Here we will explain how to create an hardware version of a task, writing its SocLib equivalent model.

For instance in the explaination of the process, we'll write a IDCT (inverse discrete cosine transform) coprocessor.

- It handles packets of 64 values, representing a block of 8*8 coefficients
- It has one input and one output fifo
- It handles 16-bit values

Coprocessor

First we'll write the coprocessor in Caba SystemC?.

Declaration for SocLib code generation

We have to declare this component to DSX:

```
class HwIdct(MwmrCoproc):
    def __init__(self, name, **d):
        MwmrCoproc.__init__(self, name, ['input'], ['output'], **d)
```

And register a Caba driver for this component:

```
# Heritage from MwmrCoprocCabaDriver is important
class idctgen(MwmrCoprocCabaDriver):
    # Here goal is to map between TaskModel fifo names and actual component's fifos
    namemap = {'output':'OUT', 'input':'IN'}
    # For all below, see [1]
    headers = 'hw_components/hw_idct.h',
    def __init__(self, node):
        MwmrCoprocCabaDriver.__init__(self, node)
        self.plugmap['output'] = "OUT"
        self.plugmap['input'] = "IN"
    def genType(self, driver):
        return 'HW_IDCT'
    def genDecl(self, driver):
        return '("%s")'%(self.node.name)
Caba.register(HwIdct, idctgen)
```

Notes:

1. CabaDriver?

Task declaration

```
idct = TaskModel(
    'idct',
    infifos = [ 'input' ],
    outfifos = [ 'output' ],
    impl = [ SwTask( 'idct',
                     stack_size = 1024,
                     sources = [ 'src/idct.c' ] ),
             HwTask( HwIdct )
           ] )
```

That's all

Now we can map this coprocessor as an hardware task