Nota 17: Digital emulation of quantum circuits, TR8

Technical report 2, April 2025

1 Introduction

The development of digital circuits that emulate the working of quantum circuits is justified by the following observations. Quantum Computation is a reality. Small size quantum computers have been designed and manufactured by big electronic companies (Google [3], IBM [4]). They are able to execute massively parallel algorithms that cannot be executed on classical digital computers - the so-called quantum supremacy [3] -. Many research departments study the definition of algorithms that might be executed on existing or future quantum computers. Nevertheless, they generally don't have the possibility to check their designs on a quantum computer. Furthermore, current quantum circuits have restricted features as regards their size (number of qubits) and the number of quantum operations they can execute, limited by the decoherence time [1]. In conclusion, the study of quantum algorithms and circuits needs the availability of design tools such as simulators, running on classical computers, or specific digital circuits that emulate the working of quantum circuits. With such tools, the working of ideal (without decoherence) quantum circuits can be checked. Obviously, the number of simulated or emulated qubits is limited (if not, what for would serve a quantum computer?). The reason is that the quantum state of a set of n qubits is modelled by the state of a digital register that stores 2^n complex numbers. The advantage of quantum circuits, with respect to digital circuits, is the existence of superposition states. The way in which a digital circuit emulates a quantum circuit is the substitution of n-qubit quantum registers by M-bit digital registers, with $M = 2 \cdot t \cdot 2^n$, being t the number of bits that encode the real part and the imaginary part of a complex number, a shift of time to hardware complexity as M is much greater than N.

In this paper, a digital circuit that emulates the working of quantum circuits is described. It is implemented on a relatively low-cost board whose main component is an FPGA including programmable logic, memory blocks, a host microprocessor and peripheral devices.

Several FPGA-based circuits that emulate the working of quantum circuits have been proposed and reported. In some cases, those circuits must be specifically synthesized and implemented in order to emulate a particular quantum circuit or a particular quantum algorithm [13] to [19]. The circuit proposed in this paper is independent of the particular circuit to be emulated. It includes a program memory that stores a description of the quantum circuit and is downloaded to the host processor from a PC through a serial channel. A similar solution is proposed in [20].

2 Background

Quantum computers are made up of small devices called qubits. As an example, charge qubits ([5], [6], [7]) are very high frequency and anharmonic oscillators whose energy levels, or oscillation frequencies, are generally restricted to the ground level, denoted by $|0\rangle$, and to the first excited level, denoted by $|1\rangle$. External microwave signals and bias currents permit to control the state of every qubit. Furthermore, neighbored qubits can be coupled, under the control of additional external signals. A set of qubits, along with the control signals applied to him, moment by moment, constitute a quantum circuit.

Consider a quantum circuit made up of n qubits q_0, q_1, …, q_{n-1}. The quantum state $|\psi\rangle$ of the circuit is defined by an expression such as

$$|\psi\rangle = a_0|00\ldots00\rangle + a_1|00\ldots01\rangle + a_2|00\ldots10\rangle + ... + a_{N-1}|11\ldots11\rangle, \quad (1)$$

where $N = 2^n$ and the coefficients a_k are complex numbers. The physical meaning of this expression is the following: when measuring the register state, the result will be 00…00 with a probability equal to $|a_0|^2$, 00…01 with a probability equal to $|a_1|^2$, and so on. Apart from the probabilistic physical interpretation, this is an enormous difference between digital and quantum circuits: the state of an n-bit digital register is a dimension-n vector over the binary field, while the state of an n-qubit quantum register is a dimension-2^n vector over the complex field C.

Whatever the type of implementation of the qubits, the behavior of a quantum circuit is defined by a unitary transformation. In matrix terms, that means that if the initial quantum state (1) is represented by the column vector

$$v = (a_0, a_1, a_2, ..., a_{N-1})^T, \quad (2)$$

then the behavior of the circuit is defined by a unitary, N by N, matrix U over the complex field, so that the final quantum state is represented by the following vector u:

$$u = Uv. \tag{3}$$

In lineal algebra, a matrix U over the complex field is called unitary iff

$$UU^{+} = U^{+}U = I, \tag{4}$$

where U^{+} is the conjugate transpose of U and I the identity matrix. Observe that, using the unitary property, the relation (3) can be inverted, that is $v = U^{+}u$.

Given an n-qubit register, the implementation of a unitary transformation U is based on the use of quantum gates that act on small sets of qubits. Some examples with $n = 3$ are given.

- An X gate, applied to qubit q_1, inverts the basic state of q_1 ($|0\rangle \rightarrow |1\rangle$, $|1\rangle \rightarrow |0\rangle$), without modifying the other qubit states. Thus, the 3-qubit register state is transformed as follows:

$$a_0|000\rangle + a_1|001\rangle + a_2|010\rangle + a_3|011\rangle + a_4|100\rangle + a_5|101\rangle + a_6|110\rangle + a_7|111\rangle \rightarrow$$

$$a_0|010\rangle + a_1|011\rangle + a_2|000\rangle + a_3|001\rangle + a_4|110\rangle + a_5|111\rangle + a_6|100\rangle + a_7|101\rangle =$$

$$a_2|000\rangle + a_3|001\rangle + a_0|010\rangle + a_1|011\rangle + a_6|100\rangle + a_7|101\rangle + a_4|110\rangle + a_5|111\rangle. \tag{5}$$

  The corresponding matrix is defined by the Kronecker product $I_2 \times X \times I_2$, where I_p stands for the p by p identity matrix and

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \tag{6}$$

  Observe that X is unitary, so that $I_2 \times X \times I_2$ is unitary.

- A CX gate, applied to qubit q_1, under the control of qubit q_0, inverts the basic states of q_1 if, and only if, the current state of q_0 is $|1\rangle$, without modifying the other qubit states. Thus, the 3-qubit register state is transformed as follows:

$$a_0|000\rangle + a_1|001\rangle + a_2|010\rangle + a_3|011\rangle + a_4|100\rangle + a_5|101\rangle + a_6|110\rangle + a_7|111\rangle \rightarrow$$

$$a_0|000\rangle + a_1|001\rangle + a_2|010\rangle + a_3|011\rangle + a_4|110\rangle + a_5|111\rangle + a_6|100\rangle + a_7|101\rangle =$$

$$a_0|000\rangle + a_1|001\rangle + a_2|010\rangle + a_3|011\rangle + a_6|100\rangle + a_7|101\rangle + a_4|110\rangle + a_5|111\rangle. \tag{7}$$

  The corresponding matrix is

$$\begin{bmatrix} I_4 & 0_{4,4} \\ 0_{4,4} & X \times I_2 \end{bmatrix} \tag{8}$$

  where $0_{p,q}$ stands for the p by q "all zero" matrix. The matrix (8) is unitary.

- A C^2X or Toffoli gate, applied to qubit q_2, under the control of qubits q_0 and q_1, inverts the basic states of q_2 if, and only if, the current state of q_0 and q_1 is $|11\rangle$. If $n > 3$, the other qubit states wouldn't be modified. Thus, the 3-qubit register state is transformed as follows:

$$a_0|000\rangle + a_1|001\rangle + a_2|010\rangle + a_3|011\rangle + a_4|100\rangle + a_5|101\rangle + a_6|110\rangle + a_7|111\rangle \rightarrow$$

$$a_0|000\rangle + a_1|001\rangle + a_2|010\rangle + a_3|011\rangle + a_4|100\rangle + a_5|101\rangle + a_6|111\rangle + a_7|110\rangle =$$

$$a_0|000\rangle + a_1|001\rangle + a_2|010\rangle + a_3|011\rangle + a_4|100\rangle + a_5|101\rangle + a_7|110\rangle + a_6|111\rangle. \tag{9}$$

  The corresponding matrix

$$\begin{bmatrix} I_6 & 0_{6,2} \\ 0_{2,6} & X \end{bmatrix} \tag{10}$$

is unitary.

Many other unary, binary and ternary operators can be defined. Consider a 2 by 2 unitary matrix

$$U = \begin{bmatrix} u_{00} & u_{01} \\ u_{10} & u_{11} \end{bmatrix}. \tag{11}$$

The following unary and binary gates can be defined.

- A U gate, applied to qubit q_1, transforms the basic states of q_1 as follows:

$$|0\rangle \rightarrow u_{00}|0\rangle + u_{10}|1\rangle,\ |1\rangle \rightarrow u_{01}|0\rangle + u_{11}|1\rangle. \tag{12}$$

The corresponding matrix is defined by the Kronecker product $I_2 \times U \times I_2$. It is unitary.

- A CU gate, applied to qubit q_1, under the control of qubit q_0 transforms the basic states of q_1 according to (12), if, and only if, the current state of q_0 is $|1\rangle$, without modifying the other qubit states. The corresponding matrix

$$\begin{bmatrix} I_4 & 0_{4,4} \\ 0_{4,4} & U \times I_2 \end{bmatrix} \tag{13}$$

is unitary.

Ternary C^2U gates could also be defined.

Other types of binary and ternary gates, that cannot be defined in the same way, are the following (with $n = 3$).

- A *SWAP* gate applied to qubits q_1 and q_2 interchanges the states of q_1 and q_2. Thus, the 3-qubit register state is transformed as follows:

$$a_0|000\rangle + a_1|001\rangle + a_2|010\rangle + a_3|011\rangle + a_4|100\rangle + a_5|101\rangle + a_6|110\rangle + a_7|111\rangle \rightarrow$$

$$a_0|000\rangle + a_2|001\rangle + a_1|010\rangle + a_3|011\rangle + a_4|100\rangle + a_6|101\rangle + a_5|110\rangle + a_7|111\rangle. \tag{14}$$

- A *CSWAP* or Fredkin gate, applied to qubits q_1 and q_2, under the control of qubit q_0, interchanges the states of q_1 and q_2 if, and only if, the state of q_0 is $|0\rangle$. Thus, the 3-qubit register state is transformed as follows:

$$a_0|000\rangle + a_1|001\rangle + a_2|010\rangle + a_3|011\rangle + a_4|100\rangle + a_5|101\rangle + a_6|110\rangle + a_7|111\rangle \rightarrow$$

$$a_0|000\rangle + a_1|001\rangle + a_2|010\rangle + a_3|011\rangle + a_4|100\rangle + a_6|101\rangle + a_5|110\rangle + a_7|111\rangle. \tag{15}$$

Thus, quantum gates are unitary operators that modify the state of a quantum register. Their implementation depends on the used technology. Consider the case of registers made up of charge qubits. The implementation of a particular gate consists of applying to some qubits and coupling circuits the control signals that actually execute the corresponding operation. Among the most commonly used unary gates are the following, with the corresponding 2 by 2 matrices:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}, T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix},$$

$$R_x(\gamma) = \begin{bmatrix} \cos\frac{\gamma}{2} & -i \cdot \sin\frac{\gamma}{2} \\ -i \cdot \sin\frac{\gamma}{2} & \cos\frac{\gamma}{2} \end{bmatrix}, R_y(\gamma) = \begin{bmatrix} \cos\frac{\gamma}{2} & -\sin\frac{\gamma}{2} \\ \sin\frac{\gamma}{2} & \cos\frac{\gamma}{2} \end{bmatrix}, R_z(\gamma) = \begin{bmatrix} e^{-i\gamma/2} & 0 \\ 0 & e^{i\gamma/2} \end{bmatrix}, R_\gamma = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\gamma} \end{bmatrix}, \tag{16}$$

where γ is a real belonging to the interval $0 \le \gamma < 2\pi$.

Assuming that the used technology permits to implement quantum gates - at least some of them - quantum circuits can be defined. First define symbols that represent the available quantum gates. Some of them are shown in Fig.1 (unary operations) and Fig.2 (binary operations). In Fig.2a the symbol of a CU gate, operating on the target qubit t under the control of qubit c, is shown. Fig.2b is the symbol of a CX gate. Fig.3c is the symbol of a *SWAP* gate.

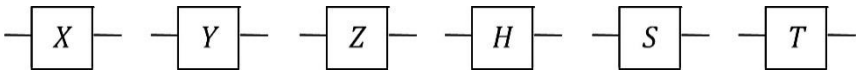

Figure 1 Unary quantum gates

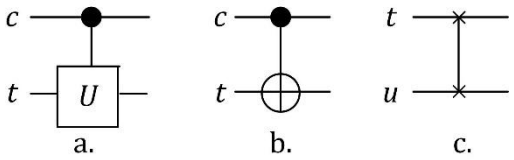

Figure 2 Binary quantum gates

In Fig.3, an example of quantum circuit is shown. It is a 4-qubit register. It executes the following operations: first execute an *H* operation on the first qubit; then execute a $CR_{\pi/2}$ operation on the first qubit under the control of the second qubit; after that, execute a $CR_{\pi/4}$ operation on the first qubit under the control of the third qubit, and so on.

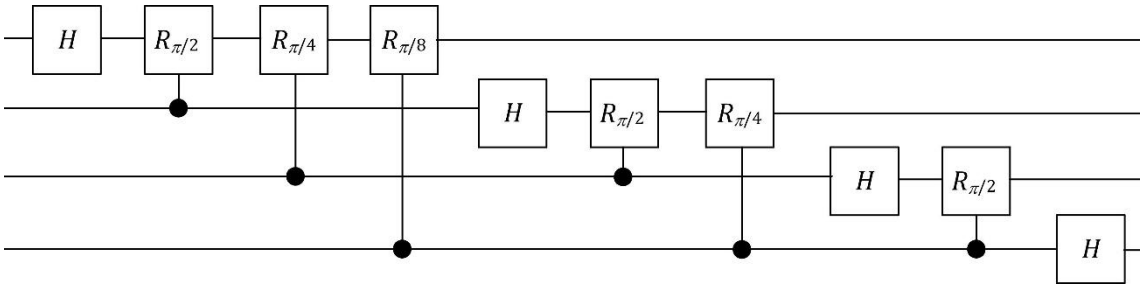

Figure 3 Quantum circuit

Notice that the horizontal lines don't represent connections. They define the chronology of the operations.

The following algebraic property constitutes the base of the quantum circuit synthesis methods and validates the emulation circuit proposed in this paper.

Property 1 Any unitary operator *U* operating on a register of *n* qubits can be expressed as a product of operators $U_1 \cdot U_2 \cdot \ldots \cdot U_M$ where every U_k is either a *CX* gate operating on one of the *n* qubits, under the control of another qubit, or a unary gate operating on one of the *n* qubits ([1], [2]).

Several methods of matrix decomposition have been proposed, for example, those based on the decomposition into 2-level operators or on the sine-cosine transform ([8], [9], [10]). In both cases, the number *M* of gate operations is smaller than 4^n.

Actually, all can be done with *CX* gates, $\{R_y(\gamma)$ gates, $\forall \gamma\}$ and $\{R_z(\gamma)$ gates, $\forall \gamma\}$. It can even be proven that, up to some previously defined maximum error, any unitary operator can be synthesized with *CX*, *H* and *T* gates ([1], [2]). In this case, the maximum number of operations depends on the predefined precision.

In conclusion, any unitary operator can be executed by a circuit that includes gates that operates on a single qubit and *CX* gates.

3 Emulator

The circuit presented in this paper allows emulating *n*-qubit quantum circuits designed with two types of quantic gates:

- $U(k)$ gates executing a unitary transformation *U* on qubit number *k*,
- $CU(l,k)$ gates executing a unitary transformation *U* on qubit number *k* under the control of qubit number *l*.

According to the conclusion of the preceding section, this set of gates is universal (it makes possible the synthesis of any unitary operation on the set of *n* qubits).

Comments

- The only considered binary gates are of type $CU(l,k)$. Other binary gates must be synthesized with $CU(l,k)$ and $U(k)$ gates. For example, a *SWAP* gate can be implemented with three *CX* gates ([2], Fig.4.7). Anyway, in an actual physical implementation, it is likely that the *SWAP* operation cannot be

executed in a single moment and must be decomposed into a sequence of more easily implementable operations.

- The same comment can be done about ternary gates. For example, a Toffoli gate can be implemented with *CX*, *H*, *T*, T^+ and *S* gates ([2], Fig.6.16).
- A limitation of the proposed emulator is that operations that involve different qubits cannot be executed simultaneously. They must be executed sequentially. This means that conclusions about the total execution time could be nonoptimized.

3.1 *U*(*k*) and *CU*(*l*,*k*) function computation

Consider an *n*-qubit register. Let *state* be a list of 2^n complex numbers that describe the current quantum state of the register. The following Python function computes the new quantum state of the register when executing the unitary transform *U*, defined by (11), on qubit number *k*, without modifying the state of the other qubits. The corresponding *N* by *N* matrix is

$$U(k) = I_{2^k} \times U \times I_{2^{n-k-1}}. \tag{17}$$

```
def U(k,u00,u01,u10,u11):
    m = 2**(n-k-1)
    p = 2**k
    for j in range (p):
        for i in range (2*m*j,(2*j+1)*m):
            a = u00*state[i] + u01*state[m+i]
            state[m+i] = u10*state[i] + u11*state[m+i]
            state[i] = a
```

This function updates the list *state* when executing the operation *U*(*k*) defined by the complex numbers u_{00}, u_{01}, u_{10} and u_{11}.

The function that updates the list *state*, when executing the unitary transform *U*, defined by (11), on qubit number *k*, under the control of qubit number *l*, without modifying the state of the other qubits, is easily deduced from the preceding function in which *ToBin* is a function that represents an integer as an *n*-bit binary number.

```
def CU(l,k,u00,u01,u10,u11):
    m = 2**(n-k-1)
    p = 2**k
    for j in range (p):
        for i in range (2*m*j,(2*j+1)*m):
            bin = ToBin(i,n)
            if bin[l] == 1:
                a = u00*state[i] + u01*state[m+i]
                state[m+i] = u10*state[i] + u11*state[m+i]
                state[i] = a
```

3.2 Digital implementation

The quantum state of a digital circuit that emulates an *n*-qubit quantum register is a memory that stores 2^n complex numbers. Assuming that each complex number is represented by a pair of *t*-bit numbers, the memory size is equal to $t \cdot 2^{n+1}$ bits. Even in the case of a few tenths of qubits, this could be a huge number, one of the challenges of quantum emulation.

The computations executed by the functions $U(k,u_{00},u_{01},u_{10},u_{11})$ and $CU(l,k,u_{00},u_{01},u_{10},u_{11})$, previously defined, are

$$next\_state(i) = u_{00} \cdot state(i) + u_{01} \cdot state(m+i),$$

$$next\_state(m+i) = u_{10} \cdot state(i) + u_{11} \cdot state(m+i). \tag{18}$$

If a dual-port memory is used, both relations can be computed at the same time. The symbol of a combinational arithmetic unit that computes (18) is shown in Fig.4.

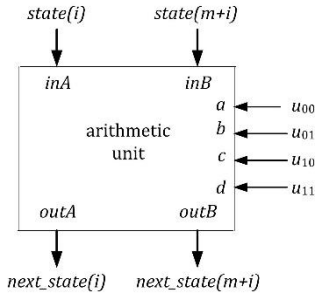

Figure 4 Arithmetic unit

The arithmetic unit of Fig.4 computes four products of two complex numbers, that can be executed in parallel, plus two additions of complex numbers in parallel. Each complex product, in turn, amounts to four products of two real numbers, that can be executed in parallel, plus an addition and a subtraction in parallel.

The module $|a|$ of all processed complex numbers $a = \alpha + i\beta$ is a probability and is smaller than or equal to 1, so that the real numbers α and β belong to the interval

$$-1 \leq \alpha, \beta \leq 1. \tag{19}$$

They are represented as t-bit fixed-point numbers:

$$\alpha = \alpha_1 \alpha_0 . \alpha_{-1} \alpha_{-2} a_{-(t-2)}. \tag{20}$$

The integer part of (20) includes two bits so that the numbers 1 and -1 can be exactly expressed as 01.00...0 and 11.00...0, respectively. Rounding operations, introducing some imprecision, must be executed in order that all processed numbers are represented with t bits. The computation time of the arithmetic unit (equations (18) and Fig.4) is equal to the delay of a parallel multiplier that computes the product of two t-bit signed numbers plus the delay of an adder that computes the sum of four t-bit signed numbers.

The structure of the proposed emulation circuit is shown in Fig.5. It consists of the following components.

- A dual-port memory that stores the quantum states, that is, 2^n 2t-bit words. The n-bit signals *address_A* and *address_B*, and the control signals w*rite_A* and *write_B*, permit to update the quantum state.
- The arithmetic unit that executes the operations (18). An additional 3-valued control signal *output*01 sets the arithmetic unit outputs to either (00.00...0, 00.00...0) when *output*01 = 2, (01.00...0, 00.00...0) when *output*01 = 1, or to the values computed by equations (18) when *output*01 = 0. It is used to initially store, within the dual-port memory, the coefficients that correspond to the basic state $|00 \ldots 0\rangle$.
- The control unit generates the n-bit signals *address_A* and *address_B*, the 1-bit control signals w*rite_A* and *write_B*, the 2-bit control signal *ouput*01 and a p-bit signal *address_M* that addresses the contents of the program memory. It reads from the program memory the operation code of the instruction to be executed. External input and output signals *start* and *ready* are used for communication purpose.
- The program memory stores a sequence of, at most, 2^p operations that will be sequentially executed. Each instruction (Fig.6) includes an operation code (*type*, *l*, *k*) and eight t-bit data that define the complex coefficients u_{00}, u_{01}, u_{10} and u_{11}.

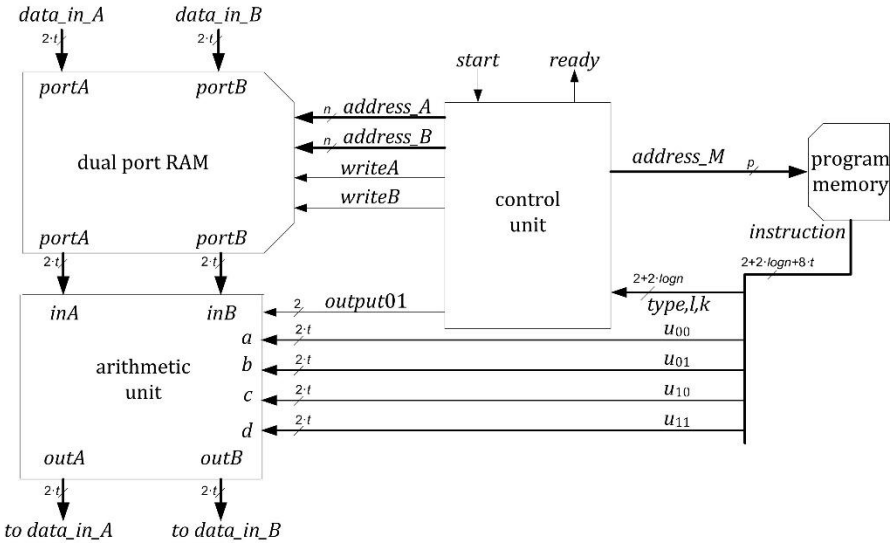

Figure 5 Block diagram

type	l	k	u_{00}_re	u_{00}_im	u_{01}_re	u_{01}_im	u_{10}_re	u_{10}_im	u_{11}_re	u_{11}_im

Figure 6 Instruction format

The emulation circuit executes four types of instructions.

- *INIT*: set the quantum state to the basic state $|00...0\rangle$.
- $U(k)$: execute a unitary operation on the qubit number k, where k belongs to the set $\{0, 1, 2, \ldots, n\text{-}1\}$.
- $CU(l,k)$: execute a unitary operation on the qubit number k, under the control of the qubit number $l \neq k$, where k and l belong to the set $\{0, 1, 2, \ldots, n\text{-}1\}$.
- *END*: execute the identity operation on the quantum state.

Assuming that l and k are encoded with *logn* bits and *type* with 2 bits, the instruction word size is

$$ws = 2 + 2 \cdot logn + 8 \cdot t. \tag{21}$$

As an example, the program memory contents that correspond to the circuit of Fig.3, initially set to the basic state $|0101\rangle$ with two additional X gates, is described by the following VHDL constant definition, with $t = 32$ and $logn < 4$:

```
constant program: memory :=
(
"00" & x"0" & x"0" & x"00000000" & x"00000000" & x"00000000" & x"00000000" & x"00000000"
                   & x"00000000" & x"00000000" & x"00000000", -- init
"01" & x"0" & x"1" & x"00000000" & x"00000000" & x"40000000" & x"00000000" & x"40000000"
                   & x"00000000" & x"00000000" & x"00000000", -- X(1)
"01" & x"0" & x"3" & x"00000000" & x"00000000" & x"40000000" & x"00000000" & x"40000000"
                   & x"00000000" & x"00000000" & x"00000000", -- X(3)
"01" & x"0" & x"0" & x"2d413ccc" & x"00000000" & x"2d413ccc" & x"00000000" & x"2d413ccc"
                   & x"00000000" & x"d2bec333" & x"00000000", -- H(0)
"10" & x"1" & x"0" & x"40000000" & x"00000000" & x"00000000" & x"00000000" & x"00000000"
                   & x"00000000" & x"00000000" & x"40000000", -- CRphi(1,0, pi/2)
"10" & x"2" & x"0" & x"40000000" & x"00000000" & x"00000000" & x"00000000" & x"00000000"
                   & x"00000000" & x"2d413ccc" & x"2d413ccc", -- CRphi(2,0, pi/4)
"10" & x"3" & x"0" & x"40000000" & x"00000000" & x"00000000" & x"00000000" & x"00000000"
                   & x"00000000" & x"3b20d79e" & x"187de2a6", -- CRphi(3,0, pi/8)
"01" & x"0" & x"1" & x"2d413ccc" & x"00000000" & x"2d413ccc" & x"00000000" & x"2d413ccc"
                   & x"00000000" & x"d2bec333" & x"00000000", -- H(1)
"10" & x"2" & x"1" & x"40000000" & x"00000000" & x"00000000" & x"00000000" & x"00000000"
                   & x"00000000" & x"00000000" & x"40000000", -- CRphi(2,1, pi/2)
"10" & x"3" & x"1" & x"40000000" & x"00000000" & x"00000000" & x"00000000" & x"00000000"
                   & x"00000000" & x"2d413ccc" & x"2d413ccc", -- CRphi(3,1, pi/4)
"01" & x"0" & x"2" & x"2d413ccc" & x"00000000" & x"2d413ccc" & x"00000000" & x"2d413ccc"
                   & x"00000000" & x"d2bec333" & x"00000000", -- H(2)
"10" & x"3" & x"2" & x"40000000" & x"00000000" & x"00000000" & x"00000000" & x"00000000"
                   & x"00000000" & x"00000000" & x"40000000", -- CRphi(3,2, pi/2)
"01" & x"0" & x"3" & x"2d413ccc" & x"00000000" & x"2d413ccc" & x"00000000" & x"2d413ccc"
```

```
                    & x"00000000" & x"d2bec333" & x"00000000", -- H(3)
"11" & x"0" & x"0" & x"00000000" & x"00000000" & x"00000000" & x"00000000" & x"00000000"
                    & x"00000000" & x"00000000" & x"00000000" -- end
);
```

The control unit is a finite state machine that sequentially reads the operation code from the program memory. It generates the dual-port memory addresses and write commands, the *output*01 signal transmitted to the arithmetic unit, and it updates the program memory address. It receives an external *start* order and generates a *ready* flag when an *END* type instruction is decoded. The following pseudocode program describes the behavior of the control unit.

```
wait on start
ready = false
address_M = 0
loop
   read_instruction
   if type == 0:
      outpu01 = 2
      {address_A = 2^(n-1), address_B = 0,
      write_A = 1, write_B = 1}
      for j in range(1, 2^(n-1)):
         output01 = 1
         {address_A = j+2^(n-1), address_B = j,
         write_A = 1, write_B = 1}
      address_M = address_M + 1
   elif type == 1:
      output01 = 0
      m = 2^(n-k-1), p = 2^k
      for j in range (p):
         for i in range (2*m*j,(2*j+1)*m):
            {address_A = i, adddress_B = m+i,
            write_A = 1, write_B = 1}
      address_M = address_M + 1
   elif type == 2:
      output01 = 0
      m = 2^(n-k-1), p = 2^k
      for j in range (p):
         for i in range (2*m*j,(2*j+1)*m):
            if i[l] == 1:
               {address_A = i, adddress_B = m+i,
               write_A = 1, write_B = 1}
      address_M = address_M + 1
   else:
      {write_A = 0, write_B = 0}
      ready = true
end loop
```

The branches of the *if* construct include at most 2^{n-1} steps. Thus, the execution time of an instruction is equal to

$$T_{inst} = 2^{n-1} \cdot T_{R\text{-}W}, \tag{22}$$

being $T_{R\text{-}W}$ the duration of a *READ-WRITE* cycle.

3.3 Input and output interfaces

Additional circuits are necessary to connect the emulator to a host processor. The program memory structure is shown in Fig.7. It is made up of eight ram blocks that store 2^p t-bit words (the unitary matrix coefficients) plus a ram block that stores 2^p cs-bit words with $cs = 2+2 \cdot logn$ (the operation codes). All processed data are t-bit length, so that the storing of an m instruction program ($m \le 2^p$) is executed in $9 \cdot m$ steps. The host processor generates (p+4)-bit addresses: bits 0 to 3 select one of the nine ram blocks and bits 4 to p-1 select a word within the selected block. The processor also generates control signals: *load_program*, set to 1 during the loading of the program memory, and to 0 during the execution of the emulation process; *start* set to 1 to initiate the emulation process; *write* set to 1 during write cycles of the program memory. The host processor reads (or is interrupted by) a flag *ready* generated by the control unit at the end of the emulation process.

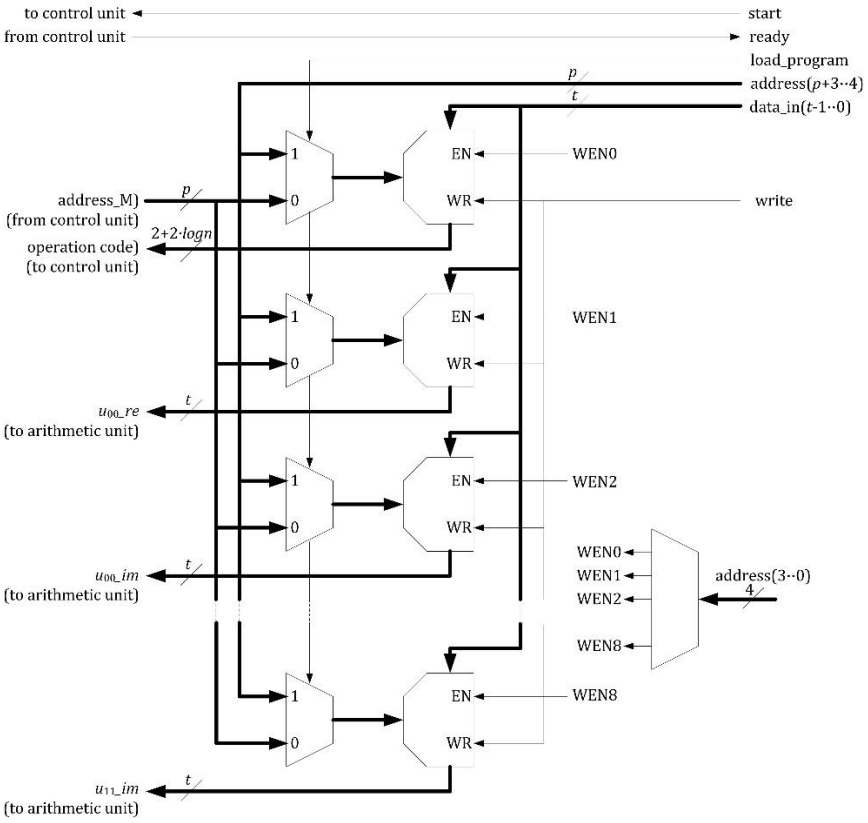

Figure 7 Program memory

When the emulation process is completed, additional connection circuits (multiplexers) permit to transfer the emulation results to the host processor. A control signal *read_state*, generated by the processor, replaces *address_A* by an external *n*-bit address (Fig.8), and inhibits the *write_A* and *write_B* commands. The *port_A* output of the dual-port memory is connected to the external data output through a multiplexor that allows reading separately the real and the imaginary part of the selected memory word (Fig.8, *re*/*im*).

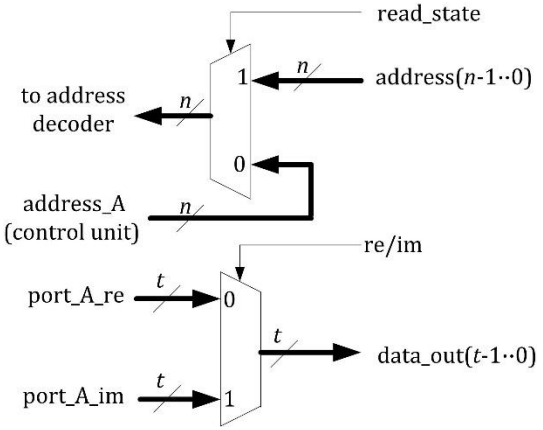

Figure 8 Dual-port memory interface

To summarize, the proposed emulator has the following parameters:

n : number of qubits,
t : number of bits of a fixed-point number,
p : number of bits of a program memory address,
logn: number of bits of a qubit index (from 0 to *n*-1),
$cs = 2 + 2 \cdot logn$: number of bits of an instruction operation code (assumed to be smaller than *t*),
$ws = cs + 8 \cdot t$: number of bits of an instruction.

Its input and output signals, apart from *reset* and *clk*, are

- *address*: an m-bit address, with $m \geq n$ and $m \geq p+4$, generated by the host microprocessor. In *write* mode, bits 0 to 3 select one of the 9 ram blocks (Fig.7) and bits 4 to $p+3$ select one of the 2^p words within the selected block. In *read* mode, bits 0 to $n-1$ select one of the 2^n words within the dual-port memory (Fig.5 and Fig.8).
- *data_input*: a t-bit word generated by the host processor. In *write* mode it is either an operation code (*type*, *l*, *k*) or a t-bit number to be stored within one of the nine ram blocks (Fig.7).
- *data_out*: a t-bit word transmitted to the host processor. In *read* mode it is either the real part or the imaginary part of the quantum state coefficient selected by the n-bit address (Fig.8).
- Control signals generated by the microprocessor:
    - *load_program*: set to 1 during the storing of the quantum program and to 0 during the quantum program execution;
    - *write*: set to 1 during *write* cycles;
    - *read_state*: set to 1 during *read* cycles;
    - *re/im*: in *read* mode it selects either the real part or the imaginary part of a quantum state coefficient;
    - *start*: order to start the quantum program execution.
- *ready*: a state signal transmitted to the host processor (a flag) indicating the termination of the quantum program execution.

A complete VHDL descriptions, including four test benches, is available in [21].

References

[1] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge, U.K.: Cambridge Univ. Press, 2010.

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

[3] Frank Arute, *Quantum supremacy using a programmable superconducting processor*, Nature, Vol 574, 24 october 2019.

[4] Quantum. Accessed: Oct., 2022. [Online]. Available: https://quantum-computing.ibm.com

[5] Roth, T. *et al.* (2022), «The Transmon Qubit for Electromagnetics Engineers», *IEEE Antennas and Propagation Magazine.*

[6] Koch, J. *et al.* (2007), «Charge insensitive qubit design derived from the Cooper pair box», *Physical Review,* n.º 76, octubre.

[7] Kockum, A. F. y Nori, F. (2019), «Quantum Bits with Josephson Junctions», en F. Tafuri (ed.), *Fundamentals and Frontiers of the Josephson Effect,* capítulo 17, Springer Series in Materials Science 28.

[8] Li, C.; Roberts, R., y Yin, X. (2013), «Decomposition of unitary matrices and quantum gates», *International Journal of Quantum Information,* n.º 11, 1350015.

[9] Sutton, B.D. Computing the complete CS decomposition. Numer. Algor. 2009 50, 33–65.

[10] Shende, V.; Bullock, S.; Markov, I. Synthesis of Quantum Logic Circuits. Comput.-Aided Des. Integr. Circuits Syst. IEEE Trans. 2006, 25, 1000–1010

[11] Negovetic G., Perkowski M., Lukac M., Buller A., "Evolving quantum circuits and an FPGA-based Quantum Computing Emulator," 5th International Workshop Boolean Problems, Freiberg, Germany, September 2002.

[12] M. Fujishima, “FPGA-based high-speed emulator of quantum computing,” in *Proc. IEEE Int. Conf. Field-Programmable Technol.*, 2003, pp. 21–26

[13] A.U. Khalid, Z. Zilic, and K. Radecka, “FPGA emulation of quantum circuits,” in *Proc. IEEE Int. Conf. Comput. Des. VLSI Comput. Processors*, 2004, pp. 310–315

[14] M. Aminian, M. Saeedi, M. S. Zamani, and M. Sedighi, “FPGA-based circuit model emulation of quantum algorithms,” in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, 2008, pp. 399–404

[15] A. Silva and O. G. Zabaleta, “FPGA quantum computing emulator using high level design tools,” in *Proc. IEEE 8th Argentine Symp. Conf. Embedded Syst.*, 2017, pp. 1–6.

[16] Y. H. Lee, M. Khalil-Hani, and M. N. Marsono, “An FPGA-based quantum computing emulation framework based on serial-parallel architecture,” *Int. J. Reconfigurable Comput.*, vol. 2016, Apr., 2016, Art. no. 5718124.

[17] N. Mahmud, E. El-Araby, and D. Caliga, “Scaling reconfigurable emulation of quantum algorithms at high precision and high throughput,” *Quantum Eng.*, vol. 1, no. 2, 2019, Art. no. e19

[18] T. Suzuki, T. Miyazaki, T. Inaritai, and T. Otsuka, “Quantum AI simulator using a hybrid CPU-FPGA approach,” 2022. Accessed: Oct. 2022. [Online]. Available: https://arxiv.org/abs/2206.09593

[19] Esam El-Araby et al., Towards Complete and Scalable Emulation of Quantum Algorithms on High-Performance Reconfigurable Computers, IEEE Tansactions on computers, Vol. 72, No. 8, august2023.

[20] J. Pilch and J. Długopolski, “An FPGA-based real quantum computer emulator,” *J. Comput. Electron.*, vol. 18, no. 1, pp. 329–342, 2019

[21] https://github.com/Marcombo/Circuitos-y-algoritmos-cuanticos