Jacek Długopolski
Marcin Sadowski
Wawrzyniec Suleja

# A PHYSICAL MODEL OF QUANTUM BIT BEHAVIOR BASED ON A PROGRAMMABLE FPGA INTEGRATED CIRCUIT

**Abstract** *The rapidly developing field of quantum computing and the ongoing lack of widely available quantum computers create the need for scientists to build their simulators. However, mathematical simulation of such circuits usually ignores many aspects and problems found in real quantum systems. In this article, the authors describe a quantum bit emulator based on FPGA integrated circuits. In this case, FPGA technology provides real-time massive parallelism of the modeled physical phenomena. The modeled QUBIT is represented using a Bloch sphere. Its quantum state is set and modified only by precise pulses of an electrical signal, and with the help of similar pulses, it manifests its current state in real time. The constructed QUBIT was additionally equipped with decoherence mechanisms and with circuits that intentionally respond to internal and external noises that distort its current quantum state. This article presents and discusses how such a physically built emulator works.*

**Keywords** quantum, qubit, FPGA, parallel, real time

**Citation** Computer Science 25(4) 2024: 1–21

## 1. Introduction

Currently, quantum physics is increasingly entering the world of information processing. The specific features of quantum phenomena (such as superposition, tunneling, or entanglement) mean that data processing carried out on the basis of quantum bits (qubits) allow achieving useful computational results in a time that would not be possible to achieve using classical methods.

However, quantum systems are still very difficult to build and maintain, so they still may not be widely available in the near future. Therefore, the idea of building a practical model of the behavior of the basic unit of a quantum computer (qubit) seems justified. After equipping such a model with a physical control system using electrical impulses, as well as a natural susceptibility to internal and external noise and sensitivity to decoherence, we will obtain a useful tool for researching and better emulating real quantum systems. By combining several such emulators, you can create more complex quantum systems and test sample quantum computing and noise reduction algorithms on them.

Effective, fault-tolerant quantum computing requires two-or-more-qubit gates with the highest possible fidelity. The recommended infidelity in the literature is below $10^{-4}$ [16]. Achieving such a level of fidelity in two-qubit gates implemented in an ion trap, such as the SWAP gate and the Mølmer-Sørensen gate, even in noise-free conditions, requires analysis not only of the internal states of the ions but also of the most relevant part of the phonon space. The authors hypothesize that complex emulations of this kind can be more efficiently conducted using parallel and real-time models based on FPGA integrated circuits, which may in the future enable more effective identification of both coherent and stochastic error sources. In addition, quantum processors also rely on classical electronic controllers to manipulate and read out the state of qubits. As the performance of the quantum processor improves, imperfections in the classical controller can become the performance bottleneck for the whole quantum system. To prevent such limitations, a systematic study of the impact of classical electrical signals on qubit fidelity is needed. To have an easy way to do such a study, we need a model close enough to real quantum systems. By controlling the qubit model only with real physical signals (pulses), it is possible to more accurately reproduce the control conditions of real quantum systems. Our proposal described in this article tries to meet all these needs.

This article describes the practical implementation of just such a model, based on a programmable FPGA integrated circuit. In addition to the qubit model (QUBIT), the article also details the control and reading module (GENMET), designed to generate appropriate electrical control pulses, as well as to read electrical pulses generated by the QUBIT itself, through which it informs the environment about its current state.

The article will also describe the idea of communication with such a single QUBIT and the method used to implement its susceptibility to decoherence and to internal and externally generated noise.

Next, the results and practical tests of the built emulator using the dedicated GENMET module will be described. The user can, of course, use other generators and measurement systems instead of GENMET. Finally, a brief description of possible applications will be presented.

## 2. Related works

At the time of writing this article, the authors had not found another similar solution in the world. Of course, many quantum simulators and emulators using mathematical apparatus are being created [1–4, 6–8, 10–15, 17], but they do not give the user the opportunity to encounter the physical problems experienced when working on building real quantum computers. The solution proposed in this article necessitates precise formation and measurement of electrical impulses that are susceptible to unpredictable physical disturbances. This provides a unique opportunity to encounter these types of problems. There are, of course, various solutions to mathematically generate artificial noise in quantum circuit simulators. An example of such a system is, for example: "Qiskit Aer - the high-performance quantum computing simulators with realistic noise models" [9]. This system, however, uses a mathematical formalism to add noise. Our system uses physical noise, which can be natural and come from within the FPGA chip itself, or it can come from an external physical function generator whose generated waveforms are also not completely immune to noise. Since we feed both control pulses signals and noise signals to our QUBIT via ordinary electrical cables, the signals flowing in them can be further naturally disturbed by external electromagnetic fields, as happens in real quantum systems under construction today. In this respect, it gives our system described here an undoubted advantage over other simulators available. The user here is getting very close to what she or he will be dealing with when operating non-ideal real quantum systems.

## 3. The idea of a quantum bit

A quantum bit (qubit) is different from a classical bit. While a classical bit is the basic smallest unit of information that can represent one of two possible classical states: '0' or '1', according to formula:

$$bit = \begin{cases} '0' \\ '1' \end{cases}$$

a qubit can take on these two classical values simultaneously and in different proportions, constituting a normalized vector in a two-dimensional Hilbert space over a complex field C in an orthogonal basis $\{|0\rangle, |1\rangle\}$. According to formula:

$$qubit = |\psi\rangle = A \cdot |0\rangle + B \cdot |1\rangle$$

where $|0\rangle$ and $|1\rangle$ represent the base states: '0' and '1' in ket notation, such that:

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

and A, B are the amplitudes of the wave functions for the states: $|0\rangle$ and $|1\rangle$, such that:

$$|A|^2 + |B|^2 = 1.$$

The square of the amplitude A modulus is the probability that the measured qubit will have state $|0\rangle$, and the square of the amplitude B modulus is the probability that the measured qubit will have state $|1\rangle$. The state of such a qubit can also be represented graphically on a Bloch sphere. If we assume that:

$$A = \cos\frac{\theta}{2}, \qquad B = e^{i\varphi}\sin\frac{\theta}{2}$$

then the qubit state can be represented graphically using the normalized state vector $|\psi\rangle$ on the three-dimensional Bloch sphere (Figure 1): Therefore, the state of the qubit

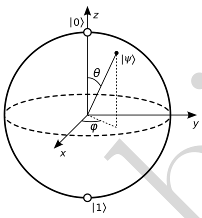

**Figure 1.** The Bloche sphere (source: wikipedia.org)

can be uniquely described using two angles: $\theta$ and $\varphi$. And this form of representing the state of the emulated qubit was adopted by the authors of this article.

## 4. Model architecture

This chapter will discuss the general architecture of the quantum bit model. In order to test the idea of the model, the entire functional set of the system was designed. It consists of a proper qubit emulator called QUBIT and a device adapted to generate and measure electric digital pulses called GENMET (GENerator-METer).

In addition to these two main elements, the set also includes: a monitor, an external generator, an oscilloscope and a simulation management computer. It is schematically shown in Figure 2. The proposed system emulates the behavior of a virtual qubit in real time and graphically displays the Bloch sphere on the monitor screen along with a vector representing the current state of this qubit. The user has the ability to rotate the qubit's state vector with appropriate input digital pulses, while at the same time the emulated qubit continuously generates its own digital output signals proportional to its current state. Thanks to this, the user has the constant opportunity to observe and read parameters of the qubit, such as its phase and superposition state, which is not directly possible in real quantum systems. Additionally, digital pulses fed into and received from the emulator's outputs can be continuously monitored on the attached oscilloscope.

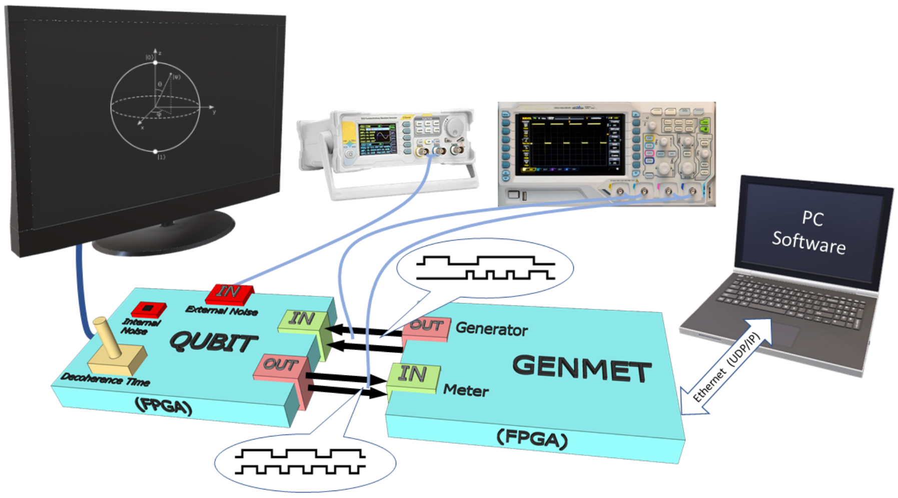

**Figure 2.** The system idea

QUBIT is also equipped with modules that emulate real physical disturbances. For this purpose, both the internal natural noises of the FPGA system (on the basis of which the emulator was created) are used, as well as special additional circuits that allow the qubit to be disrupted with a user-defined signal generated on an external function generator. Thanks to this, the user can test his quantum operations (algorithms) in the presence of a given interference waveform. It can be, for example, a sinusoidal signal of any frequency, but also white noise or another signal composed of many other harmonic waveforms. In particular, frequency, amplitude or phase modulated signals can be used.

Additionally, to ensure conditions of using the qubit emulator that are close to real ones, the emulator is equipped with a module that controls the qubit coherence time. Thanks to it, the user can set the time after which the qubit will be automatically measured and will lose its quantum state. This situation occurs when working with real quantum systems. For this reason, the user must develop quantum algorithms in such a way that they can be calculated before the system decoheres. The built-in decoherence management module is intended to enable the simulation of this type of situation.

## 4.1. QUBIT architecture

The QUBIT module, built on the basis of a programmable FPGA integrated circuit, is the main element of the system. It emulates the behavior of a qubit. The state of a qubit is represented by two angles: THETA and PHI. The THETA angle describes the state of superposition of the base states, and the PHI angle determines the phase of the $|1\rangle$ state in relation to the $|0\rangle$ state. The block diagram of the QUBIT module architecture is presented in Figure 3.

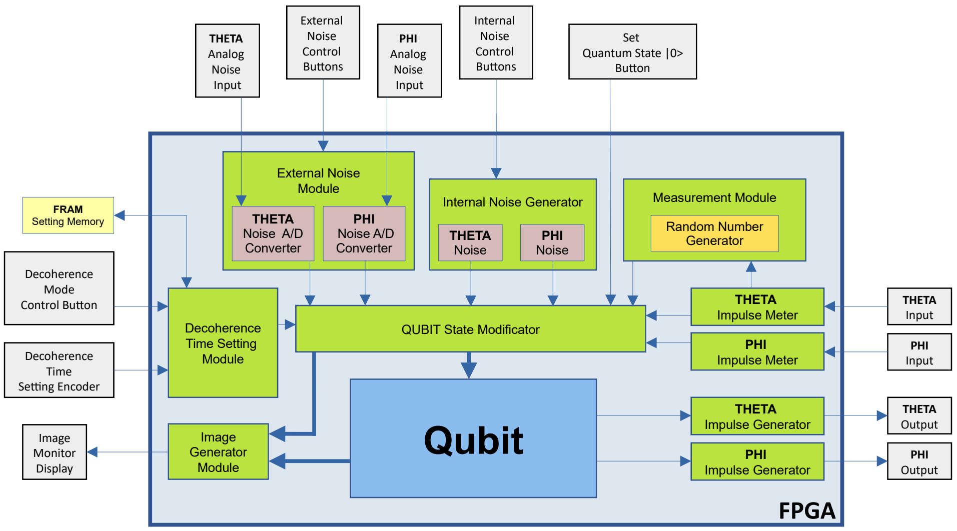

**Figure 3.** The QUBIT architecture

The QUBIT device is equipped with circuits responsible for handling internal and external noises and with a circuit responsible for emulating the phenomenon of quantum decoherence. The coherence time is set using a dedicated encoder and is stored in the non-volatile FRAM (Ferroelectric Random-Access Memory). The device is also equipped with an image generator that shows the qubit state marked on the Bloch sphere on the monitor screen. The main interaction with the qubit is through the THETA and PHI inputs and outputs.

To reset the THETA angle to zero, i.e. set the qubit to the quantum state $|0\rangle$, a pulse with a length ranging from 9000 to 9100 $\mu s$ (microseconds) must be applied to the THETA digital input.

To hide the qubit state vector in the Bloch sphere, a pulse with a length ranging from 9500 to 9600 $\mu s$ must be applied to the THETA digital input. To restore the visibility of the state vector, a pulse with a length ranging from 9700 to 9800 $\mu s$ must be applied to the THETA digital input.

To rota" the THETA angle by a specific positive angle $\alpha$, a pulse with a length T [$\mu s$] equal to the desired angle expressed in tenths of a degree should be sent to the THETA digital input, i.e. according to the formula:

$$T = \alpha \cdot 10\ [\mu s]$$

To rotate the state vector "along" the THETA angle by a specific negative angle $\alpha$, a pulse of length T [$\mu s$] must be sent to the THETA digital input, equal to the angle $\alpha$

expressed in tenths of a degree and increased by an additional 4000 $\mu s$, i.e. according to the formula:

$$T = 4000 + \alpha \cdot 10 \ [\mu s]$$

For example:

- An angle change of $+43.7°$ requires a pulse length of 437 $\mu s$,
- An angle change of $-43.7°$ requires a pulse length of 4437 $\mu s$,
- An angle change of $-143.7°$ requires a pulse length of 5437 $\mu s$.

To measure the qubit, a pulse with a length between 8000 and 8100 $\mu s$ must be applied to the THETA input. After this operation, the qubit is set to position $|0\rangle$ or position $|1\rangle$, according to the probability represented by the angle $\theta$ (THETA).

To zero the PHI angle, i.e. set the qubit phase to 0 degrees (0 °), a pulse with a length ranging from 9000 to 9100 $\mu s$ (microseconds) must be applied to the PHI digital input.

To rotate the qubit phase "along" the PHI angle by a specific positive angle $\alpha$, similarly to the THETA angle, send a pulse of length T [$\mu s$] to the PHI digital input, equal to the desired angle expressed in tenths of a degree, i.e. according to the formula:

$$T = \alpha \cdot 10 \ [\mu s]$$

To rotate the qubit phase "along" the PHI angle by a specific negative angle $\alpha$, a pulse of length T [$\mu s$] must be sent to the PHI digital input, equal to the angle $\alpha$ expressed in tenths of a degree and increased by 4000 $\mu s$, i.e. according to the formula:

$$T = 4000 + \alpha \cdot 10 \ [\mu s]$$

For example:

- An angle change of $+45.1°$ requires a pulse length of 451 $\mu s$,
- An angle change of $-45.1°$ requires a pulse length of 4451 $\mu s$,
- An angle change of $-245.3°$ requires a pulse length of 6453 $\mu s$.

The next ports in the QUBIT module are two digital outputs PHI and THETA, where appropriate electrical impulses are generated continuously and proportionally to the current state of the modeled qubit. The generated pulses for $\theta = 30°$ and $\phi = 45°$ are shown in Figure 4. A non-destructive reading of the current quantum state of the qubit involves measuring the length of the pulses generated at the THETA and PHI outputs. The lengths of these pulses show the values of the corresponding angles, expressed in tenths of a degree, minus one.

Therefore, in order to correctly read the THETA and PHI angles, the pulse lengths (T) measured in microseconds should be reduced by 1 and then divided by 10, i.e. according to the formulas:

$$THETA = \frac{T_{THETA} - 1}{10} \quad [°], \qquad PHI = \frac{T_{PHI} - 1}{10} \quad [°]$$

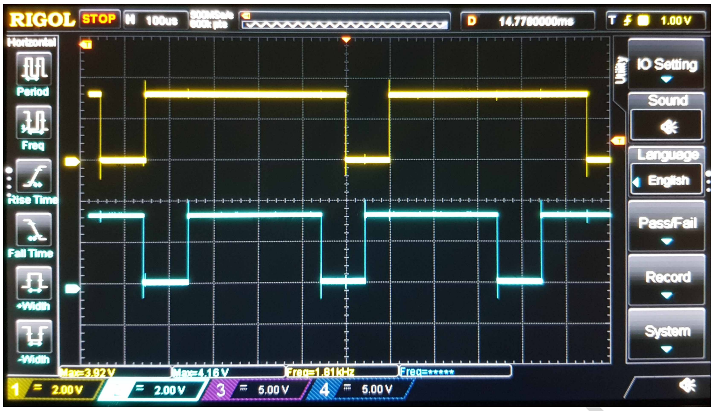

**Figure 4.** The pulses for $\theta = 30°$ (blue) and $\phi = 45°$ (yellow)

For example:

- The measured pulse of 743 $\mu s$ indicates an angle of 74.2 °
- The measured pulse of 1 $\mu s$ indicates an angle of 0 °

In "Decoherence Mode", the QUBIT module behaves like a real qubit not fully isolated from its surroundings. After switching this mode, information about the currently selected qubit coherence time and information about how much time is left until the system decoheres, appears in the upper right corner of the monitor screen. An example monitor screen in this mode is shown in Figure 5.

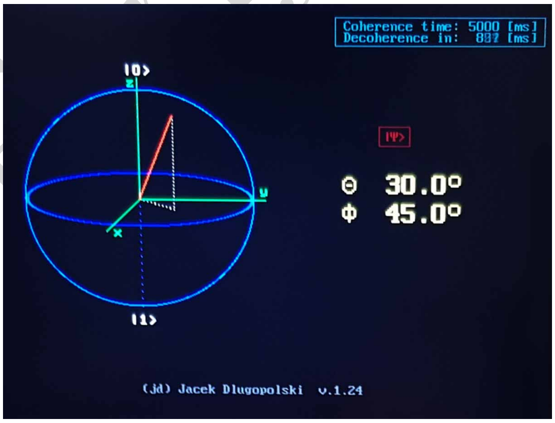

**Figure 5.** The monitor screen in the decoherence mode

The time during which the qubit is maintained in the quantum state (coherence time) can be changed by the user in the range from 1 ms to 9000 ms with a resolution of one millisecond using a dedicated encoder knob (the current setting is continuously saved in the non-volatile FRAM memory and is remembered after the power is turned off). This function only works in automatic decoherence mode, i.e. after switching the appropriate switch in the QUBIT module. In this mode, once the qubit has been set to a quantum state, it will automatically decohere after a time set according to the procedure discussed above. This will be equivalent to taking a measurement on it. After decoherence, its classical state will be either "0" or "1", according to the probabilities of the current quantum state just before decoherence. Any operations on the qubit will then no longer work until it is next set to the quantum state.

Another important function of the designed QUBIT is the "Noise" mode, in which the current quantum state of the qubit can be disturbed either by natural noise coming from the internal circuits of the FPGA or by noise artificially generated via an external function generator. Appropriate switches allow you to select the noise source. After turning on the "Noise" mode, a dedicated frame appears on the screen informing about this fact (Figure 6).

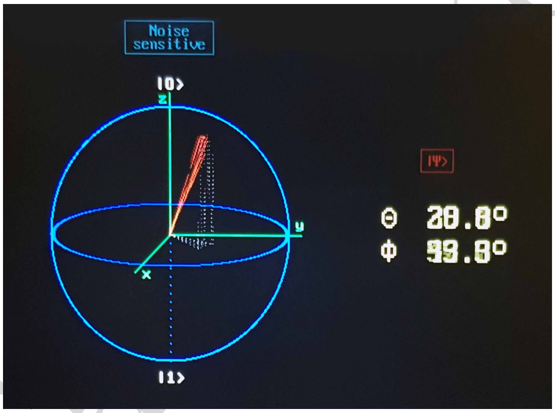

**Figure 6.** The monitor screen in the noise mode

Then, appropriate switches allow you to turn individual noises on and off, separately for each state component: THETA and PHI.

## 4.2. GENMET architecture

The QUBIT module discussed in the previous section can be controlled by any pulse generator and read by any pulse length meter. To create a complete system, a dedicated two-channel GENMET device (GENERator-METer) was built, capable of both generating and reading digital electrical impulses. With its help, the full control over the QUBIT module is gain. GENMET communicates with the user via the ETHERNET network, using the UDP/IP protocol. The BROADCAST mode used here makes it unnecessary to set an IP address. Therefore, without any configuration, the system will work in any local subnet. The block diagram of the GENMET module architecture is shown in Figure 7.

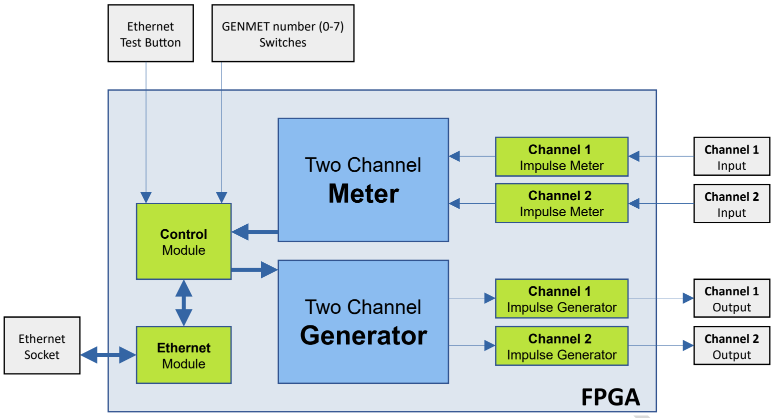

**Figure 7.** The GENMET architecture

The proposed solution allows for simultaneous operation of up to eight GENMET devices in one local Ethernet subnet. This makes it possible to build circuits consisting of up to eight qubits. GENMET is equipped with an Ethernet connector and four digital ports: two for generating pulses and two for measuring pulses.

GENMET therefore allows for the simultaneous generation of different pulses of a given duration on two independent outputs. The command to generate a pulse with specific parameters can be executed using a PC connected to the module via ETHERNET. The duration (length) of the pulses can be specified in units: from $[10ns]$ through $[\mu s]$, $[ms]$ to $[s]$. After power-on, GENMET is set to microseconds $[\mu s]$ mode. The outputs should be connected to the appropriate inputs of the QUBIT module and possibly to the appropriate oscilloscope inputs, as schematically shown in Figure 2.

GENMET, regardless of the generation of pulses, also allows for simultaneous two-channel measurement of pulses supplied to two of its inputs. When we connect these inputs to the outputs of the QUBIT module, the GENMET module will continuously measure the length of THETA and PHI pulses coming from the QUBIT module. The values of these measurements can be read at any time using a PC connected via the ETHERNET port.

By connecting a PC to GENMET and using any programming language that supports the UDP protocol, you can immediately control the virtual qubit, read its quantum state and measure it.

GENMET uses the UDP protocol and BROADCAST packets. It expects control commands on port 12000, and sends its responses to port 13000. All commands and responses are sent in text form. It does not require setting an IP address.

GENMET responds to the keyword: **genmet#**, where the "#" sign means a number from 0 to 7. Currently, it is possible to connect up to eight independent

GENMET devices to one local network. The GENMET number "#" is selected using three binary switches on the device.

After sending the "**genmet0 help**" command, we will receive a simplified response informing about the command syntax. From the computer's point of view, we see two channels generating pulses and two channels measuring pulses. We must ensure their proper association (connection) with the THETA and PHI signals of the QUBIT module.

When it comes to the generator of the GENMET device, for each command we can choose the unit in which we will give the duration of the generated pulse. The standard unit is [us] (microseconds). But there are also: [10ns], [ms] and [s] possible. Selecting [10ns] means that when specifying a pulse length of 5, we mean a pulse of 50 ns. In addition, we can specify the number of pulses in a single series using the phrase **"nr x"**, where "x" is their number (the default setting is "1", i.e. a single pulse) or instead, we can use the word **"continuous"**, and then the set pulse will be repeated continuously. The pulse parameters on individual channels are defined using the phrases **"low1 x"**, **"high1 x"** and **"low2 x"** and **"high2 x"**. Of course, "1" means the first channel and "2" means the second channel. The "x" variable should be in the range from 0 to 9999 (after power-on, the "x" values are set to 1800 $\mu s$). For example, the command:

**genmet0 start [us] nr 1 low1 400 high1 650**

will generate a single pulse on channel "1" on the GENMET0 device. After sending the command, GENMET will set the digital output of channel "1" to logical zero and wait 400 $\mu s$, then set the output to logical one and hold it for 650 $\mu s$. After this time, it will set channel "1" to logical zero again.

In order for any impulse to be generated, the word "start" must be included in the command. Missing this word or including the word "show" in the command will ensure that the transmitted command will not cause any changes to the generator outputs. GENMET will then only return information on port 13000 about the current settings. On the other hand, using the word "stop" in the command will stop the currently generated waveform and set the variable "nr" to 0. For example, the command:

**genmet0 stop high2**

will stop the signal generated on channel "2", and the command:

**genmet0 stop all**

will stop the generated signals on both channels. In turn command:

**genmet0 start [us] nr 5 low 1 400 high1 650**

will generate on channel "1" a series of five pulses discussed above. Command:

**genmet0 start [us] nr 5 low1 400 high1 650 continuous**

will generate 650 $\mu s$ pulses continuously with 400 $\mu s$ breaks. This is because when we specify **"nr 5"** and **"continuous"** concurrently, then **"continuous"** has higher priority. When we want to stop generating pulses, we should send the command without **"continuous"**, and it is best to send the command with the phrase **"nr 0"** or with the word **"stop"**. E.g. command:

**genmet0 start low1 nr 0** or **genmet0 low1 stop**

To generate a pulse on channel two, you can send the command:

**genmet0 start [us] nr 1 low2 541 high2 9000**

In this case, one pulse with a duration of 9000 $\mu s$, i.e. 9 $ms$, will appear on the second channel. The last set pulse can be repeated with the command:

**genmet0 start high2**

It repeats the last set pulse on the second channel. And the command:

**genmet0 start high1 high2** or **genmet0 start all**

repeats the last set pulses on both channels. Adding the word "show" will prevent the pulses from being generated, but information about the current settings of the two generator channels will be sent to the PC. Such information about current settings is also always sent when we invoke any valid commands. For example, the command:

**genmet0 [us] nr 1 high1 600 low1 400 start**

will return the message:

**note genmet0 starts [us] nr 1 low1 400 high1 600**

informing that a single pulse with a duration of 600 $\mu s$ was generated on the first channel.

Whereas, command:

**genmet0 show [us] nr 1 low1 400 high1 600**

will return the message:

**note genmet0 shows [us] nr 1 low1 400 high1 600**

which informs about the channel settings of the first generator, but the pulse will not be generated.

You can use any UDP terminal or any programming language to communicate with GENMET. Sending a command to GENMET from sample Python code is shown in Figure 8.

```
import socket

s=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)

s.sendto(bytes("genmet0 start [us] nr 1 low1 400 high1 2280 ", \
               "utf-8"), ('192.168.0.255', 12000))
```

**Figure 8.** Sending a command to GENMET in Python

As mentioned earlier, GENMET, in addition to generating pulses, also continuously measures the pulses fed to its two inputs. To read the last measurement, execute the command:

**genmet0 read**

and then the following text will be sent back to the PC on port 13000:

**note genmet0 reads [us] ch1: 682 ch2: 1437**

which means that a pulse with a duration of 682 $\mu s$ recently appeared on the first channel, and a pulse with a duration of 1437 $\mu s$ on the second channel. If this measurement concerns the QUBIT module, it means that two angles were measured, respectively: 68.1° and 143.6°. The measurement should (according to previous information) be reduced by 1 $\mu s$ and divided by 10. An example of reading measurement data in Python is shown on Figure 9.

The "data" variable will then contain the information received from GENMET. The presented program prints this information on the computer screen in the last line.

```
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)

r = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
r.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
r.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
r.bind(("",13000))

s.sendto(bytes("genmet0 read ", "utf-8"), ('192.168.0.255', 12000))
data,addr = r.recvfrom(13000)
print(data.decode())
```

**Figure 9.** Reading measurement data in Python

## 5. Result

Based on the assumptions and designs of individual elements described above, an actual working prototype of a physical qubit emulation system based on FGPA DE10-Lite [5] modules was built. The effect of this work is shown on Figure 10.

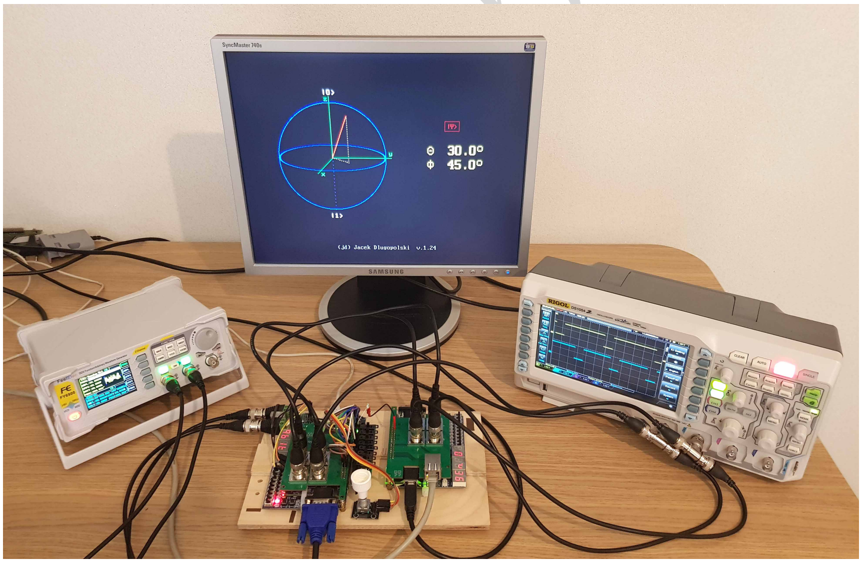

**Figure 10.** The system prototype

QUBIT and GENMET were built based on two FPGA modules, connected with each other with BNC cables and equipped with devices such as: a monitor, an oscilloscope, a function generator and a PC used to operate the system. The system built in this way was tested and it turned out to work in accordance with the original assumptions. It is possible, among others: setting the qubit coherence time and turning noise on and off, as well as performing any operations on the qubit.

## 5.1. Dedicated library

Based on the GENMET module control commands described in the previous chapter, the "jd_genmet_lib" library was developed in Python to support the QUBIT module. It provides the basic functions shown in Table 1.

**Table 1**
The "jd_genmet_lib" library functions

| | |
|---|---|
| snd(command_string) | Directly send UDP string command to the local ETHERNET network |
| set_device_name(name) | Set the device name for all of the functions below (default: "genmet0") |
| read_device_name() | Read the current device name |
| set_delay_time(t) | Set the delay time between commands in seconds (default is 0.05s) |
| setup_0_state() | Sets state to: $\|0\rangle$ |
| setup_1_state() | Sets state to: $\|1\rangle$ |
| setup_state(Theta, Phi) | Sets state to: $\cos\left(\frac{Theta}{2}\right) \cdot \|0\rangle + e^{i \cdot Phi} \cdot \sin\left(\frac{Theta}{2}\right) \cdot \|1\rangle$ "Theta" is <0 to 180> degrees and "Phi" is <0 to 360.0> degrees |
| hide_state() | Hide quantum state |
| unhide_state() | Unhide quantum state |
| read() | Read the qubit state (returns the Theta and Phi angles of a state vector) |
| measure() | Measure the qubit (returns 0 or 1) |
| degs_to_rad(Theta, Phi) | Convert Theta and Phi from degrees to radians |
| degs_to_xyz(Theta, Phi) | Convert Theta and Phi from degrees to Cartesian x,y,z |
| degs_to_amp(Theta, Phi) | Convert Theta and Phi from degrees to apmlitudes of the wave function |
| rotate_x(Angle) | Rotate X Gate "Angle" is <-360.0 to 360.0> degrees |
| rotate_y(Angle) | Rotate Y Gate "Angle" is <-360.0 to 360.0> degrees |
| rotate_z(Angle) | Rotate Z Gate "Angle" is <-360.0 to 360.0> degrees |
| rotate_theta(Angle): | Rotates Theta between <0 to 180> "Angle" is <-180.0 to 180.0> degrees |
| rotate_phi(Angle): | Rotate Phi between <0 to 360> "Angle" is <-360.0 to 360.0> degrees |
| ID() | Identity Gate |
| X() | Pauli-X (NOT) Gate |
| Y() | Pauli-Y Gate |
| Z() | Pauli-Z Gate |
| XSR() | Square root of Pauli-X Gate –¿ SQRT(NOT) |
| H() | Hadamard Gate |

The user can create new functions and sequences of operations based on this library. For example, after setting the qubit to a specific state, e.g. $\theta$=40° and $\phi$ =45°, a non-destructive reading of the quantum state is possible. Ideally, this reading should return the same value on each subsequent read attempt. Such qubit state readings can be made with the "read()" function from the library and, for example, with the simple Python program shown on Figure 11.

```
import time
import datetime
import string
import math
from jd_genmet_lib import *

def do_log(string1):
  t = datetime.datetime.fromtimestamp(time.time()).strftime('%H:%M:%S:%f')
  log = open(".\\data.txt", "a")
  print(t, " -> ", string1, file=log);log.flush()
  print(t, " -> ", string1)

log = open(".\\data.txt", "a"); print("\n\n", file=log);log.flush()
set_device_name("genmet0")
str1="   Device used:" + str(read_device_name()) + "\n"; do_log(str1)

str1=("   Theta   Phi");  do_log(str1)
for i in range(0,10):
  r=read() # Read the qubit
  Theta,Phi=r;
  str1=("   %5.1f   %5.1f" % (Theta,Phi) );do_log(str1)
  time.sleep(0.5)
```

**Figure 11.** Qubit state reading in Python

Figure 12a shows the result of sample readings without noise enabled. After turning on qubit internal noise for both parameters: $\theta$ and $\phi$, subsequent readings will be distored, as shown in Figure 12b.

a)

```
13:28:27:435917  ->     Device used:genmet0

13:28:27:435917  ->     Theta   Phi
13:28:27:503460  ->      40.0    45.0
13:28:28:055694  ->      40.0    45.0
13:28:28:607882  ->      40.0    45.0
13:28:29:160037  ->      40.0    45.0
13:28:29:712618  ->      40.0    45.0
13:28:30:264841  ->      40.0    45.0
13:28:30:817210  ->      40.0    45.0
13:28:31:369679  ->      40.0    45.0
13:28:31:921846  ->      40.0    45.0
13:28:32:474403  ->      40.0    45.0
```

b)

```
13:35:28:347785  ->     Device used:genmet0

13:35:28:347785  ->     Theta   Phi
13:35:28:412296  ->      39.8    44.9
13:35:28:963894  ->      39.8    44.8
13:35:29:516330  ->      39.9    44.8
13:35:30:068559  ->      39.6    44.9
13:35:30:621107  ->      39.8    44.8
13:35:31:173512  ->      39.8    45.2
13:35:31:726022  ->      39.9    44.9
13:35:32:278265  ->      39.9    44.8
13:35:32:830630  ->      40.1    44.9
13:35:33:383059  ->      39.8    44.6
```

**Figure 12.** Sample readings without noise a); sample reading with internal noise b)

The library also allows the conversion of parameters: $\theta$ and $\phi$ into the amplitudes of the base states $|0\rangle$ and $|1\rangle$. The function: ”degs_to_amp(Theta, Phi)” can be used for this purpose. An example of using this function is shown in Figure 13. The cnversion results are shown in Figure 14. Readings were also made here with internal noise turned on. Of course, the qubit state disturbed in this way will also affect the quantum measurement made using the ”measure()” function available in the described library.

```
import time
import datetime
import string
import math
from jd_genmet_lib import *

def do_log(string1):
  t = datetime.datetime.fromtimestamp(time.time()).strftime('%H:%M:%S:%f')
  log = open(".\\data.txt", "a")
  print(t, " -> ", string1, file=log);log.flush()
  print(t, " -> ", string1)

log = open(".\\data.txt", "a"); print("\n\n", file=log);log.flush()
set_device_name("genmet0")
str1="   Device used:" + str(read_device_name()) + "\n";do_log(str1)

str1=("   Theta   Phi        |0>            |1>");do_log(str1)
for i in range(0,10):
  r=read() # Read the qubit
  Theta,Phi=r; ar,br,bi=degs_to_amp(Theta,Phi)
  str1=("   %5.1f   %5.1f     %13.10f  %13.10f %+13.10f *i" % \
       (Theta,Phi,ar,br,bi) );do_log(str1)
  time.sleep(0.5)
```

**Figure 13.** Converting read angles to amplitudes in Python

```
13:44:16:285992  ->     Device used:genmet0

13:44:16:285992  ->     Theta   Phi        |0>            |1>
13:44:16:340292  ->      39.8    44.8      0.9402881270   0.2415233681 +0.2398430755 *i
13:44:16:891826  ->      40.1    44.8      0.9393937941   0.2432692664 +0.2415768274 *i
13:44:17:444393  ->      39.9    45.0      0.9399907318   0.2412648173 +0.2412648173 *i
13:44:17:996964  ->      39.9    44.9      0.9399907318   0.2416855361 +0.2408433635 *i
13:44:18:549387  ->      39.8    44.9      0.9402881270   0.2411043954 +0.2402642477 *i
13:44:19:101808  ->      39.8    44.9      0.9402881270   0.2411043954 +0.2402642477 *i
13:44:19:654333  ->      39.9    44.9      0.9399907318   0.2416855361 +0.2408433635 *i
13:44:20:206770  ->      39.8    44.9      0.9402881270   0.2411043954 +0.2402642477 *i
13:44:20:759131  ->      39.8    44.9      0.9402881270   0.2411043954 +0.2402642477 *i
13:44:21:311241  ->      39.8    44.8      0.9402881270   0.2415233681 +0.2398430755 *i
```

**Figure 14.** Printout of converted amplitudes.

## 5.2. Application examples

The main goal of the developed and built QUBIT was to create a research, educational and training environment for carrying out a number of experiments allowing future users of real quantum systems to become familiar with potential problems occurring in real quantum systems. Additionally, having such a device that does not require specialized low-temperature laboratory conditions makes it easier to look for algorithms and solutions to counteract the problems mentioned above.

A single QUBIT can, for example, be used to practice generating appropriate control pulses that allow the qubit state vector to rotate around any selected axis on the Bloch sphere. This will allow you to independently develop any single-qubit gates.

QUBIT is equipped with advanced physical interference features, discussed earlier. Hence, another example of using a single QUBIT is to use it for developing effective noise reduction algorithms in quantum systems. Finding an effective solution for a single qubit here can then scale to multi-qubit systems.

In addition, the device has been designed to also enable the creation of multi-qubit environments. GENMET, discussed in the previous chapters, has three special switches to set its unique name under which it is seen in the local Ethernet network. Thanks to this, up to eight independent GENMET devices can now be connected to one network (with minor modifications, this number can be significantly increased). Then each GENMET can control a separate QUBIT. This is schematically shown in Figure 15.

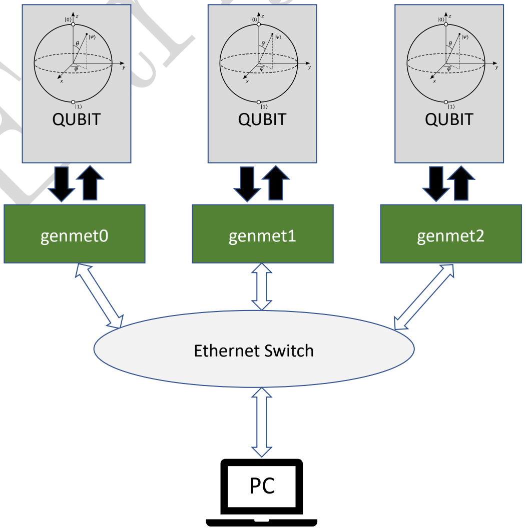

**Figure 15.** The multiQUBIT application

With such a configuration, an additional layer integrating all connected QUBITs can be created on any connected PC and based on any programming language This will make it possible to emulate entanglement between QUBITs and study various multi-qubit algorithms. Work of this type is currently underway, and the results will be presented in future articles.

## 6. Conclusion

To sum up, the constructed qubit model is much closer to the real quantum system than any other purely IT model. The change in the state of the qubit does not occur by simply sending selected values to the model, but requires the application of electrical impulses with precisely defined physical parameters, in particular the exact duration of these pulses. So this is exactly as is currently required in real quantum systems. The same applies to reading the state of a qubit. Such a readout requires a physical measurement of the duration of the pulses continuously generated by the qubit. During the process of forming and transmitting impulses, such a model is exposed to all kinds of disturbances and difficulties encountered in all kinds of real physical systems. No quantum system simulators that we are currently aware of offer this possibility.

Additionally, the presented model is equipped with mechanisms for intentionally generating noise in the system, both based on the internal noise of real qubits and based on noise usually coming from the environment. Natural noise occurring inside the FPGA circuitry is used to emulate the internal noise of the qubit. It is thanks to the naturalness of the phenomena occurring there that the process of generating internal noise is characterized by actual physical randomness.

On the other hand, any external electrical waveform generator is used to simulate ambient noise, so by selecting the appropriate waveform or combining a total noise signal from many different harmonics, many real-world phenomena that could potentially interfere with quantum systems can be physically simulated.

An additional advantage of the presented model is the ability to impose a coherence time regime, after which the model automatically loses its quantum state by making spontaneous self-measurements. This necessitates the real need to limit the number of quantum operations during one experiment session.

In addition, the ability to connect up to eight such qubits to a single local network simultaneously (and this number can be easily significantly expanded) allows the quantum circuits emulated in this way to be expanded to much more complex multi-qubit systems.

All the elements discussed above mean that the described model can be successfully used for initial emulation and study of quantum phenomena, before it is possible to safely start real experiments on real quantum systems.

## References

[1] Aminian M., Saeedi M., Saheb Zamani M., Sedighi M.: FPGA-Based Circuit Model Emulation of Quantum Algorithms. In: *2008 IEEE Computer Society Annual Symposium on VLSI*, pp. 399–404, 2008. doi: 10.1109/ISVLSI.2008.43.

[2] Burgholzer L., Bauer H., Wille R.: Hybrid Schrödinger-Feynman Simulation of Quantum Circuits With Decision Diagrams. In: *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*, IEEE, 2021. doi: 10.1109/qce52317.2021.00037.

[3] Chapeau-Blondeau F.: Modeling and Simulation of a Quantum Thermal Noise on the Qubit, *Fluctuation and Noise Letters*, vol. 21(06), 2022. doi: 10.1142/s0219477522500602.

[4] Cheng S., Cao C., Zhang C., Liu Y., Hou S.Y., Xu P., Zeng B.: Simulating noisy quantum circuits with matrix product density operators, *Physical Review Research*, vol. 3(2), 2021. doi: 10.1103/physrevresearch.3.023005.

[5] FPGA DE10-Lite. https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=218&No=1021. Accessed: 14/4/2024.

[6] Fujishima M., Inai K., Kitasho T., Hoh K.: 75- qubit Quantum Computing Emulator. In: *Extended Abstracts of the 2003 International Conference on Solid State Devices and Materials*, pp. 406–407, 2003. doi: 10.7567/SSDM.2003.P1-4.

[7] Grurl T., Fuù J., Hillmich S., Burgholzer L., Wille R.: Arrays vs. Decision Diagrams: A Case Study on Quantum Circuit Simulators. pp. 176–181, 2020. doi: 10.1109/ISMVL49045.2020.000-9.

[8] Hillmich S., Markov I.L., Wille R.: Just Like the Real Thing: Fast Weak Simulation of Quantum Computation. In: *2020 57th ACM/IEEE Design Automation Conference (DAC)*, IEEE, 2020. doi: 10.1109/dac18072.2020.9218555.

[9] IBM Qiskit Noise Models, https://qiskit.github.io/qiskit-aer/apidocs/aer_noise.html#quantum-error-functions. Accessed: 03/07/2024.

[10] IBM Quantum Composer, https://quantum-computing.ibm.com/composer/files/new. Accessed: 14/4/2024.

[11] Negovetic G., Perkowski M.A., Lukac M., Buller A.: Evolving Quantum Circuits and an FPGA-based Quantum Computing Emulator. 2002. https://api.semanticscholar.org/CorpusID:15995633.

[12] Quantum AI team and collaborators: qsim, 2020. doi: 10.5281/zenodo.4023103.

[13] Quirk: Quantum Circuit Simulator, https://algassert.com/quirk. Accessed: 14/4/2024.

[14] The Quantum Länd: Quantum Circuit Simulator, https://thequantumlaend.de/frontend/designer.php. Accessed: 14/4/2024.

[15] Wei K., Niwase R., Amano H., Yamaguchi Y., Miyoshi T.: A state vector quantum simulator working on FPGAs with extensible SATA storage. In: *2023 International Conference on Field Programmable Technology (ICFPT)*, pp. 272–273, 2023. doi: 10.1109/ICFPT59805.2023.00041.

[16] Xie T., Zhao Z., Xu S., Kong X., Yang Z., Wang M., Wang Y., Shi F., Du J.: 99.92%-Fidelity cnot Gates in Solids by Noise Filtering, *Physical Review Letters*, vol. 130, p. 030601, 2023. doi: 10.1103/PhysRevLett.130.030601.

[17] Zulehner A., Wille R.: Advanced Simulation of Quantum Computations, *arXiv*, 2018. doi: 10.48550/arXiv.1707.00865.

## Affiliations

**Jacek Długopolski**
AGH University of Krakow, Faculty of Computer Science, al. A. Mickiewicza 30, 30-059 Krakêw, Poland, dlugopol@agh.edu.pl

**Marcin Sadowski**
SONOVERO R&D, www.sonovero-rd.pl, Warsaw, Poland, msadowski@sonovero-rd.pl

**Wawrzyniec Suleja**
SONOVERO R&D, www.sonovero-rd.pl, Warsaw, Poland, wsuleja@sonovero-rd.pl

**Received**: 06.05.2024
**Revised**: 29.07.2024
**Accepted**: 29.07.2024