ADVANCES IN CYBER-PHYSICAL SYSTEMS

Vol. 3, No. 2, 2018

FPGA-BASED DIGITAL QUANTUM COPROCESSOR

Valerii Hlukhov, Bohdan Havano,

Lviv Polytechnic National University, 12, S. Bandery Str., Lviv, 79013, Ukraine
Authors' e-mail*: glukhov@polynet.lviv.ua*

Submitted on 01.12.2018

***Abstract*: Classical quantum computer is an analog probabilistic computer. The digital quantum computer that can be implemented in FPGA has been described in the article. Digital quantum coprocessor is designed to implement algorithms for execution on the analog quantum computers. A digital quantum coprocessor operates under the control of a classical computer and together they are digital quantum computer. The digital quantum coprocessor is a set of digital units called digital qubits that have multi-bit data input and single-bit data output each. The digital qubit is a wave function calculator. There is a pseudo random number generator (PRNG) in the described digital qubit to generate the probabilistic output bit. Single-bit qubit output qb has been formed by the probable reduction of its multi-bit result x to single-bit (0 or 1) according to result value x and pseudo random code k (x is an angle which determines the position of normalized vector with length 1 in polar grid, it is a result of multi-bit input data calculation). The decision about output state has been made after the functional conversion of the qubit multi-bit result x to result probability $p = \sin^2 x$ and subsequent comparison p with the pseudo random code k. Qb=1 when k < p. In the article, the inverse variant of decision option with p = arcsin (sqrt (k)) and qb = 1 when x > p has been described. This variant allows to use one PRNG for all digital qubits. Possible schemes for digital qubit and digital quantum coprocessors based on them have been discussed in the paper. The presentation of data in digital qubits and the basic operations they perform have been also considered. The results of the simulation of a four-qubit digital quantum coprocessor and the results of the qubit implementation in FPGA have been presented.**

***Index Terms*: digital quantum coprocessor, digital qubit**

I. INTRODUCTION

Classic quantum computers (QC) are analog computing devices. As for any analog calculator digital equivalent can be built for the classical analog quantum computer. It will successfully solve the same task. Such digital quantum computers form another class of computing devices – along with the devices with an access to data by addresses (ROM, RAM) and by content (associative memory, cache) form a class of devices with an access to data by probability [24]. The digital quantum computer that can be implemented in FPGA is described in the article. Digital quantum coprocessor is designed to implement algorithms for execution on the analog quantum computers. A digital quantum coprocessor operates under the control of a classical computer and together they are digital quantum computer. The digital quantum coprocessor is a set of digital units called digital qubits that have multi-bit data input and single-bit data output each. The digital qubit is a wave function calculator. There is a pseudo random number generator (PRNG) in the described digital qubit to generate the probabilistic output bit.

Possible schemes for digital qubit and digital quantum coprocessors based on them are discussed in the paper. The presentation of data in digital qubits and the basic instructions they perform are also considered. The results of the simulation of a four-qubit digital quantum coprocessor and the results of the qubit implementation in FPGA are presented.

In the literature, you can find the definition of a quantum computer – a computing device that uses the phenomena of quantum mechanics (quantum superposition, quantum entanglement) to transfer and process data. A quantum computer (as opposed to a normal classical computer) does not operate with bits (capable of accepting either 0 or 1), but qubits, which have values of both 0 and 1 simultaneously. As a result, it is possible to process all possible states simultaneously, attaining gigantic superiority over classical computers in a number of algorithms [2]. Often, the superposition principle is treated as such, which allows you to save all 2n numbers from 0 до 2n-1 in the n-qubit register at the same time. However, this statement is misleading: since the result of measuring the state of a quantum register is always one of the possible basic states, the maximum available amount of information that can be obtained from one qubit is one bit, as in the classic case. A full-fledged universal quantum computer is still a hypothetical device [2], however, recently there have been reports on the creation of quantum computers with 72 qubits [4, 2]. Also 2048 qubit The D-Wave QPU [31] is well known. It is built from a lattice of tiny loops of the metal niobium, each of which is one qubit (shown on the next page, in red). Below temperatures of 9.2 kelvin, niobium becomes a superconductor and exhibits quantum mechanical effects. When in a quantum state, current flows in both directions simultaneously, which means that the qubit is in superposition–that is, in both a 0 and a 1 state at the same time. At the end of the problem-solving process, this superposition collapses into one of the two classical states, 0 or 1.

In computational complexity theory, bounded-error quantum polynomial time (BQP) is the class of decision problems solvable by a quantum computer in polynomial

time, with an error probability of at most 1/3 for all instances.

The number of qubits in the computer is allowed to be a polynomial function of the instance size. For example, algorithms are known for factoring an n-bit integer using just over 2n qubits (Shor's algorithm [10]).

Usually, computation on a quantum computer ends with a measurement. This leads to a collapse of quantum state to one of the basis states. It can be said that the quantum state is measured to be in the correct state with high probability.

Quantum computers have gained widespread interest because some problems of practical interest are known to be in BQP, but suspected to be outside P. Some prominent examples are:

- Integer factorization (Shor's algorithm [10])
- Discrete logarithm
- Simulation of quantum systems
- Approximating the Jones polynomial at certain roots of unity [35]

Research in the field of quantum computing creates new challenges for cryptography. When using a quantum computer and Shor's algorithm [10] currently known public key cryptographic algorithms may be compromised.

This is estimated to take at least 4000 qubits (but could be more depending upon the algorithm used [34]) to factor a 2048 bit number in order to break RSA encryption with a 2048 bit key.

Today, standardization centers such as NIST and ETSI are already conducting research in creating standards for post-quantum cryptography [6, [5]. On this subject, international conferences and seminars are held, including those in Ukraine [7].

Real quantum computers are analog and probabilistic devices. Their implementation is a very difficult and expensive task. All this underlines the relevance of work on the study of the characteristics of quantum computers, the creation of their digital models and digital samples, as well as the training of specialists to work with them.

II. QUANTUM COMPUTER BASIS

If a classic computer at any moment can be in exactly one of the states $|0\rangle, |1\rangle, ..., |N-1\rangle$ (Dirac notation), then QC at each moment is simultaneously in all these basic states, and in each $|j\rangle$ state – with its complex amplitude l_j. This quantum state is called "quantum superposition" of these classical states and is denoted as the wave function $|y\rangle = \sum_{j=0}^{N-1} l_i |j\rangle$. In this case, the sum of all probabilities $P = \sum_{j=0}^{N-1} l_i^2 = 1$ [2].

The quantum state $|y\rangle$ can change in time in two fundamentally different ways: by unitary quantum operation and by measurement [4].

Any unitary transformation of the wave function $|y\rangle$ can be represented as a simple displacement of a point over the surface or sphere of unit radius (Bloch sphere for complex amplitudes, Fig. 2, [8]), or a circle of unit radius (for real amplitudes. In this case, the position of the vector at an angle of 0 radian corresponds to the state $|0\rangle$, and at an angle $\pi / 2$ radian – state $|1\rangle$). That is, the condition P = 1 is fulfilled both for the input and for the output with a unitary transformation. There are objects whose behavior is described by wave functions; therefore, quantum gates can be implemented on the basis of these objects [1], which, in turn, can serve as the element base for the construction of QC. The examples of such objects are solid-state quantum dots on semiconductors [2], where the directions of the electronic spin at a given quantum dot are used as logical qubits. The control is carried out by external potentials or a laser pulse.

It is also possible to realize the calculation of wave functions by digital methods – software or hardware (for example, on the FPGA).

The state of one qubit is written in the QC as $A|0\rangle + B|1\rangle$ – this is called a quantum superposition. For the Bloch sphere (Fig. 2) the amplitudes $A = cos(q/2)$ and $B = e^{ij}\, sin(q/2)$, for a unit circle – $A = cosq$ and $B = sinq$, respectively. Then, for the unit circle $|Y\rangle = cosq|0\rangle + sinq|1\rangle$. If the qubit is in a state $1/\sqrt{2}|0\rangle + 1/\sqrt{2}|1\rangle$, then as a result of the measurement with probability $(1/\sqrt{2})^2 = 1/2$ it can be determined as $|0\rangle$ and with the same probability as $|1\rangle$ (it is the undefined state of qubit). The indefinite state of two qubits is written as $1/2|00\rangle + 1/2|01\rangle + 1/2|10\rangle + 1/2|11\rangle$ – two qubits can be in each of their states with probability $(1/2)^2 = 1/4$.

The vector representation of a single qubit is $|a\rangle = v_0|0\rangle + v_1|1\rangle \rightarrow \begin{bmatrix} v_0 \\ v_1 \end{bmatrix}$,

The vector representation of two qubits is $|ab\rangle = a \otimes b = v_{00}|00\rangle + v_{01}|01\rangle + v_{10}|10\rangle + v_{11}|11\rangle \rightarrow \begin{bmatrix} v_{00} \\ v_{01} \\ v_{10} \\ v_{11} \end{bmatrix}$.

The action of the gate on a specific quantum state is found by multiplying the vector $|ab\rangle$, which represents the state by the matrix U representing the gate $U|ab\rangle$ [1].

In atomic theory and quantum mechanics, an atomic orbital is a mathematical function that describes the wave-like behavior of either one electron (in Fig. 3) or a pair of electrons in an atom [32]. So electron spin in Fig. 1 can be illustrated by a vector on a Bloch sphere in Fig. 2.

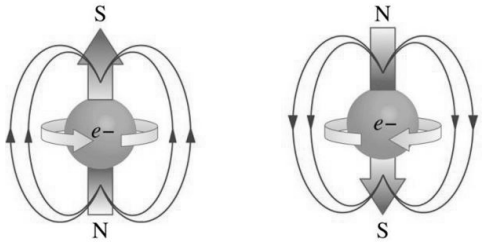


Fig. 1. An electron spin

The scientists were able to manipulate how long it would take for the electron to flip its spin and emit a photon – from one to 20 nanoseconds [33].

QC and classical computers tend to work together [12]. In addition to [12], it should be noted that in algorithms for QC a significant part of the calculations is performed by classical computers. For example, in Shor's algorithm [10, 11], the share of classic computers is the tabulation of the function $f(x) = t^x \bmod M$, where t is a prime number, M is a large number that needs to be factorized, x = 0, 1, …, N-1, N is about 2 times bigger than M. Also, a classical computer is engaged in final processing of the results and checking their reliability and suitability (QC is a probabilistic computer, and it gives the correct result with a certain probability).

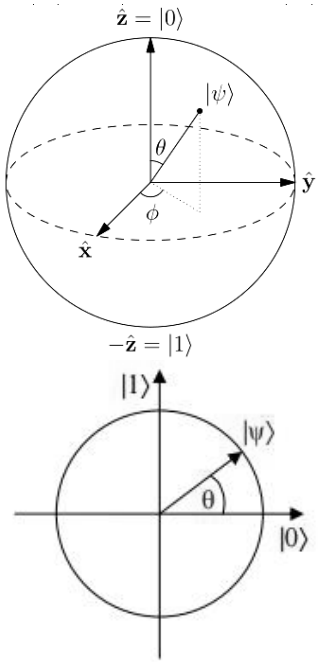


Fig. 2. Representation of a qubit in the form of a Bloch sphere for complex amplitudes (above) and a unit circle for real ones (below)

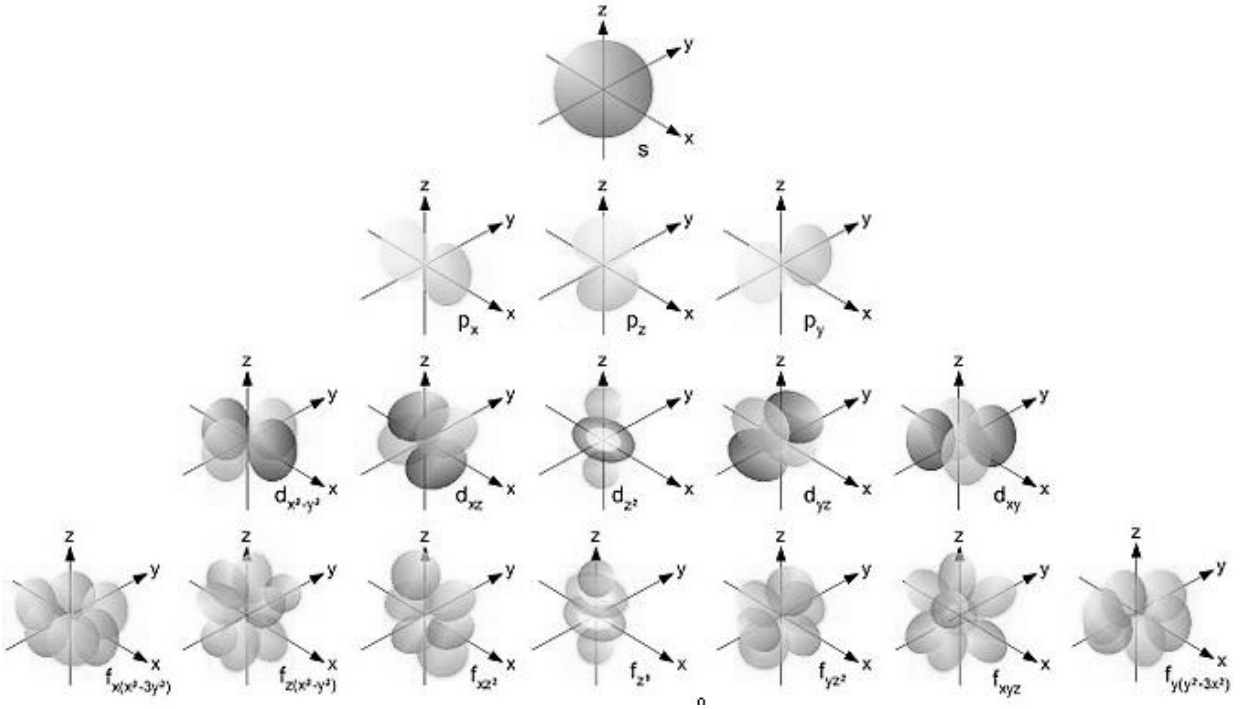


Fig. 3. The shapes of the first s-, p-, d- and f- atomic orbitals

III. QUANTUM GATES

Quantum gate (quantum logic element) is the basic element of a quantum computer, which transforms the input states of qubits in the output ones according to a certain law. It differs from ordinary logic gates by working with qubits, and, therefore, obeys quantum logic. Quantum gates, unlike many classical ones, are always reversible.

In classical computers, a set of logical two-input gates AND, OR and one-input gate NOT create a functionally complete system of functions of the algebra of logic (basis), which allows to build all possible logic schemes on their basis. Similarly, a set of quantum gates, consisting of C-NOT gate and all one-input gates, also create a basis that allows the creation of circuits of different QC, for example, for quantum Fourier transformation [14]. Unlike logic gates that process bits, quantum gates handle qubits (varying wave function coefficients that describe the state of a qubit).

There are software tools that allow you to create circuits from quantum gates and simulate their work [13].

Table 1

Preparation for factoring

			j=T=2	j=T=3	j=T=4	j=T=5	j=T=6	j=T=7
x	2^x	$2^x \bmod 15$	xmod2	xmod3	xmod4	xmod5	xmod6	xmod7
0	1	1	0	0	0	0	0	0
1	2	2	1	1	1	1	1	1
2	4	4	0	2	2	2	2	2
3	8	8	1	0	3	3	3	3
4	16	1	0	1	0	4	4	4
5	32	2	1	2	1	0	5	5
6	64	4	0	0	2	1	0	6
7	128	8	1	1	3	2	1	0
8	256	1	0	2	0	3	2	1
9	512	2	1	0	1	4	3	2
10	1024	4	0	1	2	0	4	3
11	2048	8	1	2	3	1	5	4
12	4096	1	0	0	0	2	0	5
13	8192	2	1	1	1	3	1	6
14	16384	4	0	2	2	4	2	0
15	32768	8	1	0	3	0	3	1
16	65536	1	0	1	0	1	4	2
17	131072	2	1	2	1	2	5	3
18	262144	4	0	0	2	3	0	4
19	524288	8	1	1	3	4	1	5
20	1048576	1	0	2	0	0	2	6
21	2097152	2	1	0	1	1	3	0
22	4194304	4	0	1	2	2	4	1
23	8388608	8	1	2	3	3	5	2
		N_j	5	1	5	1	1	0
		$F_J=x_{max}/T$	12	8	6	5	4	4
		$\lambda j=Nj/Fj$	0,42	0,13	0,83	0,20	0,25	0,00
		$pj=(\lambda j)^2$	0,17	0,02	0,69	0,04	0,06	0,00

IV. EXAMPLE OF USING A QUANTUM COMPUTER

The example shows how the number 15 decomposes; it was succeeded in the QC prototype [15]. The problem of decomposing on the factors of the number M reduces to the problem of determining the period of a function $y = 2^x mod M$. The course of calculations of the period for the number M = 15 is summarized in the Table **1** (it is clear that the period is 4 because $2^x mod 15 = 1, 2, 4, 8, 1, \ldots$). When finding the desired period T, the frequency of finding value $xmodT=0$, when the function $y = 2^x modM = 1$ is determined by viewing the table values with all possible periods T (T = 2, 3, 4, …, 2M): the number Nj of detected values $xmodT=0$ is counted, the frequency F_i, with which $xmodT=0$ is repeated for all selected periods T is determined, the situation $xmodT=0$ detection frequency l_i is determined and it is verified that the sum of the probabilities p_i of the wave function is 1. The wave function has the form $|y\rangle = \sum_{j=2}^{7} l_i |T_j\rangle$. To calculate the frequencies F_i in the Shor's algorithm [10], [11] the quantum Fourier transform is used [14].

After completing the work of the QC, when reading its state, a correctly defined period T = 4 will be read with the highest probability p_4 = 0.69. The result is checked by a classical computer, which it then uses to determine the multipliers of the number 15.

In quantum algorithms, usage of complex amplitudes and the representation of vectors using the Bloch sphere (Fig. 2) is the most common case. But the use of real amplitudes and the representation of vectors by means of a unit circle is also popular. For example, the American Mathematical Society used the representation of states of qubits in this way (Fig. 4 [26]), illustrating the execution of a quantum computer factorization of numbers by Shor's algorithm [10].

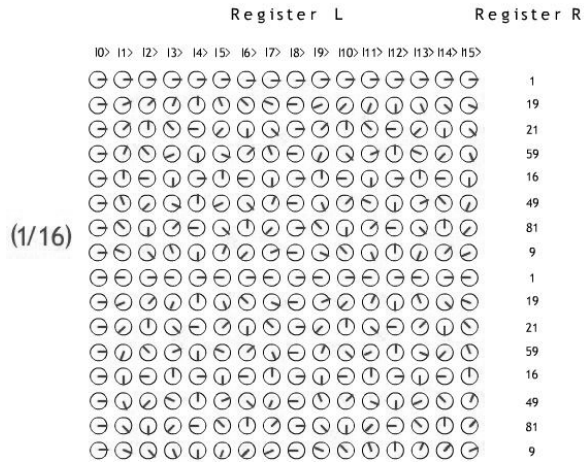


Fig. 4. The change of the state of qubits by circular diagrams

V. AN ERROR RATE FOR QUANTUM COMPUTERS

A minimum error rate for quantum computers needs to be in the range of less than 1 %, coupled with close to 100 qubits. Google seems to have achieved this so far with 72-qubit Bristlecone and its 1 % error rate for readout, 0.1 % for single-qubit gates, and 0.6 % for two-qubit gates. Quantum computers will begin to become highly useful in

solving real-world problems when we can achieve error rates of 0.1–1 % coupled with hundreds of thousand to millions of qubits. According to Google, an ideal quantum computer would have at least hundreds of millions of qubits and an error rate lower than 0.01 %. That may take several decades to achieve, even if we assume a "Moore's Law" of some kind for quantum computers [36].

VI. SOFTWARE IMPLEMENTATION OF DIGITAL QUANTUM COMPUTERS – MICROSOFT QUANTUM KATAS

Today, quantum programming plays a major role in the research of leading companies in the world. The existence of a quantum computer is no longer a science fiction, but rather a matter of time. One of these companies is Microsoft, it has not built a quantum computer yet, and even if it were, it would be very unprofitable to perform certain calculations. Therefore, a new Q # programming language was implemented, which solves the problem of the inability to teach programmers to encrypt quantum computers on a real device. It also allows you to encode a quantum computer program with the confidence that this code will work after the hardware has been created.

What is Q#? According to Microsoft: Q# is a scalable, multi-paradigm, domain-specific programming language for quantum computing. So, what do these terms actually mean? Let us dive into the details. Scalable: Q# allows us to write code that can be executed on machines of varying computing abilities. We can use it to simulate a few Qubits on our local machine, or even thousands of Qubits for an enterprise level application. Multi-paradigm: Q# is a multi-paradigm programming language. It supports both functional and imperative programming styles. Domain-specific: Q# is a programming language for quantum computing. It is to be used for writing algorithms and code snippets that are executed on quantum processors [22].

Development tools are integrated into the well-known platform programmers, Visual Studio. Features include a local simulator and cost estimator, so early experiments can determine how many qubits are needed, for example. And this is really important, because when you are studying a new programming language, you immediately want to try new knowledge in practice. Therefore, programmers do not need access to a quantum computer to use Q # and applications. Q # has a pretty good documentation and environment setup is a fairly simple process. Each programmer can access it [23].

This solution has already several libraries, which have many operations and functions, such as quantum I / O, and interact with the usual data. Currently, the simulator is a sbo machine that consists of 30 qubits (with 40 qubits at Azure in the near future). Using cost estimates, users can launch programs at an early stage on a scale, display complex programs, and see the time delay of the scheme, the use of qubits, and detailed error information.

This is an opportunity, as well as a problem at the same time. Quantum physics is a strange area of teleportation and the likelihood that it does not adhere to the rules we are familiar with. Many people do not understand quantum mechanics. Developers are no exception, people who will have to program quantum computers for application use.

However, programmers do not necessarily have to know quantum physics, in order to write programs on Q #. Just try creating a program, or use ready-made test software from libraries, and follow the compilation result. Launching Q # using the target equipment for specific challenges with cloud resource invocation. The programming environment offered by Microsoft, Q #, is very similar to what programmers are using in the present with other programming languages.

What is Quantum Katas? This is a tool by which Microsoft aims to encourage and teach programmers to use Q #. In general, it is a set of freely available training programs that can help programmers get to know Q #. Each task is a task for solving a given issue related to the topic of quantum computing. This tool is not intended to solve it at one time, or not wrongly. On the contrary, mistakenly, programmers will go deeper into the essence of quantum programming. And it really works, because when passing this program, people with zero knowledge of quantum physics, can begin to program quantum computers [23].

With this approach, programmers begin to relate to quantum programming from the other side. Why? Because there is a transition from obscure quantum theories to practical solutions, not having a quantum computer, they see the result, it works or not. Q # and Quantum Katas build a so-called proxy between programmers and quantum physics, enabling programmers to understand the essence of the work of quantum computing and what choice they are spoilt for [23].

By downloading the Quantum Development Kit, part of which is Q #, interested programmers cannot just use it and learn, but also refine it. In this way, promoting the community of developers, as well as expanding the database, adding new algorithms and documentation will definitely improve the future work of developers with Q #.

An important point is that even non-experienced programmers can begin to work with Q #, so begin to understand quantum computing by experimenting with Quantum Development Kit. This is useful not only for them, but also for quantum physics in general. Since quantum physics is considered difficult to explain and understand, and it is difficult to compare with classical physics, this is yet another step for understanding quantum physics. Especially considering that in quantum physics it is difficult to see any results of experiments, taking into account its features, then in this case, programmers still see the result of their quantum program [23].

Microsoft hopes Q # will be the point of convergence of quantum computing and programmers. Programmers do not need to know quantum physics or quantum theories. No need to spend years studying them. Simply write the Q # program, run it, and look at the result.

Learning programmers will take a lot of time, but Microsoft and their Q # is a big step forward.

Let's move on to practice and examples.

In the classical theory of computation, logic gates are used to perform operations on bits. For the manipulation of the Qbits used similar designs – quantum gates. For example, the NOT gate performs transformations 0 → 1 and 1 → 0. The quantum gate NOT is similar to its classical ancestor: it performs transformations $|0\rangle \rightarrow |1\rangle$ and $|1\rangle \rightarrow |0\rangle$. This means that after passing such a gate, the qubit from the state $a|0\rangle + b|1\rangle$ changes to the state $a|1\rangle + b|0\rangle$. The gate NOT can be written in the form of a matrix (X), which swaps 0 and 1 in the state matrix

$$X \equiv \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

As you can see, $X|0\rangle = |1\rangle$, and $X|1\rangle = |0\rangle$:

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0\times1 & + & 1\times0 \\ 1\times1 & + & 0\times0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle$$

$$X|1\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle$$

Since $|0\rangle$ and $|1\rangle$ in vector form are written as $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$ and $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$. The first column of the matrix X can be considered as a transformation of the vector $|0\rangle$, and the second column as a transformation of the vector $|1\rangle$.

It would seem that the difference from the classical case is not so great. But do not forget what we said in the previous section: measuring the state of a qubit is probabilistic in nature. As it is known from elementary probability theory, the sum of the probabilities of the complete group of incompatible events is equal to one. Therefore, $|a|^2 + |b|^2 = 1$ for the quantum state, $a|0\rangle + b|1\rangle$.

It follows that not all imaginable gates can exist in the quantum world. Here is one of the limitations: the condition for the normalization of the quantum state of a qubit, $|a|^2 + |b|^2 = 1$, must be observed both before and after the passage of the gate. In terms of matrix algebra, this condition will be satisfied if the matrix is unitary.

VII. IMPORTANT GATES: GATE Z AND HADAMARD GATE.

The Z gate works very simply: it saves the component $|0\rangle$ and changes the sign of the component $|1\rangle$. It can be written as a matrix $Z \equiv \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$ which converts the states of qubits as follows $|0\rangle \rightarrow |0\rangle$, $|1\rangle \rightarrow -|1\rangle$ (remember that the first column of the matrix describes the transformation of the vector $|0\rangle$, the second – the transformation of the vector $|1\rangle$).

The Hadamard gate creates a superposition of states $|0\rangle$ and $|1\rangle$, similar to those discussed above. Its matrix notation looks like this: $H \equiv \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$ which corresponds to the following state transformations of qubits: $|0\rangle \rightarrow \frac{|0\rangle + |1\rangle}{\sqrt{2}}$; $|1\rangle \rightarrow \frac{|0\rangle - |1\rangle}{\sqrt{2}}$. Also, for this gate the following designation is used 

VIII. ANOTHER IMPORTANT GATES

We have already dismantled the work of the NOT gate. Next in line is the CNOT (controlled-NOT, "NOT controlled") gate (Fig. 5). At its entrance serves two qubits. The first is called the manager, the second – managed. If the controlling qubit is $|0\rangle$, then the state of the controlled qubit does not change. If the controlling qubit is $|1\rangle$, then the NOT operation is applied to the managed qubit.

The CNOT operation can be interpreted in several ways. Like the gates X, Z and H, it can be written in matrix form, which is denoted by the letter U.

$$U_{CN} \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Also for this gate the following designation is used (the upper part corresponds to the control qubit, the lower part – to the controlled one):

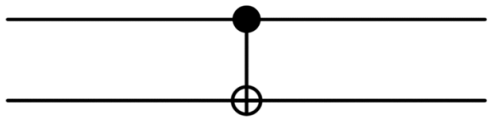


Fig. 5. CNOT gate

There are four wonderful states of Bell. One of them ($|j^+\rangle$) will be used in the quantum program below. Let's consider it.

$$|j^+\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

Suppose we measure the state of the first qubit. Result $|0\rangle$ we get with probability $\left|\frac{1}{\sqrt{2}}\right|^2$. This means that the state after the measurement $|j'\rangle = |00\rangle$, or $|1\rangle$ with the same probability (0.5), and the state after the measurement

$|j'\rangle = |11\rangle$. For the curious incident, we give the complete set of Bell states (they are the simplest cases of quantum entanglement):

$$|j^{\pm}\rangle = \frac{|00\rangle \pm |11\rangle}{\sqrt{2}} \quad |j^{\pm}\rangle = \frac{|01\rangle \pm |10\rangle}{\sqrt{2}}$$

Now, suppose that we measured the state of the second qubit. According to the same reasoning, after measuring the steam will be in the state $|00\rangle$ or $|11\rangle$. If, after this, we decide to measure the state of the first qubit, the probabilities will no longer be equal to 0.5. We will get $|0\rangle$ with a probability of 1 or 0, depending on what the measurement result was. Here it is important to understand that these results are related. The first to notice this were Albert Einstein, Boris Podolsky and Nathan Rosen (therefore, these states are sometimes called "EPR pairs"). Subsequently, their theory was developed by John Bell.

One final observation: Bell states can be generated using the Hadamard gate and the CNOT gate (Fig. 6). Hadamard's gate puts the first qubit into a superposition state. Then this qubit is fed to the control input of the CNOT gate. Table 2 shows how this process can be represented using a circuit diagram.

Table 2

Factorization

In	Out
$\|00\rangle$	$\frac{\|00\rangle + \|11\rangle}{\sqrt{2}} = \|b_{00}\rangle$
$\|01\rangle$	$\frac{\|01\rangle + \|10\rangle}{\sqrt{2}} = \|b_{01}\rangle$
$\|10\rangle$	$\frac{\|00\rangle + \|11\rangle}{\sqrt{2}} = \|b_{10}\rangle$
$\|11\rangle$	$\frac{\|01\rangle + \|10\rangle}{\sqrt{2}} = \|b_{11}\rangle$

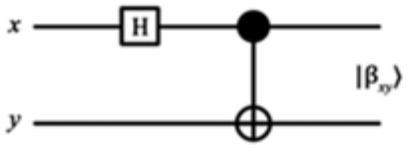


Fig. 6. Bell states generation

Let's see how to use it with Q#.

```
namespace Quantum.Bell
{
  open Microsoft.Quantum.Primitive;
  open Microsoft.Quantum.Canon;

  operation Set (desired: Result, q1: Qubit) : ()
  {
    body
    {
      let current = M(q1);
      if (desired != current)
      {
        X(q1);
      }
    }
  }
}
```

This operation converts our qubit to the selected (by us) state – 0 or 1. First, we measure the qubit (this operation is denoted by the letter M), and it collapses into the state 0 or 1. If the measured state does not correspond to the desired one, we change it with the help of the gate NOT, X. Otherwise, nothing needs to be done [21].

```
operation BellTest (count : Int, initial: Result) : (Int,Int)
{
  body
  {
    mutable numOnes = 0;
    using (qubits = Qubit[1])
    {
      for (test in 1..count)
      {
        Set (initial, qubits[0]);
        let res = M (qubits[0]);
        // Count the number of ones we saw:
        if (res == One)
        {
          set numOnes = numOnes + 1;
        }
      }
      Set(Zero, qubits[0]);
    }
    // Return number of times we saw a |0> and number
of times we saw a |1>
    return (count-numOnes, numOnes);
  }
}
```

This small piece of code is intended to test the operation we just wrote. This is a very simple program: it checks that the qubit has been transferred to the state we need.

To do this, it takes a measurement in a cycle and counts the number of results 1 using the variable numOnes.

The entry "Qubit [1]" means to "create an array of qubits from one element". The array elements are indexed from scratch. To select two qubits (we will need to do this later), we need to write "Qubit [2]". Qubits in such an array correspond to numbers 0 and 1.

In the for-loop, we set the qubit allocated for a specific initial state – One or Zero (in the Driver.cs file, to which we will soon move on, this is done explicitly). We measure this state, and if it is One, we increase the value of the counter by one. The function then returns the number of monitored states One and Zero. At the end, the qubit is placed in the Zero state (just to leave it in some known state) [21].

```
        using (var sim = new QuantumSimulator())
        {
            // Try initial values
            Result[] initials = new Result[] { Result.Zero,
Result.One };
            foreach (Result initial in initials)
            {
                var res = BellTest.Run(sim, 1000,
initial).Result;
                var (numZeros, numOnes) = res;
                System.Console.WriteLine(
                    $"Init:{initial,-4} 0s={numZeros,-4}
1s={numOnes,-4}");
            }
        }
        System.Console.WriteLine("Press any key to
continue...");
        System.Console.ReadKey();
```

This driver creates a quantum simulator and an array of initial values that need to be checked (Zero and One). Then the simulation is repeated 1000 times, and the result for debugging is displayed on the screen using the System.Console.WriteLine function

```
Init:Zero 0s=1000 1s=0
Init:One 0s=0 1s=1000
Press any key to continue…
```

If everything is in order, the display should look like the one shown above. This result means that if we transfer the initial qubit to the Zero state and perform a thousand repetitions, then the number of states $|0\rangle$ according to the observation results will be 1000. The same should be done for the One state [21].

Let's try something more interesting. Here we change the state of the qubit using the NOT gate.

```
X(qubits[0]);
let res = M (qubits[0]);
```

Then we run the program again and see that the results are reversed.

```
Init:Zero 0s=0 1s=1000
Init:One 0s=1000 1s=0
```

Then the NOT gate is replaced with the Hadamard gate (H). As a result, as we know, the qubit will go into a superposition of states, and the result of its measurement can be equal to both $|0\rangle$ and $|1\rangle$, with a certain probability.

```
H(qubits[0]);
let res = M (qubits[0]);
```

If you run the program again, we get a rather interesting result.

```
Init: Zero 0s = 484 1s = 516
Init: One 0s = 522 1s = 478
```

The number of measurement results $|0\rangle$ and $|1\rangle$ will be approximately equal.

```
operation BellTest (count : Int, initial: Result) : (Int,Int)
{
    body
    {
        mutable numOnes = 0;
        using (qubits = Qubit[2])
        {
            for (test in 1..count)
            {
                Set (initial, qubits[0]);
                Set (Zero, qubits[1]);

                H(qubits[0]);
                CNOT(qubits[0],qubits[1]);
                let res = M (qubits[0]);

                // Count the number of ones we saw:
                if (res == One)
                {
                    set numOnes = numOnes + 1;
                }
            }

            Set(Zero, qubits[0]);
            Set(Zero, qubits[1]);
        }
        // Return number of times we saw a |0> and number
of times we saw a |1>
        return (count-numOnes, numOnes);
    }
}
```

According to the diagram, the first qubit, qubits [0], needs to be passed through the Hadamard gate. As a result, he will be in superposition. Then we pass the qubits through the CNOT gate (qubits [0] – the controlling qubit, qubits [1] – controlled) and measure the result.

To understand what results to expect, we repeat once again how our state of Bell works. If we measure the first qubit, we get the value $|0\rangle$ with probability 0.5. This means that after the measurement the state will be $|j'\rangle = |00\rangle$ or $|j'\rangle = |11\rangle$ with the same probabilities (0.5) $|j'\rangle = |11\rangle$. Thus, the result of measuring the state of the second qubit will be $|0\rangle$ if the first qubit was in the state $|0\rangle$, and $|1\rangle$ if the first qubit was in the state $|1\rangle$. If the states of two qubits were successfully entangled, then our results should show that the first and second qubits are in the same states [21].

In our code, we check if the measurement result of qubits [1] is equal to the measurement result of qubits [0], using the if operator.

```
operation BellTest (count : Int, initial: Result) :
(Int,Int,Int)
```

```
{
    body
    {
        mutable numOnes = 0;
        mutable agree = 0;
        using (qubits = Qubit[2])
        {
            for (test in 1..count)
            {
                Set (initial, qubits[0]);
                Set (Zero, qubits[1]);

                H(qubits[0]);
                CNOT(qubits[0],qubits[1]);
                let res = M (qubits[0]);

                if (M (qubits[1]) == res)
                {
                    set agree = agree + 1;
                }

                // Count the number of ones we saw:
                if (res == One)
                {
                    set numOnes = numOnes + 1;
                }
            }

            Set(Zero, qubits[0]);
            Set(Zero, qubits[1]);
        }
        // Return number of times we saw a |0> and number
of times we saw a |1>
        return (count-numOnes, numOnes, agree);
    }
}
```

Before checking the results, you need to make another change to the Driver.cs file: add the variable agree.

```
using (var sim = new QuantumSimulator())
        {
            // Try initial values
            Result[] initials = new Result[] { Result.Zero,
Result.One };
            foreach (Result initial in initials)
            {
                var res = BellTest.Run(sim, 1000,
initial).Result;
                var (numZeros, numOnes, agree) = res;
                System.Console.WriteLine(
                    $"Init:{initial,-4} 0s={numZeros,-4}
1s={numOnes,-4} agree={agree,-4}");
            }
        }
        System.Console.WriteLine("Press any key to
continue...");
        System.Console.ReadKey();
```

Now the program can be run. What do these results mean? If the first qubit was initially placed in the Zero state (that is, we applied the value $|00\rangle$ to the input), then the Hadamard gate puts the qubits in the superposition state, and the measurement result is $|0\rangle$ in 50 % of cases and $|1\rangle$ in 50 %. The fulfillment of this condition can be estimated by the number of zeros and ones. If the measurement of the state of the first bit did not affect the state of the second, it would remain equal to $|0\rangle$, and consistency would be achieved only in 499 cases [21].

But, as we can see, the states of the first and second qubit are completely consistent: the number of results $|0\rangle$ and $|1\rangle$ (approximately) coincide. Thus, the results are consistent in each of the 1000 cases. This is how Bell states should work.

Init: Zero 0s = 499 1s = 501 agree = 1000
Init: One 0s = 490 1s = 510 agree = 1000

IX. PROBLEM STATEMENT

There are several approaches to design quantum computers FPGAs ([20], [25]). But there is no systematic approach to designing a digital quantum computer as a digital co-processor, which can be implemented in FPGA.

The variants of such coprocessor circuits, the variants of coding the state of its digital qubits, and the calculation of probability functions are not considered.

The errors of the results of such co-processors were not carried out.

Also, the complexity of the coprocessor implemented in the FPGA was not evaluated.

This article is an attempt to answer the above formulated tasks.

X. THE DIGITAL QUANTUM COMPUTER

A quantum computer is an analog probabilistic computer. His schemes consist only of gates. They have no memory elements. Therefore, there are no programs. Programs are performed only by a classic computer that controls a quantum computer. So the quantum computer is actually a co-processor with respect to the classic computer. The quantum computer functional scheme is induced in Fig. 7.

Table 3

Comparison of digital and analog methods of information processing

Characteristic	Analog processing methods	Digital processing methods
Speed	+	-
Versatility	–	+
Microminiaturization	–	+
Accuracy	–	+
Zoom	–	+
Transmission in space	–	+
Transmission in time (memory)	–	+
Immunity	–	+
Reliability	–	+
Testing, debugging, diagnostics	–	+

A digital quantum coprocessor consists of digital qubits. As a precursor to the class of discrete devices, the digital qubit has the advantage over the analog qubit (Table 3, '+' – advantage, '-' – disadvantage). This causes an interest in constructing precisely the digital qubits to perform probabilistic algorithms on them.

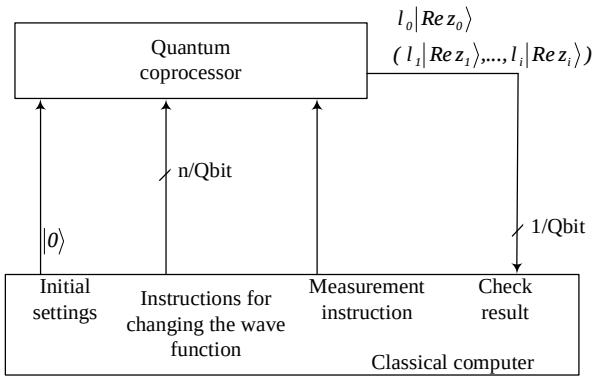


Fig. 7. A digital quantum computer

The digital quantum coprocessor consists of several qubits or it can be assumed that several single-chip digital quantum coprocessors are connected to one classical computer (Fig. 8).

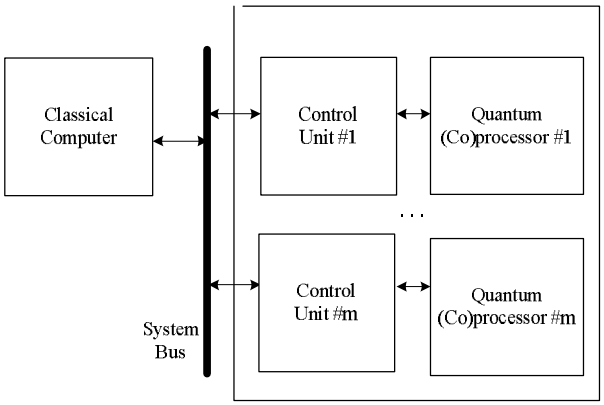


Fig. 8. A quantum computer with multiple digital quantum coprocessors

A classical computer manages the work of a quantum coprocessor, provides it with an input data, stream of instructions and checks the result of its work. Some options for constructing digital quantum computers are discussed in [20]. The classification of quantum computers is represented in Fig. 9. A generalized functional scheme of a quantum coprocessor is given in Fig. 10. Digital quantum coprocessor in Fig. 10 is represented as a set of finite-state machines (FSMs), one of them is a controller and one or more implement the functions of a digital qubit. The connection between the latters can be carried out through the pipeline registers (pRG).

The programmable switch matrix of the coprocessor makes it similar to the Complex Programmable Logic Device (CPLD [30], Fig. 11). The inputs of the matrix are the outputs of all digital qubits ($Result_1$, …, $Result_n$), from the side of the classical computer the matrix can be programmed, so that the inputs of any qubit were fed out of any other qubit ($Result_p$, …, $Result_q$).

The simplest version of a digital quantum coprocessor on a FPGA has only one digital qubit, which consistently performs operations that correspond to operations of quantum gates. The classical computer determines the sequence of these operations. The more complex coprocessor has a chain of several digital qubits (Fig. 10), as well as several qubit chains. Data transfer from one qubit to another can be done through conveyor registers.

A classical analog qubit forms only one output bit. Unlike the analog qubit in the digital qubit with a single-bit output, you can also organize a multi-bit output, which will precisely determine its current state in Fig. 16. The use of a multi-bit output by a classical computer allows you to arrange interruptions of quantum programs and call quantum procedures and functions (programs, procedures, functions, and interruptions that are executed and processed by a classical computer in the process of controlling a digital quantum coprocessor, will be called quantum ones).

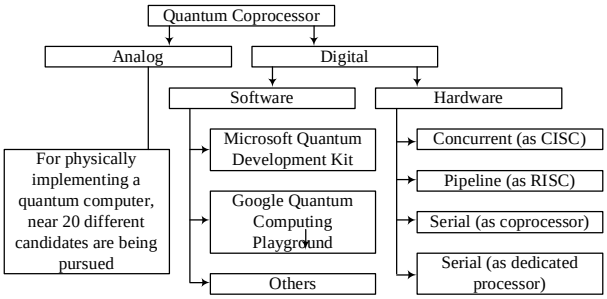


Fig. 9. Options for quantum computers design

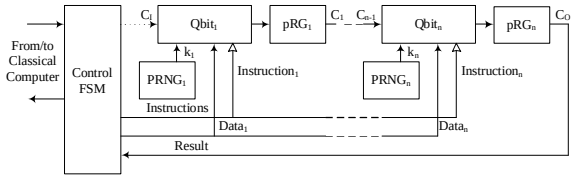


Fig. 10. A generalized functional diagram of a digital quantum coprocessor

XI. DIGITAL QUANTUM COPROCESSOR FOR REAL AMPLITUDES

To explain the principles of digital quantum coprocessor design, it will be first implemented for the case of real amplitudes.

Two methods are possible to determine the position of the vector in a unit circle – using Cartesian and polar coordinates (Fig. 12).

In this paper, to represent the position of the vector, a polar coordinate system is selected (it is necessary to specify and define only one coordinate – the angle θ (Fig. 13), in contrast to the Cartesian coordinate system, where it is necessary to specify two coordinates, x and y).

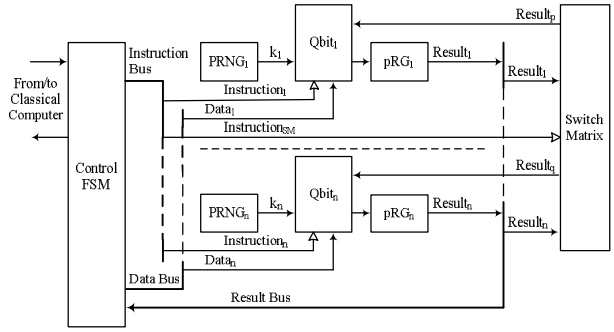


Fig. 11. A generalized functional diagram of a digital quantum coprocessor with a switching matrix

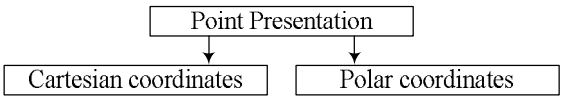


Fig. 12. Encoding formats of qubits state in a digital quantum computer

The data format that a classical computer transmits to a quantum coprocessor (angle θ and quadrant number) is shown in Fig. 14.

The codes used to set the angle θ are shown in Fig. 15. The codes xx.0..0 are special: they belong to two quadrants at once and can encode both state $|0\rangle$ and state $|1\rangle$. During the measurement, all angles are reduced to the angles of the first quadrant ($0 \le \theta \le \pi / 2$) with the codes corresponding from 00.0..0 (the code for $|0\rangle$) to 01.0..0 (the code for $|1\rangle$). The number of such codes is odd – 2^n+1, where n is the angle θ code bit number. The midpoint of the range is exactly 00.10..0 (π/4 angle, $\sin(\pi/4)=\cos(\pi/4)=1/\sqrt{2}$, so it is the point of qubit undefined state). There are 2^{n-1} codes smaller than it (from 00.0...0 to 00.01..1) and 2^{n-1} codes bigger than it (from 00.10..01 to 01.0..0), which is taken into account when measuring the state of the qubit.

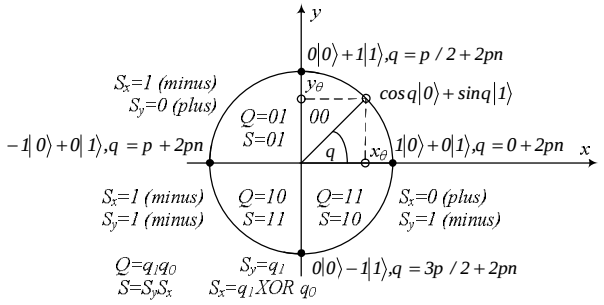


Fig. 13. Representation of a qubit state on a unit circle

Q_1	Q_0	a a ... a
2-bit		N-bit angle
Quadrant Number Q		

Fig. 14. Data exchange format

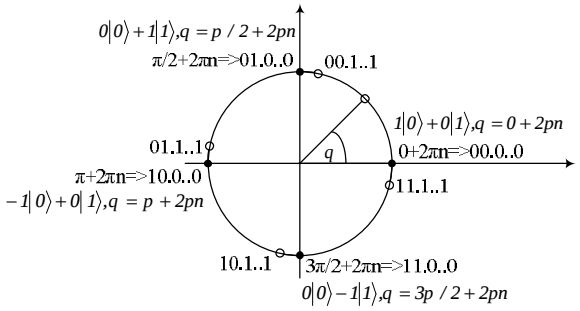


Fig. 15. Vectors position codes (angle θ codes)

Quantum gates [1] are used to convert coefficients l_j of wave functions $|y\rangle$.

XII. REALIZATION OF A DIGITAL QUBITE ON FPGA

There are several approaches to design quantum computers FPGAs [20, 25]. The peculiarity of the proposed in the paper option is the implementation of a digital quantum coprocessor and the use of polar coordinates to represent the position of the vector on a unit circle. For the realization of probabilistic functions a generator of pseudorandom codes k is used with their subsequent transformations into a form suitable for estimating the probabilities of the results as p = arcsin(sqrt(k)), where sqrt (k) is the square root of k.

A qubit that can be controlled by a classical computer, appears as a finite-state machine (FSM) (Fig. 16) with the measurement device on the output.

One of the main elements of a digital quantum coprocessor on a FPGA is the true random code generator (TRNG, external) or pseudorandom code generator (PRNG, internal, based on shift registers with linear feedback with the period 2^{32}-1 [27]), and a functional converter on its output, which is used to measure the qubit state (Fig. 20). Functional converters can be created according to well-known solutions [16, 17].

Each qubit can have its own pseudo-random code generator (the measurement scheme for this case is shown in Fig. 20) or qubites can use one generic generator, referring to it at different times (possible measurement circuits shown in Fig. 21, Fig. 25, Fig. 24, time chart in Fig. 26 illustrates this process.

The additional and optional multi-bit output (Multi-bitQbitOut) allows to interrupt digital qubit control programs performed by a classical computer. This output allows you to read the exact internal state S_0 of the digital qubit, to download the new S_1 state through the DataIn input, to perform a series of instructions starting from this state S_1, to read the exact state of S_2, in which the qubit will appear after their execution, to save S_2 in the memory of the classic computer, then to read from the memory of a classical computer previously saved qubit previous state S_0, to load it into a qubit and continue to execute instructions from this state S_0.

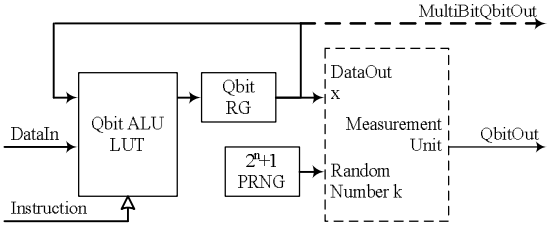


Fig. 16. Qubit as a finite-state machine (FSM)

To encode the position of the vector in the range from 0 (the code 00.00..0) to π/2 (the code 01.00..0), an odd number of codes (2n + 1) is required.

To determine the state of the qubit during the measurement using the classical method (Fig. 20), the operation $y=\sin^2 x$ must be performed (Fig. 20), then compare the code y with (pseudo) random code that can have any value in the same range (from 00.00..0 to 01.00..0). Accordingly, it is necessary to have a random code generator of the same range. The value $|1\rangle$ is formed if the code y is less (pseudo) random code k or equals to the code 01.00..0, the result $|0\rangle$ is formed if the code y is more (pseudo) random code k or equals to the code 00.00..0. If y = k – the decision about the state of the digital qubit is taken into account of additional parameters which are not discussed now. The disadvantage of this method is the presence of a functional converter $y = \sin^2 x$ at the output of each qubit.

To reduce the correlation of the results of measuring the states of different digital qubits, it is desirable to use generators of pseudorandom codes with different and large periods (for example, 2^{128}-1, 2^{19937}-1 [28] and various initial vectors.

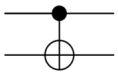
XIII. MAIN INSTRUCTIONS WHICH A DIGITAL QUBBIT MAY PERFORM

The instructions that a digital qubit can perform must be universal – form a functionally complete system of functions, that is, using a set of such instructions, you can recreate the work of any other quantum gate.

Quantum gates perform unitary operations that do not change the vector norm on the Bloch sphere; they only move a point along the sphere.

The simplest one-qubit gates:

- identical transformation
- negation (Pauli Gates) X, Y, Z 
- phase shift Rm 
- Hadamard transform H 

There are gates having two inputs (and two outputs, since the number of inputs and outputs at quantum gates must match due to the unitarity requirement). One of them is controlled NOT (C-NOT) 

There are special measurements gate  .

The gate set, consisting of the C-NOT gate and all single-qubit gates, is universal.

XIV. INSTRUCTIONS FOR DIGITAL QUBIT, WHICH WORKS WITH REAL NUMBERS

The negation instruction performs the conversion $A|0\rangle + B|1\rangle \rightarrow B|0\rangle + A|1\rangle$. Its performance for real numbers is illustrated in Fig. 17.

If the vector is in the first quadrant and its position is given by the angle α=θ, The digital qubit ALU for determining the position of the vector after inversion (to determine the angle α_{NOT}) must perform the actions $\alpha_{NO} = \pi/2 - \theta = \pi/2 - \alpha$. If π/2 is encoded as 01.0..00, then this operation is reduced to the definition of two's complement binary code for (-θ).

For the second quadrant

$\alpha=\pi/2 + \theta$ and $\alpha_{NOT}= 2\pi+\pi/2 -(\pi/2 + \theta) = \pi/2 - \alpha$.

For the third quadrant

$\alpha=\pi + \theta$ and $\alpha_{NOT}= 2\pi+\pi/2 -(\pi + \theta) = \pi/2 - \alpha$.

For the fourth quadrant

$\alpha=2\pi - \theta$ and $\alpha_{NOT}= \pi/2 + \theta = \pi/2 - \alpha$.

The phase shift of the vector with the coordinate θ by the angle φ is performed as the addition $\theta_{R\varphi} =\theta + \varphi$.

Hadamard transform sets the vector defined by the angle θ at an angle $b = arctg\dfrac{cosq - sinq}{cosq\_sinq}$. In fact, the dependence of the angles is linear (Fig. 18, for the first quadrant $\beta = \pi/4 - \theta$) and the new angle is easily calculated by the digital qubit. Repeating the Hadamard transform returns the vector to its previous state. In particular, from the position $|0\rangle$, the Hadamard transform translates a qubit to unstable equilibrium $H = \dfrac{1}{\sqrt{2}}|0\rangle + \dfrac{1}{\sqrt{2}}|1\rangle$, measurement in which will result $|0\rangle$ with the probability of 50 % or with the same probability of 50 % – the result $|1\rangle$.

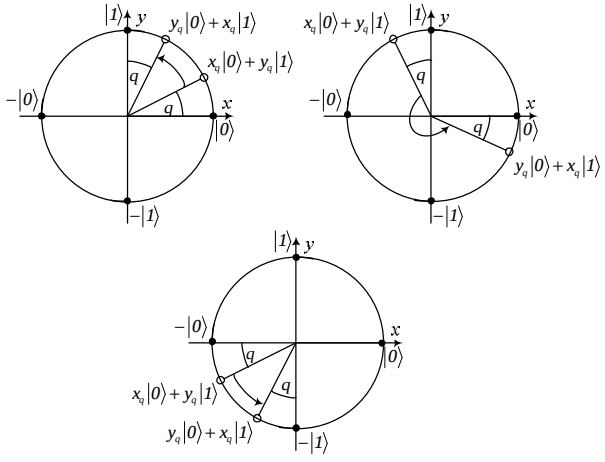


Fig. 17. Pairs of inter-inverse vectors

Controlled NOT in a digital quantum qubit in the simplest case is reduced to a sequence of two commands of a quantum program:

measurement of the state of one digital qubit;

inverting another qubit if the state of the first was $|1\rangle$.

To perform quantum interrupts, procedures, and functions, additional commands for loading an arbitrary initial angle (Load) and unloading the multi-bit current position of a digital qubit (Store, which cannot be achieved in a classic analog qubit) are used.

XV. ORGANIZATION OF QUANTUM COMPUTATIONS

On the whole, the organization of quantum computations in k-qubit (q_1, q_2, …, q_k) coprocessor is illustrated in Fig. 19:

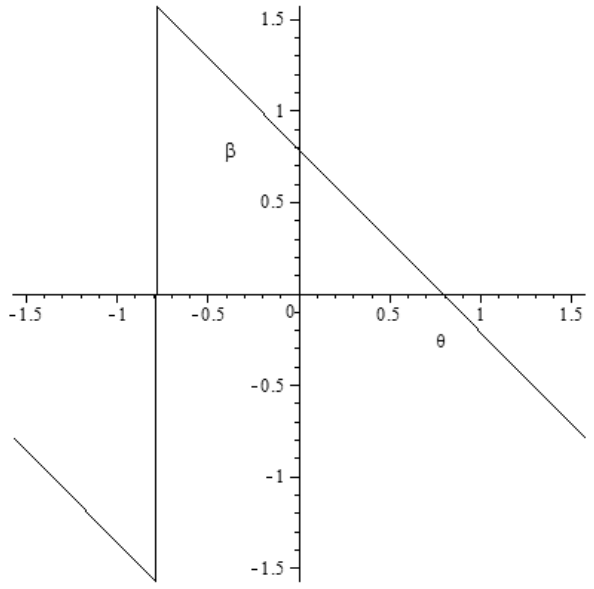


Fig. 18. Changing the angle of the vector as a result of the Hadamard transform

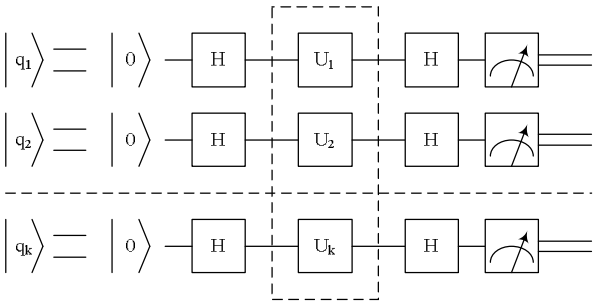


Fig. 19. Computation scheme in a quantum coprocessor

transfer all qubits to the initial state $|0\rangle$;

transfer of qubits to a state of unstable equilibrium using Hadamard transforms H;

impact on qubits either from the side of a classical computer or from other qubits with the help of the so-called phase request U_i to bring them out of a state of unstable equilibrium;

re-impact on qubits by Hadamard transform H;

qubit state measurement.

The measured result is checked by a classic computer and the process is repeated.

XVI. MULTIPLE QUBBIT DIGITAL QUANTUM COPROCESSOR FOR FPGA.

Consider an alternative way of measuring the qubit state of a digital quantum coprocessor (Fig. 21), in which the functional converter is transferred to the output of the generator of random codes (now it is a converter $y = arcsin\sqrt{x}$). Such a scheme facilitates the creation of multi-qubit digital quantum coprocessors with several comparators (Fig. 25) or with one comparator (Fig. 24).

The time diagram of the classical computer's reading of the results of a quantum coprocessor (Fig. 24) is shown in Fig. 26.

XVII.FPGA DIGITAL QUANTUM COPROCESSOR FOR COMPLEX AMPLITUDE

In contrast to the real amplitudes and the representation of the vectors by their positions on the unit circle, for which it is necessary to set only one angle θ, the use of complex amplitudes and the Bloch sphere requires working with two angles θ and φ (latitude and longitude). The principles described above for constructing a quantum coprocessor for real amplitudes can also be applied to the construction and coprocessor for complex amplitudes.

XVIII. TESTING OF DIGITAL QUANTUM COPROCESSORS

Testing of probabilistic circuits such as a digital quantum coprocessor is a very complex, long and multilevel process.

Testing involves performing a large number of experiments and evaluating the probability of each of the results. The probability of each of the research results evaluated by their actual number should be close to the estimated probability of this result.

The test sequence may be as follows:

testing a single qubit with a median position of the unit vector (at an angle π/4, after its reset and Hadamard's transformation. The probability of measuring this state as $|1\rangle$ and as $|0\rangle$ should be approximately the same (50 %) (Fig. 23);

testing a single qubit when placing a unit vector at an angle π/6. The probability of determining this as $|1\rangle$ and as $|0\rangle$ will be 75 % and 25 % respectively;

testing a single qubit when placing a unit vector at an angle π/3. The probability of determining this as $|1\rangle$ and as $|0\rangle$ will be 25 % and 75 % respectively;

testing of a multi-bit digital quantum coprocessor with a median position of unit vectors in each qubit (at an angle π/4, after their reset and execution of Hadamard's transformation).

For a four-qubit coprocessor the probability of determining each of the 16 possible states from $|0000\rangle$ to $|1111\rangle$ will be equal to 1/16 (6.25 %) (Fig. 22). Fig. 22 shows the relative deviation of the probabilities of 16 states (delta_state0, …, delta1_state15 – respectively for the states from $|0000\rangle$ to $|1111\rangle$) from the expected value 6.25 % after each successive experiment. The experiment number is postponed on the horizontal axis (extreme right point – experiment 2^{18}). As can be seen, the maximum relative deviation is near 2 % from the nominal value 6.25 % (Absolute deviation is less than 0.02*6.25 % = 0.125 %) The probability to read any coprocessor state is (6.25±0.125) %). These results are obtained using a 32-bit pseudo-random code generator and qubits with 16-bit angle codes.

Other tests are also possible.

XIX. IMPLEMENTATION OF THE DIGITAL COPROCESSOR ON FPGA

Some of the described digital coprocessors have been implemented on the FPGA Spartan 6 (Xilinx).

The 16-bit digital qubit (Fig. 16) with the measurement unit (Fig. 21) when implemented in the FPGA xc6slx150 occupies 223 of 92152 LUT (less than 1 % of resources). The clock frequency of the digital qubit exceeds 200 MHz. XC6SLX150 theoretically allows to build in one FPGA a digital quantum coprocessor with more than 400 digital qubits and 2% accuracy. To construct such a multi-bit digital quantum coprocessor, it is advisable to create an appropriate qubit IP-core generator. Parameters of the IP-core generator should be:

number of digital qubits;
angles code bit number;
pseudorandom codes generator bit number ;
start vectors of generators of pseudorandom codes;
measurement unit type ;
topology of the switching matrix;
other options.

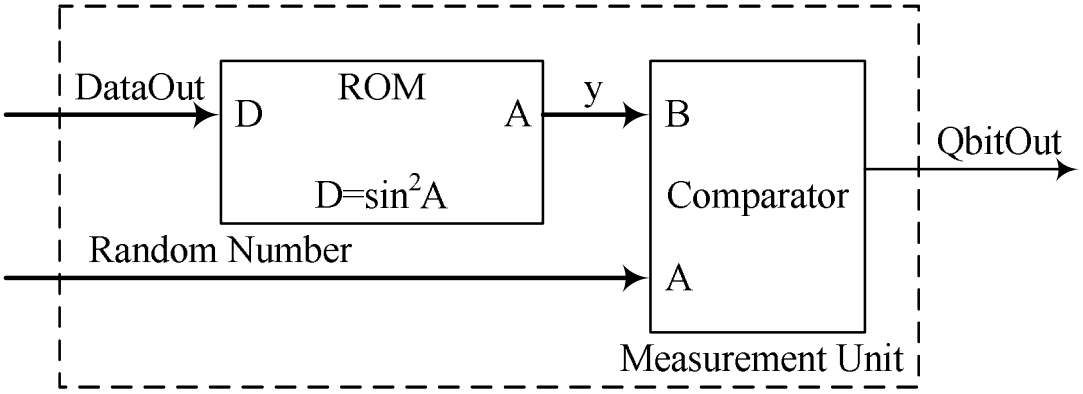


Fig. 20. Measurement of the state of the digital qubit

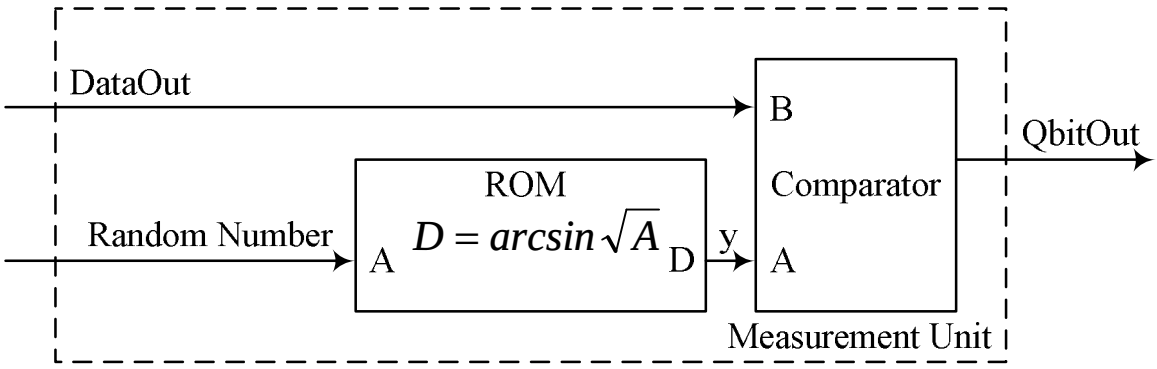


Fig. 21. An alternative way to measure the state of a digital qubit

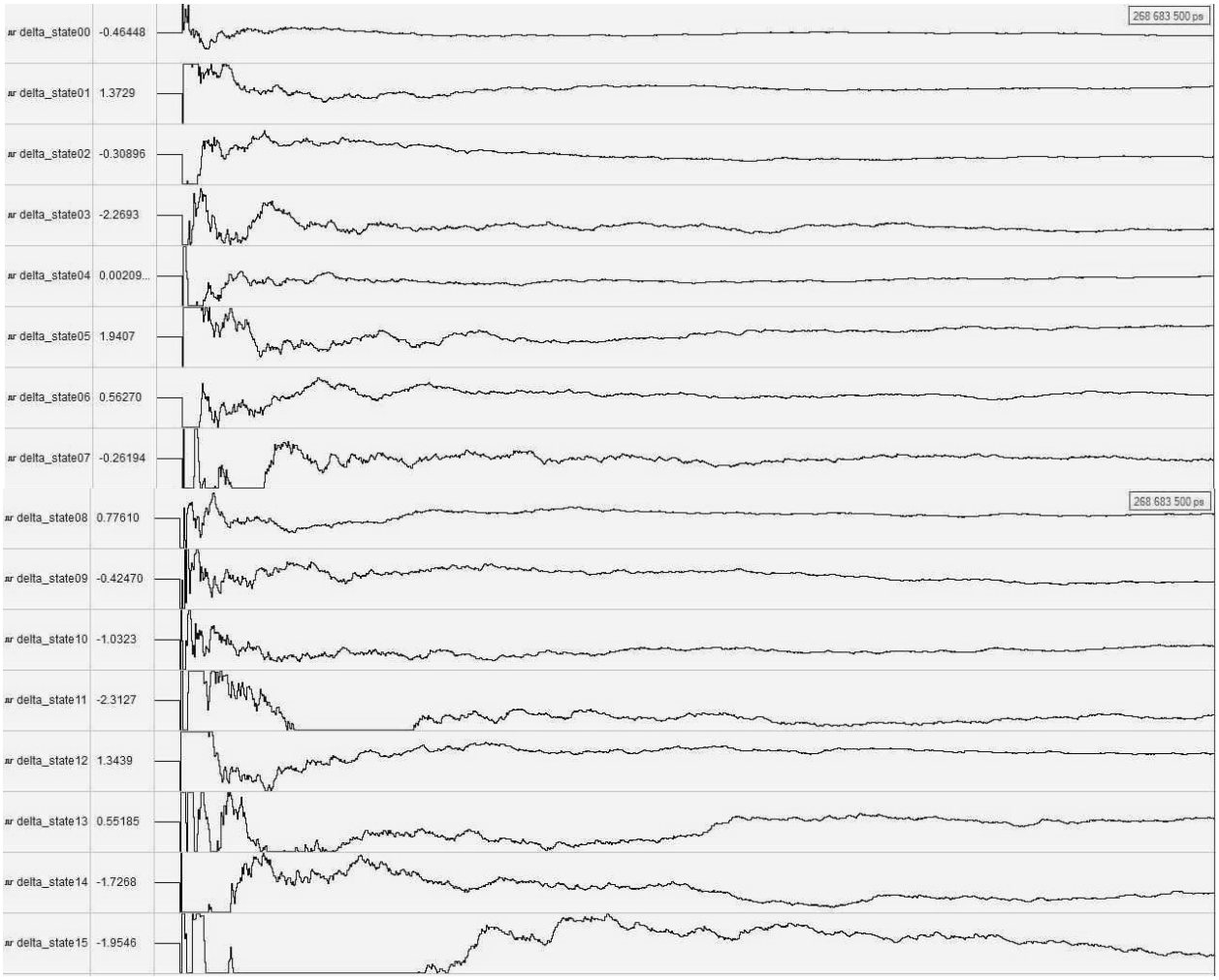


Fig. 22. The relative accuracyin % for 16-bit quantum coprocessor with 4 qubits

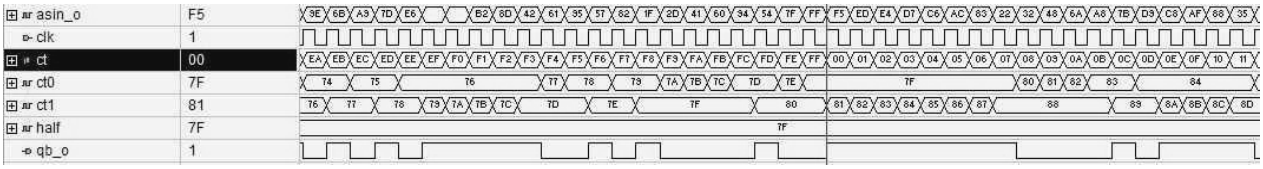


Fig. 23. Digital qubit simulation

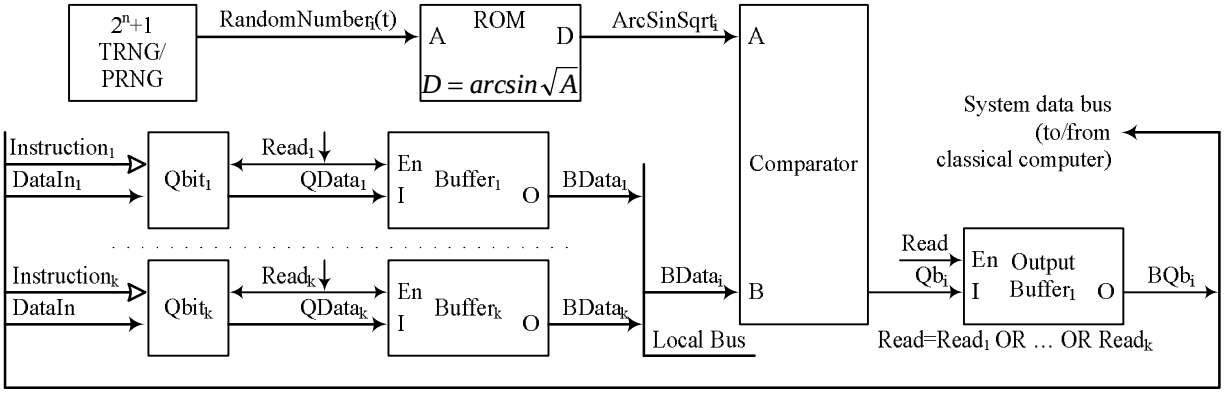


Fig. 24. Multiqubit digital quantum coprocessor with one comparator

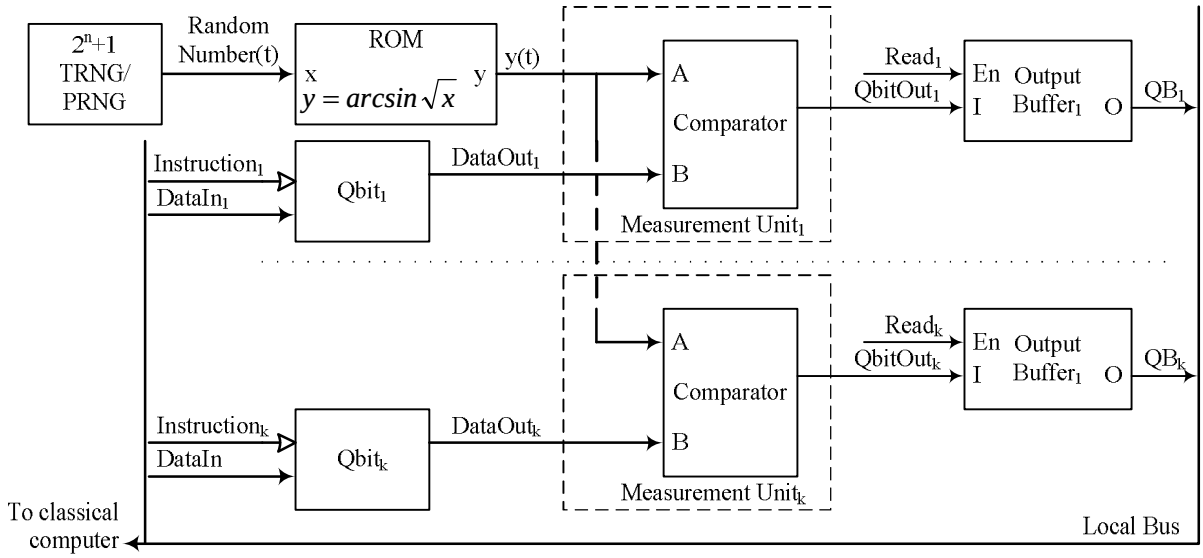


Fig. 25. Multiqubit digital quantum coprocessor

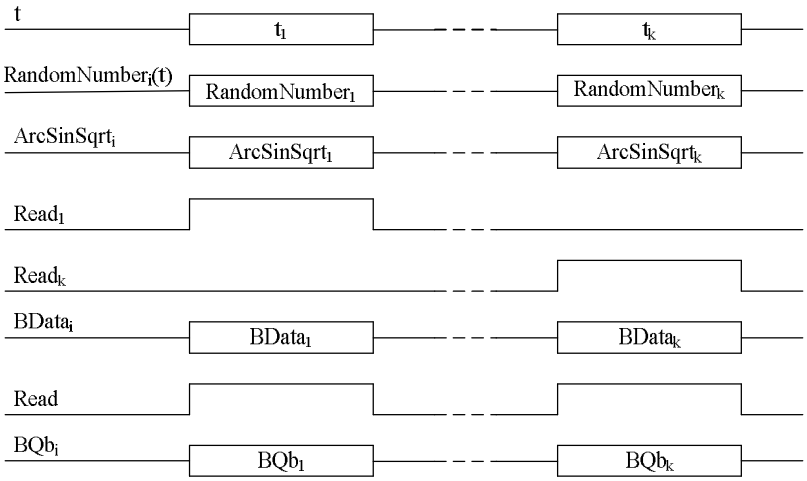


Fig. 26. Reading results from a quantum coprocessor Fig. 24

XX. CONCLUSION

The principles of constructing digital quantum coprocessors based on digital qubits are presented.

Models of the digital quantum coprocessor for its subsequent realization on the FPGA are created. The model is checked. The deviations of the results of the model from the results of the ideal quantum computer are near 2 %. It is estimated that it is theoretically possible to build on one FPGA a digital quantum coprocessor with more than 400 digital qubits.

REFERENCES

[1] Quantum logic gate. https://en.wikipedia.org/wiki/Quantum_logic_gate 08.02.2019.

[2] Quantum computing. https://en.wikipedia.org/wiki/Quantum_computing 08.02.2019.

[3] A Preview of Bristlecone, Google's New Quantum Processor. https://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html. 08.02.2019.

[4] Qubit. https://en.wikipedia.org/wiki/Qubit. 08.02.2019.

[5] ETSI White Paper No. 8. Quantum Safe Cryptography and Security. An introduction, benefits, enablers and challenges. June 2015. ISBN No. 979-10-92620-03-0 https://www.etsi.org/images/files/ETSIWhitePapers/QuantumSafeWhitepaper.pdf

[6] NISTIR 8105. L. Chen et al., Report on Post-Quantum Cryptography. National Institute of Standards and Technology. U.S. Department of Commerce.April 2016. http://dx.doi.org/10.6028/NIST.IR.8105 25.11.2018 г

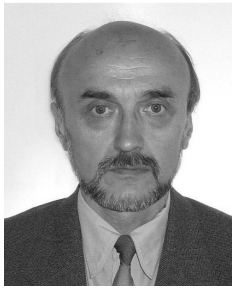
[7] 9th International IEEE Conference Dependable Systems, Services and Technologies DESSERT'2018 UKRAINE, KYIV, MAY 24-27, 2018

[8] Quantum superposition. https://en.wikipedia.org/wiki/Quantum_superposition 08.02.2019.

[9] Introduction to Quantum Computing. https://blogs.msdn.microsoft.com/uk_faculty_connection/2018/02/06/introduction-to-quantum-computing/ 08.02.2019.

[10] Peter W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. Proceedings of the 35th Annual Symposium on

Foundations of Computer Science, Santa Fe, NM, Nov. 20–22, 1994, IEEE Computer Society Press, pp. 124–134.
[11] Shor's algorithm. https://en.wikipedia.org/wiki/Shor%27s_algorithm 08.02.2019.
[12] K. A. Valiyev. Kvantovaya informatika: kompyutery. Svyaz i kriptografiya. Vestnik Rossiyskoy akademii nauk. Tom 70, No. 8, p. 688–695 (2000) (In Russian).
[13] Welcome to the Microsoft Quantum Development Kit Preview. https://docs.microsoft.com/ru-ru/quantum/?view=qsharp-preview 08.02.2019
[14] Quantum Fourier transform. https://en.wikipedia.org/wiki/Quantum_Fourier_transform 08.02.2019.
[15] Lucero, Erik; Barends, Rami; Chen, Yu; Kelly, Julian; Mariantoni, Matteo; Megrant, Anthony; O'Malley, Peter; Sank, Daniel; Vainsencher, Amit; Wenner, James; White, Ted; Yin, Yi; Cleland, Andrew N.; Martinis, John M. (2012). "Computing prime factors with a Josephson phase qubit quantum processor". Nature Physics. 8 (10): 719. arXiv:1202.5707. Bibcode:2012NatPh...8..719L.doi:10.1038/nphys2385
[16] Popov B. A.. Tesler G. S. Vychisleniye funktsiy na EVM. Spravochnik. Kiyev: Nauk. dumka, 1984. 59 p. (In Russian).
[17] V. V. Aristov. Integro-algoritmicheskiye vychisleniya. "Nauk. Dumka", 1980. 189 p. (In Russian).
[18] Quantum Computing Playground. http://www.quantumplayground.net/#/home 20.01.2019
[19] A Preview of Bristlecone, Google's New Quantum Processor. http://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html 20.01.2019
[20] M. Khalil-Hani, Y. H. Lee, M. N. Marsono. An Accurate FPGA-Based Hardware Emulation on Quantum Fourier Transform. Proceedings of the 13th Australasian Symposium on Parallel and Distributed Computing (AusPDC 2015), Sydney, Australia, 27 – 30 January 2015. Pp. 23 – 30.
[21] Q# techniques – putting it all together | Microsoft Docs (https://docs.microsoft.com/en-us/quantum/techniques/putting-it-all-together?view=qsharp-preview) 08.02.2019
[22] An introduction to Q# – Microsoft's language for quantum computing – General – The freeCodeCamp Forum (https://www.freecodecamp.org/forum/t/an-introduction-to-q-microsoft-s-language-for-quantum-computing/189814) 08.02.2019
[23] Microsoft's Quantum Programming Language, Q#, Could Help You Learn Quantum Physics | Digital Trends https://www.digitaltrends.com/computing/quantum-microsoft-q/ 08.02.2019
[24] Valeriy Hlukhov. "Kvantovyy kompyuter kak veroyatnostnyy kompyuter". Shosta mizhnarodna naukova konferencija «Modeljuvannja-2018». September 12–14, 2018 Kyiv, Ukraine. Zbirka pracj konferenciji, p. 111 – 114.
[25] Gushanskiy S. M., Pereverzev V. A. Simulation of Quantum Computing using Hardware Cores. Nauchnyy zhurnal KubGAU, №123(09), 2016. pp. http://ej.kubagro.ru/2016/09/pdf/37.pdf
[26] The Mathematics Behind Quantum Computing: Part II. http://www.ams.org/publicoutreach/feature-column/fcarc-quantum-two
[27] LFSR-Random number generator :: Overview. https://opencores.org/projects/lfsr_randgen 14.02.2019.
[28] Pseudo Random Number Generators as synthesizable VHDL code. https://github.com/jorisvr/vhdl_prng 14.02.2019
[29] Spartan-6 Family Overview. DS160 (v2.0) October 25, 2011. Product Specification. https://www.xilinx.com/support/documentation/data_sheets/ds160.pdf 14.02.2019
[30] CPLD. https://www.xilinx.com/products/silicon-devices/cpld/cpld.html 16.02.2019
[31] The D-Wave 2000Q™ Quantum Computer Technology Overview. D-Wave Systems Inc. https://www.dwavesys.com/sites/default/files/D-Wave%202000Q%20Tech%20Collateral_0117F.pdf 19.02.2019
[32] Atomic orbital. https://en.wikipedia.org/wiki/Atomic_orbital 19.02.2019
[33] Physicists control the flip of electron spin. https://phys.org/news/2005-05-physicists-flip-electron.html 19.02.2019
[34] Applying Moore's Law to Quantum Qubits https://quantumcomputingreport.com/our-take/applying-moores-law-to-quantum-qubits/ Copyright © 2019 Quantum Computing Report, All rights reserved 19.02.2019
[35] BQP https://en.wikipedia.org/wiki/BQP 19.02.2019
[36] Google Unveils 72-Qubit Quantum Computer With Low Error Rates. https://hardware.slashdot.org/story/18/03/05/2156247/google-unveils-72-qubit-quantum-computer-with-low-error-rates 26/02/2019



Valerii Hlukhov is a professor of the Department of Computer Engineering in Lviv Polytechnic National University, Ukraine. He graduated from Lviv Polytechnic Institute with the engineer degree in computer engineering in 1977. In 1991 he obtained his Ph.D. from the Institute of Modeling Problems in Power Engineering of the National Academy of Science of Ukraine. He was recognized for his outstanding contributions into special-purpose computer systems design as a Senior Scientific Researcher in 1995.

He was awarded the academic degrees of doctor of technical sciences in 2013 at Lviv Polytechnic National University. He became a Professor of Computer Engineering in 2014. He has scientific, academic and hands-on experience in the field of computer systems research and design, proven contribution into IP Cores design methodology and high-performance reconfigurable computer systems design methodology. He is experienced in computer data protection, including cryptographic algorithms, cryptographic processors design and implementation. Mr. Hlukhov is an author of more than 100 scientific papers, patents and monographs.

Bohdan Havano was born in 1994 in Sambir, Ukraine. He received the B.S. degree in computer engineering at Lviv Polytechnic National University in 2015 and M.S degree in system programming at Lviv Polytechnic National University in 2016. He has been doing scientific and research work since 2017. Currently, he is a graduate student of the Computer Engineering Department at Lviv Polytechnic National University. His research interests include architecture and data protection in cyber-physical systems.