Baltic J. Modern Computing, Vol. 4 (2016), No. 1, 89-97

Improved LabVIEW Code Generation

Evita VAVILINA, Gatis GAIGALS

Ventspils University College, Inzenieru Street 101, Ventspils, LV-3600, Latvia

evita.vavilina@venta.lv, gatis.gaigals@venta.lv

Abstract. Before implementation into hardware signal processing algorithms are tested in simulation mode. LabVIEW provides highly convenient environment for simulation development and also tools for generation of simulation environment that can include simulation itself and collection of simulation data. Despite the fact that these tools use LabVIEW for code generation, it is not easy to understand the principles of code generation and effectively develop simulation generators. This paper presents toolbox for improved LabVIEW code generation. The developed toolbox is based on standard LabVIEW code generation functions maximally simplifying the application and minimizing the necessary amount of tools for code generation.
This paper consists of theoretical part about LabVIEW code generation methods, practical part about principles of LabVIEW code generation using scripting and a graphical presentation of improved LabVIEW code generation advantages. The presented graphical results show that the improved LabVIEW code generation is simpler (thus better) and more understandable for practical realization and the code generator is clearer and more comprehensible than the original one.

Keywords: automatic programming, parallel programming, object oriented modeling, signal processing algorithms, virtual prototyping

1 Introduction

Before implementation into hardware signal processing algorithms are tested in simulation mode. Conventional simulation environments are not well suited for simulating massive parallel signal processing algorithms, which refer to parallel signal processing of an array of signals.

Most popular simulation environments, such as Matlab, use sequential signal processing simulation model, graphical environments – Simulink and LabVIEW (WEB (c)) come with development difficulties. Both graphical simulation environments provide convenient parallel signal processing path, but parallel code complexity exceeds the ability by humans to develop precise and accurate codes. Moreover, LabVIEW provides also scripting tools for generation of simulation environment, which is then well suited for development of simulation codes applicable in the most commonly used hardware for signal processing – field - programmable gate array (FPGA). The greatest advantages of FPGA are the rapid non-stop technology development – increase in speed performance and parallel code execution. Therefore simulation code should exclude any kind of loops to make full use of the FPGA advantages.

This is where LabVIEW tools for simulation code generation become efficient with unlimited, e.g. loop, replication facilities. Using LabVIEW generation tools it is possible to generate a new simulation code in LabVIEW environment. Once a generation code prototype with LabVIEW tools is created, there is no extra effort needed to generate

unlimited size or number of this kind of simulation codes. The only limit is the number of available resources on the target device. At this point the main difficulty for reaching unlimited simulation code generation is the complexity in the application of existing LabVIEW code generation tools. The principle of LabVIEW code generation tools is not easy to understand and not even worth considering effective development of simulation code generators.

In general LabVIEW is a very convenient environment for developing simple and complex signal processing algorithms graphically from the perspective of electrical engineers. LabVIEW provides error tracking tools for easy code debugging. When it comes to complex signal processing algorithm development it is more convenient and effective to use LabVIEW simulation generation tools. These tools facilitate the repetitive actions, where the main benefits are time and effort economy, as well as accuracy and correctness in generating complex codes. For example, using these tools a complex parallel signal processing simulation code with guaranteed code correctness can be generated.

As mentioned before, the provided standard LabVIEW tools are too sophisticated for an easy application and understanding in developing any kind of complexity simulation codes. Therefore, not all of the advantages provided by LabVIEW are used in full. Improved LabVIEW code generation tools simplify the development of simulation generation and greatly improve the LabVIEW code generation functionality.

2 Theory

For effective usage of FPGA advantage regarding parallel signal processing LabVIEW simulation code implementation on FPGA should avoid signal processing in loops anywhere where possible and transform them to calculations in parallel code blocks. Manual replication process takes a lot of time and may have an adverse impact on the final code accuracy. Therefore, it is effective to use one of the presented LabVIEW code generation methods.

A. LabVIEW code generation methods

LabVIEW provides two code generation methods:

a. Using prepared code in hardware description language (HDL) – VHDL or Verilog.

   Using this method the desired code functionality is defined in one of the HDL programming languages. This method is used when algorithm or application is prepared in HDL language, thus, no need to rewrite it in LabVIEW graphical code. It does not solve a problem how to correctly transfer LabVIEW code to VHDL. Traditionally third party proprietary HDL code generation tools are used. In this case LabVIEW code has to be transferred to third party environment manually taking into account environment specific code building principles.

b. Scripting

   LabVIEW provides specific scripting functions for generation of LabVIEW code – virtual instrument (VI) in the same LabVIEW environment. Scripting functions allow generating new VIs, front panel controls, block diagram objects and connections, as well as modify an existing VI (WEB (a)).

Comparing these two code generation methods it is concluded that, even though VHDL Intellectual Property (IP) offers similar functionality for code generation with various configurations, scripting is more effective. Since scripting VI generates a new VI

with highly parallel executable programme code in LabVIEW environment, it can be easily debugged and modified.

B. LabVIEW code generation using scripting functions

LabVIEW VI scripting helps programmatically generate, edit, and inspect LabVIEW code. First, VI scripting allows dynamically change the dimension of generated code by replicating once defined generation code. Secondly, VI scripting can adjust an existing simulation code working in simulation environment by automatically replacing specific functions to fit the planned hardware restrictions.

Giving the first impression about LabVIEW code generation using scripting a simple code generator is presented in Fig. 1. The generated LabVIEW code (Fig. 2) consists of While Loop and Add function. Comparing both codes it is obvious that the code generator has a bigger dimension than the generated code.

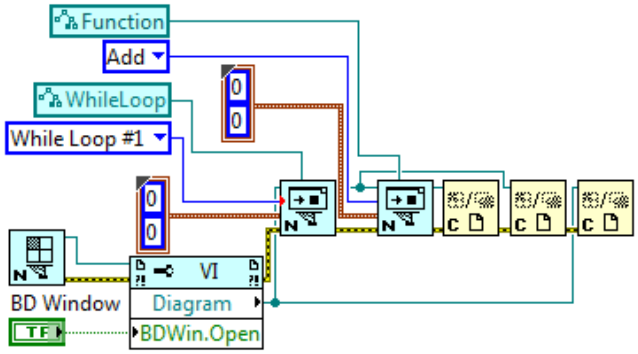

Fig. 1. LabVIEW code generator.

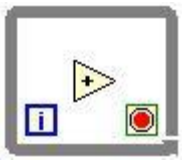

Fig. 2. Generated LabVIEW code.

When generating LabVIEW code with scripting functions, it is possible to generate all standard LabVIEW functions and complex signal processing algorithms. Since the code generator gets too complex even for a simple code generation, there are more drawbacks than benefits on using this code generation method. Thus improved LabVIEW code generation tools are necessary to make the application of scripting functions maximally effective.

3 Scripting toolbox

For full and convenient application of LabVIEW code generation functions a new LabVIEW function toolbox named Scripting is created. The Scripting toolbox (Fig. 3) includes functions which are similar to the standard LabVIEW functions or in other words to the function it generates and also some adjustment functions necessary for the realization of code generation.

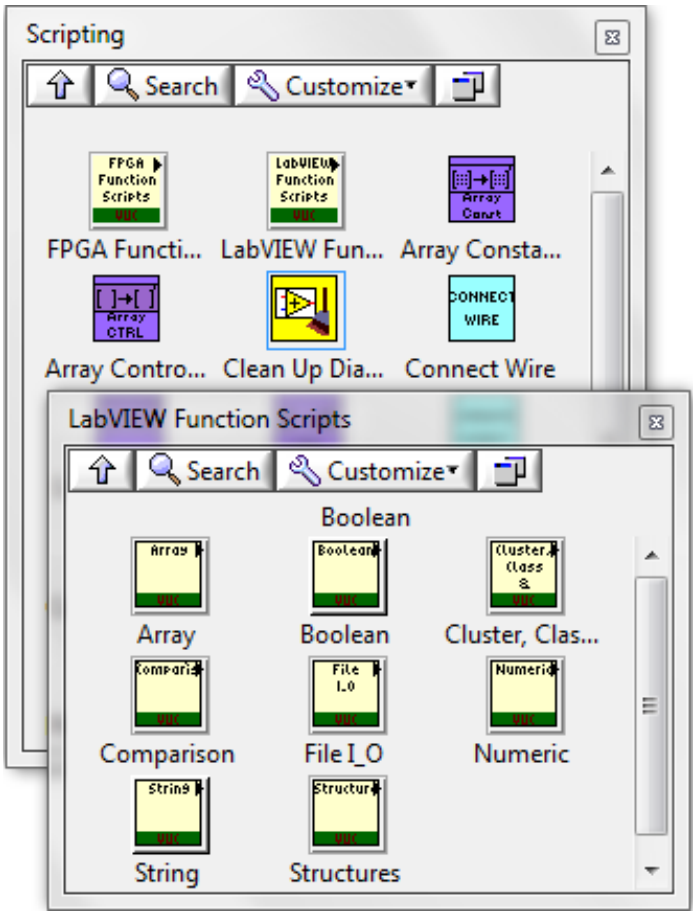

Fig. 3. LabVIEW Scripting toolbox.

In this toolbox LabVIEW function generation scripts for the most commonly used functions are developed. It is possible to develop function generation scripts for all LabVIEW function generation and use them for a general application the same way as standard LabVIEW functions. As an extra, this Scripting toolbox also includes FPGA High Throughput Math script functions for the basic FPGA math function generation.

In terms of functionality and visual representation the Scripting toolbox functions similar to the original LabVIEW functions are placed in the same directories and hierarchy levels in Function Palette as the standard LabVIEW functions. This kind of layout organization eases application of Scripting toolbox.

The Scripting toolbox function application is the same as for any standard LabVIEW function – choose function from the Function Palette and drag it into block diagram. If more information is needed, in LabVIEW Help window a small description of every Scripting toolbox function can be found.

Fig. 4 presents an icon of standard LabVIEW Add function (a) and Add generation script function (b). The developed Scripting toolbox functions all have a square form, but the visual representation is specific to the functionality, e.g. Add generation script function design is made using the same design as the standard LabVIEW Add function.

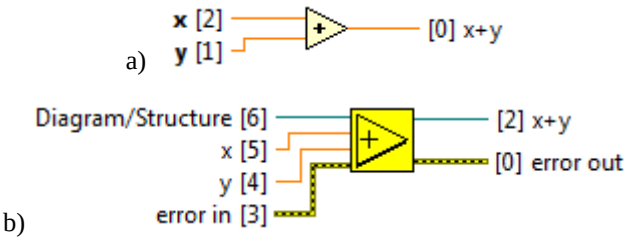

Fig. 4. LabVIEW Add function icons: a) Original LabVIEW Add function; b) LabVIEW Add generation script function.

All Scripting toolbox functions have the same terminals (with the same name) as the representing standard functions and also every scripting function has extra default terminals:

- *Diagram/Structure* (input);
- *error in* (input);
- *error out* (output).

LabVIEW standard functions dynamically adjust to the given input data type; to realize this dynamic dispatching in Scripting toolbox functions an object oriented programming technique – polymorphism is used (Rick, 2001). Polymorphism is the ability of VIs and functions to automatically adapt to accepting input data of different data types (Smith, 2000).

4 Evaluation of scripting toolbox

A classical algorithm and a good example for code generation in signal processing is Fast Fourier transform (FFT) (Smith, 2000). To demonstrate this, Fig. 5 shows one 2 - input butterfly code of LabVIEW. Butterfly code is a computation element that combines the results of smaller Discrete Fourier transforms (DFTs) into a larger DFT, or vice versa.

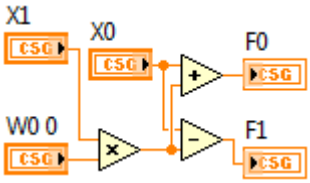

Fig. 5. One 2 - input butterfly code.

Two types of code generation tools are evaluated below.

A. Standard LabVIEW code generation functions

The preview of 1 part out of 6 of LabVIEW butterfly code generator with standard scripting functions (Fig. 6) shows that it is more complex than the generated LabVIEW butterfly code (Fig. 5).

It can be seen that even in a simple code generation case it is easier to develop a code manually not using the standard scripting functions. Even though code generation using scripting should bring benefits and make the code generation more efficient, the standard LabVIEW scripting functions are very complex even for a simple code generation.

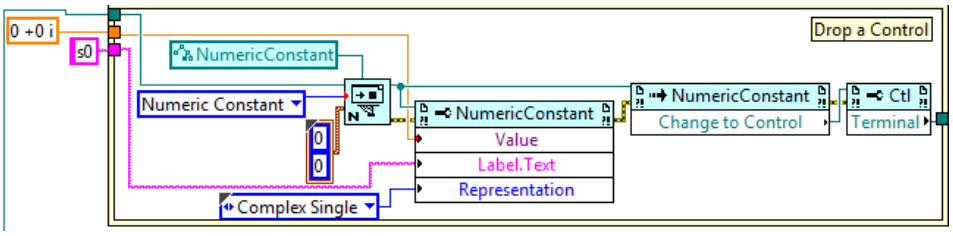

Fig. 6. One 2 - input butterfly code generator with standard LabVIEW scripting functions (1 part out of 6).

B. The developed Scripting toolbox functions

The improved LabVIEW code generation functions from Scripting toolbox are easier to use and the created butterfly code generator (Fig. 7) is clearer, visually less complex and more similar to the standard LabVIEW butterfly code (Fig. 5). Since the code generator with improved functions visually looks similar to the generated code, it is easier to understand and follow the realized algorithm.

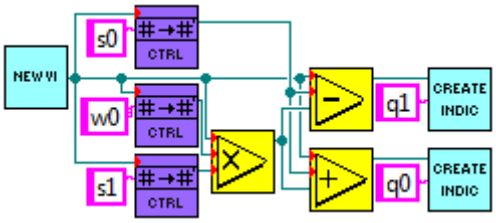

Fig. 7. One 2 - input butterfly code generator with improved scripting functions.

A simple butterfly code example with 2^1 - inputs does not show the advantages of scripting. When it comes to butterfly code with 2^{10} - inputs, the generation code development becomes very time consuming and is subject to an extreme accuracy by the developer due to its dimension.

Using Scripting toolbox a butterfly code with 2n - inputs can be automatically generated by setting only one variable in the LabVIEW front panel – *n*. Variable *n* defines the size of butterfly code. This butterfly code generator can generate butterfly code of any dimension. The only limit is the local computer processor power.

Modifying the butterfly code generator, the new code generator (Fig. 8) allows dynamically generate butterfly code of any size. In case of $n = 2$, the generated butterfly code looks like the one presented in Fig. 9.

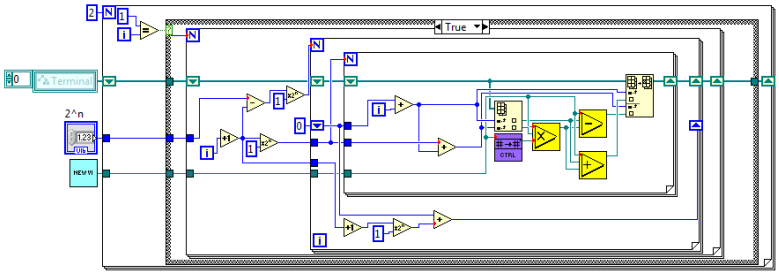

Fig. 8. Dynamical butterfly code generator with improved scripting functions.

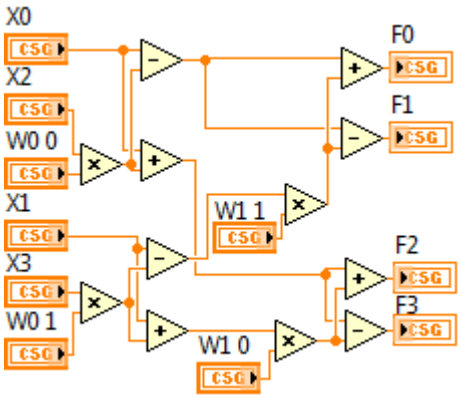

Fig. 9. Two 2 - input butterfly code.

In case of $n = 2$, the dimension of butterfly code generator and the generated code is almost the same. Since the application of butterfly code in parallel signal processing algorithms includes $n >> 2$, the generated butterfly code is more complex and with greater dimension than the code generator. The butterfly code dimension increases very rapidly, hence, screenshots of bigger generated codes are not presented in a figure here.

Advantages of Scripting toolbox

The butterfly code dimension depends on the number of inputs (2^n). The bigger the value of n, the greater is the generated code. Fig. 10 presents comparison of generated butterfly code dimensions by defining different values of n in the same butterfly code generator (Fig. 8). The dimensions were compared by analysing LabVIEW block diagrams of all generated butterfly codes.

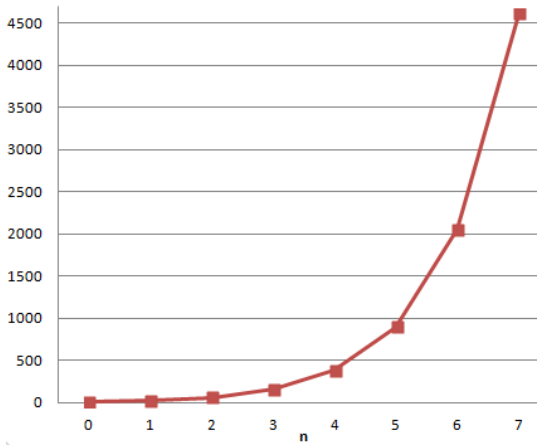

Fig. 10. Number of objects in generated butterfly code.

It can be concluded from Fig. 10 that for $n > 2$ the number of objects in the generated butterfly code increases very rapidly. For $n > 7$ the dimension of generated code increases tremendously and it is not in human power to repeat this manually.

In the same way there is no limit for generation of any other simulation or hardware implementation code. The application of the developed Scripting toolbox functions is as simple as the standard LabVIEW functions.

5 Conclusions

When it comes to development of complex signal processing algorithm simulation and its implementation into hardware, the most advantageous choice is LabVIEW code generation using scripting. In other words, LabVIEW code generation using scripting is advantageous when repetitive actions have to be done, e.g. to avoid loops and efficiently use the FPGA advantage – parallel signal processing.

Properly developed code generator allows quickly generating signal processing path of any dimension preventing human errors. Once a code generator prototype is developed, there is no extra effort needed to generate unlimited size or number of this kind of codes.

The provided standard code generation functions in LabVIEW are too complex to effectively use them in simulation code generation. The improved LabVIEW code generation functions from developed Scripting toolbox provide easy application and simulation code generation.

The Scripting toolbox provides improved functionality for the development of simulation code generators since it is easy usable, understandable and also the user developed code generator look similar to the standard LabVIEW code of the same particular signal processing algorithm.

Acknowledgement

This work has been partially supported by the ESF project Nr. 2013/0005/1DP/1.1.1.2.0/13/APIA/VIAA/049 and by the Latvian National Research Program „Cyberphysical systems, ontologies and biophotonics for safe & smart city and society.” within the project „Ontology-based knowledge engineering technologies suitable for web environment” and by the ERDF project Nr. 2DP/2.1.1.1.0/14/APIA/VIAA/072.

References

Rick, B. (2001). *Object-Oriented Programming in LabVIEW*, LabVIEW Advanced Programming Techniques Boca Raton: CRC Press LLC.

Smith, S. W. (2000). *The Scientist & Engineer's Guide to Digital Signal Processing*, California Technical Publishing.

WEB (a). Scripting language definition. http://www.webopedia.com/TERM/S/scripting_language.html [Accessed: Sept 20, 2015];

WEB (b). Polymorphism definition. http://zone.ni.com/reference/en-XX/help/371361M-01/lvconcepts/polymorphic_functions/.

WEB (c). LabVIEW popularity among IEEE publication topics. http://ieeexplore.ieee.org/search/searchresult.jsp?newsearch=true&queryText=LabVIEW.

Authors' information

Evita Vavilina received the B.S. and M.S. in electronics from Ventspils University College (VUC), Ventspils, Latvia in 2012 and 2014 respectively. She works as researcher in Engineering Research Institute "Ventspils International Radio Astronomy Centre" (VIRAC) of VUC. Her research interests include signal processing algorithm realization in graphical programming environment and development of automatic code generation tools.

Gatis Gaigals received the B.S. in electronics from Riga Technical University, Riga, Latvia in 2004 and M.S. in computer sciences from VUC, Ventspils, Latvia in 2008. He is currently pursuing the Ph.D. degree in electronics at Riga Technical University, Riga, Latvia. His current research interests include applications of compressive sampling technique. He works as researcher in VIRAC and as lecturer in VUC since 2007 and 2008 respectively.

Received March 14, 2016, accepted March 23, 2016