White Paper

Using FPGAs to Render Graphics and Drive LCD Interfaces

This paper describes the use of FPGAs to add a LCD and GUI display to any embedded system. Unlike fixed processor device implementations, this approach is scalable and can support any display interface. Graphics can be generated by any external processor, an embedded processor, or a hardware graphics acceleration engine integrated into the same FPGA design as the display graphics controller. The benefits of FPGA implementation and available tools and IP are described, and links to reference designs and third-party solution providers are given.

Introduction

Graphical displays are increasingly popular for many new applications and industries due to the dramatic increase in the use of graphical user interfaces (GUIs) in mass applications such as PCs, phones, PDAs, and other consumer devices. Put simply, users expect a GUI with every application and see great value in the ease of use that this delivers. A second factor is the cost reduction that comes with high-volume manufacturing, with the small and medium display market expected to reach 3.3 billion units and more than $27 billion per year in 2009/10 *(1)*. These volumes have resulted in a significant drop in display prices and an increase in the quality, resolutions, and sizes commonly available on the market. As a result, many applications now support the addition of a high-resolution color display and GUI where five to ten years ago it would have just been too expensive to be accepted by the market.

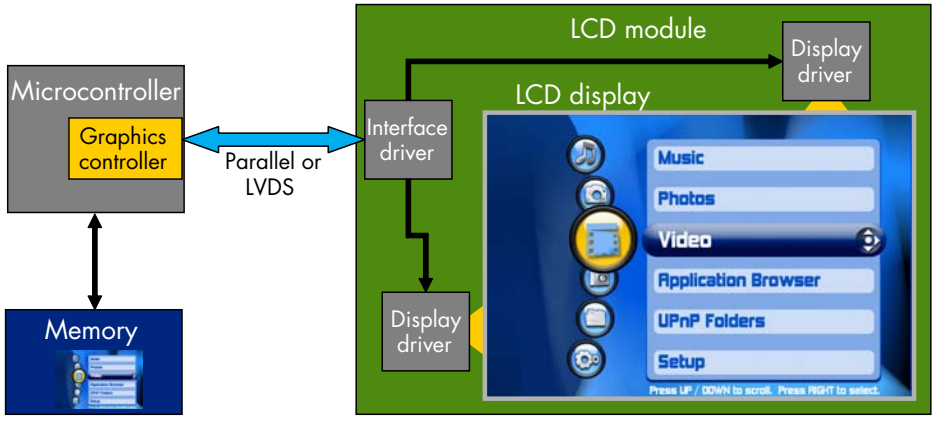
Display technology always has been a dynamic space, as every year LCD developers find new materials and ways to improve the resolution, contrast, viewing angle, and speed of their displays. In addition, there are advances in technology for touch screens, backlighting techniques, and novel displays (such as organic light-emitting diodes (OLEDs), electrophoretic (EPD) displays, and even electronic "paper").

The result of this is that display modules do not have long lifetimes—within a year or two, there will be a better, cheaper, more feature-rich display on the market and the obsolescence process will start for the older module. This may not be an issue for the consumer market space, but many other markets cannot afford to redesign their products so frequently. While the actual display may be still available, there can be problems with sourcing components as manufacturers upgrade to track the latest technology and volume requirements. Even in terms of marketing, the low resolution of an old display can be an issue-users are less likely to purchase a product that does not meet their constantly increasing expectations of a GUI.

Display Drivers and Controllers

There is a wide range of LCD drivers and controllers on the market today. One can generally distinguish between pure display drivers (without intelligence of their own), display controllers with integrated display memory (and limited/preset functionality), and general-purpose microcontrollers with integrated LCD controllers (Figure 1).

Figure 1. Typical Microcontroller-Based Display Architecture



Pure display drivers (e.g., MSM5219, UPD7225, HD44100, LC7942, etc.) generally have a high-frequency serial input and continuously require new display data in order to achieve the highest possible refresh frequency. In contrast, a display controller (e.g., HD61202/3, HD61830, SED1520, SED1330, T6963, etc.) is sent a series of graphical data and control codes, and then it manages the image (usually including fonts), internal memory, and display multiplexing on its own.

Most popular microcontroller families include one or more members that have an integrated display controller. These devices have special I/O pins to interface to the display and also may have an internal display memory to deliver maximum performance. To reduce the number of pins used and increase the number of displays supported, the microcontroller devices interface to the pure display-driver devices (usually built into a display module) using a parallel data interface or a LVDS interface.

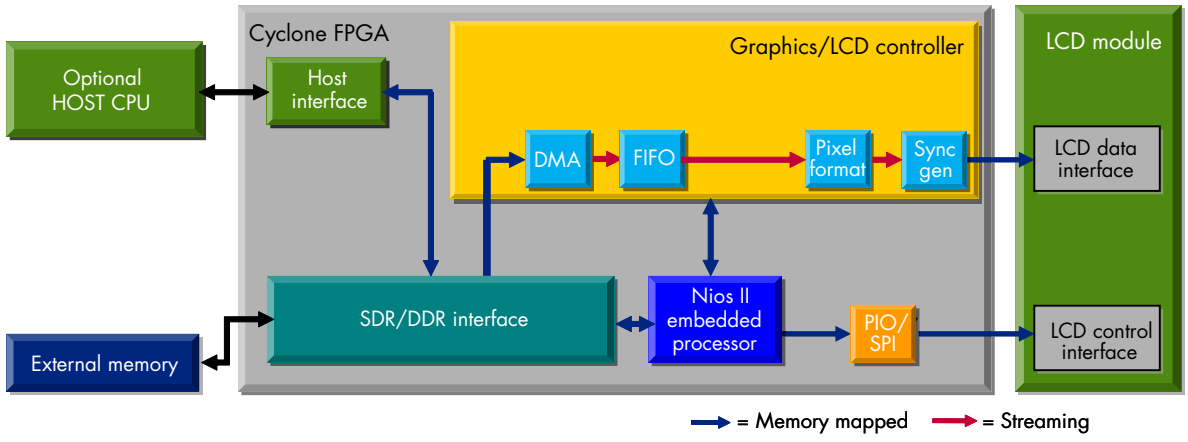
The easiest way to create a graphics system is to use a microcontroller-based device and a LCD module. As the module already contains the display-driver devices, it usually can be connected directly to the microcontroller. The image to be displayed is created and stored in the microcontroller's memory space by a program running on the microcontroller. These devices often include dedicated on-chip graphics memory and a dedicated hardware graphics engine to render computationally intensive special effects. The hardware is also usually responsible for copying the final image from memory to the LCD module interface, as doing this with the microcontroller is too expensive in terms of processing time.

However, this setup is limited by the fixed features of the microcontroller, both in terms of its capabilities as a microcontroller and as a graphics engine. Embedded applications always are limited by the features of the microcontroller, but device manufacturers mitigate this by producing a wide range of family variants with different I/O features and performance levels. Unfortunately, only a few of these family variants include a graphics engine, so the requirement for graphics severely limits the microcontroller choices available. In addition, the features of the graphics engine cannot be changed, so if the application needs to support new graphical features, a higher quality display, or a new input device (e.g., a touch screen), a higher performing, and consequently, more expensive device is usually required. This often means that the application code must be ported to a new processor architecture, which is expensive and time-consuming.

FPGA-Based Implementation

Thanks to the advent of low-cost FPGAs like the Altera® Cyclone® FPGA series, many developers have eliminated the problem of microcontroller-feature limitations by designing custom systems using a combination of soft microprocessor cores (e.g., the Nios® II embedded processor), off-the-shelf intellectual property (IP), and custom-designed logic. Should the required specification change, the system can be redesigned using Altera's SOPC Builder tool*(2)*, which literally can be done in minutes. A typical basic design (Figure 2) contains all of the same features as a microcontroller-based system (Figure 1).

Figure 2. Typical FPGA-Based Embedded System With Display Support



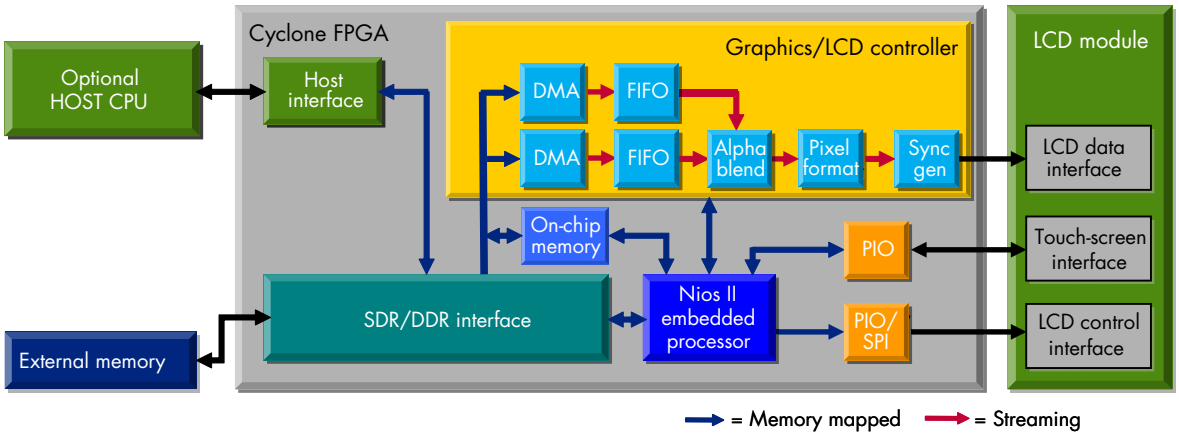
The image is rendered into memory by the Nios II processor (or the optional external CPU) and is transferred to the LCD using a direct memory access (DMA) hardware block inside the graphics controller. Due to the HDL-based controller, it is easy to modify the LCD data and control interfaces to match any current or future LCD device. This also means the system can be ported easily to newer FPGAs to reduce costs or to enhance other system features such as newer and cheaper generations of memory (e.g., DDR2, DDR3).

The flexibility of the FPGA means that it easily interfaces to other devices and systems (e.g., PCI, PCI Express, Serial RapidIO®, memory-mapped buses, coprocessor interfaces, etc.). An external host processor (Figure 1) can be used when it is more cost effective to use a high-performance microcontroller that is not enabled for graphics. This delivers performance, cost effectiveness, and a wide range of microcontroller choices without having to compromise on graphics or I/O features. The application is isolated from any changes to the graphics system and the FPGA supports additional non-standard interfaces and application-related functions.

A Scalable Solution

The most commonly requested upgrade features for embedded systems with graphical displays are color (or more colors), increased resolution, and touch-screen capability. With a fixed microcontroller, making these additions can be very difficult or even impossible, but they are relatively straightforward with a FPGA, especially when using the example designs provided with the Nios II Embedded Evaluation board *(3)*. As a simple example, the pixel format and sync-generator blocks shown in Figure 2 can be modified to support different types of display without changing the rest of the system or the software (see "Productivity" section on page 7). Figure 3 shows how the basic system and controller can be enhanced to support additional features like alpha blending and touch-screen capability.

Figure 3. FPGA-Based Embedded System With Enhanced Display Support



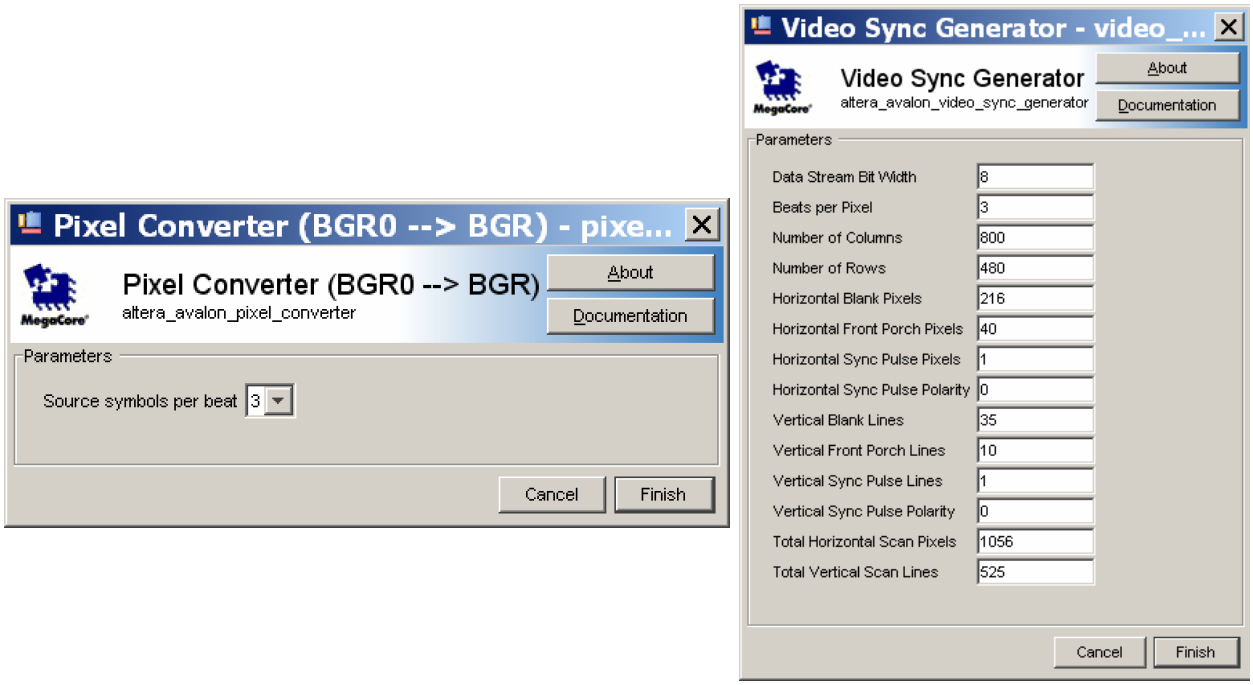
Support for many of the enhanced features shown in Figure 3 usually requires a high-performance and expensive microcontroller, but with FPGA all that is needed is the relevant hardware IP (available off the shelf or custom built) and enough additional FPGA logic gates (migrate to a larger pin-compatible device if necessary). As much of the work is done in hardware by the FPGA logic, more graphical features per clock can be supported than in a software-based solution on a traditional fixed-architecture microprocessor-based system. If the standard Nios II processor does not have the performance to run the application and render the graphics, the performance of the system can be optimized by accelerating the graphics or the application in the following ways:

- Add custom instructions to the Nios II processor to accelerate the application
- Add hardware accelerator modules to accelerate the application
- Add custom hardware to the LCD controller to accelerate graphical operations
- Implement burst access in DMA modules to optimize memory bandwidth
- Add a second processor to run the application

For more information on hardware acceleration of applications, refer to Altera's Scale System Performance web pages *(4)*.

A comparison of Figure 2 and Figure 3 shows that hardware has been added to perform alpha blending without adding any performance load (other than setup of the second DMA and blending of hardware modules) to the processor. Often the use of alpha blending includes the use of per-pixel alpha data, which may require increased data throughput from the processor and more memory space. However, with the ability to customize the system, the processor can be optimized for space and performance by implementing alpha components only where required, and by using pixel-formatting hardware modules (instead of the processor) to convert data formats as the data streams into the relevant module. For example, the pixel-formatting module provided with the Nios II Embedded Evaluation platform can strip alpha data from the pixel before it is sent to the display. As this is a SOPC Builder-compatible component, this capability can be enabled by changing a number in the graphical configuration window (Figure 4, left) rather than editing HDL. This type of component can be easily reconfigured to support other modes of operation, such as the video sync module shown in Figure 4, right.

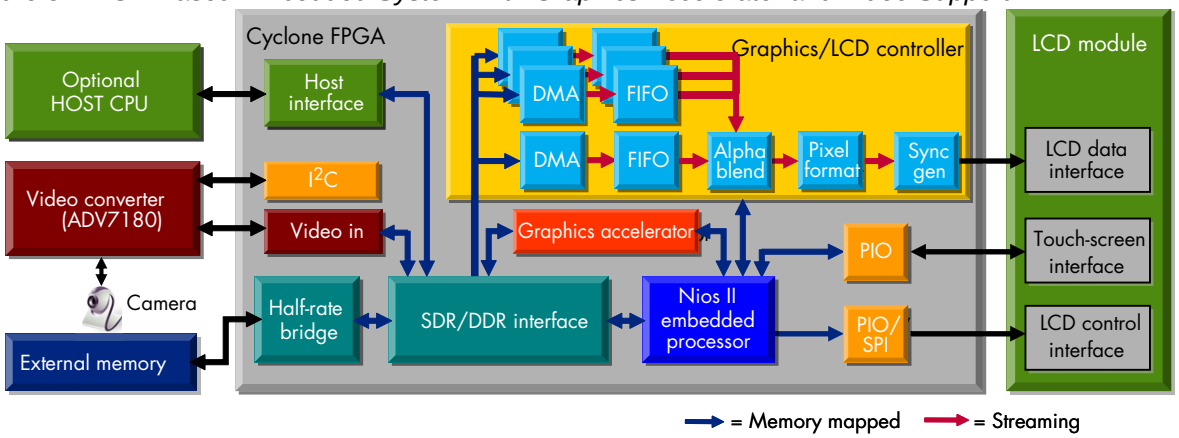
Figure 4. SOPC Component Configuration Window for the Pixel Format Module (left) and Video Sync Module Used in the Nios II Embedded Evaluation Kit (right)



It is also possible to implement additional features like picture-in-picture, background, and overlay images in the same way as alpha blending. As these are simply the replacement of data in a certain range with data from another region in memory, it is implemented easily and can support features like transparency, compressed (or palletized) source data, and scaling and storage in local on-chip memory (for high performance and reduction in bandwidth loading on the main memory system).

As the sophistication of the graphics rendering increases, it may be cost effective to license an off-the-shelf graphics accelerator. These types of graphics engines are often built into the video cards of PCs, because they access memory and process data with custom hardware to accelerate graphical operations. In these systems, the function of the Nios II processor typically is reduced to housekeeping and very high-level graphical control, leaving all of the image rendering and data movement to the graphics engine and LCD controller. Figure 5 shows a typical high-performance system with a graphics accelerator replacing many of the graphical functions previously implemented in the LCD controller.

Figure 5. FPGA-Based Embedded System With Graphics Accelerator and Video Support



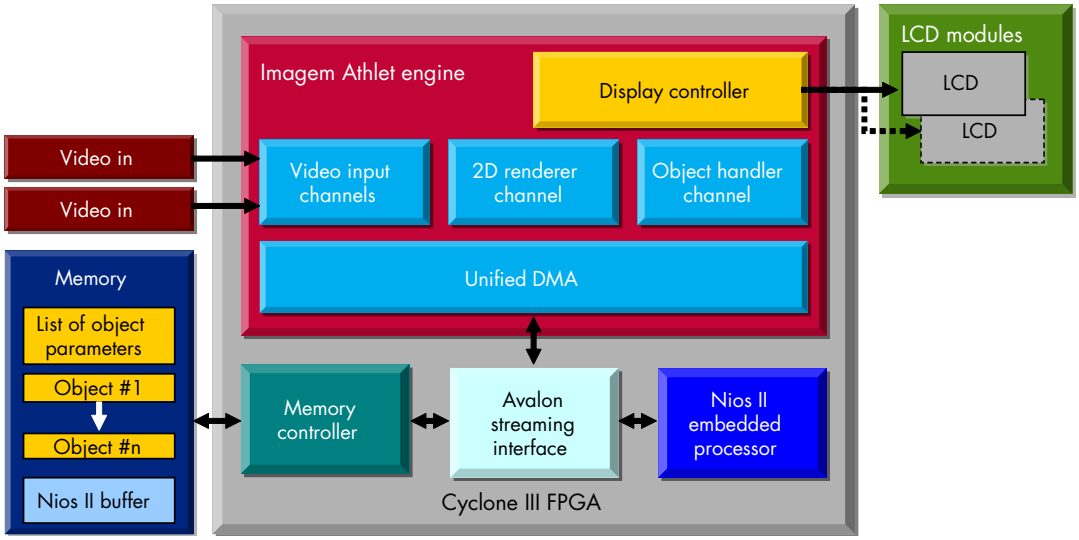
Graphics engines can be licensed from Altera's partner companies, such as TES *(5)* and Imagem *(6)*, and are available in a range of configurations to suit various application performance and feature requirements. The performance increase and quality delivered by these graphical accelerators is impressive and they deliver similar, or even better, quality graphics than the highest end microcontroller solutions, but with much lower system-clock frequencies. Examples of effects that can be implemented in FPGA logic include alpha blending, anti-aliasing, drawing and filling of geometrical shapes, scaling and color conversion of graphical elements, support for scaled fonts, and even 3D rendering.

The system shown in Figure 5 also supports video input, a streaming data conversion operation that is easily implemented in the FPGA logic. Because the video stream can be processed both "on the fly" or once the video frame is in memory, these techniques are used in applications like lane-departure detection in the automotive market *(7)*. Altera offers a suite of SOPC Builder-compatible video and image-processing IP designed specially for fast and easy development of video-processing systems *(8)*.

With all of these additional features, one possible issue is memory bandwidth. Despite the use of processing on the fly as the data streams through the system and on-chip memory for local buffers, the maximum available memory bandwidth ultimately limits the system performance. Thanks to the flexibility of FPGAs, there are many solutions to this problem. One simple solution is simply to add a second memory device. However, this usually is not cost effective, so most solutions focus on optimizing the memory controller and sequencing memory operations more efficiently so the memory bursts are not interrupted by random memory accesses. In Figure 5, a half-rate bridge (a standard Altera component) is inserted between the memory controller and the memory, allowing the system to run at half the rate of the memory; this greatly reduces the memory latencies and setup times involved with DDR-type memories. This simple change delivers higher memory bandwidth and therefore better graphics performance.

With the flexibility of the FPGA, there are many ways to deliver additional performance. For example, the Imagem Athlet engine (shown in Figure 6) reads a display list of objects and renders the image directly to the display without using a frame buffer at all. This reduces the amount of memory required, minimizes the memory bandwidth requirements, and makes it extremely easy to scale up the display resolution. The objects and their parameters are all set up by the Nios II processor but the rendering pipeline is not involved in the actual rendering process at all.

Figure 6. Architecture of the Imagem Athlet Graphics Engine



Implementation in an FPGA also enables the addition of features that are difficult or nearly impossible to add using most standard microcontrollers. Examples include multiple channels of streaming video, picture-in-picture capability, multiple-display support, image enhancement, object recognition, motion tracking, camera distortion correction, noise filtering, and support for market-specific features like flash animation files. Table 1 lists some example graphics IP components and the typical amount of Cyclone III FPGA logic elements (LEs) required by each component.

Table 1. Typical LE Requirements for Graphical IP Components

Feature	LEs
Basic LCD controller	2–3K
Multi-layer controller with alpha	4–5K
Video input with basic processing	2–3K
Hardware accelerator with basic 2D features	6–8K
Full-featured 2D hardware accelerator	11–13K
Flash Player (Imagem Technologies)	14–16K
3D Graphics Accelerator (TES)	~75K

A minimal basic graphics system (as shown in Figure 2) will fit into a Cyclone III EP3C5 FPGA (the smallest device in the family), whereas a high-performance system, similar to that shown in Figure 5, requires a EP3C16 device or larger. Memory and DSP block requirements for a simple display are minimal but they increase as hardware-accelerated features are added. Fortunately, the ratio of DSP and memory to logic in the Cyclone III family is well suited to these types of applications, so the amount of available on-chip memory and DSP resources is rarely a problem. Table 2 shows the approximate amount of LEs required by a Nios II processor-driven graphical system with a varying graphical feature set and the typical FPGA device required to support the system.

Table 2. Typical LEs and Device Requirements for a Range of Graphical Systems

Graphics System (incl. 50–100-MHz Nios II processor)	LEs	Typical Device
Minimal (basic drawing)	3–4K	EP3C5
Mid-range (+ multilayer + video capability)	5–11K	EP3C10–EP3C16
Mid-range (+ flash capability)	15–20K	EP3C16–EP3C25
High end (+ anti-aliasing, bezier curves, etc.)	14–18K	EP3C16–EP3C25
High end with 3D graphics	~80–90K	EP3C120–EP3S110

Productivity

One of the main problems that developers have with FPGAs is the fear that changing hardware and a soft processor core will reduce their productivity. Changing to a new processor architecture and toolset always involves a certain amount of work, but once over the initial familiarization stage, working with FPGA-based systems is as easy as working with any other processor tool chain. In addition, the ability to change the hardware allows much more efficient and creative engineering solutions.

With Altera's Quartus® II design software, the designer can create a whole embedded system using HDL, but it is much easier to use the integrated SOPC Builder tool. This tool enables the creation and modification of an embedded system without writing a single line of HDL code. The Nios II Embedded Evaluation Kit comes with a design example of an embedded processor system with an LCD graphics controller composed entirely of SOPC Builder components. This makes the system easy to maintain and change, and is an excellent base for developing a custom graphics controller*(9)*. As an example, a HDL-based reference design that has been optimized for the automotive market is available for download from Altera*(10)*. Altera's partner companies deliver their IP as SOPC Builder-ready components so their graphics accelerator solutions are easy to integrate into a design.

One of the main productivity issues with developing systems with graphical interfaces is making graphical and structural changes to the GUI. Typically, the GUI is developed in C, so if the code is implemented without a high degree of graphical object and user-interface structure abstraction, any changes are likely to require a significant amount of software development. Just as the use of IP and design automation (with SOPC Builder) accelerate hardware development, the same principles can accelerate GUI design. Companies like Imagem*(6)* and Altia*(11)* provide design tools and development flows that accelerate GUI design.

The first stage is to use tools that can convert graphics designed with professional graphics tools (like Adobe Photoshop) into formats that are ready to place directly into the systems memory. This enables graphics to be designed by artists and graphics designers rather than programmers, delivering much more visually impressive GUI designs and speeding up the development process. The next stage is to enable development and rapid prototyping of the GUI design (using the actual graphics) on a PC. The PC emulates the embedded environment so that the GUI and graphics can built and tested at the right resolution and color depth without writing a single line of code. This makes the design process much easier and friendlier to graphics designers and other non-technical staff. GUIs built in this way can be tested and modified easily to deliver an efficient and graphically impressive implementation in a much shorter space of time.

Once the design is complete, the tools can generate both the graphics and C code for the embedded platform code, making it easy to implement the GUI on embedded system. Any changes can be made and tested in the PC environment and then moved onto the embedded platform without writing any code. This is much quicker and easier than the endless review/restructure/rewrite cycles between the graphic designers and programmers involved in the traditional non-automated process.

Getting Started

As with any embedded system development, the easiest way to get started is to purchase a development kit targeted at the application. The Nios II Embedded Evaluation Kit (Figure 7, left)*(3)* is an easy way to learn how FPGAs and the Nios II processor can be used to develop graphics solutions. It comes with many reference hardware and software designs and is supported by all of Altera's third-party graphical solutions*(5)(6)(11)*.

For more information on the design used with the kit, refer to *AN 527: Implementing an LCD Controller (9)*.

The Sasco Holz Pablo display controller platform (Figure 7, right)*(12)* supports a wide range of displays through the use of a range of off-the-shelf add-on adapter cards and IP cores, and is ideal to evaluate and verify the flexibility of FPGAs with a variety of displays.

Figure 7. Nios II Embedded Evaluation Kit (left) and Sasco Holz Pablo Display Controller Platform Base Board (right)

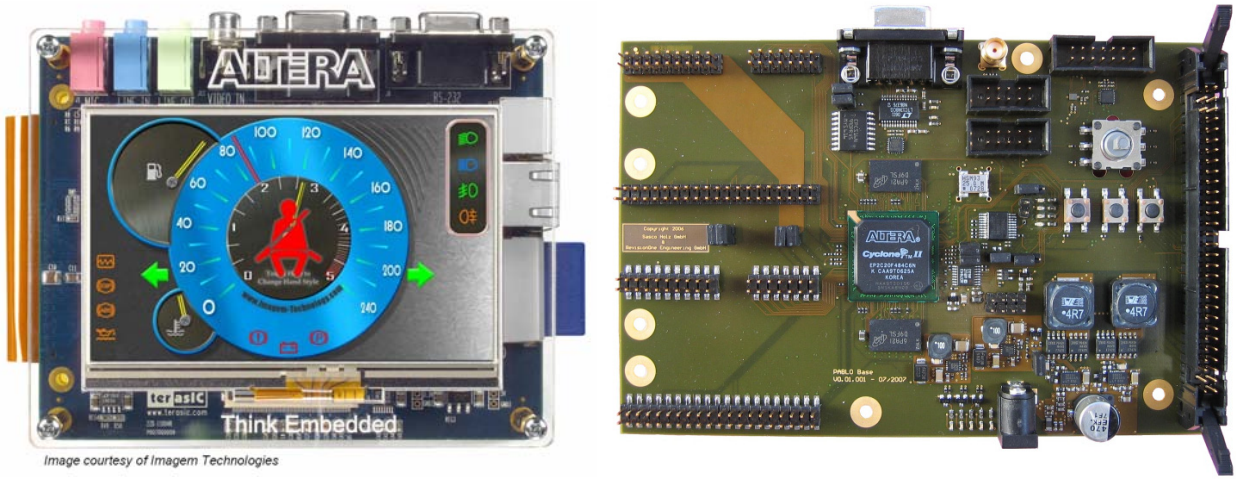


Image courtesy of Imagem Technologies

Conclusion

Having a visually impressive and easy-to-use GUI is increasingly important in a wide range of markets and products. The best way to achieve such a GUI is to use an automated development flow that enables professional graphics development, PC platform prototyping, and automated conversion to the embedded platform. While this type of flow is available for both microprocessors with graphics support and FPGA-based systems, FPGA-based designs offer much more flexibility in terms of updating the system and supporting higher resolution displays, displays with different interfaces, and new GUI features without having to port the application code to a new processor architecture. As with FPGAs, the design is IP based it can be easily migrated to larger or newer FPGAs as the market demands.

Further Information

1. DisplaySearch Quarterly Worldwide FPD Forecast Report:
   www.displaysearch.com
2. Quartus II Design Entry and Synthesis-SOPC Builder:
   www.altera.com/products/software/quartus-ii/subscription-edition/design-entry-synthesis/qts-des-ent-syn.html
3. Nios II Embedded Evaluation Kit, Cyclone III Edition:
   www.altera.com/products/devkits/altera/kit-cyc3-embedded.html/
4. Scale System Performance:
   www.altera.com/technology/embedded/fpgas/performance/emb-hi-perf.html
5. DAVE IP core:
   www.TESBV.com
6. Athlet IP core:
   www.imagem-technology.com
7. *Image-Based Driver Assistance Development Environment*:
   www.altera.com/literature/wp/wp-01091-driver-assistance.pdf
8. Image and Video Processing:
   www.altera.com/products/ip/dsp/image_video_processing/ipm-index.jsp
9. *AN 527: Implementing an LCD Controller*:
   www.altera.com/literature/an/an527.pdf
10. Automotive Graphics Controller Reference Design:
    www.altera.com/support/refdesigns/sys-sol/auto/ref-auto.html
11. Altia PhotoProto, Altia Design, DeepScreen:
    www.altia.com/products.php
12. Sasco Holz Pablo Display controller platform:
    www.sascoholz.com/designsupport/solutions/pablo

Acknowledgements

- Stefano Zammattio, Product Manager, European Product Marketing, Altera Corporation

101 Innovation Drive
San Jose, CA 95134
www.altera.com