Motivation for graphical development on DSPs

Joe Coombs, Applications Engineer, Texas Instruments

Digital signal processors (DSPs) have become ever-present in today's world, populating everything from stereo systems and cell phones to automobiles and industrial equipment. Despite this proliferation, relatively few embedded developers are DSP-focused. This apparent contradiction is easily explained: it's simply not possible to be a "DSP developer" in the same sense that one can be an "ARM developer." Each DSP device exists in its own world of registers and peripherals — low-level architectural details supported by proprietary drivers, application programming interfaces (APIs) and extensive technical documentation. The question is not whether developers can understand the architecture and tools, but how long it takes to learn how to use them. Even veteran embedded developers occasionally grapple with platform-specific details before they can bring their considerable expertise to bear on a new project.

This problem can be alleviated by graphical development tools. When developers can utilize an intuitive visual aid to abstract device-specific APIs and hardware details, it is possible to quickly design entire applications without getting bogged down in low-level technical complexities. Work in the graphical domain easily translates to multiple embedded targets with minimal changes, and the repetitious process of algorithm development is greatly accelerated.

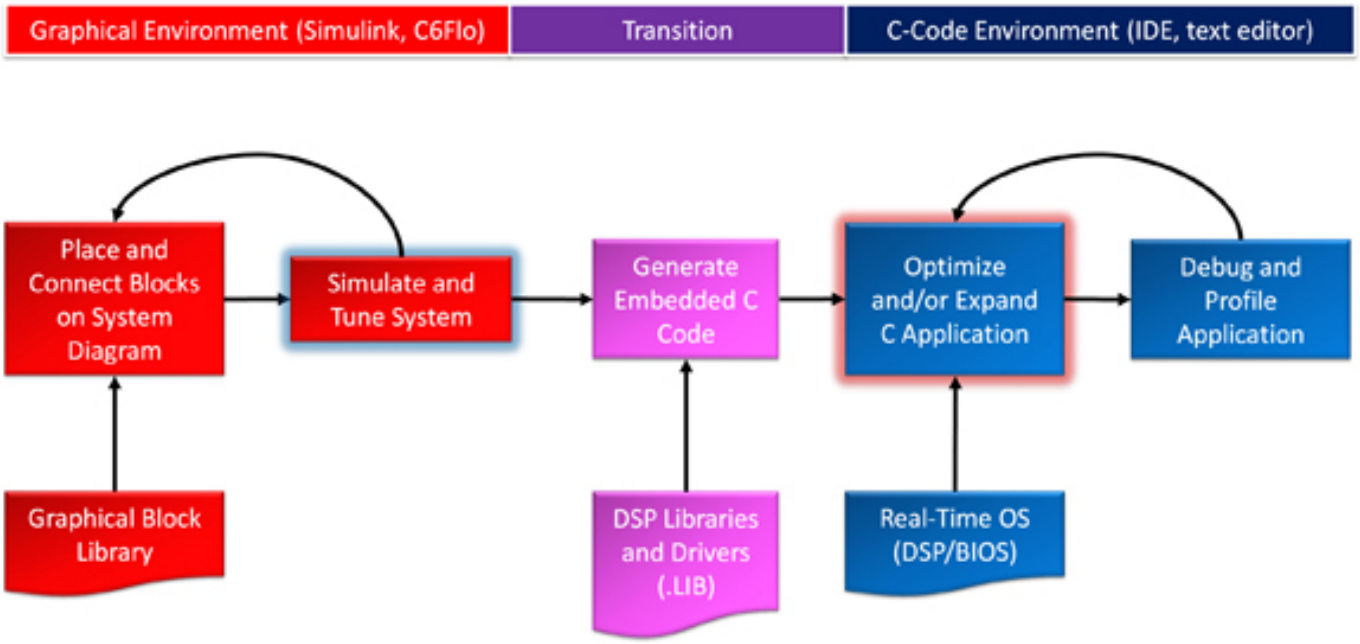

Figure 1: Development flow chart beginning in a graphical environment. Different tools (i.e. Simulink from The MathWorks, Inc and C6Flo from Texas Instruments) emphasize different steps in the process.

This high-level approach to DSP development can even be valuable to expert developers if the graphical tool produces code that meets his or her needs. Graphical development tools are sometimes viewed as props for inexperienced developers rather than important parts of a veteran's toolbox. This perception is partially validated if it's difficult to move between the graphical environment and the more traditional domain of text editors and integrated development environments (IDEs) over the course of a project. Graphically generated code must be cleanly structured, well-commented and reasonably efficient to be useable outside of the tool that spawned it. These conditions are listed in order of descending importance: algorithms can almost always be made more efficient, but the baseline code must be comprehensible before further work can begin. This key criterion allows the tool to be useful beyond the boundaries of its own Graphical User Interface (GUI) and throughout the broader development landscape.

One example of a graphical development tool that's designed with this goal in mind is C6Flo from Texas Instruments. Like other graphical development tools, C6Flo allows developers to drag and drop DSP algorithms and peripheral input/output (I/O) drivers onto a grid, then connect them together to create an intuitive block diagram that represents their DSP system. The output of the tool is a fully-formed DSP application written in plain C code that is well-suited for further development using conventional tools. The output must be good enough as-is for developers who want to stay in the graphical domain, but it must also be readable enough and malleable enough for developers who want to move beyond that niche. It can be used as a self-contained development environment, but more generally it serves to abstract low-level, device-specific details for all developers.

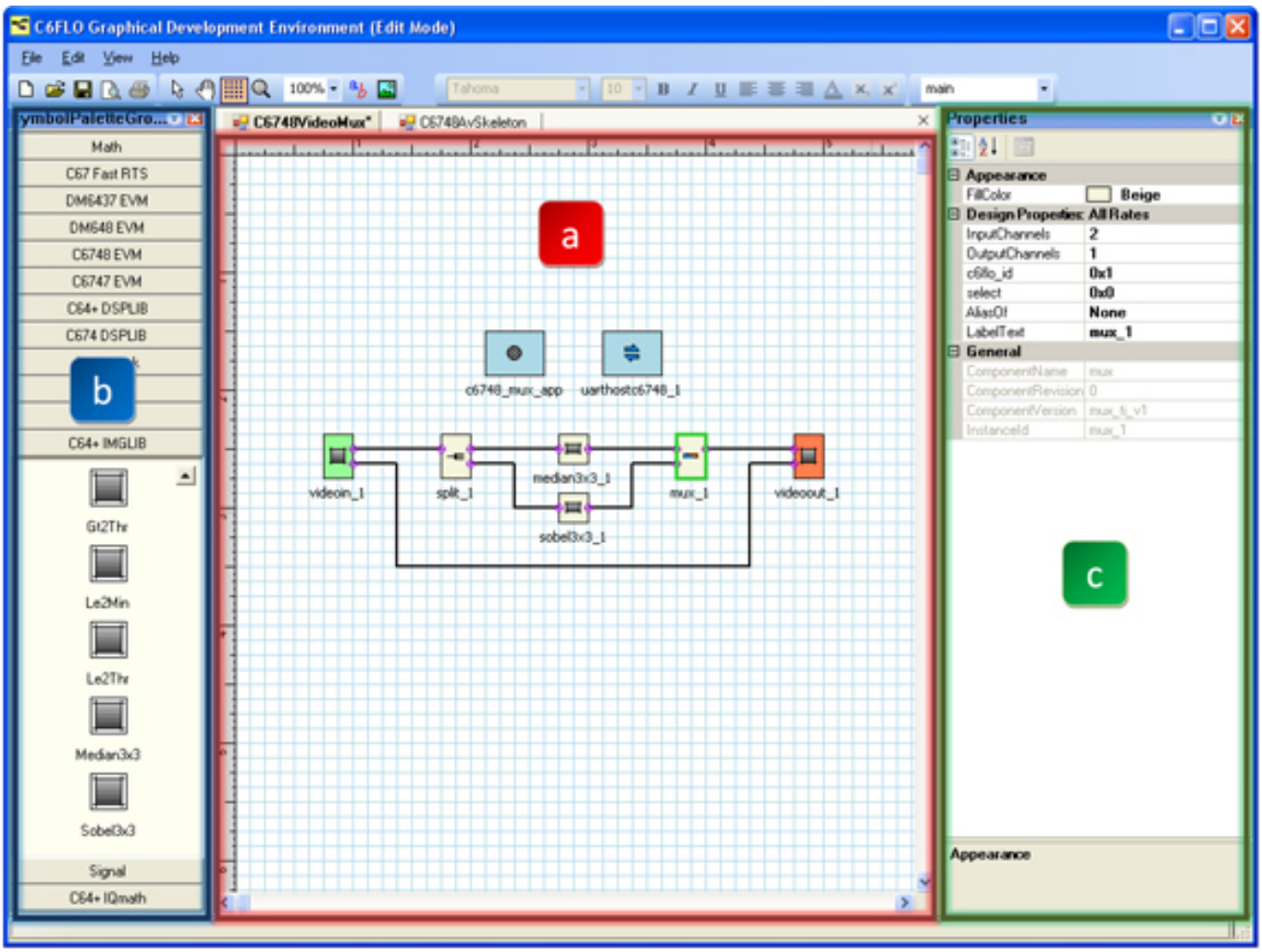

Figure 2: The C6Flo graphical environment with highlights (a) system block diagram, (b) categorized block palette, and (c) block properties pane (mux block selected)

This approach to graphical development allows DSP developers to move quickly from whiteboard sketch to working prototype without sacrificing the ability to tweak or expand their applications by hand to obtain the best possible performance.