



**ZEIDMAN  
TECHNOLOGIES**

# **SynthOS™ White Paper**

*By Bob Zeidman*

Bob Zeidman, President

Zeidman Technologies, Inc.  
15565 Swiss Creek Lane  
Cupertino, CA 95014

e-mail: [bob@zeidman.biz](mailto:bob@zeidman.biz)

Tel (408) 741-5809  
Fax (408) 741-5231  
[www.zeidman.biz](http://www.zeidman.biz)

## THE PROBLEM

Microprocessors are showing up in almost every imaginable piece of hardware, from computer monitors to network routers to automobiles to intelligent household appliances. This proliferation means that more software must be developed by more programmers. These embedded systems programmers require a real-time operating system (RTOS) to control the various tasks of the software. Over half of all embedded software developers write their own operating systems. Why is this so? Mainly because the off-the-shelf operating systems are overly complex for the majority of embedded systems being developed. These operating systems require a large memory space, which translates to higher costs for larger memory devices. They also require a steep learning curve and specialized expertise.

An embedded programmer can purchase object code for an operating system, but that code is difficult to debug when problems occur. Or the programmer can purchase source code but then must wade through tens of thousands of lines of code or more, written by someone else, to find problems when they occur. While some operating systems can be purchased outright, others require that royalties be paid for every product shipped. Royalties of this type can greatly increase the cost of a product.

Embedded Linux was heralded several years ago as a perfect solution to these problems. However, the hype has turned out to be greater than the reality. The code size is very large and requires lots of memory. Linux is designed for high-end processors with memory management units, forcing applications to use complex processors that use more power and have a higher cost than is otherwise needed. Linux was not developed as a real-time OS, meaning that the task-switching overhead is too great for many applications. Also, the code has been developed by many different people, making debugging a very difficult task.

Programmers by their nature tend to prefer to write their own code so that they can maintain it more easily. There is a need for a tool that allows embedded programmers to generate operating systems automatically and at a high level, but that still gives them full visibility to the code and full control over its development.

## SYNTHOS

SynthOS is such a tool. This revolutionary tool allows a programmer to synthesize a real-time operating system. The programmer writes code in C and adds simple SynthOS statements to the code to specify inter-task communications and operating system parameters. The output code, after synthesis, is still C code. By generating code in the same language as the input code, SynthOS allows programmers to use all of their current tools — compilers, debuggers, interpreters, emulators, etc. — for executing and debugging the resulting synthesized code. SynthOS also optimizes code and performs checking to ensure that race conditions cannot occur and that tasks have the correct priorities and frequencies.

## TECHNOLOGY

The patented technology from Zeidman Technologies that is used in SynthOS takes the concept of hardware synthesis and applies it to software. SynthOS allows the software engineer to write code for specific tasks. When one task needs to call another task, or wait for another task to complete, the programmer inserts a special primitive that looks like a C function call and that is recognized by SynthOS. The programmer also uses SynthOS to specify the parameters of each task, such as the task's priority and its period, and to specify the requirements of the operating system such as the scheduling algorithm to use. SynthOS is then run on all of the task code. SynthOS creates the appropriate mutexes, semaphores, flags, message queues, and mailboxes for each task and inserts the appropriate code at the appropriate points in the task. SynthOS also creates the RTOS to schedule execution of the tasks. Two example results are illustrated in Figure 1. The code written by the programmer is represented by the yellow areas. The code generated automatically by SynthOS is represented by the blue areas. Note that each task is mostly written by the programmer, but SynthOS inserts the RTOS data structure manipulation code and the RTOS communication code into each task. SynthOS also generates the RTOS that controls execution of each task.

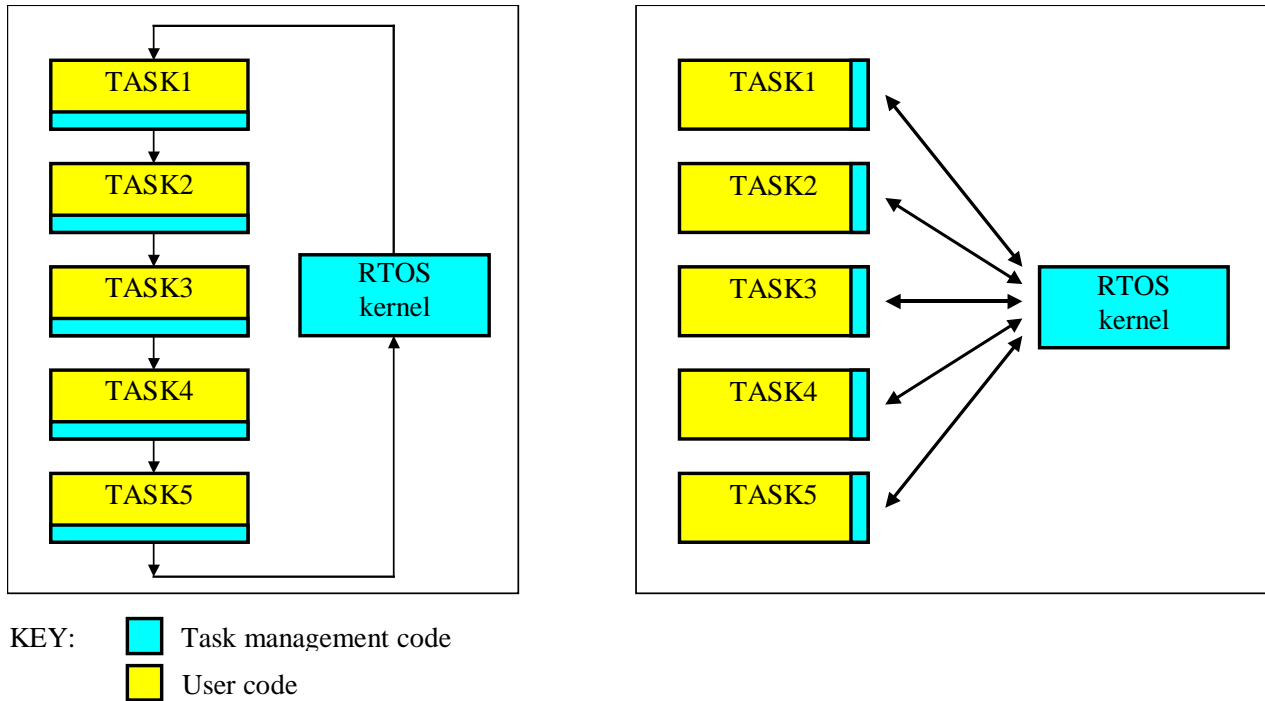


Figure 1. SynthOS code generation

SynthOS can be tuned to different processors to optimize code for these processors. SynthOS automates the process of creating the operating system so that the programmer can focus on writing the specific tasks for the device. Because the output of SynthOS is in the same language as the input, the programmer has complete visibility to everything that is going on in the operating system. All of the tools that are used to compile and debug the tasks can also be used to compile and debug the operating system.

## USER INTERFACE

SynthOS is very easy to set up and use. It is written in Java to run on any computer and has a simple command line interface. SynthOS is easily integrated into standard integrated development environments such as Eclipse and Cygwin.

## CASE STUDIES

Zeidman Technologies has performed a number of case studies for comparing the development costs and development times for using SynthOS versus other solutions in several projects as described below.

### ***Flat Panel Display***

Table 1 shows a detailed schedule for the development of firmware for a consumer flat panel display. This two-person project was a real project and the numbers for the actual schedule are accurate. The proprietary RTOS for this project was developed in-house. The SynthOS schedule is an estimate of how SynthOS could have been used to speed development and lower development costs. Using SynthOS, this flat panel would have had a savings of nearly \$43,000 in labor costs and would have gotten to market almost 10 weeks sooner.

## Flat panel display firmware development

Task	Actual Schedule		Using SynthOS Predicted Schedule	
	Man-weeks	Cost	Man-weeks	Cost
Writing firmware specification	6	\$13,846	4	\$9,231
Code cyclic executive	6	\$13,846	1	\$2,308
Code startup task	8	\$18,462	8	\$18,462
Simple interface driver tasks	6	\$13,846	5.4	\$12,462
Complex interface driver tasks	24	\$55,385	21.6	\$49,846
Simple video controller tasks	8	\$18,462	7.2	\$16,615
OSD controller tasks	8	\$18,462	7.2	\$16,615
Advanced video controller tasks	12	\$27,692	10.8	\$24,923
Internal tables for controlling video specs	2	\$4,615	2	\$4,615
NVRAM control task	2	\$4,615	1.8	\$4,154
Additional debug and integration	16.4	\$37,846	13.8	\$31,846
Code optimization	6	\$13,846	4	\$9,231
Documentation	4	\$9,231	3	\$6,923
<b>TOTAL</b>	<b>108.4</b>	<b>\$250,154</b>	<b>89.8</b>	<b>\$207,231</b>
<b>Cost savings</b>		<b>\$42,923</b>		
<b>Time-to-Market savings (weeks)</b>		<b>9.3</b>		

Table 1. Flat panel firmware development

### Zeidman Technologies vs. Wind River

Figure 2 shows the total cost of ownership for using the VxWorks RTOS from Wind River versus using SynthOS to synthesize an RTOS for an average Wind River customer. The Wind River customer pays royalties for each RTOS shipped with a product. The Zeidman Technologies customer pays for SynthOS and all of the code it generates is royalty free. For each of three years, the SynthOS TCO is about one fifth that of VxWorks.

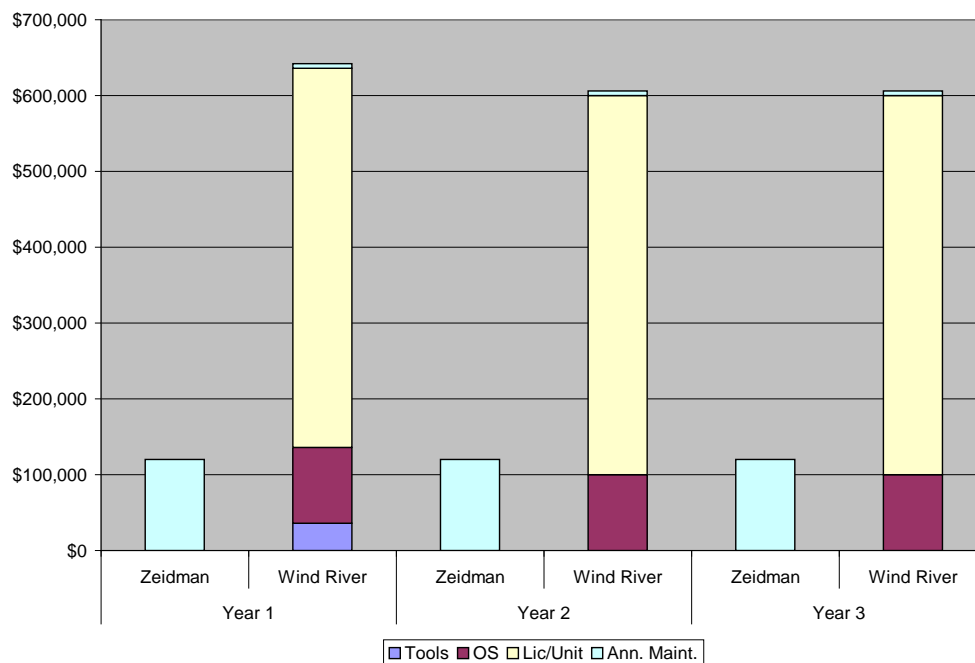


Figure 2. Three Year TCO for SynthOS vs. VxWorks

## PROJECTS

Zeidman Technologies has created several projects in order to demonstrate the usefulness of SynthOS and its advantages over off-the-shelf RTOSes. These projects are described below.

### ***Multitasking Web Server***

An Altera Cyclone EP1C20 FPGA with an embedded NIOS 32-bit soft processor was used to control a web server that was performing low-level hardware tasks while serving up Web pages. The development for this very first application of SynthOS took several weeks. The resulting RTOS kernel was about 3K bytes. This same code was re-synthesized and compiled for an 8-bit Cypress PSoC resulting in an RTOS of about 1.2K bytes.

### ***Multiprocessor “Hello World” Application***

A Xilinx Virtex II-Pro FPGA containing an embedded 32-bit PowerPC hard processor and two embedded 32-bit MicroBlaze soft processors each sent “hello world” to a single serial port. Each processor used an RTOS to control the serial port at different communication rates. The RTOS also controlled inter-processor communication to avoid race conditions. The PowerPC RTOS generated by SynthOS was about 1.5K bytes. Each MicroBlaze RTOS generated by SynthOS was less than 1K bytes.

### ***Heterogeneous Multiprocessing Robot Arm***

A Xilinx Virtex II-Pro FPGA containing an embedded 32-bit PowerPC hard processor and an embedded 32-bit MicroBlaze soft processor was used to control a serial port mouse and a multiple joint robot arm. Attempts to use uClinux, a small, embedded version of Linux promoted by Xilinx, was unsuccessful after several weeks of effort. SynthOS was able to create an RTOS for each processor in 3 days, most of which was spent setting up the workstation that ran SynthOS. The resulting RTOS for the PowerPC was 1.2K bytes while the RTOS for the MicroBlaze was less than 900 bytes.

### ***Lego Mindstorms Robot***

A Lego Mindstorms robot is shipped with 26 Kbytes of total memory. The operating system shipped with the robot uses about 22K bytes, leaving the user with only 4K bytes for applications. Many users substitute the widely available brickOS, which uses only about 11K bytes, leaving the user with 15K bytes for applications. SynthOS was synthesized for a Mindstorms robot and the resulting RTOS used only 2K bytes, leaving 24K bytes for user applications. SynthOS increased the user space by a factor of 6 and also supported more tasks and more global variables, allowing much more complexity for the applications.

### ***Comparison of RTOS Sizes***

From the above projects, summarized in Table 2, the sizes of RTOSes synthesized by SynthOS are compared to sizes of off-the-shelf RTOSes. As can be seen, in all cases the synthesized RTOS produced by SynthOS is significantly smaller than all others. The implication is that SynthOS allows systems to be implemented using fewer memory chips. SynthOS also allows systems to be integrated more easily into a single chip SOC.

RTOS	Processor (RTOS Size in Bytes)				
	Altera NIOS	Cypress PSoC	Lego RCX	Xilinx MicroBlaze	Xilinx PowerPC
<b>SynthOS</b>	3.0K	1.2K	2.0K	900	1.5K
<b>brickOS</b>			11.0K		
<b>Lego OS</b>			24.0K		
<b>Microtronix uClinux</b>				100K*	100K*
<b>MontaVista Linux</b>					500K**

\*based on estimates of compiled code

\*\*based on published numbers from the vendor

Table 2. Comparison of RTOS sizes

## **CONCLUSION**

SynthOS can greatly speed up the development time and testing time for any embedded system. The code produced by SynthOS is completely royalty free, making it very inexpensive over the life of the system. Also, the memory footprint required by the operating system is minimized because SynthOS only creates the code required for the various tasks – there is no unused or unwanted code. In many cases, system designers can port their software to a smaller, less expensive, lower power processor with SynthOS, because the optimized RTOS produced by SynthOS can run on a smaller processor than one required by an off-the-shelf RTOS.

SynthOS also has the unique ability to eliminate race conditions, a particularly difficult problem to isolate and debug in typical systems. This ability significantly reduces system debug time. Because of its ease of use, SynthOS makes multiprocessor systems on a chip a reality for the first time.

Software synthesis is the most significant development in software development since the compiler. SynthOS uses software synthesis to fulfill the promise of automatic code generation for low cost, small footprint RTOSes. By taking hardware synthesis as its model, SynthOS gives a programmer the flexibility and control over embedded system design that no off-the-shelf RTOS or programming development tool can offer, while allowing the programmer to work at a much higher level of abstraction.