Technology Mapping and Packing for Coarse-grained, Anti-fuse Based FPGAs

Chang Woo Kang, Ali Iranli, and Massoud Pedram

University of Southern California
Department of Electrical Engineering
Los Angeles CA, USA

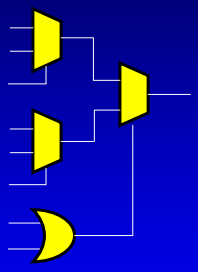
Outline

- Introduction
- Cell Library Construction
- Technology Mapping and Cell Packing
- Experimental Results
- Conclusion

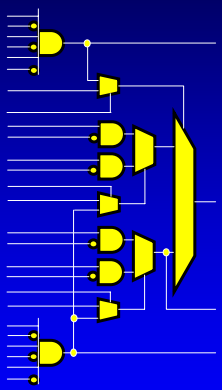
Introduction

- Coarse-grained logic block architecture is widely used in the FPGA industry
  - Xilinx Virtex series has 8 LUTs in a configurable logic block
  - QuickLogic pASIC series has 26 inputs in a logic cell

Example of Fine vs. Coarse-grained Antifuse-based FPGAs

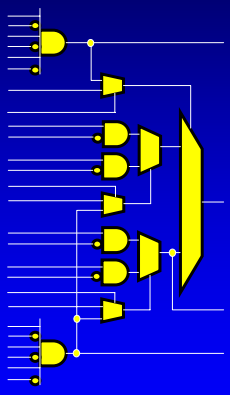


Fine Grain:
Actel ACT 2 Logic Module



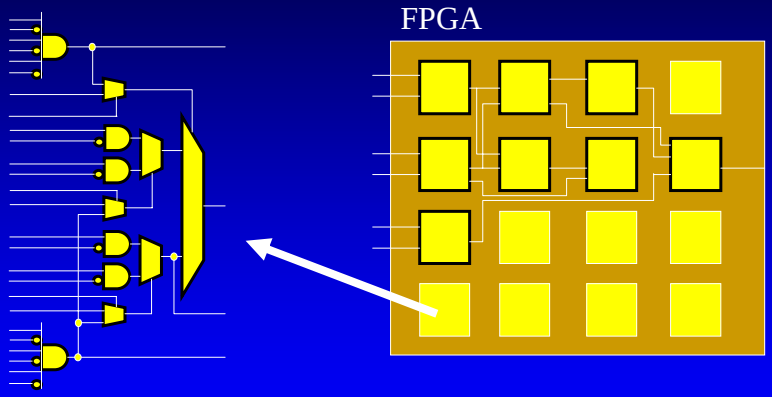
Coarse Grain:
QuickLogic pASIC Logic Cell

pASIC Logic Cell



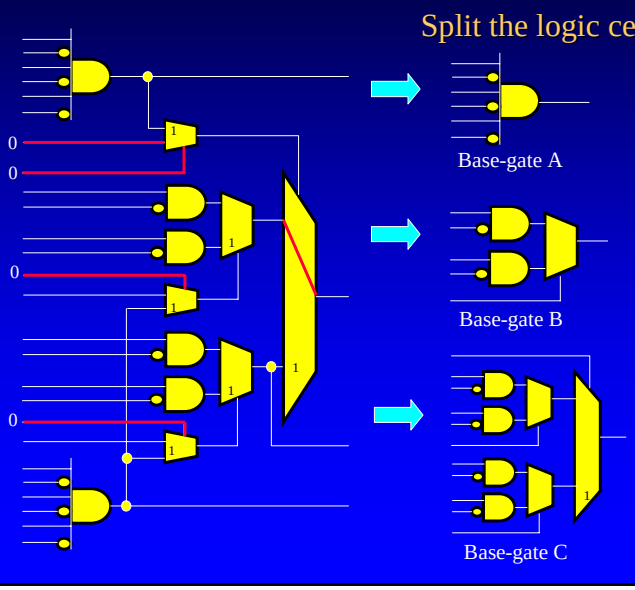
- Large fanin
  - 26 inputs, 4 outputs
- Components
  - Two 6-input AND gates
  - Four 2-input AND gates
  - Six 2:1 MUXs
- High logic capacity
  - More than 5000 library cells could be generated
- A single level of logic delay can realize many complex user functions
- Up to four logic functions can be realized with the the same logic cell

Problem Formulation

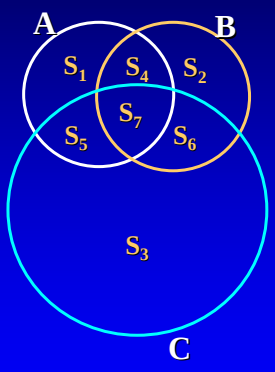


- Minimize the number of required pASIC logic cells needed to implement a given target circuit

Cell Library Construction



Cell Type Determination



- Note that the same primitive cell may be realized by more than one base gate
- A total of 205 unique primitive cells have been generated and inserted into a cell library
- A total of seven unique cell types have been defined as shown in the Venn's diagram

Cost Assignment for Primitive Cells

$$cell\_cost = \frac{s}{f \cdot c}$$

- Space usage, s : the amount of space in a pASIC cell that is used up by the primitive cell
- Freedom, f : the total number of places in a pASIC cell that the primitive cell can fit
- Coverage, c: complexity of the logic that the primitive cell can realize
- Notice that the inverter primitive cell does not have the minimum cost any more

Two-step Optimization Flow

- Perform minimum-area technology mapping by using the generated cell library
  - Use standard technology mapping algorithms (e.g., the SIS mapper)
- Pack primitive cells into pASIC logic cells in order to have the minimum number of logic cells
  - Formulate and solve as a multi-dimensional coin change problem

The Coin-Change Problem

- Problem statement
  - Let $c_1, c_2, \ldots, c_m$ be the coin types of a currency. Let C_i denote the value of coin c_i in cents and K be some integer. We assume $C_1 = 1$. The problem is to produce K cents of change by using a minimum number of coins

$$Minimize \quad \sum_{i=1}^{m} n_i \quad s.t. \quad K = \sum_{i=1}^{m} n_i C_i$$

  where n_i denotes the number of coins of type i
- Solution

$$count[K] = \begin{cases} 0 & if \ K = 0 \\ \min_{i:C_i \le K} \{count[K - C_i] + 1\} & if \ K > 0 \end{cases}$$

Different Ways of Packing Cells

	Combinations of primitive cells
1	$2S_5 + 2S_7$
2	$2S_4 + 2S_5$
3	$2S_5 + 2S_6$
…	…
i	$\sum_{j=1}^{7} C_{i,s_j} S_j$
…	…
35	$S_3 + 2S_7$
36	$S_3 + 2S_4$
37	$S_3 + S_4 + S_7$

- 37 different cases of completely utilizing a logic cell
- $C_{i,Sj}$: the number of primitive cells of type S_j in the i_{th} combination
- The packer must find optimal packing combinations in a bottom-up manner

Exact Problem Statement

- Given the different ways of packing a pASIC3 logic cell as described in the previous table and a logic netlist generated by the min-cost technology mapper, find the minimum number of pASIC3 logic cells needed to cover all primitive cells in the logic netlist, i.e.:

$$\textit{Minimize} \quad \sum_{i=1}^{37} n_i \quad \textit{s.t.} \quad \forall j : \sum_{i=1}^{37} n_i C_{i,S_j} \geq \left|S_j\right|$$

where n_i denotes the number of packings of type i and $|S_j|$ denotes the number of primitive cells of type j in the initial logic netlist

Analogy to the Coin-Change Problem

- After technology mapping, we count the number of primitive cells of each type, $|S_1|$,…, $|S_7|$. These are analogous to seven different target change counts
- The 37 different entries in the pASIC packing table are analogous to different currencies
- This is a multi-dimensional coin change problem, which can be solved optimally and efficiently by using a dynamic programming technique

Dynamic Programming Solution

$$count\left(|S_1|,...,|S_7|\right)=\begin{cases}0 & if\ \forall j,|S_j|\le 0\\ \min\limits_{i:\forall j,C_{i,S_j}\le|S_j|}\left(count\left(|S_1|-C_{i,S_1},...,|S_7|-C_{i,S_7}\right)+1\right) & otherwise\end{cases}$$

- Note that we must track remaining unpacked primitive cells for all seven cell types at the same time
- Computational complexity $O\left(\prod_{i=1}^{7}|S_i|\right)$

Experimental Results

Circuits	Primitive cell count	Greedy packing			Dynamic programming based packing			Packing improvement	
		Number of logic cells	Cell utilization (%)	CPU time (sec)	Number of logic cells	Cell utilization (%)	CPU time (sec)	Number of logic cells (%)	Cell utilization (%)
Alu2	193	76	65.87	0.08	51	100	50.27	32.9	34.1
Alu4	377	150	65.11	0.34	101	98.43	960.59	32.7	33.9
Apex6	349	154	59.86	0.37	117	80.23	306.24	24	25.4
Dalu	471	194	63.39	0.56	138	90.75	143.81	28.9	30.1
C1355	210	83	64.81	0.11	58	93.75	1.37	30.1	30.9
C1908	213	96	58.84	0.13	74	77.74	20.54	22.9	24.3
C432	108	45	65.85	0.03	31	100	13.36	31.1	34.2
C499	210	83	64.81	0.11	58	93.75	1.37	30.1	30.9
C3540*	657	268	64.22	1.2	191	91.89	56.21	28.7	30.1
C880	214	92	62.76	0.12	64	93.45	45.95	30.4	32.8
C5315*	764	333	59.97	1.94	256	79.09	42.97	23.1	24.2
C6288*	1457	664	55.19	13.11	593	61.84	19.14	10.7	10.8
C7552*	1052	413	64.94	3.7	312	86.51	86.06	24.5	24.9
Average								26.9	28.2

*The packing algorithm used segmented lists so as not to exceed the amount of available memory in the computer

Conclusions

- Proposed a minimum-area packing algorithm for coarse-grained, anti-fuse based FPGAs, comprising of library generation, technology mapping and cell packing.
- Solution of a multi-dimensional coin-change problem resulted in a polynomial time optimal solution to the cell packing problem
- Our algorithm resulted in an average of 27% fewer logic cells compared to a greedy algorithm