Xilinx Answer 64761

Bitstream Loading across the PCI Express Link in UltraScale Devices for Tandem PCIe and Partial Reconfiguration

Important Note: This downloadable PDF of an Answer Record is provided to enhance its usability and readability. It is important to note that Answer Records are Web-based content that are frequently updated as new information becomes available. You are reminded to visit the Xilinx Technical Support Website and review (Xilinx Answer 64761) for the latest version of this Answer.

Introduction

The Media Configuration Access Port (MCAP) is a new configuration interface available for UltraScale devices. This interface is integrated into the PCI Express hard block and provides access to the FPGA configuration logic through the PCI Express hard block when enabled. The MCAP can be enabled and used within the PCI Express solution IP for endpoint configurations that use either Tandem or PR over PCIe. When these options are selected, an MCAP Vendor Specific Extended Capability (VSEC) is added to the PCI Express configuration space. The MCAP VSEC can be used to load additional bitstreams and configuration command sequences into the FPGA after the initial configuration has been loaded.

This answer record provides drivers and software, for both Windows and Linux that can interact with MCAP-enabled UltraScale designs. This document provides instructions for installing and using this software along with instructions on how to use the MCAP Extended Capability. See the UltraScale Architecture Gen3 Integrated Block for PCI Express Product Guide (PG156) for details relating to your hardware implementation and designing with the PCIe IP.

PCI Express MCAP Extended Capability

When the MCAP is enabled in the PCI Express Solution IP, the MCAP Vendor Specific Extended Capability is added to the PCI Express configuration space. This capability will appear in the configuration space at a base address byte offset of 0x340 for UltraScale devices. To properly identify this capability, the PCIe Next Capability pointer linked list should be followed until the MCAP VSEC is discovered and properly identified by the PCI Express Extended Capability ID, Capability Version, and VSEC ID Register values. See the 'Identifying the PCIe MCAP Extended Capability' section for additional details.

PCIe MCAP Driver for Windows

Installing MCAP Software

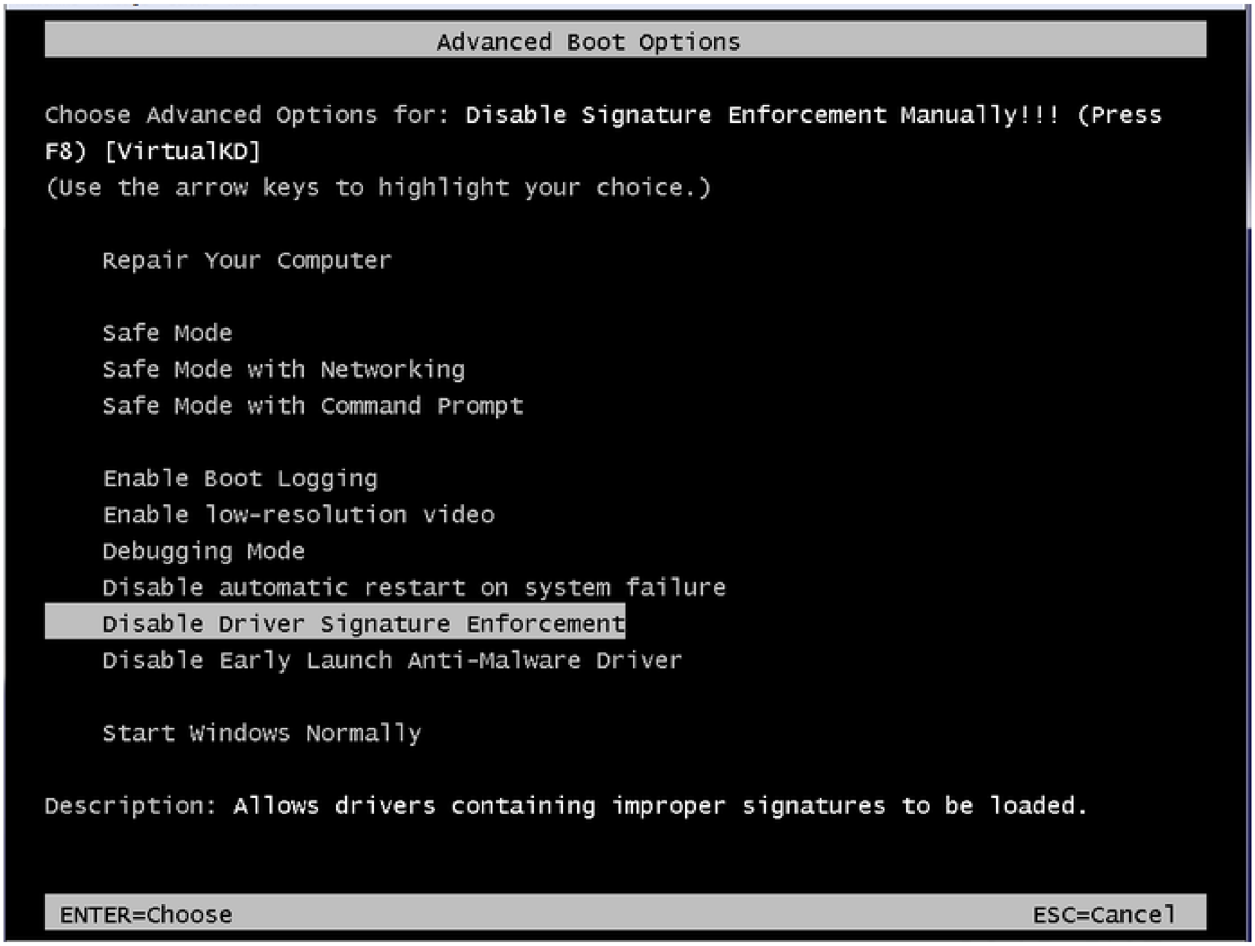
Along with this document, the following file is provided for Windows 7 64 bit OS.

Driver Installation: xilinxsetup64.exe

Disable Driver Signature Enforcement

Windows allows only drivers with valid signatures obtained from trusted certificate authorities to load in Windows 7 64 bit OS. Windows drivers provided for this design do not have a valid signature and hence user has to disable Driver Signature Enforcement on the host system every time the user wants to use these drivers on the host machine. Please follow the instructions below to disable driver signature enforcement.

1. Go to the Advanced Boot Options menu by pressing F8 after powering ON the host system.
2. Select "Disable Driver Signature Enforcement" option as shown in Figure 1, and hit Enter.



```
                         Advanced Boot Options

Choose Advanced Options for: Disable Signature Enforcement Manually!!! (Press
F8) [VirtualKD]
(Use the arrow keys to highlight your choice.)

    Repair Your Computer

    Safe Mode
    Safe Mode with Networking
    Safe Mode with Command Prompt

    Enable Boot Logging
    Enable low-resolution video
    Debugging Mode
    Disable automatic restart on system failure
    Disable Driver Signature Enforcement
    Disable Early Launch Anti-Malware Driver

    Start Windows Normally

Description: Allows drivers containing improper signatures to be loaded.


ENTER=Choose                                                     ESC=Cancel
```

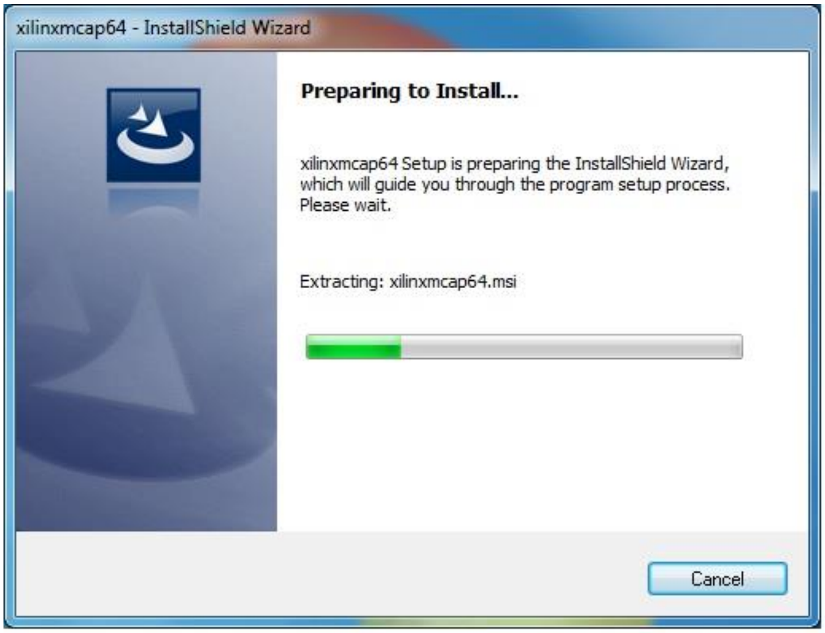
Figure 1: Disable Driver Signature Enforcement

Installing the Drivers on Windows 7 64-bit

1. Connect the board in the PCIe slot with an external power supply.
2. Power ON the board and then load the stage 1 (Tandem) or initial configuration (PR) bitstream through JTAG or from a flash device.
3. Power ON the host machine and then boot Windows.
4. Press F8 and Disable Driver Signature Enforcement.

Note: Disable Driver Signature enforcement is a very important step, otherwise the driver does not get installed.

5. After Windows has booted, navigate to the directory where setup files are present.
6. Click the setup file - xilinxsetup64.exe



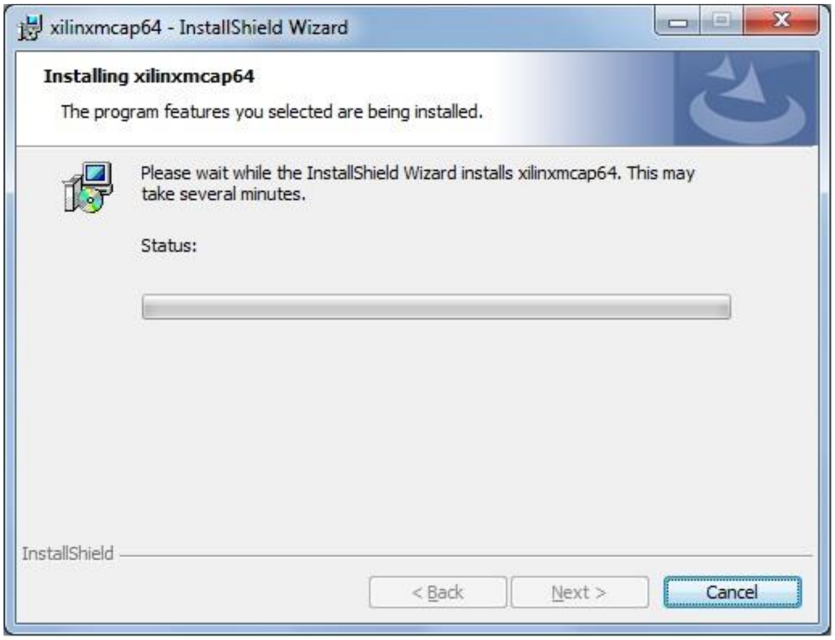
xilinxmcap64 - InstallShield Wizard

Preparing to Install...

xilinxmcap64 Setup is preparing the InstallShield Wizard, which will guide you through the program setup process. Please wait.

Extracting: xilinxmcap64.msi

Figure 2: Extracting the installer



xilinxmcap64 - InstallShield Wizard

Installing xilinxmcap64

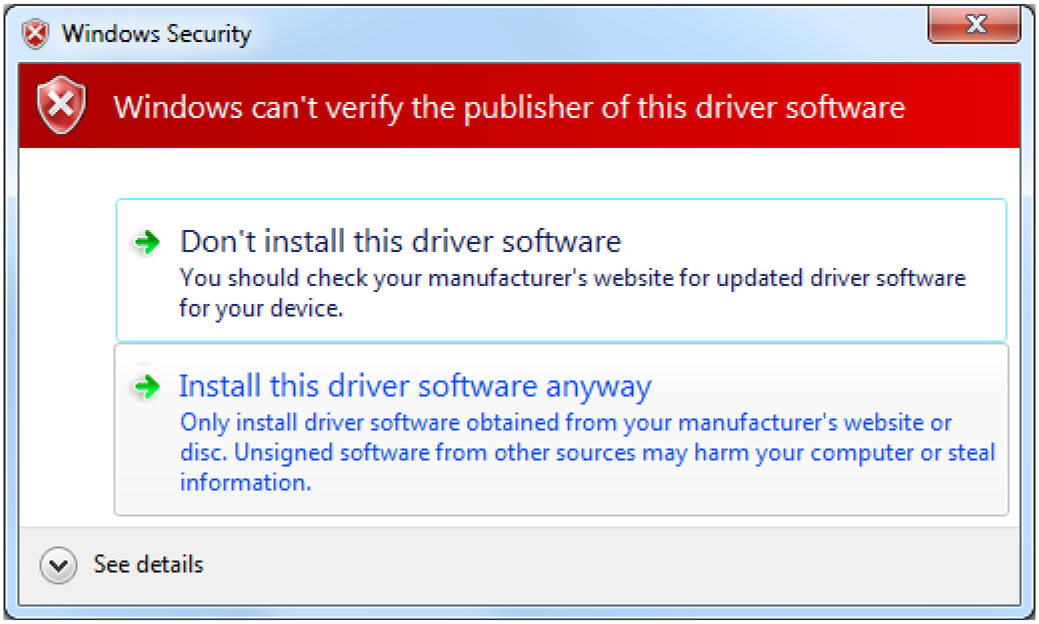
The program features you selected are being installed.

Please wait while the InstallShield Wizard installs xilinxmcap64. This may take several minutes.

Status:

Figure 3: Installing the driver

7. Click “Install this driver software anyway”



Windows Security

Windows can't verify the publisher of this driver software

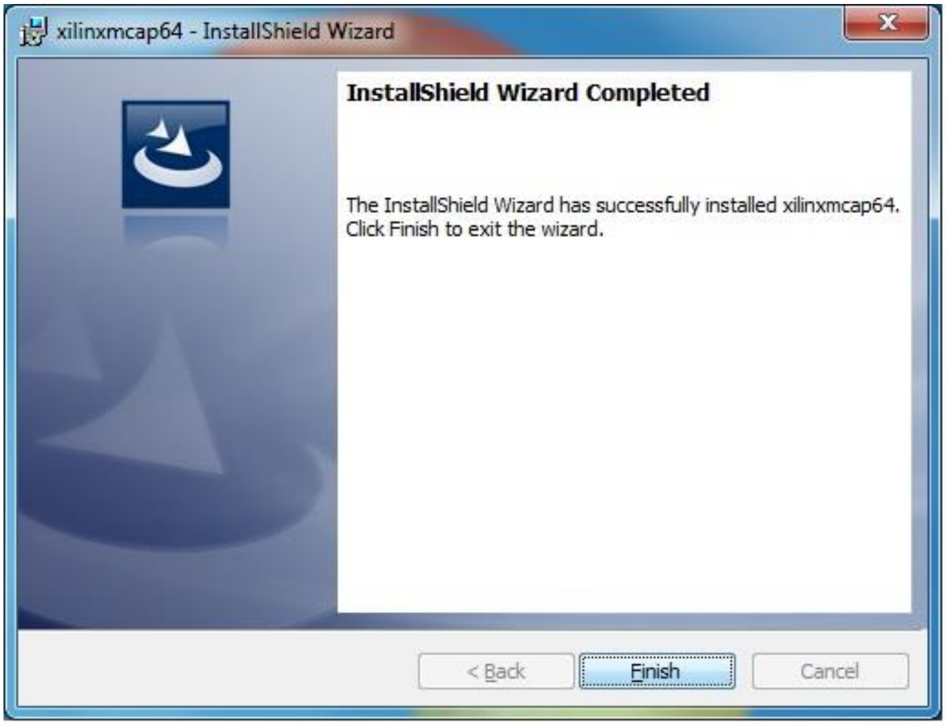
Don't install this driver software

You should check your manufacturer's website for updated driver software for your device.

Install this driver software anyway

Only install driver software obtained from your manufacturer's website or disc. Unsigned software from other sources may harm your computer or steal information.

Figure 4: Pass security check

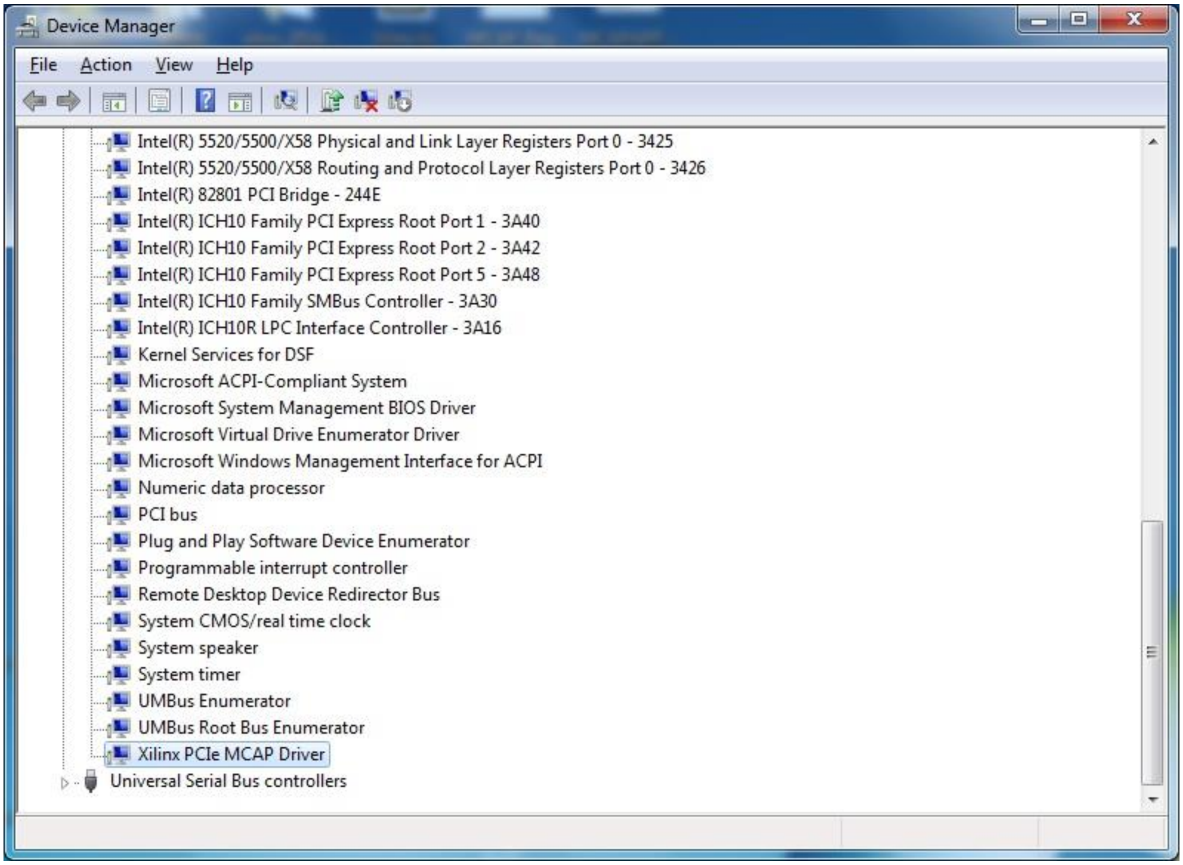


xilinxmcap64 - InstallShield Wizard

InstallShield Wizard Completed

The InstallShield Wizard has successfully installed xilinxmcap64. Click Finish to exit the wizard.

Figure 5: Installation complete



- Intel(R) 5520/5500/X58 Physical and Link Layer Registers Port 0 - 3425
- Intel(R) 5520/5500/X58 Routing and Protocol Layer Registers Port 0 - 3426
- Intel(R) 82801 PCI Bridge - 244E
- Intel(R) ICH10 Family PCI Express Root Port 1 - 3A40
- Intel(R) ICH10 Family PCI Express Root Port 2 - 3A42
- Intel(R) ICH10 Family PCI Express Root Port 5 - 3A48
- Intel(R) ICH10 Family SMBus Controller - 3A30
- Intel(R) ICH10R LPC Interface Controller - 3A16
- Kernel Services for DSF
- Microsoft ACPI-Compliant System
- Microsoft System Management BIOS Driver
- Microsoft Virtual Drive Enumerator Driver
- Microsoft Windows Management Interface for ACPI
- Numeric data processor
- PCI bus
- Plug and Play Software Device Enumerator
- Programmable interrupt controller
- Remote Desktop Device Redirector Bus
- System CMOS/real time clock
- System speaker
- System timer
- UMBus Enumerator
- UMBus Root Bus Enumerator
- Xilinx PCIe MCAP Driver

Universal Serial Bus controllers

Figure 6: Installed driver appears in "System devices"

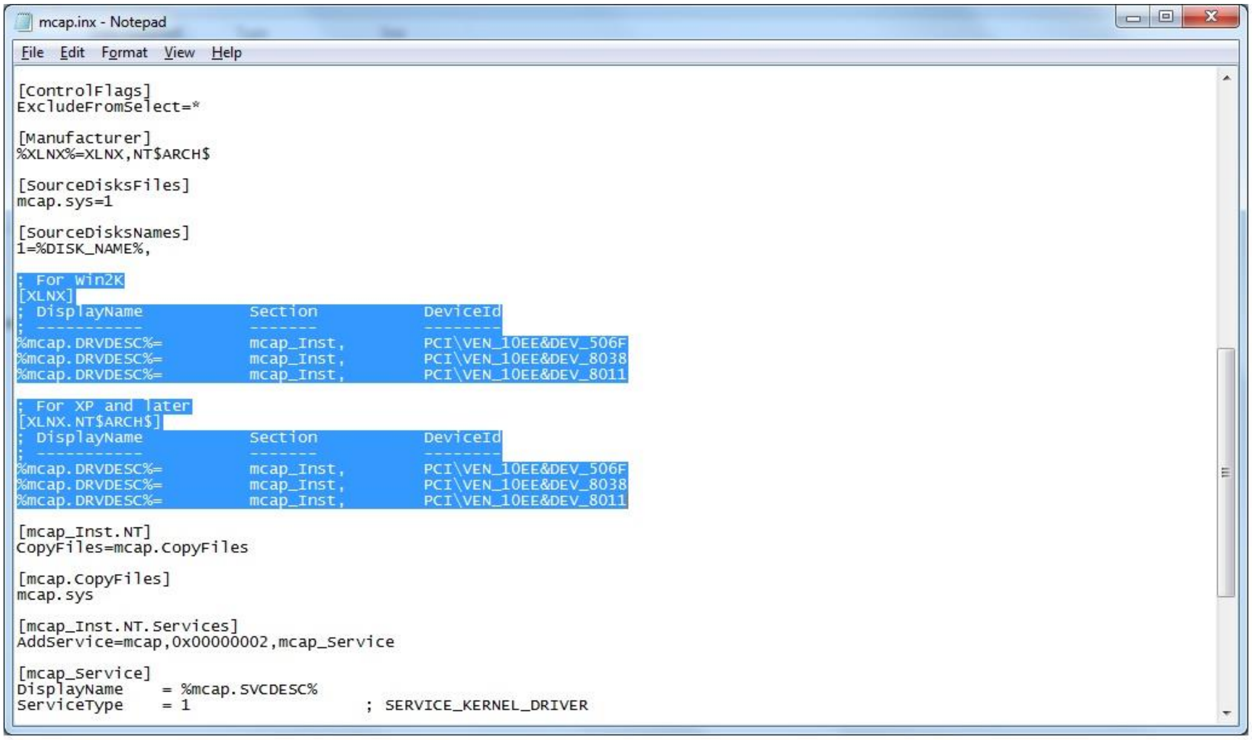
Traditionally the Vendor ID and Device ID for the hardware and associated drivers and software will be coded to a single value. This software has been designed to work with a Vendor ID of 10EE and one of the following device IDs:

- 8011
- 8038
- 506F

Users requiring their own Device ID and Vendor ID must add the same to mcap.inf present in the Driver's installation location. The default installation location is C:\Program Files (x86)\Xilinx\mcap\mcap64\

1. Open the mcap.inf file

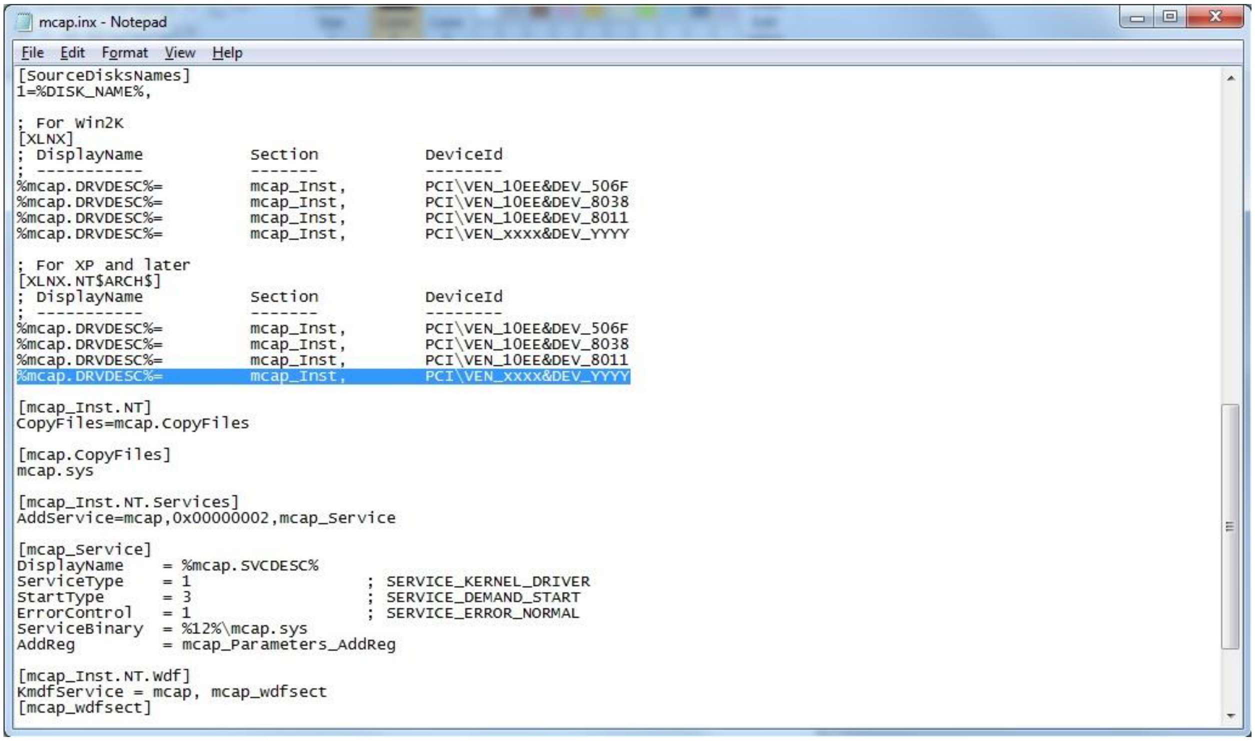
2. Check if the required VID & PID Combination is present in the mcap.inf file.



```
[ControlFlags]
ExcludeFromSelect=*

[Manufacturer]
%XLNX%=XLNX,NT$ARCH$

[SourceDisksFiles]
mcap.sys=1

[SourceDisksNames]
1=%DISK_NAME%,

; For Win2K
[XLNX]
; DisplayName           Section           DeviceId
; -----------           -------           --------
%mcap.DRVDESC%=         mcap_Inst,        PCI\VEN_10EE&DEV_506F
%mcap.DRVDESC%=         mcap_Inst,        PCI\VEN_10EE&DEV_8038
%mcap.DRVDESC%=         mcap_Inst,        PCI\VEN_10EE&DEV_8011

; For XP and later
[XLNX.NT$ARCH$]
; DisplayName           Section           DeviceId
; -----------           -------           --------
%mcap.DRVDESC%=         mcap_Inst,        PCI\VEN_10EE&DEV_506F
%mcap.DRVDESC%=         mcap_Inst,        PCI\VEN_10EE&DEV_8038
%mcap.DRVDESC%=         mcap_Inst,        PCI\VEN_10EE&DEV_8011

[mcap_Inst.NT]
CopyFiles=mcap.CopyFiles

[mcap.CopyFiles]
mcap.sys

[mcap_Inst.NT.Services]
AddService=mcap,0x00000002,mcap_Service

[mcap_Service]
DisplayName    = %mcap.SVCDESC%
ServiceType    = 1                   ; SERVICE_KERNEL_DRIVER
```

Figure 7: Mcap.inf file

3. To add a new combination Add the Highlighted line.
   The XXX and YYYY are the desired VID & PID respectively.
   Note: Add the Same line under [XLNX] and [XLNX.NT$ARCH$] sections.



```
[SourceDisksNames]
1=%DISK_NAME%,

; For Win2K
[XLNX]
; DisplayName           Section           DeviceId
; -----------           -------           --------
%mcap.DRVDESC%=         mcap_Inst,        PCI\VEN_10EE&DEV_506F
%mcap.DRVDESC%=         mcap_Inst,        PCI\VEN_10EE&DEV_8038
%mcap.DRVDESC%=         mcap_Inst,        PCI\VEN_10EE&DEV_8011
%mcap.DRVDESC%=         mcap_Inst,        PCI\VEN_XXXX&DEV_YYYY

; For XP and later
[XLNX.NT$ARCH$]
; DisplayName           Section           DeviceId
; -----------           -------           --------
%mcap.DRVDESC%=         mcap_Inst,        PCI\VEN_10EE&DEV_506F
%mcap.DRVDESC%=         mcap_Inst,        PCI\VEN_10EE&DEV_8038
%mcap.DRVDESC%=         mcap_Inst,        PCI\VEN_10EE&DEV_8011
%mcap.DRVDESC%=         mcap_Inst,        PCI\VEN_XXXX&DEV_YYYY

[mcap_Inst.NT]
CopyFiles=mcap.CopyFiles

[mcap.CopyFiles]
mcap.sys

[mcap_Inst.NT.Services]
AddService=mcap,0x00000002,mcap_Service

[mcap_Service]
DisplayName    = %mcap.SVCDESC%
ServiceType    = 1                   ; SERVICE_KERNEL_DRIVER
StartType      = 3                   ; SERVICE_DEMAND_START
ErrorControl   = 1                   ; SERVICE_ERROR_NORMAL
ServiceBinary  = %12%\mcap.sys
AddReg         = mcap_Parameters_AddReg

[mcap_Inst.NT.Wdf]
KmdfService = mcap, mcap_wdfsect
[mcap_wdfsect]
```

Figure 8: Modifying Mcap.inf file

4. Invoke Device Manager by typing devmgmt.msc at the run prompt.

5. A New device with yellow question mark with "PCIe Memory controller" should be visible.

6. Examine the VID and PID of the PCIe Memory controller. (Right Click->Properties->Detail->HW Ids)

7. If the VID and PID are equal to the one which has been added recently, right click on the device entry and click on Update Driver Software.

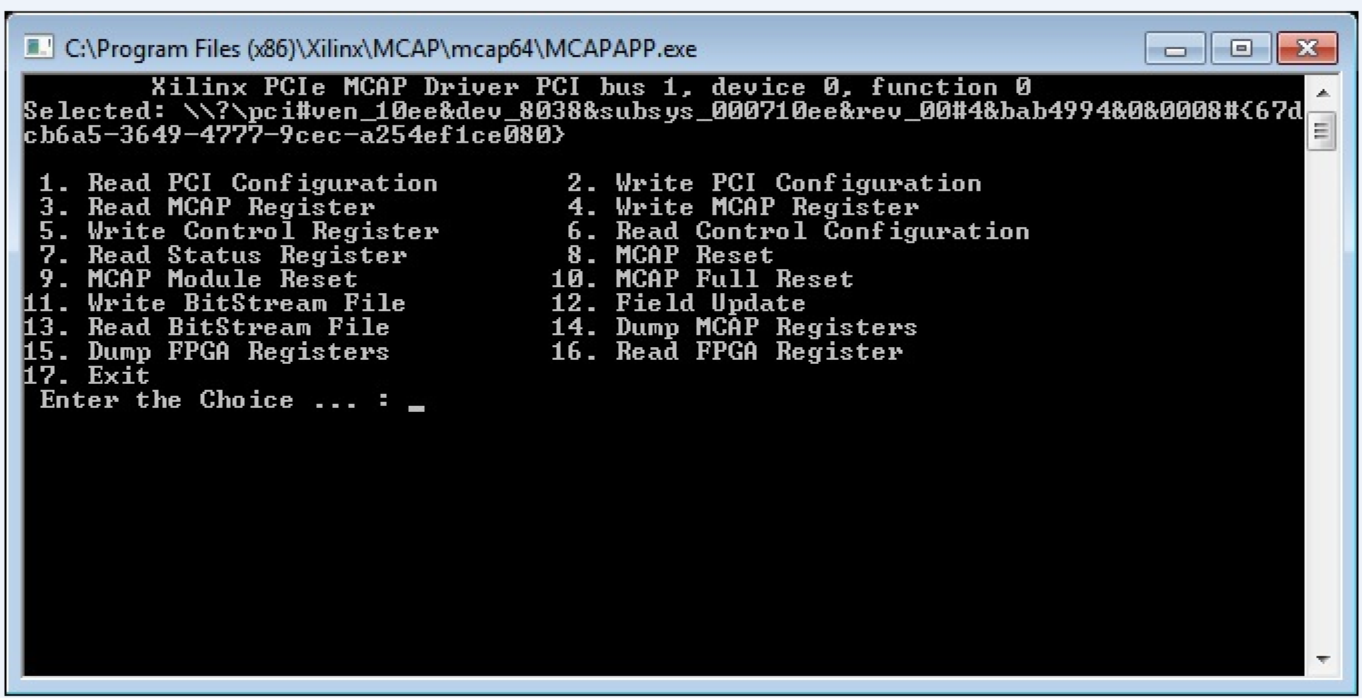
8. Select Browse My Computer for Driver Software option and manually assign the path of mcap driver to load the same.

Using MCAP Application

After installing the application, it provides options to open either the console as shown below or a GUI based application. Shortcuts to both are placed on the desktop by default.

MCAPAPPGUI.exe

Console (MCAPAPP.exe)



C:\Program Files (x86)\Xilinx\MCAP\mcap64\MCAPAPP.exe

```
        Xilinx PCIe MCAP Driver PCI bus 1, device 0, function 0
Selected: \\?\pci#ven_10ee&dev_8038&subsys_000710ee&rev_00#4&bab4994&0&0008#{67d
cb6a5-3649-4777-9cec-a254ef1ce080}

 1. Read PCI Configuration         2. Write PCI Configuration
 3. Read MCAP Register             4. Write MCAP Register
 5. Write Control Register         6. Read Control Configuration
 7. Read Status Register           8. MCAP Reset
 9. MCAP Module Reset             10. MCAP Full Reset
11. Write BitStream File          12. Field Update
13. Read BitStream File           14. Dump MCAP Registers
15. Dump FPGA Registers           16. Read FPGA Register
17. Exit
 Enter the Choice ... : _
```

Figure 9: MCAP APP Console

GUI Based Application (MCAPAPPGUI.exe)

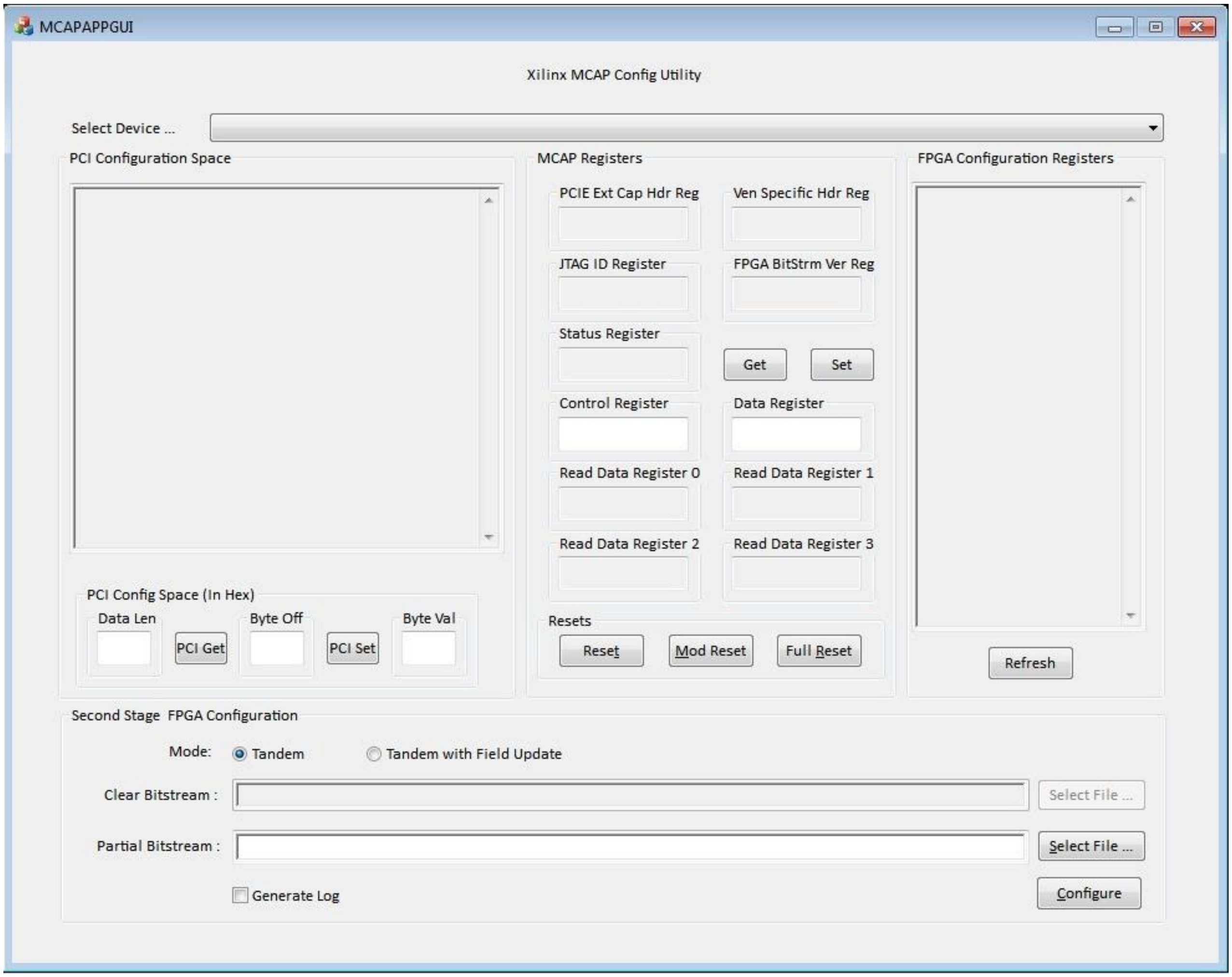
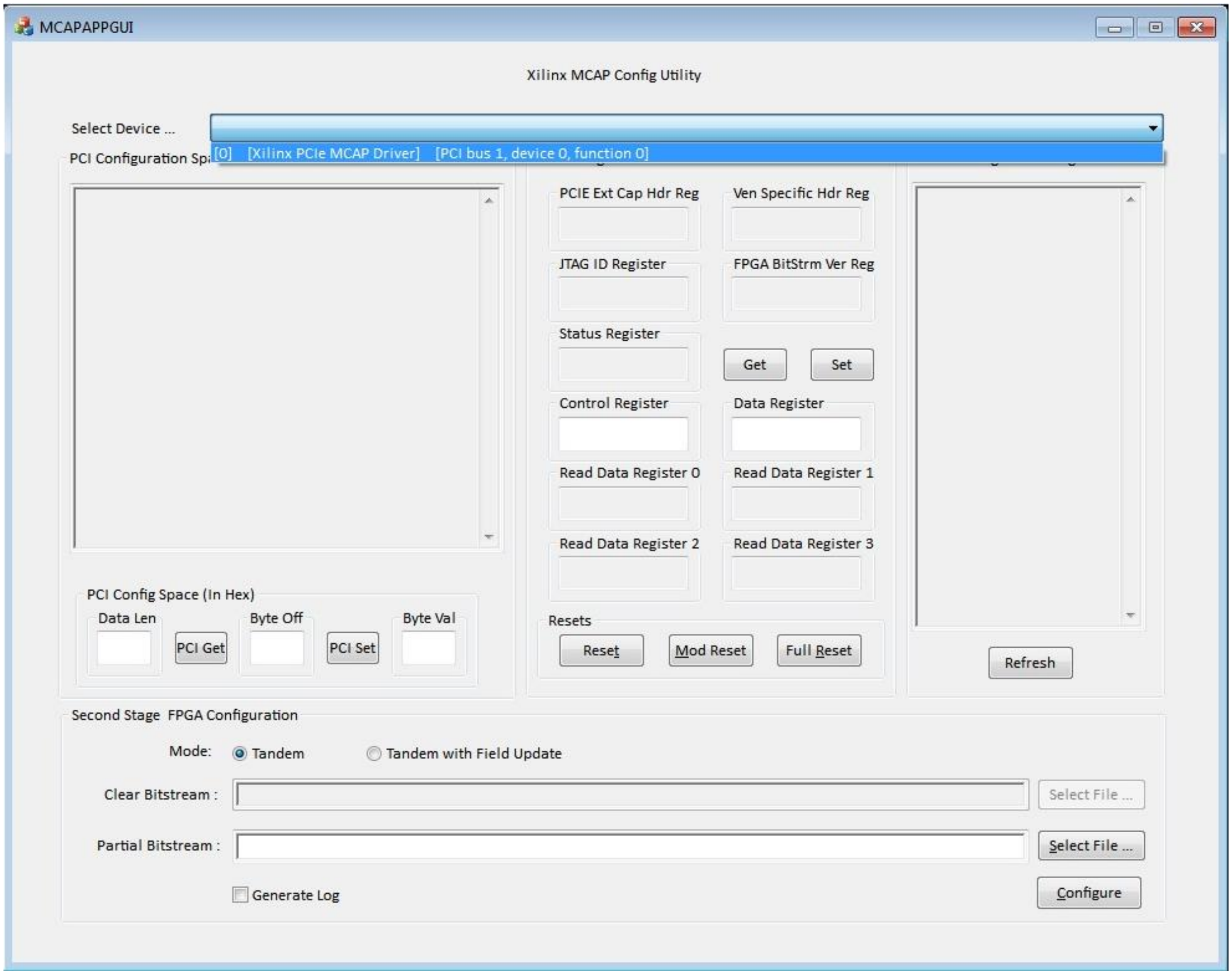


Figure 10: MCAP APP GUI

Select a device from the combo list at the top of the application.



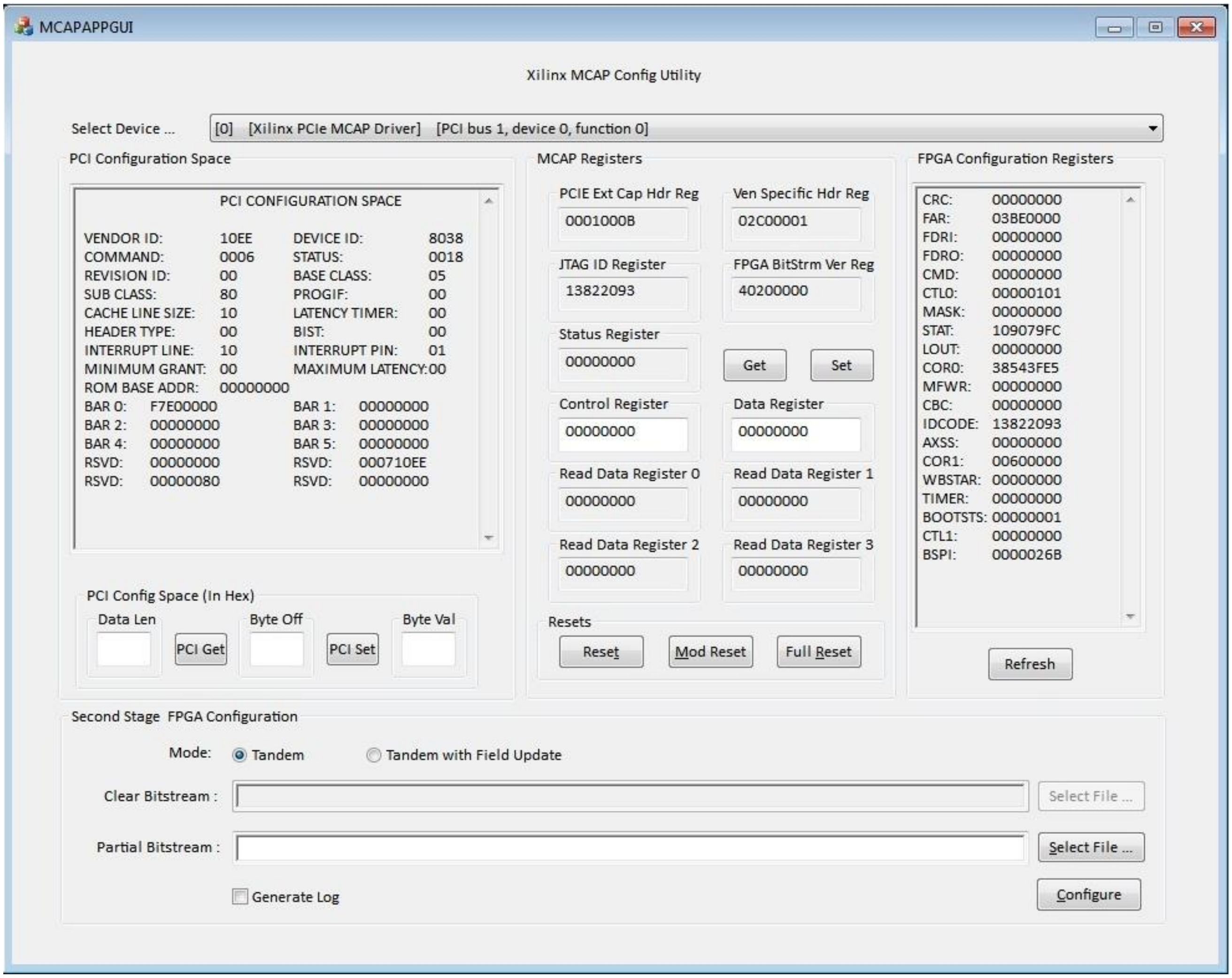
[0] [Xilinx PCIe MCAP Driver] [PCI bus 1, device 0, function 0]

Mode: ● Tandem ○ Tandem with Field Update

□ Generate Log

Figure 11: Selecting a device

Once the device is selected, the GUI is populated with the PCI Configuration Space on the left pane and the MCAP Registers in the middle pane. The FPGA Configuration Registers are populated when the “Refresh” button is clicked.



Xilinx MCAP Config Utility

Select Device ... [0] [Xilinx PCIe MCAP Driver] [PCI bus 1, device 0, function 0]

PCI Configuration Space

```
                      PCI CONFIGURATION SPACE

VENDOR ID:          10EE        DEVICE ID:          8038
COMMAND:            0006        STATUS:             0018
REVISION ID:        00          BASE CLASS:         05
SUB CLASS:          80          PROGIF:             00
CACHE LINE SIZE:    10          LATENCY TIMER:      00
HEADER TYPE:        00          BIST:               00
INTERRUPT LINE:     10          INTERRUPT PIN:      01
MINIMUM GRANT:      00          MAXIMUM LATENCY:00
ROM BASE ADDR:      00000000
BAR 0:    F7E00000              BAR 1:    00000000
BAR 2:    00000000              BAR 3:    00000000
BAR 4:    00000000              BAR 5:    00000000
RSVD:     00000000              RSVD:     000710EE
RSVD:     00000080              RSVD:     00000000
```

PCI Config Space (In Hex)

Data Len:

Byte Off:

Byte Val:

MCAP Registers

PCIE Ext Cap Hdr Reg: 0001000B

Ven Specific Hdr Reg: 02C00001

JTAG ID Register: 13822093

FPGA BitStrm Ver Reg: 40200000

Status Register: 00000000

Control Register: 00000000

Data Register: 00000000

Read Data Register 0: 00000000

Read Data Register 1: 00000000

Read Data Register 2: 00000000

Read Data Register 3: 00000000

FPGA Configuration Registers

```
CRC:      00000000
FAR:      03BE0000
FDRI:     00000000
FDRO:     00000000
CMD:      00000000
CTL0:     00000101
MASK:     00000000
STAT:     109079FC
LOUT:     00000000
COR0:     38543FE5
MFWR:     00000000
CBC:      00000000
IDCODE:   13822093
AXSS:     00000000
COR1:     00600000
WBSTAR:   00000000
TIMER:    00000000
BOOTSTS: 00000001
CTL1:     00000000
BSPI:     0000026B
```

Second Stage FPGA Configuration

Mode: ● Tandem ○ Tandem with Field Update

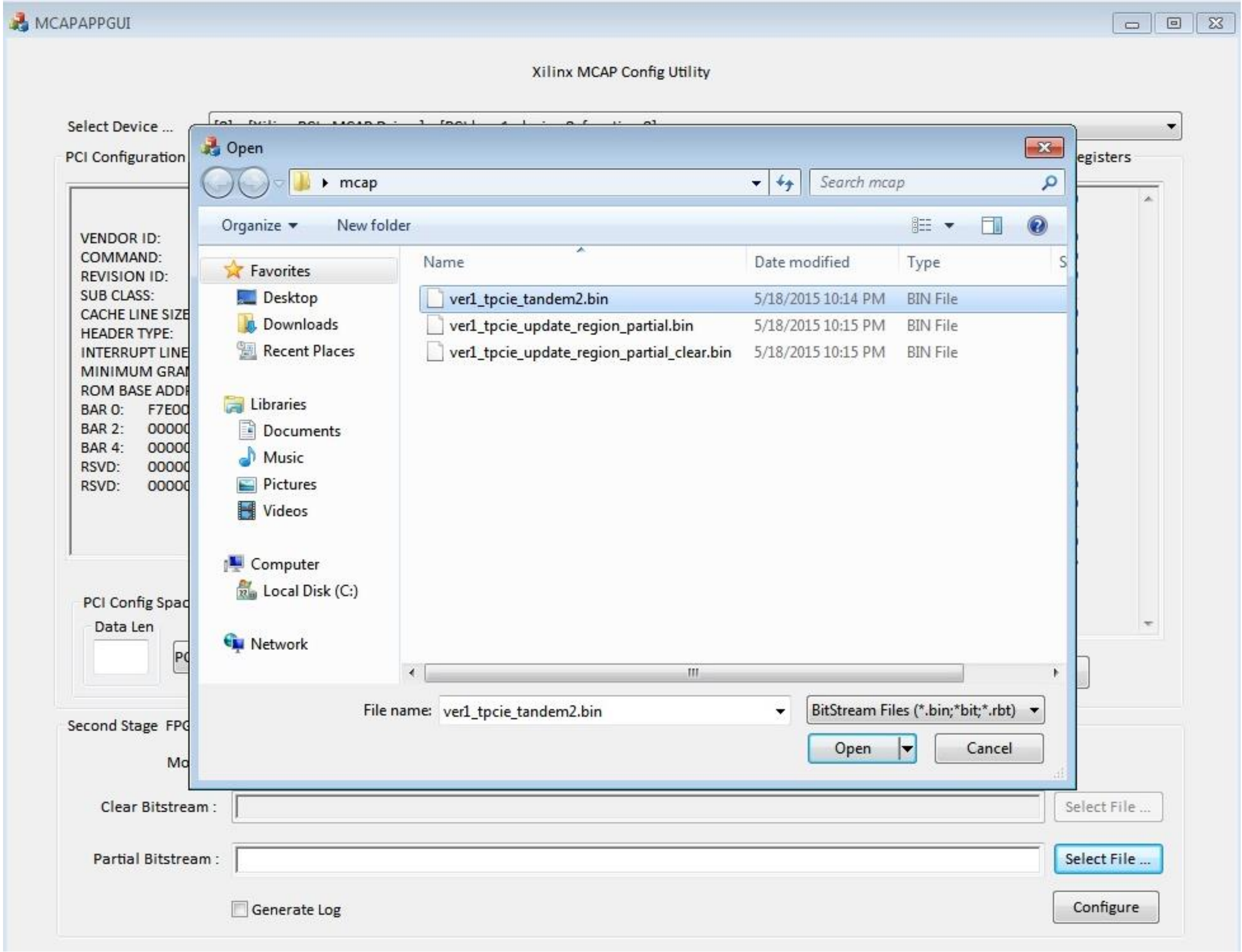
Clear Bitstream :

Partial Bitstream :

□ Generate Log

Figure 12: Information from selected device populated in GUI

To program a bit file, click “Select File.” Browse and select a file to program. Programming files with the .bin extension should be used for programming through the MCAP extended capability as these files will be 32-bit aligned and can be written directly into the MCAP registers.



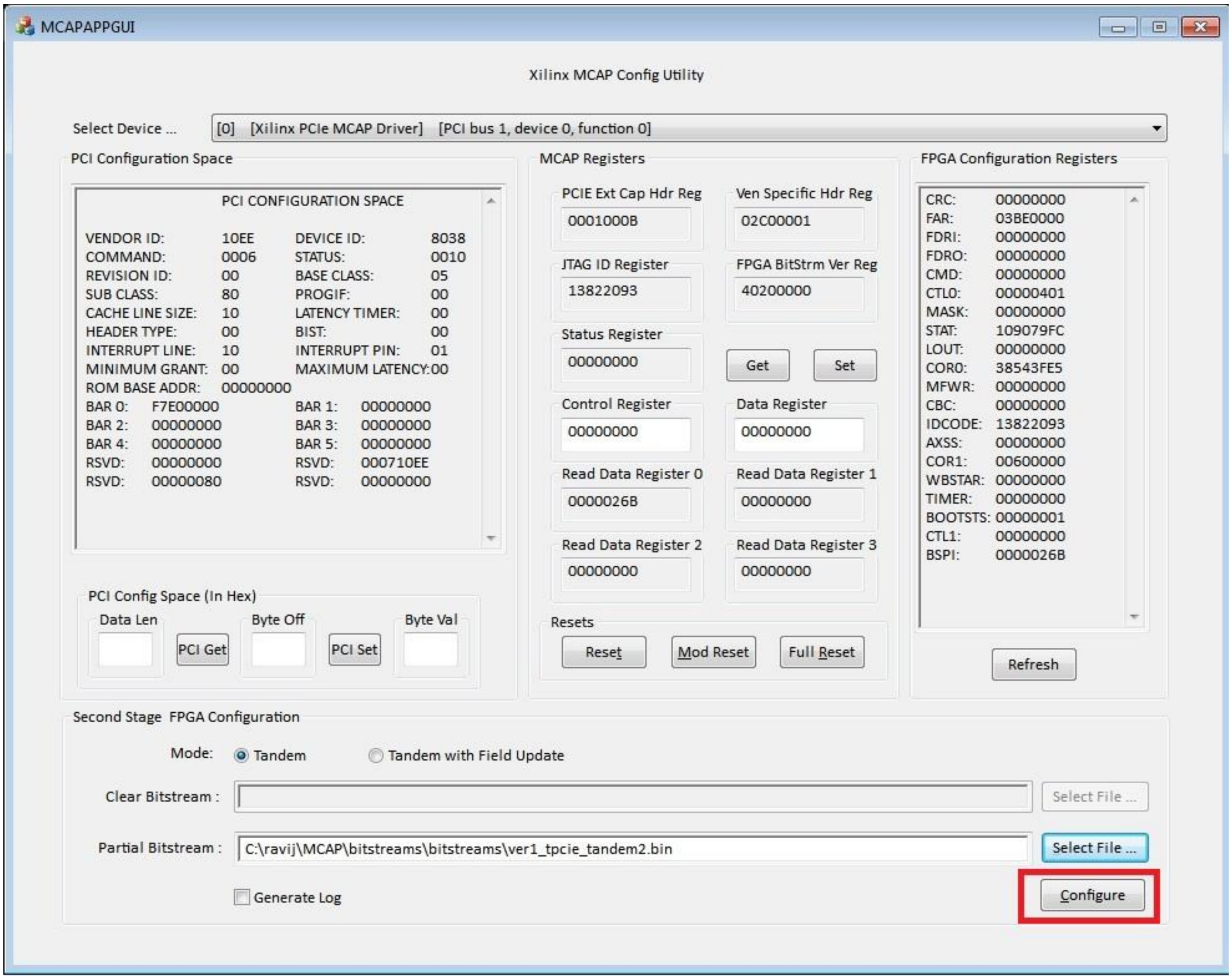
Name	Date modified	Type
ver1_tpcie_tandem2.bin	5/18/2015 10:14 PM	BIN File
ver1_tpcie_update_region_partial.bin	5/18/2015 10:15 PM	BIN File
ver1_tpcie_update_region_partial_clear.bin	5/18/2015 10:15 PM	BIN File

File name: ver1_tpcie_tandem2.bin

BitStream Files (*.bin;*bit;*.rbt)

Figure 13: Selecting a programming file

Click the “Configure” button to start programming.



MCAPAPPGUI

Xilinx MCAP Config Utility

Select Device ... [0] [Xilinx PCIe MCAP Driver] [PCI bus 1, device 0, function 0]

PCI Configuration Space

```
                    PCI CONFIGURATION SPACE

VENDOR ID:        10EE      DEVICE ID:          8038
COMMAND:          0006      STATUS:             0010
REVISION ID:      00        BASE CLASS:         05
SUB CLASS:        80        PROGIF:             00
CACHE LINE SIZE:  10        LATENCY TIMER:      00
HEADER TYPE:      00        BIST:               00
INTERRUPT LINE:   10        INTERRUPT PIN:      01
MINIMUM GRANT:    00        MAXIMUM LATENCY:00
ROM BASE ADDR:    00000000
BAR 0:     F7E00000         BAR 1:     00000000
BAR 2:     00000000         BAR 3:     00000000
BAR 4:     00000000         BAR 5:     00000000
RSVD:      00000000         RSVD:      000710EE
RSVD:      00000080         RSVD:      00000000
```

PCI Config Space (In Hex)

Data Len: [ ] PCI Get Byte Off: [ ] PCI Set Byte Val: [ ]

MCAP Registers

- PCIE Ext Cap Hdr Reg: 0001000B
- Ven Specific Hdr Reg: 02C00001
- JTAG ID Register: 13822093
- FPGA BitStrm Ver Reg: 40200000
- Status Register: 00000000
- Control Register: 00000000
- Data Register: 00000000
- Read Data Register 0: 0000026B
- Read Data Register 1: 00000000
- Read Data Register 2: 00000000
- Read Data Register 3: 00000000

Get Set

Resets: Reset Mod Reset Full Reset

FPGA Configuration Registers

```
CRC:      00000000
FAR:      03BE0000
FDRI:     00000000
FDRO:     00000000
CMD:      00000000
CTL0:     00000401
MASK:     00000000
STAT:     109079FC
LOUT:     00000000
COR0:     38543FE5
MFWR:     00000000
CBC:      00000000
IDCODE:   13822093
AXSS:     00000000
COR1:     00600000
WBSTAR:   00000000
TIMER:    00000000
BOOTSTS:  00000001
CTL1:     00000000
BSPI:     0000026B
```

Refresh

Second Stage FPGA Configuration

Mode: ● Tandem ○ Tandem with Field Update

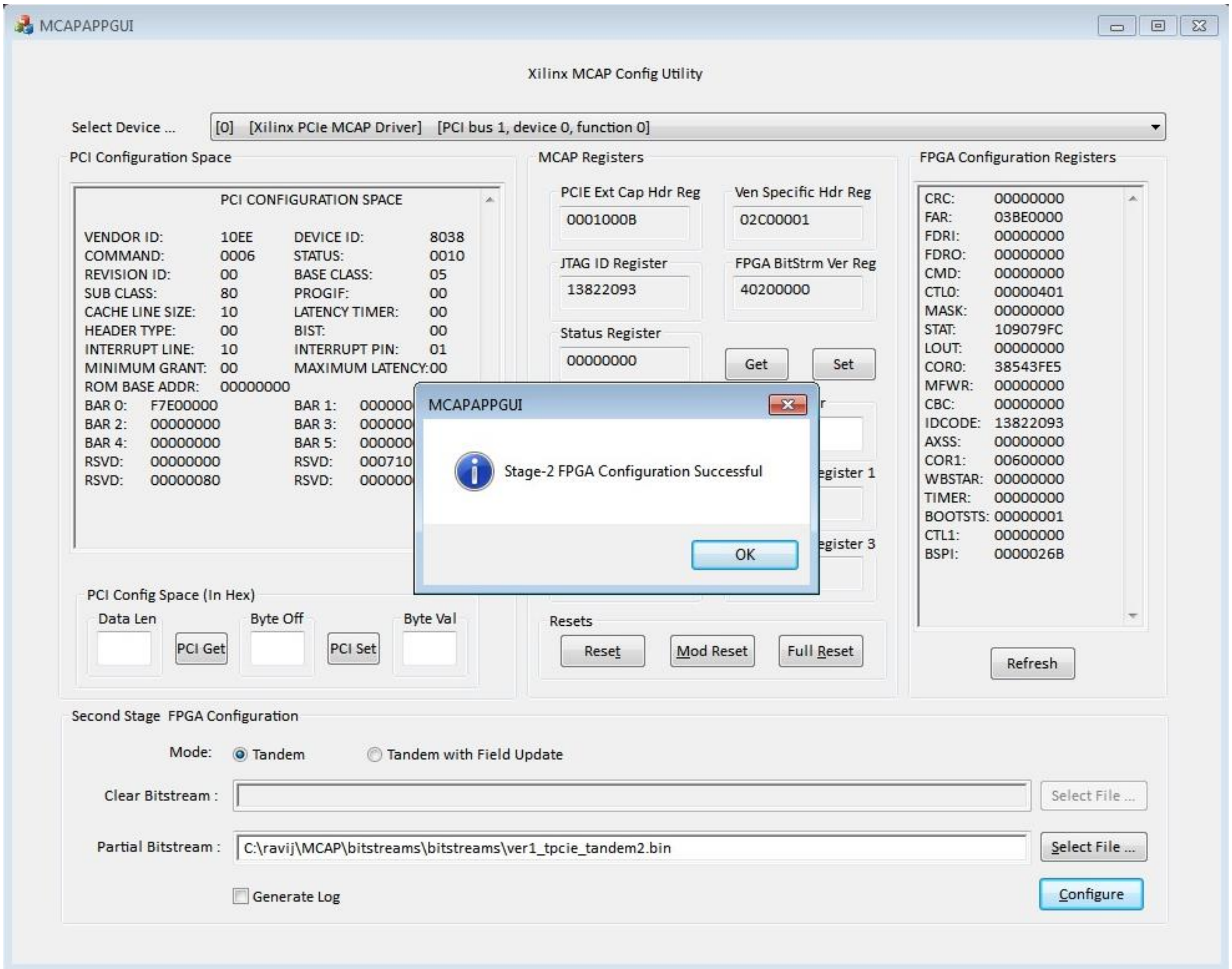
Clear Bitstream : [ ] Select File ...

Partial Bitstream : C:\ravij\MCAP\bitstreams\bitstreams\ver1_tpcie_tandem2.bin Select File ...

□ Generate Log Configure

Figure 14: Deliver the bitstream to the target device

Once bitstream programming has completed, the message below is displayed.



Select Device ... [0] [Xilinx PCIe MCAP Driver] [PCI bus 1, device 0, function 0]

```
                 PCI CONFIGURATION SPACE

VENDOR ID:       10EE        DEVICE ID:       8038
COMMAND:         0006        STATUS:          0010
REVISION ID:     00          BASE CLASS:      05
SUB CLASS:       80          PROGIF:          00
CACHE LINE SIZE: 10          LATENCY TIMER:   00
HEADER TYPE:     00          BIST:            00
INTERRUPT LINE:  10          INTERRUPT PIN:   01
MINIMUM GRANT:   00          MAXIMUM LATENCY:00
ROM BASE ADDR:   00000000
BAR 0:     F7E00000          BAR 1:     000000
BAR 2:     00000000          BAR 3:     000000
BAR 4:     00000000          BAR 5:     000000
RSVD:      00000000          RSVD:      000710
RSVD:      00000080          RSVD:      000000
```

PCIE Ext Cap Hdr Reg	0001000B
Ven Specific Hdr Reg	02C00001
JTAG ID Register	13822093
FPGA BitStrm Ver Reg	40200000
Status Register	00000000

```
CRC:     00000000
FAR:     03BE0000
FDRI:    00000000
FDRO:    00000000
CMD:     00000000
CTL0:    00000401
MASK:    00000000
STAT:    109079FC
LOUT:    00000000
COR0:    38543FE5
MFWR:    00000000
CBC:     00000000
IDCODE:  13822093
AXSS:    00000000
COR1:    00600000
WBSTAR:  00000000
TIMER:   00000000
BOOTSTS: 00000001
CTL1:    00000000
BSPI:    0000026B
```

Stage-2 FPGA Configuration Successful

Mode: ● Tandem ○ Tandem with Field Update

Clear Bitstream :

Partial Bitstream : C:\ravij\MCAP\bitstreams\bitstreams\ver1_tpcie_tandem2.bin

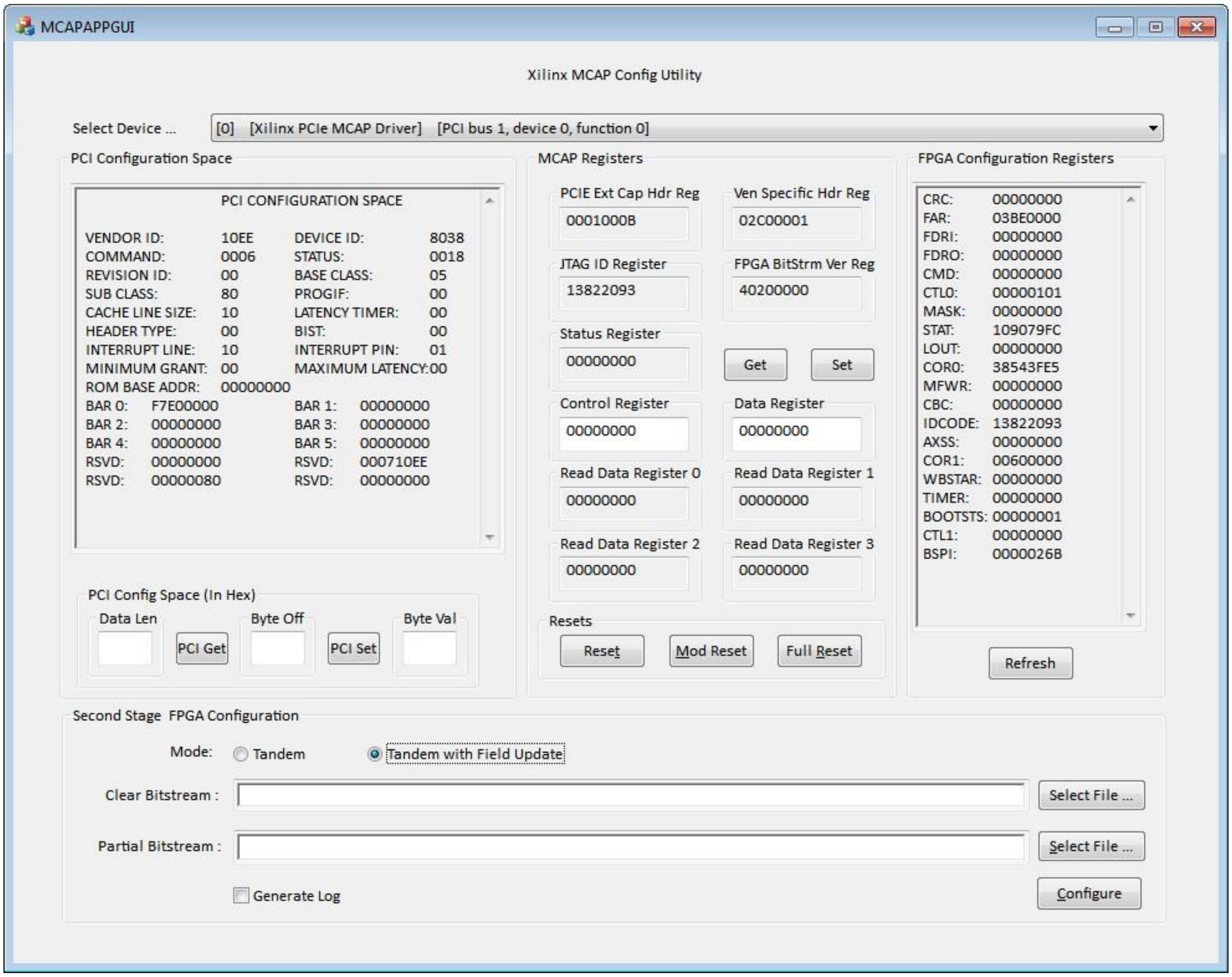
□ Generate Log

Figure 15: Configuration complete

NOTE: The windows driver is tested only on Win7 64-bit platform.

Tandem Field Update

To do a Field Update Select the “Tandem with Field Update” option at the bottom of the screen



MCAPAPPGUI

Xilinx MCAP Config Utility

Select Device ... [0] [Xilinx PCIe MCAP Driver] [PCI bus 1, device 0, function 0]

PCI Configuration Space

```
                    PCI CONFIGURATION SPACE

VENDOR ID:        10EE        DEVICE ID:          8038
COMMAND:          0006        STATUS:             0018
REVISION ID:      00          BASE CLASS:         05
SUB CLASS:        80          PROGIF:             00
CACHE LINE SIZE:  10          LATENCY TIMER:      00
HEADER TYPE:      00          BIST:               00
INTERRUPT LINE:   10          INTERRUPT PIN:      01
MINIMUM GRANT:    00          MAXIMUM LATENCY:00
ROM BASE ADDR:    00000000
BAR 0:    F7E00000            BAR 1:    00000000
BAR 2:    00000000            BAR 3:    00000000
BAR 4:    00000000            BAR 5:    00000000
RSVD:     00000000            RSVD:     000710EE
RSVD:     00000080            RSVD:     00000000
```

PCI Config Space (In Hex)

Data Len: | PCI Get | Byte Off: | PCI Set | Byte Val:

MCAP Registers

PCIE Ext Cap Hdr Reg: 0001000B	Ven Specific Hdr Reg: 02C00001
JTAG ID Register: 13822093	FPGA BitStrm Ver Reg: 40200000
Status Register: 00000000	Get / Set
Control Register: 00000000	Data Register: 00000000
Read Data Register 0: 00000000	Read Data Register 1: 00000000
Read Data Register 2: 00000000	Read Data Register 3: 00000000

Resets: Reset | Mod Reset | Full Reset

FPGA Configuration Registers

```
CRC:      00000000
FAR:      03BE0000
FDRI:     00000000
FDRO:     00000000
CMD:      00000000
CTL0:     00000101
MASK:     00000000
STAT:     109079FC
LOUT:     00000000
COR0:     38543FE5
MFWR:     00000000
CBC:      00000000
IDCODE:   13822093
AXSS:     00000000
COR1:     00600000
WBSTAR:   00000000
TIMER:    00000000
BOOTSTS:  00000001
CTL1:     00000000
BSPI:     0000026B
```

Refresh

Second Stage FPGA Configuration

Mode: ○ Tandem ● Tandem with Field Update

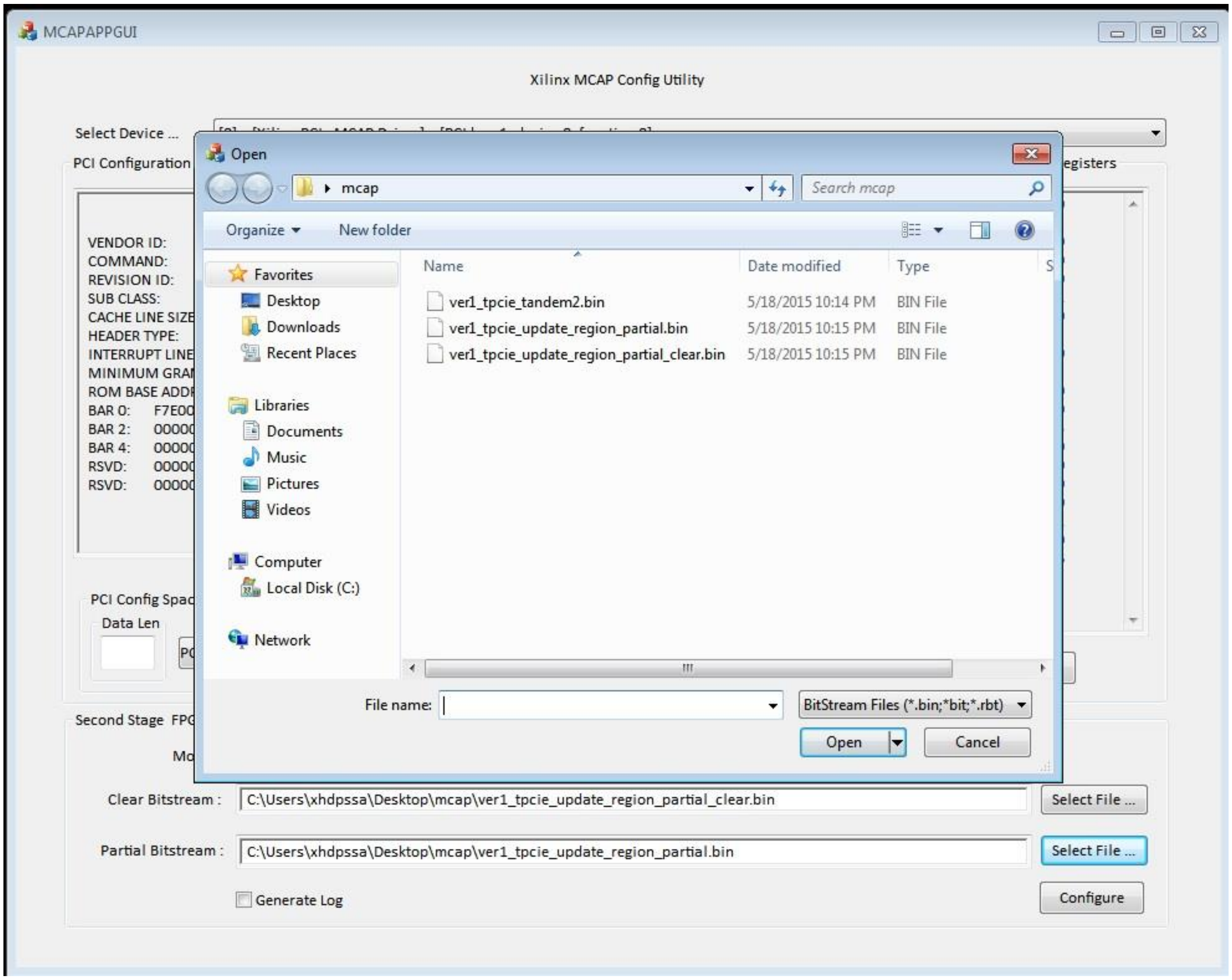
Clear Bitstream : | Select File ...

Partial Bitstream : | Select File ...

□ Generate Log | Configure

Figure 16: Tandem with Field Update

To program Clear Bitstream and Partial Bitstream, click "Select File" for both of the instances to browse and select the relevant files. Programming files with the .bin extension should be used for programming through the MCAP extended capability as these files will be 32-bit aligned and can be written directly into the MCAP registers.



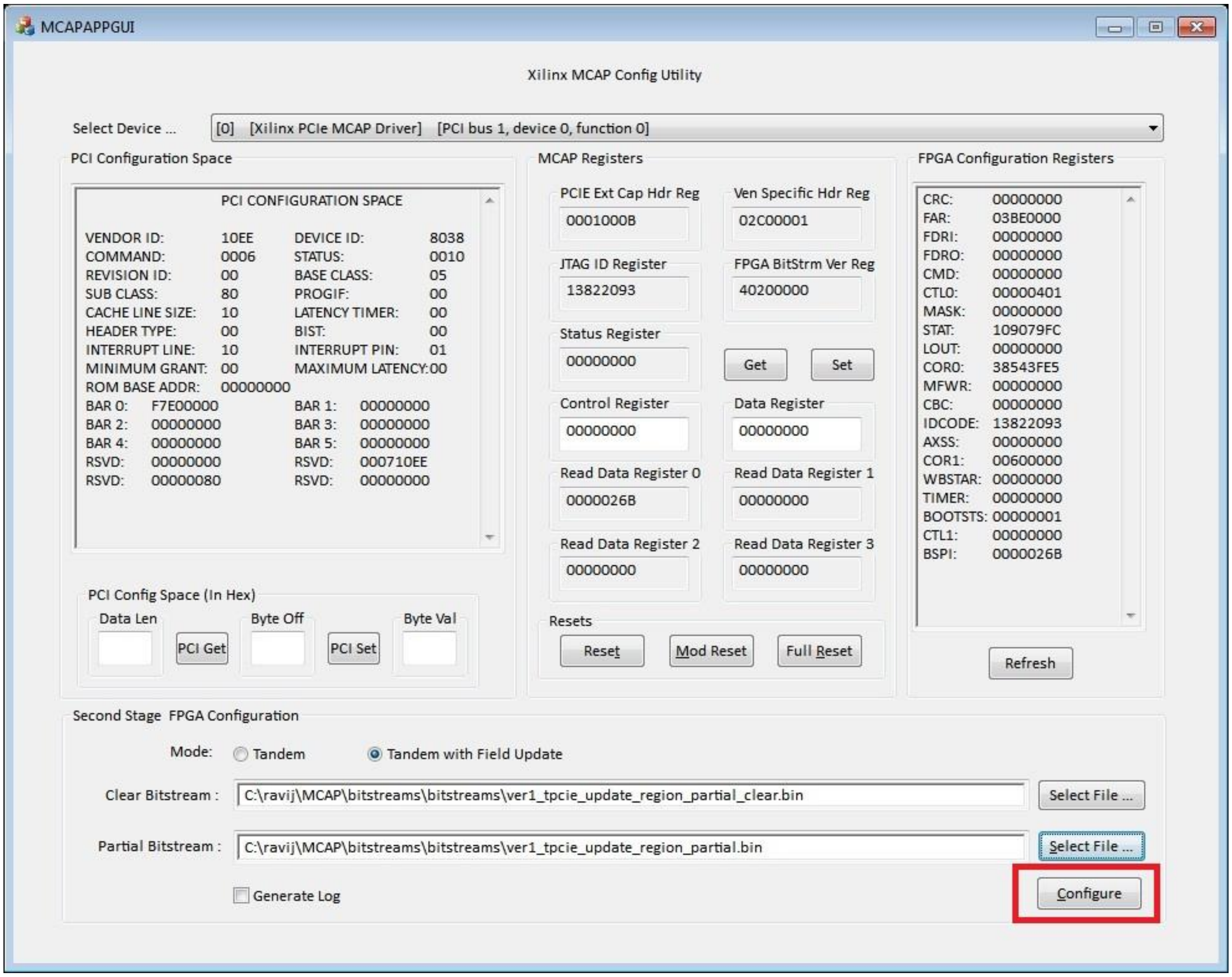
Name	Date modified	Type
ver1_tpcie_tandem2.bin	5/18/2015 10:14 PM	BIN File
ver1_tpcie_update_region_partial.bin	5/18/2015 10:15 PM	BIN File
ver1_tpcie_update_region_partial_clear.bin	5/18/2015 10:15 PM	BIN File

Clear Bitstream : C:\Users\xhdpssa\Desktop\mcap\ver1_tpcie_update_region_partial_clear.bin

Partial Bitstream : C:\Users\xhdpssa\Desktop\mcap\ver1_tpcie_update_region_partial.bin

Figure 17: File Selection for Tandem Field Update

Click the “Configure” button to perform a Field Update.



Select Device ... [0] [Xilinx PCIe MCAP Driver] [PCI bus 1, device 0, function 0]

```
                    PCI CONFIGURATION SPACE

VENDOR ID:          10EE       DEVICE ID:          8038
COMMAND:            0006       STATUS:             0010
REVISION ID:        00         BASE CLASS:         05
SUB CLASS:          80         PROGIF:             00
CACHE LINE SIZE:    10         LATENCY TIMER:      00
HEADER TYPE:        00         BIST:               00
INTERRUPT LINE:     10         INTERRUPT PIN:      01
MINIMUM GRANT:      00         MAXIMUM LATENCY:00
ROM BASE ADDR:      00000000
BAR 0:    F7E00000             BAR 1:    00000000
BAR 2:    00000000             BAR 3:    00000000
BAR 4:    00000000             BAR 5:    00000000
RSVD:     00000000             RSVD:     000710EE
RSVD:     00000080             RSVD:     00000000
```

Data Len: Byte Off: Byte Val:

PCIE Ext Cap Hdr Reg: 0001000B
Ven Specific Hdr Reg: 02C00001
JTAG ID Register: 13822093
FPGA BitStrm Ver Reg: 40200000
Status Register: 00000000
Control Register: 00000000
Data Register: 00000000
Read Data Register 0: 0000026B
Read Data Register 1: 00000000
Read Data Register 2: 00000000
Read Data Register 3: 00000000

```
CRC:      00000000
FAR:      03BE0000
FDRI:     00000000
FDRO:     00000000
CMD:      00000000
CTL0:     00000401
MASK:     00000000
STAT:     109079FC
LOUT:     00000000
COR0:     38543FE5
MFWR:     00000000
CBC:      00000000
IDCODE:   13822093
AXSS:     00000000
COR1:     00600000
WBSTAR:   00000000
TIMER:    00000000
BOOTSTS:  00000001
CTL1:     00000000
BSPI:     0000026B
```

Second Stage FPGA Configuration

Mode: ○ Tandem ● Tandem with Field Update

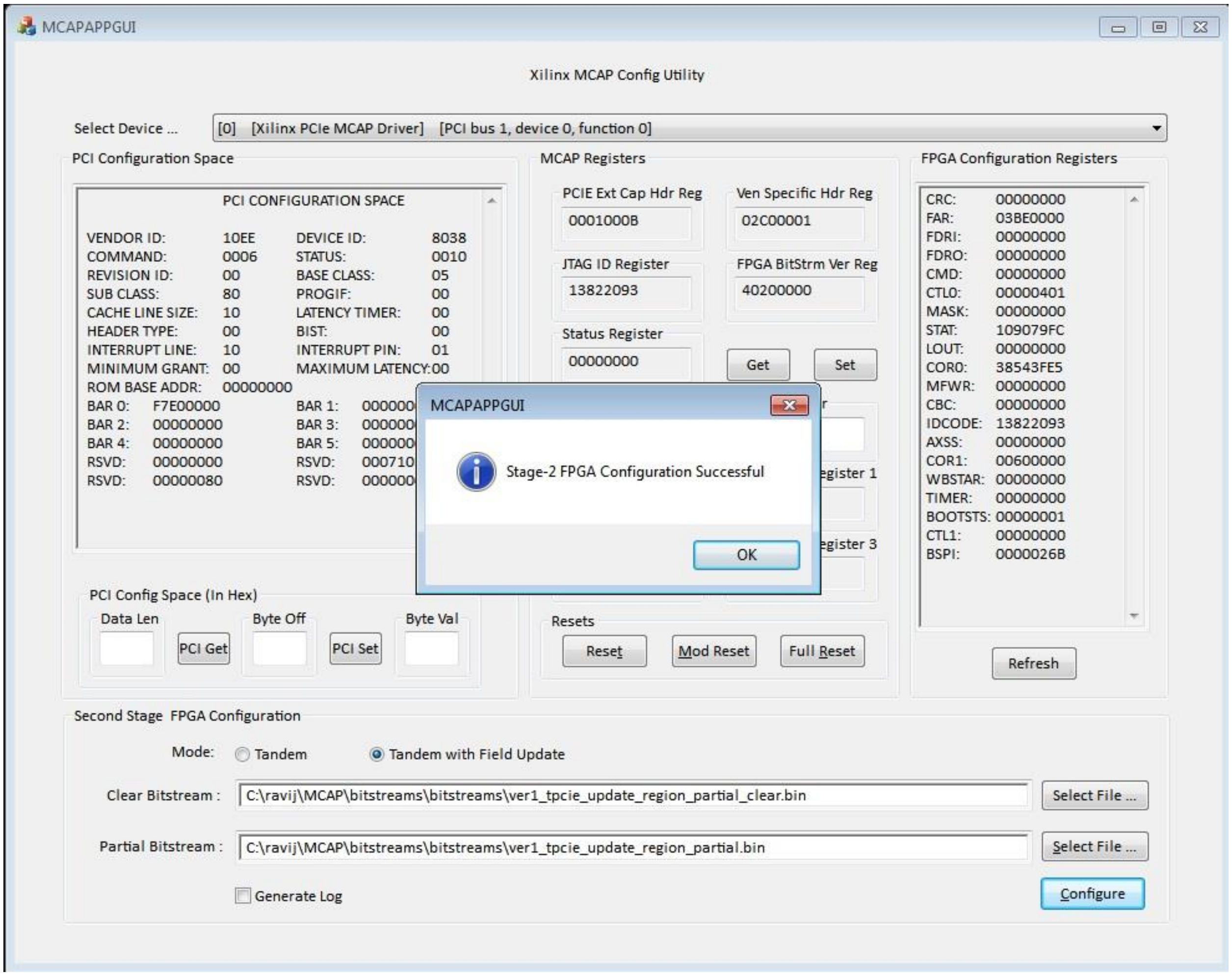
Clear Bitstream : C:\ravij\MCAP\bitstreams\bitstreams\ver1_tpcie_update_region_partial_clear.bin

Partial Bitstream : C:\ravij\MCAP\bitstreams\bitstreams\ver1_tpcie_update_region_partial.bin

□ Generate Log

Figure 18: Selecting Clear Bitstream and Partial Bitstream for a Target Device.

Once the bitstream programming is complete, the following message will be displayed.



Xilinx MCAP Config Utility

Select Device ... [0] [Xilinx PCIe MCAP Driver] [PCI bus 1, device 0, function 0]

PCI Configuration Space

```
                PCI CONFIGURATION SPACE

VENDOR ID:        10EE        DEVICE ID:          8038
COMMAND:          0006        STATUS:             0010
REVISION ID:      00          BASE CLASS:         05
SUB CLASS:        80          PROGIF:             00
CACHE LINE SIZE:  10          LATENCY TIMER:      00
HEADER TYPE:      00          BIST:               00
INTERRUPT LINE:   10          INTERRUPT PIN:      01
MINIMUM GRANT:    00          MAXIMUM LATENCY:00
ROM BASE ADDR:    00000000
BAR 0:    F7E00000            BAR 1:    000000
BAR 2:    00000000            BAR 3:    000000
BAR 4:    00000000            BAR 5:    000000
RSVD:     00000000            RSVD:     000710
RSVD:     00000080            RSVD:     000000
```

PCI Config Space (In Hex): Data Len, Byte Off, Byte Val

MCAP Registers

PCIE Ext Cap Hdr Reg	0001000B
Ven Specific Hdr Reg	02C00001
JTAG ID Register	13822093
FPGA BitStrm Ver Reg	40200000
Status Register	00000000

MCAPAPPGUI: Stage-2 FPGA Configuration Successful

FPGA Configuration Registers

```
CRC:      00000000
FAR:      03BE0000
FDRI:     00000000
FDRO:     00000000
CMD:      00000000
CTL0:     00000401
MASK:     00000000
STAT:     109079FC
LOUT:     00000000
COR0:     38543FE5
MFWR:     00000000
CBC:      00000000
IDCODE:   13822093
AXSS:     00000000
COR1:     00600000
WBSTAR:   00000000
TIMER:    00000000
BOOTSTS:  00000001
CTL1:     00000000
BSPI:     0000026B
```

Second Stage FPGA Configuration

Mode: ○ Tandem ● Tandem with Field Update

Clear Bitstream : C:\ravij\MCAP\bitstreams\bitstreams\ver1_tpcie_update_region_partial_clear.bin

Partial Bitstream : C:\ravij\MCAP\bitstreams\bitstreams\ver1_tpcie_update_region_partial.bin

□ Generate Log

Figure 19: Successful Completion of Tandem Field Update

Building Windows MCAP Software

Driver

1. Install Windows WDK 7600 or above (http://www.microsoft.com/en-us/download/details.aspx?id=11800) to build Drivers.

2. Open the Build Environment.

   - Start->All Programs->Windows Driver Kits->"DDK_VER"->Build Environments->"Select an Appropriate Environment"
   - E.g.
     Start->All Programs->Windows Driver Kits->WDK 7600.16385.1->Build Environments->"Select an Appropriate Environment"
   - x86 Free Build Environment - For 32 Bit Drivers
   - x64 Free Build Environment - For 64 Bit Driver (AMD64 Architecture)

3. Navigate to the Windows Mcap Driver software location from the command line utility.

4. Execute the command bld.bat with Appropriate Arguments to build the drivers.

- bld.bat x86 — x86 Free Build Environment - For 32 Bit Drivers
- bld.bat x64 — x64 Free Build Environment - For 64 Bit Driver (AMD64 Architecture)

Application

1. Browse to Applications Source Code Folder **MCAPAPP** or **MCAPAPPGUI**
2. Select and Open **.sln** file. (This Needs Visual Studio 2008 or Above)
3. In the Menu Bar select **Build->Configuration Manager**.
4. In the Configuration Manager select Active Configuration as Debug or Release.
5. In the Menu Bar select **Build->Build Solution** to build the Application

Note: To build MCAPGUI, Microsoft Foundation Classes must be installed on the host machine.

PCIe MCAP Driver for Linux

The MCAP Linux User Space Library is developed on top of the PCI Utilities library. This library forms the core of the MCAP interface. It has initialization routines and provides APIs for accessing MCAP registers and downloading bitstreams. It provides several other APIs to interact with the underlying hardware.

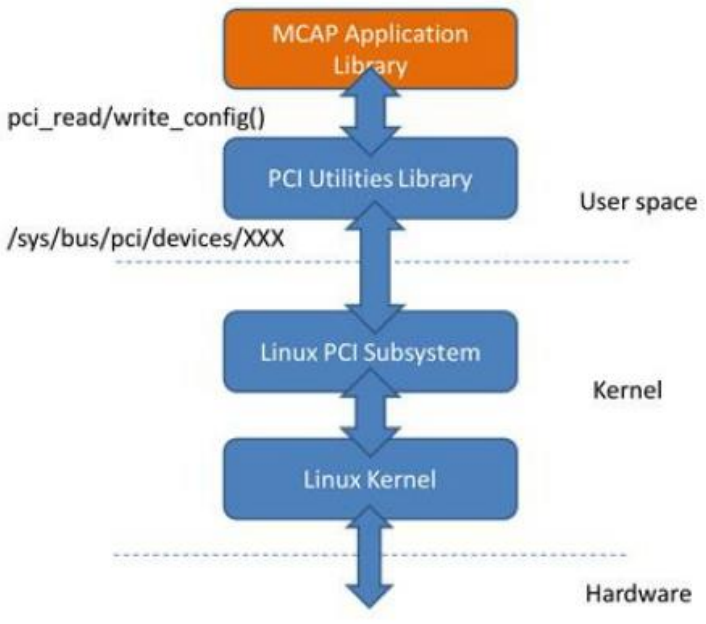


Figure 16: PCIe MCAP Driver Architecture

MCAP User Level Driver/Library

A user-level library obtains access to the target device by scanning the PCI bus using PCI Utilities library with the specific Vendor and Device IDs. This layer provides all of the necessary APIs for the user to interact with underlying hardware.

Major Design Blocks/Functionalities

Core APIs

The following APIs implement the core functionality:

- ***MCapLibInit***

  This function initializes the MCAP library by scanning the PCI bus for the MCAP device with filters of Vendor and Device IDs. When a device is found, it scans the Extended Capability Register Set to get the Register Base Offset of the MCAP Extended Capability. This function internally initializes underlying PCI Utilities library.

- ***MCapLibFree***

  This function frees all the structures allocated in *MCapLibInit().*

- ***MCapReset***

  This function performs an MCAP reset by asserting and deasserting the MCAP Reset bit in the MCAP control register.

- ***MCapModuleReset***

  This function performs an MCAP module reset by asserting and deasserting the MCAP Module Reset bit in the MCAP control register.

- ***MCapFullReset***

  This function performs both the resets mentioned above by asserting and deasserting the corresponding bits in the MCAP control register.

- ***MCapConfigureFPGA***

  This function parses the input bitstream for '.rbt' or '.bit' or '.bin' and then processes the data in the file to convert it into a format the HW expects. Then this function calls *MCapWriteBitStream()* to actually program the data into the HW.

- ***MCapWriteBitStream***

  This function will program the data processed by *MCapConfigureFPGA()* into the hardware in multiple of dwords.

- ***MCapDumpReadRegs***

  This function reads data which resides in the read data registers of the MCAP register space.

- ***MCapDumpRegs***

  This function reads and dumps all the registers of the MCAP register space.

- ***MCapAccessConfigSpace***

  This API provides the capability of reading/writing from/to the PCI Express configuration registers of MCAP device at a specified offset.

Miscellaneous Routines

- ***MCapShowDevice***

This API displays the verbose information of the MCAP device.

- ***MCapDoBusWalk***

  This API performs a bus walk to find the Register Base of the MCAP device by parsing the Extended Capability Pointers of the PCIe Configuration space.

- ***MCapProcessRBT***

  This API processes the .rbt file and organizes the data into an array such that it can be written to the hardware.

- ***MCapProcessBIN***

  This API processes the .bin file and organizes the data into an array such that it can be written to the hardware.

- ***MCapProcessBIT***

  This API processes the .bit file and organizes the data into an array such that it can be written to the hardware.

MCAP Registers

The MCAP Registers are described in the MCAP Extended Capability Register Description section of this document.

Source Files

mcap_lib.c

It implements all the functionalities of the driver.

Header Files

It has only one header file.

mcap_lib.h

It contains hash defines for the MCAP Registers and masks for accessing various bits in these registers.

Compiling the Driver

- Download the latest PCI Utilities source from https://www.kernel.org/pub/software/utils/pciutils/

  (or) Source can be cloned from the GIT repository http://git.kernel.org/cgit/utils/pciutils/pciutils.git/

- Compile the PCI Utilities by using the following command
  $pci-utils> make

- Compile the MCAP Library by the following make command
  $mcap-lib> make PCIUTILS_PATH=<PATH to PCI Utilities Source>

Note: To compile the source on machines with Fedora, zlib library (-lz) is required, hence we include the library when compiling. On Machines with Ubuntu installed, this library may not be installed by default. So, please install the library by,

Ubuntu> sudo apt-get install zlib1g-dev

- Compiling the MCAP library generates 'libmcap.a' and example elf 'mcap' built on top of the generated library.

Running the Application

Running the 'mcap.elf' with '-h' lists all the options that are available to communicate with the underlying MCAP device,

```
$Linux> ./mcap –h

Usage: mcap [options]

Options:
    -x                  Specify MCAP Device ID in hex (Mandatory)
    -p   <file>         Program Bitstream (.bin/.bit/.rbt)
    -C   <file>         Partial Reconfiguration Clear File (.bin/.bit/.rbt)
    -r                  Performs Simple Reset
    -m                  Performs Module Reset
    -f                  Performs Full Reset
    -D                  Read Data Registers
    -d                  Dump all the MCAP Registers
    -v                  verbose information of MCAP Device
    -h/H                Help
    -a <address> [type [data]] Access Device Configuration Space
                        Here type [data] - b for byte data [8 bits]
                        Here type [data] - h for half word data [16 bits]
                        Here type [data] - w for word data [32 bits]
```

NOTE*: Traditionally the Vendor ID and Device ID for the hardware and associated drivers and software will be coded to a single value. This software has been designed to use a Vendor ID of 10EE (Xilinx) and a Device ID that is passed in from the command line. Specifying MCAP Device ID option is mandatory for the application to run. For example,*

```
$Linux> ./mcap -x 0x8011
Xilinx MCAP Device Found
```

Note:

- PCI Extended Capability Registers in Linux will only be accessible with privileged user access. So, the example “elf” should be run with ROOT permissions.

- To access device configuration space, 'type' in the above syntax should be either b/h/w (byte/half-word/word). For example,

    ➔ Reading a byte
    - ./mcap -x 0x8011 -a 0x354 b

    ➔ Writing a word
    - ./mcap -x 0x8011 -a 0x354 w 0x3

Using the MCAP Extended capability

This section describes the MCAP extended capability register space and how the Host software should use the MCAP registers to perform MCAP arbitration, bitstream write, MCAP reset, and configuration register read operations.

Identifying the PCIe MCAP Extended Capability

The PCI Express specification provides a number of ID fields within the PCI Express Header space to identify a specific card, card revision, and capabilities available. Host drivers and software should validate each of these fields to ensure compatibility between the card and Host drivers/software. To aid with interoperability and lab testing, the drivers provided with this application note do not validate against all of the PCI Express ID fields. Once appropriate ID values have been selected for your application they should be fixed to specific values in the FPGA design and Host drivers and software should validate against them before accessing any PCIe devices. This will ensure that your driver and software are compatible with your specific hardware implementation and do not conflict with other PCIe devices.

Table 1 describes the PCIe ID fields that must be validated by Host drivers and software to properly discover a specific PCIe device.

Table 1: PCIe ID Fields

Name	Description	Value Width	Default Value
PCIe Vendor ID	PCIe specification defined field that should contain a PCI-SIG allocated value specific to each vendor. This value is traditionally used to identify the PCIe device manufacturer and can be modified by the user via the PCIe IP customization GUI	16-bit	0x10EE (Xilinx)
PCIe Device ID	PCIe specification defined field that contains a vendor defined value. This value is traditionally used to identify the PCIe device and can be modified by the user via the PCIe IP customization GUI. This field should be qualified by the PCIe Vendor ID prior to interpretation	16-bit	Device Dependent
PCIe Revision ID	PCIe specification defined field that contains a vendor defined value. This value is traditionally used to identify the PCIe function revision and can be modified by the user via the PCIe IP customization GUI. This field should be qualified by all previously listed ID fields prior to interpretation	8-bit	0x00
Subsystem Vendor ID	PCIe specification defined field that should contain a PCI-SIG allocated value for each subsystem vendor. This value is traditionally used to identify the vendor of a specific system or card and can be modified by the user via the PCIe IP customization GUI. This field should be qualified by all previously listed ID fields prior to interpretation.	16-bit	0x10EE (Xilinx)
Subsystem ID	PCIe specification defined field that contains a subsystem vendor defined value. This value is traditionally used to identify the specific system or card and can be modified by the user via the PCIe IP customization GUI. This field should be qualified by all previously listed ID fields prior to interpretation.	16-bit	0x0007

Once the PCIe device has been properly identified, Host drivers and software should follow the PCIe Next Capability pointer to discover additional capabilities available for the PCI Express End Point device. Each extended capability header will contain a next capability pointer to form a linked list enumerating all of the available extended capabilities. A value of 0x000 identifies the end of the extended capabilities list. To properly identify the MCAP Extended Capability, Host drivers and software should cycle through the extended capabilities until the appropriate extended capability ID fields have been identified. For UltraScale devices the base address for the MCAP VSEC will be at byte offset 0x340 within the PCIe configuration register space.

Table 2 provides a description of the ID fields that should be verified in order to identify the MCAP Extended Capability.

Table 2: Vendor Specific Extended Capability ID Fields

Name	Description	Value Width	Default Value
PCI Express Extended Capability ID	PCIe specification defined field that contains the PCIe specification defined value of 0x000B for VSEC capabilities. This field should not be modified.	16-bit	0x000B
PCI Express Extended Capability Version	PCIe specification defined field that contains the PCIe specification defined value of 0x1 for VSEC capabilities. This field should not be modified.	4-bit	0x1
MCAP VSEC ID	PCIe specification defined field that contains a value used to identify the MCAP Extended Capability. This value can be modified by changing the MCAP_VSEC_ID property on the PCIe hardblock instance. This field should be qualified by all previously listed ID fields prior to interpretation	16-bit	0x0001
MCAP VSEC Rev ID	PCIe specification defined field that contains a value used to identify the MCAP Extended Capability Revision. This value can be modified by changing the MCAP_VSEC_REV property on the PCIe hardblock instance. This field should be qualified by all previously listed ID fields prior to interpretation.	4-bit	0x0

Once the PCIe device and the MCAP extended capability have been identified, the MCAP Bitstream Version register within the MCAP Extended Capability register space can be used to identify the version of the Tandem Stage1 or PR static bitstream. This field is entirely optional, but may be used to identify differing designs or bitstreams as needed. This register is located a byte offset of 0xC0 from the MCAP VSEC base address.

Table 3 describes the MCAP Bitstream Version ID fields.

Table 3: MCAP ID Fields

Name	Description	Value Width	Default Value
MCAP Bitstream Version	Xilinx defined field that contains a value defined by the user. This field can be used to identify the bitstream version that is used for the Tandem Stage1 or PR static bitstream. This value can be modified by changing the MCAP_FPGA_BITSTREAM_VERSION property on the PCIe hardblock instance. This field should be qualified by all previously listed ID fields prior to interpretation.	32-bit value	configuration dependent

MCAP Extended Capability Register Description

The MCAP VSEC is implemented as a set of registers defined in the PCIe configuration registers space at byte offset 0x340 for UltraScale devices. Once the device and the MCAP VSEC are properly identified the registers can be used to perform MCAP operations. Table 4 and the remainder of this section describe the MCAP VSEC registers as an offset from the MCAP VSEC base address (0x340).

Table 4: MCAP Extended Capability Registers

	Byte Offset
PCI Express Extended Capability Header	00H
Vendor-Specific Header	04H
FPGA JTAG ID	08H
FPGA Bit-Stream Version	0CH
Status Register	10H
Control Register	14H
Write Data Register	18H
Read Data0 Register	1CH
Read Data1 Register	20H
Read Data2 Register	24H
Read Data3 Register	28H

MCAP PCI Express Extended Capability Header Register

The PCI Express Extended Capability Header Register is defined by the PCI-SIG and is used to identify the format of the extended capability and provide a pointer to the next extended capability of applicable. These fields are populated with appropriate values when the PCI Express core is generated. Table 5 describes the fields within the MCAP PCI Express Extended Capability Header Register.

Table 5: MCAP PCI Express Extended Capability Header Register

Bit Location	Description	Initial Value	Attribute
15:0	**PCI Express Extended Capability ID** – This field is a PCI-SIG defined ID number that indicates the nature and format of the Extended Capability. Extended Capability ID for the Vendor-Specific Capability is 000Bh.	000BH	Read Only
19:16	**Capability Version** – This field is a PCI-SIG defined version number that indicates the version of the Capability structure present. Must be 1h for this version of the specification.	1H	Read Only
31:20	**Next Capability Offset** – This field contains the offset to the next PCI Express Capability structure or 000h if no other items exist in the linked list of Capabilities. For Extended Capabilities implemented in Configuration Space, this offset is relative to the beginning of PCI-compatible Configuration Space and thus must always be either 000h (for terminating list of Capabilities) or greater than 0FFh.	Value set automatically during IP customization	Read Only

MCAP Vendor Specific Header Register

The MCAP Vendor Specific Header Register provides the identification number, revision number, and length of the MCAP VSEC. These fields are populated with appropriate values when the PCI Express core is generated. Table 6 describes the fields within the MCAP Vendor Specific Header Register.

Table 6: MCAP Vendor Specific Header Register

Bit Location	Description	Initial Value	Attribute
15:0	**VSEC ID** – This is the ID value that can be used to identify the PCIe MCAP Extended capability. This value can be modified by setting the MCAP_VSEC_ID property on the PCIe hardblock instance.	0x0001	Read Only
19:16	**VSEC Rev** – This is the Revision ID value that can be used to identify the PCIe MCAP Extended capability. This value can be modified by setting the MCAP_VSEC_REV property on the PCIe hardblock instance.	0x0	Read Only
31:20	**VSEC Length** – This field indicates the number of bytes in the entire MCAP VSEC structure, including the PCI Express Extended Capability header, the Vendor-Specific header, and the MCAP Specific registers.	0x02C	Read Only

MCAP FPGA JTAG ID Register

The MCAP JTAG ID Register contains the JTAG identification number for the FPGA. The JTAG ID is a device specific value. Table 7 describes the fields within the MCAP JTAG ID Register.

Table 7: MCAP FPGA JTAG ID Register

Bit Location	Description	Initial Value	Attribute
31:0	**FPGA JTAG ID –** This field is the FPGA JTAG ID.	Device specific JTAG ID	Read Only

MCAP FPGA Bitstream Version Register

The MCAP Bitstream Version Register provides a custom identification field that can be used to differentiate between multiple versions of the FPGA bitstream. Table 8 describes the fields within the MCAP FPGA Bitstream Version Register.

Table 8: MCAP FPGA Bitstream Version Register

Bit Location	Description	Initial Value	Attribute
31:0	**FPGA Bit-Stream Version –** This value identifies the FPGA bitstream version and can be modified by setting the MCAP_FPGA_BITSTREAM_VERSION property on the PCIe hardblock instance.	Dependent on IP Configuration	Read Only

MCAP Status Register

The MCAP Status Register is used to view the status of the MCAP during operations. With the exception of the Request for MCAP Release bit, the data values in this register are only valid when the MCAP Enable bit within the control register is set to 1.

Table 9 describes the fields within the MCAP Status Register.

Table 9: MCAP Status Register

Bit Location	Description	Initial Value	Attribute
00:00	**MCAP Error:** This field indicates that a bitstream error has occurred while the MCAP is in use. When this value is 1 a bitstream error has been detected by the FPGA configuration logic. An **MCAP Reset** must be performed to reset the FPGA configuration logic and clear the error.	1'b0	Read Only
01:01	**MCAP EOS:** This field indicates the state of the End Of Startup signal within the FPGA. The behavior of this signal is similar to the behavior of the End Of Startup (EOS) that is generated by the STARTUP primitive.	1'b0	Read Only
03:02	**Reserved**	2'b0	Read Only
04:04	**MCAP Read Complete:** This field indicates that an FPGA Configuration Register Read operation has completed and the returned data is available for reading in the **MCAP Read Data (0-3) Registers.** This bit is cleared by setting the MCAP Read Enable bit to zero or by issuing an **MCAP Module Reset**.	1'b0	Read Only
07:05	**MCAP Read Count:** This field indicates the number of data words returned by an FPGA Configuration Register Read operation. This value is valid only when **MCAP Read Complete** bit is set to 1. Legal values for this field are 1 to 4. This register should be checked when performing FPGA Configuration Register Reads to determine how many of the **MCAP Read Data (0-3) Registers** have valid data. This field is cleared by issuing an **MCAP Module Reset**.	3'b000	Read Only

08:08	**MCAP FIFO Overflow:** Data written to the **MCAP Write Data Register** is buffered in 16 word data FIFO. This field is an over flow indicator for the write data FIFO and identifies data loss during a bitstream transfer. During normal MCAP operation this FIFO will be emptied faster than a Root Port device can write data to the **MCAP Write Data Register**. This bit should only assert if the MCAP clock within the FPGA design is no longer running or is running significantly slower than is expected. This field is cleared by issuing an **MCAP Module Reset**.	1'b0	Read Only
11:09	**Reserved**	3'b0	Read Only
15:12	**MCAP FIFO Occupancy:** Data written to the **MCAP Write Data Register** is buffered in 16 word data FIFO. This field is an indicator of the fill level for the write data FIFO. During normal MCAP operation this FIFO will be emptied faster than a Root Port device can write data to the **MCAP Write Data Register**. As such this field has been truncated to 4 bits and cannot be used to express a fill level of 16. If the write data FIFO overflows the **MCAP Write FIFO Overflow** bit will be asserted. This field is reset by issuing an **MCAP Module Reset**.	4'b0	Read Only
23:16	**Reserved**	2'b0	Read Only
24:24	**Request for MCAP Release:** This bit is used to arbitrate use of the configuration logic between the PCI Express link and other interfaces internal to the FPGA that make use of the configuration logic. This signal should be used in conjunction with the **Request for MCAP by PCIe** bit within the **MCAP Control Register**. Using these signals for arbitration is described in the **MCAP Arbitration** section.	1'b0	Read Only
31:25	**Reserved**	7'b0	Read Only

MCAP Control Register

The MCAP Control Register is used to Using Control and Status register to perform MCAP operations is described with flow charts in the following section. Table 10 describes the fields within the MCAP Control Register.

Table 10: MCAP Control Register

Bit Location	Description	Initial Value	Attribute
00:00	**MCAP Enable:** When set to 1, this bit shifts control of the FPGA configuration logic to the MCAP. When this occurs the configuration logic is reset and control of the configuration logic shifts to the MCAP Control and Status Registers. After this bit is asserted the fields within the Status Register become valid. This bit must be asserted before initiating any operation on the MCAP. The MCAP Enable bit should be driven to 0 when the MCAP operation is complete or when the MCAP is not in use. Setting MCAP Enable to zero allows other configuration interfaces to make use of the configuration logic. Note: The MCAP Enable bit should not be set to 1 without properly arbitrating between other configuration interfaces that are used by the FPGA design.	1'b0	Read/Write
01:01	**MCAP Read Enable:** This bit is used during a Configuration Read Register operation to enable a read to configuration registers. To read from FPGA configuration registers, a valid register read command sequence must first be written to the **MCAP Write Data Register**. Once this is accomplished **MCAP Read Enable** bit should be asserted until **MCAP Read Complete** asserts. Once the available data has been read from the **MCAP Read Data (0-3) Registers**, the **MCAP Read Enable** bit should be set to 0 before initiating another read or write operation.	1'b0	Read/Write
03:02	**Reserved**	3'b0	Read Only

04:04	**MCAP Reset:** This bit is used to initiate an **MCAP Reset**. When this bit is asserted in conjunction with **MCAP Enable** a reset will be issued to the configuration logic. This can be used to clear the **MCAP Error** field within the **MCAP Status Register** or to recover from and unknown state.	1’b0	Read/Write
05:05	**MCAP Module Reset:** When set to 1 in conjunction with the **MCAP Enable** bit, an **MCAP Module Reset** is issued. This resets the MCAP interfacing logic, but does not reset the configuration logic. An **MCAP Module Reset** should be used to clear an **MCAP Write FIFO Overflow** error or to reset the **MCAP Status Register** values.	1’b0	Read/Write
07:06	**Reserved**	2’b0	Read Only
08:08	**Request for MCAP by PCIe:** This bit is used to arbitrate use of the configurable logic between the PCI Express link and other interfaces internal to the FPGA that make use of the configuration logic (such as the ICAP primitive). This signal should be used in conjunction with the **Request for MCAP Release** bit within the **MCAP Control Register**. Using these signals for arbitration is described in the **MCAP Arbitration** section.	1’b0	Read/Write
11:09	**Reserved**	3’b0	Read Only
12:12	**Configure MCAP Design Switch:** This bit drives PCI Express Solution IP **MCAP_Design_Switch** output. For Tandem configurations, this bit controls the design isolation muxing within the PCI Express Solution IP. While the Design Switch bit is 0, the core will return Unsupported Requests (URs) for PCIe BAR transactions. When this bit becomes ‘1’ the isolation muxing on the PCIe core will be disabled and allow PCIe traffic to pass normally. When Tandem PROM is used this bit switches automatically after the initial stage2 loading and is then controllable from Host software. When Tandem PCIe is used this bit is controlled from Host software and must be set to 1 after the initial stage2 bit file is loaded via the PCIe MCAP VSEC. When loading a Tandem field update bitstream for an unmodified Tandem field update design, design isolation in the PCIe IP must first be enabled (design switch = 1’b0). Following the loading of the field update clear and partial files the design isolation must then be disabled (design switch = 1’b0). For non-Tandem or PR only designs, the design_switch register bit passes directly through the solution IP to the MCAP_Design_Switch output. This does not affect the operation of the PCIe IP in any way.	1’b0	Read/Write
15:13	**Reserved**	3’b0	Read Only
16:16	**Write Data Register Enable:** This field enables writes to the **MCAP Write Data Register** when set to 1. When set to 0, writes to **MCAP Write Data Register** are ignored.	1’b0	Read/Write
31:17	**Reserved**	15’b0	Read Only

MCAP Write Data Register

The MCAP Write Data Register is used to write bitstream data into the configuration logic during configuration write operations. Data written to this register is buffered in a 16 word FIFO and then passed to the configuration logic when the MCAP Enable and Write Data Register Enable fields in the MCAP Control Register are set to 1. Table 11 describes the fields within the MCAP Write Data Register.

Table 11: MCAP Write Data Register

Bit Location	Description	Initial Value	Attribute
31:0	**MCAP Write Data Register:** Writes to this register are buffered in the write data FIFO and then transferred to Configuration logic when then **MCAP Enable** and **Write Data Register Enable** fields are set to 1. Reads return 32'b0.	32'b0	Read zeroes Write

MCAP Read Data (0-3) Registers

The MCAP Read Data Registers are used to return data that is read from the FPGA Configuration Registers during Configuration Register Read operations. The data in these registers is only valid when the MCAP Register Read Enable and MCAP Register Read Complete bits are set to 1. The MCAP Register Read Count field indicates which of the MCAP Read Data Registers contain valid data. Configuration Read transaction always return data starting at the MCAP Read Data 0 Register. Subsequent MCAP Read Data Registers are filled sequentially. Most FPGA Configuration Register Read operations will only return 1 word of data. Full FPGA readback is not permitted through the MCAP interface. Table 12 describes the fields within the MCAP Read Data Registers.

Table 12: MCAP Read Data Registers

Bit Location	Description	Initial Value	Attribute
31:0	**MCAP Read Data0 Register:** First DWORD returned during register read operation from FPGA Configuration controller. Format depends on FPGA configuration register read. Valid when **MCAP Register Read Complete** bit 1b and **MCAP Register Read Count** is in the range 1-4.	32'b0	Read Only
31:0	**MCAP Read Data1 Register:** Second DWORD returned during register read operation from FPGA Configuration controller. Format depends on FPGA configuration register read. Valid when **MCAP Register Read Complete** bit 1b and **MCAP Register Read Count** is in the range 2-4.	32'b0	Read Only
31:0	**MCAP Read Data2 Register:** Third DWORD returned during register read operation from FPGA Configuration controller. Format depends on FPGA configuration register read. Valid when **MCAP Register Read Complete** bit 1b and **MCAP Register Read Count** is in the range 3-4.	32'b0	Read Only
31:0	**MCAP Read Data3 Register:** Fourth DWORD returned during register read operation from FPGA Configuration controller. Format depends on FPGA configuration register read. Valid when **MCAP Register Read Complete** bit 1b and **MCAP Register Read Count** is 4.	32'b0	Read Only

MCAP Arbitration

This section discusses how to arbitrate between the MCAP and other configuration interfaces such as the ICAP. The MCAP Enable bit should only be asserted when the PCI Express core has been granted access through the use of these arbitration signals. The FPGA design should ensure that only one configuration interface is granted access to the FPGA Configuration Logic at a time. IPs, such as the Single Event Mitigation (SEM) IP, mighty also require use of configuration logic to achieve the desired functionality. When more than one configuration interface require the use of the configuration logic, an arbitration scheme between the two interfaces must be implemented in the system level design. To allow for this, the MCAP provides arbitration bits within the MCAP Control and Status Registers as well as arbitration signals to the fabric from the PCI Express Solution IP.

Table 13 describes each of these arbitration signals and how they should be used.

Table 13: Arbitration Signals

Signal Name	Direction	Description
cap_req	Fabric accessible as an output from the solution IP	Asserted via req_for_MCAP from the Host and PCIe is requesting access to the MCAP. If no other FPGA configuration interfaces are used in the design this port should be left unconnected.
cap_gnt	Fabric controllable as an input to the solution IP	Asserted by the FPGA design to grant MCAP access to the Host. If no other FPGA configuration interfaces are used in the design this port should be tied to constant 1'b1.
cap_rel	Fabric controllable as an input to the solution IP	Asserted by the FPGA design to request that the Host release the MCAP. "If no other FPGA configuration interfaces are used in the design this port should be tied to constant 1'b0.
req_for_MCAP	Host software controllable from the MCAP Control Register bit 8 (Read/Write)	Asserted by Host oftware to request access to the MCAP
req_for_release	Host software accessible from the MCAP Status Register bit 24 (Read Only)	Deasserted by FPGA logic when access to the MCAP is granted to the Host. Asserted when the Host is requested to release control of the MCAP.

The timing diagram shown in Figure 17 demonstrates how these signals can be used to arbitrate for use of the configuration logic between the FPGA hardware design and Host SW.

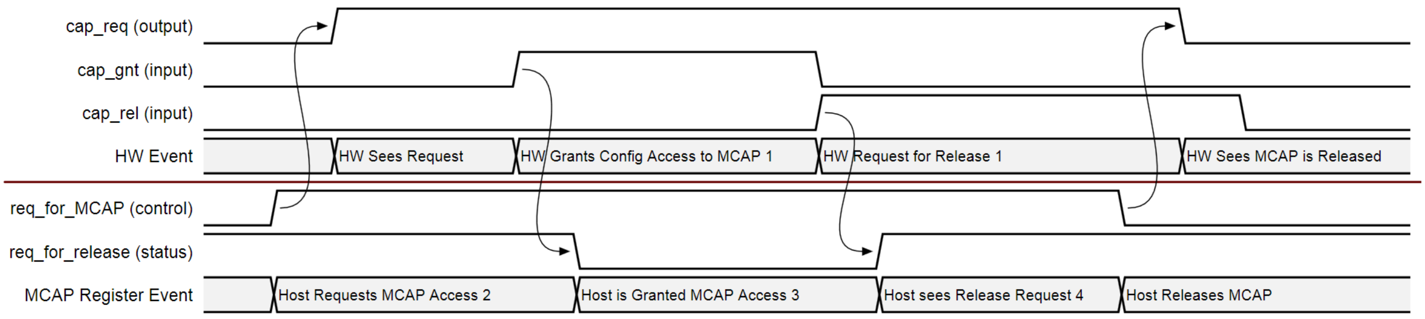


Figure 17: Configuration Logic Arbitration between the FPGA Design and Host SW

Footnotes for Figure 17:

1. During this time, Hardware should not grant Configuration Access to any other interface or peripheral.

2. Host requests access by writing the req_for_MCAP bit in the MCAP Control Register and waits for the req_for_rel bit in the MCAP Status register to deassert Host software is expected to check the req_for_release status between each atomic transaction.

3. During this time the Host has exclusive access to FPGA configuration. The Host should initiate transactions and MCAP events as desired.

4. At this point the Host should complete the current transaction and release control of the MCAP by deasserting the req_for_MCAP bit in the MCAP Control Register

Note: While all signals are synchronous to user_clk with a 1 cycle latency to the MCAP Registers, the interface should largely be considered asynchronous due to the variable delays that result from interaction with the Host System.

Figure 18 shows how Host software should use the arbitration bits within the Control and Status registers to initiate MCAP transactions. Since the flowchart below only describes arbitration for a single MCAP operation request for release, signaling is not used.

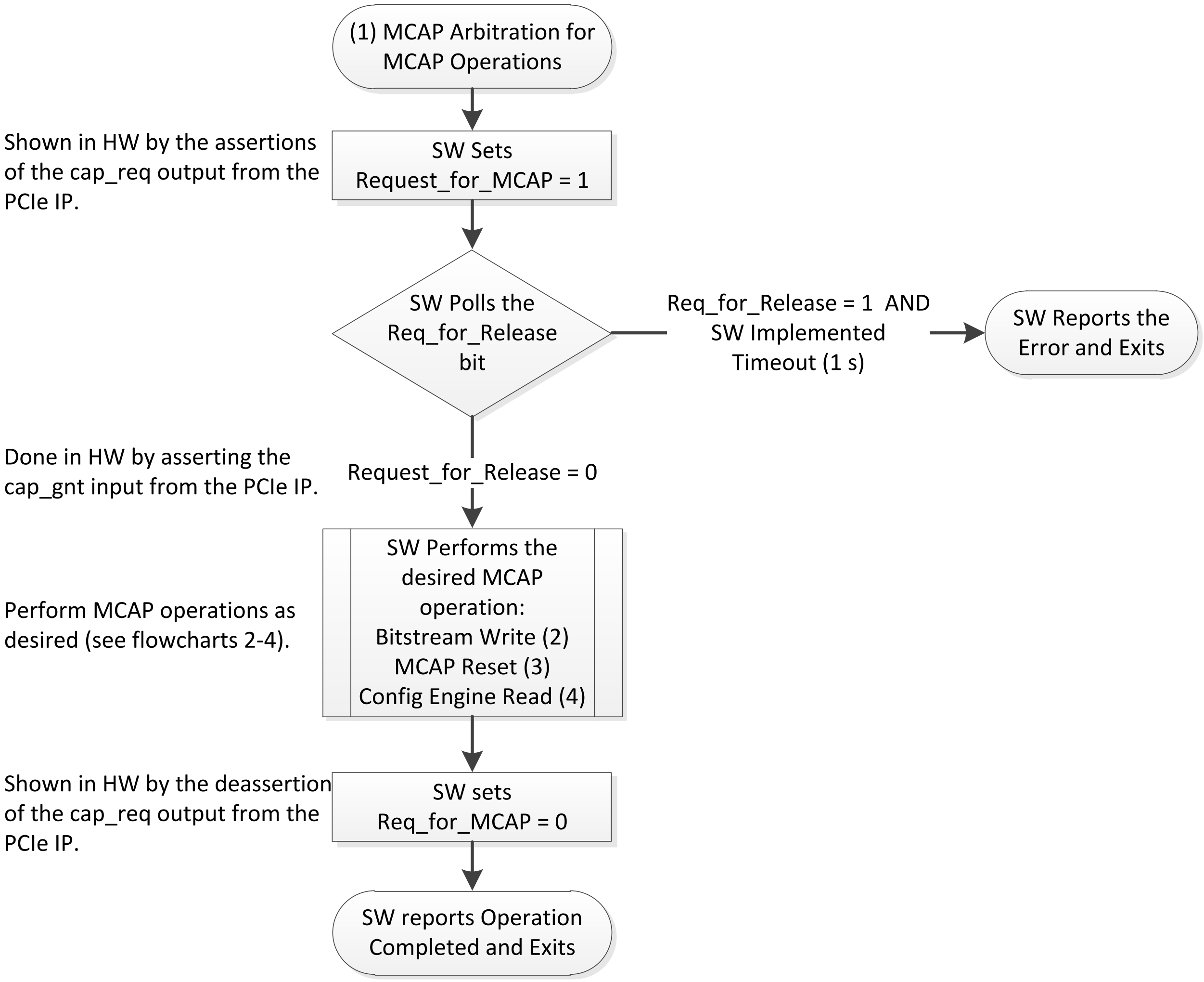

(1) MCAP Arbitration for MCAP Operations
Shown in HW by the assertions of the cap_req output from the PCIe IP.
SW Sets Request_for_MCAP = 1
SW Polls the Req_for_Release bit
Req_for_Release = 1 AND SW Implemented Timeout (1 s)
SW Reports the Error and Exits
Done in HW by asserting the cap_gnt input from the PCIe IP.
Request_for_Release = 0
Perform MCAP operations as desired (see flowcharts 2-4).
SW Performs the desired MCAP operation:
Bitstream Write (2)
MCAP Reset (3)
Config Engine Read (4)
Shown in HW by the deassertion of the cap_req output from the PCIe IP.
SW sets Req_for_MCAP = 0
SW reports Operation Completed and Exits

Figure 18: MCAP Arbitration Flowchart

Note:

- *If multiple operations are performed the software should check the Req_for_Rel bit within the MCAP Status Register. If this bit is set, the software should relinquish control of the MCAP by deasserting the Req_for_MCAP bit within the MCAP Control Register.*
- *The Software should take care to exit the program with the MCAP in an expected state. The error response and exit protocol may differ based on your system design and expected system behavior.*

Performing a Bitstream Write Operation

Figure 19 describes the steps that should be performed by the Host software to perform a Bitstream Write operation. These steps should be accomplished by PCIe Cfg read and write operations to the MCAP registers. A Bitstream Write operation can be used to load a Tandem Stage2 bitstreams, partial reconfiguration clear and partial bitstreams, or Configuration Engine command sequences into the FPGA. This operation should only be performed after access to the MCAP has been granted as described in the MCAP Arbitration section.

MCAP Arbitration (1) should have initiated prior to entering this flowchart step.

SW has exclusive access to the configuration engine.

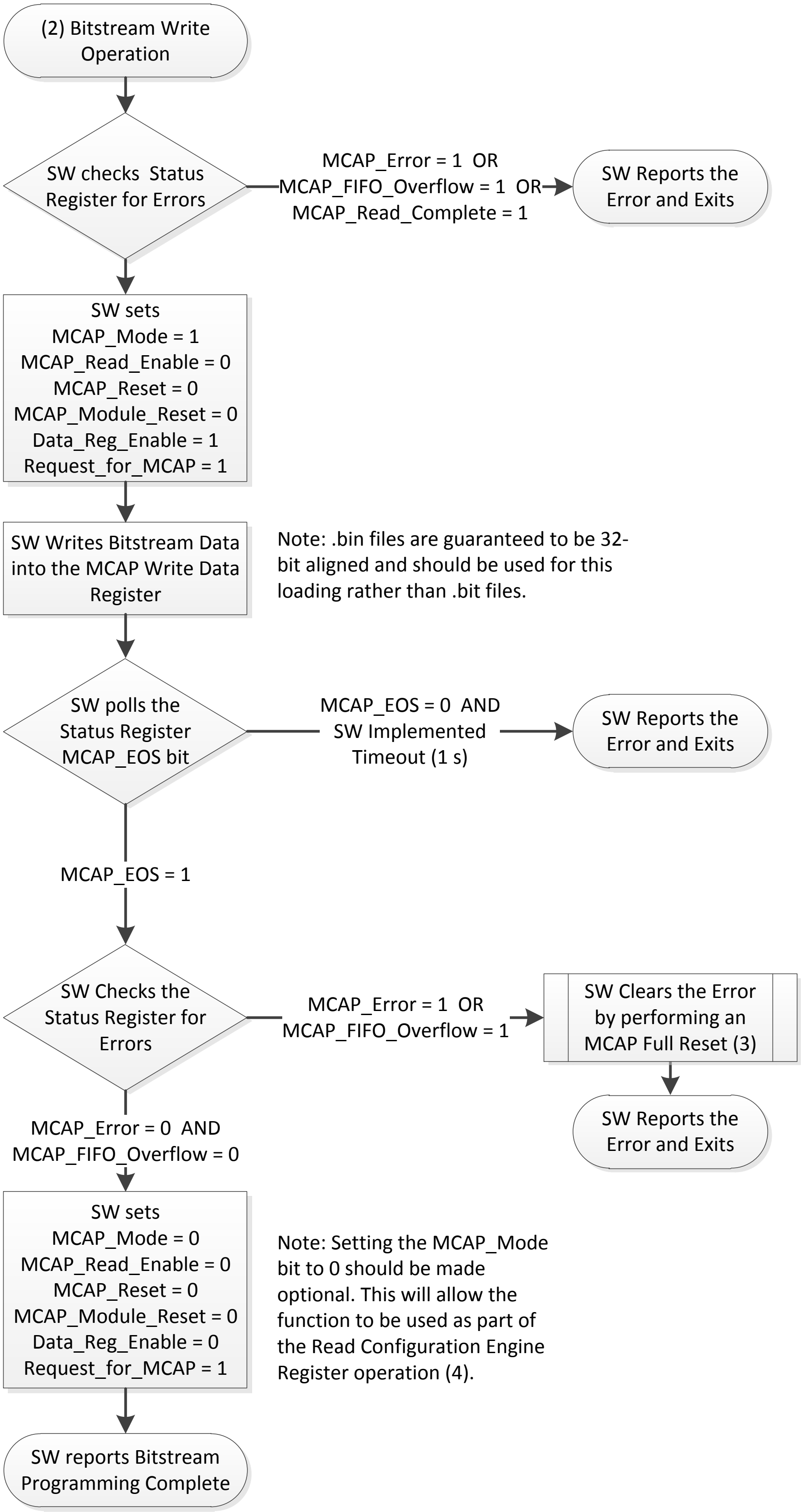
SW may optionally set the MCAP_Design_Switch = 0 bit. This is required for loading a Tandem Field Update with an unmodified PR region.(*)

For Partial Reconfiguration operations, data for both the current configuration Partial Clear file and the new configuration Partial file should be loaded.

During a configuration event MCAP_EOS will desert, when completed it will assert. MCAP_EOS is expected to be low between the loading of a Partial Clear and Partial bitstream programming files.

SW will optionally need to set MCAP_Design_Switch = 1. This is required for loading a Tandem PCIe Stage2 initial configuration and a Tandem Field Update configuration with unmodified PR region.(*)

MCAP Arbitration (1) should complete after leaving this flowchart step.



(2) Bitstream Write Operation
SW checks Status Register for Errors
MCAP_Error = 1 OR
MCAP_FIFO_Overflow = 1 OR
MCAP_Read_Complete = 1
SW Reports the Error and Exits
SW sets
MCAP_Mode = 1
MCAP_Read_Enable = 0
MCAP_Reset = 0
MCAP_Module_Reset = 0
Data_Reg_Enable = 1
Request_for_MCAP = 1
SW Writes Bitstream Data into the MCAP Write Data Register
SW polls the Status Register MCAP_EOS bit
MCAP_EOS = 0 AND
SW Implemented Timeout (1 s)
SW Reports the Error and Exits
MCAP_EOS = 1
SW Checks the Status Register for Errors
MCAP_Error = 1 OR
MCAP_FIFO_Overflow = 1
SW Clears the Error by performing an MCAP Full Reset (3)
SW Reports the Error and Exits
MCAP_Error = 0 AND
MCAP_FIFO_Overflow = 0
SW sets
MCAP_Mode = 0
MCAP_Read_Enable = 0
MCAP_Reset = 0
MCAP_Module_Reset = 0
Data_Reg_Enable = 0
Request_for_MCAP = 1
SW reports Bitstream Programming Complete

Note: .bin files are guaranteed to be 32-bit aligned and should be used for this loading rather than .bit files.

Note: Setting the MCAP_Mode bit to 0 should be made optional. This will allow the function to be used as part of the Read Configuration Engine Register operation (4).

Figure 19: Performing Bitstream Write Operation Flowchart

**Note: The MCAP_Design_Switch bit enables isolation of the PCIe IP for Tandem Configuration. If the pre-defined PR region has been modified, isolation for your reconfiguring module(s) must be considered and accounted for in your system level design.*

Note: the software should take care to exit the program with the MCAP in an expected state. The Error response and exit requirements may differ based on your system design and expected system behavior

Performing an MCAP Reset Operation

Figure 20 describes the steps that should be performed by the Host software to perform an MCAP Reset operation. These steps should be accomplished by PCIe Cfg read and write operations to the MCAP Control and Status registers. An MCAP Reset operation can be used to recover from an MCAP_Error, MCAP_FIFO_Overflow, or to reset the contents of the MCAP FIFO. This operation should only be performed after access to the MCAP has been granted as described in the MCAP Arbitration section.

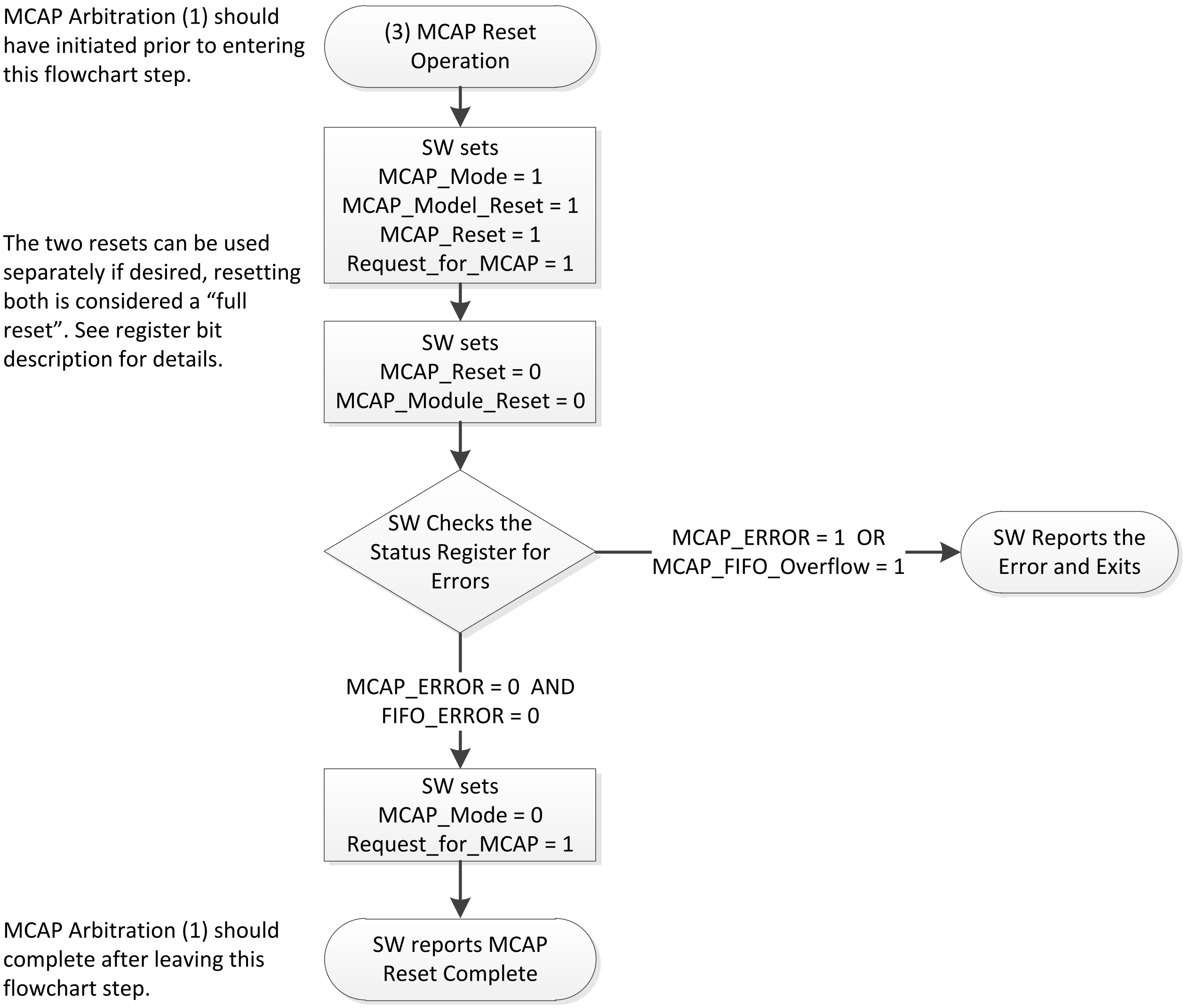

MCAP Arbitration (1) should have initiated prior to entering this flowchart step.
(3) MCAP Reset Operation
SW sets
MCAP_Mode = 1
MCAP_Model_Reset = 1
MCAP_Reset = 1
Request_for_MCAP = 1
The two resets can be used separately if desired, resetting both is considered a “full reset”. See register bit description for details.
SW sets
MCAP_Reset = 0
MCAP_Module_Reset = 0
SW Checks the Status Register for Errors
MCAP_ERROR = 1 OR
MCAP_FIFO_Overflow = 1
SW Reports the Error and Exits
MCAP_ERROR = 0 AND
FIFO_ERROR = 0
SW sets
MCAP_Mode = 0
Request_for_MCAP = 1
MCAP Arbitration (1) should complete after leaving this flowchart step.
SW reports MCAP Reset Complete

Figure 20: Performing an MCAP Reset Operation Flowchart

Note: The software should take care to exit the program with the MCAP in an expected state. The Error response and exit requirements can differ based on your system design and expected system behavior

Performing a Configuration Register Read Operation

The MCAP Registers can also be used to read the internal FPGA configuration registers. This is accomplished by writing an appropriate command sequence into the MCAP Data register, enabling read mode, and reading the data from the MCAP Read Data registers. The steps to perform this operation are described in Figure 21. This operation can be used to determine what type of error resulted in the assertion of the MCAP Error bit or to access the registers within the configuration engine.

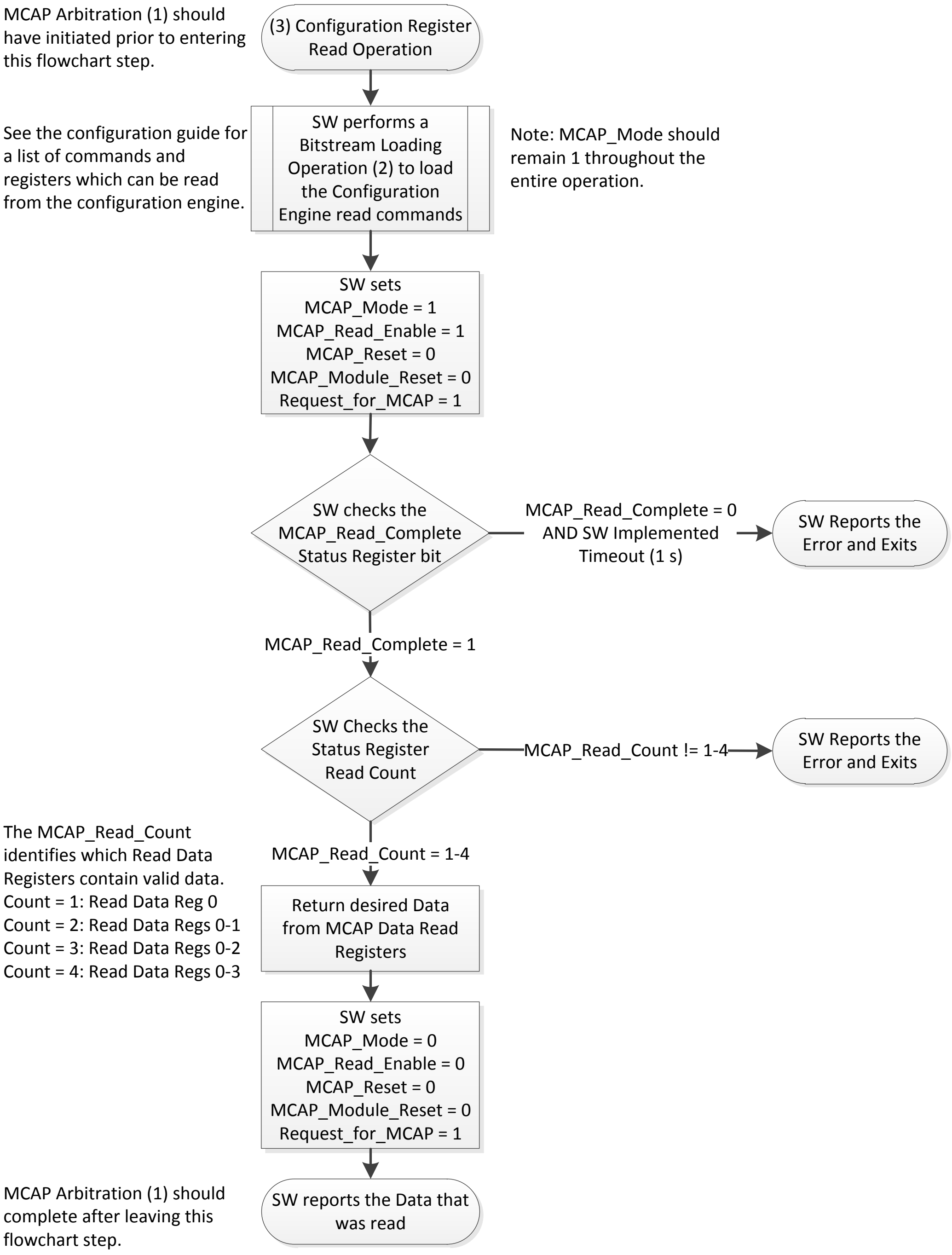

MCAP Arbitration (1) should have initiated prior to entering this flowchart step.
(3) Configuration Register Read Operation
See the configuration guide for a list of commands and registers which can be read from the configuration engine.
SW performs a Bitstream Loading Operation (2) to load the Configuration Engine read commands
Note: MCAP_Mode should remain 1 throughout the entire operation.
SW sets MCAP_Mode = 1 MCAP_Read_Enable = 1 MCAP_Reset = 0 MCAP_Module_Reset = 0 Request_for_MCAP = 1
SW checks the MCAP_Read_Complete Status Register bit
MCAP_Read_Complete = 0 AND SW Implemented Timeout (1 s)
SW Reports the Error and Exits
MCAP_Read_Complete = 1
SW Checks the Status Register Read Count
MCAP_Read_Count != 1-4
SW Reports the Error and Exits
The MCAP_Read_Count identifies which Read Data Registers contain valid data.
Count = 1: Read Data Reg 0
Count = 2: Read Data Regs 0-1
Count = 3: Read Data Regs 0-2
Count = 4: Read Data Regs 0-3
MCAP_Read_Count = 1-4
Return desired Data from MCAP Data Read Registers
SW sets MCAP_Mode = 0 MCAP_Read_Enable = 0 MCAP_Reset = 0 MCAP_Module_Reset = 0 Request_for_MCAP = 1
MCAP Arbitration (1) should complete after leaving this flowchart step.
SW reports the Data that was read

Figure 21: Performing a Configuration Register Read Operation Flowchart

Note: the software should take care to exit the program with the MCAP in an expected state. The Error response and exit requirements may differ based on your system design and expected system behavior.

References

These documents provide supplemental material useful with this document.

1. *UltraScale Architecture Gen3 Integrated Block for PCI Express: LogiCORE IP Product Guide* (PG156)
2. *UltraScale Architecture Configuration User Guide* (UG570)
3. *Vivado Design Suite User Guide: Partial Reconfiguration* (UG909)
4. *Xilinx Answer 51950 – Tandem PCIe Second Stage Bitstream Loading across the PCI Express Link for 7 Series Devices*

Revision History

06/30/2015 – Initial release
10/06/2015 – Added Tandem Field Update