KaOS

# Introduction [1]

It’s a real-time, multithreaded, preemptive operating system for the Atmel Mega32 microcontroller, which loads and executes programs from a Secure Digital or MMC card.

The system waits for you to insert a card and press the Reset button, at which point a program is loaded from the card and executed.

The system supports multiple threads, cross-thread messaging, dynamic memory allocation, dynamic thread creation, semaphores and much more.

# About the publishers

Created by two engineering students from Cornell University in New York, *Nick Clark* (Electrical and Computer Engineering major), and *Adam Liechty* (Computer Science major).

The reason they wrote this OS is the lack of a true OS for the Mega32 that dynamically loads programs from a card and that all the operating systems for the Mega32 have to be statically compiled in with a user program.

In contrast, kaOS waits for a card to be inserted and a reset button to be pressed, at which point a program is loaded from the card and executed. At any time, a new card with a new program can be inserted and run. Executing a new program doesn't require reprogramming the Atmel processor.

# High level design

The design of kaOS is broken up into two major components:

* the operating system itself
* the card reader and program loader

The card reader is accessed via the Atmel’s SPI interface by the program loader which places the program into flash memory. Once kaOS loads a program, it creates a thread for it and jumps to its main() method.

Programs can be written similar to a standard Atmel Mega32 program, except that it must include the “kaos.h” header file, which provides an interface to the threading and messaging calls to the OS.

The OS supports creation of up to 8 threads, which can be prioritized. Threads with the same priority are alternately preempted to give both equal processing time. kaOS also supports messaging between threads as a means of inter-thread communication.

# Hardware design

A circuit is built to connect the Atmel Mega32 to the SD/MMC card.

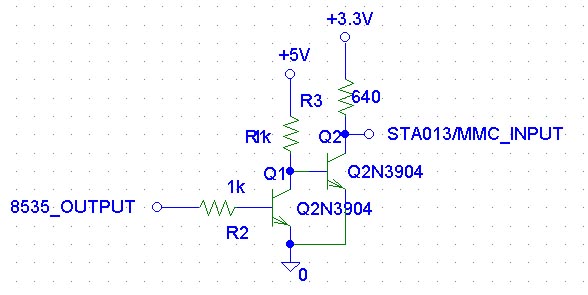
The inputs to the card are each fed through separate step-down circuits to decrease the voltage from the power supply's 5V to the 3.3V required by the card.

Figure 1 Stepdown circuit to change 5V from the Mega32 to the 3.3V required by the card

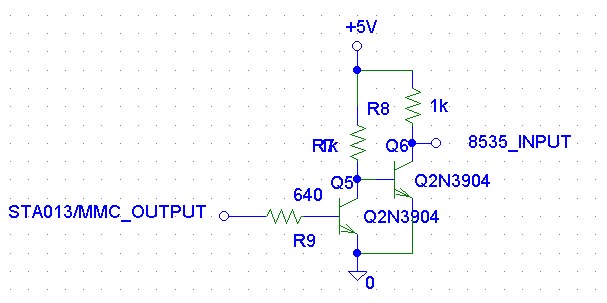
The output from the card is fed through a step-up circuit before going back to the STK500 board.

Figure 2 Step-up circuit to change 3.3V from the card to the 5V required by the Mega32

# Software design

The OS is put in the bootloader section of flash memory; as all programs on the Atmel have to be contained in the flash memory and only the bootloader can write to flash memory, this also helped in avoiding address conflicts with user programs; because the bootloader resides at the highest addresses of flash memory

### Card Programmer

The Atmel programs created in our class have been coded and compiled in CodeVision AVR Studio. CodeVision creates .ROM files when it makes a project, which are lists of 16-bit words. These files define what the contents of flash memory should be when the chip is programmed. Rather than programming the MCU, however, .ROM files are taken and used to program the SD card. A Win32 programmer and an Atmel programmer are written, separate from the OS, which programs the contents of a .ROM file to the SD card, also through the circuit built. The .ROM files consist of hundreds or thousands of lines like the following:

...

003E34:D5EF

003E35:B3BD

003E36:8E9F

...

Each line gives a program address and data value. The Win32 program parses this file and sends the length of the program followed by its binary contents through the PC's serial port to the STK500 (development board). The Atmel program then transfers what it receives on the STK500's serial port to the SD card. Data is simply written to the card sequentially. No file system is used.

### Program Loader

When kaOS boots, it begins waiting for an SD card and a reset signal. At this point, it begins reading the contents of the card into small RAM buffers, which are subsequently transferred to flash memory.

Because RAM is limited, some repurposing of memory is done when transitioning between loading a program and executing it under the OS. The buffers used to read data from the card overlap with OS data structures that are not used until after the program is loaded. Since loading and executing never overlap, we can use the same memory for both.

Once the program is loaded, kaOS creates and runs a thread that uses the address of the program's main() function as its starting point. User programs can then spawn other threads from their main() function. This is where the operating system takes over.

### Operating System

* Preemptive multithreaded operating system.
* The context switcher was borrowed from aOS.

#### OS Block Diagram

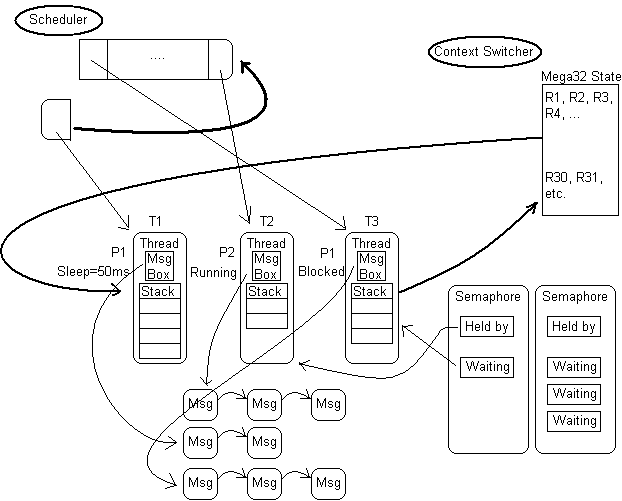


Figure 3 Block diagram of the major components of KaOS

Each thread has: an ID, priority, state, sleep duration (if any), a message box, and its own stack.

Stack: The stack size is defined by the user program and each thread can be created with a custom-sized stack. This allows for flexibility to use the Mega32's limited RAM effectively. Thus if some threads need large stacks, while others need little stack space, kaOS allows user programs to create threads tailored to these needs.

The header file, “kaos.h”, fixes a block of memory addresses for use as OS variables.

Inter-thread communication: To facilitate inter-thread communication, each thread has its own mailbox. A thread can pass a message to any other thread by calling SendMessage() with a pointer to the destination thread and a void pointer to any type of message structure. The mailboxes are implemented as a queue of messages for each thread as well as a queue of free message structs.

Synchronization: Semaphores are a necessity for synchronization in any OS. kaOS supports creation, waiting and signaling of semaphores as would be expected of an OS. They are implemented as a list of active semaphores and a list of free structs. Each semaphore holds a pointer to the threads waiting on it. When a semaphore is signaled, a waiting thread is removed from this list and reactivated.

Scheduler: The scheduler for kaOS allows priorities between 0 and 255;

* Higher priority threads always run unless they sleep.
* Equal-priority threads are swapped back and forth by the scheduler.

Mega32's Timer0 is used to call the scheduler every 10ms and schedule a new thread if necessary. The scheduler keeps all active threads in a queue. Each time the scheduler runs, it finds the highest priority active thread, choosing the least-recently scheduled in case of a tie. It removes it from the queue, schedules it, and places it at the end of the line.

# References

|  |  |
| --- | --- |
| [1] | A. L. Nick Clark, "People.ece.cornell.edu," Cornell University, 2005. [Online]. Available: http://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2005/acl34/kaos/index.html#highlevel. [Accessed 8 February 2016]. |

# Figures

[Figure 1 Stepdown circuit to change 5V from the Mega32 to the 3.3V required by the card 2](#_Toc442728641)

[Figure 2 Step-up circuit to change 3.3V from the card to the 5V required by the Mega32 2](#_Toc442728642)

[Figure 3 Block diagram of the major components of KaOS 4](#_Toc442728643)

# Note:

The source code of the KaOS Operating System can be found on the same link found in references session, including the KaOS API and the SD Application Loader.