A Logarithmic-Depth Quantum Carry-Lookahead Adder

Thomas G. Draper[*]   Samuel A. Kutin[†]   Eric M. Rains[‡]

Krysta M. Svore[§]

February 1, 2008

Abstract

We present an efficient addition circuit, borrowing techniques from the classical carry-lookahead arithmetic circuit. Our quantum carry-lookahead (QCLA) adder accepts two n-bit numbers and adds them in $O(\log n)$ depth using $O(n)$ ancillary qubits. We present both in-place and out-of-place versions, as well as versions that add modulo 2^n and modulo $2^n - 1$.

Previously, the linear-depth ripple-carry addition circuit has been the method of choice. Our work reduces the cost of addition dramatically with only a slight increase in the number of required qubits. The QCLA adder can be used within current modular multiplication circuits to reduce substantially the run-time of Shor's algorithm.

1 Introduction

With the advent of Shor's algorithms for prime factorization and the discrete logarithm problem, it is necessary to design efficient quantum arithmetic circuits. Previous quantum addition circuits include the quantum ripple-carry adder of Vedral, Barenco, and Ekert [7], which has recently been improved [1], and the transform adder [2]. Both of these approaches have depth linear in the number of input bits. We present a new adder whose depth is logarithmic in the number of input bits. The circuit size, and the number of ancillary qubits needed, are linear in the number of input bits.

Our technique is derived from classical methods that perform in time logarithmic in the number of input bits. The classical carry-lookahead (CLA) adder

[*]6013 Pontiac Street, Berwyn Heights, MD 20740. tgd@math.umd.edu

[†]Center for Communications Research, 805 Bunn Drive, Princeton, NJ 08540. kutin@idaccr.org

[‡]Mathematics, University of California, Davis, 1 Shields Avenue, Davis, CA 95616–8633. rains@math.ucdavis.edu

[§]Department of Computer Science, Columbia University, 1214 Amsterdam Avenue, New York, NY 10027–7003. kmsvore@cs.columbia.edu

[6, 5, 8] computes the carry bits in a tree-like structure, yielding a logarithmic-depth circuit. We can exploit this same structure to design a quantum CLA (QCLA) circuit to add two n-bit numbers in $O(\log n)$ depth. The QCLA adder works in $\mathbb{Z}$, and can be modified to add (mod 2^n) or (mod $2^n - 1$).

The theory of carry-lookahead addition has been known for fifty years [6], and has appeared in circuit design textbooks [4, pp. 158–161],[3, pp. 84–91]. Why, then, is this paper necessary? What are the challenges of adapting the CLA technique to a quantum circuit? There are several constraints we have to consider:

- Reversibility: We are limited to operations which do not destroy information.

- Erasure: If we use scratch space, we must explicitly erase it. We will not be able to take advantage of quantum interference if our circuit leaves extra information in scratch registers.

- Space-boundedness: We wish to minimize the use of ancillae.

- Bounded fan-out: At a given time, we can only use a wire as an input to a single quantum gate. To use multiple copies of a bit, we must explicitly perform a fan-out operation, which increases the size and depth of the circuit, and may increase the necessary space.

In Section 2.2, we discuss the classical theory of carry-lookahead addition; we then adapt this theory to the quantum setting in Sections 3 and 4. Next, in Section 5, we discuss various modified versions of the addition problem: how to add (mod 2^n) or (mod $2^n - 1$), how to compare, and how to take an incoming carry bit as input. The complexities of the various circuits are summarized in Table 1 on page 19. Finally, we close with some thoughts on future work.

2 Preliminaries

We first describe our notation throughout this paper. We then discuss the classical carry-lookahead adder.

2.1 Notation

We write the binary expansion of a number r as $r = r_{n-1}r_{n-2}\cdots r_0$, where r_0 is the low-order bit.

We generally represent negative numbers using *two's-complement arithmetic*, in which the bitwise complement r' is equal to $-r-1$. In Section 5.5, we consider *one's-complement arithmetic*, in which $r' = -r$. Note that, in this latter scheme, the all-zeros bit string and all-ones bit string both represent zero, so we have to be careful when designing reversible one's-complement arithmetic circuits.

In our circuit diagrams, time runs from left to right. We use the standard notation for quantum circuit operations: $\oplus$ for negation, and $\bullet$ for a control.

In this paper, our circuits are composed of NOT gates (also called negations), controlled-NOT (controlled-NOT) gates, and Toffoli gates. A controlled-NOT gate has a single control qubit connected to a NOT gate on the target qubit. A Toffoli gate has two qubits controlling the application of a NOT gate to the target qubit. Hence, all of our circuits are classical reversible circuits.

We will refer to the two inputs to our addition circuit as a and b. Our goal is to compute the sum s, either in place (on top of b) or out of place. We compute s by first finding c, the *carry bits*, such that $s = a \oplus b \oplus c$. (If one computes the sum using standard school-book addition, then c is the sequence of carries.)

We let $w(n)$ denote the number of ones in the binary expansion of n. We observe that

$$n - w(n) = \sum_{i=1}^{\infty} \left\lfloor \frac{n}{2^i} \right\rfloor. \tag{1}$$

We denote $\log_2$ simply by log.

2.2 Classical carry-lookahead addition

In this section, we describe the classical carry-lookahead addition circuit and our motivation for using a carry-lookahead structure. The CLA adder [5] sums two n-bit numbers in $O(\log n)$ depth. In this arithmetic circuit, partial information about the incoming carry bits is exploited to avoid a linear-time ripple-carry computation. The carry bit string can be computed using a tree structure to greatly reduce the number of required operations.

The key ingredient is the *carry status* on an interval, denoted $C[i, j]$. This status can take one of three values: k represents "kill," g represents "generate," and p represents "propagate." We begin with a discussion of the carry status $C[i, i+1]$.

Suppose we are adding a and b, and we have computed the carry bit c_i. The next carry bit c_{i+1} is the majority function $\text{MAJ}(a_i, b_i, c_i)$. The base case for this process, c_0, is assumed to be 0—see Section 5.2 for discussion of the more general problem where c_0 is an input bit.

When $a_i = b_i$, we can determine the carry bit c_{i+1} without knowing c_i. Specifically, if $a_i = b_i = 0$, then the outgoing carry bit c_{i+1} is automatically "killed" and we set $c_{i+1} = 0$; we say that $C[i, i+1] = k$. Similarly, if $a_i = b_i = 1$, then a carry bit is "generated" and $c_{i+1} = 1$ with carry status $C[i, i+1] = g$.

If $a_i \neq b_i$, then we cannot determine c_{i+1} without knowing c_i. In this case, the carry c_i is "propagated" and we set $c_{i+1} = c_i$ with carry status $C[i, i+1] = p$. Figure 1 summarizes this computation of the carry status.

Given $C[i-1, i]$ and $C[i, i+1]$, we can compute the carry status $C[i-1, i+1]$. The calculation is shown in Figure 2; we use ⊛ to denote the carry status operator. If $C[i-1, i+1] = k$, then either a carry is killed at position i, or it would be propagated at position i but has been killed at position $i-1$. Either way, if $C[i-1, i+1] = k$, we know that $c_{i+1} = 0$. Similarly, if $C[i-1, i+1] = g$, we know that $c_{i+1} = 1$. If $C[i-1, i+1] = p$, we conclude that $c_{i+1} = c_{i-1}$.

a_i	b_i	c_{i+1}	$C[i, i+1]$
0	0	0	k
0	1	c_i	p
1	0	c_i	p
1	1	1	g

Figure 1: The carry status of a_i and b_i.

			$C[i, i+1]$	
	⊛	k	p	g
	k	k	k	g
$C[i-1, i]$	p	k	p	g
	g	k	g	g

Figure 2: The carry status assignments $C[i-1, i+1]$ given the previous and current carry status values.

The carry status operator ⊛ shown in Figure 2 allows us to merge intervals: for any k satisfying $i < k < j$,

$$C[i, j] = C[i, k] \circledast C[k, j].$$

The choice of k does not affect the answer, since ⊛ is associative. By successively doubling the sizes of intervals, we can use this approach to compute $C[i, j]$ for any i, j in logarithmic depth.

We now describe the computation of the carry bits in detail. Since $C[i, j]$ can take three values, we must specify an encoding of $C[i, j]$ in bits. We define $p[i, j]$ to be 1 when $C[i, j] = \text{p}$, and we define $g[i, j]$ to be 1 when $C[i, j] = \text{g}$. The relationship between $C[i, j]$, $p[i, j]$, and $g[i, j]$ is depicted in Figure 3. Note that, in particular, we never have $p[i, j] = g[i, j] = 1$.

$C[i, j]$	$p[i, j]$	$g[i, j]$
k	0	0
g	0	1
p	1	0

Figure 3: The carry status $C[i, j]$ encoded in two bits $p[i, j]$ and $g[i, j]$.

For any i, j, $p[i, j]$ is 1 if a carry propagates from bit position i to bit position j, and 0 otherwise. Note that this occurs if and only if $a_\ell \oplus b_\ell = 1$ whenever $i \leq \ell < j$. For any k between i and j, a carry bit is propagated from bit i to bit j if a carry bit is propagated from i to bit k, and then also propagated from bit k to bit j. Thus, the computation of the propagate bits, for any $i < k < j$, is

$$p[i, j] = p[i, k] \wedge p[k, j]. \tag{2}$$

4

Next, we consider $g[i, j]$. This quantity is 1 when a carry is generated between bit positions i and j. The computation of the generate bits, for $i < k < j$, is

$$g[i, j] = g[k, j] \vee (g[i, k] \wedge p[k, j])$$
$$= g[k, j] \oplus (g[i, k] \wedge p[k, j]). \tag{3}$$

That is, either a carry bit is generated between bits k and j, or a carry bit is generated between bits i and k and then propagated from bit k to bit j. The second expression follows from the observation that $g[k, j]$ and $p[k, j]$ cannot both be equal to 1.

For all $j > 0$, $p[0, j]$ is 0, and $g[0, j]$ is the carry bit c_j. By successively doubling the sizes of the intervals under consideration, we can compute all carry bits in logarithmic depth.

3 Reversible computation of carry status

We are now ready to build a quantum adder using the CLA technique. We first explain how we can compute the carry status reversibly.

The circuit of this section has two input arrays, each of length n: P_0, initialized to $P_0[i] = p[i, i+1]$, and G, initialized to $G[i] = g[i-1, i]$. Note that the array P_0 is 0-based, but the array G is 1-based. We also use $n - w(n) - \lfloor \log n \rfloor$ ancillary bits, initialized to zero.

At the end of the computation, we want $G[i] = g[0, i] = c_i$. We also need to erase our scratch work: we must ensure that, when we're done, $P_0[i] = p[i, i+1]$ and the ancillary bits are reset to zero.

We will have roughly $\lfloor \log n \rfloor$ rounds each of four different types:

1. P-rounds: Compute $p[i, j]$ values into the ancillary space.

2. G-rounds: Set $G[j] = g[i, j]$; for each j, we choose a particular i value.[1]

3. C-rounds: Set $G[j] = c_j$.

4. P^{-1}-rounds: Erase the work done in the P-rounds.

We first describe the sequence of gates, and then we compute the circuit depth.

In P-round t, we compute all p-values of the form $p[i, j]$ where $i = 2^t m$ and $j = i + 2^t$. We refer to these values as $P_t[m]$, for $1 \le m < \lfloor n/2^t \rfloor$. We store these $\lfloor n/2^t \rfloor - 1$ values in our ancillary space. By (1), the total space needed for all of the P-rounds is $n - w(n) - \lfloor \log n \rfloor$ bits. We do not need to compute values of the form $p[0, 2^t]$, since no carry is generated at 0; in particular, when $t = \lfloor \log n \rfloor$, no computation is done.

We compute $p[i, j]$ using (2), with $k = 2^t m + 2^{t-1}$. Note that both $p[i, k]$ and $p[k, j]$ were computed in P-round $(t - 1)$, so we can write $p[i, j]$ to the

---

[1] We have $i = (j-1) \wedge j$, where $-$ denotes subtraction in $\mathbb{Z}$ and $\wedge$ denotes bitwise AND.

appropriate location using one Toffoli gate. The total number of gates is thus $n - w(n) - \lfloor \log n \rfloor$.

In G-round t, we compute all g-values of the form $g[i, j]$ where $i = 2^t m$ and $j = i + 2^t$. We store this value in the location $G[j]$. We use (3), with $k = 2^t m + 2^{t-1}$. Since $g[k, j]$ is already in location $G[j]$ after G-round $(t-1)$, we can do this computation with a single Toffoli gate, combining $g[i, k]$ (computed in G-round $(t-1)$) and $p[k, j]$ (computed in P-round $(t-1)$). The total number of gates is $n - w(n)$.

In C-round t, we compute all g-values of the form $g[0, j]$ with $j = 2^t m + 2^{t-1}$. We begin with the maximum t for which some j exists, $t = \lfloor \log \frac{2n}{3} \rfloor = 1 + \lfloor \log \frac{n}{3} \rfloor$, and work our way down to $t = 1$. We use (3) with $k = 2^t m$. Since $g[k, j]$ is already in location $G[j]$, we again need just one Toffoli gate: we require $p[k, j]$ (computed in P-round $(t-1)$) and $g[0, k]$ (computed in C-round $(t+1)$ or earlier). The total number of gates is $n - \lfloor \log n \rfloor - 1$.

Finally, in the P^{-1}-rounds, we simply repeat the same Toffolis as in the P-rounds, in reverse order.

In summary, we must perform the following steps:

1. P-rounds. For $t = 1$ to $\lfloor \log n \rfloor - 1$: for $1 \leq m < \lfloor n/2^t \rfloor$:

$$P_t[m] \oplus= P_{t-1}[2m] P_{t-1}[2m + 1].$$

2. G-rounds. For $t = 1$ to $\lfloor \log n \rfloor$: for $0 \leq m < \lfloor n/2^t \rfloor$:

$$G[2^t m + 2^t] \oplus= G[2^t m + 2^{t-1}] P_{t-1}[2m + 1].$$

3. C-rounds. For $t = \lfloor \log \frac{2n}{3} \rfloor$ down to 1: for $1 \leq m \leq \lfloor (n - 2^{t-1})/2^t \rfloor$:

$$G[2^t m + 2^{t-1}] \oplus= G[2^t m] P_{t-1}[2m].$$

4. P^{-1}-rounds. For $t = \lfloor \log n \rfloor - 1$ down to 1: for $1 \leq m < \lfloor n/2^t \rfloor$:

$$P_t[m] \oplus= P_{t-1}[2m] P_{t-1}[2m + 1].$$

The circuit consists of

$$4n - 3w(n) - 3\lfloor \log n \rfloor - 1 \tag{4}$$

Toffoli gates.

It would seem that the circuit described above would require roughly $4 \log n$ time-slices. However, we can overlap some of the computation.

We start with P-round 1, which uses the arrays P_0 and P_1. Then, P-round 2 uses the arrays P_1 and P_2. Note that G-round 1 uses the arrays G and P_0; hence, we can run G-round 1 in the same time-slice as P-round 2. In general, we can run P-round $(t + 1)$ and G-round t in parallel.

Similarly, once we have run C-round t, we are done using P_{t-1}. While we run C-round $(t - 1)$, we can run P^{-1}-round t, which uses P_{t-1} to erase P_t. We

run C-round 1 in parallel with P-round 2; we then need one additional time-slice to run P-round 1.

So, the circuit has a depth of

$$\lfloor \log n \rfloor + \left\lfloor \log \frac{n}{3} \right\rfloor + 3. \tag{5}$$

For $n \leq 3$, expression (5) overcounts the depth, since there are no P-rounds.

4 The complete quantum addition circuit

We are now ready to describe our quantum carry-lookahead addition circuit. We first discuss the out-of-place version in Section 4.1, and then the in-place version in Section 4.2.

The out-of-place version produces $n+1$ bits of output, and uses $n - w(n) - \lfloor \log n \rfloor$ ancillae. The depth is $2 \log n + O(1)$ and the size is $8n - O(\log n)$ gates.

The in-place version produces 1 bit of output, and uses $2n - w(n) - \lfloor \log n \rfloor - 1$ ancillae. The depth is $4 \log n + O(1)$ and the size is $16n - O(\log n)$ gates.

Table 1 on page 19 summarizes the complexities of these two adders, as well as the variants discussed in Section 5.

4.1 Addition out of place

We would like to add two n-bit numbers, a and b, stored in arrays A and B. We need $n+1$ bits for the output, denoted by Z, and $n - w(n) - \lfloor \log n \rfloor$ ancillary bits, denoted by X. We assume that Z and X are initialized to zero. In the end, we want Z to contain the quantity $s = a + b$.

The key relation is that the sum s is equal to $a \oplus b \oplus c$, where c is the carry string. Hence, the key step in our algorithm is to compute c, using the technique of the previous section. We compute the carry string c_1 through c_n into the bits $Z[1]$ through $Z[n]$.

The out-of-place QCLA adder proceeds as follows:

1. For $0 \leq i < n$, $Z[i+1] \oplus= A[i]B[i]$. This sets $z_{i+1} = g[i, i+1]$.

2. For $1 \leq i < n$, $B[i] \oplus= A[i]$. This sets $B[i] = p[i, i+1]$ for $i > 0$, which is what we need to run our addition circuit.

3. Run the circuit of Section 3, using X as ancillary space. Upon completion, $Z[i] = c_i$ for $i \geq 1$.

4. For $0 \leq i < n$, $Z[i] \oplus= B[i]$. Now, for $i > 0$, $Z[i] = a_i \oplus b_i \oplus c_i = s_i$. For $i = 0$, we have $Z[i] = b_i$.

5. Set $Z[0] \oplus= A[0]$. For $1 \leq i < n$, $B[i] \oplus= A[i]$. This fixes $Z[0]$, and resets B to its initial value.
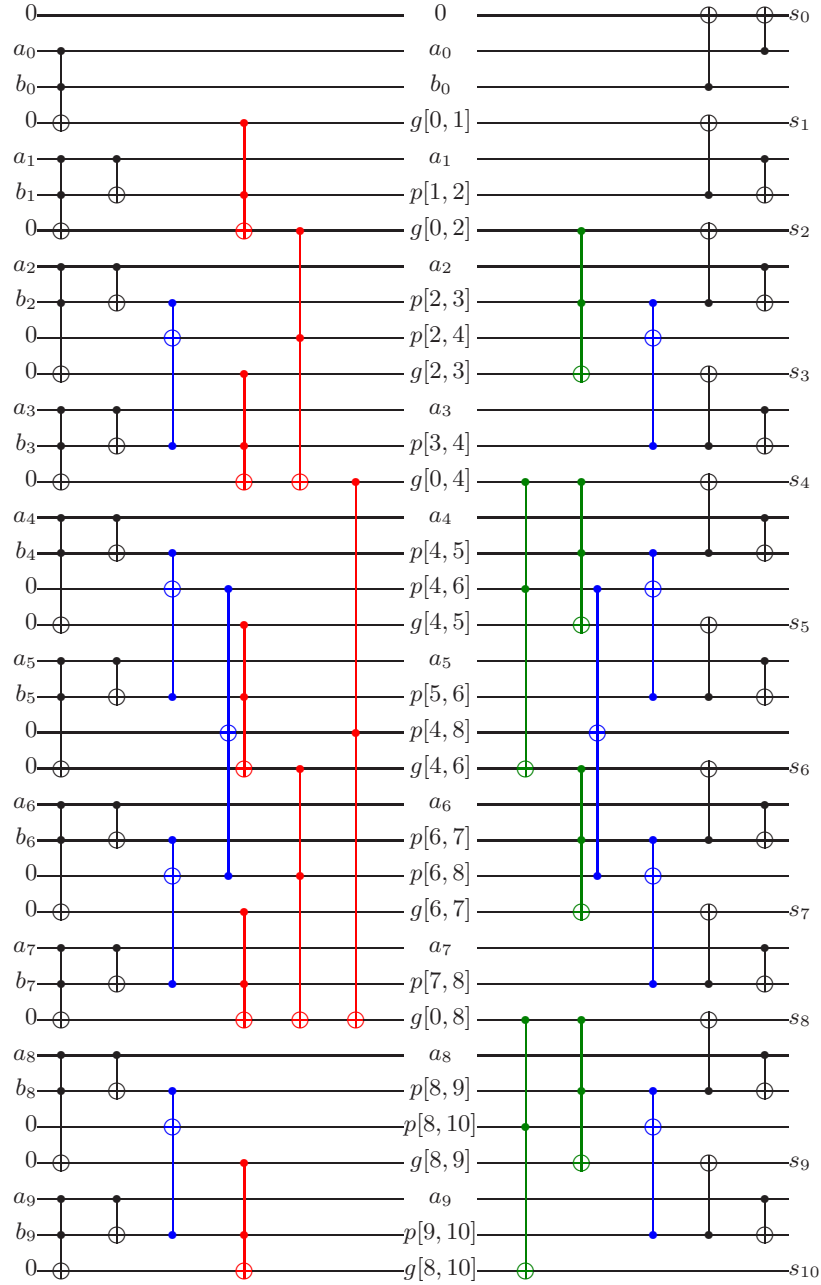
Figure 4: Out-of-place QCLA adder for 10 bits. P-rounds and P^{-1}-rounds are shown in blue. G-rounds are red, and C-rounds are green.

Aside from Step 3, each step occurs in a single time slice. So, by (5), the overall depth of the circuit is

$$\lfloor \log n \rfloor + \left\lfloor \log \frac{n}{3} \right\rfloor + 7,$$

where three of the time-slices contain controlled-NOTs and the rest contain Toffolis. For $n \leq 3$, the depth is slightly lower.

By (4), the circuit contains

$$5n - 3w(n) - 3 \lfloor \log n \rfloor - 1$$

Toffoli gates and $3n - 1$ controlled-NOTs.

The circuit for $n = 10$ is depicted in Figure 4.

4.2   Addition in place

For the in-place circuit, we begin the same way as above: we compute the carry string c into $n-1$ ancillary bits (plus one output bit for the high bit). The total ancillary space required is $2n - w(n) - \lfloor \log n \rfloor - 1$. We then write the low n bits of the sum on top of b. The key new step is the erasure of the low $n - 1$ bits of the carry string c.

Recall from Section 2.1 that we are using two's-complement arithmetic:

$$r' + r \equiv -1 \quad (\mathrm{mod}\ 2^n).$$

So, writing $s = a + b$,

$$a + s' \equiv a - a - b - 1 \equiv b' \quad (\mathrm{mod}\ 2^n).$$

Let d be the carry string generated by a and s'. We have

$$a \oplus s' \oplus d = b'$$
$$a \oplus (a \oplus b \oplus c) \oplus (-1) \oplus d = b \oplus (-1)$$
$$c = d.$$

So the carry string d, generated by adding a and s', is simply c. After we compute s, we can complement it, and then run the circuit of Section 3 in reverse to erase c.

The in-place QCLA adder proceeds as follows. We denote the $n - 1$ ancillae which store the carry string as $Z[1], \ldots, Z[n-1]$, and the remaining ancillae as X. The output bit is labeled $Z[n]$.

1. For $0 \leq i < n$, $Z[i+1] \oplus= A[i]B[i]$. This sets $Z[i+1] = g[i, i+1]$.

2. For $0 \leq i < n$, $B[i] \oplus= A[i]$. This sets $B[i] = p[i, i+1]$ for $i > 0$. Also, $B[0] = s_0$.

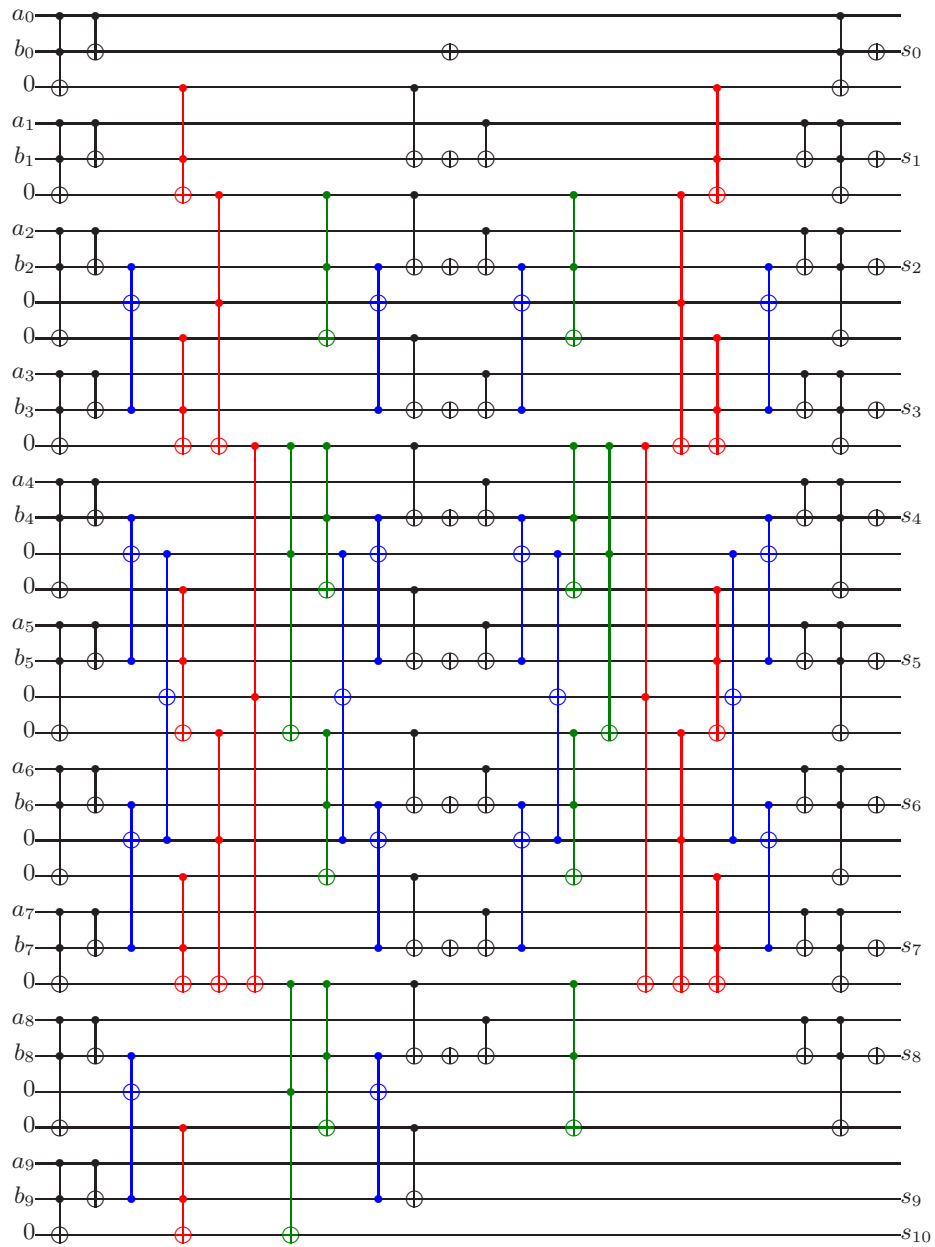3. Run the circuit of Section 3, using X as ancillary space. Upon completion, $Z[i] = c_i$ for $i \geq 1$.

9

Figure 5: In-place QCLA adder for 10 bits. P-rounds and P^{-1}-rounds are shown in blue. G-rounds are red, and C-rounds are green.

4. For $1 \leq i < n$, $B[i] \oplus= Z[i]$. Now $B[i] = s_i$.

5. For $0 \leq i < n - 1$, negate $B[i]$. Now B contains s'.

6. For $1 \leq i < n - 1$, $B[i] \oplus= A[i]$.

7. Run the circuit of Section 3 in reverse.[2] Upon completion, $Z[i+1] = a_i s'_i$ for $0 \leq i < n - 1$, and $B[i] = a_i \oplus s'_i$ for $1 \leq i < n$.

8. For $1 \leq i < n - 1$, $B[i] \oplus= A[i]$.

9. For $0 \leq i < n - 1$, $Z[i+1] \oplus= A[i]B[i]$.

10. For $0 \leq i < n - 1$, negate $B[i]$.

Each step other than 3 and 7 has depth 1. By (5), the overall depth is

$$\lfloor \log n \rfloor + \lfloor \log(n-1) \rfloor + \left\lfloor \log \frac{n}{3} \right\rfloor + \left\lceil \log \frac{n-1}{3} \right\rceil + 14,$$

where two of the time-slices contain negations, four contain controlled-NOTs, and the rest contain Toffolis. For some values of $n \leq 6$, the depth is slightly lower.

By (4), the circuit contains

$$10n - 3w(n) - 3w(n-1) - 3\lfloor \log n \rfloor - 3\lfloor \log(n-1) \rfloor - 7$$

Toffoli gates, $4n - 5$ controlled-NOTs, and $2n - 2$ negations.

In Figure 5, we show a sample in-place QCLA adder for the case $n = 10$.

5   Extensions

We now discuss various modified versions of the circuit. The simplest is one which adds (mod 2^n); we simply skip the computation of the high bit. With slightly more work, we can add (mod $2^n - 1$).

There are other constructions which use the log-depth adder as a subroutine. For these, it is useful to allow the adder to take one additional bit, an incoming carry. In this case, we wish to compute $a + b + y$, where y is either 0 or 1.

Another useful subroutine in an addition (or modular addition) circuit is comparison: Is $a \geq b$? Equivalently, is the high bit of $a - b$ zero? We discuss how one can use the log-depth adder to subtract, and we show that a comparator is of comparable complexity to an out-of-place adder.

---

[2] In Step 7, we actually reverse the $(n-1)$-bit adder, since we should not erase the high carry bit. See Section 5.1 for more discussion.

5.1 Addition (mod 2^n)

It is straightforward to add (mod 2^n); we simply do not compute the high bit of the sum. The only question is: what are the exact savings, in depth and circuit size?

Since we do not need to compute c_n, we can simply run the circuit of Section 3 on the low-order $n-1$ bits of a and b. For the out-of-place adder, this circuit leaves c_{n-1} in $Z[n-1]$, so we also need to apply the gates $Z[n-1]\oplus=a_{n-1}$ and $Z[n-1]\oplus=b_{n-1}$; we therefore add two additional controlled-NOTs. For $n > 1$, this does not increase the depth.

So, the out-of-place (mod 2^n) adder produces n output bits, and uses $(n-1)-w(n-1)-\lfloor\log(n-1)\rfloor$ ancillae. The depth is $\lfloor\log(n-1)\rfloor+\lfloor\log\frac{n-1}{3}\rfloor+7$ when $n \geq 4$, and the circuit consists of $5n-3w(n-1)-3\lfloor\log(n-1)\rfloor-6$ Toffolis and $3n-2$ controlled-NOTs.

For the in-place adder, we follow the steps in Section 4.2. However, in Step 1, our loop now stops at $i = n-2$, and, in Step 3, we run the $(n-1)$-bit adder.

Thus, the in-place (mod 2^n) adder uses $2n-2-w(n-1)-\lfloor\log(n-1)\rfloor$ ancillae. The depth is $2\lfloor\log(n-1)\rfloor+2\lfloor\log\frac{n-1}{3}\rfloor+14$ when $n \geq 5$, and the circuit size is $10n-6w(n-1)-6\lfloor\log(n-1)\rfloor-12$ Toffolis, $4n-5$ controlled-NOTs, and $2n-2$ negations.

5.2 Addition with incoming carry

Suppose we want our adder to take $2n+1$ bits of input: a, b, and a single bit y, representing an incoming carry. This is useful in various hybrid addition circuits, where we break the problem up into smaller pieces.

We can accomplish this by adding the $(n+1)$-bit numbers $2a+y$ and $2b+y$, whose sum is $2(a+b+y)$. So, the cost is roughly the same as that of an $(n+1)$-bit add. However, we use fewer operations on the low-order bit—we simply start with $c_1 = y$. The additional input bit y replaces one output bit for the out-of-place adder, and one ancillary bit for the in-place adder.

For the out-of-place adder, we save one Toffoli and two controlled-NOTs over the usual $(n + 1)$-bit adder. For the in-place adder, we save two Toffolis, one controlled-NOT, and two negations.

The same analysis applies to the (mod 2^n) adder of Section 5.1.

5.3 Subtraction

It is straightforward to use our circuit to compute $a - b$. First, complement all bits of a. Then, add as usual; we compute $a' + b$. At the end, complement all bits of a and all output bits. The result, assuming two's-complement arithmetic, is then

$$(a' + b)' = (-a - 1 + b)' = a - b.$$

A similar argument holds for one's-complement arithmetic.

Hence, the cost of subtraction is essentially the same as the cost of addition. We add two time-slices, both consisting only of negations.

5.4 Comparison

Suppose we wish to compare two numbers a and b. We compute the high bit of $a - b$. As in the subtractor, we first complement a. We then run the QCLA adder forward until we have found the high bit of $a' + b$, and then we reverse the preceding computation. This gives us a QCLA comparator.

If $n = 2^k$ for some k, then the above idea works well; we find the high bit at the end of the G-rounds, halfway through the out-of-place add. However, for $n = 2^k - 1$, we do not compute the high bit until after we're done with the C-rounds. If we just use this simple approach, the depth of our circuit turns out to be $2\lfloor \log n \rfloor + 2w(n) + 5$. We would prefer to design a comparator which has depth $2\log n + O(1)$.

So, we have to be more careful. Let $k = \lceil \log n \rceil$. If we just do and undo the P-rounds and G-rounds, we can compare two 2^k-bit numbers in depth roughly $2k$. So, we can pad our n-bit numbers by adding zeros to the front, and then use the compare circuit for 2^k-bit numbers.

After we complement a, we will have $p[i, j] = 1$ for $j > i \geq n$ and $g[i, j] = 0$ for $j > i \geq n$. We do not explicitly compute these values in our circuit; effectively, we compile the values into the circuit.

Overall, the comparator uses $2n - \lfloor \log(n-1) \rfloor - 3$ ancillae. For the explicit discussion, we suppose our input is stored in the n-long bit arrays A and B. We have $n - 1$ ancillary bits denoted $Z[1], \ldots, Z[n-1]$, and $n - \lfloor \log(n-1) \rfloor - 2$ additional ancillae denoted by X. The output bit is denoted $Z[n]$. We proceed as follows:

1. For $0 \leq i < n$, negate $A[i]$.

2. For $0 \leq i < n$, $Z[i+1] \oplus= A[i]B[i]$.

3. For $1 \leq i < n$, $B[i] \oplus= A[i]$.

4. Do the P-rounds for the 2^k-bit adder using space X; write only the values we cannot deduce at compile-time.

5. Do the G-rounds for the 2^k-bit adder; apply only those gates which affect $Z[n]$.

6. Undo the G-round gates which did not write to $Z[n]$.

7. Do the P^{-1}-rounds for the 2^k-bit adder, erasing X.

8. For $1 \leq i < n$, $B[i] \oplus= A[i]$.

9. For $0 \leq i < n - 1$, $Z[i+1] \oplus= A[i]B[i]$.

10. For $0 \leq i < n$, negate $A[i]$. Also negate $Z[n]$.

Step 5 contains $n - 1$ Toffoli gates, in depth $\lfloor \log(n-1) \rfloor + 1$. Step 6 is equivalent to inverting the G-rounds for an $(n-1)$-bit adder, and contains $n - w(n-1) - 1$ Toffoli gates in depth $\lfloor \log(n-1) \rfloor$.

Steps 4 and 7 each consist of $n - \lfloor \log(n-1) \rfloor - 2$ gates. In each case, the depth would be $\lfloor \log(n-1) \rfloor$, but each P-round after the first (and each P^{-1}-round before the last) can be done in parallel with a G-round.

The total depth for the comparator is

$$2 \lfloor \log(n-1) \rfloor + 9,$$

where two of the time-slices contain negations, two contain controlled-NOTs, and the rest are Toffolis. The overall circuit size is

$$6n - 2 \lfloor \log(n-1) \rfloor - w(n-1) - 7$$

Toffoli gates, $2n - 2$ controlled-NOT gates, and $2n + 1$ negations. When $n \leq 4$, we have slightly overcounted the depth and size. A sample comparison circuit for $n = 7$ appears in Figure 6.

If we wish to allow an incoming carry, we use the same technique as in Section 5.2. We use an $(n+1)$-bit comparator, except that the carry input replaces one of the ancillae, and we save two negations and two Toffolis.

It may seem strange that an n-bit compare would require more gates than an n-bit out-of-place add. After all, we're solving a simpler problem; we want one bit of the $(n+1)$-bit answer.

One way to look at this phenomenon is that, when we compute the high bit of the sum, we are effectively using other output bits as ancillary space. If we're "only" doing a compare, we need extra gates to erase this space. One explicit example of this is Step 9 of the compare, where we erase the generate array. For the out-of-place add, the generate array has turned into our answer, and need not be erased.

5.5   Addition (mod $2^n - 1$)

Recall from Section 2.1 that we have been working in two's-complement arithmetic, where $r' + r = -1$. With a slight increase in depth, we can modify our circuit to work in one's-complement arithmetic, where $r' + r = 0$. Equivalently, we can view one's-complement addition as addition (mod $2^n - 1$). This may prove useful for some applications, particularly when $2^n - 1$ is prime.

Note that, in one's-complement arithmetic, 0 can be represented either by the all-zeros bit string or the all-ones bit string. For in-place reversible computation, we cannot have $a + \vec{0} = a$ and $a + \vec{1} = a$. We will first describe our adder in general terms, and then discuss how we can handle this zero problem.

First, consider the computation of c_0. In the one's-complement setting, we can no longer assume c_0 to be 0; the low bit of the sum is affected by whether or not we have an overflow. We have an overflow if and only if $a + b \geq 2^n$; hence, we get $c_0 = g[0, n]$.

If $n = 2^k$ for some k, then we have computed c_0 at the end of the G-rounds. How do we compute the other carry bits? One approach follows from the cyclic invariance of addition (mod $2^n - 1$): We note that multiplication by 2^j corresponds to a cyclic shift by j. So, if we could simultaneously add at all
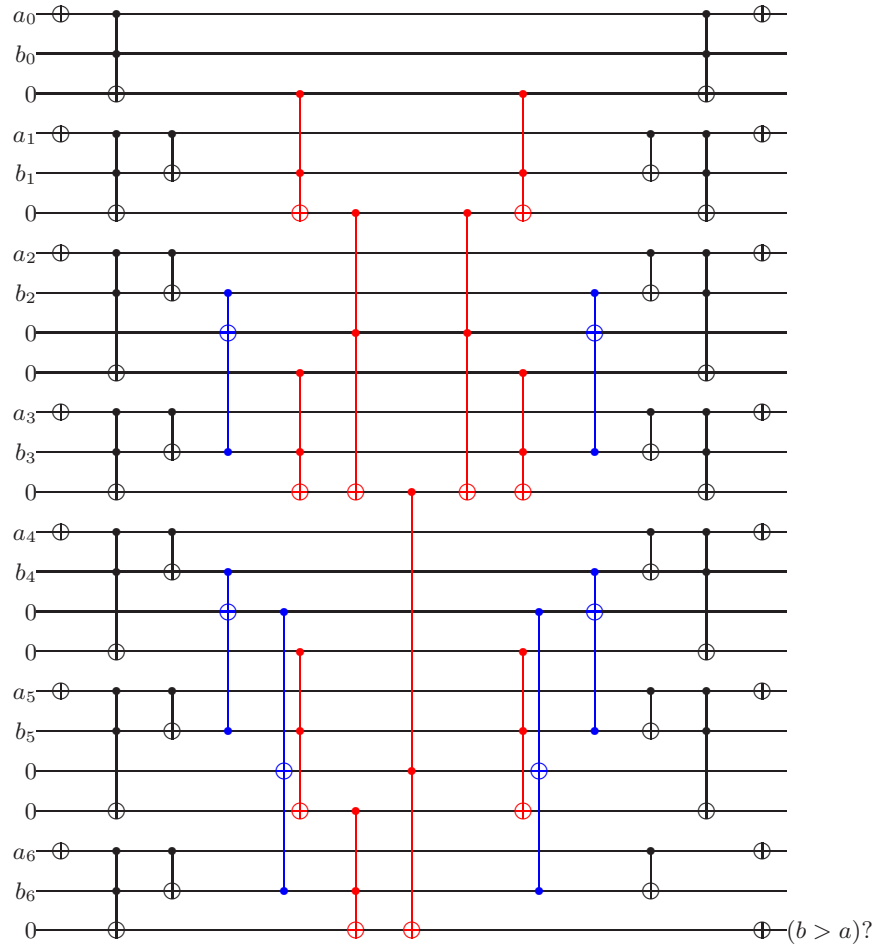
Figure 6: QCLA comparator for 7 bits. P-rounds and P^{-1}-rounds are shown in blue; G-rounds and G^{-1}-rounds are red.

15

possible cyclic shifts, we would compute all of the carry bits. This approach would have logarithmic depth, but would require $\Theta(n \log n)$ ancillary space.

A second idea is to view c_0 as an incoming carry $g[-\infty, 0]$. Our carry string is then given by $c_i = g[-\infty, i]$. Another way to look at this identity is that we are wrapping around: to compute c_i, we start at the zero position, work up to n, and then wrap back around and keep going up to i. This is the same as the cyclic shift, except that we are counting one region twice; it is not hard to see that this cannot affect our overall answer.

To do this wrap-around, we will need propagate bits of the form $p[0, 2^t]$. After we complete the P-rounds and G-rounds, we have computed $c_0 = g[-\infty, 0]$. We now add a new round: using c_0 and $p[0, 2^{k-1}]$, we use one Toffoli gate to compute $c_{2^{k-1}} = g[-\infty, 2^{k-1}]$. From here on, we do our usual C-rounds, except that each contains one extra gate computing c_{2^t}. Upon completion, we have successfully computed the carry string.

If n is not a power of 2, we need to do a bit more work to make sure we compute c_0 at the end of the G-rounds. We use the same technique as in Section 5.4.

5.5.1 Out-of-place addition (mod $2^n - 1$)

The above description, combined with the general approach in Section 4.1, yields an out-of-place one's-complement adder. We produce n bits of output, and use $n - 2$ ancillae.

The overall depth is

$$2 \lfloor \log(n - 1) \rfloor + 8,$$

where three of the time-slices contain controlled-NOTs and the rest are Toffolis. For $n \leq 2$, the depth is slightly lower.

The circuit contains $5n - 6$ Toffoli gates and $3n$ controlled-NOT gates.

Suppose we use the above circuit to add two numbers a and b which sum to $2^n - 1$. Since $a \oplus b = \vec{1}$, the generate array will be initialized to $\vec{0}$ and the propagate array to $\vec{1}$. Hence the carry string will be $\vec{0}$, and the sum will be output as $\vec{1}$. Hence, we say that this circuit uses the $\vec{1}$ representation of zero. One can check that, if one of the inputs is $\vec{1}$, the circuit also adds correctly.[3]

It might seem more natural to represent zero as $\vec{0}$. We can modify the out-of-place circuit as follows: at the end of the P-rounds, we XOR $p[0, n]$ into c_0 (this requires one additional Toffoli gate). So, after the G-rounds, we will have $c_0 = g[0, n] \oplus p[0, n]$. We then compute the C-rounds as before. Now, if $a \oplus b = \vec{1}$, the circuit will output $\vec{0}$ as the sum; we have thus given a circuit which uses the $\vec{0}$ representation of zero. Again, we can check that, if one or both inputs are $\vec{0}$, the circuit performs correctly.

The out-of-place one's-complement adder using the $\vec{0}$ representation requires $n - 2$ ancillae. The circuit contains $5n - 5$ Toffoli gates and $3n$ controlled-NOT gates. For some n, the depth goes up by one; it depends on whether

---

[3]In fact, this circuit is also correct when exactly one of the inputs is $\vec{0}$. But, if both inputs are $\vec{0}$, we output $\vec{0}$ rather than $\vec{1}$.

the computation of $p[0, n]$ can be done simultaneously with the penultimate G-round. For $n \geq 4$, the depth can be written as

$$\lfloor \log(n-1) \rfloor + \left\lfloor \log \frac{n-1}{3} \right\rfloor + 10,$$

where three of the time-slices contain controlled-NOTs and the rest are Toffolis.

5.5.2 In-place addition (mod $2^n - 1$)

Following Section 4.2, we next construct an in-place one's-complement adder. We require $2n - 2$ ancillary bits: $n - 2$ for computing the propagate bits, and n for the carry string. We first compute the carry string into our ancillae, and then write the sum on top of b. Next, to erase the carry string, we negate b, undo the addition computation, and fix b at the end.

As in the out-of-place version, we need to be careful about the representation of zero. The complementation of b introduces a slight wrinkle: if we do our first addition using $\vec{0}$ to represent zero, then we need to undo a $\vec{1}$-based addition. If we do our first addition using the $\vec{1}$ circuit, we need to undo the $\vec{0}$ circuit.

Hence, regardless of whether we represent zero by $\vec{0}$ or $\vec{1}$, the cost of the circuit is the same: we require $2n$ negations, $4n$ controlled-NOTs, and $10n - 11$ Toffolis. For $n \geq 4$, the depth is:

$$3 \lfloor \log(n-1) \rfloor + \left\lfloor \log \frac{n-1}{3} \right\rfloor + 18,$$

where two of the time-slices contain negations, four contain controlled-NOTs, and the rest contain Toffolis.

Figure 7 depicts a sample in-place one's-complement QCLA adder for the case $n = 7$.

6 Conclusions and future work

In conclusion, we have developed an efficient addition circuit using classical carry-lookahead techniques. Our QCLA adder sums two n-bit numbers in-place using $2n - w(n) - \lfloor \log n \rfloor - 1$ ancillary qubits in depth $4 \log n + O(1)$. This improves upon the previous best known addition circuits, which require linear depth. Our work dramatically improves the run-time of the arithmetic circuits required in Shor's algorithm.

The complexities of the various circuits in this paper are summarized in Tables 1 and 2. In Table 1, we assume $n = 2^k$; in Table 2, we give the general formulas. For simplicity, we count only Toffoli gates and Toffoli time-slices. Since some of the formulas are incorrect for small n, we assume $n \geq 7$. We also include two different ripple-carry adders [7, 1].

It would be interesting to apply a similar tree-like approach to other arithmetic problems, such as modular addition and multiplication. It would also be interesting to build a logarithmic-depth addition circuit using only $O(\log n)$ ancillae, or to prove that no such classical reversible circuit exists.
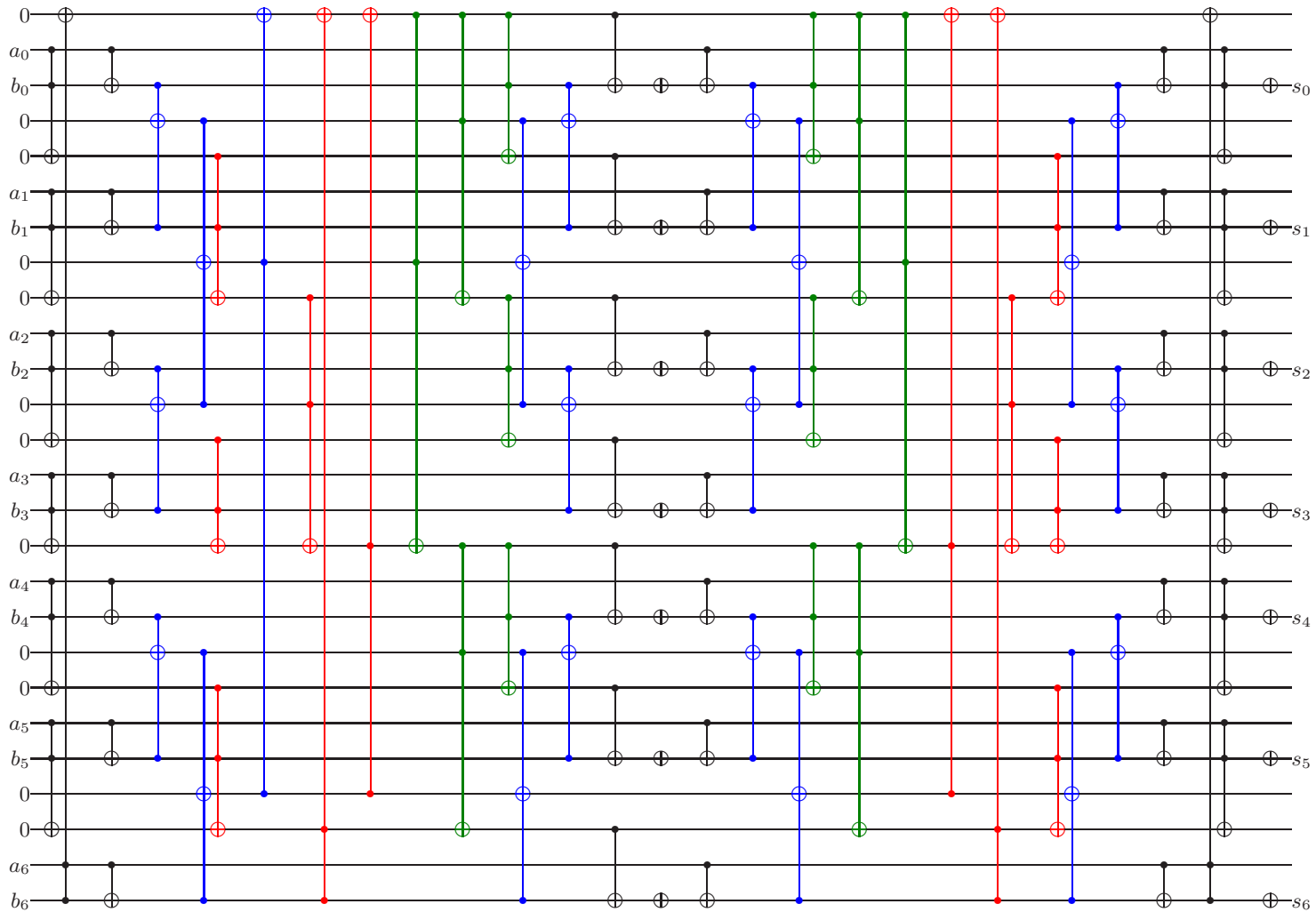
Figure 7: In-place QCLA adder (mod $2^n - 1$) for $n = 7$. P-rounds and P^{-1}-rounds are shown in blue. G-rounds are red, and C-rounds are green. This adder uses the $\vec{0}$ representation of zero. Note that the extra gate writing $p[0, n]$ to c_0 is present only during the first half of the computation.

18

Function	IP?	IC?	Input	Output	Ancillae	Size	Depth
$+$ in $\mathbb{Z}$	N	N	$2n$	$n+1$	$n-k-1$	$5n-3k-4$	$2k+2$
$+$ in $\mathbb{Z}$	N	Y	$2n+1$	$n+1$	$n-k-1$	$5n-3k-3$	$2k+2$
$+$ in $\mathbb{Z}$	Y	N	$2n$	1	$2n-k-2$	$10n-9k-7$	$4k+3$
$+$ in $\mathbb{Z}$	Y	Y	$2n+1$	1	$2n-k-2$	$10n-6k-8$	$4k+4$
$+$ $(\mathrm{mod}\ 2^n)$	N	N	$2n$	n	$n-2k$	$5n-6k-3$	$2k+1$
$+$ $(\mathrm{mod}\ 2^n)$	N	Y	$2n+1$	n	$n-k-1$	$5n-3k-5$	$2k+2$
$+$ $(\mathrm{mod}\ 2^n)$	Y	N	$2n$	0	$2n-2k-1$	$10n-12k-6$	$4k+2$
$+$ $(\mathrm{mod}\ 2^n)$	Y	Y	$2n+1$	0	$2n-k-2$	$10n-6k-10$	$4k+4$
$+$ $(\mathrm{mod}\ 2^n-1)$	N	—	$2n$	n	$n-2$	$5n-6$	$2k+3$
$+$ $(\mathrm{mod}\ 2^n-1)$	Y	—	$2n$	0	$2n-2$	$10n-11$	$4k+7$
Comparison	—	N	$2n$	1	$2n-k-2$	$6n-3k-5$	$2k+5$
Comparison	—	Y	$2n+1$	1	$2n-k-2$	$6n-2k-4$	$2k+5$
Ripple-carry [7]	Y	N	$2n$	1	n	$4n-2$	$3n-1$
Ripple-carry [1]	Y	Y	$2n+1$	1	0	$2n-1$	$2n-1$

Table 1: Circuit summary, for $n = 2^k$, where $k \geq 3$. The first column gives the function being computed. The second lists whether the computation is done in place, and the third lists whether we take an incoming carry bit as input. We then list the number of input, output, and ancillae bits, and the circuit size and depth. For the purposes of this table, we only count Toffoli gates.

Function	IP?	IC?	Input	Output	Ancillae	Size	Depth
+ in $\mathbb{Z}$	N	N	$2n$	$n+1$	$n - w(n) - \lfloor\log n\rfloor$	$5n - 3w(n) - 3\lfloor\log n\rfloor - 1$	$\lfloor\log n\rfloor + \lfloor\log\frac{n}{3}\rfloor + 4$
+ in $\mathbb{Z}$	N	Y	$2n+1$	$n+1$	$n - w(n+1) - \lfloor\log(n+1)\rfloor + 1$	$5n - 3w(n+1) - 3\lfloor\log(n+1)\rfloor + 3$	$\lfloor\log(n+1)\rfloor + \lceil\log\frac{n+1}{3}\rceil + 4$
+ in $\mathbb{Z}$	Y	N	$2n$	1	$2n - w(n) - \lfloor\log n\rfloor - 1$	$10n - 3w(n) - 3w(n-1)$ $- 3\lfloor\log n\rfloor - 3\lfloor\log(n-1)\rfloor - 7$	$\lfloor\log n\rfloor + \lfloor\log(n-1)\rfloor + \lfloor\log\frac{n}{3}\rfloor$ $+ \lceil\log\frac{n-1}{3}\rceil + 8$
+ in $\mathbb{Z}$	Y	Y	$2n+1$	1	$2n - w(n+1) - \lfloor\log(n+1)\rfloor$	$10n - 3w(n) - 3w(n+1)$ $- 3\lfloor\log n\rfloor - 3\lfloor\log(n+1)\rfloor + 1$	$\lfloor\log n\rfloor + \lfloor\log(n+1)\rfloor + \lfloor\log\frac{n}{3}\rfloor$ $+ \lceil\log\frac{n+1}{3}\rceil + 8$
+ (mod 2^n)	N	N	$2n$	n	$n - w(n-1) - \lfloor\log(n-1)\rfloor - 1$	$5n - 3w(n-1) - 3\lfloor\log(n-1)\rfloor - 6$	$\lfloor\log(n-1)\rfloor + \lceil\log\frac{n-1}{3}\rceil + 4$
+ (mod 2^n)	N	Y	$2n+1$	n	$n - w(n) - \lfloor\log n\rfloor$	$5n - 3w(n) - 3\lfloor\log n\rfloor - 2$	$\lfloor\log n\rfloor + \lfloor\log\frac{n}{3}\rfloor + 4$
+ (mod 2^n)	Y	N	$2n$	0	$2n - w(n-1) - \lfloor\log(n-1)\rfloor - 2$	$10n - 6w(n-1) - 6\lfloor\log(n-1)\rfloor - 12$	$2\lfloor\log(n-1)\rfloor + 2\lceil\log\frac{n-1}{3}\rceil + 8$
+ (mod 2^n)	Y	Y	$2n+1$	0	$2n - w(n) - \lfloor\log n\rfloor - 1$	$10n - 6w(n) - 6\lfloor\log n\rfloor - 4$	$2\lfloor\log n\rfloor + 2\lfloor\log\frac{n}{3}\rfloor + 8$
+ (mod $2^n - 1$)	N	—	$2n$	n	$n - 2$	$5n - 6$	$2\lfloor\log(n-1)\rfloor + 5$
+ (mod $2^n - 1$)	Y	—	$2n$	0	$2n - 2$	$10n - 11$	$3\lfloor\log(n-1)\rfloor + \lceil\log\frac{n-1}{3}\rceil + 12$
Comparison	—	N	$2n$	1	$2n - \lfloor\log(n-1)\rfloor - 3$	$6n - w(n-1) - 2\lfloor\log(n-1)\rfloor - 7$	$2\lfloor\log n\rfloor + 5$
Comparison	—	Y	$2n+1$	1	$2n - \lfloor\log n\rfloor - 2$	$6n - w(n) - 2\lfloor\log n\rfloor - 3$	$2\lfloor\log(n+1)\rfloor + 5$
Ripple-carry [7]	Y	N	$2n$	1	n	$4n - 2$	$3n - 1$
Ripple-carry [1]	Y	Y	$2n+1$	1	0	$2n - 1$	$2n - 1$

Table 2: Circuit summary, for $n \geq 7$. The first column gives the function being computed. The second lists whether the computation is done in place, and the third lists whether we take an incoming carry bit as input. We then list the number of input, output, and ancillae bits, and the circuit size and depth. For the purposes of this table, we only count Toffoli gates. Recall that $w(n)$ is the number of ones in the binary expansion of n.

References

[1] Steven A. Cuccaro, Thomas G. Draper, Samuel A. Kutin, and David Petrie Moulton, *A new quantum ripple-carry addition circuit*, in preparation.

[2] Thomas G. Draper, *Addition on a quantum computer*, quant-ph/0008033.

[3] Kai Hwang, *Computer arithmetic: Principles, architecture, and design*, John Wiley & Sons, 1979.

[4] M. Morris Mano, *Digital logic and computer design*, Prentice-Hall, Inc., 1979.

[5] Yu. Ofman, *On the algorithmic complexity of discrete functions*, Soviet Physics Doklady **7** (1963), no. 7, 589–591, English Translation.

[6] Staff of the Computation Laboratory, *Description of a relay calculator*, The Annals of the Computation Laboratory of Harvard University, vol. XXIV, Harvard University Press, 1949.

[7] Vlatko Vedral, Adriano Barenco, and Artur Ekert, *Quantum networks for elementary arithmetic operations*, quant-ph/9511018.

[8] A. Weinberger and J. L. Smith, *A one-microsecond adder using one-megacycle circuitry*, IRE Transactions on Electronic Computers **EC-5** (1956), no. 2.