Proceedings of the SDR 11 Technical Conference and Product Exposition,

A GRAPHICAL INTERFACE USING FIRMWARE MACRO-CELLS FOR FAST FPGA DESIGN REVISION IN SOFTWARE-DEFINED COMMUNICATIONS APPLICATIONS

Frank Raffaeli
Principal RF Engineer, National Instruments, Austin, Texas, USA; frank.raffaeli@ni.com

ABSTRACT

Achieving fast-turnaround on revision cycles is important both for research and for deployment SDR platforms. Modern large FPGAs often have prohibitively long compile times, extending for several hours. Often, the learning cycle in academic applications is curtailed by the constraint of the compile operation.

Using pre-compiled DSP macro-cells within the LabView FPGA module, a hierarchical design can be deployed from the simulation phase in a matter of seconds instead of hours. The macro-cell is a generic DSP engine that utilizes both a combinatorial and a sequential framework to combine the DSP blocks within the FPGA's infrastructure to build a re-configurable function. The macro-cells are physically configured using a combination of mux registers and scripted commands. From the user's point of view, a [library] function(s) can be written and simulated in the graphical language LabView™, then deployed to the macro-cells at run-time within a few seconds.

1. OVERVIEW

The premise of this paper assumes the design revision cycle time is largely due to the compile time of the FPGA. This method describes a process by which the FPGA can be loaded with a generic bit-file where the real-time signal processing is still with the FPGA but the functionality can be configured at run-time. The generic bit-file is built using the LabView FPGA module, and its representation is canonical and hierarchical within the LV FPGA environment. There is a graphical user interface via the LabView software, which appears as a block diagram of the communications system within the LabView interface. The user can utilize the LabView environment as a simulation engine, with all the measurement capabilities of the tool, including Frequency and Time domain analysis.

Each top-level block's functionality is configured by the user at run-time. The block diagram can be re-configured without re-compilation, provided the functionality fits within the constraints of the generic macro-cell. For full custom functionality, a complete FPGA re-compile is required.

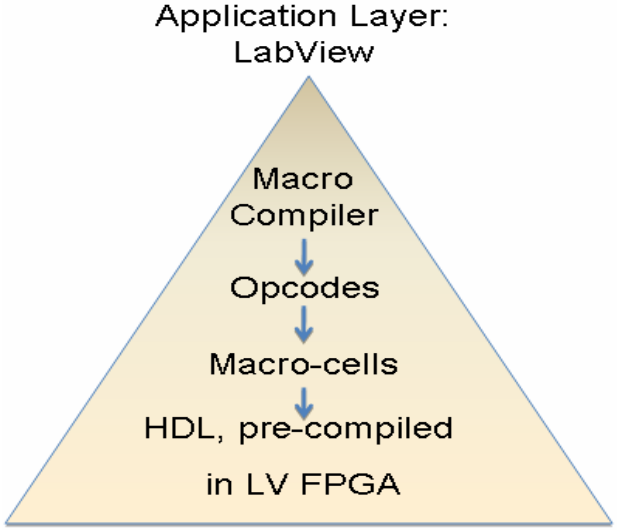

Figure 1 – structure of the macro-cell user environment

2. DESIGNING WITH MACRO-CELLS

Each macro-cell represents a block in the communication system diagram. Each functional block will require "software". In generic terms, the sequential code that runs the script [dynamic re-configuration] within the macro-cell, and the register settings that describe the signal routing and static settings are considered the "Op-codes" referenced in figure 1, above.

From the LabView interface, the user selects a function from the library, or builds a new one with the macro compiler and then loads each block with the macro-cell's script. An FPGA compile is not required since the macro compiler is only changing the op-code and register memory embedded within the macro-cell (see figure 2).

3. PRACTICAL CONSIDERATIONS

The advantage to this approach is certainly speed of design revision and short time to first implementation; however, this approach will only work if the required function fits within a macro-cell or it can be described by a series of canonical macro-cells, such as a filter. In order to examine some of the practical uses and constraints of this technique, this paper will examine two cases as follows:

- Filter design using linear-phase recursive structures
- Special functions: Oscillator.

Because a function implemented with a macro-cell must be developed using a compiler, it is handy to have a library of macro-cell functions already implemented to use as examples. To facilitate an open source environment for this work, National Instruments dedicates a web area called NI-labs [1].

4. ANATOMY OF A MACROCELL

In order to examine the pros and cons of this architecture, it may be logical to view the structure from the bottom-up.

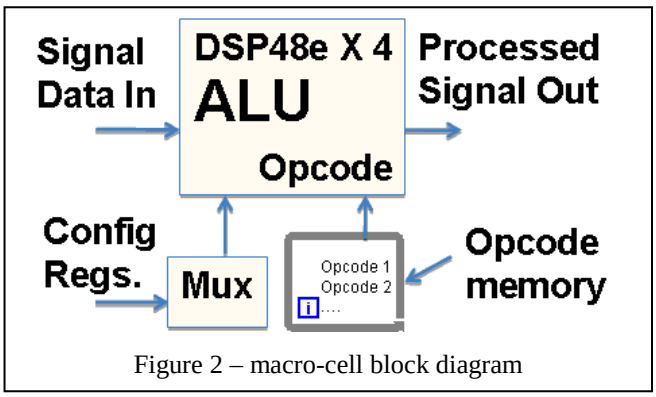

Figure 2 – macro-cell block diagram

The macro-cell consists of a main ALU, which is composed of four cross-connected DSP 48e blocks. The alumode registers of the DSP48e (see figure 4) and also several of the configuration registers are controlled via an op-code loop memory and pre-fetch sequencer that runs the sequential programming synchronously with the system clock. The mux is controlled by a static register, and although it is also configurable at run-time like the op-code registers, it does not sequence with the input clocking. The DSP48e is the main signal processing engine, and is configured such that its op-mode register can be reloaded on each clock cycle.

The sequencer has two functions: It synchronizes all four DSP48e's clocking such that the op-codes are cycle accurate, and it serves as the pre-fetch and memory controller for the op-code cycle execution.

5. OPCODE DETAIL

The following shows the basic anatomy of the 16-bit op-code. An illustration of a static configuration of the DSP48e can be found in figure 4:

B15 – enable
B [14.. 12] – slice ID
B [11.. 04] – ALU configuration
B [03.. 02] sequencing ID
B [01.. 00] dynamic mux

6. EXAMPLE IMPLEMENTATIONS

The examples chosen were implemented within the macro-cell architecture, and share a common theme: they either execute in a single clock cycle using four DSP48e blocks, or they utilize a sequencing algorithm in order to fit within four DSP48e blocks. Canonical structures are not explored.

6.1. Linear Phase Recursive Filter

If the phrase "linear phase" seems orthogonal to "recursive" it is because only a specific type of recursive filter achieves linear phase. Generically, it is referred to as a comb-resonator structure [2]. A specific case, where the resonator is a single integrator, is well-known as a CIC filter.
A macro-cell can be used to implement the generic case, if it is used in conjunction with the local memory in the macro-cell.

6.2. Oscillator

The oscillator is the simplest implementation in the macro-cell structure because it does not have a dynamic sequencing requirement. The simple oscillator topology shown in the figure below can be easily translated to a LabView implementation and simulated in a test bench.

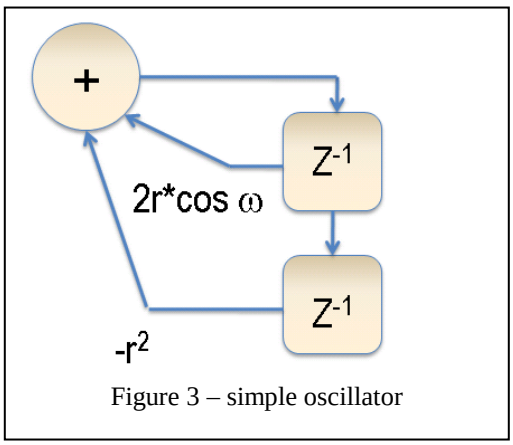

Figure 3 – simple oscillator

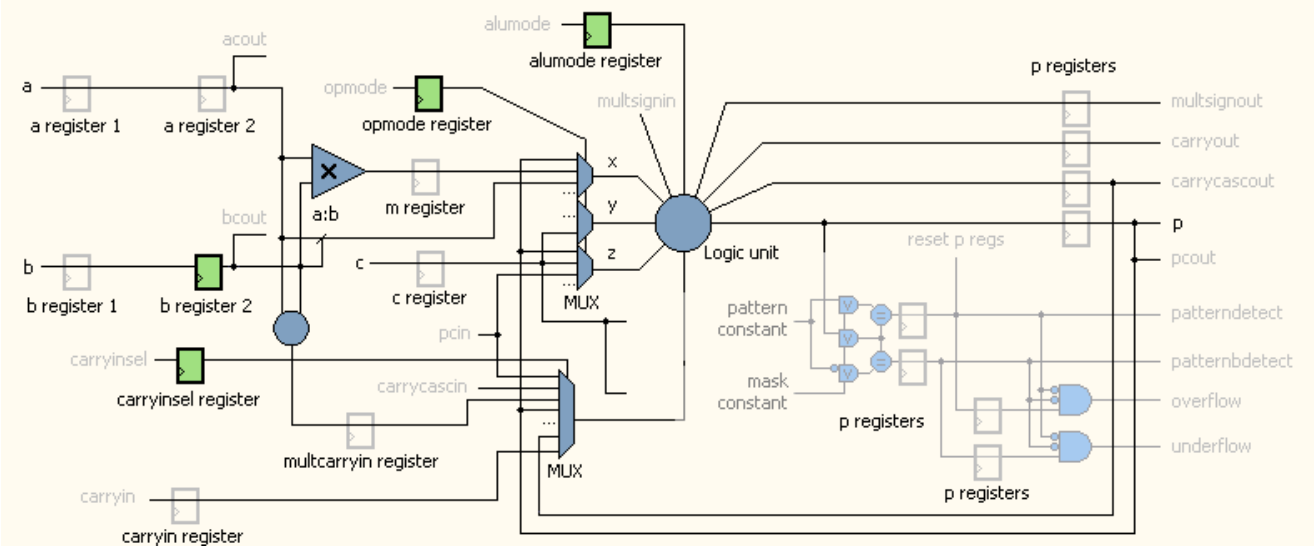

Figure 4 – detail of the DSP48e slice as configured by LabView FPGA. Highlighted blocks show enabled resources

The oscillator has two coefficients, -1 (a constant), and 2*r cos (ω). Since r is the radius of the unit circle, the only variable coefficient is the frequency scaled by a cosine function. This means it can be implemented with a simple multiply-add function. In the illustration above (figure 4), the shaded blocks show a rendering of how LabView FPGA represents the DSP48e block through the user interface. In the macro-cell implementation, the alumode register needs only to be configured by a static value, whereas in a sequential implementation such as a comb-resonator filter, the alumode register is configured by the sequencer.

The diagram in figure 5 illustrates the test bench used to build the oscillator, which is a bit more complicated, since it requires an AGC and injection-locking between the two orthogonal sections to achieve a quadrature relationship between the two sections.

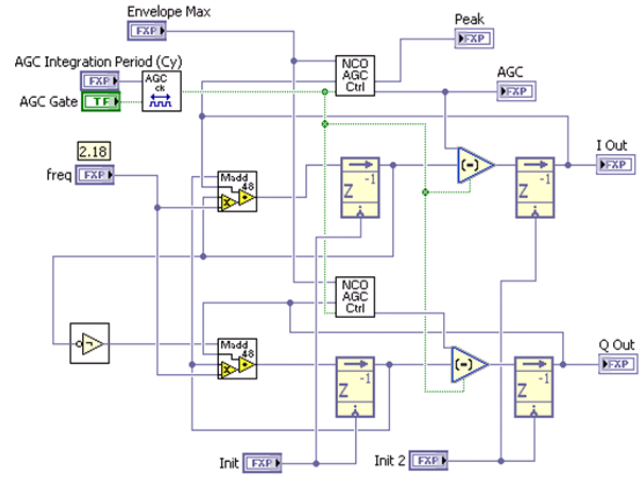

Figure 5 – oscillator test bench in LabView

7. SUMMARY

As FPGA designs become more and more complex, the need grows for techniques which can speed design iteration times without placing unnecessary constraints on the user. LabView and LabView FPGA can be used to build a design structure using generic macro-cells that can be configured at run-time. This technique can be useful if the design fits within a canonical structure or the user can make modifications to use a sequential architecture.

8. REFERENCES

[1] NI-Labs – http://www.ni.com/labs - keyword search: SDR.
[2] R. Kuc - Introduction to Digital Signal Processing pp 220-229 ISBN 0-07-035570-3.