January 27, 2022

Project: Vending Machine

ALLIANCE ASIC Flow

Omama Elrefaei| ECE617 | Fall 2021

HDL: VHDL

Software tools:
QuestaSim-64 10.4e
Quartus Prime 20.1 Lite Edition
Alliance

Contents

Introduction

It is required to build a vending machine which costs only 1.25LE for both soft drinks, and juice. It accepts 1 LE, 0.5 LE & 0.25 LE, and returns change if any.

This project describes the designing of the machine using FSM. The process of five states has been modeled using the MEALY machine model, and a clock frequency of 1MHz.

It takes three inputs, Piasters which is encoded to be “01” for 0.25LE, “10” for 0.50LE, and “11” for 1.00LE, soft drink selection, and juice selection.

It produces four outputs, drink which is encoded to be “01” for a soft drink, and “10” for juice, change25 which is up when there is a change equal to 0.25LE, change50 which is up when there is a change equal 0.50LE, and change100 which is up when there is a change equal 1.00LE.

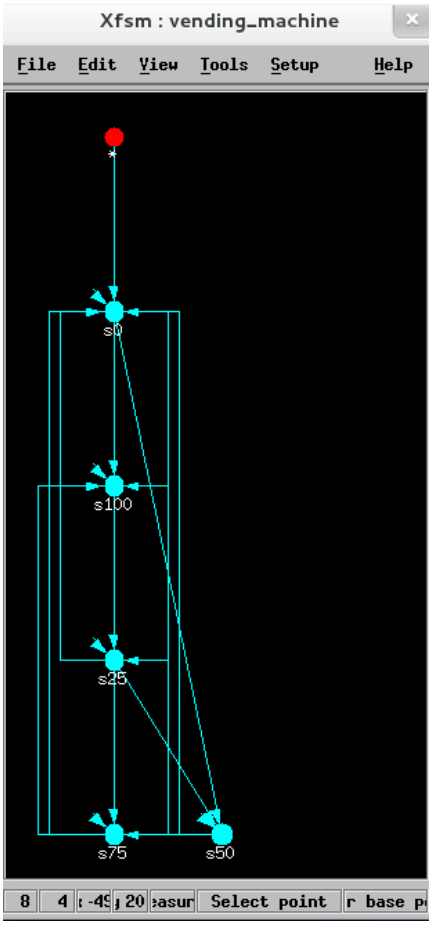
If the user cancel the order, the machine will return the money.

Chapter 1 High-Level Design

In this chapter, I wrote VHDL code and testbench for the vending machine and simulated it using QuestaSim. I used XFSM to show a graphical FSM state diagram, and Quartues to show the RTL viewer.

1.1 FSM State Machine

The finite state machine is implanted as MEALY and it consists of 5 states:



1] First state So is the starting state.

2] Second state S25 is reached through the first state when the input is 25 piasters. Also, it is reached through the fifth state if the order is canceled, and return the money.

3] Third state S50 is reached through the first, and the second state when the input is 50, and 25 piasters. It returns money if the order is canceled.

4] Fourth state S75 is reached through the second, and the third state when the input is 50, and 25 piasters. It returns money if the order is canceled.

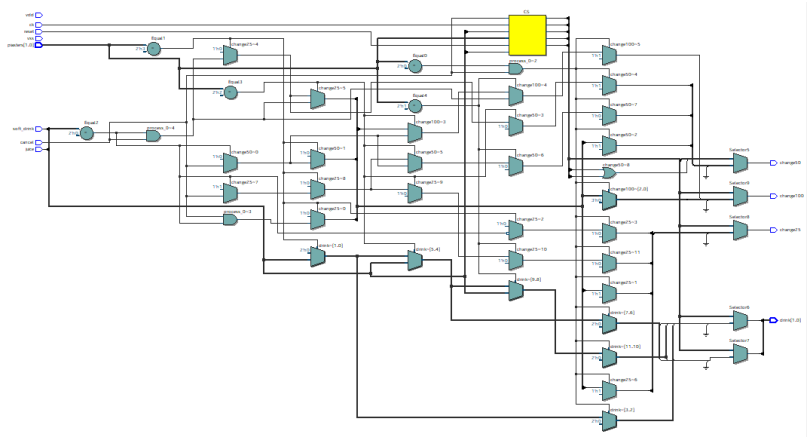
5] Fifth state S100 is reached through the first, second, and third state when the input is 100, and the fourth state regardless of the input. It returns the change, and money if the order is canceled.

1.1.1 Transitions

Current State S	Inputs				Next State S'	Outputs			
	Piasters "01" for 0.25LE "10" for 0.50LE "11" for 1.00LE	Soft Drink	Juice	Cancel		Drink "01" for Soft Drink "10" for Juice	change 25	change 50	change 100
S0	00	x	x	x	S0	00	0	0	0
	01			x	S25	00	0	0	0
	10			x	S50	00	0	0	0
	11			x	S100	00	0	0	0
S25	00	x	x	0	S25	00	0	0	0
				1	S0		1		
	01			x	S50	00	0	0	0
	10			x	S75	00	0	0	0
	11	0	1	x	S0	01	0	0	0
		1	0	x		10			
		0	0	1		00	1		1
S50	00	x	x	0	S50	00	0	0	0
				1	S0			1	
	01			x	S75	00	0	0	0
	10			x	S100	00	0	0	0
	11	0	1	x	S0	01	1	0	0
		1	0	x		10			
		0	0	0	S50	00	0	0	0
		0	0	1	S0		0	1	1
S75	00	x	x	0	S75	00	0	0	0
				1	S0		1	1	
	01			x	S100	00	0	0	0
	10	0	1	x	S0	01	0	0	0
		1	0	x		10			
		0	0	0	S75	00	0	0	0
		0	0	1	S0		1	0	1
	11	0	1	x	S0	01	0	1	0
		1	0	x		10			
		0	0	0	S75	00	0	0	0
		0	0	1	S0		1	1	1
S100	00	x	x	0	S100	00	0	0	0
				1	S0		0	0	1
	01	0	1	x	S0	01	0	0	0
		1	0	x		10			
		0	0	0	S100	00	0	0	0
		0	0	1	S0		0	0	1
	10	0	1	x	S0	01	1	0	0
		1	0	x		10			
		0	0	0	S100	00	0	0	0
		0	0	1	S0		0	1	1
	11	0	1	x	S0	01	1	1	0
		1	0	x		10			
		0	0	0	S100	00	0	0	0
		0	0	1	S25		1	1	1

2.1 RTL Viewer

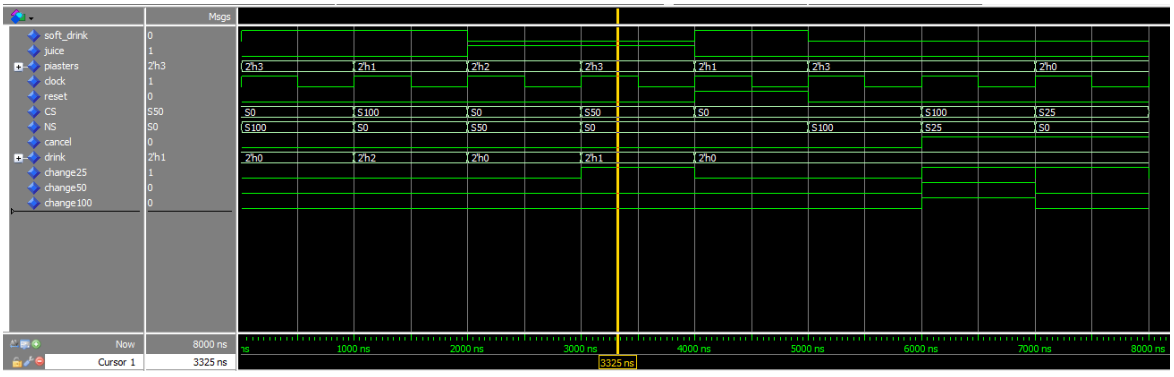
This viewer is created by running the VHDL code on (Quartus Prime 20.1 Lite Edition).



3.1 Test Strategy

Tested feature	Input						Delay after	Expected output			
	Piasters	Soft Drink	Juice	Cancel	reset	ck		drink	change25	change50	change100
Get soft drink by 1LE + 0.25LE	11	1	0	0	0	1	1 us	00	0	0	0
	11	1	0	0	0	0	1 us	00	0	0	0
	01	1	0	0	0	1	1 us	01	0	0	0
	01	1	0	0	0	0	1 us	01	0	0	0
Get juice by 0.5LE + 1LE	10	0	1	0	0	1	1 us	00	0	0	0
	10	0	1	0	0	0	1 us	00	0	0	0
	11	0	1	0	0	1	1 us	10	1	0	0
	11	0	1	0	0	0	1 us	10	1	0	0
Reset	01	1	1	0	1	1	1 us	00	0	0	0
	01	1	1	0	1	0	1 us	00	0	0	0
Select nothing after adding 1LE + 1LE	11	0	0	0	0	1	1 us	00	0	0	0
	11	0	0	0	0	0	1 us	00	0	0	0
	11	0	0	0	0	1	1 us	00	1	1	1
	11	0	0	0	0	0	1 us	00	1	1	1
	00	0	0	1	0	1	1 us	00	1	0	1
	00	0	0	1	0	0	1 us	00	1	0	0

4.1 Simulation Result



Chapter 2 **Low-Level Synthesis**

The ALLIANCE tools used in this chapter are:

i SYF an FSM synthesizer. It transforms an FSM subset (.fsm) (whereas .fsm is created by adjusting extension .vhd to .fsm) to a VHDL behavioral subset (.vbe). It chooses a state encoding algorithm and determines the logical functions of all state registers and output bits.

ii BOOM a Boolean minimizer. It performs technology-independent logic optimization.

iii BOOG a library binding and optimizer on gates. It handles only logic functions generated by BOOM, performs logic mapping to standard cell library (technology dependent), and translates Boolean functions of nodes of the Boolean network into networks of gates using only low-driving capability gates (Each node is treated independently). It uses pre-characterized cell data (paramfile.lax) to optimize area & delay. Its outputs are gate-level netlist (Alliance VHDL structural subset (VST) file). Its main disadvantage is that it is not deterministic (can have different results each time).

iv LOON a local optimizer of nets. It also uses (paramfile.lax). It operates only on critical paths to reduce the propagation time. It uses two techniques, gate replacement with higher current versions, and buffer insertion in case of very high fanout.

v XSCH: a graphical schematic viewer (.vst) files with optional color mapping.

vi FLATBEH: a behavioral from Structural. It produces a flattened (non-hierarchical) behavioral description from an input gate-level netlist.

vii PROOF: a formal verifier. It performs formal verification between the original behavioral model and the synthesized model.

Then I used QuestaSim for simulation to validate the synthesis result using the same testbench from chapter 1 and the Sxlib standard cell library.

2.1 Boolean Network Optimization

Encoding Type	No. of literals before opt.	No. of literal after opt.
ASP	211	123
Jedi	215	127
Mustang	213	147
One hot	204	145
Random	215	117

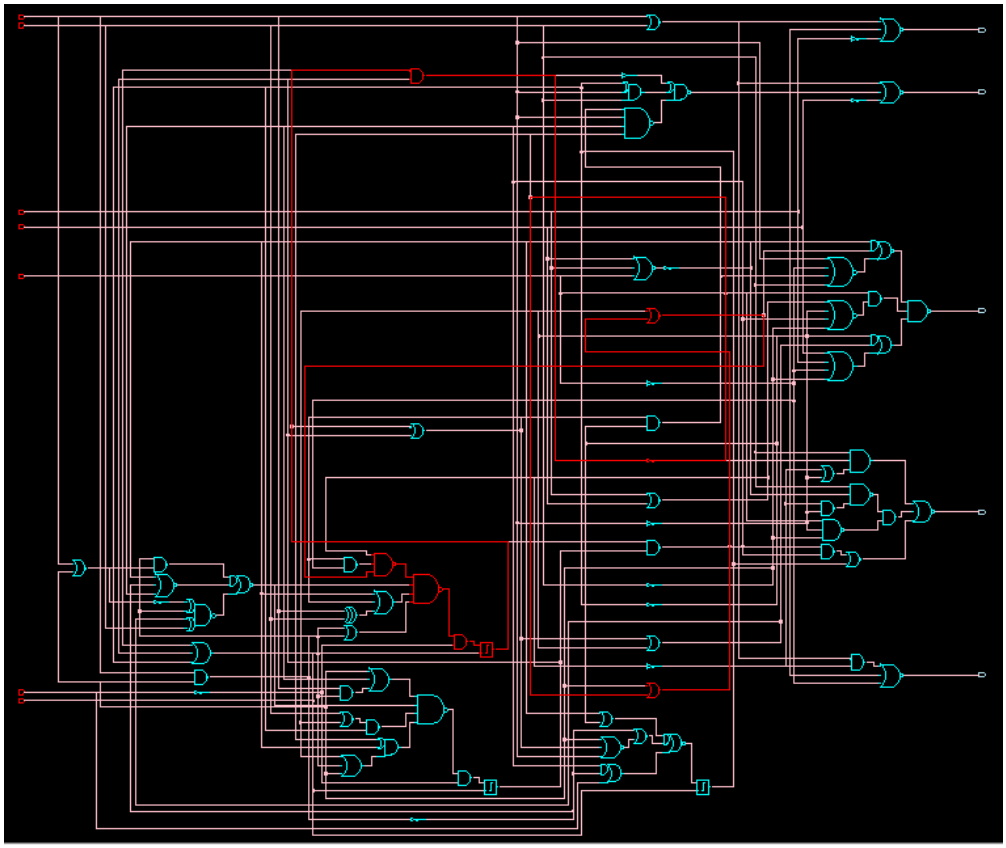
2.2 Netlist Optimization

Encoding Type	Worst RC Delay(ps)	Area (lamda)
ASP	416	101500
Jedi	342	128250
Mustang	416	111500
One hot	416	131000
Random	416	101715

The best encoding type in area is ASP, all the next steps will be done on ASP only.

2.3 Netlist Visualization

The XSCH highlights the critical path in red.

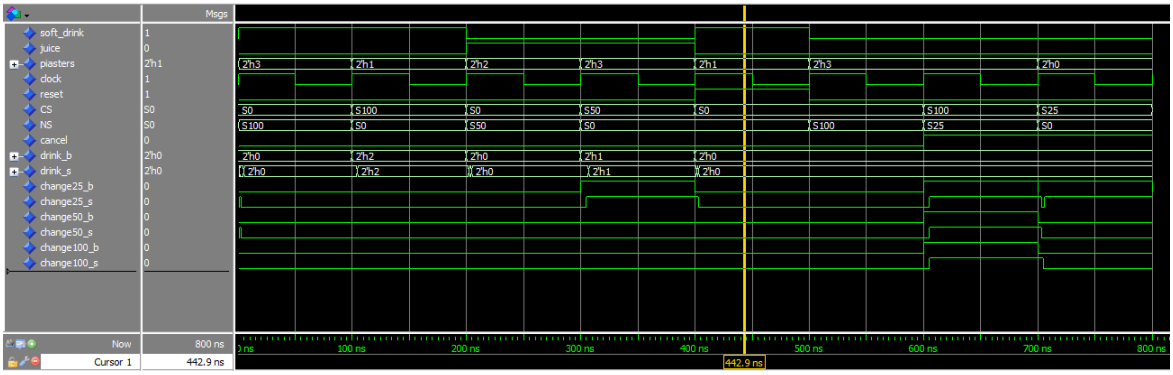


2.4 Netlist Checking

The PROOF result shows that two files are equivalent

```
------------------------------------------------------------------------------
            Formal proof with Ordered Binary Decision Diagrams between

            './vending_machinea'  and  './vending_machine_fb'
------------------------------------------------------------------------------
============================  PRIMARY OUTPUT  ================================
============================  AUXILIARY SIGNAL  ==============================
============================  REGISTER SIGNAL  ===============================
============================  EXTERNAL BUS ===================================
============================ INTERNAL BUS ====================================

                            Formal Proof : OK

pppppppppppppppppppppppprrrrrrrrrrrrroooooooooooooooooooooooooooooffffffffffffff
------------------------------------------------------------------------------
```

2.5 Delay Simulation

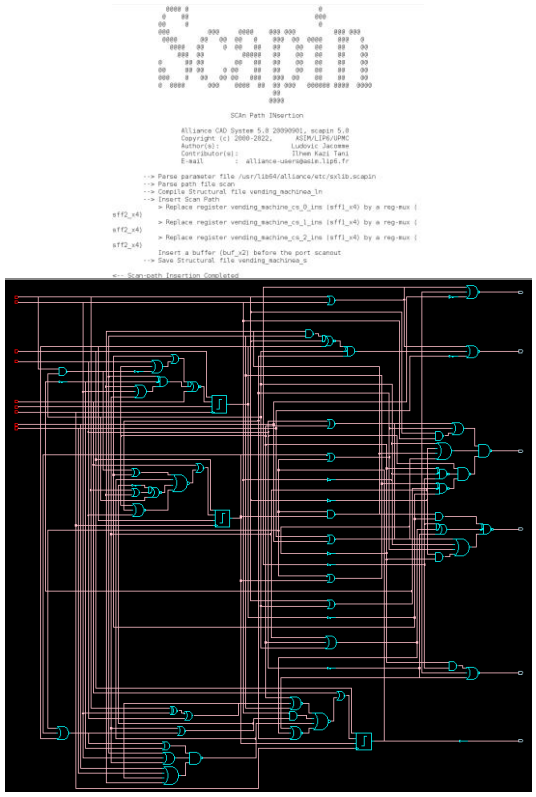


Chapter 3 **Design for test (DFT)**

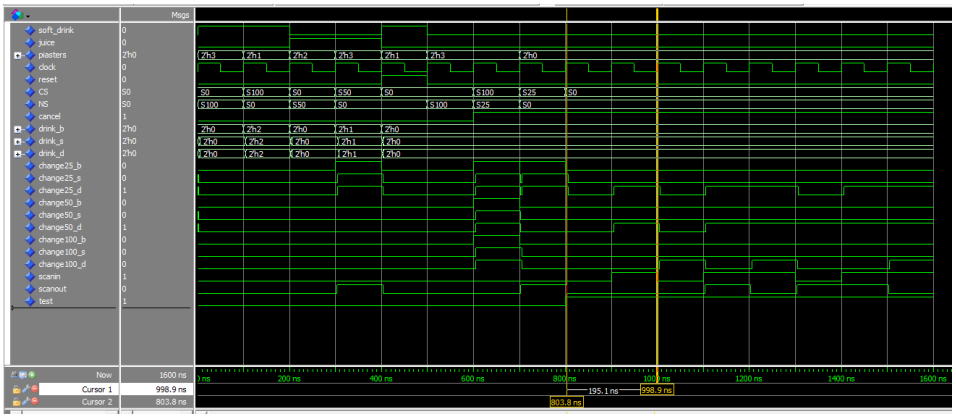
In this chapter, I used the SCAPIN tool from Alliance. It uses (scan.path) to add scan chains to synthesized gate-level netlist (.vst) file. The I used XSCH to show it's schematic.

After this, I used QuestaSim for simulation to validate the DFT result using the same testbench from Chapters 1 and 2, and the Sxlib standard cell library.

3.1 Scan-path Insertion



3.2 Scan-path Simulation



At test = 0, outputs of scan "%_d" is identical to behavioural and structural outputs. scanout has no meaning.

At test = 1, outputs of scan "%_d" have no meaning. scanout follows scanin with a delay of 3 clock cycles as I have three registers in the scan chain.

Chapter 4 **Placement and Routing**

Since there is no tool for floorplanning in Alliance, just guidelines. I constructed the chip floorplanning by hand.

The ALLIANCE tools used in this chapter are:

i OCP: a standard cell placer. It uses (.ioc) file to specify placement for external connectors (pins).
ii GRAAL: a symbolic layout editor tool. It visualizes the placement using the symbolic technology parameter file techno/techno-symb.rds.
iii NERO: an over-cell router.
iv COUGAR: a symbolic netlist extractor. It is a hierarchical extractor that is applied to a symbolic layout to get gate-level netlist (.al).
v LVX: a netlist comparator. It compares the extracted netlist from COUGAR (.al) with input netlist (.vst). Then I used GRAAL again for design rule check (DRC).

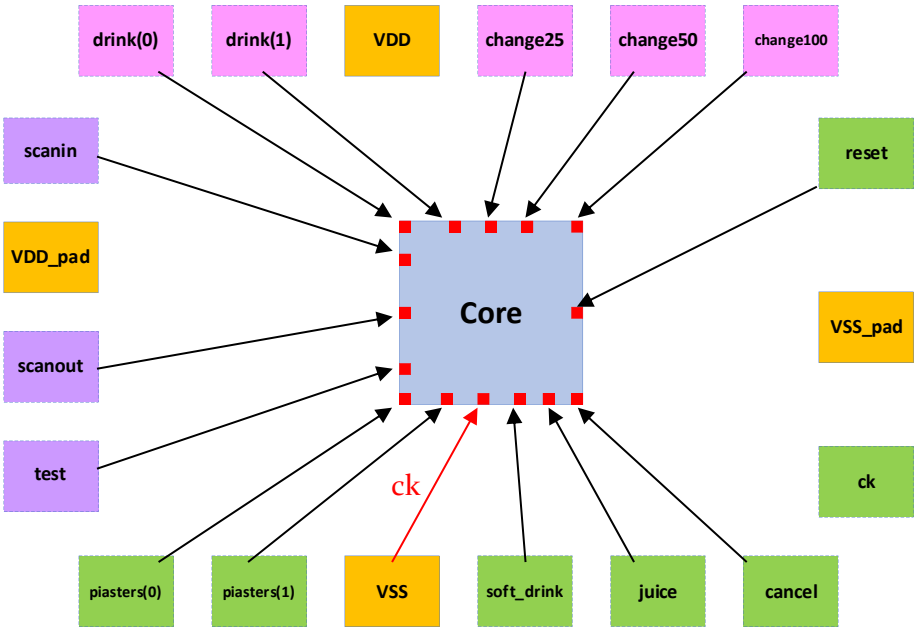
In order to complete the chip, IO pads must be placed around the core in consistence with the designed floorplan. The first step in chip assembly is to create the chip netlist, which instantiates the core netlist in addition to the needed pads (.vst).

I wrote it by hand since a top-level netlist including the pads does not exist.

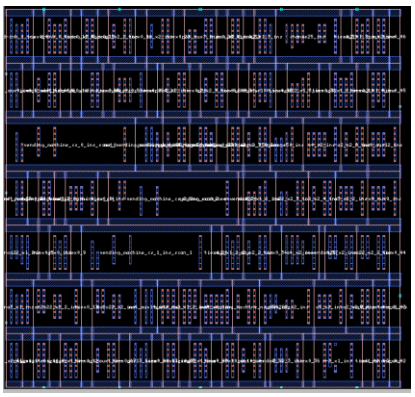
i XSCH: used to view the netlist which includes the pads
ii RING: a pad ring router. It uses (.rin) file to specify placement of pads around the core. Then I used COUGAR, and LVX again to verify the chip.
iii S2R: a symbolic-to-real layout converter. It generate (GDSII) (.cif) using the real technology parameter file techno/techno-035.rds.
iv DREAL: a real layout editor.

4.1 Floorplanning

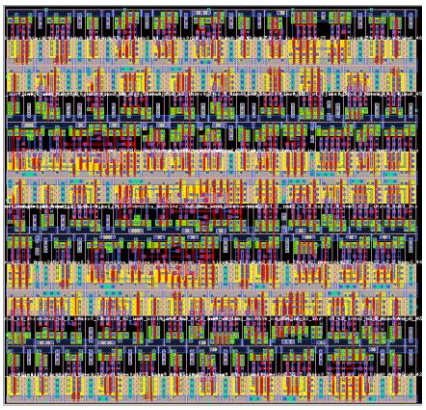
In this project there is only one block in the core, so I will determine I/O placement only.



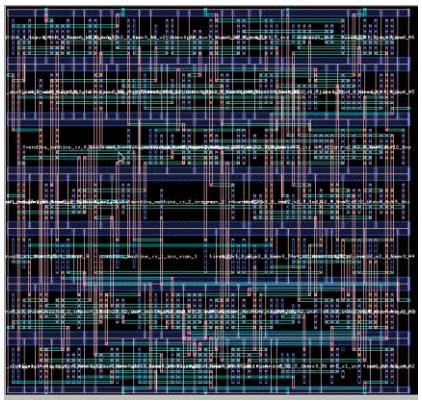
4.2 Placement



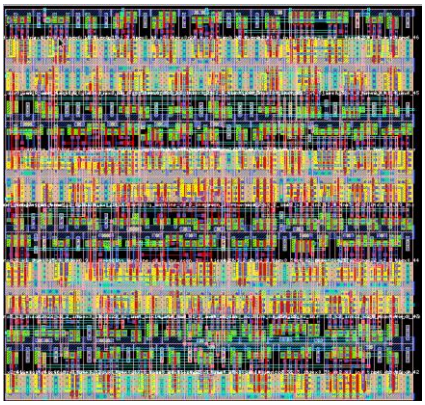
Flattened:



4.3 Routing



Flattened:



4.4 Post Layout Verification

4.6.1 Layout vs Schematic

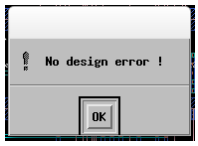
LVS compares the extracted netlist with input netlist.

```
***** Loading and flattening vending_machinea_s (vst)...

***** Loading and flattening vending_machinea_s (al)...


***** Compare Terminals ..............
***** O.K.      (0 sec)

***** Compare Instances ...........
***** O.K.      (0 sec)

***** Compare Connections ................
***** O.K.      (0 sec)

===== Terminals .......... 17
===== Instances .......... 79
===== Connectors ......... 435


***** Netlists are Identical. *****     (0 sec)
```

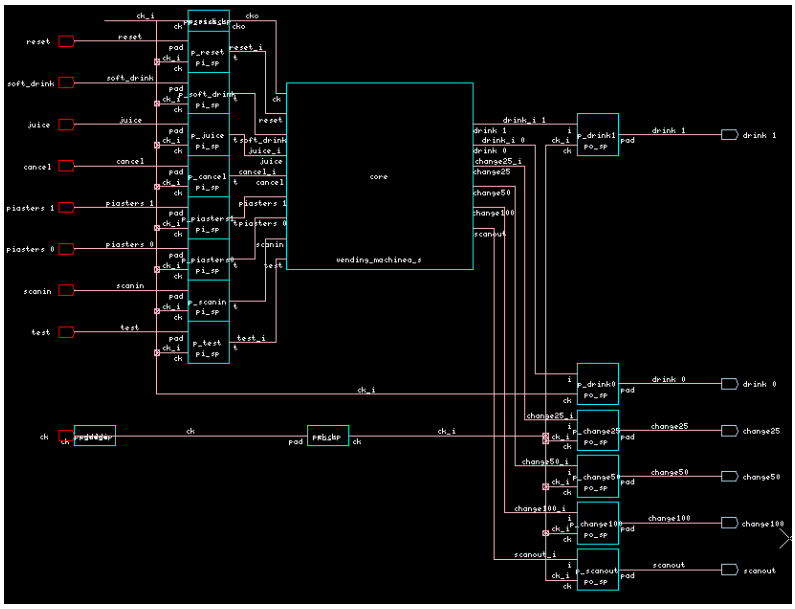
4.6.2 Design Rule Check

Physical verification checks if there are design-rule errors in the generated layout.

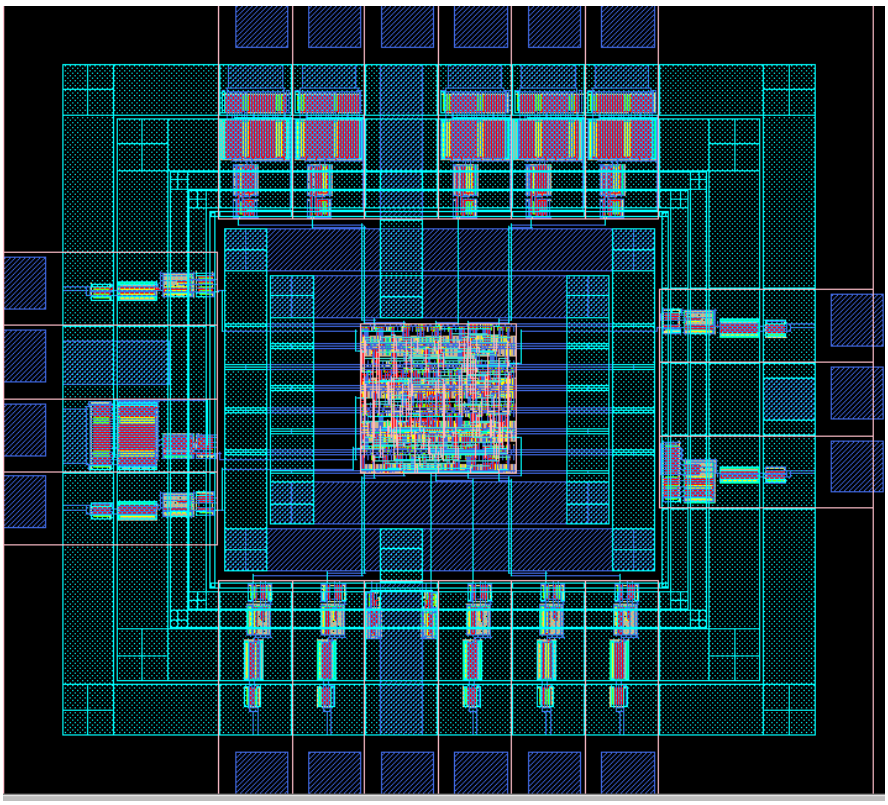
Using GRAAL:



4.5 Chip Assembly



Chip layout:



4.6 Chip Verification

4.6.1 Layout vs Schematic

```
Warning 2 : consistency checks will be disabled
***** Loading and flattening chip (vst)...

***** Loading and flattening chip_layout (al)...


***** Compare Terminals ..............
***** O.K.      (0 sec)

***** Compare Instances ...........
***** O.K.      (0 sec)

***** Compare Connections ..................
***** O.K.      (0 sec)

===== Terminals .......... 19
===== Instances .......... 91
===== Connectors ......... 532


***** Netlists are Identical. *****     (0 sec)
```

4.6.2 Design Rule Check

4.7 Symbolic-to-Real Conversion

The final chip after flattening and viewing using DREAL tool