ZWORLD WHITE paper

# Z-World's Real-Time Software Philosophy

*Z-World's real-time software philosophy harmonizes with the company's software development and hardware philosophies to provide embedded-system engineers a sophisticated software development system. This system combines remote debugging with simple, powerful, and flexible software constructs ideally suited to Z-World's line of single-board controllers.*

## Multitasking and Multiprocessing

In a single-processor, multitasking system, more than one software task appears to be executing simultaneously. In reality, a single processor can only execute one instruction at a time, so the parallel tasks interleave their execution in such a way that they seem to execute in parallel. Although the mechanics of multitasking may actually decrease processor throughput, the benefits of multitasking are significant enough to offset this use of resources.

However, programming for a multitasking environment is difficult. Poorly designed or misapplied multitasking systems incur so much overhead that the system collapses, failing to answer critical interrupts in a timely manner. Luckily, multitasking-control software can usually take advantage of natural lulls in each task's sequence of operation to allow other tasks to execute.

Although "multiprocessing" refers to hardware and "multitasking" to software, the two have much in common and often go together. "Multiprocessing" means, literally, having multiple processors. Software can execute across multiple processors in many different fashions.

For example, an operating system running multiple programs could assign each currently running program its own processor. The programs would run more or less independently of each other. However, the programs might be sharing system resources such as a network, a printer, or mass storage. The operating system would have to handle multiple, asynchronous requests from the programs to use these facilities, and would have to make sure that the programs' requests did not consist of conflicting commands for peripheral devices or corrupt each others' data.

In a different configuration, a single program could distribute multiple, separable "threads" of execution across several processors. For example, a data-acquisition program might collect data from many independent sources. Eventually, the program will have to synchronize the results of the threads, generating a composite report. To do this, the program might have to pause one thread until another thread finishes some vital operation, supplying a needed intermediate result for the first thread.

### Software Models The Outside World

An embedded controller often connects to more than one external device. A multitasking approach allows software engineers to devise a solution for each device in isolation, without having to consider the requirements of all the devices at the same time. In other words, multitasking allows software engineers to partition their software along the organizational lines of the system to be controlled, constructing an abstract software model of the real world.

## Z-World's Multitasking Systems

Although Z-World's controllers can run programs of up to 20,000 lines, they are typically used for relatively simple applications. A full-featured real-time operating system (RTOS), then, is often an unnecessary burden. So that software engineers can design a system that precisely meets their needs, whether straightforward or complex, Z-World offers three types of multitasking:

- Cooperative
- Preemptive
- Simplified preemptive

### *Preemptive Multitasking*

Preemption means that some top-priority agency—usually a timer interrupt or a supervisory task (kernel)—takes control from the task currently running, giving control of the processor to another, higher-priority task. The interrupted task has no control over when preempting may take place and no ability to stop the interrupt.

A preemptive multitasking system needs, at a minimum, a kernel to stop and start tasks. The kernel usually uses a timer interrupt from on-board timing hardware to preempt the currently active task. A kernel can also take control of the processor in response to an asynchronous interrupt from the outside world and, after determining the nature of the interrupt, decide to switch tasks.

Since a task can be interrupted at any point, preemptive multitasking is well-suited for applications that require precise timing or high speeds.

Because each task does not know when preemption may take place, software engineers must be careful when tasks share common resources such as variables, displays, storage devices, and communications lines. Cooperation and coordination among preemptable tasks are major programming concerns.

The RTK (real-time kernel), one of two kernels shipped with Z-World's Dynamic C, supports preemptive multitasking. The RTK supports prioritized preemption; only a task of higher priority than the one currently executing can interrupt. Software engineers may create as many priority levels as desired when using the RTK. The RTK also has a **`suspend`** function, with which a high-priority task voluntarily suspends itself (for a specified length of time or until awakened by other tasks) and lets lower-priority tasks execute.

### *Cooperative Multitasking*

Cooperative multitasking is the simplest, fastest, lowest-overhead multitasking possible. Cooperative multitasking has low overhead because no RTK or "supervisor" task is needed. Under cooperative multitasking, each task voluntarily gives up control so other tasks can execute.

*Cooperative multitasking is the simplest, fastest, lowest-overhead multitasking possible*

Cooperative multitasking has several advantages over other types of multitasking:

- The designer has explicit control of the points at which a task begins and ends logical subsections of its overall job.
- Programmers have complete, explicit control of tasks' interactions.
- Tasks communicate more easily.
- Programming is simplified.
- Errors in code are less likely, and are easier to isolate when they do occur.
- Errors usually degrade performance rather than halting execution.
- Indeterminate interrupt latency is lower.

Cooperative multitasking does require some tradeoffs. Compared to preemptive multitasking, cooperative multitasking's overall performance is slower, making it inappropriate for fine-tuned applications requiring high speed or exact timing.

## The `costate` Function

Z-World's Dynamic C extension **`costate`** supports cooperative multitasking. The name derives from the well-known "co-routine," a concept developed by IBM decades ago for mainframe computers. Co-routine allows some routines to stay in memory, retaining their "current state" between periods of execution instead of being reinitialized—and perhaps reloaded—each time they are called.

A costatement is simply a cooperative task. Cooperative tasks run until they encounter an explicit command in their code to suspend operation. As shown in Figure 1, this suspension may be temporary (allowing other costatements time to execute) or may continue until some condition is met.

Because cooperative-multitasking systems have no supervisor that automatically deals out processor time among the tasks, the software engineer must carefully program in such "wait" and "pause" commands to maximize a cooperative-multitasking system's performance.

Since cooperative tasks interrupt themselves at convenient points in their execution, cooperative multitasking is, in a sense, synchronous. A cooperative-multitasking system is thus subject to fewer problems arising from asynchronous interruptions than is a preemptive-multitasking system.

Since cooperative-multitasking systems need only a small amount of system code to save and restore tasks, they can usually switch between tasks more quickly than preemptive tasks can.

### *Simplified Preemptive Multitasking*

Z-World's simplified preemptive multitasking combines the relative simplicity of cooperative multitasking with the preemptive ability of a RTOS.

*Z-World's simplified preemptive multitasking combines the relative simplicity of cooperative multitasking with the preemptive ability of a RTOS*

Z-World ships a simplified real-time kernel (SRTK) with Dynamic C. Like the full RTK, the simplified RTK is prioritized and preemptive. However, it has only three levels of priority: high, low, and background. The high-priority task executes at 25-millisecond intervals, the low-priority task executes at 100-millisecond intervals, and background tasks execute only when no other tasks are executing. Because of the SRTK's fixed properties, it is compact and easy to use. The SRTK can be combined with cooperative multitasking.

```
main () { ...
   costatement A { ...
        yield
        ... }
   costatement B { ...
        yield
        ... }
   costatement C { ...
        yield
        ... }
   }
```

```
main () { ...
   costatement A { ...
        waitfor (<event>)
        ... }
   costatement B { ...
        yield
        ... }
   }
```

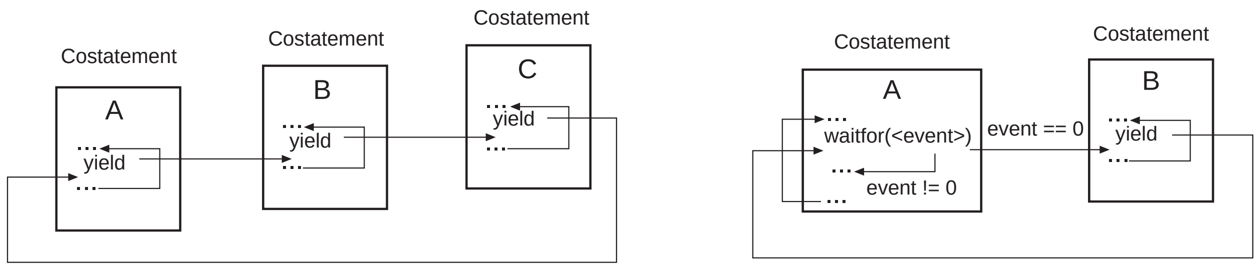

***Figure 1. Examples of Cooperative Multitasking***

## Programming Considerations

### Corrupting Shared, "Non-Atomic" Operations

Real-time programming is perhaps the hardest kind of programming to do. Problems are difficult to detect, difficult to diagnose, and difficult to fix. Verifying that a real-time system will survive all possible combinations of inputs, insults, and outages in the real world is a challenging task.

For example, in a real-time system with concurrent tasks sharing data, subtle problems can occur with variables that are stored and fetched in a "non-atomic" manner. Here "atomic" refers to an "atom" of code; that is, a segment of code that executes from start to finish as an uninterruptable unit.

If fetching or storing a variable takes several instructions, it is possible that an interrupt could occur during this sequence of instructions. Consider a floating-point variable that occupies four bytes. Storing a value in this multi-byte variable typically takes two or more instructions. If an interrupt occurs between these store instructions, and a new task of higher priority that uses the same variable takes over, then the new task will see a corrupted value, partly old and partly new.

Z-World's Dynamic C provides a C keyword, **`shared`**, that declares a variable as "atomic." That is, Dynamic C will automatically disable interrupts during stores and fetches of **`shared`** variables.

Disabling interrupts during non-atomic operations cures the problem of data corruption but at the expense of increased interrupt latency—an important consideration for a embedded controller that must respond promptly and predictably to asynchronous interrupts.

### Logical Operators

The nature of C introduces some built-in problems. For example, a potential problem arises because the logical operators **`||`** (Boolean OR) and **`&&`** (Boolean AND) are subject to "short-circuit" evaluation rules. In an OR expression, such as **`a||b||c`**, remaining terms are not evaluated if any of the preceding terms are true. In an AND expression, such as **`a&&b&&c`**, remaining terms are not evaluated if any of the preceding terms are false.

Consider the following code fragment that uses the Z-World **`DelayMs`** function, which delays a task for the specified number of milliseconds after **`DelayMs`** is called. Because of the short-circuit evaluation rules, an expression such as

```
waitfor( test && DelayMs(50L) );
```

waits until **`test`** becomes true and *then* waits 50 milliseconds more. The call to **`DelayMs`**, in other words, will not happen while **`test`** is false.

The code executes very differently, however, if the order of the terms in the expression is reversed:

```
waitfor( DelayMs(50L) && test );
```

The expression now calls **`DelayMs`**, beginning the 50-millisecond delay, and then checks to see if **`test`** is true. If **`test`** becomes true after the 50-millisecond delay has finished, the program will immediately move to the next expression (rather than calling **`DelayMs`** only *after* **`test`** is true, as in the first example). Software engineers can avoid short-circuit evaluation problems by using the bitwise OR and AND operators (**`|`** and **`&`**).

## Multiprocessing

Some applications may require more input and output ports than a single control computer can provide, but be too tightly integrated to be controlled by two separate programs running on two separate controllers. Or, in other cases, running numerous input/output lines through a large machine to and from a single controller might be physically unsafe or impractical.

Z-World's simple approach to multiprocessing relieves the software engineers of the burden of coordinating multiple, multitasking programs running on multiple processors. Z-World implements multiprocessing as a simple master-slave network of control computers linked by an inexpensive, twisted-pair RS-485 multidrop network.

The software engineer writes a single control program for the single master controller. This program uses a single set of I/O routines to control the input and output ports of both the master controllers and all the slave controllers. The slaves all run the same factory-supplied program from EPROM. The software engineer does not need to program the slave computers.

Further, each controller, whether a master or a slave, can have Z-World expansion boards attached. Again, the program running on the master controller uses a single set of function calls to operate all of the network's expansion boards.

These add-on boards could theoretically provide an additional 48 form-C relay contacts, 16 digital-to-analog (DAC) output channels, 16 conditioned analog-to-digital (ADC) input channels, 28 unconditioned analog-to-digital input channels, 32 digital-input channels, and 12 high-voltage digital-output channels.

In practice, because of physical limitations such as current draw and noise, no more than four expansion boards can be connected to each controller (see Figure 2).

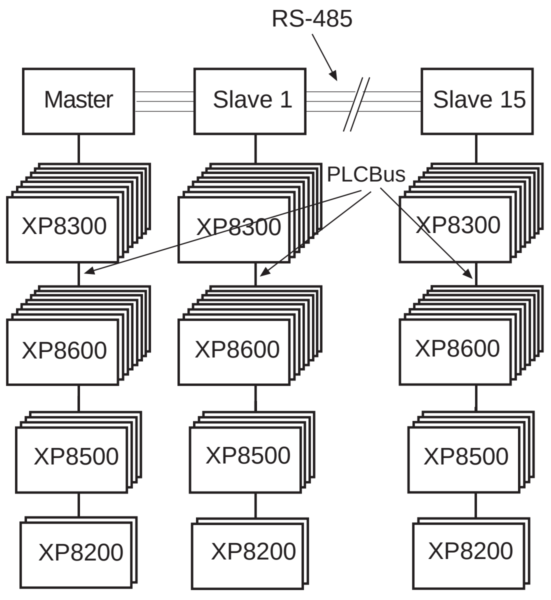

***Figure 2. Theoretical Maximum Short-Range System Expansion***

### *Short-Range And Long-Range Networks*

Z-World offers sets of function calls for short-range and long-range versions of its master-slave, multiprocessing network.

The short-range network links a maximum of 16 control computers (1 master and 15 slaves). The cable length must be short enough and the environment electrically clean enough so that no communication errors occur. The short-range network is faster than the long-range network because it does no error checking or error recovery. In the event of an error, the control computer (master or slave) detecting the error simply resets.

The long-range network can link more than 16 control computers. Because the long-range network attempts to retransmit garbled communication and otherwise recover from errors without resetting, it runs more slowly than does the short-range network.

## Z-World's Real-Time Performance Enhancements

Z-World's RTKs and the `costate` facility are only part of Z-World's real-time software philosophy; other software facilities (supplied as source code) and hardware devices (such as the watchdog timer, battery backup, and nonvolatile storage) are integral parts of Z-World's real-time system.

### *The Virtual Driver*

The virtual driver is a set of functions providing the following services:

- Periodic timer interrupts
- Second, millisecond, and tick timers
- Synchronization of the second timer with the real-time clock
- Virtual watchdog timers
- Periodic scheduling for real time kernels
- The "fastcall" execution thread
- Global initialization

### *"Fastcall" Task*

Both the RTK and the SRTK support a "fastcall" task, available with either kernel or when no kernel is running, which can execute as often as 1280 times per second. The fastcall task preempts all other tasks.

### *Global Initialization*

Initializing a control program is really an engineering task because such initializations determine the "zero state" of a complex hardware/software system. Consequently, the initialization facilities of conventional compilers generally do not provide enough programming power for embedded systems. The global initialization feature of Dynamic C executes sections in the program declared with a special keyword. This construct provides a formal, explicit means of initializing an entire embedded system. The initialization statements can be complex C code with loops, branches, and function calls, or simple assignments—whatever is required.

## Interrupts, Failure, and Recovery

Simple applications written for a PC or a UNIX workstation do not have to handle interrupts because their operating systems already handle interrupts from the standard devices usually interfaced to such programs: a keyboard, serial interfaces, disk drives, and so on.

Embedded controllers running stand-alone, real-time programs often have *no* operating system. Embedded-system designers also have to contend with a huge range of possible peripheral devices; virtually any device in the world can interface with a control computer. Furthermore, real-time applications require far more robust responses to interrupts and errors than do workstations.

In many cases, a workstation or PC operating system responds to problems with an error message and simply halts (or crashes), leaving the workstation's operator to fix the problem. Embedded controllers generally do not have operators in attendance and so must handle problems themselves.

Even a "perfect" program may still crash because of conditions that are beyond the control of the software engineer and Dynamic C. For example, blackouts, brownouts, and "spikes" on the incoming power lines can put the controller in an undetermined state. Therefore, crash detection and recovery are important issues for all embedded applications. As a rule, software engineers can expect that the logic associated with detecting and handling errors will be a substantial portion of an embedded program's code.

Z-World has derived methods and support routines for robust embedded systems. Dynamic C makes writing interrupt and error routines as easy as writing any other C function.

### *Software Failures*

While software cannot detect all types of software-related failures, certain software-related failures can be actively verified. For example, if a software engineer compiles a program with Dynamic C debugging options activated, the compiler inserts code to perform the following routine:

- Check stack integrity
- Check validity of pointers
- Check array-bounds overflow

If such errors occur, the debugging code calls a function established by the software engineer to invoke the proper error handler for each type of error.

### *Protected-Variable Recovery*

Z-World provides a language construct and support routines for low-level recovery of important variables. Recovering certain data is particularly important if the application uses nonvolatile memory to store log files because the log files (and the associated data structures) must persist over crashes.

Software engineers may declare a variable "protected." When a variable, array, or structure is protected, the compiler generates code that will perform the following routine when storing a value in the variable:

- Make a backup copy of the variable
- Set a "flag" indicating the backup copy is valid
- Store the variable
- Reset the flag

If system fails during the write to the variable, the Z-World recovery function will check the flag and reestablish the correct version.

## *Hardware Failures*

Software often cannot identify or handle hardware-related failures because the software itself runs on the failing hardware. Generally, the only verifiable hardware failure is power failure. Special hardware on Z-World's controllers detects low input voltage before the regulated, on-board power drops out. Upon detecting an impending power failure, this hardware causes an non-maskable interrupt (NMI) that the processor cannot ignore, and the processor executes the handler for the NMI. The program usually has only a few milliseconds or less between the NMI and complete power failure, however. Consequently, the NMI handler often can only store a handful of critical system parameters in some kind of nonvolatile memory. Z-World control computers offer, variously, battery-backed RAM, EEPROM, or flash EPROM as nonvolatile storage.

### Hardware Watchdog Timer

A hardware watchdog timer will reset the a control computer unless it repeatedly receives a signal from the software within a specified time—about 1.6 seconds for Z-World equipment. A correctly functioning program will periodically reset the watchdog timer to keep the processor from resetting. The system assumes that if the watchdog times out, a software failure must have occurred, and so it resets the system to allow the program to reinitialize and attempt a recovery.

### Reset and Super-Reset

A reset caused by a power failure, software failure, operator intervention, or watchdog-timer timeout causes the program to execute a reset routine.

Z-World strongly suggests that software engineers distinguish between a program's initialization during startup and a program's recovery from a genuine, run-time failure. If the program is simply starting up, the program must initialize its variables to the "zero" state. When the program is recovering from failure, however, it must restore critical system parameters saved from before the failure so that execution can resume execution at the point of failure.

For this purpose, then, the Dynamic C compiler places a time stamp in each program. The time stamp is passed as an argument to **`main()`**., which can use the time stamp to determine if a program is restarting or responding to a crash. Dynamic C has separate facilities for a startup reset to the system's zero state and a "Super Reset" that allows a program to resume where it left off.

### Enhanced Power-Failure Handler

The circuitry in Z-World's controllers that detects power failures goes beyond the industry-standard handbook design. This enhanced circuitry can distinguish between a crash resulting from a power failure (blackout or brownout) and one resulting from a watchdog timeout. This circuitry allows a Z-World controller to withstand multiple, rapid power-line insults that would overload the interrupt handler of conventional designs.

Z-World Corporate Headquarters
2900 Spafford Street, Davis, California 95616 USA
530.575.3737 • Fax: 530.753.5141

http://www.zworld.com