On Lookup Table Cascade-Based Realizations of Arbiters

Petr Mikušek, Václav Dvořák

Brno University of Technology
Faculty of Information Technology

{imikusek, dvorak}@fit.vutbr.cz

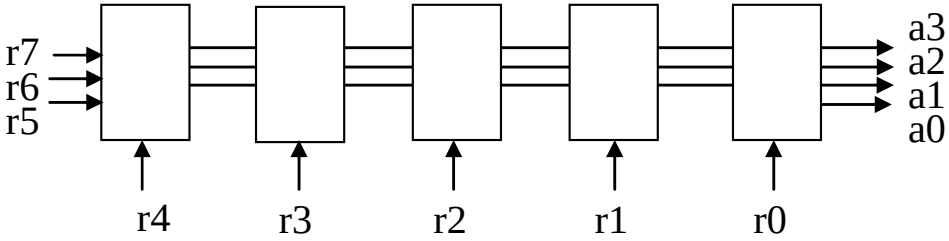
MEMICS 2008, November 14—16, Znojmo

Outline

- LUT cascade
- Iterative heuristic decomposition
- Arbiter circuits
- Heuristic Iterative Decomposition Tool
- Experimental results
- Conclusions

LUT cascade

- Regular logic
  - short development time
  - better utilization of chip area
  - easy testability



- Realization of every multiple-output Boolean function by LUT cascade was proved possible long time ago
- Direct synthesis of LUT cascades from representation in a form of Multi-Terminal Binary Decision Diagrams (MTBDD)
  - how to order variables in the diagram?
  - orderings of variables influences size and shape of the diagram

Iterative heuristic decomposition

- New technique of the iterative decomposition of integer-valued functions with an embedded heuristics to order variables
- Produces a LUT cascade and simultaneously a sub-optimal MTBDD
- Bottom-up synthesis
  - does not require knowledge of optimum variable ordering
  - order of variables is generated concurrently
- Obtained LUT cascades can be used for implementation of combinational and sequential functions in hardware, firmware, software

Example of the synthesis technique

r3	r2	r1	r0	z
0	0	0	0	4
0	0	0	1	0
0	0	1	x	1
0	1	x	x	2
1	x	x	x	3

LUT4

0:=	(2,2)
1:=	(3,3)
2:=	(0,1)
3:=	(4,1)

0	0	-	0	3	(4,1)
0	0	-	1	2	(0,1)
0	1	-	x	0	(2,2)
1	x	-	x	1	(3,3)

LUT3

0:=	(3,1)
1:=	(2,1)
2:=	(0,1)

-	0	-	0	0	(3,1)
-	0	-	1	1	(2,1)
-	1	-	x	2	(0,1)

LUT2

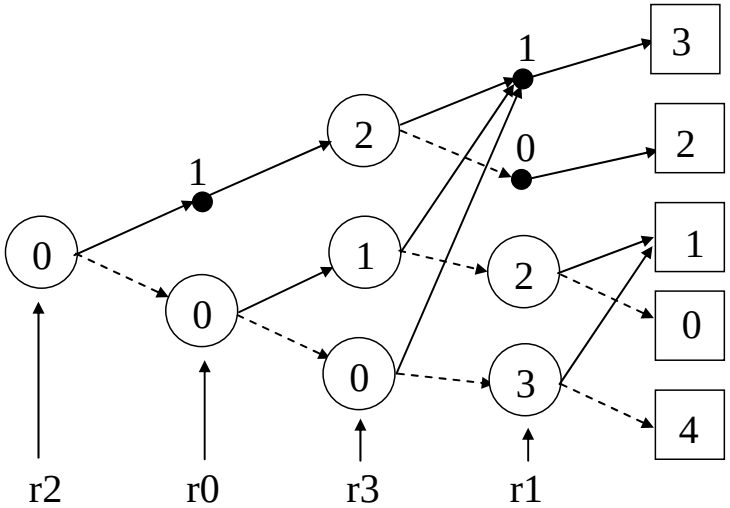
0:=	(0,1)
1:=	(2,2)

-	0	-	-	0	(0,1)
-	1	-	-	1	(2,2)

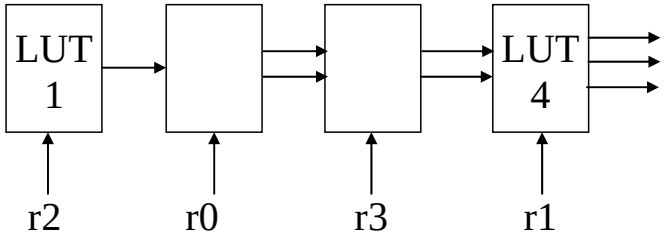
LUT1

0:=	(0,1)

-	-	-	-	0	(0,1)



a)



b)

Heuristic to select variables

- Heuristic to locally minimize the LUT cascade/MTBDD width
- At each decomposition step:
  - select variable with minimum number of decision nodes (including degenerated ones)
  - tie: select variable with the minimum number of true (non-degenerated) decision nodes
  - tie: select variable randomly

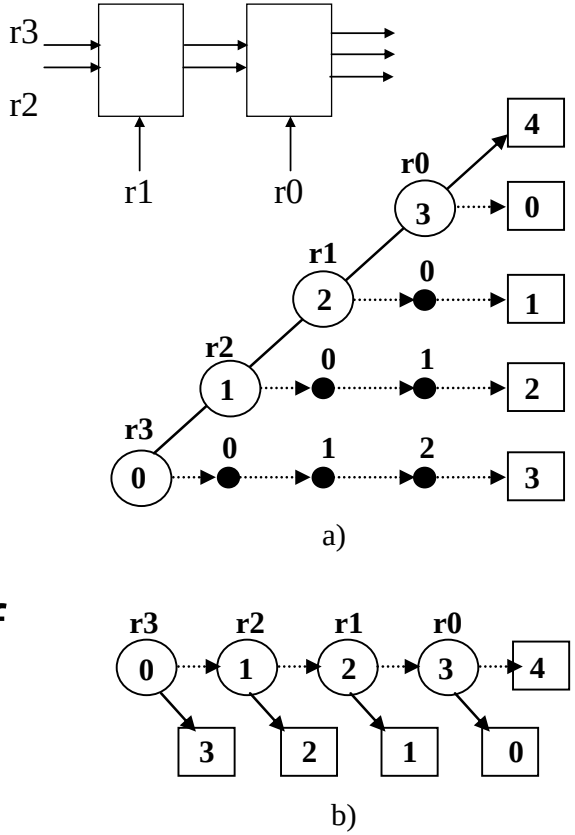
Arbiter circuits

Three representative types of arbiters:

- Fixed priority arbiter
  - Priority Encoder (PE)
- Round robin arbiter
  - Last Granted Lowest Priority scheme (LGLP)
- Matrix arbiter
  - Least Recently Served scheme (LRS)

Priority Encoder (PE)

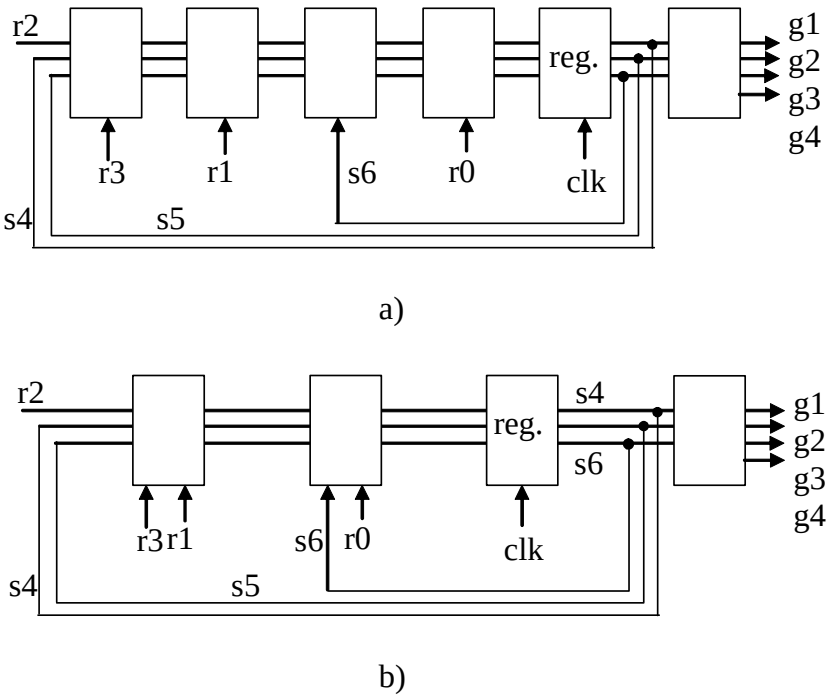
- Combinational circuit
- n request inputs, $\lceil \log_2 n \rceil + 1$ outputs
- Output is the address of the request input with the highest priority
- Input request r(n-1) has the highest priority, r0 has the lowest priority
- One address bit denotes the case of no active request



PE4 arbiter

Round robin (LGLP) arbiter

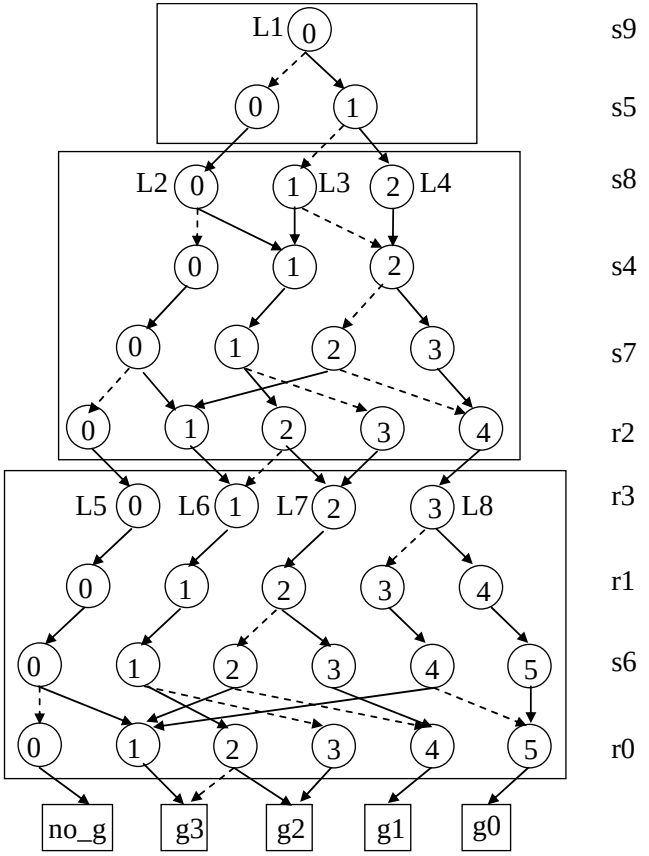
- Sequential circuit
- *n* input requests, 2*n* states S0, S1, …, S(2*n*-1), *n* grant outputs
- Even-numbered states monitors request inputs
- Odd-numbered states generates grant outputs (one grant per state)
- Priority vector is modified by cyclic shifts



LUT cascade of LGLP4 arbiter
a) 4-input LUTs b) 5-input LUTs

Matrix arbiter with LRS strategy

- Sequential circuit
- Priority matrix P
- p_{ik} = 1 means the i-th request has priority over the k-th request
- Asserted grant output g(j) resets the j-th row to all zeros and sets the j-th column to all one
- Matrix is antisymetric – store only elements above diagonal
- $(n^2 – n)/2$ state variables



MTBDD of LRS4 arbiter

Heuristic Iterative Decomposition Tool

- We developed HIDET tool (C++)
- Input in Espresso PLA format
- Loop of determining best variable can be easily parallelized (OpenMP)
- Limitation: HIDET can process only functions specified by function tables with disjunctive sub-domains (rows)
  - we can't test our program on a standard benchmark set (most of the benchmark circuits are specified by overlapping sub-domains)

```
Initialize i ← 1, M_0 ← M_in;
for i in 1 to n do
    // Determine the best variable
    v_best ← arbitrary variable from S_v,
    w_best ← size(M_i-1), d_best ← 0;
    for all variables v ∈ S_v do
        M_p ← make_pairs(M_i-1, v);
        S_p ← unique_pairs(M_p);
        S_m ← merge_compatible_pairs(S_p);
        w ← size(S_m);
        d ← number of constant pairs in S_m;
        if (w < w_best) or (w == w_best and d > d_best)
            then
        v_best ← v, w_best ← w, d_best ← d;
        endif
    endfor
    v_i ← v_best;

    // Decompose
    M_p ← make_pairs(M_i-1, v_i);
    S_p ← unique_pairs(M_p);
    S_m ← merge_compatible_pairs(S_p);
    L_i ← enumerate_pairs(S_m);
    M_i ← replace pairs in M_p by new id numbers in L_i;
    S_v ← S_v \ {v_i};
endfor
```

Experimental results (1)

- Function tables of many instances of three types of arbiters has been generated
  - PE: PE4, PE8, PE12, PE16
  - LGLP: LGLP3, LGLP4, LGLP6, LGLP8, LGLP10, LGLP12, LGLP16
  - LRS: LRS3, LRS4, LRS6
- LUT cascades synthesized by our HIDET tool
- Same circuits synthesized by Xilinx FPGA tool

Experimental results (2)

Name	In	Out	Term	One LUT	LUT cascade					FPGA					
				memory [bits]	nodes	max. width	memory [bits]	memory [%]	time [s]	LUT	levels	delay [ns]	memory [bits]	memory [%]	time [s]
PE4	4	3	5	48	10	2	50	104.17%	0.01	4	1	0.465	64	133.33%	4.31
PE8	8	4	9	1024	36	3	258	25.20%	0.01	8	2	1.302	128	12.50%	4.29
PE12	12	5	13	20480	78	4	802	3.92%	0.02	13	4	2.258	208	1.02%	4.43
PE16	16	5	17	327680	136	5	1314	0.40%	0.04	19	5	3.813	304	0.09%	4.50
LGLP3	6	3	18	192	28	4	242	126.04%	0.01	12	3	2.121	192	100.00%	4.53
LGLP4	7	3	28	384	43	4	370	96.35%	0.01	15	4	1.922	240	62.50%	4.51
LGLP6	10	4	54	4096	79	5	834	20.36%	0.06	45	6	4.413	720	17.58%	5.32
LGLP8	12	4	88	16384	99	5	930	5.68%	0.26	72	7	4.428	1152	7.03%	6.09
LGLP10	15	5	130	163840	152	5	1762	1.08%	11.15	122	7	4.989	1952	1.19%	8.70
LGLP12	17	5	180	655360	188	6	2402	0.37%	26.61	186	12	8.072	2976	0.45%	18.58
LGLP16	21	5	304	10485760	289	6	3570	0.03%	59.10	420	12	9.338	6720	0.06%	57.24
LRS3	6	3	13	192	18	3	114	59.38%	0.01	6	2	0.796	96	50.00%	4.30
LRS4	10	4	33	4096	42	3	322	7.86%	0.02	8	2	1.302	128	3.13%	4.38
LRS6	21	6	193	12582912	122	4	952	0.01%	1.73	24	2	1.302	384	0.00%	5.02
LRS8	36	8	1025	549755813888	389	4	4004	0.00%	90.95	-	-	-	-	-	-

Conclusions

- New algorithm for synthesis of LUT cascades and sub-optimal MTBDD has been presented
- Our synthesis method proved to be suitable for synthesis combinational and sequential designs with tens of variables
- Future research should address a more general specification of functions with overlapping sub-domains and/or with incompletely specified output vectors

Thank you for your attention