DM280e

Programming Manual

Version 1.3, 05-2013

Table of Contents

Section 1: General

DM280e_Initialize

Synopsis

ViStatus DM280e_Initialize (resourceName, init_options, optionString, vi)

Arguments

ViRsrc resourceName (in)

Specifies the resourceName assigned by PCIe. For example, "PXI2::0::INSTR" is the resourceName of an instrument. resourceName can also be a logical IVI name.

ViInt32 init_options (in)

bit[0] = Specifies whether to perform DM280e_Reset() the device during the initialization procedure.
bit[1] = Reserved.
bit[2] = Reserved.
bit[3] = Specifies whether to reset the lock status to unlock during the initialization procedure.

ViConstString optionString (in)

Specifies available options (case insensitive):

a. "Simulate=0" OR "Simulate=1". This option string can place the instrument in simulation mode.
b. "DriverSetup=Model:<model number>"

Example: "Simulate=0, DriverSetup=Model:DM280e"

ViSession* vi (out)

Returns an instrument handle that you can use to identify the instrument in all subsequent function calls.

Descriptions

DM28oe_Initialize creates a new VISA session to the instrument specified in the resourceName, which provides methods to control and interact with the instrument, and returns a session handle you use to identify the session in all subsequent VI function calls.

This VI also allows you to configure the instrument into known states upon initialization via the init_options:

1. Set bit[0] to "1" to reset the instrument during instrument initialization.

2. Bit [3] is meant to reset the lock status to unlock during the initialization procedure.

   Lock Status:

   a. Command FIFO[1] is used to store any data written from the PCI/PCIe bus from the backplane, keeping the instructions in proper order for execution. Result FIFO is used to keep the data to be written to the PCI/PCIe bus back to the host computer. The status of Result FIFO will be read before the data is retrieved back to the host.

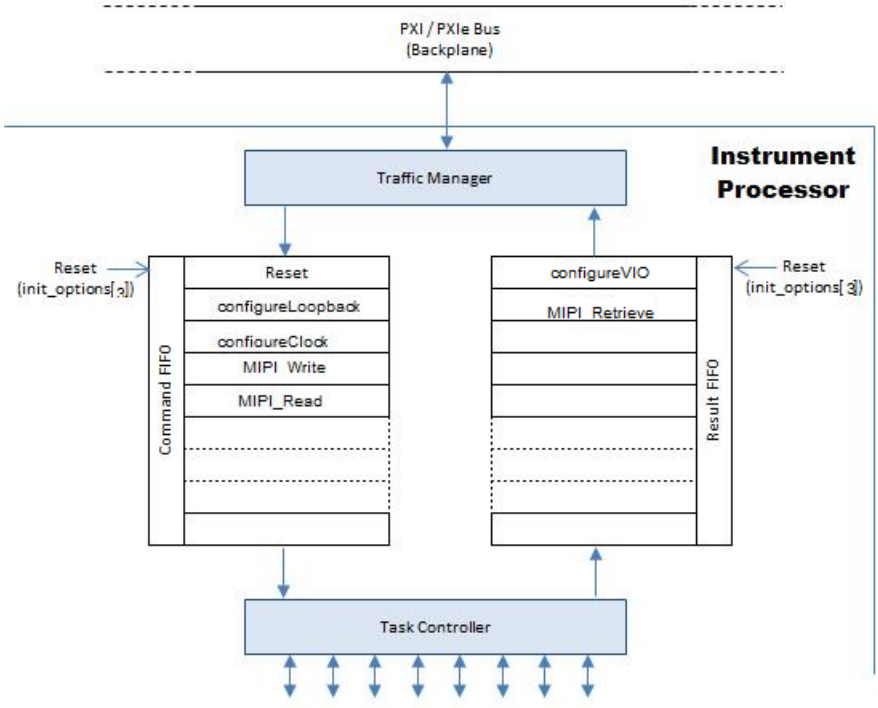

Figure 1: Command and Result FIFO

[1] FIFO is an acronym for **First In, First Out,** which is an abstraction related to ways of organizing and manipulation of data relative to time and prioritization. This expression describes the principle of a queue processing technique or servicing conflicting demands by ordering process by first-come, first-served (FCFS) behavior: where the persons leave the queue in the order they arrive, or waiting one's turn at a traffic control signal. FIFO blocks are designed in the processor of the instrument to manage data write and read to avoid traffic jam.

For example, if you call DM28oe_Configure_Loopback, the instrument driver will write this instruction into the Command FIFO for further processing. Then the instrument driver will check the status of the Result FIFO for its readiness, before retrieving the data back to the host computer.

Consider the following scenario:

a. When you call DM28oe_MIPI_RFFE_WR, the instrument driver will write this instruction into the Command FIFO for further processing. However, if the program is suddenly terminated (for whatever reasons), this instruction still remains in the Command FIFO. After the termination happens, if you reinitialize the instrument and try to do some other operations, you will always get the wrong result because DM28oe_MIPI_RFFE_WR instruction still remains in the FIFO queue. If you try to read some data, you will get error. In this case, when you DM28oe_Initialize, bits[3] needs reset.

b. An instrument can be configured to single-site or multi-site mode via DM28oe_ConfigureMultiSiteMode. By setting instrument to multi-site mode, you can actually split the available channels in the same instrument to serve multiple site testing. Example, you can use channels 0 of DM28oe to serve Site-1 and its channels 1 to serve Site-2, and both test sites can run either synchronously or asynchronously, for very flexible multi-site configuration.

   However, if you configure the instrument to single-site mode, but you actually run it in multi-site mode, the following may happen:

   > Let's imagine Site-1 is running production and are continuously writing instructions to the Command FIFO. Then another test program tries to initialize channels on Site-2 and it clears the Command FIFOs. This interruption may render Site-1 to appear "hang" as the Command FIFO is now empty.

   If the instrument is configured as multi-site mode, then it will "lock" the FIFO until the current instruction is completed before the next instruction write to the FIFO is possible.

DM28oe_Close

Synopsis

ViStatus DM28oe_Close (vi)

Arguments

ViSession vi (in)

Specifies the instrument handle.

Descriptions

DM28oe_Close closes the session specified in instrument handle. The instrument will maintain its last running state.

This function returns zero if successful and non-zero if otherwise.

DM280e_Reset

Synopsis

ViStatus DM280e_Reset (vi)

Arguments

ViSession vi (in)

Specifies the instrument handle.

Descriptions

DM280e_Reset resets the lock status and disable the configured clock frequency on the module. User needs to reconfigure the clock frequency with DM280e_CONFIGURE_MIPI_CLOCK command in order to perform operation of DM280e_MIPI_RFFE_WR and DM280e_MIPI_RFFE_RD. VIO and delay will remain unchanged.

This function returns zero if successful and non-zero if otherwise.

DM280e_ConfigureMultiSiteMode

Synopsis

ViStatus DM280e_ConfigureMultiSiteMode (vi, mode)

Arguments

ViSession vi (in)

Specifies the instrument handle.

ViInt32 mode (in)

Specifies the operation mode:

mode	Operation
0	Single-site (Default)
1	Multi-site

Descriptions

DM280e_ConfigureMultiSiteMode allows you configure the specified instrument to either in single-site or multi-site mode.

At single-site mode, lock/unlock operation is not performed hence yields better test time performance. Use bit[3] of init_options of DM280e_Initialize to clear the lock status upon initialization of instrument.

This function returns zero if successful and non-zero if otherwise.

Section 2: Configuration

DM280e_CONFIGURE_MIPI_CLOCK

Synopsis

ViStatus DM280e_CONFIGURE_MIPI_CLOCK (vi, freq_Hz)

Arguments

ViSession vi (in)

Specifies the instrument handle.

ViInt32 freq_Hz (in)

Specifies the MIPI clock speed.
32,000 <= freq_Hz <= 26,000,000.

Descriptions

DM280e_CONFIGURE_MIPI_CLOCK configures the speed of the MIPI operation.

This function returns zero if successful and non-zero if otherwise.

DM280e_CONFIGURE_LOOPBACK

Synopsis

ViStatus DM280e_CONFIGURE_LOOPBACK (vi, ch, loopback)

Arguments

ViSession vi (in)

Specifies the instrument handle.

ViInt32 ch (in)

Specifies the selected channel. The selected channel will be the master and the other channel will be slave.

ViInt32 loopback (in)

Specifies the loopback enable.
[1] enable loopback
[0] disable loopback

Descriptions

DM280e_CONFIGURE_LOOPBACK configures the loopback operation of the module.

This function returns zero if successful and non-zero if otherwise.

DM280e_CONFIGURE_MIPI_DELAY

Synopsis

ViStatus DM280e_CONFIGURE_MIPI_DELAY (vi, ch, delay)

Arguments

ViSession vi (in)

Specifies the instrument handle.

ViInt32 ch (in)

Specifies the selected channel.

ViInt32 delay (in)

Specifies the delay inserted before data sampling during read operation starts, the resolution is defined in terms of 1/2f (f = configured frequency), or half a clock cycle of configured clock frequency.

Descriptions

DM280e_CONFIGURE_MIPI_DELAY sets the delay inserted before data sampling of a read operation.

This function returns zero if successful and non-zero if otherwise.

Example:

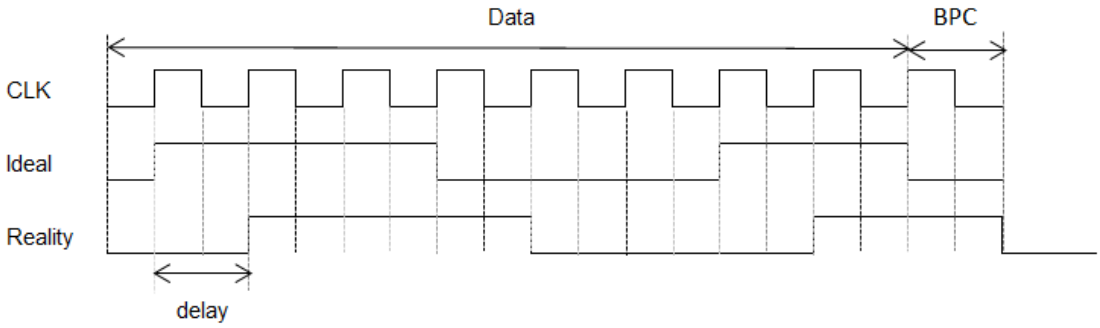

Figure 2: Ideal vs. Reality data

The distance between and Master (DM280e) and Slave (DUT) during production testing may be greater than 15cm (MIPI specification). Therefore propagation delay may impact the reading of the data sent by the slave subsystem represented in figure 2.

In figure 2, what should have been read as 0xE3 ideally, would in fact be read as 0x71. It can be seen that the delay of the signal is 2 half cycle of the clock in reality. Hence DM280e_MIPI_Configure_Delay can be set as 2 to delay when to start reading in the data and therefore to counter the effects of propagation delay.

DM280e_CONFIGURE_VOLTAGE_SUPPLY

Synopsis

ViStatus DM280e_CONFIGURE_VOLTAGE_SUPPLY (vi, target_vio, actual_vio)

Arguments

ViSession vi (in)

Specifies the instrument handle.

ViReal32 target_vio (in)

Specifies the targeted VIO for the module in volt (V).
1.45<=target_vio <= 3.9V

ViReal32* actual_vio (out)

Returns the actual VIO of the module in volt (V).

Descriptions

DM280e_CONFIGURE_VOLTAGE_SUPPLY sets the targeted VIO of the module and returns the reading of the actual VIO from the module.

This function returns zero if successful and non-zero if otherwise.

DM280e_CONFIGURE_BPC

Synopsis

ViStatus DM280e_CONFIGURE_BPC (vi, BPC)

Arguments

ViSession vi (in)

Specifies the instrument handle.

ViInt32 BPC (in)

Specifies the BPC delay inserted each command or data frame involving changing of bus ownership, in terms of the number of full speed clock cycles.
Valid range: BPC>4.

Descriptions

DM280e_CONFIGURE_BPC configures the Bus Park Cycle delay inserted when involving changing of bus ownership.

This function returns zero if successful and non-zero if otherwise.

Section 3: MIPI Operation

DM28oe_MIPI_RFFE_WR

Synopsis

ViStatus DM28oe_MIPI_RFFE_WR (vi, ch, Command, Data)

Arguments

ViSession vi (in)

Specifies the instrument handle.

ViInt32 ch (in)

Specifies the selected channel.

ViInt32 Command (in)

Specifies the command for the operation.

Command_frame [11:0] = { Slave_Address[2], Command_frame_lower }		
Register 0 Write	Command_frame_lower[7:0]	8'b{ 1, Data[6:0] }
Register Write	Command_frame_lower[7:0]	8'b{ 0,1,0, Address[4:0] }
Extended Register Write	Command_frame_lower[7:0]	8'b{ 0,0,0,0, BC[3:0] }
Extended Register Write Long	Command_frame_lower[7:0]	8'b{ 0,0,1,1,0, BC[2:0] }

1. Command frame excludes parity bit, which is handled by firmware
2. Slave address is 4 bits and common for all operations.
3. BC => byte_count for data frame, ie. If BC is zero, then M = 1, refer to data frame table

ViInt32* Data (in)

Specifies a pointer to an array corresponding to each 8 bit data frame that will be written to the channel.

Data_frame[7:0]		
Register 0 Write	-	-
Register Write	One Data_frame	-
Extended Register Write	One Data_frame for Address	Data_frame[0]= address [7:0], followed by M^2 number of Data Frames containing up to 16 bytes of data. ie: if M = 3, Data_frame[1] = data0, Data_frame[2] = data1, Data_frame[3] = data2.
Extended Register Write Long	Two Data_frame for Address	Data_frame[0]= address [15:8] and Data_frame[1] = address [7:0], followed by M^2 number of Data Frames containing up to 8 bytes of data. ie: if M = 3, Data_frame[2] = data0, Data_frame[3] = data1, Data_frame[4] = data2

1. Data frame excludes parity bit, which is handled by firmware.
2. BC => byte_count for data frame, ie. If BC is zero, then M = 1, refer to data frame table

Descriptions

DM28oe_MIPI_RFFE_WR writes the data into the channel according to the operation selected.

This function returns zero if successful and non-zero if otherwise.

Example:
This will perform the operation of Extended Register Write Long.

```
Command =(0xF<<8)|(0x6<<3)|(0x2); //Extended register write long, 3 bytes of
data
Data[0] = 0x1; //Address [15:8]
Data[1] = 0x23; //Address [7:0]
Data[2] = 0x31; //Byte 1 data
Data[3] = 0x31; //Byte 2 data
Data[4] = 0x31; //Byte 3 data
MIPI_Write("280e_1",Command, Data); //"280_1" is the channel selected.
```

For the Command, `(0xF<<8)` is the slave address, `(0x6<<3)` is the command frame and `(0x2)` is data_frame with the number of byte to be written into the channel.

Note: Refer to appendix for more details.

DM28oe_MIPI_RFFE_RD

Synopsis

ViStatus DM28oe_MIPI_RFFE_RD (vi, ch, speed, Command, Data)

Arguments

ViSession vi (in)

Specifies the instrument handle.

ViInt32 ch (in)

Specifies the selected channel.

ViInt32 speed (in)

Specifies the speed of reading.

0 = full speed, 1 = half speed.

ViInt32 Command (in)

Specifies the command of read operation.

Command_frame [11:0] = { Slave_Address[2], Command_frame_lower }		
Register Read	Command_frame_lower[7:0]	8'b{ 0,1,1, Address[4:0] }
Extended Register Read	Command_frame_lower[7:0]	8'b{ 0,0,1,0, BC[3:0] }
Extended Register Read Long	Command_frame_lower[7:0]	8'b{ 0,0,1,1,1, BC[2:0] }

1. Command frame excludes parity bit, which is handled by firmware.
2. Slave address is 4 bits and common for all operations.

ViInt32* Data (in)

The array of addresses that will be reading from the channel.

Data_frame[7:0]		
Register Read	-	-
Extended Register Read	One Data frame for Address	Data_frame[0]= address [7:0]
Extended Register Read Long	Two Data frame for Address	Data_frame[0]= address [15:8]; Data_frame[1]= address [7:0]

1. Data frame excludes parity bit, which is handled by firmware.

Descriptions

DM28oe_MIPI_RFFE_RD will read the data from the corresponding addresses.

This function returns zero if successful and non-zero if otherwise.

Example:

```
Command = (0xF<<8)|(0x2<<4)|(0x2);//Specifies command for Register Read
DataRead[0] = 0x1; //Specifies the address to read from.
MIPI_Read("280e_1",1,Command, DataRead) //"280e_1" is the channel selected,"1"
indicates half speed for the read operation.
```

This will perform the operation of Extended Register Read.
For the Command, `(0xF<<8)` is the slave address, `(0x2<<4)` is the command frame and `(0x2)` is the number of byte to be read from the channel.

Note: Refer to appendix for more details.

DM280e_MIPI_RFFE_RETRIEVE

Synopsis

ViStatus DM280e_MIPI_RFFE_RETRIEVE (vi, ch, rd_byte_data_count, rd_data, parity_check)

Arguments

ViSession vi (in)

Specifies the instrument handle.

ViInt32 ch (in)

Specifies the selected channel.

ViInt32* rd_byte_data_count (out)

Specifies the number of bytes of data retrieved from the channel.

ViInt32* rd_data(out)

Returns the array of data that retrieved from channel.

ViInt32* parity_check (out)

Returns the array of parity_check corresponding to the array of rd_data that retrieved from the channel.

1 => Check failed

0 => Check passed

Descriptions

DM280e_MIPI_RFFE_RETRIEVE retrieves and returns an array of data specified in operation DM280e_MIPI_RFFE_RD.

This function returns zero if successful and non-zero if otherwise.

Section 4: Appendix

Figure 3: RFFE Supported Command Sequences

Section 5: Revision History

1.0	APR 2013	INITIAL RELEASE
1.1	MAY 2013	ADDED DM280E_CONFIGUREMULTISITEMODE
1.2	MAY 2013	ADDED DETAILS OF DM280E_MIPI_CONFIGUREDELAY
1.3	MAY 2013	CHANGE THE VALID RANGE OF BPC

Section 6: Contact Us

To obtain service, warranty or technical assistance, please contact Aemulus.

Aemulus Corporation Berhad
Krystal Point, B-2-04, B-2-05, B-2-06 & B-2-07
303, Jalan Sultan Azlan Shah,
11900 Penang, Malaysia
Tel: +604 6446399
Fax: +604 6466799

Web: www.aemulus.com
Email: enquiry@aemulus.com

Product specifications and descriptions in this document are subject to change without prior notice.