# Protocols PCI Bus Support Test

## EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_PCI\_ROOT\_BRIDGE\_IO\_ PROTOCOL Section.

Configuration

Some checkpoints in the EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL test are device related. If the user needs to check the protocol on the specified device, the related profile needs to be updated to provide the specified information about this device.

For the format of the profile, please refer to A.2.

### PollMem()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.1.1.1 | 0xa10d3292, 0x6908, 0x446f, 0x9b, 0xfa, 0x38, 0x67, 0x75, 0xc6, 0x3e, 0x2e | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.PollMem – PollMem() with the correct value written to the destination address before delay time out returns EFI\_SUCCESS | 1. Call Mem.Write() to write specific value to destination address before the PollMem() delay times out.  2. Call PollMem() to poll the specific value on destination address. It should return EFI\_SUCCESS when required value is written to destination address. |
| 5.8.1.1.2 | 0xec6af458, 0x3dc1, 0x4022, 0xae, 0x0a, 0x7a, 0xd5, 0x61, 0x58, 0xdc, 0x5c | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.PollMem – PollMem() returns EFI\_SUCCESS immediately when required value has been written to destination address. | 1. Call Mem.Write() to write specific value to destination address before call of PollMem().  2. Call PollMem() to poll the specific value on destination address. It should return EFI\_SUCCESS immediately. |
| 5.8.1.1.3 | 0x6f82fa28, 0x8c61, 0x4af9, 0x8b, 0x77, 0xc9, 0xab, 0x26, 0x64, 0x10, 0x30 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.PollMem – PollMem() with delay as 0 returns EFI\_SUCCESS immediately. | 1. Call PollMem() to poll the specific value on destination address with delay as 0. It should return EFI\_SUCCESS immediately. |
| 5.8.1.1.4 | 0x2f0c1ddc, 0x53f3, 0x4053, 0xa8, 0xce, 0x37, 0x0f, 0xff, 0xac, 0x56, 0x05 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.PollMem – PollMem() with the invalid value written to the destination address before delay time out returns EFI\_TIME\_OUT | 1. Call Mem.Write() to write specific value to destination address before the PollMem() delay time out.  2. Call PollMem() to poll the different value on destination address. The return code should be EFI\_TIME\_OUT after delay time out. |
| 5.8.1.1.5 | 0x1d028ad2, 0xd563, 0x445e, 0x8c, 0x68, 0x92, 0x6f, 0x66, 0x35, 0x12, 0xa5 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.PollMem – PollMem() with Width as **EfiPciWidthMaximum** returns EFI\_INVALID\_PARAMETER. | 1. Call PollMem() with Width as **EfiPciWidthMaximum**. The return code should be EFI\_INVALID\_PARAMETER |
| 5.8.1.1.6 | 0x78d809be, 0xa958, 0x4c16, 0xb7, 0xbc, 0xbd, 0xb0, 0x26, 0xa0, 0x10, 0x48 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.PollMem – PollMem() with Width as **EfiPciWidthFifoUintX** returns EFI\_INVALID\_PARAMETER. | 1. Call PollMem() with Width as **EfiPciWidthFifoUintX**. The return code should be EFI\_INVALID\_PARAMETER |
| 5.8.1.1.7 | 0x87dc296a, 0xa156, 0x4601, 0x8c, 0xfb, 0x25, 0xd5, 0xa5, 0xcb, 0x64, 0x11 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.PollMem – PollMem() with Width as **EfiPciWidthFillUintX** returns EFI\_INVALID\_PARAMETER. | 1. Call PollMem() with Width as **EfiPciWidthFillUintX**. The return code should be EFI\_INVALID\_PARAMETER |
| 5.8.1.1.8 | 0x4e02eeec, 0x660d, 0x4782, 0xb2, 0xec, 0x2f, 0x5a, 0x66, 0x6c, 0xf2, 0xb7 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.PollMem – PollMem() with Width as -1 returns EFI\_INVALID\_PARAMETER. | 1. Call PollMem() with Width as -1. The return code should be EFI\_INVALID\_PARAMETER |
| 5.8.1.1.9 | 0x438d7bdd, 0x3e1b, 0x44dc, 0xb3, 0x53, 0x54, 0xf1, 0x9f, 0x02, 0x2d, 0x88 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.PollMem – PollMem() with Result as NULL returns EFI\_INVALID\_PARAMETER. | 1. Call PollMem() with Result as NULL. The return code should be EFI\_INVALID\_PARAMETER |

### PollIo()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.1.2.1 | 0x7f89a139, 0x7bba, 0x41da, 0xaa, 0x92, 0x1c, 0xe3, 0xc4, 0x77, 0x97, 0x68 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.PollIo – PollIo() with the correct value written to the destination Io address before delay time out returns EFI\_SUCCESS | 1. Call Io.Write() to write specific value to destination Io address before the PollIo() delay time out.  2. Call PollIo() to poll the specific value on destination Io address. It should return EFI\_SUCCESS when required value is written to destination address. |
| 5.8.1.2.2 | 0xf6882063, 0xc841, 0x4822, 0xa9, 0x86, 0x16, 0x7e, 0xce, 0x5b, 0x2c, 0x76 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.PollIo – PollIo() returns EFI\_SUCCESS immediately when required value has been written to destination address. | 1. Call Io.Write() to write specific value to destination address before call of PollIo().  2. Call PollIo() to poll the specific value on destination address. It should return EFI\_SUCCESS immediately. |
| 5.8.1.2.3 | 0x2ba92ffe, 0x557b, 0x4e2e, 0xa1, 0x22, 0x7c, 0x12, 0x36, 0x87, 0xdf, 0x6a | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.PollIo – PollIo() with delay as 0 returns EFI\_SUCCESS immediately. | 1. Call PollIo() to poll the specific value on destination address with delay as 0. It should return EFI\_SUCCESS immediately. |
| 5.8.1.2.4 | 0x424cfc17, 0x7335, 0x49d5, 0xb7, 0x9f, 0xa5, 0xfd, 0x90, 0xf2, 0xc5, 0x5e | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.PollIo – PollIo() with the invalid value written to the destination address before delay time out returns EFI\_TIME\_OUT | 1. Call Io.Write() to write specific value to destination address before the PollIo() delay time out.  2. Call PollIo() to poll the different value on destination address. The return code should be EFI\_TIME\_OUT after delay time out. |
| 5.8.1.2.5 | 0xb46d5e49, 0xe908, 0x4874, 0x96, 0x2f, 0xf8, 0x4e, 0x21, 0x6d, 0xcb, 0x54 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.PollIo – PollIo() with Width as **EfiPciWidthMaximum** returns EFI\_INVALID\_PARAMETER. | 1. Call PollIo() with Width as **EfiPciWidthMaximum**. The return code should be EFI\_INVALID\_PARAMETER |
| 5.8.1.2.6 | 0x90f1257b, 0x115e, 0x4d5d, 0xa1, 0x83, 0x09, 0xed, 0xc9, 0x5c, 0x18, 0x08 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.PollIo – PollIo() with Width as **EfiPciWidthFifoUintX** returns EFI\_INVALID\_PARAMETER. | 1. Call PollIo() with Width as **EfiPciWidthFifoUintX**. The return code should be EFI\_INVALID\_PARAMETER |
| 5.8.1.2.7 | 0xf557d70d, 0x4418, 0x4903, 0x8a, 0xb7, 0x66, 0x6f, 0x11, 0x1a, 0xd3, 0x37 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.PollIo – PollIo() with Width as **EfiPciWidthFillUintX** returns EFI\_INVALID\_PARAMETER. | 1. Call PollIo() with Width as **EfiPciWidthFillUintX**. The return code should be EFI\_INVALID\_PARAMETER |
| 5.8.1.2.8 | 0xd00129f5, 0x35d4, 0x4c01, 0xa7, 0x41, 0x00, 0xc7, 0xd5, 0xa5, 0x19, 0x0f | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.PollIo – PollIo() with Width as -1 returns EFI\_INVALID\_PARAMETER. | 1. Call PollIo() with Width as -1. The return code should be EFI\_INVALID\_PARAMETER |
| 5.8.1.2.9 | 0x7465fa90, 0xa357, 0x442f, 0xa8, 0xec, 0xf8, 0x86, 0x5f, 0xb6, 0xe2, 0xca | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.PollIo – PollIo() with Result as NULL returns EFI\_INVALID\_PARAMETER. | 1. Call PollIo() with Result as NULL. The return code should be EFI\_INVALID\_PARAMETER |

### Mem.Read()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.1.3.1 | 0x122320b0, 0x435d, 0x449b, 0x9c, 0xc0, 0x99, 0xd5, 0x95, 0xc9, 0xd2, 0x3d | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Mem.Read – Mem.Read() with **EfiPciWidthUintX** returns EFI\_SUCCESS. | 1. Call Mem.Read() with data width as **EfiPciWidthUintX**(X=8,16,32) to read Mem address contents to buffer. The return code should be EFI\_SUCCESS. |
| 5.8.1.3.2 | 0xc29f3981, 0x0a68, 0x48f0, 0x99, 0xfe, 0xc2, 0xe4, 0x84, 0xe8, 0xd2, 0x9d | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Mem.Read – Mem.Read() with **EfiPciWidthUintX** returns EFI\_SUCCESS. | 1. Call Mem.Read() with data width as **EfiPciWidthUintX**(X=8,16,32) to read Mem address contents to backup buffer.  2. Call Mem.Write() to write backup buffer contents to Mem address.  3. Call Mem.Read() again to read Mem address contents to another buffer. The return code should be EFI\_SUCCESS. |
| 5.8.1.3.3 | 0x57e2d8b2, 0xed4c, 0x4856, 0x82, 0xb6, 0xa0, 0xfd, 0x80, 0xd0, 0xb2, 0x55 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Mem.Read – Mem.Read() with **EfiPciWidthUintX** returns the contents written by Mem.Write(). | 1. Call Mem.Read() with data width as **EfiPciWidthUintX**(X=8,16,32) to read Mem address contents to backup buffer.  2. Call Mem.Write() to write backup buffer contents to Mem address.  3. Call Mem.Read() again to read Mem address contents to another buffer. The read contents in buffer should be the same as backup buffer. |
| 5.8.1.3.4 | 0x729ba46d, 0x7962, 0x4a2b, 0xb5, 0x20, 0xbf, 0x52, 0xa2, 0x02, 0x3c, 0xbe | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Mem.Read – Mem.Read() with **EfiPciWidthFifoUintX** returns EFI\_SUCCESS. | 1. Call Mem.Read() with data width as **EfiPciWidthFifoUintX**(X=8,16,32) to read Mem address contents to buffer. The return code should be EFI\_SUCCESS. |
| 5.8.1.3.5 | 0x701e90f7, 0xd218, 0x411f, 0xba, 0x7d, 0xb5, 0xab, 0x92, 0x2a, 0xcb, 0x93 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Mem.Read – Mem.Read() with **EfiPciWidthFifoUintX** only increases buffer for each of the count operations performed. | 1. Call Mem.Write() with **EfiPciWidthUintX** to write Buffer1 to memory address.  2. Call Mem.Read() with data width as **EfiPciWidthFifoUintX** from the same memory address to Buffer2. All units of Buffer2 should be the first unit of Buffer1. |
| 5.8.1.3.6 | 0x383c6e62, 0xf92f, 0x4719, 0x9a, 0x11, 0x70, 0x95, 0x08, 0x31, 0x19, 0xad | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Mem.Read – Mem.Read() with **EfiPciWidthFillUintX** returns EFI\_SUCCESS. | 1. Call Mem.Read() with data width as **EfiPciWidthFillUintX**(X=8,16,32) to read Mem address contents to buffer. The return code should be EFI\_SUCCESS. |
| 5.8.1.3.7 | 0x596a5971, 0x11d4, 0x43b0, 0x82, 0x4d, 0xe5, 0xcc, 0x41, 0x81, 0x9e, 0x14 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Mem.Read – Mem.Read() with **EfiPciWidthFillUintX** only increases address for each of the count operations performed. | 1. Call Mem.Write() with **EfiPciWidthUintX** to write Buffer1 to memory address.  2. Set all units of Buffer2 with the same value.  2. Call Mem.Read() with data width as **EfiPciWidthFillUintX** from the same memory address to Buffer2. The first unit of Buffer2 should be same as the last unit of Buffer1 and other units of Buffer2 should remain unchanged. |
| 5.8.1.3.8 | 0x28ba919b, 0xbc04, 0x464a, 0xbb, 0xa0, 0x87, 0xee, 0xda, 0xc1, 0x0f, 0x33 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Mem.Read – Mem.Read() with Width as **EfiPciWidthMaximum** returns EFI\_INVALID\_PARAMETER. | 1. Call Mem.Read() with Width as **EfiPciWidthMaximum**. The return code should be EFI\_INVALID\_PARAMETER |
| 5.8.1.3.9 | 0xbc884213, 0xe80e, 0x41e6, 0x81, 0x69, 0xbc, 0x46, 0x7d, 0x53, 0x40, 0x86 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Mem.Read – Mem.Read() with Width as -1 returns EFI\_INVALID\_PARAMETER. | 1. Call Mem.Read() with Width as -1. The return code should be EFI\_INVALID\_PARAMETER |
| 5.8.1.3.10 | 0x8cc49d7f, 0x87be, 0x4a2e, 0x82, 0xc0, 0xce, 0xc2, 0xbf, 0xcb, 0xb1, 0x3d | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Mem.Read – Mem.Read() with buffer as NULL returns EFI\_INVALID\_PARAMETER. | 1. Call Mem.Read() with buffer as NULL. The return code should be EFI\_INVALID\_PARAMETER |
| 5.8.1.3.11 | 0xbbf33c06, 0xa3a0, 0x4e13, 0xa3, 0xc7, 0x49, 0x23, 0x37, 0x07, 0xc9, 0x0d | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Mem.Read – Mem.Read() with unsupported Width from profile returns EFI\_INVALID\_PARAMETER. | 1. Call Mem.Read() with unsupported Width from profile. The return code should be EFI\_INVALID\_PARAMETER |

### Mem.Write()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.1.4.1 | 0x9dac86c8, 0xb700, 0x47ec, 0x95, 0x27, 0x9e, 0xf2, 0x39, 0x56, 0xbc, 0xca | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Mem.Write – Mem.Write() with **EfiPciWidthUintX** returns EFI\_SUCCESS. | 1. Call Mem.Write() with data width as **EfiPciWidthUintX**(X=8,16,32) to write buffer to Mem address. The return code should be EFI\_SUCCESS. |
| 5.8.1.4.2 | 0x1ed536a0, 0x7dbb, 0x4f97, 0xa7, 0xcd, 0xeb, 0xb4, 0xc4, 0x84, 0xab, 0x2b | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Mem.Write – Mem.Read() with **EfiPciWidthUintX** returns the contents written by Mem.Write(). | 1. Call Mem.Read() to read Mem address contents to backup buffer.  2. Call Mem.Write() with data width as **EfiPciWidthUintX**(X=8,16,32) to write backup buffer contents to Mem address.  3. Call Mem.Read() again to read Mem address contents to another buffer. The read contents in buffer should be the same as backup buffer. |
| 5.8.1.4.3 | 0xd2f05d14, 0xff03, 0x4b2d, 0x94, 0xbc, 0x11,0xd7, 0x7a, 0x56, 0x20, 0x5e | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Mem.Write – Mem.Write() with **EfiPciWidthFifoUintX** returns EFI\_SUCCESS. | 1. Call Mem.Write() with data width as **EfiPciWidthFifoUintX**(X=8,16,32) to write buffer contents to Mem address. The return code should be EFI\_SUCCESS. |
| 5.8.1.4.4 | 0x2e0a75e3, 0x04f3, 0x47f4, 0x85, 0x8f, 0x75, 0x1a, 0x29, 0xcf, 0x1c, 0x6a | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Mem.Write – Mem.Write() with **EfiPciWidthFifoUintX** only increases buffer for each of the count operations performed. | 1. Call Mem.Read() with **EfiPciWidthUintX** to read memory address contents to Buffer1.  2. Call Mem.Write() with **EfiPciWidthFifoUintX** to write Buffer1 to memory address.  3. Call Mem.Read() with data width as **EfiPciWidthUintX** from the same memory address to Buffer2. The first unit of Buffer2 should be the same as the last unit of Buffer1, and other units of Buffer2 should be the same as corresponding units of Buffer1. |
| 5.8.1.4.5 | 0xd220d6da, 0xa7b9, 0x477f, 0xa6, 0xfb, 0xc1, 0x52, 0x43, 0xe9, 0x52, 0x5e | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Mem.Write – Mem.Write() with **EfiPciWidthFillUintX** returns EFI\_SUCCESS. | 1. Call Mem.Write() with data width as **EfiPciWidthFillUintX**(X=8,16,32) to write buffer contents to Mem address. The return code should be EFI\_SUCCESS. |
| 5.8.1.4.6 | 0x8283aeec, 0x2896, 0x460b, 0x9e, 0xf1, 0xe7, 0xa6, 0x89, 0xa4, 0x8c, 0x86 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Mem.Write – Mem.Read() after Mem.Write the data using **EfiPciIoWidthFillUintX** return EFI\_SUCCESS. | 1. Call Mem.Write() with data width as **EfiPciWidthFillUintX**(X=8,16,32) to write buffer contents to Mem address.  2. Call Mem.Read() with data width as **EfiPciWidthUintX** to read Mem address contents to buffer. The return code should be EFI\_SUCCESS. |
| 5.8.1.4.7 | 0xcabf0b57, 0x7e2b, 0x40f6, 0x96, 0xa6, 0x3d, 0x4e, 0x92, 0xca, 0x5b, 0x55 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Mem.Write – Mem.Write() with **EfiPciWidthFillUintX** only increases address for each of the count operations performed. | 1. Call Mem.Write() with data width as **EfiPciWidthFillUintX**(X=8,16,32) to write Buffer1 contents to Mem address.  2. Call Mem.Read() with data width as **EfiPciWidthUintX** to read Mem address contents to Buffer2. All the units of Buffer2 should be the same as the first unit of Buffer1. |
| 5.8.1.4.8 | 0xaa2e8dd7, 0x501e, 0x4210, 0x8f, 0x10, 0xd0, 0x30, 0x78, 0x30, 0x75, 0x64 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Mem.Write – Mem.Write() with **EfiPciWidthUintX** returns EFI\_SUCCESS. | 1. Call Mem.Read() with data width as **EfiPciWidthUintX**(X=8,16,32) to write buffer back to Mem address. The return code should be EFI\_SUCCESS. |
| 5.8.1.4.9 | 0x26aa2144, 0x1c21, 0x4499, 0xb4, 0xdb, 0xda, 0xf4, 0x80, 0x07, 0xfa, 0xd9 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Mem.Write – Mem.Write() with Width as **EfiPciWidthMaximum** returns EFI\_INVALID\_PARAMETER. | 1. Call Mem.Write() with Width as **EfiPciWidthMaximum**. The return code should be EFI\_INVALID\_PARAMETER |
| 5.8.1.4.10 | 0x71b8a5d8, 0xf464, 0x416d, 0xb9, 0x73, 0x4e, 0xb0, 0xc1, 0x06, 0x94, 0x07 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Mem.Write – Mem.Write() with Width as -1 returns EFI\_INVALID\_PARAMETER. | 1. Call Mem.Write() with Width as -1. The return code should be EFI\_INVALID\_PARAMETER |
| 5.8.1.4.11 | 0x2b698420, 0x82b3, 0x43b3, 0xaa, 0x39, 0x53, 0xc2, 0x9d, 0x1d, 0x91, 0x13 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Mem.Write – Mem.Write() with buffer as NULL returns EFI\_INVALID\_PARAMETER. | 1. Call Mem.Write() with buffer as NULL. The return code should be EFI\_INVALID\_PARAMETER |
| 5.8.1.4.12 | 0xcf2417f3, 0x1491, 0x44ea, 0x93, 0xec, 0xad, 0x0b, 0x5b, 0xc0, 0x2b, 0xc6 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Mem.Write – Mem.Write() with unsupported Width from profile returns EFI\_INVALID\_PARAMETER. | 1. Call Mem.Write() with unsupported Width from profile. The return code should be EFI\_INVALID\_PARAMETER |

### Io.Read()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.1.5.1 | 0xf6d5c145, 0x15c9, 0x4bc5, 0xa5, 0x1c, 0xd5, 0xfd, 0xba, 0xf0, 0x73, 0xe9 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Io.Read – Io.Read() with **EfiPciWidthUintX** returns EFI\_SUCCESS. | 1. Call Io.Read() with data width as **EfiPciWidthUintX**(X=8,16,32) to read Io address contents to buffer. The return code should be EFI\_SUCCESS. |
| 5.8.1.5.2 | 0x12a1078a, 0xc78a, 0x446d, 0x90, 0x37, 0x22, 0xd8, 0xd0, 0x88, 0xfb, 0x2d | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Io.Read – Io.Read() with **EfiPciWidthUintX** returns EFI\_SUCCESS. | 1. Call Io.Read() with data width as **EfiPciWidthUintX**(X=8,16,32) to read Io address contents to backup buffer.  2. Call Io.Write() to write backup buffer contents to Io address.  3. Call Io.Read() again to read Io address contents to another buffer. The return code should be EFI\_SUCCESS. |
| 5.8.1.5.3 | 0xcc985605, 0x262d, 0x4954, 0xb4, 0x1c, 0xa9, 0x4c, 0xd0, 0x15, 0x7b, 0x96 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Io.Read – Io.Read() with **EfiPciWidthUintX** returns the contents written by Io.Write(). | 1. Call Io.Read() with data width as **EfiPciWidthUintX**(X=8,16,32) to read Io address contents to backup buffer.  2. Call Io.Write() to write backup buffer contents to Io address.  3. Call Io.Read() again to read Io address contents to another buffer. The read contents in buffer should be the same as backup buffer. |
| 5.8.1.5.4 | 0x0d6630e0, 0x4a9e, 0x4720, 0xa2, 0xe1, 0x4e, 0xf3, 0xef, 0x81, 0x5f, 0x41 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Io.Read – Io.Read() with **EfiPciWidthFifoUintX** returns EFI\_SUCCESS. | 1. Call Io.Read() with data width as **EfiPciWidthFifoUintX**(X=8,16,32) to read Io address contents to buffer. The return code should be EFI\_SUCCESS. |
| 5.8.1.5.5 | 0xddb273f7, 0xd3d7, 0x4ab2, 0xa2, 0x41, 0xcb, 0x78, 0x05, 0x76, 0x79, 0xe0 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Io.Read – Io.Read() with **EfiPciWidthFifoUintX** only increases buffer for each of the count operations performed. | 1. Call Io.Write() with **EfiPciWidthUintX** to write Buffer1 to Io address.  2. Call Io.Read() with data width as **EfiPciWidthFifoUintX** from the same Io address to Buffer2. All units of Buffer2 should be the first unit of Buffer1. |
| 5.8.1.5.6 | 0x349eb44d, 0x2db1, 0x4fa7, 0xa3, 0xf2, 0x1a, 0x08, 0x8d, 0xa9, 0x0e, 0x3c | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Io.Read – Io.Read() with **EfiPciWidthFillUintX** returns EFI\_SUCCESS. | 1. Call Io.Read() with data width as **EfiPciWidthFillUintX**(X=8,16,32) to read Io address contents to buffer. The return code should be EFI\_SUCCESS. |
| 5.8.1.5.7 | 0x3dcc7e09, 0x598c, 0x4fdb, 0xbb, 0x03, 0xda, 0xa6, 0x1a, 0xc9, 0x9f, 0x28 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Io.Read – Io.Read() with **EfiPciWidthFillUintX** only increases address for each of the count operations performed. | 1. Call Io.Write() with **EfiPciWidthUintX** to write Buffer1 to Io address.  2. Set all units of Buffer2 with the same value.  2. Call Io.Read() with data width as **EfiPciWidthFillUintX** from the same Io address to Buffer2. The first unit of Buffer2 should be same as the last unit of Buffer1 and other units of Buffer2 should remain unchanged. |
| 5.8.1.5.8 | 0xb7153211, 0xaf3b, 0x4a10, 0x85, 0x16, 0x5d, 0x5b, 0x13, 0x1d, 0x9e, 0x67 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Io.Read – Io.Read() with Width as **EfiPciWidthMaximum** returns EFI\_INVALID\_PARAMETER. | 1. Call Io.Read() with Width as **EfiPciWidthMaximum**. The return code should be EFI\_INVALID\_PARAMETER |
| 5.8.1.5.9 | 0x8578f6de, 0xc396, 0x42f7, 0x92, 0x42, 0x74, 0x37, 0x13, 0xdb, 0xbf, 0x6d | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Io.Read – Io.Read() with Width as -1 returns EFI\_INVALID\_PARAMETER. | 1. Call Io.Read() with Width as -1. The return code should be EFI\_INVALID\_PARAMETER |
| 5.8.1.5.10 | 0x50b7d46a, 0x73b5, 0x4bba, 0xa7, 0x36, 0x8a, 0xae, 0x97, 0x5c, 0x42, 0x6b | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Io.Read – Io.Read() with buffer as NULL returns EFI\_INVALID\_PARAMETER. | 1. Call Io.Read() with buffer as NULL. The return code should be EFI\_INVALID\_PARAMETER |
| 5.8.1.5.11 | 0xb24b8daa, 0x5ea2, 0x47d0, 0x88, 0xc0, 0x32, 0x3b, 0x26, 0x43, 0x2f, 0xbc | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Io.Read – Io.Read() with unsupported Width from profile returns EFI\_INVALID\_PARAMETER. | 1. Call Io.Read() with unsupported Width from profile. The return code should be EFI\_INVALID\_PARAMETER |

### Io.Write()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.1.6.1 | 0xa0954c3a, 0x86d9, 0x43a8, 0xb0, 0xcb, 0x13, 0xcf, 0x13, 0xe2, 0x82, 0x50 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Io.Write – Io.Write() with **EfiPciWidthUintX** returns EFI\_SUCCESS. | 1. Call Io.Write() with data width as **EfiPciWidthUintX**(X=8,16,32) to write buffer to Io address. The return code should be EFI\_SUCCESS. |
| 5.8.1.6.2 | 0xe401d5de, 0x3a4e, 0x4e21, 0xb1, 0x4c, 0x34, 0x90, 0xc6, 0xe8, 0xf3, 0xd8 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Io.Write – Io.Read() with **EfiPciWidthUintX** returns the contents written by Io.Write(). | 1. Call Io.Read() to read Io address contents to backup buffer.  2. Call Io.Write() with data width as **EfiPciWidthUintX**(X=8,16,32) to write backup buffer contents to Io address.  3. Call Io.Read() again to read Io address contents to another buffer. The read contents in buffer should be the same as backup buffer. |
| 5.8.1.6.3 | 0xef5142b5, 0xe421, 0x43b8, 0xb1, 0xd5, 0x17, 0x60, 0x46, 0x60, 0x72, 0x3a | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Io.Write – Io.Write() with **EfiPciWidthFifoUintX** returns EFI\_SUCCESS. | 1. Call Io.Write() with data width as **EfiPciWidthFifoUintX**(X=8,16,32) to write buffer contents to Io address. The return code should be EFI\_SUCCESS. |
| 5.8.1.6.4 | 0xd2f5dadf, 0x82f7, 0x4d25, 0x9a, 0x96, 0x50, 0xd5, 0xb6, 0xfe, 0x86, 0xbf | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Io.Write – Io.Write() with **EfiPciWidthFifoUintX** only increases buffer for each of the count operations performed. | 1. Call Io.Read() with **EfiPciWidthUintX** to read Io address contents to Buffer1.  2. Call Io.Write() with **EfiPciWidthFifoUintX** to write Buffer1 to Io address.  3. Call Io.Read() with data width as **EfiPciWidthUintX** from the same Io address to Buffer2. The first unit of Buffer2 should be the same as the last unit of Buffer1, and other units of Buffer2 should be the same as corresponding units of Buffer1. |
| 5.8.1.6.5 | 0xf6433206, 0xe359, 0x4a42, 0x82, 0x68, 0xb6, 0xbb, 0x68, 0x90, 0x6a, 0x3a | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Io.Write – Io.Write() with **EfiPciWidthFillUintX** returns EFI\_SUCCESS. | 1. Call Io.Write() with data width as **EfiPciWidthFillUintX**(X=8,16,32) to write buffer contents to Io address. The return code should be EFI\_SUCCESS. |
| 5.8.1.6.6 | 0x8912391c, 0xf457, 0x4e51, 0x82, 0xb4, 0xe8, 0xaf, 0x1c, 0x5a, 0x18, 0xc2 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Io.Write – Io.Write() with **EfiPciWidthFillUintX** only increases address for each of the count operations performed. | 1. Call Io.Write() with data width as **EfiPciWidthFillUintX**(X=8,16,32) to write Buffer1 contents to Io address.  2. Call Io.Read() with data width as **EfiPciWidthUintX** to read Io address contents to Buffer2. All the units of Buffer2 should be the same as the first unit of Buffer1. |
| 5.8.1.6.7 | 0xe347d0ed, 0x8fbd, 0x46c4, 0xbd, 0xfe, 0x27, 0x2f, 0x81, 0x3a, 0x84, 0x85 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Io.Write – Io.Write() with **EfiPciWidthUintX** returns EFI\_SUCCESS. | 1. Call Io.Read() with data width as **EfiPciWidthUintX**(X=8,16,32) to write buffer back to Io address. The return code should be EFI\_SUCCESS. |
| 5.8.1.6.8 | 0x21d34064, 0x9df8, 0x4edf, 0x81, 0xd8, 0xeb, 0x90, 0x9c, 0xe7, 0x53, 0xd5 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Io.Write – Io.Write() with Width as **EfiPciWidthMaximum** returns EFI\_INVALID\_PARAMETER. | 1. Call Io.Write() with Width as **EfiPciWidthMaximum**. The return code should be EFI\_INVALID\_PARAMETER |
| 5.8.1.6.9 | 0x9174967b, 0x1639, 0x46b0, 0xab, 0x66, 0x70, 0x59, 0x4e, 0x5a, 0x3f, 0x57 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Io.Write – Io.Write() with Width as -1 returns EFI\_INVALID\_PARAMETER. | 1. Call Io.Write() with Width as -1. The return code should be EFI\_INVALID\_PARAMETER |
| 5.8.1.6.10 | 0x429ab4d0, 0x8d64, 0x4308, 0xa3, 0x08, 0x3e, 0x48, 0xa5, 0x66, 0x70, 0x4b | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Io.Write – Io.Write() with buffer as NULL returns EFI\_INVALID\_PARAMETER. | 1. Call Io.Write() with buffer as NULL. The return code should be EFI\_INVALID\_PARAMETER |
| 5.8.1.6.11 | 0x3d761cee, 0x9d62, 0x4942, 0x91, 0xde, 0xa9, 0xca, 0x93, 0xe4, 0xd5, 0x31 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Io.Write – Io.Write() with unsupported Width from profile returns EFI\_INVALID\_PARAMETER. | 1. Call Io.Write() with unsupported Width from profile. The return code should be EFI\_INVALID\_PARAMETER |

### Pci.Read()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.1.7.1 | 0x0a24c289, 0xe2b2, 0x465e, 0x93, 0x03, 0x20, 0x4e, 0xae, 0x23, 0x88, 0xd5 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Pci.Read – Pci.Read() with **EfiPciWidthUintX** returns EFI\_SUCCESS. | 1. Call Pci.Read() with data width as **EfiPciWidthUintX**(X=8,16,32) to read Pci address contents to buffer. The return code should be EFI\_SUCCESS. |
| 5.8.1.7.2 | 0x6a0884db, 0x48e2, 0x4330, 0x97, 0xa7, 0xf5, 0x26, 0x92, 0x4a, 0xf5, 0xea | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Pci.Read – Pci.Read() with **EfiPciWidthUintX** returns EFI\_SUCCESS. | 1. Call Pci.Read() with data width as **EfiPciWidthUintX**(X=8,16,32) to read Pci address contents to backup buffer.  2. Call Pci.Write() to write backup buffer contents to Pci address.  3. Call Pci.Read() again to read Pci address contents to another buffer. The return code should be EFI\_SUCCESS. |
| 5.8.1.7.3 | 0x34b35b73, 0xdb30, 0x4343, 0x85, 0x9a, 0x13, 0xb9, 0xac, 0x6e, 0x88, 0x9a | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Pci.Read – Pci.Read() with **EfiPciWidthUintX** returns the contents written by Pci.Write(). | 1. Call Pci.Read() with data width as **EfiPciWidthUintX**(X=8,16,32) to read Pci address contents to backup buffer.  2. Call Pci.Write() to write backup buffer contents to Pci address.  3. Call Pci.Read() again to read Pci address contents to another buffer. The read contents in buffer should be the same as backup buffer. |
| 5.8.1.7.4 | 0x0cb1fa0c, 0xfb2d, 0x4eed, 0x8d, 0x72, 0xb1, 0x65, 0x14, 0xcf, 0x95, 0xee | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Pci.Read – Pci.Read() with **EfiPciWidthFifoUintX** returns EFI\_SUCCESS. | 1. Call Pci.Read() with data width as **EfiPciWidthFifoUintX**(X=8,16,32) to read Pci address contents to buffer. The return code should be EFI\_SUCCESS. |
| 5.8.1.7.5 | 0x95094926, 0x51ab, 0x43c1, 0xb6, 0xb3, 0x77, 0xba, 0x39, 0x8b, 0x4a, 0x94 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Pci.Read – Pci.Read() with **EfiPciWidthFifoUintX** only increases buffer for each of the count operations performed. | 1. Call Pci.Write() with **EfiPciWidthUintX** to write Buffer1 to Pci address.  2. Call Pci.Read() with data width as **EfiPciWidthFifoUintX** from the same Pci address to Buffer2. All units of Buffer2 should be the first unit of Buffer1. |
| 5.8.1.7.6 | 0xfb4b5e93, 0x494b, 0x4865, 0x9e, 0xb0, 0x8c, 0xb5, 0xeb, 0x0d, 0x86, 0x64 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Pci.Read – Pci.Read() with **EfiPciWidthFillUintX** returns EFI\_SUCCESS. | 1. Call Pci.Read() with data width as **EfiPciWidthFillUintX**(X=8,16,32) to read Pci address contents to buffer. The return code should be EFI\_SUCCESS. |
| 5.8.1.7.7 | 0x711d56d9, 0x90d4, 0x422b, 0xad, 0x2b, 0xfe, 0xe9, 0x01, 0x2c, 0xfd, 0x7a | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Pci.Read – Pci.Read() with **EfiPciWidthFillUintX** only increases address for each of the count operations performed. | 1. Call Pci.Write() with **EfiPciWidthUintX** to write Buffer1 to Pci address.  2. Set all units of Buffer2 with the same value.  2. Call Pci.Read() with data width as **EfiPciWidthFillUintX** from the same Pci address to Buffer2. The first unit of Buffer2 should be same as the last unit of Buffer1 and other units of Buffer2 should remain unchanged. |
| 5.8.1.7.8 | 0xbeed4e4f, 0xf7aa, 0x480e, 0x97, 0xfd, 0x3d, 0xd8, 0x83, 0x5f, 0x47, 0x09 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Pci.Read – Pci.Read() with Width as **EfiPciWidthMaximum** returns EFI\_INVALID\_PARAMETER. | 1. Call Pci.Read() with Width as **EfiPciWidthMaximum**. The return code should be EFI\_INVALID\_PARAMETER |
| 5.8.1.7.9 | 0x1698aaaf, 0x8a6e, 0x4a56, 0xb6, 0xd5, 0x4e, 0xa4, 0x1d, 0x12, 0x2c, 0xb3 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Pci.Read – Pci.Read() with Width as -1 returns EFI\_INVALID\_PARAMETER. | 1. Call Pci.Read() with Width as -1. The return code should be EFI\_INVALID\_PARAMETER |
| 5.8.1.7.10 | 0x201fdef9, 0xdc84, 0x4c9d, 0x85, 0x98, 0x86, 0xf7, 0xca, 0x3f, 0xef, 0x81 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Pci.Read – Pci.Read() with buffer as NULL returns EFI\_INVALID\_PARAMETER. | 1. Call Pci.Read() with buffer as NULL. The return code should be EFI\_INVALID\_PARAMETER |
| 5.8.1.7.11 | 0xe0a36a5f, 0x3be9, 0x4b11, 0x9e, 0xfb, 0x90, 0x07, 0x1c, 0x73, 0x99, 0xc9 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Pci.Read – Pci.Read() with unsupported Width from profile returns EFI\_INVALID\_PARAMETER. | 1. Call Pci.Read() with unsupported Width from profile. The return code should be EFI\_INVALID\_PARAMETER |

### Pci.Write()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.1.8.1 | 0x22abcbe1, 0x5a58, 0x47d0, 0xb7, 0x3a, 0x6d, 0x3c, 0x55, 0x7a, 0xe9, 0x7c | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Pci.Write – Pci.Write() with **EfiPciWidthUintX** returns EFI\_SUCCESS. | 1. Call Pci.Write() with data width as **EfiPciWidthUintX**(X=8,16,32) to write buffer to Pci address. The return code should be EFI\_SUCCESS. |
| 5.8.1.8.2 | 0xb4e49e1b, 0xbe09, 0x4cdc, 0xbb, 0x56, 0xaa, 0x44, 0x4b, 0x86, 0xa6, 0x4a | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Pci.Write – Pci.Read() with **EfiPciWidthUintX** returns the contents written by Pci.Write(). | 1. Call Pci.Read() to read Pci address contents to backup buffer.  2. Call Pci.Write() with data width as **EfiPciWidthUintX**(X=8,16,32) to write backup buffer contents to Pci address.  3. Call Pci.Read() again to read Pci address contents to another buffer. The read contents in buffer should be the same as backup buffer. |
| 5.8.1.8.3 | 0xd753202a, 0xbe16, 0x4a58, 0x88, 0x3a, 0xcb, 0x5b, 0x82, 0xdf, 0xb8, 0xe8 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Pci.Write – Pci.Write() with **EfiPciWidthFifoUintX** returns EFI\_SUCCESS. | 1. Call Pci.Write() with data width as **EfiPciWidthFifoUintX**(X=8,16,32) to write buffer contents to Pci address. The return code should be EFI\_SUCCESS. |
| 5.8.1.8.4 | 0x241e4d94, 0xa5a2, 0x4192, 0x93, 0x66, 0x6d, 0x25, 0x8b, 0x20, 0x9b, 0xfc | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Pci.Write – Pci.Write() with **EfiPciWidthFifoUintX** only increases buffer for each of the count operations performed. | 1. Call Pci.Read() with **EfiPciWidthUintX** to read Pci address contents to Buffer1.  2. Call Pci.Write() with **EfiPciWidthFifoUintX** to write Buffer1 to Pci address.  3. Call Pci.Read() with data width as **EfiPciWidthUintX** from the same Pci address to Buffer2. The first unit of Buffer2 should be the same as the last unit of Buffer1, and other units of Buffer2 should be the same as corresponding units of Buffer1. |
| 5.8.1.8.5 | 0xadff8bd8, 0x7efd, 0x4368, 0x9b, 0x72, 0x0e, 0x9b, 0x10, 0xca, 0x13, 0x39 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Pci.Write – Pci.Write() with **EfiPciWidthFillUintX** returns EFI\_SUCCESS. | 1. Call Pci.Write() with data width as **EfiPciWidthFillUintX**(X=8,16,32) to write buffer contents to Pci address. The return code should be EFI\_SUCCESS. |
| 5.8.1.8.6 | 0xe9a41aa8, 0xd9be, 0x4b34, 0x99, 0xab, 0x40, 0x89, 0x08, 0x76, 0xc4, 0xe0 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Pci.Write – Pci.Write() with **EfiPciWidthFillUintX** only increases address for each of the count operations performed. | 1. Call Pci.Write() with data width as **EfiPciWidthFillUintX**(X=8,16,32) to write Buffer1 contents to Pci address.  2. Call Pci.Read() with data width as **EfiPciWidthUintX** to read Pci address contents to Buffer2. All the units of Buffer2 should be the same as the first unit of Buffer1. |
| 5.8.1.8.7 | 0x91076895, 0x66a6, 0x4d26, 0x84, 0xca, 0x8d, 0x38, 0xeb, 0x96, 0xd7, 0x5f | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Pci.Write – Pci.Write() with **EfiPciWidthUintX** returns EFI\_SUCCESS. | 1. Call Pci.Read() with data width as **EfiPciWidthUintX**(X=8,16,32) to write buffer back to Pci address. The return code should be EFI\_SUCCESS. |
| 5.8.1.8.8 | 0x7ff7a44c, 0x8647, 0x46de, 0x94, 0xe9, 0xe4, 0x0d, 0x30, 0xd1, 0x52, 0x41 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Pci.Write – Pci.Write() with Width as **EfiPciWidthMaximum** returns EFI\_INVALID\_PARAMETER. | 1. Call Pci.Write() with Width as **EfiPciWidthMaximum**. The return code should be EFI\_INVALID\_PARAMETER |
| 5.8.1.8.9 | 0x5928ba78, 0x13d0, 0x48bd, 0x8f, 0xf7, 0xa6, 0xee, 0x82, 0x79, 0xef, 0xea | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Pci.Write – Pci.Write() with Width as -1 returns EFI\_INVALID\_PARAMETER. | 1. Call Pci.Write() with Width as -1. The return code should be EFI\_INVALID\_PARAMETER |
| 5.8.1.8.10 | 0xb04a41bf, 0xa881, 0x4f93, 0xb6, 0x81, 0x14, 0x5c, 0xea, 0xaf, 0xa6, 0xa8 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Pci.Write – Pci.Write() with buffer as NULL returns EFI\_INVALID\_PARAMETER. | 1. Call Pci.Write() with buffer as NULL. The return code should be EFI\_INVALID\_PARAMETER |
| 5.8.1.8.11 | 0x009e4d36, 0xdc7e, 0x45a6, 0xa7, 0xa5, 0xfa, 0x8b, 0x79, 0x11, 0xfb, 0x0c | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Pci.Write – Pci.Write() with unsupported Width from profile returns EFI\_INVALID\_PARAMETER. | 1. Call Pci.Write() with unsupported Width from profile. The return code should be EFI\_INVALID\_PARAMETER |

### CopyMem()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.1.9.1 | 0x73a0ec23, 0x176e, 0x4560, 0xb2, 0xa3, 0x77, 0x13, 0xae, 0x8e, 0x42, 0xd2 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.CopyMem – CopyMem() between non-overlapping regions regions returns EFI\_SUCCESS. | 1. Set Buffer1 with specific value. Call Mem.Write() to write Buffer1 to Address1 with count units.  2. Call CopyMem() to copy Mem from Address1 to Address1+ **BufferSize** with count units. The return code should be EFI\_SUCCESS. |
| 5.8.1.9.2 | 0x6fd31187, 0xf3e6, 0x4b1d, 0x90, 0x61, 0xdc, 0xd8, 0x36, 0x98, 0xe6, 0xfc | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.CopyMem – The data in destination address should be the same as the source address after call of CopyMem() between non-overlapping regions. | 1. Set Buffer1 with specific value. Call Mem.Write() to write Buffer1 to **Address1** with count units.  2. Call CopyMem() to copy Mem from Address1 to Address1+ **BufferSize** with count units.  3. Call Mem.Read() to read data of Address1+**BufferSize** to Buffer2. All units of Buffer2 should be the same as Buffer1. |
| 5.8.1.9.3 | 0x4110b651, 0xb45e, 0x4684, 0xae, 0x38, 0x72, 0x8d, 0x01, 0xbb, 0x00, 0x97 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.CopyMem – CopyMem() between overlapping regions with destination address > source address returns EFI\_SUCCESS. | 1. Set **Buffer1** with specific value. Call Mem.Write() to write Buffer1 to Address1 with count units.  2. Call CopyMem() to copy Mem from Address1 to Address1+ **BufferSize**/2 with count units. The return code should be EFI\_SUCCESS. |
| 5.8.1.9.4 | 0x2f84ec07, 0xa38a, 0x4db2, 0xac, 0x0f, 0x66, 0x4f, 0x91, 0x3b, 0xb3, 0xea | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.CopyMem – After call of CopyMem() between overlapping regions, the data in destination address should be the same as the buffer contents written to the source address. | 1. Set Buffer1 with specific value. Call Mem.Write() to write Buffer1 to Address1 with count units.  2. Call CopyMem() to copy Mem from Address1 to Address1+ **BufferSize**/2 with count units.  3. Call Mem.Read() to read data of Address1+**BufferSize**/2 to Buffer2. All units of Buffer2 should be the same as Buffer1. |
| 5.8.1.9.5 | 0x4081f6bf, 0xf332, 0x44de, 0xb8, 0x62, 0x19, 0xe5, 0xaa, 0xdb, 0x43, 0x7e | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.CopyMem – CopyMem() between overlapping regions with destination address < source address returns EFI\_SUCCESS. | 1. Set Buffer1 with specific value. Call Mem.Write() to write Buffer1 to Address1+ **BufferSize**/2 with count units.  2. Call CopyMem() to copy Mem from Address1+ **BufferSize**/2 to Address1 with count units. The return code should be EFI\_SUCCESS. |
| 5.8.1.9.6 | 0x8fb4d613, 0x2bde, 0x4f40, 0x9c, 0x70, 0xe1, 0x60, 0x34, 0xdc, 0x3b, 0xbc | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.CopyMem – After call of CopyMem() between overlapping regions, the data in destination address should be the same as the buffer contents written to the source address. | 1. Set Buffer1 with specific value. Call Mem.Write() to write Buffer1 to Address1+ **BufferSize**/2 with count units.  2. Call CopyMem() to copy Mem from Address1+ **BufferSize**/2 to Address1 with count units.  3. Call Mem.Read() to read data of Address1 to Buffer2. All units of Buffer2 should be the same as Buffer1. |
| 5.8.1.9.7 | 0x0bcb82fb, 0x7052, 0x4d0f, 0xad, 0x73, 0xd3, 0xe7, 0x25, 0xae, 0x46, 0xb5 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.CopyMem – CopyMem() with Width as **EfiPciWidthMaximum** returns EFI\_INVALID\_PARAMETER. | 1. Call CopyMem() with Width as **EfiPciWidthMaximum**. The return code should be EFI\_INVALID\_PARAMETER |
| 5.8.1.9.8 | 0x9f7bf606, 0xf898, 0x42f2, 0xb7, 0x7f, 0xc1, 0x39, 0xa5, 0x90, 0x65, 0x6c | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.CopyMem – CopyMem() with Width as -1 returns EFI\_INVALID\_PARAMETER. | 1. Call CopyMem() with Width as -1. The return code should be EFI\_INVALID\_PARAMETER |
| 5.8.1.9.9 | 0x5762a830, 0x4fd5, 0x4858, 0x82, 0x1f, 0x76, 0xab, 0x12, 0xe9, 0xa9, 0x80 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.CopyMem – CopyMem() with Width as **EfiPciWidthFifoUintX** returns EFI\_INVALID\_PARAMETER. | 1. Call CopyMem() with Width as **EfiPciWidthFifoUintX**. The return code should be EFI\_INVALID\_PARAMETER |
| 5.8.1.9.10 | 0x09154449, 0xd6bc, 0x47b3, 0x8a, 0x47, 0x25, 0xd3, 0x08, 0x81, 0xa5, 0x0f | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.CopyMem – CopyMem() with Width as **EfiPciWidthFillUintX** returns EFI\_INVALID\_PARAMETER. | 1. Call CopyMem() with Width as **EfiPciWidthFillUintX**. The return code should be EFI\_INVALID\_PARAMETER |
| 5.8.1.9.11 | 0x6ea5136c, 0x0060, 0x4e70, 0xa1, 0x7a, 0xc1, 0xf0, 0xbf, 0x9c, 0x74, 0x89 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.CopyMem – CopyMem() with unsupported Width from profile returns EFI\_INVALID\_PARAMETER. | 1. Call CopyMem() with unsupported Width from profile. The return code should be EFI\_INVALID\_PARAMETER. |

### Map()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.1.10.1 | 0xb5eadff4, 0x6bbc, 0x45a2, 0xb9, 0x05, 0x85, 0x49, 0x78, 0xf3, 0xa6, 0x27 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Map – Map with **EfiPciOperationBusMasterRead** returns EFI\_SUCCESS. | 1. Allocate memory to Buffer.  2. Call Map() with **EfiPciOperationBusMasterRead** to map the address of Buffer to device address. The return code should be EFI\_SUCCESS. |
| 5.8.1.10.2 | 0x93950131, 0x0bc3, 0x429d, 0xad, 0x2d, 0x10, 0x47, 0x70, 0x76, 0x6c, 0xce | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Map – Map with **EfiPciOperationBusMasterRead** returns non-0 NumberOfBytes. | 1. Allocate memory to Buffer.  2. Call Map() with **EfiPciOperationBusMasterRead** to map the address of Buffer to device address. The return value of NumberOfBytes should not be 0. |
| 5.8.1.10.3 | 0x1a041b96, 0x79ea, 0x4732, 0xb9, 0xaa, 0x1c, 0xd4, 0x3b, 0x8c, 0x36, 0xcc | [DELETED] |  |
| 5.8.1.10.4 | 0x11e33211, 0xbc86, 0x4d69, 0xb9, 0xdf, 0x2d, 0x0a, 0xb5, 0xa0, 0x94, 0x46 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Map – Map() with **EfiPciOperationBusMasterRead**64 returns EFI\_SUCCESS. | 1. Allocate memory to Buffer.  2. Call Map() with **EfiPciOperationBusMasterRead**64 to map the address of Buffer to device address. The return code should be EFI\_SUCCESS. |
| 5.8.1.10.5 | 0x42e6a8c6, 0x0b28, 0x422d, 0xae, 0x3d, 0x86, 0x4d, 0xbf, 0x7b, 0x55, 0xee | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Map – Map() with **EfiPciOperationBusMasterRead**64 returns non-0 NumberOfBytes. | 1. Allocate memory to Buffer.  2. Call Map() with **EfiPciOperationBusMasterRead**64 to map the address of Buffer to device address. The return value of NumberOfBytes should not be 0. |
| 5.8.1.10.6 | 0x84f186ad, 0x3c1e, 0x46c4, 0x95, 0x52, 0xff, 0xd9, 0xdc, 0xbf, 0x80, 0x9d | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Map – After Map() with **EfiPciOperationBusMasterRead**64, the data read from device address is the same as original data. | 1. Allocate memory to Buffer.  2. Call Map() with **EfiPciOperationBusMasterRead**64 to map the address of Buffer to device address. The data read from device address must be the same as original data. |
| 5.8.1.10.7 | 0xe10594a2, 0xfd97, 0x4383, 0x82, 0x5c, 0x62, 0x14, 0x54, 0x62, 0xd9, 0x5e | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Map – Map() with **EfiPciOperationBusMasterWrite** returns EFI\_SUCCESS. | 1. Allocate memory to Buffer.  2. Call Map() with **EfiPciOperationBusMasterWrite** to map the address of Buffer to device address. The return code should be EFI\_SUCCESS. |
| 5.8.1.10.8 | 0x07e366fc, 0x5d2e, 0x474f, 0xba, 0xd3, 0xf8, 0xe4, 0x0a, 0x50, 0xf1, 0xd9 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Map – Map() with **EfiPciOperationBusMasterWrite** returns non-0 NumberOfBytes. | 1. Allocate memory to Buffer.  2. Call Map() with **EfiPciOperationBusMasterWrite** to map the address of Buffer to device address. The return value of NumberOfBytes should not be 0. |
| 5.8.1.10.9 | 0xbceb0ddc, 0x1145, 0x4fcd, 0x89, 0x1c, 0x53, 0x2f, 0x71, 0xb1, 0xf4, 0xe7 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Map – Map() with **EfiPciOperationBusMasterWrite** does not change data in host address. | 1. Allocate memory to Buffer.  2. Call Map() with **EfiPciOperationBusMasterWrite** to map the address of Buffer to device address. Data in Buffer should not be changed. |
| 5.8.1.10.10 | 0x5288b979, 0x9a17, 0x474a, 0xaf, 0xa0, 0x68, 0x61, 0x88, 0x48, 0xb3, 0xc1 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Map – Map() with **EfiPciOperationBusMasterWrite**64 returns EFI\_SUCCESS. | 1. Allocate memory to Buffer.  2. Call Map() with **EfiPciOperationBusMasterWrite**64 to map the address of Buffer to device address. The return code should be EFI\_SUCCESS. |
| 5.8.1.10.11 | 0x65d95c94, 0xd3b9, 0x4e4b, 0x88, 0x38, 0x49, 0x96, 0x0d, 0xb8, 0xfb, 0x24 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Map – Map() with **EfiPciOperationBusMasterWrite** returns non-0 NumberOfBytes. | 1. Allocate memory to Buffer.  2. Call Map() with **EfiPciOperationBusMasterWrite**64 to map the address of Buffer to device address. The return value of NumberOfBytes should not be 0. |
| 5.8.1.10.12 | 0x29fc59bc, 0x9f0d, 0x463d, 0xb4, 0x4a, 0x5a, 0xd2, 0x2d, 0x11, 0xa2, 0x26 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Map – Map() with **EfiPciOperationBusMasterWrite**64 does not change data in host address. | 1. Allocate memory to Buffer.  2. Call Map() with **EfiPciOperationBusMasterWrite**64 to map the address of Buffer to device address. Data in Buffer should not be changed. |
| 5.8.1.10.13 | 0xb674ab5a, 0xc030, 0x4832, 0x9d, 0x69, 0xbb, 0x18, 0x27, 0xb3, 0x39, 0x8e | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Map – Map() with **EfiPciOperationBusMasterCommonBuffer** returns EFI\_SUCCESS. | 1. Call AllocateBuffer() to allocate memory to Buffer.  2. Call Map() with **EfiPciOperationBusMasterCommonBuffer** to map the address of Buffer to device address. The return code should be EFI\_SUCCESS. |
| 5.8.1.10.14 | 0xebb4be23, 0x25c7, 0x46ce, 0xb8, 0x52, 0xde, 0xc7, 0x18, 0x2a, 0xc2, 0x07 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Map – Map() with **EfiPciOperationBusMasterCommonBuffer** returns non-0 NumberOfBytes. | 1. Call AllocateBuffer() to allocate memory to Buffer.  2. Call Map() with **EfiPciOperationBusMasterCommonBuffer** to map the address of Buffer to device address. The return value of NumberOfBytes should not be 0. |
| 5.8.1.10.18 | 0x8120df74, 0xae1e, 0x47f9, 0xaa, 0x45, 0x8e, 0x70, 0xa7, 0xe3, 0x31, 0x19 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Map – Map() with **EfiPciOperationBusMasterCommonBuffer**64 returns EFI\_SUCCESS. | 1. Call AllocateBuffer() to allocate memory to Buffer.  2. Call Map() with **EfiPciOperationBusMasterCommonBuffer**64 to map the address of Buffer to device address. The return code should be EFI\_SUCCESS. |
| 5.8.1.10.19 | 0xb93854ce, 0x5237, 0x492f, 0xbd, 0x55, 0x27, 0xd3, 0x82, 0xc1, 0xce, 0x53 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Map – Map() with **EfiPciOperationBusMasterCommonBuffer**64 returns non-0 NumberOfBytes. | 1. Call AllocateBuffer() to allocate memory to Buffer.  2. Call Map() with **EfiPciOperationBusMasterCommonBuffer**64 to map the address of Buffer to device address. The return value of NumberOfBytes should not be 0. |
| 5.8.1.10.20 | 0x3ec7dc5b, 0x3c99, 0x47e1, 0x87, 0xff, 0xb2, 0x4d, 0x08, 0x95, 0x04, 0x96 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Map – After Map() with **EfiPciOperationBusMasterCommonBuffer**64, the data read from device address is the same as original data. | 1. Call AllocateBuffer() to allocate memory to Buffer.  2. Call Map() with **EfiPciOperationBusMasterCommonBuffer**64 to map the address of Buffer to device address. The data read from device address must be the same as original data. |
| 5.8.1.10.21 | 0xb4df6e6e, 0x4e30, 0x457e, 0xa1, 0xf8, 0x39, 0xf4, 0x52, 0xf6, 0x11, 0x2f | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Map – After Map() with **EfiPciOperationBusMasterCommonBuffer**64, the data in original host address remains in sync with mapped device address. | 1. Call AllocateBuffer() to allocate memory to Buffer.  2. Call Map() with **EfiPciOperationBusMasterCommonBuffer**64 to map the address of Buffer to device address.  3. Call BS.SetMem() to change contents of mapped device address. Data in host address should change also and be equal to data in device address. |
| 5.8.1.10.22 | 0xc4451e9d, 0x538e, 0x4cda, 0xa7, 0xa6, 0x0c, 0xa1, 0x50, 0x06, 0x03, 0x87 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Map – After Map() with **EfiPciOperationBusMasterCommonBuffer**64, the data in mapped device address remains in sync with original host address. | 1. Call AllocateBuffer() to allocate memory to Buffer.  2. Call Map() with **EfiPciOperationBusMasterCommonBuffer**64 to map the address of Buffer to device address.  3. Call BS.SetMem() to change contents of host address. Data in mapped device address should change also and be equal to data in device address. |
| 5.8.1.10.23 | 0xc79ed36f, 0xe0b3, 0x426c, 0x85, 0xc1, 0x7d, 0xfe, 0xb8, 0xcf, 0xdf, 0x07 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Map – Map() with invalid Operation as **EfiPciOperationMaximum** returns EFI\_INVALID\_PARAMETER. | 1. Call Map() with invalid Operation: **EfiPciOperationMaximum**. The return code should be EFI\_INVALID\_PARAMETER. |
| 5.8.1.10.24 | 0x04b07426, 0x3d17, 0x4f18, 0x8b, 0x1c, 0xbd, 0x59, 0xae, 0x99, 0xe5, 0xf8 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Map – Map() with invalid Operation as -1 returns EFI\_INVALID\_PARAMETER. | 1. Call Map() with invalid Operation: -1. The return code should be EFI\_INVALID\_PARAMETER. |
| 5.8.1.10.25 | 0xf8a42643, 0x912a, 0x4731, 0xb9, 0x04, 0x47, 0xbc, 0x87, 0x7f, 0xdd, 0xcf | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Map – Map() with HostAddress as NULL returns EFI\_INVALID\_PARAMETER. | 1. Call Map() with HostAddress as NULL. The return code should be EFI\_INVALID\_PARAMETER. |
| 5.8.1.10.26 | 0x13513dbf, 0xc4da, 0x4952, 0xa4, 0x37, 0x44, 0x22, 0x28, 0x13, 0xdb, 0xfd | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Map – Map() with NumberOfBytes as NULL returns EFI\_INVALID\_PARAMETER. | 1. Call Map() with NumberOfBytes as NULL. The return code should be EFI\_INVALID\_PARAMETER. |
| 5.8.1.10.27 | 0x8bfb7a69, 0xd816, 0x4315, 0xbe, 0x27, 0xe2, 0xa9, 0x03, 0x44, 0x69, 0x8e | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Map – Map() with DeviceAddress as NULL returns EFI\_INVALID\_PARAMETER. | 1. Call Map() with DeviceAddress as NULL. The return code should be EFI\_INVALID\_PARAMETER. |
| 5.8.1.10.28 | 0x6fe65b18, 0x7638, 0x4584, 0xb9, 0x5f, 0x90, 0x2c, 0x0f, 0x80, 0xf6, 0x9b | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Map – Map() with Mapping as NULL returns EFI\_INVALID\_PARAMETER. | 1. Call Map() with Mapping as NULL. The return code should be EFI\_INVALID\_PARAMETER. |
| 5.8.1.10.29 | 0xd6b631c7, 0xd459, 0x40cd, 0xa1, 0xca, 0x6d, 0x28, 0x7b, 0x61, 0xaa, 0xd9 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Map – Map() with **EfiPciOperationBusMasterCommonBuffer** and HostAddress + NumberofBytes > 4GB returns EFI\_UNSUPPORTED. | 1. Call Map() with HostAddress + NumberofBytes > 4GB. The return code should be EFI\_UNSUPPORTED. |
| 5.8.1.10.30 | 0x04030971, 0xedb2, 0x498b, 0x84, 0x94, 0xf0, 0x19, 0x24, 0x28, 0xd4, 0x14 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Map – Map() with **EfiPciOperationBusMasterCommonBuffer**64 and HostAddress + NumberofBytes > 4GB returns EFI\_UNSUPPORTED. | 1. Call Map() with HostAddress + NumberofBytes > 4GB. The return code should be EFI\_UNSUPPORTED. |

### Unmap()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.1.11.1 | 0xb4a084d7, 0x48de, 0x48de, 0x97, 0xa0, 0x27, 0x10, 0x07, 0x9f, 0xcc, 0x04 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Unmap – Unmap() with mapping value gotten from Map() of **EfiPciOperationBusMasterRead** returns EFI\_SUCCESS. | 1. Call Map() with **EfiPciOperationBusMasterRead** to map the address of Buffer to device address..  2. Call Unmap() to release resources of mapping. The return code should be EFI\_SUCCESS. |
| 5.8.1.11.2 | 0xa4ef56f6, 0x597b, 0x47a4, 0xa3, 0xed, 0x00, 0xba, 0x87, 0xcd, 0x47, 0xd8 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Unmap – Unmap() with mapping value gotten from Map() of **EfiPciOperationBusMasterRead** does not change contents in host address. | 1. Set specific value to Buffer.  2. Call Map() with **EfiPciOperationBusMasterRead** to map the address of Buffer to device address.  3. Call Unmap with mapping value gotten from Map(). The data in Buffer should remain unchanged. |
| 5.8.1.11.3 | 0xd211369e, 0x2b2d, 0x4d95, 0xa7, 0x30, 0x7c, 0x7c, 0xf5, 0xd6, 0xfc, 0x13 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Unmap – Unmap() with mapping value gotten from Map() of **EfiPciOperationBusMasterRead**64 returns EFI\_SUCCESS. | 1. Call Map() with **EfiPciOperationBusMasterRead**64 to map the address of Buffer to device address..  2. Call Unmap() to release resources of mapping. The return code should be EFI\_SUCCESS. |
| 5.8.1.11.4 | 0xa32ec004, 0x1e89, 0x4553, 0xac, 0x80, 0x9d, 0x3b, 0x14, 0xe6, 0x09, 0x49 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Unmap – Unmap() with mapping value gotten from Map() of **EfiPciOperationBusMasterRead**64 does not change contents in host address. | 1. Set specific value to Buffer.  2. Call Map() with **EfiPciOperationBusMasterRead**64 to map the address of Buffer to device address.  3. Call Unmap with mapping value gotten from Map(). The data in Buffer should remain unchanged. |
| 5.8.1.11.5 | 0x8a2ffff4, 0x186b, 0x4624, 0xa5, 0x4a, 0x1a, 0x8f, 0xaf, 0xe4, 0x06, 0x2a | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Unmap – Unmap() with mapping value gotten from Map() of **EfiPciOperationBusMasterWrite** returns EFI\_SUCCESS. | 1. Call Map() with **EfiPciOperationBusMasterWrite** to map the address of Buffer to device address.  2. Call Unmap() to release resources of mapping. The return code should be EFI\_SUCCESS. |
| 5.8.1.11.6 | 0x8874b727, 0x7a35, 0x4e6e, 0x96, 0x19, 0x7e, 0x5b, 0x22, 0xcb, 0x3f, 0xf8 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Unmap – Unmap() with mapping value gotten from Map() of **EfiPciOperationBusMasterWrite** does not change contents in host address. | 1. Set specific value to Buffer.  2. Call Map() with **EfiPciOperationBusMasterWrite** to map the address of Buffer to device address.  3. Call Unmap with mapping value gotten from Map(). The data in Buffer should remain unchanged. |
| 5.8.1.11.7 | 0xffd39873, 0xa3da, 0x49fd, 0xae, 0x87, 0x5c, 0x09, 0xb5, 0xa1, 0x01, 0x73 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Unmap – Unmap() with mapping value gotten from Map() of **EfiPciOperationBusMasterWrite** returns EFI\_SUCCESS. | 1. Call Map() with **EfiPciOperationBusMasterWrite** to map the address of Buffer to device address.  2. Call Unmap() to release resources of mapping. The return code should be EFI\_SUCCESS. |
| 5.8.1.11.8 | 0xd8eedc25, 0xea92, 0x4d1b, 0x8f, 0xe7, 0x7c, 0xb1, 0x87, 0xb2, 0xc0, 0xa6 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Unmap – Unmap() with mapping value gotten from Map() of **EfiPciOperationBusMasterWrite**, does not change contents in host address. | 1. Set specific value to the Buffer.  2. Call Map() with **EfiPciOperationBusMasterWrite** to map the address of the Buffer to the device address.  3. Call Unmap() with mapping value gotten from Map(). The data in the Buffer should remain unchanged. |
| 5.8.1.11.9 | 0xe543e036, 0x3948, 0x4773, 0xa8, 0x0e, 0x89, 0x2c, 0xd3, 0xcc, 0xf0, 0xdf | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Unmap – Unmap() with mapping value gotten from Map() of **EfiPciOperationBusMasterCommonBuffer** returns EFI\_SUCCESS. | 1. Call Map() with **EfiPciOperationBusMasterCommonBuffer** to map the address of the Buffer to the device address.  2. Call Unmap() to release resources of mapping. The return code should be EFI\_SUCCESS. |
| 5.8.1.11.10 | 0xd2368593, 0x122a, 0x41e7, 0x83, 0x34, 0x65, 0x7e, 0x78, 0xed, 0x12, 0xbc | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Unmap – Unmap() with mapping value gotten from Map() of **EfiPciOperationBusMasterCommonBuffer** does not change contents in host address. | 1. Call AllocateBuffer() to allocate memory to the Buffer.  2. Call Map() with **EfiPciOperationBusMasterCommonBuffer** to map the address of the Buffer to the device address.  3. Call Unmap() with mapping value gotten from Map(). The data in the Buffer should remain unchanged. |
| 5.8.1.11.11 | 0x9356285b, 0x21b2, 0x40a3, 0x95, 0xed, 0xd6, 0xfe, 0x27, 0x5a, 0x2b, 0xba | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Unmap – Unmap() with mapping value gotten from Map() of **EfiPciOperationBusMasterCommonBuffer**64 returns EFI\_SUCCESS. | 1. Call Map() with **EfiPciOperationBusMasterCommonBuffer**64 to map the address of the Buffer to the device address.  2. Call Unmap() to release resources of mapping. The return code should be EFI\_SUCCESS. |
| 5.8.1.11.12 | 0x0c44017c, 0x078d, 0x475c, 0x90, 0x0c, 0x4a, 0x36, 0xe6, 0x8b, 0x72, 0x04 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Unmap – Unmap() with mapping value gotten from Map() of **EfiPciOperationBusMasterCommonBuffer**64 does not change contents in host address | 1. Call AllocateBuffer() to allocate memory to the Buffer.  2. Call Map() with **EfiPciOperationBusMasterCommonBuffer**64 to map the address of the Buffer to the device address.  3. Call Unmap with mapping value gotten from Map(). The data in the Buffer should remain unchanged. |

### AllocateBuffer()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.1.12.1 | 0x58a99166, 0xfdbe, 0x4963, 0xb9, 0x56, 0x00, 0x4f, 0x97, 0xcc, 0xe5, 0x20 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.AllocateBuffer – AllocateBuffer() with valid parameter returns EFI\_SUCCESS. | 1. Call AllocateBuffer() with valid parameter. The return code should be EFI\_SUCCESS. |
| 5.8.1.12.2 | 0x193efb14, 0x0c2a, 0x494d, 0xa2, 0xfc, 0xe1, 0x28, 0xb0, 0xe7, 0xb6, 0x5c | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.AllocateBuffer – AllocateBuffer() with invalid memory types -1 returns EFI\_INVALID\_PARAMETER. | 1. Call AllocateBuffer() with invalid memory types -1. The return code should be EFI\_INVALID\_PARAMETER |
| 5.8.1.12.3 | 0x08d81bb3, 0x1db0, 0x4ce3, 0x8e, 0xe0, 0xa6, 0x7c, 0x46, 0xf1, 0xa8, 0x9b | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.AllocateBuffer – AllocateBuffer() with invalid memory types returns EFI\_INVALID\_PARAMETER. | 1. Call AllocateBuffer() with invalid memory types. The return code should be EFI\_INVALID\_PARAMETER |
| 5.8.1.12.4 | 0x66bd765c, 0x6b86, 0x4a29, 0xbe, 0x88, 0x10, 0xab, 0xfe, 0x5a, 0xef, 0xbd | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.AllocateBuffer – AllocateBuffer() with HostAddress as NULL returns EFI\_INVALID\_PARAMETER. | 1. Call AllocateBuffer() with HostAddress as NULL. The return code should be EFI\_INVALID\_PARAMETER |
| 5.8.1.12.5 | 0xf2e8d30e, 0x40d8, 0x4823, 0x97, 0xb2, 0x08, 0x32, 0x11, 0x9f, 0x78, 0xd3 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.AllocateBuffer – AllocateBuffer() with unsupported Attributes returns EFI\_UNSUPPORTED. | 1. Call AllocateBuffer() with unsupported Attributes. The return code should be EFI\_UNSUPPORTED. |

### FreeBuffer()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.1.13.1 | 0xf2ec6740, 0x6416, 0x4890, 0xaf, 0xe6, 0xad, 0x67, 0x91, 0xf0, 0x22, 0xaf | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.FreeBuffer – FreeBuffer() with valid parameter returns EFI\_SUCCESS. | 1. Call AllocateBuffer() to allocate memory to buffer.  2. Call FreeBuffer() to free buffer memory. The return code should be EFI\_SUCCESS. |

### Flush()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.1.14.1 | 0x8ce74cd6, 0x0409,  0x4513,  0x98, 0xdd,  0x3d, 0x0f,  0x96, 0x97,  0x4f, 0xe8 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Flush – Flush() with valid parameter returns EFI\_SUCCESS. | 1. Call Flush() with valid parameter. The return code should be EFI\_SUCCESS. |

### 

### GetAttributes()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.1.15.1 | 0x8e661c40, 0xf56f, 0x4ce8, 0x8e, 0x7e, 0xf4, 0x07, 0x28, 0x57, 0xf9, 0x5b | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.GetAttributes – GetAttributes() to get current attributes and supported attributes returns EFI\_SUCCESS. | 1. Call GetAttributes() to get current attributes and supported attributes. The return code should be EFI\_SUCCESS. |
| 5.8.1.15.2 | 0x54d94c0e, 0x70d7, 0x4a7a, 0x9e, 0x81, 0xf5, 0xb1, 0x63, 0x05, 0x93, 0xbe | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.GetAttributes – Current attributes must within Supported attributes. | 1. Call GetAttributes() to get current attributes and supported attributes.  2. Current attributes must within Supported attributes. |
| 5.8.1.15.3 | 0x727cabec, 0x1a1b, 0x4e9d, 0xb1, 0xde, 0x3b, 0x3e, 0xda, 0x55, 0x84, 0x44 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.GetAttributes – GetAttributes() to only get current attributes returns EFI\_SUCCESS. | 1. Call GetAttributes() to only get current attributes. The return code should be EFI\_SUCCESS. |
| 5.8.1.15.4 | 0x66fb3230, 0xa799, 0x4efe, 0x89, 0xfa, 0xbf, 0x86, 0xdf, 0x23, 0xb0, 0xf7 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.GetAttributes – The second call of GetAttributes() returns the same current attributes as the first time. | 1. Call GetAttributes() to get current attributes and supported attributes.  2. Call GetAttributes() for the second time to only get current attributes. It should return the same current attribute as the first time. |
| 5.8.1.15.5 | 0x2176073a, 0x7dfa, 0x463a, 0xa2, 0xf1, 0xab, 0xba, 0x92, 0x42, 0xe0, 0xea | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.GetAttributes - GetAttributes() to only get supported attributes returns EFI\_SUCCESS. | 1. Call GetAttributes() to only get supported attributes. The return code should be EFI\_SUCCESS. |
| 5.8.1.15.6 | 0x5a5c6253, 0x1202, 0x4abd, 0x95, 0x6f, 0x23, 0x0a, 0x1b, 0x2f, 0x45, 0xc0 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.GetAttributes – The second call of GetAttributes() returns the same supported attributes as the first time. | 1. Call GetAttributes() to get current attributes and supported attributes.  2. Call GetAttributes() for the second time to only get supported attributes. It should return the same supported attribute as the first time. |
| 5.8.1.15.7 | 0x8f25b1c3, 0x4571, 0x4101, 0x95, 0xf1, 0x36, 0xc1, 0xe5, 0x83, 0xc0, 0x23 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.GetAttributes – GetAttributes() with both Attributes and Supports as NULL returns EFI\_INVALID\_PARAMETER. | 1. Call GetAttributes() with both Attributes and Supports as NULL. The return code should be EFI\_INVALID\_PARAMETER |

### SetAttributes()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.1.16.1 | 0xb9ee4bd9, 0x5a92, 0x4521, 0xbf, 0xaa, 0x80, 0x7f, 0x8b, 0x20, 0xac, 0xaa | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.SetAttributes – SetAttributes() to set supported attributes returns EFI\_SUCCESS. | 1. Call GetAttributes() to get supported attributes.  2. Call SetAttributes() to set supported attributes. The return code should be EFI\_SUCCESS. |
| 5.8.1.16.2 | 0x1dbb0bee, 0x7ebf, 0x4a3f, 0xa8, 0xaf, 0xb8, 0x24, 0x76, 0x29, 0xd6, 0x7c | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.SetAttributes – SetAttributes() to set supported attributes changes current attributes as expected. | 1. Call GetAttributes() to get supported attributes.  2. Call SetAttributes() to set supported attributes.  3. Call GetAttributes() to get current attributes. The supported attributes bits should be set. |
| 5.8.1.16.3 | 0x697e0d03, 0xca02, 0x4a21, 0x87, 0xf6, 0xd5, 0xd5, 0xeb, 0xb3, 0xab, 0xdb | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.SetAttributes – SetAttributes() to set supported attributes that require a resource returns EFI\_SUCCESS. | 1. Call GetAttributes() to get supported attributes.  2. Call SetAttributes() to set MEMORY\_WRITE\_COMBINE, MEMORY\_CACHED or MEMORY\_DISABLE if they are supported. The return code should be EFI\_SUCCESS. |
| 5.8.1.16.4 | 0x1f27d46e, 0x53b4, 0x4687, 0xaa, 0x9a, 0x5d, 0x46, 0xfb, 0x05, 0xa3, 0x65 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.SetAttributes – SetAttributes() to set supported attributes changes current attributes as expected. | 1. Call GetAttributes() to get supported attributes.  2. Call SetAttributes() to set MEMORY\_WRITE\_COMBINE, MEMORY\_CACHED or MEMORY\_DISABLE if they are supported.  3. Call GetAttributes() to get current attributes. The supported attribute bits specified by SetAttributes() should be set. |
| 5.8.1.16.5 | 0x405511dd, 0x38b4, 0x4aed, 0x9a, 0x7e, 0x18, 0xaa, 0xd1, 0x21, 0x67, 0x68 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.SetAttributes – SetAttributes() with unsupported attributes that do not need resources returns EFI\_UNSUPPORTED. | 1. Call GetAttributes() to get current attributes and supported attributes.  2. Call SetAttributes() with unsupported attributes that do not need resources. The return code should be EFI\_UNSUPPORTED. |
| 5.8.1.16.6 | 0x0150f584, 0x775b, 0x422d, 0xb3, 0xd7, 0xb8, 0x0d, 0x34, 0x56, 0x26, 0x47 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.SetAttributes – SetAttributes() with unsupported attributes that need resources returns EFI\_UNSUPPORTED. | 1. Call GetAttributes() to get current attributes and supported attributes.  2. Call SetAttributes() with unsupported attributes that need resources. The return code should be EFI\_UNSUPPORTED. |
| 5.8.1.16.7 | 0xdbf3baef, 0x35e9, 0x4d10, 0x8a, 0xbb, 0xcc, 0xca, 0x70, 0x5e, 0x99, 0x86 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.SetAttributes – SetAttributes() with unsupported attributes does not change current attributes. | 1. Call GetAttributes() to get current attributes and supported attributes.  2. Call SetAttributes() with unsupported attributes that not resource.  3. Call GetAttributes() to get current attributes. It should remain unchanged. |
| 5.8.1.16.8 | 0x186fee52, 0x7b8d, 0x4589, 0x8d, 0x87, 0x8e, 0x4f, 0x6b, 0x67, 0x9c, 0x6c | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.SetAttributes – SetAttributes() with EFI\_PCI\_ATTRIBUTE\_MEMORY\_WRITE\_COMBINE and ResourceBase as NULL returns EFI\_INVALID\_PARAMETER. | 1. Call SetAttributes() with EFI\_PCI\_ATTRIBUTE\_MEMORY\_WRITE\_COMBINE and ResourceBase as NULL. The return code should be EFI\_INVALID\_PARAMETER. |
| 5.8.1.16.9 | 0x5a06217c, 0xcbf1, 0x4faa, 0x94, 0x04, 0x3b, 0xaf, 0x39, 0x6d, 0x04, 0x1d | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.SetAttributes – SetAttributes() with EFI\_PCI\_ATTRIBUTE\_MEMORY\_WRITE\_COMBINE and ResourceLength as NULL returns EFI\_INVALID\_PARAMETER. | 1. Call SetAttributes() with EFI\_PCI\_ATTRIBUTE\_MEMORY\_WRITE\_COMBINE and ResourceLength as NULL. The return code should be EFI\_INVALID\_PARAMETER. |
| 5.8.1.16.10 | 0x7d1e8194, 0x0732, 0x4ca0, 0xac, 0x50, 0xdb, 0x62, 0x18, 0xe0, 0x69, 0xdd | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.SetAttributes – SetAttributes() with EFI\_PCI\_ATTRIBUTE\_MEMORY\_CACHED and ResourceBase as NULL returns EFI\_INVALID\_PARAMETER. | 1. Call SetAttributes() with EFI\_PCI\_ATTRIBUTE\_MEMORY\_CACHED and ResourceBase as NULL. The return code should be EFI\_INVALID\_PARAMETER. |
| 5.8.1.16.11 | 0x037c66ae, 0x79a4, 0x4909, 0x93, 0xa4, 0xa6, 0xb7, 0xb8, 0xee, 0x58, 0xd6 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.SetAttributes – SetAttributes() with EFI\_PCI\_ATTRIBUTE\_MEMORY\_CACHED and ResourceLength as NULL returns EFI\_INVALID\_PARAMETER. | 1. Call SetAttributes() with EFI\_PCI\_ATTRIBUTE\_MEMORY\_CACHED and ResourceLength as NULL. The return code should be EFI\_INVALID\_PARAMETER. |
| 5.8.1.16.12 | 0x117de9ad, 0xbc79, 0x49c2, 0xa7, 0x0f, 0x80, 0xc8, 0x80, 0x48, 0x6c, 0x91 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.SetAttributes – SetAttributes() with EFI\_PCI\_ATTRIBUTE\_MEMORY\_DISABLE and ResourceBase as NULL returns EFI\_INVALID\_PARAMETER. | 1. Call SetAttributes() with EFI\_PCI\_ATTRIBUTE\_MEMORY\_DISABLE and ResourceBase as NULL. The return code should be EFI\_INVALID\_PARAMETER. |
| 5.8.1.16.13 | 0x363d5f12, 0x4c82, 0x4117, 0xa7, 0x6c, 0xc3, 0xd3, 0x70, 0x8f, 0xdb, 0xda | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.SetAttributes – SetAttributes() with EFI\_PCI\_ATTRIBUTE\_MEMORY\_DISABLE and ResourceLength as NULL returns EFI\_INVALID\_PARAMETER. | 1. Call SetAttributes() with EFI\_PCI\_ATTRIBUTE\_MEMORY\_DISABLE and ResourceLength as NULL. The return code should be EFI\_INVALID\_PARAMETER. |

### Configuration()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.1.17.1 | 0xe65742bb, 0x7693, 0x4de1, 0xb0, 0x7b, 0x74, 0xfd, 0x64, 0x43, 0x6b, 0xf5 | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Configuration – Configuration() to get the resource list returns EFI\_SUCCESS. | 1. Call Configuration() to get the resource list. The return code should be EFI\_SUCCESS. |
| 5.8.1.17.2 | 0xa5982933, 0x6b43, 0x4947, 0xb0, 0x29, 0xa8, 0xd5, 0x66, 0x72, 0xaa, 0xce | EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL.Configuration – Resource returned by Configuration points to a valid ACPI 2.0 QWord descriptor. | 1. Call Configuration() to get the Resource list. The return Resource should be a valid ACPI 2.0 QWord descriptor. |

## EFI\_PCI\_IO\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_PCI\_ IO\_ PROTOCOL Section.

Configuration

Some checkpoints in the EFI\_PCI\_IO\_PROTOCOL test are device related. If the user needs to check the protocol on the specified device, the related profile needs to be updated to provide the specified information about this device.

For the format of the profile, please refer to

��EFI\_PCI\_IO\_PROTOCOL Test Profile.

### PollMem()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.2.1.1 | 0xaef16eb4, 0x40ad, 0x4dcf, 0x8c, 0x57, 0x20, 0x92, 0xa7, 0x43, 0xa9, 0x78 | EFI\_PCI\_IO\_PROTOCOL.PollMem - PollMem() with valid value returns EFI\_SUCCESS. | 1. Call Mem.Write() to set the Alternate Value on the address.  2. Start a 3 second timer event. The event handler writes the Target Value to the address.  3. Call PollMem() for the Target Value with Delay as 5 seconds on the address -- PollMem() must return EFI\_SUCCESS with Result as the Target Value. |
| 5.8.2.1.2 | 0x6e8a67fe, 0x4ad1, 0x4317, 0xa6, 0xfe, 0x76, 0x88, 0x02, 0x49, 0x0f, 0xbc | EFI\_PCI\_IO\_PROTOCOL.PollMem - PollMem() with valid value again returns EFI\_SUCCESS. | 1. Call Mem.Write() to set the Alternate Value on the address.  2. Start a 3 second timer event. The event handler writes the Target Value to the address.  3. Call PollMem() for the Target Value with Delay as 5 seconds on the address -- PollMem() must return EFI\_SUCCESS with Result as the Target Value.  4. Call PollMem() for the Target Value again on the address. -- PollMem() must return EFI\_SUCCESS with Result as the expected value. |
| 5.8.2.1.3 | 0x3b2cfc3e, 0xf167, 0x4c1f, 0x99, 0x8e, 0x2b, 0xca, 0x0b, 0x17, 0x6d, 0x39 | EFI\_PCI\_IO\_PROTOCOL.PollMem - PollMem() with delay equals 0 and invalid destination address, returns EFI\_SUCCESS. | 1. Call Mem.Write() to set the Alternate Value on the address.  2. PollMem() for the Target Value on the address with Delay as 0. -- PollMem() must return EFI\_SUCCESS, with Result as the Alternate Value. |
| 5.8.2.1.4 | 0x600c99fb, 0x31d0, 0x4a94, 0x8e, 0xa3, 0xbd, 0x59, 0x54, 0xd0, 0xa5, 0x2b | EFI\_PCI\_IO\_PROTOCOL.PollMem - PollMem() with 5 seconds delay and invalid destination address, returns EFI\_TIMEOUT. | 1. Call Mem.Write() to set the Alternate Value on the address.  2. PollMem() for the Target on the address with Delay as 5 seconds. – PollMem() must return EFI\_TIMEOUT, with Result as the Alternate Value. |
| 5.8.2.1.5 | 0x5a9e8b1e, 0xdc0d, 0x461f, 0x9f, 0xd5, 0xf4, 0x4c, 0xb9, 0x6e, 0xff, 0xfa | EFI\_PCI\_IO\_PROTOCOL.PollMem - With Width as **EfiPciWidthMaximum**, the return status is EFI\_INVALID\_PARAMETER. | 1. Call PollMem() with Width as **EfiPciWidthMaximum**. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.1.6 | 0x3c29ad4d, 0x8bad, 0x4862, 0xab, 0x3a, 0x9b, 0xde, 0xee, 0xd6, 0x2e, 0x19 | EFI\_PCI\_IO\_PROTOCOL.PollMem - With Width as **EfiPciWidthFifoUintX**, the return status is EFI\_INVALID\_PARAMETER. | 1. Call PollMem() with Width as **EfiPciWidthFifoUintX**. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.1.7 | 0xb9b9ebdc, 0x09e9, 0x4cc6, 0xaf, 0x45, 0xf1, 0xae, 0x28, 0x06, 0x17, 0x70 | EFI\_PCI\_IO\_PROTOCOL.PollMem - With Width as **EfiPciWidthFillUintX**, the return status is EFI\_INVALID\_PARAMETER. | 1. Call PollMem() with Width as **EfiPciWidthFillUintX**. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.1.8 | 0x9007c300, 0x0782, 0x4f3e, 0xae, 0x40, 0xd5, 0x9d, 0x95, 0xce, 0x55, 0xf6 | EFI\_PCI\_IO\_PROTOCOL.PollMem - With Width as -1, the return status is EFI\_INVALID\_PARAMETER | 1. Call PollMem() with Width as -1. The return status must be EFI\_INVALID\_PARAMETER |
| 5.8.2.1.9 | 0xdb14a663, 0x3a39, 0x4cf1, 0x90, 0xe6, 0x7a, 0xfe, 0x00, 0x6c, 0x66, 0xe2 | EFI\_PCI\_IO\_PROTOCOL.PollMem - With Result as NULL, the return status is EFI\_INVALID\_PARAMETER. | 1. Call PollMem() with Result as NULL. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.1.10 | 0x47e2f242, 0xf876, 0x46ed, 0x9c, 0x91, 0x82, 0xd6, 0xd6, 0xb6, 0x7d, 0xb5 | EFI\_PCI\_IO\_PROTOCOL.PollMem - With Offset beyond the range of BAR, the return status is EFI\_UNSUPPORTED. | 1. Call PollMem() with Offset beyond the range of BAR. The return status must be EFI\_UNSUPPORTED. |
| 5.8.2.1.11 | 0x02b6ac92, 0x4984, 0x42d8, 0xab, 0xda, 0xb1, 0x87, 0x8e, 0xa0, 0xd6, 0xc8 | EFI\_PCI\_IO\_PROTOCOL.PollMem - With invalid BAR Index the return status is EFI\_UNSUPPORTED. | 1. Call PollMem() with invalid BAR Index. The return status must be EFI\_UNSUPPORTED. |
| 5.8.2.1.12 | 0x668ccc4e, 0xb0b2, 0x4980, 0xab, 0x43, 0xff, 0xfd, 0x11, 0x83, 0x91, 0x75 | EFI\_PCI\_IO\_PROTOCOL.PollMem - With Io BAR Index the return status is EFI\_UNSUPPORTED. | 1. Call PollMem() with Io BAR Index. The return status must be EFI\_UNSUPPORTED. |
| 5.8.2.1.13 | 0x47a63a3d, 0xa134, 0x4a04, 0xb0, 0xd2, 0x10, 0xf1, 0x64, 0x88, 0xb0, 0xfb | EFI\_PCI\_IO\_PROTOCOL.PollMem - With invalid Width the return status is EFI\_INVALID\_PARAMETER. | 1. Call PollMem() with invalid Width. The return status must be EFI\_INVALID\_PARAMETER. |

### PollIo()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.2.2.1 | 0x6dfeb4fd, 0xdd98, 0x40db, 0x8e, 0x42, 0x67, 0x8a, 0xfb, 0x92, 0x6a, 0xe9 | EFI\_PCI\_IO\_PROTOCOL.PollIo - PollIo() with valid value returns EFI\_SUCCESS. | 1. Call Mem.Write() to set the Alternate Value on the address.  2. Start a 3 second timer event. The event handler writes the Target Value to the address.  3. Call PollMem() for the Target Value with Delay as 5 seconds on the address -- PollMem() must return EFI\_SUCCESS with Result as the Target Value. |
| 5.8.2.2.2 | 0x427eb5db, 0x6e41, 0x4b01, 0xad, 0xb0, 0x31, 0xff, 0xd9, 0x99, 0x6a, 0x5b | EFI\_PCI\_IO\_PROTOCOL.PollIo - PollIo() with valid value again returns EFI\_SUCCESS. | 1. Call Io.Write() to set the Alternate Value on the address.  2. Start a 3 second timer event. The event handler writes the Target Value to the address.  3. Call PollIo() for the Target Value with Delay as 5 seconds on the address -- PollIo() must return EFI\_SUCCESS with Result as the Target Value.  4. Call PollIo() for the Target Value again on the address. -- PollIo() must return EFI\_SUCCESS with Result as the expected value |
| 5.8.2.2.3 | 0xdff400ef, 0x9e72, 0x448f, 0xad, 0x6b, 0xb1, 0x34, 0x25, 0x45, 0xc7, 0x02 | EFI\_PCI\_IO\_PROTOCOL.PollIo - PollIo() with delay equal 0 and invalid destination address returns EFI\_SUCCESS. | 1. Call Io.Write() to set the Alternate Value on the address.  2. PollIo() for the Target Value on the address with Delay as 0. -- PollIo() must return EFI\_SUCCESS, with Result as the Alternate Value. |
| 5.8.2.2.4 | 0x6071974c, 0x35c0, 0x4599, 0xa6, 0x53, 0xe4, 0xbe, 0xc7, 0x34, 0xf7, 0x2c | EFI\_PCI\_IO\_PROTOCOL.PollIo - PollIo() with 5 seconds delay and invalid destination address returns EFI\_TIMEOUT. | 1. Call Io.Write() to set the Alternate Value on the address.  2. PollIo() for the Target on the address with Delay as 5 seconds. – PollIo() must return EFI\_TIMEOUT, with Result as the Alternate Value. |
| 5.8.2.2.5 | 0xc113fe3f, 0x0fae, 0x4266, 0xbf, 0xb4, 0xfd, 0x41, 0xed, 0x41, 0xea, 0x39 | EFI\_PCI\_IO\_PROTOCOL.PollIo - With Width as **EfiPciWidthMaximum** the return status is EFI\_INVALID\_PARAMETER. | 1. Call PollIo() with Width as **EfiPciWidthMaximum**. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.2.6 | 0x11466e1f, 0xd7e6, 0x4622, 0x84, 0x73, 0xfd, 0x57, 0xbf, 0x2f, 0x8f, 0x8e | EFI\_PCI\_IO\_PROTOCOL.PollIo - With Width as **EfiPciWidthFifoUintX** the return status is EFI\_INVALID\_PARAMETER. | 1. Call PollIo() with Width as **EfiPciWidthFifoUintX**. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.2.7 | 0x251113eb, 0x968c, 0x4c70, 0xbf, 0xa0, 0x0d, 0xf6, 0x74, 0x7f, 0xfa, 0x9a | EFI\_PCI\_IO\_PROTOCOL.PollIo - With Width as **EfiPciWidthFillUintX** the return status is EFI\_INVALID\_PARAMETER. | 1. Call PollIo() with Width as **EfiPciWidthFillUintX**. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.2.8 | 0xc6e532e8, 0xacc8, 0x4d48, 0x84, 0x69, 0xfd, 0xb0, 0xc1, 0xe0, 0xe5, 0x34 | EFI\_PCI\_IO\_PROTOCOL.PollIo - With Width as -1 the return status is EFI\_INVALID\_PARAMETER. | 1. Call PollIo() with Width as -1. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.2.9 | 0xdd0e653a, 0x9da8, 0x4f32, 0x9d, 0x0a, 0xe3, 0x29, 0xe1, 0x17, 0x19, 0x0e | EFI\_PCI\_IO\_PROTOCOL.PollIo - With Result as NULL the return status is EFI\_INVALID\_PARAMETER. | 1. Call PollIo() with Result as NULL. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.2.10 | 0xda044ef5, 0xe73b, 0x415c, 0xaf, 0x03, 0xaf, 0x3c, 0xb0, 0x00, 0x3f, 0x45 | EFI\_PCI\_IO\_PROTOCOL.PollIo - With Offset beyond the range of BAR the return status is EFI\_UNSUPPORTED. | 1. Call PollIo() with Offset beyond the range of BAR. The return status must be EFI\_UNSUPPORTED. |
| 5.8.2.2.11 | 0x0929e753, 0x7659, 0x4b6b, 0x80, 0x1a, 0x8b, 0xd6, 0xb6, 0x37, 0x4d, 0xf6 | EFI\_PCI\_IO\_PROTOCOL.PollIo - With invalid BAR Index the return status is EFI\_UNSUPPORTED. | 1. Call PollIo() with invalid BAR Index. The return status must be EFI\_UNSUPPORTED. |
| 5.8.2.2.12 | 0x64e878f6, 0xa53d, 0x4b4f, 0xa3, 0xca, 0x18, 0x9e, 0x37, 0x23, 0x4a, 0x99 | EFI\_PCI\_IO\_PROTOCOL.PollIo - With Mem BAR Index the return status is EFI\_UNSUPPORTED. | 1. Call PollIo() with Mem BAR Index. The return status must be EFI\_UNSUPPORTED. |
| 5.8.2.2.13 | 0xf2e6563e, 0x0881, 0x4efc, 0xae, 0x69, 0x6d, 0x08, 0xdf, 0x1c, 0xb2, 0x80 | EFI\_PCI\_IO\_PROTOCOL.PollIo - With invalid Width the return status is EFI\_INVALID\_PARAMETER. | 1. Call PollIo() with invalid Width. The return status must be EFI\_INVALID\_PARAMETER. |

### Mem.Read()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.2.3.1 | 0xa52d8d69, 0x77cb, 0x4012, 0x9d, 0x3f, 0xfa, 0x19, 0xe3, 0x2f, 0x17, 0x6c | EFI\_PCI\_IO\_PROTOCOL.Mem.Read - Mem.Read() reads data out and returns EFI\_SUCCESS. | 1. Allocate a buffer that matches the size of the address range.  2. Call Mem.Read() to fill in the buffer with the predefined data units  The return status should be EFI\_SUCCESS. |
| 5.8.2.3.2 | 0x44e5c09e, 0xce91, 0x419d, 0xbe, 0xaf, 0xd6, 0x60, 0x73, 0xdf, 0x4e, 0xe3 | EFI\_PCI\_IO\_PROTOCOL.Mem.Read - Mem.Read() read out the data from the address space and returns EFI\_SUCCESS. | 1. Allocate a buffer that matches the size of the address range.  2. Call Mem.Read() to fill in the buffer with the predefined data units.  3. Call Mem.Write() to write the buffer into the address range.  4. Call Mem.Read() to read out the data in destination address range.  The return status should be EFI\_SUCCESS. |
| 5.8.2.3.3 | 0x8ac05fc7, 0x0378, 0x4b5e, 0xba, 0x48, 0xb8, 0x53, 0x3d, 0x9e, 0xf2, 0x4c | EFI\_PCI\_IO\_PROTOCOL.Mem.Read - The data read out is the same as that written in. | 1. Allocate a buffer that matches the size of the address range.  2. Call Mem.Read() to fill in the buffer with the predefined data units.  3. Call Mem.Write() to write the buffer into the address range.  4. Call Mem.Read() to read out the data in destination address range.  5. Compare the data read out with data written in. |
| 5.8.2.3.4 | 0xca3a1290, 0x652f, 0x490c, 0x8a, 0x3f, 0xea, 0x94, 0x45, 0xa4, 0xd3, 0x81 | EFI\_PCI\_IO\_PROTOCOL.Mem.Read - Mem.Read() reads out the data with **EfiPciIoWidthFifoX** returns EFI\_SUCCESS. | 1. Allocate a buffer that matches the size of the address range.  2. Call Mem.Read() to fill in the buffer with the predefined data units.  3. Call Mem.Write() to write the buffer into the address range.  4. Call Mem.Read() to read out the data using **EfiPciIoWidthFifoUintX**.  The return status should be EFI\_SUCCESS. |
| 5.8.2.3.5 | 0x99bb7423, 0xa29c, 0x442e, 0x9a, 0x29, 0x7b, 0xf8, 0xf1, 0x88, 0xa8, 0x7e | EFI\_PCI\_IO\_PROTOCOL.Mem.Read - With **EfiPciIoWidthFifoX**, the data read out is the same as the first data unit. | 1. Allocate a buffer that matches the size of the address range.  2. Call Mem.Read() to fill in the buffer with the predefined data units.  3. Call Mem.Write() to write the buffer into the address range.  4. Call Mem.Read() to read out the data using **EfiPciIoWidthFifoUintX**.  5. Compare the each data unit in the buffer with the data at the Starting Address of the address range. |
| 5.8.2.3.6 | 0x4b9fef07, 0x3a4f, 0x40a0, 0xad, 0x43, 0xd1, 0x6a, 0x59, 0x8c, 0x22, 0x04 | EFI\_PCI\_IO\_PROTOCOL.Mem.Read - Mem.Read() reads out the data with **EfiPciIoWidthFillX** and returns EFI\_SUCCESS. | 1. Allocate a bufferthat matches the size of the address range.  2. Call Mem.Read() to fill in the buffer with the predefined data units.  3. Call Mem.Write() to write the buffer into the address range.  4. Call Mem.Read() to read out the data using **EfiPciIoWidthFillUintX**.  The return status should be EFI\_SUCCESS. |
| 5.8.2.3.7 | 0xd0bb89cc, 0x3838, 0x48bd, 0xb9, 0xd8, 0x1b, 0x8e, 0x3d, 0xef, 0x77, 0xd5 | EFI\_PCI\_IO\_PROTOCOL.Mem.Read - With **EfiPciIoWidthFillX**, the data read out from the first unit in buffer equals the last unit in the address space. | 1. Allocate a bufferthat matches the size of the address range.  2. Call Mem.Read() to fill in the buffer with the predefined data units.  3. Call Mem.Write() to write the buffer into the address range.  4. Call Mem.Read() to read out the data using **EfiPciIoWidthFillUintX**.  5. Compare the first data unit in the output buffer with the last data unit in the address range. Compare other data units in the output buffer with the original data. |
| 5.8.2.3.8 | 0xd282dcc9, 0x004f, 0x4733, 0xb2, 0xa6, 0xb5, 0x56, 0x6b, 0x4c, 0xaf, 0x91 | EFI\_PCI\_IO\_PROTOCOL.Mem.Read - With Width as **EfiPciIoWidthMaximum** the return status is EFI\_INVALID\_PARAMETER. | 1. Call Mem.Read() with Width as **EfiPciWidthMaximum**. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.3.9 | 0x927ad37d, 0x5ee5, 0x4d7a, 0x9f, 0x2e, 0x49, 0x9d, 0x7a, 0x49, 0x87, 0xb9 | EFI\_PCI\_IO\_PROTOCOL.Mem.Read - With invalid Width type -1 the return status is EFI\_INVALID\_PARAMETER. | 1. Call Mem.Read() with Width as -1. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.3.10 | 0x99d41dcf, 0x75ee, 0x48bf, 0xac, 0x3d, 0x86, 0xce, 0xe6, 0x67, 0x11, 0x1c | EFI\_PCI\_IO\_PROTOCOL.Mem.Read - With Buffer as NULL the return status is EFI\_INVALID\_PARAMETER. | 1. Call Mem.Read() with Buffer as NULL. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.3.11 | 0xa6d04a84, 0x2808, 0x48c7, 0xa0, 0x0b, 0xc2, 0xd6, 0xab, 0xad, 0xd5, 0x91 | EFI\_PCI\_IO\_PROTOCOL.Mem.Read - With address out of BAR range the return status is EFI\_UNSUPPORTED. | 1. Call Mem.Read() with address out of BAR range. The return status must be EFI\_UNSUPPORTED. |
| 5.8.2.3.12 | 0xe8417927, 0xe158, 0x4094, 0x90, 0xf6, 0x03, 0x15, 0xf7, 0x2f, 0x61, 0xdc | EFI\_PCI\_IO\_PROTOCOL.Mem.Read - With address out of BAR range the return status is EFI\_UNSUPPORTED. | 1. Call Mem.Read() with address out of BAR range. The return status must be EFI\_UNSUPPORTED. |
| 5.8.2.3.13 | 0x80720d1b, 0xa3dd, 0x465f, 0x8d, 0xe8, 0x9b, 0x6b, 0xb9, 0x64, 0x76, 0xda | EFI\_PCI\_IO\_PROTOCOL.Mem.Read - With invalid BAR Index the return status is EFI\_UNSUPPORTED. | 1. Call Mem.Read() with invalid BAR Index. The return status must be EFI\_UNSUPPORTED. |
| 5.8.2.3.14 | 0x3b9e11c1, 0x6fea, 0x4742, 0x81, 0xfd, 0xf2, 0xfb, 0xd6, 0x9c, 0xb6, 0xba | EFI\_PCI\_IO\_PROTOCOL.Mem.Read - With Io **Type** BAR the return status is EFI\_UNSUPPORTED. | 1. Call Mem.Read() with Io **Type** BAR. The return status must be EFI\_UNSUPPORTED. |
| 5.8.2.3.15 | 0xa043ffdf, 0x568b, 0x4128, 0x80, 0xf8, 0x61, 0x29, 0x0c, 0xd8, 0x8d, 0x57 | EFI\_PCI\_IO\_PROTOCOL.Mem.Read - With invalid Width the return status is EFI\_INVALID\_PARAMETER. | 1. Call Mem.Read() with invalid Width. The return status must be EFI\_INVALID\_PARAMETER. |

### Mem.Write()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.2.4.1 | 0x5847e586, 0x1f02, 0x466c, 0xa8, 0x33, 0x27, 0x23, 0x0d, 0x8d, 0xd9, 0xfd | EFI\_PCI\_IO\_PROTOCOL.Mem.Write - Mem.Write() writes data to the memory address space and returns EFI\_SUCCESS. | 1. Allocate a buffer that matches the size of the address range.  2. Call Mem.Read() to fill in the buffer with the predefined data units.  3. Call Mem.Write() to write the buffer into the address range  The return status should be EFI\_SUCCESS. |
| 5.8.2.4.2 | 0x6790de90, 0x56b2, 0x456e, 0x8e, 0x7a, 0xd1, 0x65, 0x77, 0xb9, 0xce, 0x39 | EFI\_PCI\_IO\_PROTOCOL.Mem.Write - The data read out is the same as that written in. | 1. Allocate a buffer that matches the size of the address range.  2. Call Mem.Read() to fill in the buffer with the predefined data units.  3. Call Mem.Write() to write the buffer into the address range.  4. Call Mem.Read() to read out the data in destination address range.  5. Compare the data read out with data written in. |
| 5.8.2.4.3 | 0x148a380b, 0xdbe0, 0x496b, 0xbd, 0x51, 0x56, 0xe6, 0xde, 0xcc, 0xf7, 0xca | EFI\_PCI\_IO\_PROTOCOL.Mem.Write - Mem.Write() writes the data with **EfiPciIoWidthFifoX** and returns EFI\_SUCCESS. | 1. Allocate a buffer that matches the size of the address range.  2. Call Mem.Read() to fill in the buffer with the predefined data units.  3. Call Mem.Write() to write to the starting address of address range using **EfiPciIoWidthFifoUintX**.  The return status should be EFI\_SUCCESS. |
| 5.8.2.4.4 | 0xf641e745, 0x9f3c, 0x42bf, 0x94, 0x23, 0x04, 0x20, 0x56, 0x46, 0x4b, 0x6b | EFI\_PCI\_IO\_PROTOCOL.Mem.Write - With **EfiPciIoWidthFifoX**, the first data unit is the same as the last data unit. | 1. Allocate a buffer that matches the size of the address range.  2. Call Mem.Read() to fill in the buffer with the predefined data units  3. Call Mem.Write() to write to the starting address of address range using **EfiPciIoWidthFifoUintX**.  4. Call Mem.Read() to read out the data using **EfiPciIoWidthFifoUintX**.  5. Compare the data in the starting address with the last data unit. Compare other data units with original data. |
| 5.8.2.4.5 | 0xbb3f0bad, 0x6680, 0x4aaa, 0xbe, 0x39, 0x70, 0xe4, 0x13, 0x02, 0xf8, 0x5d | EFI\_PCI\_IO\_PROTOCOL.Mem.Write - Mem.Write() writes the data with **EfiPciIoWidthFillX** and returns EFI\_SUCCESS. | 1. Allocate a buffer that matches the size of the address range.  2. Call Mem.Read() to fill in the buffer with the predefined data units.  3. Call Mem.Write() to write to address range using **EfiPciIoWidthFillUintX**.  The return status should be EFI\_SUCCESS. |
| 5.8.2.4.6 | 0x787dfda9, 0xcbfd, 0x4aae, 0x82, 0x98, 0xb1, 0xd4, 0x74, 0x15, 0x89, 0xb2 | EFI\_PCI\_IO\_PROTOCOL.Mem.Write - With **EfiPciIoWidthFillX**, all the data units read out are the same as the first data unit in the address space. | 1. Allocate a buffer that matches the size of the address range.  2. Call Mem.Read() to fill in the buffer with the predefined data units.  3. Call Mem.Write() to write to address range using **EfiPciIoWidthFillUintX**.  4. Call Mem.Read() to read out the data using **EfiPciIoWidthFillUintX**.  5. Compare all the data units with the first data unit. |
| 5.8.2.4.7 | 0x2d6920fd, 0x05a9, 0x480b, 0x8c, 0x74, 0x2a, 0xfc, 0x0f, 0xa7, 0x83, 0x3a | EFI\_PCI\_IO\_PROTOCOL.Mem.Write - Mem.Write() writes back the Data and returns EFI\_SUCCESS. | 1. Allocate a buffer that matches the size of the address range.  2. Call Mem.Read() to fill in the buffer with the predefined data units.  3. Call Mem.Write() to write the buffer into the address range.  4. Call Mem.Read() to read out the data in destination address range.  5. Call Mem.Write() to write the data back.  The return status should be EFI\_SUCCESS. |
| 5.8.2.4.8 | 0x4fe0f156, 0x0cb2, 0x464a, 0xb1, 0xbd, 0x23, 0x14, 0x9e, 0x3e, 0x09, 0x60 | EFI\_PCI\_IO\_PROTOCOL.Mem.Write - With Width as **EfiPciIoWidthMaximum** the return status is EFI\_INVALID\_PARAMETER. | 1. Call Mem.Write() with Width as **EfiPciWidthMaximum**. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.4.9 | 0xb868ce7a, 0xfff0, 0x4c3c, 0x98, 0x00, 0xf5, 0xc7, 0xc2, 0x13, 0xaa, 0x09 | EFI\_PCI\_IO\_PROTOCOL.Mem.Write - With invalid Width type -1 the return status is EFI\_INVALID\_PARAMETER. | 1. Call Mem.Write() with Width as -1. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.4.10 | 0x2fe9804a, 0xa418, 0x40b7, 0xa6, 0x8c, 0xaa, 0x40, 0xc3, 0xe6, 0x2f, 0x84 | EFI\_PCI\_IO\_PROTOCOL.Mem.Write - With Buffer as NULL the return status is EFI\_INVALID\_PARAMETER. | 1. Call Mem.Write() with Buffer as NULL. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.4.11 | 0xdac9a8dc, 0x172e, 0x4c6d, 0xb2, 0xe7, 0xf1, 0x65, 0x94, 0xfe, 0x89, 0x39 | EFI\_PCI\_IO\_PROTOCOL.Mem.Write - With address out of BAR range the return status is EFI\_UNSUPPORTED. | 1. Call Mem.Write() with address out of BAR range. The return status must be EFI\_UNSUPPORTED. |
| 5.8.2.4.12 | 0x99fca122, 0xd9dc, 0x4d3b, 0xbb, 0xb0, 0x2a, 0xf5, 0x3d, 0xd1, 0x39, 0x0e | EFI\_PCI\_IO\_PROTOCOL.Mem.Write - With address out of BAR range the return status is EFI\_UNSUPPORTED. | 1. Call Mem.Write() with address out of BAR range. The return status must be EFI\_UNSUPPORTED. |
| 5.8.2.4.13 | 0xd5c1f492, 0x5dbf, 0x4b4d, 0x9e, 0x09, 0xd5, 0x1a, 0x23, 0x47, 0x37, 0xcc | EFI\_PCI\_IO\_PROTOCOL.Mem.Write - With invalid BAR Index the return status is EFI\_UNSUPPORTED. | 1. Call Mem.Write() with invalid BAR Index. The return status must be EFI\_UNSUPPORTED. |
| 5.8.2.4.14 | 0x1af1b78c, 0x8ca2, 0x4146, 0x97, 0x69, 0x94, 0x29, 0xac, 0x48, 0x11, 0x65 | EFI\_PCI\_IO\_PROTOCOL.Mem.Write - With Io **Type** BAR the return status is EFI\_UNSUPPORTED. | 1. Call Mem.Write() with Io **Type** BAR. The return status must be EFI\_UNSUPPORTED. |
| 5.8.2.4.15 | 0xa154d373, 0xc12b, 0x4939, 0xa3, 0xb2, 0xc0, 0x14, 0xc1, 0x09, 0xd3, 0x68 | EFI\_PCI\_IO\_PROTOCOL.Mem.Write - With invalid Width the return status is EFI\_INVALID\_PARAMETER. | 1. Call Mem.Write() with invalid Width. The return status must be EFI\_INVALID\_PARAMETER. |

### Io.Read()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.2.5.1 | 0x36e0b044, 0x2b2b, 0x484b, 0xb4, 0x80, 0x85, 0x99, 0xa9, 0x99, 0xa9, 0x35 | EFI\_PCI\_IO\_PROTOCOL.Io.Read - Io.Read() reads data out and returns EFI\_SUCCESS. | 1. Allocate a buffer that matches the size of the address range.  2. Call Io.Read() to fill in the buffer with the predefined data units.  The return status should be EFI\_SUCCESS. |
| 5.8.2.5.2 | 0xe65f66cb, 0xb1cb, 0x4a7a, 0x8c, 0x68, 0xb2, 0x0c, 0x69, 0x58, 0xdd, 0x6a | EFI\_PCI\_IO\_PROTOCOL.Io.Read - Io.Read() reads out the data from Io address space and returns EFI\_SUCCESS. | 1. Allocate a buffer that matches the size of the address range.  2. Call Io.Read() to fill in the buffer with the predefined data units.  3. Call Io.Write() to write the buffer into the address range.  4. Call Io.Read() to read out the data in destination address range.  The return status should be EFI\_SUCCESS. |
| 5.8.2.5.3 | 0xec27b5c5, 0x59fb, 0x4954, 0x9c, 0x51, 0xad, 0xf4, 0x46, 0x7e, 0xe7, 0xe6 | EFI\_PCI\_IO\_PROTOCOL.Io.Read - The data read out is the same as that written in. | 1. Allocate a buffer that matches the size of the address range.  2. Call Io.Read() to fill in the buffer with the predefined data units.  3. Call Io.Write() to write the buffer into the address range.  4. Call Io.Read() to read out the data in destination address range.  5. Compare the data read out with data written in. |
| 5.8.2.5.4 | 0x271e3b70, 0x6617, 0x4f5f, 0xb5, 0x12, 0x46, 0xb1, 0xe3, 0x1d, 0xe3, 0x79 | EFI\_PCI\_IO\_PROTOCOL.Io.Read - Io.Read() reads out the data with **EfiPciIoWidthFifoUnintX** and returns EFI\_SUCCESS. | 1. Allocate a buffer that matches the size of the address range.  2. Call Io.Read() to fill in the buffer with the predefined data units.  3. Call Io.Write() to write the buffer into the address range.  4. Call Io.Read() to read out the data using **EfiPciIoWidthFifoUintX**.  The return status should be EFI\_SUCCESS. |
| 5.8.2.5.5 | 0xccf3806e, 0x25fa, 0x4697, 0xb7, 0x08, 0x8d, 0xc1, 0x5b, 0x47, 0xba, 0x8d | EFI\_PCI\_IO\_PROTOCOL.Io.Read – All the data read out with **EfiPciIoWidthFifoX** is equal with the first data unit. | 1. Allocate a buffer that matches the size of the address range.  2. Call Io.Read() to fill in the buffer with the predefined data units.  3. Call Io.Write() to write the buffer into the address range.  4. Call Io.Read() to read out the data using **EfiPciIoWidthFifoUintX**.  5. Compare each data unit in the buffer with the data at the Starting Address of the address range. |
| 5.8.2.5.6 | 0x080ea87f, 0xc265, 0x4a33, 0xab, 0x09, 0x78, 0xf8, 0x94, 0x58, 0x03, 0x2b | EFI\_PCI\_IO\_PROTOCOL.Io.Read - Io.Read() reads out the data with **EfiPciIoWidthFillX** and returns EFI\_SUCCESS. | 1. Allocate a buffer that matches the size of the address range.  2. Call Io.Read() to fill in the buffer with the predefined data units.  3. Call Io.Write() to write the buffer into the address range.  4. Call Io.Read() to read out the data using **EfiPciIoWidthFillUintX**.  The return status should be EFI\_SUCCESS. |
| 5.8.2.5.7 | 0x543fda6a, 0x651a, 0x4560, 0xaf, 0xfd, 0x6a, 0x95, 0x76, 0x54, 0x07, 0x30 | EFI\_PCI\_IO\_PROTOCOL.Io.Read - Reads out the data with **EfiPciIoWidthFillX**. The first data unit eqauls the last data unit in destination address. | 1. Allocate a buffer that matches the size of the address range.  2. Call Io.Read() to fill in the buffer with the predefined data units.  3. Call Io.Write() to write the buffer into the address range.  4. Call Io.Read() to read out the data using **EfiPciIoWidthFillUintX**.  5. Compare the first data unit in the output buffer with the last data unit in the address range. Compare other data units in the output buffer with the original data. |
| 5.8.2.5.8 | 0x65b3c515, 0x1fe1, 0x4021, 0xb2, 0x02, 0xdd, 0xc9, 0x7a, 0x0d, 0xb2, 0x11 | EFI\_PCI\_IO\_PROTOCOL.Io.Read - With Width as **EfiPciIoWidthMaximum**, the return status is EFI\_INVALID\_PARAMETER. | 1. Call Io.Read() with Width as **EfiPciWidthMaximum**. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.5.9 | 0x8ef36cf9, 0x84b7, 0x4961, 0xaa, 0xcc, 0xf7, 0x41, 0x21, 0x96, 0xc0, 0xdc | EFI\_PCI\_IO\_PROTOCOL.Io.Read - With invalid Width type -1, the return status is EFI\_INVALID\_PARAMETER. | 1. Call Io.Read() with Width as -1. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.5.10 | 0x4cac979d, 0x6b8c, 0x458c, 0xb3, 0xca, 0x75, 0x30, 0x6f, 0x59, 0xa9, 0xb7 | EFI\_PCI\_IO\_PROTOCOL.Io.Read - With Buffer as NULL, the return status is EFI\_INVALID\_PARAMETER. | 1. Call Io.Read() with Buffer as NULL. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.5.11 | 0xaf51e635, 0x89c8, 0x49db, 0xa7, 0x11, 0xb6, 0xc6, 0xb8, 0x96, 0xf9, 0x79 | EFI\_PCI\_IO\_PROTOCOL.Io.Read - With address out of BAR range, the return status is EFI\_UNSUPPORTED. | 1. Call Io.Read() with address out of BAR range. The return status must be EFI\_UNSUPPORTED. |
| 5.8.2.5.12 | 0x8d878934, 0x8270, 0x48a7, 0xad, 0x51, 0x65, 0xa8, 0x8d, 0xac, 0x36, 0x93 | EFI\_PCI\_IO\_PROTOCOL.Io.Read - With address out of BAR range, the return status is EFI\_UNSUPPORTED. | 1. Call Io.Read() with address out of BAR range. The return status must be EFI\_UNSUPPORTED. |
| 5.8.2.5.13 | 0x53cc0e1e, 0xf3aa, 0x4f15, 0xaf, 0xec, 0xc3, 0x04, 0x8f, 0x0f, 0xa5, 0xb8 | EFI\_PCI\_IO\_PROTOCOL.Io.Read - With invalid BAR Index, the return status is EFI\_UNSUPPORTED. | 1. Call Io.Read() with invalid BAR Index. The return status must be EFI\_UNSUPPORTED. |
| 5.8.2.5.14 | 0x2fb4dc13, 0xb3f5, 0x4e19, 0xba, 0xe2, 0x76, 0x47, 0x10, 0x4d, 0xf7, 0x79 | EFI\_PCI\_IO\_PROTOCOL.Io.Read - With Mem **Type** BAR, the return status is EFI\_UNSUPPORTED. | 1. Call Io.Read() with Mem **Type** BAR. The return status must be EFI\_UNSUPPORTED. |
| 5.8.2.5.15 | 0x44b6de4e, 0xc968, 0x4d97, 0xbe, 0x01, 0x3f, 0xf3, 0xdd, 0xfc, 0x53, 0xe0 | EFI\_PCI\_IO\_PROTOCOL.Io.Read - With invalid Width, the return status is EFI\_INVALID\_PARAMETER. | 1. Call Io.Read() with invalid Width. The return status must be EFI\_INVALID\_PARAMETER. |

### Io.Write()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.2.6.1 | 0x7b1ed2c6, 0xa84e, 0x4858, 0xa7, 0x8b, 0xa6, 0xd9, 0x32, 0x03, 0x22, 0xbe | EFI\_PCI\_IO\_PROTOCOL.Io.Write - Io.Write() writes data to Io address space, returns EFI\_SUCCESS. | 1. Allocate a buffer that matches the size of the address range.  2. Call Io.Read() to fill in the buffer with the predefined data units.  3. Call Io.Write() to write the buffer into the address range.  The return status should be EFI\_SUCCESS. |
| 5.8.2.6.2 | 0xd1704c13, 0xd0df, 0x4f7c, 0xb8, 0xb6, 0xd9, 0x5b, 0xe6, 0xdc, 0xea, 0x87 | EFI\_PCI\_IO\_PROTOCOL.Io.Write - The data read equals the data written in. | 1. Allocate a buffer that matches the size of the address range.  2. Call Io.Read() to fill in the buffer with the predefined data units.  3. Call Io.Write() to write the buffer into the address range.  4. Call Io.Read() to read out the data in destination address range.  5. Compare the data read out with data written in. |
| 5.8.2.6.3 | 0xafb5070c, 0x1d07, 0x4df3, 0x9a, 0xd5, 0x6f, 0x91, 0x7e, 0x48, 0xc5, 0xed | EFI\_PCI\_IO\_PROTOCOL.Io.Write - Io.Write() writes the data with **EfiPciIoWidthFifoX** and returns EFI\_SUCCESS. | 1. Allocate a buffer that matches the size of the address range.  2. Call Io.Read() to fill in the buffer with the predefined data units.  3. Call Io.Write() to write to the starting address of address range using **EfiPciIoWidthFifoUintX**.  The return status should be EFI\_SUCCESS. |
| 5.8.2.6.4 | 0xee8d1797, 0x1474, 0x4d80, 0x85, 0x82, 0x35, 0x78, 0x61, 0x2b, 0x26, 0x01 | EFI\_PCI\_IO\_PROTOCOL.Io.Write - With **EfiPciIoWidthFifoUintX**, the first data unit is the last data unit. | 1. Allocate a buffer that matches the size of the address range.  2. Call Io.Read() to fill in the buffer with the predefined data units.  3. Call Io.Write() to write to the starting address of address range using **EfiPciIoWidthFifoUintX**.  4. Call Io.Read() to read out the data using **EfiPciIoWidthFifoUintX**.  5. Compare the data in the starting address with the last data unit. Compare other data units with original data. |
| 5.8.2.6.5 | 0x4a6378ee, 0x5058, 0x42b2, 0x8a, 0x03, 0x1f, 0x1c, 0xff, 0x05, 0x15, 0xcc | EFI\_PCI\_IO\_PROTOCOL.Io.Write - Io.Write() writes the data with **EfiPciIoWidthFillX** and returns EFI\_SUCCESS. | 1. Allocate a buffer that matches the size of the address range.  2. Call Io.Read() to fill in the buffer with the predefined data units.  3. Call Io.Write() to write to address range using **EfiPciIoWidthFillUintX**.  The return status should be EFI\_SUCCESS. |
| 5.8.2.6.6 | 0x15b81460, 0xbc5e, 0x4be3, 0x9c, 0xc5, 0xb6, 0x59, 0x06, 0x83, 0x28, 0x5e | EFI\_PCI\_IO\_PROTOCOL.Io.Write - With **EfiPciIoWidthFillUintX**, all the data units read out are the same as the first data units in the address space. | 1. Allocate a buffer that matches the size of the address range.  2. Call Io.Read() to fill in the buffer with the predefined data units.  3. Call Io.Write() to write to address range using **EfiPciIoWidthFillUintX**.  4. Call Io.Read() to read out the data using **EfiPciIoWidthFillUintX**.  5. Compare all the data units with the first data unit. |
| 5.8.2.6.7 | 0x8e854d61, 0x2048, 0x446f, 0xb6, 0x47, 0x3c, 0x37, 0x17, 0x14, 0xac, 0xf6 | EFI\_PCI\_IO\_PROTOCOL.Io.Write - Io.Write() writes back the data and returns EFI\_SUCCESS. | 1. Allocate a buffer that matches the size of the address range.  2. Call Io.Read() to fill in the buffer with the predefined data units.  3. Call Io.Write() to write the buffer into the address range.  4. Call Io.Read() to read out the data in destination address range.  5. Call Io.Write() to write the data back.  The return status should be EFI\_SUCCESS. |
| 5.8.2.6.8 | 0xb96af4e4, 0x988f, 0x4362, 0x8c, 0x63, 0x1f, 0x08, 0xeb, 0xfd, 0xa3, 0x5f | EFI\_PCI\_IO\_PROTOCOL.Io.Write - With Width as **EfiPciIoWidthMaximum**, return status is EFI\_INVALID\_PARAMETER. | 1. Call Io.Write() with Width as **EfiPciWidthMaximum**. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.6.9 | 0x8cb298d4, 0x5831, 0x48ce, 0x87, 0x8d, 0xf3, 0xf8, 0x20, 0x62, 0xea, 0xf3 | EFI\_PCI\_IO\_PROTOCOL.Io.Write - With invalid Width type -1, the return status is EFI\_INVALID\_PARAMETER. | 1. Call Io.Write() with Width as ‑1. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.6.10 | 0x175943ee, 0x4d2d, 0x480f, 0xa3, 0xf1, 0x88, 0xc9, 0x7c, 0x6b, 0x04, 0x77 | EFI\_PCI\_IO\_PROTOCOL.Io.Write - With Buffer as NULL, the return status is EFI\_INVALID\_PARAMETER. | 1. Call Io.Write() with Buffer as NULL. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.6.11 | 0x4617468a, 0xd228, 0x4a84, 0x88, 0x56, 0x21, 0x8c, 0x3f, 0x39, 0x46, 0xd1 | EFI\_PCI\_IO\_PROTOCOL.Io.Write - With address out of BAR range the return status is EFI\_UNSUPPORTED. | 1. Call Io.Write() with address out of BAR range. The return status must be EFI\_UNSUPPORTED. |
| 5.8.2.6.12 | 0x03dd4807, 0xe461, 0x4e97, 0x9d, 0xf9, 0xea, 0x73, 0x38, 0x15, 0xd5, 0x62 | EFI\_PCI\_IO\_PROTOCOL.Io.Write - With address out of BAR range the return status is EFI\_UNSUPPORTED. | 1. Call Io.Write() with address out of BAR range. The return status must be EFI\_UNSUPPORTED. |
| 5.8.2.6.13 | 0xd6b9d51d, 0x2676, 0x4449, 0xa4, 0xd6, 0x3d, 0xa0, 0x17, 0x36, 0x2e, 0xa6 | EFI\_PCI\_IO\_PROTOCOL.Io.Write - With invalid BAR Index the return status is EFI\_UNSUPPORTED. | 1. Call Io.Write() with invalid BAR Index. The return status must be EFI\_UNSUPPORTED. |
| 5.8.2.6.14 | 0x648a859d, 0x3b72, 0x41a6, 0x86, 0xad, 0x3f, 0xff, 0x66, 0xd8, 0x61, 0x2f | EFI\_PCI\_IO\_PROTOCOL.Io.Write - With Mem **Type** BAR the return status is EFI\_UNSUPPORTED. | 1. Call Io.Write() with Mem **Type** BAR. The return status must be EFI\_UNSUPPORTED. |
| 5.8.2.6.15 | 0xfdc9b3f3, 0x2b80, 0x4a99, 0xa9, 0xba, 0xa5, 0x5e, 0xf9, 0xf8, 0x26, 0x19 | EFI\_PCI\_IO\_PROTOCOL.Io.Write - With invalid Width the return status is EFI\_INVALID\_PARAMETER. | 1. Call Io.Write() with invalid Width. The return status must be EFI\_INVALID\_PARAMETER. |

### Pci.Read()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.2.7.1 | 0xea2a44d0, 0xc8d1, 0x465b, 0xb5, 0x50, 0x58, 0xd6, 0xef, 0x4e, 0x38, 0xd4 | EFI\_PCI\_IO\_PROTOCOL.Pci.Read - Pci.Read() reads data out into backup buffer and returns EFI\_SUCCESS. | 1. Allocate a buffer that matches the size of the address range.  2. Call Pci.Read() to fill in the buffer with the predefined data units.  The return status should be EFI\_SUCCESS. |
| 5.8.2.7.2 | 0xe30bb837, 0x1d06, 0x4ee2, 0x80, 0x85, 0x18, 0xd4, 0x6b, 0x1c, 0x99, 0x66 | EFI\_PCI\_IO\_PROTOCOL.Pci.Read - Pci.Read() reads out the data from PCI configuration space and returns EFI\_SUCCESS. | 1. Allocate a buffer that matches the size of the address range.  2. Call Pci.Read() to fill in the buffer with the predefined data units.  3. Call Pci.Write() to write the buffer into the address range.  4. Call Pci.Read() to read out the data in destination address range.  The return status should be EFI\_SUCCESS. |
| 5.8.2.7.3 | 0x2f9274d9, 0x7a14, 0x492f, 0x87, 0xc0, 0x40, 0x81, 0x4f, 0x66, 0x1b, 0xb4 | EFI\_PCI\_IO\_PROTOCOL.Pci.Read – The data read out from the PCI configuration space with **PciIoWidthUintX** equals the data written in. | 1. Allocate a buffer that matches the size of the address range.  2. Call Pci.Read() to fill in the buffer with the predefined data units.  3. Call Pci.Write() to write the buffer into the address range.  4. Call Pci.Read() to read out the data in destination address range.  5. Compare the data read out with data written in. |
| 5.8.2.7.4 | 0x59ba5b67, 0x9e17, 0x4b60, 0xb5, 0x79, 0x5f, 0xd3, 0x26, 0x16, 0xe6, 0x6a | EFI\_PCI\_IO\_PROTOCOL.Pci.Read - Pci.Read() reads out the data with **EfiPciIoWidthFifoX** and returns EFI\_SUCCESS. | 1. Allocate a buffer that matches the size of the address range.  2. Call Pci.Read() to fill in the buffer with the predefined data units.  3. Call Pci.Write() to write the buffer into the address range.  4. Call Pci.Read() to read out the data using **EfiPciIoWidthFifoUintX**.  The return status should be EFI\_SUCCESS. |
| 5.8.2.7.5 | 0xd3b49ee4, 0x131a, 0x4fa3, 0xab, 0x81, 0x9f, 0x86, 0x33, 0xdf, 0x2d, 0xc7 | EFI\_PCI\_IO\_PROTOCOL.Pci.Read - Reads out the data with **EfiPciIoWidthFifoX**. The data read out is the same as the first data unit. | 1. Allocate a buffer that matches the size of the address range.  2. Call Pci.Read() to fill in the buffer with the predefined data units.  3. Call Pci.Write() to write the buffer into the address range.  4. Call Pci.Read() to read out the data using **EfiPciIoWidthFifoUintX**.  5. Compare the each data unit in the buffer with the data at the Starting Address of the address range. |
| 5.8.2.7.6 | 0x6e5881b2, 0x262d, 0x41ec, 0xa8, 0xd4, 0xcf, 0x28, 0x71, 0x1e, 0x5c, 0x15 | EFI\_PCI\_IO\_PROTOCOL.Pci.Read - Pci.Read() reads out the data with **EfiPciIoWidthFillX** and returns EFI\_SUCCESS. | 1. Allocate a buffer that matches the size of the address range.  2. Call Pci.Read() to fill in the buffer with the predefined data units.  3. Call Pci.Write() to write the buffer into the address range.  4. Call Pci.Read() to read out the data using **EfiPciIoWidthFillUintX**.  The return status should be EFI\_SUCCESS. |
| 5.8.2.7.7 | 0x4595bbca, 0xbad7, 0x417f, 0xaf, 0x8d, 0x37, 0xec, 0x32, 0xf8, 0x03, 0x80 | EFI\_PCI\_IO\_PROTOCOL.Pci.Read - Reads out the data with **EfiPciIoWidthFillX**. The first data unit equals the last data unit in Destination address range. | 1. Allocate a buffer that matches the size of the address range.  2. Call Pci.Read() to fill in the buffer with the predefined data units.  3. Call Pci.Write() to write the buffer into the address range.  4. Call Pci.Read() to read out the data using **EfiPciIoWidthFillUintX**.  5. Compare the first data unit in the output buffer with the last data unit in the address range. Compare other data units in the output buffer with the original data. |
| 5.8.2.7.8 | 0x94d0a3d8, 0x7b61, 0x4147, 0xad, 0x9a, 0xea, 0xbb, 0x5f, 0x30, 0x59, 0xc2 | EFI\_PCI\_IO\_PROTOCOL.Pci.Read - With Width as **EfiPciIoWidthMaximum**, the return status is EFI\_INVALID\_PARAMETER. | 1. Call Pci.Read() with Width as **EfiPciWidthMaximum**. The return status must be EFI\_INVALID\_PARAMETER |
| 5.8.2.7.9 | 0x18cf01fe, 0xa703, 0x4639, 0xb8, 0xe0, 0x8e, 0xd7, 0x3c, 0xbe, 0xa0, 0xb6 | EFI\_PCI\_IO\_PROTOCOL.Pci.Read - With invalid Width type -1, the return status is EFI\_INVALID\_PARAMETER. | 1. Call Pci.Read() with Width as -1. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.7.10 | 0xa7710b95, 0x114d, 0x4096, 0xa8, 0x3c, 0xf6, 0x5f, 0x63, 0x00, 0xbd, 0xab | EFI\_PCI\_IO\_PROTOCOL.Pci.Read - With Buffer as NULL the return status is EFI\_INVALID\_PARAMETER. | 1. Call Pci.Read() with Buffer as NULL. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.7.11 | 0x147279d7, 0xf685, 0x4658, 0xb8, 0x09, 0xdf, 0xd1, 0xd7, 0x75, 0xe0, 0xb5 | EFI\_PCI\_IO\_PROTOCOL.Pci.Read - With Offset + Count \* Width > 255, the return status is EFI\_UNSUPPORTED. | 1. Call Pci.Read() with Offset + Count \* Width > 255. The return status must be EFI\_UNSUPPORTED. |
| 5.8.2.7.12 | 0xf070aeda, 0x2e6b, 0x4911, 0xae, 0x80, 0x1b, 0x21, 0xcc, 0xef, 0x30, 0x50 | EFI\_PCI\_IO\_PROTOCOL.Pci.Read - With Offset + Count \* Width > 255 the return status is EFI\_UNSUPPORTED. | 1. Call Pci.Read() with Offset + Count \* Width > 255. The return status must be EFI\_UNSUPPORTED. |
| 5.8.2.7.13 | 0x85111b07, 0x5d78, 0x4e62, 0x90, 0x48, 0x69, 0xca, 0x37, 0x4a, 0xdc, 0xb3 | EFI\_PCI\_IO\_PROTOCOL.Pci.Read - With invalid Width the return status is EFI\_INVALID\_PARAMETER. | 1. Call Pci.Read() with invalid Width. The return status must be EFI\_INVALID\_PARAMETER. |

### Pci.Write()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.2.8.1 | 0x1c65f03c, 0x6d87, 0x435e, 0x94, 0x2e, 0x41, 0x4f, 0xfa, 0x1d, 0x69, 0xb8 | EFI\_PCI\_IO\_PROTOCOL.Pci.Write - Pci.Write() writes data to the PCI configuration space and returns EFI\_SUCCESS. | 1. Allocate a buffer that matches the size of the address range.  2. Call Pci.Read() to fill in the buffer with the predefined data units.  3. Call Pci.Write() to write the buffer into the address range  The return status should be EFI\_SUCCESS. |
| 5.8.2.8.2 | 0xb175434f, 0xf038, 0x43a2, 0xa1, 0xa8, 0xef, 0xab, 0x71, 0x57, 0x7f, 0xac | EFI\_PCI\_IO\_PROTOCOL.Pci.Write – Data read out from PCI configuration space with **PciIoWidthUintX** equals the data written in. | 1. Allocate a buffer that matches the size of the address range  2. Call Pci.Read() to fill in the buffer with the predefined data units  3. Call Pci.Write() to write the buffer into the address range  4. Call Pci.Read() to read out the data in destination address range.  5. Compare the data read out with data written in. |
| 5.8.2.8.3 | 0xfbc65a77, 0xd113, 0x4584, 0xa6, 0xe0, 0x40, 0x6d, 0xc7, 0xd9, 0x24, 0x1f | EFI\_PCI\_IO\_PROTOCOL.Pci.Write - Pci.Write() writes the data with **EfiPciIoWidthFifoX** and returns EFI\_SUCCESS | 1. Allocate a buffer that matches the size of the address range.  2. Call Pci.Read() to fill in the buffer with the predefined data units.  3. Call Pci.Write() to write to the starting address of address range using **EfiPciIoWidthFifoUintX**.  The return status should be EFI\_SUCCESS. |
| 5.8.2.8.4 | 0x1dd97ca1, 0x6920, 0x41db, 0xa2, 0x0c, 0xcf, 0x62, 0x78, 0xbd, 0x07, 0x47 | EFI\_PCI\_IO\_PROTOCOL.Pci.Write - With **PciIoWidthFifoUintX**, the first data unit is equal to the last data unit, and the other data units are unchanged. | 1. Allocate a buffer that matches the size of the address range.  2. Call Pci.Read() to fill in the buffer with the predefined data units.  3. Call Pci.Write() to write to the starting address of address range using **EfiPciIoWidthFifoUintX**.  4. Call Pci.Read() to read out the data using **EfiPciIoWidthFifoUintX**.  5. Compare the data in the starting address with the last data unit. Compare other data units with original data. |
| 5.8.2.8.5 | 0x3ea04425, 0xbf3d, 0x465a, 0xbd, 0x5b, 0xf7, 0x77, 0xc5, 0x41, 0x21, 0x6a | EFI\_PCI\_IO\_PROTOCOL.Pci.Write - Pci.Write() writes the data with **EfiPciIoWidthFillX** and returns EFI\_SUCCESS. | 1. Allocate a buffer that matches the size of the address range.  2. Call Pci.Read() to fill in the buffer with the predefined data units.  3. Call Pci.Write() to write to address range using **EfiPciIoWidthFillUintX**.  The return status should be EFI\_SUCCESS. |
| 5.8.2.8.6 | 0x74ff6a17, 0xdf28, 0x434a, 0x8a, 0xd7, 0xbf, 0xa3, 0xe9, 0xc5, 0x1f, 0x12 | EFI\_PCI\_IO\_PROTOCOL.Pci.Write - With **PciIoWidthFillX**, all the data units read out are equal to the first data unit in the address space. | 1. Allocate a buffer that matches the size of the address range.  2. Call Pci.Read() to fill in the buffer with the predefined data units.  3. Call Pci.Write() to write to address range using **EfiPciIoWidthFillUintX**.  4. Call Pci.Read() to read out the data using **EfiPciIoWidthFillUintX**.  5. Compare all the data units with the first data unit. |
| 5.8.2.8.7 | 0xc355e57b, 0x93ef, 0x4ca6, 0x91, 0x2f, 0x65, 0x6e, 0x4f, 0x2e, 0x2a, 0x13 | EFI\_PCI\_IO\_PROTOCOL.Pci.Write - Pci.Write() writes data back with **EfiPciIoWidthX** and returns EFI\_SUCCESS | 1. Allocate a buffer that matches the size of the address range  2. Call Pci.Read() to fill in the buffer with the predefined data units  3. Call Pci.Write() to write the buffer into the address range  4. Call Pci.Read() to read out the data in destination address range.  5. Call Pci.Write() to write the data back.  The return status should be EFI\_SUCCESS |
| 5.8.2.8.8 | 0x8a26f93b, 0xc0a3, 0x4e08, 0x9f, 0xf1, 0xd6, 0xf1, 0xac, 0x2e, 0x63, 0x9a | EFI\_PCI\_IO\_PROTOCOL.Pci.Write - With Width as **EfiPciIoWidthMaximum** the return status is EFI\_INVALID\_PARAMETER | 1. Call Pci.Write() with Width as **EfiPciWidthMaximum**. The return status must be EFI\_INVALID\_PARAMETER |
| 5.8.2.8.9 | 0xfeab0187, 0x541b, 0x45da, 0x92, 0x1f, 0x49, 0x01, 0x00, 0xb7, 0xdd, 0x7a | EFI\_PCI\_IO\_PROTOCOL.Pci.Write - With invalid Width type -1 the return status is EFI\_INVALID\_PARAMETER | 1. Call Pci.Write() with Width as -1. The return status must be EFI\_INVALID\_PARAMETER |
| 5.8.2.8.10 | 0x686732db, 0xa12b, 0x4ed7, 0x90, 0xfb, 0x66, 0x92, 0xbb, 0xd7, 0xe8, 0x4c | EFI\_PCI\_IO\_PROTOCOL.Pci.Write - With Buffer as NULL the return status is EFI\_INVALID\_PARAMETER | 1. Call Pci.Write() with Buffer as NULL. The return status must be EFI\_INVALID\_PARAMETER |
| 5.8.2.8.11 | 0x11cf0b51, 0x6f50, 0x4bba, 0xa9, 0xd7, 0x3e, 0x53, 0x28, 0xb3, 0x1f, 0x30 | EFI\_PCI\_IO\_PROTOCOL.Pci.Write - With Offset + Count \* Width > 255 the return status is EFI\_UNSUPPORTED | 1. Call Pci.Write() with Offset + Count \* Width > 255. The return status must be EFI\_UNSUPPORTED |
| 5.8.2.8.12 | 0x4e4617a2, 0x4e8a, 0x46c8, 0xb2, 0x4b, 0xa4, 0x91, 0x55, 0xf2, 0x3a, 0x0d | EFI\_PCI\_IO\_PROTOCOL.Pci.Write - With Offset + Count \* Width > 255 the return status is EFI\_UNSUPPORTED | 1. Call Pci.Write() with Offset + Count \* Width > 255. The return status must be EFI\_UNSUPPORTED |
| 5.8.2.8.13 | 0xc6dbb28e, 0xbf42, 0x40e3, 0xbc, 0x93, 0x5f, 0x9b, 0x11, 0xa2, 0x46, 0x5f | EFI\_PCI\_IO\_PROTOCOL.Pci.Write - With invalid Width the return status is EFI\_INVALID\_PARAMETER | 1. Call Pci.Write() with invalid Width. The return status must be EFI\_INVALID\_PARAMETER |

### CopyMem()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.2.9.1 | 0x8d728b05, 0xc64e, 0x4ef0, 0x80, 0x68, 0x51, 0xbc, 0xe3, 0x9f, 0xc5, 0x0c | EFI\_PCI\_IO\_PROTOCOL.CopyMem - CopyMem() copying Data between non-overlapping regions returns EFI\_SUCCESS. | 1. Allocate a buffer, the size of which is: **BufferSize** = Address Range Size / Width */ 2 \** Width.  2. Call Mem.Write() to write the buffer into the beginning address.  3. Call CopyMem() to copy Data between non-overlapping regions.  The return status should be EFI\_SUCCESS. |
| 5.8.2.9.2 | 0x73f80e2c, 0xe2d9, 0x4c6b, 0xbe, 0xc0, 0x85, 0xd7, 0xa4, 0x27, 0x07, 0xd0 | EFI\_PCI\_IO\_PROTOCOL.CopyMem – Data copied between non‑overlapping regions is equal. | 1. Allocate a buffer, the size of which is: **BufferSize** = Address Range Size / Width */ 2 \** Width.  2. Call Mem.Write() to write the buffer into the beginning address.  3. Call CopyMem() to copy Data between non-overlapping regions.  4. Call Mem.Read() to read out the data.  5. Compare the data read out with the data written in. |
| 5.8.2.9.3 | 0x459bcee9, 0x16f7, 0x41ae, 0x81, 0x55, 0x7e, 0x49, 0xec, 0x98, 0x56, 0x7d | EFI\_PCI\_IO\_PROTOCOL.CopyMem - CopyMem() copying Data between overlapping regions (destination address > source address) returns EFI\_SUCCESS. | 1. Allocate a buffer, the size of which is: **BufferSize** = Address Range Size / Width */ 2 \** Width.  2. Call Mem.Write() to write the buffer into the beginning address.  3. Call CopyMem() to copy Data between overlapping regions (destination address > source address).  The return status should be EFI\_SUCCESS. |
| 5.8.2.9.4 | 0x9ca6f1d4, 0xfb7c, 0x416c, 0xa6, 0x09, 0x06, 0xa4, 0xcb, 0x0f, 0x44, 0x59 | EFI\_PCI\_IO\_PROTOCOL.CopyMem – When copying Data between overlapping regions (destination > source), the data is copied. | 1. Allocate a buffer, the size of which is: **BufferSize** = Address Range Size / Width */ 2 \** Width.  2. Call Mem.Write() to write the buffer into the beginning address.  3. Call CopyMem() to copy Data between overlapping regions (destination address > source address).  4. Call Mem.Read() to read out the data  5. Compare the data read out with the data written in. |
| 5.8.2.9.5 | 0xb8eb3987, 0x9915, 0x40d2, 0x93, 0xc6, 0xe1, 0x83, 0x7e, 0x49, 0x4e, 0x1a | EFI\_PCI\_IO\_PROTOCOL.CopyMem - CopyMem() copying Data between overlapping regions (destination address < source address) returns EFI\_SUCCESS. | 1. Allocate a buffer, the size of which is: **BufferSize** = Address Range Size / Width */ 2 \** Width.  2. Call Mem.Write() to write the buffer into the beginning address.  3. Call CopyMem() to copy Data between overlapping regions (destination address < source address).  The return status should be EFI\_SUCCESS. |
| 5.8.2.9.6 | 0x3294319c, 0xc3f0, 0x46f2, 0x81, 0xfd, 0x14, 0xc0, 0xd0, 0x61, 0xc4, 0x42 | EFI\_PCI\_IO\_PROTOCOL.CopyMem – When copying Data between overlapping regions (destination < source) the data is copied. | 1. Allocate a buffer, the size of which is: **BufferSize** = Address Range Size / Width */ 2 \** Width.  2. Call Mem.Write() to write the buffer into the beginning address.  3. Call CopyMem() to copy Data between overlapping regions (destination address < source address).  4. Call Mem.Read() to read out the data.  5. Compare the data read out with the data written in. |
| 5.8.2.9.7 | 0xd0b52eb3, 0x3d19, 0x4b72, 0xb5, 0xba, 0xe3, 0xa3, 0x7c, 0xd0, 0xcb, 0x93 | EFI\_PCI\_IO\_PROTOCOL.CopyMem - CopyMem() copying Data between different BARs returns EFI\_SUCCESS. | 1. Allocate a buffer, the size of which is: **BufferSize** = Address Range Size / Width/ 2 \*Width.  2. Call Mem.Write() to write the buffer into the beginning address.  3. Call CopyMem() to copy Data between different BARs.  The return status should be EFI\_SUCCESS. |
| 5.8.2.9.8 | 0xe0863095, 0x4854, 0x4099, 0x89, 0xf0, 0x01, 0xbf, 0xda, 0x41, 0xa4, 0xe3 | EFI\_PCI\_IO\_PROTOCOL.CopyMem – When copying Data between different BARs the data is copied. | 1. Allocate a buffer, the size of which is: **BufferSize** = Address Range Size / Width/ 2 \*Width.  2. Call Mem.Write() to write the buffer into the beginning address.  3. Call CopyMem() to copy Data between different BARs.  4. Call Mem.Read() to read out the data.  5. Compare the data read out with the data written in. |
| 5.8.2.9.9 | 0x45056bf8, 0xe6e4, 0x4397, 0xb7, 0xe1, 0x89, 0x0b, 0x42, 0x4d, 0xe3, 0x54 | EFI\_PCI\_IO\_PROTOCOL.CopyMem - With Width as **EfiPciIoWidthMaximum** the return status is EFI\_INVALID\_PARAMETER. | 1. Call CopyMem() with Width as **EfiPciWidthMaximum**. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.9.10 | 0xf780b74f, 0x6b93, 0x4e64, 0x8a, 0xb5, 0x05, 0x77, 0xdd, 0x99, 0xc1, 0xfa | EFI\_PCI\_IO\_PROTOCOL.CopyMem - With invalid Width type ‑1 the return status is EFI\_INVALID\_PARAMETER. | 1. Call CopyMem() with Width as -1. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.9.11 | 0xebf7fa5c, 0xb4c9, 0x406c, 0x8d, 0x12, 0x90, 0x3a, 0x81, 0x85, 0x26, 0x17 | EFI\_PCI\_IO\_PROTOCOL.CopyMem - With Width as **EfiPciWidthFifoUintX** the return status is EFI\_INVALID\_PARAMETER. | 1. Call CopyMem() with Width as **EfiPciWidthFifoUintX**. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.9.12 | 0xc07ea144, 0x18b5, 0x40e5, 0xa0, 0xa0, 0xd4, 0xca, 0x6c, 0x82, 0xc2, 0x9d | EFI\_PCI\_IO\_PROTOCOL.CopyMem - With Width as **EfiPciWidthFillUintX**, the return status is EFI\_INVALID\_PARAMETER. | 1. Call CopyMem() with Width as **EfiPciWidthFillUintX**. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.9.13 | 0x299293a3, 0xe8db, 0x43a4, 0x9b, 0x3f, 0x5e, 0x23, 0xb2, 0x9e, 0x37, 0x31 | EFI\_PCI\_IO\_PROTOCOL.CopyMem - With Source Address area out of BAR range, the return status is EFI\_UNSUPPORTED. | 1. Call CopyMem() with **Source** Address area out of BAR range. The return status must be EFI\_UNSUPPORTED. |
| 5.8.2.9.14 | 0x33c447ae, 0x5caf, 0x4904, 0xaf, 0x90, 0x66, 0x78, 0x17, 0x45, 0x0e, 0x12 | EFI\_PCI\_IO\_PROTOCOL.CopyMem - With Destination Address area out of BAR range, the return status is EFI\_UNSUPPORTED. | 1. Call CopyMem() with Destination Address area out of BAR range. The return status must be EFI\_UNSUPPORTED. |
| 5.8.2.9.15 | 0xe2dd0321, 0xac26, 0x4aac, 0xa6, 0x28, 0xc2, 0x59, 0xbc, 0x8a, 0xd5, 0x2c | EFI\_PCI\_IO\_PROTOCOL.CopyMem - With invalid Source BAR Index, return status is EFI\_UNSUPPORTED. | 1. Call CopyMem() with invalid Source BAR Index. The return status must be EFI\_UNSUPPORTED. |
| 5.8.2.9.16 | 0x110a96a1, 0x7a2e, 0x4eab, 0xbc, 0x11, 0xf3, 0xda, 0x30, 0xd0, 0xa2, 0xff | EFI\_PCI\_IO\_PROTOCOL.CopyMem - With invalid Destination, BAR Index return status is EFI\_UNSUPPORTED. | 1. Call CopyMem() with invalid Destination BAR Index. The return status must be EFI\_UNSUPPORTED. |
| 5.8.2.9.17 | 0x7d1c3de1, 0xa7b8, 0x4923, 0x94, 0x13, 0x76, 0x49, 0x05, 0x01, 0xf6, 0x9f | EFI\_PCI\_IO\_PROTOCOL.CopyMem - With Source BAR Index as an IO type BAR, the return status is EFI\_UNSUPPORTED. | 1. Call CopyMem() with Source BAR Index as an IO type BAR. The return status must be EFI\_UNSUPPORTED. |
| 5.8.2.9.18 | 0xaacfb1ec, 0xd6fb, 0x4c3a, 0xa4, 0x8c, 0x4a, 0xc2, 0x77, 0xfb, 0xc8, 0xe3 | EFI\_PCI\_IO\_PROTOCOL.CopyMem - With Destination BAR Index as an IO type BAR, the return status is EFI\_UNSUPPORTED. | 1. Call CopyMem() with Destination BAR Index as an IO type BAR. The return status must be EFI\_UNSUPPORTED. |
| 5.8.2.9.19 | 0x83b30e84, 0x528f, 0x420d, 0x87, 0x48, 0x7d, 0x96, 0x36, 0x8e, 0x33, 0x58 | EFI\_PCI\_IO\_PROTOCOL.CopyMem - With invalid Width, the return status is EFI\_INVALID\_PARAMETER. | 1. Call CopyMem() with invalid Width. The return status must be EFI\_INVALID\_PARAMETER. |

### Map()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.2.10.1 | 0x720e6fdc, 0x91c8, 0x4fd5, 0xb5, 0xde, 0xb1, 0xcc, 0x3b, 0x0c, 0x0c, 0xba | EFI\_PCI\_IO\_PROTOCOL.Map - Map() with Bus Master Read returns EFI\_SUCCESS. | 1. Allocate a buffer (4K + 1 Bytes).  2. Fill in the buffer with some data.  3. Call Map() with Operation as **EfiPciIoOperationBusMasterRead** to map this range to a new DMA capable location. The return status should be EFI\_SUCCESS. |
| 5.8.2.10.2 | 0xbf7f859c, 0x20e5, 0x4418, 0x8e, 0x21, 0x87, 0x60, 0x60, 0x58, 0x73, 0xa2 | EFI\_PCI\_IO\_PROTOCOL.Map - Map() with Bus Master Read, mapped bytes are > 0. | 1. Allocate a buffer (4K + 1 Bytes).  2. Fill in the buffer with some data.  3. Call Map() with Operation as **EfiPciIoOperationBusMasterRead** to map this range to a new DMA capable location.  4. Check if the number of bytes mapped great than 0. |
| 5.8.2.10.3 | 0xd56b3a96, 0x7c58, 0x4209, 0x85, 0xe9, 0x90, 0xb2, 0x07, 0x90, 0x6d, 0x55 | EFI\_PCI\_IO\_PROTOCOL.Map - Map() with Bus Master Read, the mapped area equals original area. | 1. Allocate a buffer (4K + 1 Bytes).  2. Fill in the buffer with some data.  3. Call Map() with Operation as **EfiPciIoOperationBusMasterRead** to map this range to a new DMA capable location.  4. Check if data of mapped area ishe same as the data of original area. |
| 5.8.2.10.4 | 0x5539608f, 0xed60, 0x4172, 0x94, 0x4e, 0xe9, 0x4a, 0x0f, 0x61, 0xf7, 0xe8 | EFI\_PCI\_IO\_PROTOCOL.Map - Map() with Bus Master Write returns EFI\_SUCCESS. | 1. Allocate a buffer (4K + 1 Bytes) .  2. Fill in the buffer with some data.  3. Call Map() with Operation as **EfiPciIoOperationBusMasterWrite** to map this range to a new DMA capable location  The return status should be EFI\_SUCCESS. |
| 5.8.2.10.5 | 0xb4019165, 0x7b45, 0x4ec4, 0xa7, 0xeb, 0xc5, 0x67, 0x71, 0x07, 0xd9, 0x4c | EFI\_PCI\_IO\_PROTOCOL.Map - Map() with Bus Master Write, mapped bytes are > 0. | 1. Allocate a buffer (4K + 1 Bytes).  2. Fill in the buffer with some data.  3. Call Map() with Operation as **EfiPciIoOperationBusMasterWrite** to map this range to a new DMA capable location.  4. Check if the number of bytes mapped great than 0. |
| 5.8.2.10.6 | 0x6b4e9c1e, 0xa1e7, 0x4cf5, 0x8d, 0x0f, 0xdd, 0x68, 0x80, 0xcd, 0x8f, 0x43 | EFI\_PCI\_IO\_PROTOCOL.Map - Map() with Bus Master Write, original data remains unchanged immediatelyafter mapping. | 1. Allocate a buffer (4K + 1 Bytes) .  2. Fill in the buffer with some data.  3. Call Map() with Operation as **EfiPciIoOperationBusMasterWrite** to map this range to a new DMA capable location.  4. Check if the data of the original area is unchanged. |
| 5.8.2.10.7 | 0x9a37eb62, 0x4bab, 0x4fce, 0x81, 0x9d, 0x0d, 0x80, 0x42, 0xea, 0x46, 0x7e | EFI\_PCI\_IO\_PROTOCOL.Map - Map() with Bus Master Common, the Buffer returns EFI\_SUCCESS. | 1. Allocate a buffer (4K + 1 Bytes)  2. Fill in the buffer with some data  3. Call Map() with Operation as **EfiPciIoOperationBusMasterCommon** to map this range to a new DMA capable location  The return status should be EFI\_SUCCESS. |
| 5.8.2.10.8 | 0x4d562d9c, 0xb028, 0x43ff, 0xb7, 0xfc, 0x92, 0xdb, 0x62, 0x40, 0xd5, 0x9a | EFI\_PCI\_IO\_PROTOCOL.Map - Map() with Bus Master Common, the Buffer mapped bytes are > 0. | 1. Allocate a buffer (4K + 1 Bytes) .  2. Fill in the buffer with some data.  3. Call Map() with Operation as **EfiPciIoOperationBusMasterCommon** to map this range to a new DMA capable location.  4. Check if the number of bytes mapped are greater than 0. |
| 5.8.2.10.9 | 0x8bd3ecc4, 0x43ea, 0x4f9e, 0x84, 0x79, 0x8c, 0x36, 0xde, 0x51, 0x13, 0x2f | EFI\_PCI\_IO\_PROTOCOL.Map - Map() with Bus Master Common, the Buffer mapped area equalsthe original area after mapping. | 1. Allocate a buffer (4K + 1 Bytes).  2. Fill in the buffer with some data.  3. Call Map() with Operation as **EfiPciIoOperationBusMasterCommon** to map this range to a new DMA capable location.  4. Check if the data of mapped areais is the same as the data of the original area. |
| 5.8.2.10.10 | 0x673d01f2, 0xdabf, 0x49bb, 0xbe, 0xc5, 0xe7, 0xa0, 0x3a, 0xd7, 0x71, 0xbc | EFI\_PCI\_IO\_PROTOCOL.Map - Map() with Bus Master Common, the Buffer data of the original area is sync’d with the mapped area. | 1. Allocate a buffer (4K + 1 Bytes).  2. Fill in the buffer with some data.  3. Call Map() with Operation as **EfiPciIoOperationBusMasterCommon** to map this range to a new DMA capable location.  4. Call SetMem() to fill in mapped address with some fixed data.  5. Check if the data of the original area is synchronized with the mapped area. |
| 5.8.2.10.11 | 0xbd5fcf21, 0xdb42, 0x4f4f, 0xb0, 0xfb, 0x56, 0x62, 0xd5, 0x1a, 0xba, 0x68 | EFI\_PCI\_IO\_PROTOCOL.Map - Map() with Bus Master Common, the Buffer data of the mapped area syncs with original area. | 1. Allocate a buffer (4K + 1 Bytes) .  2. Fill in the buffer with some data.  3. Call Map() with Operation as **EfiPciIoOperationBusMasterCommon** to map this range to a new DMA capable location.  4. Call SetMem() to fill in original address with some fixed data.  5. Check if the data of mapped area is synchronized with the original area. |
| 5.8.2.10.12 | 0xe2fa9ae5, 0xea93, 0x48b2, 0xba, 0x85, 0xa3, 0x74, 0xe2, 0xdb, 0xe2, 0xaf | EFI\_PCI\_IO\_PROTOCOL.Map - Mapping with Operation as **EfiPciIoOperationMaximum** returns a status of EFI\_INVALID\_PARAMETER. | 1. Call Map() with Operation as **EfiPciIoOperationMaximum**. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.10.13 | 0x3b337461, 0x98da, 0x4117, 0xab, 0xef, 0x57, 0x60, 0x34, 0xfd, 0xc6, 0x22 | EFI\_PCI\_IO\_PROTOCOL.Map - Mapping with Operation as **EfiPciIoOperationMaximum** + 1 returns a status of EFI\_INVALID\_PARAMETER. | 1. Call Map() with Operation as **EfiPciIoOperationMaximum** + 1. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.10.14 | 0xdce36bfb, 0xde48, 0x4f84, 0x9d, 0xc1, 0xde, 0x92, 0xa4, 0x40, 0x50, 0xbb | EFI\_PCI\_IO\_PROTOCOL.Map - Mapping with Operation as -1 returns a status of EFI\_INVALID\_PARAMETER. | 1. Call Map() with Operation as -1. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.10.15 | 0x8aa3c1cb, 0x5c8d, 0x4a74, 0x83, 0x81, 0x4b, 0x15, 0x3a, 0xf8, 0xff, 0x17 | EFI\_PCI\_IO\_PROTOCOL.Map - Mapping with HostAddress as NULL returns a status of EFI\_INVALID\_PARAMETER | 1. Call Map() with HostAddress as NULL. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.10.16 | 0x495cff3e, 0x5f7a, 0x4888, 0x85, 0x9f, 0xb7, 0x26, 0x0b, 0xb4, 0x18, 0xaf | EFI\_PCI\_IO\_PROTOCOL.Map - Mapping with NumberOfBytes as NULL returns a status of EFI\_INVALID\_PARAMETER | 1. Call Map() with NumberOfBytes as NULL. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.10.17 | 0x7e34b406, 0x0821, 0x4b95, 0xa4, 0x18, 0xc2, 0x0e, 0xfc, 0xfc, 0x00, 0xef | EFI\_PCI\_IO\_PROTOCOL.Map - Mapping with DeviceAddress as NULL returns a status of EFI\_INVALID\_PARAMETER | 1. Call Map() with DeviceAddress as NULL. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.10.18 | 0x6b450eae, 0x225c, 0x4ff1, 0x93, 0xd1, 0x55, 0xf9, 0xae, 0x35, 0x3e, 0xf8 | EFI\_PCI\_IO\_PROTOCOL.Map - Mapping with Mapping as NULL returns a status of EFI\_INVALID\_PARAMETER. | 1. Call Map() with Mapping as NULL. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.10.19 | 0x07a924a7, 0xe637, 0x4f46, 0x9b, 0x3c, 0x04, 0x63, 0x86, 0xfb, 0xf6, 0xf0 | EFI\_PCI\_IO\_PROTOCOL.Map - Mapping with HostAddress + NumberOfByte > 4G returns a status of EFI\_UNSUPPORTED. | 1. Call Map() with HostAddress + NumberOfByte > 4G. The return status must be EFI\_UNSUPPORTED. |

### Unmap()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.2.11.1 | 0xd9f80cd4, 0x8f0b, 0x4a27, 0x99, 0x16, 0x1a, 0x47, 0xfd, 0x8f, 0x07, 0x25 | EFI\_PCI\_IO\_PROTOCOL.Unmap - Unmap() area mapped wih **BusMasterRead** returns EFI\_SUCCESS | 1. Allocate a buffer (4K + 1 Bytes)  2. Fill in the buffer with some data  3. Call Map() with Operation as **EfiPciIoOperationBusMasterRead** to map this range to a new DMA capable location  4. Call Unmap() to release the mapped resources  The return status should be EFI\_SUCCESS |
| 5.8.2.11.2 | 0x8f86dbbf, 0xcc86, 0x40d0, 0x89, 0xb3, 0x97, 0xd6, 0xf1, 0xe8, 0xd7, 0x80 | EFI\_PCI\_IO\_PROTOCOL.Unmap - Unmap()leaves data in the original area mapped wih **BusMasterRead** unchangedafter Unmap | 1. Allocate a buffer (4K + 1 Bytes)  2. Fill in the buffer with some data  3. Call Map() with Operation as **EfiPciIoOperationBusMasterRead** to map this range to a new DMA capable location  4. Call Unmap() to release the mapped resources  5. Check if the data in original area remains unchanged |
| 5.8.2.11.3 | 0xab8555aa, 0x8c45, 0x4bec, 0x90, 0x9a, 0xad, 0xc7, 0xfe, 0xe9, 0xaf, 0xf4 | EFI\_PCI\_IO\_PROTOCOL.Unmap - Unmap() area mapped wih **BusMasterWrite** returns EFI\_SUCCESS | 1. Allocate a buffer (4K + 1 Bytes)  2. Fill in the buffer with some data  3. Call Map() with Operation as **EfiPciIoOperationBusMasterWrite** to map this range to a new DMA capable location  4. Call Unmap() to release the mapped resources  The return status should be EFI\_SUCCESS |
| 5.8.2.11.4 | 0xa6537c2a, 0x34bc, 0x4604, 0x81, 0x48, 0xb1, 0x41, 0x70, 0x46, 0x86, 0xe4 | EFI\_PCI\_IO\_PROTOCOL.Unmap - Unmap() leaves data in the original area mapped wih **BusMasterWrite** equal with the data written in mapped area | 1. Allocate a buffer (4K + 1 Bytes)  2. Fill in the buffer with some data  3. Call Map() with Operation as **EfiPciIoOperationBusMasterWrite** to map this range to a new DMA capable location  4. Call Unmap() to release the mapped resources  5. Check if the data in the original area is equal with the data in mapped area |
| 5.8.2.11.5 | 0x79009fa0, 0x5b72, 0x4e82, 0x84, 0x84, 0x3a, 0x21, 0xe0, 0x57, 0x93, 0xb9 | EFI\_PCI\_IO\_PROTOCOL.Unmap - Unmap() area mapped wih Bus Master Common Read returns EFI\_SUCCESS | 1. Allocate a buffer (4K + 1 Bytes)  2. Fill in the buffer with some data  3. Call Map() with Operation as **EfiPciIoOperationBusMasterCommon** to map this range to a new DMA capable location  4. Call Unmap() to release the mapped resources  The return status should be EFI\_SUCCESS |
| 5.8.2.11.6 | 0xda153716, 0xcd62, 0x4612, 0xae, 0x11, 0x71, 0x5e, 0x97, 0xeb, 0x6a, 0x9a | EFI\_PCI\_IO\_PROTOCOL.Unmap - Unmap() leaves data in the original area mapped wih Bus Master Common Read unchanged after Unmap | 1. Allocate a buffer (4K + 1 Bytes)  2. Fill in the buffer with some data  3. Call Map() with Operation as **EfiPciIoOperationBusMasterCommon** to map this range to a new DMA capable location  4. Call Unmap() to release the mapped resources  5. Check if the data in the original area remains unchanged |

### AllocateBuffer()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.2.12.1 | 0x841e89ab, 0x9c60, 0x48e5, 0xae, 0x7d, 0x51, 0x21, 0xf5, 0x08, 0xe1, 0x0c | EFI\_PCI\_IO\_PROTOCOL.AllocateBuffer - AllocateBuffer() with correct Parameter status returns EFI\_SUCCESS | 1. Call AllocateBuffer() with the following parameters having multiple enumerated values:  **MemoryType** – **EfiBootServicesData** and **EfiRuntimeServicesData**  **Attributes** – 0, EFI\_PCI\_ATTRIBUTE\_MEMORY\_WRITE\_COMBINE EFI\_PCI\_ATTRIBUTE\_MEMORY\_CACHED, and EFI\_PCI\_ATTRIBUTE\_MEMORY\_WRITE\_COMBINE | EFI\_PCI\_ATTRIBUTE\_MEMORY\_CACHED  The return status should be EFI\_SUCCESS |
| 5.8.2.12.2 | 0x576894ad, 0x9229, 0x4078, 0xa9, 0x69, 0x70, 0x0e, 0x6e, 0x04, 0x4b, 0xb3 | EFI\_PCI\_IO\_PROTOCOL.AllocateBuffer - With invalid memory type the status is EFI\_INVALID\_PARAMETER | 1. Call AllocateBuffer() with invalid memory type. The return status must be EFI\_INVALID\_PARAMETER |
| 5.8.2.12.3 | 0xa0c5c95e, 0xf251, 0x4c00, 0x9f, 0xdf, 0x9c, 0x88, 0xa2, 0xaa, 0x45, 0x6b | EFI\_PCI\_IO\_PROTOCOL.AllocateBuffer - With HostAddress as NULL the status is EFI\_INVALID\_PARAMETER | 1. Call AllocateBuffer() with HostAddress as NULL. The return status must be EFI\_INVALID\_PARAMETER |
| 5.8.2.12.4 | 0xfacb1e1b, 0x0327, 0x4341, 0xa9, 0x42, 0x4d, 0xb9, 0x9f, 0x1d, 0xe5, 0x68 | EFI\_PCI\_IO\_PROTOCOL.AllocateBuffer - With invalid Attributes the status is EFI\_UNSUPPORTED | 1. Call AllocateBuffer() with invalid Attributes. The return status must be EFI\_UNSUPPORTED |

### 

### FreeBuffer()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.2.13.1 | 0x00312f50, 0x721c, 0x4085, 0x82, 0x63, 0x04, 0xd1, 0x1f, 0x37, 0x2c, 0x6c | EFI\_PCI\_IO\_PROTOCOL.FreeBuffer - FreeBuffer() return status is EFI\_SUCCESS | 1. Call AllocateBuffer() with the following parameters having multiple enumerated values:  **MemoryType** – **EfiBootServicesData** and **EfiRuntimeServicesData**  **Attributes** – 0, EFI\_PCI\_ATTRIBUTE\_MEMORY\_WRITE\_COMBINE EFI\_PCI\_ATTRIBUTE\_MEMORY\_CACHED, and EFI\_PCI\_ATTRIBUTE\_MEMORY\_WRITE\_COMBINE | EFI\_PCI\_ATTRIBUTE\_MEMORY\_CACHED  2. Call FreeBuffer()  The return status should be EFI\_SUCCESS |

### Flush()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.2.14.1 | 0x2c9f36a3, 0x4cab, 0x4434, 0xa8, 0xc1, 0x7b, 0xf6, 0x3c, 0x46, 0x8f, 0x05 | EFI\_PCI\_IO\_PROTOCOL.Flush - Flush() return status is EFI\_SUCCESS | 1. Call Flush()  The return status should be EFI\_SUCCESS |

### GetLocation()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.2.15.1 | 0xfb478a8e, 0x58e2, 0x41b9, 0x89, 0x35, 0x71, 0x7b, 0x5a, 0x90, 0xa1, 0x84 | EFI\_PCI\_IO\_PROTOCOL.GetLocation - GetLocation() return status is EFI\_SUCCESS | 1. Call GetLocation()  The return status should be EFI\_SUCCESS |
| 5.8.2.15.2 | 0x07b74ac9, 0x96f4, 0x4d00, 0x94, 0xbd, 0x09, 0x60, 0xd4, 0xe9, 0xa6, 0xe7 | EFI\_PCI\_IO\_PROTOCOL.GetLocation - GetLocation()returns a BusNumber < 256 | 1. Call GetLocation()  2. Check if the returned BusNumber is less than 256 |
| 5.8.2.15.3 | 0xaf7155de, 0x45f4, 0x4b97, 0xb4, 0xac, 0x07, 0x1a, 0x53, 0x43, 0x32, 0x48 | EFI\_PCI\_IO\_PROTOCOL.GetLocation - GetLocation()returns a DeviceNumber < 32 | 1. Call GetLocation()  2. Check if the returned DeviceNumber is less than 32 |
| 5.8.2.15.4 | 0x838f7bf6, 0xfa36, 0x4149, 0x92, 0x29, 0xce, 0x60, 0x8a, 0x66, 0x35, 0x61 | EFI\_PCI\_IO\_PROTOCOL.GetLocation - GetLocation()returns a FunctionNumber < 8 | 1. Call GetLocation()  2. Check if the returned FunctionNumber is less than 8 |
| 5.8.2.15.5 | 0xa5510fe8, 0x2178, 0x47e6, 0x9e, 0xcc, 0xe9, 0x0b, 0x92, 0xcf, 0x1b, 0xbb | EFI\_PCI\_IO\_PROTOCOL.GetLocation - With SegmentNumber as NULL, the status is EFI\_INVALID\_PARAMETER | 1. Call GetLocation() with SegmentNumber as NULL. The return status must be EFI\_INVALID\_PARAMETER |
| 5.8.2.15.6 | 0x2a1ff8b2, 0xc540, 0x4f12, 0x9c, 0x06, 0x36, 0x8d, 0x45, 0x7c, 0x02, 0x7c | EFI\_PCI\_IO\_PROTOCOL.GetLocation - With BusNumber as NULL, the status is EFI\_INVALID\_PARAMETER | 1. Call GetLocation() with BusNumber as NULL. The return status must be EFI\_INVALID\_PARAMETER |
| 5.8.2.15.7 | 0x5e74e7e0, 0x36b0, 0x4c5d, 0x88, 0xb8, 0xb7, 0x52, 0xad, 0x2c, 0xbf, 0x61 | EFI\_PCI\_IO\_PROTOCOL.GetLocation - With DeviceNumber as NULL, the status is EFI\_INVALID\_PARAMETER | 1. Call GetLocation() with DeviceNumber as NULL. The return status must be EFI\_INVALID\_PARAMETER |
| 5.8.2.15.8 | 0xb37cb86c, 0xdd05, 0x4082, 0xa6, 0xf1, 0x8c, 0xf9, 0xc3, 0x46, 0x77, 0x7a | EFI\_PCI\_IO\_PROTOCOL.GetLocation - With FunctionNumber as NULL, the status is EFI\_INVALID\_PARAMETER | 1. Call GetLocation() with FunctionNumber as NULL. The return status must be EFI\_INVALID\_PARAMETER |

### Attributes()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.2.16.1 | 0x33ca89a5, 0xefa8, 0x4f52, 0x84, 0xf6, 0x2e, 0x95, 0x26, 0x23, 0xb0, 0xe1 | EFI\_PCI\_IO\_PROTOCOL.Attributes - Call Attributes() get current attribute status must be EFI\_SUCCESS | 1. Call Attributes() with **EfiPciIoAttributeOperationGet** to get the current attributes of the PCI controller  The return status should be EFI\_SUCCESS |
| 5.8.2.16.2 | 0xa11652df, 0x8818, 0x4a05, 0xbe, 0xd9, 0x27, 0xf9, 0xe5, 0xad, 0x78, 0x3c | EFI\_PCI\_IO\_PROTOCOL.Attributes - Call Attributes() get supported attribute status must be EFI\_SUCCESS | 1. Call Attributes() with **EfiPciIoAttributeOperationSupported** to get the supported attributes of the PCI controller  The return status should be EFI\_SUCCESS |
| 5.8.2.16.3 | 0x69ce5213, 0x7180, 0x4beb, 0x9f, 0x39, 0x1d, 0x1f, 0x17, 0x00, 0x59, 0x9a | EFI\_PCI\_IO\_PROTOCOL.Attributes - Current attributes should in supported attributes | 1. Call Attributes() with **EfiPciIoAttributeOperationGet** to get the current attributes of the PCI controller  2. Call Attributes() with **EfiPciIoAttributeOperationSupported** to get the supported attributes of the PCI controller  3. Check if the current attributes is a subset of Supported attributes |
| 5.8.2.16.4 | 0xfac8ddb3, 0xbfae, 0x40ff, 0xb7, 0x31, 0x26, 0x8e, 0x58, 0x29, 0x25, 0xb0 | EFI\_PCI\_IO\_PROTOCOL.Attributes - Call Attributes() set Attributes as Supported attributes return status must be EFI\_SUCCESS | 1. Call Attributes() with **EfiPciIoAttributeOperationSet** with a supported attribute of the PCI controller.  The return status should be EFI\_SUCCESS |
| 5.8.2.16.5 | 0xf8e48da6, 0x72e2, 0x4905, 0xa7, 0x19, 0xe3, 0xa5, 0x77, 0xca, 0xa2, 0xa8 | EFI\_PCI\_IO\_PROTOCOL.Attributes - Set Attributes as supported attributes the attributes should really be cleared | 1. Call Attributes() with **EfiPciIoAttributeOperationSet** with a supported attribute of the PCI controller.  2. Call Attributes() with EfiPciIoAttributeOperationGet to get the attributes of the PCI controller  3. Check if the gotten attributes is the same as the set ones. |
| 5.8.2.16.6 | 0x02cab1a9, 0x4be9, 0x4c47, 0xb2, 0x75, 0xca, 0xed, 0x59, 0x62, 0x1f, 0x41 | EFI\_PCI\_IO\_PROTOCOL.Attributes - Call Attributes() set Attributes as 0 return status must be EFI\_SUCCESS | 1. Call Attributes() with **EfiPciIoAttributeOperationSet** with an **attribute** value of 0 to clear all attributes. The return status should be EFI\_SUCCESS |
| 5.8.2.16.7 | 0x88791167, 0xb9f3, 0x42ae, 0x84, 0xd1, 0xa1, 0xb6, 0xd3, 0xeb, 0xb8, 0x2f | EFI\_PCI\_IO\_PROTOCOL.Attributes - Set Attributes as 0 the attributes should really be cleared | 1. Call Attributes() with **EfiPciIoAttributeOperationSet** with an Attributes value of 0 to clear all attributes.  2. Call Attributes() with EfiPciIoAttributeOperationGet to get the attributes of the PCI controller  3. Check if the gotten attributes is the same as that of set |
| 5.8.2.16.8 | 0x04479c23, 0xc700, 0x439f, 0xb7, 0x42, 0x91, 0x9a, 0x6b, 0x2e, 0x71, 0x5a | EFI\_PCI\_IO\_PROTOCOL.Attributes - Call Attributes() enable Attributes as original attributes return status must be EFI\_SUCCESS | 1. Call Attributes() with **EfiPciIoAttributeOperationEnable** with supported attributes  The return status should be EFI\_SUCCESS |
| 5.8.2.16.9 | 0x1d011f3e, 0xaa23, 0x4b0b, 0xb1, 0x65, 0x8f, 0x6f, 0x21, 0xf3, 0x85, 0x6d | EFI\_PCI\_IO\_PROTOCOL.Attributes - enable Attributes as original attributes the attributes should really be **Enabled** | 1. Call Attributes() with **EfiPciIoAttributeOperationEnable** with supported attributes  2. Call Attributes() with **EfiPciIoAttributeOperationGet** to get the attributes  3. Check if the attributes value is the same as original attributes |
| 5.8.2.16.10 | 0x35e690e9, 0xd037, 0x41a1, 0x93, 0x44, 0x86, 0x78, 0x02, 0xe2, 0x37, 0xfc | EFI\_PCI\_IO\_PROTOCOL.Attributes - Call Attributes() disable original attributes return status must be EFI\_SUCCESS | 1. Call Attributes() with EfiPciIoAttributeOperationDisable with supported attributes  The return status should be EFI\_SUCCESS |
| 5.8.2.16.11 | 0xb7376265, 0xfb7f, 0x410c, 0x99, 0xb5, 0x5b, 0x17, 0x37, 0x41, 0xf7, 0x03 | EFI\_PCI\_IO\_PROTOCOL.Attributes - Disable original attributes the attributes should really be disabled | 1. Call Attributes() with EfiPciIoAttributeOperationDisable with supported attributes  2. Call Attributes() with **EfiPciIoAttributeOperationGet** to get the attributes  3. Check if the attributes is 0 |
| 5.8.2.16.12 | 0x00c4352a, 0x0747, 0x4175, 0x8d, 0xa6, 0xd1, 0xad, 0xc7, 0x30, 0x31, 0xf4 | EFI\_PCI\_IO\_PROTOCOL.Attributes - Call Attributes() set original attributes return status must be EFI\_SUCCESS | 1. Call Attributes() with **EfiPciIoAttributeOperationSet** with original attributes  The return status should be EFI\_SUCCESS |
| 5.8.2.16.13 | 0x7ba1d37a, 0xa654, 0x4738, 0x96, 0x98, 0x11, 0x1b, 0x4b, 0x43, 0xad, 0x6c | EFI\_PCI\_IO\_PROTOCOL.Attributes - Set original attributes the attributes should really be set | 1. Call Attributes() with **EfiPciIoAttributeOperationSet** with original attributes  2. Call Attributes() with **EfiPciIoAttributeOperationGet** to get the attributes  3. Check if the attributes is the same as original attributes |
| 5.8.2.16.14 | 0xca3478fa, 0x7a9a, 0x4452, 0x93, 0x23, 0x98, 0xda, 0xe1, 0xf9, 0x17, 0xde | EFI\_PCI\_IO\_PROTOCOL.Attributes - With Operation as **EfiPciIoAttributeOperationMaximum** status must be EFI\_INVALID\_PARAMETER | 1. Call Attributes() with Operation as **EfiPciIoAttributeOperationMaximum**. The return status must be EFI\_INVALID\_PARAMETER |
| 5.8.2.16.15 | 0xf09e9c22, 0xd061, 0x4a52, 0xa6, 0xea, 0xa9, 0x4a, 0x90, 0x2e, 0x15, 0x0e | EFI\_PCI\_IO\_PROTOCOL.Attributes - With Operation as **EfiPciIoAttributeOperationMaximum** + 1 status must be EFI\_INVALID\_PARAMETER | 1. Call Attributes() with Operation as **EfiPciIoAttributeOperationMaximum** + 1. The return status must be EFI\_INVALID\_PARAMETER |
| 5.8.2.16.16 | 0x1a5371a2, 0x9f8f, 0x4a0a, 0x90, 0x3c, 0x61, 0xca, 0xf0, 0x47, 0xc4, 0x30 | EFI\_PCI\_IO\_PROTOCOL.Attributes - With Operation as -1 the status must be EFI\_INVALID\_PARAMETER | 1. Call Attributes() with Operation as -1. The return status must be EFI\_INVALID\_PARAMETER |
| 5.8.2.16.17 | 0x63c39f67, 0xb02f, 0x4f78, 0x88, 0x49, 0x63, 0x3a, 0xa9, 0x0b, 0xfd, 0xd8 | EFI\_PCI\_IO\_PROTOCOL.Attributes - With Operation as EfiPciIoAttributeOperationGet and Result as NULL then the status must be EFI\_INVALID\_PARAMETER | 1. Call Attributes() with Operation as **EfiPciIoAttributeOperationGet** and Result as NULL. The return status must be EFI\_INVALID\_PARAMETER |
| 5.8.2.16.18 | 0xacfb1410, 0x3824, 0x42f0, 0x89, 0xfe, 0x93, 0x0c, 0xda, 0xb7, 0xe0, 0x3a | EFI\_PCI\_IO\_PROTOCOL.Attributes - With Operation as **EfiPciIoAttributeOperationSupported** and Result as NULL, status is EFI\_INVALID\_PARAMETER | 1. Call Attributes() with Operation as **EfiPciIoAttributeOperationSupported** and Result as NULL. The return status must be EFI\_INVALID\_PARAMETER |
| 5.8.2.16.19 | 0xabcd2d94, 0x9389, 0x49a5, 0x91, 0xd7, 0x91, 0x83, 0x0b, 0x80, 0xfe, 0xc2 | EFI\_PCI\_IO\_PROTOCOL.Attributes - Setting unsupported Attributes returns a status of EFI\_UNSUPPORTED | 1. Find unsupported attributes by this device  2. Call Attributes() with Operation as **EfiPciIoAttributeOperationSet** and unsupported Attributes. The return status must be EFI\_UNSUPPORTED |
| 5.8.2.16.20 | 0xdbe5ef54, 0x5b5e, 0x45e8, 0x9f, 0x8b, 0x9d, 0xa5, 0x72, 0xdb, 0xcd, 0xb7 | EFI\_PCI\_IO\_PROTOCOL.Attributes - Enabling unsupported Attributes returns a status of EFI\_UNSUPPORTED | 1. Find unsupported attributes by this device  2. Call Attributes() with Operation as **EfiPciIoAttributeOperationEnable** and unsupported Attributes. The return status must be EFI\_UNSUPPORTED |
| 5.8.2.16.21 | 0x781416ce, 0xc545, 0x4542, 0xb5, 0xd8, 0xbc, 0xc0, 0xc4, 0xe0, 0x2a, 0x52 | EFI\_PCI\_IO\_PROTOCOL.Attributes - Disabling unsupported Attributes returns a status of EFI\_UNSUPPORTED | 1. Find unsupported attributes by this device  2. Call Attributes() with Operation as **EfiPciIoAttributeOperationDisable** and unsupported Attributes. The return status must be EFI\_UNSUPPORTED |

### GetBarAttributes()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.2.17.1 | 0xbc76b1a7, 0x767b, 0x4f5c, 0x94, 0x03, 0x34, 0x40, 0xfb, 0xd9, 0x40, 0x95 | EFI\_PCI\_IO\_PROTOCOL.GetBarAttributes - Calling GetBarAttributes() returns a status of EFI\_SUCCESS. | 1. Call GetBarAttributes() with a valid BAR Index and a valid Resources pointer.  The return status should be EFI\_SUCCESS. |
| 5.8.2.17.2 | 0x8414d9a1,0x0339,  0x4d7c,  0xa2,0xa4,  0x45,0x3d,  0xd6,0x8d,  0x6b,0x5f | EFI\_PCI\_IO\_PROTOCOL.GetBarAttributes - Calling GetBarAttributes() with only Supports is NULL returns status of EFI\_SUCCESS. | 1. Call GetBarAttributes() with a valid BAR Index and NULL Supports.  The return status should be EFI\_SUCCESS. |
| 5.8.2.17.3 | 0x211c1b15, 0xc4ce, 0x452d, 0x96, 0x93, 0xec, 0xf4, 0xc2, 0x3d, 0x20, 0xfe | EFI\_PCI\_IO\_PROTOCOL.GetBarAttributes - Calling GetBarAttributes() with only Resource is NULL returns a status of EFI\_SUCCESS. | 1. Call GetBarAttributes() with a valid BAR Index and NULL Resources pointer.  The return status should be EFI\_SUCCESS. |
| 5.8.2.17.4 | 0xcb909d56, 0x1d18, 0x44b5, 0xb0, 0x30, 0xa2, 0x58, 0x30, 0x9e, 0xd6, 0x6c | EFI\_PCI\_IO\_PROTOCOL.GetBarAttributes - The Resource Descriptor List is valid. | 1. Call GetBarAttributes() with a valid BAR Index and a valid Resources pointer.  2. Check that the returned resource descriptor is valid. |
| 5.8.2.17.5 | 0xc0d61a6d, 0x5d07, 0x4748, 0x9f, 0x14, 0x78, 0x00, 0xb6, 0xcf, 0x4b, 0x47 | EFI\_PCI\_IO\_PROTOCOL.GetBarAttributes - The attributes are in Device Supported Attributes. | 1. Call GetBarAttributes() with a valid BAR Index and a valid Resources pointer.  2. Call Attributes() with **EfiPciIoAttributeOperationSupported** to get the supported attributes of the PCI controller.  3. Check that the current attributes are a subset of Supported attributes. |
| 5.8.2.17.6 | 0x50f8ec56, 0xc28c, 0x417c, 0x8f, 0x43, 0x43, 0xfd, 0xfc, 0xbd, 0x4e, 0xdf | EFI\_PCI\_IO\_PROTOCOL.GetBarAttributes - With invalid BAR Index the status is EFI\_UNSUPPORTED. | 1. Call GetBarAttributes() with invalid BAR Index. The return status must be EFI\_UNSUPPORTED. |
| 5.8.2.17.7 | 0xf52eed93, 0x6c9d, 0x4008, 0xad, 0x9d, 0xe9, 0xab, 0xc8, 0xa4, 0x88, 0x01 | EFI\_PCI\_IO\_PROTOCOL.GetBarAttributes - With both Supports and Resources as NULL status is EFI\_INVALID\_PARAMETER. | 1. Call GetBarAttributes() with both Supports and Resources as NULL. The return status must be EFI\_INVALID\_PARAMETER. |

### SetBarAttributes()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.8.2.18.1 | 0x51ec0763, 0x0edb, 0x4ad3, 0xb1, 0x0c, 0x2d, 0x3f, 0x88, 0x34, 0x78, 0x44 | EFI\_PCI\_IO\_PROTOCOL.SetBarAttributes - Calling SetBarAttributes() returns a status of EFI\_SUCCESS. | 1. Call GetBarAttributes() with a valid BAR Index to get the BAR supported attributes resource.  2. Call SetBarAttributes() with BAR Supported attributes and resource information.  The return status should be EFI\_SUCCESS. |
| 5.8.2.18.2 | 0x9cbd1e01, 0x86a4, 0x4b9f, 0xbb, 0x00, 0x3e, 0xff, 0xfb, 0x35, 0xf3, 0xbd | EFI\_PCI\_IO\_PROTOCOL.SetBarAttributes - With invalid BAR Index, the status is EFI\_UNSUPPORTED. | 1. Call SetBarAttributes() with invalid BAR Index. The return status must be EFI\_UNSUPPORTED. |
| 5.8.2.18.3 | 0x445e37a9, 0xc8e7, 0x402b, 0xb7, 0xf8, 0x93, 0x96, 0xa0, 0xbd, 0x5e, 0xc5 | EFI\_PCI\_IO\_PROTOCOL.SetBarAttributes - With Offset as NULL, the status is EFI\_INVALID\_PARAMETER. | 1. Call SetBarAttributes() with Offset as NULL. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.18.4 | 0x32edd10b, 0x4a81, 0x4a98, 0x8b, 0x7a, 0xef, 0x1b, 0x9a, 0xe8, 0x25, 0x69 | EFI\_PCI\_IO\_PROTOCOL.SetBarAttributes - With Length as NULL the status is EFI\_INVALID\_PARAMETER. | 1. Call SetBarAttributes() with Length as NULL. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.8.2.18.5 | 0xfbb0d8fc, 0xffcf, 0x4562, 0xba, 0x86, 0x1f, 0x9b, 0x41, 0x45, 0x1f, 0x9c | EFI\_PCI\_IO\_PROTOCOL.SetBarAttributes - With Offset + Length out of the BAR resource range, the status is EFI\_UNSUPPORTED. | 1. Call SetBarAttributes() with Offset + Length out of the BAR resourcde range. The return status must be EFI\_UNSUPPORTED. |
| 5.8.2.18.6 | 0x48602f8b, 0xbb69, 0x4421, 0xb0, 0x21, 0x5a, 0x10, 0x78, 0x5b, 0xba, 0xf9 | EFI\_PCI\_IO\_PROTOCOL.SetBarAttributes - With unsupported Attributes the status is EFI\_UNSUPPORTED. | 1. Find unsupported attributes by this device  2. Call SetBarAttributes() with unsupported Attributes. The return status must be EFI\_UNSUPPORTED. |