Microsoft

winhec

Workshop Taipei 2016

Any questions please contact winhectpe@microsoft.com

Microsoft

Access Hardware Buses in User Mode

Devin Wong
Software Engineer

Partner Enablement Team

Contents

Low power buses on Windows

GPIO

- Used as an interrupt or for general I/O
- Buttons, switches, lights, system wake, etc.
- Interrupts for SPB, serial, and other buses

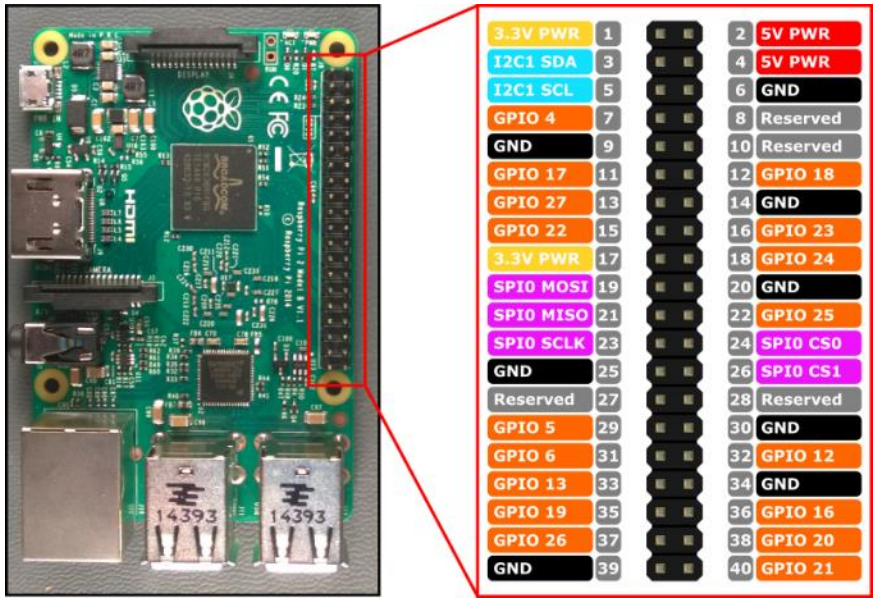
I^2C, SPI

- Simple Peripheral Buses (SPB) = I^2C, SPI
- I^2C: 100Kbps-3.4Mbps, SPI: Up to 160Mbps
- Typically used for:
  - Input / HID
  - Radios
  - Sensors
  - Power management

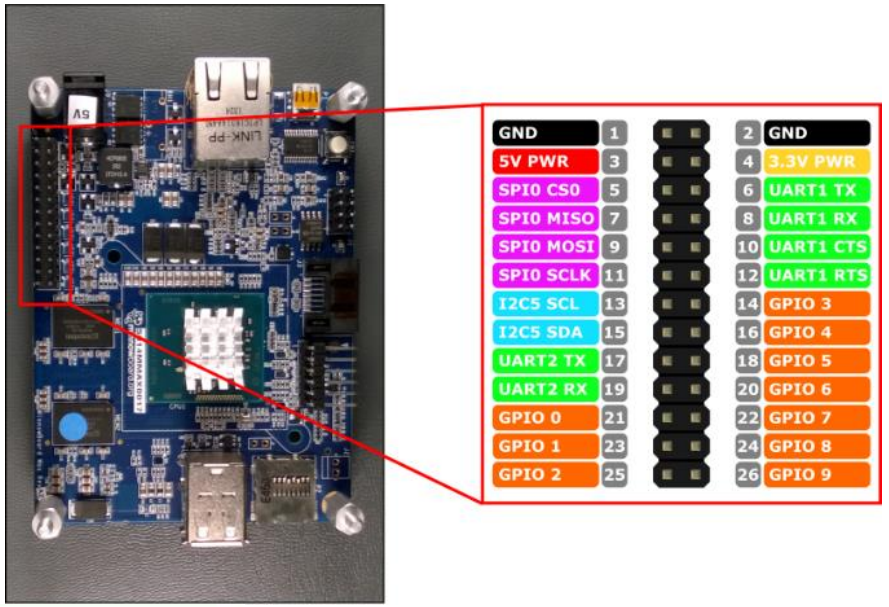
UART

- High Speed UART
- Up to 20Mbps
- Typically used for:
  - Bluetooth,
  - GPS

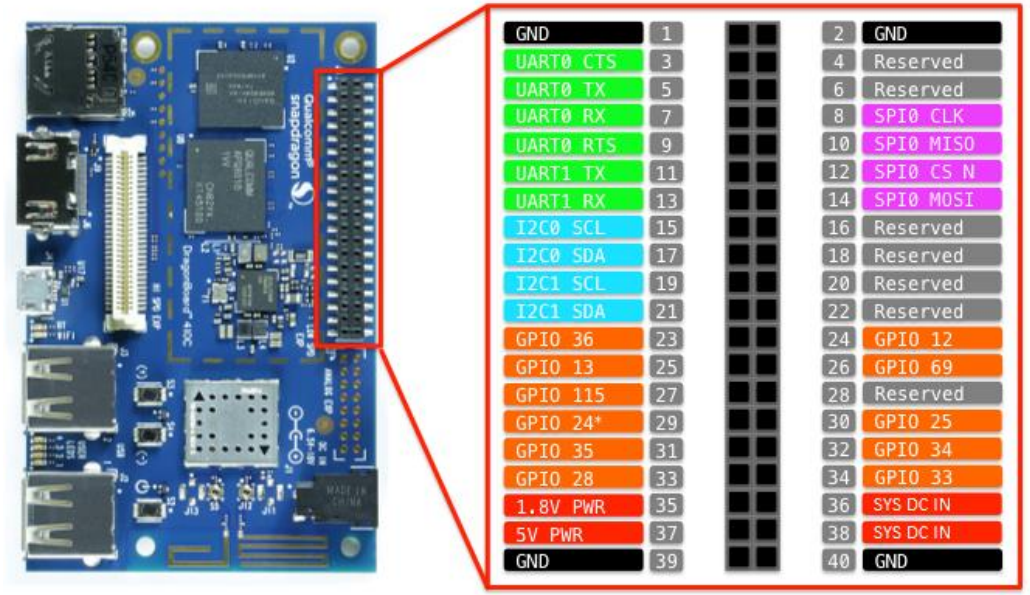
Hardware Pins on a dev board



Raspberry Pi 2

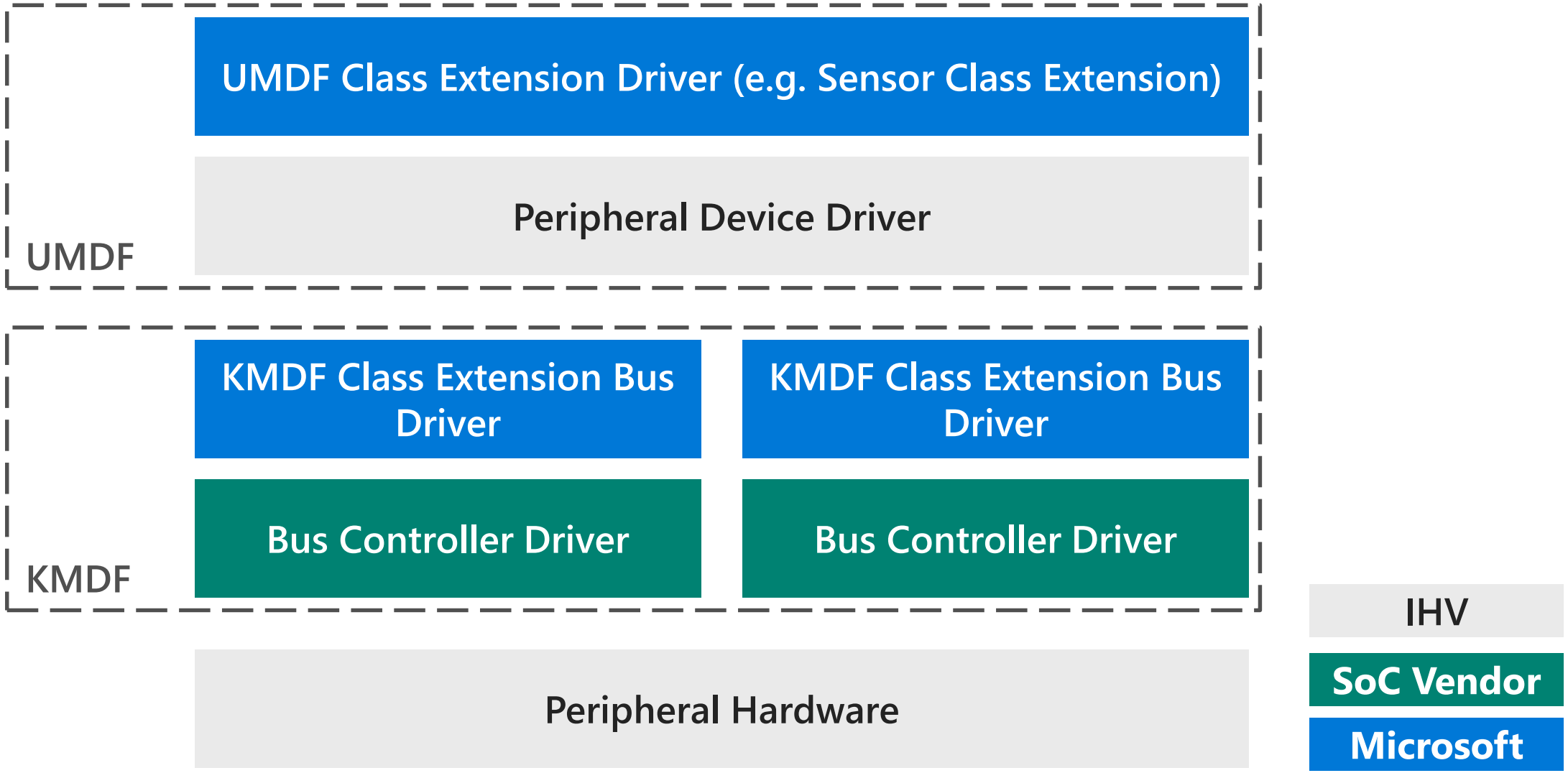


MinnowBoard MAX

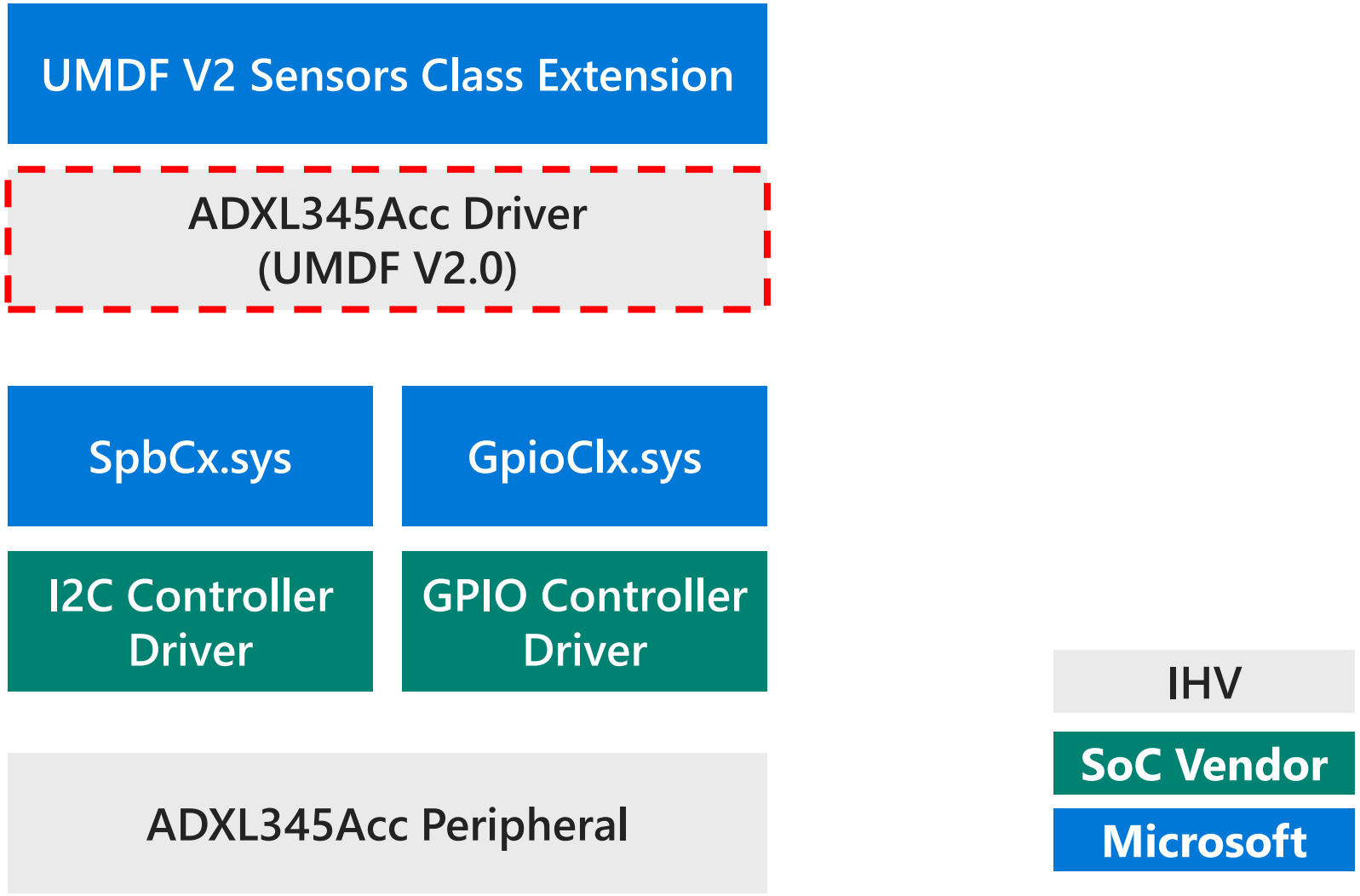


Qualcomm Snapdragon 410

Typical device driver architecture in desktop



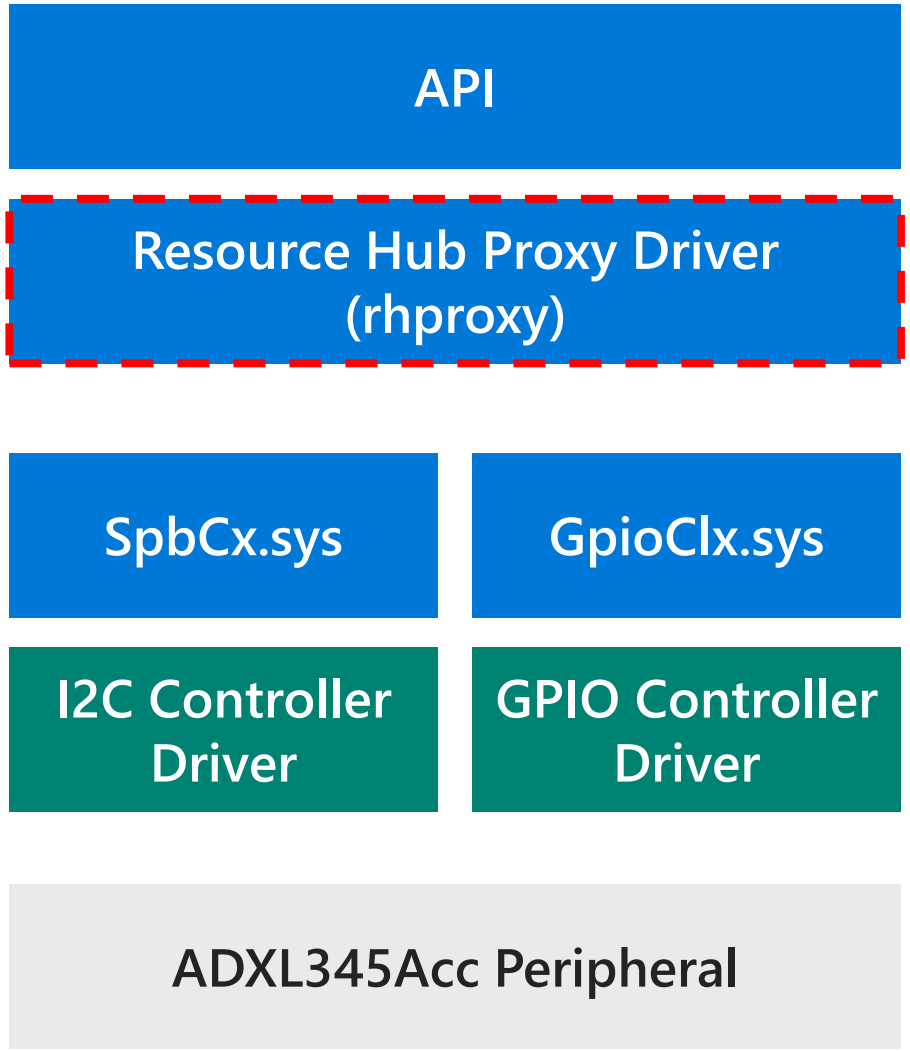
Example: Accelerometer sensor in desktop



It becomes easier in Win 10 IoT

- Windows 10 IoT Core contains new APIs for accessing GPIO, I2C, SPI, and UART directly from user mode
- New driver called *rhProxy* exposes *GpioClx* and *SpbCx* resources to user mode
- No additional drivers needed
- Detailed document “Enabling Usermode Access to GPIO, I2C, SPI, UART” by Jordan Rhee can be obtained from Connect site.

Example: Accelerometer sensor in Win 10 IoT

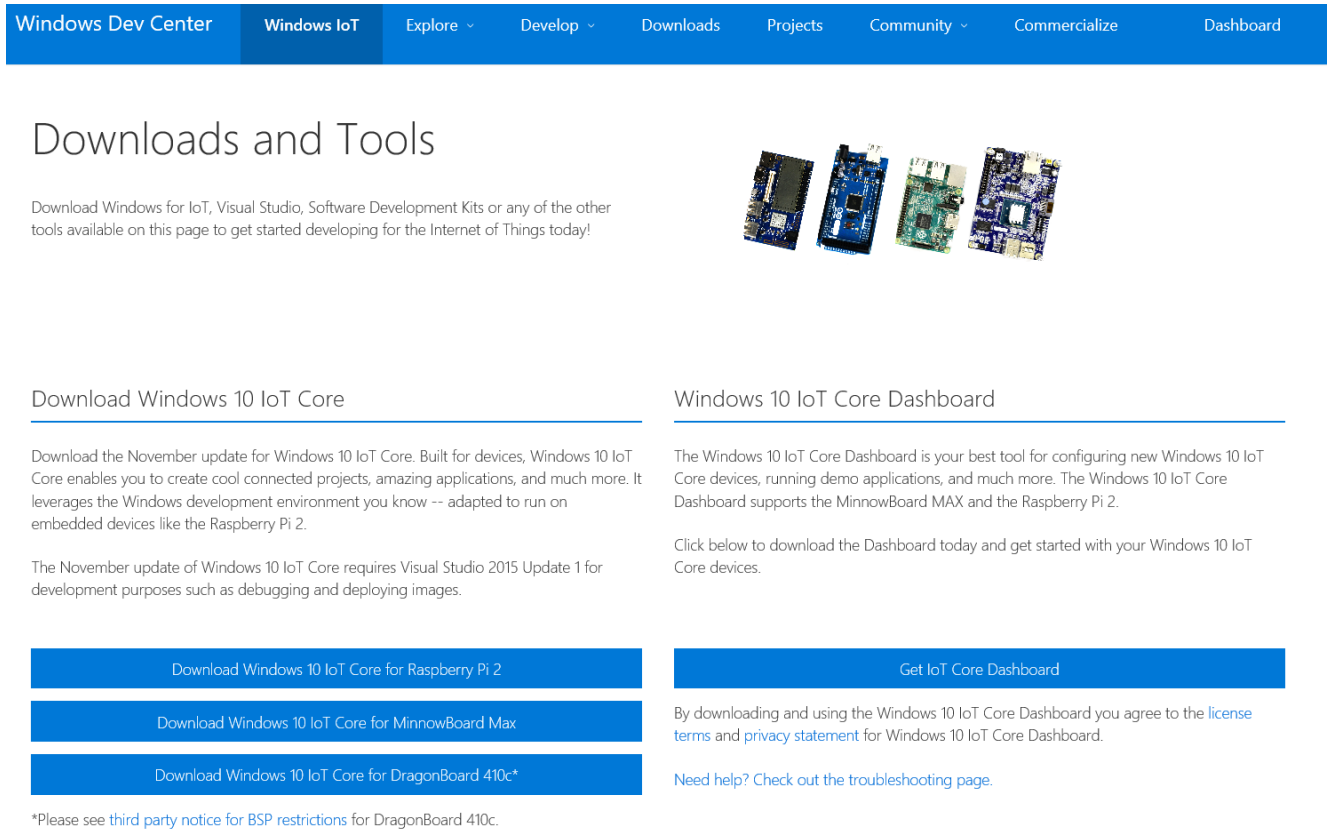


End-to-End Example on Raspberry Pi 2 and Minnowboard MAX

Downloads and Tools

http://ms-iot.github.io/content/en-US/Downloads.htm

Download Windows 10 IoT Core, Visual Studio with SDK, WDK or any of the other tools available



ACPI Device Declaration

Rhproxy device node declaration on Raspberry Pi 2

```
Scope (\_SB)
    {
     Device(RHPX)
        {
            Name(_HID, "MSFT8000")
            Name(_CID, "MSFT8000")
            Name(_UID, 1)
…
```

_HID – Hardware Id. Set this to a vendor-specific hardware ID.
_CID – Compatible Id. Must be “MSFT8000”.
_UID – Unique Id. Set to 1.

Declaring SPI

Raspberry Pi has one exposed SPI buses

SPI0 has two hardware chip select lines One SPISerialBus() resource declaration is required for each chip select line for each bus

The mapping between bus friendly name and resource index is specified in the DSD(Device Specific Data)

SPI0 – with 2 chip select lines

```
// Index 0
SPISerialBus(                   // SCKL - GPIO 11 - Pin 23
                                // MOSI - GPIO 10 - Pin 19
                                // MISO - GPIO 9  - Pin 21
                                // CE0  - GPIO 8  - Pin 24
    0,                          // Device selection (CE0)
    PolarityLow,                // Device selection polarity
    FourWireMode,               // wiremode
    0,                          // databit len: placeholder
    ControllerInitiated,        // slave mode
    0,                          // conn. speed: placeholder
    ClockPolarityLow,           // clock polarity: placeholde
    ClockPhaseFirst,            // clock phase: placeholder
    "\\_SB.SPI0",               // RscSource: SPI bus ctrl na
    0,                          // ResourceSourceIndex
                                // Resource usage
    )                           // Vendor Data
```

```
// Index 1
SPISerialBus(                   // SCKL - GPIO 11 - Pin 23
                                // MOSI - GPIO 10 - Pin 19
                                // MISO - GPIO 9  - Pin 21
                                // CE1  - GPIO 7  - Pin 26
    1,                          // Device selection (CE1)
    PolarityLow,                // Device selection polarity
    FourWireMode,               // wiremode
    0,                          // databit len: placeholder
    ControllerInitiated,        // slave mode
    0,                          // conn. speed: placeholder
    ClockPolarityLow,           // clock polarity: placeholder
    ClockPhaseFirst,            // clock phase: placeholder
    "\\_SB.SPI0",               // RscSource: SPI bus ctrl name
    0,                          // ResourceSourceIndex
                                // Resource usage
    )                           // Vendor Data
```

SPI0 Corresponding DSD settings

Package(2) { "bus-SPI-SPI0", Package() { 0, 1 }},

//creates a bus named "*SPI0*" with two chip select lines – resource indexes 0 and 1

Package(2) { "SPI0-MinClockInHz", 7629 },

Package(2) { "SPI0-MaxClockInHz", 125000000 },

//the *MinClockInHz* and *MaxClockInHz* properties specify the minimum and maximum clock speeds that are supported by the controller

Package(2) { "SPI0-SupportedDataBitLengths", Package() { 8 }},

//the *SupportedDataBitLengths* property lists the data bit lengths supported by the controller

Declaring I2C

+ Corresponding DSD

- Next, we declare the I2C resources.
  - Raspberry Pi 2 exposes a single I2C bus on pins 3 and 5.

```
// Index 3
I2CSerialBus(                   // Pin 3 (GPIO2, SDA1), 5 (GPIO3, SCL1)
    0xFFFF,                     // SlaveAddress: placeholder
    ,                           // SlaveMode: default to
ControllerInitiated
    0,                          // ConnectionSpeed: placeholder
    ,                           // Addressing Mode: placeholder
    "\\_SB.I2C1",               // ResourceSource: I2C bus controller
name
    ,
    ,
    )                           // VendorData
```

Declaring GPIO

- Next, we declare all the GPIO pins

-Declare all pins on exposed headers.

-Declare pins that are connected to useful onboard functions like buttons and LEDs.

-Do not declare pins that are reserved for system functions

-Do not declare pins that are not connected to anything.

Declaring GPIO

+ Corresponding DSD

```
// Index 4 – GPIO 4
GpioIO(Shared, PullUp, , , , “\\_SB.GPI0”, , , , ) { 4 }
GpioInt(Edge, ActiveBoth, Shared, PullUp, 0, “\\_SB.GPI0”,) { 4 }
// Index 6 – GPIO 5
GpioIO(Shared, PullUp, , , , “\\_SB.GPI0”, , , , ) { 5 }
GpioInt(Edge, ActiveBoth, Shared, PullUp, 0, “\\_SB.GPI0”,) { 5 }
```

Declaring UART

+ Corresponding DSD

- The following UART declaration is from MinnowBoard Max as an example

```
// Index 2
UARTSerialBus(            // Pin 17, 19 of JP1, for SIO_UART2
    115200,                 // InitialBaudRate: in bits ber second
    ,                       // BitsPerByte: default to 8 bits
    ,                       // StopBits: Defaults to one bit
    0xfc,                   // LinesInUse: 8 1-bit flags to declare line enabled
    ,                       // IsBigEndian: default to LittleEndian
    ,                       // Parity: Defaults to no parity
    ,                       // FlowControl: Defaults to no flow control
    32,                     // ReceiveBufferSize
    32,                     // TransmitBufferSize
    "\\_SB.URT2",           // ResourceSource: UART bus controller name
    ,
    ,
    ,
)
```

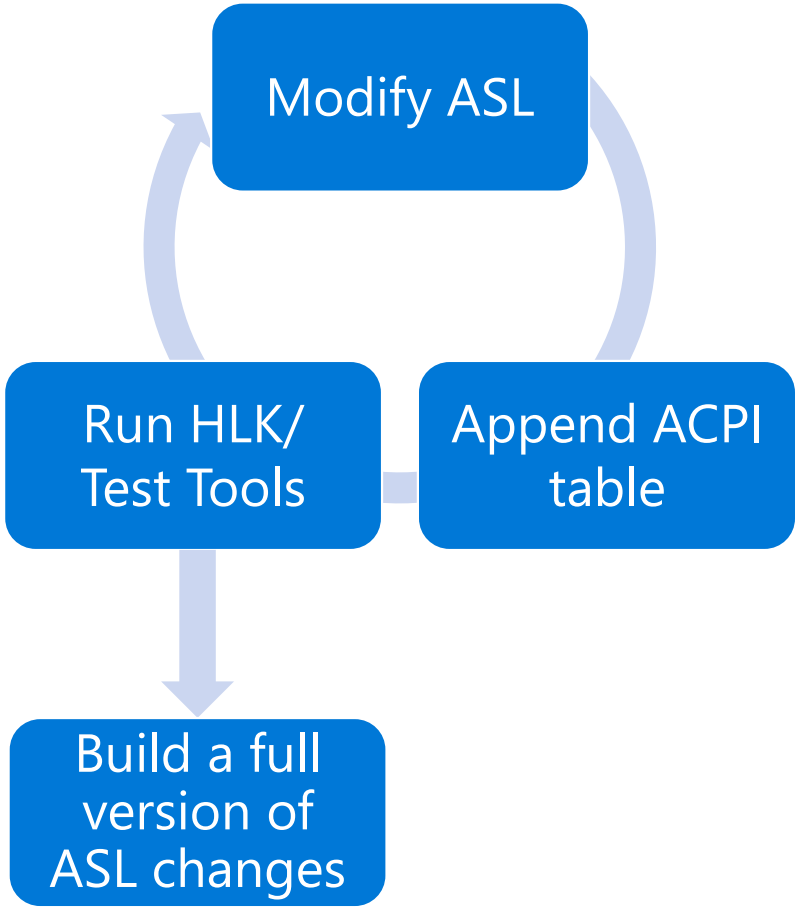
Verification of ASL

Verification

May need to modify the ASL file a few times at development stage

We recommend using ACPITABL.dat during development and validation as it does not require a full UEFI rebuild to test ASL changes.

Run the Hardware Lab Kit (HLK) tests to verify that all resources are exposed correctly and the underlying buses meet the functional contract of the API



1. Modify your asl file

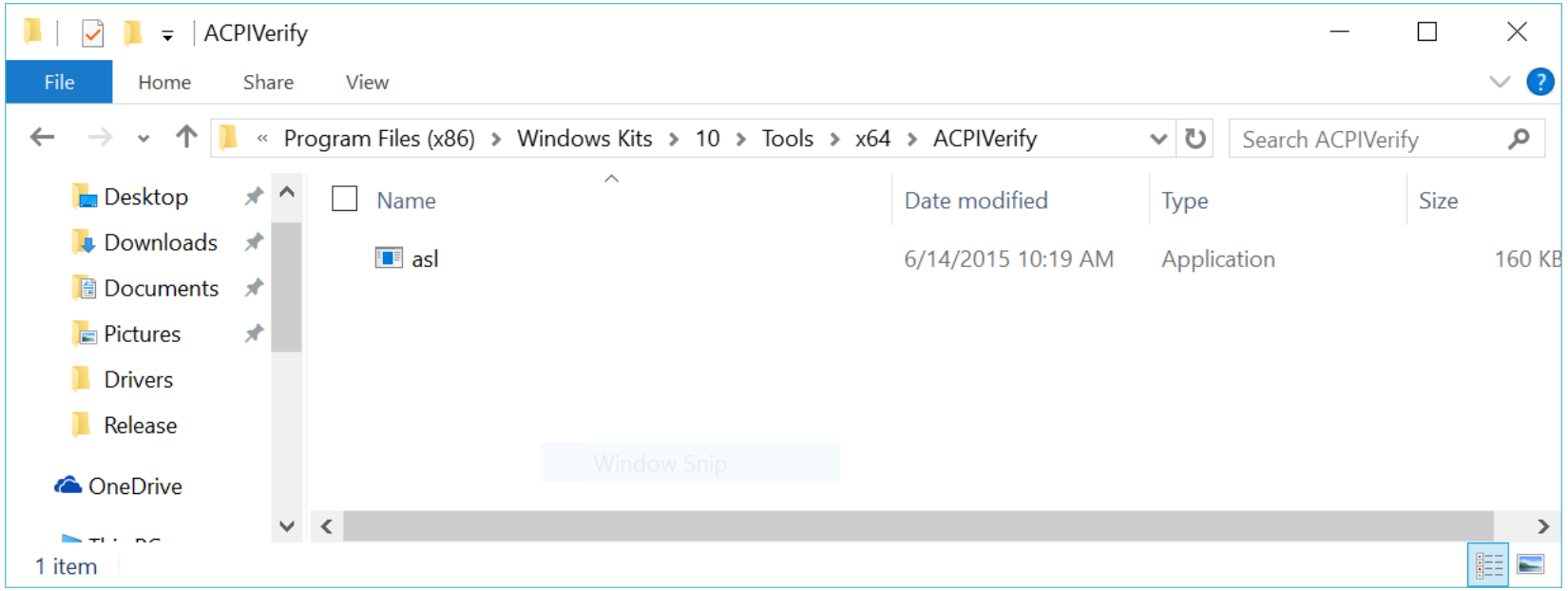
Create a file named yourboard.asl and put the RHPX device node inside a DefinitionBlock

```
DefinitionBlock ("ACPITABL.dat", "SSDT", 1,
"MSFT", "RHPROXY", 1)
{
    Scope (\_SB)
    {
        Device(RHPX)
        {
        ...
        }
    }
}
```

2. Get asl.exe

Download the WDK and get asl.exe

Asl.exe locates at “C:\Program Files (x86)\Windows Kits\10\Tools\x86\ACPIVerify” by default



3. Generate ACPITABL.dat

Copy your yourboard.asl to the folder of asl.exe

Open up a command prompt and navigate to the asl compiler directory

Run following command

asl.exe yourboard.asl

4. Copy the resulting ACPITABL.dat file

An ACPITABL.dat file will be generated in the same directory as the ASL file

Copy the resulting ACPITABL.dat file to c:\windows\system32 on your system under test.

5. Turn on testsigning

Turn on testsigning on your system under test

bcdedit /set testsigning on

Use bcdedit to check status

```
C:\Windows\system32>bcdedit /set testsigning on
The operation completed successfully.
```

```
C:\Windows\system32>bcdedit

Windows Boot Manager
--------------------
identifier              {bootmgr}
device                  partition=\Device\HarddiskVolume2
path                    \EFI\Microsoft\Boot\bootmgfw.efi
description             Windows Boot Manager
locale                  en-US
inherit                 {globalsettings}
default                 {current}
resumeobject            {42b450c2-3055-11e5-b17d-f46e80558b4b}
displayorder            {current}
toolsdisplayorder       {memdiag}
timeout                 30

Windows Boot Loader
-------------------
identifier              {current}
device                  partition=C:
path                    \Windows\system32\winload.efi
description             Windows 10
locale                  en-US
inherit                 {bootloadersettings}
recoverysequence        {42b450c4-3055-11e5-b17d-f46e80558b4b}
recoveryenabled         Yes
testsigning             Yes
isolatedcontext         Yes
```

6. Reboot the system under test

The system will append the ACPI tables defined in ACPITABL.dat to the system firmware tables.

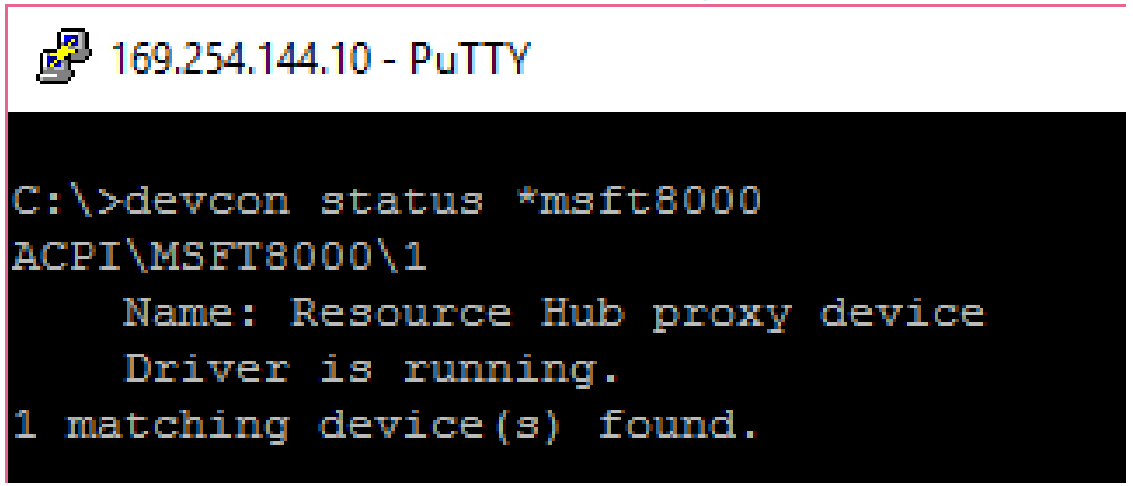
7. Verify device node was added

Sanity check

GpioTestTool	http://ms-iot.github.io/content/en-US/win10/samples/GpioTestTool.htm
I2cTestTool	http://ms-iot.github.io/content/en-US/win10/samples/I2cTestTool.htm
SpiTestTool	http://ms-iot.github.io/content/en-US/win10/samples/SpiTestTool.htm
MinComm (Serial)	https://github.com/ms-iot/samples/tree/develop/MinComm

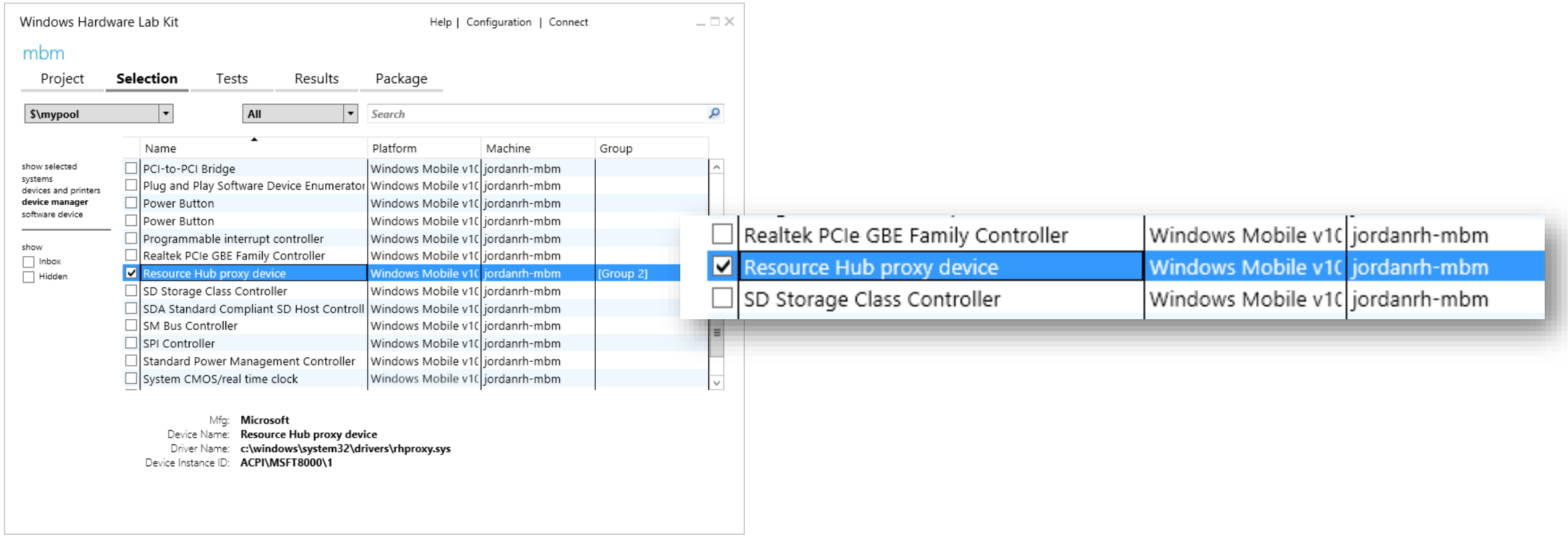
Verify that the RHPX device node was added to the system

*devcon status *msft8000*



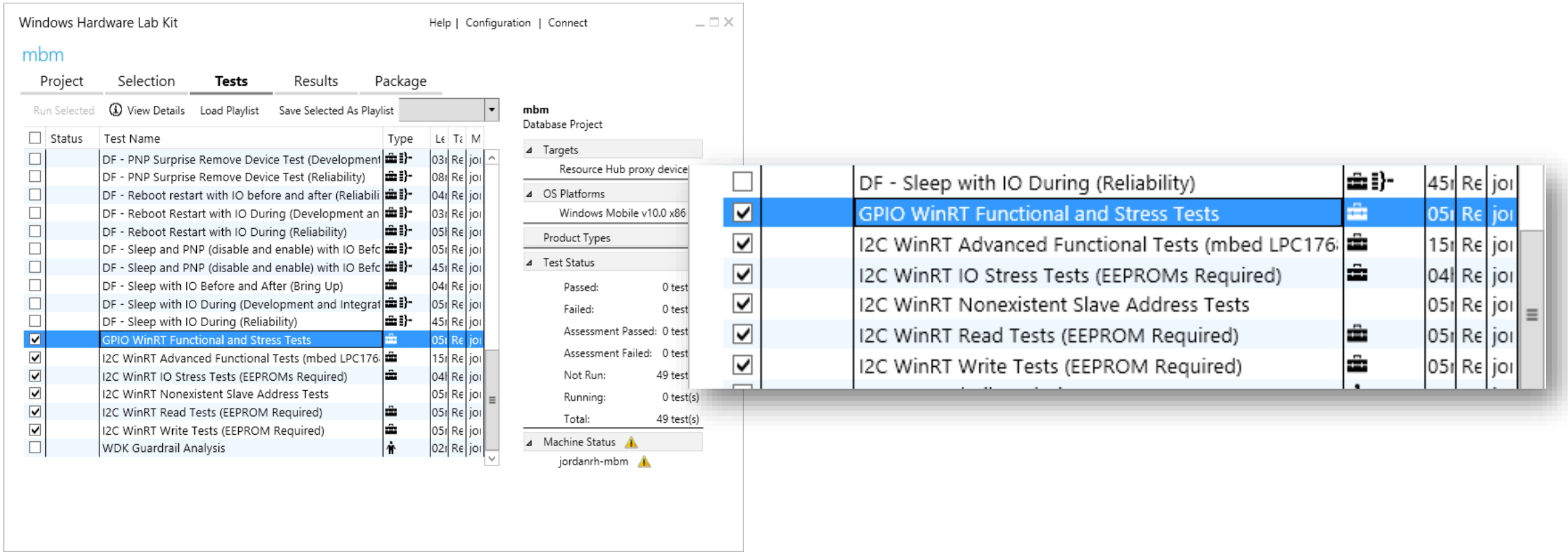
Run the HLK Tests

In the HLK manager, select “Resource Hub Proxy device”



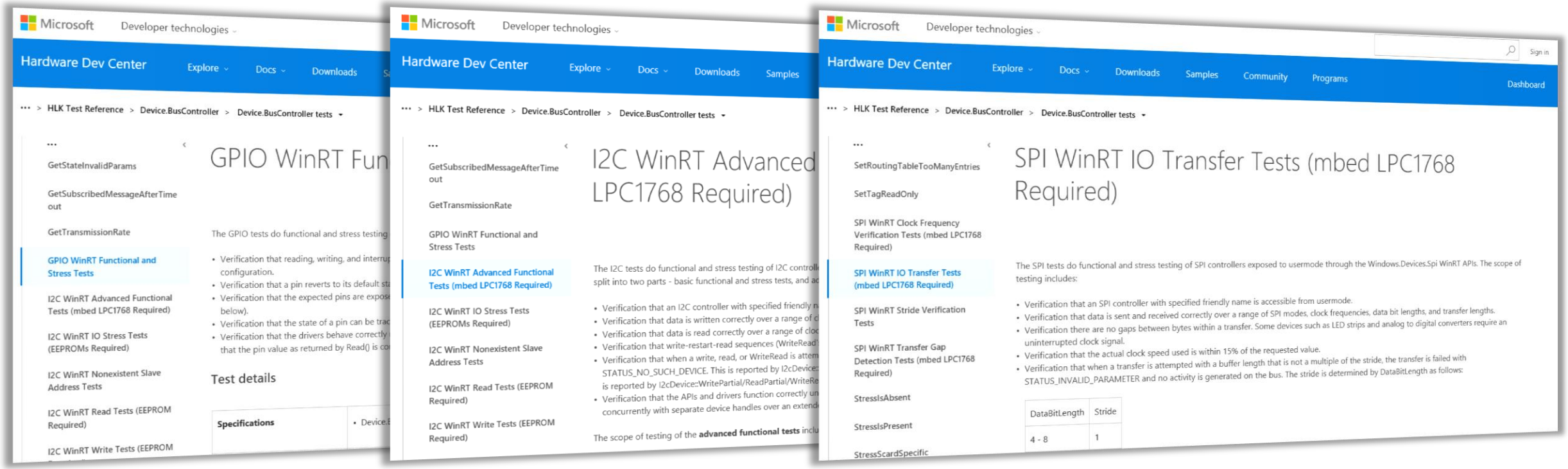
Select Desired Tests in HLK

Then click the Tests tab, and select I2C WinRT, GPIO WinRT, and SPI WinRT tests and run



Full Instructions to run HLK on MSDN

GPIO	https://msdn.microsoft.com/en-us/library/windows/hardware/mt591939(v=vs.85).aspx
I2C	https://msdn.microsoft.com/en-us/library/windows/hardware/mt591936(v=vs.85).aspx
SPI	https://msdn.microsoft.com/en-us/library/windows/hardware/mt591929(v=vs.85).aspx



Case study - DSD must be a child of Device()

Erroneous

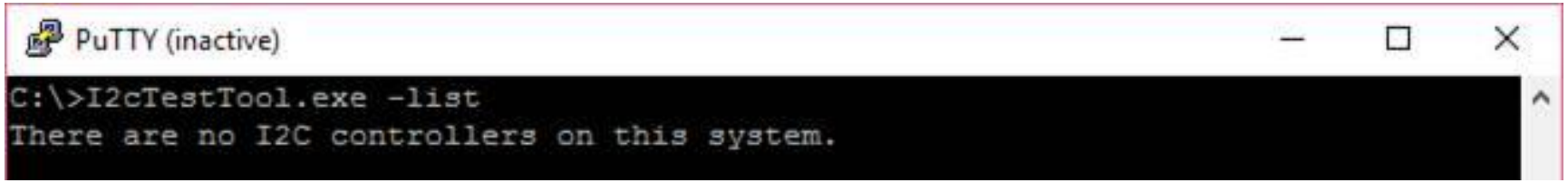
```
Device(MTST)
    {
        Name(_HID, "MSFT8000")
        Name(_CID, "MSFT8000")
        Name(_UID, One)
        Method(_CRS, 0x0, Serialized)
        {
            Name(UBUF, ResourceTemplate()
            {
                SPISerialBus(0, PolarityLow, FourWireMode, 0x8,
ControllerInitiated, 0x7a1200, ClockPolarityLow,
ClockPhaseSecond, "\\_SB.SPI1", 0, ResourceConsumer, , )
                I2CSerialBus(0xff, ControllerInitiated, 0x61a80,
AddressingMode7Bit, "\\_SB.I2C1", 0, ResourceConsumer, , )
            })
            Name(_DSD, Package(0x2)
            //erroneous!! cannot be a child of _CRS. Must be child of
Device()
            {
```

Correct

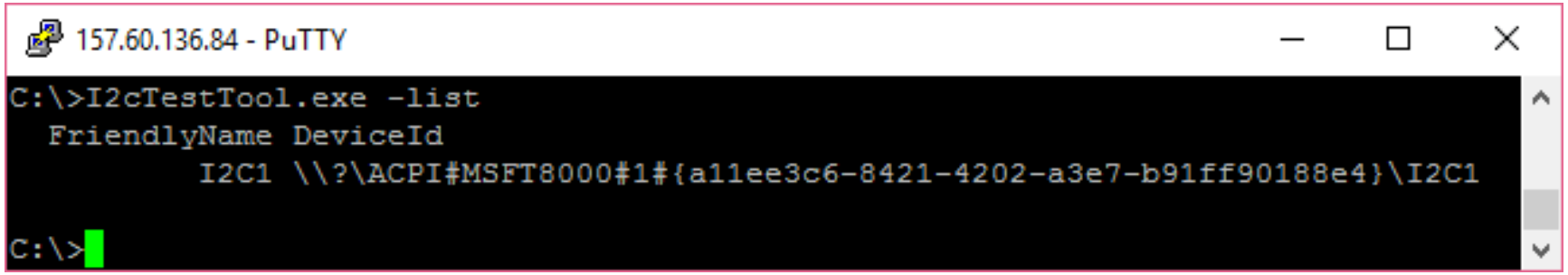
```
Device(MTST)
    {
        Name(_HID, "MSFT8000")
        Name(_CID, "MSFT8000")
        Name(_UID, One)
        Method(_CRS, 0x0, Serialized)
        {
            Name(UBUF, ResourceTemplate()
            {
                SPISerialBus(0, PolarityLow, FourWireMode, 0x8,
ControllerInitiated, 0x7a1200, ClockPolarityLow,
ClockPhaseSecond, "\\_SB.SPI1", 0, ResourceConsumer, , )
                I2CSerialBus(0xff, ControllerInitiated, 0x61a80,
AddressingMode7Bit, "\\_SB.I2C1", 0, ResourceConsumer, , )
                ...
            })
          ...
        }
        Name(_DSD, Package(0x2)     //Correct!!
        {
```

Use I2cTestTool to verify

If I2C is declared incorrectly

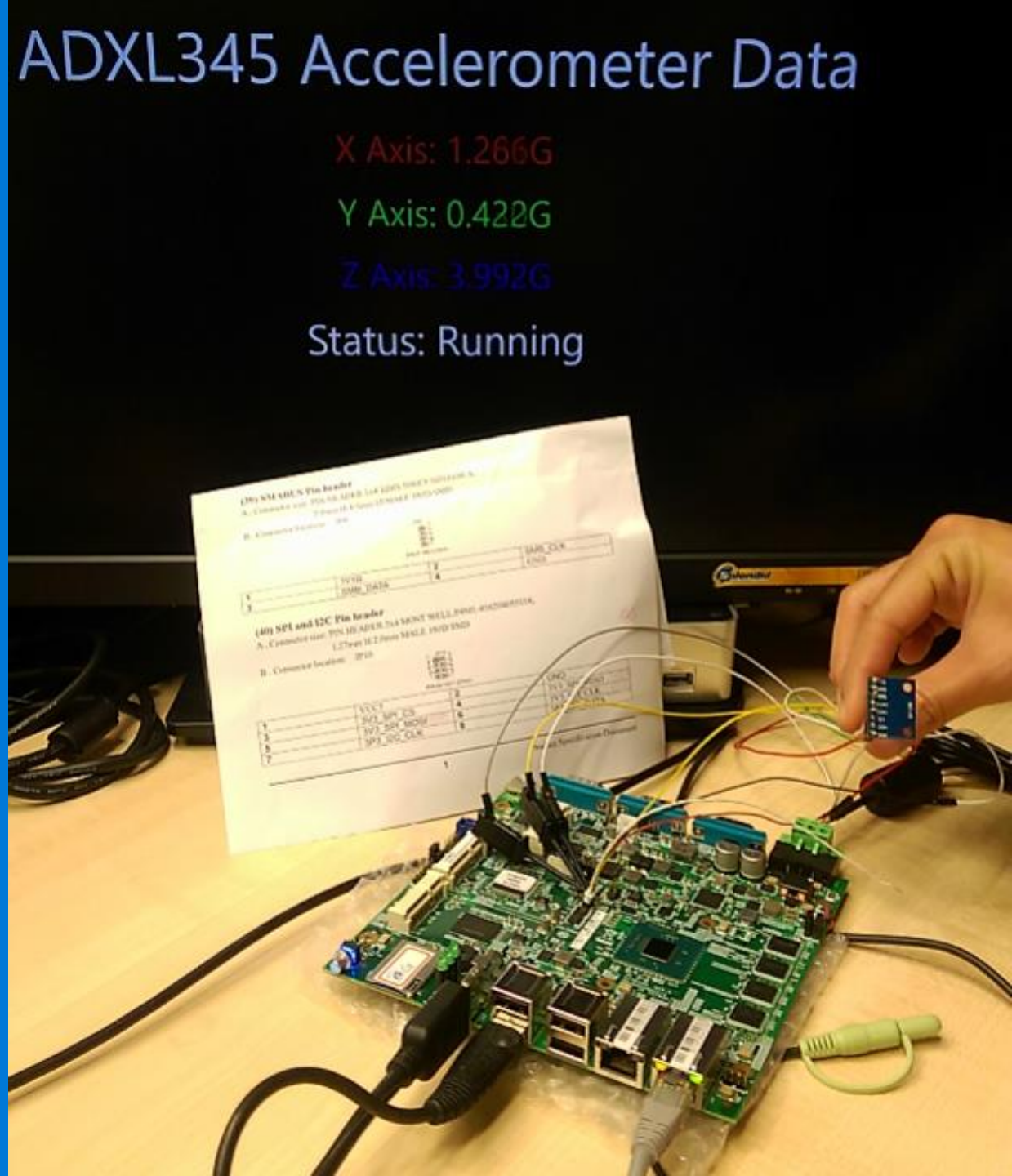


If declaration is correct



Demo – Reading ADXL345 data in User Mode

Demo – Reading ADXL345 data in User Mode



Resources and References

ACPI 5.0 specification	http://acpi.info/spec.htm
Asl.exe (Microsoft ASL Compiler)	https://msdn.microsoft.com/en-us/library/windows/hardware/dn551195(v=vs.85).aspx
Windows.Devices.Gpio	https://msdn.microsoft.com/en-us/library/windows/apps/windows.devices.gpio.aspx
Windows.Devices.I2c	https://msdn.microsoft.com/en-us/library/windows/apps/windows.devices.i2c.aspx
Windows.Devices.Spi	https://msdn.microsoft.com/en-us/library/windows/apps/windows.devices.spi.aspx
Windows.Devices.SerialCommunication	https://msdn.microsoft.com/en-us/library/windows/apps/windows.devices.serialcommunication.aspx
Test Authoring and Execution Framework (TAEF)	https://msdn.microsoft.com/en-us/library/windows/hardware/hh439725%28v=vs.85%29.aspx
SpbCx	https://msdn.microsoft.com/en-us/library/windows/hardware/hh450906(v=vs.85).aspx
GpioClx	https://msdn.microsoft.com/en-us/library/windows/hardware/hh439508(v=vs.85).aspx
SerCx	https://msdn.microsoft.com/en-us/library/windows/hardware/ff546939(v=vs.85).aspx
MITT I2C Tests	https://msdn.microsoft.com/en-us/library/windows/hardware/dn919852(v=vs.85).aspx
GpioTestTool	http://ms-iot.github.io/content/en-US/win10/samples/GpioTestTool.htm
I2cTestTool	http://ms-iot.github.io/content/en-US/win10/samples/I2cTestTool.htm
SpiTestTool	http://ms-iot.github.io/content/en-US/win10/samples/SpiTestTool.htm
MinComm (Serial)	https://github.com/ms-iot/samples/tree/develop/MinComm
Hardware Lab Kit (HLK)	https://msdn.microsoft.com/en-us/library/windows/hardware/dn930814(v=vs.85).aspx

Calls to Action

Join WinHEC LINE Community
@winhec

We want to hear from you!

Please Complete the Evaluation Form and return it to our reception.

Your input is highly important to us!
Thank you!! ☺