A Novel Wirelength-Driven Packing Algorithm for FPGAs with Adaptive Logic Modules

Speaker: Po-Yi Hsu

Sheng-Kai Wu, Po-Yi Hsu, Wai-Kei Mak

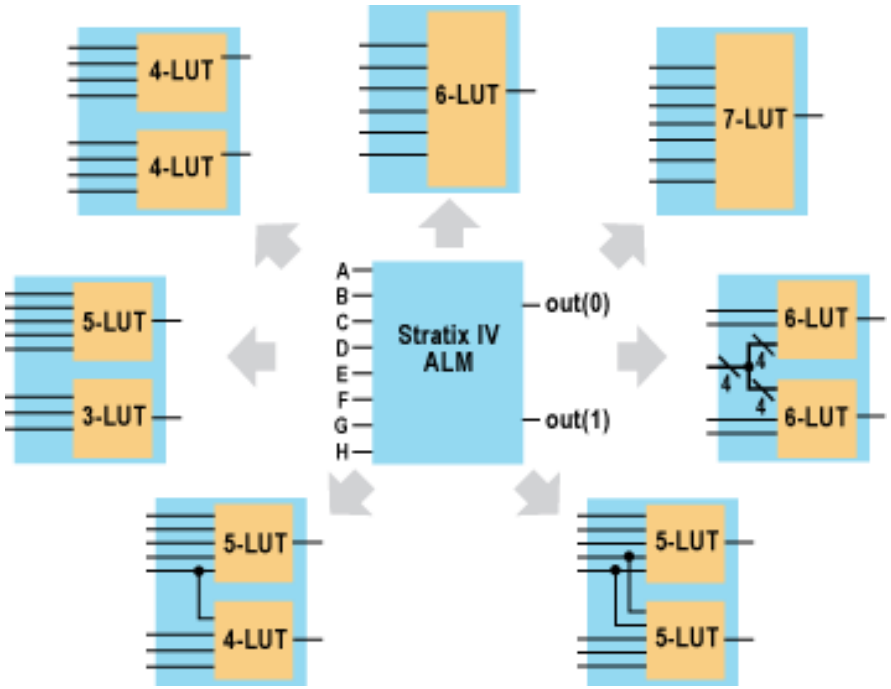
Dept. of CS, National Tsing Hua University

Outline

- Introduction
- Preliminaries
- Algorithm
- Experimental results
- Conclusion

Fracturability of ALM

- ALM can serve as a 6-input LUT or two smaller LUTs if the total distinct inputs is less or equal to 8 under certain constraints.



(From http://www.altera.com)

Recent Research of ALM

- [Hutton FPL04] reported that the ALM architecture led to a 15% timing improvement and 12% area reduction on average versus a standard BLE4 architecture.
- How to merge two LUTs into one ALM so that it will not adversely affect the quality for later placement and routing is an important issue.
- In this work, we propose a novel packing algorithm for ALM-based FPGA targeting directly at wirelength minimization.

Outline

- Introduction
- Preliminaries
  - The idea of safe clustering
  - Problem formulation
- Algorithm
- Experimental results
- Conclusion

The idea of safe clustering

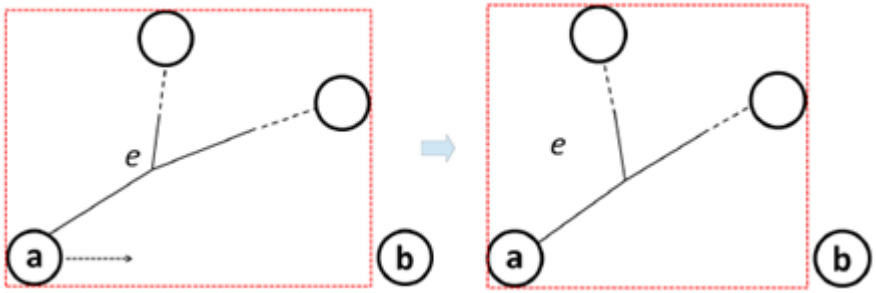
- The concept of safe clustering for effectively reducing the problem size in ASIC placement was introduced in [Yan TCAD12].

- Safe clustering guarantees that the clustering will not adversely affect the final total wirelength.

Gradient functions

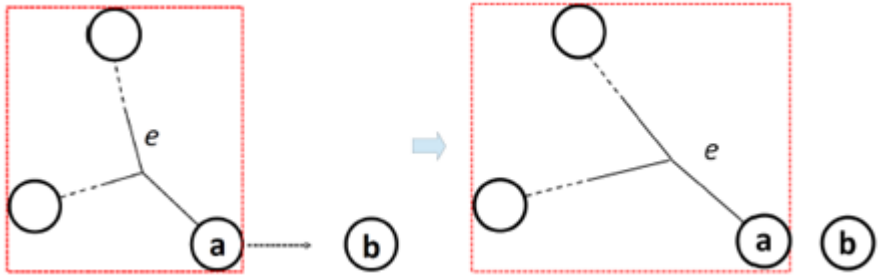
- Given a hypergraph $G(V, E)$, where V is the set of nodes and E is the set of hyperedges corresponding to the nets.
- P: the set of all possible valid placements
- E_v: the set of hyperedges incident to v

$p \in P$, $e \in E$, a pair of nodes $\{a, b\}$ with a on the left of b



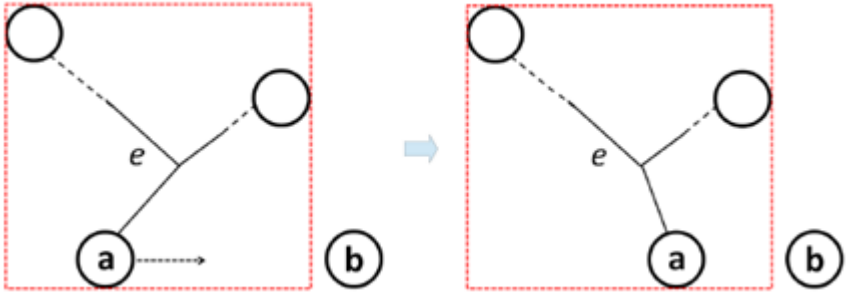
if a is the rightmost vertex of e

$$\Delta_a(p, e) = w_e$$



if a is the only leftmost vertex of e

$$\Delta_a(p, e) = -w_e$$



otherwise

$$\Delta_a(p, e) = 0$$

Total wirelength gradient function

- From the above gradient function, the total wirelength gradient function is defined as follows:

$$F_{ab}(p) = min(\sum_{e \in E_a} \Delta_a(p, e), \sum_{e \in E_b} \Delta_b(p, e))$$

- If all possible placements Fab is not greater than zero. It is safe to cluster a and b.

$$max(F_{ab}(p_1), F_{ab}(p_2), ..., F_{ab}(p_l)) \leq 0 \quad p_i \in P$$

- However, it is not practical to generate all possible placements when considering clustering a and b.

Selective enumeration

- A selective placement enumeration approach is proposed in [Yan TCAD12].

- It only enumerates the placements that might generate worse wirelength if we merge a and b.

All placement $|P| = \infty$

Only consider V_{ab} nodes connected with at least one of *a* or *b*

For each node $v \in V_{ab}$
(1) v is on the left of a
(2) v is between a and b
(3) v is on the right of b

$3^{|V_{ab}|}$

[Yan TCAD12] has proven that case(2) will never be worse than case(1) or case(3).

Only two possible locations for v $2^{|V_{ab}|}$

[Yan TCAD12] identified a subset of nodes in *Vab* for which it is unnecessary to consider both possible positions for them.

Final placements that need to be enumerated to check the safeness $2^{|V_{ab}|-\alpha}$

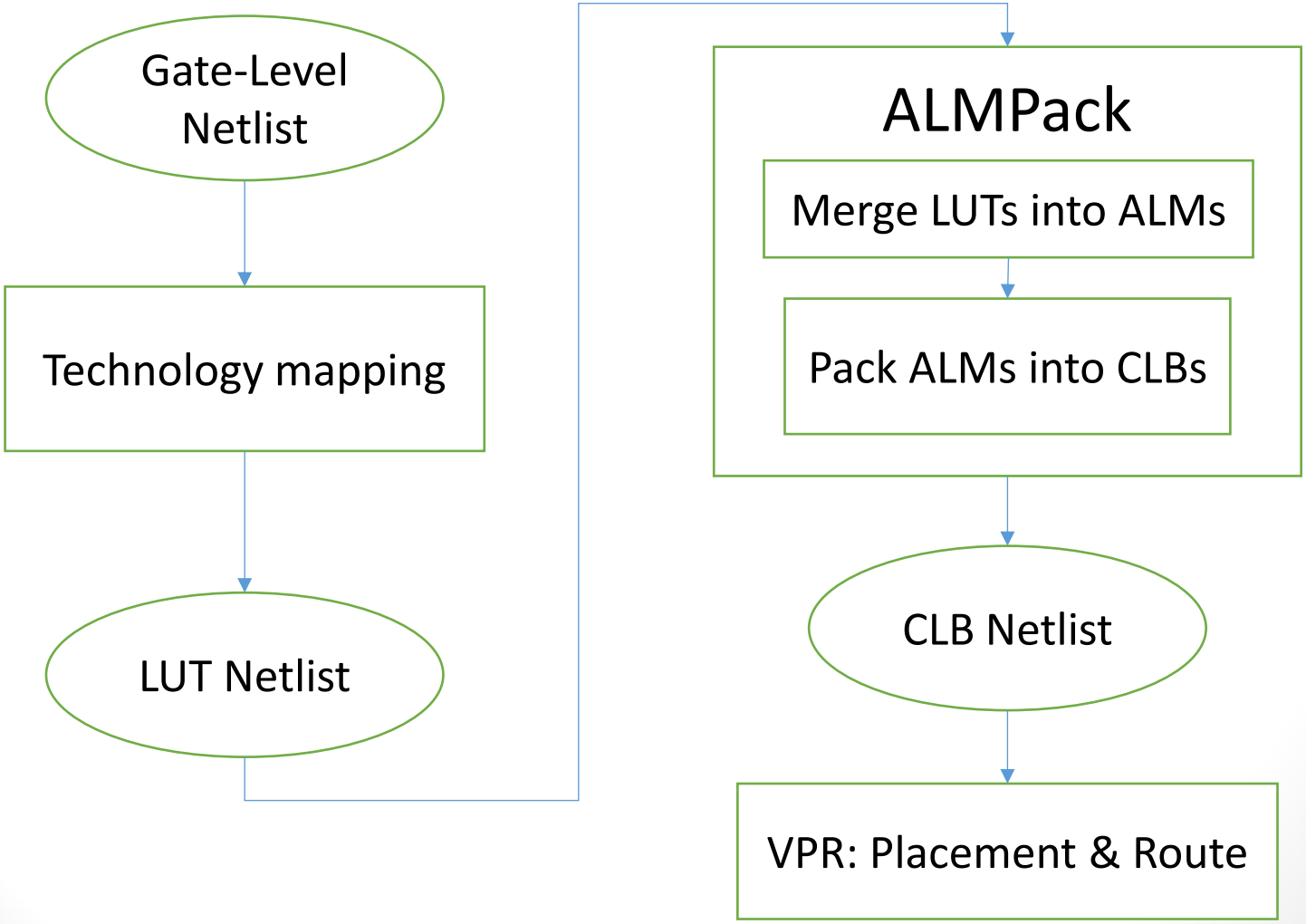
Problem formulation

- Given:
  - A mapped netlist of 6-input LUTs
- Objective:
  - Merge the LUTs into ALMs under the ALM architecture constraint and cluster ALMs into CLBs so as to optimize the expected wirelength after place and route.

Outline

- Introduction
- Preliminaries
- Algorithm
  -Merge LUTs into ALMs
  -CLB clustering
- Experimental results
- Conclusion

Overall flow

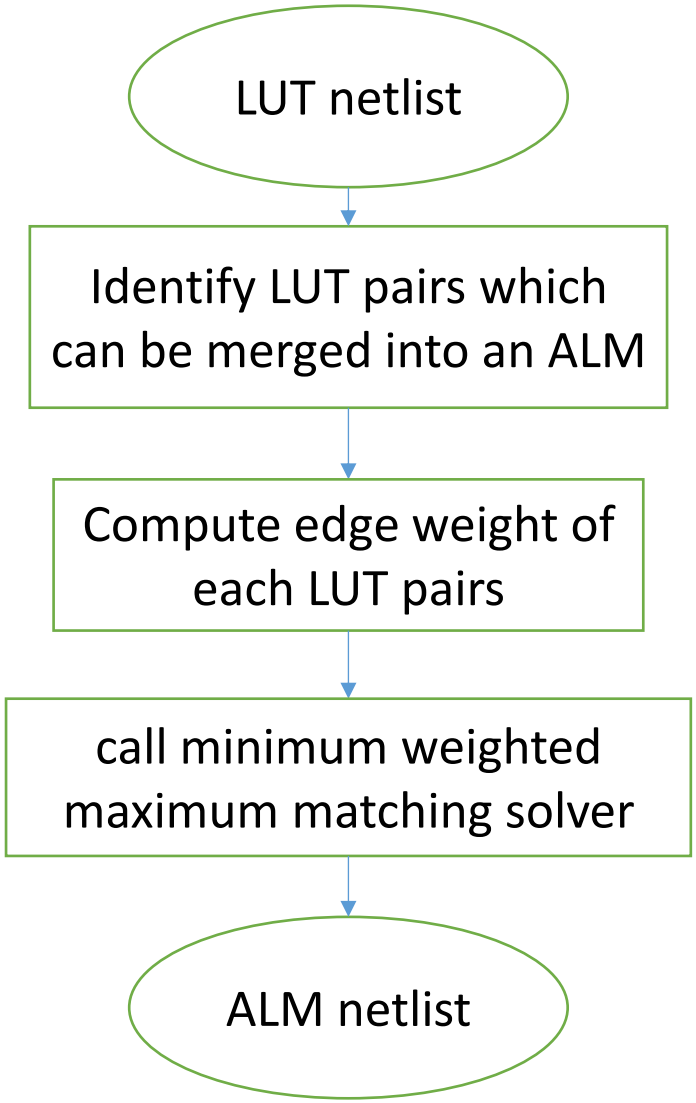


Merge LUTs into ALMs

- We model the wirelength-driven ALM formation problem as a ***minimum weighted maximum matching*** problem.

- We construct a weighted undirected graph where each node corresponding to a LUT, and there is an edge between two nodes if and only if the corresponding LUTs can be merged into an ALM.

Merge LUTs into ALMs flow

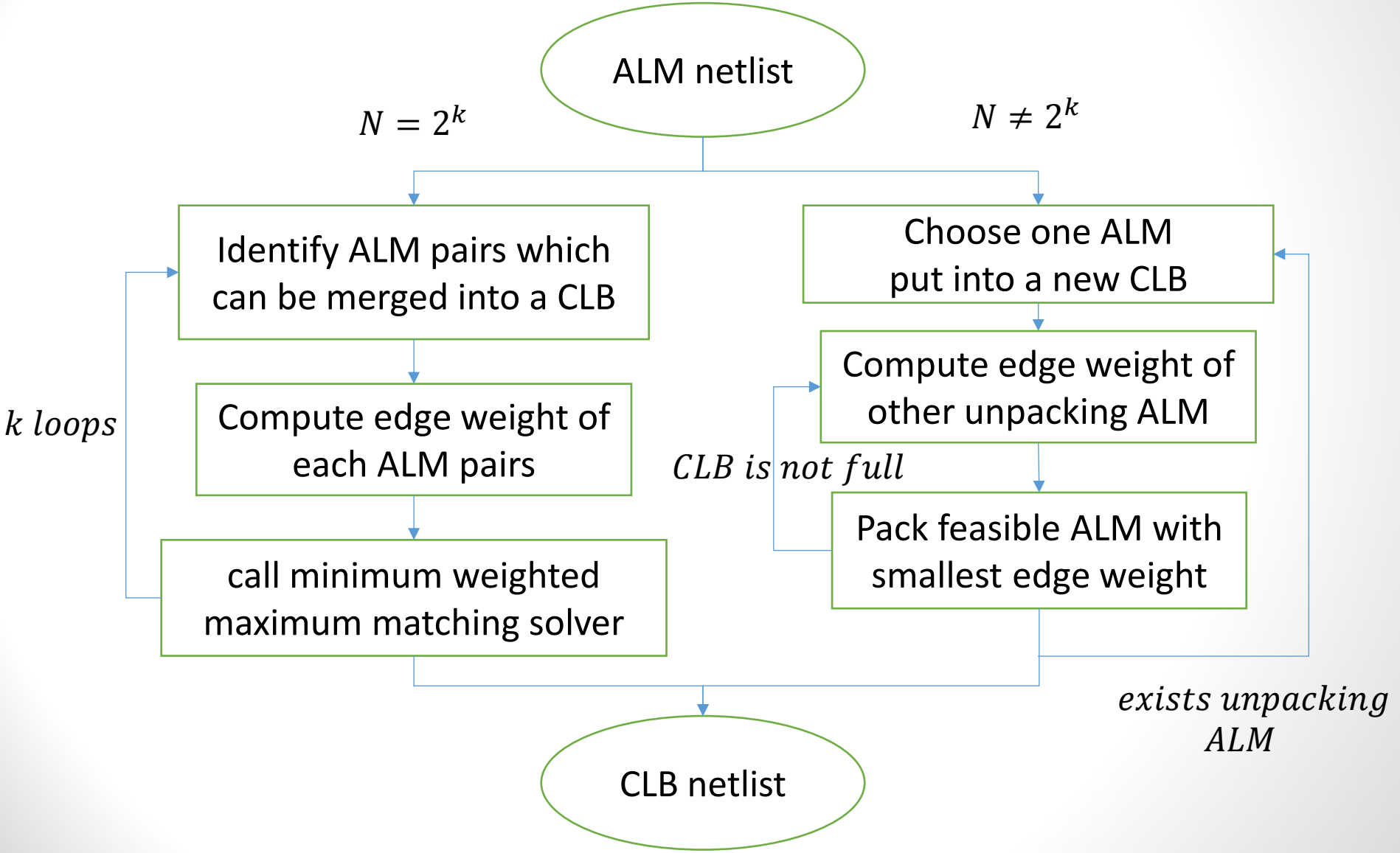


Edge weight

- We define the edge weight between two nodes a and b as follows:

$$weight(a,b) = \begin{cases} 0, & \text{if } 2^{|V_{ab}|-\alpha_{ab}} \leq enum\_bound \\ & \text{and } \forall i \; F_{ab}(p_i) \leq 0 \\ \frac{(\sum_{i=1 s.t. F_{ab}(p_i)>0}^{L_{ab}} F_{ab}(p_i))}{L_{ab}}, & \text{if } 2^{|V_{ab}|-\alpha_{ab}} \leq enum\_bound \\ & \text{and } \exists i \text{ s.t. } F_{ab}(p_i) > 0 \\ B - \beta \frac{|NET(a) \cap NET(b)|}{|NET(a) \cup NET(b)|}, & \text{if } 2^{|V_{ab}|-\alpha_{ab}} > enum\_bound \end{cases}$$

Pack ALMs into CLBs



Outline

- Introduction
- Preliminaries
- Algorithm
- Experimental results
  - ALM-based FPGA (compare to AAPack)
  - Traditional BLE4 FPGA (compare to T-VPack)
- Conclusion

Environment Setup

- We implemented our packing algorithm, ALMPack, using C++ on an Ubuntu workstation with 8 GB memory and 2.13 GHz CPU.
- 20 largest MCNC benchmarks for the experiments.

	Exp. 1	Exp. 2
Logic element	ALM	4-LUT
Cluster size(N)	8	8
#inputs of CLB(I)	34	22
Placement Algorithm	Timing-driven	Timing-driven
Routing Algorithm	Timing-driven	Timing-driven
Segment length	4	4
Switch type	Buffered	Buffered
F_c(pad,input,output)	(1,0.5,0.25)	(1,0.5,0.25)
F_s	3	3

ARCHITECTURE AND EXPERIMENTAL SETTINGS

ALM-based FPGA

Bench.	# LUTs	# CLBs		min. channel width		wirelength		delay(10^{-8} sec)	
		AAPack	ALMPack	track	ALMPack	AAPack	ALMpack	AAPack	ALMPack
ex5p	745	54	57	58	48	5372	4503	0.824	0.698
spla	2072	166	187	106	86	27651	24123	1.4491	1.172
alu4	803	65	70	62	58	7395	6956	0.9479	0.795
apex2	1058	88	85	74	72	11538	10554	1.073	1.002
apex4	787	65	63	72	72	8904	7198	1.1377	0.851
des	555	46	46	40	38	4999	4063	0.4674	0.461
ex1010	2703	224	221	114	94	35318	32048	1.1002	1.235
misex3	818	65	66	62	60	7349	7170	0.8506	0.905
pdc	2417	195	206	116	96	33328	31907	1.1725	1.341
seq	960	77	76	74	74	9011	8804	0.6801	0.788
bigkey	579	50	50	44	40	4560	2796	0.4346	0.293
clma	3911	330	328	88	74	41977	31467	1.399	1.247
dsip	689	43	43	38	34	3791	3055	0.3653	0.356
diffeq	660	55	54	44	30	4251	2628	0.5957	0.695
elliptic	1795	140	138	74	50	14711	9946	0.8975	0.969
frisc	1797	133	132	88	74	18962	14221	1.11	1.175
s298	780	62	62	60	58	6314	5859	1.158	1.075
s38417	2781	217	215	54	42	17742	15217	1.308	1.335
s38584	2504	195	185	64	44	18278	14812	0.8522	0.824
tseng	660	51	49	40	28	3740	2217	0.5054	0.586
Avg. Impv.			-0.47%		14.54%		17.97%		2.14%

ALM-based FPGA

- It shows that the final wirelength is improved in all cases and is 17.97% shorter on average using ALMpack.
- Reduced the minimum channel width in 18 of the 20 benchmarks and never increased the minimum channel width.
- We ran AAPack in default mode which optimizes both area and timing, but we still obtained 2.14% improvement for delay on average with comparable area.
- Our delay improvement will be more significant if we route the designs under the same channel widths as AAPack.

Traditional BLE4 FPGA

Bench.	# LUTs	# CLBs		min. channel width		wirelength		delay(10^{-8} sec)	
		T-VPack	ALMPack	T-VPack	ALMPack	T-Vpack	ALMpack	T-Vpack	ALMPack
ex5p	892	116	112	46	42	8214	7355	1.60548	1.22603
spla	3016	386	377	54	52	30214	28341	1.89437	1.9539
alu4	1205	155	151	42	38	9842	9111	1.25202	1.4911
apex2	1441	187	181	46	42	13004	11145	1.44227	1.1804
apex4	1061	138	135	50	44	9825	9006	1.43892	1.2831
des	1238	156	155	28	28	17292	16311	1.1487	1.722
ex1010	3854	513	496	72	58	47525	37386	2.75033	1.7329
misex3	1173	151	150	42	36	9401	8780	1.4199	1.235
pdc	3435	442	430	62	58	38730	36819	2.34965	2.6949
seq	1361	177	172	46	42	12430	11701	1.205	1.1713
bigkey	1146	144	144	18	12	14177	11482	0.5724	0.519
clma	5621	706	703	60	50	55226	40757	1.8288	1.796
dsip	1368	171	171	18	12.00	14155	9780.00	0.6116	0.61
diffeq	981	123	123	22	18.00	4671	3289.00	0.7795	0.871
elliptic	2050	261	257	34	28	17088	11946	1.2081	1.255
frisc	2282	288	286	50	40.00	19404	16656.00	1.3278	1.443
s298	1053	134	133	38	36.00	8746	8347.00	1.5905	1.431
s38417	4978	623	623	32	20.00	23754	15717.00	0.9948	1.031
s38584	4497	563	559	30	20.00	24731	13934.00	1.6917	0.815
tseng	779	98	98	16	12.00	4208	3017.00	0.7555	0.691
Avg. Impv.			1.4		16.59		17.57		3.62

Traditional BLE4 FPGA

- Use WireMap implemented in ABC [10] to generate the netlists of 4-LUTs and compare our algorithm with T-VPack [9].
- We reduced the wirelength for all 20 benchmarks with 17.57% improvement on average.
- We achieved better minimum channel width in 18 of the 20 benchmarks with 16.59% improvement on average.
- Although ALMpack does not directly target to minimize delay, it still obtained 3.71% delay improvement on average compared to the timing-driven packing algorithm, T-VPack, while reducing the channel width by 16.59%.

Outline

- Introduction
- Preliminaries
- Algorithm
- Experimental results
- Conclusion

Conclusion

- We proposed a novel wirelength-driven algorithm to merge the LUTs and pack the ALMs to ensure that it will not adversely affect the final wirelength.

- The experimental results show that our packing algorithm consistently outperforms AAPack for ALM-based FPGA and T-VPack for traditional FPGA by a large margin.

Thank you for listening.