

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

For instance in the explanation 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
- Input is 32-bit coefficients, output is 8-bit pixels

Coprocessor

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

see [trunk/dsx/lib/soclib/modules/fifo_idct/caba](#)

Declaration for SocLib code generation

We have to declare this component to soclib build system, see [?wiki:SoclibDesc:](#)

in trunk/dsx/lib/soclib/modules/fifo_idct/caba/metadata/fifo_idct.sd:

```
Module('caba:fifo_idct',
      classname = 'dsx::caba::FifoIdct',
      header_files = [
        "../source/include/fifo_idct.h",
        "../../../include/fifo_idct.h",
      ],
      implementation_files = [
        "../source/src/fifo_idct.cpp",
      ],
      ports = [
        Port('caba:fifo_output', 'p_to_ctrl'),
        Port('caba:fifo_input', 'p_from_ctrl'),
        Port('caba:bit_in', 'p_resetn', auto = 'resetn'),
        Port('caba:clock_in', 'p_clk', auto = 'clock'),
      ],
      uses = [
        Uses('caba:base_module'),
      ],
      instance_parameters = [
        parameter.Int('latency'),
      ],
      tpl_parameters = [
        parameter.Type('word_t'),
      ],
    )
```

Task declaration

Now we define a task, with a software implementation, and a mwmr coprocessor implementation:

in idct.task:

```
TaskModel(
  'hw_idct',
  ports = {
    'input': MwmrInput(64*4),
    'output': MwmrOutput(64),
  },
)
```

```

impls = [
    SwTask( 'idct',
        stack_size = 4096,
        sources = [ 'idct.c' ],
        defines = [ 'WIDTH', 'HEIGHT' ] ),
    MwmrCoprocc(
        module = 'caba:fifo_idct',
        from_coprocc = [ 'output:to_ctrl' ],
        to_coprocc = [ 'input:from_ctrl' ],
        config = [],
        status = [],
        latency = 128,
        word_t = 'uint32_t' )
] )

```

Creating the coprocessor in the netlist

In our netlist, we now have to create the coprocessor and its controller. There is a helper function to do this:

```

import dsx
import soclib

# ....

# The helper function needs the task implementation: get the task model
idct = dsx.TaskModel.getBy_name('hw_idct').get_impl(soclib.HwTask)

# The we can call the helper function, it returns the controller
# and the coprocessor components freshly instantiated

# first argument is your current platform (the one from soclib.Architecture())
# second and third arguments are coprocessor and controller names
ctrl, coproc = idct.instantiate(arch, 'idct0', 'idct0_ctrl') # here names are "idct0" and "i

# Connections between the controller and the coprocessor has been completed (fifos, config,
# Anything else is to be done by the designer.
# For instance, connecting the controller to the interconnect, and assigning a segment to the

ctrl.add_segment('idct0_ctrl', 0x70400000, 0x100, False)
ctrl.vci_initiator // vgm.to_initiator.new()
ctrl.vci_target // vgm.to_target.new()

```

That's all

Now we can map this coprocessor as an hardware task:

```

import dsx

# ....

tcg = dsx.Tcg(
    ....
    dsx.Task( 'idct0', 'hw_idct', ... ),
    ....
)

....

mapper.map( "idct0",
            coprocessor = "idct0",
            controller = "idct0_ctrl")

```