Nota 9: Digital emulation of 3-qubit quantum circuits

Quantum algorithms can be simulated by general purpose digital computers. However, this type of simulation does not deal efficiently with the parallelism inherent to quantum algorithms. For that reason, nowadays, the design of FPGA-based, special purpose emulators, is considered.

As a very initial step toward the study and development of circuits that emulate the working of a quantum algorithms, a digital circuit that executes quantum operations on a 3-qubit register is reported.

1. Representation of complex numbers

Complex numbers $x = a+bi$ are represented by pairs of fixed-point 2's complement integers:

$$a = a_9\, a_8\, .\, a_7\, a_6\, a_5\, a_4\, a_3\, a_2\, a_1\, a_0,\ b = b_9\, b_8\, .\, b_7\, b_6\, b_5\, b_4\, b_3\, b_2\, b_1\, b_0.$$

Taking into account that the module $|x|$ of all processed numbers x is smaller than or equal to 1, the coefficients a and b belong to the interval

$$-1 \le a, b \le 1,$$

and must be represented with two integer bits. A relatively low precision (eight fractional bits) has been chosen.

Define $A = a \cdot 2^8$ and $B = b \cdot 2^8$. They belong to the interval

$$-2^8 \le A, B \le 2^8,$$

and their product $P = A \cdot B$ belongs to the interval

$$-2^{16} \le P \le 2^{16},$$

so that, in 2's complement,

$$P = P_{17}\, P_{16}\, P_{15}\, P_{14} \cdots P_0.$$

For example,

$$1100\cdots0 = -2^{17} + 2^{16} = -2^{16},\ 0100\cdots0 = 2^{16}.$$

Thus, the product $a{\cdot}b$ is equal to

$$a{\cdot}b = A{\cdot}B{\cdot}2^{-16} = P_{17}\, P_{16}\, .P_{15}\, P_{14} \cdots P_0 \cong P_{17}\, P_{16}\, .\, P_{15}\, P_{14} \cdots P_8.$$

The following VHDL entity (`nota9.vhd`) computes $a{\cdot}b$:

```
ENTITY FP_MUL IS
   PORT(
      A, B: IN SIGNED(n-1 DOWNTO 0);
      C: OUT SIGNED(n-1 DOWNTO 0)
      );
END FP_MUL;

ARCHITECTURE CIRCUIT OF FP_MUL IS
   SIGNAL long_A, long_B: SIGNED(2*n-1 DOWNTO 0);
   SIGNAL long_C: SIGNED(4*n-1 DOWNTO 0);
BEGIN
   long_A(2*n-1 DOWNTO n)<= (OTHERS => A(n-1));
   long_A(n-1 DOWNTO 0)<= A;
   long_B(2*n-1 DOWNTO n)<= (OTHERS => B(n-1));
   long_B(n-1 DOWNTO 0)<= B;
   long_C <=  long_A*long_B;
   C <= long_C(2*n-3 DOWNTO n-2);
```

2. Representation of quantum register states

The quantum state of a 3-qubit register (q_0, q_1, q_2) is represented by a pair (S, T) of Q_STATE-type signals, where

```
TYPE Q_STATE IS ARRAY (NATURAL RANGE <>) OF SIGNED(n-1 DOWNTO 0);
```

Thus, the quantum state of a 3-qubit register is

$$(S(0)+iT(0))|000\rangle + (S(1)+iT(1))|001\rangle + (S(2)+iT(2))|010\rangle +$$

$$(S(3)+iT(3))|011\rangle + (S(4)+iT(4))|100\rangle + (S(5)+iT(5))101\rangle +$$

$$(S(6)+iT(6))|110\rangle + (S(7)+iT(7))|111\rangle.$$

3. Quantum gates

Quantum gates are predefined operations that modify the quantum state. A fundamental property of quantum operations is that any unitary operator on an n-qubit register can be decomposed into 1-qubit unitary operations and 2-qubit CX operations ([1], [2]).

3.1. Unary operators

In matrix form, the unary operators are defined by 2×2 matrices over the complex field:

$$U = \begin{bmatrix} v_{00} + iw_{00} & v_{01} + iw_{01} \\ v_{10} + iw_{10} & v_{11} + iw_{11} \end{bmatrix},$$

where v_{jk} and w_{jk} are real numbers.

Assume that the operator U actuates on qubit q_0, without modifying the state of qubits q_1 and q_2. The corresponding operation on the 3-qubit register (q_0, q_1, q_2) is defined by the following 8×8 matrix (the symbol "×" stands for the Kronecker product):

$$\begin{bmatrix} v_{00} + iw_{00} & v_{01} + iw_{01} \\ v_{10} + iw_{10} & v_{11} + iw_{11} \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \qquad (1)$$

After executing the operation (1) on the 3-qubit register (q_0, q_1, q_2), initially in state

$$(S(0)+iT(0))\,|000\rangle + (S(1)+iT(1))\,|001\rangle + \ldots + (S(7)+iT(7))\,|111\rangle, \quad (2)$$

the new register state will be

$$(v_{00} + iw_{00})\,(S(0)+iT(0)) + (v_{01} + iw_{01})\,(S(4)+iT(4)))\,|000\rangle +$$

$$(v_{00} + iw_{00})\,(S(1)+iT(1)) + (v_{01} + iw_{01})\,(S(5)+iT(5)))\,|001\rangle +$$

$$(v_{00} + iw_{00})\,(S(2)+iT(2)) + (v_{01} + iw_{01})\,(S(6)+iT(6)))\,|010\rangle +$$

$$(v_{00} + iw_{00})\,(S(3)+iT(3)) + (v_{01} + iw_{01})\,(S(7)+iT(7)))\,|011\rangle +$$

$$(v_{10} + iw_{10})\,(S(0)+iT(0)) + (v_{11} + iw_{11})\,(S(4)+iT(4)))\,|100\rangle +$$

$$(v_{10} + iw_{10})\,(S(1)+iT(1)) + (v_{11} + iw_{11})\,(S(5)+iT(5)))\,|101\rangle +$$

$$(v_{10} + iw_{10})\,(S(2)+iT(2)) + (v_{11} + iw_{11})\,(S(6)+iT(6)))\,|110\rangle +$$

$(v_{10} + iw_{10})$ $(S(3)+iT(3))+$ $(v_{11} + iw_{11})$ $(S(7)+iT(7)))$ $|111\rangle$.

In other words, the new values of $S(k)$, $T(k)$, $S(k+4)$ and $T(k+4)$, with $k = 0$ to 3, are

$S(k) \Leftarrow v_{00}S(k) - w_{00}T(k) + v_{01}S(k+4) - w_{01}T(k+4)$,

$T(k) \Leftarrow v_{00}T(k) + w_{00}S(k) + v_{01}T(k+4) + w_{01}S(k+4)$,

$S(k+4) \Leftarrow v_{10}S(k) - w_{10}T(k) + v_{11}S(k+4) - w_{11}T(k+4)$,

$T(k+4) \Leftarrow v_{10}T(k) + w_{10}S(k) + v_{11}T(k+4) + w_{11}S(k+4)$. (3)

Several particular cases are analyzed.

3.1.1. R_φ gate

$$U = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\varphi} \end{bmatrix},$$

so that

$v_{00} = 1$, $w_{00} = 0$, $v_{01} = 0$, $w_{01} = 0$, $v_{10} = 0$, $w_{10} = 0$, $v_{11} = \cos\varphi$, $w_{11} = \sin\varphi$.

According to (3), the operations are ($k = 0$ to 3):

$S(k) \Leftarrow S(k)$, $T(k) \Leftarrow T(k)$,

$S(k+4) \Leftarrow (\cos\varphi)S(k+4) - (\sin\varphi)T(k+4)$,

$T(k+4) \Leftarrow (\cos\varphi)T(k+4) + (\sin\varphi)S(k+4)$. (4)

3.1.2. H gate

$$U = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix},$$

so that

$v_{00} = \frac{1}{\sqrt{2}}$, $w_{00} = 0$, $v_{01} = \frac{1}{\sqrt{2}}$, $w_{01} = 0$, $v_{10} = \frac{1}{\sqrt{2}}$, $w_{10} = 0$, $v_{11} = -\frac{1}{\sqrt{2}}$, $w_{11} = 0$.

Thus, the operations are ($k = 0$ to 3)

$S(k) \Leftarrow \frac{1}{\sqrt{2}} S(k) + \frac{1}{\sqrt{2}} S(k+4)$,

$T(k) \Leftarrow \frac{1}{\sqrt{2}} T(k) + \frac{1}{\sqrt{2}} T(k+4),$

$S(k+4) \Leftarrow \frac{1}{\sqrt{2}} S(k) - \frac{1}{\sqrt{2}} S(k+4),$

$T(k+4) \Leftarrow \frac{1}{\sqrt{2}} T(k) - \frac{1}{\sqrt{2}} T(k+4).$ (5)

3.1.3. Global phase $e^{i\varphi}$

$$U = \begin{bmatrix} e^{i\varphi} & 0 \\ 0 & e^{i\varphi} \end{bmatrix},$$

so that

$$v_{00} = \cos\varphi,\ w_{00} = \sin\varphi,\ v_{01} = 0,\ w_{01} = 0,$$

$$v_{10} = 0,\ w_{10} = 0,\ v_{11} = \cos\varphi,\ w_{11} = \sin\varphi.$$

Thus (k = 0 to 7),

$S(k) \Leftarrow (\cos\varphi)S(k) - (\sin\varphi)T(k),\ T(k) \Leftarrow (\cos\varphi)T(k) + (\sin\varphi)S(k).$ (6)

3.1.4. $R_x(\varphi)$ gate

$$U = \begin{bmatrix} cos\frac{\varphi}{2} & -isin\frac{\varphi}{2} \\ -isin\frac{\varphi}{2} & cos\frac{\varphi}{2} \end{bmatrix},$$

so that

$$v_{00} = \cos\varphi/2,\ w_{00} = 0,\ v_{01} = 0,\ w_{01} = -\sin\varphi/2,$$

$$v_{10} = 0,\ w_{10} = -\sin\varphi/2,\ v_{11} = \cos\varphi/2,\ w_{11} = 0.$$

Thus (k = 0 to 3),

$S(k) \Leftarrow (\cos\varphi/2)S(k) + (\sin\varphi/2)T(k+4),$

$T(k) \Leftarrow (\cos\varphi/2)T(k) - (\sin\varphi/2)S(k+4),$

$S(k+4) \Leftarrow (\sin\varphi/2)T(k) + (\cos\varphi/2)S(k+4),$

$T(k+4) \Leftarrow -(\sin\varphi/2)S(k) + (\cos\varphi/2)T(k+4).$ (7)

3.1.5. $R_y(\varphi)$ gate

$$U = \begin{bmatrix} cos\frac{\varphi}{2} & -sin\frac{\varphi}{2} \\ sin\frac{\varphi}{2} & cos\frac{\varphi}{2} \end{bmatrix},$$

so that

$$v_{00} = \cos\varphi/2,\ w_{00} = 0,\ v_{01} = -\sin\varphi/2,\ w_{01} = 0,$$

$$v_{10} = \sin\varphi/2,\ w_{10} = 0,\ v_{11} = \cos\varphi/2,\ w_{11} = 0.$$

Thus (k = 0 to 3),

$S(k)$ <= $(\cos\varphi/2)S(k) - (\sin\varphi/2)S(k+4)$,

$T(k)$ <= $(\cos\varphi/2)T(k) - (\sin\varphi/2)T(k+4)$,

$S(k+4)$ <= $(\sin\varphi/2)S(k) + (\cos\varphi/2)S(k+4)$,

$T(k+4)$ <= $(\sin\varphi/2)T(k) + (\cos\varphi/2)T(k+4)$. (8)

3.1.6. $R_z(\varphi)$ gate

$$U = \begin{bmatrix} e^{-i\varphi/2} & 0 \\ 0 & e^{i\varphi/2} \end{bmatrix},$$

so that

$$v_{00} = \cos\varphi/2,\ w_{00} = -\sin\varphi/2,\ v_{01} = 0,\ w_{01} = 0,$$

$$v_{10} = 0,\ w_{10} = 0,\ v_{11} = \cos\varphi/2,\ w_{11} = \sin\varphi/2.$$

Thus (k = 0 to 3),

$S(k)$ <= $(\cos\varphi/2)S(k) + (\sin\varphi/2)T(k)$,

$T(k)$ <= $(\cos\varphi/2)T(k) - (\sin\varphi/2)S(k)$,

$S(k+4)$ <= $(\cos\varphi/2)S(k+4) - (\sin\varphi/2)T(k+4)$,

$T(k+4)$ <= $(\cos\varphi/2)T(k+4) + (\sin\varphi/2)S(k+4)$. (9)

3.1.7. Pauli operators

The Pauli operators X, Y and Z are particular cases of rotations: $X = e^{i\pi/2}R_x(\pi)$, $Y = e^{i\pi/2}R_y(\pi)$, $Z = e^{i\pi/2}R_z(\pi)$. For example, the X gate executes the following operations (k = 0 to 3):

$S(k) <= S(k+4)$, $T(k) <= T(k+4)$, $S(k+4) <= S(k)$, $T(k+4) <= T(k)$.

3.2. *CU* gate

Consider a CU gate that actuates on q_1 (target qubit) under the control of q_0 (control qubit), without modifying the state of q_2. The transformation of the register (q_0, q_1, q_2) state is defined by the following 8×8 matrix:

$$\begin{bmatrix} 1 & 0 & & 0 & 0 \\ 0 & 1 & & 0 & 0 \\ 0 & 0 & v_{00}+iw_{00} & v_{01}+iw_{01} \\ 0 & 0 & v_{10}+iw_{10} & v_{11}+iw_{11} \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \qquad (10)$$

Thus, after executing the operation (10) on the 3-qubit register (q_0, q_1, q_2), initially in state (2), the new register state will be

$(S(0)+iT(0))\,|000\rangle + (S(1)+iT(1)))\,|001\rangle +$

$(S(2)+iT(2))\,|010\rangle + (S(3)+iT(3)))\,|011\rangle +$

$(v_{00}+iw_{00})\,(S(4)+iT(4)) + (v_{01}+iw_{01})\,(S(6)+iT(6)))\,|100\rangle +$

$(v_{00}+iw_{00})\,(S(5)+iT(5)) + (v_{01}+iw_{01})\,(S(7)+iT(7)))\,|101\rangle +$

$(v_{10}+iw_{10})\,(S(4)+iT(4)) + (v_{11}+iw_{11})\,(S(6)+iT(6)))\,|110\rangle +$

$(v_{10}+iw_{10})\,(S(5)+iT(5)) + (v_{11}+iw_{11})\,(S(7)+iT(7)))\,|111\rangle$. (11)

The new values of $S(k)$ and $T(k)$ are

$S(k) <= S(k)$, $T(k) <= T(k)$, k = 0 to 3,

$S(4) <= v_{00}S(4) - w_{00}T(4) + v_{01}S(6) - w_{01}T(6)$,

$T(4) <= v_{00}T(4) + w_{00}S(4) + v_{01}T(6) + w_{01}S(6)$,

$S(5) <= v_{00}S(5) - w_{00}T(5) + v_{01}S(7) - w_{01}T(7)$,

$T(5) <= v_{00}T(5) + w_{00}S(5) + v_{01}T(7) - w_{01}S(7)$,

$S(6) <= v_{10}S(4) - w_{10}T(4) + v_{11}S(6) - w_{11}T(6)$,

$T(6) <= v_{10}T(4) + w_{10}S(4) + v_{11}T(6) + w_{11}S(6)$,

$S(7) <= v_{10}S(5) - w_{10}T(5) + v_{11}S(7) - w_{11}T(7)$,

$T(7) <= v_{10}T(5) + w_{10}S(5) + v_{11}T(7) + w_{11}S(7)$. (12)

Several particular cases are analyzed.

3.2.1. CR_{φ} gate

As was seen above (Sec.3.1.1)

$v_{00} = 1$, $w_{00} = 0$, $v_{01} = 0$, $w_{01} = 0$, $v_{10} = 0$, $w_{10} = 0$, $v_{11} = \cos\varphi$, $w_{11} = \sin\varphi$.

According to (12), the operations are

$S(k) <= S(k)$, $T(k) <= T(k)$, $k = 0$ to 5,

$S(6) <= (\cos\varphi)S(6) - (\sin\varphi)T(6)$,

$T(6) <=(\cos\varphi)T(6) + (\sin\varphi)S(6)$,

$S(7) <= (\cos\varphi)S(7) - (\sin\varphi)T(7)$,

$T(7) <= (\cos\varphi)T(7) + (\sin\varphi)S(7)$. (13)

3.2.2. CX gate

The transformation of the register (q_0, q_1, q_2) state is defined by the following 8×8 matrix:

$$\begin{bmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 0 & 1\\0 & 0 & 1 & 0\end{bmatrix} \times \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}.$$

Thus, it executes the following operations:

$S(k) <= S(k)$, $T(k) <= T(k)$, if $k = 0$ to 3,

$S(4) <= S(6)$, $S(5) <= S(7)$, $S(6) <= S(4)$, $S(7) <= S(5)$,

$T(4) <= T(6)$, $T(5)<= T(7)$, $T(6) <= T(4)$, $T(7) <= T(5)$. (14)

3.2. *SWAP* gate

A *SWAP* gate, executed on qubits q_0 and q_1, without modifying the state of qubit q_3, is defined by the following matrix:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

It executes the following operations:

$S(k) <= S(k)$, $T(k) <= T(k)$, if $k = 0, 1, 6$ and 7,

$S(2) <= S(4)$, $T(2) <= T(4)$,

$S(3) <= S(5)$, $T(3) <= T(5)$,

$S(4) <= S(2)$, $T(4) <= T(2)$,

$S(5) <= S(3)$, $T(5) <= T(3)$. (15)

3.3. Ternary operators

The Toffoli or C^2X gate, applied to q_3 (target qubit), under the control of q_1 and q_2 (control qubits), is defined by the following matrix:

$$C^2X = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

It executes the following operations:

$S(k) <= S(k)$, $T(k) <= T(k)$, if $k = 0$ to 5,

$S(6) <= S(7)$, $T(6) <= T(7)$,

$S(7) <= S(6)$, $T(7) <= T(6)$. (16)

The Fredkin or *CSWAP* gate, applied to q_2 and q_3 (target qubits), under the control of q_0 (control qubit), is defined by the following matrix:

$$CSWAP = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

It executes the following operations:

$S(k) <= S(k)$, $T(k) <= T(k)$, if k = 0 to 4, and k = 7,

$S(5) <= S(6)$, $T(5) <= T(6)$,

$S(6) <= S(5)$, $T(6) <= T(5)$. (17)

3.4 Conclusions and comments

- Up to this point, only unary operations executed on q_0, *CU* gates executed on q_1 under the control of q_0, C^2X gates executed on q_2 under the control of q_0 and q_1, *CSWAP* gates executed on q_1 and q_2 under the control of q_0, have been considered. In the case of the same operations, applied to other qubits (for example, unary operations on q_0 or q_1), the conclusions are similar. Equations (4) to (9), or (13) to (17), must be modified, but they all execute the same arithmetic operation

  $$aX + bY, \tag{18}$$

  where

  $$X, Y \in \{S(k), T(k), k = 0 \text{ to } 7\},$$

  and a and b are predefined constants (1, 0, $\cos\varphi$, $\sin\varphi$, - $\sin\varphi$, ...).
- It can be demonstrated (theorem 6.1, corollary 6.1 and Fig.6.12 of [1]) that any unitary operator on an n-qubit register can be decomposed into global phase $e^{i\varphi}$ operations (Sec.3.1.3), 1-qubit rotations about the

coordinate axis (Sec.3.1.4, Sec.3.1.5 and Sec.3.1.6) and 2-qubit CX operations (Sec.3.2). The set of operations defined above includes those operations; thus, it permits to synthesize any operation on a 3-qubit register.

4. Digital emulation

Conceptually, the design of a digital circuit that executes a quantum algorithm, based on the operators defined above, is straightforward. A completely parallel circuit is made up of a register that stores sixteen fixed point numbers, sixteen $aX + bY$ computation resources, and a control unit that defines the connections, at each step of the algorithm execution (Fig.1).

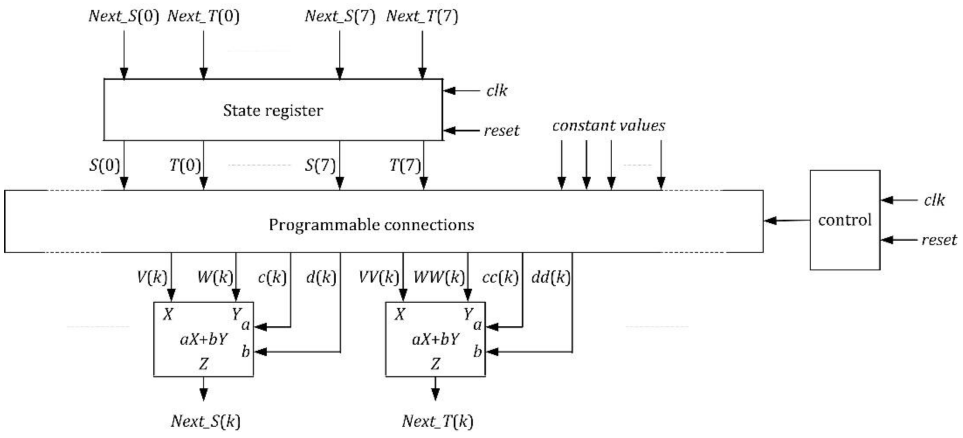

Figure 1 Digital circuit

As an example, the VHDL file `nota9_1.vhd` describes the circuit of Fig.2, with connections that correspond to the Quantum Fourier Transform on 3 qubits.

The data path is modelled by a process (`state_register`) and by sixteen components (`computation_resources`) that execute the operation (18). The control unit consists of a process (`steps`), in this example an 8-state counter, and a process (`connections`), that stores a set of tables within a case instruction; they program the connections, in function of the operation under execution:

NOP (step 0), $H(q_0)$ (step 1), $CR_{\pi/2}(q_1, q_0)$ (step 2), $CR_{\pi/4}(q_2, q_0)$ (step 3), $H(q_1)$ (step 4), $CR_{\pi/2}(q_2, q_1)$ (step 5), $H(q_2)$ (step 6), NOP (step 7).

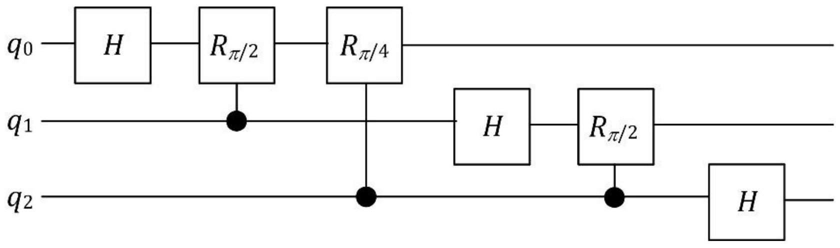

Figure 2 Quantum Fourier transform (3 qubits)

```
ENTITY QFT_3 IS
PORT(
 clk, reset: IN STD_LOGIC;
 S_out, T_out: OUT Q_STATE(0 TO 7)
);
END QFT_3;

ARCHITECTURE circuit OF QFT_3 IS
   COMPONENT FP_MUL_ADD IS
   PORT(
      X, Y, A, B: IN SIGNED(n-1 DOWNTO 0);
      C: OUT SIGNED(n-1 DOWNTO 0)
      );
   END COMPONENT;
   SIGNAL S, T, Next_S, Next_T, V, VV, W, WW, C, D, CC, DD: Q_STATE(0 TO
7);
   TYPE step IS RANGE 0 TO 7;
   SIGNAL current_step: step;
BEGIN

--DATA PATH
   state_register: PROCESS (clk, reset, Next_S, Next_T)
   BEGIN
      IF reset = '1' THEN
         FOR i IN 0 TO 4 LOOP
         S(i) <= "00"&x"00"; T(i) <= "00"&x"00";
         END LOOP;
         S(5) <= "01"&x"00"; T(5) <= "00"&x"00";
         S(6) <= "00"&x"00"; T(6) <= "00"&x"00";
         S(7) <= "00"&x"00"; T(7) <= "00"&x"00";
       ELSIF clk'event AND clk = '1' THEN
         S <= Next_S; T <= Next_T;
       END IF;
   END PROCESS;
   S_out <= S; T_out <= T;
   computation_resources: FOR k IN 0 TO 7 GENERATE
```

```
      comp1: FP_MUL_ADD PORT MAP(X => V(k) , Y => W(k), A => C(k) ,
         B => D(k), C => Next_S(k));
      comp2: FP_MUL_ADD PORT MAP(X => VV(k), Y => WW(k), A => CC(k),
         B => DD(k), C => Next_T(k));
   END GENERATE;

--CONTROL UNIT
   steps: PROCESS (clk, reset)
   BEGIN
   IF reset = '1' THEN current_step <= 0;
   ELSIF clk'event AND clk = '1' THEN
      CASE current_step IS
         WHEN 0 => current_step <= 1;
         WHEN 1 => current_step <= 2;
         WHEN 2 => current_step <= 3;
         WHEN 3 => current_step <= 4;
         WHEN 4 => current_step <= 5;
         WHEN 5 => current_step <= 6;
         WHEN 6 => current_step <= 7;
         WHEN 7 => current_step <= 7;
      END CASE;
   END IF;
   END PROCESS;

   connections: PROCESS (current_step, S, T)
   BEGIN
   CASE current_step IS
   WHEN 0 =>
   -- NOP
   V(0) <= S(0); W(0) <= zero; C(0) <= one; D(0) <= zero;
   VV(0) <= T(0); WW(0) <= zero; CC(0) <= one; DD(0) <= zero;
   V(1) <= S(1); W(1) <= zero; C(1) <= one; D(1) <= zero;
   VV(1) <= T(1); WW(1) <= zero; CC(1) <= one; DD(1) <= zero;
   V(2) <= S(2); W(2) <= zero; C(2) <= one; D(2) <= zero;
   VV(2) <= T(2); WW(2) <= zero; CC(2) <= one; DD(2) <= zero;
   V(3) <= S(3); W(3) <= zero; C(3) <= one; D(3) <= zero;
   VV(3) <= T(3); WW(3) <= zero; CC(3) <= one; DD(3) <= zero;
   V(4) <= S(4); W(4) <= zero; C(4) <= one; D(4) <= zero;
   VV(4) <= T(4); WW(4) <= zero; CC(4) <= one; DD(4) <= zero;
   V(5) <= S(5); W(5) <= zero; C(5) <= one; D(5) <= zero;
   VV(5) <= T(5); WW(5) <= zero; CC(5) <= one; DD(5) <= zero;
   V(6) <= S(6); W(6) <= zero; C(6) <= one; D(6) <= zero;
   VV(6) <= T(6); WW(6) <= zero; CC(6) <= one; DD(6) <= zero;
   V(7) <= S(7); W(7) <= zero; C(7) <= one; D(7) <= zero;
   VV(7) <= T(7); WW(7) <= zero; CC(7) <= one; DD(7) <= zero;
   WHEN 1 =>
   -- H0
   V(0) <= S(0); W(0) <= S(4); C(0) <= inv_sqrt_2; D(0) <= inv_sqrt_2;
   VV(0) <= T(0); WW(0) <= T(4); CC(0) <= inv_sqrt_2; DD(0) <= inv_sqrt_2;
   ...
   WHEN 2 =>
   -- CR_PI/2_10
   ...
   WHEN 3 =>
   -- CR_PI/4_20
   ...
   WHEN 4 =>
   -- H1
   ...
   WHEN 5 =>
   -- CR_PI/2_21
   ...
   WHEN 6 =>
   -- H2
   ...
   WHEN 7 =>
   -- NOP
   ...
   END CASE;
   END PROCESS;
END circuit;
```

The initial state, on reset, has been chosen equal to $|101\rangle$, so that the initial values are

$$S = (0, 0, 0, 0, 1, 0, 0, 0),\ T = (0, 0, 0, 0, 0, 0, 0, 0).$$

A test of the entity *QFT_3* gives the following result (hexadecimal notation):

$$Next\_S = (059, 3A6, 000, 000, 3BF, 040, 040, 3BF),$$

$$Next\_T = (000, 000, 059, 3A6, 3BF, 040, 3BF, 040).$$

Taking into account the chosen representation of the complex numbers (Sec.1), the result is

$0.347|000\rangle - 0.351|001\rangle + 0.347i|010\rangle - 0.351i|011\rangle - 0.253(1+i)|100\rangle + 0.25(1+i)|101\rangle + (0.25-0.253i)|110\rangle + (-0.253 + 0.25i)|111\rangle$.

In order to check the result, the same operation has been simulated with the Python package *cirq* (`nota9_1.py`). The initial state is set to $|101\rangle$ with X gates.

```
import cirq
j = [1,0,1]
j1, j2, j3 = cirq.LineQubit.range(3)
qft = cirq.Circuit()
if j[0] == 1:
    qft.append(cirq.X(j1))
if j[1] == 1:
    qft.append(cirq.X(j2))
if j[2] == 1:
    qft.append(cirq.X(j3))
CRpi_2 = cirq.S.controlled()
CRpi_4 = cirq.T.controlled()
qft.append([cirq.H(j1),CRpi_2(j2, j1), CRpi_4(j3,j1)])
qft.append([cirq.H(j2),CRpi_2(j3, j2)])
qft.append([cirq.H(j3)])
#print(qft)
un_simulador = cirq.Simulator()
estado_final = un_simulador.simulate(qft)
print(estado_final)
```

This is the result:

```
qubits: (cirq.LineQubit(2), cirq.LineQubit(1), cirq.LineQubit(0))
output vector:
0.354|000⟩ + (-0.25-0.25j)|001⟩ + 0.354j|010⟩ + (0.25-0.25j)|011⟩
- 0.354|100⟩ + (0.25+0.25j)|101⟩ - 0.354j|110⟩ + (-0.25+0.25j)|111⟩
```

Apart from precision errors (fixed point numbers with only 8 fractional bits) and different enumeration orders of the qubits (q_2, q_1, q_0 vs. q_0, q_1, q_2), the results are the same.

The VHDL and Python files that correspond to this note (`nota9_1.vhd` and `nota9_1.py`) are available at the web site associated to [1].

5. Comments

- The proposed architecture (Fig.1) is a straightforward translation of a computation scheme based on unary and binary resources. This implementation could be suitable in the case of circuits with a small number of qubits. Consider a circuit consisting of N qubits, and assume that the fixed-point numbers are represented with n bits. The state register of Fig.1 stores $n{\cdot}2^{N+1}$ bits. If, as before, N = 3 and n = 10, the state register stores 160 bits. Nevertheless, the state register size grows exponentially in function of the number N of qubits. In the case of a circuit that includes e.g. 30 qubits, with 32-bit fixed-number coefficients, the state register must store $64{\cdot}2^{30}$ bits, that is, 8 Gbytes. Obviously, it is not possible to emulate a quantum circuit with hundredth of qubits. For example, with n = 32, N = 100, $n{\cdot}2^{N+1} = 2^{106}$ bits. This is precisely the reason why quantum circuits can execute algorithms that a digital circuit cannot.
- Instead of executing, at each step, 16 operations $aX+bY$ – in general 2^{N+1} operations - , other solutions can be considered. In fact, all classical digital circuit implementation methods can be used (parallelism vs.

sequentialization, pipeline). This same comment applies to the programmable connections. In the preceding example (QFT-3 entity), the connections between the state register and the computation resources are executed in parallel, and their implementation is assumed to be executed by the synthesis tools. The corresponding array must be able to connect more than sixteen 10-bit sources to sixty-four 10-bit targets. All classical digital circuit implementation methods can be considered (parallelism vs. sequentialization, multiplexers, buses).

- A conclusion of the preceding comments is that the emulation of quantum circuits including more than a few qubits, is only possible with digital electronic boards that include, among others, programmable devices, connection resources and large memories. Examples of such systems are reported in [3].

References

[1] J.P.Deschamps, Computación Cuántica, Marcombo, Barcelona, 2023.

[2] M.A.Nielsen and I.L.Chuang, Quantum Computation and Quantum Information*,* Cambridge, Cambridge University Press, 2010.

[3] N.Mahmud, E.El-Araby and D.Caliga, Scaling reconfigurable emulation of quantum algorithms at high precision and high throughput, Quantum Engineering, Wiley, 2019,1:e19.