Some Thoughts of Free-AND and NAND Gates

1 Evaluate AND (OR) gates for free

1.1 General idea

Supposing we map each wire bit to a L-bit random number (aka a symbol), let generator $\mathcal{G}$ hold a secret threshold t such that a L-bit symbol s ($s \neq t$) represents 1 if $s > t$, otherwise s represents 0. Thus the generator will represent 1 with an arbitary number from $[0, t-1]$, and 0 from $[t+1, 2^L - 1]$.

As an evaluator $\mathcal{E}$, it will evaluate an AND gate by outputting $o = min(s, s')$, where s, s' are input symbols of the AND gate, respectively. Finally $\mathcal{E}$ will learn the real output (i.e., 0 or 1) by comparing o and t, where t is received from the generator $\mathcal{G}$. Similarlly we are able to evaluate OR gates for free by replacing min function with max.

1.2 Used in circuit

This design of AND/OR gate can be evaluated for free when there is no XOR/NOT gates in a ciruit. Otherwise, we still need extra costs to convert different symbol representations. As I found, to convert from XOR gates to AND gates, a generator needs to transfer 2 entries per wire; while from AND gates to XOR gates, it needs to transfer 3 entries per wire. Therefore it costs a little bit more than the state-of-art technique (aka 'half-gate'). Because half-gate technique can be combined with free-XORs and requires only 2 entries transportation for each 1-bit AND gate. But my method may perform better in a circuit with more AND/OR gates, and perform equally in a circuit which only need conversion from XOR to AND gates. For example, in a L-bit equals circuit (which is composed of L 1-bit XOR gates and $L-1$ 1-bit AND gates), we are still able to evaluate XOR and AND gates for free seperately. But from outputs of XOR to inputs of AND, the generator needs to transfer $2(L-1)$ entries to the evaluator.

1.3 Concerns

I still have some concerns about such method, especially the security aspect. I wonder I need more time to think it deeply.

2 Extended to NAND gates

I have another idea to extend such design to NAND gates. After extending each symbol to $2L$ bits, we denote each input as $r\|\alpha$, where r is the original symbol representing 1 or 0, and α is *offset* to achieve NOT function: we set α as $r_0 \oplus \alpha = r_1$ and $r_1 \oplus \alpha = r_0$.

Therefore we are able to implement a NAND gate now:

$$o = min(r, r') \oplus \tilde{\alpha}\|\tilde{\alpha}$$

, where $\tilde{\alpha} = (min(r, r') == r)?\alpha : \alpha'$.