Reversible Circuit Synthesis Using a Cycle-Based Approach

13

MEHDI SAEEDI, MORTEZA SAHEB ZAMANI,
MEHDI SEDIGHI, and ZAHRA SASANIAN
Amirkabir University of Technology

Reversible logic has applications in various research areas, including signal processing, cryptography and quantum computation. In this article, direct NCT-based synthesis of a given k-cycle in a cycle-based synthesis scenario is examined. To this end, a set of seven building blocks is proposed that reveals the potential of direct synthesis of a given permutation to reduce both quantum cost and average runtime. To synthesize a given large cycle, we propose a decomposition algorithm to extract the suggested building blocks from the input specification. Then, a synthesis method is introduced that uses the building blocks and the decomposition algorithm. Finally, a hybrid synthesis framework is suggested that uses the proposed cycle-based synthesis method in conjunction with one of the recent NCT-based synthesis approaches which is based on Reed-Muller (RM) spectra.

The time complexity and the effectiveness of the proposed synthesis approach are analyzed in detail. Our analyses show that the proposed hybrid framework leads to a better quantum cost in the worst-case scenario compared to the previously presented methods. The proposed framework always converges and typically synthesizes a given specification very fast compared to the available synthesis algorithms. Besides, the quantum costs of benchmark functions are improved about 20% on average (55% in the best case).

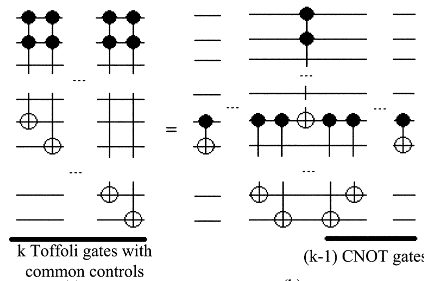
ACM Reference Format:

Saeedi, M., Saheb Zamani, M., Sedighi, M., and Sasanian, Z. 2010. Reversible circuit synthesis using a cycle-baed approach. ACM J. Emerg. Technol. Comput. Syst. 6, 4, Article 13 (December 2010), 26 pages. DOI = 10.1145/1877745.1877747 http://doi.acm.org/10.1145/1877745.1877747

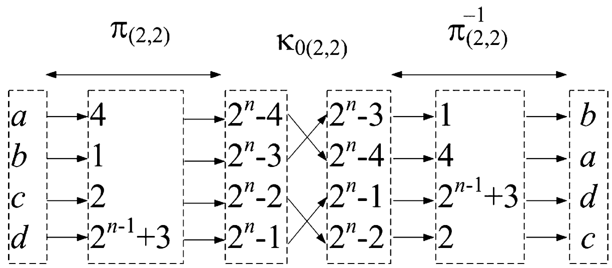
A preliminary version of this article was presented at ASPDAC [Sasanian et al. 2009].

Authors' address: M. Saeedi (corresponding author), M. Saheb Zamani, M. Sedighi, and Z. Sasanian, Quantum Design Automation Lab, Computer Engineering Department, Amirkabir University of Technology, Tehran, Iran; email: msaeedi@aut.ac.ir.

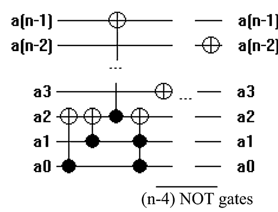
DOI 10.1145/1877745.1877747 http://doi.acm.org/10.1145/1877745.1877747



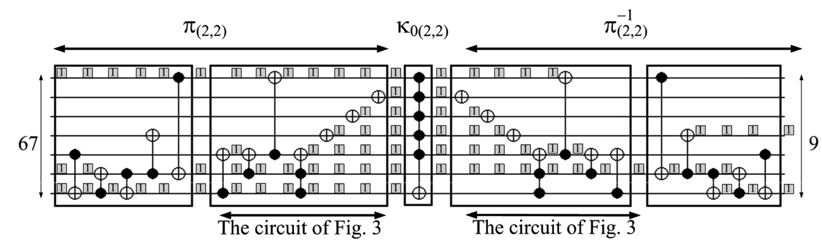
(a)
(b)

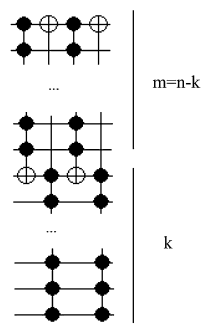


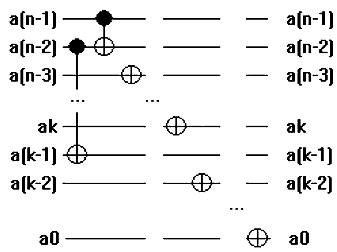
$\pi_{(2,2)}$
$\kappa_{0(2,2)}$
$\pi^{-1}_{(2,2)}$
a b c d
4 1 2 $2^{n-1}+3$
2^n-4 2^n-3 2^n-2 2^n-1
2^n-3 2^n-4 2^n-1 2^n-2
1 4 $2^{n-1}+3$ 2
b a d c


a(n-1)
a(n-2)
...
a3
a2
a1
a0
...
a(n-1)
a(n-2)
a3
a2
a1
a0
(n-4) NOT gates

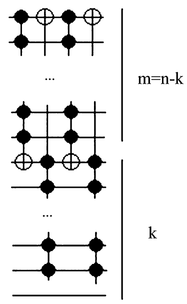
)

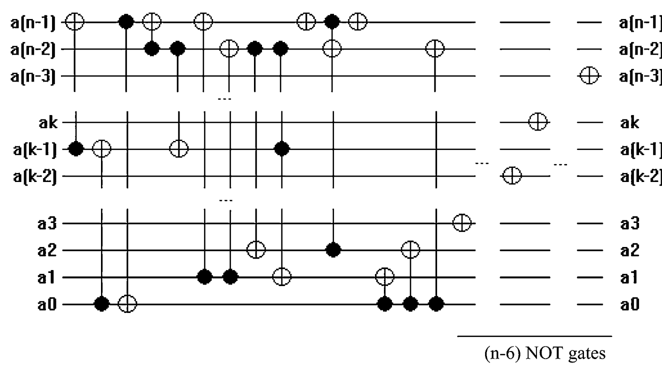

$\pi_{(2,2)}$
$\kappa_{0(2,2)}$
$\pi_{(2,2)}^{-1}$
67
9
The circuit of Fig. 3
The circuit of Fig. 3

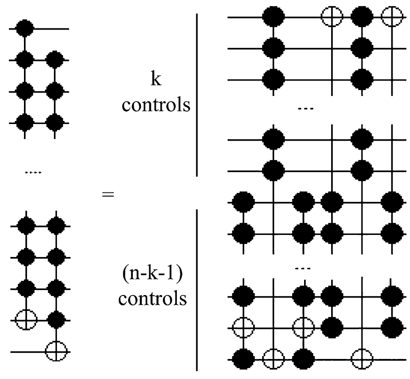

...
m=n-k
...
k



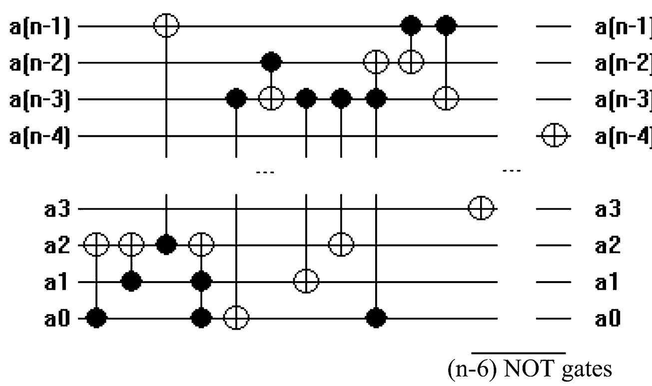
(n-3) NOT gates

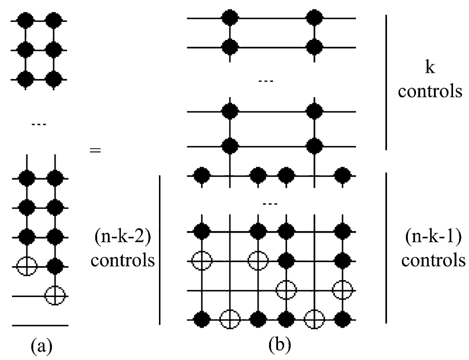



a(n-1)
a(n-2)
a(n-3)
ak
a(k-1)
a(k-2)
a3
a2
a1
a0
(n-6) NOT gates

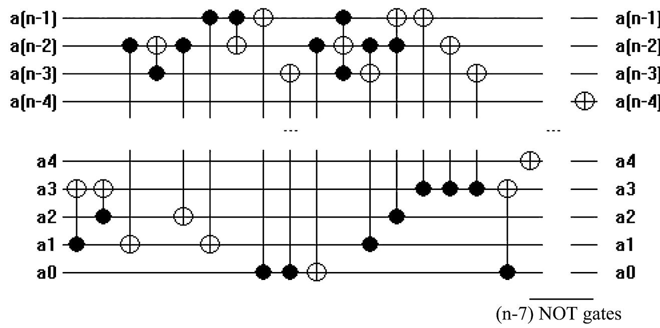

k
controls
(n-k-1)
controls

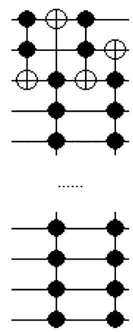
(a) (b)

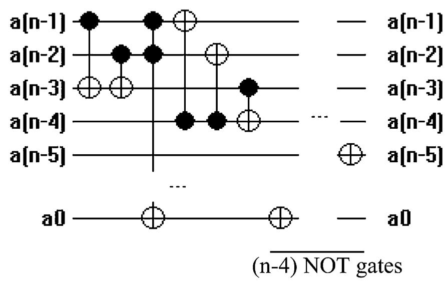

a(n-1)
a(n-2)
a(n-3)
a(n-4)
a3
a2
a1
a0
(n-6) NOT gates

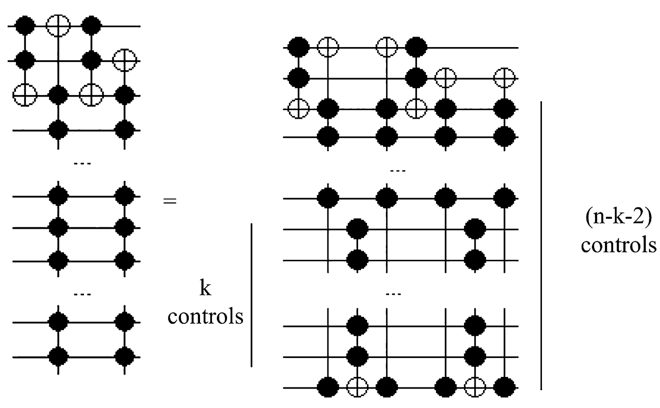


...
=
(n-k-2)
controls
(a)
...
k
controls
...
(n-k-1)
controls
(b)


a(n-1)
a(n-2)
a(n-3)
a(n-4)
...
a4
a3
a2
a1
a0
...
a(n-1)
a(n-2)
a(n-3)
a(n-4)
a4
a3
a2
a1
a0
(n-7) NOT gates

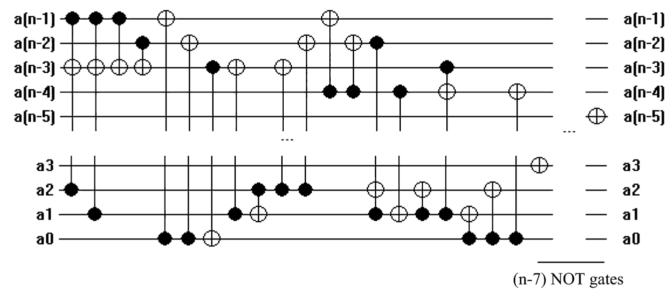

......


a(n-1)
a(n-2)
a(n-3)
a(n-4)
a(n-5)
...
a0
...
a(n-1)
a(n-2)
a(n-3)
a(n-4)
a(n-5)
a0
(n-4) NOT gates


...
...
=
k
controls
...
...
(n-k-2)
controls

(a)

(b)


a(n-1)
a(n-2)
a(n-3)
a(n-4)
a(n-5)
a3
a2
a1
a0
a(n-1)
a(n-2)
a(n-3)
a(n-4)
a(n-5)
a3
a2
a1
a0
(n-7) NOT gates

Step1
Fix 0 and 2^i terms use a pre-process stage as done in [Shende et al. 2003].

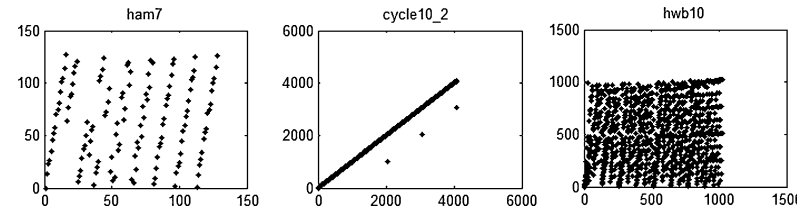
Step2
if $n < 7$

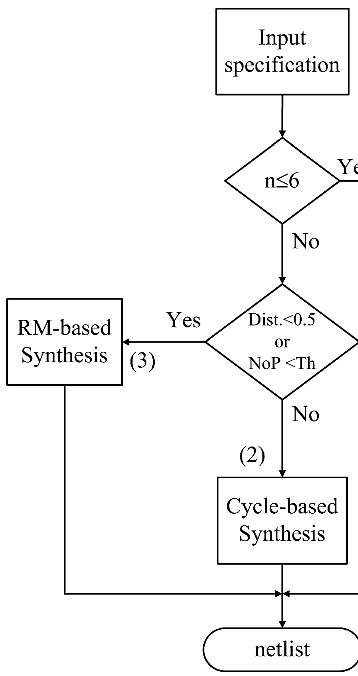
1- Decompose the input permutation into a set of 2-cycles.
2- Apply $Syn_{2,2}$ to synthesize all 2-cycles

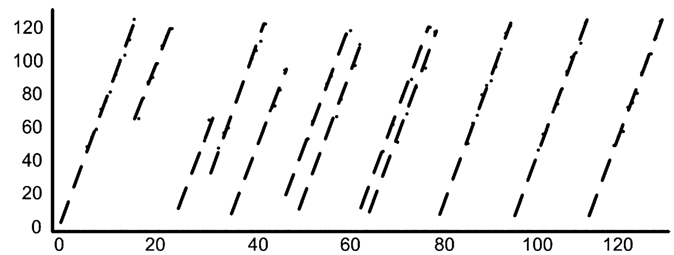
else

1- Decompose the input permutation into a set of 5, 4, 3, and 2 cycles
2- Synthesize all disjoint 5-cycle pairs ($Syn_{5,5}$)
3- Synthesize single 5-cycles (Syn_5)
4- Synthesize all disjoint 3-cycle pairs ($Syn_{3,3}$)
5- Synthesize single 3-cycles (Syn_3)
6- Synthesize all disjoint 4-cycle pairs ($Syn_{4,4}$)
7- Synthesize all disjoint 4-cycle and 2-cycle pairs ($Syn_{4,2}$)
8- Synthesize all disjoint 2-cycle pairs ($Syn_{2,2}$)

)


ham7
cycle10_2
hwb10
150
100
50
0
6000
4000
2000
1500
1000
500


Input specification
n≤6
Yes
(1)
Cycle-based Synthesis
No
Random Driver
Dist.<0.5
or
NoP <Th
Yes
RM-based Synthesis
(3)
No
(2)
Cycle-based Synthesis
netlist



120
100
80
60
40
20
0
0
20
40
60
80
100
120