DSX has some support for flattened device trees.

DSX can convert:

- a given flattened device tree to a soclib caba platform,
- a given soclib caba platform to a flattened device tree.

As all the needed data is **not** present in the components metadata, these conversions involve custom translator code. Therefore, support for translation is limited to known modules, i.e. modules having an associated *handler*.

From a platform to a device tree

For a given platform, we may extract the corresponding device tree as seen for a given mapping table (a device tree can represent only one address-space):

Import converter and built-in handlers:

```
from dsx.contrib.dts_platform.platform_to_dts import PlatformToDts
import dsx.contrib.dts_platform.platform_handlers
```

Convert the platform, for a given mapping table:

```
dtsgen = PlatformToDts(arch, mapping_table)
```

Then we can retrieve the device tree

```
device_tree = dtsgen.create_device_tree()
```

This device tree itself can be serialized to different formats. There are drivers for this task

Device tree serialization as ascii source (`dts`)

Import the driver

```
from dsx.device_tree.dts import Driver
```

Call it for the device tree:

```
driver = Driver(
            outdir = '.',
            parent = None,
            output_file = "my_platform.dts",
            )

device_tree.generate(driver)
```

Device tree serialization as device tree blob (`dtb`)

Import the driver

```
from dsx.device_tree.dtb import Driver
```

Call it for the device tree:

```
driver = Driver(
            outdir = '.',
            parent = None,
            output_file = "my_platform.dtb",
            )

device_tree.generate(driver)
```

From a device tree to a platform

Likewise, you may convert a device tree source or blob to a platform.

The architecture generated will be flat around a VGMN.

Programmatically

import modules

```
from dsx.contrib.dts_platform.dts_to_platform import DtsToPlatform
import dsx.contrib.dts_platform.handlers
from soclib.platform import PfDriver
```

Parse a dts and retrieve corresponding platform

```
pgen = DtsToPlatform("test_3.dts", binary = 'kernel-soclib-mips.out')
pf = pgen.create_caba_platform()
```

Generate the simulator, as usual:

```
pf.generate(PfDriver("hard"))
```

With a command-line one-liner

This tool can even create a netlist with GDB or memchecker-enabled platforms

```
 $ python -m dsx.contrib.dts_platform.converter -h
Usage: converter.py [-o output_dir] [-b binary] file.dts

Options:
  -h, --help     show this help message and exit
  -o OUTPUT_DIR  Output directory
  -b BINARY      Binary kernel to load
  --gdb          Whether to use GDB server
  --memchecker   Whether to use Memchecker
```

For instance:

```
 $ python -m dsx.contrib.dts_platform.converter -o platform -b kernel.out --gdb my_platform.dts
```