Multidiszciplináris tudományok, 11. kötet. (2021) 5 sz. pp. 264-268 **https://doi.org/10.35925/j.multi.2021.5.28**

VISUAL PROGRAMMING ENVIRONMENTS FOR DIGITAL SIGNAL PROCESSORS

Attila Károly Varga
associate professor, University of Miskolc, Institute of Automation and Infocommunication
3515 Miskolc, Miskolc-Egyetemváros, e-mail: varga.attila@uni-miskolc.hu

Abstract
The first step in implementing digital signal processor (DSP) applications is to test the functional operation of the algorithm. Once we are convinced that it fully fulfills the required task, the second step, implementation, can follow. In the vast majority of cases, this is done in a high-level C / C ++ language. We are not yet optimizing at this stage of development. If the code works completely flawlessly and runs at the right speed, no further refinement is needed, we're done with the development. We have several tools at our disposal for developing programs running on DSP. This software package is based on the C / C ++ language, with numerous pre-written routing collections and optimization options to support development work. However, there are development tools that can be used more easily and most quickly. Some of these tools will be discussed in this publication.

Keywords: *digital signal processor, programming environment, DSP development, application builder*

1. DSP development using the Matlab software package

Matlab is a well-known software package that tries to cover a very wide range. Initially based on simulations based on mathematical calculations, the program has already claimed a place in the development systems market. Although laboratory development and modeling is still the main profile, numerous additions have been made to the basic package since its inception. The Simulink is entirely for visual (block) programming. From the beginning, countless toolboxes (plus toolbars) have appeared and new versions have already included hardware support. This following will actually show how Simulink supports DSP development. (Kuo and Gan, 2005)

The Simulink is the graphical part of Matlab, ie we do not have to describe program lines, but arrange blocks in a workspace with connections, then after setting the parameters we run the simulation and evaluate the result.

To create a new model, you must first start Simulink. This is possible using the simulink statement typed in the Matlab command window. The simulink window opens, where you can browse through the different block collections. In the work area, the selected blocks should be placed by unraveling them in the tree structure, then grabbing and dragging them onto the board. Once all the selected items are on the tab and we have arranged them accordingly, we need to connect the blocks. The procedure is as follows: place the cursor on the output of a block and press the left mouse button. While holding down the button, move the mouse to the input you want to connect to the output, and then release the button. After the connections have been established, all that is left is to set the simulation parameters, and then to parameterize the blocks placed on the sheet (see Fig. 1). Parameterizing the blocks is quite simple, all you have to do is double-click on the given block and fill in the dialog box that appears. However, you should make sure that the simulation settings and the block settings are consistent. That is, if we use

(and mostly use) a signal generator, do not set a simulation step that does not comply with the sampling law, or we will not get what we want.

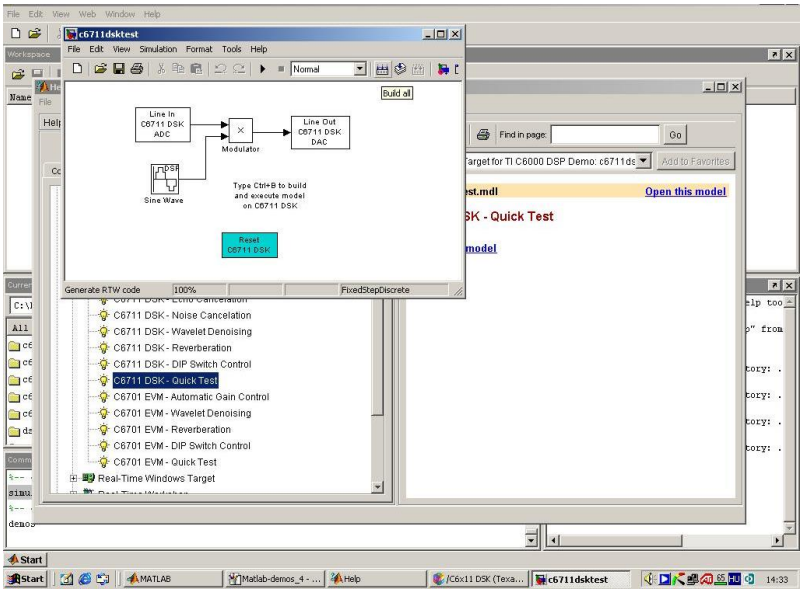

Figure 1. *Compiling a MATLAB Simulink program*

There are also a number of pre-built sample programs in Matlab's Help to make programming easier to learn. The interface between Matlab and Code Composer Studio (CCS) also allows you to run the model as a model instead of downloading it to the DSP and running it there as a program. Of course, this also requires that the CCS be installed on the computer and that the DSP be connected (type matching is absolutely necessary). (Elrajoubi et al., 2017)

2. DSP development using VisSim

VisSim is a name formed from the words Visual Simulation. This name is a development environment created and marketed by Visual Solutions Inc. VisSim is a development and simulation program that provides a graphical environment. All the tools needed for model building are available under the different menu items.

As its name suggests, it was mainly developed for simulating algorithms, but the development tool also has code generation capabilities. In the VisSim development system, we can develop programs for the DSPs. One of the unique capabilities of VisSim is that it can be used to develop programs that require very little memory. For example, a closed-loop PID controller with an encoder input and a digital as well as a PWM output uses only 1.3K ROM and 155 bytes of RAM on an MSP430. Another advantage of VisSim is that it has an interface to both Matlab / Simulink and CCS (code-composer-studio.software…), as well as a C code generator.

The graphical interface (see Fig. 2) of the program is completely similar in its basic features to other programs, VisSim also uses menus and dialogs to communicate between the user and the program, and a scroll bar to facilitate navigation. The user interface includes the menu bar, toolbar, status bar, workspace, and an auxiliary window.

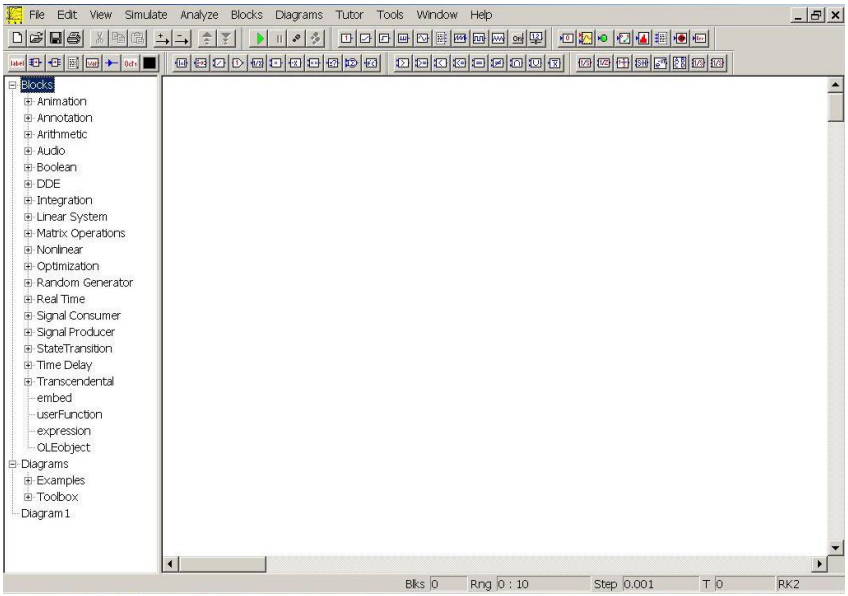

Figure 2. *VisSim user interface*

Basic blocks are basic instructions, blocks organized into different topics essentially represent pre-written routines, while user routines correspond to compressed blocks. The sub-simulation assembled from the blocks may be repeated, or it may be necessary to lock it in a black box for transparency. You can do this by selecting the items you want to merge and choosing Edit — Create Compound Block.

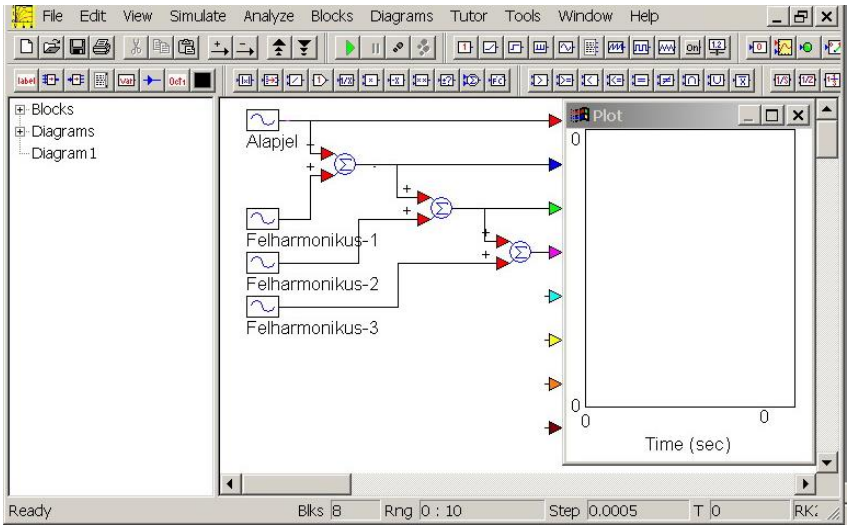

Figure 3. *Completed simulation in VisSim*

Then you need to enter the name of the new complex block, set security functions, and add a bitmap image to the new block.You can view the contents of compressed blocks by double-clicking on the block. However, if you want to extract a compressed block, the Edit — Dissolve Compound Block menu

item will help you. (Dahnoun, 2000; TMS320C6000 Code Composer Studio Tutorial, 2000; VisSim User's Guide 8.0, 2010)

3. The VAB development environment

Visual Application Builder (VAB) is a visual development environment created by Hyperception that allows us to quickly develop programs for Texas Instruments DSP. We use the workspace to organize each block into a system, thus creating our program. The workspace is bounded by the menu bar at the top and the status bar by the bottom. When you start VAB, you automatically place a blank worksheet on the workspace. The worksheet is the area where the program is created by connecting the blocks.

The VAB menu commands are located at the top of the window and contain the menu items and their submenu items in the form of a drop-down menu. Of course, these functions can also be placed on a toolbar tray in the form of a button, or some of them are there automatically. The toolbar is located below the menu command line, but can also be moved to another point in the workspace.

Of course, we also have the option to customize the menu if we wish. All you have to do is right-click anywhere in the workspace. In response, VAB pops up the VAB Customize window. Here you can select which menu commands you want to place in the shortcut menu that you click on the right mouse button, and you can also separate them with a separator (SEPARATOR keyword).

One of the most important things when programming is to always keep in mind the memory size of a given DSK (which is limited). This memory is what our program will use at runtime. To make the most of this memory, we must first understand how VAB handles the available memory. Just be careful about the amount of memory on your hardware, as this is the area of the DSP's real-time operating system core during DSP operation, reducing the amount of free memory. This allows data transfer to and from the DSP. Each DSP-based block we place on our worksheet requires a certain amount of memory. Blocks are essentially subroutines written on a DSP assembler that have variables and parameters that, of course, take up space in memory at run time (e.g., a sine function generator needs a measure of amplitude and frequency). Some blocks require memory space (DSP to PC Buffer) for data storage. These memory requirements, of course, add up.

Native blocks are designed to run as fast as possible and require very little free memory. These two properties of native blocks make them ideal for use. This, in turn, means that we have to pay attention to their input data. If a native block receives input data with more than one frame size, only the very first frame is processed, while the other data is lost. If you use a mixture of vector-based and native blocks, be sure to adjust the frame size of the vector-based blocks more and more. (Chassing, 2004)

4. Summary

Digital signal processor circuits include the tools needed for emulation, which allows development tools to monitor program execution and monitor program activity in real time. The Integrated Developer Environment (IDE) allows you to edit, modify, and track user programs. One of the best development environments for DSP development is CCS, an integrated development environment (IDE) that supports microcontrollers and embedded processors, as well as a toolkit used to develop and debug embedded applications. It supports all phases of application development: source code writing, debugging, and real-time tracing. It includes code generation tools and real-time analysis

capabilities. In this publication, however, I did not want to focus on CCS, but presented 3 visual development environments that are easy to master and not far below the effectiveness of CCS in terms of the efficiency of the code they generate.

New versions of Simulink already include a number of great features. These include, on the one hand, the existence of hardware-specific blocks and, on the other hand, the interface to CCS.

In VisSim, as in other development environments, there are always basic instructions and pre-written routines (functions, procedures), and user routines can be defined. The model can be built from blocks, connections can be defined, and then the finished system can be simulated and analyzed using the program.

VAB contains a number of very effective block components called native blocks. PC-based blocks entrust the processing of the algorithm to the PC. We mostly use PC blocks to display data. These blocks cannot be connected directly to DSP-based native and vector blocks, although VAB provides the ability to upload data and process and display it on a PC. Vector blocks can be set as a frame size parameter and can work with more than one frame size, unlike native blocks.

5. Acknowledgements

The research work was carried out as part of the EFOP-3.6.1-16-00011 “Younger and Renewing University – Innovative Knowledge City – institutional development of the University of Miskolc aiming at intelligent specialisation” project implemented in the framework of the Szechenyi 2020 program. The realization of this project is supported by the European Union, co-financed by the European Social Fund.

References

[1] Kuo, S., M., Gan, W. S. (2005). *Digital Signal Processors*. Upper SaddleRiver, NJ: Prentice-Hall.

[2] Elrajoubi, A., Ang, S. S., Abushaiba, A.: *TMS320F28335 DSP programming using MATLAB Simulink embedded coder: Techniques and advancements*, 2017 IEEE 18th Workshop on Control and Modeling for Power Electronics (COMPEL), pp. 1-7. **https://doi.org/10.1109/COMPEL.2017.8013418**

[3] https://code-composer-studio.software.informer.com

[4] Dahnoun, N. (2000). *Digital Signal Processing implementation using the TMS320C6000™ DSP Platform*. Pentice Hall 2000. ISBN 0201-61916-4

[5] TMS320C6000 Code Composer Studio Tutorial; Literature Number: SPRU301C, February 2000.

[6] Visual Solutions, Inc.: VisSim User's Guide 8.0, Westford, 2010.

[7] Chassing, R. (2004). *Digital Signal Processing and Applications with the C6713 and C6416 DSK*. Wiley-Interscience, ISBN 978-0471690078 **https://doi.org/10.1002/0471704075**