Reach customer wins “Product of the Year” in the Mobile Electronics Category at SEMA

The ISIS inTOUCH™ hand-held wireless remote is a hit at the world’s premier automotive specialty products trade event

Background

The ISIS inTOUCH Intelligent Multiplex System, a joint development between Littelfuse and I Squared Engineering, replaces traditional vehicle wiring harnesses for custom car enthusiasts and limousine manufacturers. It allows users to program circuits that control electrical functions within a vehicle (for example, starting the car, controlling interior and exterior lights, manipulating heat controls, locking the car). I Squared Engineering needed to add a color touch interface without doing a tremendous amount of research and development.

Critical Issues

To eliminate *any* possibility of failure due to software bugs or inadequate testing, I Squared Engineering wanted a pocket-sized touch screen that did not contain a complex operating system. According to President Chris Loubier, “I would *absolutely not* use a touch screen that involved Windows, Linux, or UNIX, or any other operating system. I wanted something that ran bare metal without dynamic memory allocation - the idea of getting the blue screen of death behind the wheel of a car is not good!”

Solution

I Squared Engineering asked Reach to develop a display module based on a popular 4.3” LCD size being used by top portable gaming systems, GPS navigation, and other hand-held devices. They used this as the foundation for ISIS inTOUCH, a hand-held wireless customer interface that can control any vehicle function. While Reach developed the 4.3” display module, I Squared Engineering focused on their core competencies: power control modules and associated firmware.

Results

ISIS inTOUCH was recently named Product of the Year in the Mobile Electronics Category at SEMA 2008, the world’s premier automotive specialty products trade event. Loubier said, “Seeing the wireless touch screen tied into the ISIS distribution center thrilled editors at SEMA. They told us, ‘This changes the game in the restoration, aftermarket, and commercial vehicle space.’” They started shipping their first production units in February 2009.

See the compete ISIS solution demonstrated in a video shot at SEMA (http://www.isispower.com/V8_interview.php). If you are only interested in seeing the 4.3” display module, start watching the video at 14 minutes and 18 seconds.

Jay Harris, Global Director of Business Development, Automotive Business Unit at Littelfuse demonstrates ISIS inTOUCH at SEMA.

Should I use a full-blown, embedded operating system, like Windows CE, CE Linux or QNX?

There is a common belief that still lingers: that a full-blown operating system is required to implement a modern graphical color touch interface. Certainly Microsoft would have you believe this is the case. Today, there are other options.

If you're simply upgrading an existing product with color touch technology, it's not likely that a full-blown, embedded operating system would be your logical choice.

A complex operating system is appropriate if:

- You're designing a new product and it needs a high-level interconnection with other, similar OS-based systems.
- The system will have off-the-shelf peripherals, such as a fingerprint reader, attached to it.
- You're well-versed in the operating system you're going to use, or are willing to commit the time to become comfortable with it.

Considerations

Processing power: A full-blown operating system needs a full-blown single-board computer (SBC) to run on. If your current product is based on a microcontroller, you have to make a major hardware platform change.

Operating system experience: You and your team must be (or become) familiar enough with the operating system to make accurate risk assessments. A typical statement of concern: "Is there some implementation detail that I don't know about and don't know I have to consider? They say it'll do what I need it to do, but will it? If I have a problem, who do I call?"

Start-up and shut-down time: Boot time and shut-down time are issues to consider. High-end operating systems are not inherently designed to do either of these as quickly as your application might require.

Memory leaks: The specter of memory leaks is almost inevitable with high-end operating systems. Consider this issue if memory leaks could pose a problem for your application. For the uninitiated, memory leaks and the consequent low-level processor abort is the major cause of PC "crashes."

Reliability: The more code, the more bugs. It is just that simple. Operating systems inherently have a lot of code.

+ Advantages of embedded operating systems

Industry-standard, embedded operating systems have their advantages. Among them:

- They're industry standards.
- High level networking connectivity features are built in.
- You can connect to centralized databases using standard software interfaces.
- You can connect to PCs on a peer-to-peer basis.
- They permit the use of Flash animation.
- They're "future-proof." Maybe your product doesn't need networking, database connectivity, Flash, or any of that now, but, if you think those features will be required in the future, a full-blown operating system could be the way to go.

- Disadvantages of embedded operating systems

Some projects require the power and versatility of high-end operating systems, but such systems also have substantial disadvantages:

Increased complexity: Full-blown embedded operating systems are extremely complex. Unless you and your team already possess deep expertise in their use, you'll spend months gaining the necessary knowledge and experience.

Boot time: In most embedded applications, boot time is a big issue. Without special programming, full-blown operating systems require 15 – 20 seconds to boot. You can improve that, but to do so requires an expert level of knowledge.

Shut-down issues: What happens when power fails and the system is not shut down "gracefully"? Is shut-down time important to your application? If so, be aware that high-end operating systems require substantial time to shut down correctly. As with boot time, shut- down times can be reduced, but doing so requires substantial know-how.

Test complexity: For an embedded test, all the permutations of the run-time environment must be identified and incorporated into the test suite. This is a huge job and it's rarely fully- accomplished, as is obvious whenever an embedded "system" crashes.

"Is there some implementation detail that I don't know about and don't know I have to consider? They say it'll do what I need it to do, but will it? If I have a problem, who do I call?"

Interface responsiveness: We have become used to our PC's having the occasional "hiccup," as one of its many background processes eats up processor cycles. However, if your microwave oven took varying times to respond to the "start" button, you'd probably think it was broken, or on its last legs. Embedded user interfaces need consistent response times and this is more difficult to achieve with a complex multitasking operating system.

Sample Windows CE Code Used to Create a Frame and Add a Button

```
int CMainFrame::
OnCreate(LPCREATESTRUCT lpCreateStruct)
{
  if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
      return -1;
  // Add the buttons and adornments to the
CommandBar.
  if (!InsertButtons(tbButtons, nNumButtons,
IDR_MAINFRAME, nNumImages) ||
      !AddAdornments(dwAdornmentFlags))
  {
     TRACE0("Failed to add toolbar buttons\n");
      return -1;
  }
  return 0;
}
```

COLOR
TOUCH
CONTROL
HANDBOOK

Read an in-depth discussion of your Buy vs. Build choices in our white paper, ***Need help adding a graphical color control interface to your product?*** Download it at www.reachtech.