SLM FPGA-IP
CADフロー、シミュレーション手順
慶應環境編

2021/8・9

ふんが

docker環境の構築

- axionなど強力なマシンでsudoerに入れてもらう
- sudo docker run -v /home/hlab/hunga/slm_work:/opt/SLM-FLOW/work -it --rm slm_cad
- これで/home/hlab/hunga/slm_workがdocker環境と共有される。
- 自分の共有環境に/home/hlab/hunga/slm_work/templateをコピーする。
- 最新の環境は常にtemplateに入れておくようにするので、更新の度にコピーする。
- 以下、６ページまでの処理はdocker環境内で行う。

論理合成

- example.hdlのConfiguration Dataを生成する場合を考える
- APP/example/hdl/というディレクトリを作り、この中に対象となる設計example.hdlを置く。
- 1_logicsynthesisの下で
  ./odin.sh example
- 2_mappingの下で
  ./run_abc_if_func.sh example
- 3_packplaceの下で
  ./run1st.sh exampleを実行
  2_mapping/result/example/example.clustered.placeからiomapファイルをエディットする。

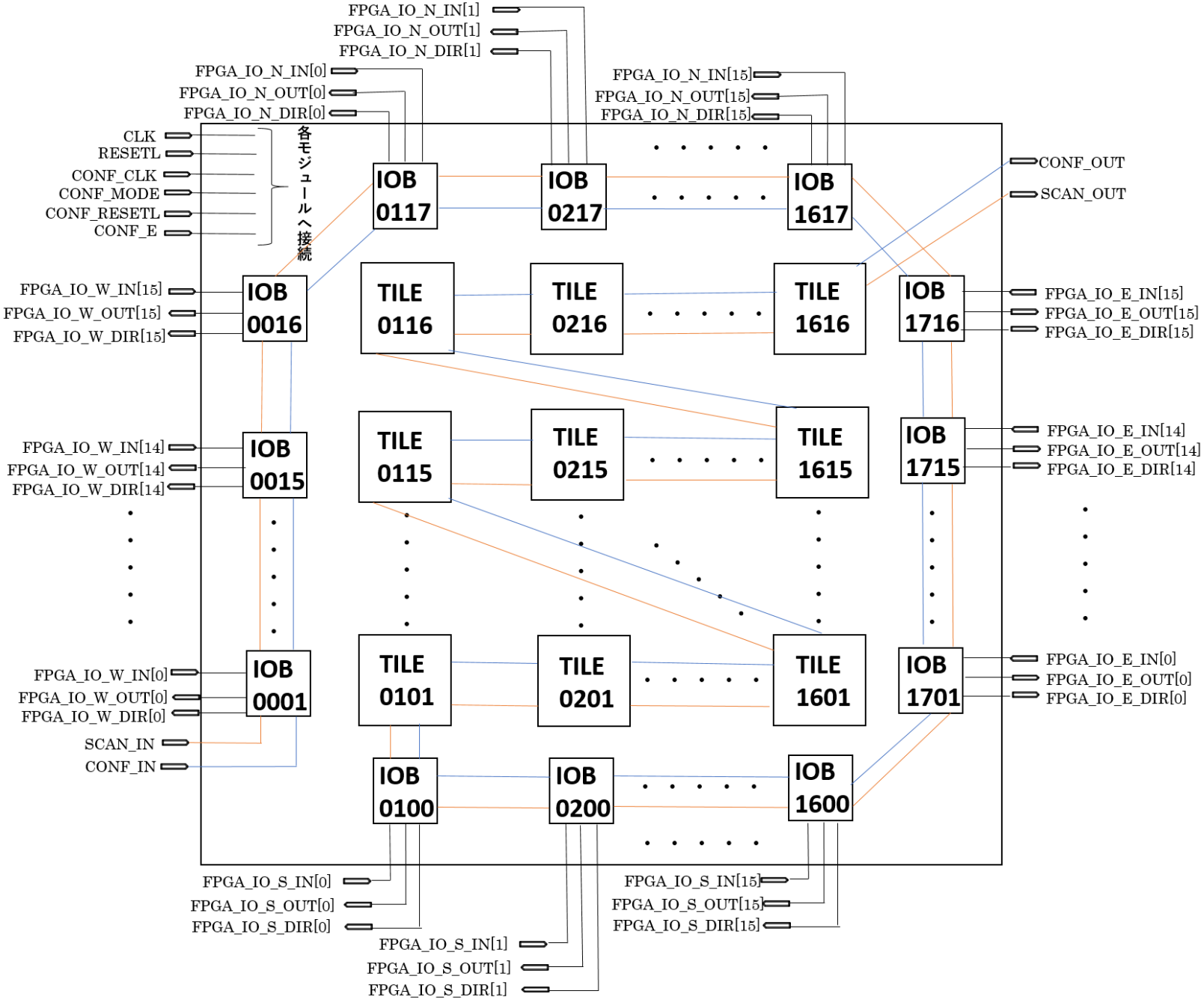
入出力ピンの割り当て

example.clustered.placeの中身

```
#block name     x       y       subblk  block number
#----------    --      --      ------  ------------
n29             8       1       0       #0
top^b~0         5       0       0       #1
top^b~1         6       0       0       #2
top^a~1         2       0       0       #3
top^a~0         1       0       0       #4
top^ck          8       17      0       #5
top^b~2         7       0       0       #6
top^a~2         3       0       0       #7
top^FF_NODE~5   16      1       0       #8
top^b~3         8       0       0       #9
top^a~3         4       0       0       #10
```

入出力ピン以外（n29とかFF..とか）は削除
入出力ピン名の最初は常にtop
a[0]はtop^a~0になる
x,y座標は次のページ参照
subblkは常に 0 でいいようだ

ckは、自動的にFPGAのクロックに接続されるので
ここに書かなくてよい。
しかし、削除するとエラーになるので
このまま残しておく。



x, yを17,1にすると
FPGA_IO_E[0]に
割り当てられる。

RISC-Vのアドレスマップ
（7ページ）と
照合してI/Oの場所を
決める。

作ったexample.iomapは
3_packplaceの下に置く

配置配線、bitstream file生成

- 3_packplaceの下で
  ./runmap.sh example
- 4_routingの下で
  ./run.sh example
- 5_bitstreamの下で
  ./run.sh example

bitstream fileは4_routing/resultの下
  example.betstream.frame
  I/O mapは
  example_iomap.txtに表示されるので確認のこと。

  上記でdocker環境下　熊大CADはおしまい。

RISC-Vによる制御

- 現在はいい加減なメモリマップで以下のように設定している。

アドレスh	信号名	R/W	意味
0020_0000	CSTART	W	Configuration Start (1bitのみ有効)
0020_0004	BREAK	W	Break（1bitのみ有効）
0020_0008	Status	R	6'b0,MCOUNT,ERRL,SCAN_MODE,STATE
0020_000c	SCAN_MODE	W	Scan once
0020_0010	DIR0	W	Direction for E,S 0:in/1:out
0020_0014	DIR1	W	Direction for W,N 0:in/1:out
0020_0020	Data out0	W	Data out for E,S
0020_0024	Data out1	W	Data out for W,N
0020_0030	Data_in0	R	Data in for E,S
0020_0034	Data_in1	R	Data in for W,N

E,S : E[15:0],S[15:0]
W,N: W[15:0],N[15:0]
FPGAにデータを出力したいとき：　DIRに１を書き、Data_outに書く
FPGAからデータを入力したいとき：DIRに０を書き、Data_inから読み込む

シミュレーション

- /home/hunga/wjupiter/vcs_sim: SLMのVerilog 記述
- /home/hunga/wjupiter/vcs_sim/mem: Data/Conf. Memory
- /home/hunga/wjupiter/vcs_sim/base: RISC-V

memにdocker環境で生成したexample.bitstream.frameをコピー

- ./m288t8.py example.bitstream.frame out.dat

により8ビット幅に変換

vcs_simにて、./tbase.shを実行することでncverilogが実行される。

RISC-Vのプログラム

- 計算機構成で使っているものをそのまま利用
  - http://www.am.ics.keio.ac.jp/parthenon/
- base内で、例えばadder.asmのアセンブル
  - make adder

  → imem.dat: 命令メモリの初期化ファイルが生成される。
  先のページの./tbase.shでシミュレーション

プログラム例

```
 lui x3,0x00200
sw x0,x3,0                ←　これでConfiguration Start
addi x4,x0,5
loop:  lw x1,x3,8
andi x1,x1,0xff
bne x1,x4,loop     ←　Configurationが終了すると状態が5になるので待つ
addi x1,x0,0xff
sw x1,x3,0x10     ←　Sの下8ビットを出力に設定
addi x1,x0,0x56   ←　入力データとして５＋６を計算させる
sw x1,x3,0x20
lw x1,x3,0x30       ←加算結果を読み出す
sw x1,x3,0x80    ←　現在のテストベンチでは80番地に書くと表示される
lend: beq x0,x0,lend
```