Control gates as building blocks for reversible computers

A. De Vos[1], B. Desoete[2], F. Janiak[3], and A. Nogawski[3]

[1] Universiteit Gent and Imec v.z.w., B-9000 Gent, Belgium
[2] Universiteit Gent, B-9000 Gent, Belgium
[3] Politechnika Łódzka, PL-90-924 Łódź, Poland

Abstract. In principle, any reversible logic circuit can be built by using a single building block (having three logic inputs and three logic outputs). We demonstrate that, for a flexible design, it is more advantageous to use a broad class of reversible gates, called control gates. They form a generalization of Feynman's three gates (i.e. the `NOT`, the `CONTROLLED NOT`, and the `CONTROLLED CONTROLLED NOT`). As an illustration, two reversible 4-bit carry-look-ahead adders in 0.8 μm c-MOS have been built.

1 Introduction

Classical computing machines using logically irreversible gates unavoidably generate heat. This is due to the fact that each loss of one bit of information is accompanied by an increase of the environment's entropy by an amount $k \log(2)$, where k is Boltzmann's constant. This means that an amount of thermal energy equal to $kT \log(2)$ is transferred to the environment (at temperature T). According to Landauer's principle [1] [2] [3], it is possible to construct a computer that dissipates an arbitrarily small amount of heat. A necessary condition is that no information is thrown away. Therefore, logical reversibility is a necessary (although not sufficient) condition for physical reversibility.

Fredkin and Toffoli [4] [5] have shown that a logically reversible basic building block should have three binary inputs (say A, B, and C) and three binary outputs (say P, Q, and R). An arbitrary boolean function can be implemented using exclusively such gate. Storme et al. [6] have shown that not less than 38,976 different logic gates (all with three inputs and three outputs) are candidates to play the role of universal reversible building block. Instead of working with a single block, one can equally well use a set of building blocks. Feynman [7] [8] has proposed the use of three fundamental gates (See Table 1):

– the `NOT` gate,
– the `CONTROLLED NOT` gate, and
– the `CONTROLLED CONTROLLED NOT` gate.

In the present paper, we develop a design strategy that uses even more than three building units. We call the new reversible units control gates.

Table 1. Feynman's three basic truth tables: (a) `NOT`, (b) `CONTROLLED NOT`, (c) `CONTROLLED CONTROLLED NOT`

A	P
0	1
1	0

(a)

AB	PQ
0 0	0 0
0 1	0 1
1 0	1 1
1 1	1 0

(b)

ABC	PQR
0 0 0	0 0 0
0 0 1	0 0 1
0 1 0	0 1 0
0 1 1	0 1 1
1 0 0	1 0 0
1 0 1	1 0 1
1 1 0	1 1 1
1 1 1	1 1 0

(c)

2 Simple control gates

2.1 Definition

A gate with k inputs $(A_1, A_2, ..., A_k)$ and k outputs $(P_1, P_2, ..., P_k)$, satisfying

$$\begin{aligned} P_i &= A_i \qquad \text{for all } i \in \{1, 2, ..., k-1\} \\ P_k &= f(A_1, A_2, ..., A_{k-1}) \texttt{ XOR } A_k \ , \end{aligned}$$

with f an arbitrary boolean function of $k-1$ boolean arguments, is called a simple control gate. The number k is called the width of the gate. The logic inputs A_1, A_2, ..., A_{k-1} are named the controlling bits, whereas the input A_k is the controlled bit. Finally, the function f is called the control function.

2.2 Properties

We first demonstrate that any simple control gate is reversible. For this purpose, we cascade two identical simple control gates, yielding

$$P_k = [\ f(A_1, A_2, ..., A_{k-1}) \texttt{ XOR } f(A_1, A_2, ..., A_{k-1})\ ] \texttt{ XOR } A_k \ ,$$

and thus $P_i = A_i$ for all i, because of the two boolean identities X `XOR` $X = 0$ and 0 `XOR` $Y = Y$. The result is thus the k-bit follower. In other words: any simple control gate is its own inverse, and thus is necessarily reversible.

Cascading two arbitrary simple control gates (one with control function f' and one with control function f'') results in a new simple control gate, with control function f' `XOR` f''. Therefore the simple control gates of width k together with the operation cascading form a group. The group has $2^{2^{k-1}}$ elements. It is abelian, because of the boolean identity X `XOR` $Y = Y$ `XOR` X. We note that

- the `NOT` gate is a simple control gate with $k = 1$ and f a function with zero arguments: $f(.) = 1$;
- the `CONTROLLED NOT` is a simple control gate with $k = 2$ and $f(A_1) = A_1$;
- the `CONTROLLED CONTROLLED NOT` is a simple control gate with $k = 3$ and $f(A_1, A_2) = A_1$ `AND` A_2.

2.3 Implementation

The function

$$P_k = f(A_1, A_2, ..., A_{k-1}) \texttt{ XOR } A_k$$

is equivalent with

$$\begin{aligned} P_k = \ & \texttt{NOT } A_k \text{ if } f(A_1, A_2, ..., A_{k-1}) = 1 \\ & A_k \quad \text{ if } \texttt{NOT } f(A_1, A_2, ..., A_{k-1}) = 1 \end{aligned}$$

and thus can be built into a square geometry, provided we use dual line electronics, i.e. any signal X is accompanied by its counterpart `NOT` X. Fig. 1 shows P_k is connected to A_k if $f = 0$ but is connected to $\overline{A_k}$ (short notation for `NOT` A_k) if $f = 1$. Because a boolean function $f(A_1, A_2, ..., A_{k-1})$ can always be written

- either as an '`OR` of `AND`s' (often referred to as 'sum of minterms')
- or as an '`AND` of `OR`s' (often referred to as 'product of maxterms'),

we can implement $P_k = f(A_1, A_2, ..., A_{k-1})$ `XOR` A_k in the square, using

- either four parallel connections of series connections of switches
- or four series connections of parallel connections of switches
- or a combination of both.

Each switch is composed of one n-MOS transistor in parallel with one p-MOS transistor (forming together a transmission gate). This leads to a reversible electronic implementation in dual-line pass-transistor logic: so-called r-MOS technology [9]. Such logic is naturally suited for adiabatic addressing [10] [11] [12]. All energy supplied to the outputs P_k and $\overline{P_k}$ comes from the inputs A_k and $\overline{A_k}$, i.e. not from separate power lines.

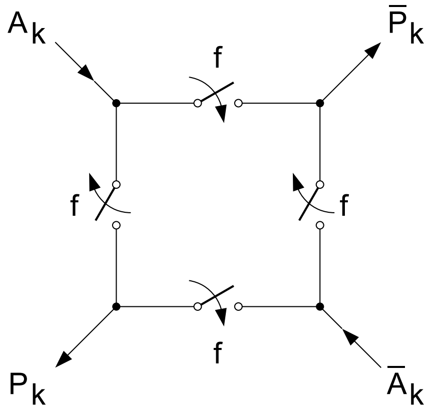


Fig. 1. Implementation of the function f `XOR` A_k, with the help of four switches

Table 2. Truth table: (a) original (irreversible) table, (b) reversible version

A	B	C	S
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

(a)

A	B	C	D	P	Q	R	S
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	1
0	1	1	0	0	1	1	1
0	1	1	1	0	1	1	0
1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	1
1	0	1	0	1	0	1	0
1	0	1	1	1	0	1	1
1	1	0	0	1	1	0	1
1	1	0	1	1	1	0	0
1	1	1	0	1	1	1	1
1	1	1	1	1	1	1	0

(b)

As an example, suppose we have to implement truth Table 2a: a function $S(A,B,C)$ of three arguments. The appropriate control gate has a total of $k=4$ inputs, $k-1=3$ of which are control bits (A,B,C) controlling the k th $=4$ th input bit D. We extend Table 2a into Table 2b, where we have $P=A$, $Q=B$, $R=C$, and where the fourth output bit S satisfies

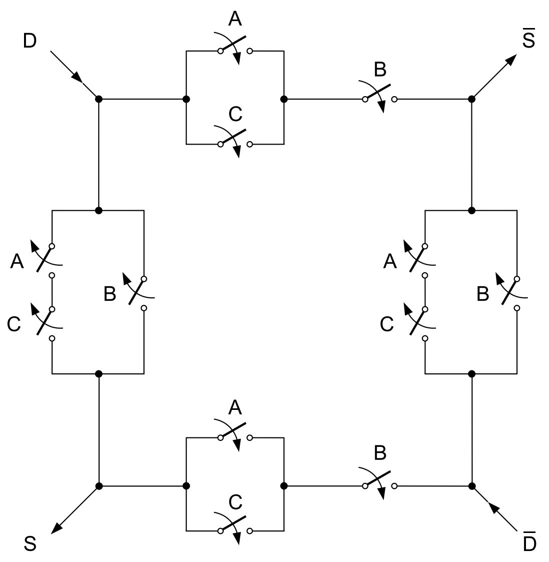
$$S = ((\overline{A} \texttt{ AND } B \texttt{ AND } C) \texttt{ OR } (A \texttt{ AND } B \texttt{ AND } \overline{C}) \texttt{ OR } (A \texttt{ AND } B \texttt{ AND } C)) \texttt{ XOR } D \ ,$$

indeed realizing (after presetting $D=0$) truth Table 2a. Fig. 2a displays an optimized implementation of the example, making use of

$$S = ((A \texttt{ OR } C) \texttt{ AND } B) \texttt{ XOR } D \ ,$$

needing only 12 switches.

An alternative approach makes use of standard cells, where the particular function $f(A,B,C)$, to be `XOR`ed with D, is hardwired by the vias between the `Metal1` and `Metal2` layers of the chip. These vias are displayed as small black squares in Fig. 2b. The programmable gate however needs $2^{k+1}-4=28$ switches.



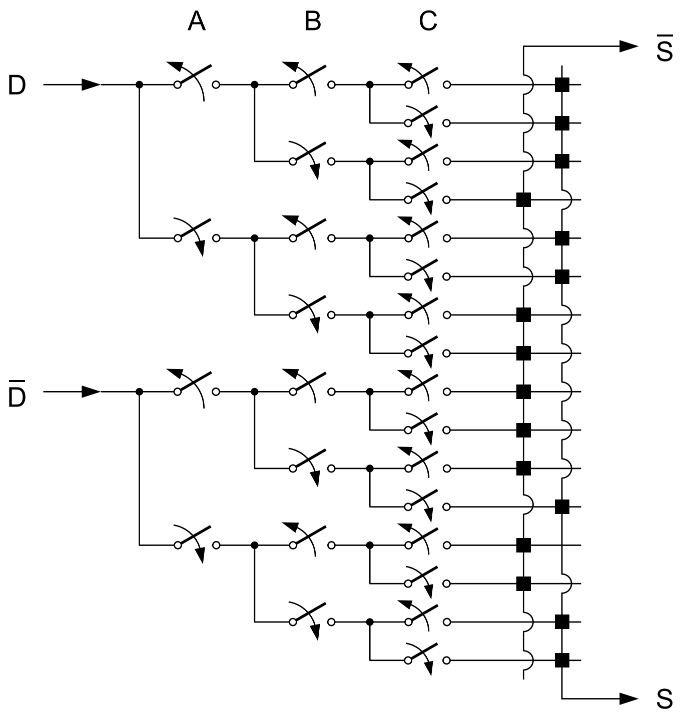


Fig. 2. Implementation of boolean Table 2 using (a) 12 switches, (b) 28 switches

3 Control gates

3.1 Definition

When we cascade k simple control gates (one of width k, one of width $k-1$, ..., and one of width 1), in the way of Fig. 3, we have a new gate of width k. Because each output is only one boolean function f away from the inputs, its logic depth is only 1. We call such gates control gates, as each output P_i is either equal to the controlled bit A_i or to its inverse $\overline{A_i}$, depending on the value of its $i-1$ controlling inputs A_1, A_2, ..., A_{i-1}.

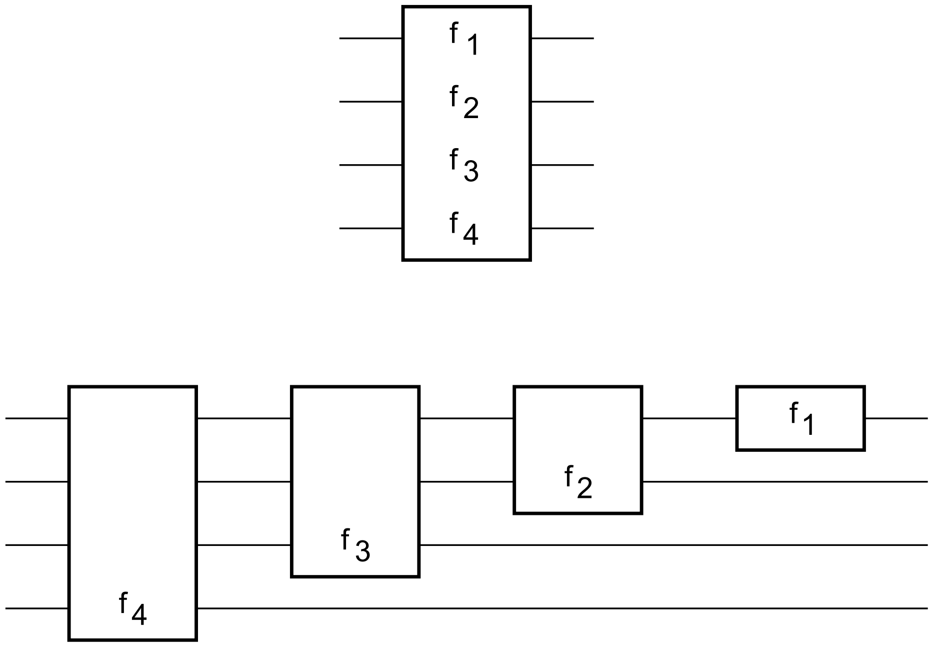


Fig. 3. Decomposition of a control gate into simple control gates

We thus come to the definition of a control gate: a logic gate with k inputs $(A_1, A_2, ..., A_k)$ and k outputs $(P_1, P_2, ..., P_k)$, satisfying

$$P_i = f_i(A_1, A_2, ..., A_{i-1}) \texttt{ XOR } A_i \text{ for all } i \in \{1, 2, ..., k\},$$

with f_i arbitrary boolean functions of $(i-1)$ arguments, is called a control gate.

Note that a control gate with width k has $(k-1)$ controlling bits (A_1, A_2, ..., A_{k-1}) as well as $(k-1)$ controlled bits (A_2, A_3, ..., A_k).

We remark that the above definition is somewhat more general than the preliminary definition presented at *Patmos 2000* [12].

3.2 Properties

As any control gate is composed of simple control gates and any simple control gate is reversible, the control gate is thus also reversible. The inverse of a simple control gate is equal to itself. This is not the case with an arbitrary control gate. The inverse of the control gate of Fig. 3 consists of first putting the simple control gate f_1, then the simple control gate f_2, etc. Now the cascading of two simple control gates of different width is not commutative. Thus an arbitrary control gate and its reverse are not necessarily equal, the simple building blocks appearing in opposite order.

Two control gates (one with control functions f_i' and one with control functions f_i''), when cascaded, form a new control gate, with control functions

$$\begin{aligned} f_i(A_1, A_2, ..., A_{i-1}) \quad &= \quad f_i'(A_1, A_2, ..., A_{i-1}) \texttt{ XOR} \\ f_i''(\ f_1'(.) \texttt{ XOR } A_1,\ f_2'(A_1) &\texttt{ XOR } A_2,\ ...,\ f_{i-1}'(A_1, A_2, ..., A_{i-2}) \texttt{ XOR } A_{i-1}\ ) \ . \end{aligned}$$

Thus, the control gates of width k, together with the cascading operation, form a group. This group has 2^{2^k-1} elements and is solvable, but not abelian.

4 Carry-look-ahead adder

To demonstrate the flexibility of using control gates, we present here, as an example, a 4-bit carry-look-ahead adder, as an alternative to the classical, i.e. ripple adder. An n-bit ripple adder consists of $2n$ gates of type `CONTROLLED NOT` and $2n$ gates of the `CONTROLLED CONTROLLED NOT` type [12]. Its logic depth increases with increasing n. In order to make the calculation less deep, and thus faster, we replace the ripple adder by a carry-look-ahead adder.

For the carry-look-ahead (or c.l.a.) [13], we first need to implement the calculation of the n generator bits G_i and the n propagator bits P_i from the n addend bits A_i and the n augend bits B_i:

$$\begin{aligned} G_i &= A_i \texttt{ AND } B_i \\ P_i &= A_i \texttt{ XOR } B_i \ . \end{aligned}$$

Next we need to calculate the n carry-out bits C_i from the single carry-in bit C_0, the n generator bits, and the n propagator bits. In its simplest form, the 4-bit carry-look-ahead adder implements the following equations:

$$\begin{aligned} C_1 &= G_0 \texttt{ OR } (P_0 \texttt{ AND } C_0) \\ C_2 &= G_1 \texttt{ OR } (P_1 \texttt{ AND } (G_0 \texttt{ OR } (P_0 \texttt{ AND } C_0))) \\ C_3 &= G_2 \texttt{ OR } (P_2 \texttt{ AND } (G_1 \texttt{ OR } (P_1 \texttt{ AND } (G_0 \texttt{ OR } (P_0 \texttt{ AND } C_0))))) \\ C_4 &= G_3 \texttt{ OR } (P_3 \texttt{ AND } (G_2 \texttt{ OR } (P_2 \texttt{ AND } (G_1 \texttt{ OR } (P_1 \texttt{ AND } (G_0 \texttt{ OR } (P_0 \texttt{ AND } C_0))))))) \ . \end{aligned}$$

This can be performed by a control gate with $2n + 1$ bits controlling n other bits (i.e. $k = 3n + 1$). The electronic implementation of this gate consists of n squares, counting $8n(n + 2)$ transistors.

In the third and final step, the adder calculates the n sum bits:

$$S_i = P_i \texttt{ XOR } C_i \ .$$

5 Results

Putting the three parts (calculation of (G_i, P_i), of C_{i+1}, and of S_i) together, we see that the logic depth d of the resulting n-bit c.l.a. adder is 3, independent of n. Note that we consider the `NOT` as a gate of zero depth. Indeed, in dual line hardware, the `NOT` gate is merely an interchange of the two lines and thus costs neither silicon area, nor time delay, nor power dissipation.

Fig. 4 shows the 4-bit c.l.a. adder. For sake of clarity, the 8 preset input lines and the 12 garbage output lines are not shown, nor are the inverters (i.e. the `NOT` gates). Each logic gate has an equal number of logic inputs and logic outputs, a number we call the width w of the gate. The full circuit has depth $d = 3$, width $w = 17$, and transistor count $t = 320$. For an arbitrary n, we have $d = 3$, $w = 4n + 1$ and $t = 8n(n + 6)$. For comparison: the ripple adder has $d = n + 1$, $w = 3n + 1$ and $t = 48n$. Thus for any number $n > 2$, the c.l.a. adder is less deep (and thus faster) than its ripple counterpart. At the other side, for any number n, the c.l.a. circuit is more complex than the ripple circuit, the hardware overhead becoming quite substantial for large n.

Fig. 5 shows the 4-bit implementation in the 0.8 μm c-MOS n-well technology *CYE* of *Austria Mikro Systeme*. The n-MOS transistors have length L equal to 0.8 μm and width W equal to 2 μm. The p-MOS transistors have $L = 0.8$ μm and $W = 6$ μm. The threshold voltages are 0.85 volt (n-MOS) and $-$ 0.75 volt (p-MOS). The whole chip (bonding pads included) measures 1.9 mm $\times$ 1.2 mm. The chip has been tested successfully, with power supply voltage $V_{dd} = -V_{ss}$ ranging from 1 volt to 3 volts. Fig. 6a shows the experimental transient output C_4 for augend $B = 1101$ and addend A changing quasi-adiabatically from 0010 to 0011 with charging time $\tau = 50$ μs. Fig. 6b shows the power dissipation estimated by `Spectre` simulations (including parasitics) for $V_{dd} = -V_{ss} = 2$ V, as a function of τ.

A c.l.a. chip, applying hardware-programmed control gates, is designed. It contains as many as $t = \frac{64}{3}(4^n + 3n - 1) = 5696$ transistors and measures 2.5 mm $\times$ 2.0 mm.

In the recent literature, other 4-bit c.l.a. adders [14] [15], and even an 8-bit [16] c.l.a. adder with adiabatic/reversible gates have been presented. Our design should not at all be considered as just one more such a circuit. Our c.l.a. adders should be regarded as specific examples of the design philosophy we have developed: reversible control gate logic.

Acknowledgement

The authors thank the *Invomec* division of *Imec v.z.w.* (Leuven, Belgium) and the *Europractice* organisation, for processing the chips at *Austria Mikro Systeme.*

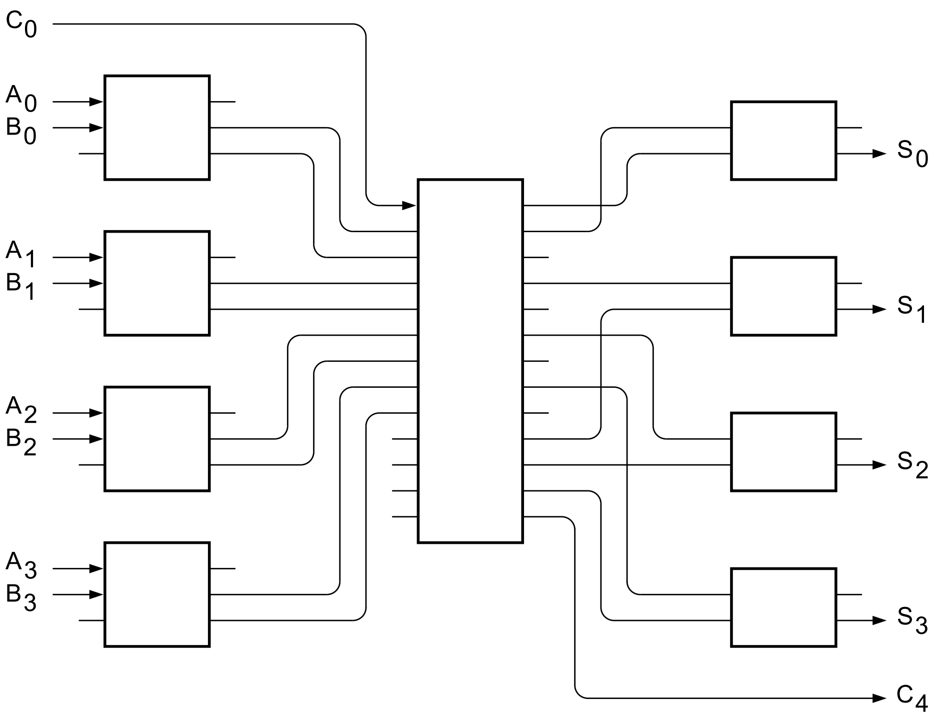


Fig. 4. Schematic diagram of reversible carry-look-ahead four-bit adder

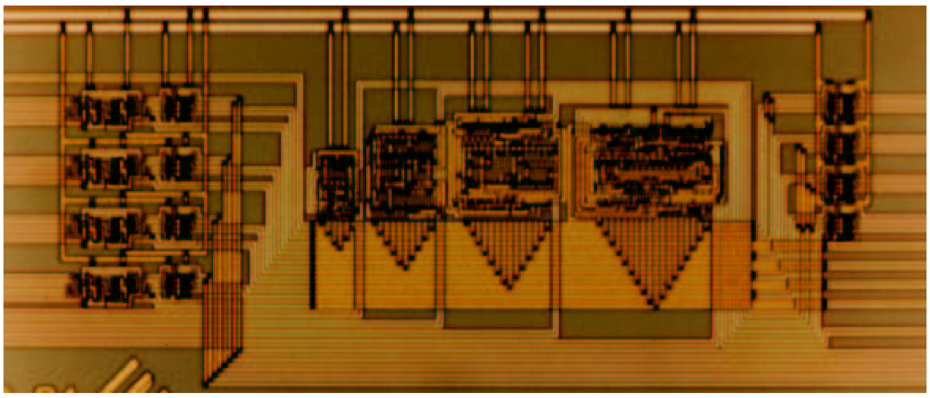
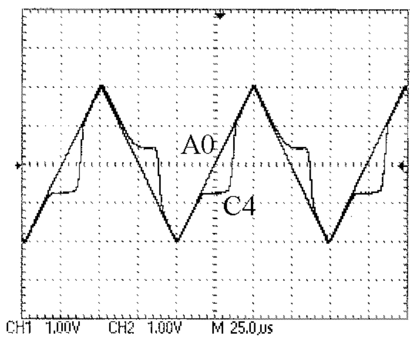
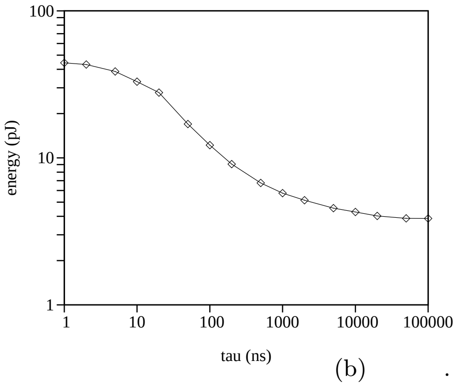


Fig. 5. Microscope photograph of c-MOS reversible carry-look-ahead four-bit adder



(a)



(b) .

Fig. 6. Oscilloscope curve and `Spectre` simulation of carry-look-ahead four-bit adder

References

1. R. Landauer: Irreversibility and heat generation in the computing process. I.B.M. Journal of Research and Development **5** (1961) 183–191
2. C. Bennett and R. Landauer: The fundamental physical limits of computation. Sc. American **253** (July 1985) 38–46
3. R. Landauer: Information is physical. Physics Today **44** (May 1991) 23–29
4. T. Toffoli: Reversible computing. In: J. De Bakker and J. Van Leeuwen (eds.): 7 th Colloquium on Automata, Languages and Programming, Springer, Berlin (1980) 632–644
5. E. Fredkin and T. Toffoli: Conservative logic. Int. Journal of Theoretical Physics **21** (1982) 219–253
6. L. Storme, A. De Vos, and G. Jacobs: Group theoretical aspects of reversible logic gates. Journal of Universal Computer Science **5** (1999) 307–321
7. R. Feynman: Quantum mechanical computers. Optics News **11** (1985) 11–20
8. R. Feynman: Feynman lectures on computation (A. Hey and R. Allen, eds.). Addison-Wesley, Reading (1996)
9. A. De Vos: Reversible computing. Progress in Quantum Electronics **23** (1999) 1–49
10. B. Desoete and A. De Vos: Optimal charging of capacitors. In: A. Trullemans and J. Sparsø (eds.): Proc. 8 th Int. Workshop Patmos, Lyngby (Oct. 1998) 335–344
11. A. De Vos and B. Desoete: Equipartition principles in finite-time thermodynamics. Journal of Non-Equilibrium Thermodynamics **25** (2000) 1–13
12. A. De Vos, B. Desoete, A. Adamski, P. Pietrzak, M. Sibiński, and T. Widerski: Design of reversible logic circuits by means of control gates. In: D. Soudris, P. Pirsch, and E. Barke (eds.): Proc. 10 th Int. Workshop Patmos, Göttingen (Sept. 2000) 255–264
13. H. Taub: Digital circuits and microprocessors. Mc Graw Hill, Auckland (1982)
14. S. Kim and M. Papaefthymiou: True single-phase energy-recovering logic for low-power, high-speed VLSI. In: Proc. 1998 Int. Symposium on Low Power Electronics & Design, Monterey (August 1998) 167–172
15. S. Kim and M. Papaefthymiou: Pipelined DSP design with true single-phase energy-recovering logic style. In: Proc. I.E.E.E. Alessandro Volta Memorial Workshop on Low Power Design, Como (March 1999) 135–143
16. S. Kim and M. Papaefthymiou: Low-energy adder design with a single-phase source-coupled adiabatic logic. In: J. Sparsø and D. Soudris (eds.): Proc. 9 th Int. Workshop Patmos, Kos (Oct. 1999) 93–102