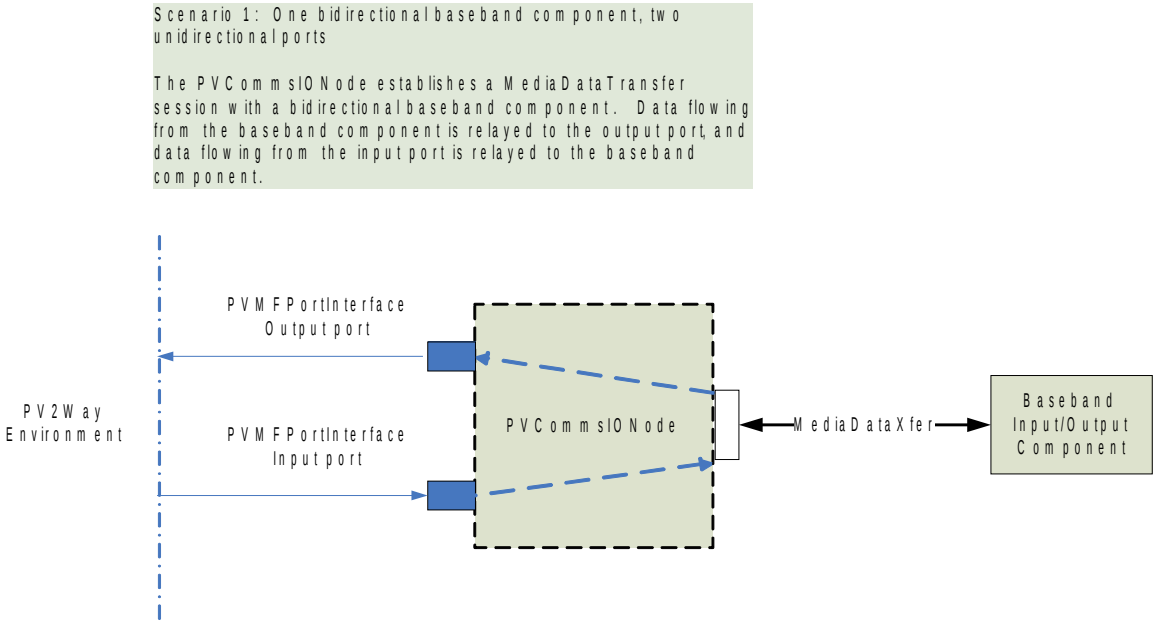

Figure 4: One MIO Component / Two Ports

An instance of PVCommsIONode should be created using the CPVCommsIONodeFactory class as detailed in the PV2Way API document.

3. Implementing Baseband Comm IO components

3.1. PvmiMIOControl Interface

All baseband comms components must implement the PvmiMIOControl interface. This provides a framework for the PVCommsIONode to programmatically start, stop and otherwise control the component. The component should return an implementation of the PvmiMediaTransfer interface via its ::CreateMediaTransfer() method. This PvmiMediaTransfer implementation is responsible for managing the actual transmission and reception of data.

3.2. PvmiMediaTransfer and Data Transfer Models

Currently, the PVCommsIONode supports the data push model for both input and output data. The PVCommsIONode is responsible for pushing output data to the baseband component, and the baseband component is responsible for pushing input data to the

PVCommsIONode. As this relates to the PvmiMediaTransfer interface, the baseband component should push input data by calling its peer's (PVCommsIONode) writeAsync() method. Accordingly, the PVCommsIONode should push output data to the baseband component by calling its peer's (baseband component) writeAsync() method.

Threading models for target environments may dictate that a pull model, requiring the use of the readAsync() methods, be used for data transfer and future support for this is planned.

3.3. PvmiCapabilityAndConfig Interface

All Baseband MIO components must implement the PvmiCapabilityAndConfig interface and expose a basic set of values that can be retrieved by a peer. At a minimum, a peer should be able to retrieve, and if necessary, set and enumerate values for the following keys:

…/input_formats;valtype=int32	If the component supports media output, it should allow a query on the current value for this key, and if more than one format is supported, it should allow enumeration and setting of this key as well. Format types are located in the file "pvmf_format_types.h". For H.324, PVMF_H223 format must be supported.
…/output_formats;valtype=int32	If the component supports media input, it should allow a query on the current value for this key, and if more than one format is supported, it should allow enumeration and setting of this key as well. Format types are located in the file "pvmf_format_types.h". For H.324, PVMF_H223 format must be supported.
…/input/transfer_model;valtype=uint32	If the component supports media output, it should allow a query on the current value for this key. The key should be read-only, and a value of 0 indicates the component uses a data pull model for output data, 1 for data push. Currently, the PVCommsIONode requires that this value be 1.
…/output/transfer_model;valtype=uint32	If the component supports media input, it should allow a query on the current value for this key. The key should be read-only, and a value of 0 indicates the component uses a data pull model for input data, 1 for data push. Currently, the PVCommsIONode requires that this value be 1.

4. FAQ

Q1. What do Comm source and Comm sink mean?

A1: Comm source and Comm sink refer to the MIO component(s) responsible for receiving data from the peer and sending data to the peer respectively.

Q2. Does this mean we need two mios in real VT case? One for reading data from modem(source) one for sending the data to modem(sink)?

A2: The Comm source and Comm sink may be part of the same MIO or may be separate MIOs. The implementation depends on what makes more sense for the driver integration. The PVCommsIONode can handle both cases. Please refer use cases described in section 2.

Q3. What are the various loopback options with pv2way engine ?

A3: The following are the loopback options. Descriptions can also be found in the pv2way API document.

PV_LOOPBACK_NONE: No loopback of data is involved here. This option is to be used for any point to point connection over 3g/sockets/other.

PV_LOOPBACK_COMM: Data is looped back external to the pv2way engine. But the pv2way engine needs to be aware of this mode so that call setup can be altered appropriately (MSD disabled).

PV_LOOPBACK_ENGINE: This was intended for looping back media at the engine level, without any involvement of the protocol stack. Currently not implemented.

PV_LOOPBACK_MUX: Loops back multiplexed data at the output of the protocol stack. So, multiplexed data is not sent to the comm mio in this case.

Q4: If we want to develop our real VT COMM MIO, do we need to set the loopback mode value to PV_LOOPBACK_NONE?

A4: Yes, if you want to connect Point-Point. If you need to test loopback at the Comm MIO / Driver level, you would have to set the loopback mode to PV_LOOPBACK_COMM.

Q5: If we change the mode to PV_LOOPBACK_NONE, will the 2way engine still connect to the PvmiMIOCommLoopback? Or how we add our real VT MIO into 2way engine?

A5: No. Changing the value does not change the COMM MIO that is being used. You would add the real Comm MIO(s) to the pv2way engine by using PVCommsIONodeFactory to create the PVCommsIONode that can control the real Comm MIO(s). Then use Terminal->Connect() to pass the PVCommsIONode to the pv2way engine.

Q6: From the view of 2 way engine behavior , what's the difference between PV_LOOPBACK_NONE and PV_LOOPBACK_COMM?

A6: The main difference is that the Master/Slave Determination procedure is skipped in the case of PV_LOOPBACK_COMM. The behaviour is similar as far as sending and receiving data is concerned. If you set the loopback mode (wrongly) as PV_LOOPBACK_NONE while using any loopback MIO, it will result in Master/Slave Determination timing out and the Connect command would fail.

Q7: From the view of our real COMM MIO, what's the difference between PV_LOOPBACK_NONE and PV_LOOPBACK_COMM?

A7: The real Comm MIO would send/receive data with the remote terminal while using PV_LOOPBACK_NONE, and would loop data back to itself while using PV_LOOPBACK_COMM.