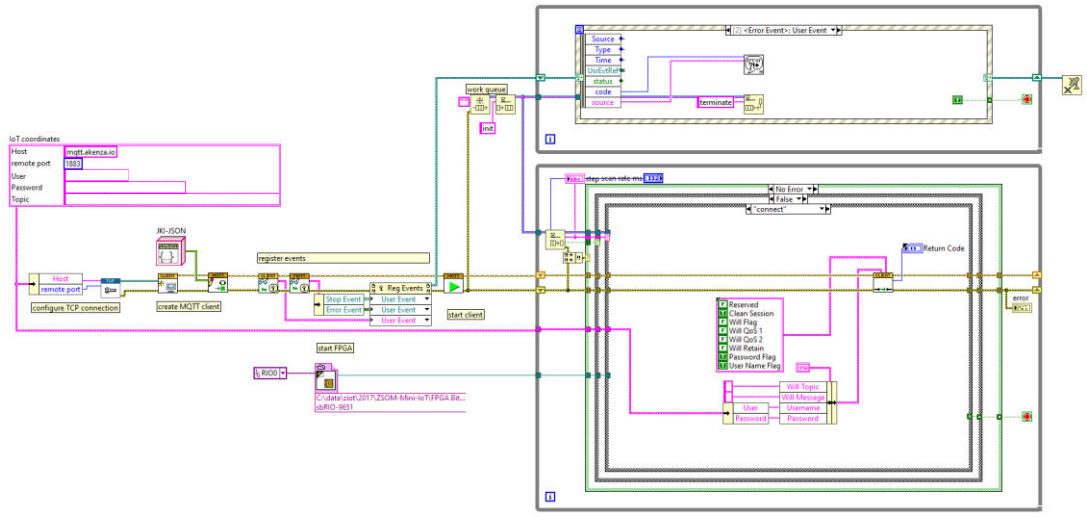
Whitepaper

The Charm and Benefit of Graphical Programming with LabVIEW



This whitepaper is ideal for anyone who is looking for a quick and easy introduction to graphical programming and wants to familiarize themselves with LabVIEW and its data-centric principle. The advantages and disadvantages of this graphical method are also addressed, as well as the importance of the community. And last but not least, there is an outlook on how NI R&D envisions the future of LabVIEW in the world of smart software assistants.

Contents

LabVIEW stands for "**Lab**oratory **V**irtual **I**nstrumentation **E**ngineering **W**orkbench". It is a development environment designed for measurement and control tasks. Since its first release on the Mac in 1986, LabVIEW has been developed as an interface to measurement and automation devices. This language is comfortably abstract, yet hardware-oriented and has grown up with physical signals. Over the years, NI developed scalable and modular hardware: the PXI, the CompactRIO and the single board RIO, to name a few. This led to the NI platform, which seamlessly combines hardware and software and measurably simplifies application development thanks to a high level of abstraction.

1 What is the difference between LabVIEW and "G"?

The key and strength of LabVIEW lies in "G", the graphical programming notation. "G" is the graphical code designed for data flow that we create in the LabVIEW development environment and which is based on a mathematical graph network. Many consider LabVIEW and "G" to be one and the same, and for the sake of simplicity we will now do the same here. Interested readers can find further details on LabVIEW in this WIKIPEDIA entry or in this technical article.

2 An Easy-to-Understand Analogy to Reality

Let's take a closer look at the LabVIEW terminology and basic philosophy. Every program in LabVIEW is a virtual instrument: a **VI.** So if I create a program called "**Energy Measurement**", the program file is called: **Energy Measurement.vi**.

LabVIEW was developed from the outset by engineers for engineers. This is why many terms build a bridge to the everyday life of technical developers, such as an oscilloscope.

Fig. 1 | In the real instrument (left, image: Tektronix), electrons flow in wires between components. In the virtual instrument (right, image: NI), data flows between nodes.

A real oscilloscope has signal inputs and a screen output with a control panel. In LabVIEW, there are different types of inputs and outputs that generally mimic what we can see on a measurement device. In the current oscilloscope example, a front panel shows what I see on the control panel of the real thing: knobs, buttons and a screen with the signal on it.

Let's take a look under the hood. In the real instrument, electrons flow in wires from component A to component B. As with the real instrument, we can also look inside the virtual instrument. There are also wires. But instead of electrons, it is data that flows from node to node. In LabVIEW, the actual code is a software diagram analogous to the hardware circuit, which functions according to the data flow.

3 A "real" Programming Language ?

A LabVIEW diagram, as we call this circuit diagram, is continuously translated into machine code and executed on the target system at the touch of a button. Even though it is graphical and not text-oriented, LabVIEW still uses familiar, classic programming principles. For example, constructs such as data types, variables, loops, recursion, branching, event handling and object-oriented approaches can also be found here. What's more, at any point in the graphical code, we can directly access underlying hardware or functions of the operating system, e.g. "wait 10 milliseconds". All in all, LabVIEW differs from purely model-based approaches or low- and no-code. The question in the title can be answered with "Yes".

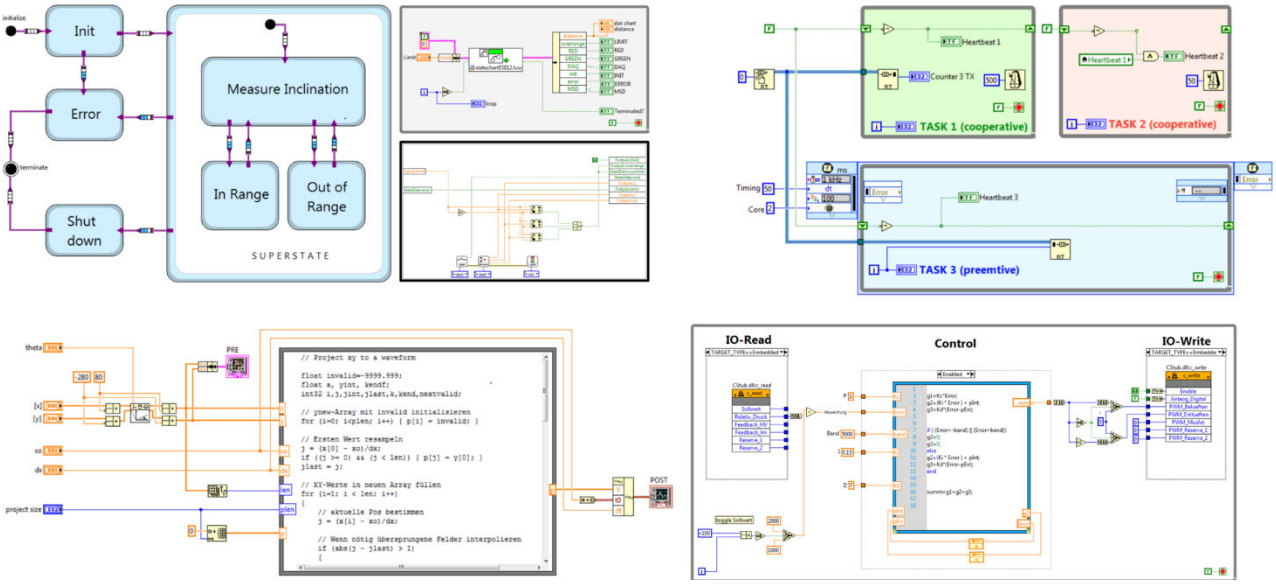


Fig. 2 | Perhaps this is the wrong question in view of the many programming models that can be flexibly embedded and executed in LabVIEW?

4 A Graph and Data as the underlying Architecture

LabVIEW is data-driven and defines data as the main concept behind every program. Instead of sequential instructions, it is the data flow that determines the order of execution. Behind this is a mathematical construct of a petri network, which uses graphs with nodes and edges to represent process states. So the principle behind LabVIEW is a powerful combination of a graph model, a data driven philosophy and real-time execution, which allows us to master several time domains at once: from milliseconds to nanoseconds.

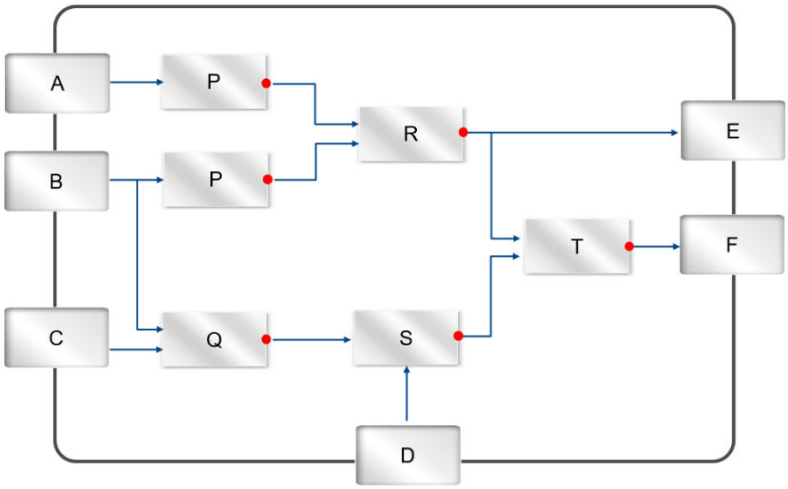


Fig. 3 | A powerful Petri net of nodes and edges works behind LabVIEW.
It is executed in parallel by nature.

In addition, LabVIEW can be used across the entire value chain.

1. Proof of Concept (PoC)
2. Minimum Viable Products
3. Prototype
4. Series Product
5. Testing

This makes it a good example of how a language and development environment that has matured over decades can keep pace with today's most modern trends and requirements.

5 The Secret behind LabVIEW's Productivity

Programming in LabVIEW is measurably faster compared to text-based languages. The reason for this lies in the way our brain works and how it perceives graphical code. Our neurons and synapses show a certain similarity to the nodes and edges of a LabVIEW program. A graphical block diagram visually representing the flow of data between nodes, seems more intuitive for some software developers than individual, consecutive text instructions.

6 Advantages and Disadvantages of LabVIEW

Advantages

1. The graphical user interface is flexible and easy to use. Most engineers and scientists can learn the application very quickly.
2. LabVIEW offers a universal platform and powerful libraries and frameworks for numerous applications in different areas.
3. Different time domains can be mastered using the same approach:
   from [ms] (PC) to [µs] (microcontroller) to [ns] (FPGA).
4. Thanks to the same language for the entire value chain (proof-of-concept, minimum viable products MVP's, prototypes, series product and test), the code developed in these different phases can be reused.
5. LabVIEW can be used on third-party hardware and integrates C/C++, Python and M code. Nowadays, some parts of the environment are planned to become open source.
6. There is a large, vibrant and globally active community.

Disadvantages

1. LabVIEW is proprietary. Some companies may not want to use a product that comes from a single source and does not follow the open source philosophy.
2. Its simplicity makes it tempting to "play" and thus to create ad-hoc programs that later grow organically. However, serious software engineering and a scalable architecture are a prerequisite for large applications, even with graphical approaches.
3. Operating costs - although in line with similar industry products - should be considered before implementation. LabVIEW is not a low-cost solution! However, the LabVIEW Community Edition is now free for Students (Reference: NI-Connect Keynote 2024).
4. Graphic code comes at a cost, especially in the price-sensitive embedded sector. Reason: overhead and hardware. Low-cost microcontrollers are out of reach. The minimum requirement for a reliable application is the SOM and this hardware has its price.
5. For those who have been used to text programming for a long time, graphical programming may take some time to get used to.
6. LabVIEW users will have to wait for the advantages of generative AI, which is currently used in text-based programming. At the time of writing (mid 2024), NI R&D is working on it: code name "Nigel"

7 The power of the NI Community - it's ok to have fun!

The true power of NI and LabVIEW lies in its globally active community. It exchanges ideas in countless forums and maintains local user groups. The emotional closeness of these programmers to LabVIEW is always in the air. In addition to many advantages, we often struggle with disadvantages in everyday life. On the whole, however, the community agrees that LabVIEW supports our way of thinking, makes us strong, productive and successful and that graphical programming is simply fun. The former NI slogan comes to mind: "It's ok to have fun!"

Fig. 4 | The Swiss LabVIEW User Group founded in spring 2024.

The Swiss LabVIEW User Group is one such local group. It was founded in spring of 2024 by Daniel Roth (PI Electronics AG) and Marco Schmid (Schmid Elektronik AG). A simple idea turned into a successful event with over 100 participants, 20 speakers, 6 tracks, 20 presentations, 7 LabVIEW training sessions and 2 hands-on workshops on team & leadership. The relaxed and electrifying startup feeling was magical... For the Swiss LabVIEW community, this event was an important and positive gathering that filled a multi-year vacuum. The three most important findings:

1. NI remains strategically independent in the new constellation under EMERSON and R&D is once again actively driving LabVIEW forward.
2. The community spirit is reminiscent of the early days of NIDays, when the presentations were similarly "nerdy".
3. We as LabVIEW users in the community are ready to increase our knowledge by sharing.

Become a member of the following two local user groups on the NI website:
Swiss LabVIEW User Group and Swiss LabVIEW Embedded User Group!

8 The Future of LabVIEW with Generative AI

LabVIEW has long been a pioneer in the world of modern software development, giving its users a competitive edge year after year that was well ahead of its time. Together with NI's consistent hardware strategy, this led to the highly efficient development platform that we know and appreciate today and made it easy to get started with an otherwise complicated technology. Even today, this makes the often individual LabVIEW user in his company a successful "NI fox" who can quickly solve even complex tasks.

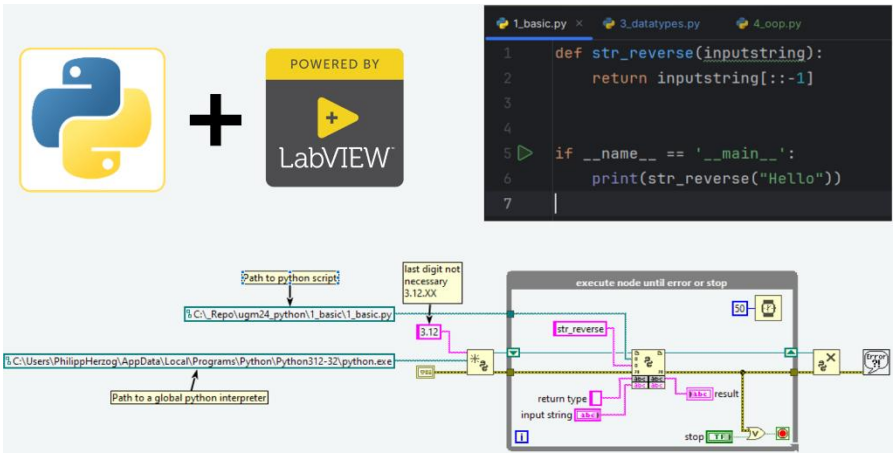


Fig. 5 | Why choose between LabVIEW and Python, when both languages can be elegantly combined? (Image: Herzog Engineering)

NI has neglected the academic market in recent years, losing its former de facto lead to Python. With Python, a new, equally easy-to-learn and very powerful programming language came into play and took over the role of top dog. And it is open source! Combined with the recent trend of programming assistants, this is likely to lead to a quantum leap and could overtake previously successful paradigms such as graphical programming.

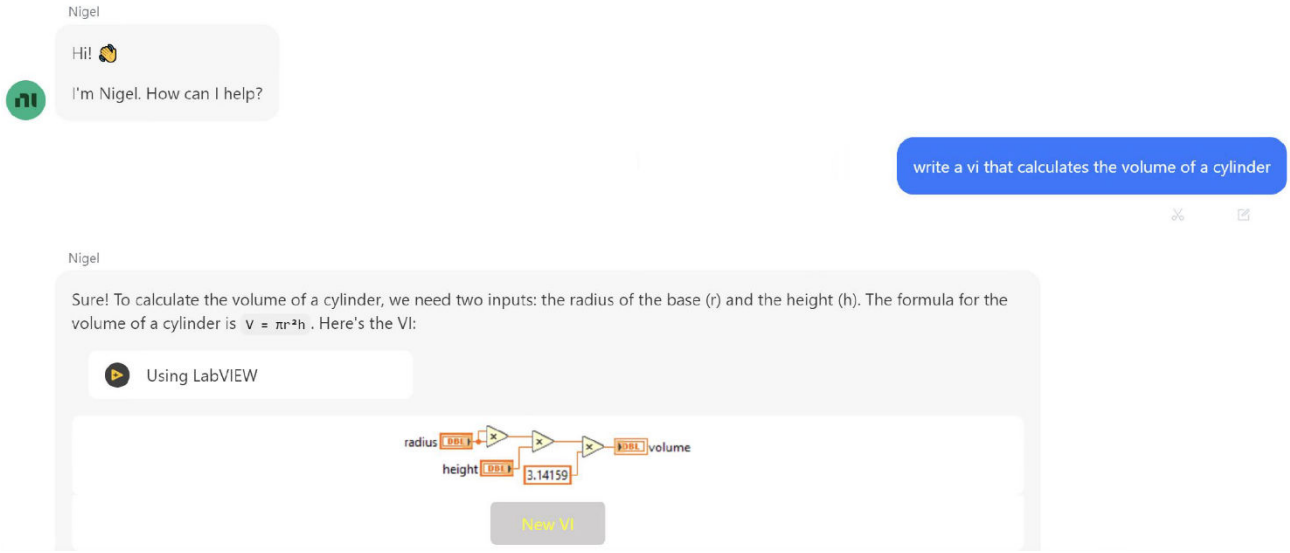


Fig. 6 | The NI version of an AI assistant (Image: NI)

What if the graphical power and charm of LabVIEW could be combined with the efficiency of AI-based coding? The answer from NI R&D is "Nigel"! This future smart assistant was presented for the first time at NI Connect 2023 and again in 2024. Nigel is NI's mascot: the eagle in the former blue National Instruments logo. At the Swiss LabVIEW User Group Event, we presented how this intelligent assistant feels in everyday programming. Nigel can do something like:

- Write VI's! Typical case: I need a 5th order low-pass filter with a cut-off frequency of 50kHz. As a result, I get a code snippet that I drag & drop into my application.
- Analyze VI's that have been inherited from a previous user
- Write VI descriptions and generate icons
- Accelerate Quick Drop
- Read device specifications and write tests for them
- VI Set up unit tests

Conclusion: Nigel will offer programmers in the domain-specific LabVIEW development environment what current language models deliver for text-based languages.

9 Glossary

Chapter 1	LabVIEW	Stands for graphical programming with the language «G»
	Language «G»	Graphical code based on data flow
	Graph	Mathematical network of nodes and edges
Chapter 2	VI	VI = Virtual Instrument = Function block in LabVIEW
	Oscilloscope	Electronic measuring device
	Hardware Circuit	Graphical representation of an electronic circuit
	Data flow	It is not the sequence that determines the programme flow, but the data
Chapter 3	Machine code	A computer program in a form that the microprocessor understands
	Target System	The computer on which machine code is executed
	Programming Principles	The basics of a typical programming language such as constants, variables, branching, loops, recursion, etc
	Operating System	A collection of programs for the operation of a microcontroller
	Model-based	Creating executable software from abstract models
	Low-Code	Visual, simple programming
	No-Code	Programming via configuration of graphic blocks
Chapter 4	Petri network	Graphical representation of process states with nodes and edges
	Graph model	The maths model underlying the graph (nodes & edges). model
	Data-driven	All organisation and administration revolve around data
	Real-time	Specified time that certain processes may consume
	Proof of Concept	Basis for deciding whether a project can be implemented
	Minimum Viable Products	Only have the most essential functions to test an idea. The short form is called: MVP
	Prototype	Tangible representation of a design concept, close to the series product
	Series Product	Can be multiplied by production in a series
	Testing	The function of hardware and/or software is verified
Chapter 5	Neurons	A nerve cell as the basic unit of our nervous system
	Synapses	A connection between two nerve cells
	Nodes	Here: a connection point in computer science
	Edges	A connection between two nodes

Chapter 6	Frameworks	A basic structure for software
	Microcontroller	Here: small, powerful computers for embedded systems
	FPGA	Reconfigurable logic: Field Programmable Gate Array
	C/C++	C is the most common language for microcontrollers. C++ is object-orientated.
	Python	Universal, higher programming language, often runs interpreted.
	M-Code	Code generated in M, optimised for mathematical problems
	Community	A community of people pursuing the same goal
	Proprietary	Software that is manufacturer-specific, such as NI in this case.
	Open Source	Publicly accessible code that is shared in communities
	Software Engineering	Processes and technology for the production and development of software
	Software Architecture	Modularisation, frameworks, object orientation, design patterns
	Embedded	Combination of hardware and software for embedded systems
	SOM	Here: the System on Module (sbRIO9651) from NI
Chapter 7	Local LabVIEW User Groups	A local group of LabVIEW users who exchange information and share experiences on a specific topic.
	Swiss LabVIEW User Group	The user group founded by PI Electronics and Schmid Elektronik, whose members are mainly from the DACH region.
	Swiss LabVIEW Embedded User Group	The user group founded by Schmid Elektronik focusses on graphically programmable embedded solutions around the NI SOM
Chapter 8	NI Connect	The NI Event in Austin, which has replaced the former NIWeek
Chapter 9	Generative AI	Artificial intelligence (AI) models that generate digital content
	Copilot	An AI-supporting tool that is used in MS Office
	Nigel	The AI Assistant, which can be used within the LabVIEW development environment in the future. As of today, It is still under development.
	Quick-Drop	Rapid insertion of VI from the pallet
	Unit Tests	Detects errors within software
	Domain specific	Here in the context of domain-specific programming languages. In contrast to general languages such as C/C++, they are designed to solve problems in a very specific area. In LabVIEW, for example, the acquisition, processing, storage and communication of data and signals is very simple and possible at a high level of abstraction.

Münchwilen, Switzerland, June 2025, Marco Schmid, marco.schmid@schmid-elektronik.ch