Parallel Addition And Subtraction In Constant Time *

Matthew Zupan

13/06/2023

Contents

1 Introduction **3**

2 Constant Time Addition **5**

3 Constant Time Subtraction **11**

4 Space Complexity **12**

5 References **13**

1 Introduction

Performing parallel binary addition in constant time within this paper is contingent in the use of a slightly different logical model than was considered in the $O(log_2(n))$ time version from [1]. The difference here is that we now allow for arbitrary splitting of wires in a single step and also allow for the possibility to use a type of OR gate which can perform the OR operation on as many inputs as necessary in a single step, illustrated respectively in Figure 1.1 and 1.2. This newly described version of the OR gate will be referred to as "concurrent OR" or in a shorter way as "COR". Note that signal propagation delay in wires themselves are not taken to account in this model, though for large travel distances this delay would pose a physical limitation to the model. Here we allow for addition of integers of size $n \in \mathbb{Z} > 2$ and use of left to right indexing beginning from 1 and ending at n.

The immediate consequence of the COR operation is that we can convert logical statements into disjunctive normal form, described as a disjunction of conjunctions (OR together many groups of AND terms), for example:

$$z = [x_1 \wedge x_2 \wedge x_3] \vee [x_4 \wedge x_5 \wedge x_6] \vee [x_7 \wedge x_8 \wedge x_9] \vee ... \vee [x_{m-2} \wedge x_{m-1} \wedge x_m]$$

Through use of DeMorgans' Law, we can take all square bracketed terms consisting of the AND operation separately and convert them into a COR operation:

$$[x_1 \wedge x_2 \wedge x_3] = \neg[\neg x_1 \vee \neg x_2 \vee \neg x_3]$$

Thus after negating all literals within the bracket simultaneously in parallel, performing COR of these negated terms to produce one single output, then negating this output, we have reduced the problem to computing:

$$z = [y_1] \vee [y_2] \vee [y_3] \vee ... \vee [y_m]$$

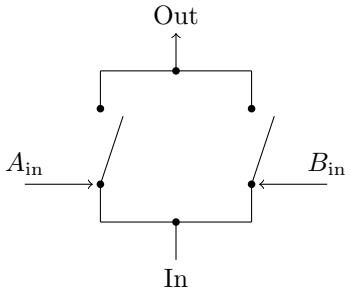
Which can be done in one single step using COR again on these reduced terms, taking an over all time of 4 steps (1. Negate all literals, 2. COR all literals within terms, 3. Negate all reduced terms, 4. COR all reduced literals).

Even though logical statements and questions can be converted into the preceeding normal form, it is known that many cases exist where this conversion requires exponential space. Exponential space complexity will be considered infeasible, putting a damper on our ability within this model to answer every logical question in constant time.

The feasibility of this model is debatable, but we can consider two contexts for which it is applicable, electrical engineering aside. Abstractly, the structure of an OR gate provides for two switches in parallel, such that if one switch A, or switch B, or both are closed, then the output is a logical 1. If both are open,

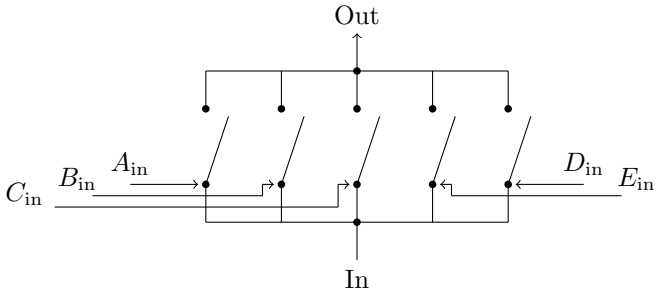
then the output is a logical 0. This is shown as follows in Figure 1.1.

Figure 1.1: Standard OR Gate as Two Parallel Switches.



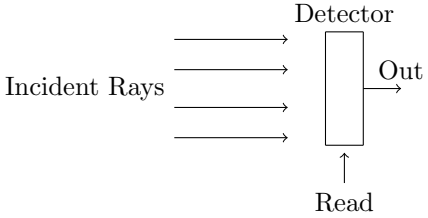
Thus the first context for which we can extend the model, is to allow for multiple switches in parallel, to create our "concurrent OR". Illustrated in Figure 1.2.

Figure 1.2: Concurrent OR Gate with Five Parallel Switches.



The second context is that of a detector. Say we choose to use a photodetector. If we convert logical inputs which are a 1 into light and simultaneously direct each beam of light at the detector, then as long as the detector receives at least one incident signal in conjunction with a separate guiding "read" signal, the detector will output a logical 1 and 0 otherwise. This concept is illustrated in Figure 1.3.

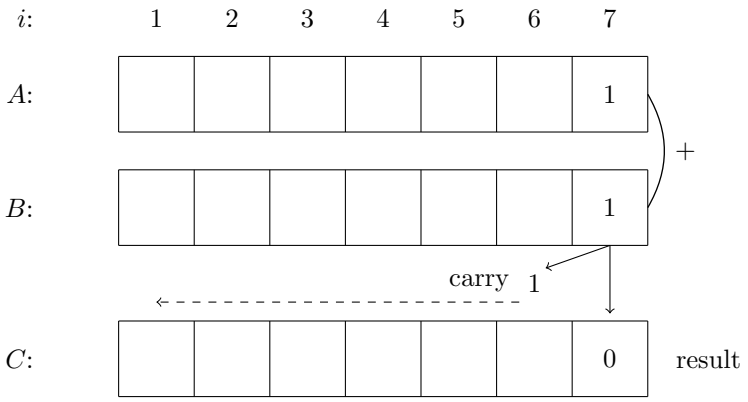
Figure 1.3: Photodetector Concurrent OR.



2 Constant Time Addition

The addition of two integers A and B requires that for each digit at index i, we add together A_i and B_i into result C_i along with an input carry c_{i+1} which has been the result of addition from digits in A and B at index $i+1$, illustrated in Figure 2.1.

Figure 2.1: Addition With Carry.



Of course, we do not have to wait for the carry to propagate down during the addition. To determine whether index i will receive a carry of 1, we consider the following two cases (where n is the length of A or B, padded to be the same size):

- Case 1: Index $i+1$ generates a carry, $(i < n)$. This means index i being immediately on the left must receive this carry.

OR

- Case 2: Some index $i+k$ generates a carry AND all indices $i < j < i+k$ propagate this carry, $(i < n-2,\ k > 1,\ i+k \leq n)$. Thus we also must receive this carry.

We will now identify how to convert the previous statements into binary logic, for case 1 and 2, illustrated in Figures 2.2 and 2.3 respectively.

Figure 2.2: Case 1, $i = 3$.

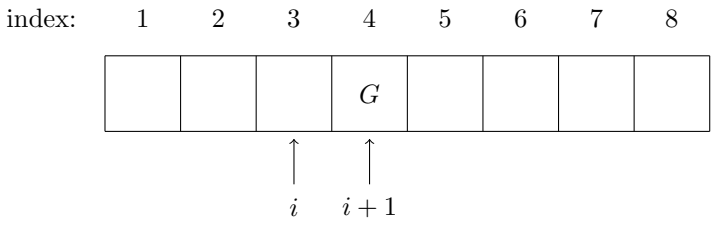
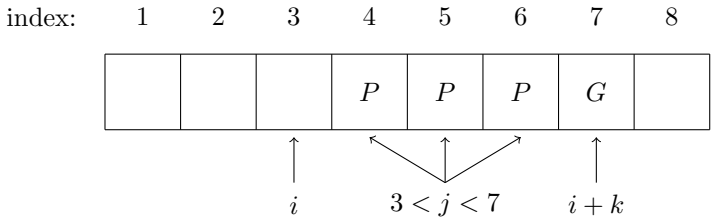


Figure 2.2: Case 2, $i = 3$, $k = 4$.



The proposition "$i + 1$ generates a carry" can only occur when both $A_{i+1} = 1$ AND $B_{i+1} = 1$. Thus we can resolve this by computing $A_{i+1} \wedge B_{i+1}$.

The proposition "$i + k$ generates a carry" can only occur when both $A_{i+k} = 1$ AND $B_{i+k} = 1$. Thus we can resolve this by computing $A_{i+k} \wedge B_{i+k}$.

The proposition "j propagates a carry" can occur in two cases where [$A_j = 1$ AND $B_j = 0$] OR [$A_j = 0$ AND $B_j = 1$]. This can be resolved by computing $(\neg A_j \wedge B_j) \vee (A_j \wedge \neg B_j)$, the circuit components of which are drawn in Figure 2.3 and 2.4 (Where an isolated open circle is the single delay gate).

Figure 2.3: Compute "Generates a Carry" $\rightarrow \{0, 1\}$.

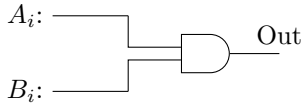
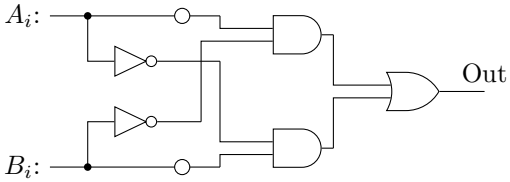


Figure 2.4: Compute "Propagates a Carry" $\rightarrow \{0, 1\}$.

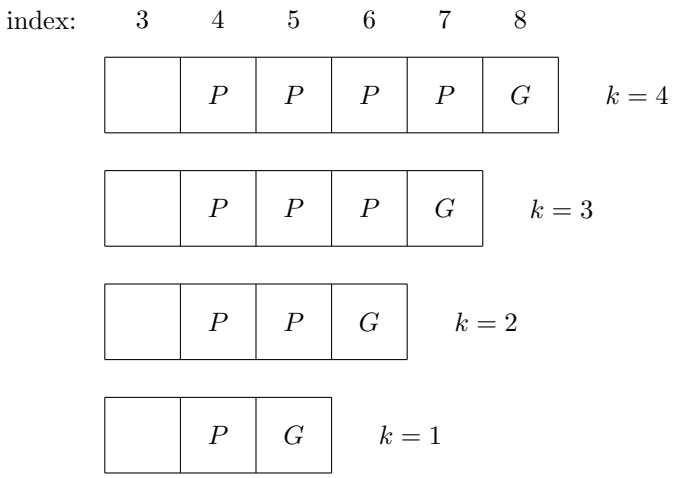


Labelling generators as G and propagators as P, the usefulness of DeMorgans' law becomes apparent in this situation in case 2. Given we require $i+1=P$ AND $i+2=P$ AND ... AND $i+k-1=P$ AND $i+k=G$, we can convert this series of chained AND statements into the newly familiar COR operation via the process $\neg[\neg(i+1=P)\vee\neg(i+2=P)\vee...\vee\neg(i+k-1=P)\vee\neg(i+k=G)]$. Substituting in the appropriate logical propositions in place leads to the following equation:

$$\neg[\neg((\neg A_{i+1}\wedge B_{i+1})\vee(A_{i+1}\wedge\neg B_{i+1}))\vee\neg((\neg A_{i+2}\wedge B_{i+2})\vee(A_{i+2}\wedge\neg B_{i+2}))\vee ...\vee\neg((\neg A_{i+k-1}\wedge B_{i+k-1})\vee(A_{i+k-1}\wedge\neg B_{i+k-1}))\vee\neg(A_{i+k}\wedge B_{i+k})].$$

It becomes clear that within case 2, we must separately check all k for which $i+1<k\leq n$ may generate a carry, and therefore for all j in as propagators in between. Thus for a given digit at index i, case 2 is broken down into $n-i-1$ subcases. This can be done by forming $n-i-1$ copies of the data from all $A_{i+k}\wedge B_{i+k}$ and $(\neg A_j\wedge B_j)\vee(A_j\wedge\neg B_j)$, checking all subcases separately, and then performing a COR operation on the results, as we will find only one instance is true if a carry is received. This subcase breakdown is illustrated in Figure 2.5.

Figure 2.5: Case 2 Subcase (Test-For) Breakdown, $i=3$.



Finally, we know that if index i receives a carry, then case 1 or case 2 must be true. While both cases cannot be true at the same time, it is sufficient to use OR between the results of these cases, as we are only concerned with showing that satisfying a single case results in a carry. If none are true, then COR will reflect this with an output of 0.

With all previous components established, we can summarise and combine into

a final circuit representing the entire series of operations leading to the carry result from A and B as follows, note carry at index 1 may result in the result of addition requiring an index 0 in C, thus extend the range of i to $0 \leq i \leq n$, A_0 and B_0 are padded with 0 here and C_n receives no carry and C_{n-1} only includes case 1.

BEGIN

- $X := A_i \wedge B_i$ for each $1 \leq i \leq n$, then delay for 2 time steps.
  - Split X into 2 separate streams, $Y := X\neg[2 : n]$
  (negate all elements) and $Z := X[1 : n]$, delay Z by 5 time steps.
    - $Z'_{n-1} := Z[n-1]$, delayed by a further 2 time steps.
- $R := (\neg A_i \wedge B_i) \vee (A_i \wedge \neg B_i)$ for each i, $(A_i \oplus B_i)$.
  - Split R into 2 separate streams, $S := R\neg[1 : n-1]$
  (negate all elements) and $T := R$, delay T by 4 time steps.
    - Delay $T[n]$ by a further 3 time steps.

Then $\forall j$, $0 \leq j \leq n-2$:

- $Y'_j[0 : n-2-j] := \mathrm{COR}(S[0 : k], Y[k])$, $\forall\ j \leq k \leq n-2$.
  - $Z'_j := \mathrm{COR}(\neg Y'_j[0 : n-2-j], Z[j])$, (negate all elements of Y'_j, we can write this in a shorter form: $Z'_j := \mathrm{COR}(\neg Y'_j, Z[j])$).

Lastly, $\forall j$, $0 \leq j \leq n-1$:

- $W := \oplus(Z'_j, T_j)$.

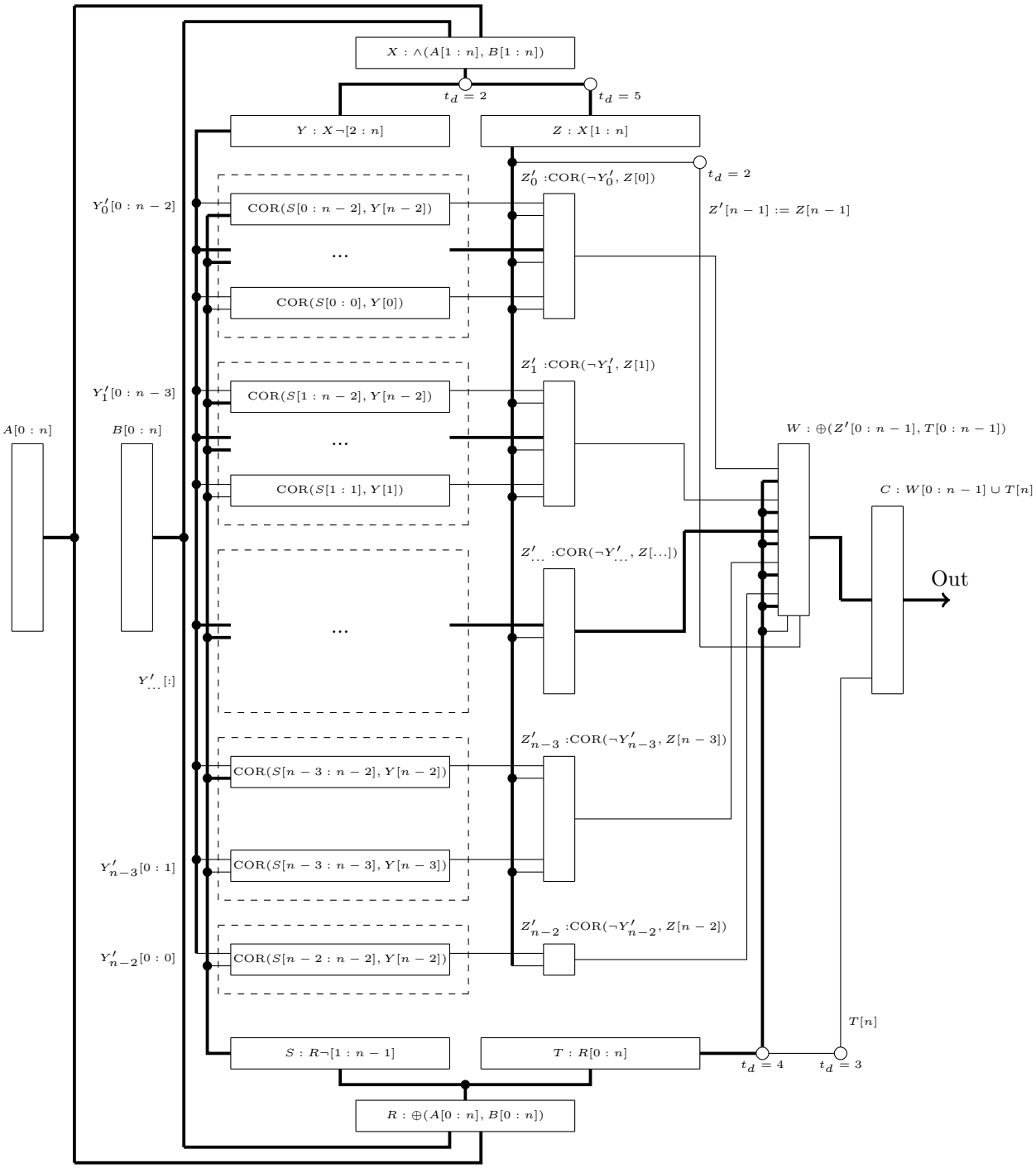
Thus the output C is defined to be:

- $W_j \cup T[n]$.

END

From the previous, we will illustrate the entire process in Figure 2.6 and then offer an explanation.

Figure 2.6: Constant Time Addition Circuit.



Comping X as $A_i \wedge B_i$ for $1 \leq i \leq n$ allows us to share work between cases 1 and 2. Splitting X into Z provides case 1 carry information for digits 0 to $n-1$. Splitting into Y allows us to use the same work to determine generators in case 2 for digits 0 to $n-2$. Y also negates each element for use in the later COR operation. Furthermore, since Y and Z are of different sizes (case 2 does not compute carry information for digit $n-1$), we require saving $Z[n]$ for later as $Z'[n-1]$.

Computing R as $A_i \oplus B_i$ for $0 \leq i \leq n$ allows us to share work between case 2 and the final addition of result + carry, where $\oplus$ computes the result component. Thus we split R into S and T, where T contains the result. S contains the negated propagator information from case 2 for all digits 1 to $n-1$ for later use in the later COR operation.

Each digit contains its own case 1 and case 2 computations, with case 2 being broken down into subcases. Thus we have Y'_j containing the set of subcases for the jth digit. Each subcase is COR'd together. However, given we seek to know if any subcase is true, as well as case 1, we can combine Y_j into a single result with the respective case 1 carry information by defining Z'_j as the negation of each subcase COR'd with the respective Z_j component.

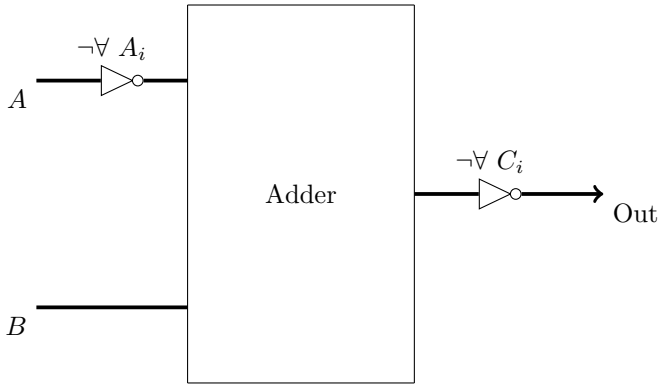
Now we seek to add together the carry information with the result information for all digits 0 to $n-1$, with each digit in W defined to be $\oplus$ between each digit in Z' and T. Now the final result, which is the output C is composed of the results from W, union with $T[n]$. As noted before, the last digit n does not receive a carry.

Time delays inserted between circuit components are to ensure that the signals are synchronised and arriving concurrently. This is a theoretic requirement, but in practice, signal delays over wires of different distances may be a timing hazard. Thin wires within Figure 2.6 represent a single element of data to be transferred, while thicker wires represent a bus, abstractly grouping together a collection of many wires to improve diagram readability. This circuit takes 10 time steps to perform the computation $A + B$, giving us constant time.

3 Constant Time Subtraction

Subtraction in this context requires the value $A > B$. As per [1], we only require to negate all inputs to A, $(\neg A)$, run $\neg A$ and B through the constant time adder, and then negate all outputs $\neg C$. This simple modification allows for re-use of the adder and only takes 2 time steps longer to perform the computation $A - B$. Using an abstract representation of the adder from Fig 2.6, we can illustrate this modification here in Fig 3.1.

Figure 3.1: Constant Time Subtraction Circuit.



Briefly, the diagram represents with thick lines a group of wires corresponding to all digit inputs A, B and outputs C. The single NOT gate on the left abstractly represents the NOT operation for all input digits to A, and the single NOT gate on the right represents the same for all output digits of C. This circuit takes a total of 12 time steps and is therefore constant time.

4 Space Complexity

Given both the addition and subtraction circuits are nearly identical, it is sufficient to analyse the big O space complexity of the addition circuit alone. A carefree calculation of the space complexity identifies that the set of Y' components requires the largest amount of space, with all other components requiring some constant multiplied by n space. We have $O(n)$ Y' components, each with at most $O(n)$ COR operations on at most $O(n)$ elements. Thus while the number of COR gates in the set Y' is $O(n^2)$, we can take into account the wire complexity as inputs, thus obtaining an overall space complexity of $O(n^3)$.

While this space is not ideal, it falls within the models acceptance criteria of feasibility by not being an exponential space circuit, for larger n however, this model would not be practical. The author has currently given no further consideration towards an improvement in the space complexity.

5 References

[1] Zupan, Matthew, "Parallel Addition and Subtraction in $O(log_2(n))$ Time," 27, Nov. 2022.