Using Synopsys Machine Learning and AI tools to optimize PPA with reduced designer effort

Caspar Röper - SoC Backend Engineer
Dream Chip Technologies GmbH

Who we are

- **Who we are**
- Testing Machine Learning Macro Placement (MLMP)
- Determining minimum diesize
- Using Design Space Optimization AI (DSO.ai) to improve power
- Conclusion

Dream Chip Technologies

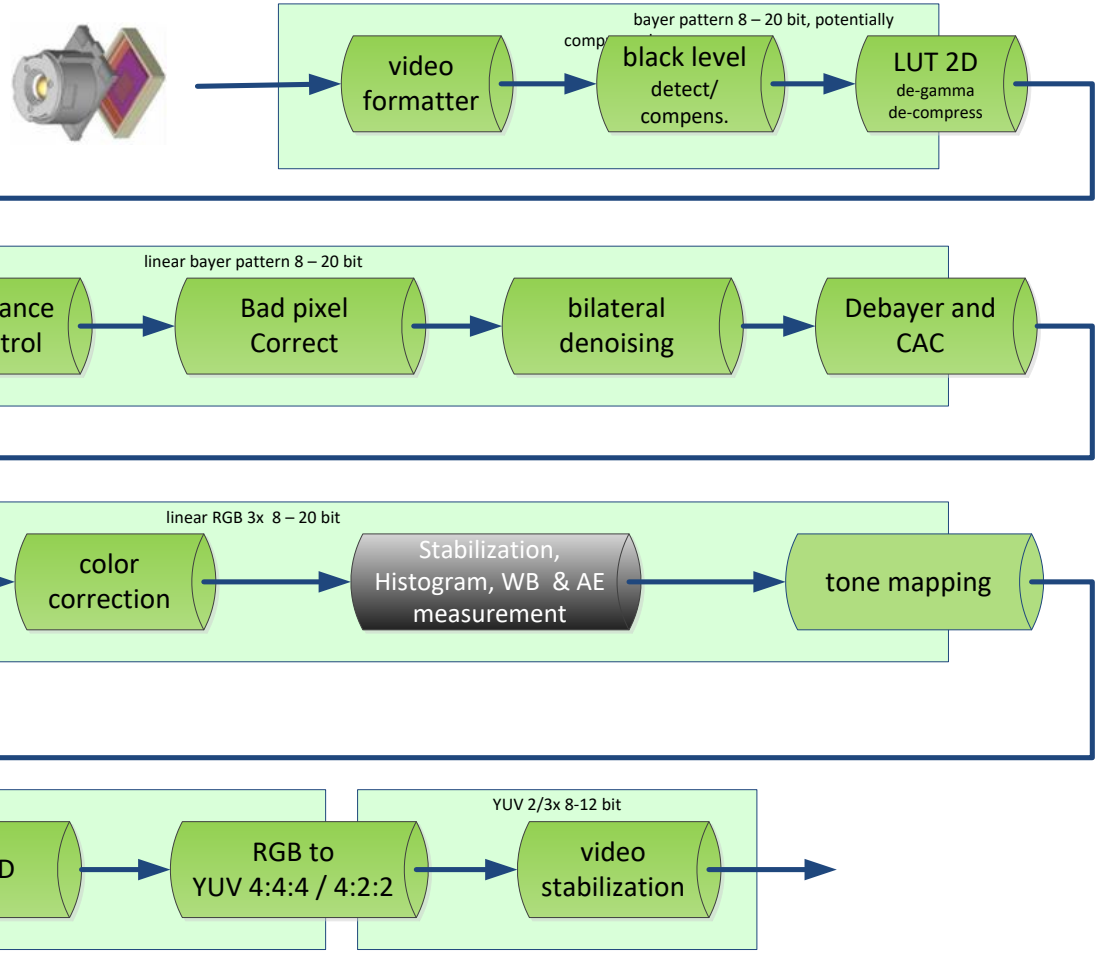
Our company

- Chip solutions from spec to GDSII
- Focus on vision processing, automotive
- ~120 Employees across 4 sites
- Founded in 2009, history goes back to 1990
- Tiny Broadcast cameras, full systems & software

- Complete pipeline starting from image sensor data
- Machine and human vision output
- Low power consumption
- Low compute resource requirements (< 5 MIPS / frame)
- Low area requirements
- Very low latency

Dream Chip Technologies

Our ISP IP

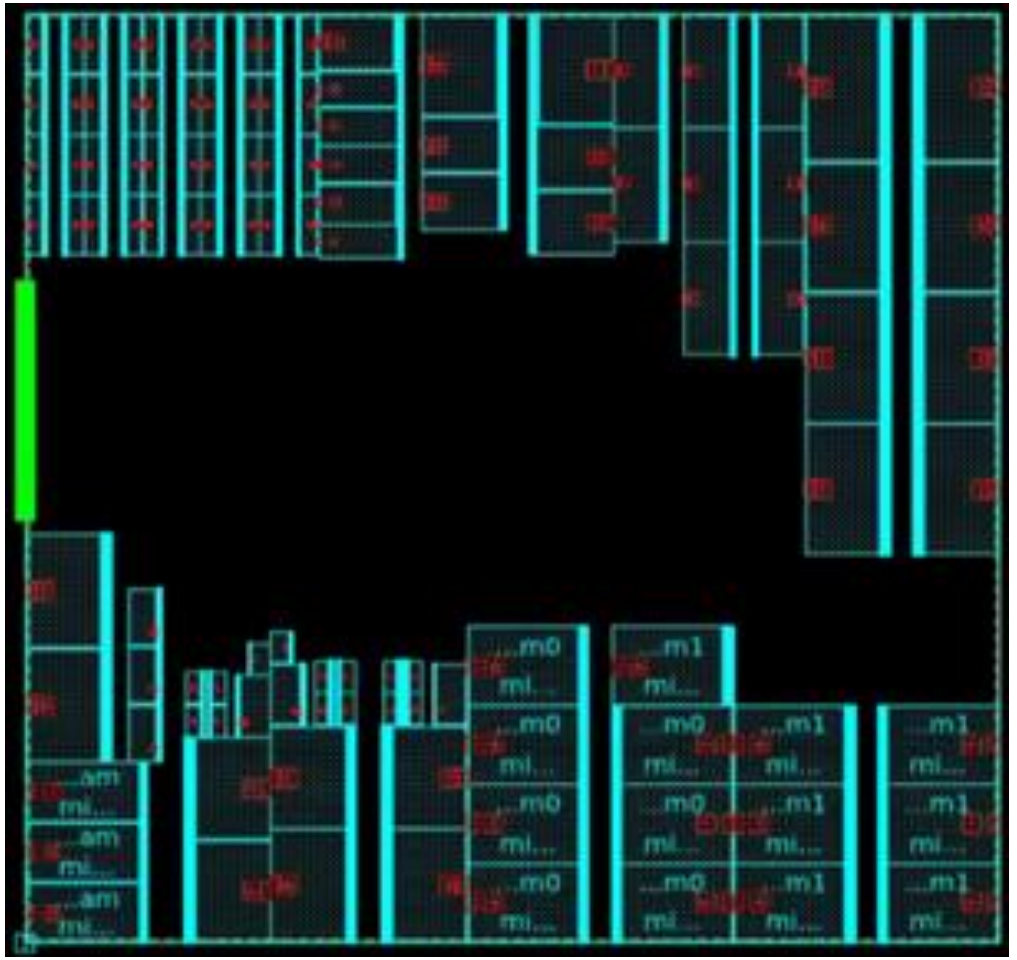


Testing MLMP

- Who we are
- **Testing Machine Learning Macro Placement (MLMP)**
- Determining minimum diesize
- Using Design Space Optimization AI (DSO.ai) to improve power
- Conclusion

Testing Machine Learning Macro Placement (MLMP)

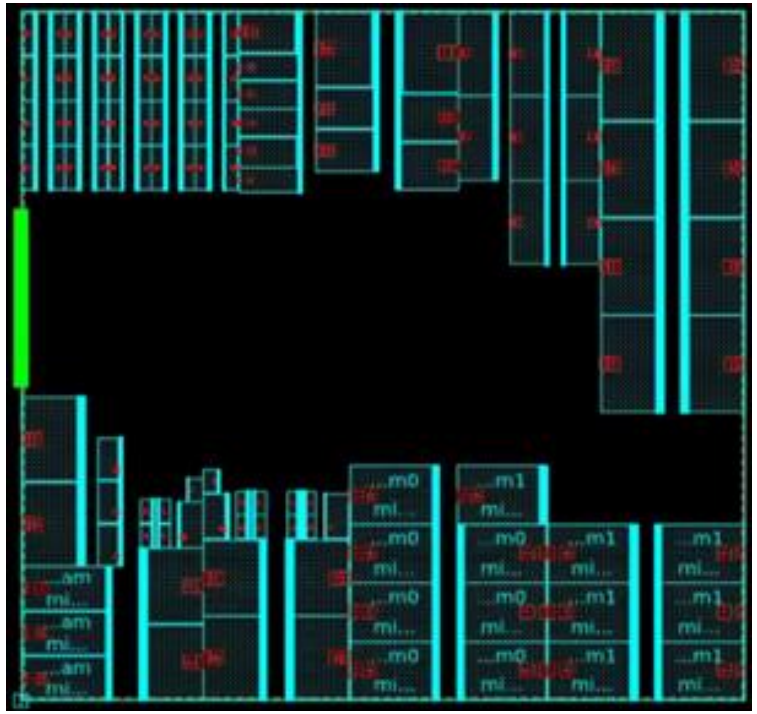
- Our ISP, synthesized in 7nm technology
- 113 memories of various sizes
- ~ 1.25M stdcell instances
- First, manually created floorplan



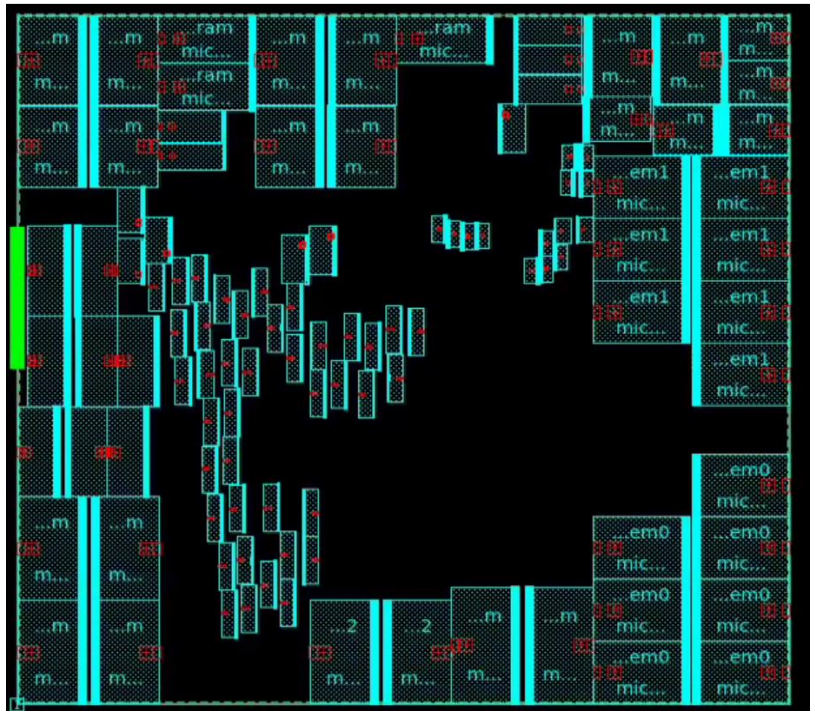
Testing Machine Learning Macro Placement (MLMP)

- Automated RAM placement (not MLMP)
  - Same area
  - Same netlist
  - Similar pin positions
- Works…
  - Timing is clean
  - Only few Design Rule Check (DRC) violations
- …but
  - Pin access is obstructed
  - No continuous stdcell area

Manual floorplan



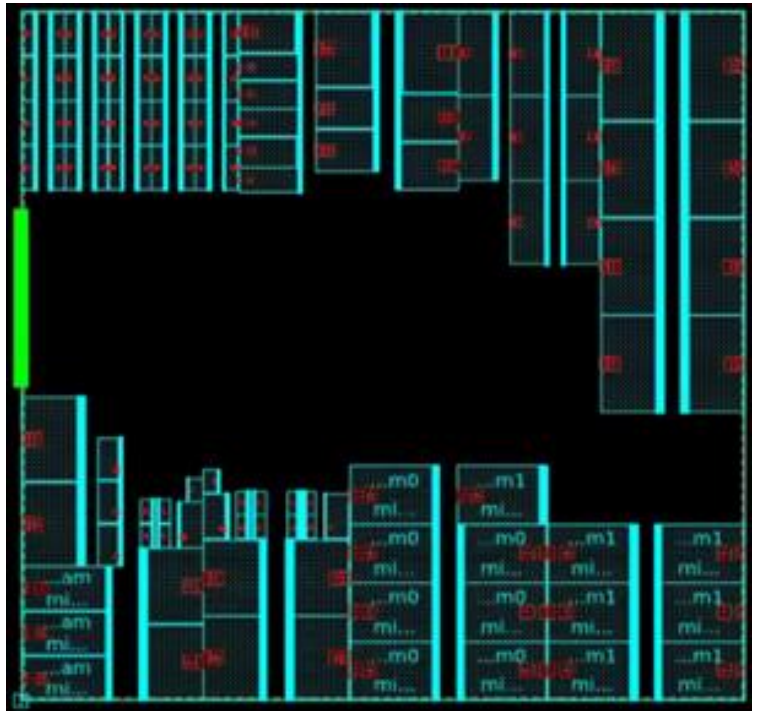
Automated Placement



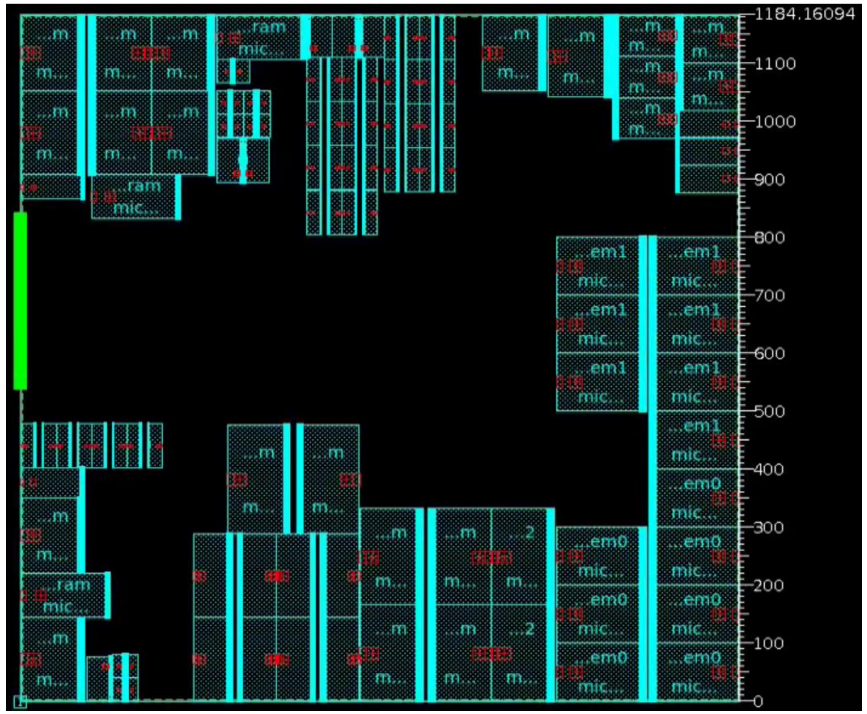
Testing Machine Learning Macro Placement (MLMP)

- Machine Learning Macro Placement (MLMP)
  - Will create parallel placement jobs (12 here)
  - Will iterate on the best one & improve it further
  - Controllable via parameters (placement on_edge / hybrid)
- Works…
  - Timing is clean
  - Few DRC violations
- …and with some tuning
  - Pins are accessible
  - Continuous stdcell area

Manual floorplan



MLMP



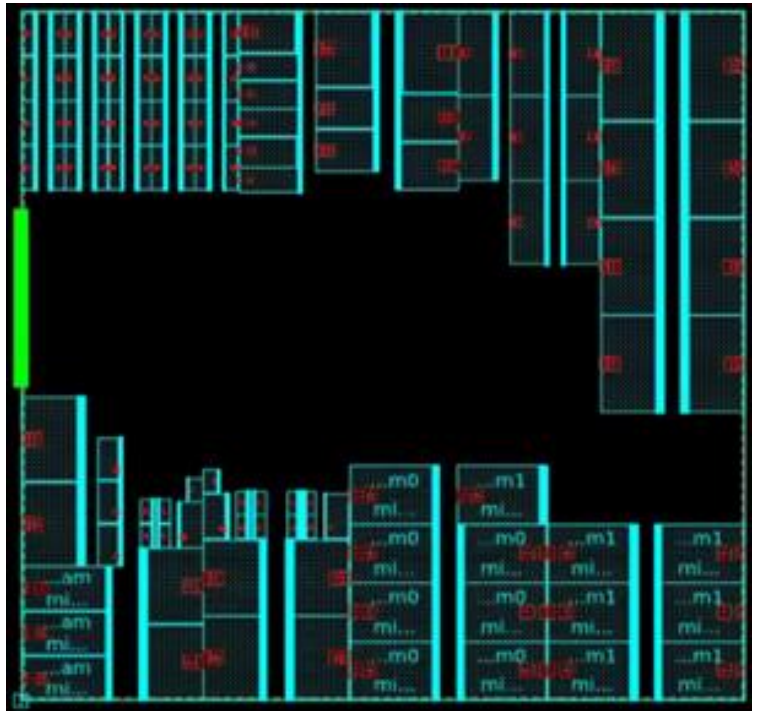
Testing Machine Learning Macro Placement (MLMP)

- Machine Learning Macro Placement (MLMP) can create memory placements that fulfill following criteria when compared to human designs:
  - Similar timing performance
  - Similar amount of design rule violations
  - Have no pin access issues
  - Have a continuous stdcell area
- This is important because
  - It can save us manual design work
  - **It should allow us to automatically determine a reasonable layout for any floorplan size**

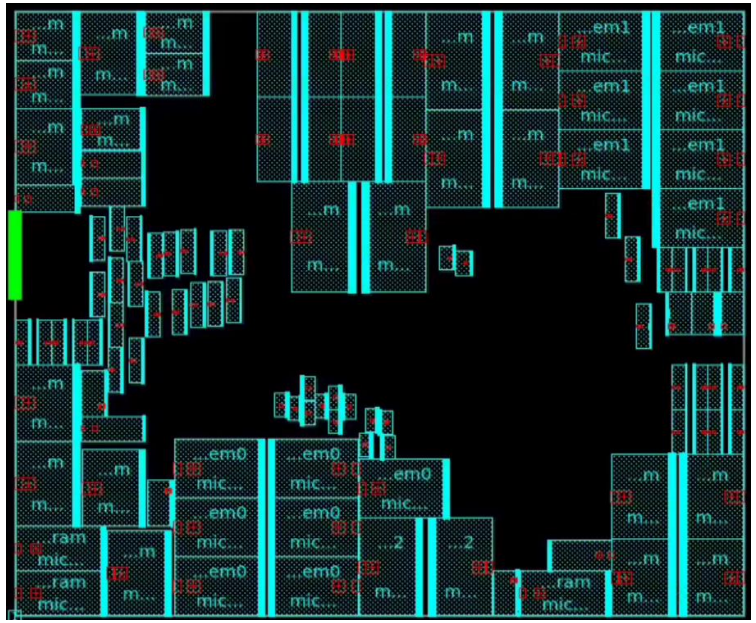
Testing Machine Learning Macro Placement (MLMP)

- Machine Learning Macro Placement (MLMP)
  - Reduced area (-13%)
  - Same netlist
  - Similar pin positions
  - Target frequency doubled
- Timing violated, but probably fixable
  - < 100 Failing end points for setup / hold
- Some problems with congestion
  - 820 DRC violations

Manual floorplan



MLMP, reduced size



Determining minimum diesize

- Who we are
- Testing Machine Learning Macro Placement (MLMP)
- **Determining minimum diesize**
- Using Design Space Optimization AI (DSO.ai) to improve power
- Conclusion

Determining minimum diesize

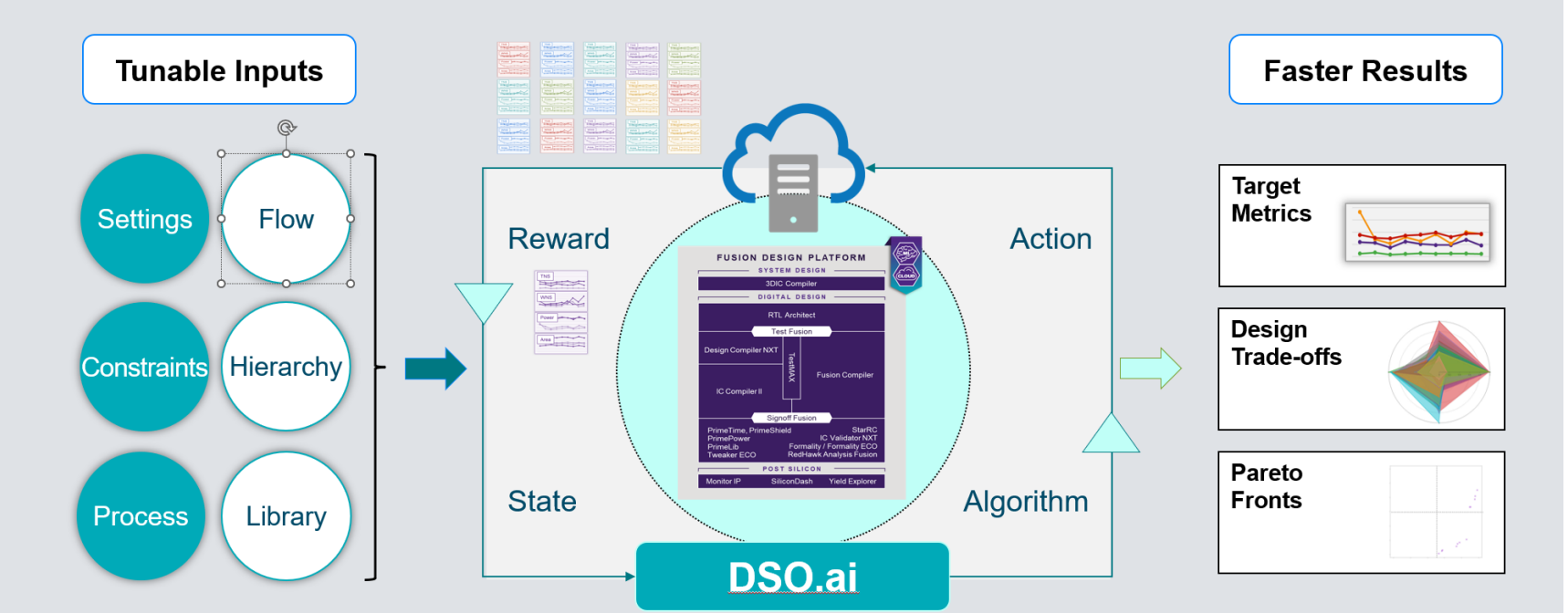
- Since MLMP can create usable floorplans automatically, we should be able to determine minimum floorplan size by providing only a small number of variable inputs:
  - Width
  - Height
  - Pin positioning is derived from height
- To measure if a floorplan size is feasible, we need some metric to compare them
  - We chose congestion
- This could be done manually or in a scripted fashion, but preferably, a tool should do the work & evaluation for us

Determining minimum diesize

- Design Space Optimization AI (DSO.ai) can do that
  - User provided input parameters (or defaults) are changed per-run
  - Runs are executed and evaluated based on a given metric / cost function
- Baseline flow is needed first, of course

DSO.ai – Design Space Optimization Loop

Uses reinforcement-learning to navigate the design-technology solution space



Determining minimum diesize

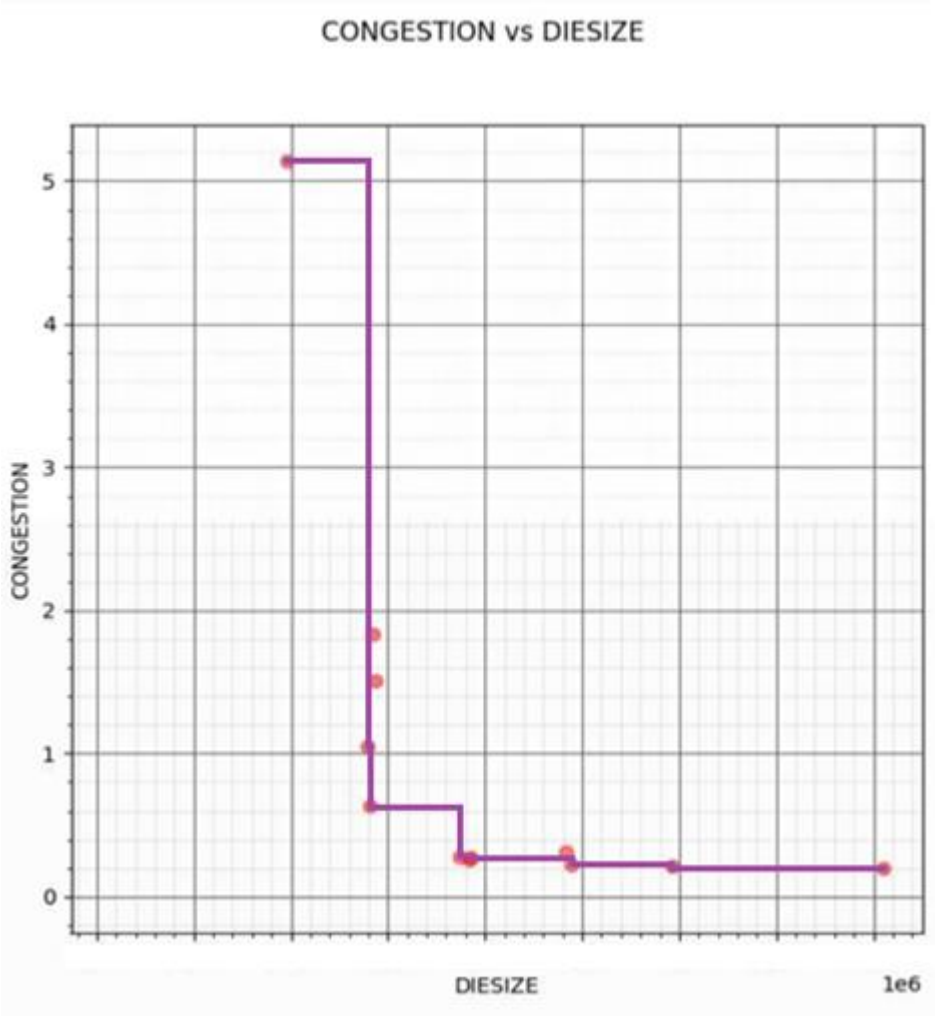
- Our setup for DSO.ai
  - Width & Height as parameters
  - Can take four discrete values
  - All other parameters fixed
  - Macro placement is done by MLMP, starting & evaluating runs is done by DSO.ai

  →16 possible combinations + baseline run = 17 runs in total
  - Only metric of interest is congestion in this scenario

- Beware: DSO.ai will start 17 variants. In each, MLMP will create several placement jobs (12 in our case) for a total of 17x12=204 placement jobs – assuming 16 threads per job, that are 3264 threads → sufficient compute resources are needed

Determining minimum diesize

- Result is the distribution of reported congestion over calculated diesize
- From the results, diesizes below certain area are not worth pursuing



Using DSO.ai to improve power

- Who we are
- Testing Machine Learning Macro Placement (MLMP)
- Determining minimum diesize
- **Using Design Space Optimization AI (DSO.ai) to improve power**
- Conclusion

Using DSO.ai to improve power

- Switching over to “next” generation of our ISP
  - Stdcells: 1.25 M insts → 1.35 M insts
  - Memories: 113 insts → 190 insts
- Task: Improve Power, Performance & Area (PPA)
- Use automation where possible

Using DSO.ai to improve power

- MLMP used to create an initial floorplan
- At nearly 70% memory area, MLMP did create a floorplan with several ‘holes’
- So MLMP placement was used as starting point for manual refinement

MLMP

Manually refined floorplan

Using DSO.ai to improve power

- Our setup for DSO.ai
  - Width and height fixed (using manually refined floorplan with MLMP as starting point)
  - Optimization targets are
    - Setup register to register worst negative slack (R2R_WNS)
    - Stdcell power (active + leakage)
  - For each of the “compile”, “clock” and “route” steps, 20 runs are started
  - 16 threads used per run

- Beware: In this config, DSO.ai will start 20 jobs in parallel. That means 20x16=320 threads are used – again, sufficient compute resources are needed
  - The three phases (“compile”, “clock” and “route”) are run in order, i.e., the “clock” phase is only started once all “compile” runs are done
  - If one of the “compile” runs is done on a slower machine, this delays the whole DSO run
  - Therefore, homogenous compute servers should be used

Using DSO.ai to improve power

- Starting from the baseline, reductions of 11% in stdcell leakage power are possible
- Very little variance in dynamic power
- At the same time, achievable frequency (PBA_R2R_WNS) slightly improves
- PBA_R2R_WNS would have been a better metric, as it is less pessimistic

```
dso_shell> report_session_results -anchor_baseline -num_runs 20
```

score_route	DYN_PWR_NOMEM	LEAKAGE_STD_CELL	R2R_WNS	STATUS	PBA_TNS	PBA_WNS	PBA_R2R_WNS	PBA_R2R_TNS	ID	BLOCK_SAVE
0.780	86063000.0	85210806.5	-0.073	DONE	-4.852	-0.261	-0.030	-0.392	4c07e522	score_route:0
0.782	86056000.0	85592255.3	-0.072	DONE	-5.025	-0.259	-0.023	-0.391	622ade31	score_route:1
0.782	85955000.0	85344435.3	-0.074	DONE	-5.188	-0.260	-0.022	-0.322	d5f6ea0e	score_route:2
0.784	85955000.0	85586085.0	-0.073	DONE	-5.318	-0.258	-0.031	-0.383	8abe292b	score_route:3
0.784	86056000.0	85222803.4	-0.074	DONE	-5.712	-0.261	-0.027	-0.403	8acee41e	score_route:4
0.791	86350000.0	85500298.6	-0.071	DONE	-6.027	-0.296	-0.029	-0.984	c8956e06	score_route:5
0.793	86151000.0	85418931.8	-0.074	DONE	-5.284	-0.262	-0.028	-0.375	cf955e37	score_route:6
0.798	86071000.0	85263897.2	-0.077	DONE	-4.958	-0.261	-0.024	-0.352	f6dcca0d	score_route:7
0.802	86147000.0	81719997.4	-0.089	DONE	-5.284	-0.260	-0.029	-0.733	81435d29	score_route:8
0.805	85958000.0	85849220.0	-0.077	DONE	-5.274	-0.260	-0.050	-0.381	e588a72d	score_route:9
0.805	85686000.0	85019784.5	-0.084	DONE	-6.108	-0.276	-0.021	-0.244	6a85f226	
0.805	86052000.0	85605470.3	-0.077	DONE	-6.170	-0.261	-0.026	-0.209	eb7e9424	
0.807	86354000.0	85399671.8	-0.076	DONE	-5.273	-0.292	-0.022	-0.531	5f4fec05	
0.808	85950000.0	85267847.7	-0.081	DONE	-6.016	-0.261	-0.033	-0.647	0d05890f	
0.813	86453000.0	85552427.0	-0.076	DONE	-6.264	-0.291	-0.038	-1.264	989cf407	
0.814	86251000.0	85449010.0	-0.078	DONE	-5.733	-0.296	-0.040	-0.700	344f0f23	
0.815	86354000.0	85818549.7	-0.076	DONE	-5.616	-0.290	-0.027	-0.583	c247502e	
0.821	86353000.0	85541108.5	-0.079	DONE	-5.564	-0.289	-0.031	-0.560	2e49e920	
0.823	85791000.0	82501359.8	-0.091	DONE	-7.012	-0.278	-0.019	-1.416	58960617	
0.823	86047000.0	85518056.1	-0.082	DONE	-5.834	-0.259	-0.027	-0.221	ce39343b	
1.000	87763000.0	91301792.5	-0.084	DONE	-5.228	-0.220	-0.032	-0.393	24ff9c00	user_baseline

Conclusion

- Who we are
- Testing Machine Learning Macro Placement (MLMP)
- Determining minimum diesize
- Using Design Space Optimization AI (DSO.ai) to improve power
- **Conclusion**

Conclusion

- MLMP
  - Can create floorplans competing with human designed ones
  - For designs with high memory area / stdcell area ratio, MLMP has its limits
  - Still, it can be helpful as a starting point to create a refined floorplan
- DSO.ai
  - Can help run parameter exploration
  - Needs correct setup
    - Working base flow
    - Proper selection of cost function is critical
  - Can be used to improve PPA with little designer intervention

	Leakage	Dynamic	PBA_R2R_WNS
Baseline	91.3 mW	877 mW	-32 ps
Best DSO.ai	81.7 mW	861 mW	-29 ps
Difference	-11 %	-2 %	same

- Both
  - Trade designer effort for compute runtime


THANK YOU
YOUR
INNOVATION
YOUR
COMMUNITY