ISSN 2072-5981
doi: 10.26907/mrsej

Magnetic Resonance in Solids

Electronic Journal

Volume ***27***
Issue ***3***
Article No ***25310***
1-13 pages
2025
doi: **10.26907/mrsej-25310**

http://mrsej.kpfu.ru
http://mrsej.elpub.ru

Established and published by Kazan University[*]
Endorsed by International Society of Magnetic Resonance (ISMAR)
Registered by Russian Federation Committee on Press (#015140),
August 2, 1996
First Issue appeared on July 25, 1997

© Kazan Federal University (KFU)[†]

"*Magnetic Resonance in Solids. Electronic Journal*" (**MRSej**) is a peer-reviewed, all electronic journal, publishing articles which meet the highest standards of scientific quality in the field of basic research of a magnetic resonance in solids and related phenomena.

Indexed and abstracted by
Web of Science (ESCI, Clarivate Analytics, from 2015),White List (from 2023)
Scopus (Elsevier, from 2012), *RusIndexSC* (*eLibrary*, from 2006), *Google Scholar*, *DOAJ*, *ROAD*, *CyberLeninka* (from 2006), *SCImago Journal & Country Rank*, etc.

[*] Address: "Magnetic Resonance in Solids. Electronic Journal", Kazan Federal University; Kremlevskaya str., 18; Kazan 420008, Russia
[†] In Kazan University the Electron Paramagnetic Resonance (EPR) was discovered by Zavoisky E.K. in 1944.

NMR experiment control system with an interval programmable generator

A.S. Alexandrov*, D.L. Melnikova, D.S. Ivanov, V.D. Skirda

Kazan Federal University, Kazan 420008, Russia

**aaleksan@kpfu.ru*

(received November 29, 2025; revised December 23, 2025; accepted December 26, 2025; published December 30, 2025)

Exceptionally high demands on the time control accuracy and the experiment flexibility are imposed by modern nuclear magnetic resonance (NMR) spectroscopy and magnetic resonance imaging (MRI). A critical component of NMR equipment is a specialized device responsible for synchronizing and controlling all nodes and devices during the experiment. In this paper, a device control system with interval programmable generator (programmator) of arbitrary pulse sequences implemented on programmable logic integrated circuits (FPGAs) is proposed. The designed system architecture provides simultaneous control of up to 245 external devices in mode both asynchronous and synchronous with the pulse sequence via special instruction words. A universal instruction format used for both internal and external devices provides a simple way to connect any new external device to the control system. The internal instructions set, which includes intervals, nested loops and macros allow to develop and implement the highest complexity pulse sequences in NMR experiments. The duration of the interval set by a single instruction can take values from 20 ns to 0.334 seconds. For any fragments of the pulse sequence, it is possible to form cycles with up to 2^{16} repetitions, including nested (cycle within cycle) cycles with a nest depth of 16. Thus, usage a memory for only 2048 instruction words allows to reach the total duration of the generated sequence 10^{80} s (with a 20 ns resolution).

PACS: 71.70.Ch, 75.10.Dg, 76.30.Kg, 71.70.Ej.

Keywords: nuclear magnetic resonance, pulse sequences, programmator, synchronization, programmable logic integrated circuits.

1. Introduction

Modern nuclear magnetic resonance (NMR) equipment is highly complex and include a set of different devices: high-frequency (HF) generators, power amplifiers, heterodyne and super-heterodyne receiving amplifiers, HF switches, gradient pulse amplifiers, shimming coil current sources, and various controllers (sample temperature, sample rotation speed, reflected power of the power amplifier, gradient pulse current, and so on). Earlier, among the devices one could distinguish a pulse sequence generator. However, today it is insufficient to generate only video pulses that determine the radio frequency (RF) pulse sequence. To conduct a modern NMR experiment one requires synchronicity between all devices involved in generating and recieving the NMR signal. In other words, it is necessary to ensure stable time intervals not only between RF pulses, but also to ensure synchronous, and in some cases, coherent relations between many other actions: magnetic field gradient pulse generators, changes in their amplitudes and signs, setup of the RF and ADC phases, etc. At the same time, fulfillment of the coherence condition, a synchronicity of all specified actions with the resonance frequency, will provide the ability to generate RF pulses modulated in amplitude, frequency, and phase, to use arbitrary phase sequences and phase cycles in the experiment, to ensure synchronous registration and digital quadrature detection of the signal, and to apply digital signal processing methods. Devices requiring indicated synchronicity will henceforth be called synchronous devices and a special device that provided this synchronicity, a programmable generator or simple, a programmator.

It is necessary to note that for each manufacturer of NMR equipment, the programmator is unique and deeply integrated system. It has own hardware and software interface, and cannot be purchased separately and used to develop new devices, since this will require using the entire system including software. The only commercially available exception here is "PulseBlaster" pulse generator by SpinCore Technologies [1]. It has excellent time and frequency characteristics, but has a limited model series and, most importantly, has a closed specification. This brings some limits when use "PulseBlaster" for designing a new NMR equipment especially mobile or bench-top ones. Another one promising system is "OPENCORE NMR", a scientific and open-source project by Takeda [2, 3]. The OPENCORE NMR system has an open specification and based on the following ideology: all functions of NMR devices that can be implemented digitally are implemented on programmable logic integrated circuits (FPGAs).

A significant disadvantage of both systems is that they have no simple way to add a new type of synchronous device. This ability is an important feature in terms of developing new equipment or implementing new NMR techniques. If a developer or a researcher requires a new receiver or transmitter or gradient channel, he must just plug-in a new device without the need to rebuild the entire system from scratch. The progress in MRI is driven by an increase in the number of transmitting-receiving and gradient channels. For example, tens new devices had to be connected to implement PatLock [4]. Hennig and Schultz themselves write that the system is rebuilt each time, and it's a significant achievement when, in addition to three linear gradients, several nonlinear ones with a corresponding array of receiving coils are added.

The purpose of this paper is to present a development designed to significantly simplify all procedures for expanding the functionality of NMR equipment. This applies not only to the manufacturing of new setups, but also to the research phase, when new ideas arise about the need to conduct a particular experiment, with new control elements or effects on the object being studied.

In the following sections, we will outline the main concepts of the proposed control system. Section 2 describes the structure (the architecture) of the device control system, the key component of which is a device control unit. Section 3 further explores the design and features of the device control unit. Sections 4 are devoted to the main component of the device control unit – the interval programmer. Section 5 presents the features and specifications of the second key component of the device management system – the physical interface for connecting and synchronizing devices. In the last section 6 consideration about FPGA model selection for implementing the control system is given.

2. Devices control system

Let us consider the general control scheme of the NMR equipment (Figure 1). The device control unit (DCU) performs a number of functions. It contains the programmator itself, a computer communication interface, and several devices essential for the NMR equipment, such as an RF pulse generator and a digital detector.

Obviously, the pulse sequence designing, measurement data storage, processing, logging, and visualization are performed on the computer. The computer communication interface ensures fault-free data transmission and reception with high throughput. This is important due to the large volumes of measured data and can optimize the control unit's intermediate buffer memory for temporary storage of measured data.

According to the block diagram shown in Figure 1, devices are controlled by device control

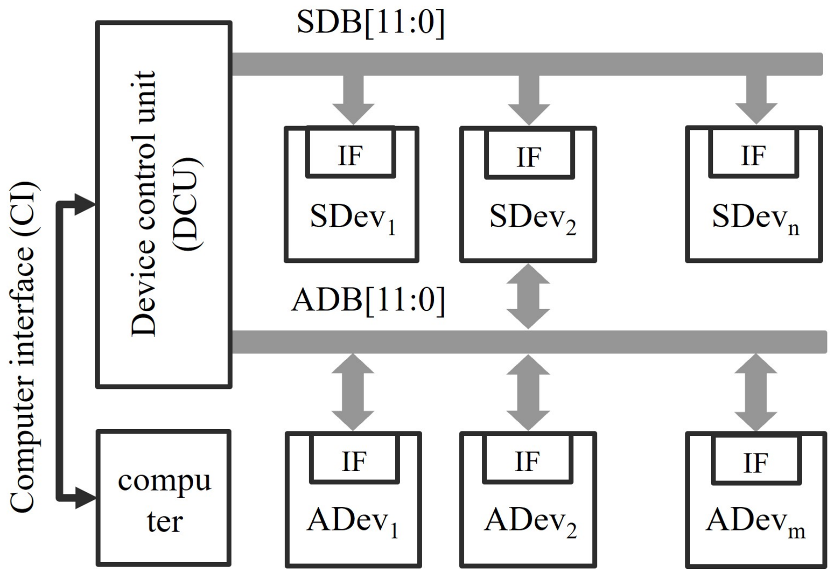


Figure 1. Block diagram of device control system.

unit (DCU) via two parallel buses: the synchronous device bus (SDB) and the asynchronous device bus (ADB). Synchronous (SDev$_i$) and asynchronous (ADev$_i$) devices are connected to the buses and have a "response" interface IF. The interface node IF is also an inherent part of the control system. It has a unique address, allows any device to receive instructions via these buses (either ADB or SDB or both), and generates control signals for the device's analog circuitry according to the argument of the instruction. For this purpose, IF may has a DAC and/or ADC onboard depending on the requirements of a specific device. These instructions generated by DCU contain the address ("to") and execution information ("how") in their argument. A timed sequence of instructions sent over control buses carry out the synchronization of all devices.

It is worth noting that to generate arbitrary pulse sequences in real time, instructions to read information from devices over the SDB bus must be disabled. Nevertheless, the devices, both synchronous and asynchronous, require feedback (information reading), such as measuring the output stage temperature, reflected power level, sample temperature, etc. This task is accomplished using an asynchronous device bus (ADB), which is controlled by the DCU and enables the writing and reading of instructions to devices independently of the pulse sequence execution. The implementation of identically structured SDB and ADB interfaces allows any device to be connected to either of the two buses, or both at once.

3. Device control unit

Detailed implementation of the NMR device control unit is shown in Figure 2. The main modules of the control unit are: an interval programmator, a digital synchronous detector (DSD), a digital low-pass filter (LPF), and a radio-frequency pulse generator (RFG). These modules are designed as asynchronous devices, i.e., they are connected to ADB, have their own unique address, and receive asynchronous instructions via ADB.

The programmator is a relatively simple digital device. It includes a random-access memory for storing instruction words, an address counter for pointing current instruction, and an internal instruction decoder. Its main task is to set the instructions to the output synchronous bus at exact time without any delays. At the current level of technology, the optimal solution for implementing a programmator is a programmable logic integrated circuit (FPGA). The

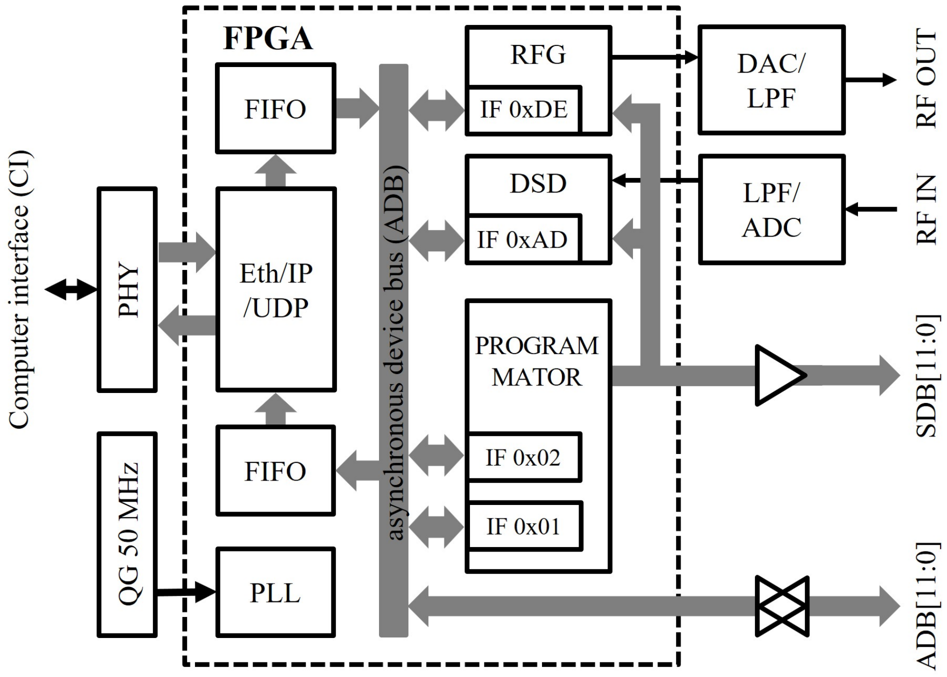


Figure 2. Block diagram of device control unit (DCU).

operating logic of the FPGA is specified by setting (programming) a specific configuration of connections of logical elements and modules inside the FPGA into a given electrical circuit. It is important that such an implementation of a complex electrical circuit inside the FPGA can be, if necessary, reprogrammed, either partially or completely [5, 6]. Application of modern FPGAs in NMR equipment allows one to solve many important problems, including not only the formation of pulse sequences, but also the formation of the RF pulses themselves with the specification of their frequency, phase and amplitude with an accuracy of up to ns and low jitter [7]. The architecture of modern FPGAs also allows one to realize algorithms of digital signal filtering in real time, and perform frequency synthesis and multiplication functions [7–9]. All mentioned features make FPGAs very promising for use in complex NMR equipment, including MRI [10–12].

The asynchronous RF pulse generator (RFG) with address 0xDE, shown in the block diagram of Figure 2, generates excitation RF pulses. To generate RF pulses of required frequency and phase, the RPG structure includes direct digital synthesis (DDS) oscillator. Two multipliers connected consequently to DDS digital output used to set required amplitude and shape of the RF pulse. The RFG device is interfaced with the analog circuitry of the NMR transmitter path via a DAC. This DAC could, in principle, be a part of the transmitter. The digital synchronous detector (DSD) with address 0xAD performs synchronous detection and pre-filtering of the digitized NMR signal. The DSD device is interfaced with the NMR receiver path via an ADC to digitize the NMR signal at pre-selected intermediate frequency. The resonance frequency needs to be converted to the intermediate frequency by the mixer circuitry in the receiver path.

The 1000BASE-T Ethernet standard was chosen as the computer communication interface (CI on Figure 2), with a maximum speed of up to 1 Gbps when connected via Category 5e twisted-pair cable. This interface was originally designed (see, for example, [13]) for stable data transmission over long distances in conditions of various external interference. Consequently, it is virtually free of the drawbacks inherent in other interfaces (PCI, USB), such as the inability to automatically restore the connection when exposed to strong interference. Hardware solutions for implementing Ethernet are currently very widely used and represent well-established and inexpensive single chip network controllers with a standardized GMII/MII format output.

The physical layer (PHY) of the Ethernet interface is implemented using a physical layer controller supporting the GMII protocol (e.g., RLT8211). The Ethernet data link, network, and transport (UPD – User Datagram Protocol) layers are implemented on the FPGA. After sequentially decoding messages from Ethernet layers are transited directly to the asynchronous device bus (ADB) via asynchronous FIFO queues. UDP transport layer is chosen for the following reasons: firstly, the programmator is not intended to operate in highly branched networks, and secondly, implementing UDP on the FPGA requires significantly fewer FPGA resources.

4. Programmator

4.1. Base principals

The two main functions of programmator are follows: a) receiving/writing the entire specified sequence of executed actions from the control computer as a single package of instruction words into its random-access memory (RAM); b) in automatic mode or upon an external command such as "start", sequentially selecting instructions from the memory stack at a high clock frequency and distributing them among all devices of the NMR equipment. Each instruction word contains an address byte and a set of bytes (at least three) containing information on the value and sign of the specified amplitude, phase, frequency, time interval, number of repetitions, etc. In other words, each instruction word contains information on "who should execute and how to execute". Moreover, all instruction words intended for setting time intervals have their own specific address ("interval" instruction), at which a delay timer is automatically started to form the corresponding time interval in the programmed sequence of actions. Immediately after the timer is started, all other instructions scheduled for execution in the current time interval are sequentially selected from memory and sent to the appropriate devices. Upon selecting the next "interval" instruction indicating next time interval the programmator switches to a wait mode. A new value will be written to the timer only after executing previous "interval" instruction. Thus, in each time interval, only those instructions stored in the memory stack immediately after the current "interval" instruction, but before the next one, are executed.

Execution of instructions for generating cycles or macros is organized in a similar manner. The "cycle" instruction contains a cycle start pointer (an address in the programmator's RAM) and the number of repetitions in the cycle. The "macro" instruction specifies the start address of the pulse sequence fragment designated as a macro. Each "cycle" and "macro" instruction can be assigned its own index (name). This allows for generating multiple macros and cycles, including nested ones, in a single sequence. Thus, the programmator essentially acts as a device manager, generating instructions such as "who, when, and how" to execute the programmed action. Instruction such as "interval", "cycle", and "macro" are internal instructions of the programmator itself. These internal instructions executed sequentially create a time scale (schedule) for other instructions. This not only optimizes the resources of the programmator as an electronic device, but most importantly, correlates with the general principles of NMR signal generation. According to which [14, 15] the response of a spin system to various pulsed exposures (RF, magnetic fields, etc.) is resulted in the generation of signals whose occurrence and duration are, in one or another way, related to the duration and timing of all previous interactions. The use of an interevent interval counter, rather than an absolute time counter, is one of the features of the designed programmator.

Beyond the devices such as a transmitter, receiver, ADC, magnetic field gradient generators, etc., NMR equipment may include devices that are not directly involved in the generating and receiving of the NMR signal. Those devices can logically be called asynchronous. These can include various kinds of controllers of the external magnetic field, experimental conditions (temperature, pressure, etc.), changing the sample or its position, etc. Clearly, the coordination of asynchronous devices does not require such serious time constraints and may well allow delays in control instructions supported by modern computers ($\sim$10 ms). As a result, in most cases such devices are controlled directly from a computer via any available interface. However, as the number of such auxiliary devices increases, the number of different connections and interfaces also increases. That is greatly complicates the configuration of a new NMR equipment. An effective solution in this case would be to connect all asynchronous devices to the computer via a separate bidirectional interface. That would support a sufficient number of parallel connections, and would be controlled, if necessary, not only directly by the computer, but also by the programmator.

If a uniform instruction word format is chosen for synchronous and asynchronous devices, it would be logical to implement such an interface node directly in the programmator. Thus, control of any device initially operating as an asynchronous device can be transferred to control by instructions generated synchronously with the execution of the pulse sequence, and vice versa.

4.2. Design

Figure 3 shows the block diagram of the interval programmator. Whole programmator circuitry is clocked by a single direct (CLK) and inverse ($\overline{CLK}$) clock signal generated by the FPGA internal PLL (phase-locked loop) generator. As it was told previously programmator has two ADB interfaces IF1 and IF2. IF2 interface is used to receive state change instruction from ADB or return the programmer current state. The programmator has three main states: “start”, “stop”, and “load”. State is changed when an instruction with a specific argument is received by IF2. The argument is decoded, and the state decoder generates control signals (LOAD, START) for changes in the programmator state. In the “stop” state, the address counter is cleared and blocked (by CE = 0), the synchronous output bus is inactive. In the “load” state the programmator is prepared to receive an instruction sequence. The sequence is received from ADB by IF1 and then written to instruction memory RAM via SEQ[11:0] bus.

In the “start” state, the address counter increments its value (ADB[11:0]) each clock cycle, and the instructions from the RAM are selected and set on the memory output bus CMD[31:0] one by one each cycle. The most significant byte (address/instruction code) of the output memory bus is decoded by instruction decoder. This determines whether the code corresponds to the internal instruction set of the programmator. If so, a corresponding strobe is generated: TI – for execution TIME instruction; CY/EL – for execution CYCLE/ELCYC instructions; MA/OR – for execution MACRO/ORCAM instructions. If not, the instruction is considered as external, the EX strobe generated and goes to the output buffer of the SDB (EXT) and then to the output of the SDB within four cycles. The address counter is blocked for the duration of these four cycles, i.e., the programmator does not proceed to execute the next instruction word.

The basis for generating an arbitrary pulse sequence is the internal interval instruction TIME (here and below, only instruction mnemonics are given, not their binary representations, which, in principle, can be chosen arbitrarily). When this instruction is executed, its argument is loaded into the decrementing counter (located in the TIME block) and the counter is activated. The address counter is not blocked, and the programmator proceeds to the next instruction, and so

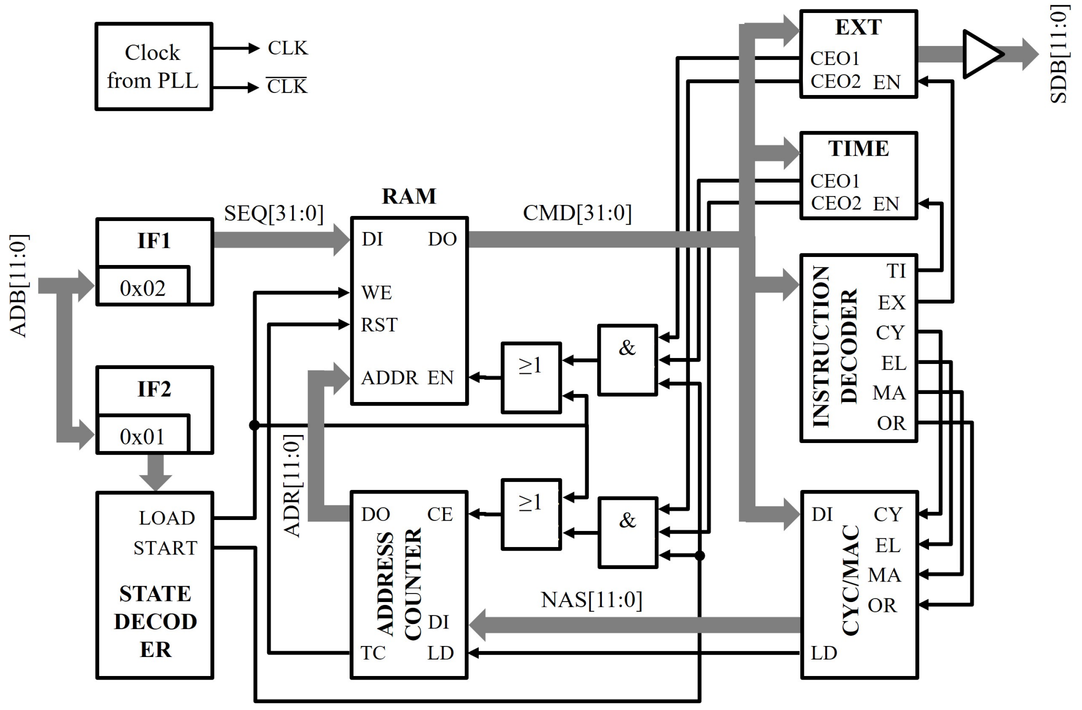


Figure 3. Block diagram of interval programmator.

on until the next TIME instruction appears at CMD[31:0]. If the interval counter is still active, the address counter and RAM output will be blocked (by CEO1, CEO2 = 0) until the counter finished. Thus, execution of the next TIME instruction will begin after a time interval equal to the period of the main clock frequency T_{clk} multiplied by the argument of the current TIME instruction. TIME instructions sequentially located in memory (RAM) form time intervals. At the beginning of their execution, all instructions immediately following the TIME instruction will be sent to the corresponding addresses (devices), in the sequence in which they will be selected from RAM as a result of incrementing of the address counter output. The minimum duration of the time interval is equal to the period of the main clock frequency $T_{\min} = T_{\text{clk}}$, and the maximum duration will be determined by the counter bit depth n: $T_{\max} = (2n - 1)T_{\text{clk}}$. Summary data on the implemented internal instructions are given in the table 1.

Table 1. Internal programmator instructions.

Instruction	**Code**	**Argument**	**Description**
IDLE	0x00	–	no action
TIME	0xF1	$time[23:0]$	execute a time interval of duration $time$
MACRO	0xF6	$addr[11:0]$	go to an instruction located in RAM at the $addr$
ORCAM	0xF7	–	return from executing last macros
CYCLE	0xF3	$n[15:0]$	begin a cycle of $n+1$ repetitions
ELCYC	0xF4	–	return to the beginning of cycle or end cycle

The next required internal instructions are the CYCLE (cycle start) instruction and ELCYC (cycle end) instruction. The CYCLE instruction argument is the number of cycle repetitions. For the ELCYC instruction, the argument is irrelevant. When the CYCLE instruction is executed, its argument is placed in the decrementing cycle counter (located in the CYC/MAC block), and the counter is activated. The address following immediately after the current address of CYCLE instruction is also pushed onto LIFO-1 stack (located in the CYC/MAC block). The cycle counter value is decremented during the ELCYC instruction execution, and the previously stored value of the cycle start address from the LIFO-1 stack (NAS[11:0]) is loaded (LD = 1) into the instruction address counter. As a result, the pulse sequence execution returns to the cycle start. The cycle exit occurs when the cycle counter reaches the "0" state, this state blocks overwriting address counter during ELCYC instruction execution. Thus, the number of cycle repetitions will be equal to $N + 1$, where N is the CYCLE instruction argument. The cycle counter is selected as 16-bit and, accordingly, the maximum repetitions are 2^{16}.

An important feature is the ability to form nested loops, with a nesting depth equal to the LIFO-1 stack length (in our case, 16). When the CYCLE instruction (CYCLE-2) is encountered again within a loop, its argument (N2) is loaded into the cycle counter, and the current value of the cycle counter of the previously executed CYCLE instruction (CYCLE-1) is placed on the LIFO-2 stack (memory of the number of remaining repetitions). The values in the LIFO-1 stack are also shifted forward by one position, and a new loop start address is written. The pulse sequence then proceeds to the repeated execution of the fragment determined by the boundaries of the nested loop (CYCLE-2). Upon completion of the nested loop (execution of the specified number of repetitions $N2 + 1$), the address values of the mentioned memory stacks are shifted back by one position, and execution of the outer loop continues. Thus, the fragment of the sequence designated by the boundaries of the nested loop will be repeated $(N2 + 1)(N1 + 1)$ times, where $N1$ is the specified number of cycles in the CYCLE-1 instruction.

At first glance, it might seem that the three instructions described (TIME, CYCLE and ELCYC) are, in principle, more than sufficient for generating sequences of virtually any complexity and duration. This is especially true given the ability to create nested loops (nesting depth 16) and define independent cycles for individual fragments of a pulse sequence. However, often may arise situations where a complex fragment of a pulse sequence must be repeated once at any designated time point(s) in the current sequence of actions. In this case, it makes sense to store this fragment in the programmator RAM once. Then, during execution, at the required time point, use the corresponding pointer to rewrite the address counter to the beginning of this section for execution within the given time interval and subsequent continuation of the programmed sequence. This not only optimizes the memory requirements for the programmator, but also ensures the convenience of generating such sequences, both at the level of the pulse sequence developer and at the level of the operator (user). To implement this function, two more internal instructions are provided: the MACRO instruction – an unconditional jump to any specified address of RAM, and the ORCAM instruction, which indicates the end of the selected fragment of the sequence.

When executing the MACRO instruction, the address counter is loaded (LD = 1) with the value (NAS[11:0]) specified in the MACRO instruction argument, causing the programmator to begin execute the instructions stored in memory at the specified address. The address immediately following the MACRO instruction for returning from the macro is stored in a separate register during execution the MACRO instruction. Return to this stored address occurs when the ORCAM instruction appears at the RAM output. Described circuitry is located in the CYC/MAC block and share several elements with CYCLE circuitry.

Thus, the proposed programmator structure allows any fragment of a pulse sequence to be repeated multiple times by adding two instructions at the beginning and end of the selected fragment: CYCLE and ELCYC, respectively. Moreover, since they are executed independently, there can be multiple such fragments in the overall sequence. The only limitation is the capacity of RAM memory, or more precisely, the number of addresses in it. Also, by designating the boundaries of any fragment of the pulse sequence first with the MACRO instruction and then with the ORCAM instruction, this selected fragment can be repeated in any other time interval of the pulse sequence. It is necessary simply to insert a MACRO instruction with corresponding argument in the desired time interval to create such a fragment. It is permissible to designate several fragments at once with macros.

5. Instruction words and control bus structure

Each instruction word in the programmator memory consists of two fields: 1) a unique instruction/device address code; 2) an instruction argument field. As already noted, the number of devices in modern NMR systems is relatively large; however, in practice, this value never exceeds several dozen. Therefore, a one byte is sufficient for the instruction code field. When selecting the argument field width, it is important to follow the principle that, to ensure high performance, the whole value range of internal instructions arguments should fit within the length of one instruction word. Based on these considerations, a 24-bit data word appears optimal. For example, for the TIME instruction, this allows to set argument values from 0 to $2^{24} - 1$. So, using a 24-bit interval counter at a clock frequency of 50 MHz, this will ensure the generation of time intervals up to 0.334 sec with a resolution of 20 ns. At a clock frequency of 10 MHz, quite acceptable for NMR time resolution of 100 ns is provided. Maximum value of a single time interval in this case is 1.677 seconds. Programming even longer time intervals is easily achieved by composing them from several consecutive TIME instructions. For the CYCLE instruction argument only the 16 least significant bits of the three bytes of the argument field was used. This corresponds to a maximum of 65536 repetitions of a single cycle, and 2^{256} repetitions for 16 nested cycles.

When discussing information transfer between various devices and components of the NMR instrument, it is important to ensure that the instruction word is written to the device via the ADB or SDB in a relatively small number of clock cycles, and that the physical buses do not have too many conductive wires. Based on these considerations, the optimal transmission is one byte of the instruction word per clock cycle. In this case, the bus should contain eight independent data lines DT[7:0] and three service lines for signal management: DE – the "data enabled" signal; RE – the "read enabled" signal; CLK – the bus clock signal. Therefore, the required bit depth is 12, and the time required to transfer one instruction word in four clock cycles is 80 ns at a clock frequency of 50 MHz. Figure 4 shows the timing diagram for writing an instruction word to a device via ADB/SDB interface. The start of writing an instruction word to the device is initiated on the negative edge of the clock signal when $DE = 0$ and $RE = 0$. At this point, the most significant byte of the instruction word - the instruction code - is written (designated as CMD1 and CMD2 in Figure 4). In the following three clock cycles, $DE = 1$, and 3 bytes of the instruction word argument field (designated B2-B0 in Figure 4) are sequentially written to the device.

Reading a device occurs when $RE = 1$, which is only available for ADB. The data read diagram will look similar to the write diagram shown, differing only in that the instruction code byte on the bus (i.e., the read request) is generated by the device control unit (Figure 2), and the three data bytes (the response) are generated by the device whose unique code matches the code written in the address byte.

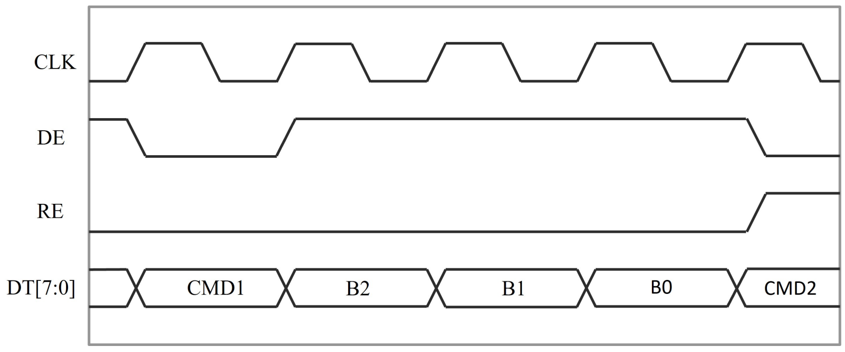


Figure 4. Instruction word write time diagram

At this point it is worth to pay attention to another one feature of designed system. Time to rewrite entire instruction RAM would be equal to length of the RAM multiplied by time need to write one word. Thus, to renew the operating pulse sequence one need $2048 * 80\,\text{ns} = 163\,\mu\text{s}$ at a clock frequency of 50 MHz, i.e., almost instantly in terms of typical for NMR waiting times.

6. Device control unit verification

The described control system can, in principle, be implemented on various modern FPGAs. We will outline a number of critical FPGA characteristics required for implementing the developed control system. For the programmator implementation, the most critical FPGA component is the instructions memory. The FPGA model must contain integrated block RAM. The minimum required capacity can be estimated at approximately 4 KB, which would be sufficient for, for example, 1000 4-byte instruction words. Another critical element of the programmer is the interval counter for the TIME instruction, since its bit width, along with the clock frequency, determines the resolution of the time interval and the maximum interval value specified by a single instruction.

The RF pulse generator module requires the ability to operate at a high frequency of over 250 MHz for DDS, which in turn requires a PLL generator integrated into the FPGA to generate such a high clock frequency. In addition, the RFG contains multipliers, a set of multi-bit registers (about 24) and ROM tables (for storing the shapes of generated oscillations and the shapes of RF pulse envelopes), which in turn consumes a significant number of FPGA cells.

Finally, implementing a computer communication system requires a large register set (approximately 256 for decoding network protocol messages) and the ability to operate at frequencies of approximately 125 MHz. A FPGA that meets these minimum requirements can be used to implement the control system. In our case, we selected the Xilinx XC6SLX9, the entry-level device in the SPARTAN 6 series. It contains 4800 flip-flops, 18 kB of block RAM, two PLL generators, eight integrated 18-bit multipliers, and can operate at frequencies up to 1080 MHz.

The FPGA configuration was designed using Xilinx's ISE Design Suite development system. The configuration was verified through various tests in a logic simulator. A printed circuit board was designed and assembled for this configuration (Figure 5). Then the performance of the control system was verified.

Let us give several examples of the designed control system capabilities. Figure 6 shows RFG output signal while generating three RF pulses different shapes: rectangular, Gaussian, and SINC. The pulse duration was $64\,\mu$s, with an interval between pulses was $400\,\mu$s. Such values are large for relaxation measurements, but are typical for imaging.

Figure 7 shows a receiver output signal during the action of generated Carr-Purcell-Meiboom-

Gill (CPMG) pulse sequence; the $\pi/2$ pulse duration was 16 μs, with an interval between pulse was 2 ms. At the figure one can clearly observe the echo signals from Midel-7131 oil sample.

These examples demonstrate the correspondence between the specified and observed parameters of the generated sequence and, therefore, demonstrate the operability of the control system.

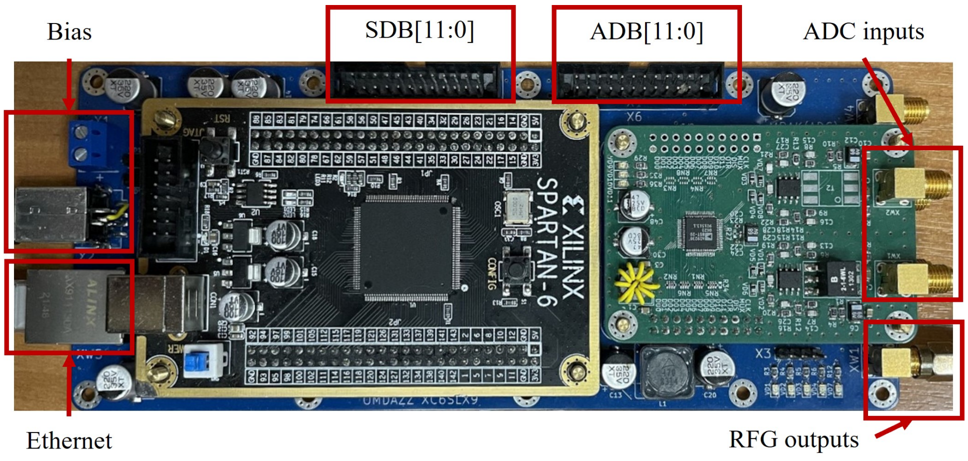
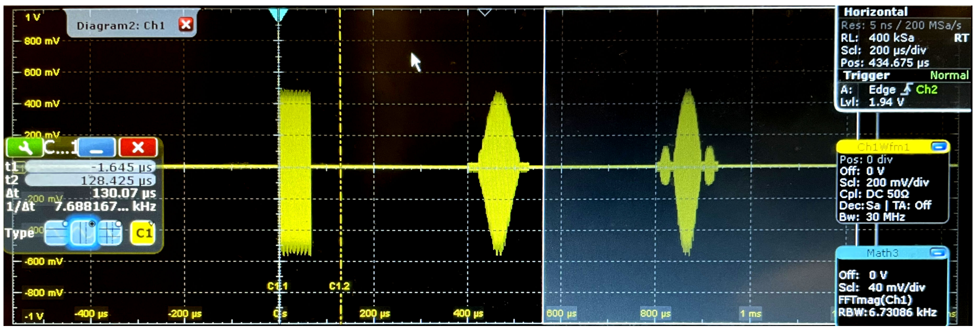


Figure 5. Assembled printed circuit board of the control system



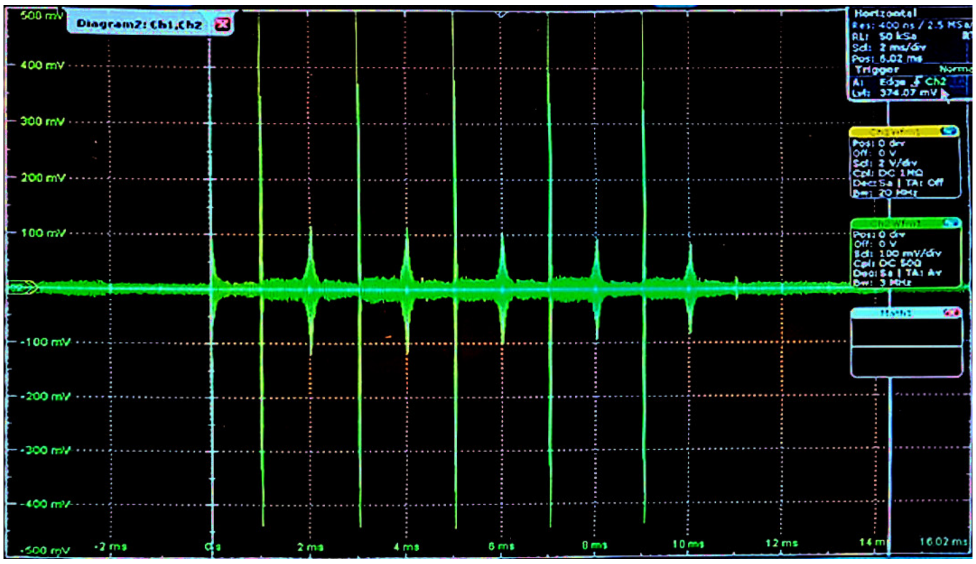
t1 -1.645 µs
t2 128.425 µs
Δt 130.07 µs
1/Δt 7.688167... kHz

Horizontal
Res: 5 ns / 200 MSa/s
RL: 400 kSa RT
Scl: 200 µs/div
Pos: 434.675 µs
Trigger Normal
A: Edge Ch2
Lvl: 1.94 V

Ch1Wfm1
Pos: 0 div
Off: 0 V
Scl: 200 mV/div
Cpl: DC 50Ω
Dec: Sa | TA: Off
Bw: 30 MHz

Math3
Off: 0 V
Scl: 40 mV/div
FFTmag(Ch1)
RBW: 6.73086 kHz

Figure 6. RFG output signal during generating sequence of various shape RF pulses



Horizontal
Res: 400 ns / 2.5 MSa/s
RL: 50 kSa
Scl: 2 ms/div
Pos: 6.02 ms
Trigger Normal
A: Edge Ch2
Lvl: 374.07 mV

Ch1Wfm1
Pos: 0 div
Off: 0 V
Scl: 2 V/div
Cpl: DC 1MΩ
Dec: Sa | TA: Off
Bw: 20 MHz

Ch2Wfm1
Pos: 0 div
Off: 0 V
Scl: 100 mV/div
Cpl: DC 50Ω
Dec: Sa | TA: Av
Bw: 3 MHz

Math1

Figure 7. Receiver output signal during the action of generated Carr-Purcell-Meiboom-Gill (CPMG) pulse sequence

7. Conclusion

The NMR device control system was designed. The interval programmator, RF pulse generator and digital detector are implemented on a Xilinx XC6SLX9 FPGA. Taking into account some of the resources allocated for encoding internal instructions, the control system enables synchronization of up to 245 external devices, including those configured on the same FPGA. These include, in particular, an RF pulse generator (with specified duration, amplitude, and phase) and a digital synchronous detector. The duration of one interval in the pulse train can take value up to 0.334 seconds in 20 ns increments at a clock frequency of 50 MHz. The clock frequency can be changed if necessary. The programmator can generate nested cyclic sequences with up to 2^{16} repetitions and up to 16 nested loops. The instruction structure, including nested loops and macros, enables the development of pulse sequences of the highest complexity. The programmator interface architecture with external devices allows for easy connection to external devices, regardless of whether they are synchronous or asynchronous with respect to the programmed pulse sequence.

The interface nodes, which allow to connect existing or newly developed devices to the control system are implemented on a Xilinx XC95144 CPLDs. The response interface utilizes approximately 30% of the CPLD resources. The remaining resources can be used to generate device-specific digital control circuits, including simple on/off circuits, trigger circuits, or simple digital processing circuits (addition, subtraction, and accumulation). Depending on the requirements of a specific device interface node may has a DAC and/or ADC onboard.

It is worth noting that reliable communication with devices via synchronous and asynchronous buses is ensured at frequencies up to 50 MHz. However, the programmator and devices implemented on the FPGA can operate at higher frequencies. The designed system architecture allows the optimization (reducing) of the clock frequency for both control buses for just external devices.

Acknowledgments

This work was funded by the subsidy allocated to Kazan Federal University for the state assignment in the sphere of scientific activities number FZSM-2023-0016.

References

1. *PulseBlaster - Programmable Pulse and Delay Generator PCIe Board SP46, Owner's Manual*, SpinCore Technologies Inc., Gainesville, FL 32653, USA (2025).
2. Takeda K., *Rev. of Sci. Instr.* **78**, 033103 (2007).
3. Takeda K., *J. of Magn. Reson.* **192**, 218 (2008).
4. Hennig J., Welz A. M., Schultz G., Korvink J., Liu Z., Speck O., Zaitsev M., *Magnetic Resonance Materials in Physics, Biology and Medicine* **21**, 5 (2008).
5. Monmasson E., Cirstea M. N., *IEEE Trans. on Indust. Electr.* **54**, 1824 (2007).
6. Kuon I., Tessier R., Rose J., *Foundations and Trends in Electr. Des. Autom.* **2**, 135 (2008).
7. Tayler M. C., Bodenstedt S., *J. of Magn. Reson.* **362**, 107665 (2024).
8. Harris M., Harris L., *Digital Design and Computer Architecture* (Elsevier, 2007).
9. Sun L., Savory J. J., Warncke K., *Concepts in Magn. Reson. Part B* **43**, 100 (2013).

10. Othman M., Abdullah N., Rusli N., in *2010 IEEE Symp. on Indust. Electr. and Appl. (ISIEA)* (2010) pp. 623–628.

11. Li L., Wyrwicz A. M., *J. of Magn. Reson.* **255**, 51 (2015).

12. Gebhardt P., Wehner J., Weissler B., Botnar R., Marsden P., Schulz V., *Phys. in Med. and Biology* **61**, 3500 (2016).

13. Law D., Dove D., D'Ambrosia J., Hajduczenia M., Laubach M., Carlson S., *IEEE Commun. Magazine* **51**, 88 (2013).

14. Abragam A., *The principles of nuclear magnetism* (Oxford university press, 1961).

15. Slichter C. P., *Principles of magnetic resonance* (Springer Science & Business Media, 2013).