Windows® 10 IoT Core Pro Running on Intel® Architecture Platforms

By Sean D. Liming and John R. Malin
Annabooks – www.annabooks.com

October 2016

Windows 10 IoT Core is a cut down version of Windows 10 that only runs Universal Windows Platform (UWP) and traditional command line applications. Since the launch of Windows 10 last year, we have had a number of clients interested in the Windows 10 IoT Core. Although the core version could be used, the clients were trying to use Intel® Core™ i-Series processors. One of the biggest things holding Windows IoT Core back has been the limited processor and board support to a few hobbyist platforms: Raspberry Pi2/3 (Broadcom BCM2837 ARM® Cortex® A7), Dragonboard 410c (Qualcom Quad-core ARM® Cortex® A53), and the MinnowBoard Max (Intel® Atom™ E3800). The only industrial board comes from Toradex, which has the Nvidia® Tegra 3 ARM® Cortex®-A9. The reason for the limiting support is very logical. Learning from the difficult lessons about Windows CE porting, Microsoft is controlling the port of the IoT Core. OEMs who want IoT-Core to run on a specific processor must contact the processor vendor. The processor vendor can work directly with Microsoft to get a port created. The OEM will get the BSP from the processor vendor to build the image. The Microsoft + processor vendor approach saves the OEM time, money, and headaches in trying to port to a processor. The drawback is a chicken and egg. It leaves it to the processor vendor to pursue the support. If the processor vendor doesn't see the value, a port never gets made.

ARM processors come in different flavors and require some porting. Intel processors are a set architecture. Intel has a range of different processors to address specific devices, but in the end they all the same basic platform architecture. This makes the support for the Intel Atom E3800 very interesting since there are many industrial PC platforms using the E3800 family. With all of our clients in mind, we set out to investigate what was possible.

All about the Firmware

The first two major releases of IoT Core where 32-bit images. The Minnowboard Max requires the firmware to be changed from 64-bit to 32-bit for the platform to boot the IoT Core image. If you tried to boot the image on any E3800 off-the-shelf platform, the system would never boot to the image. The strange partitioning that we see on the disk doesn't match a normal Windows partitioning scheme. With very little information from Microsoft, it was unknown if something was done special to the Minnowboard Max firmware or if there was a 32/64-bit issue. With the release of the IoT Core version 14393 with 64-bit support, the door begins to crack open. We finally learned that the image lays out a GPT (GUID partition table) partitioning scheme on the disk, thus UEFI boot is required. Most newer PC platforms have 64-bit UEFI firmware support. A change of the Minnowboard Max firmware to 64-bit proved that the 64-bit image could boot and run. The basic firmware rule:

- To boot 64-bit image, you had to have UEFI 64-bit firmware.
- To boot 32-bit image, you have to have UEFI 32-bit firmware.

The next step was to try the image on an E3800 platform, and the image didn't boot. It appears the firmware for the platform didn't fully support pure UEFI boot.

Since this could happen to anyone, here is the basic test to determine if the board can do a UEFI boot.

1. Download and install the Windows ADK
2. Run the Deployment and Imaging Tools Environment command prompt that the ADK installs.
3. Create a 64-bit WinPE disk by typing in the following:

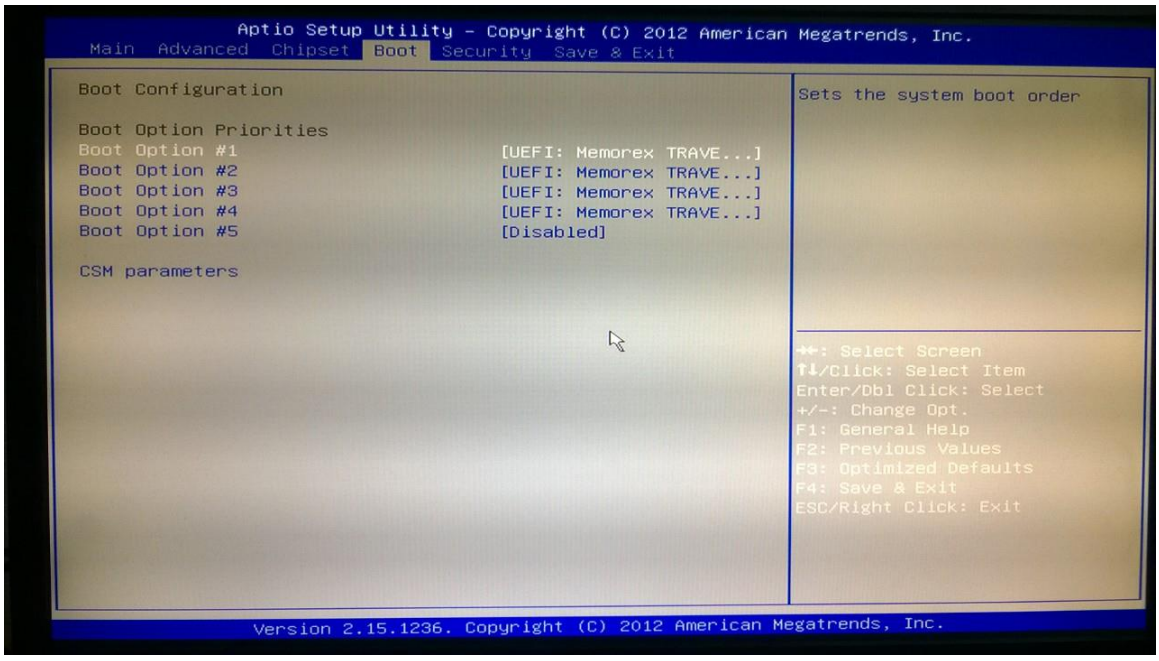
Copyright © 2016 Annabooks, LLC. All rights reserved

copype.cmd amd64 c:\WinPE

4. This will create a 64-bit WinPE image in the c:\WinPE directory. Plug in a USB flash disk
5. Run diskpart.exe to clean off the USB flash disk and set it up to boot the WinPE image using the following commands:.

    a. **List disk** <pay attention to disk size information to identify the USB drive; double check so that you don't format your main system drive>
    b. Select Disk <disk number identified from the list disk command above>
    c. Clean
    d. Create part pri
    e. Select part 1
    f. Format quick fs=ntfs
    g. Active
    h. Assign
    i. Exit

6. Copy the entire contents of the c:\winpe\media folder to the USB flash disk.
7. Safely eject the USB flash disk, remove the flash disk, and plug the flash disk into the target.
8. Boot the target and go to the BIOS.
9. Go to the Boot section and hopefully you see a UEFI boot option for the USB flash disk. Sometimes there is a legacy vs UEFI boot option in the advanced section. In one BIOS, we found a legacy only vs legacy/UEFI combo. You will also want to disable any CSM option. In any case try to set the system to boot the USB flash drive via UEFI.



If you can boot to the WinPE image, the next step is to try an IoT Core 64-bit image from a USB flash disk. The only way to get an IoT Core 64-bit image is to build one. We will save the details for building an IoT Core image for another time. Needless to say our E3800 platform failed to boot the WinPE image with pure UEFI enabled and legacy boot turned off. The E3800 platform only supported legacy boot, which supports 32- and 64-bit boot just fine, but doesn't help us with IoT Core.

Copyright © 2016 Annabooks, LLC. All rights reserved

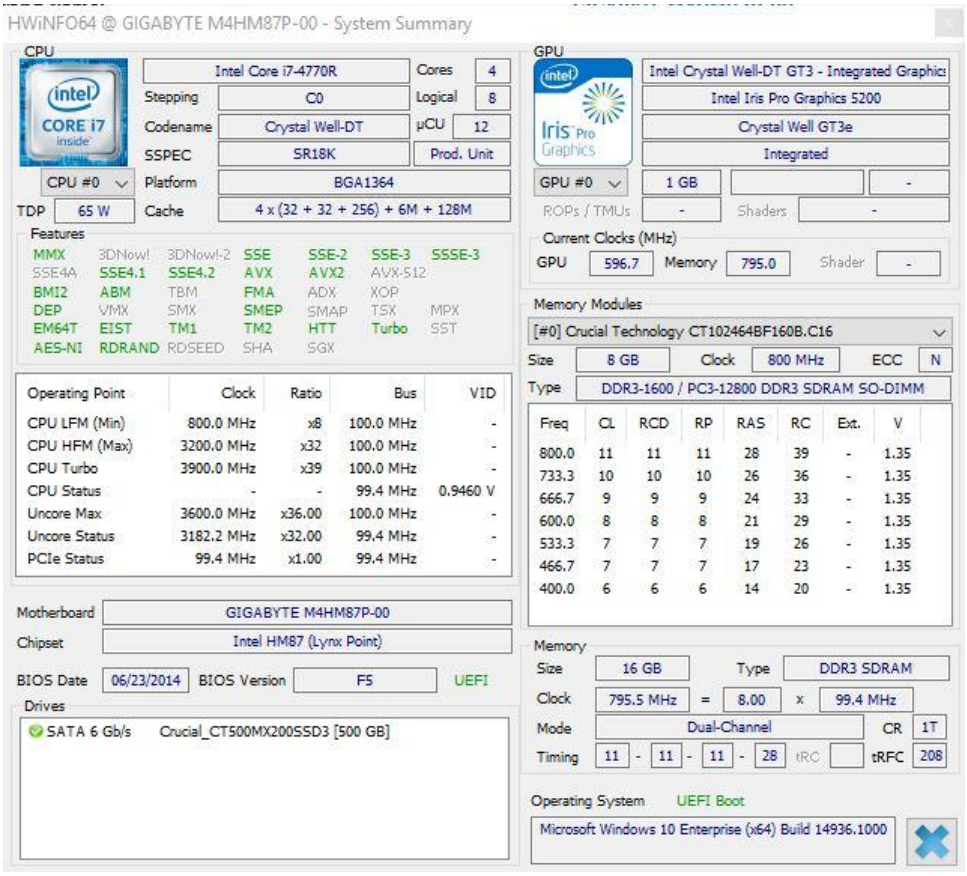
A couple more notes: First some BIOS options call out CSM (Compatibility Support Module). Too boot pure UEFI and turn off legacy mode, CSM must be disabled. Second, we had another system that had a BIOS with a “Legacy Boot only” or a “Legacy/UEFI boot” option. For this case, it is unknown which boot method is actually going to be used at boot time. There is, however, a utility on GIT hub called “detectefi.exe”. This utility can run from WinPE, and it simply tells you if the OS booted from Legacy or UEFI.

The Core i-Series Test

We had a platform with an i7-4770R that had Windows 10 with the GPT partition scheme. Since the Windows Kernel is still the Windows Kernel and Intel Architecture is still Intel Architecture, out of pure curiosity we attempted to boot the IoT Core 64-bit image from USB flash disk, and the IoT Core image booted up successfully. Microsoft has some information on MSDN for IOT Core x86/x64 processor support:

- 400 MHz or faster.
- Compatible with the x86 or x64 instruction set.
- Supports PAE, NX and SSE2.
- Supports CMPXCHG16b, LAHF/SAHF, and PrefetchW for 64-bit OS installation
- Intel Joule, Bay Trail M/D/I

Using a utility like the HWINFO, we can see that the i7 has some of the qualifications.



We had another platform with an old i3-2357M processor, which could also boot the IoT Core image. The results do not imply official support for Intel Core i-Series. The fact that the image can run on something other than Intel Atom E3800 series processors is very promising, and opens up the potential for many industrial PC boards to support IoT Core.

Copyright © 2016 Annabooks, LLC. All rights reserved

The Work is in the Driver Packages

Taking this i-Series to the next step, we decided to add the devices drivers that are unique to the i7-4770R platform. Part of porting Windows CE also involved creating device drivers. Device driver development involved getting down into the bits of the hardware, JTAG emulators, and other debug tools. The process is slow and many drivers had to be created from scratch. Those working with Windows XP Embedded and later versions took advantage of the already developed off-the-shelf PC device drivers, and once in a while a device driver had to be created for unique peripheral hardware. Since IoT Core is Windows, the same PC drivers that work in Windows 10 Home, Pro, and Enterprise should work with IoT Core. The operative word, as always, is “should”.

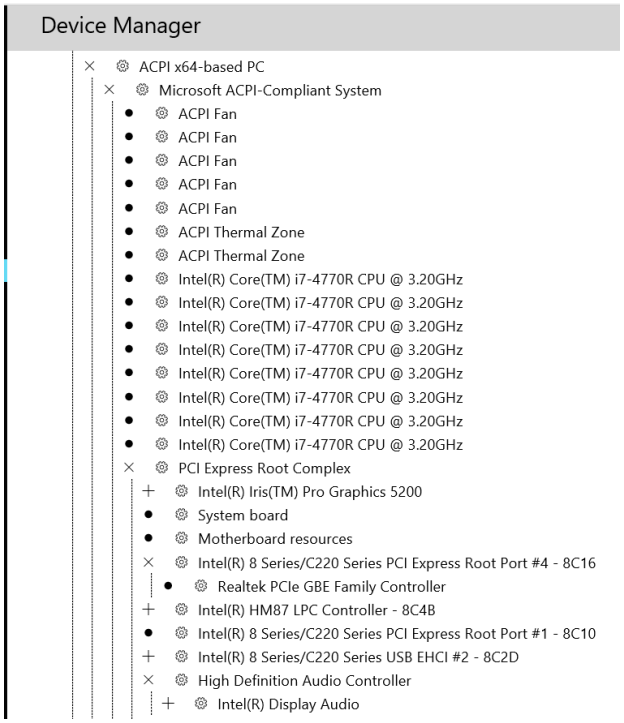
To get the drivers into the image, the drivers must be packaged into CAB file. The IoT Core 14393 build tools have a nice script that converts the INF file to a package XML. The package XML file, along with all the driver files, are then put into a CAB file. The conversion from INF to XML worked for some drivers, but failed for others. Here are a couple examples:

- Wireless – The driver had a .sys file. The INF parser didn’t catch the correct install directory to be c:\windows\system32\drivers. Even though the predefined system environment variable showed %12% in the INF, the parser put the .sys file into c:\windows\system32.

- Intel Graphics Driver – The graphics driver has 224 files that are scattered in the c:\windows and c:\Program Files directories. The script tool immediately complained that the INF file had folders that were not resolved, but continued with the conversion to XML. Reviewing the package XML, we could see that many paths were incorrect. We would have to hand modify the XML file to make this work. The problem is that most of the files are supporting the control panel interface. Since IoT Core has all the traditional GDI support removed, meaning there is no control panel, these files are a waste of space. Needless to say we went through a process that involved using the XP Embedded Component Designer INF import feature, our old XPe tools, and the device manager driver file list to edit the INF and XML file so that the CAB file creation utility would accept it. In the end the graphics driver launched successfully in the IoT Core image. Who knows how long it would take to strip out the files not needed but still allow the driver to run. Intel has released two Embedded Graphics Driver kits (IEGD and EMGD) for the Intel Atom processors. These kits produced a graphics driver with a small set of files; just enough for embedded platforms. There is the simple graphics driver for IoT Core that supports the E3800, but it might be a long while until there is a simple driver for i-Series processors.

Once we had all the device driver packages into CAB files, we were able to build an image with support for all the drivers on the i7-4770R platform.

Copyright © 2016 Annabooks, LLC. All rights reserved



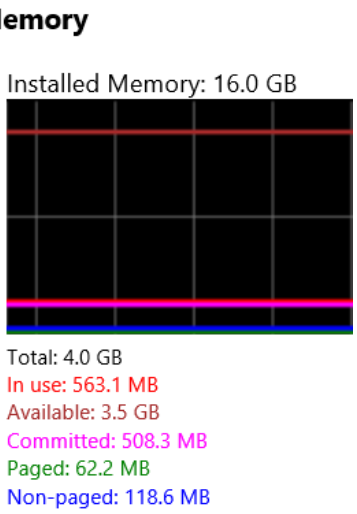
Some Things to Note

There are a couple of things were learned along the way:

1. Device drivers collide at build time. We built the IoT Core image using the MinnowBoard Max BSP. The first build of the image failed with a collision between the graphics driver we were adding and the graphics driver that came with the MinnowBoard MaxBSP. IoT Core only allows for one video driver to be present in the image, which makes sense since the goal is to keep the image small. This means you cannot create a one size fits all image. We also ran into a filter driver on the i3-2357M platform that conflicted with an internal driver. Since the filter driver was a Microsoft driver we opted for the internal driver.

2. The i7-4770R platform has 16GB of RAM, but IoT Core only uses 4GB. The Intel Atom's max RAM size is 4GB, and it looks like the kernel was built-in setting telling the kernel to only use 4GB. The limit on memory is a drawback.

Copyright © 2016 Annabooks, LLC. All rights reserved



3. After the initial boot, a reboot was needed to get the device portal up and running. This might be a quirk of the 14393 release.

Summary – The Door Begins to Crack Open

When Windows CE was first released in 1996, it only had support for SH3 and MIPS processors. When x86 support was added with Window CE 2.0 and was demonstrated to run on embedded PC platforms, the flood gate opened, and Windows CE became a popular embedded operating system. History is kind of repeating itself. Since the latest PCs support 64-bit UEFI firmware, the IoT Core 14393 releases with 64-bit support will allow IoT Core to be available for more platforms. The official limitation is support for Intel Atom E3800 platforms. After our investigation, it looks like i3, i5, and i7 platforms could also be supported, but a 4GB memory limitation might be a problem. There might be other problems that have yet to be uncovered, but the results so far look very promising.

Windows is registered trademarks of Microsoft Corporation
All other copyrighted, registered, and trademarked material remains the property of the respective owners.

Copyright © 2016 Annabooks, LLC. All rights reserved