White Paper

WP108

RabbitSys

Conquering Remote Reliability Problems

As an embedded systems engineer, you can use all the help you can get to create a reliable system. Unfortunately, interrupts are missed, the stack overflows, interrupts fire in unanticipated order, and the uncontrolled world external to your microcontroller will do its best to crash your system. System reliability overshadows the goals of any embedded project. RabbitSys increases the reliability of your embedded system. How RabbitSys achieves this reliability will be explained throughout this paper as we introduce and describe each component of this revolutionary 8-bit system. RabbitSys' rich featureset includes:

- **Remote application update**
- **Remote configuration**
- **Remote application monitoring**
- **Remote problem detection and diagnosis**
- **Hardware supported application protection**

RabbitSys does more than just increase system reliability; RabbitSys increases your productivity during the development and debug cycle. RabbitSys saves you time during development by providing more information during debugging and greatly decreasing program download time. When development is done, RabbitSys assures the integrity of your deployed product by safeguarding access to all embedded system resources, such as memory and I/O ports.

In addition to saving time during development, RabbitSys saves you time and money after you deploy your product. Without RabbitSys, the fix to a software problem in the field could require you to book a flight, send an engineer off-site, lose valuable time from today's project and, most importantly, make your customer wait. Instead, by using RabbitSys' reliable remote update feature, you will remove the need to "roll a truck" to fix a software problem.

Most new software packages require a learning curve to achieve proficiency, with the steepness of the learning curve dependent on various factors. RabbitSys was designed to be easy to use. Existing Dynamic C applications can be compiled under RabbitSys with little or no code changes to benefit in full from the increased protection, reliability, and convenience that RabbitSys offers. New features, such as the program monitor or the event handler, enhance your application's capabilities in the field, making it more resistent to external transient errors. As you will see, these features create a solid foundation for your application, and we'll show you how to take advantage of them. The RabbitSys application programming interface (API) is simple and straight forward. It does not matter whether you are migrating your existing program or creating a RabbitSys application from scratch. In other words, Dynamic C users will have no trouble making the switch to RabbitSys. Those new to Dynamic C will find that using the power of RabbitSys requires little effort from the programmer.

Fixing Reliability Problems

Reliability problems in an embedded system often follow with customer descriptions like “It hangs up once in a while and I have to reset it.” or “It goes crazy a couple times a month for no apparent reason.” Invariably, these sporadic problems too often result from subtle bugs in software that do not show up during quality assurance testing. Software reliability problems stem from two basic issues: data corruption and flow control bugs. The effects of these two troublemakers propagate and compound the original problem resulting in a completely unstable system. A problem with data corruptions soon leads to bad logic, and faulty logic soon leads to non-sensical data. Although no system can fix the logic or data of an errant program, RabbitSys allows you to hone in on problems that commonly plague deployed systems, such as:

- Stack or static data corruption from errant pointers or buffer overrun.
- Stack overflow caused by cascading interrupts or unforeseen events.
- Buffer overrun from boundary assumptions.
- Bad logic that causes control flow to go astray.

Let us take for example a typical embedded program that calls a memory copy routine to retrieve serial data from a buffer. As is often the case, the programmer mistakenly assumes that the data will never exceed some threshold value, say 128 bytes. Let’s say the serial data overflows the buffer, changing the variable that drives a motor control state machine. The state machine then either goes into the wrong or invalid state and eventually results in a crash of the program and leaves the motor running out of control.

For the scenario above, RabbitSys makes it possible for you to take control and bring the system back into a safe state, restarting or halting the program at your discretion. Upon detecting the errant program, RabbitSys logs what went wrong and calls routines to put the controller (and the motor) back into a safe state. RabbitSys also sends alerts to you via e-mail or pager, to notify you that something has gone terribly wrong and gives you to power to correct it. In the next few sections, we’ll look at the various components that make this recovery possible.

The RabbitSys Attack Plan

At the core of RabbitSys lies its event-driven kernel that drives various system components including a Monitor, Console, TCP/IP Stack, and Remote Upload Utility. As the figure below illustrates, the design of RabbitSys is simple without sacrificing function. The RabbitSys kernel takes control when the system starts up. The kernel then verifies and starts your application. Note that each of the components shown plays a critical role in the battle against reliability problems. The RabbitSys attack plan consists of three phases:

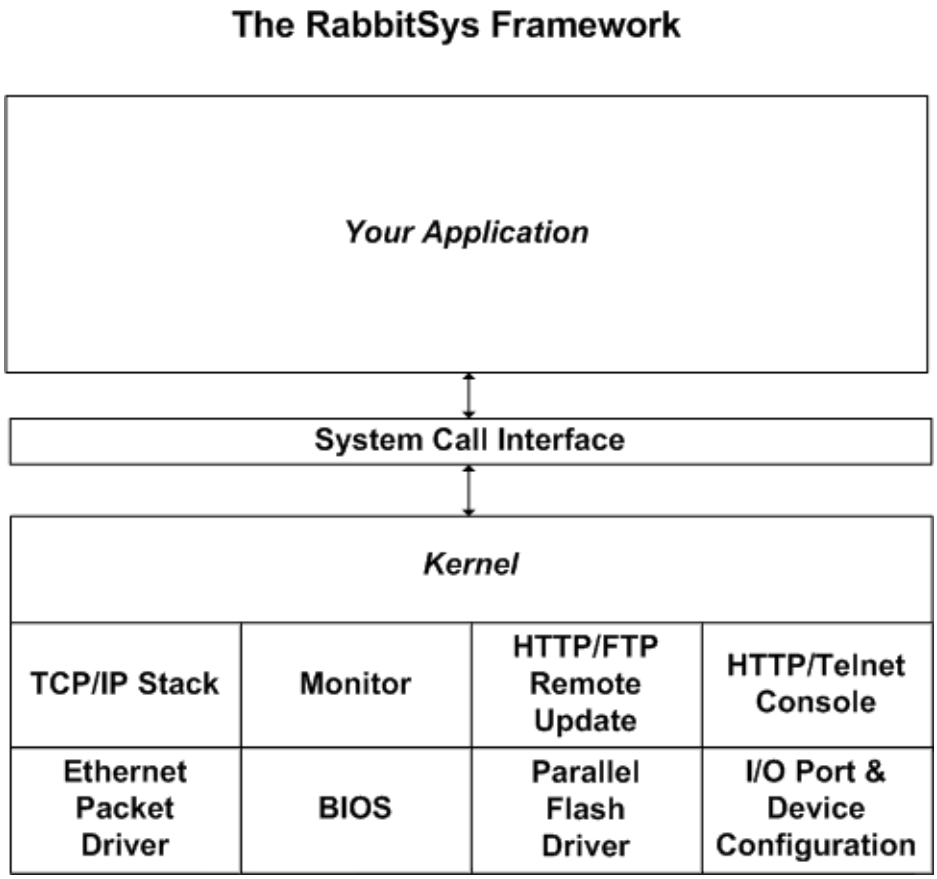
Detect the problem on the target. When an unexpected event or error occurs like a system violation, the RabbitSys Monitor logs it and the kernel takes action. This action includes sending e-mail for notification and resetting or stopping the application depending on configuration and the severity of the error.

Diagnose the problems using the RabbitSys Console to create watches on application data and configure the monitor to log additional events, and even trace program execution remotely. The console has an easy to use interface that can be accessed via HTTP or telnet.

Deliver new application code to the target that fixes the issue through Remote Program Update.

The RabbitSys Framework

This section describes the interaction of your application and RabbitSys.



Your Application

Your program has direct access to the rest of the system via the system call interface. This design gives the maximum control and power to the user application without undermining system protection. The protected execution environment created by the RabbitSys kernel monitors the behavior of your program when out in the field. Even if your program fails, whether quietly or in some very destructive manner, RabbitSys protects the core system to ensure remote accessibility so you can correct the problem.

RabbitSys allows your application to make use of Dynamic C's full-featured debug kernel. During development, the debug kernel is compiled into your program. When your program is ready for deployment, the debug kernel can be completely removed from your final product if desired. For more information about our debug kernel, please see the *Dynamic C User's Manual*.

System Call Interface - SysCall

Your program accesses RabbitSys through the system call interface. When a system call request occurs, RabbitSys verifies the type of the system call, associated parameters, and device handles before servicing the call. Pointers passed across the system call interface are checked to ensure protection against accidental corruption of system resources. This pointer checking is just one example of the kind of cross checking that RabbitSys performs on your application's operation. The cross checking of software is one of the primary methods of achieving system reliability.

The Event-Driven Kernel

A powerful event handler drives the RabbitSys kernel. Since system events require decisive action, it is easy to see how useful this RabbitSys feature can be in an embedded system. RabbitSys predefines system events and gives you the ability to add your own user-defined events. For user-defined events, you can customize your program's behavior by registering a call-back function for that event. API functions allow events to be scheduled, removed, and queried. Some examples of events that drive the system are:

Timer events are used by the kernel for driving virtual watchdogs. User timer events can be added for custom functionality.

Shutdown events are triggered when a fatal error has occurred in your program. By hooking in a shut-down callback routine you can ensure well-defined system behavior when the unexpected occurs.

Alert events are triggered when a program error occurs. When an alert event occurs, the kernel can take corrective action by sending e-mail, resetting your application, or even stopping it depending on the severity of the error.

Remote Program Update

Whether your RabbitSys-enabled target is located across the room, across the country, or across the world, uploading an updated program from your office is a quick, simple matter. Whether you are adding features or maybe fixing a bug in critical code, RabbitSys saves you considerable time and resources. As previously mentioned, the remote update removes the need to "roll a truck" to fix your software problems.

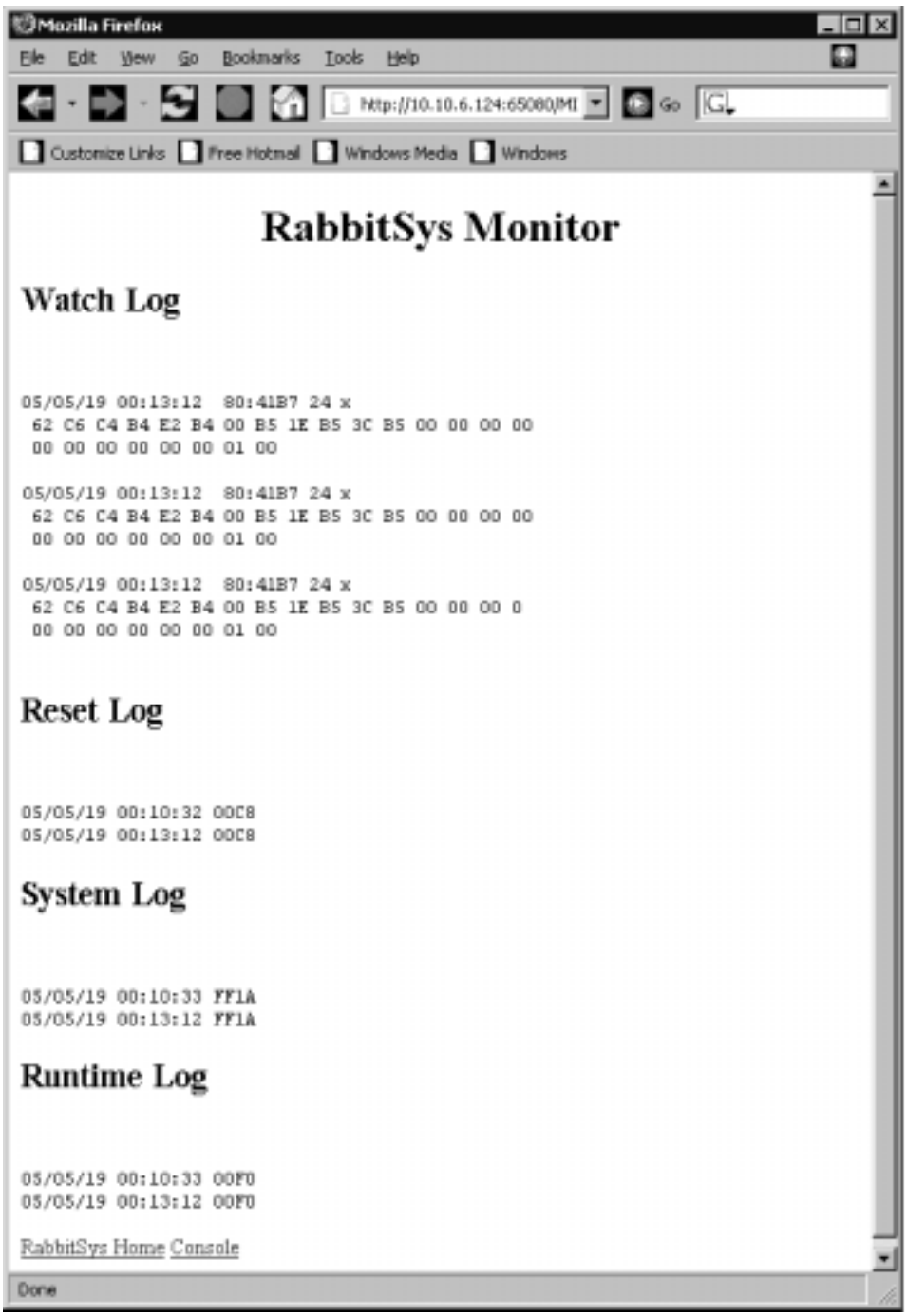
RabbitSys natively supports application update through TCP/IP, serial, and the RabbitSys API. For the sake of flexibility, there are several different methods for remote update of user program code:

- Update via system mode HTTP server.
- Update via system mode FTP server.
- Update via the remote upload API. This API allows you to utilize a different network protocol for your own custom update solution. For instance, you might want to enhance RabbitSys' remote update with HTTPS and the Secure Socket Layer (SSL) for added security.

The RabbitSys network servers run at high port addresses to avoid interfering with your application's HTTP and FTP servers. The system servers are periodically checked while your program is running to see if an update has been requested. The screen shot below shows the web interface for remote update.

Monitor

RabbitSys maintains an audit trail to support detection and diagnosis of system reliability problems. The RabbitSys API for the monitor provides the ability to query program status, log and handle run-time errors, and respond to critical system errors. The monitor tracks hardware and software resets, runtime errors, and system access violations. The monitor can also be configured to log or watch the contents of memory locations for diagnostic debugging. Your program can also register callback functions to shut down properly, to handle errors, or to send custom alerts. The screen shot below shows the web interface for accessing this system information from the monitor:



The monitor is accessible via both the console and HTTP. There are five types of monitor logs: watch, reset, run time, system and fatal (not shown). Monitor logs provides information on the state of a running or crashed program such as time, address, data, and cause.

Watch logs trace sections of memory, allowing you to monitor specific values in a running application. In the image above, the watch log is displaying the Ethernet MAC address of the board (retrieved from your program's map file).

The reset log records all software and hardware resets, which assists in crash recovery. These entries are time and date stamped and may provide useful information if a program develops a hard to track bug. RabbitSys inspects the GCSR register on restart and records its value to determine the cause of reset including power failure and watchdog timeouts. User mode run-time errors behave similar to a watchdog timeout and can either reset the board or take other corrective action based on user configuration. Customizable recovery allows the system to take other corrective action like sending e-mails alerts or putting attached devices into a safe state.

The last three monitor logs, run-time, system, and fatal, provide different levels of error reporting. The default behavior for all of these errors is to send alert notification via e-mail and shut down your application. However, custom behavior can be defined. Errors that are detected within your program such as divide-by-zero or your own custom-defined error cause run-time errors. With run-time errors, your program can register a recovery call-back that allows you to take corrective action, shutting down the program at your discretion. System and fatal errors such as attempts to corrupt system data, corrupt the system stack, or execute system code are detected by RabbitSys. Since system space cannot be corrupted by an application program, RabbitSys continues to function even after a fatal program error. It assumes control and the monitor will respond to requests for information through either the console or the RabbitSys HTTP server. When an error causes your program to shut down, you can use the console to restart it by clearing the associated log.

Console

The console provides a machine-friendly, yet human-readable, command-line interface enabled during system bootup that is active in RabbitSys at all times. You can communicate with the console directly over serial or through TCP/IP using Telnet, HTTP, or FTP. The console provides the interface through which you can configure and view network settings, configure login, access the monitor, update your program, check version, add watches, reset the program and even reset RabbitSys itself. As if that wasn't enough, when you remotely compile and download a new binary with debugging enabled, RabbitSys will allow you to debug the program directly through Dynamic C. Since the console is accessible via common networking tools like FTP and telnet, your RabbitSys-enabled board can be accessed without having to install yet another proprietary software tool. Since remote update works through FTP, you can use the native client in your PC's operating system to upload new firmware. The screen shot pictured below shows the commands available from the console.

HTTP Server

As you have already seen, RabbitSys comes with an HTTP server that provides complete access to remote upload, the console and the monitor. The ability to use a standard browser to interface with the target board offers you a quick, convenient communications channel from anywhere on the Internet. RabbitSys allows your application to register HTTP pages and callback functions. This feature allows you to control your application remotely.

Tasking Support

RabbitSys supports both preemptive and cooperative multitasking. It supports real-time environments such as µC/OS-II, as well as Dynamic C slice statements and costate constructs. RabbitSys also provides stack switching services to give your program flexible task management.

To support user-level tasking, e.g., µC/OS-II or slice statements, RabbitSys provides a means by which your program can safely hook into the periodic interrupt. RabbitSys monitors the amount of time that your program code runs and will prevent your code from violating system time constraints.

Alternatively, for applications that require finer grain control over system running times, RabbitSys provides a system tick function that must be called manually. Instead of hooking to the periodic interrupt, RabbitSys works cooperatively with your program. By calling the system tick, the system runs during the normal course of user program execution. RabbitSys expects to be called periodically and may halt the user program if not allowed to run. The tick function executes rapidly in order to incur as little time penalty as possible.

The System Tick

The system tick drives RabbitSys. On each invocation, the system tick hits both the primary and secondary watchdog timers. The tick function then runs the various subcomponents of RabbitSys as needed. The ability to associate user-defined code with timer events, run-time errors, and system shutdown increases system reliability by allowing you to respond appropriately to problems like a random jump into system code or a stack imbalance. If the tick is not called frequently enough, either the primary or secondary watchdog will fire, and the system will take corrective action by shutting down and restarting your program. As mentioned previously, this behavior is configurable.

Network Configuration and UDP Network Discovery

By default, RabbitSys enabled boards are configured with DHCP for automatic network configuration. As long as your local area network provides a DHCP server, this feature works seamlessly.

RabbitSys also allows you to detect RabbitSys enabled devices on a local area network via UDP. This UDP discovery allows you to query your network for all of the RabbitSys enabled devices present. The interface is accessible via Dynamic C. Through the discovery mechanism, you can retrieve the board's network address and immediately start communicating with it.

The combination of DHCP and UDP Discovery give you immediate access to the board as soon as you attach it to your network.

Dynamic DNS

Dynamic DNS allows the IP address of the RabbitSys-enabled target to change and still be accessible using a static domain name. If your target board is acting as a web server, outside users do not have to know the changed IP address in order to contact the server.

Remote Network Configuration

Remote network configuration gives you the ability to change the board's setting in a reliable way. Using the remote console, you configure your network settings. You then tell the board to test the new settings. The board then waits for a remote connection to be made to the console and sends out an e-mail. If neither of these verifying events occurs within 5 minutes, then the controller falls back to the previous network settings.

Hardware Independent Drivers

RabbitSys enabled boards preload the drivers for core components such as Ethernet and parallel flash. This feature protects you from the volatile component markets that manufacture such devices. Each driver provides a clear, device independent interface for configuration and use. This interface allows the underlying parts to be changed without having to redesign, much less recompile your program.

Interrupt Protection

By default, the system handles interrupts. However, your program can register an interrupt service routine (ISR). The Rabbit microprocessor supports 4 interrupts levels. In user mode, your program can operate at

priority 0, 1 or 2; only the system operates at priority 3. The hardware downgrades priority 3 in user mode to priority 2 to ensure the system can gain control if needed.

When running in user mode, the Rabbit microprocessor will generate non-maskable interrupts for memory access violations, stack violations, or when executing an intrusion detection instruction (IDET). These interrupts help to ensure reliable operation of the system as a whole and are indications that the user program is behaving poorly.

For interrupts that need rapid response, your program may request direct access by registering your own interrupt service routine. RabbitSys only revokes direct access to the interrupt in the event that your program fails. By registering a shutdown callback function, you ensure that the shutdown behavior leaves the system in a stable state.

Hardware-Enabled Protection

Starting with the Rabbit 3000A, the Rabbit family of microprocessors share many advanced hardware features, several of which provide the foundation for RabbitSys.

System/User Mode

RabbitSys makes full use of the the Rabbit microprocessor's system/user mode of operation. Using this two-tiered mode, RabbitSys provides data memory protection for both global and local data as well as prevents accidental execution of system code. These features are provided by system memory block protection (in blocks of 64K and 4K), system stack protection, and a system intrusion detection instruction (IDET). Using these features, RabbitSys protects your program from critical errors that may occur in your code by firing a system-level interrupt when such a violation occurs.

Memory Protection

The Rabbit microprocessor can inhibit writes to physical memory. Each 64K memory blocks can be individually protected; two of these blocks can be subdivided and protected with a granularity of 4 KB. When a write attempts to access protected memory, a write protection interrupt occurs.

Stack overflow and underflow are also detected by the Rabbit microprocessor. Low and high stack limits can be set on 256-byte boundaries, which if crossed, could trigger a stack violation interrupt.

Secondary Watchdog

RabbitSys also uses the secondary watchdog interrupt for finer control of the program's execution. As opposed to forcing a hard reset of the system, the secondary watchdog gives RabbitSys the ability to gracefully shutdown and restart your program, ensuring that the program is not left in a bad state due to a primary watchdog timeout. When the secondary watchdog is fired, the event manager takes over and calls shutdown code you associated with the shutdown event and then performs a software reset. If your program is so severely corrupted that even its shutdown routines do not respond, the primary watchdog forces a hard reset of your program.

Flexible I/O Protection

RabbitSys also protects the I/O devices needed to ensure connectivity without sacrificing execution speed. RabbitSys achieves this by transferring control of all non-critical devices over to your program. This feature maximizes that amount of on-board I/O that your program can access directly. The I/O API provides simple, uniform system calls to access attached devices and peripherals that your program controls.

Processor resources are allocated in many different configurations, depending on the board type. Port enable registers protect access to all parallel and serial ports. I/O register permissions are set by RabbitSys to ensure availability to resources like Ethernet, DMA, and other high priority I/O. Your program can request direct access to these protected resources.

RabbitSys provides three levels of protection for I/O registers and these permissions are configured when the system ID block is programmed. The first level of protection allows your program to access a set of I/O registers (e.g., serial port A registers) directly in code. This level is the most permissive and is the default for devices not critical to RabbitSys operation. The second level of permission requires that access to the device go through system calls, but direct access is forbidden. The second level allows RabbitSys to protect I/O pins that are shared with system critical ones like Ethernet. The third level of permission does not allow any access to the device and is necessary to protect system only registers for hardware like the memory management unit.

For devices with level 1 or level 2 permission that generate interrupts, RabbitSys also allows your program to register its own interrupt service routines. This feature gives you the ability to write time critical handling code for real-time applications.

The following table summarizes these permission levels:

Table 1. RabbitSys Permission Levels

Device Protection	System Owned	User access via syscall	User Owned	Example
Level 1	Yes	Yes	Yes	Serial Port B
Level 2	Yes	Yes	No	Parallel Port E with Ethernet
Level 3	Yes	No	No	MMU registers, Write protect registers

Integration with Dynamic C

Dynamic C integrates seamlessly with RabbitSys, and includes a new mode of compilation that targets user programs for RabbitSys. The Dynamic C IDE will provide functionality that will download a RabbitSys binary to a target board. Once a target has RabbitSys installed, whether from the factory or custom loaded by a user, Dynamic C will communicate directly with RabbitSys and use RabbitSys functionality to download, execute, and debug user-level programs.

Summary

As the world becomes more connected, the market for Internet-enabled devices grows and creates new opportunities and challenges. You can gain a competitive edge by using RabbitSys in your application. Not only will productivity increase during development, which will decrease time to market, the assurance that your system works reliably will let you rest at ease.

Z-World, Inc.

2900 Spafford Street
Davis, California 95616-6800
USA

Telephone: (530) 757-3737
Fax: (530) 757-3792

www.zworld.com

Rabbit Semiconductor

2932 Spafford Street
Davis, California 95616-6800
USA

Telephone: (530) 757-8400
Fax: (530) 757-8402

www.rabbitsemiconductor.com