A LATCH FOR A PAIR OF BINARY BITS

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims priority and the benefit of PCT/CN2024/085517, filed 2 April 2024, and PCT/CN2024/119240, filed 16 September 2024, PCT/CN2025/084849, filed on 26 March 2025, the entireties of which are incorporated herein by reference.

TECHNICAL FIELD OF THE INVENTION

The invention relates to logic circuits for manipulating bits in input binary strings to produce arithmetic output. The invention relates particularly to latches.

BACKGROUND OF THE INVENTION

A processor comprises logic circuits for performing arithmetic operations on numbers in the form of binary strings, i.e. addition, subtraction, multiplication and division. One of the most basic logic circuits is the binary adder.

Adders and multipliers may be produced as processors hardwired in arrays to take input of binary data in pre-determined string lengths and give output of a maximum string lengths.

Latches and flip-lops have been used in more dynamic circuits for taking in greater string lengths and manipulating the strings to reduce the burden of hardwiring large sets of arrays, such as in allowing bit shifting into positions of higher order to perform the calculations or re-use the output or a previous calculation in a subsequent one. The data manipulation is not by software but hardware design, although calling these processors digital. The signals representing in zeros and nonzeros comply to voltage levels but the logic circuits reacting to the input signals are analogue circuits.

A classical latch is a circuit that is able to store a single bit of information until a new input signal replaces the stored bit. Latches are sequential circuits arranged to have low states and may be made of logic gates such as AND, OR, NOT, NAND, and NOR gates.

Figure 1a shows a prior art SR Latch made of 2 cross-coupled NOR gate with the truth table in Figure 1b, while Figure 2a shows a 2 cross-coupled NAND gate with the truth table in Figure 2b. There are two inputs, S for Set and R for Reset, and two outputs Q, Q'. For the NOR SR Latch in Figure 1a, a nonzero signal at the S input (with R=0) sets the output Q to 1 (with Q'=0), while a nonzero signal at the R input (with S=0) resets the output Q to 0 (with Q'=1). When both S and R inputs are at 0, the previous states of Q and Q' are latched. On the other hand, when both S and R inputs are at 1, the latch is said to be in an "undefined" or clear state (Q=Q'=0), as shown in Figure 1b. For the NAND SR Latch in Figure 2a, a nonzero signal at the S input (with R=0) sets the output Q to 0 (with Q'=1), while a nonzero signal at the R input (with S=0) resets the output Q to 1 (with Q'=0). When both S and R inputs are at 1, the previous states of Q and Q' are latched. On the other hand, When both S and R inputs are at 0, the latch is said to be in an "invalid" state (Q=Q'=1), as shown in Figure 2b.

Figure 3a shows a prior art Gated D Latch and Figure 3b is the corresponding Truth Table. A D latch is similar to SR latch with some modifications made. Here, the inputs are complements of each other. The D latch stands for "data latch" as this latch stores single bit temporarily.

For dynamically storing generalized binary numbers which comprise of positive and negative bits, the classical latches and flip-flops may not be appropriate. For data latching or data storage of generalized binary numbers, the classical latches and flip-flops cannot be applicable because there are two bits, a positive x -bit and a negative y -bit, to be latched or stored for each "paired-bit" of a generalized binary number. For example, with the SR latch (Set/Reset latch) and the gated D latch (gated data latch), one may observe there exists a certain extent of circuit redundancy because the output must be $Q=0$ (with $\bar{Q}=1$) or Q=1 (with $\bar{Q}=0$), where an overbar for Q, ie. $\bar{Q}$, does not indicate a negative value, but it rather denotes the complementary state, or the negation of Q. Another problem with the classical SR latch is that it is possible to have continuously oscillating outputs of 0 and 1 if both Set and Reset switches simultaneously from 1 to 0 for a SR latch with

NOR gates, or from 0 to 1 for a SR latch with NAND gates. Hence, it is necessary to ensure both Set and Reset not to have states of 1's simultaneously.

It processors, information of generalized binary numbers must be supplied and processed in paired bits representing a combined value. For example, one of the bit represents a value to be subtracted from the other bit. A string of such pair bits provides two binary number from which one is subtracted from the other. In such cases, use of two classical latches to memorize any such pair bits do not have sufficiently uniform response to input signals and miscalculations from one latch responding to a changed input value faster than the other can happen.

In binary strings in unpaired regular use need only make sure sufficient time is provided for each latch to respond to one input value and give an output also of one value. However, where each value is represented by such a pair of strings, the number of factors to consider whether all the reactions to be given in a series of latches have been given are greater, and the chances of calculations made on incompletely updated latch response is higher.

Therefore, it is desirable to propose a latch design with improved, uniform response for two-bit values of a paired-bit.

It is therefore desirable to propose methods and related circuits that provide a possibility of relieving the need to wait for the signal of a carry-over bit.

SUMMARY OF THE INVENTION

In a first aspect the invention proposes latch circuit for receiving a pair of bits, comprising:

a first latch input for one of the bits in the pair and a first latch output, a second latch input for the other one of the bits in the pair and a second latch output

(1) the first latch input being one of two pins of a first upstream logic gate, the first latch output being the output pin of a first downstream logic gate;

the output of the first upstream logic gate connected to one of two input pins of the first downstream logic gate;

the first latch output connected to the other one of two pins of the first upstream logic gate as a feedback;

(2) the second latch input being one of two pins of a second upstream logic gate, the second latch output being the output pin of a second downstream logic gate;

the output of the second upstream logic gate connected to one of two input pins of the second downstream logic gate;

the second latch output connected to the other one of two pins of the second upstream logic gate as a feedback;

(3) a midstream logic gate having two input pins each connected to one of the output of the first upstream logic gate and the output of the second upstream logic gate;

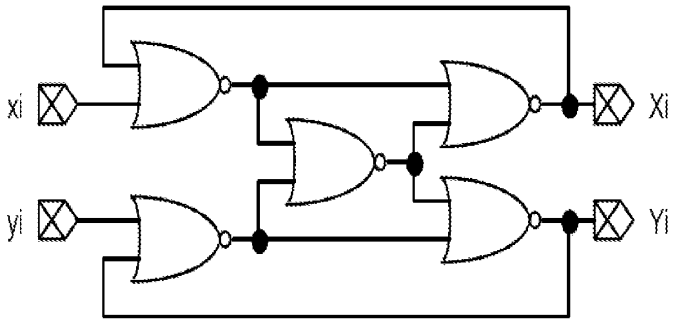
the output of the midstream logic gate connected the other of the two input pins of the second downstream logic gate and the other of the two input pins of the first downstream logic gate.

Optionally, each of the first and second upstream logic gates, the midstream logic gate and the first and second downstream logic gates being NOR gate.

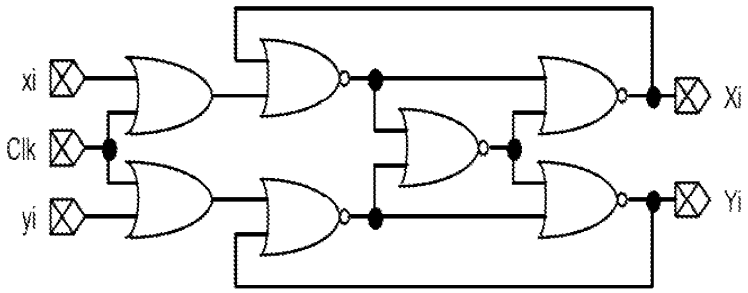
Optionally, each of the first and second upstream logic gates, the midstream logic gate and the first and second downstream logic gates being NAND gate.

Preferably, the latch circuit comprises one of the following:

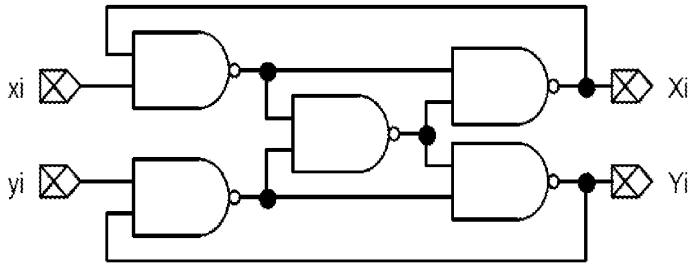
Subcircuit FFXY1



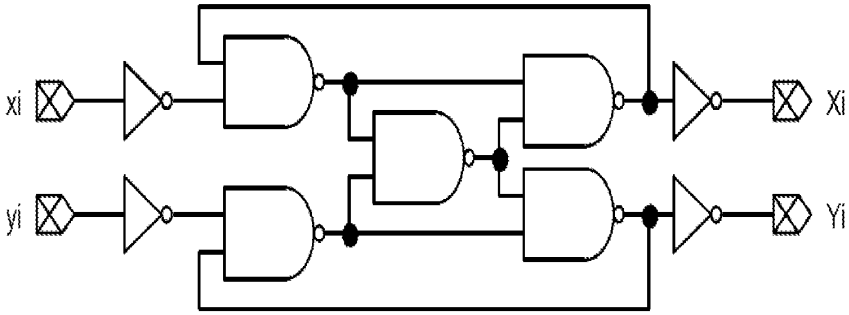
Subcircuit FFGXY1



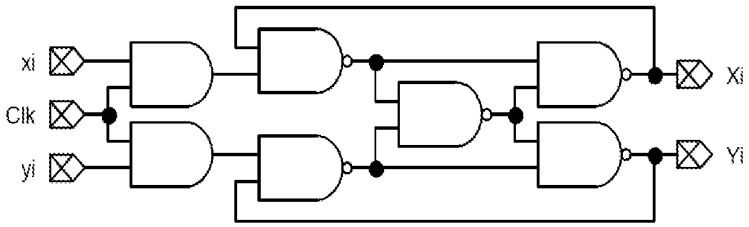
Subcircuit FFXY2



Subcircuit FFXY3



Subcircuit FFGXY2



BRIEF DESCRIPTION OF THE FIGURES

It will be convenient to further describe the present invention with respect to the accompanying drawings that illustrate possible arrangements of the invention, in which like integers refer to like parts. Other arrangements of the invention are possible, and consequently the particularity of the accompanying drawings is not to be understood as superseding the generality of the preceding description of the invention.

Figure 1a is a circuit diagram of a prior art;
Figure 1b is the Truth Table of the circuit of Figure 1;
Figure 2a is a circuit diagram of a prior art;
Figure 2b is the Truth Table of the circuit of Figure 2;
Figure 3a is a circuit diagram of a prior art;
Figure 3b is the Truth Table of the circuit of Figure 3;
Figure 4a is the Truth Table of the generalized NOR XY latch;
Figure 4b is the circuit diagram corresponding to Figure 4a;
Figure 5a is the Truth Table of the generalized NAND XY latch;
Figure 5b is the circuit diagram corresponding to Figure 5a;
Figure 6a is the Truth Table of the generalized NAND XY latch converted to function the role of a generalized NOR XY latch;
Figure 6b is the circuit diagram corresponding to Figure 6a;
Figure 7a is the Truth Table of the generalized gated NOR XY latch;
Figure 7b is the circuit diagram corresponding to Figure 7a;
Figure 8a is the Truth Table of the generalized gated NAND XY latch;
Figure 8b is the circuit diagram corresponding to Figure 8a;

DETAILED DESCRIPTION OF SPECIFIC EMBODIMENTS

Embodiments described in the following paragraphs are founded on a proposed General Equivalence Principle and Generalised Binary Number Format, which have

to be explained first before describing any embodiment in detail, a binary string Transformation called the Specific Equivalence Principle.

Equivalence Principle (General)

Any positive whole number with zero excluded (i.e. natural number) can be expressed as the sum of two positive whole number components of smaller magnitude, provided that the whole number is not 1. However, if the component numbers may be integers of positive and negative numbers, then any positive whole number can be expressed as multiple different pairs of component integer numbers, both of which may be positive or negative, or either one be positive and the other negative. The sum of two integers may be expressed as many sets of four component numbers that add up to equal sum.

Accordingly, given a pair of binary strings to be added up to find their sum, it is possible to manipulate the zeros and nonzeros between the pair of binary strings so as to provide a different pair of binary strings but which add up to equal sum. The principle of picking different sets (not just a pair) of component numbers the sum of which is a constant number is the proposed General Equivalence Principle.

Generalised Binary Number Format and the process of Generalization

A proposed process of 'generalization' is used to re-express or re-cast a normal binary string representing a certain number into a set or a pair of binary strings that add up to the same number. The paired binary strings are called 'substrings' for distinction, and are specifically given the notations x and y this description. The value of each substring may be read like a typical binary string.

The x substring and y substring may be transformed into another pair of x substring and y substring that have different values but nevertheless add up to equal sum, by using transformations that are designed to observe the Equivalence Principle.

While it is possible in some embodiments that both the x substring and y substring of a pair are positive values, it is preferred that the x substring represents a positive value and the y substring represents a 'negative' value as this configuration provides a larger range of values for more flexible manipulation. Hence, x substring – y substring = the value of the original binary string. However, it is also preferably that the value of the y substring is an absolute value, and that the value is to be subtracted from the x substring, rather than the y substring being a negative value. This is because a negative number in binary and in a processor environment is expressed in Two's Complement, and tends to have many nonzeros that are potential cause of carry-over values. The embodiment is able to deal with the many nonzeros, but it is not convenient for explaining the embodiment or to check for errors with the many nonzeros to deal with.

Accordingly, it would be more accurate to refer to the y substring as subtrahend to the x substring as minuend. However, referring to 'minuends' and 'subtrahends' many times makes the description a little convoluted. Therefore, in the following paragraphs, a y substring has an absolute value but may be referred to as a 'negative' binary substring or a 'negative component' substring, the value of which is 'negative', and a nonzero bit in a y substring as a 'negative' bit, which is a little more intuitive to most readers. Conversely, an x substring may be referred to as a positive component substring, the value of which is positive, and a nonzero bit in an x substring as a positive bit.

'String' and 'number' are used interchangeably in this description when describing the binary.

An augend binary number and an addend binary number are provided, which are converted each into a generalised binary number, i.e. $(n_w)_1$ and $(n_w)_2$. Each generalised binary number comprises pair of substrings, x_1, y_1, and x_2, y_2. Generalisation of a n-bit binary string n_w can be presented in mathematical notations as follows:

$$n_w = n_w^+ + n_w^- = a_{n-1}^+ a_{n-2}^+ a_{n-3}^+ \ldots a_2^+ a_1^+ a_0^+ - a_{n-1}^- a_{n-2}^- a_{n-3}^- \ldots a_2^- a_1^- a_0^-$$
$$= \begin{pmatrix} a_{n-1}^+ a_{n-2}^+ a_{n-3}^+ \ldots a_2^+ a_1^+ a_0^+ \\ a_{n-1}^- a_{n-2}^- a_{n-3}^- \ldots a_2^- a_1^- a_0^- \end{pmatrix} \quad (1)$$

where

$a_i^+ = \langle 0,1 \rangle$ and $a_i^- = \langle 0, \bar{1} \rangle$, in which $i = 0,1,...,n-1$; and

<, > indicates a set of possible values for the variable.

$\bar{1}$ is a shorthand for a nonzero bit in the y substring, consisting of a bar over a digit 1.

The weight of each bit in the binary strings is unity or a positive-integer power of 2. Hence, n_w can be expressed as:

$$n_w = \sum_{i=0}^{n-1} 2^i a_i^+ - \sum_{i=0}^{n-1} 2^i a_i^- = \sum_{i=0}^{n-1} 2^i \left(a_i^+ - a_i^- \right) \quad (2)$$

The domain of n_w covers a range from $-(2^n - 1)$ to $2^n - 1$.

An algorithm based on the above may be implemented by the circuit block shown as Figure 6, which takes in an 8-bit binary string and outputs a pair of x substring and y substring, each in 8-bit. Inside the circuit block of Figure 6 is the logic circuit shown as Figure 7. When a number in the form of a binary string is fed into the logic circuit in of Figure 7, a replica of the binary string is made and provided as the x substring, to which is assigned an all-zero binary string of the same length for the y substring. The pairing completes the process of generalization. However, if the sign bit of the binary string is nonzero, the replica of the binary string for the x substring stops short of this sign bit. The nonzero sign bit is placed in the same bit-position (the most significant) in the otherwise all-zero y substring.

Therefore, a binary string 1000 0011 (-125 in decimal) is generalised into an x substring of 0000 0011 (3 in decimal) and a y substring of 1000 0000 (an absolute number 128 in decimal). As mentioned, the y substring is subtrahend to the x substring as minuend, i.e. x - y = $3 - 128 = -125$.

Upon the pairing, the original binary string has become “generalised” because the string’s value is no longer represented by a binary string but by one of many possible pairs of x and y substring as subtrahend that give and identical sum, in accordance to the General Equivalence Principle.

Preferably, the x substring and y substring are each allocated a string length longer than that of the original binary string by one bit. This extra bit allows the maximum value of the x substring to bigger by one binary order, such that the value of the original binary string falls easily within a large minuend value from which the value of y substring may be subtracting to give the value of the original binary string.

A “Specific” Equivalent Principle Conversion

A particularly useful conversion based on the Equivalence Principle is described in some detail in Box 1. It is easier to appreciate the use of the Generalised Binary Number Format by considering what is called in this description the Specific Equivalence Principle (SEP) conversion.

Box 1

The *Specific Equivalence Principle* in a generalized binary number system states that:

- *Any substring of* $m\ (\geq 1)$ *1’s can be replaced by an 1 (more significant) and an* $\bar{1}$ *(less significant) separated by m-1 zeroes.*
- *Similarly, any substring of* $m(\geq 1)\ \bar{1}$*’s can be replaced by an* $\bar{1}$ *(more significant) and an 1 (less significant) separated by m-1 zeroes.*

The effect of the SEP conversion removes all adjacent nonzeros in a substring and can be applied separately to each bit-position in parallel processes.

Taking the positive string as $x_i = \langle 0,1 \rangle$ and the negative string as $y_i = \langle 0,\overline{1} \rangle$, the *Specific Equivalence Principle* for a substring of *m* (≥ 1) 1's can be expressed by Equations (3a) and (3b), while that for a substring of *m* (≥ 1) $\overline{1}$'s can be expressed by Equations (4a) and (4b).

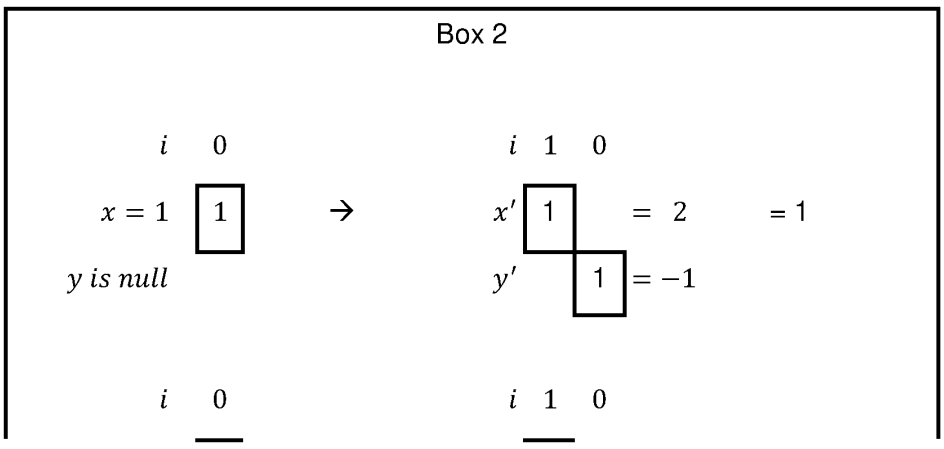
$$X_i = \overline{x}_i x_{i-1} \quad ; \quad Y_i = x_i \overline{x}_{i-1} \tag{3a,3b}$$

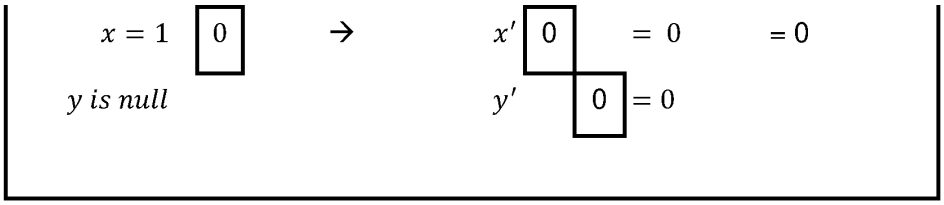
$$X_i = y_i \overline{y}_{i-1} \quad ; \quad Y_i = \overline{y}_i y_{i-1} \tag{4a,4b}$$

The left in Figure 9 is the corresponding circuit block diagram 901 while the right drawing in Figure 9 is the underlying logic circuit unit 903 for SEP conversion of an x substring. Similarly, the left drawing in Figure 10 is the corresponding circuit block diagram while the right drawing in Figure 10 is the underlying logic circuit unit for SEP conversion of an y substring. The corresponding Truth Table is presented in Figure 11, where the left half of the table relates to conversion of a nonzero in the positive component x substring, and the right half of the Truth Table relates to conversion of the negative sign component y substring.

Box 2 below shows the SEP conversion, which is based on the possibility of re-expressing a positive bit having a nonzero value of 1 as a greater number offset with a smaller and negative number, such as $x - y = 2 - 1 = 1$, while observing the Equivalence Principle. For completeness, Box 2 shows that if the bit has zero value the output is zero at a higher order. Actually, this is determined passively from the active re-positioning of nonzero bits in the output.

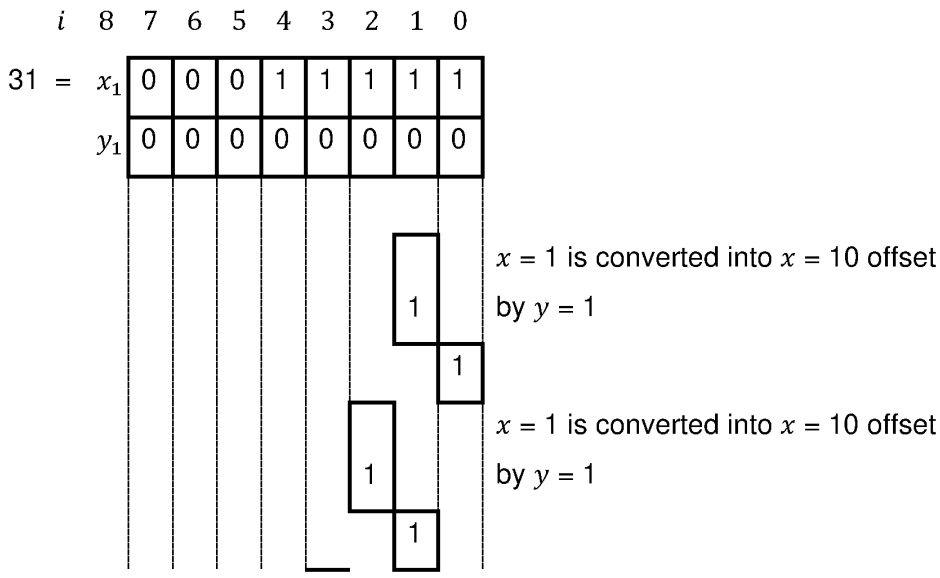
Box 2

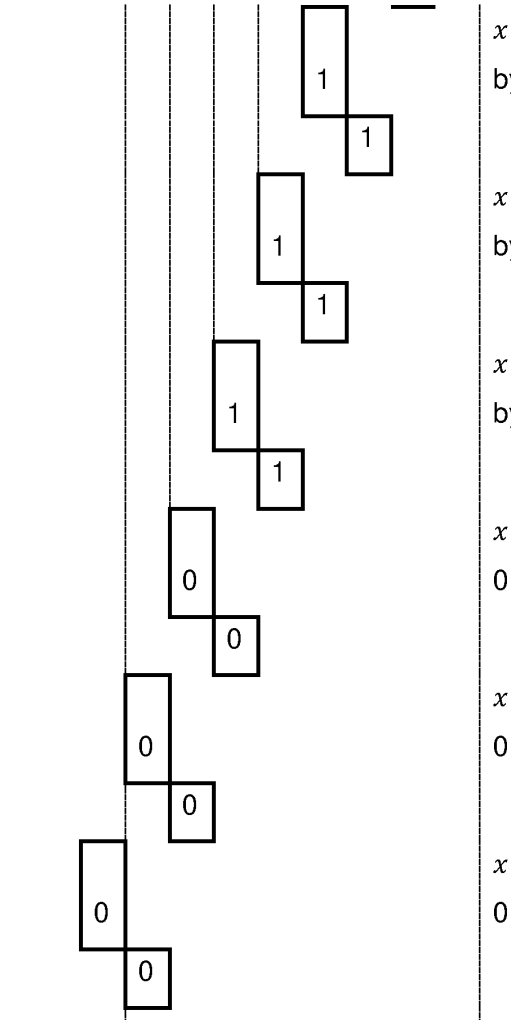




An example of applying the SEP conversion in Box 2 to a binary string is given in the following paragraphs. Taking as example the number 31, which in 8-bit binary format is 0001 1111, generalization re-expresses 0001 1111 as an x substring = 0001 1111, paired with an all-zero y substring = 0000 0000 as there is no sign bit. Box 3 below shows the SEP conversion shown in Box 2 applied to the x substring. The x substring is converted into 0010 0000 (32 in decimal) and the y substring is converted into 0000 0001 (1 in decimal). Therefore, $x - y$ in the decimal system is $32 - 1 = 31$, giving the same sum and observing the General Equivalence Principle. For convenience, these substrings may be written as 0010 000$\bar{1}$ in the recommended shorthand described in Equation (1).

Box 3





x = 1 is converted into x = 10 offset by y = 1

x = 1 is converted into x = 10 offset by y = 1

x = 1 is converted into x = 10 offset by y = 1

x = 0 is converted into x = 00 and y = 0

x = 0 is converted into x = 00 and y = 0

x = 0 is converted into x = 00 and y = 0

x_1' =	0	0	0	1	1	1	1	1
y_1' =		0	0	0	1	1	1	1	1

These results are obtained by compiling the output of the SEP conversion of each bit-position.

32 =	0	0	0	1	0	0	0	0	**0**
-1 =	**0**	0	0	0	0	0	0	0	1

Where x = 1 an y = 1 in the same bit-position, the two nonzero bits cancel out into zero, i.e., simplification.

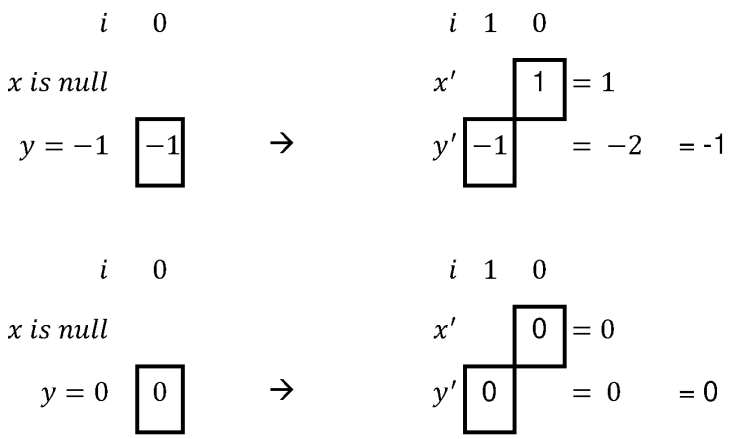
As described already in a preceding paragraph, it can be seen that the overall effect of the SEP conversion lies in the removal of all immediately adjacent nonzero bits in

the x substring, but without changing the sum 31, i.e. 32-1. The SEP conversion must be applied consistently to every bit-position in the x substring.

Using the shorthand $\bar{1}$ to denote a nonzero bit in the y substring, a pair of substrings x =0001 0000 and y = 0000 0001 may be written in a single line as 0001 000$\bar{1}$. There cannot be a 1 and a $\bar{1}$ in the same bit-position as they cancel out and becomes zeros.

The SEP conversion may have to be applied also to every bit-position the y substring too, although there is nothing to demonstrate in the example of an all-zero y substring, Box 4 below shows the SEP conversion in Box 2 expressed for the y substring.

Box 4



The example in Box 4 has a final step of simplifying the SEP conversion output, which is not desirable for it adds an additional step to the overall embodiment.

Box 5 below relates to a variation of the SEP conversion, which produces outputs from inputs according to the Truth Table in Figure 11 but without the need of a simplification step after SEP conversion. The corresponding logic circuits are shown as Figure 9 and Figure 10.

While the SEP conversion in Box 2 and Box 4 reads a single bit for input and gives two bits for output, the SEP conversion in Box 5 reads two bits for input, one from the current position $i = 1$ and the other from the lower position $i = 0$. If there is no position $i = 0$ because $i = 1$ is already the smallest bit at the start of a binary string, a zero value is prescribed to the non-existent bit in position $i = 0$. Furthermore, although the result of the SEP conversion of Box 5 may include bits in more than one bit-position, only the bits in position $i = 1$ are taken compiled into the Truth Table. To draw the reader's attention to this, SEP conversion output bits in position $i = 1$ are shown unshaded, while adjacent bits not included into the compilation of the Truth Table are shown shaded.

Box 5

Cases	i 1 0		i 2 1 0		Equivalent
i.	x 0 0	$= 0$	x' 0 0 y' 0 0	$=$	$0 = 0$
ii.	x 0 1	$= 1$	x' 1 0 y' 0 1	$=$ $=$	$2 = 1$ -1
iii.	x 1 0	$= 2$	x' 1 0 0 y' 1 0	$=$ $=$	$4 = 2$ -2
iv.	x 1 1 x' 1 1 y' 1 1	$= 3$ $=$	x' 1 0 0 y' 0 1	$=$ $=$	$4 = 3$ -1

Cases	i 1 0		i 2 1 0		Equivalent
v.	y 0 0	$= 0$	x' 0 0 y' 0 0	$=$	$0 = 0$
vi.	y 0 1	$= -1$	x' 0 1 y' 1 0	$=$	$1 = -1$ -2
vii.	y 1 0	$= -2$	x' 1 0	$=$	$2 = -2$

viii. $\overline{y\ 1\ 1} = -3 \quad x' \quad 1\ 1 \quad = \quad x' \quad 0\ 1 \quad = \quad 1 \quad = \quad -3$

$y' \quad 1\ 1 \qquad y' \quad 1\ 0\ 0 \qquad -4$

Cases i to iv in Box 5 show SEP conversions of all the four possible combinations of two bits in the positive component x substring, i.e. 00, 01, 10, 11. The two bits are adjacent to each other in the same substring. Conversely, cases v to viii in Box 5 shows the SEP conversion of all the four possible combinations of two bits in the negative component y substring, i.e. 00, 01, 10, 11.

An example of applying the SEP conversion in Box 5 to a binary string is shown in Box 6. Taking as example the number 23, which in an 8-bit binary format is 0001 0111 and which generalizes into an x substring = 0001 0111 and a y substring = 0000 0000.

Upon SEP conversion of each bit based on the transformation in Box 6, the x substring becomes 0010 1000 (40 in decimal), and the y substring becomes 0001 0001 (17 in decimal). The difference between $x - y$ in the decimal system is 40 – 17 = 23 (General Equivalence Principle observed). These substrings may be written as $001\bar{1}\ 100\bar{1}$ in the recommended shorthand.

Data Latching for Generalized Binary Numbers

Hence, a "generalized latch" for a paired-bit XY, each from one of the x and y substrings is proposed. There are two different versions, the generalized basic NOR XY latch with the logic truth and logic circuit shown in Figure 4a and Figure 4b, while the generalized basic NAND XY latch where the logic truth and logic circuit are shown in Figure 5a and Figure 5b, respectively.

Referring to the truth table shown as Figure 4a and Figure 4b, the generalized basic NOR XY latch has some unique characteristics that (i) the output states X_i, Y_i

assume the input states x_i, y_i if both inputs states are different; (ii) $X_i = Y_i = 0$ if $x_i = y_i = 1$; (iii) X_i, Y_i memorize (or store) the preceding states if $x_i = y_i = 0$, and further simplify $X_i = Y_i = 1$ to $X_i = Y_i = 0$ in the next cycle. It should be highlighted that all inputs and present states take only one cycle to reach stable output states except for (i) $x_i = 0$, $y_i = 1$, $X_i = 1$ (present state); and (ii) $x_i = 1$, $y_i = 0$, $Y_i = 1$ (present state) which require two cycles to reach stable outputs.

Similarly, referring to the truth table and circuit diagram shown as Figure 5a and Figure 5b, the generalized basic NAND XY latch has some unique characteristics that (i) the output states X_i, Y_i assume the input states x_i, y_i if both inputs states are different; (ii) $X_i = Y_i = 1$ if $x_i = y_i = 0$; and (iii) X_i, Y_i memorize (or store) the preceding states if $x_i = y_i = 1$, and further simplify $X_i = Y_i = 0$ to $X_i = Y_i = 1$ in the next cycle. It should also be highlighted that all inputs and present states take only one cycle to reach stable output states except for (i) $x_i = 0$, $y_i = 1$, $Y_i = 0$ (present state); and (ii) $x_i = 1$, $y_i = 0$, $X_i = 0$ (present state) which require two cycles to reach stable outputs.

In reality, the NOR XY latch with the truth tables shown in Figure 4a and logic circuit in Figure 4b is complementary to the NAND XY latch with the truth table in Figure 5a and logic circuit in Figure 5b. The outputs in the table shown in Figure 5a and Figure 5b can be converted to the corresponding outputs in the table shown as Figure 4a and Figure 4b by simply adding an inverter to each of the inputs and outputs, as shown in the truth table and logic circuit shown as Figure 6a and Figure 6b.

In addition, the generalized NOR and NAND XY latches have overcome a long-lasting inherent defect of the classical SR latch well-known in the prior art, that it is possible to have continuously oscillating outputs of 0 and 1 if both Set and Reset switch simultaneously (i) from 1 to 0 for a classical SR latch with NOR gates, or (ii) from 0 to 1 for a classical SR latch with NAND gates. Take the classical SR latch

with NOR gates as an example, the outputs of Q and Q' are both 0's if the current states of S and R are both 1's, see Figure 1b. Upon switching both S and R from 1's to 0's simultaneously, the current states of Q and Q' (both are 0's) are fed as inputs to the NOR gates which produce concurrently 1's for both Q and Q's as outputs, see Figure 1a. The 1's are then fed again as inputs to the NOR gates which produce 0's. The outputs are further fed again as inputs and the condition continues while the outputs interchange between 0's and 1's continuously, so long as S and R remain as 0's. This ever-lasting oscillation of outputs between 0's and 1's is due to the feedback connection between the outputs and inputs of the NOR gates. This defect of the classical SR latch is inherent well-known in the prior art and it has not been resolved hitherto. With the new generalized NOR XY latch, the ever-lasting output oscillation is resolved because the outputs, so fed as inputs in the next iteration, do not directly yield the outputs, see Figure 4b. Instead, the inputs pass through a level of two NOR gates, a level of an intermediate NOR gate and a further level of two NOR gates before yielding the outputs. The presence of intermediate and further NOR gates prevent the output oscillation from taking place.

By intentionally earthing one of the inputs of the generalized NOR and NAND XY latches, say y=0 for instance, these generalized XY latches can be used to replace the classical SR latch to store a single bit data. With such replacements, the x input takes the role of either S (or R) depending on the design requirements. Because S and R in the classical SR latches are complementary to each other which indeed store or memorize the information of a single bit only, the role of x-bit input and its corresponding X-bit output (having the y-bit input earthed or y=0) are completely perfect to replace the function of the classical SR latches. The redundant role of S and R can be solved. Yet the input-output oscillation issue of the classical SR latch is completely resolved.

"Generalized Gated" Latches

The word "generalised" does not refer to a thing that is common, known and seen everywhere, but to the generalised numbers each having a value represented generally by a large possibility of different pairs of binary strings.

Similar to the classical gated latches, an enable or a clock for generalized gated latches can be introduced to the generalized basic NOR and NAND XY latches such that a gate is in place to control the onset or offset of the latches. The logic truth and logic circuit are presented in Figure 7a and Figure 7b for the generalized gated NOR XY latch, while they are presented in the truth table and logic circuit shown as Figure 8a and Figure 8b for the generalized gated NAND XY latch.

Similarly, the table shown as Figure 8a and Figure 8b can be made equivalent to the table shown as Figure 7a and Figure 7b by adding a signal inverter to the inputs and outputs, except the clock.

Regardless of non-gated or gated conditions, the generalized NOR and NAND latches for generalized binary numbers are data latches that work in a similar manner with respect to the classical SR latch and the D latch. Nevertheless, there are at least five key advantages for the new generalized NOR and NAND latches comparing to the classical latches as follows:

- no bit redundancy;
- instead of storing a single bit data, a two-bit generalized binary number data can be stored;
- no oscillating outputs;
- contrary to the classical SR latch, it is not required to ensure Set and Reset to assume complementary 0 and 1 states, or vice versa; and
- data latching is performed with bit simplification where all $x_i = y_i = 1$ is synchronized to $x_i = y_i = 0$ or vice versa, depending on whether it is a NOR XY latch or a NAND XY latch.

In all the tables, it can be seen that the inputs to latch of the paired bits are shown as columns (1) and (2), and every case except for the latches cases, the outputs become the same as the inputs in the last two columns.

For the latch case, in some embodiments, the outputs becomes fixed, memorized or latched. The latching is always triggered by zero inputs of both bits for the generalized NOR XY latch, See rows 1 to 4 of Figure 4a. Regardless of the existing state of the outputs when 0,0 is applied, that state will persist. This is shown with the four output combinations in row 1 to 4, persisting through time represented from column (3), (4), to (5) ,(6) and ending in (7),(8).

Row 5 shows what happens when the inputs of 0,1 are sent to the latch when the present state of the latch is 0,0, shown in column (3),(4). The output of the latch reacts to the input and becomes 0,1 shown in columns (5),(6), and remain 0,1 at time represented by columns (7),(8).

Row 6 is the same as row 5 except that the state of the latch when the inputs of 0,1 shown as columns (1),(2) are applied when the latch is in a present state of 0,1. The latch output remains 0,1, shown in column (3),(4), columns (5),(6), and is stable remain 0,1 at time represented by columns (7),(8).

Row 7 shows what happens when the inputs of 0,1 are sent to the latch when the present state of the latch is 1,0, shown in column (3),(4). The outputs of the latch react to the inputs and become 0,0 shown in columns (5),(6), and finally becomes 0,1 shown as columns (7),(8).

Row 8 shows what happens when the inputs of 0,1 are sent to the latch when the present state of the latch are 1,1, shown in column (3),(4). The outputs of the latch react to the inputs and become 0,0 shown in columns (5),(6), and finally become 0,1 shown as columns (7),(8).

All the Truth Tables show the change that the respective latch goes through, so that one may appreciate the change that takes places is uniform to both output leads for the bit-pair. If the changes in both leads are not in tandem the changes of random error from bit misrepresentation will be huge

The other matters of timing when the latch settles into the final state in reacting to an input is a matter current circuit designs are able to deal with very well. The embodiments only propose a way of ensuring the two pair-bit values when processed or memorized by the latch are able to respond and change in tandem.

While there has been described in the foregoing description preferred embodiments of the present invention, it will be understood by those skilled in the technology concerned that many variations or modifications in details of design, construction or operation may be made without departing from the scope of the present invention as claimed.

CLAIMS

1. A latch circuit for receiving a pair of bits, comprising:

a first latch input for one of the bit in the pair and a first latch output, a second latch input for the other one of the bit in the pair and a second latch output

(1) the first latch input being one of two pins of a first upstream logic gate, the first latch output being the output pin of a first downstream logic gate;

the output of the first upstream logic gate connected to one of two input pins of the first downstream logic gate;

the first latch output connected to the other one of two pins of the first upstream logic gate as a feedback;

(2) the second latch input being one of two pins of a second upstream logic gate, the second latch output being the output pin of a second downstream logic gate;

the output of the second upstream logic gate connected to one of two input pins of the second downstream logic gate;

the second latch output connected to the other one of two pins of the second upstream logic gate as a feedback;

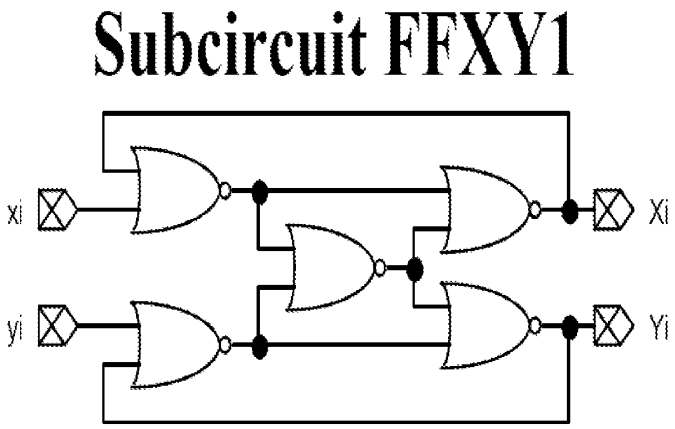
(3) a midstream logic gate having two input pins each connected to one of the output of the first upstream logic gate and the output of the second upstream logic gate;

the output of the midstream logic gate connected the other of the two input pins of the second downstream logic gate and the other of the two input pins of the first downstream logic gate.

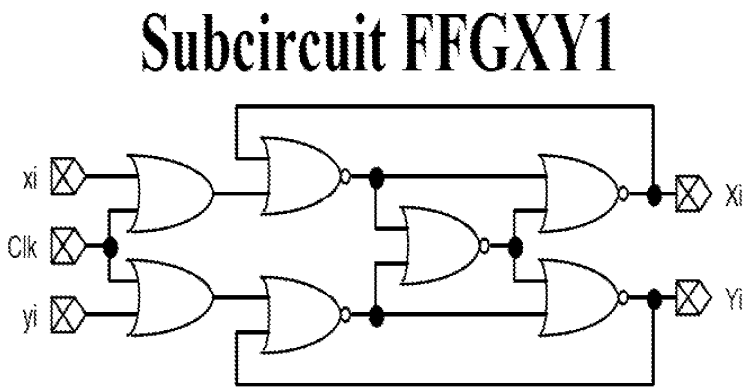
2. A latch circuit for receiving a pair of bits as claimed in claim 1, wherein each of the first and second upstream logic gates, the midstream logic gate and the first and second downstream logic gates being NOR gate.

3. A latch circuit for receiving a pair of bits as claimed in claim 1, wherein each of the first and second upstream logic gates, the midstream logic gate and the first and second downstream logic gates being NAND gate.

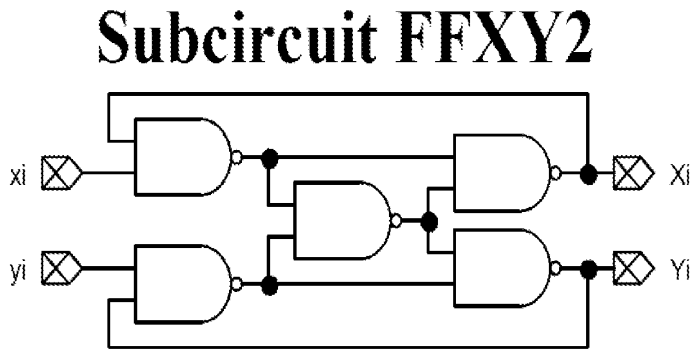
4. A latch circuit for receiving a pair of bits as claimed in claim 2, comprising the logic layout of **:**



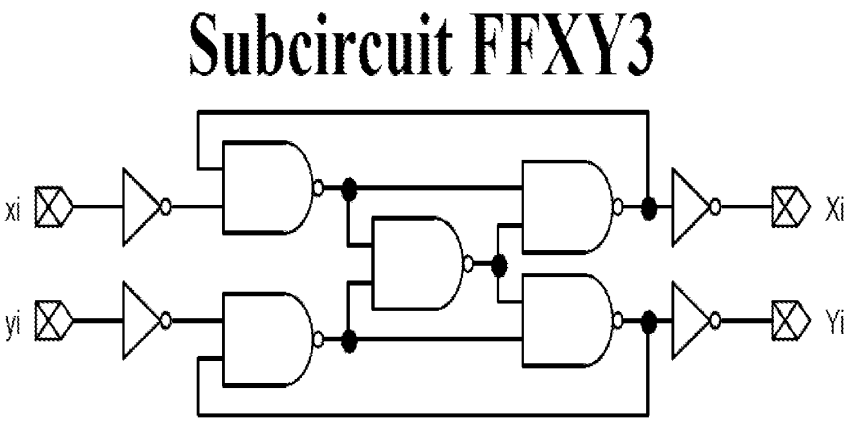
5. A latch circuit for receiving a pair of bits as claimed in claim 4, comprising the logic layout of :



6. A latch circuit for receiving a pair of bits as claimed in claim 3, comprising the logic layout of :



7. A latch circuit for receiving a pair of bits as claimed in claim 6, comprising the logic layout of :



8. A latch circuit for receiving a pair of bits as claimed in claim 6, comprising the logic layout of :

