DeviceTree vtechstudy : Lesson_4

In Lesson_4/ folder, run these commands:

- **make**
- **../../../../gendtb.py -f tlmm.dts**
- **../../../../gendtb.py -f overlay-tlmm-1.dts**

Oops, we have a build error:

stderr:

Error: ./overlay-tlmm-1.dts.pp:4.7-11.3 Label or path tlmm not found

FATAL ERROR: Syntax error parsing input tree

DTC could not generate [./overlay-tlmm-1.dtb]

Q.1) Open overlay-tlmm-1.dts and pinctrl.dtsi in separate editor windows. Other than the number of nodes in each file, what jumps out as an obvious difference between the two files?

A) &tlmm is missing from pinctrl.dtsi

B) &tlmm is missing from overlay-tlmm-1.dts

C) /dts-v1/; is present in overlay-tlmm-1.dts

D) /dts-v1/; is present in pinctrl.dtsi

Q.2) What does /dts-v1/; tell dtc about the file being compiled?

A) this is a DeviceTree source file

B) this is a DeviceTree overlay source file

C) this is a DeviceTree overlay version 1 source file

D) this is a DeviceTree version 1 source file

Now open tlmm.dts in an editor window and review. Since pinctrl.dtsi is #include from tlmm.dts file, /dts-v1/ in pinctrl.dtsi is not required, as it is already present in tlmm.dts.

In overlay-tlmm-1.dts file, add “/plugin/;” on line 3 after /dts-v1/; and re-run this command:

- **../../../../gendtb.py -f overlay-tlmm-1.dts**

The previous build error should be resolved.

Q.3) /plugin/ in the DTS file is required to

A) allow 3rd-party tools “plug-ins” to access the data

B) allow dt-schema to correctly process the file

C) support overlay mechanism

D) force every node in the file to have a phandle defined

Running the command **./lesson_4 -f tlmm.dtb -t** yields this output:

./lesson_4 running...

[main] read dtb_blob[tlmm.dtb]..size[19728]

[dump_nodes][/tlmm/qup_l0_13]..mux[0](0)..mux[1](1)

[dump_nodes][/tlmm/qup_l1_13]..mux[0](1)..mux[1](1)

starting overlay loop..primary_blob[d00dfeed]..bsize[19728]

Unable to stat file[overlay-tlmm-1.dtbo], exiting...

The file extension “.dtbo” is by convention used to indicate the file is an overlay file and not a complete DTB file by itself. Gendtb.py allows an over-ride on the extension. Add “-e dtbo” to the gendtb.py command-line to generate the compiled overlay file correctly.

Re-running the above lesson_4 command-line should yield this output:

./lesson_4 running...

[main] read dtb_blob[tlmm.dtb]..size[19728]

[dump_nodes][/tlmm/qup_l0_13]..mux[0](0)..mux[1](1)

[dump_nodes][/tlmm/qup_l1_13]..mux[0](1)..mux[1](1)

starting overlay loop..primary_blob[d00dfeed]..bsize[19728]

overlay[0][overlay-tlmm-1.dtbo] about to be merged...

fdt_check_header(0x5556cbf6e1d0) returned (-9)

example() returned [FDT_ERR_BADMAGIC]...

There is now a file, overlay-tlmm-1.dtbo.merged present in the folder. This is the base .dtb file (tlmm.dtb) with the overlay-tlmm-1.dtbo file merged into it. The application writes this to the filesystem as a debugging breadcrumb (which is now useful). This would obviously not be possible on a target, consider what you might use instead if you had to debug this on a target? Open this file with a hexdump utility.

Q.4) What value is present in the magic field of the header?

A) 00 00 00 00

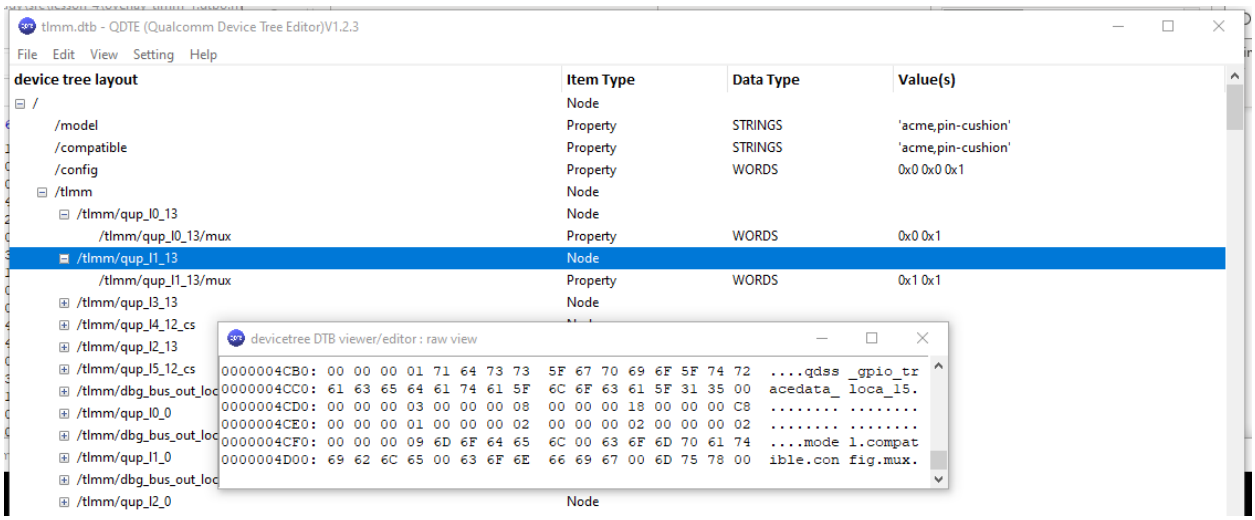
B) FF FF FF FF

C) 00 00 4E 28

D) 00 00 00 38

If you scroll to the bottom of the file in the hex viewer you will notice there is no block of ascii data at the end of the blob, this is another give-away that this is a corrupted file. For an unknown reason the underlying FdtLib API(s) failed silently. This is one of the reasons we perform basic sanity checks on a blob before attempting to use it as a valid DTB.

Add /plugin/; to line 3 of all of the overlay.dts files in this folder. This doesn't completely resolve the overlay issue but we are one step closer. Open tlmm.dtb in QDTE and open the raw window, scrolling to the bottom. If you open a couple of the /tlmm/qup nodes you should see something like this:



This is a somewhat large (20K) DTB and it only has 4 properties. According to ls -l output it is 19728 bytes in length.

If you consider that the runtime overlay mechanism is somewhat similar to what ld.so does on Linux when loading and running a dynamically linked program, can you start to imagine what might be missing and why the overlay mechanism isn't working yet? Review the dtc command-line arguments.

Q.5) What do we have to add to the DTC_ARGS variable in gendtb.py to generate a blob that can be used with the overlay mechanism?

A) -S

B) -p

C) -a

D) -@

E) -H

F) -R

G) -A

H) -s

I) -T

Make the required change in gendtb.py and run the following commands. Note: make your addition to the beginning of the variable string so you don't upset the order of the other parameters.

- **../../../../gendtb.py -f tlmm.dts**
- **../../../../gendtb.py -f overlay-tlmm-1.dts -e dtbo**

Now, when lesson_4 command is run **(./lesson_4 -f tlmm.dtb -t**), you should see this output:

./lesson_4 running...

[main] read dtb_blob[tlmm.dtb]..size[48878]

[dump_nodes][/tlmm/qup_l0_13]..mux[0](0)..mux[1](1)

[dump_nodes][/tlmm/qup_l1_13]..mux[0](1)..mux[1](1)

starting overlay loop..primary_blob[d00dfeed]..bsize[48878]

overlay[0][overlay-tlmm-1.dtbo] about to be merged...

[dump_nodes][/tlmm/qup_l0_13]..mux[0](1)..mux[1](2)

[dump_nodes][/tlmm/qup_l1_13]..mux[0](2)..mux[1](2)

example() returned [FDT_ERR_QC_NOERROR]...

The first pair of dump_nodes output shows the mux values for the first two qup nodes as they appear in tlmm.dtb. The second pair of dump_nodes output shows the same mux node values have been changed by the merging of the overlay-tlmm-1.dtbo file over the base .dtb file.

Open merged file with QDTE. Note: on the open dialog you will have to change the filename mask to be *.* to show this file. It should load correctly because it is a valid DTB file.

Q.6) What is the new root node that has been added? Note: you can use the raw view to see the exact string name if it is unclear. _____

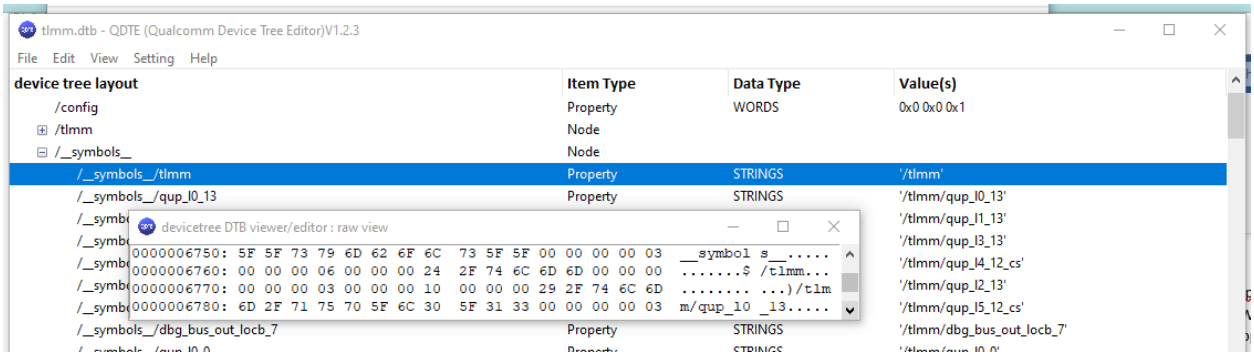
Explore some of the qup nodes under /tlmm and you should note something else is different.

Q.7) What property has been added to each qup node? _____

As we discovered in the previous lesson, phandles, while useful, come at a performance cost, both in terms of memory and speed of lookup. We can begin to quantify the memory cost by looking at the size of the tlmm.dtb file when it is built to support overlay. The new size is 48878 bytes, more than double the original size.

In the raw window goto offset 0xD8, this is the start of the first phandle node and as a phandle is a uint32_t, each phandle property adds exactly 16 bytes to the DTB. There are approximately 422 qup nodes in the DTB, so the phandle property being added for each qup node accounts for 6752 additional bytes, far short of the 29150 byte delta. What else is going on? In the main QDTE window expand the new root node and start to explore these properties.

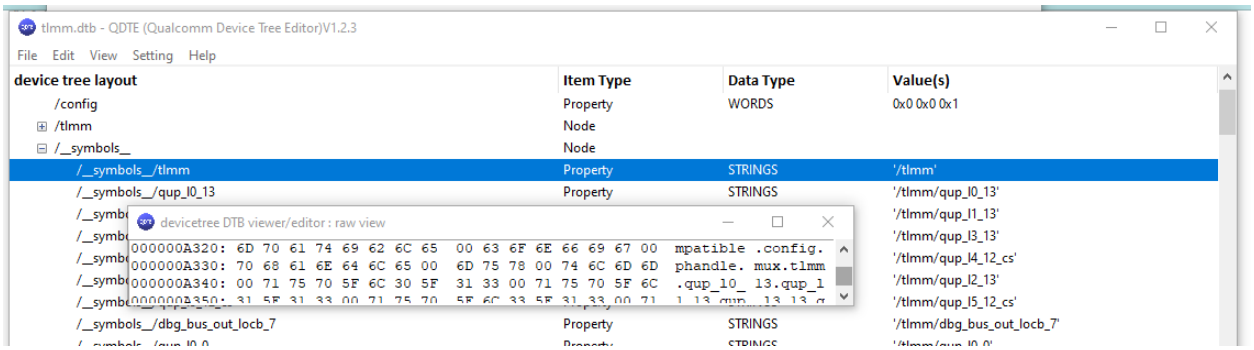
In QDTE, make sure you have tlmm.dtb loaded and have the raw window open. From Edit menu use Find command to search for “symbols”, the raw window should move forward to offset 0x6750, with the first property structure in the __symbols__ node starting at offset 0x675C, it should look something like this screenshot.



Recalling a previous lesson, the property structure has these fields:

Field	Offset	Value
tag	0x675C	3
len	0x6760	6
nameoff	0x6764	0x24
data[0]	0x6768	/tlmm

From the off_dt_strings field of the header we discover the strings block starts at offset 0xA318. Scroll to 0xA33C (0xA318 + 0x24) to see the property name:



Adding the symbol table to the DTB, which is required to support overlays, comes with a very stiff memory cost. Each node that has a label causes dtc to generate a possible phandle for that node. The term possible is used because the phandle won't actually end up in the generated .dtb file unless another node references the phandle, using the syntax we learned in an earlier lesson. Introduce overlay support and the inclusion of the __symbols__ node and now every possible phandle becomes a real phandle because an overlay file can potentially access any of these nodes. While I haven't done the math to account for every byte in the delta, you should be convinced that the __symbols__ node accounts for the bulk of additional memory in the DTB.

Since this is so important to understand, we will make this point specifically: every phandle in the symbol table results in the name appearing twice, once as a property name and then as the value for that property and the value is the full path from the root node, not just the name of the property.

Edit lesson_4.c to add the second overlay file. Use gendtb.py to correctly generate the second overlay .dtbo file. Re-run the lesson_4 command using the same command-line as before. The output should look something like this:

./lesson_4 running...

[main] read dtb_blob[tlmm.dtb]..size[48878]

[dump_nodes][/tlmm/qup_l0_13]..mux[0](0)..mux[1](1)

[dump_nodes][/tlmm/qup_l1_13]..mux[0](1)..mux[1](1)

starting overlay loop..primary_blob[d00dfeed]..bsize[48878]

overlay[0][overlay-tlmm-1.dtbo] about to be merged...

[dump_nodes][/tlmm/qup_l0_13]..mux[0](1)..mux[1](2)

[dump_nodes][/tlmm/qup_l1_13]..mux[0](2)..mux[1](2)

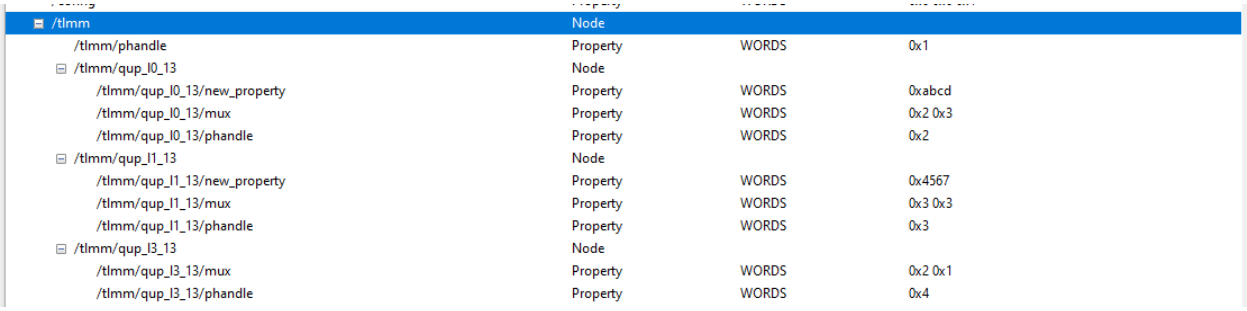
overlay[1][overlay-tlmm-2.dtbo] about to be merged...

[dump_nodes][/tlmm/qup_l0_13]..mux[0](2)..mux[1](3)

[dump_nodes][/tlmm/qup_l1_13]..mux[0](3)..mux[1](3)

example() returned [FDT_ERR_QC_NOERROR]...

Using QDTE, open the second merged file and expand the first three qup nodes under /tlmm, the edit window should look something like this:



/tlmm	Node		
/tlmm/phandle	Property	WORDS	0x1
/tlmm/qup_l0_13	Node		
/tlmm/qup_l0_13/new_property	Property	WORDS	0xabcd
/tlmm/qup_l0_13/mux	Property	WORDS	0x2 0x3
/tlmm/qup_l0_13/phandle	Property	WORDS	0x2
/tlmm/qup_l1_13	Node		
/tlmm/qup_l1_13/new_property	Property	WORDS	0x4567
/tlmm/qup_l1_13/mux	Property	WORDS	0x3 0x3
/tlmm/qup_l1_13/phandle	Property	WORDS	0x3
/tlmm/qup_l3_13	Node		
/tlmm/qup_l3_13/mux	Property	WORDS	0x2 0x1
/tlmm/qup_l3_13/phandle	Property	WORDS	0x4

The second overlay file, besides modifying the mux values also adds a new property to the qup nodes. This is legal for an overlay file to do. Consider the consequences though. Open the second overlay .dts file in an editor and note that the new properties were added after the existing property. However, when the overlay was merged the properties ended up in front of the existing property. This implies that you have no control over the property ordering when you add properties in an overlay.

A second implication is that every node in the DTB after the node(s) with properties added will now be at a new offset. This is also the case when a property value gets modified and the new value is bigger or smaller, e.g. a string, or a list property that has additional values added. It is very dangerous to use a DTB in memory for awhile and then apply an overlay to it. Any offsets or values that may be cached from before the overlay was applied are probably no longer valid.