Mark3 Realtime Kernel

Generated by Doxygen 1.8.1.2

Tue Feb 5 2013 20:29:38

Contents

Chapter 1

The Mark3 Realtime Kernel

```
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/ __/|__|_||__| __\  __||__| __\  __||__| __\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
```

The Mark3 Realtime Kernel is a completely free, open-source, real-time operating system aimed at bringing multi-tasking to microcontroller systems without MMUs.

It uses modern programming languages and concepts (it's written entirely in C++) to minimize code duplication, and its object-oriented design enhances readibility. The API is simple - there are only six functions required to set up the kernel, initialize threads, and start the scheduler.

The source is fully-documented with example code provided to illustrate concepts. The result is a performant RTOS, which is easy to read, easy to understand, and easy to extend to fit your needs.

But Mark3 is bigger than just a real-time kernel, it also contains a number of class-leading features:

- Device driver HAL which provides a meaningful abstraction around device-specific peripherals.
- Capable recursive-make driven build system which can be used to build all libraries, examples, tests, and documentation for any number of targets from the command-line.
- Graphics and UI code designed to simplify the implementation of systems using displays, keypads, joysticks, and touchscreens
- Standards-based custom communications protocol used to simplify the creation of host tools
- A bulletproof, well-documented bootloader for AVR microcontrollers

Chapter 2

Preface

2.1 Who should read this

As the cover clearly states, this is a book about the Mark3 real-time kernel. I assume that if you're reading this book you have an interest in some, if not all, of the following subjects:

- Embedded systems
- Real-time systems
- Operating system kernel design

And if you're interested in those topics, you're likely familiar with C and C++ and the more you know, the easier you'll find this book to read. And if C++ scares you, and you don't like embedded, real-time systems, you're probably looking for another book. If you're unfamiliar with RTOS fundamentals, I highly suggest searching through the vast amount of RTOS-related articles on the internet to familiarize yourself with the concepts.

2.2 Why Mark3?

My first job after graduating from university in 2005 was with a small company that had a very old-school, low-budget philosophy when it came to software development. Every make-or-buy decision ended with "make" when it came to tools. It was the kind of environment where vendors cost us money, but manpower was free. In retrospect, we didn't have a ton of business during the time that I worked there, and that may have had something to do with the fact that we were constantly K_SHORT on ready cash for things we could code ourselves.

Early on, I asked why we didn't use industry-standard tools - like JTAG debuggers or IDEs. One senior engineer scoffed that debuggers were tools for wimps - and something that a good programmer should be able to do without. After all - we had serial ports, GPIOs, and a bi-color LED on our boards. Since these were built into the hardware, they didn't cost us a thing. We also had a single software "build" server that took 5 minutes to build a 32k binary on its best days, so when we had to debug code, it was a painful process of trial and error, with lots of Youtube between iterations. We complained that tens of thousands of dollars of productivity was being flushed away that could have been solved by implementing a proper build server - and while we eventually got our wish, it took far more time than it should have.

Needless to say, software development was painful at that company. We made life hard on ourselves purely out of pride, and for the right to say that we walked "up-hills both ways through 3 feet of snow, everyday". Our code was tied ever-so-tightly to our hardware platform, and the system code was indistinguishable from the application. While we didn't use an RTOS, we had effectively implemented a 3-priority threading scheme using a carefully designed interrupt nesting scheme with event flags and a while(1) superloop running as a background thread. Nothing was abstracted, and the code was always optimized for the platform, presumably in an effort to save on code size and wasted cycles. I asked why we didn't use an RTOS in any of our systems and received dismissive scoffs - the overhead from thread switching and maintaining multiple threads could not be tolerated in our systems according to our chief engineers. In retrospect, our ad-hoc system was likely as large as my smallest kernel, and had just as much context switching (althrough it was hidden by the compiler).

And every time a new iteration of our product was developed, the firmware took far too K_LONG to bring up, because the algorithms and data structures had to be re-tooled to work with the peripherals and sensors attached to the new boards. We worked very hard in an attempt to reinvent the wheel, all in the name of producing "efficient" code.

Regardless, I learned a lot about software development.

Most important, I learned that good design is the key to good software; and good design doesn't have to come at a price. In all but the smallest of projects, the well-designed, well-abstracted code is not only more portable, but it's usually smaller, easier to read, and easier to reuse.

Also, since we had all the time in the world to invest in developing our own tools, I gained a lot of experience building them, and making use of good, free PC tools that could be used to develop and debug a large portion of our code. I ended up writing PC-based device and peripheral simulators, state-machine frameworks, and abstractions for our horrible ad-hoc system code. At the end of the day, I had developed enough tools that I could solve a lot of our development problems without having to re-inventing the wheel at each turn. Gaining a background in how these tools worked gave me a better understanding of how to use them - making me more productive at the jobs that I've had since.

I am convinced that designing good software takes honest effort up-front, and that good application code cannot be written unless it is based on a solid framework. Just as the wise man builds his house on rocks, and not on sand, wise developers write applications based on a well-defined platforms. And while you can probably build a house using nothing but a hammer and sheer will, you can certainly build one a lot faster with all the right tools.

This conviction lead me to development my first RTOS kernel in 2009 - FunkOS. It is a small, yet surprisingly full-featured kernel. It has all the basics (semaphores, mutexes, round-robin and preemptive scheduling), and some pretty advanced features as well (device drivers and other middleware). However, it had two major problems - it doesn't scale well, and it doesn't support many devices.

While I had modest success with this kernel (it has been featured on some blogs, and still gets around 125 downloads a month), it was nothing like the success of other RTOS kernels like uC/OS-II and FreeRTOS. To be honest, as a one-man show, I just don't have the resources to support all of the devices, toolchains, and evaluation boards that a real vendor can. I had never expected my kernel to compete with the likes of them, and I don't expect Mark3 to change the embedded landscape either.

My main goal with Mark3 was to solve the technical shortfalls in the FunkOS kernel by applying my experience in kernel development. As a result, Mark3 is better than FunkOS in almost every way; it scales better, has lower interrupt latency, and is generally more thoughtfully designed (all at a small cost to code size).

Another goal I had was to create something easy to understand, that could be documented and serve as a good introduction to RTOS kernel design. The end result of these goals is the kernel as presented in this book - a full source listing of a working OS kernel, with each module completely documented and explained in detail.

Finally, I wanted to prove that a kernel written entirely in C++ could perform just as well as one written in C, without incurring any extra overhead. Comparing the same configuration of Mark2 to Mark3, the code size is remarkably similar, and the execution performance is just as good. Not only that, but there are fewer lines of code. The code is more readable and easier to understand as a result of making use of object-oriented concepts provided by C++. Applications are easier to write because common concepts are encapsulated into objects (Threads, Semaphores, Mutexes, etc.) with their own methods and data, as opposed to APIs which rely on lots of explicit pointer-passing, type casting, and other operations that are typically considered "unsafe" or "advaned topics" in C.

Chapter 3

Can you Afford an RTOS?

Of course, since you're reading the manual for an RTOS that I've been developing for the last few years, you can guess that the conclusion that I draw is a resounding "yes".

If your code is of any sort of non-trivial complexity (say, at least a few-thousand lines), then a more appropriate question would be "can you afford <em>not</em> to use an RTOS in your system?".

In K_SHORT, there are simply too many benefits of an RTOS to ignore.

- Sophisticated synchronization objects
- The ability to efficiently block and wait
- Enhanced responsiveness for high-priority tasks
- Built in timers
- Built in efficient memory management

Sure, these features have a cost in code space and RAM, but from my experience the cost of trying to code around a lack of these features will cost you as much - if not more. The results are often far less maintainable, error prone, and complex. And that simply adds time and cost. Real developers ship, and the RTOS is quickly becoming one of the standard tools that help keep developers shipping.

3.1 Intro

(Note - this article was written for the C-based Mark2 kernel, which is slightly different. While the general principles are the same, the numbers are not an 100% accurate reflection of the current costs of the Mark3 kernel.)

One of the main arguments against using an RTOS in an embedded project is that the overhead incurred is too great to be justified. Concerns over "wasted" RAM caused by using multiple stacks, added CPU utilization, and the "large" code footprint from the kernel cause a large number of developers to shun using a preemptive RTOS, instead favoring a non-preemptive, application-specific solution.

I believe that not only is the impact negligible in most cases, but that the benefits of writing an application with an RTOS can lead to savings around the board (code size, quality, reliability, and development time). While these other benefits provide the most compelling case for using an RTOS, they are far more challenging to demonstrate in a quantitative way, and are clearly documented in numerous industry-based case studies.

While there is some overhead associated with an RTOS, the typical arguments are largely unfounded when an RTOS is correctly implemented in a system. By measuring the true overhead of a preemptive RTOS in a typical application, we will demonstrate that the impact to code space, RAM, and CPU usage is minimal, and indeed acceptable for a wide range of CPU targets.

To illustrate just how little an RTOS impacts the size of an embedded software design we will look at a typical microcontroller project and analyze the various types of overhead associated with using a pre-emptive realtime kernel versus a similar non-preemptive event-based framework.

RTOS overhead can be broken into three distinct areas:

- Code space: The amount of code space eaten up by the kernel (static)
- Memory overhead: The RAM associated wtih running the kernel and application threads.
- Runtime overhead: The CPU cycles required for the kernel's functionality (primarily scheduling and thread switching)

While there are other notable reasons to include or avoid the use of an RTOS in certain applications (determinism, responsiveness, and interrupt latency among others), these are not considered in this discussion - as they are difficult to consider for the scope of our "canned" application. Application description:

For the purpose of this comparison, we first create an application using the standard preemptive Mark3 kernel with 2 system threads running: A foreground thread and a background thread. This gives three total priority levels in the system - the interrupt level (high), and two application priority threads (medium and low), which is quite a common paradigm for microcontroller firmware designs. The foreground thread processes a variety of time-critical events at a fixed frequency, while the background thread processes lower priority, aperiodic events. When there are no background thread events to process, the processor enters its low-power mode until the next interrupt is acknowledged.

The contents of the threads themselves are unimportant for this comparison, but we can assume they perform a variety of I/O using various user-input devices and a serial graphics display. As a result, a number of Mark3 device drivers are also implemented.

The application is compiled for an ATMega328p processor which contains 32kB of code space in flash, and 2kB of RAM, which is a lower-mid-range microcontroller in Atmel's 8-bit AVR line of microcontrollers. Using the WinAVR GCC compiler with -O2 level optimizations, an executable is produced with the following code/RAM utilization:

31600 Bytes Code Space 2014 Bytes RAM

An alternate version of this project is created using a custom "super-loop" kernel, which uses a single application thread and provides 2 levels of priority (interrupt and application). In this case, the event handler processes the different priority application events to completion from highest to lowest priority.

This approach leaves the application itself largely unchanged. Using the same optimization levels as the preemptive kernel, the code compiles as follows:

29904 Bytes Code Space 1648 Bytes RAM

3.2 Memory overhead:

At first glance, the difference in RAM utilization seems quite a lot higher for the preemptive mode version of the application, but the raw numbers don't tell the whole story.

The first issue is that the cooperative-mode total does not take into account the system stack - whereas these values are included in the totals for RTOS version of the project. As a result, some further analysis is required to determine how the stack sizes truly compare.

In cooperative mode, there is only one thread of execution - so considering that multiple event handlers are executed in turn, the stack requirements for cooperative mode is simply determined by those of the most stack-intensive event handler.

In contrast, the preemptive kernel requires a separate stack for each active thread, and as a result the stack usage of the system is the sum of the stacks for all threads.

Since the application and idle events are the same for both preemptive and cooperative mode, we know that their (independent) stack requirements will be the same in both cases.

For cooperative mode, we see that the idle thread stack utilization is lower than that of the application thread, and so the application thread's determines the stack size requirement. Again, with the preemptive kernel the stack utilization is the sum of the stacks defined for both threads.

As a result, the difference in overhead between the two cases becomes the extra stack required for the idle thread - which in our case is (a somewhat generous) 64 bytes.

The numbers still don't add up completely, but looking into the linker output we see that the rest of the difference comes from the extra data structures used to declare the threads in preemptive mode.

With this taken into account, the true memory cost of a 2-thread system ends up being around 150 bytes of RA-M - which is less than 8% of the total memory available on this particular microcontroller. Whether or not this is reasonable certainly depends on the application, but more importantly, it is not so unreasonable as to eliminate an RTOS-based solution from being considered.

3.3 Code Space Overhead:

The difference in code space overhead between the preemptive and cooperative mode solutions is less of an issue. Part of this reason is that both the preemptive and cooperative kernels are relatively small, and even an average target device (like the Atmega328 we've chosen) has plenty of room.

Mark3 can be configured so that only features necessary for the application are included in the RTOS - you only pay for the parts of the system that you use. In this way, we can measure the overhead on a feature-by-feature basis, which is shown below for the kernel as configured for this application:

3466 Bytes

The configuration tested in this comparison uses the thread/port module with timers, drivers, and semaphores, for a total kernel size of $\sim$3.5KB, with the rest of the code space occupied by the application.

The custom cooperative-mode framework has a similar structure which is broken down by module as follows:

1850 Bytes

As can be seen from the compiler's output, the difference in code space between the two versions of the application is about 1.7kB - or about 5% of the available code space on the selected processor. While nearly all of this comes from the added overhead of the kernel, the rest of the difference comes the changes to the application necessary to facilitate the different frameworks.

3.4 Runtime Overhead

On the cooperative kernel, the overhead associated with running the thread is the time it takes the kernel to notice a pending event flag and launch the appropriate event handler, plus the timer interrupt execution time.

Similarly, on the preemptive kernel, the overhead is the time it takes to switch contexts to the application thread, plus the timer interrupt execution time.

The timer interrupt overhead is similar for both cases, so the overhead then becomes the difference between the following:

Preemptive mode:

- Posting the semaphore that wakes the high-priority thread
- Performing a context switch to the high-priority thread

Cooperative mode:

- Setting the high-priority thread's event flag
- Acknowledging the event from the event loop

Using the cycle-accurate AVR simulator, we find the end-to-end event sequence time to be 20.4us for the cooperative mode scheduler and 44.2us for the preemptive, giving a difference of 23.8us.

With a fixed high-priority event frequency of 33Hz, we achieve a runtime overhead of 983.4us per second, or 0.-0983% of the total available CPU time. Now, obviously this value would expand at higher event frequencies and/or slower CPU frequencies, but for this typical application we find the difference in runtime overhead to be neglible for a preemptive system. Analysis:

For the selected test application and platform, including a preemptive RTOS is entirely reasonable, as the costs are low relative to a non-preemptive kernel solution. But these costs scale relative to the speed, memory and code space of the target processor. Because of these variables, there is no "magic bullet" environment suitable for every application, but Mark3 attempts to provide a framework suitable for a wide range of targets.

On the one hand, if these tests had been performed on a higher-end microcontroller such as the ATMega1284p (containing 128kB of code space and 16kB of RAM), the overhead would be in the noise. For this type of resource-rich microcontroller, there would be no reason to avoid using the Mark3 preemptive kernel.

Conversely, using a lower-end microcontroller like an ATMega88pa (which has only 8kB of code space and 1k-B of RAM), the added overhead would likely be prohibitive for including a preemptive kernel. In this case, the cooperative-mode kernel would be a better choice.

As a rule of thumb, if one budgets 10% of a microcontroller's code space/RAM for a preemptive kernel's overhead, you should only require at minimum a microcontroller with 16k of code space and 2kB of RAM as a base platform for an RTOS. Unless there are serious constraints on the system that require much better latency or responsiveness than can be achieved with RTOS overhead, almost any modern platform is sufficient for hosting a kernel. In the event you find yourself with a microprocessor with external memory, there should be no reason to avoid using an RTOS at all.

Chapter 4

Superloops

4.1 Intro to Superloops

Before we start taking a look at designing a real-time operating system, it's worthwhile taking a look through one of the most-common design patterns that developers use to manage task execution in embedded systems - Superloops.

Systems based on superloops favor the system control logic baked directly into the application code, usually under the guise of simplicity, or memory (code and RAM) efficiency. For simple systems, superloops can definitely get the job done. However, they have some serious limitations, and are not suitable for every kind of project. In a lot of cases you can squeak by using superloops - especially in extremely constrained systems, but in general they are not a solid basis for reusable, portable code.

Nonetheless, a variety of examples are presented here- from the extremely simple, to cooperative and liimted-preemptive multitasking systems, all of which are examples are representative of real-world systems that I've either written the firmware for, or have seen in my experience.

4.2 The simplest loop

Let's start with the simplest embedded system design possible - an infinite loop that performs a single task repeatedly:

```
int main()
{
    while(1)
    {
        Do_Something();
    }
}
```

Here, the code inside the loop will run a single function forever and ever. Not much to it, is there? But you might be surprised at just how much embedded system firmware is implemented using essentially the same mechanism - there isn't anything wrong with that, but it's just not that interesting.

While the execution timeline for this program is equally boring, for the sake of completeness it would look like this:

Despite its simplicity we can see the beginnings of some core OS concepts. Here, the while(1) statement can be logically seen as the he operating system kernel - this one control statement determines what tasks can run in the system, and defines the constraints that could modify their execution. But at the end of the day, that's a big part of what a kernel is - a mechanism that controls the execution of application code.

The second concept here is the task. This is application code provided by the user to perform some useful purpose in a system. In this case Do_something() represents that task - it could be monitoring blood pressure, reading a sensor and writing its data to a terminal, or playing an MP3; anything you can think of for an embedded system to do. A simple round-robin multi-tasking system can be built off of this example by simply adding additional tasks in sequence in the main while-loop. Note that in this example the CPU is always busy running tasks - at no time is the CPU idle, meaning that it is likely burning a lot of power.

While we conceptually have two separate pieces of code involved here (an operating system kernel and a set of running tasks), they are not logically separate. The OS code is indistinguishable from the application. It's like a single-celled organism - everything is crammed together within the walls of an indivisible unit; and specialized to perform its given function relying solely on instinct.

4.3 Interrupt-Driven Super-loop

In the previous example, we had a system without any way to control the execution of the task- it just runs forever. There's no way to control when the task can (or more importantly can't) run, which greatly limits the usefulness of the system. Say you only want your task to run every 100 miliseconds - in the previous code, you have to add a hard-coded delay at the end of your task's execution to ensure your code runs only when it should.

Fortunately, there is a much more elegant way to do this. In this example, we introduce the concept of the synchronization object. A Synchronization object is some data structure which works within the bounds of the operating system to tell tasks when they can run, and in many cases includes special data unique to the synchronization event. There are a whole family of synchronization objects, which we'll get into later. In this example, we make use of the simplest synchronization primitive - the global flag.

With the addition of synchronization brings the addition of event-driven systems. If you're programming a microcontroller system, you generally have scores of peripherals available to you - timers, GPIOs, ADCs, UARTs, ethernet, USB, etc. All of which can be configured to provide a stimulus to your system by means of interrupts. This stimulus gives us the ability not only to program our micros to do_something(), but to do_something() if-and-only-if a corresponding trigger has occurred.

The following concepts are shown in the example below:

```
volatile K_BOOL something_to_do = false;

__interrupt__ My_Interrupt_Source(void)
{
    something_to_do = true;
}

int main()
{
    while(1)
    {
        if( something_to_do )
        {
            Do_something();
            something_to_do = false;
        }
        else
        {
            Idle();
        }
    }
}
```

So there you have it - an event driven system which uses a global variable to synchronize the execution of our task based on the occurrence of an interrupt. It's still just a bare-metal, OS-baked-into-the-aplication system, but it's introduced a whole bunch of added complexity (and control!) into the system.

The first thing to notice in the source is that the global variable, something_to_do, is used as a synchronization object. When an interrupt occurs from some external event, triggering the My_Interrupt_Source() ISR, program flow in main() is interrupted, the interrupt handler is run, and something_to_do is set to true, letting us know that when we get back to main(), that we should run our Do_something() task.

Another new concept at play here is that of the idle function. In general, when running an event driven system, there are times when the CPU has no application tasks to run. In order to minimize power consumption, CPUs usually contain instructions or registers that can be set up to disable non-essential subsets of the system when there's nothing to do. In general, the sleeping system can be re-activated quickly as a result of an interrupt or other external stimulus, allowing normal processing to resume.

Now, we could just call Do_something() from the interrupt itself - but that's generally not a great solution. In general, the more time we spend inside an interrupt, the more time we spend with at least some interrupts disabled. As a result, we end up with interrupt latency. Now, in this system, with only one interrupt source and only one task this might not be a big deal, but say that Do_something() takes several seconds to complete, and in that time several other interrupts occur from other sources. While executing in our K_LONG-running interrupt, no other interrupts can be processed - in many cases, if two interrupts of the same type occur before the first is processed, one of these interrupt events will be lost. This can be utterly disastrous in a real-time system and should be avoided at all costs. As a result, it's generally preferable to use synchronization objects whenever possible to defer processing outside of the ISR.

Another OS concept that is implicitly introduced in this example is that of task priority. When an interrupt occurs, the normal execution of code in main() is preempted: control is swapped over to the ISR (which runs to completion), and then control is given back to main() where it left off. The very fact that interrupts take precedence over what's running shows that main is conceptually a "low-priority" task, and that all ISRs are "high-priority" tasks. In this example, our "high-priority" task is setting a variable to tell our "low-priority" task that it can do something useful. We will investigate the concept of task priority further in the next example.

Preemption is another key principle in embedded systems. This is the notion that whatever the CPU is doing when an interrupt occurs, it should stop, cache its current state (referred to as its context), and allow the high-priority event to be processed. The context of the previous task is then restored its state before the interrupt, and resumes processing. We'll come back to preemption frequently, since the concept comes up frequently in RTOS-based systems.

4.4 Cooperative multi-tasking

Our next example takes the previous example one step further by introducing cooperative multi-tasking:

```
// Bitfield values used to represent three distinct tasks
#define TASK_1_EVENT (0x01)
#define TASK_2_EVENT (0x02)
#define TASK_3_EVENT (0x04)

volatile K_UCHAR event_flags = 0;

// Interrupt sources used to trigger event execution

__interrupt__  My_Interrupt_1(void)
{
    event_flags |= TASK_1_EVENT;
}

__interrupt__ My_Interrupt_2(void)
{
    event_flags |= TASK_2_EVENT;
}

__interrupt__ My_Interrupt_3(void)
{
    event_flags |= TASK_3_EVENT;
}

// Main tasks
int main(void)
{
    while(1)
    {
        while(event_flags)
        {
            if( event_flags & TASK_1_EVENT)
            {
                Do_Task_1();
                event_flags &= ~TASK_1_EVENT;
            } else if( event_flags & TASK_2_EVENT) {
                Do_Task_2();
                event_flags &= ~TASK_2_EVENT;
            } else if( event_flags & TASK_3_EVENT) {
                Do_Task_3();
                event_flags &= ~TASK_3_EVENT;
            }
        }
        Idle();
    }
}
```

This system is very similar to what we had before - however the differences are worth discussing. First, we have stimulus from multiple interrupt sources: each ISR is responsible for setting a single bit in our global event flag, which is then used to control execution of individual tasks from within main().

Next, we can see that tasks are explicitly given priorities inside the main loop based on the logic of the if/else if structure. As K_LONG as there is something set in the event flag, we will always try to execute Task1 first, and only when Task1 isn't set will we attempt to execute Task2, and then Task 3. This added logic provides the notion of priority. However, because each of these tasks exist within the same context (they're just different functions called from our main control loop), we don't have the same notion of preemption that we have when dealing with interrupts.

That means that even through we may be running Task2 and an event flag for Task1 is set by an interrupt, the CPU still has to finish processing Task2 to completion before Task1 can be run. And that's why this kind of scheduling is referred to ascooperative multitasking: we can have as many tasks as we want, but unless they cooperate by means of returning back to main, the system can end up with high-priority tasks getting starved for CPU time by lower-priority, K_LONG-running tasks.

This is one of the more popular Os-baked-into-the-application approaches, and is widely used in a variety of real-time embedded systems.

4.5 Hybrid cooperative/preemptive multi-tasking

The final variation on the superloop design utilizes software-triggered interrupts to simulate a hybrid cooperative/preemptive multitasking system. Consider the example code below.

```
// Bitfields used to represent high-priority tasks.  Tasks in this group
// can preempt tasks in the group below - but not eachother.
#define HP_TASK_1        (0x01)
#define HP_TASK_2        (0x02)

volatile K_UCHAR hp_tasks = 0;

// Bitfields used to represent low-priority tasks.
#define LP_TASK_1        (0x01)
#define LP_TASK_2        (0x02)

volatile K_UCHAR lp_tasks = 0;

// Interrupt sources, used to trigger both high and low priority tasks.
__interrupt__ System_Interrupt_1(void)
{
    // Set any of the other tasks from here...
    hp_tasks |= HP_TASK_1;
    // Trigger the SWI that calls the High_Priority_Tasks interrupt handler
    SWI();
}

__interrupt__ System_Interrupt_n...(void)
{
    // Set any of the other tasks from here...
}


// Interrupt handler that is used to implement the high-priority event context
__interrupt__ High_Priority_Tasks(void)
{
    // Enabled every interrupt except this one
    Disable_My_Interrupt();
    Enable_Interrupts();
    while( hp_tasks)
    {
        if( hp_tasks & HP_TASK_1)
        {
            HP_Task1();
            hp_tasks &= ~HP_TASK_1;
        }
        else if (hp_tasks & HP_TASK_2)
        {
            HP_Task2();
            hp_tasks &= ~HP_TASK_2;
        }
    }
    Restore_Interrupts();
    Enable_My_Interrupt();
}
```

```
// Main loop, used to implement the low-priority events
int main(void)
{
    // Set the function to run when a SWI is triggered
    Set_SWI(High_Priority_Tasks);

    // Run our super-loop
    while(1)
    {
        while (lp_tasks)
        {
            if (lp_tasks & LP_TASK_1)
            {
                LP_Task1();
                lp_tasks &= ~LP_TASK_1;
            }
            else if (lp_tasks & LP_TASK_2)
            {
                LP_Task2();
                lp_tasks &= ~LP_TASK_2;
            }
        }
        Idle();
    }
}
```

In this example, High_Priority_Tasks() can be triggered at any time as a result of a software interrupt (SWI),. When a high-priority event is set, the code that sets the event calls the SWI as well, which instantly preempts whatever is happening in main, switching to the high-priority interrupt handler. If the CPU is executing in an interrupt handler already, the current ISR completes, at which point control is given to the high priority interrupt handler.

Once inside the HP ISR, all interrupts (except the software interrupt) are re-enabled, which allows this interrupt to be preempted by other interrupt sources, which is called interrupt nesting. As a result, we end up with two distinct execution contexts (main and HighPriorityTasks()), in which all tasks in the high-priority group are guaranteed to preempt main() tasks, and will run to completion before returning control back to tasks in main(). This is a very basic preemptive multitasking scenario, approximating a "real" RTOS system with two threads of different priorities.

4.6 Problems with superloops

As mentioned earlier, a lot of real-world systems are implemented using a superloop design; and while they are simple to understand due to the limited and obvious control logic involved, they are not without their problems.

Hidden Costs

It's difficult to calculate the overhead of the superloop and the code required to implement workarounds for blocking calls, scheduling, and preemption. There's a cost in both the logic used to implement workarounds (usually involving state machines), as well as a cost to maintainability that comes with breaking up into chunks based on execution time instead of logical operations. In moderate firmware systems, this size cost can exceed the overhead of a reasonably well-featured RTOS, and the deficit in maintainability is something that is measurable in terms of lost productivity through debugging and profiling.

Tightly-coupled code

Because the control logic is integrated so closely with the application logic, a lot of care must be taken not to compromise the separation between application and system code. The timing loops, state machines, and architecture-specific control mechanisms used to avoid (or simulate) preemption can all contribute to the problem. As a result, a lot of superloop code ends up being difficult to port without effectively simulating or replicating the underlying system for which the application was written. Abstraction layers can mitigate the risks, but a lot of care should be taken to fully decouple the application code from the system code.

No blocking calls

In a super-loop environment, there's no such thing as a blocking call or blocking objects. Tasks cannot stop mid-execution for event-driven I/O from other contexts - they must always run to completion. If busy-waiting and polling are used as a substitute, it increases latency and wastes cycles. As a result, extra code complexity is often times necessary to work-around this lack of blocking objects, often times through implementing additional state machines. In a large enough system, the added overhead in code size and cycles can add up.

Difficult to guarantee responsiveness

Without multiple levels of priority, it may be difficult to guarantee a certain degree of real-time responsiveness without added profiling and tweaking. The latency of a given task in a priority-based cooperative multitasking system is the length of the longest task. Care must be taken to break tasks up into appropriate sized chunks in order to ensure that higher-priority tasks can run in a timely fashion - a manual process that must be repeated as new tasks are added in the system. Once again, this adds extra complexity that makes code larger, more difficult to understand and maintain due to the artificial subdivision of tasks into time-based components.

Limited preemption capability

As shown in the example code, the way to gain preemption in a superloop is through the use of nested interrupts. While this isn't unwiedly for two levels of priority, adding more levels beyond this is becomes complicated. In this case, it becomes necessary to track interrupt nesting manually, and separate sets of tasks that can run within given priority loops - and deadlock becomes more difficult to avoid.

Chapter 5

Mark3 Overview

5.1 Intro

The following section details the overall design of Mark3, the goals I've set out to achieve, the features that I've intended to provide, as well as an introduction to the programming concepts used to make it happen.

5.2 Features

Mark3 is a fully-featured real-time kernel, and is feature-competitive with other open-source and commercial RTOS's in the embedded arena.

The key features of this RTOS are:

- Flexible Scheduler
  - Unlimited number of threads with 8 priority levels
  - Unlimited threads per priority level
  - Round-robin scheduling for threads at each priority level
  - Time quantum scheduling for each thread in a given priority level
- Configurable stacks for each Thread
- Resource protection:
  - Integrated mutual-exclusion semaphores (Mutex)
  - Priority-inheritance on Mutex objects to prevent priority inversion
- Synchronization Objects
  - Binary and counting Semaphore to coordinate thread execution
- Efficient Timers
  - The RTOS is tickless, the OS only wakes up when a timer expires, not at a regular interval
  - One-shot and periodic timers with event callbacks
  - Timers are high-precision and K_LONG-counting (about 68000 seconds when used with a 16us resolution timer)
- Driver API
  - A hardware abstraction layer is provided to simplify driver development
- Robust Interprocess Communications
  - Threadsafe global Message pool and configurable message queues

5.3 Design Goals

Lightweight

Mark3 can be configured to have an extremely low static memory footprint. Each thread is defined with its own stack, and each thread structure can be configured to take as little as 26 bytes of RAM. The complete Mark3 kernel with all features, setup code, a serial driver, and the Mark3 protocol libraries comes in at under 9K of code space and 1K of RAM on atmel AVR.

Modular

Each system feature can be enabled or disabled by modifying the kernel configuration header file. Include what you want, and ignore the rest to save code space and RAM.

Easily Portable

Mark3 should be portable to a variety of 8, 16 and 32 bit architectures without MMUs. Porting the OS to a new architecture is relatively straightforward, requiring only device-specific implementations for the lowest-level operations such as context switching and timer setup.

Easy To Use

Mark3 is small by design - which gives it the advantage that it's also easy to develop for. This manual, the code itself, and the Doxygen documentation in the code provide ample documentation to get you up to speed quickly. Because you get to see the source, there's nothing left to assumption.

Simple to Understand

Not only is the Mark3 API rigorously documented (hey - that's what this book is for!), but the architecture and naming conventions are intuitive - it's easy to figure out where code lives, and how it works. Individual modules are small due to the "one feature per file" rule used in development. This makes Mark3 an ideal platform for learning about aspects of RTOS design.

Chapter 6

Getting Started

6.1 Kernel Setup

This section details the process of defining threads, initializing the kernel, and adding threads to the scheduler.

If you're at all familiar with real-time operating systems, then these setup and initialization steps should be familiar. I've tried very hard to ensure that as much of the heavy lifting is hidden from the user, so that only the bare minimum of calls are required to get things started.

The examples presented in this chapter are real, working examples taken from the ATmega328p port.

First, you'll need to create the necessary data structures and functions for the threads:

1. Create a Thread object for all of the "root" or "initial" tasks.
2. Allocate stacks for each of the Threads
3. Define an entry-point function for each Thread

This is shown in the example code below:

```
//---------------------------------------------------------------------------
#include "thread.h"
#include "kernel.h"

//1) Create a thread object for all of the "root" or "initial" tasks
static Thread AppThread;
static Thread IdleThread;

//2) Allocate stacks for each thread
#define STACK_SIZE_APP      (192)
#define STACK_SIZE_IDLE     (128)

static K_UCHAR aucAppStack[STACK_SIZE_APP];
static K_UCHAR aucIdleStack[STACK_SIZE_IDLE];

//3) Define entry point functions for each thread
void AppThread(void);
void IdleThread(void);
```

Next, we'll need to add the required kernel initialization code to main. This consists of running the Kernel's init routine, initializing all of the threads we defined, adding the threads to the scheduler, and finally calling Kernel::Start(), which transfers control of the system to the RTOS.

These steps are illustrated in the following example.

```
int main(void)
{
    //1) Initialize the kernel prior to use
    Kernel::Init();

    //2) Initialize all of the threads we've defined
    AppThread.Init( aucAppStack,
                    STACK_SIZE_APP,
                    1,
                    (void*)AppEntry,
                    NULL );

    IdleThread.Init( aucIdleStack,
                     STACK_SIZE_IDLE,
                     0,
                     4,
                     (void*)IdleEntry,
                     NULL );

    //3) Add the threads to the scheduler
    AppThread.Start();
    IdleThread.Start();

    //4) Give control of the system to the kernel
    Kernel::Start();
}
```

Not much to it, is there? There are a few noteworthy points in this code, though.

In order for the kernel to work properly, a system must always contain an idle thread; that is, a thread at priority level 0 that never blocks. This thread is responsible for performing any of the low-level power management on the CPU in order to maximize battery life in an embedded device. The idle thread must also never block, and it must never exit. Either of these operations will cause undefined behavior in the system.

The App thread is at a priority level greater-than 0. This ensures that as K_LONG as the App thread has something useful to do, it will be given control of the CPU. In this case, if the app thread blocks, control will be given back to the Idle thread, which will put the CPU into a power-saving mode until an interrupt occurs.

Stack sizes must be large enough to accommodate not only the requirements of the threads, but also the requirements of interrupts - up to the maximum interrupt-nesting level used. Stack overflows are super-easy to run into in an embedded system; if you encounter strange and unexplained behavior in your code, chances are good that one of your threads is blowing its stack.

6.2 Threads

Mark3 Threads act as independent tasks in the system. While they share the same address-space, global data, device-drivers, and system peripherals, each thread has its own set of CPU registers and stack, collectively known as the thread's **context**. The context is what allows the RTOS kernel to rapidly switch between threads at a high rate, giving the illusion that multiple things are happening in a system, when really, only one thread is executing at a time.

6.2.1 Thread Setup

Each instance of the Thread class represents a thread, its stack, its CPU context, and all of the state and metadata maintained by the kernel. Before a Thread will be scheduled to run, it must first be initialized with the necessary configuration data.

The Init function gives the user the opportunity to set the stack, stack size, thread priority, entry-point function, entry-function argument, and round-robin time quantum:

Thread stacks are pointers to blobs of memory (usually K_CHAR arrays) carved out of the system's address space. Each thread must have a stack defined that's large enough to handle not only the requirements of local variables in the thread's code path, but also the maximum depth of the ISR stack.

Priorities should be chosen carefully such that the shortest tasks with the most strict determinism requirements are executed first - and are thus located in the highest priorities. Tasks that take the longest to execute (and require the least degree of responsiveness) must occupy the lower thread priorities. The idle thread must be the only thread occupying the lowest priority level.

The thread quantum only aplies when there are multiple threads in the ready queue at the same priority level. This interval is used to kick-off a timer that will cycle execution between the threads in the priority list so that they each get a fair chance to execute.

The entry function is the function that the kernel calls first when the thread instance is first started. Entry functions have at most one argument - a pointer to a data-object specified by the user during initialization.

An example thread initailization is shown below:

```
Thread clMyThread;
K_UCHAR aucStack[192];

void AppEntry(void)
{
    while(1)
    {
        // Do something!
    }
}

...
{
    clMyThread.Init(aucStack,
                    192,
                    1,
                    4,
                    (void*)AppEntry,
                    NULL );

}
```

Once a thread has been initialized, it can be added to the scheduler by calling:

```
clMyThread.Start();
```

The thread will be placed into the Scheduler's queue at the designated priority, where it will wait its turn for execution.

6.2.2 Entry Functions

Mark3 Threads should not run-to-completion - they should execute as infinite loops that perform a series of tasks, appropriately partitioned to provide the responsiveness characteristics desired in the system.

The most basic Thread loop is shown below:

```
void Thread( void *param )
{
    while(1)
    {
        // Do Something
    }
}
```

Threads can interact with eachother in the system by means of synchronization objects (Semaphore), mutual-exclusion objects (Mutex), Inter-process messaging (MessageQueue), and timers (Timer).

Threads can suspend their own execution for a predetermined period of time by using the static Thread::Sleep() method. Calling this will block the Thread's executin until the amount of time specified has ellapsed. Upon expiry, the thread will be placed back into the ready queue for its priority level, where it awaits its next turn to run.

6.3 Timers

Timer objects are used to trigger callback events periodic or on a one-shot (alarm) basis.

While extremely simple to use, they provide one of the most powerful execution contexts in the system. The timer callbacks execute from within the timer callback ISR in an interrupt-enabled context. As such, timer callbacks are considered higher-priority than any thread in the system, but lower priority than other interrupts. Care must be taken to ensure that timer callbacks execute as quickly as possible to minimize the impact of processing on the throughput of tasks in the system. Wherever possible, heavy-lifting should be deferred to the threads by way of semaphores or messages.

Below is an example showing how to start a periodic system timer which will trigger every second:

```
{
    Timer clTimer;
    clTimer.Init();

    clTimer.Start( 1000,
                   1,
                   MyCallback,
                   (void*)&my_data );

    ... // Keep doing work in the thread
}

// Callback function, executed from the timer-expiry context.
void MyCallBack( Thread *pclOwner_, void *pvData_ )
{
    LED.Flash(); // Flash an LED.
}
```

6.4 Semaphores

Semaphores are used to synchronized execution of threads based on the availability (and quantity) of application-specific resources in the system. They are extremely useful for solving producer-consumer problems, and are the method-of-choice for creating efficient, low latency systems, where ISRs post semaphores that are handled from within the context of individual threads. (Yes, Semaphores can be posted - but not pended - from the interrupt context).

The following is an example of the producer-consumer usage of a binary semaphore:

```
Semaphore clSemaphore; // Declare a semaphore shared between a
          producer and a consumer thread.

void Producer()
{
    clSemaphore.Init(0, 1);
    while(1)
    {
        // Do some work, create something to be consumed

        // Post a semaphore, allowing another thread to consume the data
        clSemaphore.Post();
    }
}

void Consumer()
{
    // Assumes semaphore initialized before use...
    While(1)
    {
        // Wait for new data from the producer thread
        clSemaphore.Pend();

        // Consume the data!
    }
}
```

And an example of using semaphores from the ISR context to perform event- driven processing.

```
Semaphore clSemaphore;

__interrupt__ MyISR()
{
    clSemaphore.Post(); // Post the interrupt.  Lightweight when
       uncontested.
}

void MyThread()
{
    clSemaphore.Init(0, 1); // Ensure this is initialized before the MyISR
       interrupt is enabled.
    while(1)
    {
        // Wait until we get notification from the interrupt
        clSemaphore.Pend();

        // Interrupt has fired, do the necessary work in this thread's context
        HeavyLifting();
    }
}
```

6.5 Mutexes

Mutexes (Mutual exclusion objects) are provided as a means of creating "protected sections" around a particular resource, allowing for access of these objects to be serialized. Only one thread can hold the mutex at a time - other threads have to wait until the region is released by the owner thread before they can take their turn operating on the protected resource. Note that mutexes can only be owned by threads - they are not available to other contexts (i.e. interrupts). Calling the mutex APIs from an interrupt will cause catastrophic system failures.

Note that these objects are also not recursive- that is, the owner thread can not attempt to claim a mutex more than once.

Prioritiy inheritence is provided with these objects as a means to avoid prioritiy inversions. Whenever a thread at a priority than the mutex owner blocks on a mutex, the priority of the current thread is boosted to the highest-priority waiter to ensure that other tasks at intermediate priorities cannot artificically prevent progress from being made.

Mutex objects are very easy to use, as there are only three operations supported: Initialize, Claim and Release. An example is shown below.

```
Mutex clMutex;  // Create a mutex globally.

void Init()
{
    // Initialize the mutex before use.
    clMutex.Init();
}

// Some function called from a thread
void Thread1Function()
{
    clMutex.Claim();

    // Once the mutex is owned, no other thread can
    // enter a block protect by the same mutex

    my_protected_resource.do_something();
    my_protected_resource.do_something_else();

    clMutex.Release();
}

// Some function called from another thread
void Thread2Function()
{
    clMutex.Claim();

    // Once the mutex is owned, no other thread can
    // enter a block protect by the same mutex

    my_protected_resource.do_something();
    my_protected_resource.do_different_things();

    clMutex.Release();
}
```

6.6 Messages

Sending messages between threads is the key means of synchronizing access to data, and the primary mechanism to perform asynchronous data processing operations.

Sending a message consists of the following operations:

- Obtain a Message object from the global message pool
- Set the message data and event fields
- Send the message to the destination message queue

While receiving a message consists of the following steps:

- Wait for a messages in the destination message queue

- Process the message data
- Return the message back to the global message pool

These operations, and the various data objects involved are discussed in more detail in the following section.

6.6.1 Message Objects

Message objects are used to communicate arbitrary data between threads in a safe and synchronous way.

The message object consists of an event code field and a data field. The event code is used to provide context to the message object, while the data field (essentially a void $*$ data pointer) is used to provide a payload of data corresponding to the particular event.

Access to these fields is marshalled by accessors - the transmitting thread uses the SetData() and SetCode() methods to seed the data, while the receiving thread uses the GetData() and GetCode() methods to retrieve it.

By providing the data as a void data pointer instead of a fixed-size message, we achieve an unprecedented measure of simplicity and flexibility. Data can be either statically or dynamically allocated, and sized appropriately for the event without having to format and reformat data by both sending and receiving threads. The choices here are left to the user - and the kernel doesn't get in the way of efficiency.

It is worth noting that you can send messages to message queues from within ISR context. This helps maintain consistency, since the same APIs can be used to provide event-driven programming facilities throughout the whole of the OS.

6.6.2 Global Message Pool

To maintain efficiency in the messaging system (and to prevent over-allocation of data), a global pool of message objects is provided. The size of this message pool is specified in the implementation, and can be adjusted depending on the requirements of the target application as a compile-time option.

Allocating a message from the message pool is as simple as calling the GlobalMessagePool::Pop() Method.

Messages are returned back to the GlobalMessagePool::Push() method once the message contents are no longer required.

One must be careful to ensure that discarded messages always are returned to the pool, otherwise a resource leak can occur, which may cripple the operating system's ability to pass data between threads.

6.6.3 Message Queues

Message objects specify data with context, but do not specify where the messages will be sent. For this purpose we have a MessageQueue object. Sending an object to a message queue involves calling the MessageQueue::Send() method, passing in a pointer to the Message object as an argument.

When a message is sent to the queue, the first thread blocked on the queue (as a result of calling the Message-Queue Receive() method) will wake up, with a pointer to the Message object returned.

It's worth noting that multiple threads can block on the same message queue, providing a means for multiple threads to share work in parallel.

6.6.4 Messaging Example

```
// Message queue object shared between threads
MessageQueue clMsgQ;

// Function that initializes the shared message queue
void MsgQInit()
{
    clMsgQ.Init();
}
```

```
// Function called by one thread to send message data to
// another
void TxMessage()
{
    // Get a message, initialize its data
    Message *pclMesg = GlobalMessagePool::Pop();

    pclMesg->SetCode(0xAB);
    pclMesg->SetData((void*)some_data);

    // Send the data to the message queue
    clMsgQ.Send(pclMesg);
}

// Function called in the other thread to block until
// a message is received in the message queue.
void RxMessage()
{
    Message *pclMesg;

    // Block until we have a message in the queue
    pclMesg = clMsgQ.Receive();

    // Do something with the data once the message is received
    pclMesg->GetCode();

    // Free the message once we're done with it.
    GlobalMessagePool::Push(pclMesg);
}
```

6.7 Sleep

There are instances where it may be necessary for a thread to poll a resource, or wait a specific amount of time before proceeding to operate on a peripheral or volatile piece of data.

While the Timer object is generally a better choice for performing time-sensitive operations (and certainly a better choice for periodic operations), the Thread::Sleep() method provides a convenient (and efficient) mechanism that allows for a thread to suspend its execution for a specified interval.

Note that when a thread is sleeping it is blocked, during which other threads can operate, or the system can enter its idle state.

```
int GetPeripheralData();
{
    int value;
    // The hardware manual for a peripheral specifies that
    // the "foo()" method will result in data being generated
    // that can be captured using the "bar()" method.
    // However, the value only becomes valid after 10ms

    peripheral.foo();
    Thread::Sleep(10);  // Wait 10ms for data to become valid
    value = peripheral.bar();
    return value;
}
```

6.8 Round-Robin Quantum

Threads at the same thread priority are scheduled using a round-robin scheme. Each thread is given a timeslice (which can be configured) of which it shares time amongst ready threads in the group. Once a thread's timeslice has expired, the next thread in the priority group is chosen to run until its quantum has expired - the cycle continues over and over so K_LONG as each thread has work to be done.

By default, the round-robin interval is set at 4ms.

This value can be overridden by calling the thread's SetQuantum() with a new interval specified in milliseconds.

Chapter 7

Build System

Mark3 is distributed with a recursive makefile build system, allowing the entire source tree to be built into a series of libraries with simple make commands.

The way the scripts work, every directory with a valid makefile is scanned, as well as all of its subdirectories. The build then generates binary components for all of the components it finds -libraries and executables. All libraries that are generated can then be imported into an application using the linker without having to copy-and-paste files on a module-by-module basis. Applications built during this process can then be loaded onto a device directly, without requiring a GUI-based IDE. As a result, Mark2 integrates well with 3rd party tools for continuous-integration and automated testing.

This modular framework allows for large volumes of libraries and binaries to be built at once - the default build script leverages this to build all of the examples and unit tests at once, linking against the pre-built kernel, services, and drivers. Whatever can be built as a library is built as a library, promoting reuse throughout the platform, and enabling Mark3 to be used as a platform, with an ecosystem of libraries, services, drivers and applications.

7.1 Source Layout

One key aspect of Mark2 is that system features are organized into their own separate modules. These modules are further grouped together into folders based on the type of features represented:

```
Root              Base folder, contains recursive makefiles for build system
    bootloader    Mark2 Bootloader code for AVR
    build         Makefile support for various platforms
    doc           Documentation (including this)
    drivers       Device driver code
    example       Example applications
    kernel        Basic Mark2 Components (the focus of this manual)
        cpu       CPU-specific porting code
    services      Utility code and services, extended system features
    stage         Staging directory, where the build system places artifacts
    tests         Unit tests, written as C/C++ applications
```

7.2 Building the kernel

The base.mak file determines how the kernel, drivers, and libraries are built, for what targets, and with what options. Most of these options can be copied directly from the options found in your IDE managed projects. Below is an overview of the main variables used to configure the build.

```
STAGE       - Location in the filesystem where the build output is stored
ROOT_DIR    - The location of the root source tree
ARCH        - The CPU architecture to build against
VARIANT     - The variant of the above CPU to target
TOOLCHAIN   - Which toolchain to build with (dependent on ARCH and VARIANT)
```

Build.mak contains the logic which is used to perform the recursive make in all directories. Unless you really know what you're doing, it's best to leave this as-is.

You must make sure that all required paths are set in your system environment variables so that they are accessible through from the command-line.

Once configured, you can build the source tree using the various make targets:

- make headers
  - copy all headers in each module's /public subdirectory to the location specified by STAGE environment variable's ./inc subdirectory.
- make library
  - regenerate all objects copy marked as libraries (i.e. the kernel + drivers). Resulting binaries are copied into STAGE's ./lib subdirectory.
- make binary
  - build all executable projects in the root directory structure. In the default distribution, this includes the basic set of demos.

To add new components to the recursive build system, simply add your code into a new folder beneath the root install location.

Source files, the module makefile and private header files go directly in the new folder, while public headers are placed in a ./public subdirectory. Create a ./obj directory to hold the output from the builds.

The contents of the module makefile looks something like this:

```
# Include common prelude make file
include $(ROOT_DIR)base.mak

# If we're building a library, set IS_LIB and LIBNAME
# If we're building an app, set IS_APP and APPNAME
IS_LIB=1
LIBNAME=mylib

#this is the list of the source modules required to build the kernel
CPP_SOURCE = mylib.cpp \
             someotherfile.cpp

# Similarly, C-language source would be under the C_SOURCE variable.

# Include the rest of the script that is actually used for building the
# outputs
include $(ROOT_DIR)build.mak
```

Once you've placed your code files in the right place, and configured the makefile appropriately, a fresh call to make headers, make library, then make binary will guarantee that your code is built.

Now, you can still copy-and-paste the required kernel, port, and drivers, directly into your application avoiding the whole process of using make from the command line. To do this, run "make source" from the root directory in svn, and copy the contents of /stage/src into your project. This should contain the source to the kernel, all drivers, and all services that are in the tree - along with the necessary header files.

Chapter 8

License

8.1 License

Chapter 9

Profiling Results

The following profiling results were obtained using an ATMega328p @ 16MHz.

The test cases are designed to make use of the kernel profiler, which accurately measures the performance of the fundamental system APIs, in order to provide information for user comparison, as well as to ensure that regressions are not being introduced into the system.

9.1 Date Performed

Sun Jan 27 10:36:50 EST 2013

9.2 Compiler Information

The kernel and test code used in these results were built using the following compiler: ./profile.sh: 55: ./profile.sh: /home/moslevin/atmel/bin/avr-gcc: not found

9.3 Profiling Results

- Semaphore Initialization: 2356994244 cycles (averaged over 82 iterations)
- Semaphore Post (uncontested): 148 cycles (averaged over 82 iterations)
- Semaphore Pend (uncontested): 68 cycles (averaged over 82 iterations)
- Semaphore Flyback Time (Contested Pend): 1668 cycles (averaged over 82 iterations)
- Mutex Init: 4190211955 cycles (averaged over 82 iterations)
- Mutex Claim: 204 cycles (averaged over 82 iterations)
- Mutex Release: 47 cycles (averaged over 82 iterations)
- Thread Initialize: 7897 cycles (averaged over 82 iterations)
- Thread Start: 798 cycles (averaged over 82 iterations)
- Context Switch: 171 cycles (averaged over 82 iterations)
- Thread Schedule: 70 cycles (averaged over 82 iterations)

Chapter 10

Class Index

10.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 11

Class Index

11.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 12

File Index

12.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 13

Class Documentation

13.1 BlockHeap Class Reference

Single-block-size heap.

`#include <fixed_heap.h>`

Public Member Functions

- void ∗ Create (void ∗pvHeap_, K_USHORT usSize_, K_USHORT usBlockSize_)

  Create a single list heap in the blob of memory provided, with the selected heap size, and the selected number of blocks.
- void ∗ Alloc ()

  Allocate a block of memory from this heap.
- void Free (void ∗pvData_)

  Free a previously allocated block of memory.
- K_BOOL IsFree ()

  Returns the state of a heap - whether or not it has free elements.

Protected Attributes

- K_USHORT m_usBlocksFree

  Number of blocks free in the heap.

Private Attributes

- DoubleLinkList m_clList

  Linked list used to manage the blocks.

13.1.1 Detailed Description

Single-block-size heap.

Definition at line 29 of file fixed_heap.h.

13.1.2 Member Function Documentation

13.1.2.1 void ∗ BlockHeap::Alloc ( )

Allocate a block of memory from this heap.

Returns

pointer to a block of memory, or 0 on failure

Definition at line 83 of file fixed_heap.cpp.

13.1.2.2 void ∗ BlockHeap::Create (void ∗ *pvHeap_*, K_USHORT *usSize_*, K_USHORT *usBlockSize_*)

Create a single list heap in the blob of memory provided, with the selected heap size, and the selected number of blocks.

Will create as many blocks as will fit in the usSize_ parameter

Parameters

*pvHeap_*	Pointer to the heap data to initialize
*usSize_*	Size of the heap range in bytes
*usBlockSize_*	Size of each heap block in bytes

Returns

Pointer to the next heap element to initialize

Definition at line 48 of file fixed_heap.cpp.

13.1.2.3 void BlockHeap::Free (void ∗ *pvData_*)

Free a previously allocated block of memory.

Parameters

*pvData_*	Pointer to a block of data previously allocated off the heap.

Definition at line 102 of file fixed_heap.cpp.

13.1.2.4 K_BOOL BlockHeap::IsFree ( ) [inline]

Returns the state of a heap - whether or not it has free elements.

Returns

true if the heap is not full, false if the heap is full

Definition at line 74 of file fixed_heap.h.

The documentation for this class was generated from the following files:

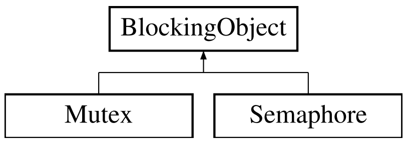
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/fixed_heap.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/fixed_heap.cpp

13.2 BlockingObject Class Reference

Class implementing thread-blocking primatives.

`#include <blocking.h>`

Inheritance diagram for BlockingObject:



Protected Member Functions

- void Block (Thread ∗pclThread_)
- void UnBlock (Thread ∗pclThread_)

Protected Attributes

- ThreadList m_clBlockList

  ThreadList which is used to hold the list of threads blocked on a given object.

13.2.1 Detailed Description

Class implementing thread-blocking primatives.

Used for implementing things like semaphores, mutexes, message queues, or anything else that could cause a thread to suspend execution on some external stimulus.

Definition at line 65 of file blocking.h.

13.2.2 Member Function Documentation

13.2.2.1 void BlockingObject::Block (Thread ∗ *pclThread_*) `[protected]`

Parameters

*pclThread_*	Pointer to the thread object that will be blocked.

Blocks a thread on this object. This is the fundamental operation performed by any sort of blocking operation in the operating system. All semaphores/mutexes/sleeping/messaging/etc ends up going through the blocking code at some point as part of the code that manages a transition from an "active" or "waiting" thread to a "blocked" thread.

The steps involved in blocking a thread (which are performed in the function itself) are as follows;

1) Remove the specified thread from the current owner's list (which is likely one of the scheduler's thread lists) 2) Add the thread to this object's thread list 3) Setting the thread's "current thread-list" point to reference this object's threadlist.

Definition at line 36 of file blocking.cpp.

13.2.2.2 void BlockingObject::UnBlock (Thread ∗ *pclThread_*) `[protected]`

Parameters

*pclThread_*	Pointer to the thread to unblock.

Unblock a thread that is already blocked on this object, returning it to the "ready" state by performing the following steps:

1) Removing the thread from this object's threadlist 2) Restoring the thread to its "original" owner's list

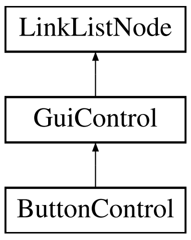
Definition at line 52 of file blocking.cpp.

The documentation for this class was generated from the following files:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/blocking.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/blocking.cpp

13.3 ButtonControl Class Reference

Inheritance diagram for ButtonControl:



Public Member Functions

- virtual void Init ()

  Initiailize the control - must be called before use.
- virtual void Draw ()

  Redraw the control "cleanly".
- virtual GuiReturn_t ProcessEvent (GuiEvent_t ∗pstEvent_)

  Process an event sent to the control.
- virtual void Activate (bool bActivate_)

  Activate or deactivate the current control - used when switching from one active control to another.
- void **SetBGColor** (COLOR eColor_)
- void **SetLineColor** (COLOR eColor_)
- void **SetFillColor** (COLOR eColor_)
- void **SetTextColor** (COLOR eColor_)
- void **SetActiveColor** (COLOR eColor_)
- void **SetFont** (Font_t ∗pstFont_)
- void **SetCaption** (const K_CHAR ∗szCaption_)
- void **SetCallback** (ButtonCallback pfCallback_, void ∗pvData_)

Private Attributes

- const K_CHAR ∗ **m_szCaption**
- Font_t ∗ **m_pstFont**
- COLOR **m_uBGColor**
- COLOR **m_uActiveColor**
- COLOR **m_uLineColor**

- COLOR **m_uFillColor**
- COLOR **m_uTextColor**
- bool **m_bState**
- void ∗ **m_pvCallbackData**
- ButtonCallback **m_pfCallback**

Additional Inherited Members

13.3.1 Detailed Description

Definition at line 32 of file control_button.h.

13.3.2 Member Function Documentation

13.3.2.1 void ButtonControl::Activate (bool *bActivate_*) [virtual]

Activate or deactivate the current control - used when switching from one active control to another.

Parameters

*bActivate_*	- true to activate, false to deactivate

Implements GuiControl.

Definition at line 215 of file control_button.cpp.

13.3.2.2 void ButtonControl::Draw ( ) [virtual]

Redraw the control "cleanly".

Subclass specific.

Implements GuiControl.

Definition at line 39 of file control_button.cpp.

13.3.2.3 void ButtonControl::Init ( ) [virtual]

Initiailize the control - must be called before use.

Implementation is subclass specific.

Implements GuiControl.

Definition at line 25 of file control_button.cpp.

13.3.2.4 GuiReturn_t ButtonControl::ProcessEvent (GuiEvent_t ∗ *pstEvent_*) [virtual]

Process an event sent to the control.

Subclass specific implementation.

Parameters

*pstEvent_*	Pointer to a struct containing the event data

Implements GuiControl.

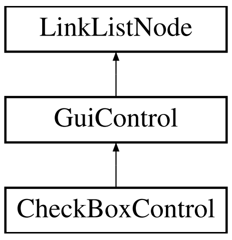
Definition at line 117 of file control_button.cpp.

The documentation for this class was generated from the following files:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_button.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_button.cpp

13.4 CheckBoxControl Class Reference

Inheritance diagram for CheckBoxControl:



Public Member Functions

- virtual void Init ()

  Initiailize the control - must be called before use.
- virtual void Draw ()

  Redraw the control "cleanly".
- virtual GuiReturn_t ProcessEvent (GuiEvent_t ∗pstEvent_)

  Process an event sent to the control.
- virtual void Activate (bool bActivate_)

  Activate or deactivate the current control - used when switching from one active control to another.
- void **SetFont** (Font_t ∗pstFont_)
- void **SetCaption** (const char ∗szCaption_)
- void **SetCheck** (bool bChecked_)
- void **SetFontColor** (COLOR uFontColor_)
- void **SetBoxColor** (COLOR uBoxColor_)
- void **SetBackColor** (COLOR uBackColor_)
- bool **IsChecked** (void)

Private Attributes

- const char ∗ **m_szCaption**
- COLOR **m_uBackColor**
- COLOR **m_uBoxColor**
- COLOR **m_uFontColor**
- Font_t ∗ **m_pstFont**
- bool **m_bChecked**

Additional Inherited Members

13.4.1 Detailed Description

Definition at line 29 of file control_checkbox.h.

13.4.2 Member Function Documentation

13.4.2.1 virtual void CheckBoxControl::Activate (bool *bActivate_*) [inline], [virtual]

Activate or deactivate the current control - used when switching from one active control to another.

Parameters

*bActivate_*	- true to activate, false to deactivate

Implements GuiControl.

Definition at line 35 of file control_checkbox.h.

13.4.2.2 void CheckBoxControl::Draw ( ) [virtual]

Redraw the control "cleanly".

Subclass specific.

Implements GuiControl.

Definition at line 59 of file control_checkbox.cpp.

13.4.2.3 void CheckBoxControl::Init ( ) [virtual]

Initiailize the control - must be called before use.

Implementation is subclass specific.

Implements GuiControl.

Definition at line 53 of file control_checkbox.cpp.

13.4.2.4 GuiReturn_t CheckBoxControl::ProcessEvent (GuiEvent_t * *pstEvent_*) [virtual]

Process an event sent to the control.

Subclass specific implementation.

Parameters

*pstEvent_*	Pointer to a struct containing the event data

Implements GuiControl.

Definition at line 112 of file control_checkbox.cpp.

The documentation for this class was generated from the following files:

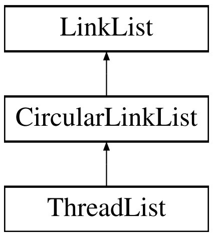
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_checkbox.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_checkbox.cpp

13.5 CircularLinkList Class Reference

Circular-linked-list data type, inherited from the base LinkList type.

`#include <ll.h>`

Inheritance diagram for CircularLinkList:



Public Member Functions

- virtual void Add (LinkListNode ∗node_)

  Add the linked list node to this linked list.
- virtual void Remove (LinkListNode ∗node_)

  Add the linked list node to this linked list.
- void PivotForward ()

  Pivot the head of the circularly linked list forward (Head = Head->next, Tail = Tail->next)
- void PivotBackward ()

  Pivot the head of the circularly linked list backward (Head = Head->prev, Tail = Tail->prev)

Additional Inherited Members

13.5.1 Detailed Description

Circular-linked-list data type, inherited from the base LinkList type.

Definition at line 201 of file ll.h.

13.5.2 Member Function Documentation

13.5.2.1 void CircularLinkList::Add (**LinkListNode** ∗ *node_*) [virtual]

Add the linked list node to this linked list.

Parameters

*node_*	Pointer to the node to add

Implements LinkList.

Reimplemented in ThreadList.

Definition at line 89 of file ll.cpp.

13.5.2.2 void CircularLinkList::Remove (**LinkListNode** ∗ *node_*) [virtual]

Add the linked list node to this linked list.

Parameters

*node_*	Pointer to the node to remove

Implements LinkList.

Reimplemented in ThreadList.

Definition at line 114 of file ll.cpp.

The documentation for this class was generated from the following files:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/ll.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/ll.cpp

13.6 DCPU Class Reference

DCPU emulator, used for running code out of EEPROM, RAM, or other memory interfaces than FLASH.

`#include <dcpu.h>`

Public Member Functions

- void Init (K_USHORT ∗pusRAM_, K_USHORT usRAMSize_, const K_USHORT ∗pusROM_, K_USHORT usROMSize_)

    Initialize the CPU emulator, specifying which driver supplies the memory read interface.
- void RunOpcode ()

    Execute the next opcode at the VM's current PC.
- DCPU_Registers ∗ GetRegisters ()

    Return a pointer to the VM's register structure.
- void SendInterrupt (K_USHORT usMessage_)

    Send an interrupt to the CPU with a given message.
- void AddPlugin (DCPUPlugin ∗pclPlugin_)

    Add a plugin to the CPU.

Private Member Functions

- void **SET** ()
- void **ADD** ()
- void **SUB** ()
- void **MUL** ()
- void **MLI** ()
- void **DIV** ()
- void **DVI** ()
- void **MOD** ()
- void **MDI** ()
- void **AND** ()
- void **BOR** ()
- void **XOR** ()
- void **SHR** ()
- void **ASR** ()
- void **SHL** ()
- bool **IFB** ()
- bool **IFC** ()
- bool **IFE** ()
- bool **IFN** ()
- bool **IFG** ()
- bool **IFA** ()
- bool **IFL** ()
- bool **IFU** ()
- void **ADX** ()
- void **SBX** ()

- void **STI** ()
- void **STD** ()
- void **JSR** ()
- void **INT** ()
- void **IAG** ()
- void **IAS** ()
- void RFI ()
- void IAQ ()
- void HWN ()
- void **HWQ** ()
- void **HWI** ()
- K_UCHAR GetOperand (K_UCHAR ucOpType_, K_USHORT ∗∗pusResult_)
- void ProcessInterruptQueue ()

  Process the next interrupt in the Queue.

Private Attributes

- DCPU_Registers m_stRegisters

  CPU Register file.
- K_USHORT ∗ a

  Temporary "a" operand pointer.
- K_USHORT ∗ b

  Temporary "b" operand pointer.
- K_USHORT m_usTempA

  Local-storage for staging literal "a" values.
- K_USHORT ∗ m_pusRAM

  Pointer to the RAM buffer.
- K_USHORT m_usRAMSize

  Size to the RAM (including stack)
- K_USHORT ∗ m_pusROM

  Pointer to the CPU ROM storage.
- K_USHORT m_usROMSize

  Size of the ROM.
- K_ULONG m_ulCycleCount

  Current cycle count.
- K_BOOL m_bInterruptQueueing

  CPU flag indicating whether or not interrupts are queued.
- K_UCHAR m_ucQueueLevel

  Current interrupt Queue level.
- K_USHORT m_ausInterruptQueue [8]

  Interrupt queue.
- DoubleLinkList m_clPluginList

  Linked-list of plug-ins.

13.6.1 Detailed Description

DCPU emulator, used for running code out of EEPROM, RAM, or other memory interfaces than FLASH.

Definition at line 359 of file dcpu.h.

13.6.2 Member Function Documentation

13.6.2.1 void DCPU::AddPlugin (DCPUPlugin * *pclPlugin_*)

Add a plugin to the CPU.

Parameters

*pclPlugin_*	Pointer to the plugin object to add

Definition at line 944 of file dcpu.cpp.

13.6.2.2 K_UCHAR DCPU::GetOperand (K_UCHAR *ucOpType_*, K_USHORT ** *pusResult_*) [private]

Parameters

*ucOpType_*	The operand type, as specified in DCPU_Argument
*pusResult_*	Pointer to the pointer that corresponds to the argument's location in memory.

Definition at line 721 of file dcpu.cpp.

13.6.2.3 DCPU_Registers * DCPU::GetRegisters ( ) [inline]

Return a pointer to the VM's register structure.

Returns

Pointer to the VM's register structure

Definition at line 391 of file dcpu.h.

13.6.2.4 void DCPU::HWN ( ) [private]

Returns the number of connected hardware devices to "a"

Definition at line 641 of file dcpu.cpp.

13.6.2.5 void DCPU::IAQ ( ) [private]

Add an interrupt to the interrupt queue if non-zero, if a = 0 then interrupts will be triggered as normal

Interrupts queued

Interrups triggered

Definition at line 623 of file dcpu.cpp.

13.6.2.6 void DCPU::Init (K_USHORT * *pusRAM_*, K_USHORT *usRAMSize_*, const K_USHORT * *pusROM_*, K_USHORT *usROMSize_*)

Initialize the CPU emulator, specifying which driver supplies the memory read interface.

This allows us to abstract RAM/FLASH/EEPROM or other memory. The VM must be initialized before any other method in the class is run.

Parameters

*pusRAM_*	Pointer to the CPU's RAM buffer
*usRAMSize_*	Size of the RAM Buffer in words
*pusROM_*	Pointer to the CPU's ROM buffer
*usROMSize_*	Size of the ROM buffer in words

Definition at line 696 of file dcpu.cpp.

13.6.2.7 void DCPU::RFI ( ) [private]

Disables interrupt queueing, pop A from the stack, then pops PC from the stack. By disabling interrupt Queueing, we're essentially re-enabling interrupts.

Definition at line 608 of file dcpu.cpp.

13.6.2.8 void DCPU::SendInterrupt (K_USHORT *usMessage_*)

Send an interrupt to the CPU with a given message.

Parameters

*usMessage_*	Message to send along with the interrupt

Definition at line 918 of file dcpu.cpp.

13.6.3 Member Data Documentation

13.6.3.1 DoubleLinkList DCPU::m_clPluginList [private]

Linked-list of plug-ins.

Definition at line 489 of file dcpu.h.

The documentation for this class was generated from the following files:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/dcpu.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/dcpu.cpp

13.7 DCPU_Registers Struct Reference

Structure defining the DCPU hardware registers.

```
#include <dcpu.h>
```

Public Attributes

- union {
    struct {
        K_USHORT **A**
        K_USHORT **B**
        K_USHORT **C**
        K_USHORT **X**
        K_USHORT **Y**
        K_USHORT **Z**
        K_USHORT **I**

```
        K_USHORT J
        K_USHORT PC
        K_USHORT SP
        K_USHORT EX
        K_USHORT IA
      }
      K_USHORT ausRegisters [12]
    };
```

13.7.1 Detailed Description

Structure defining the DCPU hardware registers.

Definition at line 72 of file dcpu.h.

The documentation for this struct was generated from the following file:

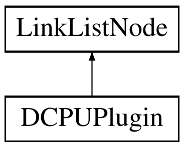
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/dcpu.h

13.8 DCPUPlugin Class Reference

Class used to provide the hardware device abstraction between the DCPU-16 emulator/VM and the host system.

`#include <dcpu.h>`

Inheritance diagram for DCPUPlugin:



Public Member Functions

- void Init (K_USHORT usDeviceNumber_, K_ULONG ulHWID_, K_ULONG ulVID_, K_USHORT usVersion_, DCPU_Callback pfCallback_)

  Initialize the DCPU plugin extension.
- void Enumerate (DCPU_Registers ∗pstRegisters_)

  Perform hardware enumeration to the target VM specified by the register set.
- void Interrupt (DCPU ∗pclCPU_)

  Execute the hardware callback.
- K_USHORT GetDeviceNumber ()

  Return the device number associated with this plugin.

Private Attributes

- K_USHORT m_usDeviceNumber

  Location of the device on the "bus".
- K_ULONG m_ulHWID

  Hardware ID.
- K_ULONG m_ulVID

Vendor ID.
- K_USHORT m_usVersion

  Hardware Version.
- DCPU_Callback m_pfCallback

  HWI Callback.

Friends

- class **DCPUPluginList**

Additional Inherited Members

13.8.1 Detailed Description

Class used to provide the hardware device abstraction between the DCPU-16 emulator/VM and the host system.

Definition at line 267 of file dcpu.h.

13.8.2 Member Function Documentation

13.8.2.1 void DCPUPlugin::Enumerate (**DCPU_Registers** ∗ *pstRegisters_*) `[inline]`

Perform hardware enumeration to the target VM specified by the register set.

Parameters

*pstRegisters_*	Pointer to the VM's CPU registers, which are filled with enumeration data. See the DCPU 1.7 spec for details.

Definition at line 311 of file dcpu.h.

13.8.2.2 K_USHORT DCPUPlugin::GetDeviceNumber ( ) `[inline]`

Return the device number associated with this plugin.

Returns

Device number associated with this plugin

Definition at line 339 of file dcpu.h.

13.8.2.3 void DCPUPlugin::Init (K_USHORT *usDeviceNumber_*, K_ULONG *ulHWID_*, K_ULONG *ulVID_*, K_USHORT *usVersion_*, **DCPU_Callback** *pfCallback_*) `[inline]`

Initialize the DCPU plugin extension.

Plug

Parameters

*usDevice-Number_*	Unique plugin device enumeration associated with this plugin
*ulHWID_*	Unique hardware type identifier
*ulVID_*	Hardware Vendor ID
*usVersion_*	Version identifier for this hardware piece
*pfCallback_*	Callback function invoked from the VM when a HWI instruction is called on this device. This is essentially the interrupt handler.

Definition at line 288 of file dcpu.h.

13.8.2.4 void DCPUPlugin::Interrupt (DCPU ∗ *pclCPU_*) [inline]

Execute the hardware callback.

Parameters

pclCPU	Pointer to the VM triggering the interrupt

Definition at line 327 of file dcpu.h.

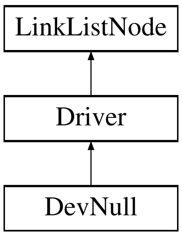
The documentation for this class was generated from the following file:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/dcpu.h

13.9 DevNull Class Reference

This class implements the "default" driver (/dev/null)

Inheritance diagram for DevNull:



Public Member Functions

- virtual void Init ()

  Initialize a driver, must be called prior to use.
- virtual K_UCHAR Open ()

  Open a device driver prior to use.
- virtual K_UCHAR Close ()

  Close a previously-opened device driver.
- virtual K_USHORT Read (K_USHORT usBytes_, K_UCHAR ∗pucData_)

  Read a specified number of bytes from the device into a specific buffer.
- virtual K_USHORT Write (K_USHORT usBytes_, K_UCHAR ∗pucData_)

  Write a payload of data of a given length to the device.
- virtual K_USHORT Control (K_USHORT usEvent_, void ∗pvDataIn_, K_USHORT usSizeIn_, void ∗pvDataOut_, K_USHORT usSizeOut_)

  This is the main entry-point for device-specific io and control operations.

Additional Inherited Members

13.9.1 Detailed Description

This class implements the "default" driver (/dev/null)

Definition at line 40 of file driver.cpp.

13.9.2 Member Function Documentation

13.9.2.1 virtual K_UCHAR DevNull::Close ( ) [inline], [virtual]

Close a previously-opened device driver.

Returns

Driver-specific return code, 0 = OK, non-0 = error

Implements Driver.

Definition at line 45 of file driver.cpp.

13.9.2.2 virtual K_USHORT DevNull::Control (K_USHORT *usEvent_*, void * *pvDataIn_*, K_USHORT *usSizeIn_*, void * *pvDataOut_*, K_USHORT *usSizeOut_*) [inline], [virtual]

This is the main entry-point for device-specific io and control operations.

This is used for implementing all "side-channel" communications with a device, and any device-specific IO operations that do not conform to the typical POSIX read/write paradigm. Use of this funciton is analagous to the non-POSIX (yet still common) devctl() or ioctl().

Parameters

*usEvent_*	Code defining the io event (driver-specific)
*pvDataIn_*	Pointer to the intput data
*usSizeIn_*	Size of the input data (in bytes)
*pvDataOut_*	Pointer to the output data
*usSizeOut_*	Size of the output data (in bytes)

Returns

Driver-specific return code, 0 = OK, non-0 = error

Implements Driver.

Definition at line 53 of file driver.cpp.

13.9.2.3 virtual K_UCHAR DevNull::Open ( ) [inline], [virtual]

Open a device driver prior to use.

Returns

Driver-specific return code, 0 = OK, non-0 = error

Implements Driver.

Definition at line 44 of file driver.cpp.

13.9.2.4 virtual K_USHORT DevNull::Read (K_USHORT *usBytes_*, K_UCHAR * *pucData_*) [inline], [virtual]

Read a specified number of bytes from the device into a specific buffer.

Depending on the driver-specific implementation, this may be a number less than the requested number of bytes read, indicating that there there was less input than desired, or that as a result of buffering, the data may not be available.

Parameters

*usBytes_*	Number of bytes to read (<= size of the buffer)
*pucData_*	Pointer to a data buffer receiving the read data

Returns

Number of bytes actually read

Implements Driver.

Definition at line 47 of file driver.cpp.

13.9.2.5 virtual K_USHORT DevNull::Write (K_USHORT *usBytes_*, K_UCHAR * *pucData_*) `[inline], [virtual]`

Write a payload of data of a given length to the device.

Depending on the implementation of the driver, the amount of data written to the device may be less than the requested number of bytes. A result less than the requested size may indicate that the device buffer is full, indicating that the user must retry the write at a later point with the remaining data.

Parameters

*usBytes_*	Number of bytes to write (<= size of the buffer)
*pucData_*	Pointer to a data buffer containing the data to write

Returns

Number of bytes actually written

Implements Driver.

Definition at line 50 of file driver.cpp.

The documentation for this class was generated from the following file:

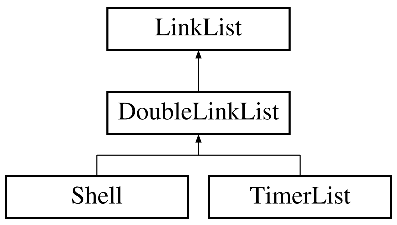
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/driver.cpp

13.10 DoubleLinkList Class Reference

Doubly-linked-list data type, inherited from the base LinkList type.

`#include <ll.h>`

Inheritance diagram for DoubleLinkList:



Public Member Functions

- DoubleLinkList ()

Default constructor - initializes the head/tail nodes to NULL.
- virtual void Add (LinkListNode ∗node_)

  Add the linked list node to this linked list.
- virtual void Remove (LinkListNode ∗node_)

  Add the linked list node to this linked list.

Additional Inherited Members

13.10.1 Detailed Description

Doubly-linked-list data type, inherited from the base LinkList type.

Definition at line 170 of file ll.h.

13.10.2 Member Function Documentation

13.10.2.1 void DoubleLinkList::Add (LinkListNode ∗ *node_*) [virtual]

Add the linked list node to this linked list.

Parameters

*node_*	Pointer to the node to add

Implements LinkList.

Definition at line 40 of file ll.cpp.

13.10.2.2 void DoubleLinkList::Remove (LinkListNode ∗ *node_*) [virtual]

Add the linked list node to this linked list.

Parameters

*node_*	Pointer to the node to remove

Implements LinkList.

Definition at line 64 of file ll.cpp.

The documentation for this class was generated from the following files:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/ll.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/ll.cpp

13.11 DrawBitmap_t Struct Reference

Defines a bitmap.

`#include <draw.h>`

Public Attributes

- K_USHORT usX

  Leftmost pixel.

- K_USHORT usY

    Uppermost pixel.
- K_USHORT usWidth

    Width of the bitmap in pixels.
- K_USHORT usHeight

    Height of the bitmap in pixels.
- K_UCHAR ucBPP

    Bits-per-pixel.
- K_UCHAR ∗ pucData

    Pixel data pointer.

13.11.1 Detailed Description

Defines a bitmap.

Definition at line 117 of file draw.h.

The documentation for this struct was generated from the following file:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/draw.h

13.12 DrawCircle_t Struct Reference

Defines a circle.

`#include <draw.h>`

Public Attributes

- K_USHORT usX

    Center X pixel.
- K_USHORT usY

    Center Y pixel.
- K_USHORT usRadius

    Radius in pixels.
- COLOR uLineColor

    Color of the circle perimeter.
- K_BOOL bFill

    Whether or not to fill the interior of the circle.
- COLOR uFillColor

    Fill color for the circle.

13.12.1 Detailed Description

Defines a circle.

Definition at line 92 of file draw.h.

The documentation for this struct was generated from the following file:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/draw.h

13.13 DrawEllipse_t Struct Reference

Defines a ellipse.

`#include <draw.h>`

Public Attributes

- K_USHORT usX

  Center X pixel.
- K_USHORT usY

  Center Y pixel.
- K_USHORT usHeight

  Height of the ellipse.
- K_USHORT usWidth

  Width of the ellipse.
- COLOR uColor

  Color of the ellipse perimeter.

13.13.1 Detailed Description

Defines a ellipse.

Definition at line 105 of file draw.h.

The documentation for this struct was generated from the following file:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/draw.h

13.14 DrawLine_t Struct Reference

Defines a simple line.

`#include <draw.h>`

Public Attributes

- K_USHORT usX1

  Starting X coordinate.
- K_USHORT usX2

  Ending X coordinate.
- K_USHORT usY1

  Starting Y Coordinate.
- K_USHORT usY2

  Ending Y coordinate.
- COLOR uColor

  Color of the pixel.

13.14.1 Detailed Description

Defines a simple line.

Definition at line 66 of file draw.h.

The documentation for this struct was generated from the following file:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/draw.h

13.15 DrawMove_t Struct Reference

Simple 2D copy/paste.

```
#include <draw.h>
```

Public Attributes

- K_USHORT usSrcX
  Source X pixel (leftmost)
- K_USHORT usSrcY
  Source Y pixel (topmost)
- K_USHORT usDstX
  Destination X pixel (leftmost)
- K_USHORT usDstY
  Destination Y pixel (topmost)
- K_USHORT usCopyHeight
  Number of rows to copy.
- K_USHORT usCopyWidth
  Number of columns to copy.

13.15.1 Detailed Description

Simple 2D copy/paste.

Moves a bitmap specified by the given source coordinates on-surface to the destination coordinates.

Definition at line 172 of file draw.h.

The documentation for this struct was generated from the following file:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/draw.h

13.16 DrawPoint_t Struct Reference

Defines a pixel.

```
#include <draw.h>
```

Public Attributes

- K_USHORT usX
  X coordinate of the pixel.
- K_USHORT usY

Y coordinate of the pixel.

- COLOR uColor

  Color of the pixel.

13.16.1 Detailed Description

Defines a pixel.

Definition at line 55 of file draw.h.

The documentation for this struct was generated from the following file:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/draw.h

13.17 DrawPoly_t Struct Reference

Defines the structure of an arbitrary polygon.

```
#include <draw.h>
```

Public Attributes

- K_USHORT usNumPoints

  Number of points in the polygon.
- COLOR uColor

  Color to use for lines/fill.
- K_BOOL bFill

  Display as wireframe or filled.
- DrawVector_t ∗ pstVector

  Vector points making the polygon.

13.17.1 Detailed Description

Defines the structure of an arbitrary polygon.

Can be used to specify the

Definition at line 199 of file draw.h.

The documentation for this struct was generated from the following file:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/draw.h

13.18 DrawRectangle_t Struct Reference

Defines a rectangle.

```
#include <draw.h>
```

Public Attributes

- K_USHORT usLeft

  Leftmost pixel of the rectangle.

- K_USHORT usTop

  Topmost pixel of the rectangle.
- K_USHORT usRight

  Rightmost pixel of the rectangle.
- K_USHORT usBottom

  Bottom pixel of the rectangle.
- COLOR uLineColor

  Color of the line.
- K_BOOL bFill

  Whether or not to floodfill the interior.
- COLOR uFillColor

  Color of the interior of the rectangle.

13.18.1 Detailed Description

Defines a rectangle.

Definition at line 78 of file draw.h.

The documentation for this struct was generated from the following file:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/draw.h

13.19 DrawStamp_t Struct Reference

Defines a 1-bit 2D bitmap of arbitrary resolution.

```
#include <draw.h>
```

Public Attributes

- K_USHORT usX

  Leftmost pixel.
- K_USHORT usY

  Uppermost pixel.
- K_USHORT usWidth

  Width of the stamp.
- K_USHORT usHeight

  Height of the stamp.
- COLOR uColor

  Color of the stamp.
- K_UCHAR ∗ pucData

  Pointer to the stamp data.

13.19.1 Detailed Description

Defines a 1-bit 2D bitmap of arbitrary resolution.

Definition at line 130 of file draw.h.

The documentation for this struct was generated from the following file:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/draw.h

13.20 DrawText_t Struct Reference

Defines a bitmap-rendered string.

```
#include <draw.h>
```

Public Attributes

- K_USHORT usLeft

  Leftmost pixel of the text.
- K_USHORT usTop

  Uppermost pixel of the text.
- COLOR uColor

  Color of the text.
- Font_t ∗ pstFont

  Pointer to the font used to render the text.
- const K_CHAR ∗ pcString

  ASCII String to render.

13.20.1 Detailed Description

Defines a bitmap-rendered string.

Definition at line 144 of file draw.h.

The documentation for this struct was generated from the following file:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/draw.h

13.21 DrawVector_t Struct Reference

Specifies a single 2D point.

```
#include <draw.h>
```

Public Attributes

- K_USHORT **usX**
- K_USHORT **usY**

13.21.1 Detailed Description

Specifies a single 2D point.

When used in arrays, this provides a way to draw vector paths, which form the basis of the polygon data structures.

Definition at line 188 of file draw.h.

The documentation for this struct was generated from the following file:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/draw.h

13.22 DrawWindow_t Struct Reference

Defines the active window - establishes boundaries for drawing on the current display.

`#include <draw.h>`

Public Attributes

- K_USHORT usLeft

  Left boundary.
- K_USHORT usRight

  Right boundary.
- K_USHORT usTop

  Upper boundary.
- K_USHORT usBottom

  Bottom boundary.

13.22.1 Detailed Description

Defines the active window - establishes boundaries for drawing on the current display.

Only pixels drawn inside the surface boundaries are rendered to the output

Definition at line 159 of file draw.h.

The documentation for this struct was generated from the following file:

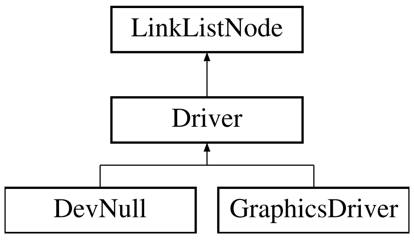
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/draw.h

13.23 Driver Class Reference

Base device-driver class used in hardware abstraction.

`#include <driver.h>`

Inheritance diagram for Driver:



Public Member Functions

- virtual void Init ()=0

  Initialize a driver, must be called prior to use.
- virtual K_UCHAR Open ()=0

  Open a device driver prior to use.
- virtual K_UCHAR Close ()=0

  Close a previously-opened device driver.

- virtual K_USHORT Read (K_USHORT usBytes_, K_UCHAR ∗pucData_)=0

  Read a specified number of bytes from the device into a specific buffer.
- virtual K_USHORT Write (K_USHORT usBytes_, K_UCHAR ∗pucData_)=0

  Write a payload of data of a given length to the device.
- virtual K_USHORT Control (K_USHORT usEvent_, void ∗pvDataIn_, K_USHORT usSizeIn_, void ∗pvDataOut_, K_USHORT usSizeOut_)=0

  This is the main entry-point for device-specific io and control operations.
- void SetName (const K_CHAR ∗pcName_)

  Set the path for the driver.
- const K_CHAR ∗ GetPath ()

  Returns a string containing the device path.

Private Attributes

- const K_CHAR ∗ m_pcPath

  string pointer that holds the driver path (name)

Additional Inherited Members

13.23.1 Detailed Description

Base device-driver class used in hardware abstraction.

All other device drivers inherit from this class

Definition at line 121 of file driver.h.

13.23.2 Member Function Documentation

13.23.2.1 K_UCHAR Driver::Close ( ) [pure virtual]

Close a previously-opened device driver.

Returns

Driver-specific return code, 0 = OK, non-0 = error

Implemented in DevNull.

13.23.2.2 K_USHORT Driver::Control (K_USHORT *usEvent_*, void ∗ *pvDataIn_*, K_USHORT *usSizeIn_*, void ∗ *pvDataOut_*, K_USHORT *usSizeOut_*) [pure virtual]

This is the main entry-point for device-specific io and control operations.

This is used for implementing all "side-channel" communications with a device, and any device-specific IO operations that do not conform to the typical POSIX read/write paradigm. Use of this funciton is analagous to the non-POSIX (yet still common) devctl() or ioctl().

Parameters

*usEvent_*	Code defining the io event (driver-specific)
*pvDataIn_*	Pointer to the intput data
*usSizeIn_*	Size of the input data (in bytes)
*pvDataOut_*	Pointer to the output data
*usSizeOut_*	Size of the output data (in bytes)

Returns

Driver-specific return code, 0 = OK, non-0 = error

Implemented in DevNull.

13.23.2.3 const K_CHAR ∗ Driver::GetPath ( ) `[inline]`

Returns a string containing the device path.

Returns

pcName_ Return the string constant representing the device path

Definition at line 231 of file driver.h.

13.23.2.4 K_UCHAR Driver::Open ( ) `[pure virtual]`

Open a device driver prior to use.

Returns

Driver-specific return code, 0 = OK, non-0 = error

Implemented in DevNull.

13.23.2.5 K_USHORT Driver::Read (K_USHORT *usBytes_*, K_UCHAR ∗ *pucData_*) `[pure virtual]`

Read a specified number of bytes from the device into a specific buffer.

Depending on the driver-specific implementation, this may be a number less than the requested number of bytes read, indicating that there there was less input than desired, or that as a result of buffering, the data may not be available.

Parameters

*usBytes_*	Number of bytes to read (<= size of the buffer)
*pucData_*	Pointer to a data buffer receiving the read data

Returns

Number of bytes actually read

Implemented in DevNull.

13.23.2.6 void Driver::SetName (const K_CHAR ∗ *pcName_*) `[inline]`

Set the path for the driver.

Name must be set prior to access (since driver access is name-based).

Parameters

*pcName_*	String constant containing the device path

Definition at line 222 of file driver.h.

13.23.2.7 K_USHORT Driver::Write (K_USHORT *usBytes_*, K_UCHAR ∗ *pucData_*) `[pure virtual]`

Write a payload of data of a given length to the device.

Depending on the implementation of the driver, the amount of data written to the device may be less than the requested number of bytes. A result less than the requested size may indicate that the device buffer is full, indicating that the user must retry the write at a later point with the remaining data.

Parameters

*usBytes_*	Number of bytes to write (<= size of the buffer)
*pucData_*	Pointer to a data buffer containing the data to write

Returns

Number of bytes actually written

Implemented in DevNull.

The documentation for this class was generated from the following file:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/driver.h

13.24 DriverList Class Reference

List of Driver objects used to keep track of all device drivers in the system.

`#include <driver.h>`

Static Public Member Functions

- static void Init ()

  Initialize the list of drivers.
- static void Add (Driver ∗pclDriver_)

  Add a Driver object to the managed global driver-list.
- static void Remove (Driver ∗pclDriver_)

  Remove a driver from the global driver list.
- static Driver ∗ FindByPath (const K_CHAR ∗m_pcPath)

  Look-up a driver in the global driver-list based on its path.

Static Private Attributes

- static DoubleLinkList m_clDriverList

  LinkedList object used to implementing the driver object management.

13.24.1 Detailed Description

List of Driver objects used to keep track of all device drivers in the system.

By default, the list contains a single entity, "/dev/null".

Definition at line 244 of file driver.h.

13.24.2 Member Function Documentation

13.24.2.1 DriverList::Add (Driver * *pclDriver_*) [inline], [static]

Add a Driver object to the managed global driver-list.

Parameters

*pclDriver_*	pointer to the driver object to add to the global driver list.

Definition at line 264 of file driver.h.

13.24.2.2 Driver * DriverList::FindByPath (const K_CHAR * *m_pcPath*) [static]

Look-up a driver in the global driver-list based on its path.

In the event that the driver is not found in the list, a pointer to the default "/dev/null" object is returned. In this way, unimplemented drivers are automatically stubbed out.

Definition at line 97 of file driver.cpp.

13.24.2.3 void DriverList::Init ( ) [static]

Initialize the list of drivers.

Must be called prior to using the device driver library.

Definition at line 88 of file driver.cpp.

13.24.2.4 void DriverList::Remove (Driver * *pclDriver_*) [inline], [static]

Remove a driver from the global driver list.

Parameters

*pclDriver_*	Pointer to the driver object to remove from the global table

Definition at line 274 of file driver.h.

The documentation for this class was generated from the following files:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/driver.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/driver.cpp

13.25 FixedHeap Class Reference

Fixed-size-block heap allocator with multiple block sizes.

`#include <fixed_heap.h>`

Public Member Functions

- void Create (void *pvHeap_, HeapConfig *pclHeapConfig_)
  Creates a heap in a provided blob of memory with lists of fixed-size blocks configured based on the associated configuration data.
- void * Alloc (K_USHORT usSize_)
  Allocate a blob of memory from the heap.

Static Public Member Functions

- static void Free (void ∗pvNode_)

  Free a previously-allocated block of memory to the heap it was originally allocated from.

Private Attributes

- HeapConfig ∗ m_paclHeaps

  Pointer to the configuration data used by the heap.

13.25.1 Detailed Description

Fixed-size-block heap allocator with multiple block sizes.

Definition at line 104 of file fixed_heap.h.

13.25.2 Member Function Documentation

13.25.2.1 void ∗ FixedHeap::Alloc (K_USHORT *usSize_*)

Allocate a blob of memory from the heap.

If no appropriately-sized data block is available, will return NULL. Note, this API is thread- safe, and interrupt safe.

Parameters

*usSize_*	Size (in bytes) to allocate from the heap

Returns

Pointer to a block of data allocated, or 0 on error.

Definition at line 130 of file fixed_heap.cpp.

13.25.2.2 void FixedHeap::Create (void ∗ *pvHeap_*, HeapConfig ∗ *pclHeapConfig_*)

Creates a heap in a provided blob of memory with lists of fixed-size blocks configured based on the associated configuration data.

A heap must be created before it can be allocated/freed.

Parameters

*pvHeap_*	Pointer to the data blob that will contain the heap
*pclHeapConfig_*	Pointer to the array of config objects that define how the heap is laid out in memory, and how many blocks of what size are included. The objects in the array must be initialized, starting from smallest block-size to largest, with the final entry in the table have a 0-block size, indicating end-of-configuration.

Definition at line 113 of file fixed_heap.cpp.

13.25.2.3 void FixedHeap::Free (void ∗ *pvNode_*) [static]

Free a previously-allocated block of memory to the heap it was originally allocated from.

This must point to the block of memory at its originally-returned pointer, and not an address within an allocated blob (as supported by some allocators).

Parameters

*pvNode_*	Pointer to the previously-allocated block of memory

Definition at line 160 of file fixed_heap.cpp.

The documentation for this class was generated from the following files:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/fixed_heap.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/fixed_heap.cpp

13.26 Font_t Struct Reference

Public Attributes

- K_UCHAR **ucSize**
- K_UCHAR **ucFlags**
- K_UCHAR **ucStartChar**
- K_UCHAR **ucMaxChar**
- K_CHAR ∗ **szName**
- const FONT_STORAGE_TYPE ∗ **pucFontData**

13.26.1 Detailed Description

Definition at line 43 of file font.h.

The documentation for this struct was generated from the following file:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/font.h

13.27 GlobalMessagePool Class Reference

Implements a list of message objects shared between all threads.

`#include <message.h>`

Static Public Member Functions

- static void Init ()

  Initialize the message queue prior to use.
- static void Push (Message ∗pclMessage_)

  Return a previously-claimed message object back to the global queue.
- static Message ∗ Pop ()

  Pop a message from the global queue, returning it to the user to be populated before sending by a transmitter.

Static Private Attributes

- static Message m_aclMessagePool [8]

  Array of message objects that make up the message pool.
- static DoubleLinkList m_clList

  Linked list used to manage the Message objects.

13.27.1 Detailed Description

Implements a list of message objects shared between all threads.

Definition at line 157 of file message.h.

13.27.2 Member Function Documentation

13.27.2.1 Message ∗ GlobalMessagePool::Pop ( ) [static]

Pop a message from the global queue, returning it to the user to be populated before sending by a transmitter.

Returns

Pointer to a Message object

Definition at line 69 of file message.cpp.

13.27.2.2 void GlobalMessagePool::Push (Message ∗ *pclMessage_*) [static]

Return a previously-claimed message object back to the global queue.

Used once the message has been processed by a receiver.

Parameters

*pclMessage_*	Pointer to the Message object to return back to the global queue

Definition at line 57 of file message.cpp.

The documentation for this class was generated from the following files:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/message.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/message.cpp

13.28 Glyph_t Struct Reference

Public Attributes

- K_UCHAR ucWidth

  Width of this font glyph in pixels.
- K_UCHAR ucHeight

  Height of this font glyph in pixels.
- K_UCHAR ucVOffset

  Vertical offset of this glyph.
- K_UCHAR aucData [1]

  Glyph data array.

13.28.1 Detailed Description

Definition at line 26 of file font.h.

The documentation for this struct was generated from the following file:

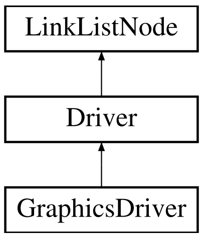
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/font.h

13.29 GraphicsDriver Class Reference

Defines the base graphics driver class, which is inherited by all other graphics drivers.

```
#include <graphics.h>
```

Inheritance diagram for GraphicsDriver:



Public Member Functions

- virtual void DrawPixel (DrawPoint_t ∗pstPoint_)

  Draw a single pixel to the display.
- virtual void ReadPixel (DrawPoint_t ∗pstPoint_)

  Read a single pixel from the display.
- virtual void ClearScreen ()

  Clear the screen (initializes to all black pixels)
- virtual void Point (DrawPoint_t ∗pstPoint_)

  Draw a pixel to the display.
- virtual void Line (DrawLine_t ∗pstLine_)

  Draw a line to the display using Bresenham's line drawing algorithm.
- virtual void Rectangle (DrawRectangle_t ∗pstRectangle_)

  Draws a rectangle on the display.
- virtual void Circle (DrawCircle_t ∗pstCircle_)

  Draw a circle to the display.
- virtual void Ellipse (DrawEllipse_t ∗pstEllipse_)

  Draw an ellipse to the display.
- virtual void Bitmap (DrawBitmap_t ∗pstBitmap_)

  Draw an RGB image on the display.
- virtual void Stamp (DrawStamp_t ∗pstStamp_)

  Draws a stamp (a 1-bit bitmap) on the display.
- virtual void Move (DrawMove_t ∗pstMove_)

  Move a the contents from one rectangle on screen to another rectangle, specified by the values of the input structure.
- virtual void TriangleWire (DrawPoly_t ∗pstPoly_)

  Draw a wireframe triangle to the display.
- virtual void TriangleFill (DrawPoly_t ∗pstPoly_)

  Draw a filled triangle to the display.
- virtual void **Polygon** (DrawPoly_t ∗pstPoly_)
- virtual void Text (DrawText_t ∗pstText_)

  Draw a string of text to the display using a bitmap font.
- virtual K_USHORT **TextWidth** (DrawText_t ∗pstText_)
- void SetWindow (DrawWindow_t ∗pstWindow_)

  Set the drawable window of the screen.
- void ClearWindow ()

  Clear the window - resetting the boundaries to the entire drawable area of the screen.

Protected Attributes

- K_USHORT **m_usResX**
- K_USHORT **m_usResY**
- K_USHORT **m_usLeft**
- K_USHORT **m_usTop**
- K_USHORT **m_usRight**
- K_USHORT **m_usBottom**
- K_UCHAR **m_ucBPP**

Additional Inherited Members

13.29.1 Detailed Description

Defines the base graphics driver class, which is inherited by all other graphics drivers.

Per-pixel rendering functions for all raster operations is provided by default. These can be overridden with more efficient hardware-supported operations where available.

Definition at line 32 of file graphics.h.

13.29.2 Member Function Documentation

13.29.2.1 void GraphicsDriver::Bitmap (**DrawBitmap_t** ∗ *pstBitmap_*) `[virtual]`

Draw an RGB image on the display.

Parameters

*pstBitmap_*	- pointer to the bitmap object to display

Definition at line 300 of file graphics.cpp.

13.29.2.2 void GraphicsDriver::Circle (**DrawCircle_t** ∗ *pstCircle_*) `[virtual]`

Draw a circle to the display.

Parameters

*pstCircle_*	- pointer to the circle to draw

Definition at line 176 of file graphics.cpp.

13.29.2.3 void GraphicsDriver::DrawPixel (**DrawPoint_t** ∗ *pstPoint_*) `[inline]`, `[virtual]`

Draw a single pixel to the display.

Parameters

*pstPoint_*	Structure containing the pixel data (color/location) to be written.

Definition at line 49 of file graphics.h.

13.29.2.4 void GraphicsDriver::Ellipse (DrawEllipse_t * *pstEllipse_*) `[virtual]`

Draw an ellipse to the display.

Parameters

*pstEllipse_*	- pointer to the ellipse to draw on the display

Definition at line 248 of file graphics.cpp.

13.29.2.5 void GraphicsDriver::Line (DrawLine_t * *pstLine_*) `[virtual]`

Draw a line to the display using Bresenham's line drawing algorithm.

Parameters

*pstLine_*	- pointer to the line structure

Definition at line 48 of file graphics.cpp.

13.29.2.6 void GraphicsDriver::Move (DrawMove_t * *pstMove_*) `[virtual]`

Move a the contents from one rectangle on screen to another rectangle, specified by the values of the input structure.

Parameters

*pstMove_*	- object describing the graphics movement operation (framebuffer operations only).

Definition at line 438 of file graphics.cpp.

13.29.2.7 void GraphicsDriver::Point (DrawPoint_t * *pstPoint_*) `[virtual]`

Draw a pixel to the display.

Parameters

*pstPoint_*	- pointer to the struct containing the pixel to draw

Definition at line 42 of file graphics.cpp.

13.29.2.8 void GraphicsDriver::ReadPixel (DrawPoint_t * *pstPoint_*) `[inline], [virtual]`

Read a single pixel from the display.

Parameters

*pstPoint_*	Structure containing the pixel location of the pixel to be read. The color value will contain the value from the display when read.

Definition at line 58 of file graphics.h.

13.29.2.9 void GraphicsDriver::Rectangle (DrawRectangle_t * *pstRectangle_*) `[virtual]`

Draws a rectangle on the display.

Parameters

*pstRectangle_*	- pointer to the rectangle struct

Definition at line 131 of file graphics.cpp.

13.29.2.10 void GraphicsDriver::SetWindow (**DrawWindow_t** ∗ *pstWindow_*)

Set the drawable window of the screen.

Parameters

*pstWindow_*	- pointer to the window struct defining the drawable area

Definition at line 882 of file graphics.cpp.

13.29.2.11 void GraphicsDriver::Stamp (**DrawStamp_t** ∗ *pstStamp_*) `[virtual]`

Draws a stamp (a 1-bit bitmap) on the display.

Parameters

*pstStamp_*	- pointer to the stamp object to draw

Definition at line 399 of file graphics.cpp.

13.29.2.12 void GraphicsDriver::Text (**DrawText_t** ∗ *pstText_*) `[virtual]`

Draw a string of text to the display using a bitmap font.

Parameters

*pstText_*	- pointer to the text object to render

Definition at line 499 of file graphics.cpp.

13.29.2.13 void GraphicsDriver::TriangleFill (**DrawPoly_t** ∗ *pstPoly_*) `[virtual]`

Draw a filled triangle to the display.

Parameters

*pstPoly_*	Pointer to the polygon to draw.

Definition at line 655 of file graphics.cpp.

13.29.2.14 void GraphicsDriver::TriangleWire (**DrawPoly_t** ∗ *pstPoly_*) `[virtual]`

Draw a wireframe triangle to the display.

Parameters

*pstPoly_*	Pointer to the polygon to draw.

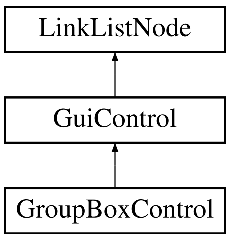
Definition at line 630 of file graphics.cpp.

The documentation for this class was generated from the following files:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/graphics.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/graphics.cpp

13.30 GroupBoxControl Class Reference

Inheritance diagram for GroupBoxControl:



Public Member Functions

- virtual void Init ()

    Initiailize the control - must be called before use.
- virtual void Draw ()

    Redraw the control "cleanly".
- virtual GuiReturn_t ProcessEvent (GuiEvent_t ∗pstEvent_)

    Process an event sent to the control.
- virtual void Activate (bool bActivate_)

    Activate or deactivate the current control - used when switching from one active control to another.
- void **SetPanelColor** (COLOR eColor_)
- void **SetLineColor** (COLOR eColor_)
- void **SetFontColor** (COLOR eColor_)
- void **SetFont** (Font_t ∗pstFont_)
- void **SetCaption** (const K_CHAR ∗pcCaption_)

Private Attributes

- COLOR **m_uPanelColor**
- COLOR **m_uLineColor**
- COLOR **m_uFontColor**
- Font_t ∗ **m_pstFont**
- const K_CHAR ∗ **m_pcCaption**

Additional Inherited Members

13.30.1 Detailed Description

Definition at line 29 of file control_groupbox.h.

13.30.2 Member Function Documentation

13.30.2.1 virtual void GroupBoxControl::Activate (bool *bActivate_*) `[inline], [virtual]`

Activate or deactivate the current control - used when switching from one active control to another.

Parameters

*bActivate_*	- true to activate, false to deactivate

Implements GuiControl.

Definition at line 38 of file control_groupbox.h.

13.30.2.2 void GroupBoxControl::Draw ( ) `[virtual]`

Redraw the control "cleanly".

Subclass specific.

Implements GuiControl.

Definition at line 30 of file control_groupbox.cpp.

13.30.2.3 virtual void GroupBoxControl::Init ( ) `[inline], [virtual]`

Initiailize the control - must be called before use.

Implementation is subclass specific.

Implements GuiControl.

Definition at line 32 of file control_groupbox.h.

13.30.2.4 virtual GuiReturn_t GroupBoxControl::ProcessEvent (GuiEvent_t * *pstEvent_*) `[inline], [virtual]`

Process an event sent to the control.

Subclass specific implementation.

Parameters

*pstEvent_*	Pointer to a struct containing the event data

Implements GuiControl.

Definition at line 37 of file control_groupbox.h.

The documentation for this class was generated from the following files:

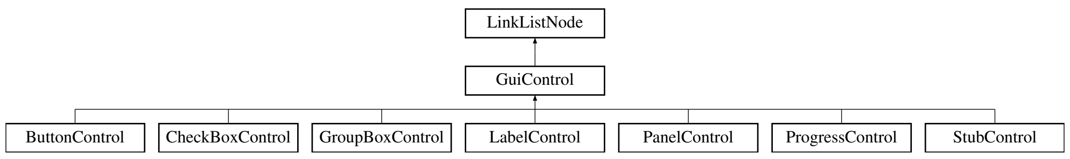
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_groupbox.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_groupbox.cpp

13.31 GuiControl Class Reference

GUI Control Base Class.

`#include <gui.h>`

Inheritance diagram for GuiControl:



Public Member Functions

- virtual void Init ()=0

  Initiailize the control - must be called before use.
- virtual void Draw ()=0

  Redraw the control "cleanly".
- virtual GuiReturn_t ProcessEvent (GuiEvent_t ∗pstEvent_)=0

  Process an event sent to the control.
- void SetTop (K_USHORT usTop_)

  Set the location of the topmost pixel of the control.
- void SetLeft (K_USHORT usLeft_)

  Set the location of the leftmost pixel of the control.
- void SetHeight (K_USHORT usHeight_)

  Set the height of the control (in pixels)
- void SetWidth (K_USHORT usWidth_)

  Set the width of the control (in pixels)
- void SetZOrder (K_UCHAR ucZ_)

  Set the Z-order (depth) of the control.
- void SetControlIndex (K_UCHAR ucIdx_)

  Set the index of the control, used for cycling through focus (ala tab order in VB).
- K_USHORT GetTop ()

  Return the topmost pixel of the control.
- K_USHORT GetLeft ()

  Return the leftmost pixel of the control.
- K_USHORT GetHeight ()

  Get the height of the control in pixels.
- K_USHORT GetWidth ()

  Get the width of the control in pixels.
- K_UCHAR GetZOrder ()

  Return the Z-order of the control.
- K_UCHAR GetControlIndex ()

  Return the Control Index of the control.
- K_BOOL IsStale ()

  Return whether or not the control needs to be redrawn or not.
- void GetControlOffset (K_USHORT ∗pusX_, K_USHORT ∗pusY_)

  Return the absolute offset of the control within an event surface.
- K_BOOL IsInFocus ()

  Return whether or not the current control has the focus in the window.
- virtual void Activate (bool bActivate_)=0

  Activate or deactivate the current control - used when switching from one active control to another.

Protected Member Functions

- void SetParentControl (GuiControl ∗pclParent_)

  Set the parent control of this control.
- void SetParentWindow (GuiWindow ∗pclWindow_)

  Set the parent window of this control.
- GuiControl ∗ GetParentControl ()

  Return the pointer to the control's currently-assigned parent control.
- GuiWindow ∗ GetParentWindow ()

  Get the parent window of this control.
- void ClearStale ()

  Clear the stale flag for this control.
- void SetStale ()

  Signal that the object needs to be redrawn.
- void SetAcceptFocus (bool bFocus_)

  Tell the control whether or not to accept focus.
- bool AcceptsFocus ()

  Returns whether or not this control accepts focus.

Private Attributes

- K_BOOL m_bStale

  true if the control is stale and needs to be redrawn, false otherwise
- K_BOOL m_bAcceptsFocus

  Whether or not the control accepts focus or not.
- K_UCHAR m_ucZOrder

  The Z-Order (depth) of the control.
- K_UCHAR m_ucControlIndex

  Index of the control in the window.
- K_USHORT m_usTop

  Topmost location of the control on the window.
- K_USHORT m_usLeft

  Leftmost location of the control on the window.
- K_USHORT m_usWidth

  Width of the control in pixels.
- K_USHORT m_usHeight

  Height of the control in pixels.
- GuiControl ∗ m_pclParentControl

  Pointer to the parent control.
- GuiWindow ∗ m_pclParentWindow

  Pointer to the parent window associated with this control.

Friends

- class **GuiWindow**
- class **GuiEventSurface**

Additional Inherited Members

13.31.1 Detailed Description

GUI Control Base Class.

This class is the common ancestor to all GUI control elements. It defines a base set of properties common to all controls, as well as methods for initialization, event handling, and redrawing. Controls are directly related to Windows, which are used to manage and organize controls.

Definition at line 521 of file gui.h.

13.31.2 Member Function Documentation

13.31.2.1 void GuiControl::Activate (bool *bActivate_*) `[pure virtual]`

Activate or deactivate the current control - used when switching from one active control to another.

Parameters

*bActivate_*	- true to activate, false to deactivate

Implemented in StubControl, LabelControl, ButtonControl, PanelControl, GroupBoxControl, ProgressControl, and CheckBoxControl.

13.31.2.2 void GuiControl::ClearStale ( ) `[inline]`, `[protected]`

Clear the stale flag for this control.

Should only be done after a redraw has been completed

Definition at line 724 of file gui.h.

13.31.2.3 void GuiControl::Draw ( ) `[pure virtual]`

Redraw the control "cleanly".

Subclass specific.

Implemented in StubControl, LabelControl, ButtonControl, PanelControl, GroupBoxControl, ProgressControl, and CheckBoxControl.

13.31.2.4 K_UCHAR GuiControl::GetControlIndex ( ) `[inline]`

Return the Control Index of the control.

Returns

The control index of the control

Definition at line 631 of file gui.h.

13.31.2.5 void GuiControl::GetControlOffset (K_USHORT ∗ *pusX_*, K_USHORT ∗ *pusY_*)

Return the absolute offset of the control within an event surface.

This function will traverse through all of the object's parents, and their parents, until the root control and root window are identified. The absolute pixel locations of the Topmost (Y) and Leftmost (X) pixels are populated in the

Parameters

*pusX_*	Pointer to the K_USHORT containing the leftmost pixel
*pusY_*	Pointer to the K_USHORT containing the topmost pixel

Definition at line 543 of file gui.cpp.

13.31.2.6 K_USHORT GuiControl::GetHeight ( ) [inline]

Get the height of the control in pixels.

Returns

Height of the control in pixels

Definition at line 610 of file gui.h.

13.31.2.7 K_USHORT GuiControl::GetLeft ( ) [inline]

Return the leftmost pixel of the control.

Returns

Leftmost pixel of the control

Definition at line 603 of file gui.h.

13.31.2.8 GuiControl ∗ GuiControl::GetParentControl ( ) [inline], [protected]

Return the pointer to the control's currently-assigned parent control.

Returns

Pointer to the Control's currently assigned parent control.

Definition at line 708 of file gui.h.

13.31.2.9 GuiWindow ∗ GuiControl::GetParentWindow ( ) [inline], [protected]

Get the parent window of this control.

Returns

Pointer to the control's window

Definition at line 716 of file gui.h.

13.31.2.10 K_USHORT GuiControl::GetTop ( ) [inline]

Return the topmost pixel of the control.

Returns

Topmost pixel of the control

Definition at line 596 of file gui.h.

13.31.2.11 K_USHORT GuiControl::GetWidth ( ) [inline]

Get the width of the control in pixels.

Returns

Width of the control in pixels

Definition at line 617 of file gui.h.

13.31.2.12 K_UCHAR GuiControl::GetZOrder ( ) [inline]

Return the Z-order of the control.

Returns

Z-order of the control

Definition at line 624 of file gui.h.

13.31.2.13 void GuiControl::Init ( ) [pure virtual]

Initiailize the control - must be called before use.

Implementation is subclass specific.

Implemented in StubControl, ButtonControl, PanelControl, LabelControl, ProgressControl, CheckBoxControl, and GroupBoxControl.

13.31.2.14 K_BOOL GuiControl::IsInFocus ( ) [inline]

Return whether or not the current control has the focus in the window.

Returns

true if this control is in focus, false otherwise

Definition at line 660 of file gui.h.

13.31.2.15 K_BOOL GuiControl::IsStale ( ) [inline]

Return whether or not the control needs to be redrawn or not.

Returns

true - control needs redrawing, false - control is intact.

Definition at line 638 of file gui.h.

13.31.2.16 GuiReturn_t GuiControl::ProcessEvent (GuiEvent_t * *pstEvent_*) [pure virtual]

Process an event sent to the control.

Subclass specific implementation.

Parameters

*pstEvent_*	Pointer to a struct containing the event data

Implemented in StubControl, LabelControl, ButtonControl, PanelControl, GroupBoxControl, ProgressControl, and CheckBoxControl.

13.31.2.17 void GuiControl::SetControlIndex (K_UCHAR *ucIdx_*) [inline]

Set the index of the control, used for cycling through focus (ala tab order in VB).

Parameters

*ucIdx_*	Focus index of the control

Definition at line 589 of file gui.h.

13.31.2.18 void GuiControl::SetHeight (K_USHORT *usHeight_*) [inline]

Set the height of the control (in pixels)

Parameters

*usHeight_*	Height of the control in pixels

Definition at line 567 of file gui.h.

13.31.2.19 void GuiControl::SetLeft (K_USHORT *usLeft_*) [inline]

Set the location of the leftmost pixel of the control.

Parameters

*usLeft_*	Leftmost pixel of the control

Definition at line 560 of file gui.h.

13.31.2.20 void GuiControl::SetParentControl (GuiControl * *pclParent_*) [inline], [protected]

Set the parent control of this control.

When a control has its parent set, it is considered "nested" within that control. Moving the control will thus result in all of its child controls to become invalidated, thus requiring redraws. The control's object offsets (Top, Bottom, Height, and Width) also become relative to the origin of the parent control.

Parameters

*pclParent_*	Pointer to the control's parent control

Definition at line 689 of file gui.h.

13.31.2.21 void GuiControl::SetParentWindow (GuiWindow * *pclWindow_*) [inline], [protected]

Set the parent window of this control.

All controls within the same window are all associated together, and share events targetted towards a specific window. Event tabbing, focus, and Z-ordering is also shared between controls within a window.

Parameters

*pclWindow_*	Pointer to the control's parent window.

Definition at line 700 of file gui.h.

13.31.2.22 void GuiControl::SetTop (K_USHORT *usTop_*) [inline]

Set the location of the topmost pixel of the control.

Parameters

*usTop_*	Topmost pixel of the control

Definition at line 553 of file gui.h.

13.31.2.23 void GuiControl::SetWidth (K_USHORT *usWidth_*) [inline]

Set the width of the control (in pixels)

Parameters

*usWidth_*	Width of the control in pixels

Definition at line 574 of file gui.h.

13.31.2.24 void GuiControl::SetZOrder (K_UCHAR *ucZ_*) [inline]

Set the Z-order (depth) of the control.

Parameters

*ucZ_*	Z order of the control

Definition at line 581 of file gui.h.

13.31.3 Member Data Documentation

13.31.3.1 K_UCHAR GuiControl::m_ucControlIndex [private]

Index of the control in the window.

This is used for setting focus when transitioning from control to control on a window

Definition at line 753 of file gui.h.

13.31.3.2 K_UCHAR GuiControl::m_ucZOrder [private]

The Z-Order (depth) of the control.

Only the highest order controls are visible at any given location

Definition at line 749 of file gui.h.

The documentation for this class was generated from the following files:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/gui.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/gui.cpp

13.32 GuiEvent_t Struct Reference

Public Attributes

- K_UCHAR **ucEventType**
- K_UCHAR **ucTargetID**
- union {
  KeyEvent_t **stKey**
  MouseEvent_t **stMouse**
  TouchEvent_t **stTouch**
  JoystickEvent_t **stJoystick**
  TimerEvent_t **stTimer**
  };

13.32.1 Detailed Description

Definition at line 183 of file gui.h.

The documentation for this struct was generated from the following file:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/gui.h

13.33 GuiEventSurface Class Reference

GUI Event Surface Object.

`#include <gui.h>`

Public Member Functions

- void Init ()

  Initialize an event surface before use.
- void AddWindow (GuiWindow ∗pclWindow_)

  Add a window to the event surface.
- void RemoveWindow (GuiWindow ∗pclWindow_)

  Remove a window from the event surface.
- K_BOOL SendEvent (GuiEvent_t ∗pstEvent_)

  Send an event to this window surface.
- K_BOOL ProcessEvent ()

  Process an event in the event queue.
- K_UCHAR GetEventCount ()

  Get the count of pending events in the event surface's queue.
- GuiWindow ∗ FindWindowByName (const K_CHAR ∗szName_)

  Return a pointer to a window by name, or NULL on failure.

Private Member Functions

- void CopyEvent (GuiEvent_t ∗pstDst_, GuiEvent_t ∗pstSrc_)

  Copy the contents of one message structure to another.

Private Attributes

- DoubleLinkList m_clWindowList

  List of windows managed on this event surface.
- MessageQueue m_clMessageQueue

  Message queue used to manage window events.

13.33.1 Detailed Description

GUI Event Surface Object.

An event surface is the lowest-level UI object. It maintains a list of windows which are associated with it, and manages the transmission and routing of events to each window, and their appropriate controls

All windows located on the event surface are assumed to share a common display, and coordinate frame. In this way, multiple GUIs can be implemented in the system, each tied to separate physical or virtual displays.

Definition at line 441 of file gui.h.

13.33.2 Member Function Documentation

13.33.2.1 void GuiEventSurface::AddWindow (GuiWindow ∗ *pclWindow_*)

Add a window to the event surface.

Parameters

*pclWindow_*	Pointer to the window object to add to the sruface

Definition at line 409 of file gui.cpp.

13.33.2.2 void GuiEventSurface::CopyEvent (GuiEvent_t ∗ *pstDst_*, GuiEvent_t ∗ *pstSrc_*) [private]

Copy the contents of one message structure to another.

Parameters

*pstDst_*	Destination event pointer
*pstSrc_*	Source event pointer

Definition at line 530 of file gui.cpp.

13.33.2.3 void GuiEventSurface::Init ( ) [inline]

Initialize an event surface before use.

Must be called prior to any other object methods.

Definition at line 448 of file gui.h.

13.33.2.4 K_BOOL GuiEventSurface::ProcessEvent ( )

Process an event in the event queue.

If no events are pending, the call will block until an event is available.

Definition at line 461 of file gui.cpp.

13.33.2.5 void GuiEventSurface::RemoveWindow (GuiWindow ∗ *pclWindow_*)

Remove a window from the event surface.

Parameters

*pclWindow_*	Pointer to the window object to remove from the surface

Definition at line 417 of file gui.cpp.

13.33.2.6 K_BOOL GuiEventSurface::SendEvent (GuiEvent_t ∗ *pstEvent_*)

Send an event to this window surface.

The event will be forwraded to all windows managed by this service.

Parameters

*pstEvent_*	Pointer to an event to send

Returns

true on success, false on failure

Definition at line 425 of file gui.cpp.

The documentation for this class was generated from the following files:

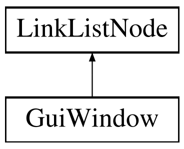
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/gui.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/gui.cpp

13.34 GuiWindow Class Reference

Basic Window Class.

`#include <gui.h>`

Inheritance diagram for GuiWindow:



Public Member Functions

- void Init ()

  Initialize the GUI Window object prior to use.
- void SetDriver (GraphicsDriver ∗pclDriver_)

  Set the graphics driver to use for rendering controls on the window.
- GraphicsDriver ∗ GetDriver ()

  Set the graphics driver to use for rendering controls on the window.
- void AddControl (GuiControl ∗pclControl_, GuiControl ∗pclParent_)

  Assign a GUI Control to this window object.

- void RemoveControl (GuiControl ∗pclControl_)

  Removes a previously-added control from the Window.
- K_UCHAR GetMaxZOrder ()

  Returns the highest Z-Order of all controls attached to this window.
- void Redraw (K_BOOL bRedrawAll_)

  Redraw objects in the window.
- void ProcessEvent (GuiEvent_t ∗pstEvent_)

  Process an event sent to this window.
- void SetFocus (GuiControl ∗pclControl_)

  Set the control used to accept "focus" events.
- K_BOOL IsInFocus (GuiControl ∗pclControl_)

  Return whether or not the selected control is in focus or not.
- void SetTop (K_USHORT usTop_)

  Set the location of the topmost pixel of the window.
- void SetLeft (K_USHORT usLeft_)

  Set the location of the leftmost pixel of the window.
- void SetHeight (K_USHORT usHeight_)

  Set the height of the window (in pixels)
- void SetWidth (K_USHORT usWidth_)

  Set the width of the window (in pixels)
- K_USHORT GetTop ()

  Return the topmost pixel of the window.
- K_USHORT GetLeft ()

  Return the leftmost pixel of the window.
- K_USHORT GetHeight ()

  Get the height of the window in pixels.
- K_USHORT GetWidth ()

  Get the width of the window in pixels.
- K_UCHAR GetZOrder ()

  Get the Z-order of the window on the event surface.
- void SetZOrder (K_UCHAR ucZ_)

  Set the Z-order of the window on the event surface.
- void CycleFocus (bool bForward_)

  Cycle the focus to the next active control in the window.
- void SetName (const K_CHAR ∗szName_)

  Set the name for this window.
- const K_CHAR ∗ GetName ()

  Return the name of this window.

Private Attributes

- K_USHORT m_usTop

  Topmost pixel of the window on the event surface.
- K_USHORT m_usLeft

  Leftmost pixel of the window on the event surface.
- K_USHORT m_usHeight

  Height of the window in pixels.
- K_USHORT m_usWidth

  Width of the window in pixels.
- K_UCHAR m_ucZ

Z-order of the window on the event surface.

- const K_CHAR ∗ m_szName

  Name applied to this window.
- DoubleLinkList m_clControlList

  List of controls managed by this window.
- GuiControl ∗ m_pclInFocus

  Pointer to the control in event focus.
- K_UCHAR m_ucControlCount

  Number of controls in this window.
- GraphicsDriver ∗ m_pclDriver

  Graphics driver for this window.

Additional Inherited Members

13.34.1 Detailed Description

Basic Window Class.

A Window is loosely defined as a container of controls, all sharing a coordinate reference coordinate frame. Events are managed on a per-window basis, and each window is isolated from eachother.

Definition at line 219 of file gui.h.

13.34.2 Member Function Documentation

13.34.2.1 GuiWindow::AddControl (**GuiControl** ∗ *pclControl_*, **GuiControl** ∗ *pclParent_*)

Assign a GUI Control to this window object.

Adding an object to a window ensures that the object will be drawn on the specific window surface, and ensures that events directed to this window will be forwarded to the controls appropriately.

Parameters

*pclControl_*	Pointer to the control object to add
*pclParent_*	Pointer to the control's "parent" object (or NULL)

Definition at line 28 of file gui.cpp.

13.34.2.2 void GuiWindow::CycleFocus (bool *bForward_*)

Cycle the focus to the next active control in the window.

Parameters

*bForward_*	- Cycle to the next control when true, previous control when false

Definition at line 279 of file gui.cpp.

13.34.2.3 **GraphicsDriver** ∗ GuiWindow::GetDriver ( ) `[inline]`

Set the graphics driver to use for rendering controls on the window.

Returns

Pointer to the Window's graphics driver

Definition at line 248 of file gui.h.

13.34.2.4 K_USHORT GuiWindow::GetHeight ( ) [inline]

Get the height of the window in pixels.

Returns

Height of the window in pixels

Definition at line 375 of file gui.h.

13.34.2.5 K_USHORT GuiWindow::GetLeft ( ) [inline]

Return the leftmost pixel of the window.

Returns

Leftmost pixel of the window

Definition at line 368 of file gui.h.

13.34.2.6 K_UCHAR GuiWindow::GetMaxZOrder ( )

Returns the highest Z-Order of all controls attached to this window.

Returns

The highest Z-Order used by controls in this window

Definition at line 62 of file gui.cpp.

13.34.2.7 K_USHORT GuiWindow::GetTop ( ) [inline]

Return the topmost pixel of the window.

Returns

Topmost pixel of the window

Definition at line 361 of file gui.h.

13.34.2.8 K_USHORT GuiWindow::GetWidth ( ) [inline]

Get the width of the window in pixels.

Returns

Width of the window in pixels

Definition at line 382 of file gui.h.

13.34.2.9 void GuiWindow::Init ( ) [inline]

Initialize the GUI Window object prior to use.

Must be called before calling other methods on this object

Definition at line 227 of file gui.h.

13.34.2.10 K_BOOL GuiWindow::IsInFocus (GuiControl ∗ *pclControl_*) [inline]

Return whether or not the selected control is in focus or not.

Parameters

*pclControl_*	Pointer to the control object to evaluate

Returns

true - the selected control is the active control on the window false - otherwise

Definition at line 319 of file gui.h.

13.34.2.11 void GuiWindow::ProcessEvent (GuiEvent_t ∗ *pstEvent_*)

Process an event sent to this window.

This method handles all of the plumbing required to target the event towards specific controls, or all controls in the window depending on the event payload.

Definition at line 128 of file gui.cpp.

13.34.2.12 void GuiWindow::Redraw (K_BOOL *bRedrawAll_*)

Redraw objects in the window.

Typically, only the affected controls will need to be redrawn, but in some cases (such as window initialization), the entire window will need to be redrawn cleanly. This behavior is defined by the value of the bRedrawAll_ parameter.

Definition at line 86 of file gui.cpp.

13.34.2.13 GuiWindow::RemoveControl (GuiControl ∗ *pclControl_*)

Removes a previously-added control from the Window.

Parameters

*pclControl_*	Pointer to the control object to remove

Definition at line 41 of file gui.cpp.

13.34.2.14 void GuiWindow::SetDriver (GraphicsDriver ∗ *pclDriver_*) [inline]

Set the graphics driver to use for rendering controls on the window.

Parameters

*pclDriver_*	Pointer to the graphics driver

Definition at line 240 of file gui.h.

13.34.2.15 void GuiWindow::SetFocus (GuiControl * *pclControl_*)

Set the control used to accept "focus" events.

Such events include keyboard events.

Parameters

*pclControl_*	Pointer to the control object to set focus on.

Definition at line 271 of file gui.cpp.

13.34.2.16 void GuiWindow::SetHeight (K_USHORT *usHeight_*) [inline]

Set the height of the window (in pixels)

Parameters

*usHeight_*	Height of the window in pixels

Definition at line 347 of file gui.h.

13.34.2.17 void GuiWindow::SetLeft (K_USHORT *usLeft_*) [inline]

Set the location of the leftmost pixel of the window.

Parameters

*usLeft_*	Leftmost pixel of the window

Definition at line 340 of file gui.h.

13.34.2.18 void GuiWindow::SetTop (K_USHORT *usTop_*) [inline]

Set the location of the topmost pixel of the window.

Parameters

*usTop_*	Topmost pixel of the window

Definition at line 333 of file gui.h.

13.34.2.19 void GuiWindow::SetWidth (K_USHORT *usWidth_*) [inline]

Set the width of the window (in pixels)

Parameters

*usWidth_*	Width of the window in pixels

Definition at line 354 of file gui.h.

13.34.3 Member Data Documentation

13.34.3.1 GraphicsDriver* GuiWindow::m_pclDriver [private]

Graphics driver for this window.

Definition at line 425 of file gui.h.

The documentation for this class was generated from the following files:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/gui.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/gui.cpp

13.35 HeapConfig Class Reference

Heap configuration object.

```
#include <fixed_heap.h>
```

Public Attributes

- K_USHORT m_usBlockSize

    Block size in bytes.
- K_USHORT m_usBlockCount

    Number of blocks to create @ this size.

Protected Attributes

- BlockHeap m_clHeap

    BlockHeap object used by the allocator.

Friends

- class **FixedHeap**

13.35.1 Detailed Description

Heap configuration object.

Definition at line 90 of file fixed_heap.h.

The documentation for this class was generated from the following file:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/fixed_heap.h

13.36 JoystickEvent_t Struct Reference

Joystick UI event structure.

```
#include <gui.h>
```

Public Attributes

- union {
    K_USHORT usRawData
        Raw joystick data.
    struct {
        unsigned int bUp:1
            D-pad UP state.
        unsigned int bDown:1
            D-pad DOWN state.
        unsigned int bLeft:1
            D-pad LEFT state.
        unsigned int bRight:1
            D-pad RIGHT state.
        unsigned int bButton1:1
            Joystick Button1 state.
        unsigned int bButton2:1
            Joystick Button2 state.
        unsigned int bButton3:1
            Joystick Button3 state.
        unsigned int bButton4:1
            Joystick Button4 state.
        unsigned int bButton5:1
            Joystick Button5 state.
        unsigned int bButton6:1
            Joystick Button6 state.
        unsigned int bButton7:1
            Joystick Button7 state.
        unsigned int bButton8:1
            Joystick Button8 state.
        unsigned int bButton9:1
            Joystick Button9 state.
        unsigned int bButton10:1
            Joystick Button10 state.
        unsigned int bSelect:1
            Start button state.
        unsigned int bStart:1
            Select button state.
    }
  };

13.36.1 Detailed Description

Joystick UI event structure.

Definition at line 144 of file gui.h.

The documentation for this struct was generated from the following file:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/gui.h

13.37 Kernel Class Reference

Class that encapsulates all of the kernel startup functions.

```
#include <kernel.h>
```

Static Public Member Functions

- static void Init (void)

  Kernel Initialization Function, call before any other OS function.
- static void Start (void)

  Start the kernel; function never returns.

13.37.1 Detailed Description

Class that encapsulates all of the kernel startup functions.

Definition at line 40 of file kernel.h.

13.37.2 Member Function Documentation

13.37.2.1 Kernel::Init (void) `[static]`

Kernel Initialization Function, call before any other OS function.

Initializes all global resources used by the operating system. This must be called before any other kernel function is invoked.

Definition at line 43 of file kernel.cpp.

13.37.2.2 Kernel::Start (void) `[static]`

Start the kernel; function never returns.

Start the operating system kernel - the current execution context is cancelled, all kernel services are started, and the processor resumes execution at the entrypoint for the highest-priority thread.

You must have at least one thread added to the kernel before calling this function, otherwise the behavior is undefined.

Definition at line 68 of file kernel.cpp.

The documentation for this class was generated from the following files:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kernel.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kernel.cpp

13.38 KernelSWI Class Reference

Class providing the software-interrupt required for context-switching in the kernel.

```
#include <kernelswi.h>
```

Static Public Member Functions

- static void Config (void)

  Configure the software interrupt - must be called before any other software interrupt functions are called.
- static void Start (void)

  Enable ("Start") the software interrupt functionality.
- static void Stop (void)

  Disable the software interrupt functionality.
- static void Clear (void)

Clear the software interrupt.

- static void Trigger (void)

  Call the software interrupt.

- static K_UCHAR DI ()

  Disable the SWI flag itself.

- static void RI (K_UCHAR bEnable_)

  Restore the state of the SWI to the value specified.

13.38.1 Detailed Description

Class providing the software-interrupt required for context-switching in the kernel.

Definition at line 32 of file kernelswi.h.

13.38.2 Member Function Documentation

13.38.2.1 K_UCHAR KernelSWI::DI ( ) `[static]`

Disable the SWI flag itself.

Returns

previous status of the SWI, prior to the DI call

Definition at line 50 of file kernelswi.cpp.

13.38.2.2 void KernelSWI::RI (K_UCHAR *bEnable_*) `[static]`

Restore the state of the SWI to the value specified.

Parameters

*bEnable_*	true - enable the SWI, false - disable SWI

Definition at line 58 of file kernelswi.cpp.

The documentation for this class was generated from the following files:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kernelswi.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kernelswi.cpp

13.39 KernelTimer Class Reference

Hardware timer interface, used by all scheduling/timer subsystems.

`#include <kerneltimer.h>`

Static Public Member Functions

- static void Config (void)

  Initializes the kernel timer before use.

- static void Start (void)

  Starts the kernel time (must be configured first)

- static void Stop (void)

    Shut down the kernel timer, used when no timers are scheduled.
- static K_UCHAR DI (void)

    Disable the kernel timer's expiry interrupt.
- static void RI (K_UCHAR bEnable_)

    Retstore the state of the kernel timer's expiry interrupt.
- static void EI (void)

    Enable the kernel timer's expiry interrupt.
- static K_ULONG SubtractExpiry (K_ULONG ulInterval_)

    Subtract the specified number of ticks from the timer's expiry count register.
- static K_ULONG TimeToExpiry (void)

    Returns the number of ticks remaining before the next timer expiry.
- static K_ULONG SetExpiry (K_ULONG ulInterval_)

    Resets the kernel timer's expiry interval to the specified value.
- static K_ULONG GetOvertime (void)

    Return the number of ticks that have elapsed since the last expiry.
- static void ClearExpiry (void)

    Clear the hardware timer expiry register.

Static Private Member Functions

- static K_USHORT Read (void)

    Safely read the current value in the timer register.

13.39.1 Detailed Description

Hardware timer interface, used by all scheduling/timer subsystems.

Definition at line 33 of file kerneltimer.h.

13.39.2 Member Function Documentation

13.39.2.1 K_ULONG KernelTimer::GetOvertime (void) [static]

Return the number of ticks that have elapsed since the last expiry.

Returns

Number of ticks that have elapsed after last timer expiration

Definition at line 94 of file kerneltimer.cpp.

13.39.2.2 K_USHORT KernelTimer::Read (void) [static], [private]

Safely read the current value in the timer register.

Returns

Value held in the timer register

Definition at line 57 of file kerneltimer.cpp.

13.39.2.3 void KernelTimer::RI (K_UCHAR *bEnable_*) [static]

Retstore the state of the kernel timer's expiry interrupt.

Parameters

*bEnable_*	1 enable, 0 disable

Definition at line 137 of file kerneltimer.cpp.

13.39.2.4 K_ULONG KernelTimer::SetExpiry (K_ULONG *ulInterval_*) [static]

Resets the kernel timer's expiry interval to the specified value.

Parameters

*ulInterval_*	Desired interval in ticks to set the timer for

Returns

Actual number of ticks set (may be less than desired)

Definition at line 100 of file kerneltimer.cpp.

13.39.2.5 K_ULONG KernelTimer::SubtractExpiry (K_ULONG *ulInterval_*) [static]

Subtract the specified number of ticks from the timer's expiry count register.

Returns the new expiry value stored in the register.

Parameters

*ulInterval_*	Time (in HW-specific) ticks to subtract

Returns

Value in ticks stored in the timer's expiry register

Definition at line 71 of file kerneltimer.cpp.

13.39.2.6 K_ULONG KernelTimer::TimeToExpiry (void) [static]

Returns the number of ticks remaining before the next timer expiry.

Parameters

Time	before next expiry in platform-specific ticks

Definition at line 78 of file kerneltimer.cpp.

The documentation for this class was generated from the following files:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kerneltimer.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kerneltimer.cpp

13.40 KeyEvent_t Struct Reference

Keyboard UI event structure definition.

```
#include <gui.h>
```

Public Attributes

- K_UCHAR ucKeyCode

  8-bit value representing a keyboard scan code
- union {

    K_UCHAR ucFlags

      Flags indicating modifiers to the event.

    struct {

      unsigned int bKeyState:1

        Key is being pressed or released.

      unsigned int bShiftState:1

        Whether or not shift is pressed.

      unsigned int bCtrlState:1

        Whether or not CTRL is pressed.

      unsigned int bAltState:1

        Whether or not ALT it pressed.

      unsigned int bWinState:1

        Whether or not the Window/Clover key is pressed.

      unsigned int bFnState:1

        Whether or not a special function key is pressed.

    }

  };

13.40.1 Detailed Description

Keyboard UI event structure definition.

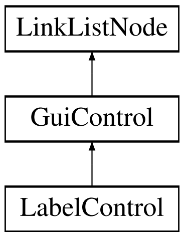
Definition at line 80 of file gui.h.

The documentation for this struct was generated from the following file:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/gui.h

13.41 LabelControl Class Reference

Inheritance diagram for LabelControl:



Public Member Functions

- virtual void Init ()

Initiailize the control - must be called before use.
- virtual void Draw ()

  Redraw the control "cleanly".
- virtual GuiReturn_t ProcessEvent (GuiEvent_t ∗pstEvent_)

  Process an event sent to the control.
- virtual void Activate (bool bActivate_)

  Activate or deactivate the current control - used when switching from one active control to another.
- void **SetBackColor** (COLOR eColor_)
- void **SetFontColor** (COLOR eColor_)
- void **SetFont** (Font_t ∗pstFont_)
- void **SetCaption** (const K_CHAR ∗pcData_)

Private Attributes

- Font_t ∗ **m_pstFont**
- const K_CHAR ∗ **m_pcCaption**
- COLOR **m_uBackColor**
- COLOR **m_uFontColor**

Additional Inherited Members

13.41.1 Detailed Description

Definition at line 30 of file control_label.h.

13.41.2 Member Function Documentation

13.41.2.1 virtual void LabelControl::Activate (bool *bActivate_*) `[inline], [virtual]`

Activate or deactivate the current control - used when switching from one active control to another.

Parameters

*bActivate_*	- true to activate, false to deactivate

Implements GuiControl.

Definition at line 40 of file control_label.h.

13.41.2.2 void LabelControl::Draw ( ) `[virtual]`

Redraw the control "cleanly".

Subclass specific.

Implements GuiControl.

Definition at line 26 of file control_label.cpp.

13.41.2.3 virtual void LabelControl::Init ( ) `[inline], [virtual]`

Initiailize the control - must be called before use.

Implementation is subclass specific.

Implements GuiControl.

Definition at line 33 of file control_label.h.

13.41.2.4 virtual GuiReturn_t LabelControl::ProcessEvent (GuiEvent_t ∗ *pstEvent_*) [inline], [virtual]

Process an event sent to the control.

Subclass specific implementation.

Parameters

*pstEvent_*	Pointer to a struct containing the event data

Implements GuiControl.

Definition at line 39 of file control_label.h.

The documentation for this class was generated from the following files:

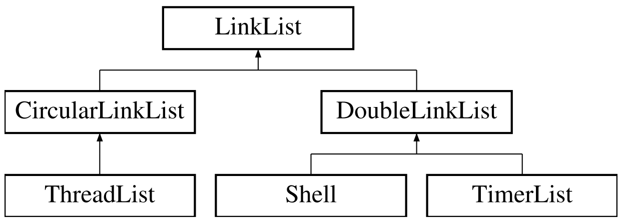
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_label.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_label.cpp

13.42 LinkList Class Reference

Abstract-data-type from which all other linked-lists are derived.

`#include <ll.h>`

Inheritance diagram for LinkList:



Public Member Functions

- void Init ()

  Clear the linked list.
- virtual void Add (LinkListNode ∗node_)=0

  Add the linked list node to this linked list.
- virtual void Remove (LinkListNode ∗node_)=0

  Add the linked list node to this linked list.
- LinkListNode ∗ GetHead ()

  Get the head node in the linked list.
- LinkListNode ∗ GetTail ()

  Get the tail node of the linked list.

Protected Attributes

- LinkListNode ∗ **m_pstHead**
- LinkListNode ∗ **m_pstTail**

13.42.1 Detailed Description

Abstract-data-type from which all other linked-lists are derived.

Definition at line 117 of file ll.h.

13.42.2 Member Function Documentation

13.42.2.1 void LinkList::Add (**LinkListNode** ∗ *node_*) [pure virtual]

Add the linked list node to this linked list.

Parameters

*node_*	Pointer to the node to add

Implemented in CircularLinkList, DoubleLinkList, and ThreadList.

13.42.2.2 **LinkListNode** ∗ LinkList::GetHead ( ) [inline]

Get the head node in the linked list.

Returns

Pointer to the head node in the list

Definition at line 154 of file ll.h.

13.42.2.3 **LinkListNode** ∗ LinkList::GetTail ( ) [inline]

Get the tail node of the linked list.

Returns

Pointer to the tail node in the list

Definition at line 163 of file ll.h.

13.42.2.4 void LinkList::Remove (**LinkListNode** ∗ *node_*) [pure virtual]

Add the linked list node to this linked list.

Parameters

*node_*	Pointer to the node to remove

Implemented in CircularLinkList, DoubleLinkList, and ThreadList.

The documentation for this class was generated from the following file:

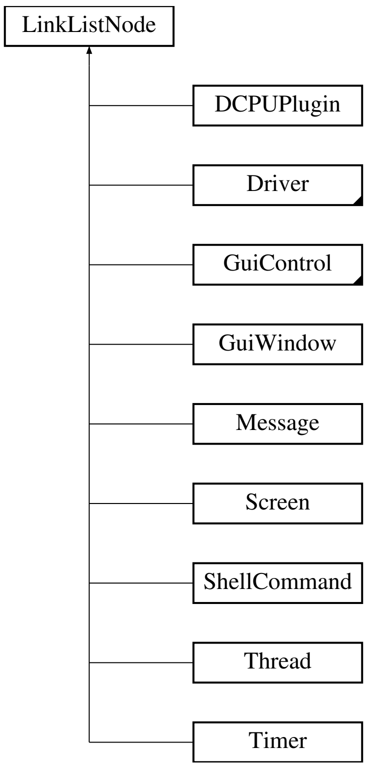
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/ll.h

13.43 LinkListNode Class Reference

Basic linked-list node data structure.

```
#include <ll.h>
```

Inheritance diagram for LinkListNode:



Public Member Functions

- LinkListNode $*$ GetNext (void)

  Returns a pointer to the next node in the list.
- LinkListNode $*$ GetPrev (void)

  Returns a pointer to the previous node in the list.

Protected Member Functions

- void ClearNode ()

  Initialize the linked list node, clearing its next and previous node.

Protected Attributes

- LinkListNode $*$ **next**
- LinkListNode $*$ **prev**

Friends

- class **LinkList**
- class **DoubleLinkList**
- class **CircularLinkList**

13.43.1 Detailed Description

Basic linked-list node data structure.

This data is managed by the linked-list class types, and can be used transparently between them.

Definition at line 75 of file ll.h.

13.43.2 Member Function Documentation

13.43.2.1 LinkListNode ∗ LinkListNode::GetNext (void) [inline]

Returns a pointer to the next node in the list.

Returns

a pointer to the next node in the list.

Definition at line 97 of file ll.h.

13.43.2.2 LinkListNode ∗ LinkListNode::GetPrev (void) [inline]

Returns a pointer to the previous node in the list.

Returns

a pointer to the previous node in the list.

Definition at line 106 of file ll.h.

The documentation for this class was generated from the following files:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/ll.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/ll.cpp

13.44 MemUtil Class Reference

String and Memory manipulation class.

`#include <memutil.h>`

Static Public Member Functions

- static void DecimalToHex (K_UCHAR ucData_, char ∗szText_)

  Convert an 8-bit unsigned binary value as a hexadecimal string.
- static void DecimalToString (K_UCHAR ucData_, char ∗szText_)

  Convert an 8-bit unsigned binary value as a decimal string.
- static K_UCHAR Checksum8 (const void ∗pvSrc_, K_USHORT usLen_)

  Compute the 8-bit addative checksum of a memory buffer.
- static K_USHORT Checksum16 (const void ∗pvSrc_, K_USHORT usLen_)

  Compute the 16-bit addative checksum of a memory buffer.
- static K_USHORT StringLength (const char ∗szStr_)

  Compute the length of a string in bytes.
- static bool CompareStrings (const char ∗szStr1_, const char ∗szStr2_)

  Compare the contents of two zero-terminated string buffers to eachother.

- static void CopyMemory (void ∗pvDst_, const void ∗pvSrc_, K_USHORT usLen_)

  Copy one buffer in memory into another.
- static void CopyString (char ∗szDst_, const char ∗szSrc_)

  Copy a string from one buffer into another.
- static K_SHORT StringSearch (const char ∗szBuffer_, const char ∗szPattern_)

  Search for the presence of one string as a substring within another.
- static bool CompareMemory (const void ∗pvMem1_, const void ∗pvMem2_, K_USHORT usLen_)

  Compare the contents of two memory buffers to eachother.
- static void SetMemory (void ∗pvDst_, K_UCHAR ucVal_, K_USHORT usLen_)

  Initialize a buffer of memory to a specified 8-bit pattern.

13.44.1 Detailed Description

String and Memory manipulation class.

Utility method class implementing common memory and string manipulation functions, without relying on an external standard library implementation which might not be available on some toolchains, may be closed source, or may not be thread-safe.

Definition at line 37 of file memutil.h.

13.44.2 Member Function Documentation

13.44.2.1 static K_USHORT MemUtil::Checksum16 (const void ∗ *pvSrc_*, K_USHORT *usLen_*) [static]

Compute the 16-bit addative checksum of a memory buffer.

Parameters

*pvSrc_*	Memory buffer to compute a 16-bit checksum of.
*usLen_*	Length of the buffer in bytes.

Returns

16-bit checksum of the memory block.

Definition at line 108 of file memutil.cpp.

13.44.2.2 static K_USHORT MemUtil::Checksum8 (const void ∗ *pvSrc_*, K_USHORT *usLen_*) [static]

Compute the 8-bit addative checksum of a memory buffer.

Parameters

*pvSrc_*	Memory buffer to compute a 8-bit checksum of.
*usLen_*	Length of the buffer in bytes.

Returns

8-bit checksum of the memory block.

Definition at line 92 of file memutil.cpp.

13.44.2.3 static bool MemUtil::CompareMemory (const void ∗ *pvMem1_*, const void ∗ *pvMem2_*, K_USHORT *usLen_*) [static]

Compare the contents of two memory buffers to eachother.

Parameters

*pvMem1_*	First buffer to compare
*pvMem2_*	Second buffer to compare
*usLen_*	Length of buffer (in bytes) to compare

Returns

true if the buffers match, false if they do not.

Definition at line 235 of file memutil.cpp.

13.44.2.4 static bool MemUtil::CompareStrings (const char ∗ *szStr1_*, const char ∗ *szStr2_*) [static]

Compare the contents of two zero-terminated string buffers to eachother.

Parameters

*szStr1_*	First string to compare
*szStr2_*	Second string to compare

Returns

true if strings match, false otherwise.

Definition at line 140 of file memutil.cpp.

13.44.2.5 static void MemUtil::CopyMemory (void ∗ *pvDst_*, const void ∗ *pvSrc_*, K_USHORT *usLen_*) [static]

Copy one buffer in memory into another.

Parameters

*pvDst_*	Pointer to the destination buffer
*pvSrc_*	Pointer to the source buffer
*usLen_*	Number of bytes to copy from source to destination

Definition at line 166 of file memutil.cpp.

13.44.2.6 static void MemUtil::CopyString (char ∗ *szDst_*, const char ∗ *szSrc_*) [static]

Copy a string from one buffer into another.

Parameters

*szDst_*	Pointer to the buffer to copy into
*szSrc_*	Pointer to the buffer to copy data from

Definition at line 183 of file memutil.cpp.

13.44.2.7 static void MemUtil::DecimalToHex (K_UCHAR *ucData_*, char ∗ *szText_*) [static]

Convert an 8-bit unsigned binary value as a hexadecimal string.

Parameters

*ucData_*	Value to convert into a string
*szText_*	Destination string buffer (3 bytes minimum)

Definition at line 28 of file memutil.cpp.

13.44.2.8 static void MemUtil::DecimalToString (K_UCHAR *ucData_*, char ∗ *szText_*) [static]

Convert an 8-bit unsigned binary value as a decimal string.

Parameters

*ucData_*	Value to convert into a string
*szText_*	Destination string buffer (4 bytes minimum)

Definition at line 61 of file memutil.cpp.

13.44.2.9 static void MemUtil::SetMemory (void ∗ *pvDst_*, K_UCHAR *ucVal_*, K_USHORT *usLen_*) [static]

Initialize a buffer of memory to a specified 8-bit pattern.

Parameters

*pvDst_*	Destination buffer to set
*ucVal_*	8-bit pattern to initialize each byte of destination with
*usLen_*	Length of the buffer (in bytes) to initialize

Definition at line 256 of file memutil.cpp.

13.44.2.10 static K_USHORT MemUtil::StringLength (const char ∗ *szStr_*) [static]

Compute the length of a string in bytes.

Parameters

*szStr_*	Pointer to the zero-terminated string to calculate the length of

Returns

length of the string (in bytes), not including the 0-terminator.

Definition at line 125 of file memutil.cpp.

13.44.2.11 static K_SHORT MemUtil::StringSearch (const char ∗ *szBuffer_*, const char ∗ *szPattern_*) [static]

Search for the presence of one string as a substring within another.

Parameters

*szBuffer_*	Buffer to search for pattern within
*szPattern_*	Pattern to search for in the buffer

Returns

Index of the first instance of the pattern in the buffer, or -1 on no match.

Definition at line 200 of file memutil.cpp.

The documentation for this class was generated from the following files:

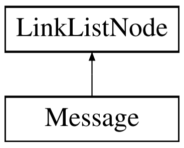
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/memutil.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/memutil.cpp

13.45 Message Class Reference

Class to provide message-based IPC services in the kernel.

`#include <message.h>`

Inheritance diagram for Message:



Public Member Functions

- void Init ()
  Initialize the data and code in the message.
- void SetData (void ∗pvData_)
  Set the data pointer for the message before transmission.
- void ∗ GetData ()
  Get the data pointer stored in the message upon receipt.
- void SetCode (K_USHORT usCode_)
  Set the code in the message before transmission.
- K_USHORT GetCode ()
  Return the code set in the message upon receipt.

Private Attributes

- void ∗ m_pvData
  Pointer to the message data.
- K_USHORT m_usCode
  Message code, providing context for the message.

Additional Inherited Members

13.45.1 Detailed Description

Class to provide message-based IPC services in the kernel.

Definition at line 99 of file message.h.

13.45.2 Member Function Documentation

13.45.2.1 K_USHORT Message::GetCode ( ) `[inline]`

Return the code set in the message upon receipt.

Returns

User code set in the object

Definition at line 143 of file message.h.

13.45.2.2 void ∗ Message::GetData ( ) `[inline]`

Get the data pointer stored in the message upon receipt.

Returns

Pointer to the data set in the message object

Definition at line 125 of file message.h.

13.45.2.3 Message::SetCode (K_USHORT *usCode_*) `[inline]`

Set the code in the message before transmission.

Parameters

*usCode_*	Data code to set in the object

Definition at line 134 of file message.h.

13.45.2.4 void Message::SetData (void ∗ *pvData_*) `[inline]`

Set the data pointer for the message before transmission.

Parameters

*pvData_*	Pointer to the data object to send in the message

Definition at line 116 of file message.h.

The documentation for this class was generated from the following file:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/message.h

13.46 MessageQueue Class Reference

List of messages, used as the channel for sending and receiving messages between threads.

```
#include <message.h>
```

Public Member Functions

- void Init ()

Initialize the message queue prior to use.
- Message ∗ Receive ()

  Receive a message from the message queue.
- Message ∗ Receive (K_ULONG ulTimeWaitMS_)

  Receive a message from the message queue.
- void Send (Message ∗pclSrc_)

  Send a message object into this message queue.
- K_USHORT GetCount ()

  Return the number of messages pending in the "receive" queue.

Private Attributes

- Semaphore m_clSemaphore

  Counting semaphore used to manage thread blocking.
- DoubleLinkList m_clLinkList

  List object used to store messages.

13.46.1 Detailed Description

List of messages, used as the channel for sending and receiving messages between threads.

Definition at line 201 of file message.h.

13.46.2 Member Function Documentation

13.46.2.1 K_USHORT MessageQueue::GetCount ( )

Return the number of messages pending in the "receive" queue.

Returns

Count of pending messages in the queue.

Definition at line 147 of file message.cpp.

13.46.2.2 Message ∗ MessageQueue::Receive ( )

Receive a message from the message queue.

If the message queue is empty, the thread will block until a message is available.

Returns

Pointer to a message object at the head of the queue

Definition at line 88 of file message.cpp.

13.46.2.3 Message ∗ MessageQueue::Receive (K_ULONG *ulWaitTimeMS_*)

Receive a message from the message queue.

If the message queue is empty, the thread will block until a message is available for the duration specified. If no message arrives within that duration, the call will return with NULL.

Parameters

*ulWaitTimeMS_*	The amount of time in ms to wait for a message before timing out and unblocking the waiting thread.

Returns

Pointer to a message object at the head of the queue or NULL on timeout.

Definition at line 108 of file message.cpp.

13.46.2.4 void MessageQueue::Send (**Message** ∗ *pclSrc_*)

Send a message object into this message queue.

Will un-block the first waiting thread blocked on this queue if that occurs.

Parameters

*pclSrc_*	Pointer to the message object to add to the queue

Definition at line 130 of file message.cpp.

The documentation for this class was generated from the following files:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/message.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/message.cpp

13.47 MouseEvent_t Struct Reference

Mouse UI event structure.

`#include <gui.h>`

Public Attributes

- K_USHORT usX

  absolute X location of the mouse (pixel)
- K_USHORT usY

  absolute Y location of the mouse (pixel)
- union {
  K_UCHAR ucFlags
  modifier flags for the event
  struct {
  unsigned int bLeftState:1
  State of the left mouse button.
  unsigned int bRightState:1
  State of the right mouse button.
  unsigned int bMiddleState:1
  State of the middle mouse button.
  unsigned int bScrollUp:1
  State of the scroll wheel (UP)
  unsigned int bScrollDown:1
  State of the scroll wheel (DOWN)
  }
  };

13.47.1 Detailed Description

Mouse UI event structure.

Definition at line 102 of file gui.h.

The documentation for this struct was generated from the following file:

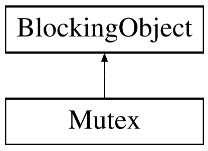
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/gui.h

13.48 Mutex Class Reference

Mutual-exclusion locks, based on BlockingObject.

`#include <mutex.h>`

Inheritance diagram for Mutex:



Public Member Functions

- void Init ()

    Initialize a mutex object for use - must call this function before using the object.
- void Claim ()

    Claim the mutex.
- bool Claim (K_ULONG ulWaitTimeMS_)
- void WakeMe (Thread ∗pclOwner_)

    Wake a thread blocked on the mutex.
- void **SetExpired** (bool bExpired_)
- void Release ()

    Release the mutex.

Private Member Functions

- K_UCHAR WakeNext ()

    Wake the next thread waiting on the Mutex.

Private Attributes

- K_UCHAR **m_bReady**
- K_UCHAR **m_ucMaxPri**
- Thread ∗ **m_pclOwner**
- bool **m_bExpired**

Additional Inherited Members

13.48.1 Detailed Description

Mutual-exclusion locks, based on BlockingObject.

Definition at line 68 of file mutex.h.

13.48.2 Member Function Documentation

13.48.2.1 void Mutex::Claim ( )

Claim the mutex.

When the mutex is claimed, no other thread can claim a region protected by the object.

Definition at line 96 of file mutex.cpp.

13.48.2.2 bool Mutex::Claim (K_ULONG *ulWaitTimeMS_*)

Parameters

*ulWaitTimeMS_*	

Returns

true - mutex was claimed within the time period specified false - mutex operation timed-out before the claim operation.

Definition at line 100 of file mutex.cpp.

13.48.2.3 void Mutex::Release ( )

Release the mutex.

When the mutex is released, another object can enter the mutex-protected region.

Definition at line 190 of file mutex.cpp.

13.48.2.4 void Mutex::WakeMe (Thread ∗ *pclOwner_*)

Wake a thread blocked on the mutex.

This is an internal function used for implementing timed mutexes relying on timer callbacks. Since these do not have access to the private data of the mutex and its base classes, we have to wrap this as a public method - do not use this for any other purposes.

Parameters

*pclOwner_*	Thread to unblock from this object.

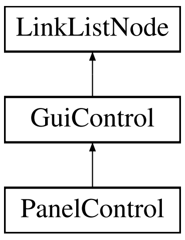
Definition at line 55 of file mutex.cpp.

The documentation for this class was generated from the following files:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/mutex.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/mutex.cpp

13.49 PanelControl Class Reference

Inheritance diagram for PanelControl:



Public Member Functions

- virtual void Init ()

    Initiailize the control - must be called before use.
- virtual void Draw ()

    Redraw the control "cleanly".
- virtual GuiReturn_t ProcessEvent (GuiEvent_t ∗pstEvent_)

    Process an event sent to the control.
- virtual void Activate (bool bActivate_)

    Activate or deactivate the current control - used when switching from one active control to another.
- void **SetColor** (COLOR eColor_)

Private Attributes

- COLOR **m_uColor**

Additional Inherited Members

13.49.1 Detailed Description

Definition at line 33 of file control_panel.h.

13.49.2 Member Function Documentation

13.49.2.1 virtual void PanelControl::Activate (bool *bActivate_*) [inline], [virtual]

Activate or deactivate the current control - used when switching from one active control to another.

Parameters

*bActivate_*	- true to activate, false to deactivate

Implements GuiControl.

Definition at line 39 of file control_panel.h.

13.49.2.2 void PanelControl::Draw ( ) [virtual]

Redraw the control "cleanly".

Subclass specific.

Implements GuiControl.

Definition at line 26 of file control_panel.cpp.

13.49.2.3 virtual void PanelControl::Init ( ) [inline], [virtual]

Initiailize the control - must be called before use.

Implementation is subclass specific.

Implements GuiControl.

Definition at line 36 of file control_panel.h.

13.49.2.4 virtual GuiReturn_t PanelControl::ProcessEvent (GuiEvent_t * *pstEvent_*) [inline], [virtual]

Process an event sent to the control.

Subclass specific implementation.

Parameters

*pstEvent_*	Pointer to a struct containing the event data

Implements GuiControl.

Definition at line 38 of file control_panel.h.

The documentation for this class was generated from the following files:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_panel.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_panel.cpp

13.50 Profiler Class Reference

System profiling timer interface.

`#include <kprofile.h>`

Static Public Member Functions

- static void Init ()

  Initialize the global system profiler.
- static void Start ()

  Start the global profiling timer service.
- static void Stop ()

  Stop the global profiling timer service.
- static K_USHORT Read ()

  Read the current tick count in the timer.
- static void Process ()

  Process the profiling counters from ISR.
- static K_ULONG GetEpoch ()

  Return the current timer epoch.

Static Private Attributes

- static K_ULONG **m_ulEpoch**

13.50.1 Detailed Description

System profiling timer interface.

Definition at line 37 of file kprofile.h.

13.50.2 Member Function Documentation

13.50.2.1 void Profiler::Init (void) [static]

Initialize the global system profiler.

Must be called prior to use.

Definition at line 32 of file kprofile.cpp.

The documentation for this class was generated from the following files:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kprofile.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kprofile.cpp

13.51 ProfileTimer Class Reference

Profiling timer.

`#include <profile.h>`

Public Member Functions

- void Init ()

  Initialize the profiling timer prior to use.
- void Start ()

  Start a profiling session, if the timer is not already active.
- void Stop ()

  Stop the current profiling session, adding to the cumulative time for this timer, and the total iteration count.
- K_ULONG GetAverage ()

  Get the average time associated with this operation.
- K_ULONG GetCurrent ()

  Return the current tick count held by the profiler.

Private Member Functions

- K_ULONG ComputeCurrentTicks (K_USHORT usCount_, K_ULONG ulEpoch_)

  Figure out how many ticks have elapsed in this iteration.

Private Attributes

- K_ULONG m_ulCumulative

  Cumulative tick-count for this timer.
- K_ULONG m_ulCurrentIteration

  Tick-count for the current iteration.
- K_USHORT m_usInitial

  Initial count.
- K_ULONG m_ulInitialEpoch

  Initial Epoch.
- K_USHORT m_usIterations

  Number of iterations executed for this profiling timer.
- K_UCHAR m_bActive

  Wheter or not the timer is active or stopped.

13.51.1 Detailed Description

Profiling timer.

This class is used to perform high-performance profiling of code to see how K_LONG certain operations take. Useful in instrumenting the performance of key algorithms and time-critical operations to ensure real-timer behavior.

Definition at line 69 of file profile.h.

13.51.2 Member Function Documentation

13.51.2.1 K_ULONG ProfileTimer::ComputeCurrentTicks (K_USHORT *usCount_*, K_ULONG *ulEpoch_*) `[private]`

Figure out how many ticks have elapsed in this iteration.

Parameters

*usCount_*	Current timer count
*ulEpoch_*	Current timer epoch

Returns

Current tick count

Definition at line 106 of file profile.cpp.

13.51.2.2 K_ULONG ProfileTimer::GetAverage ( )

Get the average time associated with this operation.

Returns

Average tick count normalized over all iterations

Definition at line 79 of file profile.cpp.

13.51.2.3 K_ULONG ProfileTimer::GetCurrent ( )

Return the current tick count held by the profiler.

Valid for both active and stopped timers.

Returns

The currently held tick count.

Definition at line 89 of file profile.cpp.

13.51.2.4 void ProfileTimer::Init (void)

Initialize the profiling timer prior to use.

Can also be used to reset a timer that's been used previously.

Definition at line 37 of file profile.cpp.

13.51.2.5 void ProfileTimer::Start (void)

Start a profiling session, if the timer is not already active.

Has no effect if the timer is already active.

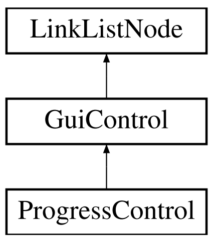
Definition at line 46 of file profile.cpp.

The documentation for this class was generated from the following files:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/profile.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/profile.cpp

13.52 ProgressControl Class Reference

Inheritance diagram for ProgressControl:



Public Member Functions

- virtual void Init ()

  Initiailize the control - must be called before use.
- virtual void Draw ()

  Redraw the control "cleanly".
- virtual GuiReturn_t ProcessEvent (GuiEvent_t ∗pstEvent_)

  Process an event sent to the control.
- virtual void Activate (bool bActivate_)

  Activate or deactivate the current control - used when switching from one active control to another.
- void **SetBackColor** (COLOR eColor_)
- void **SetProgressColor** (COLOR eColor_)
- void **SetBorderColor** (COLOR eColor_)
- void **SetProgress** (K_UCHAR ucProgress_)

Private Attributes

- COLOR **m_uBackColor**
- COLOR **m_uProgressColor**
- COLOR **m_uBorderColor**
- K_UCHAR **m_ucProgress**

Additional Inherited Members

13.52.1 Detailed Description

Definition at line 30 of file control_progress.h.

13.52.2 Member Function Documentation

13.52.2.1 virtual void ProgressControl::Activate (bool *bActivate_*) [inline], [virtual]

Activate or deactivate the current control - used when switching from one active control to another.

Parameters

*bActivate_*	- true to activate, false to deactivate

Implements GuiControl.

Definition at line 36 of file control_progress.h.

13.52.2.2 void ProgressControl::Draw ( ) [virtual]

Redraw the control "cleanly".

Subclass specific.

Implements GuiControl.

Definition at line 36 of file control_progress.cpp.

13.52.2.3 void ProgressControl::Init ( ) [virtual]

Initiailize the control - must be called before use.

Implementation is subclass specific.

Implements GuiControl.

Definition at line 27 of file control_progress.cpp.

13.52.2.4 GuiReturn_t ProgressControl::ProcessEvent (GuiEvent_t * *pstEvent_*) [virtual]

Process an event sent to the control.

Subclass specific implementation.

Parameters

*pstEvent_*	Pointer to a struct containing the event data

Implements GuiControl.

Definition at line 102 of file control_progress.cpp.

The documentation for this class was generated from the following files:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_progress.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_progress.cpp

13.53 Quantum Class Reference

Static-class used to implement Thread quantum functionality, which is a key part of round-robin scheduling.

`#include <quantum.h>`

Static Public Member Functions

- static void UpdateTimer ()

  This function is called to update the thread quantum timer whenever something in the scheduler has changed.
- static void AddThread (Thread ∗pclThread_)

  Add the thread to the quantum timer.
- static void RemoveThread ()

  Remove the thread from the quantum timer.

Static Private Member Functions

- static void SetTimer (Thread ∗pclThread_)

  Set up the quantum timer in the timer scheduler.

Static Private Attributes

- static Timer **m_clQuantumTimer**
- static K_UCHAR **m_bActive**

13.53.1 Detailed Description

Static-class used to implement Thread quantum functionality, which is a key part of round-robin scheduling.

Definition at line 39 of file quantum.h.

13.53.2 Member Function Documentation

13.53.2.1 void Quantum::AddThread (**Thread** ∗ *pclThread_*) `[static]`

Add the thread to the quantum timer.

Only one thread can own the quantum, since only one thread can be running on a core at a time.

Definition at line 70 of file quantum.cpp.

13.53.2.2 void Quantum::RemoveThread (void) `[static]`

Remove the thread from the quantum timer.

This will cancel the timer.

Definition at line 87 of file quantum.cpp.

13.53.2.3 void Quantum::SetTimer (**Thread** ∗ *pclThread_*) `[static], [private]`

Set up the quantum timer in the timer scheduler.

This creates a one-shot timer, which calls a static callback in quantum.cpp that on expiry will pivot the head of the threadlist for the thread's priority. This is the mechanism that provides round-robin scheduling in the system.

Parameters

*pclThread_*	Pointer to the thread to set the Quantum timer on

Definition at line 60 of file quantum.cpp.

13.53.2.4 void Quantum::UpdateTimer (void) `[static]`

This function is called to update the thread quantum timer whenever something in the scheduler has changed.

This can result in the timer being re-loaded or started. The timer is never stopped, but if may be ignored on expiry.

Definition at line 100 of file quantum.cpp.

The documentation for this class was generated from the following files:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/quantum.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/quantum.cpp

13.54 Scheduler Class Reference

Priority-based round-robin Thread scheduling, using ThreadLists for housekeeping.

`#include <scheduler.h>`

Static Public Member Functions

- static void Init ()

  Intiailize the scheduler, must be called before use.
- static void Schedule ()

  Run the scheduler, determines the next thread to run based on the current state of the threads.
- static void Add (Thread ∗pclThread_)

  Add a thread to the scheduler at its current priority level.
- static void Remove (Thread ∗pclThread_)

  Remove a thread from the scheduler at its current priority level.
- static void SetScheduler (K_UCHAR bEnable_)

  Set the active state of the scheduler.
- static Thread ∗ GetCurrentThread ()

  Return the pointer to the currently-running thread.
- static Thread ∗ GetNextThread ()

  Return the pointer to the thread that should run next, according to the last run of the scheduler.
- static ThreadList ∗ GetThreadList (K_UCHAR ucPriority_)

  Return the pointer to the active list of threads that are at the given priority level in the scheduler.
- static ThreadList ∗ GetStopList ()

  Return the pointer to the list of threads that are in the scheduler's stopped state.
- static K_UCHAR IsEnabled ()

  Return the current state of the scheduler - whether or not scheudling is enabled or disabled.

Static Private Attributes

- static K_UCHAR m_bEnabled

  Scheduler's state - enabled or disabled.
- static ThreadList m_clStopList

  ThreadList for all stopped threads.
- static ThreadList m_aclPriorities [NUM_PRIORITIES]

  ThreadLists for all threads at all priorities.
- static K_UCHAR m_ucPriFlag

  Bitmap flag for each.

13.54.1 Detailed Description

Priority-based round-robin Thread scheduling, using ThreadLists for housekeeping.

Definition at line 61 of file scheduler.h.

13.54.2 Member Function Documentation

13.54.2.1 void Scheduler::Add (**Thread** ∗ *pclThread_*) `[static]`

Add a thread to the scheduler at its current priority level.

Parameters

*pclThread_*	Pointer to the thread to add to the scheduler

Definition at line 77 of file scheduler.cpp.

13.54.2.2 static **Thread**∗ Scheduler::GetCurrentThread ( ) `[inline]`, `[static]`

Return the pointer to the currently-running thread.

Returns

Pointer to the currently-running thread

Definition at line 118 of file scheduler.h.

13.54.2.3 static **Thread**∗ Scheduler::GetNextThread ( ) `[inline]`, `[static]`

Return the pointer to the thread that should run next, according to the last run of the scheduler.

Returns

Pointer to the next-running thread

Definition at line 126 of file scheduler.h.

13.54.2.4 static **ThreadList**∗ Scheduler::GetStopList ( ) `[inline]`, `[static]`

Return the pointer to the list of threads that are in the scheduler's stopped state.

Returns

Pointer to the ThreadList containing the stopped threads

Definition at line 144 of file scheduler.h.

13.54.2.5 static ThreadList∗ Scheduler::GetThreadList (K_UCHAR *ucPriority_*) [inline], [static]

Return the pointer to the active list of threads that are at the given priority level in the scheduler.

Parameters

*ucPriority_*	Priority level of

Returns

Pointer to the ThreadList for the given priority level

Definition at line 136 of file scheduler.h.

13.54.2.6 K_UCHAR Scheduler::IsEnabled ( ) [inline], [static]

Return the current state of the scheduler - whether or not scheudling is enabled or disabled.

Returns

true - scheduler enabled, false - disabled

Definition at line 154 of file scheduler.h.

13.54.2.7 void Scheduler::Remove (Thread ∗ *pclThread_*) [static]

Remove a thread from the scheduler at its current priority level.

Parameters

*pclThread_*	Pointer to the thread to be removed from the scheduler

Definition at line 84 of file scheduler.cpp.

13.54.2.8 Scheduler::Schedule ( ) [static]

Run the scheduler, determines the next thread to run based on the current state of the threads.

Note that the next-thread chosen from this function is only valid while in a critical section.

Definition at line 60 of file scheduler.cpp.

13.54.2.9 void Scheduler::SetScheduler (K_UCHAR *bEnable_*) [inline], [static]

Set the active state of the scheduler.

When the scheduler is disabled, the *next thread* is never set; the currently running thread will run forever until the scheduler is enabled again. Care must be taken to ensure that we don't end up trying to block while the scheduler is disabled, otherwise the system ends up in an unusable state.

Parameters

*bEnable_*	true to enable, false to disable the scheduler

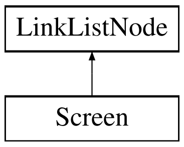
Definition at line 111 of file scheduler.h.

The documentation for this class was generated from the following files:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/scheduler.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/scheduler.cpp

13.55 Screen Class Reference

Inheritance diagram for Screen:



Public Member Functions

- void Activate ()

  This is called when a new screen needs to be created.
- void Deactivate ()

  This is called when a screen is torn-down.
- void SetWindowAffinity (const K_CHAR ∗szWindowName_)

  Indicate by name which window this screen is to be bound.
- void SetName (const K_CHAR ∗szName_)

  Set the name of the current screen.
- const K_CHAR ∗ GetName ()

  Return the name of the current screen.

Protected Member Functions

- void SetManager (ScreenManager ∗pclScreenManager_)

  Function called by the ScreenManager to set the screen affinity.

Protected Attributes

- const K_CHAR ∗ **m_szName**
- ScreenManager ∗ **m_pclScreenManager**
- GuiWindow ∗ **m_pclWindow**

Private Member Functions

- virtual void **Create** ()=0
- virtual void **Destroy** ()=0

Friends

- class **ScreenManager**

13.55.1 Detailed Description

Definition at line 31 of file screen.h.

13.55.2 Member Function Documentation

13.55.2.1 void Screen::Activate ( ) `[inline]`

This is called when a new screen needs to be created.

This calls the underlying virtual "create" method, which performs all control object initialization and allocation. Calling a redraw(true) on the bound window will result in the new window being rendered to display.

Definition at line 40 of file screen.h.

13.55.2.2 void Screen::Deactivate ( ) `[inline]`

This is called when a screen is torn-down.

Essentially removes the controls from the named window and deallocates any memory used to build up the screen.

Definition at line 47 of file screen.h.

The documentation for this class was generated from the following files:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/screen.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/screen.cpp

13.56 ScreenList Class Reference

Public Member Functions

- void Add (Screen ∗pclScreen_)

  Add a screen to the screen list.
- void Remove (Screen ∗pclScreen_)

  Remove a screen from the screen list.
- Screen ∗ GetHead ()

  Get the beginning of the screen list.

Private Attributes

- DoubleLinkList m_clList

  Double link-list used to manage screen objects.

13.56.1 Detailed Description

Definition at line 84 of file screen.h.

The documentation for this class was generated from the following file:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/screen.h

13.57 ScreenManager Class Reference

Public Member Functions

- void AddScreen (Screen ∗pclScreen_)

  Add a new screen to the screen manager.
- void RemoveScreen (Screen ∗pclScreen_)

  Remove an existing screen from the screen manager.
- void SetEventSurface (GuiEventSurface ∗pclSurface_)

  Set the event surface on which this screen manager's screens will be displayed.
- GuiWindow ∗ FindWindowByName (const K_CHAR ∗m_szName_)

  Return a pointer to a window by name.
- Screen ∗ FindScreenByName (const K_CHAR ∗m_szName_)

  Return a pointer to a screen by name.

Private Attributes

- ScreenList m_clScreenList

  Screen list object used to manage individual screens.
- GuiEventSurface ∗ m_pclSurface

  Pointer to the GUI Event Surface on which the screens are displayed.

13.57.1 Detailed Description

Definition at line 109 of file screen.h.

The documentation for this class was generated from the following files:

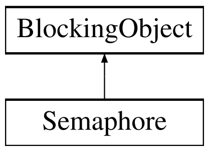
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/screen.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/screen.cpp

13.58 Semaphore Class Reference

Counting semaphore, based on BlockingObject base class.

`#include <semaphore.h>`

Inheritance diagram for Semaphore:



Public Member Functions

- void Init (K_USHORT usInitVal_, K_USHORT usMaxVal_)

  Initialize a semaphore before use.
- void Post ()

  Increment the semaphore count.
- void Pend ()

Decrement the semaphore count.
- K_USHORT GetCount ()

  Return the current semaphore counter.
- bool Pend (K_ULONG ulWaitTimeMS_)

  Decrement the semaphore count.
- void WakeMe (Thread ∗pclChosenOne_)

  Wake a thread blocked on the semaphore.
- void SetExpired (bool bExpired_)

  Set the semaphore expired flag on this object.
- bool **GetExpired** ()

Private Member Functions

- K_UCHAR WakeNext ()

  Wake the next thread waiting on the semaphore.

Private Attributes

- K_USHORT **m_usValue**
- K_USHORT **m_usMaxValue**
- bool **m_bExpired**

Additional Inherited Members

13.58.1 Detailed Description

Counting semaphore, based on BlockingObject base class.

Definition at line 37 of file semaphore.h.

13.58.2 Member Function Documentation

13.58.2.1 K_USHORT Semaphore::GetCount ( )

Return the current semaphore counter.

This can be used by a thread to bypass blocking on a semaphore - allowing it to do other things until a non-zero count is returned, instead of blocking until the semaphore is posted.

Returns

The current semaphore counter value.

Definition at line 213 of file semaphore.cpp.

13.58.2.2 void Semaphore::Init (K_USHORT *usInitVal_*, K_USHORT *usMaxVal_*)

Initialize a semaphore before use.

Must be called before post/pend operations.

Parameters

*usInitVal_*	Initial value held by the semaphore
*usMaxVal_*	Maximum value for the semaphore

Definition at line 84 of file semaphore.cpp.

13.58.2.3 void Semaphore::Pend ( )

Decrement the semaphore count.

If the count is zero, the thread will block until the semaphore is pended.

Definition at line 143 of file semaphore.cpp.

13.58.2.4 bool Semaphore::Pend (K_ULONG *ulWaitTimeMS_*)

Decrement the semaphore count.

If the count is zero, the thread will block until the semaphore is pended. If the specified interval expires before the thread is unblocked, then the status is returned back to the user.

Returns

true - semaphore was acquired before the timeout false - timeout occurred before the semaphore was claimed.

Definition at line 148 of file semaphore.cpp.

13.58.2.5 void Semaphore::SetExpired (bool *bExpired_*) `[inline]`

Set the semaphore expired flag on this object.

\

Definition at line 112 of file semaphore.h.

13.58.2.6 void Semaphore::WakeMe (Thread ∗ *pclChosenOne_*)

Wake a thread blocked on the semaphore.

This is an internal function used for implementing timed semaphores relying on timer callbacks. Since these do not have access to the private data of the semaphore and its base classes, we have to wrap this as a public method - do not use this for any other purposes.

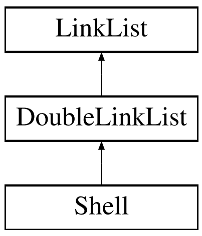
Definition at line 57 of file semaphore.cpp.

The documentation for this class was generated from the following files:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/semaphore.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/semaphore.cpp

13.59 Shell Class Reference

Inheritance diagram for Shell:



Public Member Functions

- void AddCommand (ShellCommand ∗pclCommand_)
  Add a new command to the list of commands supported in the shell.
- void RunCommand (const K_CHAR ∗pcCommand_)
  Parse the command string and run the associated command.

Private Attributes

- int **m_argc**
- void ∗ **m_argv** [MAX_ARGV]

Additional Inherited Members

13.59.1 Detailed Description

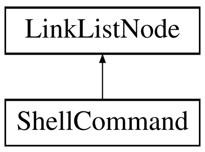
Definition at line 32 of file shell_command.h.

The documentation for this class was generated from the following file:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/shell_command.h

13.60 ShellCommand Class Reference

Inheritance diagram for ShellCommand:



Public Member Functions

- **ShellCommand** (const K_CHAR ∗pcCommand_, ShellHandler ∗pfHandler_)
- void **SetCommand** (const K_CHAR ∗pcCommand_)
- void **SetHandler** (ShellHandler ∗pfHandler_)
- const K_CHAR ∗ **GetCommand** ()
- int **RunHandler** (int argc_, void ∗argv_)

Private Attributes

- const K_CHAR ∗ **m_pcCommand**
- ShellHandler **m_pfHandler**

Additional Inherited Members

13.60.1 Detailed Description

Definition at line 12 of file shell_command.h.

The documentation for this class was generated from the following file:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/shell_command.h

13.61 Slip Class Reference

Object used to frame communications over an abstract device using the serial-line internet protocol (SLIP).

```
#include <slip.h>
```

Public Member Functions

- void SetDriver (Driver ∗pclDriver_)

  Set the driver to attach to this object.
- Driver ∗ GetDriver ()

  Return the pointer to the driver attached to this object.
- void WriteData (K_UCHAR ucChannel_, K_CHAR ∗aucBuf_, K_USHORT usLen_)

  Write a packet of data in the FunkenSlip format.
- K_USHORT ReadData (K_UCHAR ∗pucChannel_, K_CHAR ∗aucBuf_, K_USHORT usLen_)

  Read a packet from a specified device, parse, and copy to a specified output buffer.
- void WriteVector (K_UCHAR ucChannel_, SlipDataVector ∗astData_, K_USHORT usLen_)

  Write a single message composed of multiple data-vector fragments.
- void SendAck ()

  Send an acknowledgement character to the host.
- void SendNack ()

  Send a negative-acknowledgement character to the host.

Static Public Member Functions

- static K_USHORT EncodeByte (K_UCHAR ucChar_, K_UCHAR ∗aucBuf_)

  Encode a single byte into a stream, returning the size of the encoded value (either 1 or 2 bytes).
- static K_USHORT DecodeByte (K_UCHAR ∗ucChar_, K_UCHAR ∗aucBuf_)

  Decode a byte from a stream into a specified value.

Private Member Functions

- void **WriteByte** (K_UCHAR ucData_)

Private Attributes

- Driver ∗ **m_pclDriver**

13.61.1 Detailed Description

Object used to frame communications over an abstract device using the serial-line internet protocol (SLIP).

Definition at line 66 of file slip.h.

13.61.2 Member Function Documentation

13.61.2.1 K_USHORT Slip::DecodeByte (K_UCHAR * *ucChar_*, K_UCHAR * *aucBuf_*) [static]

Decode a byte from a stream into a specified value.

Returns the number of bytes from the source array that were processed, (1 or 2), or 0 if an end-of-packet (192) was encountered.

Parameters

*ucChar_*	Destination K_CHAR
*aucBuf_*	Source buffer

Returns

# bytes read, or 0 on terminating character (192)

Definition at line 56 of file slip.cpp.

13.61.2.2 K_USHORT Slip::EncodeByte (K_UCHAR *ucChar_*, K_UCHAR * *aucBuf_*) [static]

Encode a single byte into a stream, returning the size of the encoded value (either 1 or 2 bytes).

Parameters

*ucChar_*	Character to encode
*aucBuf_*	Buffer to encode into

Returns

# bytes read

Definition at line 34 of file slip.cpp.

13.61.2.3 Driver* Slip::GetDriver ( ) [inline]

Return the pointer to the driver attached to this object.

Returns

Pointer to the driver attached

Definition at line 81 of file slip.h.

13.61.2.4 K_USHORT Slip::ReadData (K_UCHAR * *pucChannel_*, K_CHAR * *aucBuf_*, K_USHORT *usLen_*)

Read a packet from a specified device, parse, and copy to a specified output buffer.

Parameters

*pucChannel_*	Pointer to a uchar that stores the message channel
*aucBuf_*	Buffer where the message will be decoded
*usLen_*	Length of the buffer to decode

Returns

data bytes read, 0 on failure.

Definition at line 104 of file slip.cpp.

13.61.2.5 void Slip::SetDriver (**Driver** ∗ *pclDriver_*) `[inline]`

Set the driver to attach to this object.

Parameters

*pclDriver_*	Pointer to the driver to attach

Definition at line 74 of file slip.h.

13.61.2.6 void Slip::WriteData (K_UCHAR *ucChannel_*, K_CHAR ∗ *aucBuf_*, K_USHORT *usLen_*)

Write a packet of data in the FunkenSlip format.

Returns the number of bytes from the source array that were processed, (1 or 2), or 0 if an end-of-packet (192) was encountered.

Parameters

*ucChannel_*	Channel to encode the packet to
*aucBuf_*	Payload to encode
*usLen_*	Length of payload data

Definition at line 164 of file slip.cpp.

13.61.2.7 void Slip::WriteVector (K_UCHAR *ucChannel_*, **SlipDataVector** ∗ *astData_*, K_USHORT *usLen_*)

Write a single message composed of multiple data-vector fragments.

Allows for transmitting complex data structures without requiring buffering. This operation is zero-copy.

Parameters

*ucChannel_*	Message channel
*astData_*	Pointer to the data vector
*usLen_*	Number of elements in the data vector

Definition at line 223 of file slip.cpp.

The documentation for this class was generated from the following files:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slip.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slip.cpp

13.62 SlipDataVector Struct Reference

Data structure used for vector-based SLIP data transmission.

`#include <slip.h>`

Public Attributes

- K_UCHAR ucSize

  Size of the data buffer.
- K_UCHAR ∗ pucData

  Pointer to the data buffer.

13.62.1 Detailed Description

Data structure used for vector-based SLIP data transmission.

Allows for building and transmitting complex data structures without having to copy data into intermediate buffers.

Definition at line 55 of file slip.h.

The documentation for this struct was generated from the following file:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slip.h

13.63 SlipMux Class Reference

Static-class which implements a multiplexed stream of SLIP data over a single interface.

```
#include <slip_mux.h>
```

Static Public Member Functions

- static void Init (K_CHAR ∗pcDriverPath_, K_USHORT usRxSize_, K_UCHAR ∗aucRx_, K_USHORT usTx-Size_, K_UCHAR ∗aucTx_)

  Attach a driver to the Slip-stream multiplexer and initialize the internal data associated with the module.
- static void InstallHandler (K_UCHAR ucChannel_, Slip_Channel pfHandler_)

  Install a slip handler function for the given communication channel.
- static void MessageReceive ()

  Wait for a valid packet to arrive, and call the appropriate handler function for the channel the message was attached to.
- static Driver ∗ GetDriver ()

  Return the pointer of the current driver used by the SlipMux module.
- static MessageQueue ∗ GetQueue ()

  Return the pointer to the message queue attached to the slip mux channel.
- static void SetQueue (MessageQueue ∗pclMessageQueue_)

  Set the message queue that will receive the notification when the slip mux channel has received data.
- static Slip ∗ GetSlip ()

  Return the pointer to the SlipMux' Slip object.

Static Private Attributes

- static MessageQueue ∗ **m_pclMessageQueue**
- static Driver ∗ **m_pclDriver**
- static Slip_Channel **m_apfChannelHandlers** [SLIP_CHANNEL_COUNT] = {0}
- static K_UCHAR **m_aucData** [SLIP_BUFFER_SIZE]
- static Semaphore **m_clSlipSem**
- static Slip **m_clSlip**

13.63.1 Detailed Description

Static-class which implements a multiplexed stream of SLIP data over a single interface.

Definition at line 43 of file slip_mux.h.

13.63.2 Member Function Documentation

13.63.2.1 static **Driver**∗ SlipMux::GetDriver ( ) `[inline], [static]`

Return the pointer of the current driver used by the SlipMux module.

Returns

Pointer to the current handle owned by SlipMux

Definition at line 91 of file slip_mux.h.

13.63.2.2 static **MessageQueue**∗ SlipMux::GetQueue ( ) `[inline], [static]`

Return the pointer to the message queue attached to the slip mux channel.

Returns

Pointer to the message Queue

Definition at line 99 of file slip_mux.h.

13.63.2.3 static **Slip**∗ SlipMux::GetSlip ( ) `[inline], [static]`

Return the pointer to the SlipMux' Slip object.

Returns

Pointer to the Slip object

Definition at line 117 of file slip_mux.h.

13.63.2.4 void SlipMux::Init (K_CHAR ∗ *pcDriverPath_*, K_USHORT *usRxSize_*, K_UCHAR ∗ *aucRx_*, K_USHORT *usTxSize_*, K_UCHAR ∗ *aucTx_*) `[static]`

Attach a driver to the Slip-stream multiplexer and initialize the internal data associated with the module.

Must be called before any of the other functions in this module are called.

Parameters

*pcDriverPath_*	Filesystem path to the driver to attach to
*usRxSize_*	Size of the RX Buffer to attach to the driver
*aucRx_*	Pointer to the RX Buffer to attach to the driver
*usTxSize_*	Size of the TX Buffer to attach to the driver
*aucTx_*	Pointer to the TX Buffer to attach to the driver

Definition at line 59 of file slip_mux.cpp.

13.63.2.5 void SlipMux::InstallHandler (K_UCHAR *ucChannel_*, Slip_Channel *pfHandler_*) [static]

Install a slip handler function for the given communication channel.

Parameters

*ucChannel_*	Channel to attach the handler to
*pfHanler_*	Pointer to the handler function to attach

Definition at line 76 of file slip_mux.cpp.

13.63.2.6 void SlipMux::MessageReceive (void) [static]

Wait for a valid packet to arrive, and call the appropriate handler function for the channel the message was attached to.

This is essentially the entry point for a thread whose purpose is to service slip Rx data.

Definition at line 85 of file slip_mux.cpp.

13.63.2.7 static void SlipMux::SetQueue (MessageQueue ∗ *pclMessageQueue_*) [inline], [static]

Set the message queue that will receive the notification when the slip mux channel has received data.

Parameters

*pclMessage-Queue_*	Pointer to the message queue to use for notification.

Definition at line 108 of file slip_mux.h.

The documentation for this class was generated from the following files:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slip_mux.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slip_mux.cpp

13.64 SlipTerm Class Reference

Class implementing a simple debug terminal interface.

`#include <slipterm.h>`

Public Member Functions

- void Init ()

  Initialize the terminal by opening a handle to the serial interface attached at /dev/tty.
- void PrintLn (const char ∗szLine_)

  Print a string of text to the SLIP interface, multiplexed using the FunkenSlip terminal channel.
- void PrintLn (K_UCHAR ucSeverity_, const char ∗szLine_)

  Print a string of text to the SLIP interface, but only if the current logging verbosity level is greater than or equal to the specified message severity.
- void SetVerbosity (K_UCHAR ucLevel_)

  Set the logging verbosity level - the minimum severity level that will be printed to the terminal.

Private Member Functions

- K_USHORT StrLen (const char ∗szString_)

    Quick 'n' dirty StrLen functionality used for printing the string.

Private Attributes

- K_UCHAR m_ucVerbosity
- Slip m_clSlip

    Slip object that this module interfaces with.

13.64.1 Detailed Description

Class implementing a simple debug terminal interface.

This is useful for printf style debugging.

Definition at line 40 of file slipterm.h.

13.64.2 Member Function Documentation

13.64.2.1 void SlipTerm::Init (void)

Initialize the terminal by opening a handle to the serial interface attached at /dev/tty.

Must be called prior to using the print functionality.

Definition at line 26 of file slipterm.cpp.

13.64.2.2 void SlipTerm::PrintLn (const char ∗ *szLine_*)

Print a string of text to the SLIP interface, multiplexed using the FunkenSlip terminal channel.

Parameters

*szLine_*	String to print

Definition at line 44 of file slipterm.cpp.

13.64.2.3 void SlipTerm::PrintLn (K_UCHAR *ucSeverity_*, const char ∗ *szLine_*)

Print a string of text to the SLIP interface, but only if the current logging verbosity level is greater than or equal to the specified message severity.

Parameters

*ucSeverity_*	Message severity level, 0 = highest severity
*szLine_*	String to print

Definition at line 56 of file slipterm.cpp.

13.64.2.4 void SlipTerm::SetVerbosity (K_UCHAR *ucLevel_*) `[inline]`

Set the logging verbosity level - the minimum severity level that will be printed to the terminal.

The higher the number, the more chatty the output.

Definition at line 81 of file slipterm.h.

13.64.2.5 K_USHORT SlipTerm::StrLen (const char ∗ *szString_*) [private]

Quick 'n' dirty StrLen functionality used for printing the string.

Returns

Length of the string (in bytes)

Definition at line 33 of file slipterm.cpp.

13.64.3 Member Data Documentation

13.64.3.1 K_UCHAR SlipTerm::m_ucVerbosity [private]

Verbosity level. Messages with a severity level greater than this Are not displayed.

Definition at line 92 of file slipterm.h.

The documentation for this class was generated from the following files:

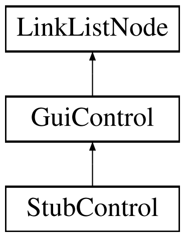
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slipterm.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slipterm.cpp

13.65 StubControl Class Reference

Stub control class, used for testing out the GUI framework where certain controls have not yet been implemented.

```
#include <gui.h>
```

Inheritance diagram for StubControl:



Public Member Functions

- virtual void Init ()

  Initiailize the control - must be called before use.
- virtual void Draw ()

  Redraw the control "cleanly".
- virtual GuiReturn_t ProcessEvent (GuiEvent_t ∗pstEvent_)

  Process an event sent to the control.
- virtual void Activate (bool bActivate_)

  Activate or deactivate the current control - used when switching from one active control to another.

Additional Inherited Members

13.65.1 Detailed Description

Stub control class, used for testing out the GUI framework where certain controls have not yet been implemented.

Definition at line 779 of file gui.h.

13.65.2 Member Function Documentation

13.65.2.1 virtual void StubControl::Activate (bool *bActivate_*) [inline], [virtual]

Activate or deactivate the current control - used when switching from one active control to another.

Parameters

*bActivate_*	- true to activate, false to deactivate

Implements GuiControl.

Definition at line 785 of file gui.h.

13.65.2.2 virtual void StubControl::Draw ( ) [inline], [virtual]

Redraw the control "cleanly".

Subclass specific.

Implements GuiControl.

Definition at line 783 of file gui.h.

13.65.2.3 virtual void StubControl::Init ( ) [inline], [virtual]

Initiailize the control - must be called before use.

Implementation is subclass specific.

Implements GuiControl.

Definition at line 782 of file gui.h.

13.65.2.4 virtual GuiReturn_t StubControl::ProcessEvent (GuiEvent_t * *pstEvent_*) [inline], [virtual]

Process an event sent to the control.

Subclass specific implementation.

Parameters

*pstEvent_*	Pointer to a struct containing the event data

Implements GuiControl.

Definition at line 784 of file gui.h.

The documentation for this class was generated from the following file:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/gui.h

13.66 SystemHeap Class Reference

Static Public Member Functions

- static void **Init** (void)
- static void ∗ **Alloc** (K_USHORT usSize_)
- static void **Free** (void ∗pvData_)

Static Private Attributes

- static K_UCHAR **m_pucRawHeap** [HEAP_RAW_SIZE]
- static HeapConfig **m_pclSystemHeapConfig** [HEAP_NUM_SIZES+1]
- static FixedHeap **m_clSystemHeap**
- static bool **m_bInit**

13.66.1 Detailed Description

Definition at line 185 of file system_heap.h.

The documentation for this class was generated from the following files:

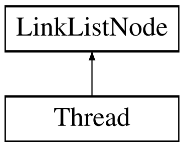
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/system_heap.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/system_heap.cpp

13.67 Thread Class Reference

Object providing fundamental multitasking support in the kernel.

`#include <thread.h>`

Inheritance diagram for Thread:



Public Member Functions

- void Init (K_UCHAR ∗paucStack_, K_USHORT usStackSize_, K_UCHAR ucPriority_, ThreadEntry_t pfEntryPoint_, void ∗pvArg_)

  Initialize a thread prior to its use.
- void Start ()

  Start the thread - remove it from the stopped list, add it to the scheduler's list of threads (at the thread's set priority), and continue along.
- void SetName (const K_CHAR ∗szName_)

  Set the name of the thread - this is purely optional, but can be useful when identifying issues that come along when multiple threads are at play in a system.
- ThreadList ∗ GetOwner (void)

  Return the ThreadList where the thread belongs when it's in the active/ready state in the scheduler.
- ThreadList ∗ GetCurrent (void)

Return the ThreadList where the thread is currently located.

- K_UCHAR GetPriority (void)

  Return the priority of the current thread.
- K_UCHAR **GetCurPriority** (void)
- void SetQuantum (K_USHORT usQuantum_)

  Set the thread's round-robin execution quantum.
- K_USHORT GetQuantum (void)

  Get the thread's round-robin execution quantum.
- void SetCurrent (ThreadList ∗pclNewList_)

  Set the thread's current to the specified thread list.
- void SetOwner (ThreadList ∗pclNewList_)

  Set the thread's owner to the specified thread list.
- void SetPriority (K_UCHAR ucPriority_)

  Set the priority of the Thread (running or otherwise) to a different level.
- void InheritPriority (K_UCHAR ucPriority_)

  Allow the thread to run at a different priority level (temporarily) for the purpose of avoiding priority inversions.
- void Exit ()

  Remove the thread from being scheduled again.
- void SetID (K_UCHAR ucID_)

  Set an 8-bit ID to uniquely identify this thread.
- K_UCHAR GetID ()

  Return the 8-bit ID corresponding to this thread.
- K_USHORT GetStackSlack ()

  Performs a (somewhat lengthy) check on the thread stack to check the amount of stack margin (or "slack") remaining on the stack.

Static Public Member Functions

- static void Sleep (K_ULONG ulTimeMs_)

  Put the thread to sleep for the specified time (in milliseconds).
- static void Yield (void)

  Yield the thread - this forces the system to call the scheduler and determine what thread should run next.

Private Member Functions

- void **SetPriorityBase** (K_UCHAR ucPriority_)

Static Private Member Functions

- static void ContextSwitchSWI (void)

  This code is used to trigger the context switch interrupt.

Private Attributes

- K_UCHAR ∗ m_paucStackTop

  Pointer to the top of the thread's stack.
- K_UCHAR ∗ m_paucStack

  Pointer to the thread's stack.
- K_USHORT m_usStackSize

  Size of the stack (in bytes)

- K_USHORT m_usQuantum

  Thread quantum (in milliseconds)
- K_UCHAR m_ucThreadID

  Thread ID.
- K_UCHAR m_ucPriority

  Default priority of the thread.
- K_UCHAR m_ucCurPriority

  Current priority of the thread (priority inheritence)
- ThreadEntry_t m_pfEntryPoint

  The entry-point function called when the thread starts.
- void ∗ m_pvArg

  Pointer to the argument passed into the thread's entrypoint.
- const K_CHAR ∗ m_szName

  Thread name.
- ThreadList ∗ m_pclCurrent

  Pointer to the thread-list where the thread currently resides.
- ThreadList ∗ m_pclOwner

  Pointer to the thread-list where the thread resides when active.

Friends

- class **ThreadPort**

Additional Inherited Members

13.67.1 Detailed Description

Object providing fundamental multitasking support in the kernel.

Definition at line 64 of file thread.h.

13.67.2 Member Function Documentation

13.67.2.1 void Thread::ContextSwitchSWI (void) [static], [private]

This code is used to trigger the context switch interrupt.

Called whenever the kernel decides that it is necessary to swap out the current thread for the "next" thread.

Definition at line 277 of file thread.cpp.

13.67.2.2 void Thread::Exit ( )

Remove the thread from being scheduled again.

The thread is effectively destroyed when this occurs. This is extremely useful for cases where a thread encounters an unrecoverable error and needs to be restarted, or in the context of systems where threads need to be created and destroyed dynamically.

This must not be called on the idle thread.

Definition at line 116 of file thread.cpp.

13.67.2.3 ThreadList ∗ Thread::GetCurrent (void) `[inline]`

Return the ThreadList where the thread is currently located.

Returns

Pointer to the thread's current list

Definition at line 131 of file thread.h.

13.67.2.4 K_UCHAR Thread::GetID ( ) `[inline]`

Return the 8-bit ID corresponding to this thread.

Returns

Thread's 8-bit ID, set by the user

Definition at line 268 of file thread.h.

13.67.2.5 ThreadList ∗ Thread::GetOwner (void) `[inline]`

Return the ThreadList where the thread belongs when it's in the active/ready state in the scheduler.

Returns

Pointer to the Thread's owner list

Definition at line 122 of file thread.h.

13.67.2.6 K_UCHAR Thread::GetPriority (void) `[inline]`

Return the priority of the current thread.

Returns

Priority of the current thread

Definition at line 141 of file thread.h.

13.67.2.7 K_USHORT Thread::GetQuantum (void) `[inline]`

Get the thread's round-robin execution quantum.

Returns

The thread's quantum

Definition at line 169 of file thread.h.

13.67.2.8 K_USHORT Thread::GetStackSlack ( )

Performs a (somewhat lengthy) check on the thread stack to check the amount of stack margin (or "slack") remaining on the stack.

If you're having problems with blowing your stack, you can run this function at points in your code during development to see what operations cause problems. Also useful during development as a tool to optimally size thread stacks.

Returns

The amount of slack (unused bytes) on the stack

! ToDo: Take into account stacks that grow up

Definition at line 178 of file thread.cpp.

13.67.2.9 void Thread::InheritPriority (K_UCHAR *ucPriority_*)

Allow the thread to run at a different priority level (temporarily) for the purpose of avoiding priority inversions.

This should only be called from within the implementation of blocking-objects.

Parameters

*ucPriority_*	New Priority to boost to.

Definition at line 270 of file thread.cpp.

13.67.2.10 void Thread::Init (K_UCHAR ∗ *paucStack_*, K_USHORT *usStackSize_*, K_UCHAR *ucPriority_*, ThreadEntry_t *pfEntryPoint_*, void ∗ *pvArg_*)

Initialize a thread prior to its use.

Initialized threads are placed in the stopped state, and are not scheduled until the thread's start method has been invoked first.

Parameters

*paucStack_*	Pointer to the stack to use for the thread
*ucStackSize_*	Size of the stack (in bytes)
*ucPriority_*	Priority of the thread (0 = idle, 7 = max)
*pfEntryPoint_*	This is the function that gets called when the thread is started
*pvArg_*	Pointer to the argument passed into the thread's entrypoint function.

< Default round-robin thread quantum of 4ms

Definition at line 39 of file thread.cpp.

13.67.2.11 void Thread::SetCurrent (ThreadList ∗ *pclNewList_*) [inline]

Set the thread's current to the specified thread list.

Parameters

*pclNewList_*	Pointer to the threadlist to apply thread ownership

Definition at line 179 of file thread.h.

13.67.2.12 void Thread::SetID (K_UCHAR *ucID_*) [inline]

Set an 8-bit ID to uniquely identify this thread.

Parameters

*ucID_*	8-bit Thread ID, set by the user

Definition at line 259 of file thread.h.

13.67.2.13 void Thread::SetName (const K_CHAR ∗ *szName_*) [inline]

Set the name of the thread - this is purely optional, but can be useful when identifying issues that come along when multiple threads are at play in a system.

Parameters

*szName_*	Char string containing the thread name

Definition at line 111 of file thread.h.

13.67.2.14 void Thread::SetOwner (ThreadList ∗ *pclNewList_*) [inline]

Set the thread's owner to the specified thread list.

Parameters

*pclNewList_*	Pointer to the threadlist to apply thread ownership

Definition at line 188 of file thread.h.

13.67.2.15 void Thread::SetPriority (K_UCHAR *ucPriority_*)

Set the priority of the Thread (running or otherwise) to a different level.

This activity involves re-scheduling, and must be done so with due caution, as it may effect the determinism of the system.

This should *always* be called from within a critical section to prevent system issues.

Parameters

*ucPriority_*	New priority of the thread

Definition at line 233 of file thread.cpp.

13.67.2.16 void Thread::SetQuantum (K_USHORT *usQuantum_*) [inline]

Set the thread's round-robin execution quantum.

Parameters

*usQuantum_*	Thread's execution quantum (in milliseconds)

Definition at line 160 of file thread.h.

13.67.2.17 void Thread::Sleep (K_ULONG *ulTimeMs_*) [static]

Put the thread to sleep for the specified time (in milliseconds).

Actual time slept may be longer (but not less than) the interval specified.

Parameters

*ulTimeMs_*	Time to sleep (in ms)

Definition at line 156 of file thread.cpp.

13.67.2.18 void Thread::Yield (void) [static]

Yield the thread - this forces the system to call the scheduler and determine what thread should run next.

This is typically used when threads are moved in and out of the scheduler.

Definition at line 199 of file thread.cpp.

The documentation for this class was generated from the following files:

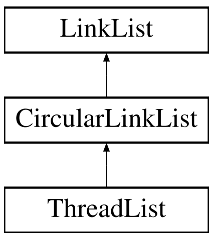
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/thread.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/thread.cpp

13.68 ThreadList Class Reference

This class is used for building thread-management facilities, such as schedulers, and blocking objects.

`#include <threadlist.h>`

Inheritance diagram for ThreadList:



Public Member Functions

- ThreadList ()

  Default constructor - zero-initializes the data.
- void SetPriority (K_UCHAR ucPriority_)

  Set the priority of this threadlist (if used for a scheduler).
- void SetFlagPointer (K_UCHAR ∗pucFlag_)

  Set the pointer to a bitmap to use for this threadlist.
- void Add (LinkListNode ∗node_)

  Add a thread to the threadlist.
- void Add (LinkListNode ∗node_, K_UCHAR ∗pucFlag_, K_UCHAR ucPriority_)

  Add a thread to the threadlist, specifying the flag and priority at the same time.
- void Remove (LinkListNode ∗node_)

  Remove the specified thread from the threadlist.
- Thread ∗ HighestWaiter ()

  Return a pointer to the highest-priority thread in the thread-list.

Private Attributes

- K_UCHAR m_ucPriority

  Priority of the threadlist.
- K_UCHAR ∗ m_pucFlag

  Pointer to the bitmap/flag to set when used for scheduling.

Additional Inherited Members

13.68.1 Detailed Description

This class is used for building thread-management facilities, such as schedulers, and blocking objects.

Definition at line 34 of file threadlist.h.

13.68.2 Member Function Documentation

13.68.2.1 void ThreadList::Add (LinkListNode ∗ *node_*) [virtual]

Add a thread to the threadlist.

Parameters

*node_*	Pointer to the thread (link list node) to add to the list

Reimplemented from CircularLinkList.

Definition at line 46 of file threadlist.cpp.

13.68.2.2 void ThreadList::Add (LinkListNode ∗ *node_*, K_UCHAR ∗ *pucFlag_*, K_UCHAR *ucPriority_*)

Add a thread to the threadlist, specifying the flag and priority at the same time.

Parameters

*node_*	Pointer to the thread to add (link list node)
*pucFlag_*	Pointer to the bitmap flag to set (if used in a scheduler context), or NULL for non-scheduler.
*ucPriority_*	Priority of the threadlist

Definition at line 62 of file threadlist.cpp.

13.68.2.3 Thread ∗ ThreadList::HighestWaiter ( )

Return a pointer to the highest-priority thread in the thread-list.

Returns

Pointer to the highest-priority thread

Definition at line 87 of file threadlist.cpp.

13.68.2.4 void ThreadList::Remove (LinkListNode ∗ *node_*) [virtual]

Remove the specified thread from the threadlist.

Parameters

*node_*	Pointer to the thread to remove

Reimplemented from CircularLinkList.

Definition at line 71 of file threadlist.cpp.

13.68.2.5 void ThreadList::SetFlagPointer (K_UCHAR ∗ *pucFlag_*)

Set the pointer to a bitmap to use for this threadlist.

Once again, only needed when the threadlist is being used for scheduling purposes.

Parameters

*pucFlag_*	Pointer to the bitmap flag

Definition at line 40 of file threadlist.cpp.

13.68.2.6 void ThreadList::SetPriority (K_UCHAR *ucPriority_*)

Set the priority of this threadlist (if used for a scheduler).

Parameters

*ucPriority_*	Priority level of the thread list

Definition at line 34 of file threadlist.cpp.

The documentation for this class was generated from the following files:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/threadlist.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/threadlist.cpp

13.69 ThreadPort Class Reference

Class defining the architecture specific functions required by the kernel.

`#include <threadport.h>`

Static Public Member Functions

- static void StartThreads ()

  Function to start the scheduler, initial threads, etc.

Static Private Member Functions

- static void InitStack (Thread ∗pstThread_)

  Initialize the thread's stack.

Friends

- class **Thread**

13.69.1 Detailed Description

Class defining the architecture specific functions required by the kernel.

This is limited (at this point) to a function to start the scheduler, and a function to initialize the default stack-frame for a thread.

Definition at line 167 of file threadport.h.

13.69.2 Member Function Documentation

13.69.2.1 void ThreadPort::InitStack (Thread ∗ *pstThread_*) `[static], [private]`

Initialize the thread's stack.

Parameters

*pstThread_*	Pointer to the thread to initialize

Definition at line 37 of file threadport.cpp.

The documentation for this class was generated from the following files:

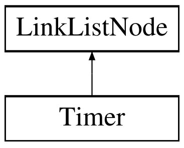
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/threadport.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/threadport.cpp

13.70 Timer Class Reference

Timer - an event-driven execution context based on a specified time interval.

`#include <timerlist.h>`

Inheritance diagram for Timer:



Public Member Functions

- Timer ()

  Default Constructor - zero-initializes all internal data.
- void Start (K_UCHAR bRepeat_, K_ULONG ulIntervalMs_, TimerCallback_t pfCallback_, void ∗pvData_)

  Start a timer using default ownership, using repeats as an option, and millisecond resolution.
- void Stop ()

  Stop a timer already in progress.
- void SetFlags (K_UCHAR ucFlags_)

  Set the timer's flags based on the bits in the ucFlags_ argument.
- void SetCallback (TimerCallback_t pfCallback_)

  Define the callback function to be executed on expiry of the timer.
- void SetData (void ∗pvData_)

  Define a pointer to be sent to the timer callbcak on timer expiry.
- void SetOwner (Thread ∗pclOwner_)

Set the owner-thread of this timer object (all timers must be owned by a thread).

- void SetIntervalTicks (K_ULONG ulTicks_)

  Set the timer expiry in system-ticks (platform specific!)
- void SetIntervalSeconds (K_ULONG ulSeconds_)

  ! The next three cost us 330 bytes of flash on AVR...
- void SetIntervalMSeconds (K_ULONG ulMSeconds_)

  Set the timer expiry interval in milliseconds (platform agnostic)
- void SetIntervalUSeconds (K_ULONG ulUSeconds_)

  Set the timer expiry interval in microseconds (platform agnostic)

Private Attributes

- K_UCHAR m_ucFlags

  Flags for the timer, defining if the timer is one-shot or repeated.
- TimerCallback_t m_pfCallback

  Pointer to the callback function.
- K_ULONG m_ulInterval

  Interval of the timer in timer ticks.
- K_ULONG m_ulTimeLeft

  Time remaining on the timer.
- Thread ∗ m_pclOwner

  Pointer to the owner thread.
- void ∗ m_pvData

  Pointer to the callback data.

Friends

- class **TimerList**

Additional Inherited Members

13.70.1 Detailed Description

Timer - an event-driven execution context based on a specified time interval.

This inherits from a LinkListNode for ease of management by a global TimerList object.

Definition at line 78 of file timerlist.h.

13.70.2 Member Function Documentation

13.70.2.1 void Timer::SetCallback (TimerCallback_t *pfCallback_*) [inline]

Define the callback function to be executed on expiry of the timer.

Parameters

*pfCallback_*	Pointer to the callback function to call

Definition at line 114 of file timerlist.h.

13.70.2.2 void Timer::SetData (void ∗ *pvData_*) [inline]

Define a pointer to be sent to the timer callbcak on timer expiry.

Parameters

*pvData_*	Pointer to data to pass as argument into the callback

Definition at line 123 of file timerlist.h.

13.70.2.3 void Timer::SetFlags (K_UCHAR *ucFlags_*) [inline]

Set the timer's flags based on the bits in the ucFlags_ argument.

Parameters

Flags	to assign to the timer object

Definition at line 105 of file timerlist.h.

13.70.2.4 void Timer::SetIntervalMSeconds (K_ULONG *ulMSeconds_*)

Set the timer expiry interval in milliseconds (platform agnostic)

Parameters

*ulMSeconds_*	Time in milliseconds

Definition at line 270 of file timerlist.cpp.

13.70.2.5 void Timer::SetIntervalSeconds (K_ULONG *ulSeconds_*)

! The next three cost us 330 bytes of flash on AVR...

Set the timer expiry interval in seconds (platform agnostic)

Parameters

*ulSeconds_*	Time in seconds

Definition at line 264 of file timerlist.cpp.

13.70.2.6 void Timer::SetIntervalTicks (K_ULONG *ulTicks_*)

Set the timer expiry in system-ticks (platform specific!)

Parameters

*ulTicks_*	Time in ticks

Definition at line 256 of file timerlist.cpp.

13.70.2.7 void Timer::SetIntervalUSeconds (K_ULONG *ulUSeconds_*)

Set the timer expiry interval in microseconds (platform agnostic)

Parameters

*ulUSeconds_*	Time in microseconds

Definition at line 276 of file timerlist.cpp.

13.70.2.8 void Timer::SetOwner (Thread * *pclOwner_*) [inline]

Set the owner-thread of this timer object (all timers must be owned by a thread).

Parameters

*pclOwner_*	Owner thread of this timer object

Definition at line 133 of file timerlist.h.

13.70.2.9 void Timer::Stop (void)

Stop a timer already in progress.

Has no effect on timers that have already been stopped.

Definition at line 250 of file timerlist.cpp.

The documentation for this class was generated from the following files:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/timerlist.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/timerlist.cpp

13.71 TimerEvent_t Struct Reference

Timer UI event structure.

`#include <gui.h>`

Public Attributes

- K_USHORT usTicks

  Number of clock ticks (arbitrary) that have elapsed.

13.71.1 Detailed Description

Timer UI event structure.

Definition at line 177 of file gui.h.

The documentation for this struct was generated from the following file:

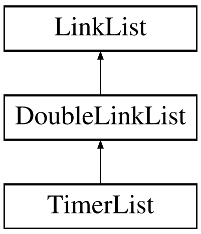
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/gui.h

13.72 TimerList Class Reference

TimerList class - a doubly-linked-list of timer objects.

`#include <timerlist.h>`

Inheritance diagram for TimerList:



Public Member Functions

- void Init ()

  Initialize the TimerList object.
- void Add (Timer ∗pclListNode_)

  Add a timer to the TimerList.
- void Remove (Timer ∗pclListNode_)

  Remove a timer from the TimerList, cancelling its expiry.
- void Process ()

  Process all timers in the timerlist as a result of the timer expiring.

Private Attributes

- K_ULONG m_ulNextWakeup

  The time (in system clock ticks) of the next wakeup event.
- K_UCHAR m_bTimerActive

  Whether or not the timer is active.

Additional Inherited Members

13.72.1 Detailed Description

TimerList class - a doubly-linked-list of timer objects.

Definition at line 198 of file timerlist.h.

13.72.2 Member Function Documentation

13.72.2.1 void TimerList::Add (Timer ∗ *pclListNode_*)

Add a timer to the TimerList.

Parameters

*pclListNode_*	Pointer to the Timer to Add

Definition at line 55 of file timerlist.cpp.

13.72.2.2 void TimerList::Init (void)

Initialize the TimerList object.

Must be called before using the object.

Reimplemented from LinkList.

Definition at line 48 of file timerlist.cpp.

13.72.2.3 void TimerList::Process (void)

Process all timers in the timerlist as a result of the timer expiring.

This will select a new timer epoch based on the next timer to expire. ToDo - figure out if we need to deal with any overtime here.

Definition at line 110 of file timerlist.cpp.

13.72.2.4 void TimerList::Remove (Timer ∗ *pclListNode_*)

Remove a timer from the TimerList, cancelling its expiry.

Parameters

*pclListNode_*	Pointer to the Timer to remove

Definition at line 95 of file timerlist.cpp.

The documentation for this class was generated from the following files:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/timerlist.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/timerlist.cpp

13.73 TimerScheduler Class Reference

"Static" Class used to interface a global TimerList with the rest of the kernel.

`#include <timerlist.h>`

Static Public Member Functions

- static void Init ()

  Initialize the timer scheduler.
- static void Add (Timer ∗pclListNode_)

  Add a timer to the timer scheduler.
- static void Remove (Timer ∗pclListNode_)

  Remove a timer from the timer scheduler.
- static void Process ()

  This function must be called on timer expiry (from the timer's ISR context).

Static Private Attributes

- static TimerList m_clTimerList

  TimerList object manipulated by the Timer Scheduler.

13.73.1 Detailed Description

"Static" Class used to interface a global TimerList with the rest of the kernel.

Definition at line 248 of file timerlist.h.

13.73.2 Member Function Documentation

13.73.2.1 void TimerScheduler::Add (Timer ∗ *pclListNode_*) [inline], [static]

Add a timer to the timer scheduler.

Adding a timer implicitly starts the timer as well.

Parameters

*pclListNode_*	Pointer to the timer list node to add

Definition at line 267 of file timerlist.h.

13.73.2.2 void TimerScheduler::Init (void) [inline], [static]

Initialize the timer scheduler.

Must be called before any timer, or timer-derived functions are used.

Definition at line 257 of file timerlist.h.

13.73.2.3 void TimerScheduler::Process (void) [inline], [static]

This function must be called on timer expiry (from the timer's ISR context).

This will result in all timers being updated based on the epoch that just elapsed. New timer epochs are set based on the next timer to expire.

Definition at line 289 of file timerlist.h.

13.73.2.4 void TimerScheduler::Remove (Timer ∗ *pclListNode_*) [inline], [static]

Remove a timer from the timer scheduler.

May implicitly stop the timer if this is the only active timer scheduled.

Parameters

*pclListNode_*	Pointer to the timer list node to remove

Definition at line 278 of file timerlist.h.

The documentation for this class was generated from the following files:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/timerlist.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/timerlist.cpp

13.74 TouchEvent_t Struct Reference

Touch UI event structure.

```
#include <gui.h>
```

Public Attributes

- K_USHORT usX

    Absolute touch location (pixels)
- K_USHORT usY

    Absolute touch location (pixels)
- union {
    K_USHORT ucFlags
      Modifier flags.
    struct {
      unsigned int bTouch:1
        Whether or not touch is up or down.
    }
  };

13.74.1 Detailed Description

Touch UI event structure.

Definition at line 125 of file gui.h.

The documentation for this struct was generated from the following file:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/gui.h

13.75 UnitTest Class Reference

Class used to implement a simple unit-testing framework.

`#include <unit_test.h>`

Public Member Functions

- void SetName (const K_CHAR ∗szName_)

    Set the name of the test object.
- void Start ()

    Start a new test iteration.
- void Pass ()

    Stop the current iteration (if started), and register that the test was successful.
- void Fail ()

    Stop the current iterations (if started), and register that the current test failed.
- void Complete ()

    Complete the test.
- const K_CHAR ∗ GetName ()

    Get the name of the tests associated with this object.
- K_BOOL GetResult ()

    Return the result of the last test.
- K_USHORT GetPassed ()

    Return the total number of test points/iterations passed.
- K_USHORT GetFailed ()

    Return the number of failed test points/iterations.
- K_USHORT GetTotal ()

    Return the total number of iterations/test-points executed.

Private Attributes

- const K_CHAR ∗ m_szName

  Name of the tests performed.
- K_BOOL m_bIsActive

  Whether or not the test is active.
- K_UCHAR m_bComplete

  Whether or not the test is complete.
- K_BOOL m_bStatus

  Status of the last-run test.
- K_USHORT m_usIterations

  Number of iterations executed.
- K_USHORT m_usPassed

  Number of iterations that have passed.

13.75.1 Detailed Description

Class used to implement a simple unit-testing framework.

Definition at line 28 of file unit_test.h.

13.75.2 Member Function Documentation

13.75.2.1 void UnitTest::Complete ( ) `[inline]`

Complete the test.

Once a test has been completed, no new iterations can be started (i.e Start()/Pass()/Fail() will have no effect).

Definition at line 72 of file unit_test.h.

13.75.2.2 K_USHORT UnitTest::GetFailed ( ) `[inline]`

Return the number of failed test points/iterations.

Returns

Failed test point/iteration count

Definition at line 108 of file unit_test.h.

13.75.2.3 const K_CHAR ∗ UnitTest::GetName ( ) `[inline]`

Get the name of the tests associated with this object.

Returns

Name of the test

Definition at line 81 of file unit_test.h.

13.75.2.4 K_USHORT UnitTest::GetPassed ( ) [inline]

Return the total number of test points/iterations passed.

Returns

Count of all successful test points/iterations

Definition at line 99 of file unit_test.h.

13.75.2.5 K_BOOL UnitTest::GetResult ( ) [inline]

Return the result of the last test.

Returns

Status of the last run test (false = fail, true = pass)

Definition at line 90 of file unit_test.h.

13.75.2.6 K_USHORT UnitTest::GetTotal ( ) [inline]

Return the total number of iterations/test-points executed.

Returns

Total number of ierations/test-points executed

Definition at line 117 of file unit_test.h.

13.75.2.7 void UnitTest::SetName (const K_CHAR ∗ *szName_*) [inline]

Set the name of the test object.

Parameters

*szName_*	Name of the tests associated with this object

Definition at line 41 of file unit_test.h.

The documentation for this class was generated from the following files:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/unit_test.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/unit_test.cpp

13.76 WriteBuffer16 Class Reference

This class is used to provide a general-purpose, fully thread-safe circular buffer implementation which can be used for creating tracebuffers, data logging queues, transaction queues, etc.

`#include <writebuf16.h>`

Public Member Functions

- void SetBuffers (K_USHORT ∗pusData_, K_USHORT usSize_)

Assign the data to be used as storage for this circular buffer.

- void SetCallback (WriteBufferCallback pfCallback_)

  Set the callback function to be called when the buffer hits 50% of its capacity, and again when the buffer rolls over completely.
- void WriteData (K_USHORT ∗pusBuf_, K_USHORT usLen_)

  Write an array of values to the circular buffer.
- void WriteVector (K_USHORT ∗∗ppusBuf_, K_USHORT ∗pusLen_, K_UCHAR ucCount_)

  Write a multi-part vector to the circular buffer.

Private Attributes

- K_USHORT ∗ m_pusData

  Pointer to the circular buffer data.
- volatile K_USHORT m_usSize

  Size of the buffer.
- volatile K_USHORT m_usHead

  Current head element (where data is written)
- volatile K_USHORT m_usTail

  Current tail element (where data is read)
- WriteBufferCallback m_pfCallback

  Buffer callback function.

13.76.1 Detailed Description

This class is used to provide a general-purpose, fully thread-safe circular buffer implementation which can be used for creating tracebuffers, data logging queues, transaction queues, etc.

We use it for implementing a debug print journal.

Definition at line 37 of file writebuf16.h.

13.76.2 Member Function Documentation

13.76.2.1 void WriteBuffer16::SetBuffers (K_USHORT ∗ *pusData_*, K_USHORT *usSize_*) [inline]

Assign the data to be used as storage for this circular buffer.

Parameters

*pusData_*	Pointer to the array of data to be managed as a circular buffer by this object.
*usSize_*	Size of the buffer in 16-bit elements

Definition at line 50 of file writebuf16.h.

13.76.2.2 void WriteBuffer16::SetCallback (WriteBufferCallback *pfCallback_*) [inline]

Set the callback function to be called when the buffer hits 50% of its capacity, and again when the buffer rolls over completely.

Parameters

*pfCallback_*	Function pointer to call whenever the buffer has reached 50% capacity, or has rolled over completely.

Definition at line 69 of file writebuf16.h.

13.76.2.3 void WriteBuffer16::WriteData (K_USHORT ∗ *pusBuf_*, K_USHORT *usLen_*)

Write an array of values to the circular buffer.

Parameters

*pusBuf_*	Source data array to write to the circular buffer
*usLen_*	Length of the source data array in 16-bit elements

Definition at line 25 of file writebuf16.cpp.

13.76.2.4 void WriteBuffer16::WriteVector (K_USHORT ∗∗ *ppusBuf_*, K_USHORT ∗ *pusLen_*, K_UCHAR *ucCount_*)

Write a multi-part vector to the circular buffer.

Parameters

*ppusBuf_*	Pointer to the array of source data pointers
*pusLen_*	Array of buffer lengths
*ucCount_*	Number of source-data arrays to write to the buffer

Definition at line 37 of file writebuf16.cpp.

The documentation for this class was generated from the following files:

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/writebuf16.h
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/writebuf16.cpp

Chapter 14

File Documentation

14.1 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/blocking.cpp File Reference

Implementation of base class for blocking objects.

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "kernel_debug.h"
#include "blocking.h"
#include "thread.h"
```

Macros

- #define **__FILE_ID__** BLOCKING_CPP

14.1.1 Detailed Description

Implementation of base class for blocking objects.

Definition in file blocking.cpp.

14.2 blocking.cpp

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "kernel_debug.h"

#include "blocking.h"
#include "thread.h"

//---------------------------------------------------------------------------
#if defined __FILE_ID__
```

```
    #undef __FILE_ID__
#endif
#define __FILE_ID__     BLOCKING_CPP

#if KERNEL_USE_SEMAPHORE || KERNEL_USE_MUTEX
//---------------------------------------------------------------------------
void BlockingObject::Block(Thread *pclThread_)
{
    KERNEL_ASSERT( pclThread_ );
    KERNEL_TRACE_1( STR_THREAD_BLOCK_1, (K_USHORT)pclThread_->GetID() );

    // Remove the thread from its current thread list (the "owner" list)
    // ... And add the thread to this object's block list
    Scheduler::Remove(pclThread_);
    m_clBlockList.Add(pclThread_);

    // Set the "current" list location to the blocklist for this thread
    pclThread_->SetCurrent(&m_clBlockList);

}

//---------------------------------------------------------------------------
void BlockingObject::UnBlock(Thread *pclThread_)
{
    KERNEL_ASSERT( pclThread_ );
    KERNEL_TRACE_1( STR_THREAD_UNBLOCK_1, (K_USHORT)pclThread_->GetID() );

    // Remove the thread from its current thread list (the "owner" list)
    pclThread_->GetCurrent()->Remove(pclThread_);

    // Put the thread back in its active owner's list.  This is usually
    // the ready-queue at the thread's original priority.
    Scheduler::Add(pclThread_);

    // Tag the thread's current list location to its owner
    pclThread_->SetCurrent(pclThread_->GetOwner());
}

#endif
```

14.3 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/blocking.h File Reference

Blocking object base class declarations.

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "ll.h"
#include "threadlist.h"
#include "thread.h"
```

Classes

- class BlockingObject

  Class implementing thread-blocking primatives.

14.3.1 Detailed Description

Blocking object base class declarations. A Blocking object in Mark3 is essentially a thread list. Any blocking object implementation (being a semaphore, mutex, event flag, etc.) can be built on top of this class, utilizing the provided functions to manipulate thread location within the Kernel.

Blocking a thread results in that thread becoming de-scheduled, placed in the blocking object's own private list of threads which are waiting on the object.

Unblocking a thread results in the reverse: The thread is moved back to its original location from the blocking list.

The only difference between a blocking object based on this class is the logic used to determine what consitutes a Block or Unblock condition.

For instance, a semaphore Pend operation may result in a call to the Block() method with the currently-executing thread in order to make that thread wait for a semaphore Post. That operation would then invoke the UnBlock() method, removing the blocking thread from the semaphore's list, and back into the the appropriate thread inside the scheduler.

Care must be taken when implementing blocking objects to ensure that critical sections are used judiciously, otherwise asynchronous events like timers and interrupts could result in non-deterministic and often catastrophic behavior.

Definition in file blocking.h.

14.4 blocking.h

```
/*===========================================================================
     _____     _____     _____     _____
 ___|    _|__ __|_    |__ __|__   |__ __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|     |_____|     |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#ifndef __BLOCKING_H__
#define __BLOCKING_H__

#include "kerneltypes.h"
#include "mark3cfg.h"

#include "ll.h"
#include "threadlist.h"
#include "thread.h"

#if KERNEL_USE_MUTEX || KERNEL_USE_SEMAPHORE

//---------------------------------------------------------------------------
class BlockingObject
{
protected:
    void Block(Thread *pclThread_ );

    void UnBlock(Thread *pclThread_);

    ThreadList m_clBlockList;
};

#endif

#endif
```

14.5 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_button.cpp File Reference

GUI Button Control Implementation.

```
#include "control_button.h"
#include "gui.h"
```

14.5.1 Detailed Description

GUI Button Control Implementation. Basic pushbutton control with an up/down state.

Definition in file control_button.cpp.

14.6 control_button.cpp

```
/*===========================================================================
     _____     _____     _____     _____
 ___|    _|__ __|_    |__ __|__   |__ __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "control_button.h"
#include "gui.h"


void ButtonControl::Init()
{
    m_szCaption = "Button";
    m_pstFont = NULL;
    m_uBGColor = COLOR_GREY50;
    m_uActiveColor = COLOR_GREY25;
    m_uLineColor = COLOR_GREY62;
    m_uTextColor = COLOR_WHITE;
    m_bState = false;
    m_pfCallback = NULL;
    m_pvCallbackData = NULL;
    SetAcceptFocus(true);
}
//---------------------------------------------------------------------------
void ButtonControl::Draw()
{
    DrawText_t stText;
    DrawLine_t stLine;

    GraphicsDriver *pclDriver = GetParentWindow()
->GetDriver();

    K_USHORT usXOffset = 0;
    K_USHORT usHalfWidth = 0;
    K_USHORT usYOffset = 0;

    // Get the location of the control relative to elements higher in the
 heirarchy
    GetControlOffset(&usXOffset, &usYOffset);

    // Draw the rounded-off rectangle
    stLine.usX1 = GetLeft() + usXOffset;
    stLine.usX2 = stLine.usX1 + GetWidth() - 1;
    stLine.usY1 = GetTop() + usYOffset;
    stLine.usY2 = stLine.usY1;
    stLine.uColor = m_uLineColor;
    pclDriver->Line(&stLine);

    stLine.usY1 = GetTop() + GetHeight() + usYOffset - 1;
    stLine.usY2 = stLine.usY1;
    pclDriver->Line(&stLine);

    stLine.usX1 = GetLeft() + usXOffset;
    stLine.usX2 = stLine.usX1;
    stLine.usY1 = GetTop() + usYOffset + 1;
    stLine.usY2 = GetTop() + GetHeight() - 2;
    pclDriver->Line(&stLine);

    stLine.usX1 = GetLeft() + GetWidth() + usXOffset - 1;
    stLine.usX2 = stLine.usX1;
    pclDriver->Line(&stLine);

    // Draw a rectangle before the text if the BG is specified.
    {
        DrawRectangle_t stRect;
        stRect.usLeft = GetLeft() + usXOffset + 1;
        stRect.usRight = GetLeft() + GetWidth() +
usXOffset - 2;
        stRect.usTop = GetTop() + usYOffset + 1;
        stRect.usBottom = GetTop() + GetHeight() +
usYOffset - 2;
        stRect.bFill = true;

        if (m_bState)
        {
            stRect.uFillColor = m_uActiveColor;
        }
        else
        {
            stRect.uFillColor = m_uBGColor;
        }

        if (GetParentWindow()->IsInFocus(this))
        {
            stRect.uLineColor = m_uLineColor;
        }
        else
        {
            stRect.uLineColor = m_uFillColor;
        }

        pclDriver->Rectangle(&stRect);
    }

    // Draw the Text
    stText.pstFont = m_pstFont;
    stText.pcString = m_szCaption;
    stText.uColor = m_uTextColor;
    usHalfWidth = pclDriver->TextWidth(&stText);
    usHalfWidth >>= 1;
    stText.usLeft = GetLeft() + (GetWidth()>>1) -
usHalfWidth + usXOffset;
    stText.usTop = GetTop() + usYOffset;
    pclDriver->Text(&stText);
}

//---------------------------------------------------------------------------
GuiReturn_t ButtonControl::ProcessEvent( GuiEvent_t
 *pstEvent_ )
{
    K_USHORT usXOffset, usYOffset;

    GetControlOffset(&usXOffset, &usYOffset);

    GUI_DEBUG_PRINT("ButtonControl::ProcessEvent\n");

    switch (pstEvent_->ucEventType)
    {
        case EVENT_TYPE_KEYBOARD:
        {
            // If this is a space bar or an enter key, behave like a mouse
 click.
            if ((KEYCODE_SPACE == pstEvent_->stKey.ucKeyCode) ||
                (KEYCODE_RETURN == pstEvent_->stKey.ucKeyCode))
            {
                if (pstEvent_->stKey.bKeyState)
                {
                    m_bState = true;
                }
                else
                {
                    m_bState = false;
                    if (m_pfCallback)
                    {
                        m_pfCallback(m_pvCallbackData);
                    }
                }
                SetStale();
            }
        }
            break;
        case EVENT_TYPE_MOUSE:
        {
            // Is this control currently in the "active"/pressed state?
            if (m_bState)
            {
                // Check to see if the movement is out-of-bounds based on the
 coordinates.
                // If so, de-activate the control
                if (pstEvent_->stMouse.bLeftState)
                {
                    if ((pstEvent_->stMouse.usX < GetLeft() +
usXOffset) ||
                        (pstEvent_->stMouse.usX >= GetLeft() +
usXOffset + GetWidth()-1) ||
                        (pstEvent_->stMouse.usY < GetTop() + usYOffset
) ||
                        (pstEvent_->stMouse.usY >= GetTop() +
usYOffset + GetHeight() - 1))
```

```
                    {
                        m_bState = false;
                        SetStale();
                    }
                }
                // left button state is now up, and the control was previously active.
                // Run the event callback for the mouse, and go from there.
                else
                {
                    if ((pstEvent_->stMouse.usX >= GetLeft() + usXOffset) &&
                        (pstEvent_->stMouse.usX < GetLeft() + usXOffset + GetWidth()-1) &&
                        (pstEvent_->stMouse.usY >= GetTop() + usYOffset) &&
                        (pstEvent_->stMouse.usY < GetTop() + usYOffset + GetHeight() - 1))
                    {
                        m_bState = false;
                        SetStale();
                        if (m_pfCallback)
                        {
                            m_pfCallback(m_pvCallbackData);
                        }
                    }
                }
            }
            else if (!m_bState)
            {
                // If we registered a down-click in the bounding box, set the state of the
                // control to activated.
                if (pstEvent_->stMouse.bLeftState)
                {
                    if ((pstEvent_->stMouse.usX >= GetLeft() + usXOffset) &&
                        (pstEvent_->stMouse.usX < GetLeft() + usXOffset + GetWidth()-1) &&
                        (pstEvent_->stMouse.usY >= GetTop() + usYOffset) &&
                        (pstEvent_->stMouse.usY < GetTop() + usYOffset + GetHeight() - 1))
                    {
                        m_bState = true;
                        SetStale();
                    }
                }
            }

            if (!IsInFocus())
            {
                GetParentWindow()->SetFocus(this);
                SetStale();
            }

        }
            break;
    }

}

//---------------------------------------------------------------------------
void ButtonControl::Activate( bool bActivate_ )
{
    // When we de-activate the control, simply disarm the control and force
    // a redraw
    if (!bActivate_)
    {
        m_bState = false;
    }
    SetStale();
}
```

14.7 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_button.h File Reference

GUI Button Control.

```
#include "gui.h"
#include "kerneltypes.h"
#include "draw.h"
#include "font.h"
```

Classes

- class ButtonControl

Typedefs

- typedef void(∗ **ButtonCallback**)(void ∗pvData_)

14.7.1 Detailed Description

GUI Button Control. Basic pushbutton control with an up/down state.

Definition in file control_button.h.

14.8 control_button.h

```

/*===========================================================================
     _____       _____        _____       _____
 ___|    _|__ __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|       |_____|       |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#ifndef __CONTROL_BUTTON_H__
#define __CONTROL_BUTTON_H__

#include "gui.h"
#include "kerneltypes.h"
#include "draw.h"
#include "font.h"

typedef void (*ButtonCallback)( void *pvData_ );

class ButtonControl : public GuiControl
{
public:

    virtual void Init();
    virtual void Draw();
    virtual GuiReturn_t ProcessEvent( GuiEvent_t *
pstEvent_ );
    virtual void Activate( bool bActivate_ );

    void SetBGColor( COLOR eColor_ )        { m_uBGColor = eColor_; }
    void SetLineColor( COLOR eColor_ )      { m_uLineColor = eColor_; }
    void SetFillColor( COLOR eColor_ )      { m_uFillColor = eColor_; }
    void SetTextColor( COLOR eColor_ )      { m_uTextColor = eColor_; }
    void SetActiveColor( COLOR eColor_ )    { m_uActiveColor = eColor_; }

    void SetFont( Font_t *pstFont_ )        { m_pstFont = pstFont_; }

    void SetCaption( const K_CHAR *szCaption_ )     { m_szCaption = szCaption_;
}

    void SetCallback( ButtonCallback pfCallback_, void *pvData_ )
        { m_pfCallback = pfCallback_; m_pvCallbackData = pvData_; }
private:

    const K_CHAR *m_szCaption;
    Font_t *m_pstFont;
    COLOR   m_uBGColor;
    COLOR   m_uActiveColor;
    COLOR   m_uLineColor;
    COLOR   m_uFillColor;
    COLOR   m_uTextColor;
    bool    m_bState;

    void *m_pvCallbackData;
    ButtonCallback m_pfCallback;
};


#endif

```

14.9 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_checkbox.cpp File Reference

Checkbox Control.

```
#include "gui.h"
#include "kerneltypes.h"
#include "draw.h"
#include "font.h"
#include "control_checkbox.h"
```

Macros

- #define **TEXT_X_OFFSET** (13)

Variables

- static const K_UCHAR **aucBox** []
- static const K_UCHAR **aucCheck** []

14.9.1 Detailed Description

Checkbox Control. A binary On/Off switch control

Definition in file control_checkbox.cpp.

14.9.2 Variable Documentation

14.9.2.1 const K_UCHAR aucBox[ ] [static]

Initial value:

```
{ 0xFF,
  0x81,
  0x81,
  0x81,
  0x81,
  0x81,
  0x81,
  0xFF }
```

Definition at line 31 of file control_checkbox.cpp.

14.9.2.2 const K_UCHAR aucCheck[ ] [static]

Initial value:

```
{ 0,
  0,
  0x3C,
  0x3C,
  0x3C,
  0x3C,
  0,
  0 }
```

Definition at line 42 of file control_checkbox.cpp.

14.10 control_checkbox.cpp

```
00001 /*============================================================================
00002      _____        _____        _____        _____
00003  ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
00004 |    \  /  | ||    \      ||     |     ||  |/ /    ||___   |
00005 |     \/   | ||     \     ||     \     ||     \    ||___   |
00006 |__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\ __||______|
00007     |_____|      |_____|      |_____|      |_____|
00008
00009 --[Mark3 Realtime Platform]--------------------------------------------------
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 ===========================================================================*/
00021 #include "gui.h"
00022 #include "kerneltypes.h"
00023 #include "draw.h"
00024 #include "font.h"
00025 #include "control_checkbox.h"
00026
00027 //---------------------------------------------------------------------------
00028 #define TEXT_X_OFFSET       (13)
00029
00030 //---------------------------------------------------------------------------
00031 static const K_UCHAR aucBox[] =
00032 { 0xFF,
00033   0x81,
00034   0x81,
00035   0x81,
00036   0x81,
00037   0x81,
00038   0x81,
00039   0xFF };
00040
00041 //---------------------------------------------------------------------------
00042 static const K_UCHAR aucCheck[] =
00043 { 0,
00044   0,
00045   0x3C,
00046   0x3C,
00047   0x3C,
00048   0x3C,
00049   0,
00050   0 };
00051
00052 //---------------------------------------------------------------------------
00053 void CheckBoxControl::Init()
00054 {
00055     SetAcceptFocus(true);
00056 }
00057
00058 //---------------------------------------------------------------------------
00059 void CheckBoxControl::Draw()
00060 {
00061     GraphicsDriver *pclDriver = GetParentWindow()
      ->GetDriver();
00062     K_USHORT usX, usY;
00063     K_USHORT usTextWidth;
00064
00065     GetControlOffset(&usX, &usY);
00066
00067     // Draw the box, (and check, if necessary)
00068     {
        DrawRectangle_t stRect;
        stRect.uLineColor = m_uBackColor;
        stRect.uFillColor = m_uBackColor;
        stRect.usTop = usY + GetTop() + ((GetHeight() - 4)
>> 1);
        stRect.usLeft = usX + GetLeft();
        stRect.usRight = stRect.usLeft + 8;
        stRect.usBottom = stRect.usTop + 8;
        stRect.bFill = true;
        pclDriver->Rectangle(&stRect);
    }

    {
        DrawStamp_t stStamp;
        stStamp.uColor = m_uBoxColor;
        stStamp.usY = usY + GetTop() + ((GetHeight() - 4) >>
1);
        stStamp.usX = usX + GetLeft();
        stStamp.usWidth = 8;
        stStamp.usHeight = 8;
        stStamp.pucData = (K_UCHAR*)aucBox;
        pclDriver->Stamp(&stStamp);

        if (m_bChecked)
        {
            stStamp.pucData = (K_UCHAR*)aucCheck;
            pclDriver->Stamp(&stStamp);
        }
    }

    // Draw the caption
    {
        DrawText_t stText;
        stText.usLeft = usX + GetLeft() + TEXT_X_OFFSET;
        stText.usTop = usY + GetTop();
        stText.uColor = m_uFontColor;
        stText.pstFont = m_pstFont;
        stText.pcString = m_szCaption;

        usTextWidth = pclDriver->TextWidth(&stText);
        pclDriver->Text(&stText);
    }
}

//---------------------------------------------------------------------------
GuiReturn_t CheckBoxControl::ProcessEvent(
GuiEvent_t *pstEvent_ )
{
    K_USHORT usXOffset, usYOffset;

    GetControlOffset(&usXOffset, &usYOffset);

    GUI_DEBUG_PRINT("ButtonControl::ProcessEvent\n");

    switch (pstEvent_->ucEventType)
    {
        case EVENT_TYPE_KEYBOARD:
        {
            // If this is a space bar or an enter key, behave like a mouse
 click.
            if ((KEYCODE_SPACE == pstEvent_->stKey.ucKeyCode) ||
                (KEYCODE_RETURN == pstEvent_->stKey.ucKeyCode))
            {
                if (pstEvent_->stKey.bKeyState)
                {
                    m_bChecked = true;
                }
                else
                {
                    m_bChecked = false;
                }
                SetStale();
            }
        }
            break;
        case EVENT_TYPE_MOUSE:
        {
            // Is this control currently in the "active"/pressed state?
            if (m_bChecked)
            {
                // Check to see if the movement is out-of-bounds based on the
 coordinates.
                // If so, de-activate the control
                if (pstEvent_->stMouse.bLeftState)
                {
                    if ((pstEvent_->stMouse.usX >= GetLeft() +
usXOffset) &&
                        (pstEvent_->stMouse.usX < GetLeft() +
     usXOffset + GetWidth()-1) &&
                        (pstEvent_->stMouse.usY >= GetTop() +
     usYOffset) &&
                        (pstEvent_->stMouse.usY < GetTop() + usYOffset
      + GetHeight() - 1))
                    {
                        m_bChecked = false;
                        SetStale();
                    }
                }
            }
            else if (!m_bChecked)
            {
                // If we registered a down-click in the bounding box, set the
      state of the
                // control to activated.
                if (pstEvent_->stMouse.bLeftState)
                {
                    if ((pstEvent_->stMouse.usX >= GetLeft() +
     usXOffset) &&
                        (pstEvent_->stMouse.usX < GetLeft() +
     usXOffset + GetWidth()-1) &&
                        (pstEvent_->stMouse.usY >= GetTop() +
     usYOffset) &&
                        (pstEvent_->stMouse.usY < GetTop() + usYOffset
      + GetHeight() - 1))
                    {
                        m_bChecked = true;
                        SetStale();
                    }
                }
            }

            if (!IsInFocus())
            {
                GetParentWindow()->SetFocus(this);
                SetStale();
            }
        }
            break;
    }
}
```

14.11 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_checkbox.h File Reference

Checkbox Control.

```
#include "gui.h"
#include "kerneltypes.h"
#include "draw.h"
#include "font.h"
```

Classes

- class CheckBoxControl

14.11.1 Detailed Description

Checkbox Control. A binary On/Off switch control

Definition in file control_checkbox.h.

14.12 control_checkbox.h

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#ifndef __CONTROL_CHECKBOX_H__
#define __CONTROL_CHECKBOX_H__

#include "gui.h"
#include "kerneltypes.h"
#include "draw.h"
#include "font.h"

class CheckBoxControl : public GuiControl
{
public:
    virtual void Init();
    virtual void Draw();
    virtual GuiReturn_t ProcessEvent( GuiEvent_t *
pstEvent_ );
    virtual void Activate( bool bActivate_ )  { SetStale(); }

    void SetFont( Font_t *pstFont_ )         { m_pstFont    = pstFont_; }
    void SetCaption( const char *szCaption_ ) { m_szCaption = szCaption_; }
    void SetCheck( bool bChecked_ )           { m_bChecked   = bChecked_; }
    void SetFontColor( COLOR uFontColor_ )    { m_uFontColor = uFontColor_; }
    void SetBoxColor( COLOR uBoxColor_ )      { m_uBoxColor  = uBoxColor_; }
    void SetBackColor( COLOR uBackColor_ )    { m_uBackColor = uBackColor_; }
    bool IsChecked( void )                    { return m_bChecked; }

private:
    const char *m_szCaption;
    COLOR m_uBackColor;
    COLOR m_uBoxColor;
    COLOR m_uFontColor;
    Font_t *m_pstFont;
    bool m_bChecked;
};

#endif

```

14.13 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_groupbox.cpp File Reference

GUI GroupBox Control Implementation.

```
#include "gui.h"
#include "kerneltypes.h"
#include "draw.h"
#include "graphics.h"
#include "control_groupbox.h"
```

Macros

- #define **BORDER_OFFSET** (4)
- #define **TEXT_X_OFFSET** (8)
- #define **TEXT_Y_OFFSET** (0)

14.13.1 Detailed Description

GUI GroupBox Control Implementation.

Definition in file control_groupbox.cpp.

14.14 control_groupbox.cpp

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "gui.h"
#include "kerneltypes.h"
#include "draw.h"
#include "graphics.h"
#include "control_groupbox.h"

#define BORDER_OFFSET           (4)
#define TEXT_X_OFFSET           (8)
#define TEXT_Y_OFFSET           (0)

//---------------------------------------------------------------------------
void GroupBoxControl::Draw()
{
    GUI_DEBUG_PRINT( "GroupBoxControl::Draw()\n");
    GraphicsDriver *pclDriver = GetParentWindow()
->GetDriver();
    K_USHORT usX, usY;
    K_USHORT usTextWidth;

    GetControlOffset(&usX, &usY);

    // Draw the background panel
    {
        DrawRectangle_t stRectangle;
        stRectangle.usTop = GetTop() + usY;
        stRectangle.usBottom = stRectangle.usTop + GetHeight
() -1;
        stRectangle.usLeft = GetLeft() + usX;
        stRectangle.usRight = stRectangle.usLeft + GetWidth
() -1;
        stRectangle.bFill = true;
        stRectangle.uLineColor = m_uPanelColor;
        stRectangle.uFillColor = m_uPanelColor;

        pclDriver->Rectangle(&stRectangle);
    }

    // Draw the caption
    {
        DrawText_t stText;
        stText.usLeft = usX + TEXT_X_OFFSET;
        stText.usTop = usY + TEXT_Y_OFFSET;
        stText.uColor = m_uFontColor;
        stText.pstFont = m_pstFont;
        stText.pcString = m_pcCaption;

        usTextWidth = pclDriver->TextWidth(&stText);
        pclDriver->Text(&stText);
    }

    // Draw the lines surrounding the panel
    {
        DrawLine_t stLine;

        stLine.uColor = m_uLineColor;
        stLine.usY1 = GetTop() + usY + BORDER_OFFSET;
        stLine.usY2 = GetTop() + usY + GetHeight() -
BORDER_OFFSET - 1;
        stLine.usX1 = usX + BORDER_OFFSET;
        stLine.usX2 = usX + BORDER_OFFSET;
        pclDriver->Line(&stLine);

        stLine.usY1 = GetTop() + usY + BORDER_OFFSET;
        stLine.usY2 = GetTop() + usY + GetHeight() -
BORDER_OFFSET - 1;
        stLine.usX1 = usX + GetWidth() - BORDER_OFFSET - 1;
        stLine.usX2 = usX + GetWidth() - BORDER_OFFSET - 1;
        pclDriver->Line(&stLine);

        stLine.usY1 = GetTop() + usY + GetHeight() -
     BORDER_OFFSET - 1;
        stLine.usY2 = GetTop() + usY + GetHeight() -
     BORDER_OFFSET - 1;
        stLine.usX1 = usX + BORDER_OFFSET;
        stLine.usX2 = usX + GetWidth() - BORDER_OFFSET - 1;
        pclDriver->Line(&stLine);

        stLine.usY1 = GetTop() + BORDER_OFFSET - 1;
        stLine.usY2 = GetTop() + BORDER_OFFSET - 1;
        stLine.usX1 = usX + BORDER_OFFSET;
        stLine.usX2 = usX + TEXT_X_OFFSET - 2;
        pclDriver->Line(&stLine);

        stLine.usX1 = usX + TEXT_X_OFFSET + usTextWidth;
        stLine.usX2 = usX + GetWidth() - BORDER_OFFSET - 1;
        pclDriver->Line(&stLine);
    }


}
```

14.15 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_groupbox.h File Reference

GUI Group Box Control.

```
#include "gui.h"
#include "kerneltypes.h"
#include "draw.h"
```

Classes

- class GroupBoxControl

14.15.1 Detailed Description

GUI Group Box Control. A groupbox control is essentially a panel with a text caption, and a lined border.

Definition in file control_groupbox.h.

14.16 control_groupbox.h

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__    |__  __| __   |__  ______
|    \  /  | ||    \      ||     |      ||  |/ /      ||___   |
|     \/   | ||     \     ||     \      ||     \      ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#ifndef __CONTROL_GROUPBOX_H__
#define __CONTROL_GROUPBOX_H__

#include "gui.h"
#include "kerneltypes.h"
#include "draw.h"

class GroupBoxControl : public GuiControl
{
```

```
public:
    virtual void Init() { m_uLineColor = COLOR_BLACK;
                          m_uFontColor = COLOR_GREY25;
                          m_uPanelColor = COLOR_GREY75;
                          SetAcceptFocus(false); }
    virtual void Draw();
    virtual GuiReturn_t ProcessEvent( GuiEvent_t *
pstEvent_ ) {};
    virtual void Activate( bool bActivate_ ) {}

    void SetPanelColor( COLOR eColor_ ) { m_uPanelColor = eColor_; }
    void SetLineColor( COLOR eColor_ ) { m_uLineColor = eColor_; }
    void SetFontColor( COLOR eColor_ ) { m_uFontColor = eColor_; }
    void SetFont( Font_t *pstFont_ ) { m_pstFont = pstFont_; }
    void SetCaption( const K_CHAR *pcCaption_ ) { m_pcCaption = pcCaption_; }
private:
    COLOR m_uPanelColor;
    COLOR m_uLineColor;
    COLOR m_uFontColor;

    Font_t *m_pstFont;
    const K_CHAR *m_pcCaption;
};

#endif

```

14.17 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_label.h File Reference

GUI Label Control.

```
#include "gui.h"
#include "kerneltypes.h"
#include "draw.h"
#include "font.h"
```

Classes

- class LabelControl

14.17.1 Detailed Description

GUI Label Control. A label control is a static text eliment, specified by a font, a color, and a string to overlay at a given location.

Definition in file control_label.h.

14.18 control_label.h

```
/*===========================================================================
     _____     _____     _____     _____
 ___|    _|__ __|_    |__ __|__   |__ __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|     |_____|     |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#ifndef __CONTROL_LABEL_H__
#define __CONTROL_LABEL_H__

#include "gui.h"
```

```
#include "kerneltypes.h"
#include "draw.h"
#include "font.h"

class LabelControl : public GuiControl
{
public:
    virtual void Init() { m_uBackColor = COLOR_BLACK;
                          m_uFontColor = COLOR_WHITE;
                          m_pstFont = NULL;
                          m_pcCaption = "";
                          SetAcceptFocus(false); }
    virtual void Draw();
    virtual GuiReturn_t ProcessEvent( GuiEvent_t *
pstEvent_ ) {}
    virtual void Activate( bool bActivate_ ) {}

    void SetBackColor( COLOR eColor_ )          { m_uBackColor = eColor_; }
    void SetFontColor( COLOR eColor_ )          { m_uFontColor = eColor_; }
    void SetFont( Font_t *pstFont_ )          { m_pstFont = pstFont_; }
    void SetCaption( const K_CHAR *pcData_ )    { m_pcCaption = pcData_; }

private:
    Font_t *m_pstFont;
    const K_CHAR *m_pcCaption;
    COLOR m_uBackColor;
    COLOR m_uFontColor;

};

#endif
```

14.19 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_panel.cpp File Reference

GUI Panel Control Implementation.

```
#include "gui.h"
#include "kerneltypes.h"
#include "draw.h"
#include "graphics.h"
#include "control_panel.h"
```

14.19.1 Detailed Description

GUI Panel Control Implementation.

Definition in file control_panel.cpp.

14.20 control_panel.cpp

```
/*===========================================================================
     _____       _____       _____       _____
 ___|   _|__  __|_    |__ __|__   |__ __| __  |__  ______
|    \  /  | ||    \     ||     |     ||  |/ /     ||___   |
|     \/   | ||     \    ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "gui.h"
#include "kerneltypes.h"
#include "draw.h"
#include "graphics.h"
#include "control_panel.h"
```

```

//---------------------------------------------------------------------------
void PanelControl::Draw()
{
    GUI_DEBUG_PRINT( "PanelControl::Draw()\n");
    GraphicsDriver *pclDriver = GetParentWindow()->GetDriver();
    DrawRectangle_t stRectangle;
    K_USHORT usX, usY;

    GetControlOffset(&usX, &usY);

    stRectangle.usTop = GetTop() + usY;
    stRectangle.usBottom = stRectangle.usTop + GetHeight() -1;
    stRectangle.usLeft = GetLeft() + usX;
    stRectangle.usRight = stRectangle.usLeft + GetWidth() -1;
    stRectangle.bFill = true;
    stRectangle.uLineColor = m_uColor;
    stRectangle.uFillColor = m_uColor;

    pclDriver->Rectangle(&stRectangle);
}
```

14.21 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_panel.h File Reference

GUI Panel Control.

```
#include "gui.h"
#include "kerneltypes.h"
#include "draw.h"
```

Classes

- class PanelControl

14.21.1 Detailed Description

GUI Panel Control. The "panel" is probably the simplest control that can be implemented in a GUI. It serves as a dock for other controls, and also as an example for implementing more complex controls.

A panel is essentially a flat rectangle, specified by a control's typical top/left/height/width parameters, and a color value.

Definition in file control_panel.h.

14.22 control_panel.h

```
/*===========================================================================
     _____     _____     _____     _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  _____
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||_____|
    |_____|      |_____|     |_____|     |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#ifndef __CONTROL_PANEL_H__
#define __CONTROL_PANEL_H__

```

```
#include "gui.h"
#include "kerneltypes.h"
#include "draw.h"

class PanelControl : public GuiControl
{
public:
    virtual void Init() { m_uColor = COLOR_BLACK; SetAcceptFocus
    (false); }
    virtual void Draw();
    virtual GuiReturn_t ProcessEvent( GuiEvent_t *
pstEvent_ ) {};
    virtual void Activate( bool bActivate_ ) {}

    void SetColor( COLOR eColor_ ) { m_uColor = eColor_; }

private:
    COLOR m_uColor;

};

#endif

```

14.23 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_progress.cpp File Reference

GUI Progress Bar Control.

```
#include "gui.h"
#include "kerneltypes.h"
#include "draw.h"
#include "control_progress.h"
```

14.23.1 Detailed Description

GUI Progress Bar Control. A simple progress bar control using lines and rectangles to display the status of an operation from initialization to completion

Definition in file control_progress.cpp.

14.24 control_progress.cpp

```
/*===========================================================================
     _____     _____     _____     _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  _____
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||_____|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "gui.h"
#include "kerneltypes.h"
#include "draw.h"
#include "control_progress.h"

//---------------------------------------------------------------------------
void ProgressControl::Init()
{
    m_uBackColor = COLOR_BLACK;
    m_uBorderColor = COLOR_GREY75;
    m_uProgressColor = COLOR_GREEN;
    SetAcceptFocus(false);
}

```

```
//---------------------------------------------------------------------------
void ProgressControl::Draw()
{
    GraphicsDriver *pclDriver = GetParentWindow()
->GetDriver();
    DrawRectangle_t stRect;
    DrawLine_t stLine;

    K_USHORT usX, usY;
    K_USHORT usProgressWidth;

    GetControlOffset(&usX, &usY);

    // Draw the outside of the progress bar region
    stLine.uColor = m_uBorderColor;
    stLine.usX1 = usX + GetLeft() + 1;
    stLine.usX2 = usX + GetLeft() + GetWidth() - 2;
    stLine.usY1 = usY + GetTop();
    stLine.usY2 = usY + GetTop();
    pclDriver->Line(&stLine);

    stLine.usY1 = usY + GetTop() + GetHeight() - 1;
    stLine.usY2 = usY + GetTop() + GetHeight() - 1;
    pclDriver->Line(&stLine);

    stLine.usY1 = usY + GetTop() + 1;
    stLine.usY2 = usY + GetTop() + GetHeight() - 2;
    stLine.usX1 = usX + GetLeft();
    stLine.usX2 = usX + GetLeft();
    pclDriver->Line(&stLine);

    stLine.usX1 = usX + GetLeft() + GetWidth() - 1;
    stLine.usX2 = usX + GetLeft() + GetWidth() - 1;
    pclDriver->Line(&stLine);

    // Draw the "completed" portion
    usProgressWidth = (K_USHORT)( ( ( ( (K_ULONG)m_ucProgress) * (GetWidth
()-2) ) + 50 ) / 100);
    stRect.usTop = usY + GetTop() + 1;
    stRect.usBottom = usY + GetTop() + GetHeight() - 2;
    stRect.usLeft = usX + GetLeft() + 1;
    stRect.usRight = stRect.usLeft + usProgressWidth - 1;
    stRect.bFill = true;
    stRect.uLineColor = m_uProgressColor;
    stRect.uFillColor = m_uProgressColor;
    pclDriver->Rectangle(&stRect);

    // Draw the "incomplete" portion
    stRect.usLeft = stRect.usRight + 1;
    stRect.usRight = usX + GetLeft() + GetWidth() - 2;
    stRect.bFill = true;
    stRect.uLineColor = m_uBackColor;
    stRect.uFillColor = m_uBackColor;
    pclDriver->Rectangle(&stRect);

}

//---------------------------------------------------------------------------
void ProgressControl::SetProgress( K_UCHAR ucProgress_ )
{
    m_ucProgress = ucProgress_;
    if (m_ucProgress > 100)
    {
        m_ucProgress;
    }
    SetStale();
}

//---------------------------------------------------------------------------
GuiReturn_t ProgressControl::ProcessEvent(
GuiEvent_t *pstEvent_)
{
    return GUI_EVENT_OK;
}
```

14.25 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_progress.h File Reference

GUI Progress Bar Control.

```
#include "gui.h"
#include "kerneltypes.h"
#include "draw.h"
#include "font.h"
```

Classes

- class ProgressControl

14.25.1 Detailed Description

GUI Progress Bar Control. A simple progress bar control using lines and rectangles to display the status of an operation from initialization to completion

Definition in file control_progress.h.

14.26 control_progress.h

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \     ||     |     ||  |/ /     ||___   |
|     \/   | ||     \    ||      \    ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#ifndef __CONTROL_PROGRESS_H__
#define __CONTROL_PROGRESS_H__

#include "gui.h"
#include "kerneltypes.h"
#include "draw.h"
#include "font.h"

class ProgressControl : public GuiControl
{
public:
    virtual void Init();
    virtual void Draw();
    virtual GuiReturn_t ProcessEvent( GuiEvent_t *
pstEvent_ );
    virtual void Activate( bool bActivate_ ) {}

    void SetBackColor( COLOR eColor_ )  { m_uBackColor = eColor_; }
    void SetProgressColor( COLOR eColor_ )  { m_uProgressColor = eColor_; }
    void SetBorderColor( COLOR eColor_ )    { m_uBorderColor = eColor_; }

    void SetProgress( K_UCHAR ucProgress_ );

private:
    COLOR m_uBackColor;
    COLOR m_uProgressColor;
    COLOR m_uBorderColor;
    K_UCHAR m_ucProgress;
};

#endif

```

14.27 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/dcpu.cpp File Reference

Portable DCPU-16 CPU emulator.

```
#include "dcpu.h"
#include "kerneltypes.h"
#include "ll.h"
```

Macros

- #define **CORE_DEBUG** 0
- #define **DBG_PRINT**(...)

Variables

- static const K_UCHAR aucBasicOpcodeCycles [ ]

  Define the number of cycles that each "basic" opcode takes to execute.
- static const K_UCHAR aucExtendedOpcodeCycles [ ]

  Define the number of cycles that each "extended" opcode takes to execute.

14.27.1 Detailed Description

Portable DCPU-16 CPU emulator. The DCPU-16 is the in-game CPU used in the upcoming game 0x10^c, from the creators of the wildly successful Minecraft. While the DCPU is supposed to be part of the game, it has serious potential for use in all sorts of embedded applications.

The fact that DCPU is a very lightweight VM to implement and contains built-in instructions for accessing hardware peripheras and handling external interrupts lends itself to being used on microcontrollers.

Unlike a lot of embedded CPUs, DCPU-16 assembly is extremely simple to learn, since it has a very limited number of opcodes (37), each of which provide the same register/memory addressing modes for all operands. There are also only 2 opcode formats which make interpreting opcodes very efficient.

The DCPU-16 is extended using a variable number of "external hardware devices" which communicate with the CPU core using interrupts. These devices are enumerated on startup, and since there is no defined format for how these devices work, we can hijack this interface to provide a way for the DCPU to access resources supplied by the OS (i.e Timers, Drivers), or the hardware directly. This also lends itself to inter-VM communications (multiple DCPUs communicating with eachother in different OS threads). There's an immense amount of flexibility here - applications from debugging to scripting to runtime-configuration are all easily supported by this machine.

But what is a platform without tools support? Fortunately, the hype around 0x10c is building - and a development community for this platform has grown immensely. There are a number of compilers, assemblers, and IDEs, many of which support virtualized hardware extensions. One of the compilers is a CLANG/LLVM backend, which should allow for very good C language support.

I had attempted to do something similar by creating a VM based on the 8051 (see the Funk51 project on source-forge), but that project was at least four times as large - and the tools support was very spotty. There were C compilers, but there was a lot of shimming required to produce output that was suitable for the VM. Also, the lack of a native host interface (interrupts, hardware bus enumerations, etc.) forced a non-standard approach to triggering native methods by writing commands to a reserved chunk of memory and writing to a special "trigger" address to invoke the native system. Using a DCPU-16 based simulator addresses this in a nice, clean way by providing modern tools, and a VM infrastruture tailored to be interfaced with a host.

Regarding this version of the DCPU emulator - it's very simple to use. Program binaries are loaded into buffers in the host CPU's RAM, with the host also providing a separate buffer for DCPU RAM. The size of the DCPU RAM buffer will contain both the RAM area, as well as the program stack, so care must be taken to ensure that the stack doesn't overflow. The DCPU specification allows for 64K words (128KB) of RAM and ROM each, but this implementation allows us to tailor the CPU for more efficient or minimal environments.

In the future, this emulator will be extended to provide a mechanism to allow programs to be run out of flash, EEPROM, or other interfaces via the Mark3 Drivers API.

Once the program has been loaded into the host's address space, the DCPU class can be initialized.

```
// Use 16-bit words for 16-bit emulator.
K_USHORT ausRAM[ RAM_SIZE ];
K_USHORT ausROM[ ROM_SIZE ];
{
    class DCPU clMyDCPU;

    // Read program code into ausROM buffer here

    // Initialize the DCPU emulator
    clMyDCPU.Init( ausROM, RAM_SIZE, ausROM, ROM_SIZE );
}
```

Once the emulator has been initialized, the VM can be run one opcode at a time, as in the following example.

```
while(1)
{
    clMyCPU.RunOpcode();
}
```

To inspect the contents of the VM's registers, call the GetRegisters() method. This is useful for printing the CPU state on a regular basis, or using the PC value to determine when to end execution, or to provide an offset for disassembling the current opcode.

```
DCPU_Registers *pstRegisters;
pstRegisters = clMyCPU.GetRegisters();
```

ToDo: Implement the hardware bus interface - provide functions to register devices/drivers/etc. with the VM prior to initialization. Hook into the interrupt system to provide communication from the VM back to the host processor.

Definition in file dcpu.cpp.

14.28 dcpu.cpp

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \    ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\ __||_______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "dcpu.h"
#include "kerneltypes.h"
#include "ll.h"

#define CORE_DEBUG 0

//---------------------------------------------------------------------------
#if CORE_DEBUG
  #define DBG_PRINT(...)        printf(__VA_ARGS__)
#else
  #define DBG_PRINT(...)
#endif

//---------------------------------------------------------------------------
static const K_UCHAR aucBasicOpcodeCycles[] =
{
    0,      // OP_NON_BASIC = 0
    1,      // OP_SET
    2,      // OP_ADD
    2,      // OP_SUB
    2,      // OP_MUL
    2,      // OP_MLI
    3,      // OP_DIV
    3,      // OP_DVI,
    3,      // OP_MOD,
    3,      // OP_MDI,
    1,      // OP_AND,
    1,      // OP_BOR,
    1,      // OP_XOR,
    1,      // OP_SHR,
    1,      // OP_ASR,
    1,      // OP_SHL,
    2,      // OP_IFB,
    2,      // OP_IFC,
    2,      // OP_IFE,
    2,      // OP_IFN,
    2,      // OP_IFG,
    2,      // OP_IFA,
    2,      // OP_IFL,
    2,      // OP_IFU,
    0,      // OP_18,
    0,      // OP_19,
    3,      // OP_ADX,
    3,      // OP_SBX,
    0,      // OP_1C,
    0,      // OP_1D,
    2,      // OP_STI,
    2,      // OP_STD
};

//---------------------------------------------------------------------------
static const K_UCHAR aucExtendedOpcodeCycles[] =
{
    0,  // "RESERVED",
    3,  // "JSR",
    0,  // "UNDEFINED"
    0,  // "UNDEFINED"
    0,  // "UNDEFINED"
    0,  // "UNDEFINED"
    0,  // "UNDEFINED"
    0,  // "UNDEFINED"
    4,  // "INT",
    1,  // "IAG",
    1,  // "IAS",
    3,  // "RFI",
    2,  // "IAQ",
    0,  // "UNDEFINED"
    0,  // "UNDEFINED"
    0,  // "UNDEFINED"
    2,  // "HWN",
    4,  // "HWQ",
    4,  // "HWI",
    0,  // "UNDEFINED"
    0,  // "UNDEFINED"
    0,  // "UNDEFINED"
    0,  // "UNDEFINED"
    0,  // "UNDEFINED"
    0,  // "UNDEFINED"
    0,  // "UNDEFINED"
    0,  // "UNDEFINED"
    0,  // "UNDEFINED"
    0,  // "UNDEFINED"
    0,  // "UNDEFINED"
    0,  // "UNDEFINED"
    0,  // "UNDEFINED"
};

//---------------------------------------------------------------------------
void DCPU::SET()
{
    DBG_PRINT("SET\n");
    *b = *a;
}

//---------------------------------------------------------------------------
void DCPU::ADD()
{
    K_ULONG ulTemp;
    DBG_PRINT("ADD\n");

    ulTemp = (K_ULONG)*a + (K_ULONG)*b;
    if (ulTemp >= 65536)
    {
        m_stRegisters.EX = 0x0001;
    }
    else
    {
        m_stRegisters.EX = 0;
    }

    *b = *b + *a;
}

//---------------------------------------------------------------------------
void DCPU::SUB()
{
    K_LONG lTemp;
    DBG_PRINT("SUB\n");

    lTemp = (K_LONG)*b - (K_LONG)*a;
    if (lTemp < 0)
    {
        m_stRegisters.EX = 0xFFFF;
    }
    else
    {
        m_stRegisters.EX = 0;
    }

    *b = *b - *a;
}

//---------------------------------------------------------------------------
void DCPU::MUL()
{
    K_ULONG ulTemp;

    DBG_PRINT("MUL\n");
    ulTemp = (((K_ULONG)*a * (K_ULONG)*b));
    m_stRegisters.EX = (K_USHORT)(ulTemp >> 16);
    *b = (K_USHORT)(ulTemp & 0x0000FFFF);
}

//---------------------------------------------------------------------------
void DCPU::MLI()
{
    K_LONG lTemp;

    DBG_PRINT("MLI\n");
    lTemp = ((K_LONG)(*(K_SHORT*)a) * (K_LONG)(*(K_SHORT*)b));
    m_stRegisters.EX = (K_USHORT)(lTemp >> 16);
    *b = (K_USHORT)(lTemp & 0x0000FFFF);
}

//---------------------------------------------------------------------------
void DCPU::DIV()
{
    K_USHORT usTemp;

    DBG_PRINT("DIV\n");
    if (*a == 0)
    {
        *b = 0;
        m_stRegisters.EX = 0;
    }
    else
    {
        usTemp = (K_USHORT)((((K_ULONG)*b) << 16) / (K_ULONG)*a);
        *b = *b / *a;
        m_stRegisters.EX = usTemp;
    }
}

//---------------------------------------------------------------------------
void DCPU::DVI()
{
    K_USHORT usTemp;

    DBG_PRINT("DVI\n");
    if (*a == 0)
    {
        *b = 0;
        m_stRegisters.EX = 0;
    }
    else
    {
        usTemp = (K_USHORT)((((K_LONG)*((K_SHORT*)b)) << 16) / (K_LONG)(*(
K_SHORT*)a));
        *b = (K_USHORT)(*(K_SHORT*)b / *(K_SHORT*)a);
        m_stRegisters.EX = usTemp;

    }
}
```

```

//---------------------------------------------------------------------------
void DCPU::MOD()
{
    DBG_PRINT("MOD\n");
    if (*a == 0)
    {
        *b = 0;
    }
    else
    {
        *b = *b % *a;
    }
}

//---------------------------------------------------------------------------
void DCPU::MDI()
{
    DBG_PRINT("MDI\n");
    if (*b == 0)
    {
        *a = 0;
    }
    else
    {
        *b = (K_USHORT)(*((K_SHORT*)b) % *((K_SHORT*)a));
    }
}

//---------------------------------------------------------------------------
void DCPU::AND()
{
    DBG_PRINT("AND\n");
    *b = *b & *a;
}

//---------------------------------------------------------------------------
void DCPU::BOR()
{
    DBG_PRINT("BOR\n");
    *b = *b | *a;
}

//---------------------------------------------------------------------------
void DCPU::XOR()
{
    DBG_PRINT("XOR\n");
    *b = *b ^ *a;
}

//---------------------------------------------------------------------------
void DCPU::SHR()
{
    K_USHORT usTemp = (K_USHORT)((((K_ULONG)*b) << 16) >> (K_ULONG)*a);

    DBG_PRINT("SHR\n");
    *b = *b >> *a;
    m_stRegisters.EX = usTemp;
}

//---------------------------------------------------------------------------
void DCPU::ASR()
{
    K_USHORT usTemp = (K_USHORT)((((K_LONG)*b) << 16) >> (K_LONG)*a);

    DBG_PRINT("ASR\n");
    *b = (K_USHORT)(*(K_SHORT*)b >> *(K_SHORT*)a);
    m_stRegisters.EX = usTemp;
}
//---------------------------------------------------------------------------
void DCPU::SHL()
{
    K_USHORT usTemp = (K_USHORT)((((K_ULONG)*b) << (K_ULONG)*a) >> 16);

    DBG_PRINT("SHL\n");
    *b = *b << *a;
    m_stRegisters.EX = usTemp;
}

//---------------------------------------------------------------------------
bool DCPU::IFB()
{
    DBG_PRINT("IFB\n");
    if ((*b & *a) != 0)
    {
        return true;
    }
    return false;
}

//---------------------------------------------------------------------------
bool DCPU::IFC()
{
    DBG_PRINT("IFC\n");
    if ((*b & *a) == 0)
    {
        return true;
    }
    return false;
}

//---------------------------------------------------------------------------
bool DCPU::IFE()
{
    DBG_PRINT("IFE\n");
    if (*b == *a)
    {
        return true;
    }
    return false;
}

//---------------------------------------------------------------------------
bool DCPU::IFN()
{
    DBG_PRINT("IFN\n");
    if (*b != *a)
    {
        return true;
    }
    return false;
}

//---------------------------------------------------------------------------
bool DCPU::IFG()
{
    DBG_PRINT("IFG\n");
    if (*b > *a)
    {
        return true;
    }
    return false;
}

//---------------------------------------------------------------------------
bool DCPU::IFA()
{
    DBG_PRINT("IFA\n");
    if (*((K_SHORT*)b) > *((K_SHORT*)a))
    {
        return true;
    }
    return false;
}

//---------------------------------------------------------------------------
bool DCPU::IFL()
{
    DBG_PRINT("IFL\n");
    if (*b < *a)
    {
        return true;
    }
    return false;
}

//---------------------------------------------------------------------------
bool DCPU::IFU()
{
    DBG_PRINT("IFU\n");
    if (*(K_SHORT*)b < *(K_SHORT*)a)
    {
        return true;
    }
    return false;
}

//---------------------------------------------------------------------------
void DCPU::ADX()
{
    K_ULONG ulTemp;
    DBG_PRINT("ADX\n");
    ulTemp = (K_ULONG)*b + (K_ULONG)*a + (K_ULONG)m_stRegisters.
EX;
```

```
    if (ulTemp >= 0x10000)
    {
        m_stRegisters.EX = 1;
    }
    else
    {
        m_stRegisters.EX = 0;
    }

    *b = ((K_USHORT)(ulTemp & 0x0000FFFF));
}

//---------------------------------------------------------------------------
void DCPU::SBX()
{
    K_LONG lTemp;
    DBG_PRINT("SBX\n");
    lTemp = (K_LONG)*b - (K_LONG)*a + (K_LONG)m_stRegisters.EX;
    if (lTemp < 0 )
    {
        m_stRegisters.EX = 0xFFFF;
    }
    else
    {
        m_stRegisters.EX = 0;
    }

    *b = ((K_USHORT)(lTemp & 0x0000FFFF));
}

//---------------------------------------------------------------------------
void DCPU::STI()
{
    DBG_PRINT("STI\n");
    *b = *a;
    m_stRegisters.I++;
    m_stRegisters.J++;
}

//---------------------------------------------------------------------------
void DCPU::STD()
{
    DBG_PRINT("STD\n");
    *b = *a;
    m_stRegisters.I--;
    m_stRegisters.J--;
}

//---------------------------------------------------------------------------
void DCPU::JSR()
{
    DBG_PRINT("JSR\n");
    m_pusRAM[ --m_stRegisters.SP ] = m_stRegisters
.PC;
    m_stRegisters.PC = *b;
}

//---------------------------------------------------------------------------
void DCPU::INT()
{
    DBG_PRINT("INT\n");

    if (m_stRegisters.IA == 0)
    {
        // If IA is not set, return out.
        return;
    }

    // Either acknowledge the interrupt immediately, or queue it.
    if (m_bInterruptQueueing == false)
    {
        m_pusRAM[ --m_stRegisters.SP ] = m_stRegisters
.PC;
        m_pusRAM[ --m_stRegisters.SP ] = m_stRegisters
.A;

        m_stRegisters.A = *a;
        m_stRegisters.PC = m_stRegisters.IA;
        m_bInterruptQueueing = true;
    }
    else
    {
        // Add interrupt message to the queue
        m_ausInterruptQueue[ ++m_ucQueueLevel
] = *a;
    }
}
```

```

//---------------------------------------------------------------------------
void DCPU::ProcessInterruptQueue()
{
    // If there's an interrupt address specified, queueing is disabled, and
    // the queue isn't empty
    if (m_stRegisters.IA && !m_bInterruptQueueing
 && m_ucQueueLevel)
    {
        m_pusRAM[ --m_stRegisters.SP ] = m_stRegisters
.PC;
        m_pusRAM[ --m_stRegisters.SP ] = m_stRegisters
.A;

        m_stRegisters.A = m_ausInterruptQueue[
m_ucQueueLevel-- ];
        m_stRegisters.PC = m_stRegisters.IA;

        m_bInterruptQueueing = true;
    }
}


//---------------------------------------------------------------------------
void DCPU::IAG()
{
    DBG_PRINT("IAG\n");

    *a = m_stRegisters.IA;
}

//---------------------------------------------------------------------------
void DCPU::IAS()
{
    DBG_PRINT("IAS\n");

    m_stRegisters.IA = *a;
}

//---------------------------------------------------------------------------
void DCPU::RFI()
{
    DBG_PRINT("RFI\n");

    m_bInterruptQueueing = false;

    m_stRegisters.A  = m_pusRAM[ m_stRegisters
.SP++ ];
    m_stRegisters.PC = m_pusRAM[ m_stRegisters
.SP++ ];

}

//---------------------------------------------------------------------------
void DCPU::IAQ()
{
    DBG_PRINT("IAQ\n");

    if (*a)
    {
        m_bInterruptQueueing = true;
    }
    else
    {
        m_bInterruptQueueing = false;
    }
}

//---------------------------------------------------------------------------
void DCPU::HWN()
{
    LinkListNode *pclNode;

    DBG_PRINT("HWN\n");
    m_usTempA = 0;
    pclNode = m_clPluginList.GetHead();
    while (pclNode)
    {
        m_usTempA++;
        pclNode = pclNode->GetNext();
    }

    *a = m_usTempA;
}

//---------------------------------------------------------------------------
void DCPU::HWQ()
```

```
{
    DBG_PRINT("HWQ\n");
    DCPUPlugin *pclPlugin;
    pclPlugin = (DCPUPlugin*)m_clPluginList.GetHead();

    while (pclPlugin)
    {
        if (pclPlugin->GetDeviceNumber() == *a)
        {
            pclPlugin->Enumerate(&m_stRegisters);
            break;
        }
        pclPlugin = (DCPUPlugin*)pclPlugin->GetNext();
    }
}

//---------------------------------------------------------------------------
void DCPU::HWI()
{
    DBG_PRINT("HWI\n");

    DCPUPlugin *pclPlugin;
    pclPlugin = (DCPUPlugin*)m_clPluginList.GetHead();

    while (pclPlugin)
    {
        if (pclPlugin->GetDeviceNumber() == *a)
        {
            pclPlugin->Interrupt(this);
            break;
        }
        pclPlugin = (DCPUPlugin*)pclPlugin->GetNext();
    }
}

//---------------------------------------------------------------------------
void DCPU::Init(  K_USHORT *pusRAM_, K_USHORT usRAMSize_,
                    const K_USHORT *pusROM_, K_USHORT usROMSize_ )
{
    m_stRegisters.PC = 0;
    m_stRegisters.SP = usRAMSize_ ;
    m_stRegisters.A = 0;
    m_stRegisters.B = 0;
    m_stRegisters.C = 0;
    m_stRegisters.X = 0;
    m_stRegisters.Y = 0;
    m_stRegisters.Z = 0;
    m_stRegisters.I = 0;
    m_stRegisters.J = 0;
    m_stRegisters.EX = 0;
    m_stRegisters.IA = 0;
    m_ulCycleCount = 0;

    m_pusROM = (K_USHORT*)pusROM_;
    m_usROMSize = usROMSize_;

    m_pusRAM = pusRAM_;
    m_usRAMSize = usRAMSize_;
}

//---------------------------------------------------------------------------
K_UCHAR DCPU::GetOperand( K_UCHAR ucOpType_, K_USHORT **pusResult_ )
{
    K_UCHAR ucRetVal = 0;
    switch (ucOpType_)
    {
        case ARG_A: case ARG_B: case ARG_C: case ARG_X:
        case ARG_Y: case ARG_Z: case ARG_I: case ARG_J:
            *pusResult_ = &m_stRegisters.ausRegisters[ ucOpType_ - ARG_A ];
            break;

        case ARG_BRACKET_A: case ARG_BRACKET_B: case ARG_BRACKET_C: case ARG_BRACKET_X:
        case ARG_BRACKET_Y: case ARG_BRACKET_Z: case ARG_BRACKET_I: case ARG_BRACKET_J:
            *pusResult_ = &m_pusRAM[ m_stRegisters.ausRegisters[ ucOpType_ - ARG_BRACKET_A ] ];
            break;

        case ARG_WORD_A: case ARG_WORD_B: case ARG_WORD_C: case ARG_WORD_X:
        case ARG_WORD_Y: case ARG_WORD_Z: case ARG_WORD_I: case ARG_WORD_J:
        {
            K_USHORT usTemp = m_pusROM[ m_stRegisters.PC++
 ];
            usTemp += m_stRegisters.ausRegisters[ ucOpType_ -
ARG_WORD_A ];
            *pusResult_ = &m_pusRAM[ usTemp ];
            ucRetVal = 1;
        }
            break;
        case ARG_PUSH_POP_SP:
            if (*pusResult_ == a)
            {
                a = &m_pusRAM[ m_stRegisters.SP++ ];
            }
            else
            {
                b = &m_pusRAM[ --m_stRegisters.SP ];
            }
            break;
        case ARG_PEEK_SP:
            *pusResult_ = &m_pusRAM[ m_stRegisters.SP ];
            break;
        case ARG_WORD_SP:
        {
            K_USHORT usTemp = m_pusROM[ m_stRegisters.PC++
 ];
            usTemp += m_stRegisters.SP;
            *pusResult_ = &m_pusRAM[ usTemp ];
            ucRetVal++;
        }
            break;
        case ARG_SP:
            *pusResult_ = &(m_stRegisters.SP);
            break;
        case ARG_PC:
            *pusResult_ = &(m_stRegisters.PC);
            break;
        case ARG_EX:
            *pusResult_ = &(m_stRegisters.EX);
            break;
        case ARG_NEXT_WORD:
            *pusResult_ = &m_pusRAM[ m_pusROM[ m_stRegisters
.PC++ ] ];
            ucRetVal++;
            break;
        case ARG_NEXT_LITERAL:
            *pusResult_ = &m_pusROM[ m_stRegisters.PC++ ];
            ucRetVal++;
            break;

        case ARG_LITERAL_0:
            *pusResult_ = &m_usTempA;
            m_usTempA = (K_USHORT)(-1);
            break;
        case ARG_LITERAL_1: case ARG_LITERAL_2: case ARG_LITERAL_3: case
ARG_LITERAL_4:
        case ARG_LITERAL_5: case ARG_LITERAL_6: case ARG_LITERAL_7: case
ARG_LITERAL_8:
        case ARG_LITERAL_9: case ARG_LITERAL_A: case ARG_LITERAL_B: case
ARG_LITERAL_C:
        case ARG_LITERAL_D: case ARG_LITERAL_E: case ARG_LITERAL_F: case
ARG_LITERAL_10:
        case ARG_LITERAL_11: case ARG_LITERAL_12: case ARG_LITERAL_13: case
ARG_LITERAL_14:
        case ARG_LITERAL_15: case ARG_LITERAL_16: case ARG_LITERAL_17: case
ARG_LITERAL_18:
        case ARG_LITERAL_19: case ARG_LITERAL_1A: case ARG_LITERAL_1B: case
ARG_LITERAL_1C:
        case ARG_LITERAL_1D: case ARG_LITERAL_1E: case ARG_LITERAL_1F:
            *pusResult_ = &m_usTempA;
            m_usTempA = (K_USHORT)(ucOpType_ - ARG_LITERAL_1);
            break;
        default:
            break;
    }
    return ucRetVal;
}

//---------------------------------------------------------------------------
void DCPU::RunOpcode()
{
    // Fetch the opcode @ the current program counter
    K_USHORT usWord = m_pusROM[ m_stRegisters.PC++ ];
    K_UCHAR ucOp = (K_UCHAR)DCPU_NORMAL_OPCODE_MASK(
usWord);
    K_UCHAR ucA = (K_UCHAR)DCPU_A_MASK(usWord);
    K_UCHAR ucB = (K_UCHAR)DCPU_B_MASK(usWord);
    K_UCHAR ucSize = 1;

    // Decode the opcode
    if (ucOp)
    {
        bool bRunNext = true;

        a = &m_usTempA;
        b = 0;

        // If this is a "basic" opcode, decode "a" and "b"
        ucSize += GetOperand( ucA , &a );
        ucSize += GetOperand( ucB, &b );

        // Add the cycles to the runtime clock
        m_ulCycleCount += (K_ULONG)aucBasicOpcodeCycles
[ ucOp ];
        m_ulCycleCount += (ucSize - 1);

        // Execute the instruction once we've decoded the opcode and
        // processed the arguments.
        switch (DCPU_NORMAL_OPCODE_MASK(usWord))
        {
            case OP_SET:  SET();      break;
            case OP_ADD:  ADD();      break;
            case OP_SUB:  SUB();      break;
            case OP_MUL:  MUL();      break;
            case OP_MLI:  MLI();      break;
            case OP_DIV:  DIV();      break;
            case OP_DVI:  DVI();      break;
            case OP_MOD:  MOD();      break;
            case OP_MDI:  MDI();      break;
            case OP_AND:  AND();      break;
            case OP_BOR:  BOR();      break;
            case OP_XOR:  XOR();      break;
            case OP_SHR:  SHR();      break;
            case OP_ASR:  ASR();      break;
            case OP_SHL:  SHL();      break;
            case OP_IFB:  bRunNext = IFB();   break;
            case OP_IFC:  bRunNext = IFC();   break;
            case OP_IFE:  bRunNext = IFE();   break;
            case OP_IFN:  bRunNext = IFN();   break;
            case OP_IFG:  bRunNext = IFG();   break;
            case OP_IFA:  bRunNext = IFA();   break;
            case OP_IFL:  bRunNext = IFL();   break;
            case OP_IFU:  bRunNext = IFU();   break;
            case OP_ADX:  ADX();      break;
            case OP_SBX:  SBX();      break;
            case OP_STI:  STI();      break;
            case OP_STD:  STD();      break;
            default:    break;
        }

        // If we're not supposed to run the next instruction (i.e. skip it
        // due to failed condition), adjust the PC.
        if (!bRunNext)
        {
            // Skipped branches take an extra cycle
            m_ulCycleCount++;

            // Skip the next opcode
            usWord = m_pusROM[ m_stRegisters.PC++ ];
            if (DCPU_NORMAL_OPCODE_MASK(usWord))
            {
                DBG_PRINT( "Skipping Basic Opcode: %X\n",
DCPU_NORMAL_OPCODE_MASK(usWord));
                // If this is a "basic" opcode, decode "a" and "b" - we do this
 to make sure our
                // PC gets adjusted properly.
                GetOperand( DCPU_A_MASK(usWord), &a );
                GetOperand( DCPU_B_MASK(usWord), &b );
            }
            else
            {
                DBG_PRINT( "Skipping Extended Opcode: %X\n",
DCPU_EXTENDED_OPCODE_MASK(usWord));
                GetOperand( DCPU_A_MASK(usWord), &a );
            }
        }
    }
    else
    {
        // Extended opcode.  These only have a single argument, stored in the
        // "a" field.
        GetOperand( ucA, &a );
        m_ulCycleCount++;

        // Execute the "extended" instruction now that the opcode has been
        // decoded, and the arguments processed.
        switch (ucB)
        {
            case OP_EX_JSR:    JSR();  break;
            case OP_EX_INT:        INT();  break;
            case OP_EX_IAG:        IAG();  break;
            case OP_EX_IAS:        IAS();  break;
            case OP_EX_RFI:        RFI();  break;
            case OP_EX_IAQ:        IAQ();   break;
            case OP_EX_HWN:        HWN();   break;
            case OP_EX_HWQ:        HWQ();  break;
            case OP_EX_HWI:        HWI();  break;
            default:    break;
        }
    }

    // Process an interrupt from the queue (if there is one)
    ProcessInterruptQueue();
}

//---------------------------------------------------------------------------
void DCPU::SendInterrupt( K_USHORT usMessage_ )
{
    if (m_stRegisters.IA == 0)
    {
        // If IA is not set, return out.
        return;
    }

    // Either acknowledge the interrupt immediately, or queue it.
    if (m_bInterruptQueueing == false)
    {
        m_pusRAM[ --m_stRegisters.SP ] = m_stRegisters
.PC;
        m_pusRAM[ --m_stRegisters.SP ] = m_stRegisters
.A;

        m_stRegisters.A = usMessage_;
        m_stRegisters.PC = m_stRegisters.IA;
        m_bInterruptQueueing = true;
    }
    else
    {
        // Add interrupt message to the queue
        m_ausInterruptQueue[ ++m_ucQueueLevel
] = usMessage_;
    }
}

//---------------------------------------------------------------------------
void DCPU::AddPlugin( DCPUPlugin *pclPlugin_ )
{
    m_clPluginList.Add( (LinkListNode*)pclPlugin_
);
}
```

14.29 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/dcpu.h File Reference

DCPU-16 emulator.

```
#include "kerneltypes.h"
#include "ll.h"
```

Classes

- struct DCPU_Registers

  Structure defining the DCPU hardware registers.
- class DCPUPlugin

  Class used to provide the hardware device abstraction between the DCPU-16 emulator/VM and the host system.
- class DCPU

  DCPU emulator, used for running code out of EEPROM, RAM, or other memory interfaces than FLASH.

Macros

- #define DCPU_NORMAL_OPCODE_MASK(x) ((K_USHORT)(x & 0x001F))

  DCPU v1.7 CPU emulator.
- #define **DCPU_EXTENDED_OPCODE_MASK**(x) ((K_USHORT)((x >> 5) & 0x001F))
- #define **DCPU_A_MASK**(x) ((K_USHORT)((x >> 10) & 0x003F))
- #define **DCPU_B_MASK**(x) ((K_USHORT)((x >> 5) & 0x001F))
- #define **DCPU_BUILD_NORMAL**(x, y, z) (((K_USHORT)(x) & 0x001F) | ((K_USHORT)(y) & 0x001F) << 5 | ((K_USHORT)(z) & 0x003F) << 10)
- #define **DCPU_BUILD_EXTENDED**(x, y) (((K_USHORT)(x & 0x001F) << 5) | ((K_USHORT)(y & 0x003F) << 10))

Typedefs

- typedef void(∗ DCPU_Callback)(DCPU ∗pclVM_)

  Callback function type used to implement HWI for VM->Host communications.

Enumerations

- enum DCPU_OpBasic {
  OP_NON_BASIC = 0, OP_SET, OP_ADD, OP_SUB,
  OP_MUL, OP_MLI, OP_DIV, OP_DVI,
  OP_MOD, OP_MDI, OP_AND, OP_BOR,
  OP_XOR, OP_SHR, OP_ASR, OP_SHL,
  OP_IFB, OP_IFC, OP_IFE, OP_IFN,
  OP_IFG, OP_IFA, OP_IFL, OP_IFU,
  OP_18, OP_19, OP_ADX, OP_SBX,
  OP_1C, OP_1D, OP_STI, OP_STD }

  DCPU Basic Opcodes.
- enum DCPU_OpExtended {
  OP_EX_RESERVED = 0, OP_EX_JSR, OP_EX_2, OP_EX_3,
  OP_EX_4, OP_EX_5, OP_EX_6, OP_EX_7,
  OP_EX_INT, OP_EX_IAG, OP_EX_IAS, OP_EX_RFI,
  OP_EX_IAQ, OP_EX_D, OP_EX_E, OP_EX_F,
  OP_EX_HWN, OP_EX_HWQ, OP_EX_HWI, OP_EX_13,
  OP_EX_14, OP_EX_15, OP_EX_16, OP_EX_17,
  OP_EX_18, OP_EX_19, OP_EX_1A, OP_EX_1B,
  OP_EX_1C, OP_EX_1D, OP_EX_1E, OP_EX_1F }

  DCPU Extended opcodes.
- enum DCPU_Argument {
  ARG_A = 0, **ARG_B**, **ARG_C**, **ARG_X**,
  ARG_Y, **ARG_Z**, **ARG_I**, **ARG_J**,
  ARG_BRACKET_A, **ARG_BRACKET_B**, **ARG_BRACKET_C**, **ARG_BRACKET_X**,
  ARG_BRACKET_Y, **ARG_BRACKET_Z**, **ARG_BRACKET_I**, **ARG_BRACKET_J**,
  ARG_WORD_A, **ARG_WORD_B**, **ARG_WORD_C**, **ARG_WORD_X**,
  ARG_WORD_Y, **ARG_WORD_Z**, **ARG_WORD_I**, **ARG_WORD_J**,
  ARG_PUSH_POP_SP, **ARG_PEEK_SP**, **ARG_WORD_SP**, **ARG_SP**,
  ARG_PC, **ARG_EX**, **ARG_NEXT_WORD**, **ARG_NEXT_LITERAL**,
  ARG_LITERAL_0, **ARG_LITERAL_1**, **ARG_LITERAL_2**, **ARG_LITERAL_3**,
  ARG_LITERAL_4, **ARG_LITERAL_5**, **ARG_LITERAL_6**, **ARG_LITERAL_7**,
  ARG_LITERAL_8, **ARG_LITERAL_9**, **ARG_LITERAL_A**, **ARG_LITERAL_B**,
  ARG_LITERAL_C, **ARG_LITERAL_D**, **ARG_LITERAL_E**, **ARG_LITERAL_F**,
  ARG_LITERAL_10, **ARG_LITERAL_11**, **ARG_LITERAL_12**, **ARG_LITERAL_13**,
  ARG_LITERAL_14, **ARG_LITERAL_15**, **ARG_LITERAL_16**, **ARG_LITERAL_17**,
  ARG_LITERAL_18, **ARG_LITERAL_19**, **ARG_LITERAL_1A**, **ARG_LITERAL_1B**,
  ARG_LITERAL_1C, **ARG_LITERAL_1D**, **ARG_LITERAL_1E**, **ARG_LITERAL_1F** }

Argument formats.

14.29.1 Detailed Description

DCPU-16 emulator.

Definition in file dcpu.h.

14.29.2 Macro Definition Documentation

14.29.2.1 #define DCPU_NORMAL_OPCODE_MASK(*x*) ((K_USHORT)(x & 0x001F))

DCPU v1.7 CPU emulator.

Basic opcode format: [aaaaaabbbbbooooo]

Where: - aaaaaa 6-bit source argument

- bbbbb 5-bit destination argument
- o is the opcode itself in a

If oooo = 0, then it's an "extended" opcode

Extended opcode format: [aaaaaaoooooxxxxx]

Where:

- xxxxx = all 0's - (basic opcode)
- ooooo = an extended opcode
- aaaaaa = the argument

Definition at line 48 of file dcpu.h.

14.29.3 Enumeration Type Documentation

14.29.3.1 enum DCPU_OpBasic

DCPU Basic Opcodes.

Enumerator:

OP_NON_BASIC special instruction - see below

OP_SET b, a | sets b to a

OP_ADD b, a | sets b to b+a, sets EX to 0x0001 if there's an overflow, 0x0 otherwise

OP_SUB b, a | sets b to b-a, sets EX to 0xffff if there's an underflow, 0x0 otherwise

OP_MUL b, a | sets b to b∗a, sets EX to ((b∗a)>>16)&0xffff (treats b, a as unsigned)

OP_MLI b, a | like MUL, but treat b, a as signed

OP_DIV b, a | sets b to b/a, sets EX to ((b<<16)/a)&0xffff. if a==0, sets b and EX to 0 instead. (treats b, a as unsigned)

OP_DVI b, a | like DIV, but treat b, a as signed. Rounds towards 0

OP_MOD b, a | sets b to ba. if a==0, sets b to 0 instead.

OP_MDI b, a | like MOD, but treat b, a as signed. (MDI -7, 16 == -7)

OP_AND b, a | sets b to b&a

OP_BOR b, a | sets b to b|a

OP_XOR b, a | sets b to b^a

OP_SHR b, a | sets b to b>>>a, sets EX to ((b<<16)>>a)&0xffff (logical shift)

OP_ASR b, a | sets b to b>>a, sets EX to ((b<<16)>>>a)&0xffff (arithmetic shift) (treats b as signed)

OP_SHL b, a | sets b to b<<a, sets EX to ((b<<a)>>16)&0xffff

OP_IFB b, a | performs next instruction only if (b&a)!=0

OP_IFC b, a | performs next instruction only if (b&a)==0

OP_IFE b, a | performs next instruction only if b==a

OP_IFN b, a | performs next instruction only if b!=a

OP_IFG b, a | performs next instruction only if b>a

OP_IFA b, a | performs next instruction only if b>a (signed)

OP_IFL b, a | performs next instruction only if b<a

OP_IFU b, a | performs next instruction only if b<a (signed)

OP_18 **UNDEFINED**

OP_19 **UNDEFINED**

OP_ADX b, a | sets b to b+a+EX, sets EX to 0x0001 if there is an over-flow, 0x0 otherwise

OP_SBX b, a | sets b to b-a+EX, sets EX to 0xFFFF if there is an under-flow, 0x0 otherwise

OP_1C **UNDEFINED**

OP_1D **UNDEFINED**

OP_STI b, a | sets b to a, then increases I and J by 1

OP_STD b, a | sets b to a, then decreases I and J by 1

Definition at line 99 of file dcpu.h.

14.29.3.2 enum DCPU_OpExtended

DCPU Extended opcodes.

Enumerator:

OP_EX_JSR a - pushes the address of the next instruction to the stack, then sets PC to a

OP_EX_2 **UNDEFINED**

OP_EX_3 **UNDEFINED**

OP_EX_4 **UNDEFINED**

OP_EX_5 **UNDEFINED**

OP_EX_6 **UNDEFINED**

OP_EX_7 **UNDEFINED**

OP_EX_INT Invoke software interrupt "a".

OP_EX_IAG Get interrupt address in "a".

OP_EX_IAS Set interrupt address from "a".

OP_EX_RFI Disables interrupt queueing, pops A from the stack, then pops PC from the stack.

OP_EX_IAQ if a is nonzero, interrupts will be added to the queue instead of triggered. if a is zero, interrupts will be triggered as normal again

OP_EX_D **UNDEFINED**

OP_EX_E **UNDEFINED**

OP_EX_F **UNDEFINED**

OP_EX_HWN Sets "a" to number of connected HW devices.

OP_EX_HWQ Set registers with information about hardware at index "a".

OP_EX_HWI Send an interrupt to hardware interface "a".

OP_EX_13 **UNDEFINED**

OP_EX_14 **UNDEFINED**

OP_EX_15 **UNDEFINED**

OP_EX_16 **UNDEFINED**

OP_EX_17 **UNDEFINED**

OP_EX_18 **UNDEFINED**

OP_EX_19 **UNDEFINED**

OP_EX_1A **UNDEFINED**

OP_EX_1B **UNDEFINED**

OP_EX_1C **UNDEFINED**

OP_EX_1D **UNDEFINED**

OP_EX_1E **UNDEFINED**

OP_EX_1F **UNDEFINED**

Definition at line 139 of file dcpu.h.

14.30 dcpu.h

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#ifndef __DCPU_H__
#define __DCPU_H__

#include "kerneltypes.h"
#include "ll.h"

//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
// Macros to access individual elements from within an opcode
#define DCPU_NORMAL_OPCODE_MASK( x ) \
        ((K_USHORT)(x & 0x001F))

#define DCPU_EXTENDED_OPCODE_MASK( x ) \
        ((K_USHORT)((x >> 5) & 0x001F))

#define DCPU_A_MASK( x ) \
        ((K_USHORT)((x >> 10) & 0x003F))

#define DCPU_B_MASK( x ) \
        ((K_USHORT)((x >> 5) & 0x001F))

//---------------------------------------------------------------------------
// Macros to emit opcodes in the normal/extended formats
#define DCPU_BUILD_NORMAL( x, y, z ) \
        ( ((K_USHORT)(x) & 0x001F) | ((K_USHORT)(y) & 0x001F) << 5 |
 ((K_USHORT)(z) & 0x003F) << 10 )

#define DCPU_BUILD_EXTENDED( x, y ) \
        ( ((K_USHORT)(x & 0x001F) << 5) | ((K_USHORT)(y & 0x003F) << 10) )

//---------------------------------------------------------------------------
typedef struct
{
    union
    {
        struct
        {
            K_USHORT A;
            K_USHORT B;
            K_USHORT C;
            K_USHORT X;
            K_USHORT Y;
            K_USHORT Z;
            K_USHORT I;
            K_USHORT J;
            K_USHORT PC;
            K_USHORT SP;
            K_USHORT EX;
            K_USHORT IA;
        };
        K_USHORT ausRegisters[12];
    };
} DCPU_Registers;

//---------------------------------------------------------------------------
typedef enum
{
    OP_NON_BASIC = 0,
    OP_SET,
    OP_ADD,
    OP_SUB,
    OP_MUL,
    OP_MLI,
    OP_DIV,
    OP_DVI,
    OP_MOD,
    OP_MDI,
    OP_AND,
    OP_BOR,
    OP_XOR,
    OP_SHR,
    OP_ASR,
    OP_SHL,
    OP_IFB,
    OP_IFC,
    OP_IFE,
    OP_IFN,
    OP_IFG,
    OP_IFA,
    OP_IFL,
    OP_IFU,
    OP_18,
    OP_19,
    OP_ADX,
    OP_SBX,
    OP_1C,
    OP_1D,
    OP_STI,
    OP_STD
} DCPU_OpBasic;

//---------------------------------------------------------------------------
typedef enum
{
    OP_EX_RESERVED = 0,
    OP_EX_JSR,
    OP_EX_2,
    OP_EX_3,
    OP_EX_4,
    OP_EX_5,
    OP_EX_6,
    OP_EX_7,
    OP_EX_INT,
    OP_EX_IAG,
    OP_EX_IAS,
    OP_EX_RFI,
    OP_EX_IAQ,
    OP_EX_D,
    OP_EX_E,
    OP_EX_F,
    OP_EX_HWN,
    OP_EX_HWQ,
    OP_EX_HWI,
    OP_EX_13,
    OP_EX_14,
    OP_EX_15,
    OP_EX_16,
    OP_EX_17,
    OP_EX_18,
    OP_EX_19,
    OP_EX_1A,
    OP_EX_1B,
    OP_EX_1C,
    OP_EX_1D,
    OP_EX_1E,
    OP_EX_1F
} DCPU_OpExtended;

//---------------------------------------------------------------------------
typedef enum
{
    ARG_A = 0,
    ARG_B,
    ARG_C,
    ARG_X,
    ARG_Y,
    ARG_Z,
    ARG_I,
    ARG_J,

    ARG_BRACKET_A,
    ARG_BRACKET_B,
    ARG_BRACKET_C,
    ARG_BRACKET_X,
    ARG_BRACKET_Y,
    ARG_BRACKET_Z,
    ARG_BRACKET_I,
    ARG_BRACKET_J,

    ARG_WORD_A,
    ARG_WORD_B,
    ARG_WORD_C,
    ARG_WORD_X,
    ARG_WORD_Y,
    ARG_WORD_Z,
    ARG_WORD_I,
    ARG_WORD_J,

    ARG_PUSH_POP_SP,
    ARG_PEEK_SP,
    ARG_WORD_SP,
    ARG_SP,
    ARG_PC,
    ARG_EX,
    ARG_NEXT_WORD,
    ARG_NEXT_LITERAL,

    ARG_LITERAL_0,
    ARG_LITERAL_1,
    ARG_LITERAL_2,
    ARG_LITERAL_3,
    ARG_LITERAL_4,
    ARG_LITERAL_5,
    ARG_LITERAL_6,
    ARG_LITERAL_7,
    ARG_LITERAL_8,
    ARG_LITERAL_9,
    ARG_LITERAL_A,
    ARG_LITERAL_B,
    ARG_LITERAL_C,
    ARG_LITERAL_D,
    ARG_LITERAL_E,
    ARG_LITERAL_F,
    ARG_LITERAL_10,
    ARG_LITERAL_11,
    ARG_LITERAL_12,
    ARG_LITERAL_13,
    ARG_LITERAL_14,
    ARG_LITERAL_15,
    ARG_LITERAL_16,
    ARG_LITERAL_17,
    ARG_LITERAL_18,
    ARG_LITERAL_19,
    ARG_LITERAL_1A,
    ARG_LITERAL_1B,
    ARG_LITERAL_1C,
    ARG_LITERAL_1D,
    ARG_LITERAL_1E,
    ARG_LITERAL_1F

} DCPU_Argument;

//---------------------------------------------------------------------------
class DCPU; // Forward declaration - required by the plugin class

//---------------------------------------------------------------------------
typedef void (*DCPU_Callback)(DCPU *pclVM_);

//---------------------------------------------------------------------------
```

```
class DCPUPlugin : public LinkListNode
{
public:
    void Init(  K_USHORT usDeviceNumber_,
                K_ULONG ulHWID_,
                K_ULONG ulVID_,
                K_USHORT usVersion_,
                DCPU_Callback pfCallback_)
    {
        m_ulHWID = ulHWID_;
        m_ulVID = ulVID_;
        m_usDeviceNumber = usDeviceNumber_;
        m_usVersion = usVersion_;
        m_pfCallback = pfCallback_;
    }

    void Enumerate( DCPU_Registers *pstRegisters_ )
    {
        pstRegisters_->A = (K_USHORT)(m_ulHWID & 0x0000FFFF);
        pstRegisters_->B = (K_USHORT)((m_ulHWID >> 16) & 0x0000FFFF);
        pstRegisters_->C = m_usVersion;
        pstRegisters_->X = (K_USHORT)(m_ulVID & 0x0000FFFF);
        pstRegisters_->Y = (K_USHORT)((m_ulVID >> 16) & 0x0000FFFF);
    }

    void Interrupt( DCPU *pclCPU_ )
    {
        m_pfCallback(pclCPU_);
    }

    K_USHORT GetDeviceNumber()
    {
        return m_usDeviceNumber;
    }

    friend class DCPUPluginList;
private:
    K_USHORT      m_usDeviceNumber;
    K_ULONG       m_ulHWID;
    K_ULONG       m_ulVID;
    K_USHORT      m_usVersion;

    DCPU_Callback m_pfCallback;
};

//---------------------------------------------------------------------------
class DCPU
{
public:
    void Init( K_USHORT *pusRAM_, K_USHORT usRAMSize_, const K_USHORT *
pusROM_, K_USHORT usROMSize_ );

    void RunOpcode();

    DCPU_Registers *GetRegisters() { return &
m_stRegisters; }

    void SendInterrupt( K_USHORT usMessage_ );

    void AddPlugin( DCPUPlugin *pclPlugin_ );

private:

    // Basic opcodes
    void SET();
    void ADD();
    void SUB();
    void MUL();
    void MLI();
    void DIV();
    void DVI();
    void MOD();
    void MDI();
    void AND();
    void BOR();
    void XOR();
    void SHR();
    void ASR();
    void SHL();
    bool IFB();
    bool IFC();
    bool IFE();
    bool IFN();
    bool IFG();
    bool IFA();
    bool IFL();
    bool IFU();
    void ADX();
    void SBX();
    void STI();
    void STD();

    // Extended opcodes
    void JSR();
    void INT();
    void IAG();
    void IAS();
    void RFI();
    void IAQ();
    void HWN();
    void HWQ();
    void HWI();

    K_UCHAR GetOperand( K_UCHAR ucOpType_, K_USHORT **pusResult_ );


    void ProcessInterruptQueue();

    DCPU_Registers m_stRegisters;

    K_USHORT *a;
    K_USHORT *b;

    K_USHORT m_usTempA;

    K_USHORT *m_pusRAM;
    K_USHORT m_usRAMSize;

    K_USHORT *m_pusROM;
    K_USHORT m_usROMSize;

    K_ULONG  m_ulCycleCount;

    K_BOOL   m_bInterruptQueueing;
    K_UCHAR  m_ucQueueLevel;
    K_USHORT m_ausInterruptQueue[ 8 ];

    DoubleLinkList m_clPluginList;
};

#endif
```

14.31 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/debug_tokens.h File Reference

Hex codes/translation tables used for efficient string tokenization.

Macros

- #define BLOCKING_CPP 0x0001 /∗ SUBSTITUTE="blocking.cpp" ∗/

  Source file names start at 0x0000.
- #define **DRIVER_CPP** 0x0002 /∗ SUBSTITUTE="driver.cpp" ∗/
- #define **KERNEL_CPP** 0x0003 /∗ SUBSTITUTE="kernel.cpp" ∗/
- #define **LL_CPP** 0x0004 /∗ SUBSTITUTE="ll.cpp" ∗/
- #define **MESSAGE_CPP** 0x0005 /∗ SUBSTITUTE="message.cpp" ∗/
- #define **MUTEX_CPP** 0x0006 /∗ SUBSTITUTE="mutex.cpp" ∗/
- #define **PROFILE_CPP** 0x0007 /∗ SUBSTITUTE="profile.cpp" ∗/
- #define **QUANTUM_CPP** 0x0008 /∗ SUBSTITUTE="quantum.cpp" ∗/
- #define **SCHEDULER_CPP** 0x0009 /∗ SUBSTITUTE="scheduler.cpp" ∗/
- #define **SEMAPHORE_CPP** 0x000A /∗ SUBSTITUTE="semaphore.cpp" ∗/
- #define **THREAD_CPP** 0x000B /∗ SUBSTITUTE="thread.cpp" ∗/
- #define **THREADLIST_CPP** 0x000C /∗ SUBSTITUTE="threadlist.cpp" ∗/
- #define **TIMERLIST_CPP** 0x000D /∗ SUBSTITUTE="timerlist.cpp" ∗/
- #define **KERNELSWI_CPP** 0x000E /∗ SUBSTITUTE="kernelswi.cpp" ∗/
- #define **KERNELTIMER_CPP** 0x000F /∗ SUBSTITUTE="kerneltimer.cpp" ∗/

- #define **KPROFILE_CPP** 0x0010 /∗ SUBSTITUTE="kprofile.cpp" ∗/
- #define **THREADPORT_CPP** 0x0011 /∗ SUBSTITUTE="threadport.cpp" ∗/
- #define BLOCKING_H 0x1000 /∗ SUBSTITUTE="blocking.h" ∗/

  Header file names start at 0x1000.
- #define **DRIVER_H** 0x1001 /∗ SUBSTITUTE="driver.h" ∗/
- #define **KERNEL_H** 0x1002 /∗ SUBSTITUTE="kernel.h" ∗/
- #define **KERNELTYPES_H** 0x1003 /∗ SUBSTITUTE="kerneltypes.h" ∗/
- #define **LL_H** 0x1004 /∗ SUBSTITUTE="ll.h" ∗/
- #define **MANUAL_H** 0x1005 /∗ SUBSTITUTE="manual.h" ∗/
- #define **MARK3CFG_H** 0x1006 /∗ SUBSTITUTE="mark3cfg.h" ∗/
- #define **MESSAGE_H** 0x1007 /∗ SUBSTITUTE="message.h" ∗/
- #define **MUTEX_H** 0x1008 /∗ SUBSTITUTE="mutex.h" ∗/
- #define **PROFILE_H** 0x1009 /∗ SUBSTITUTE="profile.h" ∗/
- #define **PROFILING_RESULTS_H** 0x100A /∗ SUBSTITUTE="profiling_results.h" ∗/
- #define **QUANTUM_H** 0x100B /∗ SUBSTITUTE="quantum.h" ∗/
- #define **SCHEDULER_H** 0x100C /∗ SUBSTITUTE="scheduler.h" ∗/
- #define **SEMAPHORE_H** 0x100D /∗ SUBSTITUTE="semaphore.h" ∗/
- #define **THREAD_H** 0x100E /∗ SUBSTITUTE="thread.h" ∗/
- #define **THREADLIST_H** 0x100F /∗ SUBSTITUTE="threadlist.h" ∗/
- #define **TIMERLIST_H** 0x1010 /∗ SUBSTITUTE="timerlist.h" ∗/
- #define **KERNELSWI_H** 0x1011 /∗ SUBSTITUTE="kernelswi.h ∗/
- #define **KERNELTIMER_H** 0x1012 /∗ SUBSTITUTE="kerneltimer.h ∗/
- #define **KPROFILE_H** 0x1013 /∗ SUBSTITUTE="kprofile.h" ∗/
- #define **THREADPORT_H** 0x1014 /∗ SUBSTITUTE="threadport.h" ∗/
- #define STR_PANIC 0x2000 /∗ SUBSTITUTE="!Panic!" ∗/

  Indexed strings start at 0x2000.
- #define **STR_MARK3_INIT** 0x2001 /∗ SUBSTITUTE="Initializing Kernel Objects" ∗/
- #define **STR_KERNEL_ENTER** 0x2002 /∗ SUBSTITUTE="Starting Kernel" ∗/
- #define **STR_THREAD_START** 0x2003 /∗ SUBSTITUTE="Switching to First Thread" ∗/
- #define **STR_START_ERROR** 0x2004 /∗ SUBSTITUTE="Error starting kernel - function should never return" ∗/
- #define **STR_THREAD_CREATE** 0x2005 /∗ SUBSTITUTE="Creating Thread" ∗/
- #define **STR_STACK_SIZE_1** 0x2006 /∗ SUBSTITUTE=" Stack Size: %1" ∗/
- #define **STR_PRIORITY_1** 0x2007 /∗ SUBSTITUTE=" Priority: %1" ∗/
- #define **STR_THREAD_ID_1** 0x2008 /∗ SUBSTITUTE=" Thread ID: %1" ∗/
- #define **STR_ENTRYPOINT_1** 0x2009 /∗ SUBSTITUTE=" EntryPoint: %1" ∗/
- #define **STR_CONTEXT_SWITCH_1** 0x200A /∗ SUBSTITUTE="Context Switch To Thread: %1" ∗/
- #define **STR_IDLING** 0x200B /∗ SUBSTITUTE="Idling CPU" ∗/
- #define **STR_WAKEUP** 0x200C /∗ SUBSTITUTE="Waking up" ∗/
- #define **STR_SEMAPHORE_PEND_1** 0x200D /∗ SUBSTITUTE="Semaphore Pend: %1" ∗/
- #define **STR_SEMAPHORE_POST_1** 0x200E /∗ SUBSTITUTE="Semaphore Post: %1" ∗/
- #define **STR_MUTEX_CLAIM_1** 0x200F /∗ SUBSTITUTE="Mutex Claim: %1" ∗/
- #define **STR_MUTEX_RELEASE_1** 0x2010 /∗ SUBSTITUTE="Mutex Release: %1" ∗/
- #define **STR_THREAD_BLOCK_1** 0x2011 /∗ SUBSTITUTE="Thread %1 Blocked" ∗/
- #define **STR_THREAD_UNBLOCK_1** 0x2012 /∗ SUBSTITUTE="Thread %1 Unblocked" ∗/
- #define **STR_ASSERT_FAILED** 0x2013 /∗ SUBSTITUTE="Assertion Failed" ∗/
- #define **STR_SCHEDULE_1** 0x2014 /∗ SUBSTITUTE="Scheduler chose %1" ∗/
- #define **STR_THREAD_START_1** 0x2015 /∗ SUBSTITUTE="Thread Start: %1" ∗/
- #define **STR_THREAD_EXIT_1** 0x2016 /∗ SUBSTITUTE="Thread Exit: %1" ∗/
- #define **STR_UNDEFINED** 0xFFFF /∗ SUBSTITUTE="UNDEFINED" ∗/

14.31.1 Detailed Description

Hex codes/translation tables used for efficient string tokenization. We use this for efficiently encoding strings used for kernel traces, debug prints, etc. The upside - this is really fast and efficient for encoding strings and data. Downside? The tools need to parse this header file in order to convert the enumerated data into actual strings, decoding them.

Definition in file debug_tokens.h.

14.32 debug_tokens.h

```
/*===========================================================================
     _____     _____     _____     _____
 ___|    _|__ __|_    |__ __|__   |__ __| __  |__ ______
|    \  /  | ||    \     ||     |     ||  |/ /    ||___   |
|     \/   | ||     \    ||     \     ||     \    ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]-------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#ifndef __DEBUG_TOKENS_H__
#define __DEBUG_TOKENS_H__
//---------------------------------------------------------------------------
#define BLOCKING_CPP        0x0001      /* SUBSTITUTE="blocking.cpp" */
#define DRIVER_CPP          0x0002      /* SUBSTITUTE="driver.cpp" */
#define KERNEL_CPP          0x0003      /* SUBSTITUTE="kernel.cpp" */
#define LL_CPP              0x0004      /* SUBSTITUTE="ll.cpp" */
#define MESSAGE_CPP         0x0005      /* SUBSTITUTE="message.cpp" */
#define MUTEX_CPP           0x0006      /* SUBSTITUTE="mutex.cpp" */
#define PROFILE_CPP         0x0007      /* SUBSTITUTE="profile.cpp" */
#define QUANTUM_CPP         0x0008      /* SUBSTITUTE="quantum.cpp" */
#define SCHEDULER_CPP       0x0009      /* SUBSTITUTE="scheduler.cpp" */
#define SEMAPHORE_CPP       0x000A      /* SUBSTITUTE="semaphore.cpp" */
#define THREAD_CPP          0x000B      /* SUBSTITUTE="thread.cpp" */
#define THREADLIST_CPP      0x000C      /* SUBSTITUTE="threadlist.cpp" */
#define TIMERLIST_CPP       0x000D      /* SUBSTITUTE="timerlist.cpp" */
#define KERNELSWI_CPP       0x000E      /* SUBSTITUTE="kernelswi.cpp" */
#define KERNELTIMER_CPP     0x000F      /* SUBSTITUTE="kerneltimer.cpp" */
#define KPROFILE_CPP        0x0010      /* SUBSTITUTE="kprofile.cpp" */
#define THREADPORT_CPP      0x0011      /* SUBSTITUTE="threadport.cpp" */

//---------------------------------------------------------------------------
#define BLOCKING_H          0x1000      /* SUBSTITUTE="blocking.h" */
#define DRIVER_H            0x1001      /* SUBSTITUTE="driver.h" */
#define KERNEL_H            0x1002      /* SUBSTITUTE="kernel.h" */
#define KERNELTYPES_H       0x1003      /* SUBSTITUTE="kerneltypes.h" */
#define LL_H                0x1004      /* SUBSTITUTE="ll.h" */
#define MANUAL_H            0x1005      /* SUBSTITUTE="manual.h" */
#define MARK3CFG_H          0x1006      /* SUBSTITUTE="mark3cfg.h" */
#define MESSAGE_H           0x1007      /* SUBSTITUTE="message.h" */
#define MUTEX_H             0x1008      /* SUBSTITUTE="mutex.h" */
#define PROFILE_H           0x1009      /* SUBSTITUTE="profile.h" */
#define PROFILING_RESULTS_H 0x100A      /* SUBSTITUTE="profiling_results.h" */
#define QUANTUM_H           0x100B      /* SUBSTITUTE="quantum.h" */
#define SCHEDULER_H         0x100C      /* SUBSTITUTE="scheduler.h" */
#define SEMAPHORE_H         0x100D      /* SUBSTITUTE="semaphore.h" */
#define THREAD_H            0x100E      /* SUBSTITUTE="thread.h" */
#define THREADLIST_H        0x100F      /* SUBSTITUTE="threadlist.h" */
#define TIMERLIST_H         0x1010      /* SUBSTITUTE="timerlist.h" */
#define KERNELSWI_H         0x1011      /* SUBSTITUTE="kernelswi.h */
#define KERNELTIMER_H       0x1012      /* SUBSTITUTE="kerneltimer.h */
#define KPROFILE_H          0x1013      /* SUBSTITUTE="kprofile.h" */
#define THREADPORT_H        0x1014      /* SUBSTITUTE="threadport.h" */

//---------------------------------------------------------------------------
#define STR_PANIC               0x2000      /* SUBSTITUTE="!Panic!" */
#define STR_MARK3_INIT          0x2001      /* SUBSTITUTE="Initializing Kernel
 Objects" */
#define STR_KERNEL_ENTER        0x2002      /* SUBSTITUTE="Starting Kernel" */
#define STR_THREAD_START        0x2003      /* SUBSTITUTE="Switching to First
 Thread" */
#define STR_START_ERROR         0x2004      /* SUBSTITUTE="Error starting
 kernel - function should never return" */
#define STR_THREAD_CREATE       0x2005      /* SUBSTITUTE="Creating Thread" */
#define STR_STACK_SIZE_1        0x2006      /* SUBSTITUTE="  Stack Size: %1" */
#define STR_PRIORITY_1          0x2007      /* SUBSTITUTE="  Priority: %1" */
```

```
#define STR_THREAD_ID_1         0x2008      /* SUBSTITUTE="  Thread ID: %1" */
#define STR_ENTRYPOINT_1        0x2009      /* SUBSTITUTE="  EntryPoint: %1" */
#define STR_CONTEXT_SWITCH_1    0x200A      /* SUBSTITUTE="Context Switch To
 Thread: %1" */
#define STR_IDLING              0x200B      /* SUBSTITUTE="Idling CPU" */
#define STR_WAKEUP              0x200C      /* SUBSTITUTE="Waking up" */
#define STR_SEMAPHORE_PEND_1    0x200D      /* SUBSTITUTE="Semaphore Pend: %1"
*/
#define STR_SEMAPHORE_POST_1    0x200E      /* SUBSTITUTE="Semaphore Post: %1"
*/
#define STR_MUTEX_CLAIM_1       0x200F      /* SUBSTITUTE="Mutex Claim: %1" */
#define STR_MUTEX_RELEASE_1     0x2010      /* SUBSTITUTE="Mutex Release: %1"
*/
#define STR_THREAD_BLOCK_1      0x2011      /* SUBSTITUTE="Thread %1 Blocked"
*/
#define STR_THREAD_UNBLOCK_1    0x2012      /* SUBSTITUTE="Thread %1 Unblocked"
 */
#define STR_ASSERT_FAILED       0x2013      /* SUBSTITUTE="Assertion Failed" */
#define STR_SCHEDULE_1          0x2014      /* SUBSTITUTE="Scheduler chose %1"
*/
#define STR_THREAD_START_1      0x2015      /* SUBSTITUTE="Thread Start: %1" */
#define STR_THREAD_EXIT_1       0x2016      /* SUBSTITUTE="Thread Exit: %1" */

//---------------------------------------------------------------------------
#define STR_UNDEFINED           0xFFFF      /* SUBSTITUTE="UNDEFINED" */
#endif
```

14.33 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/draw.h File Reference

Raster graphics APIs Description: Implements basic drawing functionality.

```
#include "kerneltypes.h"
#include "font.h"
#include "colorspace.h"
```

Classes

- struct DrawPoint_t

  Defines a pixel.
- struct DrawLine_t

  Defines a simple line.
- struct DrawRectangle_t

  Defines a rectangle.
- struct DrawCircle_t

  Defines a circle.
- struct DrawEllipse_t

  Defines a ellipse.
- struct DrawBitmap_t

  Defines a bitmap.
- struct DrawStamp_t

  Defines a 1-bit 2D bitmap of arbitrary resolution.
- struct DrawText_t

  Defines a bitmap-rendered string.
- struct DrawWindow_t

  Defines the active window - establishes boundaries for drawing on the current display.
- struct DrawMove_t

  Simple 2D copy/paste.
- struct DrawVector_t

  Specifies a single 2D point.

- struct DrawPoly_t

    Defines the structure of an arbitrary polygon.

Enumerations

- enum **DisplayEvent_t** {
  DISPLAY_EVENT_SET_PIXEL = 0x00, **DISPLAY_EVENT_GET_PIXEL**, **DISPLAY_EVENT_CLEAR**, **DISPLAY_EVENT_LINE**,
  DISPLAY_EVENT_RECTANGLE, **DISPLAY_EVENT_CIRCLE**, **DISPLAY_EVENT_ELLIPSE**, **DISPLAY_EVENT_BITMAP**,
  DISPLAY_EVENT_STAMP, **DISPLAY_EVENT_TEXT**, **DISPLAY_EVENT_MOVE**, **DISPLAY_EVENT_POLY** }

14.33.1 Detailed Description

Raster graphics APIs Description: Implements basic drawing functionality. This forms a hardware abstraction layer which requires a backend for rendering.

Definition in file draw.h.

14.34 draw.h

```
/*===========================================================================
     _____     _____     _____     _____
 ___|    _|__ __|_    |__ __|__   |__ __| __  |__  ______
|    \  /  | ||    \     ||     |     ||  |/ /     ||___   |
|     \/   | ||     \    ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
//---------------------------------------------------------------------------

#ifndef __DRAW_H_
#define __DRAW_H_

#include "kerneltypes.h"
#include "font.h"
#include "colorspace.h"

//---------------------------------------------------------------------------
// Event definitions for 2D hardware accelerated graphics functions
typedef enum
{
    //--[Mandatory for a display driver]-----------------------------------
    DISPLAY_EVENT_SET_PIXEL = 0x00,
    DISPLAY_EVENT_GET_PIXEL,

    //--[Optional if supported in hardware]--------------------------------
    DISPLAY_EVENT_CLEAR,
    DISPLAY_EVENT_LINE,
    DISPLAY_EVENT_RECTANGLE,
    DISPLAY_EVENT_CIRCLE,
    DISPLAY_EVENT_ELLIPSE,
    DISPLAY_EVENT_BITMAP,
    DISPLAY_EVENT_STAMP,
    DISPLAY_EVENT_TEXT,
    DISPLAY_EVENT_MOVE,
    DISPLAY_EVENT_POLY
} DisplayEvent_t;

//---------------------------------------------------------------------------
typedef struct
{
    K_USHORT usX;
    K_USHORT usY;
    COLOR uColor;
} DrawPoint_t;
```

```

//---------------------------------------------------------------------------
typedef struct
{
    K_USHORT usX1;
    K_USHORT usX2;
    K_USHORT usY1;
    K_USHORT usY2;
    COLOR uColor;
} DrawLine_t;
//---------------------------------------------------------------------------
typedef struct
{
    K_USHORT usLeft;
    K_USHORT usTop;
    K_USHORT usRight;
    K_USHORT usBottom;
    COLOR uLineColor;
    K_BOOL bFill;
    COLOR uFillColor;
} DrawRectangle_t;
//---------------------------------------------------------------------------
typedef struct
{
    K_USHORT usX;
    K_USHORT usY;
    K_USHORT usRadius;
    COLOR uLineColor;
    K_BOOL bFill;
    COLOR uFillColor;
} DrawCircle_t;
//---------------------------------------------------------------------------
typedef struct
{
    K_USHORT usX;
    K_USHORT usY;
    K_USHORT usHeight;
    K_USHORT usWidth;
    COLOR uColor;
} DrawEllipse_t;
//---------------------------------------------------------------------------
typedef struct
{
    K_USHORT usX;
    K_USHORT usY;
    K_USHORT usWidth;
    K_USHORT usHeight;
    K_UCHAR ucBPP;
    K_UCHAR *pucData;
} DrawBitmap_t;
//---------------------------------------------------------------------------
typedef struct
{
    K_USHORT usX;
    K_USHORT usY;
    K_USHORT usWidth;
    K_USHORT usHeight;
    COLOR uColor;
    K_UCHAR *pucData;
} DrawStamp_t;     // monochrome stamp, bitpacked 8bpp

//---------------------------------------------------------------------------
typedef struct
{
    K_USHORT usLeft;
    K_USHORT usTop;
    COLOR uColor;
    Font_t *pstFont;
    const K_CHAR *pcString;
} DrawText_t;

//---------------------------------------------------------------------------
typedef struct
{
    K_USHORT usLeft;
    K_USHORT usRight;
    K_USHORT usTop;
    K_USHORT usBottom;
} DrawWindow_t;

//---------------------------------------------------------------------------
typedef struct
{
    K_USHORT usSrcX;
    K_USHORT usSrcY;
    K_USHORT usDstX;
    K_USHORT usDstY;
    K_USHORT usCopyHeight;
    K_USHORT usCopyWidth;
} DrawMove_t;

//---------------------------------------------------------------------------
typedef struct
{
    K_USHORT usX;
    K_USHORT usY;
} DrawVector_t;

//---------------------------------------------------------------------------
typedef struct
{
    K_USHORT     usNumPoints;
    COLOR        uColor;
    K_BOOL       bFill;
    DrawVector_t *pstVector;
} DrawPoly_t;

#endif //__DRAW_H_
```

14.35 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/driver.cpp File Reference

Device driver/hardware abstraction layer.

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "kernel_debug.h"
#include "driver.h"
```

Classes

- class DevNull

  This class implements the "default" driver (/dev/null)

Macros

- #define **__FILE_ID__** DRIVER_CPP

Functions

- static K_UCHAR **DrvCmp** (const K_CHAR ∗szStr1_, const K_CHAR ∗szStr2_)

Variables

- static DevNull **clDevNull**

14.35.1 Detailed Description

Device driver/hardware abstraction layer.

Definition in file driver.cpp.

14.36 driver.cpp

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "kernel_debug.h"
#include "driver.h"

//---------------------------------------------------------------------------
#if defined __FILE_ID__
    #undef __FILE_ID__
#endif
#define __FILE_ID__     DRIVER_CPP

//---------------------------------------------------------------------------
#if KERNEL_USE_DRIVER

DoubleLinkList DriverList::m_clDriverList
;

class DevNull : public Driver
{
public:
    virtual void Init() { SetName("/dev/null"); };
    virtual K_UCHAR Open() { return 0; }
    virtual K_UCHAR Close() { return 0; }

    virtual K_USHORT Read( K_USHORT usBytes_,
    K_UCHAR *pucData_){ return 0; }

    virtual K_USHORT Write( K_USHORT usBytes_,
    K_UCHAR *pucData_) { return 0; }

    virtual K_USHORT Control( K_USHORT usEvent_,
        void *pvDataIn_,
        K_USHORT usSizeIn_,
        void *pvDataOut_,
        K_USHORT usSizeOut_ ) { return 0; }

};

//---------------------------------------------------------------------------
static DevNull clDevNull;

//---------------------------------------------------------------------------
static K_UCHAR DrvCmp( const K_CHAR *szStr1_, const K_CHAR *szStr2_ )
{
    K_CHAR *szTmp1 = (K_CHAR*) szStr1_;
    K_CHAR *szTmp2 = (K_CHAR*) szStr2_;

    while (*szTmp1 && *szTmp2)
    {
        if (*szTmp1++ != *szTmp2++)
        {
            return 0;
        }
    }

    // Both terminate at the same length
    if (!(*szTmp1) && !(*szTmp2))
    {
        return 1;
    }

    return 0;
}

//---------------------------------------------------------------------------
void DriverList::Init()
{
    // Ensure we always have at least one entry - a default in case no match
    // is found (/dev/null)
    clDevNull.Init();
    Add(&clDevNull);
}

//---------------------------------------------------------------------------
Driver *DriverList::FindByPath( const K_CHAR *
m_pcPath )
{
    KERNEL_ASSERT( m_pcPath );
    Driver *pclTemp = static_cast<Driver*>(m_clDriverList
.GetHead());

    while (pclTemp)
    {
        if(DrvCmp(m_pcPath, pclTemp->GetPath()))
        {
            return pclTemp;
        }
        pclTemp = static_cast<Driver*>(pclTemp->GetNext());
    }
    return &clDevNull;
}

#endif
```

14.37 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/driver.h File Reference

Driver abstraction framework.

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "ll.h"
```

Classes

- class Driver

  Base device-driver class used in hardware abstraction.
- class DriverList

  List of Driver objects used to keep track of all device drivers in the system.

14.37.1 Detailed Description

Driver abstraction framework.

14.37.2 Intro

This is the basis of the driver framework. In the context of Mark3, drivers don't necessarily have to be based on physical hardware peripherals. They can be used to represent algorithms (such as random number generators), files, or protocol stacks. Unlike FunkOS, where driver IO is protected automatically by a mutex, we do not use this kind of protection - we leave it up to the driver implementor to do what's right in its own context. This also frees up the driver to implement all sorts of other neat stuff, like sending messages to threads associated with the driver. Drivers are implemented as character devices, with the standard array of posix-style accessor methods for reading, writing, and general driver control.

A global driver list is provided as a convenient and minimal "filesystem" structure, in which devices can be accessed by name.

14.37.3 Driver Design

A device driver needs to be able to perform the following operations: -Initialize a peripheral -Start/stop a peripheral -Handle I/O control operations -Perform various read/write operations

At the end of the day, that's pretty much all a device driver has to do, and all of the functionality that needs to be presented to the developer.

We abstract all device drivers using a base-class which implements the following methods: -Start/Open -Stop/Close -Control -Read -Write

A basic driver framework and API can thus be implemented in five function calls - that's it! You could even reduce that further by handling the initialize, start, and stop operations inside the "control" operation.

14.37.4 Driver API

In C++, we can implement this as a class to abstract these event handlers, with virtual void functions in the base class overridden by the inherited objects.

To add and remove device drivers from the global table, we use the following methods:

```
void DriverList::Add( Driver *pclDriver_ );
void DriverList::Remove( Driver *pclDriver_ );
```

DriverList::Add()/Remove() takes a single arguments the pointer to he object to operate on.

Once a driver has been added to the table, drivers are opened by NAME using DriverList::FindBy-Name("/dev/name"). This function returns a pointer to the specified driver if successful, or to a built in /dev/null device if the path name is invalid. After a driver is open, that pointer is used for all other driver access functions.

This abstraction is incredibly useful any peripheral or service can be accessed through a consistent set of APIs, that make it easy to substitute implementations from one platform to another. Portability is ensured, the overhead is negligible, and it emphasizes the reuse of both driver and application code as separate entities.

Consider a system with drivers for I2C, SPI, and UART peripherals - under our driver framework, an application can initialize these peripherals and write a greeting to each using the same simple API functions for all drivers:

```
pclI2C  = DriverList::FindByName("/dev/i2c");
pclUART = DriverList::FindByName("/dev/tty0");
pclSPI  = DriverList::FindByName("/dev/spi");

pclI2C->Write(12,"Hello World!");
pclUART->Write(12, "Hello World!");
pclSPI->Write(12, "Hello World!");
```

Definition in file driver.h.

14.38 driver.h

```
/*===========================================================================
     _____     _____     _____     _____
 ___|    _|__ __|_    |__ __|__   |__ __| __  |__  ______
|    \  /  | ||    \    ||     |    ||  |/ /    ||___   |
|     \/   | ||     \   ||     \    ||     \    ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "kerneltypes.h"
#include "mark3cfg.h"

#include "ll.h"

#ifndef __DRIVER_H__
```

```
#define __DRIVER_H__

#if KERNEL_USE_DRIVER

class DriverList;
//---------------------------------------------------------------------------
class Driver : public LinkListNode
{
public:
    virtual void Init() = 0;

    virtual K_UCHAR Open() = 0;

    virtual K_UCHAR Close() = 0;

    virtual K_USHORT Read( K_USHORT usBytes_,
                                 K_UCHAR *pucData_) = 0;

    virtual K_USHORT Write( K_USHORT usBytes_,
                                  K_UCHAR *pucData_) = 0;

    virtual K_USHORT Control( K_USHORT usEvent_,
                                    void *pvDataIn_,
                                    K_USHORT usSizeIn_,
                                    void *pvDataOut_,
                                    K_USHORT usSizeOut_ ) = 0;

    void SetName( const K_CHAR *pcName_ ) { m_pcPath = pcName_;
}

    const K_CHAR *GetPath() { return m_pcPath; }

private:

    const K_CHAR *m_pcPath;
};

//---------------------------------------------------------------------------
class DriverList
{
public:
    static void Init();

    static void Add( Driver *pclDriver_ ) { m_clDriverList
.Add(pclDriver_); }

    static void Remove( Driver *pclDriver_ ) { m_clDriverList
.Remove(pclDriver_); }

    static Driver *FindByPath( const K_CHAR *m_pcPath );

private:

    static DoubleLinkList m_clDriverList;
};

#endif //KERNEL_USE_DRIVER

#endif
```

14.39 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/fixed_heap.cpp File Reference

Fixed-block-size memory management.

```
#include "kerneltypes.h"
#include "fixed_heap.h"
#include "threadport.h"
```

14.39.1 Detailed Description

Fixed-block-size memory management. This allows a user to create heaps containing multiple lists, each list containing a linked-list of blocks that are each the same size. As a result of the linked-list format, these heaps are very fast - requiring only a linked list pop/push to allocated/free memory. Array traversal is required to allow for the optimal heap to be used. Blocks are chosen from the first heap with free blocks large enough to fulfill the request.

Only simple malloc/free functionlality is supported in this implementation, no complex vector-allocate or reallocation functions are supported.

Heaps are protected by critical section, and are thus thread-safe.

When creating a heap, a user supplies an array of heap configuration objects, which determines how many objects of what size are available.

The configuration objects are defined from smallest list to largest, the memory to back the heap is supplied as a pointer to a "blob" of memory which will be used to create the underlying heap objects that make up the heap internal data structures. This blob must be large enough to contain all of the requested heap objects, with all of the additional metadata required to manage the objects.

Multiple heaps can be created using this library (heaps are not singleton).

Definition in file fixed_heap.cpp.

14.40 fixed_heap.cpp

```
/*===========================================================================
     _____     _____     _____     _____
 ___|    _|__ __|_    |__ __|__   |__ __| __  |__  ______
|    \  /  | ||    \    ||     |     ||  |/ /     ||___   |
|     \/   | ||     \   ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\ __||__|\__\ __||______|
    |_____|      |_____|     |_____|     |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "kerneltypes.h"
#include "fixed_heap.h"
#include "threadport.h"

//---------------------------------------------------------------------------
void *BlockHeap::Create( void *pvHeap_, K_USHORT usSize_,
K_USHORT usBlockSize_ )
{
    K_USHORT usNodeCount = usSize_ /
                                  (usBlockSize_ + sizeof(LinkListNode
) + sizeof(void*));
    void *pvNode = pvHeap_;
    void *pvMaxNode = (void*)((K_ULONG)pvHeap_ + (K_ULONG)usSize_);
    m_clList.Init();

    // Create a heap (linked-list nodes + byte pool) in the middle of
    // the data blob
    for (K_USHORT i = 0; i < usNodeCount; i++ )
    {
        // Create a pointer back to the source list.
        BlockHeap **pclTemp = (BlockHeap**)(pvNode + sizeof(
LinkListNode));
        *pclTemp = (BlockHeap*)(this);

        // Add the node to the block list
        m_clList.Add( (LinkListNode*)pvNode );

        // Move the pointer in the pool to point to the next block to allocate
        pvNode += (usBlockSize_ + sizeof(LinkListNode) + sizeof(
BlockHeap*));

        // Bail if we would be going past the end of the allocated space...
        if ((K_ULONG)pvNode >= (K_ULONG)pvMaxNode)
        {
            break;
        }
    }
    m_usBlocksFree = usNodeCount;

    // Return pointer to end of heap (used for heap-chaining)
    return pvNode;
}

//---------------------------------------------------------------------------
void *BlockHeap::Alloc()
```

```
{
    LinkListNode *pclNode = m_clList.GetHead();

    // Return the first node from the head of the list
    if (pclNode)
    {
        m_clList.Remove( pclNode );
        m_usBlocksFree--;

        // Account for block-management metadata
        return (void*)pclNode + sizeof(LinkListNode) + sizeof(void
*);
    }

    // Or null, if the heap is empty.
    return 0;
}

//---------------------------------------------------------------------------
void BlockHeap::Free( void* pvData_ )
{
    // Compute the address of the original object (class metadata included)
    LinkListNode *pclNode = (LinkListNode*)(pvData_ -
sizeof(LinkListNode) - sizeof(void*));

    // Add the object back to the block data pool
    m_clList.Add(pclNode);
    m_usBlocksFree++;
}

//---------------------------------------------------------------------------
void FixedHeap::Create( void *pvHeap_, HeapConfig *
pclHeapConfig_ )
{
    K_USHORT i = 0;
    void *pvTemp = pvHeap_;
    while( pclHeapConfig_[i].m_usBlockSize != 0)
    {
        pvTemp = pclHeapConfig_[i].m_clHeap.Create
                    (pvTemp,
                     (pclHeapConfig_[i].m_usBlockSize +sizeof(LinkListNode
) + sizeof(void*)) *
                     pclHeapConfig_[i].m_usBlockCount,
                     pclHeapConfig_[i].m_usBlockSize );
        i++;
    }
    m_paclHeaps = pclHeapConfig_;
}

//---------------------------------------------------------------------------
void *FixedHeap::Alloc( K_USHORT usSize_ )
{
    void *pvRet = 0;
    K_USHORT i = 0;

    // Go through all heaps, trying to find the smallest one that
    // has a free item to satisfy the allocation
    while (m_paclHeaps[i].m_usBlockSize != 0)
    {
        CS_ENTER();
        if ((m_paclHeaps[i].m_usBlockSize >= usSize_) && m_paclHeaps
[i].m_clHeap.IsFree() )
        {
            // Found a match
            pvRet = m_paclHeaps[i].m_clHeap.Alloc();

        }
        CS_EXIT();

        // Return an object if found
        if (pvRet)
        {
            return pvRet;
        }
        i++;
    }

    // Or null on no-match
    return pvRet;
}

//---------------------------------------------------------------------------
void FixedHeap::Free( void *pvNode_ )
{
    // Compute the pointer to the block-heap this block belongs to, and
    // return it.
    CS_ENTER();
    BlockHeap **pclHeap = (BlockHeap**)(pvNode_ - sizeof(
BlockHeap*));
    (*pclHeap)->Free(pvNode_);
    CS_EXIT();
}


```

14.41 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/fixed_heap.h File Reference

Fixed-block-size heaps.

```
#include "kerneltypes.h"
#include "ll.h"
```

Classes

- class BlockHeap

  Single-block-size heap.
- class HeapConfig

  Heap configuration object.
- class FixedHeap

  Fixed-size-block heap allocator with multiple block sizes.

14.41.1 Detailed Description

Fixed-block-size heaps.

Definition in file fixed_heap.h.

14.42 fixed_heap.h

```
/*============================================================================
     _____     _____     _____     _____
 ___|    _|__ __|_    |__ __|__   |__ __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]---------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
============================================================================*/
#ifndef __FIXED_HEAP_H__
#define __FIXED_HEAP_H__

#include "kerneltypes.h"
#include "ll.h"

//----------------------------------------------------------------------------
class BlockHeap
{
public:
    void *Create( void *pvHeap_, K_USHORT usSize_, K_USHORT usBlockSize_
);

    void *Alloc();

    void Free( void* pvData_ );

    K_BOOL IsFree() { return m_usBlocksFree != 0; }

protected:
    K_USHORT m_usBlocksFree;

private:
    DoubleLinkList m_clList;
};


class FixedHeap;

//---------------------------------------------------------------------------
class HeapConfig
{
public:
    K_USHORT m_usBlockSize;
    K_USHORT m_usBlockCount;
    friend class FixedHeap;
protected:
    BlockHeap m_clHeap;
};

//---------------------------------------------------------------------------
class FixedHeap
{
public:
    void Create( void *pvHeap_, HeapConfig *pclHeapConfig_ );

    void *Alloc( K_USHORT usSize_ );

    static void Free( void *pvNode_ );

private:
    HeapConfig *m_paclHeaps;
};

#endif

```

14.43 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/font.h File Reference

Font structure definitions.

```
#include "kerneltypes.h"
#include "fontport.h"
```

Classes

- struct Glyph_t
- struct Font_t

Macros

- #define GLYPH_SIZE(x) (((K_USHORT)((x->ucWidth + 7) >> 3) * (K_USHORT)(x->ucHeight)) + sizeof(Glyph_t) - 1)

  The size of the glyph is the width∗height (in bytes), plus the overhead of the struct parameters.

14.43.1 Detailed Description

Font structure definitions.

Definition in file font.h.

14.44 font.h

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#ifndef __FONT_H__
#define __FONT_H__

#include "kerneltypes.h"
#include "fontport.h"

//---------------------------------------------------------------------------
typedef struct
{
    K_UCHAR ucWidth;
    K_UCHAR ucHeight;
    K_UCHAR ucVOffset;
    K_UCHAR aucData[1];
} Glyph_t;

//---------------------------------------------------------------------------
#define GLYPH_SIZE(x) \
    (((K_USHORT)((x->ucWidth + 7) >> 3) * (K_USHORT)(x->ucHeight)) +
 sizeof(Glyph_t) - 1)

//---------------------------------------------------------------------------
typedef struct
{
    K_UCHAR ucSize;
    K_UCHAR ucFlags;
    K_UCHAR ucStartChar;
    K_UCHAR ucMaxChar;
    K_CHAR *szName;
    const FONT_STORAGE_TYPE *pucFontData;
} Font_t;

#endif

```

14.45 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/graphics.cpp File Reference

Generic graphics driver implementation.

```
#include "kerneltypes.h"
#include "graphics.h"
#include "draw.h"
#include "driver.h"
#include "colorspace.h"
#include "font.h"
```

14.45.1 Detailed Description

Generic graphics driver implementation.

Definition in file graphics.cpp.

14.46 graphics.cpp

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /    ||___   |
|     \/   | ||     \     ||     \     ||     \    ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\ __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "kerneltypes.h"
#include "graphics.h"
#include "draw.h"
#include "driver.h"
#include "colorspace.h"
#include "font.h"

//---------------------------------------------------------------------------
void GraphicsDriver::ClearScreen()
{
    DrawPoint_t stPoint;
    stPoint.uColor = COLOR_BLACK;

    for (stPoint.usX = 0; stPoint.usX < m_usResX; stPoint.usX++)
    {
        for (stPoint.usY = 0; stPoint.usY < m_usResY; stPoint.usY++)
        {
            // Pixel Write
            DrawPixel(&stPoint);
        }
    }
}
//---------------------------------------------------------------------------
void GraphicsDriver::Point(DrawPoint_t *
pstPoint_)
{
    DrawPixel(pstPoint_);
}

//---------------------------------------------------------------------------
void GraphicsDriver::Line(DrawLine_t *pstLine_)
{
    // Bresenham Line drawing algorithm, adapted from:
    // www.cs.unc.edu/~mcmillan/comp136/Lecture6/Lines.html

    DrawPoint_t stPoint;
    K_SHORT sX1 = (K_SHORT)pstLine_->usX1;
    K_SHORT sX2 = (K_SHORT)pstLine_->usX2;
    K_SHORT sY1 = (K_SHORT)pstLine_->usY1;
    K_SHORT sY2 = (K_SHORT)pstLine_->usY2;
    K_SHORT sDeltaY = sY2 - sY1;
    K_SHORT sDeltaX = sX2 - sX1;
    K_CHAR cStepx, cStepy;
    stPoint.uColor = pstLine_->uColor;

    if (sDeltaY < 0)
    {
        sDeltaY = -sDeltaY;
        cStepy = -1;
    }
    else
    {
        cStepy = 1;
    }

    if (sDeltaX < 0)
    {
        sDeltaX = -sDeltaX;
        cStepx = -1;
    }
    else
    {
        cStepx = 1;
    }

    // Scale by a factor of 2 in each direction
    sDeltaY <<= 1;
    sDeltaX <<= 1;

    stPoint.usX = sX1;
    stPoint.usY = sY1;
    DrawPixel(&stPoint);

    if (sDeltaX > sDeltaY)
    {
        K_SHORT sFraction = sDeltaY - (sDeltaX >> 1);

        while (sX1 != sX2)
        {
            if (sFraction >= 0)
            {
                sY1 += cStepy;
                sFraction -= sDeltaX;
            }
            sX1 += cStepx;
            sFraction += sDeltaY;

            stPoint.usX = sX1;
            stPoint.usY = sY1;
            DrawPixel(&stPoint);
        }
    }
    else
    {
        K_SHORT sFraction = sDeltaX - (sDeltaY >> 1);
        while (sY1 != sY2)
        {
            if (sFraction >= 0)
            {
                sX1 += cStepx;
                sFraction -= sDeltaY;
            }
            sY1 += cStepy;
            sFraction += sDeltaX;

            stPoint.usX = sX1;
            stPoint.usY = sY1;
            DrawPixel(&stPoint);
        }
    }
}

//---------------------------------------------------------------------------
void GraphicsDriver::Rectangle(DrawRectangle_t *pstRectangle_)
{
    DrawPoint_t stPoint;

    // if drawing a background fill color (optional)
    if (pstRectangle_->bFill == true)
    {
        stPoint.uColor = pstRectangle_->uFillColor;
        for (stPoint.usX = pstRectangle_->usLeft; stPoint.usX <= pstRectangle_->usRight; stPoint.usX++)
        {
            for (stPoint.usY = pstRectangle_->usTop; stPoint.usY <= pstRectangle_->usBottom; stPoint.usY++)
            {
                DrawPixel(&stPoint);
            }
        }
    }

    // Draw four orthogonal lines...
    stPoint.uColor = pstRectangle_->uLineColor;
    stPoint.usY = pstRectangle_->usTop;
    for (stPoint.usX = pstRectangle_->usLeft; stPoint.usX <= pstRectangle_->usRight; stPoint.usX++)
    {
        DrawPixel(&stPoint);
    }

    stPoint.usY = pstRectangle_->usBottom;
    for (stPoint.usX = pstRectangle_->usLeft; stPoint.usX <= pstRectangle_->usRight; stPoint.usX++)
    {
        DrawPixel(&stPoint);
    }

    stPoint.usX = pstRectangle_->usLeft;
    for (stPoint.usY = pstRectangle_->usTop; stPoint.usY <= pstRectangle_->usBottom; stPoint.usY++)
    {
        DrawPixel(&stPoint);
    }

    stPoint.usX = pstRectangle_->usRight;
    for (stPoint.usY = pstRectangle_->usTop; stPoint.usY <=
pstRectangle_->usBottom; stPoint.usY++)
    {
        DrawPixel(&stPoint);
    }
}

//---------------------------------------------------------------------------
void GraphicsDriver::Circle(DrawCircle_t *
pstCircle_)
{
    DrawPoint_t stPoint;
    K_SHORT sX;
    K_SHORT sY;
    K_ULONG ulRadSquare;

    K_ULONG ulXSquare;
    K_ULONG ulYSquare;

    // Get the radius squared value...
    ulRadSquare = (K_ULONG)pstCircle_->usRadius;
    ulRadSquare *= ulRadSquare;

    // Look at the upper-right quarter of the circle
    for (sX = 0; sX <= (K_SHORT)pstCircle_->usRadius; sX++)
    {
        ulXSquare = (K_ULONG)sX;
        ulXSquare *= ulXSquare;
        for (sY = 0; sY <= (K_SHORT)pstCircle_->usRadius; sY++)
        {
            ulYSquare = (K_ULONG)sY;
            ulYSquare *= ulYSquare;

            // if filled...
            if (pstCircle_->bFill == true)
            {
                stPoint.uColor = pstCircle_->uFillColor;
                if (ulXSquare + ulYSquare <= ulRadSquare)
                {
                    // Draw the fill color at the appropriate locations
 (quadrature...)
                    stPoint.usX = pstCircle_->usX + sX;
                    stPoint.usY = pstCircle_->usY + sY;
                    DrawPixel(&stPoint);
                    stPoint.usX = pstCircle_->usX - sX;
                    stPoint.usY = pstCircle_->usY + sY;
                    DrawPixel(&stPoint);
                    stPoint.usX = pstCircle_->usX + sX;
                    stPoint.usY = pstCircle_->usY - sY;
                    DrawPixel(&stPoint);
                    stPoint.usX = pstCircle_->usX - sX;
                    stPoint.usY = pstCircle_->usY - sY;
                    DrawPixel(&stPoint);
                }
            }
            // Check for edge...
            if (
                ((ulXSquare + ulYSquare) >= (ulRadSquare-pstCircle_->usRadius
))  &&
                ((ulXSquare + ulYSquare) <= (ulRadSquare+pstCircle_->usRadius
))
               )
            {
                stPoint.uColor = pstCircle_->uLineColor;

                // Draw the fill color at the appropriate locations
 (quadrature...)
                stPoint.usX = pstCircle_->usX + sX;
                stPoint.usY = pstCircle_->usY + sY;
                DrawPixel(&stPoint);
                stPoint.usX = pstCircle_->usX - sX;
                stPoint.usY = pstCircle_->usY + sY;
                DrawPixel(&stPoint);
                stPoint.usX = pstCircle_->usX + sX;
                stPoint.usY = pstCircle_->usY - sY;
                DrawPixel(&stPoint);
                stPoint.usX = pstCircle_->usX - sX;
                stPoint.usY = pstCircle_->usY - sY;
                DrawPixel(&stPoint);
            }
        }
    }
}

//---------------------------------------------------------------------------
void GraphicsDriver::Ellipse(DrawEllipse_t
*pstEllipse_)
{
    DrawPoint_t stPoint;
    K_SHORT sX;
    K_SHORT sY;
    K_ULONG ulRadius;
    K_ULONG ulHSquare;
    K_ULONG ulVSquare;
    K_ULONG ulXSquare;
    K_ULONG ulYSquare;

    ulHSquare = (K_ULONG)pstEllipse_->usWidth;
    ulHSquare *= ulHSquare;

    ulVSquare = (K_ULONG)pstEllipse_->usHeight;
    ulVSquare *= ulVSquare;

    ulRadius = ulHSquare * ulVSquare;

    for (sX = 0; sX <= (K_SHORT)pstEllipse_->usWidth; sX++)
    {
        ulXSquare = (K_ULONG)sX;
        ulXSquare *= ulXSquare;
        ulXSquare *= ulHSquare;

        for (sY = 0; sY <= (K_SHORT)pstEllipse_->usHeight; sY++)
        {
            ulYSquare = (K_ULONG)sY;
            ulYSquare *= ulYSquare;
            ulYSquare *= ulVSquare;

            if ((ulXSquare + ulYSquare) <= ulRadius)
            {
                // Draw the fill color at the appropriate locations
 (quadrature...)
                stPoint.usX = pstEllipse_->usX + sX;
                stPoint.usY = pstEllipse_->usY + sY;
                DrawPixel(&stPoint);
                stPoint.usX = pstEllipse_->usX - sX;
                stPoint.usY = pstEllipse_->usY + sY;
                DrawPixel(&stPoint);
                stPoint.usX = pstEllipse_->usX + sX;
                stPoint.usY = pstEllipse_->usY - sY;
                DrawPixel(&stPoint);
                stPoint.usX = pstEllipse_->usX - sX;
                stPoint.usY = pstEllipse_->usY - sY;
                DrawPixel(&stPoint);
            }
        }
    }
}

//---------------------------------------------------------------------------
void GraphicsDriver::Bitmap(DrawBitmap_t *
pstBitmap_)
{
    K_USHORT usRow;
    K_USHORT usCol;

    K_USHORT usIndex;

    K_UCHAR ucRed = 0;
    K_UCHAR ucBlue = 0;
    K_UCHAR ucGreen = 0;

    DrawPoint_t stPoint;

    usIndex = 0;
    for (usRow = pstBitmap_->usY; usRow < (pstBitmap_->usY + pstBitmap_->
usHeight); usRow++)
    {
        for (usCol = pstBitmap_->usX; usCol < (pstBitmap_->usX +
pstBitmap_->usWidth); usCol++)
        {

            stPoint.usX = usCol;
            stPoint.usY = usRow;

            // Build the color based on the bitmap value...  This algorithm
            // is slow, but it automatically converts any 8/16/24 bit bitmap
 into the
            // current colorspace defined...
            switch(pstBitmap_->ucBPP)
            {
                case 1:
                {
                    // 3:2:3, RGB
                    ucRed     = ((pstBitmap_->pucData[usIndex]) & 0xE0)
<< 1;
```

```
                    ucGreen   = ((pstBitmap_->pucData[usIndex]) & 0x18) << 3;
                    ucBlue    = ((pstBitmap_->pucData[usIndex]) & 0x07) << 5;
                }
                    break;
                case 2:
                {
                    K_USHORT usTemp;
                    usTemp = pstBitmap_->pucData[usIndex];
                    usTemp <<= 8;
                    usTemp |= pstBitmap_->pucData[usIndex + 1];

                    // 5:6:5, RGB
                    ucRed     = (K_UCHAR)((usTemp >> 11) & 0x001F) << 3;
                    ucGreen   = (K_UCHAR)((usTemp >> 5) & 0x003F)  << 2;
                    ucBlue    = (K_UCHAR)(usTemp & 0x001F) << 3;
                }
                    break;
                case 3:
                {
                    K_ULONG ulTemp;
                    ulTemp = pstBitmap_->pucData[usIndex];
                    ulTemp <<= 8;
                    ulTemp |= pstBitmap_->pucData[usIndex + 1];
                    ulTemp <<= 8;
                    ulTemp |= pstBitmap_->pucData[usIndex + 2];

                    // 8:8:8 RGB
                    ucRed    = (K_UCHAR)((ulTemp & 0x00FF0000) >> 16);
                    ucGreen  = (K_UCHAR)((ulTemp & 0x0000FF00) >> 8);
                    ucBlue   = (K_UCHAR)((ulTemp & 0x000000FF));
                }
                    break;
                default:
                    break;
            }

            // Convert the R,G,B values into the correct colorspace for display
#if DRAW_COLOR_2BIT
            //1-bit
            ucRed >>= 7;
            ucGreen >>= 7;
            ucBlue >>= 7;
#elif DRAW_COLOR_8BIT
            //3:2:3 R:G:B
            ucRed >>= 5;
            ucGreen >>= 6;
            ucBlue >>= 5;
#elif DRAW_COLOR_16BIT
            //5:6:5 R:G:B
            ucRed >>= 3;
            ucGreen >>= 2;
            ucBlue >>= 3;
#elif DRAW_COLOR_24BIT
            // No conversion required
#endif
            // Build the color.
            stPoint.uColor = RGB_COLOR(ucRed,ucGreen,ucBlue);

            // Draw the point.
            DrawPixel(&stPoint);

            // Stamps are opaque, don't fill in the BG
            usIndex += m_ucBPP / 8;
        }
    }
}

//---------------------------------------------------------------------------
void GraphicsDriver::Stamp(DrawStamp_t * pstStamp_)
{
    K_USHORT usRow;
    K_USHORT usCol;
    K_USHORT usShift;
    K_USHORT usIndex;
    DrawPoint_t stPoint;

    usIndex = 0;
    for (usRow = pstStamp_->usY; usRow < (pstStamp_->usY + pstStamp_->usHeight); usRow++)
    {
        usShift = 0x80;
        for (usCol = pstStamp_->usX; usCol < (pstStamp_->usX + pstStamp_->usWidth); usCol++)
        {
            // If the packed bit in the bitmap is a "1", draw the color.
            if (pstStamp_->pucData[usIndex] & usShift)
            {
                stPoint.usX = usCol;
                stPoint.usY = usRow;
                stPoint.uColor = pstStamp_->uColor;
                DrawPixel(&stPoint);
            }
            // Stamps are opaque, don't fill in the BG

            // Shift to the next bit in the field
            usShift >>= 1;

            // Rollover - next bit in the bitmap.
            // This obviously works best for stamps that are multiples of 8x8
            if (usShift == 0)
            {
                usShift = 0x80;
                usIndex++;
            }
        }
    }
}

//---------------------------------------------------------------------------
void GraphicsDriver::Move( DrawMove_t *pstMove_ )
{
    DrawPoint_t stPoint;
    K_LONG sX;
    K_LONG sY;
    K_LONG sXInc = 0;
    K_LONG sYInc = 0;

    K_BOOL bLeftToRight = false;
    K_BOOL bTopToBottom = false;

    if (pstMove_->usSrcX > pstMove_->usDstX)
    {
        bLeftToRight = true;
    }
    if (pstMove_->usSrcY > pstMove_->usDstY)
    {
        bTopToBottom = true;
    }

    if (bLeftToRight)
    {
        sXInc++;
    }
    else
    {
        sXInc--;
        pstMove_->usSrcX += pstMove_->usCopyWidth - 1;
        pstMove_->usDstX += pstMove_->usCopyWidth - 1;
    }

    if (bTopToBottom)
    {
        sYInc++;
    }
    else
    {
        sYInc--;
        pstMove_->usSrcY += pstMove_->usCopyHeight - 1;
        pstMove_->usDstY += pstMove_->usCopyHeight - 1;
    }

    // Hideously inefficient memory move...
    for (sX = 0; sX < pstMove_->usCopyWidth; sX++)
    {
        for (sY = 0; sY < pstMove_->usCopyHeight; sY++)
        {
            // Read from source (value read into the point struct)
            stPoint.usY = (K_USHORT)((K_LONG)pstMove_->usSrcY + ((
K_LONG)sY * sYInc));
            stPoint.usX = (K_USHORT)((K_LONG)pstMove_->usSrcX + ((
K_LONG)sX * sXInc));
            ReadPixel(&stPoint);

            // Copy to dest
            stPoint.usY = (K_USHORT)((K_LONG)pstMove_->usDstY + ((
K_LONG)sY * sYInc));
            stPoint.usX = (K_USHORT)((K_LONG)pstMove_->usDstX + ((
K_LONG)sX * sXInc));
            DrawPixel(&stPoint);
        }
    }
```

```
}

//---------------------------------------------------------------------------
void GraphicsDriver::Text(DrawText_t *pstText_)
{
    K_USHORT usX, usY;
    K_USHORT usStartX;
    K_USHORT usStartY;
    K_USHORT usCharOffsetX;
    K_USHORT usCharIndex = 0;
    K_UCHAR *pucData = (K_UCHAR*)pstText_->pstFont->pucFontData;
    DrawPoint_t stPoint;

    // set the color for this element.
    stPoint.uColor = pstText_->uColor;

    usCharOffsetX = 0;

    // Draw every character in the string, one at a time
    while (pstText_->pcString[usCharIndex] != 0)
    {
        K_USHORT usOffset = 0;

        K_UCHAR ucWidth;
        K_UCHAR ucHeight;
        K_UCHAR ucVOffset;
        K_UCHAR ucBitmask;

        // Read the glyphs from memory until we arrive at the one we wish to
 print
        for (usX = 0; usX < pstText_->pcString[usCharIndex]; usX++)
        {
            // Glyphs are variable-sized for efficiency - to look up a
 particular
            // glyph, we must traverse all preceding glyphs in the list
            ucWidth  = Font_ReadByte(usOffset, pucData);
            ucHeight = Font_ReadByte(usOffset + 1, pucData);

            // Adjust the offset to point to the next glyph
            usOffset += ((((K_USHORT)ucWidth + 7) >> 3) * (K_USHORT)ucHeight)
                        + (sizeof(Glyph_t) - 1);
        }

        // Header information:  glyph size and vertical offset
        ucWidth   = Font_ReadByte(usOffset++, pucData);
        ucHeight  = Font_ReadByte(usOffset++, pucData);
        ucVOffset = Font_ReadByte(usOffset++, pucData);

        usStartY = pstText_->usTop + (K_USHORT)ucVOffset;
        usStartX = pstText_->usLeft;

        // Draw the font from left->right, top->bottom
        for (   usY = usStartY;
                usY < usStartY + (K_USHORT)ucHeight;
                usY++ )
        {
            K_UCHAR ucTempChar = Font_ReadByte(usOffset, pucData);
            ucBitmask = 0x80;

            for (   usX = usCharOffsetX + usStartX;
                    usX < usCharOffsetX + usStartX + (K_USHORT)ucWidth;
                    usX++ )
            {
                if (!ucBitmask)
                {
                    ucBitmask = 0x80;
                    usOffset++;
                    ucTempChar = Font_ReadByte(usOffset, pucData);
                }

                if (ucTempChar & ucBitmask)
                {
                    // Update the location
                    stPoint.usX = usX;
                    stPoint.usY = usY;

                    // Draw the point.
                    DrawPixel(&stPoint);
                }

                ucBitmask >>= 1;
            }

            usOffset++;
        }

        // Next character
        usCharIndex++;
        usCharOffsetX += (K_USHORT)ucWidth + 1;
    }
}

//---------------------------------------------------------------------------
K_USHORT GraphicsDriver::TextWidth(DrawText_t *pstText_)
{
    K_USHORT usCharOffsetX;
    K_USHORT usCharIndex = 0;
    K_USHORT usX;
    K_UCHAR *pucData = (K_UCHAR*)pstText_->pstFont->pucFontData;

    usCharOffsetX = 0;

    // Draw every character in the string, one at a time
    while (pstText_->pcString[usCharIndex] != 0)
    {
        K_USHORT usOffset = 0;

        K_UCHAR ucWidth;
        K_UCHAR ucHeight;

        // Read the glyphs from memory until we arrive at the one we wish to
 print
        for (usX = 0; usX < pstText_->pcString[usCharIndex]; usX++)
        {
            // Glyphs are variable-sized for efficiency - to look up a
 particular
            // glyph, we must traverse all preceding glyphs in the list
            ucWidth  = Font_ReadByte(usOffset, pucData);
            ucHeight = Font_ReadByte(usOffset + 1, pucData);

            // Adjust the offset to point to the next glyph
            usOffset += ((((K_USHORT)ucWidth + 7) >> 3) * (K_USHORT)ucHeight)
                        + (sizeof(Glyph_t) - 1);
        }

        // Header information:  glyph size and vertical offset
        ucWidth   = Font_ReadByte(usOffset, pucData);
        usOffset += (sizeof(Glyph_t) - 1);

        // Next character
        usCharIndex++;
        usCharOffsetX += (K_USHORT)ucWidth + 1;
    }

    return usCharOffsetX;
}

//---------------------------------------------------------------------------
void GraphicsDriver::TriangleWire(DrawPoly_t
 *pstPoly_)
{
    DrawLine_t stLine;

    stLine.uColor = pstPoly_->uColor;

    stLine.usX1 = pstPoly_->pstVector[0].usX;
    stLine.usY1 = pstPoly_->pstVector[0].usY;
    stLine.usX2 = pstPoly_->pstVector[1].usX;
    stLine.usY2 = pstPoly_->pstVector[1].usY;
    Line(&stLine);

    stLine.usX1 = pstPoly_->pstVector[1].usX;
    stLine.usY1 = pstPoly_->pstVector[1].usY;
    stLine.usX2 = pstPoly_->pstVector[2].usX;
    stLine.usY2 = pstPoly_->pstVector[2].usY;
    Line(&stLine);

    stLine.usX1 = pstPoly_->pstVector[2].usX;
    stLine.usY1 = pstPoly_->pstVector[2].usY;
    stLine.usX2 = pstPoly_->pstVector[0].usX;
    stLine.usY2 = pstPoly_->pstVector[0].usY;
    Line(&stLine);
}
//---------------------------------------------------------------------------
void GraphicsDriver::TriangleFill(DrawPoly_t
 *pstPoly_)
{
    // Drawing a raster-filled triangle:
    K_UCHAR ucMaxEdge = 0;
    K_UCHAR ucMinEdge1 = 0, ucMinEdge2 = 0;
    K_SHORT sMax = 0;
    K_SHORT sTemp;

    K_SHORT sDeltaX1, sDeltaX2;
    K_SHORT sDeltaY1, sDeltaY2;
    K_CHAR cStepX1, cStepX2;
    K_CHAR cStepY;
    K_SHORT sX1, sX2, sX3, sY1, sY2, sY3;
    K_SHORT sTempX1, sTempY1, sTempX2, sTempY2;
    K_SHORT sFraction1;
    K_SHORT sFraction2;
    K_SHORT i;
    DrawPoint_t stPoint;

    // Figure out which line segment is the longest
    sTemp = (K_SHORT)pstPoly_->pstVector[0].usY - (K_SHORT)pstPoly_->
pstVector[1].usY;
    if( sTemp < 0 )     { sTemp = -sTemp; }
    if( sTemp > sMax ) { sMax  = sTemp; ucMaxEdge = 0; ucMinEdge1 = 1;
ucMinEdge2 = 2;}

    sTemp = (K_SHORT)pstPoly_->pstVector[1].usY - (K_SHORT)pstPoly_->
pstVector[2].usY;
    if( sTemp < 0 )     { sTemp = -sTemp; }
    if( sTemp > sMax ) { sMax  = sTemp; ucMaxEdge = 1; ucMinEdge1 = 2;
ucMinEdge2 = 0; }

    sTemp = (K_SHORT)pstPoly_->pstVector[2].usY - (K_SHORT)pstPoly_->
pstVector[0].usY;
    if( sTemp < 0 )     { sTemp = -sTemp; }
    if( sTemp > sMax ) { sMax  = sTemp; ucMaxEdge = 2; ucMinEdge1 = 0;
ucMinEdge2 = 1;}

    // Label the vectors and copy into temporary signed buffers
    sX1 = (K_SHORT)pstPoly_->pstVector[ucMaxEdge].usX;
    sX2 = (K_SHORT)pstPoly_->pstVector[ucMinEdge1].usX;
    sX3 = (K_SHORT)pstPoly_->pstVector[ucMinEdge2].usX;

    sY1 = (K_SHORT)pstPoly_->pstVector[ucMaxEdge].usY;
    sY2 = (K_SHORT)pstPoly_->pstVector[ucMinEdge1].usY;
    sY3 = (K_SHORT)pstPoly_->pstVector[ucMinEdge2].usY;

    // Figure out whether or not we're drawing up-down or down-up
    sDeltaY1 = sY1 - sY2;
    if (sDeltaY1 < 0) { cStepY = -1; sDeltaY1 = -sDeltaY1; } else { cStepY = 1;
 }

    sDeltaX1 = sX1 - sX2;
    if (sDeltaX1 < 0) { cStepX1 = -1; sDeltaX1 = -sDeltaX1; } else { cStepX1 =
1; }

    sDeltaY2 = sY1 - sY3;
    if (sDeltaY2 < 0) { cStepY = -1; sDeltaY2 = -sDeltaY2; } else { cStepY = 1;
 }

    sDeltaX2 = sX1 - sX3;
    if (sDeltaX2 < 0) { cStepX2 = -1; sDeltaX2 = -sDeltaX2; } else { cStepX2 =
1; }

    sDeltaX1 <<=1;
    sDeltaX2 <<=1;
    sDeltaY1 <<=1;
    sDeltaY2 <<=1;

    sFraction1 = sDeltaX1;// - (sDeltaY1 >> 1);
    sFraction2 = sDeltaX2;// - (sDeltaY2 >> 1);

    sTempY1 = sY1;
    sTempY2 = sY1;
    sTempX1 = sX1;
    sTempX2 = sX1;

    stPoint.uColor = pstPoly_->uColor;

    if( sDeltaY2 != 0 )
    {
        while (sTempY2 != sY3)
        {
            stPoint.usY = sTempY2;
            if( sTempX1 < sTempX2 ) {
                for( i = sTempX1; i <= sTempX2; i++) {
                    stPoint.usX = i;
                    Point(&stPoint);
                }
            } else {
                for( i = sTempX2; i <= sTempX1; i++ ) {
                    stPoint.usX = i;
                    Point(&stPoint);
                }
            }

            while (sFraction2 >= sDeltaY2)
            {
                sTempX2 -= cStepX2;
                sFraction2 -= sDeltaY2;
            }
            sTempY2 -= cStepY;
            sFraction2 += sDeltaX2;

            while (sFraction1 >= sDeltaY1)
            {
                sTempX1 -= cStepX1;
                sFraction1 -= sDeltaY1;
            }
            sTempY1 -= cStepY;
            sFraction1 += sDeltaX1;
        }
    }

    sDeltaY2 = sY3 - sY2;
    sDeltaX2 = sX3 - sX2;

    if (sDeltaX2 < 0) { cStepX2 = -1; sDeltaX2 = -sDeltaX2; } else { cStepX2 =
1; }
    if (sDeltaY2 < 0) { cStepY = -1;  sDeltaY2 = -sDeltaY2; } else { cStepY  =
1; }

    sDeltaX2 <<=1;
    sDeltaY2 <<=1;

    sFraction2 = sDeltaX2; // - (sDeltaY2 >> 1);

    sTempY2 = sY3;
    sTempX2 = sX3;

    if( sDeltaY2 != 0)
    {
        while (sTempY2 != sY2)
        {
            stPoint.usY = sTempY2;
            if( sTempX1 < sTempX2 ) {
                for( i = sTempX1; i <= sTempX2; i++) {
                    stPoint.usX = i;
                    Point(&stPoint);
                }
            } else {
                for( i = sTempX2; i <= sTempX1; i++ ) {
                    stPoint.usX = i;
                    Point(&stPoint);
                }
            }

            while (sFraction2 >= sDeltaY2)
            {
                sTempX2 -= cStepX2;
                sFraction2 -= sDeltaY2;
            }
            sTempY2 -= cStepY;
            sFraction2 += sDeltaX2;

            while (sFraction1 >= sDeltaY1)
            {
                sTempX1 -= cStepX1;
                sFraction1 -= sDeltaY1;
            }
            sTempY1 -= cStepY;
            sFraction1 += sDeltaX1;
        }
    }
}

//---------------------------------------------------------------------------
void GraphicsDriver::Polygon(DrawPoly_t *pstPoly_)
{
    K_USHORT i,j,k;
    K_BOOL bState = false;

    DrawPoly_t   stTempPoly;
    DrawVector_t astTempVec[3];

    if (pstPoly_->usNumPoints < 3)
    {
        return;
    }

    stTempPoly.uColor = pstPoly_->uColor;
    stTempPoly.bFill = pstPoly_->bFill;
    stTempPoly.pstVector = astTempVec;
    stTempPoly.usNumPoints = 3;

    astTempVec[0].usX = pstPoly_->pstVector[0].usX;
    astTempVec[1].usX = pstPoly_->pstVector[1].usX;
    astTempVec[0].usY = pstPoly_->pstVector[0].usY;
    astTempVec[1].usY = pstPoly_->pstVector[1].usY;

    j = 2;
    astTempVec[2].usX = pstPoly_->pstVector[pstPoly_->usNumPoints
 - 1].usX;
    astTempVec[2].usY = pstPoly_->pstVector[pstPoly_->usNumPoints
 - 1].usY;

    k = pstPoly_->usNumPoints - 2;

    if( pstPoly_->bFill )
    {
        TriangleFill(&stTempPoly);
    }
    else
    {
        TriangleWire(&stTempPoly);
    }

    // Filled polygon/wireframe polygon using triangle decomp.
    for(i = 0; i < pstPoly_->usNumPoints - 3; i++)
    {
        astTempVec[0].usX = astTempVec[1].usX;
        astTempVec[1].usX = astTempVec[2].usX;
        astTempVec[0].usY = astTempVec[1].usY;
        astTempVec[1].usY = astTempVec[2].usY;

        if( !bState )
        {
            bState = true;
            astTempVec[2].usX = pstPoly_->pstVector[j].usX;
            astTempVec[2].usY = pstPoly_->pstVector[j].usY;
            j++;
        }
        else
        {
            bState = false;
            astTempVec[2].usX = pstPoly_->pstVector[k].usX;
            astTempVec[2].usY = pstPoly_->pstVector[k].usY;
            k--;
        }
        if( pstPoly_->bFill )
        {
            TriangleFill(&stTempPoly);
        }
        else
        {
            TriangleWire(&stTempPoly);
        }
    }
}

//---------------------------------------------------------------------------
void GraphicsDriver::SetWindow(DrawWindow_t
 *pstWindow_)
{
    if ((pstWindow_->usLeft <= pstWindow_->usRight) &&
        (pstWindow_->usRight < m_usResX) &&
        (pstWindow_->usLeft < m_usResX))
    {
        m_usLeft = pstWindow_->usLeft;
        m_usRight = pstWindow_->usRight;
    }

    if ((pstWindow_->usTop <= pstWindow_->usBottom) &&
        (pstWindow_->usTop < m_usTop)  &&
        (pstWindow_->usBottom < m_usBottom))
    {
        m_usTop = pstWindow_->usTop;
        m_usBottom = pstWindow_->usBottom;
    }

}

//---------------------------------------------------------------------------
void GraphicsDriver::ClearWindow()
{
    m_usLeft = 0;
    m_usTop = 0;
    m_usRight = m_usResX - 1;
    m_usBottom = m_usResY - 1;
}
```

14.47 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/graphics.h File Reference

Graphics driver class declaration.

```
#include "driver.h"
#include "draw.h"
```

Classes

- class GraphicsDriver

  Defines the base graphics driver class, which is inherited by all other graphics drivers.

14.47.1 Detailed Description

Graphics driver class declaration.

Definition in file graphics.h.

14.48 graphics.h

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#ifndef __GRAPHICSX_H__
#define __GRAPHICSX_H__

#include "driver.h"
#include "draw.h"

//---------------------------------------------------------------------------
class GraphicsDriver : public Driver
{
public:
//---------------------------------------------------------------------------
/*
    The base graphics driver does not implement the set of
    virtual methods inherited from the Driver class.  This
    is left to the actual hardware implementation.
*/
//---------------------------------------------------------------------------

    virtual void DrawPixel(DrawPoint_t *pstPoint_) {};

    virtual void ReadPixel(DrawPoint_t *pstPoint_) {};

//---------------------------------------------------------------------------
/*
    Raster operations defined using per-pixel rendering.
    Can be overridden in inheriting classes.
*/
//---------------------------------------------------------------------------
    virtual void ClearScreen();

    virtual void Point(DrawPoint_t *pstPoint_);

    virtual void Line(DrawLine_t *pstLine_);

    virtual void Rectangle(DrawRectangle_t *
pstRectangle_);
```

```

    virtual void Circle(DrawCircle_t *pstCircle_);

    virtual void Ellipse(DrawEllipse_t *pstEllipse_);

    virtual void Bitmap(DrawBitmap_t *pstBitmap_);

    virtual void Stamp(DrawStamp_t *pstStamp_);

    virtual void Move(DrawMove_t *pstMove_ );

    virtual void TriangleWire(DrawPoly_t *pstPoly_);

    virtual void TriangleFill(DrawPoly_t *pstPoly_);

    virtual void Polygon(DrawPoly_t *pstPoly_);

    virtual void Text(DrawText_t *pstText_);

    virtual K_USHORT TextWidth(DrawText_t *pstText_);

    void SetWindow( DrawWindow_t *pstWindow_ );

    void ClearWindow();
protected:

    K_USHORT m_usResX;
    K_USHORT m_usResY;

    K_USHORT m_usLeft;
    K_USHORT m_usTop;
    K_USHORT m_usRight;
    K_USHORT m_usBottom;

    K_UCHAR m_ucBPP;
};

#endif

```

14.49 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/gui.cpp File Reference

Graphical User Interface classes and data structure definitions.

```
#include "message.h"
#include "kerneltypes.h"
#include "gui.h"
#include "system_heap.h"
#include "fixed_heap.h"
#include "memutil.h"
#include <stdio.h>
```

14.49.1 Detailed Description

Graphical User Interface classes and data structure definitions.

Definition in file gui.cpp.

14.50 gui.cpp

```
/*===========================================================================
     _____     _____     _____     _____
 ___|    _|__ __|_    |__ __|__   |__ __| __  |__  _____
|    \  /  | ||    \     ||     |     ||  |/ /     ||___   |
|     \/   | ||     \    ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||_______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]-----------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "message.h"
#include "kerneltypes.h"
#include "gui.h"
#include "system_heap.h"
#include "fixed_heap.h"
#include "memutil.h"

#include <stdio.h>
//---------------------------------------------------------------------------
void GuiWindow::AddControl( GuiControl *
pclControl_, GuiControl *pclParent_ )
{
    GUI_DEBUG_PRINT("GuiWindow::AddControl\n");

    m_clControlList.Add(static_cast<LinkListNode*>(
pclControl_));
    m_pclInFocus = pclControl_;
    m_ucControlCount++;

    pclControl_->SetParentWindow(this);
    pclControl_->SetParentControl(pclParent_);
}

//---------------------------------------------------------------------------
void GuiWindow::RemoveControl( GuiControl *
pclControl_ )
{
    GUI_DEBUG_PRINT("GuiWindow::RemoveControl\n");

    if (pclControl_->GetPrev())
    {
        m_pclInFocus = static_cast<GuiControl*>(
pclControl_->GetPrev());
    }
    else if (pclControl_->GetNext())
    {
        m_pclInFocus = static_cast<GuiControl*>(
pclControl_->GetNext());
    }
    else
    {
        m_pclInFocus = NULL;
    }
    m_clControlList.Remove(static_cast<LinkListNode*>(
pclControl_));
    m_ucControlCount--;
}

//---------------------------------------------------------------------------
K_UCHAR GuiWindow::GetMaxZOrder()
{
    GUI_DEBUG_PRINT("GuiWindow::GetMaxZOrder\n");

    LinkListNode *pclTempNode;
    K_UCHAR ucZ = 0;
    K_UCHAR ucTempZ;

    pclTempNode = m_clControlList.GetHead();

    while (pclTempNode)
    {
        ucTempZ = (static_cast<GuiControl*>(pclTempNode))->GetZOrder(
);
        if (ucTempZ > ucZ)
        {
            ucZ = ucTempZ;
        }
        pclTempNode = pclTempNode->GetNext();
    }

    return ucZ;
}

//---------------------------------------------------------------------------
void GuiWindow::Redraw( K_BOOL bRedrawAll_ )
{
    GUI_DEBUG_PRINT("GuiWindow::Redraw\n");

    K_UCHAR ucControlsLeft = m_ucControlCount;
    K_UCHAR ucCurrentZ = 0;
    K_UCHAR ucMaxZ;

    ucMaxZ = GetMaxZOrder();

    // While there are still controls left to process (and we're less than
    // the maximum Z-order, just a sanity check.), redraw each object that
    // has its stale flag set, or all controls if the bRedrawAll_ parameter
    // is true.
    while (ucControlsLeft && (ucCurrentZ <= ucMaxZ))
    {
        LinkListNode *pclTempNode;

        pclTempNode = m_clControlList.GetHead();
        while (pclTempNode)
        {
            GuiControl* pclTempControl = static_cast<GuiControl*>(pclTempNode);
            if (pclTempControl->GetZOrder() == ucCurrentZ)
            {
                if ((bRedrawAll_) || (pclTempControl->IsStale()))
                {
                    pclTempControl->Draw();
                    pclTempControl->ClearStale();
                }

                ucControlsLeft--;
            }

            pclTempNode = pclTempNode->GetNext();
        }
        ucCurrentZ++;
    }
    GUI_DEBUG_PRINT("  Current Z: %d\n", ucCurrentZ);
    GUI_DEBUG_PRINT("  Controls Left: %d\n", ucControlsLeft);
}

//---------------------------------------------------------------------------
void GuiWindow::ProcessEvent( GuiEvent_t * pstEvent_ )
{
    GUI_DEBUG_PRINT("GuiWindow::ProcessEvent\n");

    // If the event is for broadcast - send it to all controls,
    // without regard to order.
    if ((TARGET_ID_BROADCAST == pstEvent_->ucTargetID)
        || (TARGET_ID_BROADCAST_Z == pstEvent_->ucTargetID))
    {
        GUI_DEBUG_PRINT("  TARGET_ID_BROADCAST(_Z)\n");

        LinkListNode *pclTempNode;
        pclTempNode = m_clControlList.GetHead();

        while (pclTempNode)
        {
            GuiReturn_t eRet;
            eRet = (static_cast<GuiControl*>(pclTempNode))->ProcessEvent(pstEvent_);
            if (GUI_EVENT_CONSUMED == eRet)
            {
                break;
            }
            pclTempNode = pclTempNode->GetNext();
        }
    }
    // Send the event only to the currently-selected object.
    else if (TARGET_ID_FOCUS == pstEvent_->ucTargetID)
    {
        GUI_DEBUG_PRINT("  TARGET_ID_FOCUS\n");
        GuiReturn_t eReturn = GUI_EVENT_OK;

        // Try to let the control process the event on its own
        if (m_pclInFocus)
        {
            eReturn = m_pclInFocus->ProcessEvent(pstEvent_);
        }

        // If the event was not consumed, use default logic to process the event
        if (GUI_EVENT_CONSUMED != eReturn)
        {
            if (EVENT_TYPE_KEYBOARD == pstEvent_->ucEventType)
            {
                if (KEYCODE_TAB == pstEvent_->stKey.ucKeyCode)
                {
                    if (pstEvent_->stKey.bKeyState)
                    {
                        CycleFocus(true);
                    }
                }
            }
            else if (EVENT_TYPE_JOYSTICK == pstEvent_->ucEventType)
            {
                if (pstEvent_->stJoystick.bUp || pstEvent_->stJoystick.bLeft)
                {
                    // Cycle focus *backwards*
                    CycleFocus(false);
                }
                else if (pstEvent_->stJoystick.bRight || pstEvent_->stJoystick.bDown)
                {
                    // Cycle focus *forewards*
                    CycleFocus(true);
                }
            }
        }
    }
    else if (TARGET_ID_HIGH_Z == pstEvent_->ucTargetID)
    {
        GUI_DEBUG_PRINT("  TARGET_ID_HIGH_Z\n");

        K_USHORT usTargetX, usTargetY;
        K_USHORT usOffsetX, usOffsetY;
        K_UCHAR ucMaxZ = 0;

        LinkListNode *pclTempNode;
        pclTempNode = m_clControlList.GetHead();

        switch (pstEvent_->ucEventType)
        {
            case EVENT_TYPE_MOUSE:
            case EVENT_TYPE_TOUCH:
            {
                GuiControl *pclTargetControl = NULL;

                // Read the target X/Y coordinates out of the event struct
                if (EVENT_TYPE_TOUCH == pstEvent_->ucEventType)
                {
                    usTargetX = pstEvent_->stTouch.usX;
                    usTargetY = pstEvent_->stTouch.usY;
                }
                else
                {
                    usTargetX = pstEvent_->stMouse.usX;
                    usTargetY = pstEvent_->stMouse.usY;
                }

                // Go through every control on the window, checking to see if the
                // event falls within the bounding box
                while (pclTempNode)
                {
                    GuiReturn_t eRet;
                    GuiControl *pclControl = (static_cast<GuiControl*>(pclTempNode));

                    pclControl->GetControlOffset(&usOffsetX, &usOffsetY);

                    // Compare event coordinates to bounding box (with offsets)
                    if ( ((usTargetX >= (usOffsetX + pclControl->GetLeft()) &&
                          (usTargetX <= (usOffsetX + pclControl->GetLeft() + pclControl->GetWidth() - 1)))) &&
                         ((usTargetY >= (usOffsetY + pclControl->GetTop()) &&
                          (usTargetY <= (usOffsetY + pclControl->GetTop() + pclControl->GetHeight() - 1)))) )
                    {
                        // If this control is higher in Z-Order, set this as the newest
                        // candidate control to accept the event
                        if (pclControl->GetZOrder() >= ucMaxZ)
                        {
                            pclTargetControl = pclControl;
                            ucMaxZ = pclControl->GetZOrder();
                        }
                    }

                    pclTempNode = pclTempNode->GetNext();
                }

                // If a suitable control was found on the event surface, pass the event off
                // for processing.
                if (pclTargetControl)
                {
                    // If the selected control is different from the current in-focus
                    // control, then deactive that control.
                    if (m_pclInFocus && (m_pclInFocus != pclTargetControl))
                    {
                        m_pclInFocus->Activate(false);
                        m_pclInFocus = NULL;
                    }
                    (static_cast<GuiControl*>(pclTargetControl))->ProcessEvent(pstEvent_);
                }
            }
                break;
            default:
                break;
        }
    }
}
//---------------------------------------------------------------------------
void GuiWindow::SetFocus( GuiControl *pclControl_ )
{
    GUI_DEBUG_PRINT("GuiWindow::SetFocus\n");

    m_pclInFocus = pclControl_;
}

//---------------------------------------------------------------------------
void GuiWindow::CycleFocus( bool bForward_ )
{
    GUI_DEBUG_PRINT("GuiWindow::CycleFocus\n");

    // Set starting point and cached copy of current nodes
    LinkListNode *pclTempNode = static_cast<GuiControl*>(m_clControlList.GetHead());
    LinkListNode *pclStartNode = m_pclInFocus;

    if (bForward_)
    {
        // If there isn't a current focus node, set the focus to the beginning
        // of the list
        if (!m_pclInFocus)
        {
            m_pclInFocus = static_cast<GuiControl*>(pclTempNode);
            if (!m_pclInFocus)
            {
                return;
            }
            pclTempNode = static_cast<GuiControl*>(m_pclInFocus);
            pclStartNode = NULL;
        }
        else
        {
            // Deactivate the control that's losing focus
            static_cast<GuiControl*>(m_pclInFocus)->Activate(false);

            // Otherwise start with the next node
            pclStartNode = pclStartNode->GetNext();
        }

        // Go through the whole control list and find the next one to accept
        // the focus
        while (pclTempNode && pclTempNode != pclStartNode)
        {
            if (static_cast<GuiControl*>(pclTempNode)->AcceptsFocus())
            {
                m_pclInFocus = static_cast<GuiControl*>(pclTempNode);
                m_pclInFocus->Activate(true);
                SetFocus(m_pclInFocus);
                return;
            }
            pclTempNode = pclTempNode->GetNext();
        }

        pclTempNode = static_cast<GuiControl*>(m_clControlList.GetHead());
        while (pclTempNode && pclTempNode != pclStartNode)
        {
            if (static_cast<GuiControl*>(pclTempNode)->AcceptsFocus())
            {
                m_pclInFocus = static_cast<GuiControl*>(
pclTempNode);
                m_pclInFocus->Activate(true);
                SetFocus(m_pclInFocus);
                return;
            }
            pclTempNode = pclTempNode->GetNext();
        }
    }
    else
    {
        pclTempNode = static_cast<GuiControl*>(m_clControlList
.GetTail());
        pclStartNode = m_pclInFocus;

        // If there isn't a current focus node, set the focus to the end
        // of the list
        if (!m_pclInFocus)
        {
            m_pclInFocus = static_cast<GuiControl*>(
pclTempNode);
            if (!m_pclInFocus)
            {
                return;
            }
            pclTempNode = static_cast<GuiControl*>(m_pclInFocus
);
            pclStartNode = NULL;
        }
        else
        {
            // Deactivate the control that's losing focus
            static_cast<GuiControl*>(m_pclInFocus)->
Activate(false);

            // Otherwise start with the previous node
            pclStartNode = pclStartNode->GetPrev();
        }

        // Go through the whole control list and find the next one to accept
        // the focus
        while (pclTempNode && pclTempNode != pclStartNode)
        {
            if (static_cast<GuiControl*>(pclTempNode)->AcceptsFocus())
            {
                m_pclInFocus = static_cast<GuiControl*>(
pclTempNode);
                m_pclInFocus->Activate(true);
                SetFocus(m_pclInFocus);
                return;
            }
            pclTempNode = pclTempNode->GetPrev();
        }

        pclTempNode = static_cast<GuiControl*>(m_clControlList
.GetTail());
        while (pclTempNode && pclTempNode != pclStartNode)
        {
            if (static_cast<GuiControl*>(pclTempNode)->AcceptsFocus())
            {
                m_pclInFocus = static_cast<GuiControl*>(
pclTempNode);
                m_pclInFocus->Activate(true);
                SetFocus(m_pclInFocus);
                return;
            }
            pclTempNode = pclTempNode->GetPrev();
        }
    }
}
//---------------------------------------------------------------------------
GuiWindow *GuiEventSurface::FindWindowByName
( const K_CHAR *szName_ )
{
    LinkListNode *pclTempNode = static_cast<LinkListNode
*>(m_clWindowList.GetHead());

    while (pclTempNode)
    {
        if (MemUtil::CompareStrings(szName_,
static_cast<GuiWindow*>(pclTempNode)->GetName()))
        {
             return static_cast<GuiWindow*>(pclTempNode);
        }
        pclTempNode = pclTempNode->GetNext();
    }

    return NULL;
}

//---------------------------------------------------------------------------
void GuiEventSurface::AddWindow( GuiWindow *
pclWindow_ )
{
    GUI_DEBUG_PRINT("GuiEventSurface::AddWindow\n");

    m_clWindowList.Add(static_cast<LinkListNode*>(pclWindow_))
;
}

//---------------------------------------------------------------------------
void GuiEventSurface::RemoveWindow( GuiWindow
 *pclWindow_ )
{
    GUI_DEBUG_PRINT("GuiEventSurface::RemoveWindow\n");

    m_clWindowList.Remove(static_cast<LinkListNode*>(
pclWindow_));
}

//---------------------------------------------------------------------------
K_BOOL GuiEventSurface::SendEvent( GuiEvent_t
 *pstEvent_ )
{
    GUI_DEBUG_PRINT("GuiEventSurface::SendEvent\n");

    // Allocate a message from the global message pool
    Message *pclMessage = GlobalMessagePool::Pop()
;

    // No messages available? Return a failure
    if (!pclMessage)
    {
        return false;
    }

    // Allocate a copy of the event from the heap
    GuiEvent_t *pstEventCopy = static_cast<GuiEvent_t*>(
SystemHeap::Alloc(sizeof(GuiEvent_t)));

    // If the allocation fails, push the message back to the global pool and
 bail
    if (!pstEventCopy)
    {
        GlobalMessagePool::Push(pclMessage);
        return false;
    }

    // Copy the source event into the destination event buffer
    CopyEvent(pstEventCopy, pstEvent_);

    // Set the new event as the message payload
    pclMessage->SetData(static_cast<void*>(pstEventCopy));

    // Send the event to the message queue
    m_clMessageQueue.Send(pclMessage);

    return true;
}

//---------------------------------------------------------------------------
K_BOOL GuiEventSurface::ProcessEvent()
{
    GUI_DEBUG_PRINT("GuiEventSurface::ProcessEvent\n");

    // read the event from the queue (blocking call)
    Message *pclMessage = m_clMessageQueue.Receive
();
    GuiEvent_t stLocalEvent;

    // If we failed to get something from the queue,
    // bail out
    if (!pclMessage)
    {
        return false;
    }

    // Copy the event data from the message into a local copy
    CopyEvent(&stLocalEvent,
        static_cast<GuiEvent_t*>(pclMessage->GetData()));

    // Free the message and event as soon as possible, since
    // they are shared system resources
    SystemHeap::Free(pclMessage->GetData());
    GlobalMessagePool::Push(pclMessage);

    // Special case check - target ID is the highest Z-ordered window(s) ONLY.
    if (stLocalEvent.ucTargetID == TARGET_ID_BROADCAST_Z)
    {
        LinkListNode* pclTempNode = m_clWindowList.
GetHead();
        LinkListNode* pclHighestNode = pclTempNode;
        K_UCHAR ucMaxZ = 0;

        while (pclTempNode)
        {
            if (ucMaxZ < (static_cast<GuiWindow*>(pclTempNode))->GetZOrder() )
            {
                ucMaxZ = static_cast<GuiWindow*>(pclTempNode)->
GetZOrder();
            }
            pclTempNode = pclTempNode->GetNext();
        }

        // Iterate through all windows again - may have multiple windows
        // at the same z-order.
        pclTempNode = m_clWindowList.GetHead();
        while (pclTempNode)
        {
            if (ucMaxZ == (static_cast<GuiWindow*>(pclTempNode))->GetZOrder())
            {
                (static_cast<GuiWindow*>(pclTempNode))->ProcessEvent(&
stLocalEvent);
            }
            pclTempNode = pclTempNode->GetNext();
        }
    }
    // Broadcast the event - sending it to *all* windows.  Let the individual
    // windows figure out what to do with the events.
    else
    {
        LinkListNode* pclTempNode = m_clWindowList.
GetHead();
        while (pclTempNode)
        {
            (static_cast<GuiWindow*>(pclTempNode))->ProcessEvent(&
stLocalEvent);
            pclTempNode = pclTempNode->GetNext();
        }
    }

    // Return out
    return true;
}

//---------------------------------------------------------------------------
void GuiEventSurface::CopyEvent( GuiEvent_t
 *pstDst_, GuiEvent_t *pstSrc_ )
{
    GUI_DEBUG_PRINT("GuiEventSurface::CopyEvent\n");
    K_UCHAR *pucDst_ = (K_UCHAR*)pstDst_;
    K_UCHAR *pucSrc_ = (K_UCHAR*)pstSrc_;
    K_UCHAR i;
    for (i = 0; i < sizeof(GuiEvent_t); i++)
    {
        *pucDst_++ = *pucSrc_++;
    }
}

//---------------------------------------------------------------------------
void GuiControl::GetControlOffset( K_USHORT *pusX_,
 K_USHORT *pusY_ )
{
    GUI_DEBUG_PRINT("GuiControl::GetControlOffset\n");
    GuiControl *pclTempControl = m_pclParentControl
;
    *pusX_ = 0;
    *pusY_ = 0;
    while (pclTempControl)
    {
        *pusX_ += pclTempControl->GetLeft();
        *pusY_ += pclTempControl->GetTop();
        pclTempControl = pclTempControl->GetParentControl();
    }

    if (m_pclParentWindow)
    {
        *pusX_ += m_pclParentWindow->GetLeft();
```

```
            *pusY_ += m_pclParentWindow->GetTop();
        }
    }
```

14.51 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/gui.h File Reference

Graphical User Interface classes and data structure declarations.

```
#include "kerneltypes.h"
#include "ll.h"
#include "driver.h"
#include "graphics.h"
#include "message.h"
#include "keycodes.h"
```

Classes

- struct KeyEvent_t

  Keyboard UI event structure definition.
- struct MouseEvent_t

  Mouse UI event structure.
- struct TouchEvent_t

  Touch UI event structure.
- struct JoystickEvent_t

  Joystick UI event structure.
- struct TimerEvent_t

  Timer UI event structure.
- struct GuiEvent_t
- class GuiWindow

  Basic Window Class.
- class GuiEventSurface

  GUI Event Surface Object.
- class GuiControl

  GUI Control Base Class.
- class StubControl

  Stub control class, used for testing out the GUI framework where certain controls have not yet been implemented.

Macros

- #define **GUI_DEBUG** (0)
- #define **GUI_DEBUG_PRINT**
- #define EVENT_STATE_UP (0)

  Event state defintions, used for determining whether or not a button or key is in the "up" or "down" contact state.
- #define **EVENT_STATE_DOWN** (1)
- #define MAX_WINDOW_CONTROLS (251)

  Maximum number of controls per window.
- #define TARGET_ID_BROADCAST_Z (252)

  Broadcast event to all controls in the topmost window.
- #define TARGET_ID_BROADCAST (253)

  Send event to all controls in all windows.
- #define TARGET_ID_FOCUS (254)

Send event to the in-focus control.

- #define TARGET_ID_HIGH_Z (255)

  Send event to the highest Z-order control.

Enumerations

- enum GuiEventType_t {
  EVENT_TYPE_KEYBOARD, EVENT_TYPE_MOUSE, EVENT_TYPE_TOUCH, EVENT_TYPE_JOYSTICK,
  EVENT_TYPE_TIMER, EVENT_TYPE_COUNT }

  Enumeration defining the various UI event codes.
- enum **GuiReturn_t** {
  GUI_EVENT_OK = 0, **GUI_EVENT_CONSUMED**, **GUI_EVENT_CANCEL**, **GUI_EVENT_RETRY**,
  GUI_EVENT_COUNT }

14.51.1 Detailed Description

Graphical User Interface classes and data structure declarations.

Definition in file gui.h.

14.51.2 Enumeration Type Documentation

14.51.2.1 enum GuiEventType_t

Enumeration defining the various UI event codes.

Enumerator:

EVENT_TYPE_KEYBOARD	Keypress event.
EVENT_TYPE_MOUSE	Mouse movement or click event.
EVENT_TYPE_TOUCH	Touchscreen movement event.
EVENT_TYPE_JOYSTICK	Joystick event.
EVENT_TYPE_TIMER	Timer event.
EVENT_TYPE_COUNT	Count of different event types supported.

Definition at line 65 of file gui.h.

14.52 gui.h

```
/*============================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||_____|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]-------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
==========================================================================*/
#ifndef __GUI_H__
#define __GUI_H__

#include "kerneltypes.h"
#include "ll.h"
#include "driver.h"
#include "graphics.h"
```

```

#include "message.h"

#include "keycodes.h"

#define GUI_DEBUG           (0)

#if GUI_DEBUG
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>

    #define GUI_DEBUG_PRINT     printf
#else
    #define GUI_DEBUG_PRINT
#endif


//---------------------------------------------------------------------------
#define EVENT_STATE_UP          (0)
#define EVENT_STATE_DOWN        (1)

//---------------------------------------------------------------------------
#define MAX_WINDOW_CONTROLS     (251)

#define TARGET_ID_BROADCAST_Z   (252)
#define TARGET_ID_BROADCAST     (253)
#define TARGET_ID_FOCUS         (254)
#define TARGET_ID_HIGH_Z        (255)


//---------------------------------------------------------------------------
typedef enum
{
    EVENT_TYPE_KEYBOARD,
    EVENT_TYPE_MOUSE,
    EVENT_TYPE_TOUCH,
    EVENT_TYPE_JOYSTICK,
    EVENT_TYPE_TIMER,
//---
    EVENT_TYPE_COUNT
} GuiEventType_t;

//---------------------------------------------------------------------------
typedef struct
{
    K_UCHAR ucKeyCode;
    union
    {
        K_UCHAR ucFlags;
        struct
        {
            unsigned int bKeyState:1;
            unsigned int bShiftState:1;
            unsigned int bCtrlState:1;
            unsigned int bAltState:1;
            unsigned int bWinState:1;
            unsigned int bFnState:1;
        };
    };
} KeyEvent_t;

//---------------------------------------------------------------------------
typedef struct
{
    K_USHORT usX;
    K_USHORT usY;

    union
    {
        K_UCHAR ucFlags;
        struct
        {
            unsigned int bLeftState:1;
            unsigned int bRightState:1;
            unsigned int bMiddleState:1;
            unsigned int bScrollUp:1;
            unsigned int bScrollDown:1;
        };
    };
} MouseEvent_t;

//---------------------------------------------------------------------------
typedef struct
{
    K_USHORT usX;
    K_USHORT usY;

    union
    {
        K_USHORT ucFlags;
        struct
        {
            unsigned int bTouch:1;
        };
    };
} TouchEvent_t;

//---------------------------------------------------------------------------
typedef struct
{
    union
    {
        K_USHORT usRawData;
        struct
        {
            unsigned int bUp:1;
            unsigned int bDown:1;
            unsigned int bLeft:1;
            unsigned int bRight:1;

            unsigned int bButton1:1;
            unsigned int bButton2:1;
            unsigned int bButton3:1;
            unsigned int bButton4:1;
            unsigned int bButton5:1;
            unsigned int bButton6:1;
            unsigned int bButton7:1;
            unsigned int bButton8:1;
            unsigned int bButton9:1;
            unsigned int bButton10:1;

            unsigned int bSelect:1;
            unsigned int bStart:1;
        };
    };
} JoystickEvent_t;

//---------------------------------------------------------------------------
typedef struct
{
    K_USHORT usTicks;
} TimerEvent_t;

//---------------------------------------------------------------------------
typedef struct
{
    K_UCHAR ucEventType;
    K_UCHAR ucTargetID;
    union
    {
        KeyEvent_t         stKey;
        MouseEvent_t     stMouse;
        TouchEvent_t     stTouch;
        JoystickEvent_t stJoystick;
        TimerEvent_t     stTimer;
    };

} GuiEvent_t;

//---------------------------------------------------------------------------
typedef enum
{
    GUI_EVENT_OK = 0,
    GUI_EVENT_CONSUMED,
    GUI_EVENT_CANCEL,
    GUI_EVENT_RETRY,
//---
    GUI_EVENT_COUNT
} GuiReturn_t;

class GuiControl;

//---------------------------------------------------------------------------
class GuiWindow : public LinkListNode
{

public:
    void Init()
    {
        m_ucControlCount = 0;
        m_pclDriver = NULL;
        m_szName = "";
    }

    void SetDriver( GraphicsDriver *pclDriver_ ) { m_pclDriver = pclDriver_; }

    GraphicsDriver *GetDriver() { return m_pclDriver; }

    void AddControl( GuiControl *pclControl_, GuiControl *pclParent_ );

    void RemoveControl( GuiControl *pclControl_ );

    K_UCHAR GetMaxZOrder();

    void Redraw( K_BOOL bRedrawAll_ );

    void ProcessEvent( GuiEvent_t *pstEvent_ );

    void SetFocus( GuiControl *pclControl_ );

    K_BOOL IsInFocus( GuiControl *pclControl_ )
    {
        if (m_pclInFocus == pclControl_)
        {
            return true;
        }
        return false;
    }

    void SetTop( K_USHORT usTop_ )         { m_usTop = usTop_; }

    void SetLeft( K_USHORT usLeft_ )       { m_usLeft = usLeft_; }

    void SetHeight( K_USHORT usHeight_ ) { m_usHeight = usHeight_; }

    void SetWidth( K_USHORT usWidth_ )   { m_usWidth = usWidth_; }

    K_USHORT GetTop()             { return m_usTop; }

    K_USHORT GetLeft()            { return m_usLeft; }

    K_USHORT GetHeight()        { return m_usHeight; }

    K_USHORT GetWidth()          { return m_usWidth; }

    K_UCHAR GetZOrder()          { return m_ucZ; }

    void SetZOrder( K_UCHAR ucZ_ ) { m_ucZ = ucZ_; }

    void CycleFocus( bool bForward_ );

    void SetName( const K_CHAR *szName_ ) { m_szName = szName_; }

    const K_CHAR *GetName() { return m_szName; }

private:
    K_USHORT m_usTop;
    K_USHORT m_usLeft;
    K_USHORT m_usHeight;
    K_USHORT m_usWidth;

    K_UCHAR  m_ucZ;
    const K_CHAR  *m_szName;

    DoubleLinkList m_clControlList;
    GuiControl *m_pclInFocus;
    K_UCHAR m_ucControlCount;
    GraphicsDriver *m_pclDriver;
};

//---------------------------------------------------------------------------
class GuiEventSurface
{
public:
    void Init() { m_clMessageQueue.Init(); }

    void AddWindow( GuiWindow *pclWindow_ );

    void RemoveWindow( GuiWindow *pclWindow_ );

    K_BOOL SendEvent( GuiEvent_t *pstEvent_ );

    K_BOOL ProcessEvent();

    K_UCHAR GetEventCount() { return m_clMessageQueue
.GetCount(); }

    GuiWindow *FindWindowByName( const K_CHAR *szName_
 );


private:
    void CopyEvent( GuiEvent_t *pstDst_, GuiEvent_t
 *pstSrc_ );

private:
    DoubleLinkList m_clWindowList;

    MessageQueue   m_clMessageQueue;
};

//---------------------------------------------------------------------------
class GuiControl : public LinkListNode
{
public:
    virtual void Init() = 0;

    virtual void Draw() = 0;

    virtual GuiReturn_t ProcessEvent( GuiEvent_t *
pstEvent_ ) = 0;

    void SetTop( K_USHORT usTop_ )         { m_usTop = usTop_; }

    void SetLeft( K_USHORT usLeft_ )      { m_usLeft = usLeft_;
}

    void SetHeight( K_USHORT usHeight_ ) { m_usHeight =
usHeight_; }

    void SetWidth( K_USHORT usWidth_ )   { m_usWidth =
usWidth_; }

    void SetZOrder( K_UCHAR ucZ_ )          { m_ucZOrder =
ucZ_; }

    void SetControlIndex( K_UCHAR ucIdx_ ) { m_ucControlIndex
 = ucIdx_; }

    K_USHORT GetTop()              { return m_usTop; }

    K_USHORT GetLeft()            { return m_usLeft; }

    K_USHORT GetHeight()       { return m_usHeight; }

    K_USHORT GetWidth()          { return m_usWidth; }

    K_UCHAR  GetZOrder()       { return m_ucZOrder; }

    K_UCHAR  GetControlIndex()   { return m_ucControlIndex
; }

    K_BOOL   IsStale()            { return m_bStale; }

    void GetControlOffset( K_USHORT *pusX_, K_USHORT *pusY_ );

    K_BOOL  IsInFocus()
    {
        return m_pclParentWindow->IsInFocus(this);
    }

    virtual void Activate( bool bActivate_ ) = 0;

protected:
    friend class GuiWindow;
    friend class GuiEventSurface;

    void SetParentControl( GuiControl *pclParent_ ) {
 m_pclParentControl = pclParent_; }

    void SetParentWindow( GuiWindow *pclWindow_ )    {
m_pclParentWindow  = pclWindow_; }

    GuiControl *GetParentControl()
{ return m_pclParentControl; }

    GuiWindow *GetParentWindow()                     {
return m_pclParentWindow; }

    void ClearStale()                                { m_bStale
 = false; }

    void SetStale()                                { m_bStale
= true; }

    void SetAcceptFocus( bool bFocus_ )            {
m_bAcceptsFocus = bFocus_; }

    bool AcceptsFocus()                            { return
m_bAcceptsFocus; }
private:
    K_BOOL   m_bStale;

    K_BOOL   m_bAcceptsFocus;

    K_UCHAR  m_ucZOrder;

    K_UCHAR  m_ucControlIndex;

    K_USHORT m_usTop;

    K_USHORT m_usLeft;

    K_USHORT m_usWidth;

    K_USHORT m_usHeight;

    GuiControl *m_pclParentControl;

    GuiWindow  *m_pclParentWindow;
};

//---------------------------------------------------------------------------
class StubControl : public GuiControl
{
public:
    virtual void Init() {   }
    virtual void Draw() {   }
    virtual GuiReturn_t ProcessEvent( GuiEvent_t *
pstEvent_ ) { return GUI_EVENT_OK; }
    virtual void Activate( bool bActivate_ ) { }
};

#endif
```

14.53 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kernel.cpp File Reference

Kernel initialization and startup code.

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "kernel.h"
#include "scheduler.h"
#include "thread.h"
#include "threadport.h"
#include "timerlist.h"
#include "message.h"
#include "driver.h"
#include "profile.h"
#include "kprofile.h"
#include "tracebuffer.h"
#include "kernel_debug.h"
```

Macros

- #define **__FILE_ID__** KERNEL_CPP

14.53.1 Detailed Description

Kernel initialization and startup code.

Definition in file kernel.cpp.

14.54 kernel.cpp

```
/*===========================================================================
     _____     _____     _____     _____
 ___|    _|__ __|_    |__ __|__   |__ __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "kerneltypes.h"
#include "mark3cfg.h"

#include "kernel.h"
#include "scheduler.h"
#include "thread.h"
#include "threadport.h"
#include "timerlist.h"
#include "message.h"
#include "driver.h"
#include "profile.h"
#include "kprofile.h"
#include "tracebuffer.h"
#include "kernel_debug.h"

//---------------------------------------------------------------------------
#if defined __FILE_ID__
    #undef __FILE_ID__
#endif
#define __FILE_ID__     KERNEL_CPP

//---------------------------------------------------------------------------
void Kernel::Init(void)
{
#if KERNEL_USE_DEBUG
    TraceBuffer::Init();
#endif
    KERNEL_TRACE( STR_MARK3_INIT );

    // Initialize the global kernel data - scheduler, timer-scheduler, and
    // the global message pool.
    Scheduler::Init();
#if KERNEL_USE_DRIVER
    DriverList::Init();
#endif
#if KERNEL_USE_TIMERS
    TimerScheduler::Init();
#endif
#if KERNEL_USE_MESSAGE
    GlobalMessagePool::Init();
#endif
#if KERNEL_USE_PROFILER
    Profiler::Init();
#endif
}

//---------------------------------------------------------------------------
void Kernel::Start(void)
{
    KERNEL_TRACE( STR_THREAD_START );
    ThreadPort::StartThreads();
    KERNEL_TRACE( STR_START_ERROR );
}
```

14.55 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kernel.h File Reference

Kernel initialization and startup class.

`#include "kerneltypes.h"`

Classes

- class Kernel

  Class that encapsulates all of the kernel startup functions.

14.55.1 Detailed Description

Kernel initialization and startup class. The Kernel namespace provides functions related to initializing and starting up the kernel.

The Kernel::Init() function must be called before any of the other functions in the kernel can be used.

Once the initial kernel configuration has been completed (i.e. first threads have been added to the scheduler), the Kernel::Start() function can then be called, which will transition code execution from the "main()" context to the threads in the scheduler.

Definition in file kernel.h.

14.56 kernel.h

```
/*===========================================================================
     _____     _____     _____     _____
 ___|    _|__ __|_    |__ __|__   |__ __| __  |__  _____
|    \  /  | ||    \    ||     |    ||  |/ /    ||___   |
|     \/   | ||     \   ||     \    ||     \    ||___   |
|__/\__/|__|_||__|\__\ __||__|\__\ __||__|\__\ __||_____|
    |_____|      |_____|     |_____|     |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#ifndef __KERNEL_H__
#define __KERNEL_H__

#include "kerneltypes.h"
//---------------------------------------------------------------------------
class Kernel
{
public:
    static void Init(void);

    static void Start(void);
};

#endif

```

14.57 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kernel_debug.h File Reference

Macros and functions used for assertions, kernel traces, etc.

```
#include "debug_tokens.h"
#include "mark3cfg.h"
#include "tracebuffer.h"
```

Macros

- #define **__FILE_ID__** 0
- #define **KERNEL_TRACE**(x)
- #define **KERNEL_TRACE_1**(x, arg1)
- #define **KERNEL_TRACE_2**(x, arg1, arg2)
- #define **KERNEL_ASSERT**(x)

14.57.1 Detailed Description

Macros and functions used for assertions, kernel traces, etc.

Definition in file kernel_debug.h.

14.58 kernel_debug.h

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#ifndef __KERNEL_DEBUG_H__
#define __KERNEL_DEBUG_H__

#include "debug_tokens.h"
#include "mark3cfg.h"
#include "tracebuffer.h"

//---------------------------------------------------------------------------
#if KERNEL_USE_DEBUG

//---------------------------------------------------------------------------
#define __FILE_ID__         STR_UNDEFINED

//---------------------------------------------------------------------------
#define KERNEL_TRACE( x )   \
{   \
    K_USHORT ausMsg__[5]; \
    ausMsg__[0] = 0xACDC;  \
    ausMsg__[1] = __FILE_ID__; \
    ausMsg__[2] = __LINE__; \
    ausMsg__[3] = TraceBuffer::Increment() ; \
    ausMsg__[4] = (K_USHORT)(x) ; \
    TraceBuffer::Write(ausMsg__, 5); \
};

//---------------------------------------------------------------------------
#define KERNEL_TRACE_1( x, arg1 ) \
{   \
    K_USHORT ausMsg__[6]; \
    ausMsg__[0] = 0xACDC;  \
    ausMsg__[1] = __FILE_ID__; \
    ausMsg__[2] = __LINE__; \
    ausMsg__[3] = TraceBuffer::Increment(); \
    ausMsg__[4] = (K_USHORT)(x); \
    ausMsg__[5] = arg1; \
    TraceBuffer::Write(ausMsg__, 6); \
}

```

```
//---------------------------------------------------------------------------
#define KERNEL_TRACE_2( x, arg1, arg2 ) \
{   \
    K_USHORT ausMsg__[7]; \
    ausMsg__[0] = 0xACDC;  \
    ausMsg__[1] = __FILE_ID__; \
    ausMsg__[2] = __LINE__; \
    ausMsg__[3] = TraceBuffer::Increment(); \
    ausMsg__[4] = (K_USHORT)(x); \
    ausMsg__[5] = arg1; \
    ausMsg__[6] = arg2; \
    TraceBuffer::Write(ausMsg__, 7); \
}

//---------------------------------------------------------------------------
#define KERNEL_ASSERT( x ) \
{       \
    if( ( x ) == false ) \
    {   \
        K_USHORT ausMsg__[5];   \
        ausMsg__[0] = 0xACDC;   \
        ausMsg__[1] = __FILE_ID__;  \
        ausMsg__[2] = __LINE__; \
        ausMsg__[3] = TraceBuffer::Increment(); \
        ausMsg__[4] = STR_ASSERT_FAILED;    \
        TraceBuffer::Write(ausMsg__, 5); \
    }   \
}

#else
//---------------------------------------------------------------------------
#define __FILE_ID__         0
//---------------------------------------------------------------------------
#define KERNEL_TRACE( x )
//---------------------------------------------------------------------------
#define KERNEL_TRACE_1( x, arg1 )
//---------------------------------------------------------------------------
#define KERNEL_TRACE_2( x, arg1, arg2 )
//---------------------------------------------------------------------------
#define KERNEL_ASSERT( x )

#endif // KERNEL_USE_DEBUG

#endif
```

14.59 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kernelswi.cpp File Reference

Kernel Software interrupt implementation for ATMega328p.

```
#include "kerneltypes.h"
#include "kernelswi.h"
#include <avr/io.h>
#include <avr/interrupt.h>
```

14.59.1 Detailed Description

Kernel Software interrupt implementation for ATMega328p.

Definition in file kernelswi.cpp.

14.60 kernelswi.cpp

```
/*===========================================================================
     _____     _____     _____     _____
 ___|    _|__ __|_    |__ __|__   |__ __| __  |__  _____
|    \  /  | ||    \     ||     |     ||  |/ /    ||___   |
|     \/   | ||     \    ||     \     ||  |  \    ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\ __||_______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "kerneltypes.h"
#include "kernelswi.h"

#include <avr/io.h>
#include <avr/interrupt.h>

//---------------------------------------------------------------------------
void KernelSWI::Config(void)
{
    PORTD &= ~0x04; // Clear INT0
    DDRD |= 0x04;    // Set PortD, bit 2 (INT0) As Output
    EICRA |= (1 << ISC00) | (1 << ISC01);    // Rising edge on INT0
}

//---------------------------------------------------------------------------
void KernelSWI::Start(void)
{
    EIFR &= ~(1 << INTF0);    // Clear any pending interrupts on INT0
    EIMSK |= (1 << INT0);    // Enable INT0 interrupt (as K_LONG as I-bit is
 set)
}

//---------------------------------------------------------------------------
void KernelSWI::Stop(void)
{
    EIMSK &= ~(1 << INT0);    // Disable INT0 interrupts
}

//---------------------------------------------------------------------------
K_UCHAR KernelSWI::DI()
{
    K_UCHAR bEnabled = ((EIMSK & (1 << INT0)) != 0);
    EIMSK &= ~(1 << INT0);
    return bEnabled;
}

//---------------------------------------------------------------------------
void KernelSWI::RI(K_UCHAR bEnable_)
{
    if (bEnable_)
    {
        EIMSK |= (1 << INT0);
    }
    else
    {
        EIMSK &= ~(1 << INT0);
    }
}

//---------------------------------------------------------------------------
void KernelSWI::Clear(void)
{
    EIFR &= ~(1 << INTF0);     // Clear the interrupt flag for INT0
}

//---------------------------------------------------------------------------
void KernelSWI::Trigger(void)
{
    //if(Thread_IsSchedulerEnabled())
    {
        PORTD &= ~0x04;
        PORTD |= 0x04;
    }
}
```

14.61 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kernelswi.h File Reference

Kernel Software interrupt declarations.

```
#include "kerneltypes.h"
```

Classes

- class KernelSWI

  Class providing the software-interrupt required for context-switching in the kernel.

14.61.1 Detailed Description

Kernel Software interrupt declarations.

Definition in file kernelswi.h.

14.62 kernelswi.h

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /    ||___   |
|     \/   | ||     \     ||     \     ||     \    ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\ __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "kerneltypes.h"
#ifndef __KERNELSWI_H_
#define __KERNELSWI_H_

//---------------------------------------------------------------------------
class KernelSWI
{
public:
    static void Config(void);

    static void Start(void);

    static void Stop(void);

    static void Clear(void);

    static void Trigger(void);

    static K_UCHAR DI();

    static void RI(K_UCHAR bEnable_);
};


#endif // __KERNELSIW_H_
```

14.63 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kerneltimer.cpp File Reference

Kernel Timer Implementation for ATMega328p.

```
#include "kerneltypes.h"
#include "kerneltimer.h"
#include <avr/io.h>
#include <avr/interrupt.h>
```

Macros

- #define **TCCR1B_INIT** ((1 << WGM12) | (1 << CS12))

- #define **TIMER_IMSK** (1 << OCIE1A)
- #define **TIMER_IFR** (1 << OCF1A)

14.63.1 Detailed Description

Kernel Timer Implementation for ATMega328p.

Definition in file kerneltimer.cpp.

14.64 kerneltimer.cpp

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "kerneltypes.h"
#include "kerneltimer.h"

#include <avr/io.h>
#include <avr/interrupt.h>

#define TCCR1B_INIT      ((1 << WGM12) | (1 << CS12))
#define TIMER_IMSK       (1 << OCIE1A)
#define TIMER_IFR        (1 << OCF1A)

//---------------------------------------------------------------------------
void KernelTimer::Config(void)
{
    TCCR1B = TCCR1B_INIT;
}

//---------------------------------------------------------------------------
void KernelTimer::Start(void)
{
    TCNT1 = 0;
    TIFR1 &= ~TIMER_IFR;
    TIMSK1 |= TIMER_IMSK;
    TCCR1B |= (1 << CS12);      // Enable count...
}

//---------------------------------------------------------------------------
void KernelTimer::Stop(void)
{
    TIFR1 &= ~TIMER_IFR;
    TIMSK1 &= ~TIMER_IMSK;
    TCCR1B &= ~(1 << CS12);      // Disable count...
    TCNT1 = 0;
    OCR1A = 0;
}

//---------------------------------------------------------------------------
K_USHORT KernelTimer::Read(void)
{
    volatile K_USHORT usRead1;
    volatile K_USHORT usRead2;

    do {
        usRead1 = TCNT1;
        usRead2 = TCNT1;
    } while (usRead1 != usRead2);

    return usRead1;
}

//---------------------------------------------------------------------------
K_ULONG KernelTimer::SubtractExpiry(K_ULONG
ulInterval_)
{
    OCR1A -= (K_USHORT)ulInterval_;
    return (K_ULONG)OCR1A;
}

//-----------------------------------------------------------------------------
K_ULONG KernelTimer::TimeToExpiry(void)
{
    K_USHORT usRead = KernelTimer::Read();
    K_USHORT usOCR1A = OCR1A;

    if (usRead >= usOCR1A)
    {
        return 0;
    }
    else
    {
        return (K_ULONG)(usOCR1A - usRead);
    }
}

//-----------------------------------------------------------------------------
K_ULONG KernelTimer::GetOvertime(void)
{
    return KernelTimer::Read();
}

//-----------------------------------------------------------------------------
K_ULONG KernelTimer::SetExpiry(K_ULONG ulInterval_)
{
    K_USHORT usSetInterval;
    if (ulInterval_ > 65535)
    {
        usSetInterval = 65535;
    }
    else
    {
        usSetInterval = (K_USHORT)ulInterval_ ;
    }
    OCR1A = usSetInterval;
    return (K_ULONG)usSetInterval;
}

//-----------------------------------------------------------------------------
void KernelTimer::ClearExpiry(void)
{
    OCR1A = 65535;                  // Clear the compare value
}

//-----------------------------------------------------------------------------
K_UCHAR KernelTimer::DI(void)
{
    K_UCHAR bEnabled = ((TIMSK1 & (TIMER_IMSK)) != 0);
    TIFR1 &= ~TIMER_IFR;        // Clear interrupt flags
    TIMSK1 &= ~TIMER_IMSK;      // Disable interrupt
    return bEnabled;
}

//-----------------------------------------------------------------------------
void KernelTimer::EI(void)
{
    KernelTimer::RI(0);
}

//-----------------------------------------------------------------------------
void KernelTimer::RI(K_UCHAR bEnable_)
{
    if (bEnable_)
    {
        TIMSK1 |= (1 << OCIE1A);    // Enable interrupt
    }
    else
    {
        TIMSK1 &= ~(1 << OCIE1A);
    }
}
```

14.65 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kerneltimer.h File Reference

Kernel Timer Class declaration.

```
#include "kerneltypes.h"
```

Classes

- class KernelTimer

    Hardware timer interface, used by all scheduling/timer subsystems.

Macros

- #define **SYSTEM_FREQ** ((K_ULONG)16000000)
- #define **TIMER_FREQ** ((K_ULONG)(SYSTEM_FREQ / 256))

14.65.1 Detailed Description

Kernel Timer Class declaration.

Definition in file kerneltimer.h.

14.66 kerneltimer.h

```
/*============================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]---------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
==========================================================================*/
#include "kerneltypes.h"
#ifndef __KERNELTIMER_H_
#define __KERNELTIMER_H_

//----------------------------------------------------------------------------
#define SYSTEM_FREQ     ((K_ULONG)16000000)
#define TIMER_FREQ      ((K_ULONG)(SYSTEM_FREQ / 256)) // Timer ticks per
 second...

//----------------------------------------------------------------------------
class KernelTimer
{
public:
    static void Config(void);

    static void Start(void);

    static void Stop(void);

    static K_UCHAR DI(void);

    static void RI(K_UCHAR bEnable_);

    static void EI(void);

    static K_ULONG SubtractExpiry(K_ULONG ulInterval_);

    static K_ULONG TimeToExpiry(void);

    static K_ULONG SetExpiry(K_ULONG ulInterval_);

    static K_ULONG GetOvertime(void);

    static void ClearExpiry(void);

private:
    static K_USHORT Read(void);

};

#endif //__KERNELTIMER_H_
```

14.67 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kerneltypes.h File Reference

Basic data type primatives used throughout the OS.

Macros

- #define **K_BOOL** unsigned char
- #define **K_CHAR** char
- #define **K_UCHAR** unsigned char
- #define **K_USHORT** unsigned short
- #define **K_SHORT** short
- #define **K_ULONG** unsigned long
- #define **K_LONG** long

14.67.1 Detailed Description

Basic data type primatives used throughout the OS.

Definition in file kerneltypes.h.

14.68 kerneltypes.h

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /    ||___   |
|     \/   | ||     \     ||     \     ||     \    ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\ __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#ifndef __KERNELTYPES_H__
#define __KERNELTYPES_H__

#if defined(bool)
    #define K_BOOL          bool
#else
    #define K_BOOL          unsigned char
#endif

#define K_CHAR          char
#define K_UCHAR         unsigned char
#define K_USHORT        unsigned short
#define K_SHORT         short
#define K_ULONG         unsigned long
#define K_LONG          long

#endif
```

14.69 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/keycodes.h File Reference

Standard ASCII keyboard codes.

```
#include "kerneltypes.h"
```

Enumerations

- enum **KEYCODE** {
KEYCODE_LBUTTON = 0x01, **KEYCODE_RBUTTON**, **KEYCODE_CANCEL**, **KEYCODE_MBUTTON**,
KEYCODE_BACK = 0x08, **KEYCODE_TAB**, **KEYCODE_CLEAR** = 0x0C, **KEYCODE_RETURN**,
KEYCODE_SHIFT = 0x10, **KEYCODE_CONTROL**, **KEYCODE_MENU**, **KEYCODE_PAUSE**,
KEYCODE_CAPITAL, **KEYCODE_ESCAPE** = 0x1B, **KEYCODE_SPACE**, **KEYCODE_PRIOR**,
KEYCODE_NEXT, **KEYCODE_END**, **KEYCODE_HOME**, **KEYCODE_LEFT**,
KEYCODE_UP, **KEYCODE_RIGHT**, **KEYCODE_DOWN**, **KEYCODE_SELECT**,
KEYCODE_PRINT, **KEYCODE_EXECUTE**, **KEYCODE_SNAPSHOT**, **KEYCODE_INSERT**,
KEYCODE_DELETE, **KEYCODE_HELP** = 0x2F, **KEYCODE_0**, **KEYCODE_1**,
KEYCODE_2, **KEYCODE_3**, **KEYCODE_4**, **KEYCODE_5**,
KEYCODE_6, **KEYCODE_7**, **KEYCODE_8**, **KEYCODE_9**,
KEYCODE_A, **KEYCODE_B**, **KEYCODE_C**, **KEYCODE_D**,
KEYCODE_E, **KEYCODE_F**, **KEYCODE_G**, **KEYCODE_H**,
KEYCODE_I, **KEYCODE_J**, **KEYCODE_K**, **KEYCODE_L**,
KEYCODE_M, **KEYCODE_N**, **KEYCODE_O**, **KEYCODE_P**,
KEYCODE_Q, **KEYCODE_R**, **KEYCODE_S**, **KEYCODE_T**,
KEYCODE_U, **KEYCODE_V**, **KEYCODE_W**, **KEYCODE_X**,
KEYCODE_Y, **KEYCODE_Z**, **KEYCODE_NUMPAD0** = 0x60, **KEYCODE_NUMPAD1**,
KEYCODE_NUMPAD2, **KEYCODE_NUMPAD3**, **KEYCODE_NUMPAD4**, **KEYCODE_NUMPAD5**,
KEYCODE_NUMPAD6, **KEYCODE_NUMPAD7**, **KEYCODE_NUMPAD8**, **KEYCODE_NUMPAD9**,
KEYCODE_SEPARATOR = 0x6C, **KEYCODE_SUBTRACT**, **KEYCODE_DECIMAL**, **KEYCODE_DIVIDE**,
KEYCODE_F1, **KEYCODE_F2**, **KEYCODE_F3**, **KEYCODE_F4**,
KEYCODE_F5, **KEYCODE_F6**, **KEYCODE_F7**, **KEYCODE_F8**,
KEYCODE_F9, **KEYCODE_F10**, **KEYCODE_F11**, **KEYCODE_F12**,
KEYCODE_F13, **KEYCODE_F14**, **KEYCODE_F15**, **KEYCODE_F16**,
KEYCODE_F17, **KEYCODE_F18**, **KEYCODE_F19**, **KEYCODE_F20**,
KEYCODE_F21, **KEYCODE_F22**, **KEYCODE_F23**, **KEYCODE_F24**,
KEYCODE_NUMLOCK = 0x90, **KEYCODE_SCROLL**, **KEYCODE_LSHIFT** = 0xA0, **KEYCODE_RSHIFT**,
KEYCODE_LCONTROL, **KEYCODE_RCONTROL**, **KEYCODE_LMENU**, **KEYCODE_RMENU**,
KEYCODE_PLAY = 0xFA, **KEYCODE_ZOOM** }

14.69.1 Detailed Description

Standard ASCII keyboard codes.

Definition in file keycodes.h.

14.70 keycodes.h

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \     ||     |     ||  |/ /     ||___   |
|     \/   | ||     \    ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#ifndef __KEYCODES_H__
#define __KEYCODES_H__

#include "kerneltypes.h"

typedef enum
{
    KEYCODE_LBUTTON = 0x01,
    KEYCODE_RBUTTON,
    KEYCODE_CANCEL,
    KEYCODE_MBUTTON,
    KEYCODE_BACK = 0x08,
    KEYCODE_TAB,
    KEYCODE_CLEAR = 0x0C,
    KEYCODE_RETURN,
    KEYCODE_SHIFT = 0x10,
    KEYCODE_CONTROL,
    KEYCODE_MENU,
    KEYCODE_PAUSE,
    KEYCODE_CAPITAL,
    KEYCODE_ESCAPE = 0x1B,
    KEYCODE_SPACE,
    KEYCODE_PRIOR,
    KEYCODE_NEXT,
    KEYCODE_END,
    KEYCODE_HOME,
    KEYCODE_LEFT,
    KEYCODE_UP,
    KEYCODE_RIGHT,
    KEYCODE_DOWN,
    KEYCODE_SELECT,
    KEYCODE_PRINT,
    KEYCODE_EXECUTE,
    KEYCODE_SNAPSHOT,
    KEYCODE_INSERT,
    KEYCODE_DELETE,
    KEYCODE_HELP = 0x2F,
    KEYCODE_0,
    KEYCODE_1,
    KEYCODE_2,
    KEYCODE_3,
    KEYCODE_4,
    KEYCODE_5,
    KEYCODE_6,
    KEYCODE_7,
    KEYCODE_8,
    KEYCODE_9,
    KEYCODE_A,
    KEYCODE_B,
    KEYCODE_C,
    KEYCODE_D,
    KEYCODE_E,
    KEYCODE_F,
    KEYCODE_G,
    KEYCODE_H,
    KEYCODE_I,
    KEYCODE_J,
    KEYCODE_K,
    KEYCODE_L,
    KEYCODE_M,
    KEYCODE_N,
    KEYCODE_O,
    KEYCODE_P,
    KEYCODE_Q,
    KEYCODE_R,
    KEYCODE_S,
    KEYCODE_T,
    KEYCODE_U,
    KEYCODE_V,
    KEYCODE_W,
    KEYCODE_X,
    KEYCODE_Y,
    KEYCODE_Z,
    KEYCODE_NUMPAD0 = 0x60,
    KEYCODE_NUMPAD1,
    KEYCODE_NUMPAD2,
    KEYCODE_NUMPAD3,
    KEYCODE_NUMPAD4,
    KEYCODE_NUMPAD5,
    KEYCODE_NUMPAD6,
    KEYCODE_NUMPAD7,
    KEYCODE_NUMPAD8,
    KEYCODE_NUMPAD9,
    KEYCODE_SEPARATOR = 0x6C,
    KEYCODE_SUBTRACT,
    KEYCODE_DECIMAL,
    KEYCODE_DIVIDE,
    KEYCODE_F1,
    KEYCODE_F2,
    KEYCODE_F3,
    KEYCODE_F4,
    KEYCODE_F5,
    KEYCODE_F6,
    KEYCODE_F7,
    KEYCODE_F8,
    KEYCODE_F9,
    KEYCODE_F10,
    KEYCODE_F11,
    KEYCODE_F12,
    KEYCODE_F13,
    KEYCODE_F14,
    KEYCODE_F15,
    KEYCODE_F16,
    KEYCODE_F17,
    KEYCODE_F18,
    KEYCODE_F19,
    KEYCODE_F20,
    KEYCODE_F21,
    KEYCODE_F22,
    KEYCODE_F23,
    KEYCODE_F24,
    KEYCODE_NUMLOCK = 0x90,
    KEYCODE_SCROLL,
    KEYCODE_LSHIFT = 0xA0,
    KEYCODE_RSHIFT,
    KEYCODE_LCONTROL,
    KEYCODE_RCONTROL,
    KEYCODE_LMENU,
    KEYCODE_RMENU,
    KEYCODE_PLAY = 0xFA,
    KEYCODE_ZOOM
} KEYCODE;

#endif //__KEYCODES_H_
```

14.71 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kprofile.cpp File Reference

ATMega328p Profiling timer implementation.

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "profile.h"
#include "kprofile.h"
#include "threadport.h"
#include <avr/io.h>
#include <avr/interrupt.h>
```

Functions

- **ISR** (TIMER0_OVF_vect)

14.71.1 Detailed Description

ATMega328p Profiling timer implementation.

Definition in file kprofile.cpp.

14.72 kprofile.cpp

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
```

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "profile.h"
#include "kprofile.h"
#include "threadport.h"
#include <avr/io.h>
#include <avr/interrupt.h>

#if KERNEL_USE_PROFILER
K_ULONG Profiler::m_ulEpoch;

//---------------------------------------------------------------------------
void Profiler::Init()
{
    TCCR0A = 0;
    TCCR0B = 0;
    TIFR0 = 0;
    TIMSK0 = 0;
    m_ulEpoch = 0;
}

//---------------------------------------------------------------------------
void Profiler::Start()
{
    TIFR0 = 0;
    TCNT0 = 0;
    TCCR0B |= (1 << CS01);
    TIMSK0 |= (1 << TOIE0);
}

//---------------------------------------------------------------------------
void Profiler::Stop()
{
    TIFR0 = 0;
    TCCR0B &= ~(1 << CS01);
    TIMSK0 &= ~(1 << TOIE0);
}
//---------------------------------------------------------------------------
K_USHORT Profiler::Read()
{
    K_USHORT usRet;
    CS_ENTER();
    TCCR0B &= ~(1 << CS01);
    usRet = TCNT0;
    TCCR0B |= (1 << CS01);
    CS_EXIT();
    return usRet;
}

//---------------------------------------------------------------------------
void Profiler::Process()
{
    CS_ENTER();
    m_ulEpoch++;
    CS_EXIT();
}

//---------------------------------------------------------------------------
ISR(TIMER0_OVF_vect)
{
    Profiler::Process();
}

#endif
```

14.73 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kprofile.h File Reference

Profiling timer hardware interface.

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "ll.h"
```

Classes

- class Profiler

  System profiling timer interface.

Macros

- #define **TICKS_PER_OVERFLOW** (256)
- #define **CLOCK_DIVIDE** (8)

14.73.1 Detailed Description

Profiling timer hardware interface.

Definition in file kprofile.h.

14.74 kprofile.h

```
/*===========================================================================
     _____     _____     _____     _____
 ___|    _|__ __|_    |__  __|__   |__  __| __  |__  _____
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||_____|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "ll.h"

#ifndef __KPROFILE_H__
#define __KPROFILE_H__

#if KERNEL_USE_PROFILER

//---------------------------------------------------------------------------
#define TICKS_PER_OVERFLOW              (256)
#define CLOCK_DIVIDE                    (8)

//---------------------------------------------------------------------------
class Profiler
{
public:
    static void Init();

    static void Start();

    static void Stop();

    static K_USHORT Read();

    static void Process();

    static K_ULONG GetEpoch(){ return m_ulEpoch; }
private:

    static K_ULONG m_ulEpoch;
};

#endif //KERNEL_USE_PROFILER

#endif

```

14.75 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/ll.cpp File Reference

Core Linked-List implementation, from which all kernel objects are derived.

```
#include "kerneltypes.h"
#include "ll.h"
#include "kernel_debug.h"
```

Macros

- #define **__FILE_ID__** LL_CPP

14.75.1 Detailed Description

Core Linked-List implementation, from which all kernel objects are derived.

Definition in file ll.cpp.

14.76 ll.cpp

```
/*===========================================================================
     _____     _____     _____     _____
 ___|    _|__ __|_    |__ __|__   |__ __| __  |__ ______
|    \  /  | ||    \    ||     |    ||  |/ /    ||___   |
|     \/   | ||     \   ||     \    ||     \    ||___   |
|__/\__/|__|_||__|\__\ __||__|\__\ __||__|\__\ __||______|
    |_____|      |_____|     |_____|     |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "kerneltypes.h"
#include "ll.h"
#include "kernel_debug.h"

//---------------------------------------------------------------------------
#if defined __FILE_ID__
    #undef __FILE_ID__
#endif
#define __FILE_ID__     LL_CPP

//---------------------------------------------------------------------------
void LinkListNode::ClearNode()
{
    next = NULL;
    prev = NULL;
}

//---------------------------------------------------------------------------
void DoubleLinkList::Add(LinkListNode *node_)
{
    KERNEL_ASSERT( node_ );

    // Add a node to the end of the linked list.
    if (!m_pstHead)
    {
        // If the list is empty, initilize the nodes
        m_pstHead = node_;
        m_pstTail = node_;

        m_pstHead->prev = NULL;
        m_pstTail->next = NULL;
        return;
    }

    // Move the tail node, and assign it to the new node just passed in
    m_pstTail->next = node_;
    node_->prev = m_pstTail;
    node_->next = NULL;
    m_pstTail = node_;
}

//---------------------------------------------------------------------------
void DoubleLinkList::Remove(LinkListNode *
node_)
{
    KERNEL_ASSERT( node_ );

    if (node_->prev)
    {
        node_->prev->next = node_->next;
    }
    if (node_->next)
    {
        node_->next->prev = node_->prev;
    }
    if (node_ == m_pstHead)
    {
        m_pstHead = node_->next;
    }
    if (node_ == m_pstTail)
    {
        m_pstTail = node_->prev;
    }

    node_->ClearNode();
}

//---------------------------------------------------------------------------
void CircularLinkList::Add(LinkListNode *node_
)
{
    KERNEL_ASSERT( node_ );

    // Add a node to the end of the linked list.
    if (!m_pstHead)
    {
        // If the list is empty, initilize the nodes
        m_pstHead = node_;
        m_pstTail = node_;

        m_pstHead->prev = m_pstHead;
        m_pstHead->next = m_pstHead;
        return;
    }

    // Move the tail node, and assign it to the new node just passed in
    m_pstTail->next = node_;
    node_->prev = m_pstTail;
    node_->next = m_pstHead;
    m_pstTail = node_;
    m_pstHead->prev = node_;
}

//---------------------------------------------------------------------------
void CircularLinkList::Remove(LinkListNode
*node_)
{
    KERNEL_ASSERT( node_ );

    // Check to see if this is the head of the list...
    if ((node_ == m_pstHead) && (m_pstHead == m_pstTail))
    {
        // Clear the head and tail pointers - nothing else left.
        m_pstHead = NULL;
        m_pstTail = NULL;
        return;
    }

    // This is a circularly linked list - no need to check for connection,
    // just remove the node.
    node_->next->prev = node_->prev;
    node_->prev->next = node_->next;

    if (node_ == m_pstHead)
    {
        m_pstHead = m_pstHead->next;
    }
    if (node_ == m_pstTail)
    {
        m_pstTail = m_pstTail->prev;
    }
    node_->ClearNode();
}

```

```
//---------------------------------------------------------------------------
void CircularLinkList::PivotForward()
{
    if (m_pstHead)
    {
        m_pstHead = m_pstHead->next;
        m_pstTail = m_pstTail->next;
    }
}

//---------------------------------------------------------------------------
void CircularLinkList::PivotBackward()
{
    if (m_pstHead)
    {
        m_pstHead = m_pstHead->prev;
        m_pstTail = m_pstTail->prev;
    }
}
```

14.77 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/ll.h File Reference

Core linked-list declarations, used by all kernel list types.

```
#include "kerneltypes.h"
```

Classes

- class LinkListNode

  Basic linked-list node data structure.
- class LinkList

  Abstract-data-type from which all other linked-lists are derived.
- class DoubleLinkList

  Doubly-linked-list data type, inherited from the base LinkList type.
- class CircularLinkList

  Circular-linked-list data type, inherited from the base LinkList type.

Macros

- #define **NULL** (0)
- #define SAFE_UNLINK (0)

  "Safe unlinking" performs extra checks on data to make sure that there are no consistencies when performing node operations.

14.77.1 Detailed Description

Core linked-list declarations, used by all kernel list types. At the heart of RTOS data structures are linked lists. Having a robust and efficient set of linked-list types that we can use as a foundation for building the rest of our kernel types allows us to keep our RTOS code efficient and logically-separated.

So what data types rely on these linked-list classes?

-Threads -ThreadLists -The Scheduler -Timers, -The Timer Scheduler -Blocking objects (Semaphores, Mutexes, etc...)

Pretty much everything in the kernel uses these linked lists. By having objects inherit from the base linked-list node type, we're able to leverage the double and circular linked-list classes to manager virtually every object type in the system without duplicating code. These functions are very efficient as well, allowing for very deterministic behavior in our code.

Definition in file ll.h.

14.78 ll.h

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___  |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#ifndef __LL_H__
#define __LL_H__

#include "kerneltypes.h"

//---------------------------------------------------------------------------
#ifndef NULL
#define NULL        (0)
#endif

//---------------------------------------------------------------------------
#define SAFE_UNLINK     (0)

//---------------------------------------------------------------------------
class LinkList;
class DoubleLinkList;
class CircularLinkList;

//---------------------------------------------------------------------------
class LinkListNode
{
protected:

    LinkListNode *next;
    LinkListNode *prev;

    void ClearNode();

public:
    LinkListNode *GetNext(void) { return next; };

    LinkListNode *GetPrev(void) { return prev; };

    friend class LinkList;
    friend class DoubleLinkList;
    friend class CircularLinkList;
};

//---------------------------------------------------------------------------
class LinkList
{
protected:
    LinkListNode *m_pstHead;
    LinkListNode *m_pstTail;

public:
    void Init(){ m_pstHead = NULL; m_pstTail = NULL; }

    virtual void Add(LinkListNode *node_) = 0;

    virtual void Remove(LinkListNode *node_) = 0;

    LinkListNode *GetHead() { return m_pstHead; }

    LinkListNode *GetTail() { return m_pstTail; }
};

//---------------------------------------------------------------------------
class DoubleLinkList : public LinkList
{
public:
    DoubleLinkList() { m_pstHead = NULL; m_pstTail = NULL; };

    virtual void Add(LinkListNode *node_);

    virtual void Remove(LinkListNode *node_);
};

//---------------------------------------------------------------------------
class CircularLinkList : public LinkList
{
```

```
public:
    CircularLinkList() { m_pstHead = NULL; m_pstTail = NULL; };

    virtual void Add(LinkListNode *node_);

    virtual void Remove(LinkListNode *node_);

    void PivotForward();

    void PivotBackward();
};

#endif
```

14.79 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/manual.h File Reference

Ascii-format documentation, used by doxygen to create various printable and viewable forms.

14.79.1 Detailed Description

Ascii-format documentation, used by doxygen to create various printable and viewable forms.

Definition in file manual.h.

14.80 manual.h

```
/*===========================================================================
     _____     _____     _____     _____
 ___|    _|__  __|_    |__  __|__  |__  __| __  |__  _____
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||  \       ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||_______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
```

14.81 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/mark3cfg.h File Reference

Mark3 Kernel Configuration.

Macros

- #define KERNEL_USE_TIMERS (1)

  The following options is related to all kernel time-tracking.
- #define KERNEL_USE_QUANTUM (1)

  Do you want to enable time quanta? This is useful when you want to have tasks in the same priority group share time in a controlled way.
- #define KERNEL_USE_SEMAPHORE (1)

  Do you want the ability to use counting/binary semaphores for thread synchronization? Enabling this features provides fully-blocking semaphores and enables all API functions declared in semaphore.h.
- #define KERNEL_USE_MESSAGE (1)

  Enable inter-thread messaging using named mailboxes.

- #define KERNEL_USE_MUTEX (1)

    Do you want the ability to use mutual exclusion semaphores (mutex) for resource/block protection? Enabling this feature provides mutexes, with priority inheritence, as declared in mutex.h.
- #define KERNEL_USE_SLEEP (1)

    Do you want to be able to set threads to sleep for a specified time? This enables the Thread::Sleep() API.
- #define KERNEL_USE_DRIVER (1)

    Enabling device drivers provides a posix-like filesystem interface for peripheral device drivers.
- #define KERNEL_USE_THREADNAME (1)

    Provide Thread method to allow the user to set a name for each thread in the system.
- #define KERNEL_USE_DYNAMIC_THREADS (1)

    Provide extra Thread methods to allow the application to create (and more importantly destroy) threads at runtime.
- #define KERNEL_USE_PROFILER (1)

    Provides extra classes for profiling the performance of code.
- #define KERNEL_USE_DEBUG (0)

    Provides extra logic for kernel debugging, and instruments the kernel with extra asserts, and kernel trace functionality.

14.81.1 Detailed Description

Mark3 Kernel Configuration. This file is used to configure the kernel for your specific application in order to provide the optimal set of features for a given use case.

Since you only pay the price (code space/RAM) for the features you use, you can usually find a sweet spot between features and resource usage by picking and choosing features a-la-carte. This config file is written in an "interactive" way, in order to minimize confusion about what each option provides, and to make dependencies obvious.

As of 7.6.2012 on AVR, these are the costs associated with the various features:

Base Kernel: 2888 bytes Tickless Timers: 1194 bytes Semaphores: 224 bytes Message Queues: 332 bytes (+ Semaphores) Mutexes: 290 bytes Thread Sleep: 162 bytes (+ Semaphores/Timers) Round-Robin: 304 bytes (+ Timers) Drivers: 144 bytes Dynamic Threads: 68 bytes Thread Names: 8 bytes Profiling Timers: 624 bytes

Definition in file mark3cfg.h.

14.81.2 Macro Definition Documentation

14.81.2.1 #define KERNEL_USE_DRIVER (1)

Enabling device drivers provides a posix-like filesystem interface for peripheral device drivers.

When enabled, the size of the filesystem table is specified in DRIVER_TABLE_SIZE. Permissions are enforced for driver access by thread ID and group when DRIVER_USE_PERMS are enabled.

Definition at line 118 of file mark3cfg.h.

14.81.2.2 #define KERNEL_USE_DYNAMIC_THREADS (1)

Provide extra Thread methods to allow the application to create (and more importantly destroy) threads at runtime.

Useful for designs implementing worker threads, or threads that can be restarted after encountering error conditions.

Definition at line 133 of file mark3cfg.h.

14.81.2.3 #define KERNEL_USE_MESSAGE (1)

Enable inter-thread messaging using named mailboxes.

If per-thread mailboxes are defined, each thread is allocated a default mailbox of a depth specified by THREAD_M-AILBOX_SIZE.

Definition at line 88 of file mark3cfg.h.

14.81.2.4 #define KERNEL_USE_MUTEX (1)

Do you want the ability to use mutual exclusion semaphores (mutex) for resource/block protection? Enabling this feature provides mutexes, with priority inheritence, as declared in mutex.h.

Enabling per-thread mutex automatically allocates a mutex for each thread.

Definition at line 99 of file mark3cfg.h.

14.81.2.5 #define KERNEL_USE_PROFILER (1)

Provides extra classes for profiling the performance of code.

Useful for debugging and development, but uses an additional timer.

Definition at line 139 of file mark3cfg.h.

14.81.2.6 #define KERNEL_USE_QUANTUM (1)

Do you want to enable time quanta? This is useful when you want to have tasks in the same priority group share time in a controlled way.

This allows equal tasks to use unequal amounts of the CPU, which is a great way to set up CPU budgets per thread in a round-robin scheduling system. If enabled, you can specify a number of ticks that serves as the default time period (quantum). Unless otherwise specified, every thread in a priority will get the default quantum.

Definition at line 68 of file mark3cfg.h.

14.81.2.7 #define KERNEL_USE_SEMAPHORE (1)

Do you want the ability to use counting/binary semaphores for thread synchronization? Enabling this features provides fully-blocking semaphores and enables all API functions declared in semaphore.h.

If you have to pick one blocking mechanism, this is the one to choose. By also enabling per-thread semaphores, each thread will receive it's own built-in semaphore.

Definition at line 80 of file mark3cfg.h.

14.81.2.8 #define KERNEL_USE_THREADNAME (1)

Provide Thread method to allow the user to set a name for each thread in the system.

Adds to the size of the thread member data.

Definition at line 125 of file mark3cfg.h.

14.81.2.9 #define KERNEL_USE_TIMERS (1)

The following options is related to all kernel time-tracking.

-timers provide a way for events to be periodically triggered in a lightweight manner. These can be periodic, or one-shot.

-Thread Quantum (used for round-robin scheduling) is dependent on this module, as is Thread Sleep functionality.

Definition at line 56 of file mark3cfg.h.

14.82 mark3cfg.h

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#ifndef __MARK3CFG_H__
#define __MARK3CFG_H__

#define KERNEL_USE_TIMERS               (1)

#if KERNEL_USE_TIMERS
    #define KERNEL_USE_QUANTUM          (1)
#else
    #define KERNEL_USE_QUANTUM          (0)
#endif

#define KERNEL_USE_SEMAPHORE            (1)

#if KERNEL_USE_SEMAPHORE
    #define KERNEL_USE_MESSAGE          (1)
#else
    #define KERNEL_USE_MESSAGE          (0)
#endif

#define KERNEL_USE_MUTEX                (1)

#if KERNEL_USE_TIMERS && KERNEL_USE_SEMAPHORE
    #define KERNEL_USE_SLEEP            (1)
#else
    #define KERNEL_USE_SLEEP            (0)
#endif


#define KERNEL_USE_DRIVER               (1)

#define KERNEL_USE_THREADNAME           (1)

#define KERNEL_USE_DYNAMIC_THREADS      (1)

#define KERNEL_USE_PROFILER             (1)

#define KERNEL_USE_DEBUG                (0)


#endif
```

14.83 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/memutil.cpp File Reference

Implementation of memory, string, and conversion routines.

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "kernel_debug.h"
#include "memutil.h"
```

14.83.1 Detailed Description

Implementation of memory, string, and conversion routines.

Definition in file memutil.cpp.

14.84 memutil.cpp

```
/*===========================================================================
     _____     _____     _____     _____
 ___|    _|__ __|_    |__ __|__   |__ __| __  |__  ______
|    \  /  | ||    \    ||     |     ||  |/ /     ||___   |
|     \/   | ||     \   ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "kernel_debug.h"
#include "memutil.h"

//---------------------------------------------------------------------------
void MemUtil::DecimalToHex( K_UCHAR ucData_, char *szText_
 )
{
    K_UCHAR ucTmp = ucData_;
    K_UCHAR ucMax;

    KERNEL_ASSERT( szText_ );

    if (ucTmp >= 0x10)
    {
        ucMax = 2;
    }
    else
    {
        ucMax = 1;
    }

    ucTmp = ucData_;
    szText_[ucMax] = 0;
    while (ucMax--)
    {
        if ((ucTmp & 0x0F) <= 9)
        {
            szText_[ucMax] = '0' + (ucTmp & 0x0F);
        }
        else
        {
            szText_[ucMax] = 'A' + ((ucTmp & 0x0F) - 10);
        }
        ucTmp>>=4;
    }
}

//---------------------------------------------------------------------------
void MemUtil::DecimalToString( K_UCHAR ucData_, char *
szText_ )
{
    K_UCHAR ucTmp = ucData_;
    K_UCHAR ucMax;

    KERNEL_ASSERT(szText_);

    // Find max index to print...
    if (ucData_ >= 100)
    {
        ucMax = 3;
    }
    else if (ucData_ >= 10)
    {
        ucMax = 2;
    }
    else
    {
        ucMax = 1;
    }

    szText_[ucMax] = 0;
    while (ucMax--)
    {
        szText_[ucMax] = '0' + (ucTmp % 10);
        ucTmp/=10;
    }
}

//---------------------------------------------------------------------------
```

```
// Basic checksum routines
K_UCHAR  MemUtil::Checksum8( const void *pvSrc_, K_USHORT
usLen_ )
{
    K_UCHAR ucRet = 0;
    K_UCHAR *pcData = (K_UCHAR*)pvSrc_;

    KERNEL_ASSERT(pvSrc_);

    // 8-bit CRC, computed byte at a time
    while (usLen_--)
    {
        ucRet += *pcData++;
    }
    return ucRet;
}

//---------------------------------------------------------------------------
K_USHORT MemUtil::Checksum16( const void *pvSrc_, K_USHORT
usLen_ )
{
    K_USHORT usRet = 0;
    K_UCHAR *pcData = (K_UCHAR*)pvSrc_;

    KERNEL_ASSERT(pvSrc_);

    // 16-bit CRC, computed byte at a time
    while (usLen_--)
    {
        usRet += *pcData++;
    }
    return usRet;
}

//---------------------------------------------------------------------------
// Basic string routines
K_USHORT MemUtil::StringLength( const char *szStr_ )
{
    K_UCHAR *pcData = (K_UCHAR*)szStr_;
    K_USHORT usLen = 0;

    KERNEL_ASSERT(szStr_);

    while (*pcData++)
    {
        usLen++;
    }
    return usLen;
}

//---------------------------------------------------------------------------
bool    MemUtil::CompareStrings( const char *szStr1_,
const char *szStr2_ )
{
    char *szTmp1 = (char*) szStr1_;
    char *szTmp2 = (char*) szStr2_;

    KERNEL_ASSERT(szStr1_);
    KERNEL_ASSERT(szStr2_);

    while (*szTmp1 && *szTmp2)
    {
        if (*szTmp1++ != *szTmp2++)
        {
            return false;
        }
    }

    // Both terminate at the same length
    if (!(*szTmp1) && !(*szTmp2))
    {
        return true;
    }

    return false;
}

//---------------------------------------------------------------------------
void MemUtil::CopyMemory( void *pvDst_, const void *pvSrc_,
K_USHORT usLen_ )
{
    char *szDst = (char*) pvDst_;
    char *szSrc = (char*) pvSrc_;

    KERNEL_ASSERT(pvDst_);
    KERNEL_ASSERT(pvSrc_);

    // Run through the strings verifying that each character matches
    // and the lengths are the same.
    while (usLen_--)
    {
        *szDst++ = *szSrc++;
    }
}

//---------------------------------------------------------------------------
void MemUtil::CopyString( char *szDst_, const char *szSrc_ )
{
    char *szDst = (char*) szDst_;
    char *szSrc = (char*) szSrc_;

    KERNEL_ASSERT(szDst_);
    KERNEL_ASSERT(szSrc_);

    // Run through the strings verifying that each character matches
    // and the lengths are the same.
    while (*szSrc)
    {
        *szDst++ = *szSrc++;
    }
}

//---------------------------------------------------------------------------
K_SHORT MemUtil::StringSearch( const char *szBuffer_,
const char *szPattern_ )
{
    char *szTmpPat = (char*)szPattern_;
    K_SHORT i16Idx = 0;
    K_SHORT i16Start;
    KERNEL_ASSERT( szBuffer_ );
    KERNEL_ASSERT( szPattern_ );

    // Run through the big buffer looking for a match of the pattern
    while (szBuffer_[i16Idx])
    {
        // Reload the pattern
        i16Start = i16Idx;
        szTmpPat = (char*)szPattern_;
        while (*szTmpPat && szBuffer_[i16Idx])
        {
            if (*szTmpPat != szBuffer_[i16Idx])
            {
                break;
            }
            szTmpPat++;
            i16Idx++;
        }
        // Made it to the end of the pattern, it's a match.
        if (*szTmpPat == '\0')
        {
            return i16Start;
        }
        i16Idx++;
    }

    return -1;
}

//---------------------------------------------------------------------------
bool MemUtil::CompareMemory( const void *pvMem1_, const
void *pvMem2_, K_USHORT usLen_ )
{
    char *szTmp1 = (char*) pvMem1_;
    char *szTmp2 = (char*) pvMem2_;

    KERNEL_ASSERT(pvMem1_);
    KERNEL_ASSERT(pvMem2_);

    // Run through the strings verifying that each character matches
    // and the lengths are the same.
    while (usLen_--)
    {
        if (*szTmp1++ != *szTmp2++)
        {
            return false;
        }
    }
    return true;
}

//---------------------------------------------------------------------------
void MemUtil::SetMemory( void *pvDst_, K_UCHAR ucVal_,
K_USHORT usLen_ )
{
    char *szDst = (char*)pvDst_;

    KERNEL_ASSERT(pvDst_);

    while (usLen_--)
    {
        *szDst++ = ucVal_;
    }
}



```

14.85 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/memutil.h File Reference

Utility class containing memory, string, and conversion routines.

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "kernel_debug.h"
```

Classes

- class MemUtil
  String and Memory manipulation class.

14.85.1 Detailed Description

Utility class containing memory, string, and conversion routines.

Definition in file memutil.h.

14.86 memutil.h

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#ifndef __MEMUTIL_H__
#define __MEMUTIL_H__

#include "kerneltypes.h"
#include "mark3cfg.h"
#include "kernel_debug.h"

//---------------------------------------------------------------------------
class MemUtil
{

public:

    //-----------------------------------------------------------------------
    static void DecimalToHex( K_UCHAR ucData_, char *szText_ );

    //-----------------------------------------------------------------------
    static void DecimalToString( K_UCHAR ucData_, char *szText_
);
```

```
    //---------------------------------------------------------------------------
    static K_UCHAR  Checksum8( const void *pvSrc_, K_USHORT usLen_ );

    //---------------------------------------------------------------------------
    static K_USHORT Checksum16( const void *pvSrc_, K_USHORT usLen_ )
;

    //---------------------------------------------------------------------------
    static K_USHORT StringLength( const char *szStr_ );

    //---------------------------------------------------------------------------
    static bool CompareStrings( const char *szStr1_, const char *
szStr2_ );

    //---------------------------------------------------------------------------
    static void CopyMemory( void *pvDst_, const void *pvSrc_,
K_USHORT usLen_ );

    //---------------------------------------------------------------------------
    static void CopyString( char *szDst_, const char *szSrc_ );

    //---------------------------------------------------------------------------
    static K_SHORT StringSearch( const char *szBuffer_, const char
*szPattern_ );

    //---------------------------------------------------------------------------
    static bool CompareMemory( const void *pvMem1_, const void *
pvMem2_, K_USHORT usLen_ );

    //---------------------------------------------------------------------------
    static void SetMemory( void *pvDst_, K_UCHAR ucVal_, K_USHORT
usLen_ );

};


#endif //__MEMUTIL_H__




```

14.87 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/message.cpp File Reference

Inter-thread communications via message passing.

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "message.h"
#include "threadport.h"
#include "kernel_debug.h"
#include "timerlist.h"
```

Macros

- #define **__FILE_ID__** MESSAGE_CPP

14.87.1 Detailed Description

Inter-thread communications via message passing.

Definition in file message.cpp.

14.88 message.cpp

```
/*===========================================================================
     _____     _____     _____     _____
 ___|    _|__ __|_    |__ __|__   |__ __| __  |__  _____
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||_____|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "kerneltypes.h"
#include "mark3cfg.h"

#include "message.h"
#include "threadport.h"
#include "kernel_debug.h"

//---------------------------------------------------------------------------
#if defined __FILE_ID__
    #undef __FILE_ID__
#endif
#define __FILE_ID__     MESSAGE_CPP


#if KERNEL_USE_MESSAGE

#if KERNEL_USE_TIMERS
    #include "timerlist.h"
#endif

Message GlobalMessagePool::m_aclMessagePool[8];
DoubleLinkList GlobalMessagePool::m_clList;

//---------------------------------------------------------------------------
void GlobalMessagePool::Init()
{
    K_UCHAR i;
    for (i = 0; i < 8; i++)
    {
        GlobalMessagePool::m_aclMessagePool[i].Init();
        GlobalMessagePool::m_clList.Add(&(GlobalMessagePool::m_aclMessagePool[i]));
    }
}

//---------------------------------------------------------------------------
void GlobalMessagePool::Push( Message *pclMessage_ )
{
    KERNEL_ASSERT( pclMessage_ );

    CS_ENTER();

    GlobalMessagePool::m_clList.Add(pclMessage_);

    CS_EXIT();
}

//---------------------------------------------------------------------------
Message *GlobalMessagePool::Pop()
{
    Message *pclRet;
    CS_ENTER();

    pclRet = static_cast<Message*>( GlobalMessagePool::m_clList.GetHead() );
    GlobalMessagePool::m_clList.Remove( static_cast<LinkListNode*>( pclRet ) );

    CS_EXIT();
    return pclRet;
}

//---------------------------------------------------------------------------
void MessageQueue::Init()
{
    m_clSemaphore.Init(0, 100);
}
```

```

//---------------------------------------------------------------------------
Message *MessageQueue::Receive()
{
    Message *pclRet;

    // Block the current thread on the counting semaphore
    m_clSemaphore.Pend();

    CS_ENTER();

    // Pop the head of the message queue and return it
    pclRet = static_cast<Message*>( m_clLinkList.GetHead
() );
    m_clLinkList.Remove(static_cast<Message*>(pclRet));

    CS_EXIT();

    return pclRet;
}

#if KERNEL_USE_TIMERS
//---------------------------------------------------------------------------
Message *MessageQueue::Receive( K_ULONG
ulTimeWaitMS_ )
{
    Message *pclRet;

    // Block the current thread on the counting semaphore
    if (!m_clSemaphore.Pend(ulTimeWaitMS_))
    {
        return NULL;
    }

    CS_ENTER();

    // Pop the head of the message queue and return it
    pclRet = static_cast<Message*>( m_clLinkList.GetHead
() );
    m_clLinkList.Remove(static_cast<Message*>(pclRet));

    CS_EXIT();

    return pclRet;
}
#endif
//---------------------------------------------------------------------------
void MessageQueue::Send( Message *pclSrc_ )
{
    KERNEL_ASSERT( pclSrc_ );

    CS_ENTER();

    // Add the message to the head of the linked list
    m_clLinkList.Add( pclSrc_ );

    // Post the semaphore, waking the blocking thread for the queue.
    m_clSemaphore.Post();

    CS_EXIT();
}


//---------------------------------------------------------------------------
K_USHORT MessageQueue::GetCount()
{
    return m_clSemaphore.GetCount();
}
#endif //KERNEL_USE_MESSAGE
```

14.89 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/message.h File Reference

Inter-thread communication via message-passing.

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "ll.h"
#include "semaphore.h"
#include "timerlist.h"
```

Classes

- class Message

    Class to provide message-based IPC services in the kernel.

- class GlobalMessagePool

    Implements a list of message objects shared between all threads.

- class MessageQueue

    List of messages, used as the channel for sending and receiving messages between threads.

14.89.1 Detailed Description

Inter-thread communication via message-passing. Embedded systems guru Jack Ganssle once said that without a robust form of interprocess communications (IPC), an RTOS is just a toy. Mark3 implements a form of IPC to provide safe and flexible messaging between threads.

Using kernel-managed IPC offers significant benefits over other forms of data sharing (i.e. Global variables) in that it avoids synchronization issues and race conditions common to the practice. Using IPC also enforces a more disciplined coding style that keeps threads decoupled from one another and minimizes global data preventing careless and hard-to-debug errors.

14.89.2 Using Messages, Queues, and the Global Message Pool

```
// Declare a message queue shared between two threads
MessageQueue my_queue;

int main()
{
    ...
    // Initialize the message queue
    my_queue.init();
    ...
}

void Thread1()
{
    // Example TX thread - sends a message every 10ms
    while(1)
    {
        // Grab a message from the global message pool
        Message *tx_message = GlobalMessagePool::Pop
      ();

        // Set the message data/parameters
        tx_message->SetCode( 1234 );
        tx_message->SetData( NULL );

        // Send the message on the queue.
        my_queue.Send( tx_message );
        Thread::Sleep(10);
    }
}

void Thread2()
{
    while()
    {
        // Blocking receive - wait until we have messages to process
        Message *rx_message = my_queue.Recv();

        // Do something with the message data...

        // Return back into the pool when done
        GlobalMessagePool::Push(rx_message);

    }
}
```

Definition in file message.h.

14.90 message.h

```
/*===========================================================================
     _____     _____     _____     _____
 ___|    _|__ __|_    |__ __|__   |__ __| __  |__  ______
|    \  /  | ||    \     ||     |     ||  |/ /     ||___   |
|     \/   | ||     \    ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#ifndef __MESSAGE_H__
#define __MESSAGE_H__

#include "kerneltypes.h"
#include "mark3cfg.h"

#include "ll.h"
#include "semaphore.h"

#if KERNEL_USE_MESSAGE

#if KERNEL_USE_TIMERS
    #include "timerlist.h"
#endif

//---------------------------------------------------------------------------
class Message : public LinkListNode
{
public:
    void Init() { m_pvData = NULL; m_usCode = 0; }

    void SetData( void *pvData_ ) { m_pvData = pvData_; }

    void *GetData() { return m_pvData; }

    void SetCode( K_USHORT usCode_ ) { m_usCode = usCode_; }

    K_USHORT GetCode() { return m_usCode; }
private:

    void *m_pvData;

    K_USHORT m_usCode;
};

//---------------------------------------------------------------------------
class GlobalMessagePool
{
public:
    static void Init();

    static void Push( Message *pclMessage_ );

    static Message *Pop();

private:
    static Message m_aclMessagePool[8];

    static DoubleLinkList m_clList;
};

//---------------------------------------------------------------------------
class MessageQueue
{
public:
    void Init();

    Message *Receive();

#if KERNEL_USE_TIMERS
```

```

    Message *Receive( K_ULONG ulTimeWaitMS_ );
#endif

    void Send( Message *pclSrc_ );


    K_USHORT GetCount();
private:

    Semaphore m_clSemaphore;

    DoubleLinkList m_clLinkList;
};

#endif //KERNEL_USE_MESSAGE

#endif
```

14.91 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/mutex.cpp File Reference

Mutual-exclusion object.

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "blocking.h"
#include "mutex.h"
#include "kernel_debug.h"
```

Macros

- #define **__FILE_ID__** MUTEX_CPP

Functions

- void **TimedMutex_Calback** (Thread *pclOwner_, void *pvData_)

14.91.1 Detailed Description

Mutual-exclusion object.

Definition in file mutex.cpp.

14.92 mutex.cpp

```
/*============================================================================
     _____     _____     _____     _____
 ___|    _|__ __|_    |__ __|__   |__ __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|     |_____|     |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "kerneltypes.h"
#include "mark3cfg.h"

#include "blocking.h"
```

```
#include "mutex.h"
#include "kernel_debug.h"
//---------------------------------------------------------------------------
#if defined __FILE_ID__
    #undef __FILE_ID__
#endif
#define __FILE_ID__     MUTEX_CPP


#if KERNEL_USE_MUTEX

#if KERNEL_USE_TIMERS

//---------------------------------------------------------------------------
void TimedMutex_Calback(Thread *pclOwner_, void *pvData_)
{
    Mutex *pclMutex = static_cast<Mutex*>(pvData_);

    // Indicate that the semaphore has expired on the thread
    pclMutex->SetExpired(true);

    // Wake up the thread that was blocked on this semaphore.
    pclMutex->WakeMe(pclOwner_);

    if (pclOwner_->GetPriority() > Scheduler::GetCurrentThread
()->GetPriority())
    {
        Thread::Yield();
    }
}

//---------------------------------------------------------------------------
void Mutex::WakeMe(Thread *pclOwner_)
{
    // Remove from the semaphore waitlist and back to its ready list.
    UnBlock(pclOwner_);
}

#endif

//---------------------------------------------------------------------------
K_UCHAR Mutex::WakeNext()
{
    Thread *pclChosenOne = NULL;

    // Get the highest priority waiter thread
    pclChosenOne = m_clBlockList.HighestWaiter();

    // Unblock the thread
    UnBlock(pclChosenOne);

    // The chosen one now owns the mutex
    m_pclOwner = pclChosenOne;

    // Signal a context switch if it’s a greater than or equal to the current
 priority
    if (pclChosenOne->GetPriority() >= Scheduler::GetCurrentThread
()->GetPriority())
    {
        return 1;
    }
    return 0;
}

//---------------------------------------------------------------------------
void Mutex::Init()
{
    // Reset the data in the mutex
    m_bReady = 1;             // The mutex is free.
    m_ucMaxPri = 0;           // Set the maximum priority inheritence state
    m_pclOwner = NULL;        // Clear the mutex owner
}

//---------------------------------------------------------------------------
#if KERNEL_USE_TIMERS
    void Mutex::Claim()
    {
        Claim(0);
    }
    bool Mutex::Claim(K_ULONG ulWaitTimeMS_)
#else
    void Mutex::Claim()
#endif
{
    KERNEL_TRACE_1( STR_MUTEX_CLAIM_1, (K_USHORT)g_pstCurrent->GetID() );

    K_UCHAR bSchedule = 0;
    Thread *pclThread;

#if KERNEL_USE_TIMERS
    Timer clTimer;

    m_bExpired = false;
#endif

    // Disable the scheduler while claiming the mutex - we're dealing with all
    // sorts of private thread data, can't have a thread switch while messing
    // with internal data structures.
    Scheduler::SetScheduler(0);

    // Get the current thread pointer
    pclThread = Scheduler::GetCurrentThread();

    // Check to see if the mutex is claimed or not
    if (m_bReady != 0)
    {
        // Mutex isn't claimed, claim it.
        m_bReady = 0;
        m_ucMaxPri = pclThread->GetPriority();
        m_pclOwner = pclThread;
    }
    else
    {
        // The mutex is claimed already - we have to block now.  Move the
        // current thread to the list of threads waiting on the mutex.
#if KERNEL_USE_TIMERS
        if (ulWaitTimeMS_)
        {
            clTimer.Start(0, ulWaitTimeMS_, (TimerCallback_t)
TimedMutex_Calback, (void*)this);
        }
#endif

        Block(pclThread);

        // Check if priority inheritence is necessary.  We do this in order
        // to ensure that we don't end up with priority inversions in case
        // multiple threads are waiting on the same resource.
        if(m_ucMaxPri <= pclThread->GetPriority())
        {
            m_ucMaxPri = pclThread->GetPriority();

            {
                Thread *pclTemp = static_cast<Thread*>(
m_clBlockList.GetHead());
                while(pclTemp)
                {
                    pclTemp->InheritPriority(m_ucMaxPri);
                    if(pclTemp == static_cast<Thread*>(m_clBlockList.GetTail())
 )
                    {
                        break;
                    }
                    pclTemp = static_cast<Thread*>(pclTemp->GetNext
());

                }
            }
        }

        // Switch Threads when we exit the critical section.
        bSchedule = 1;
    }

    // Done with thread data -reenable the scheduler
    Scheduler::SetScheduler(1);

    if (bSchedule)
    {
        // Switch threads if this thread acquired the mutex
        Thread::Yield();
    }

#if KERNEL_USE_TIMERS
    if (ulWaitTimeMS_)
    {
        clTimer.Stop();
    }
    return (m_bExpired == 0);
#endif
}

//---------------------------------------------------------------------------
void Mutex::Release()
```

```
{
    KERNEL_TRACE_1( STR_MUTEX_RELEASE_1, (K_USHORT)g_pstCurrent->GetID() )
;

    K_UCHAR bSchedule = 0;
    Thread *pclThread;

    // Disable the scheduler while we deal with internal data structures.
    Scheduler::SetScheduler(0);
    pclThread = Scheduler::GetCurrentThread();

    // Restore the thread's original priority
    if (pclThread->GetCurPriority() != pclThread->GetPriority())
    {
        pclThread->SetPriority(pclThread->GetPriority());

        // In this case, we want to reschedule
        bSchedule = 1;
    }

    // No threads are waiting on this semaphore?
    if (m_clBlockList.GetHead() == NULL)
    {
        // Re-initialize the mutex to its default values
        m_bReady = 1;
        m_ucMaxPri = 0;
        m_pclOwner = NULL;
    }
    else
    {
        // Wake the highest priority Thread pending on the mutex
        if(WakeNext())
        {
            // Switch threads if it's higher or equal priority than the current
 thread
            bSchedule = 1;
        }
    }

    // Must enable the scheduler again in order to switch threads.
    Scheduler::SetScheduler(1);
    if(bSchedule)
    {
        // Switch threads if a higher-priority thread was woken
        Thread::Yield();
    }
}

#endif //KERNEL_USE_MUTEX
```

14.93 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/mutex.h File Reference

Mutual exclusion class declaration.

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "blocking.h"
#include "timerlist.h"
```

Classes

- class Mutex

  Mutual-exclusion locks, based on BlockingObject.

14.93.1 Detailed Description

Mutual exclusion class declaration. Resource locks are implemented using mutual exclusion semaphores (Mutex-_t). Protected blocks can be placed around any resource that may only be accessed by one thread at a time. If additional threads attempt to access the protected resource, they will be placed in a wait queue until the resource becomes available. When the resource becomes available, the thread with the highest original priority claims the resource and is activated. Priority inheritance is included in the implementation to prevent priority inversion. Always ensure that you claim and release your mutex objects consistently, otherwise you may end up with a deadlock scenario that's hard to debug.

14.93.2 Initializing

Initializing a mutex object by calling:

```
clMutex.Init();
```

14.93.3 Resource protection example

```
clMutex.Claim();
...
<resource protected block>
...
clMutex.Release();
```

Definition in file mutex.h.

14.94 mutex.h

```
/*===========================================================================
     _____       _____       _____       _____
 ___|   _|__  __|_    |__ __|__   |__ __| __  |__ ______
|    \  / | ||    \     ||     |     ||  |/ /    ||___   |
|     \/  | ||     \    ||     \     ||    \     ||___  |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|     |_____|     |_____|     |_____|

--[Mark3 Realtime Platform]-------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#ifndef __MUTEX_H_
#define __MUTEX_H_

#include "kerneltypes.h"
#include "mark3cfg.h"

#include "blocking.h"

#if KERNEL_USE_MUTEX

#if KERNEL_USE_TIMERS
#include "timerlist.h"
#endif

//---------------------------------------------------------------------------
class Mutex : public BlockingObject
{
public:
    void Init();

    void Claim();

#if KERNEL_USE_TIMERS

    bool Claim(K_ULONG ulWaitTimeMS_);

    void WakeMe( Thread *pclOwner_ );

    void SetExpired( bool bExpired_ ) { m_bExpired = bExpired_; }
#endif

    void Release();

private:

    K_UCHAR WakeNext();

    K_UCHAR m_bReady;
    K_UCHAR m_ucMaxPri;
    Thread *m_pclOwner;

#if KERNEL_USE_TIMERS
    bool    m_bExpired;
#endif
};

#endif //KERNEL_USE_MUTEX

#endif //__MUTEX_H_

```

14.95 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/profile.cpp File Reference

Code profiling utilities.

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "profile.h"
#include "kprofile.h"
#include "threadport.h"
#include "kernel_debug.h"
```

Macros

- #define **__FILE_ID__** PROFILE_CPP

14.95.1 Detailed Description

Code profiling utilities.

Definition in file profile.cpp.

14.96 profile.cpp

```
/*============================================================================
     _____     _____     _____     _____
 ___|    _|__ __|_    |__ __|__   |__ __| __  |__ _____
|    \  /  | ||    \    ||     |    ||  |/ /    ||___   |
|     \/   | ||     \   ||     \    ||     \    ||___   |
|__/\__/|__|_||__|\__\ __||__|\__\ __||__|\__\ __||_____|
    |_____|      |_____|     |_____|     |_____|

--[Mark3 Realtime Platform]---------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
============================================================================*/
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "profile.h"
#include "kprofile.h"
#include "threadport.h"
#include "kernel_debug.h"
//----------------------------------------------------------------------------
#if defined __FILE_ID__
    #undef __FILE_ID__
#endif
#define __FILE_ID__     PROFILE_CPP


#if KERNEL_USE_PROFILER

```

```
//---------------------------------------------------------------------------
void ProfileTimer::Init()
{
    m_ulCumulative = 0;
    m_ulCurrentIteration = 0;
    m_usIterations = 0;
    m_bActive = 0;
}

//---------------------------------------------------------------------------
void ProfileTimer::Start()
{
    if (!m_bActive)
    {
        CS_ENTER();
        m_ulCurrentIteration = 0;
        m_ulInitialEpoch = Profiler::GetEpoch();
        m_usInitial = Profiler::Read();
        CS_EXIT();
        m_bActive = 1;
    }
}

//---------------------------------------------------------------------------
void ProfileTimer::Stop()
{
    if (m_bActive)
    {
        K_USHORT usFinal;
        K_ULONG ulEpoch;
        CS_ENTER();
        usFinal = Profiler::Read();
        ulEpoch = Profiler::GetEpoch();
        // Compute total for current iteration...
        m_ulCurrentIteration = ComputeCurrentTicks(usFinal, ulEpoch);
        m_ulCumulative += m_ulCurrentIteration;
        m_usIterations++;
        CS_EXIT();
        m_bActive = 0;
    }
}

//---------------------------------------------------------------------------
K_ULONG ProfileTimer::GetAverage()
{
    if (m_usIterations)
    {
        return m_ulCumulative / (K_ULONG)m_usIterations;
    }
    return 0;
}

//---------------------------------------------------------------------------
K_ULONG ProfileTimer::GetCurrent()
{

    if (m_bActive)
    {
        K_USHORT usCurrent;
        K_ULONG ulEpoch;
        CS_ENTER();
        usCurrent = Profiler::Read();
        ulEpoch = Profiler::GetEpoch();
        CS_EXIT();
        return ComputeCurrentTicks(usCurrent, ulEpoch);
    }
    return m_ulCurrentIteration;
}

//---------------------------------------------------------------------------
K_ULONG ProfileTimer::ComputeCurrentTicks(K_USHORT usCurrent_, K_ULONG ulEpoch_)
{
    K_ULONG ulTotal;
    K_ULONG ulOverflows;

    ulOverflows = ulEpoch_ - m_ulInitialEpoch;

    // More than one overflow...
    if (ulOverflows > 1)
    {
        ulTotal = ((K_ULONG)(ulOverflows-1) * TICKS_PER_OVERFLOW)
                + (K_ULONG)(TICKS_PER_OVERFLOW - m_usInitial) +
                (K_ULONG)usCurrent_;
    }
    // Only one overflow, or one overflow that has yet to be processed
    else if (ulOverflows || (usCurrent_ < m_usInitial))
    {
        ulTotal = (K_ULONG)(TICKS_PER_OVERFLOW - m_usInitial) +
                (K_ULONG)usCurrent_;
    }
    // No overflows, none pending.
    else
    {
        ulTotal = (K_ULONG)(usCurrent_ - m_usInitial);
    }

    return ulTotal;
}

#endif
```

14.97 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/profile.h File Reference

High-precision profiling timers.

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "ll.h"
```

Classes

- class ProfileTimer

  Profiling timer.

14.97.1 Detailed Description

High-precision profiling timers. Enables the profiling and instrumentation of performance-critical code. Multiple timers can be used simultaneously to enable system-wide performance metrics to be computed in a lightweight manner.

Usage:

```
ProfileTimer clMyTimer;
int i;

clMyTimer.Init();

// Profile the same block of code ten times
for (i = 0; i < 10; i++)
{
    clMyTimer.Start();
    ...
    //Block of code to profile
    ...
    clMyTimer.Stop();
}

// Get the average execution time of all iterations
ulAverageTimer = clMyTimer.GetAverage();

// Get the execution time from the last iteration
ulLastTimer = clMyTimer.GetCurrent();
```

Definition in file profile.h.

14.98 profile.h

```
/*===========================================================================
     _____       _____       _____       _____
 ___|    _|__  __|_    |__ __|__   |__ __| __  |__  ______
|    \  /  | ||   \     ||     |    ||  |/ /    ||___   |
|     \/   | ||    \    ||     \    ||     \    ||___   |
|__/\__/|__|_||__|\__\ __||__|\__\ __||__|\__\ __||______|
    |_____|     |_____|     |_____|     |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#ifndef __PROFILE_H__
#define __PROFILE_H__

#include "kerneltypes.h"
#include "mark3cfg.h"
#include "ll.h"

#if KERNEL_USE_PROFILER

class ProfileTimer
{

public:
    void Init();

    void Start();

    void Stop();

    K_ULONG GetAverage();

    K_ULONG GetCurrent();

private:

    K_ULONG ComputeCurrentTicks(K_USHORT usCount_, K_ULONG
ulEpoch_);

    K_ULONG m_ulCumulative;
    K_ULONG m_ulCurrentIteration;
    K_USHORT m_usInitial;
    K_ULONG m_ulInitialEpoch;
    K_USHORT m_usIterations;
    K_UCHAR m_bActive;
};

#endif // KERNEL_USE_PROFILE

#endif
```

14.99 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/quantum.cpp File Reference

Thread Quantum Implementation for Round-Robin Scheduling.

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "timerlist.h"
#include "thread.h"
#include "quantum.h"
#include "kernel_debug.h"
```

Macros

- #define **__FILE_ID__** QUANTUM_CPP

Functions

- static void **QuantumCallback** (Thread ∗pclThread_, void ∗pvData_)

Variables

- static volatile K_BOOL **bAddQuantumTimer**

14.99.1 Detailed Description

Thread Quantum Implementation for Round-Robin Scheduling.

Definition in file quantum.cpp.

14.100 quantum.cpp

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "kerneltypes.h"
#include "mark3cfg.h"

#include "timerlist.h"
#include "thread.h"
#include "quantum.h"
#include "kernel_debug.h"
//---------------------------------------------------------------------------
#if defined __FILE_ID__
    #undef __FILE_ID__
#endif
#define __FILE_ID__     QUANTUM_CPP

#if KERNEL_USE_QUANTUM

//---------------------------------------------------------------------------
static volatile K_BOOL bAddQuantumTimer;    // Indicates that a timer add is pending

//---------------------------------------------------------------------------
Timer Quantum::m_clQuantumTimer;    // The global timernodelist_t object
K_UCHAR Quantum::m_bActive;
//---------------------------------------------------------------------------
static void QuantumCallback(Thread *pclThread_, void *pvData_)
{
    // Validate thread pointer, check that source/destination match (it's
    // in its real priority list).  Also check that this thread was part of
    // the highest-running priority level.
    if (pclThread_->GetPriority() >= Scheduler::GetCurrentThread()->GetPriority())
    {
        if (pclThread_->GetCurrent()->GetHead() != pclThread_->GetCurrent()->GetTail() )
        {
            bAddQuantumTimer = true;
            pclThread_->GetCurrent()->PivotForward();
        }
    }
}

//---------------------------------------------------------------------------
void Quantum::SetTimer(Thread *pclThread_)
{
    m_clQuantumTimer.SetIntervalMSeconds(pclThread_->GetQuantum());
    m_clQuantumTimer.SetFlags(TIMERLIST_FLAG_ONE_SHOT
      );
    m_clQuantumTimer.SetData(NULL);
    m_clQuantumTimer.SetCallback((TimerCallback_t)QuantumCallback);
    m_clQuantumTimer.SetOwner(pclThread_);
}

//---------------------------------------------------------------------------
void Quantum::AddThread(Thread *pclThread_)
{
    if (m_bActive)
    {
        return;
    }
    // If this isn't the only thread in the list.
    if ( pclThread_->GetCurrent()->GetHead() !=
           pclThread_->GetCurrent()->GetTail() )
    {
        Quantum::SetTimer(pclThread_);
        TimerScheduler::Add(&m_clQuantumTimer);
        m_bActive = 1;
    }
}

//---------------------------------------------------------------------------
void Quantum::RemoveThread(void)
{
    if (!m_bActive)
    {
        return;
    }

    // Cancel the current timer
    TimerScheduler::Remove(&m_clQuantumTimer);
    m_bActive = 0;
}

//---------------------------------------------------------------------------
void Quantum::UpdateTimer(void)
{
    // If we have to re-add the quantum timer (more than 2 threads at the
    // high-priority level...)
    if (bAddQuantumTimer)
    {
        // Trigger a thread yield - this will also re-schedule the
        // thread *and* reset the round-robin scheduler.
        Thread::Yield();
        bAddQuantumTimer = false;
    }
}

#endif //KERNEL_USE_QUANTUM
```

14.101 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/quantum.h File Reference

Thread Quantum declarations for Round-Robin Scheduling.

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "thread.h"
#include "timerlist.h"
```

Classes

- class Quantum

  Static-class used to implement Thread quantum functionality, which is a key part of round-robin scheduling.

14.101.1 Detailed Description

Thread Quantum declarations for Round-Robin Scheduling.

Definition in file quantum.h.

14.102 quantum.h

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#ifndef __KQUANTUM_H__
#define __KQUANTUM_H__

#include "kerneltypes.h"
#include "mark3cfg.h"

#include "thread.h"
#include "timerlist.h"

#if KERNEL_USE_QUANTUM
class Timer;

class Quantum
{
public:
    static void UpdateTimer();

    static void AddThread( Thread *pclThread_ );

    static void RemoveThread();

private:
    static void SetTimer( Thread *pclThread_ );

    static Timer m_clQuantumTimer;
    static K_UCHAR m_bActive;
};

#endif //KERNEL_USE_QUANTUM

#endif
```

14.103 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/scheduler.cpp File Reference

Strict-Priority + Round-Robin thread scheduler implementation.

```
#include "kerneltypes.h"
#include "ll.h"
#include "scheduler.h"
#include "thread.h"
#include "kernel_debug.h"
```

Macros

- #define **__FILE_ID__** SCHEDULER_CPP

Variables

- Thread ∗ **g_pstNext**

- Thread ∗ **g_pstCurrent**
- K_UCHAR **g_ucFlag**
- static const K_UCHAR **aucCLZ** [16] = {-1,0,1,1,2,2,2,2,3,3,3,3,3,3,3,3}

14.103.1 Detailed Description

Strict-Priority + Round-Robin thread scheduler implementation.

Definition in file scheduler.cpp.

14.104 scheduler.cpp

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_     |__  __|__    |__  __| __   |__  ______
|    \  /  | ||    \       ||     |      ||  |/ /     ||___   |
|     \/   | ||     \      ||     \      ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "kerneltypes.h"
#include "ll.h"
#include "scheduler.h"
#include "thread.h"
#include "kernel_debug.h"
//---------------------------------------------------------------------------
#if defined __FILE_ID__
    #undef __FILE_ID__
#endif
#define __FILE_ID__     SCHEDULER_CPP

//---------------------------------------------------------------------------
Thread *g_pstNext;
Thread *g_pstCurrent;

//---------------------------------------------------------------------------
K_UCHAR Scheduler::m_bEnabled;
ThreadList Scheduler::m_clStopList;
ThreadList Scheduler::m_aclPriorities[
NUM_PRIORITIES];
K_UCHAR Scheduler::m_ucPriFlag;

K_UCHAR g_ucFlag;
//---------------------------------------------------------------------------
static const K_UCHAR aucCLZ[16] ={-1,0,1,1,2,2,2,2,3,3,3,3,3,3,3,3};

//---------------------------------------------------------------------------
void Scheduler::Init()
{
    m_ucPriFlag = 0;
    for (int i = 0; i < NUM_PRIORITIES; i++)
    {
        m_aclPriorities[i].SetPriority(i);
        m_aclPriorities[i].SetFlagPointer(&
m_ucPriFlag);
    }
    g_ucFlag = m_ucPriFlag;
}

//---------------------------------------------------------------------------
void Scheduler::Schedule()
{
    K_UCHAR ucPri = 0;

    // Figure out what priority level has ready tasks (8 priorities max)
    ucPri = aucCLZ[m_ucPriFlag >> 4 ];
    if (ucPri == 0xFF) { ucPri = aucCLZ[m_ucPriFlag & 0x0F]; }
    else { ucPri += 4; }

    // Get the thread node at this priority.
    g_pstNext = (Thread*)( m_aclPriorities[ucPri].GetHead(
) );
    g_ucFlag = m_ucPriFlag;

    KERNEL_TRACE_1( STR_SCHEDULE_1, (K_USHORT)g_pstNext->GetID() );
}

//---------------------------------------------------------------------------
void Scheduler::Add(Thread *pclThread_)
{
    m_aclPriorities[pclThread_->GetPriority()].Add
(pclThread_);
    g_ucFlag = m_ucPriFlag;
}

//---------------------------------------------------------------------------
void Scheduler::Remove(Thread *pclThread_)
{
    m_aclPriorities[pclThread_->GetPriority()].Remove
(pclThread_);
    g_ucFlag = m_ucPriFlag;
}
```

14.105 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/scheduler.h File Reference

Thread scheduler function declarations.

```
#include "kerneltypes.h"
#include "thread.h"
```

Classes

- class Scheduler

  Priority-based round-robin Thread scheduling, using ThreadLists for housekeeping.

Macros

- #define **NUM_PRIORITIES** (8)

Variables

- Thread ∗ **g_pstNext**
- Thread ∗ **g_pstCurrent**

14.105.1 Detailed Description

Thread scheduler function declarations. This scheduler implements a very flexible type of scheduling, which has become the defacto industry standard when it comes to real-time operating systems. This scheduling mechanism is referred to as priority round- robin.

From the name, there are two concepts involved here:

1) Priority scheduling:

Threads are each assigned a priority, and the thread with the highest priority which is ready to run gets to execute.

2) Round-robin scheduling:

Where there are multiple ready threads at the highest-priority level, each thread in that group gets to share time, ensuring that progress is made.

The scheduler uses an array of ThreadList objects to provide the necessary housekeeping required to keep track of threads at the various priorities. As s result, the scheduler contains one ThreadList per priority, with an additional list to manage the storage of threads which are in the "stopped" state (either have been stopped, or have not been started yet).

Definition in file scheduler.h.

14.106 scheduler.h

```
/*==========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]-------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
==========================================================================*/
#ifndef __SCHEDULER_H__
#define __SCHEDULER_H__

#include "kerneltypes.h"
#include "thread.h"

extern Thread *g_pstNext;
extern Thread *g_pstCurrent;

#define NUM_PRIORITIES              (8)
//--------------------------------------------------------------------------
class Scheduler
{
public:
    static void Init();

    static void Schedule();

    static void Add(Thread *pclThread_);

    static void Remove(Thread *pclThread_);

    static void SetScheduler(K_UCHAR bEnable_){ m_bEnabled
 = bEnable_; }

    static Thread *GetCurrentThread(){ return
g_pstCurrent; }

    static Thread *GetNextThread(){ return g_pstNext; }

    static ThreadList *GetThreadList(K_UCHAR ucPriority_
){ return &m_aclPriorities[ucPriority_]; }

    static ThreadList *GetStopList(){ return &m_clStopList
; }

    static K_UCHAR IsEnabled(){ return m_bEnabled; }

private:
    static K_UCHAR m_bEnabled;

    static ThreadList m_clStopList;

    static ThreadList m_aclPriorities[NUM_PRIORITIES];

    static K_UCHAR m_ucPriFlag;
};
#endif

```

14.107 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/screen.cpp File Reference

Higher level window management framework.

```
#include "kerneltypes.h"
#include "screen.h"
#include "gui.h"
#include "memutil.h"
```

14.107.1 Detailed Description

Higher level window management framework.

Definition in file screen.cpp.

14.108 screen.cpp

```
/*===========================================================================
       _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  _____
|    \  /  | ||    \      ||     |      ||  |/ /     ||___   |
|     \/   | ||     \     ||     \      ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||_____|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]-------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "kerneltypes.h"
#include "screen.h"
#include "gui.h"
#include "memutil.h"

//---------------------------------------------------------------------------
void Screen::SetManager( ScreenManager * pclScreenManager_ )
{
    m_pclScreenManager = pclScreenManager_;
}

//---------------------------------------------------------------------------
void Screen::SetWindowAffinity( const K_CHAR * szWindowName_ )
{
    m_pclWindow = m_pclScreenManager->FindWindowByName( szWindowName_ );
}

//---------------------------------------------------------------------------
GuiWindow *ScreenManager::FindWindowByName( const K_CHAR *m_szName_ )
{
    return m_pclSurface->FindWindowByName( m_szName_ );
}

//---------------------------------------------------------------------------
Screen *ScreenManager::FindScreenByName( const K_CHAR *szName_ )
{
    LinkListNode *pclTempNode = static_cast<LinkListNode *>(m_clScreenList.GetHead());

    while (pclTempNode)
    {
        if (MemUtil::CompareStrings(szName_, static_cast<Screen*>(pclTempNode)->GetName()))
        {
            return static_cast<Screen*>(pclTempNode);
        }
        pclTempNode = pclTempNode->GetNext();
    }

    return NULL;
}

```

14.109 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/screen.h File Reference

Higher level window management framework.

```
#include "kerneltypes.h"
#include "gui.h"
#include "ll.h"
```

Classes

- class Screen
- class ScreenList
- class ScreenManager

14.109.1 Detailed Description

Higher level window management framework.

Definition in file screen.h.

14.110 screen.h

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \     ||     |     ||  |/ /     ||___   |
|     \/   | ||     \    ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#ifndef __SCREEN_H__
#define __SCREEN_H__

#include "kerneltypes.h"
#include "gui.h"
#include "ll.h"

//---------------------------------------------------------------------------
class ScreenList;
class ScreenManager;

//---------------------------------------------------------------------------
class Screen : public LinkListNode
{
public:
    void Activate()             { Create(); }

    void Deactivate()           { Destroy(); }

    void SetWindowAffinity( const K_CHAR *szWindowName_ );

    void SetName( const K_CHAR *szName_ )           { m_szName = szName_
; }

    const K_CHAR *GetName()                         { return m_szName; }

protected:
    friend class ScreenManager;

    void SetManager( ScreenManager *pclScreenManager_ );

    const K_CHAR    *m_szName;
    ScreenManager   *m_pclScreenManager;
    GuiWindow       *m_pclWindow;

private:

    virtual void Create() = 0;
    virtual void Destroy() = 0;

};

//---------------------------------------------------------------------------
class ScreenList
{
public:
    ScreenList()                        { m_clList.Init(); }

    void Add( Screen *pclScreen_ )      { m_clList.Add(pclScreen_); }

    void Remove( Screen *pclScreen_)    { m_clList.Remove(pclScreen_); }

    Screen *GetHead()                   { return static_cast<Screen*>(m_clList.GetHead()); }

private:
    DoubleLinkList  m_clList;
};

//---------------------------------------------------------------------------
class ScreenManager
{
public:

    ScreenManager() { m_pclSurface = NULL; }

    void AddScreen( Screen *pclScreen_ )        { m_clScreenList.Add(pclScreen_);
                                                  pclScreen_->SetManager(this); }

    void RemoveScreen( Screen *pclScreen_)      { m_clScreenList.Remove(pclScreen_);
                                                  pclScreen_->SetManager(NULL); }

    void SetEventSurface( GuiEventSurface *pclSurface_ ) { m_pclSurface = pclSurface_; }

    GuiWindow *FindWindowByName( const K_CHAR *m_szName_ );

    Screen *FindScreenByName( const K_CHAR *m_szName_ );

private:

    ScreenList m_clScreenList;
    GuiEventSurface *m_pclSurface;
};

#endif
```

14.111 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/semaphore.cpp File Reference

Semaphore Blocking-Object Implemenation.

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "semaphore.h"
#include "blocking.h"
#include "kernel_debug.h"
#include "timerlist.h"
```

Macros

- #define **__FILE_ID__** SEMAPHORE_CPP

Functions

- void **TimedSemaphore_Callback** (Thread ∗pclOwner_, void ∗pvData_)

14.111.1 Detailed Description

Semaphore Blocking-Object Implemenation.

Definition in file semaphore.cpp.

14.112 semaphore.cpp

```
/*===========================================================================
     _____     _____     _____     _____
 ___|    _|__ __|_    |__  __|__   |__  __| __  |__  _____
|    \  /  | ||    \     ||     |     ||  |/ /     ||___   |
|     \/   | ||     \    ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||_______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "kerneltypes.h"
#include "mark3cfg.h"

#include "semaphore.h"
#include "blocking.h"
#include "kernel_debug.h"
//---------------------------------------------------------------------------
#if defined __FILE_ID__
    #undef __FILE_ID__
#endif
#define __FILE_ID__     SEMAPHORE_CPP

#if KERNEL_USE_SEMAPHORE

#if KERNEL_USE_TIMERS
#include "timerlist.h"

//---------------------------------------------------------------------------
void TimedSemaphore_Callback(Thread *pclOwner_, void *pvData_)
{
    Semaphore *pclSemaphore = static_cast<Semaphore*>(pvData_
);

    // Indicate that the semaphore has expired on the thread
    pclSemaphore->SetExpired(true);

    // Wake up the thread that was blocked on this semaphore.
    pclSemaphore->WakeMe(pclOwner_);

    if (pclOwner_->GetPriority() > Scheduler::GetCurrentThread
()->GetPriority())
    {
        Thread::Yield();
    }
}

//---------------------------------------------------------------------------
void Semaphore::WakeMe(Thread *pclChosenOne_)
{
    // Remove from the semaphore waitlist and back to its ready list.
    UnBlock(pclChosenOne_);
}

#endif // KERNEL_USE_TIMERS

//---------------------------------------------------------------------------
```

```
K_UCHAR Semaphore::WakeNext()
{
    Thread *pclChosenOne;

    pclChosenOne = m_clBlockList.HighestWaiter();

    // Remove from the semaphore waitlist and back to its ready list.
    UnBlock(pclChosenOne);

    // Call a task switch only if higher priority thread
    if (pclChosenOne->GetPriority() > Scheduler::GetCurrentThread
()->GetPriority())
    {
        return 1;
    }
    return 0;
}

//---------------------------------------------------------------------------
void Semaphore::Init(K_USHORT usInitVal_, K_USHORT usMaxVal_)
{
    // Copy the paramters into the object - set the maximum value for this
    // semaphore to implement either binary or counting semaphores, and set
    // the initial count.  Clear the wait list for this object.
    m_usValue = usInitVal_;
    m_usMaxValue = usMaxVal_;
#if KERNEL_USE_TIMERS
    m_bExpired = false;
#endif
}

//---------------------------------------------------------------------------
void Semaphore::Post()
{
    KERNEL_TRACE_1( STR_SEMAPHORE_POST_1, (K_USHORT)g_pstCurrent->GetID()
);

    K_UCHAR bThreadWake = 0;

    // Increment the semaphore count - we can mess with threads so ensure this
    // is in a critical section.  We don't just disable the scheudler since
    // we want to be able to do this from within an interrupt context as well.
    CS_ENTER();

    // If nothing is waiting for the semaphore
    if (m_clBlockList.GetHead() == NULL)
    {
        // Check so see if we've reached the maximum value in the semaphore
        if (m_usValue < m_usMaxValue)
        {
            // Increment the count value
            m_usValue++;
        }
    }
    else
    {
        // Otherwise, there are threads waiting for the semaphore to be
        // posted, so wake the next one (highest priority goes first).
        bThreadWake = WakeNext();
    }

    CS_EXIT();

    // if bThreadWake was set, it means that a higher-priority thread was
    // woken.  Trigger a context switch to ensure that this thread gets
    // to execute next.
    if (bThreadWake)
    {
        Thread::Yield();
    }
}

#if !KERNEL_USE_TIMERS
//---------------------------------------------------------------------------
    // No timers, no timed pend
    void Semaphore::Pend()
#else
//---------------------------------------------------------------------------
    // Redirect the untimed pend API to the timed pend, with a null timeout.
    void Semaphore::Pend()
    {
        Pend(0);
    }
//---------------------------------------------------------------------------
    bool Semaphore::Pend( K_ULONG ulWaitTimeMS_ )
#endif
{
    KERNEL_TRACE_1( STR_SEMAPHORE_PEND_1, (K_USHORT)g_pstCurrent->GetID()
);

    // Decrement the semaphore count - if 0, wait.
    K_UCHAR bThreadWait = 0;
    Thread *pclThread;

#if KERNEL_USE_TIMERS
    Timer clSemTimer;

    m_bExpired = false;
#endif

    // Once again, messing with thread data - ensure
    // we're doing all of these operations from within a thread-safe context.
    CS_ENTER();

    // Get the current thread pointer.
    pclThread = Scheduler::GetCurrentThread();

    // Check to see if we need to take any action based on the semaphore count
    if (m_usValue != 0)
    {
        // The semaphore count is non-zero, we can just decrement the count
        // and go along our merry way.
        m_usValue--;
    }
    else
    {
        // The semaphore count is zero - we need to block the current thread
        // and wait until the semaphore is posted from elsewhere.
#if KERNEL_USE_TIMERS
            if (ulWaitTimeMS_)
            {
                clSemTimer.Start(0, ulWaitTimeMS_, TimedSemaphore_Callback
, (void*)this);
            }
#endif
            Block(pclThread);
            bThreadWait = 1;
    }

    // If bThreadWait was set, it means that the current thread is blocked.
    // We need to call a context switch to ensure the highest-priority
    // ready thread gets to run next.
    if (bThreadWait)
    {
        // Switch Threads immediately
        Thread::Yield();
    }

    CS_EXIT();


#if KERNEL_USE_TIMERS
    if (ulWaitTimeMS_)
    {
        clSemTimer.Stop();
    }
    return (m_bExpired == 0);
#endif
}

//---------------------------------------------------------------------------
K_USHORT Semaphore::GetCount()
{
    K_USHORT usRet;
    CS_ENTER();
    usRet = m_usValue;
    CS_EXIT();
    return usRet;
}

#endif
```

14.113 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/semaphore.h File Reference

Semaphore Blocking Object class declarations.

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "blocking.h"
#include "threadlist.h"
```

Classes

- class Semaphore

    Counting semaphore, based on BlockingObject base class.

14.113.1 Detailed Description

Semaphore Blocking Object class declarations.

Definition in file semaphore.h.

14.114 semaphore.h

```
/*===========================================================================
     _____     _____     _____     _____
 ___|    _|__ __|_    |__ __|__   |__ __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#ifndef __SEMAPHORE_H__
#define __SEMAPHORE_H__

#include "kerneltypes.h"
#include "mark3cfg.h"

#include "blocking.h"
#include "threadlist.h"

#if KERNEL_USE_SEMAPHORE

//---------------------------------------------------------------------------
class Semaphore : public BlockingObject
{
public:
    void Init(K_USHORT usInitVal_, K_USHORT usMaxVal_);

    void Post();

    void Pend();


    K_USHORT GetCount();

#if KERNEL_USE_TIMERS

    bool Pend( K_ULONG ulWaitTimeMS_);

    void WakeMe(Thread *pclChosenOne_);

    void SetExpired(bool bExpired_) { m_bExpired = bExpired_; }

    bool GetExpired() { return m_bExpired; }
#endif

private:

    K_UCHAR WakeNext();

    K_USHORT m_usValue;
    K_USHORT m_usMaxValue;

#if KERNEL_USE_TIMERS
    bool m_bExpired;
#endif

};

#endif //KERNEL_USE_SEMAPHORE

#endif
```

14.115 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slip.cpp File Reference

Serial Line IP framing code.

```
#include "kerneltypes.h"
#include "slip.h"
#include "driver.h"
```

Macros

- #define FRAMING_BYTE (192)

  Byte indicating end-of-frame.
- #define FRAMING_ENC_BYTE (219)

  Byte used to indicate substitution.
- #define FRAMING_SUB_BYTE (220)

  Byte to substitute for framing byte.
- #define FRAMING_SUB_ENC_BYTE (221)

  Byte to substitute for the substitute-byte.
- #define ACchar (69)

  Acknowledgement character.
- #define NACchar (96)

  Non-acknowledgement character.

14.115.1 Detailed Description

Serial Line IP framing code.

Definition in file slip.cpp.

14.116 slip.cpp

```
/*============================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "kerneltypes.h"
#include "slip.h"
#include "driver.h"
```

```

//---------------------------------------------------------------------------
#define FRAMING_BYTE            (192)
#define FRAMING_ENC_BYTE        (219)
#define FRAMING_SUB_BYTE        (220)
#define FRAMING_SUB_ENC_BYTE    (221)

//---------------------------------------------------------------------------
#define ACchar                (69)
#define NACchar               (96)

//---------------------------------------------------------------------------
K_USHORT Slip::EncodeByte( K_UCHAR ucChar_, K_UCHAR *aucBuf_ )
{
    K_USHORT usLen = 1;
    switch (ucChar_)
    {
        case FRAMING_BYTE:
            aucBuf_[0] = FRAMING_ENC_BYTE;
            aucBuf_[1] = FRAMING_SUB_BYTE;
            usLen = 2;
            break;
        case FRAMING_ENC_BYTE:
            aucBuf_[0] = FRAMING_ENC_BYTE;
            aucBuf_[1] = FRAMING_SUB_ENC_BYTE;
            usLen = 2;
            break;
        default:
            aucBuf_[0] = ucChar_;
    }
    return usLen;
}

//---------------------------------------------------------------------------
K_USHORT Slip::DecodeByte( K_UCHAR *ucChar_, K_UCHAR *aucBuf_ )
{
    K_USHORT usLen = 1;

    if (aucBuf_[0] == FRAMING_ENC_BYTE)
    {
        if(aucBuf_[1] == FRAMING_SUB_BYTE)
        {
            *ucChar_ = FRAMING_BYTE;
            usLen = 2;
        }
        else if(aucBuf_[1] == FRAMING_SUB_ENC_BYTE)
        {
            *ucChar_ = FRAMING_ENC_BYTE;
            usLen = 2;
        }
        else
        {
            *ucChar_ = 0;
            usLen = 0;
        }
    }
    else if (aucBuf_[0] == FRAMING_BYTE)
    {
        usLen = 0;
        *ucChar_ = 0;
    }
    else
    {
        *ucChar_ = aucBuf_[0];
    }
    return usLen;
}

//---------------------------------------------------------------------------
void Slip::WriteByte( K_UCHAR ucData_)
{
    K_USHORT usSize = 0;
    K_USHORT usIdx = 0;
    K_UCHAR aucBuf[2];
    usSize = EncodeByte(ucData_, aucBuf);
    while (usIdx < usSize)
    {
        usIdx += m_pclDriver->Write(usSize, &aucBuf[usIdx]);
    }
}

//---------------------------------------------------------------------------
K_USHORT Slip::ReadData(K_UCHAR *pucChannel_, K_CHAR *aucBuf_,
K_USHORT usLen_)
{
    K_USHORT usReadCount;
    K_UCHAR ucTempCount;
    K_USHORT usValid = 0;
    K_USHORT usCRC;
    K_USHORT usCRC_Calc = 0;
    K_USHORT usLen;
    K_UCHAR *pucSrc = (K_UCHAR*)aucBuf_;
    K_UCHAR *pucDst = (K_UCHAR*)aucBuf_;

    usReadCount = m_pclDriver->Read(usLen_, (K_UCHAR*)aucBuf_);

    while (usReadCount)
    {
        K_UCHAR ucRead;
        ucTempCount = DecodeByte(&ucRead, pucSrc);

        *pucDst = ucRead;

        // Encountered a FRAMING_BYTE - end of message
        if (!ucTempCount)
        {
            break;
        }

        // Add to the CRC
        usCRC_Calc += ucRead;

        // Adjust iterators, source, and destination pointers.
        usReadCount -= ucTempCount;
        pucSrc += ucTempCount;
        pucDst++;
        usValid++;
    }

    // Ensure we have enough data to try a match.
    if (usValid < 5) {
        return 0;
    }

    usCRC_Calc -= aucBuf_[usValid-2];
    usCRC_Calc -= aucBuf_[usValid-1];

    usLen = ((K_USHORT)aucBuf_[1]) << 8;
    usLen += ((K_USHORT)aucBuf_[2]);
    usCRC = ((K_USHORT)aucBuf_[usValid-2]) << 8;
    usCRC += ((K_USHORT)aucBuf_[usValid-1]);

    if (usCRC != usCRC_Calc)
    {
        return 0;
    }

    *pucChannel_ = aucBuf_[0];

    return usLen;
}

//---------------------------------------------------------------------------
void Slip::WriteData(K_UCHAR ucChannel_, K_CHAR *aucBuf_,
K_USHORT usLen_)
{
    K_UCHAR aucTmp[2];
    K_USHORT usCRC = 0;

    // Lightweight protocol built on-top of SLIP.
    // 1) Channel ID (8-bit)
    // 2) Data Size (16-bit)
    // 3) Data blob
    // 4) CRC16 (16-bit)
    aucTmp[0] = FRAMING_BYTE;
    while( !m_pclDriver->Write(1, aucTmp) ) {}

    if (!usLen_)     // Read to end-of-line (\0)
    {
        K_UCHAR *pucBuf = (K_UCHAR*)aucBuf_;
        while (*pucBuf != '\0')
        {
            usLen_++;
            pucBuf++;
        }
    }

    WriteByte(ucChannel_);
    usCRC = ucChannel_;

    WriteByte((K_UCHAR)(usLen_ >> 8));
    usCRC += (usLen_ >> 8);

    WriteByte((K_UCHAR)(usLen_ & 0x00FF));
    usCRC += (usLen_ & 0x00FF);

    while (usLen_--)
    {
        WriteByte(*aucBuf_);
        usCRC += (K_USHORT)*aucBuf_;
        aucBuf_++;
    }

    WriteByte((K_UCHAR)(usCRC >> 8));
    WriteByte((K_UCHAR)(usCRC & 0x00FF));

    aucTmp[0] = FRAMING_BYTE;
    while( !m_pclDriver->Write(1, aucTmp) ) {}
}

//---------------------------------------------------------------------------
void Slip::SendAck()
{
    WriteByte(ACchar);
}

//---------------------------------------------------------------------------
void Slip::SendNack()
{
    WriteByte(NACchar);
}

//---------------------------------------------------------------------------
void Slip::WriteVector(K_UCHAR ucChannel_, SlipDataVector
 *astData_, K_USHORT usLen_)
{
    K_UCHAR aucTmp[2];
    K_USHORT usCRC = 0;
    K_UCHAR i, j;
    K_USHORT usTotalLen = 0;

    // Calculate the total length of all message fragments
    for (i = 0; i < usLen_; i++)
    {
        usTotalLen += astData_[i].ucSize;
    }

    // Send a FRAMING_BYTE to start framing a message
    aucTmp[0] = FRAMING_BYTE;
    while( !m_pclDriver->Write(1, aucTmp) ) {}

    // Write a the channel
    WriteByte(ucChannel_);
    usCRC = ucChannel_;

    // Write the length
    WriteByte((K_UCHAR)(usTotalLen >> 8));
    usCRC += (usTotalLen >> 8);

    WriteByte((K_UCHAR)(usTotalLen & 0x00FF));
    usCRC += (usTotalLen & 0x00FF);

    // Write the message fragments
    for (i = 0; i < usLen_; i++)
    {
        K_UCHAR *aucBuf = astData_[i].pucData;
        for (j = 0; j < astData_[i].ucSize; j++ )
        {
            WriteByte(*aucBuf);
            usCRC += (K_USHORT)*aucBuf;
            aucBuf++;
        }
    }

    // Write the CRC
    WriteByte((K_UCHAR)(usCRC >> 8));
    WriteByte((K_UCHAR)(usCRC & 0x00FF));

    // Write the end-of-message
    aucTmp[0] = FRAMING_BYTE;
    while( !m_pclDriver->Write(1, aucTmp) ) {}
}
```

14.117 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slip.h File Reference

Serial Line IP framing code.

```
#include "kerneltypes.h"
#include "driver.h"
```

Classes

- struct SlipDataVector

  Data structure used for vector-based SLIP data transmission.
- class Slip

  Object used to frame communications over an abstract device using the serial-line internet protocol (SLIP).

Enumerations

- enum SlipChannel {
  SLIP_CHANNEL_TERMINAL = 0, SLIP_CHANNEL_UNISCOPE, SLIP_CHANNEL_NVM, SLIP_CHANNEL-_RESET,
  SLIP_CHANNEL_GRAPHICS, SLIP_CHANNEL_HID, **SLIP_CHANNEL_COUNT** }

14.117.1 Detailed Description

Serial Line IP framing code. Also includes code to frame data in FunkenSlip format for use with SlipTerm on a host PC.

FunkenSlip uses SLIP-framed messages with a pre-defined packet format as follows:

[Channel][Size][Data Buffer][CRC8]

Channel is 1 byte, indicating the type of data carried in the message Size is 2 bytes, indicating the length of the binary blob that follows Data Buffer is n bytes, this is the packet data CRC16 is 2 byte, Providing an error detection mechanism

Definition in file slip.h.

14.117.2 Enumeration Type Documentation

14.117.2.1 enum SlipChannel

Enumerator:

SLIP_CHANNEL_TERMINAL	ASCII text mode terminal.
SLIP_CHANNEL_UNISCOPE	Uniscope VM command channel.
SLIP_CHANNEL_NVM	Non-volatile memory configuration.
SLIP_CHANNEL_RESET	Channel used to reset the device...
SLIP_CHANNEL_GRAPHICS	Encoded drawing commands.
SLIP_CHANNEL_HID	HID commands.

Definition at line 37 of file slip.h.

14.118 slip.h

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "kerneltypes.h"
#include "driver.h"

#ifndef __SLIP_H__
#define __SLIP_H__

//---------------------------------------------------------------------------
typedef enum
{
    SLIP_CHANNEL_TERMINAL = 0,
    SLIP_CHANNEL_UNISCOPE,
    SLIP_CHANNEL_NVM,
    SLIP_CHANNEL_RESET,
    SLIP_CHANNEL_GRAPHICS,
    SLIP_CHANNEL_HID,
//---
    SLIP_CHANNEL_COUNT
} SlipChannel;

//---------------------------------------------------------------------------
typedef struct
{
    K_UCHAR ucSize;
    K_UCHAR *pucData;
}SlipDataVector;

//---------------------------------------------------------------------------
class Slip
{
public:
    void SetDriver( Driver *pclDriver_ ){ m_pclDriver =
pclDriver_; }

    Driver *GetDriver() { return m_pclDriver; }

    static K_USHORT EncodeByte( K_UCHAR ucChar_, K_UCHAR *aucBuf_ );

    static K_USHORT DecodeByte( K_UCHAR *ucChar_, K_UCHAR *aucBuf_ );

    void WriteData( K_UCHAR ucChannel_, K_CHAR *aucBuf_, K_USHORT
usLen_ );

    K_USHORT ReadData( K_UCHAR *pucChannel_, K_CHAR *aucBuf_, K_USHORT
usLen_ );

    void WriteVector( K_UCHAR ucChannel_, SlipDataVector
 *astData_, K_USHORT usLen_ );

    void SendAck();

    void SendNack();

private:
    void WriteByte(K_UCHAR ucData_);
    Driver *m_pclDriver;
};

#endif
```

14.119 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slip_mux.cpp File Reference

FunkenSlip Channel Multiplexer.

```
#include "kerneltypes.h"
#include "driver.h"
#include "drvUART.h"
#include "slip.h"
#include "slip_mux.h"
#include "message.h"
```

Functions

- static void SlipMux_CallBack (Driver ∗pclDriver_)

14.119.1 Detailed Description

FunkenSlip Channel Multiplexer. Demultiplexes FunkenSlip packets transmitted over a single serial channel, and provides an abstraction to attach handlers for each event type.

Definition in file slip_mux.cpp.

14.119.2 Function Documentation

14.119.2.1 static void SlipMux_CallBack (Driver ∗ *pclDriver_*) [static]

Parameters

*pclDriver_*	Pointer to the driver data for the port triggering the callback

Definition at line 43 of file slip_mux.cpp.

14.120 slip_mux.cpp

```
00001 /*===========================================================================
00002      _____        _____        _____        _____
00003  ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
00004 |    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
00005 |     \/   | ||     \     ||     \     ||     \     ||___   |
00006 |__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
00007     |_____|      |_____|      |_____|      |_____|
00008
00009 --[Mark3 Realtime Platform]--------------------------------------------------
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 ===========================================================================*/
00022 #include "kerneltypes.h"
00023 #include "driver.h"
00024 #include "drvUART.h"
00025 #include "slip.h"
00026 #include "slip_mux.h"
00027 #include "message.h"
00028
00029 //---------------------------------------------------------------------------
00030 MessageQueue *SlipMux::m_pclMessageQueue;
00031 K_UCHAR SlipMux::m_aucData[SLIP_BUFFER_SIZE];
00032 Driver *SlipMux::m_pclDriver;
00033 Slip_Channel SlipMux::m_apfChannelHandlers[SLIP_CHANNEL_COUNT] = {0};
00034 Semaphore SlipMux::m_clSlipSem;
00035 Slip SlipMux::m_clSlip;
00036
00037 //---------------------------------------------------------------------------
00043 static void SlipMux_CallBack( Driver *pclDriver_)
00044 {
00045     Message *pclMsg = GlobalMessagePool::Pop();
00046     if (pclMsg)
00047     {
00048         pclDriver_->Control(CMD_SET_RX_DISABLE, 0, 0, 0, 0);
00049
```

```
        // Send a message to the queue, letting it know that there's a
        // pending slip message that needs to be processed
        pclMsg->SetCode(SLIP_RX_MESSAGE_ID);
        pclMsg->SetData(NULL);
        SlipMux::GetQueue()->Send(pclMsg);
    }
}

//---------------------------------------------------------------------------
void SlipMux::Init(K_CHAR *pcDriverPath_, K_USHORT usRxSize_, K_UCHAR *aucRx_, K_USHORT usTxSize_, K_UCHAR *aucTx_)
{
    m_pclDriver = DriverList::FindByPath(pcDriverPath_);
    m_pclMessageQueue = NULL;

    m_clSlip.SetDriver(m_pclDriver);
    m_clSlipSem.Init(0, 1);

    m_pclDriver->Control(CMD_SET_BUFFERS, (void*)aucRx_, usRxSize_, (void*)aucTx_, usTxSize_);
    m_pclDriver->Control(CMD_SET_RX_CALLBACK, (void*)SlipMux_CallBack, 0, 0, 0);
    {
        K_UCHAR ucEscape = 192;
        m_pclDriver->Control(CMD_SET_RX_ESCAPE, (void*)&ucEscape, 1, 0, NULL);
    }
}

//---------------------------------------------------------------------------
void SlipMux::InstallHandler( K_UCHAR ucChannel_, Slip_Channel pfHandler_ )
{
    if (pfHandler_)
    {
        m_apfChannelHandlers[ucChannel_] = pfHandler_;
    }
}

//---------------------------------------------------------------------------
void SlipMux::MessageReceive(void)
{
    K_USHORT usLen;
    K_UCHAR ucChannel;

    usLen = m_clSlip.ReadData( &ucChannel, (K_CHAR*)m_aucData, SLIP_BUFFER_SIZE );
    if (usLen && (m_apfChannelHandlers[ucChannel] != NULL))
    {
        m_apfChannelHandlers[ucChannel]( m_pclDriver, ucChannel, &(m_aucData[3]), usLen);
    }

    // Re-enable the driver once we're done.
    m_pclDriver->Control( CMD_SET_RX_ENABLE, 0, 0, 0, 0 );
}

```

14.121 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slip_mux.h File Reference

FunkenSlip Channel Multiplexer.

```
#include "kerneltypes.h"
#include "driver.h"
#include "semaphore.h"
#include "message.h"
#include "slip.h"
```

Classes

- class SlipMux

  Static-class which implements a multiplexed stream of SLIP data over a single interface.

Macros

- #define **SLIP_BUFFER_SIZE** (32)
- #define **SLIP_RX_MESSAGE_ID** (0xD00D)

Typedefs

- typedef void(∗ **Slip_Channel**)(Driver ∗pclDriver_, K_UCHAR ucChannel_, K_UCHAR ∗pucData_, K_USHORT usLen_)

14.121.1 Detailed Description

FunkenSlip Channel Multiplexer. Demultiplexes FunkenSlip packets transmitted over a single serial channel

Definition in file slip_mux.h.

14.122 slip_mux.h

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "kerneltypes.h"
#include "driver.h"
#include "semaphore.h"
#include "message.h"
#include "slip.h"

#ifndef __SLIP_MUX_H__
#define __SLIP_MUX_H__

//---------------------------------------------------------------------------
#define SLIP_BUFFER_SIZE    (32)

#define SLIP_RX_MESSAGE_ID  (0xD00D)

//---------------------------------------------------------------------------
typedef void (*Slip_Channel)( Driver *pclDriver_, K_UCHAR ucChannel_,
K_UCHAR *pucData_, K_USHORT usLen_ );

//---------------------------------------------------------------------------
class SlipMux
{
public:
    static void Init(K_CHAR *pcDriverPath_, K_USHORT usRxSize_, K_UCHAR *
aucRx_, K_USHORT usTxSize_, K_UCHAR *aucTx_);

    static void InstallHandler( K_UCHAR ucChannel_, Slip_Channel
pfHandler_ );

    static void MessageReceive();

    static Driver *GetDriver(){ return m_pclDriver; }

    static MessageQueue *GetQueue(){ return
m_pclMessageQueue; }

    static void SetQueue( MessageQueue *pclMessageQueue_ )
        { m_pclMessageQueue = pclMessageQueue_; }


    static Slip *GetSlip(){ return &m_clSlip; }

private:
    static MessageQueue *m_pclMessageQueue;
    static Driver *m_pclDriver;
    static Slip_Channel m_apfChannelHandlers[SLIP_CHANNEL_COUNT];
    static K_UCHAR m_aucData[SLIP_BUFFER_SIZE];
    static Semaphore m_clSlipSem;
    static Slip m_clSlip;
};

#endif
```

14.123 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slipterm.cpp File Reference

Serial debug interface using SLIP protocol, and FunkenSlip multiplexing.

```
#include "kerneltypes.h"
#include "slip.h"
#include "slipterm.h"
```

14.123.1 Detailed Description

Serial debug interface using SLIP protocol, and FunkenSlip multiplexing.

Definition in file slipterm.cpp.

14.124 slipterm.cpp

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "kerneltypes.h"
#include "slip.h"
#include "slipterm.h"

//---------------------------------------------------------------------------
void SlipTerm::Init()
{
    m_clSlip.SetDriver( DriverList::FindByPath("/dev/tty" ) );
    m_ucVerbosity = SEVERITY_DEBUG;
}

//---------------------------------------------------------------------------
K_USHORT SlipTerm::StrLen( const char *szLine_ )
{
    K_USHORT i=0;
    while (szLine_[i] != 0 )
    {
        i++;
    }
    return i;
}

//---------------------------------------------------------------------------
void SlipTerm::PrintLn( const char *szLine_ )
{
    SlipDataVector astData[2];
    astData[0].pucData = (K_UCHAR*)szLine_;
    astData[0].ucSize = StrLen(szLine_);
    astData[1].pucData = (K_UCHAR*)"\r\n";
    astData[1].ucSize = 2;

    m_clSlip.WriteVector(SLIP_CHANNEL_TERMINAL
, astData, 2);
}

//---------------------------------------------------------------------------
void SlipTerm::PrintLn( K_UCHAR ucSeverity_, const char *
szLine_ )
{
    if (ucSeverity_ <= m_ucVerbosity)
    {
        PrintLn( szLine_ );
    }
}
```

14.125 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slipterm.h File Reference

Serial debug interface using SLIP serial, and Funkenslip serial port multiplexing.

```
#include "kerneltypes.h"
#include "driver.h"
#include "slip.h"
```

Classes

- class SlipTerm

  Class implementing a simple debug terminal interface.

Macros

- #define **SEVERITY_DEBUG** 4
- #define **SEVERITY_INFO** 3
- #define **SEVERITY_WARN** 2
- #define **SEVERITY_CRITICAL** 1
- #define **SEVERITY_CATASTROPHIC** 0
- #define **__SLIPTERM_H__**

14.125.1 Detailed Description

Serial debug interface using SLIP serial, and Funkenslip serial port multiplexing.

Definition in file slipterm.h.

14.126 slipterm.h

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "kerneltypes.h"
#include "driver.h"
#include "slip.h"
```

```

//---------------------------------------------------------------------------
#define SEVERITY_DEBUG              4
#define SEVERITY_INFO               3
#define SEVERITY_WARN               2
#define SEVERITY_CRITICAL           1
#define SEVERITY_CATASTROPHIC       0

//---------------------------------------------------------------------------
#ifndef __SLIPTERM_H__
#define __SLIPTERM_H__

class SlipTerm
{
public:
    void Init();

    void PrintLn( const char *szLine_ );

    void PrintLn( K_UCHAR ucSeverity_, const char *szLine_ );

    void SetVerbosity( K_UCHAR ucLevel_ ) { m_ucVerbosity
 = ucLevel_; }
private:
    K_USHORT StrLen( const char *szString_ );

    K_UCHAR m_ucVerbosity;


    Slip m_clSlip;
};

#endif
```

14.127 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/system_heap.cpp File Reference

Global system-heap implementation.

```
#include "kerneltypes.h"
#include "system_heap_config.h"
#include "system_heap.h"
```

14.127.1 Detailed Description

Global system-heap implementation. Provides a system-wide malloc/free paradigm allocation scheme.

Definition in file system_heap.cpp.

14.128 system_heap.cpp

```
/*===========================================================================
     _____     _____     _____     _____
 ___|    _|__ __|_    |__ __|__   |__ __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|       |_____|       |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "kerneltypes.h"
#include "system_heap_config.h"
#include "system_heap.h"

#if USE_SYSTEM_HEAP

//---------------------------------------------------------------------------
```

```
K_UCHAR    SystemHeap::m_pucRawHeap[ HEAP_RAW_SIZE ];
HeapConfig SystemHeap::m_pclSystemHeapConfig[ HEAP_NUM_SIZES
 + 1];
FixedHeap  SystemHeap::m_clSystemHeap;
bool       SystemHeap::m_bInit;

//---------------------------------------------------------------------------
void SystemHeap::Init(void)
{
#if HEAP_NUM_SIZES > 0
    m_pclSystemHeapConfig[0].m_usBlockSize  = HEAP_BLOCK_SIZE_1
;
    m_pclSystemHeapConfig[0].m_usBlockCount = HEAP_BLOCK_COUNT_1
;
#endif
#if HEAP_NUM_SIZES > 1
    m_pclSystemHeapConfig[1].m_usBlockSize  = HEAP_BLOCK_SIZE_2;
    m_pclSystemHeapConfig[1].m_usBlockCount = HEAP_BLOCK_COUNT_2;
#endif
#if HEAP_NUM_SIZES > 2
    m_pclSystemHeapConfig[2].m_usBlockSize  = HEAP_BLOCK_SIZE_3;
    m_pclSystemHeapConfig[2].m_usBlockCount = HEAP_BLOCK_COUNT_3;
#endif
#if HEAP_NUM_SIZES > 3
    m_pclSystemHeapConfig[3].m_usBlockSize  = HEAP_BLOCK_SIZE_4;
    m_pclSystemHeapConfig[3].m_usBlockCount = HEAP_BLOCK_COUNT_4;
#endif
#if HEAP_NUM_SIZES > 4
    m_pclSystemHeapConfig[4].m_usBlockSize  = HEAP_BLOCK_SIZE_5;
    m_pclSystemHeapConfig[4].m_usBlockCount = HEAP_BLOCK_COUNT_5;
#endif
#if HEAP_NUM_SIZES > 5
    m_pclSystemHeapConfig[5].m_usBlockSize  = HEAP_BLOCK_SIZE_6;
    m_pclSystemHeapConfig[5].m_usBlockCount = HEAP_BLOCK_COUNT_6;
#endif
#if HEAP_NUM_SIZES > 6
    m_pclSystemHeapConfig[6].m_usBlockSize  = HEAP_BLOCK_SIZE_7;
    m_pclSystemHeapConfig[6].m_usBlockCount = HEAP_BLOCK_COUNT_7;
#endif
#if HEAP_NUM_SIZES > 7
    m_pclSystemHeapConfig[7].m_usBlockSize  = HEAP_BLOCK_SIZE_8;
    m_pclSystemHeapConfig[7].m_usBlockCount = HEAP_BLOCK_COUNT_8;
#endif
#if HEAP_NUM_SIZES > 8
    m_pclSystemHeapConfig[8].m_usBlockSize  = HEAP_BLOCK_SIZE_9;
    m_pclSystemHeapConfig[8].m_usBlockCount = HEAP_BLOCK_COUNT_9;
#endif
#if HEAP_NUM_SIZES > 9
    m_pclSystemHeapConfig[9].m_usBlockSize  = HEAP_BLOCK_SIZE_10;
    m_pclSystemHeapConfig[9].m_usBlockCount = HEAP_BLOCK_COUNT_10
;
#endif
#if HEAP_NUM_SIZES > 10
    m_pclSystemHeapConfig[10].m_usBlockSize  = HEAP_BLOCK_SIZE_11;
    m_pclSystemHeapConfig[10].m_usBlockCount =
HEAP_BLOCK_COUNT_11;
#endif
#if HEAP_NUM_SIZES > 11
    m_pclSystemHeapConfig[11].m_usBlockSize  = HEAP_BLOCK_SIZE_12;
    m_pclSystemHeapConfig[11].m_usBlockCount =
HEAP_BLOCK_COUNT_12;
#endif
#if HEAP_NUM_SIZES > 12
    m_pclSystemHeapConfig[12].m_usBlockSize  = HEAP_BLOCK_SIZE_13;
    m_pclSystemHeapConfig[12].m_usBlockCount =
HEAP_BLOCK_COUNT_13;
#endif
#if HEAP_NUM_SIZES > 13
    m_pclSystemHeapConfig[13].m_usBlockSize  = HEAP_BLOCK_SIZE_14;
    m_pclSystemHeapConfig[13].m_usBlockCount =
HEAP_BLOCK_COUNT_14;
#endif
#if HEAP_NUM_SIZES > 14
    m_pclSystemHeapConfig[14].m_usBlockSize  = HEAP_BLOCK_SIZE_15;
    m_pclSystemHeapConfig[14].m_usBlockCount =
HEAP_BLOCK_COUNT_15;
#endif
#if HEAP_NUM_SIZES > 15
    m_pclSystemHeapConfig[15].m_usBlockSize  = HEAP_BLOCK_SIZE_16;
    m_pclSystemHeapConfig[15].m_usBlockCount =
HEAP_BLOCK_COUNT_16;
#endif
#if HEAP_NUM_SIZES > 16
    m_pclSystemHeapConfig[16].m_usBlockSize  = HEAP_BLOCK_SIZE_17;
    m_pclSystemHeapConfig[16].m_usBlockCount =
HEAP_BLOCK_COUNT_17;
#endif
#if HEAP_NUM_SIZES > 17
    m_pclSystemHeapConfig[17].m_usBlockSize  = HEAP_BLOCK_SIZE_18;
    m_pclSystemHeapConfig[17].m_usBlockCount =
HEAP_BLOCK_COUNT_18;
#endif
#if HEAP_NUM_SIZES > 18
    m_pclSystemHeapConfig[18].m_usBlockSize  = HEAP_BLOCK_SIZE_19;
    m_pclSystemHeapConfig[18].m_usBlockCount =
HEAP_BLOCK_COUNT_19;
#endif
#if HEAP_NUM_SIZES > 19
    m_pclSystemHeapConfig[19].m_usBlockSize  = HEAP_BLOCK_SIZE_20;
    m_pclSystemHeapConfig[19].m_usBlockCount =
HEAP_BLOCK_COUNT_20;
#endif
#if HEAP_NUM_SIZES > 20
    m_pclSystemHeapConfig[20].m_usBlockSize  = HEAP_BLOCK_SIZE_21;
    m_pclSystemHeapConfig[20].m_usBlockCount =
HEAP_BLOCK_COUNT_21;
#endif

    m_pclSystemHeapConfig[HEAP_NUM_SIZES].m_usBlockSize
 = 0;
    m_pclSystemHeapConfig[HEAP_NUM_SIZES].m_usBlockCount
 = 0;

    m_clSystemHeap.Create((void*)m_pucRawHeap, m_pclSystemHeapConfig);

    m_bInit = true;
}

//---------------------------------------------------------------------------
void *SystemHeap::Alloc(K_USHORT usSize_)
{
    if (!m_bInit)
    {
        return NULL;
    }
    return m_clSystemHeap.Alloc(usSize_);
}

//---------------------------------------------------------------------------
void SystemHeap::Free(void* pvBlock_)
{
    if (!m_bInit)
    {
        return;
    }
    m_clSystemHeap.Free(pvBlock_);
}

#endif // USE_SYSTEM_HEAP
```

14.129 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/system_heap.h File Reference

Global system-heap implmentation.

```
#include "system_heap_config.h"
#include "fixed_heap.h"
```

Classes

- class SystemHeap

Macros

- #define HEAP_RAW_SIZE_1 ((HEAP_BLOCK_SIZE_1 + sizeof(LinkListNode) + sizeof(void∗)) ∗ HEAP_BL-OCK_COUNT_1)

  Really ugly computations used to auto-size the heap footprint based on the user-configuration data.

- #define **HEAP_RAW_SIZE_2** ((HEAP_BLOCK_SIZE_2 + sizeof(LinkListNode) + sizeof(void∗)) ∗ HEAP_BL-OCK_COUNT_2)
- #define **HEAP_RAW_SIZE_3** ((HEAP_BLOCK_SIZE_3 + sizeof(LinkListNode) + sizeof(void∗)) ∗ HEAP_BL-OCK_COUNT_3)
- #define **HEAP_RAW_SIZE_4** 0
- #define **HEAP_RAW_SIZE_5** 0
- #define **HEAP_RAW_SIZE_6** 0
- #define **HEAP_RAW_SIZE_7** 0
- #define **HEAP_RAW_SIZE_8** 0
- #define **HEAP_RAW_SIZE_9** 0
- #define **HEAP_RAW_SIZE_10** 0
- #define **HEAP_RAW_SIZE_11** 0
- #define **HEAP_RAW_SIZE_12** 0
- #define **HEAP_RAW_SIZE_13** 0
- #define **HEAP_RAW_SIZE_14** 0
- #define **HEAP_RAW_SIZE_15** 0
- #define **HEAP_RAW_SIZE_16** 0
- #define **HEAP_RAW_SIZE_17** 0
- #define **HEAP_RAW_SIZE_18** 0
- #define **HEAP_RAW_SIZE_19** 0
- #define **HEAP_RAW_SIZE_20** 0
- #define **HEAP_RAW_SIZE_21** 0
- #define **HEAP_RAW_SIZE**

14.129.1 Detailed Description

Global system-heap implmentation. Provides a basic malloc()/free() allocation scheme.

Definition in file system_heap.h.

14.129.2 Macro Definition Documentation

14.129.2.1 #define HEAP_RAW_SIZE

Value:

```
HEAP_RAW_SIZE_1 + \
HEAP_RAW_SIZE_2 + \
HEAP_RAW_SIZE_3 + \
HEAP_RAW_SIZE_4 + \
HEAP_RAW_SIZE_5 + \
HEAP_RAW_SIZE_6 + \
HEAP_RAW_SIZE_7 + \
HEAP_RAW_SIZE_8 + \
HEAP_RAW_SIZE_9 + \
HEAP_RAW_SIZE_10 + \
HEAP_RAW_SIZE_11 + \
HEAP_RAW_SIZE_12 + \
HEAP_RAW_SIZE_13 + \
HEAP_RAW_SIZE_14 + \
HEAP_RAW_SIZE_15 + \
HEAP_RAW_SIZE_16 + \
HEAP_RAW_SIZE_17 + \
HEAP_RAW_SIZE_18 + \
HEAP_RAW_SIZE_19 + \
HEAP_RAW_SIZE_20 + \
HEAP_RAW_SIZE_21
```

Definition at line 161 of file system_heap.h.

14.129.2.2 #define HEAP_RAW_SIZE_1 ((HEAP_BLOCK_SIZE_1 + sizeof(LinkListNode) + sizeof(void∗)) ∗ HEAP_BLOCK_COUNT_1)

Really ugly computations used to auto-size the heap footprint based on the user-configuration data.

(don’t touch this!!!)

Definition at line 35 of file system_heap.h.

14.130 system_heap.h

```
00001 /*=============================================================================
00002      _____     _____     _____     _____
00003  ___|    _|__ __|_    |__ __|__   |__ __| __  |__ ______
00004 |    \  /  | ||    \     ||     |     ||  |/ /     ||___   |
00005 |     \/   | ||     \    ||     \     ||  |  \     ||___   |
00006 |__/\__/|__|_||__|\__\ __||__|\__\ __||__|\__\ __||______|
00007     |_____|      |_____|     |_____|     |_____|
00008
00009 --[Mark3 Realtime Platform]--------------------------------------------------
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 ===========================================================================*/
00021 #ifndef __SYSTEM_HEAP_H__
00022 #define __SYSTEM_HEAP_H__
00023
00024 #include "system_heap_config.h"
00025 #include "fixed_heap.h"
00026
00027 #if USE_SYSTEM_HEAP
00028
00029 //---------------------------------------------------------------------------
00034 #if HEAP_NUM_SIZES > 0
00035     #define HEAP_RAW_SIZE_1 ((HEAP_BLOCK_SIZE_1 + sizeof(LinkListNode) +
       sizeof(void*)) * HEAP_BLOCK_COUNT_1 )
00036 #else
00037     #define HEAP_RAW_SIZE_1 0
00038 #endif
00039
00040 #if HEAP_NUM_SIZES > 1
00041     #define HEAP_RAW_SIZE_2 ((HEAP_BLOCK_SIZE_2 + sizeof(LinkListNode) +
       sizeof(void*)) * HEAP_BLOCK_COUNT_2 )
00042 #else
00043     #define HEAP_RAW_SIZE_2 0
00044 #endif
00045
00046 #if HEAP_NUM_SIZES > 2
00047     #define HEAP_RAW_SIZE_3 ((HEAP_BLOCK_SIZE_3 + sizeof(LinkListNode) +
       sizeof(void*)) * HEAP_BLOCK_COUNT_3 )
00048 #else
00049     #define HEAP_RAW_SIZE_3 0
00050 #endif
00051
00052 #if HEAP_NUM_SIZES > 3
00053     #define HEAP_RAW_SIZE_4 ((HEAP_BLOCK_SIZE_4 + sizeof(LinkListNode) +
       sizeof(void*)) * HEAP_BLOCK_COUNT_4 )
00054 #else
00055     #define HEAP_RAW_SIZE_4 0
00056 #endif
00057
00058 #if HEAP_NUM_SIZES > 4
00059     #define HEAP_RAW_SIZE_5 ((HEAP_BLOCK_SIZE_5 + sizeof(LinkListNode) +
       sizeof(void*)) * HEAP_BLOCK_COUNT_5 )
00060 #else
00061     #define HEAP_RAW_SIZE_5 0
00062 #endif
00063
00064 #if HEAP_NUM_SIZES > 5
00065     #define HEAP_RAW_SIZE_6 ((HEAP_BLOCK_SIZE_6 + sizeof(LinkListNode) +
       sizeof(void*)) * HEAP_BLOCK_COUNT_6 )
00066 #else
00067     #define HEAP_RAW_SIZE_6 0
00068 #endif
00069
00070 #if HEAP_NUM_SIZES > 6
00071     #define HEAP_RAW_SIZE_7 ((HEAP_BLOCK_SIZE_7 + sizeof(LinkListNode) +
       sizeof(void*)) * HEAP_BLOCK_COUNT_7 )
00072 #else
00073     #define HEAP_RAW_SIZE_7 0
00074 #endif
```

```

#if HEAP_NUM_SIZES > 7
    #define HEAP_RAW_SIZE_8 ((HEAP_BLOCK_SIZE_8 + sizeof(LinkListNode) +
 sizeof(void*)) * HEAP_BLOCK_COUNT_8 )
#else
    #define HEAP_RAW_SIZE_8 0
#endif

#if HEAP_NUM_SIZES > 8
    #define HEAP_RAW_SIZE_9 ((HEAP_BLOCK_SIZE_9 + sizeof(LinkListNode) +
 sizeof(void*)) * HEAP_BLOCK_COUNT_9 )
#else
    #define HEAP_RAW_SIZE_9 0
#endif

#if HEAP_NUM_SIZES > 9
    #define HEAP_RAW_SIZE_10 ((HEAP_BLOCK_SIZE_10 + sizeof(LinkListNode) +
 sizeof(void*)) * HEAP_BLOCK_COUNT_10 )
#else
    #define HEAP_RAW_SIZE_10 0
#endif

#if HEAP_NUM_SIZES > 10
    #define HEAP_RAW_SIZE_11 ((HEAP_BLOCK_SIZE_11 + sizeof(LinkListNode) +
 sizeof(void*)) * HEAP_BLOCK_COUNT_11 )
#else
    #define HEAP_RAW_SIZE_11 0
#endif

#if HEAP_NUM_SIZES > 11
    #define HEAP_RAW_SIZE_12 ((HEAP_BLOCK_SIZE_12 + sizeof(LinkListNode) +
 sizeof(void*)) * HEAP_BLOCK_COUNT_12 )
#else
    #define HEAP_RAW_SIZE_12 0
#endif

#if HEAP_NUM_SIZES > 12
    #define HEAP_RAW_SIZE_13 ((HEAP_BLOCK_SIZE_13 + sizeof(LinkListNode) +
 sizeof(void*)) * HEAP_BLOCK_COUNT_13 )
#else
    #define HEAP_RAW_SIZE_13 0
#endif

#if HEAP_NUM_SIZES > 13
    #define HEAP_RAW_SIZE_14 ((HEAP_BLOCK_SIZE_14 + sizeof(LinkListNode) +
 sizeof(void*)) * HEAP_BLOCK_COUNT_14 )
#else
    #define HEAP_RAW_SIZE_14 0
#endif

#if HEAP_NUM_SIZES > 14
    #define HEAP_RAW_SIZE_15 ((HEAP_BLOCK_SIZE_15 + sizeof(LinkListNode) +
 sizeof(void*)) * HEAP_BLOCK_COUNT_15 )
#else
    #define HEAP_RAW_SIZE_15 0
#endif

#if HEAP_NUM_SIZES > 15
    #define HEAP_RAW_SIZE_16 ((HEAP_BLOCK_SIZE_16 + sizeof(LinkListNode) +
 sizeof(void*)) * HEAP_BLOCK_COUNT_16 )
#else
    #define HEAP_RAW_SIZE_16 0
#endif

#if HEAP_NUM_SIZES > 16
    #define HEAP_RAW_SIZE_17 ((HEAP_BLOCK_SIZE_17 + sizeof(LinkListNode) +
 sizeof(void*)) * HEAP_BLOCK_COUNT_17 )
#else
    #define HEAP_RAW_SIZE_17 0
#endif

#if HEAP_NUM_SIZES > 17
    #define HEAP_RAW_SIZE_18 ((HEAP_BLOCK_SIZE_18 + sizeof(LinkListNode) +
 sizeof(void*)) * HEAP_BLOCK_COUNT_18 )
#else
    #define HEAP_RAW_SIZE_18 0
#endif

#if HEAP_NUM_SIZES > 18
    #define HEAP_RAW_SIZE_19 ((HEAP_BLOCK_SIZE_19 + sizeof(LinkListNode) +
 sizeof(void*)) * HEAP_BLOCK_COUNT_19 )
#else
    #define HEAP_RAW_SIZE_19 0
#endif

#if HEAP_NUM_SIZES > 19
    #define HEAP_RAW_SIZE_20 ((HEAP_BLOCK_SIZE_20 + sizeof(LinkListNode) +
      sizeof(void*)) * HEAP_BLOCK_COUNT_20 )
#else
    #define HEAP_RAW_SIZE_20 0
#endif

#if HEAP_NUM_SIZES > 20
    #define HEAP_RAW_SIZE_21 ((HEAP_BLOCK_SIZE_21 + sizeof(LinkListNode) +
      sizeof(void*)) * HEAP_BLOCK_COUNT_21 )
#else
    #define HEAP_RAW_SIZE_21 0
#endif

//---------------------------------------------------------------------------
#define HEAP_RAW_SIZE   \
HEAP_RAW_SIZE_1 + \
HEAP_RAW_SIZE_2 + \
HEAP_RAW_SIZE_3 + \
HEAP_RAW_SIZE_4 + \
HEAP_RAW_SIZE_5 + \
HEAP_RAW_SIZE_6 + \
HEAP_RAW_SIZE_7 + \
HEAP_RAW_SIZE_8 + \
HEAP_RAW_SIZE_9 + \
HEAP_RAW_SIZE_10 + \
HEAP_RAW_SIZE_11 + \
HEAP_RAW_SIZE_12 + \
HEAP_RAW_SIZE_13 + \
HEAP_RAW_SIZE_14 + \
HEAP_RAW_SIZE_15 + \
HEAP_RAW_SIZE_16 + \
HEAP_RAW_SIZE_17 + \
HEAP_RAW_SIZE_18 + \
HEAP_RAW_SIZE_19 + \
HEAP_RAW_SIZE_20 + \
HEAP_RAW_SIZE_21

//---------------------------------------------------------------------------
class SystemHeap
{
public:
    static void  Init(void);
    static void* Alloc(K_USHORT usSize_);
    static void  Free(void *pvData_);

private:
    static K_UCHAR m_pucRawHeap[ HEAP_RAW_SIZE ];
    static HeapConfig m_pclSystemHeapConfig[ HEAP_NUM_SIZES
      + 1 ];
    static FixedHeap m_clSystemHeap;
    static bool m_bInit;
};

#endif // USE_SYSTEM_HEAP

#endif // __SYSTEM_HEAP_H__
```

14.131 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/system_heap_-config.h File Reference

System heap configuration - defines the block sizes and counts used to fulfill system/service allocations.

```
#include "kerneltypes.h"
```

Macros

- #define USE_SYSTEM_HEAP (1)

  Set this to "1" if you want the system heap to be built as part of this library.
- #define HEAP_NUM_SIZES (3)

  Define the number of heap block sizes that we want to have attached to our system heap.
- #define HEAP_BLOCK_SIZE_1 ((K_USHORT) 8)

  Define the block sizes for each of the fixed-size blocks that will be managed by our heaps.
- #define **HEAP_BLOCK_SIZE_2** ((K_USHORT) 16)

- #define **HEAP_BLOCK_SIZE_3** ((K_USHORT) 24)
- #define **HEAP_BLOCK_SIZE_4** ((K_USHORT) 32)
- #define **HEAP_BLOCK_SIZE_5** ((K_USHORT) 48)
- #define **HEAP_BLOCK_SIZE_6** ((K_USHORT) 64)
- #define **HEAP_BLOCK_SIZE_7** ((K_USHORT) 96)
- #define **HEAP_BLOCK_SIZE_8** ((K_USHORT) 128)
- #define **HEAP_BLOCK_SIZE_9** ((K_USHORT) 192)
- #define **HEAP_BLOCK_SIZE_10** ((K_USHORT) 256)
- #define HEAP_BLOCK_COUNT_1 ((K_USHORT) 4)

  Define the number of blocks in each bin, tailored for a particular application.
- #define **HEAP_BLOCK_COUNT_2** ((K_USHORT) 4)
- #define **HEAP_BLOCK_COUNT_3** ((K_USHORT) 2)
- #define **HEAP_BLOCK_COUNT_4** ((K_USHORT) 2)
- #define **HEAP_BLOCK_COUNT_5** ((K_USHORT) 2)
- #define **HEAP_BLOCK_COUNT_6** ((K_USHORT) 2)
- #define **HEAP_BLOCK_COUNT_7** ((K_USHORT) 1)
- #define **HEAP_BLOCK_COUNT_8** ((K_USHORT) 1)
- #define **HEAP_BLOCK_COUNT_9** ((K_USHORT) 1)
- #define **HEAP_BLOCK_COUNT_10** ((K_USHORT) 1)

14.131.1 Detailed Description

System heap configuration - defines the block sizes and counts used to fulfill system/service allocations.

Definition in file system_heap_config.h.

14.131.2 Macro Definition Documentation

14.131.2.1 #define HEAP_BLOCK_SIZE_1 ((K_USHORT) 8)

Define the block sizes for each of the fixed-size blocks that will be managed by our heaps.

Must be defined in incrementing order.

Definition at line 44 of file system_heap_config.h.

14.132 system_heap_config.h

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#ifndef __SYSTEM_HEAP_CONFIG_H__
#define __SYSTEM_HEAP_CONFIG_H__

#include "kerneltypes.h"

//---------------------------------------------------------------------------
#define USE_SYSTEM_HEAP     (1)

//---------------------------------------------------------------------------
#define HEAP_NUM_SIZES      (3)

//---------------------------------------------------------------------------
```

```
#define HEAP_BLOCK_SIZE_1        ((K_USHORT) 8)
#define HEAP_BLOCK_SIZE_2        ((K_USHORT) 16)
#define HEAP_BLOCK_SIZE_3        ((K_USHORT) 24)
#define HEAP_BLOCK_SIZE_4        ((K_USHORT) 32)
#define HEAP_BLOCK_SIZE_5        ((K_USHORT) 48)
#define HEAP_BLOCK_SIZE_6        ((K_USHORT) 64)
#define HEAP_BLOCK_SIZE_7        ((K_USHORT) 96)
#define HEAP_BLOCK_SIZE_8        ((K_USHORT) 128)
#define HEAP_BLOCK_SIZE_9        ((K_USHORT) 192)
#define HEAP_BLOCK_SIZE_10       ((K_USHORT) 256)

//-----------------------------------------------------------------------------
#define HEAP_BLOCK_COUNT_1       ((K_USHORT) 4)
#define HEAP_BLOCK_COUNT_2       ((K_USHORT) 4)
#define HEAP_BLOCK_COUNT_3       ((K_USHORT) 2)
#define HEAP_BLOCK_COUNT_4       ((K_USHORT) 2)
#define HEAP_BLOCK_COUNT_5       ((K_USHORT) 2)
#define HEAP_BLOCK_COUNT_6       ((K_USHORT) 2)
#define HEAP_BLOCK_COUNT_7       ((K_USHORT) 1)
#define HEAP_BLOCK_COUNT_8       ((K_USHORT) 1)
#define HEAP_BLOCK_COUNT_9       ((K_USHORT) 1)
#define HEAP_BLOCK_COUNT_10      ((K_USHORT) 1)

#endif

```

14.133 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/thread.cpp File Reference

Platform-Independent thread class Definition.

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "thread.h"
#include "scheduler.h"
#include "kernelswi.h"
#include "timerlist.h"
#include "semaphore.h"
#include "quantum.h"
#include "kernel_debug.h"
```

Macros

- #define **__FILE_ID__** THREAD_CPP

Functions

- static void ThreadSleepCallback (Thread ∗pclOwner_, void ∗pvData_)
  This callback is used to wake up a thread once the interval has expired.

14.133.1 Detailed Description

Platform-Independent thread class Definition.

Definition in file thread.cpp.

14.134 thread.cpp

```
/*=============================================================================
     _____        _____        _____       _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
```

```
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \    ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "kerneltypes.h"
#include "mark3cfg.h"

#include "thread.h"
#include "scheduler.h"
#include "kernelswi.h"
#include "timerlist.h"
#include "semaphore.h"
#include "quantum.h"
#include "kernel_debug.h"
//---------------------------------------------------------------------------
#if defined __FILE_ID__
    #undef __FILE_ID__
#endif
#define __FILE_ID__     THREAD_CPP

//---------------------------------------------------------------------------
void Thread::Init(  K_UCHAR *paucStack_,
                K_USHORT usStackSize_,
                K_UCHAR ucPriority_,
                ThreadEntry_t pfEntryPoint_,
                void *pvArg_ )
{
    static K_UCHAR ucThreadID = 0;

    KERNEL_ASSERT( paucStack_ );
    KERNEL_ASSERT( pfEntryPoint_ );

    m_ucThreadID = ucThreadID++;

    KERNEL_TRACE_1( STR_STACK_SIZE_1, usStackSize_ );
    KERNEL_TRACE_1( STR_PRIORITY_1, (K_UCHAR)ucPriority_ );
    KERNEL_TRACE_1( STR_THREAD_ID_1, (K_USHORT)m_ucThreadID );
    KERNEL_TRACE_1( STR_ENTRYPOINT_1, (K_USHORT)pfEntryPoint_ );

    // Initialize the thread parameters to their initial values.
    m_paucStack = paucStack_;
    m_paucStackTop = TOP_OF_STACK(paucStack_, usStackSize_);

    m_usStackSize = usStackSize_;

#if KERNEL_USE_QUANTUM
    m_usQuantum = 4;
#endif

    m_ucPriority = ucPriority_;
    m_ucCurPriority = m_ucPriority;
    m_pfEntryPoint = pfEntryPoint_;
    m_pvArg = pvArg_;

#if KERNEL_USE_THREADNAME
    m_szName = NULL;
#endif

    // Call CPU-specific stack initialization
    ThreadPort::InitStack(this);

    // Add to the global "stop" list.
    CS_ENTER();
    m_pclOwner = Scheduler::GetThreadList(m_ucPriority);
    m_pclCurrent = Scheduler::GetStopList();
    m_pclCurrent->Add(this);
    CS_EXIT();
}

//---------------------------------------------------------------------------
void Thread::Start(void)
{
    // Remove the thread from the scheduler's "stopped" list, and add it
    // to the scheduler's ready list at the proper priority.
    KERNEL_TRACE_1( STR_THREAD_START_1, (K_USHORT)m_ucThreadID );

    CS_ENTER();
    Scheduler::GetStopList()->Remove(this);
    Scheduler::Add(this);
    m_pclOwner = Scheduler::GetThreadList(
m_ucPriority);
    m_pclCurrent = m_pclOwner;
    if (m_ucPriority >= Scheduler::GetCurrentThread
()->GetCurPriority())
    {
#if KERNEL_USE_QUANTUM
        // Deal with the thread Quantum
        Quantum::RemoveThread();
        Quantum::AddThread(this);
#endif
    }
    if (m_ucPriority > Scheduler::GetCurrentThread
()->GetPriority())
    {
        Thread::Yield();
    }
    CS_EXIT();
}

#if KERNEL_USE_DYNAMIC_THREADS
//---------------------------------------------------------------------------
void Thread::Exit()
{
    K_UCHAR bReschedule = 0;

    KERNEL_TRACE_1( STR_THREAD_EXIT_1, m_ucThreadID );

    CS_ENTER();

    // If this thread is the actively-running thread, make sure we run the
    // scheduler again.
    if (this == Scheduler::GetCurrentThread())
    {
        bReschedule = 1;
    }

    // Remove the thread from scheduling
    Scheduler::Remove(this);

    CS_EXIT();

    if (bReschedule)
    {
        // Choose a new "next" thread if we must
        Thread::Yield();
    }
}
#endif

#if KERNEL_USE_SLEEP
//---------------------------------------------------------------------------
static void ThreadSleepCallback( Thread *pclOwner_,
void *pvData_ )
{
    Semaphore *pclSemaphore = static_cast<Semaphore*>(pvData_
);

    // Post the semaphore, which will wake the sleeping thread.
    pclSemaphore->Post();
}

//---------------------------------------------------------------------------
void Thread::Sleep(K_ULONG ulTimeMs_)
{
    Timer clTimer;
    Semaphore clSemaphore;

    // Create a semaphore that this thread will block on
    clSemaphore.Init(0, 1);

    // Create a one-shot timer that will call a callback that posts the
    // semaphore, waking our thread.
    clTimer.SetIntervalMSeconds(ulTimeMs_);
    clTimer.SetCallback(ThreadSleepCallback);
    clTimer.SetData((void*)&clSemaphore);
    clTimer.SetFlags(TIMERLIST_FLAG_ONE_SHOT);

    // Add the new timer to the timer scheduler, and block the thread
    TimerScheduler::Add(&clTimer);
    clSemaphore.Pend();
}
#endif // KERNEL_USE_SLEEP

//---------------------------------------------------------------------------
K_USHORT Thread::GetStackSlack()
{
    K_USHORT usCount = 0;

    CS_ENTER();

    for (usCount = 0; usCount < m_usStackSize; usCount++)
    {
        if (m_paucStack[usCount] != 0xFF)
        {
            break;
        }
    }

    CS_EXIT();

    return usCount;
}

//---------------------------------------------------------------------------
void Thread::Yield()
{
    CS_ENTER();

    // Run the scheduler
    Scheduler::Schedule();

    // Only switch contexts if the new task is different than the old task
    if (Scheduler::GetCurrentThread() !=
Scheduler::GetNextThread())
    {
#if KERNEL_USE_QUANTUM
        // new thread scheduled.  Stop current quantum timer (if it exists),
        // and restart it for the new thread (if required).
        Quantum::RemoveThread();
        Quantum::AddThread(g_pstNext);
#endif

        Thread::ContextSwitchSWI();
    }

    CS_EXIT();
}

//---------------------------------------------------------------------------
void Thread::SetPriorityBase(K_UCHAR ucPriority_)
{
     GetCurrent()->Remove(this);

     SetCurrent(Scheduler::GetThreadList(
m_ucPriority));

     GetCurrent()->Add(this);
}

//---------------------------------------------------------------------------
void Thread::SetPriority(K_UCHAR ucPriority_)
{
    K_UCHAR bSchedule = 0;
    CS_ENTER();
    // If this is the currently running thread, it's a good idea to reschedule
    // Or, if the new priority is a higher priority than the current thread's.
    if ((g_pstCurrent == this) || (ucPriority_ > g_pstCurrent->GetPriority
()))
    {
        bSchedule = 1;
    }
    CS_EXIT();

    Scheduler::Remove(this);

    m_ucCurPriority = ucPriority_;
    m_ucPriority = ucPriority_;

    CS_ENTER();
    Scheduler::Add(this);
    CS_EXIT();

    if (bSchedule)
    {
        CS_ENTER();
        Scheduler::Schedule();
#if KERNEL_USE_QUANTUM
        // new thread scheduled.  Stop current quantum timer (if it exists),
        // and restart it for the new thread (if required).
        Quantum::RemoveThread();
        Quantum::AddThread(g_pstNext);
#endif
        CS_EXIT();
        Thread::ContextSwitchSWI();
    }
}

//---------------------------------------------------------------------------
void Thread::InheritPriority(K_UCHAR ucPriority_)
{
    SetOwner(Scheduler::GetThreadList(
ucPriority_));
    m_ucCurPriority = ucPriority_;
}

//---------------------------------------------------------------------------
void Thread::ContextSwitchSWI()
{
    // Call the context switch interrupt if the scheduler is enabled.
    if (Scheduler::IsEnabled() == 1)
    {
        KERNEL_TRACE_1( STR_CONTEXT_SWITCH_1, (K_USHORT)g_pstNext->GetID()
 );
        KernelSWI::Trigger();
    }
}


```

14.135 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/thread.h File Reference

Platform independent thread class declarations.

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "ll.h"
#include "threadlist.h"
#include "scheduler.h"
#include "threadport.h"
#include "quantum.h"
```

Classes

- class Thread

  Object providing fundamental multitasking support in the kernel.

Macros

- #define THREAD_QUANTUM_DEFAULT (4)

  Suggested default thread quantum.

Typedefs

- typedef void(∗ ThreadEntry_t)(void ∗pvArg_)

  Function pointer type used for thread entrypoint functions.

14.135.1 Detailed Description

Platform independent thread class declarations. Threads are an atomic unit of execution, and each instance of the thread class represents an instance of a program running of the processor. The Thread is the fundmanetal user-facing object in the kernel - it is what makes multiprocessing possible from application code.

In Mark3, threads each have their own context - consisting of a stack, and all of the registers required to multiplex a processor between multiple threads.

The Thread class inherits directly from the LinkListNode class to facilitate efficient thread management using Double, or Double-Circular linked lists.

Definition in file thread.h.

14.136 thread.h

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#ifndef __THREAD_H__
#define __THREAD_H__

#include "kerneltypes.h"
#include "mark3cfg.h"

#include "ll.h"
#include "threadlist.h"
#include "scheduler.h"
#include "threadport.h"
#include "quantum.h"

//---------------------------------------------------------------------------
#define THREAD_QUANTUM_DEFAULT      (4)

//---------------------------------------------------------------------------
typedef void (*ThreadEntry_t)(void *pvArg_);

//---------------------------------------------------------------------------
class ThreadPort;

//---------------------------------------------------------------------------
class Thread : public LinkListNode
{
public:
    void Init(K_UCHAR *paucStack_,
              K_USHORT usStackSize_,
              K_UCHAR ucPriority_,
              ThreadEntry_t pfEntryPoint_,
              void *pvArg_ );

    void Start();

#if KERNEL_USE_THREADNAME

    void SetName(const K_CHAR *szName_) { m_szName = szName_; };
#endif

    ThreadList *GetOwner(void) { return m_pclOwner;
 };

    ThreadList *GetCurrent(void) { return m_pclCurrent
; } ;

    K_UCHAR GetPriority(void) { return m_ucPriority; };

    K_UCHAR GetCurPriority(void) { return m_ucCurPriority; };

#if KERNEL_USE_QUANTUM

    void SetQuantum( K_USHORT usQuantum_ ) { m_usQuantum =
 usQuantum_; }

    K_USHORT GetQuantum(void) { return m_usQuantum; };
#endif

    void SetCurrent( ThreadList *pclNewList_ ) {
m_pclCurrent = pclNewList_; };

```

```
    void SetOwner( ThreadList *pclNewList_ ) { m_pclOwner
 = pclNewList_; };


    void SetPriority(K_UCHAR ucPriority_);

    void InheritPriority(K_UCHAR ucPriority_);

#if KERNEL_USE_DYNAMIC_THREADS

    void Exit();
#endif

#if KERNEL_USE_SLEEP

    static void Sleep(K_ULONG ulTimeMs_);
#endif

    static void Yield(void);

    void SetID( K_UCHAR ucID_ ) { m_ucThreadID = ucID_; }

    K_UCHAR GetID() { return m_ucThreadID; }


    K_USHORT GetStackSlack();

    friend class ThreadPort;

private:
    static void ContextSwitchSWI(void);

    void SetPriorityBase(K_UCHAR ucPriority_);

    K_UCHAR *m_paucStackTop;

    K_UCHAR *m_paucStack;

    K_USHORT m_usStackSize;

#if KERNEL_USE_QUANTUM

    K_USHORT m_usQuantum;
#endif

    K_UCHAR m_ucThreadID;

    K_UCHAR m_ucPriority;

    K_UCHAR m_ucCurPriority;

    ThreadEntry_t m_pfEntryPoint;

    void *m_pvArg;

#if KERNEL_USE_THREADNAME

    const K_CHAR *m_szName;
#endif

    ThreadList *m_pclCurrent;

    ThreadList *m_pclOwner;
};

#endif
```

14.137 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/threadlist.cpp File Reference

Thread linked-list definitions.

```
#include "kerneltypes.h"
#include "ll.h"
#include "threadlist.h"
#include "thread.h"
#include "kernel_debug.h"
```

Macros

- #define **__FILE_ID__** THREADLIST_CPP

14.137.1 Detailed Description

Thread linked-list definitions.

Definition in file threadlist.cpp.

14.138 threadlist.cpp

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "kerneltypes.h"
#include "ll.h"
#include "threadlist.h"
#include "thread.h"
#include "kernel_debug.h"
//---------------------------------------------------------------------------
#if defined __FILE_ID__
    #undef __FILE_ID__
#endif
#define __FILE_ID__     THREADLIST_CPP

//---------------------------------------------------------------------------
void ThreadList::SetPriority(K_UCHAR ucPriority_)
{
    m_ucPriority = ucPriority_;
}

//---------------------------------------------------------------------------
void ThreadList::SetFlagPointer( K_UCHAR *pucFlag_)
{
    m_pucFlag = pucFlag_;
}

//---------------------------------------------------------------------------
void ThreadList::Add(LinkListNode *node_) {
    CircularLinkList::Add(node_);

    // If the head of the list isn't empty,
    if (m_pstHead != NULL)
    {
        // We've specified a bitmap for this threadlist
        if (m_pucFlag)
        {
            // Set the flag for this priority level
            *m_pucFlag |= (1 << m_ucPriority);
        }
    }
}

//---------------------------------------------------------------------------
void ThreadList::Add(LinkListNode *node_, K_UCHAR *
pucFlag_, K_UCHAR ucPriority_) {
    // Set the threadlist's priority level, flag pointer, and then add the
    // thread to the threadlist
    SetPriority(ucPriority_);
    SetFlagPointer(pucFlag_);
    Add(node_);
}

//---------------------------------------------------------------------------
void ThreadList::Remove(LinkListNode *node_) {
    // Remove the thread from the list
    CircularLinkList::Remove(node_);

    // If the list is empty...
    if (!m_pstHead)
    {
        // Clear the bit in the bitmap at this priority level
        if (m_pucFlag)
        {
            *m_pucFlag &= ~(1 << m_ucPriority);
        }
    }
}

//---------------------------------------------------------------------------
Thread *ThreadList::HighestWaiter()
{
    Thread *pclTemp = static_cast<Thread*>(GetHead());
    Thread *pclChosen = pclTemp;

    K_UCHAR ucMaxPri = 0;

    // Go through the list, return the highest-priority thread in this list.
    while(1)
    {
        // Compare against current max-priority thread
        if (pclTemp->GetPriority() >= ucMaxPri)
        {
            ucMaxPri = pclTemp->GetPriority();
            pclChosen = pclTemp;
        }

        // Break out if this is the last thread in the list
        if (pclTemp == static_cast<Thread*>(GetTail()))
        {
            break;
        }

        pclTemp = static_cast<Thread*>(pclTemp->GetNext());
    }
    return pclChosen;
}
```

14.139 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/threadlist.h File Reference

Thread linked-list declarations.

```
#include "kerneltypes.h"
#include "ll.h"
```

Classes

- class ThreadList

  This class is used for building thread-management facilities, such as schedulers, and blocking objects.

14.139.1 Detailed Description

Thread linked-list declarations.

Definition in file threadlist.h.

14.140 threadlist.h

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__   __|__   |__  __| __   |__  ______
|    \  /  | ||    \     ||     |     ||  |/ /     ||___   |
|     \/   | ||     \    ||     \     ||  |  \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#ifndef __THREADLIST_H__
#define __THREADLIST_H__

#include "kerneltypes.h"
#include "ll.h"

class Thread;

class ThreadList : public CircularLinkList
{
public:
    ThreadList() { m_ucPriority = 0; m_pucFlag = NULL; };

    void SetPriority(K_UCHAR ucPriority_);

    void SetFlagPointer(K_UCHAR *pucFlag_);

    void Add(LinkListNode *node_);

    void Add(LinkListNode *node_, K_UCHAR *pucFlag_, K_UCHAR ucPriority_);

    void Remove(LinkListNode *node_);

    Thread *HighestWaiter();
private:

    K_UCHAR m_ucPriority;

    K_UCHAR *m_pucFlag;
};

#endif
```

14.141 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/threadport.cpp File Reference

ATMega328p Multithreading.

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "thread.h"
#include "threadport.h"
#include "kernelswi.h"
#include "kerneltimer.h"
#include "timerlist.h"
#include "quantum.h"
#include <avr/io.h>
#include <avr/interrupt.h>
```

Functions

- static void **Thread_Switch** (void)
- ISR (INT0_vect) __attribute__((signal

  SWI using INT0 - used to trigger a context switch.
- ISR (TIMER1_COMPA_vect)

  Timer interrupt ISR - causes a tick, which may cause a context switch.

Variables

- Thread ∗ **g_pstCurrentThread**
- **naked**

14.141.1 Detailed Description

ATMega328p Multithreading.

Definition in file threadport.cpp.

14.142 threadport.cpp

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "thread.h"
#include "threadport.h"
#include "kernelswi.h"
#include "kerneltimer.h"
#include "timerlist.h"
#include "quantum.h"
#include <avr/io.h>
#include <avr/interrupt.h>

//---------------------------------------------------------------------------
Thread *g_pstCurrentThread;

//---------------------------------------------------------------------------
void ThreadPort::InitStack(Thread *pclThread_)
{
    // Initialize the stack for a Thread
    K_USHORT usAddr;
    K_UCHAR *pucStack;
    K_USHORT i;

    // Get the address of the thread's entry function
    usAddr = (K_USHORT)(pclThread_->m_pfEntryPoint);

    // Start by finding the bottom of the stack
    pucStack = (K_UCHAR*)pclThread_->m_paucStackTop;

    // clear the stack, and initialize it to a known-default value (easier
    // to debug when things go sour with stack corruption or overflow)
    for (i = 0; i < pclThread_->m_usStackSize; i++)
    {
        pclThread_->m_paucStack[i] = 0xFF;
    }

    // Our context starts with the entry function
    PUSH_TO_STACK(pucStack, (K_UCHAR)(usAddr & 0x00FF));
    PUSH_TO_STACK(pucStack, (K_UCHAR)((usAddr >> 8) & 0x00FF));

    // R0
    PUSH_TO_STACK(pucStack, 0x00);    // R0

    // Push status register and R1 (which is used as a constant zero)
    PUSH_TO_STACK(pucStack, 0x80);  // SR
    PUSH_TO_STACK(pucStack, 0x00);  // R1

    // Push other registers
    for (i = 2; i <= 23; i++) //R2-R23
    {
        PUSH_TO_STACK(pucStack, i);
    }

    // Assume that the argument is the only stack variable
    PUSH_TO_STACK(pucStack, (K_UCHAR)(((K_USHORT)(pclThread_->m_pvArg)) & 0x00FF));    //R24
    PUSH_TO_STACK(pucStack, (K_UCHAR)((((K_USHORT)(pclThread_->m_pvArg))>>8) & 0x00FF)); //R25

    // Push the rest of the registers in the context
    for (i = 26; i <=31; i++)
    {
        PUSH_TO_STACK(pucStack, i);
    }

    // Set the top o' the stack.
    pclThread_->m_paucStackTop = (K_UCHAR*)pucStack;

    // That's it!  the thread is ready to run now.
}

//---------------------------------------------------------------------------
static void Thread_Switch(void)
{
    g_pstCurrent = g_pstNext;
}


//---------------------------------------------------------------------------
void ThreadPort::StartThreads()
{
    KernelSWI::Config();                 // configure the task switch SWI
    KernelTimer::Config();                 // configure the kernel timer

    Scheduler::SetScheduler(1);            // enable the scheduler
    Scheduler::Schedule();                 // run the scheduler - determine the first thread to run

    Thread_Switch();                     // Set the next scheduled thread to the current thread

    KernelTimer::Start();              // enable the kernel timer
    KernelSWI::Start();                  // enable the task switch SWI

    // Restore the context...
    Thread_RestoreContext();        // restore the context of the first running thread
    ASM("reti");                    // return from interrupt - will return to the first scheduled thread
}

//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
ISR(INT0_vect) __attribute__ ( ( signal, naked ) );
ISR(INT0_vect)
{
    Thread_SaveContext();        // Push the context (registers) of the current task
    Thread_Switch();            // Switch to the next task
    Thread_RestoreContext();    // Pop the context (registers) of the next task
    ASM("reti");                // Return to the next task
}

//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
ISR(TIMER1_COMPA_vect)
{
#if KERNEL_USE_TIMERS
    TimerScheduler::Process();
#endif
#if KERNEL_USE_QUANTUM
    Quantum::UpdateTimer();
#endif
}
```

14.143 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/threadport.h File Reference

ATMega328p Multithreading support.

```
#include "kerneltypes.h"
#include "thread.h"
#include <avr/io.h>
#include <avr/interrupt.h>
```

Classes

- class ThreadPort

  Class defining the architecture specific functions required by the kernel.

Macros

- #define ASM(x) asm volatile(x);

  ASM Macro - simplify the use of ASM directive in C.
- #define SR_ 0x3F

  Status register define - map to 0x003F.
- #define SPH_ 0x3E

  Stack pointer define.
- #define **SPL_** 0x3D
- #define TOP_OF_STACK(x, y) (K_UCHAR∗) (((K_USHORT)x) + (y-1))

  Macro to find the top of a stack given its size and top address.
- #define PUSH_TO_STACK(x, y) ∗x = y; x--;

  Push a value y to the stack pointer x and decrement the stack pointer.
- #define Thread_SaveContext()

  Save the context of the Thread.
- #define Thread_RestoreContext()

  Restore the context of the Thread.
- #define CS_ENTER()

  These macros must be used in pairs !
- #define CS_EXIT()

  Exit critical section (restore status register)
- #define ENABLE_INTS() ASM("sei");

  Initiate a contex switch without using the SWI.
- #define **DISABLE_INTS**() ASM("cli");

14.143.1 Detailed Description

ATMega328p Multithreading support.

Definition in file threadport.h.

14.143.2 Macro Definition Documentation

14.143.2.1 #define CS_ENTER( )

Value:

```
{ \
volatile K_UCHAR x; \
x = _SFR_IO8(SR_); \
ASM("cli");
```

These macros *must* be used in pairs !

Enter critical section (copy status register, disable interrupts)

Definition at line 142 of file threadport.h.

14.143.2.2 #define CS_EXIT( )

Value:

```
_SFR_IO8(SR_) = x;\
}
```

Exit critical section (restore status register)

Definition at line 149 of file threadport.h.

14.144 threadport.h

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#ifndef __THREADPORT_H_
#define __THREADPORT_H_

#include "kerneltypes.h"
#include "thread.h"

#include <avr/io.h>
#include <avr/interrupt.h>

//---------------------------------------------------------------------------
#define ASM(x)      asm volatile(x);

#define SR_         0x3F

#define SPH_        0x3E
#define SPL_        0x3D


//---------------------------------------------------------------------------
#define TOP_OF_STACK(x, y)        (K_UCHAR*) ( ((K_USHORT)x) + (y-1) )

#define PUSH_TO_STACK(x, y)        *x = y; x--;

//---------------------------------------------------------------------------
#define Thread_SaveContext() \
ASM("push r0"); \
ASM("in r0, __SREG__"); \
ASM("cli"); \
ASM("push r0"); \
ASM("push r1"); \
ASM("clr r1"); \
ASM("push r2"); \
ASM("push r3"); \
ASM("push r4"); \
ASM("push r5"); \
ASM("push r6"); \
ASM("push r7"); \
ASM("push r8"); \
ASM("push r9"); \
ASM("push r10"); \
ASM("push r11"); \
ASM("push r12"); \
ASM("push r13"); \
ASM("push r14"); \
ASM("push r15"); \
ASM("push r16"); \
ASM("push r17"); \
ASM("push r18"); \
ASM("push r19"); \
ASM("push r20"); \
ASM("push r21"); \
ASM("push r22"); \
ASM("push r23"); \
ASM("push r24"); \
ASM("push r25"); \
ASM("push r26"); \
ASM("push r27"); \
ASM("push r28"); \
ASM("push r29"); \
ASM("push r30"); \
ASM("push r31"); \
ASM("lds r26, g_pstCurrent"); \
ASM("lds r27, g_pstCurrent + 1"); \
ASM("adiw r26, 4"); \
ASM("in    r0, 0x3D"); \
ASM("st    x+, r0"); \
ASM("in    r0, 0x3E"); \
ASM("st    x+, r0");

//---------------------------------------------------------------------------
#define Thread_RestoreContext() \
ASM("lds r26, g_pstCurrent"); \
ASM("lds r27, g_pstCurrent + 1");\
ASM("adiw r26, 4"); \
ASM("ld     r28, x+"); \
ASM("out 0x3D, r28"); \
ASM("ld     r29, x+"); \
ASM("out 0x3E, r29"); \
ASM("pop r31"); \
ASM("pop r30"); \
ASM("pop r29"); \
ASM("pop r28"); \
ASM("pop r27"); \
ASM("pop r26"); \
ASM("pop r25"); \
ASM("pop r24"); \
ASM("pop r23"); \
ASM("pop r22"); \
ASM("pop r21"); \
ASM("pop r20"); \
ASM("pop r19"); \
ASM("pop r18"); \
ASM("pop r17"); \
ASM("pop r16"); \
ASM("pop r15"); \
ASM("pop r14"); \
ASM("pop r13"); \
ASM("pop r12"); \
ASM("pop r11"); \
ASM("pop r10"); \
ASM("pop r9"); \
ASM("pop r8"); \
ASM("pop r7"); \
ASM("pop r6"); \
ASM("pop r5"); \
ASM("pop r4"); \
ASM("pop r3"); \
ASM("pop r2"); \
ASM("pop r1"); \
ASM("pop r0"); \
ASM("out __SREG__, r0"); \
ASM("pop r0");

//------------------------------------------------------------------------
//------------------------------------------------------------------------
#define CS_ENTER()    \
{ \
volatile K_UCHAR x; \
x = _SFR_IO8(SR_); \
ASM("cli");
//---------------------------------------------------------------------------
#define CS_EXIT() \
_SFR_IO8(SR_) = x;\
}

//---------------------------------------------------------------------------
#define ENABLE_INTS()        ASM("sei");
#define DISABLE_INTS()       ASM("cli");

//---------------------------------------------------------------------------
class Thread;
class ThreadPort
{
public:
    static void StartThreads();
    friend class Thread;
private:

    static void InitStack(Thread *pstThread_);
};

#endif //__ThreadPORT_H_
```

14.145 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/timerlist.cpp File Reference

Timer data structure + scheduler implementations.

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "timerlist.h"
#include "kerneltimer.h"
#include "threadport.h"
#include "kernel_debug.h"
```

Macros

- #define **__FILE_ID__** TIMERLIST_CPP
- #define TL_FUDGE_FACTOR (9)

  Number of ticks to account for overhead when performing Time->tick computations.

14.145.1 Detailed Description

Timer data structure + scheduler implementations.

Definition in file timerlist.cpp.

14.145.2 Macro Definition Documentation

14.145.2.1 #define TL_FUDGE_FACTOR (9)

Number of ticks to account for overhead when performing Time->tick computations.

This must be calibrated on a per-device basis. This value is currently Set up for a 16-bit timer, with a 256 prescaler, 16MHz clock, on an ATMega328p (i.e. ARDUINO UNO).

Definition at line 44 of file timerlist.cpp.

14.146 timerlist.cpp

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "kerneltypes.h"
#include "mark3cfg.h"

#include "timerlist.h"
#include "kerneltimer.h"
#include "threadport.h"
#include "kernel_debug.h"
//---------------------------------------------------------------------------
#if defined __FILE_ID__
    #undef __FILE_ID__
#endif
#define __FILE_ID__     TIMERLIST_CPP

#if KERNEL_USE_TIMERS

//---------------------------------------------------------------------------
#define TL_FUDGE_FACTOR             (9)

TimerList TimerScheduler::m_clTimerList;
//---------------------------------------------------------------------------
void TimerList::Init(void)
{
    m_bTimerActive = 0;
    m_ulNextWakeup = 0;
}

//---------------------------------------------------------------------------
void TimerList::Add(Timer *pclListNode_)
{
    K_LONG lDelta;
    K_UCHAR bStart = 0;
     CS_ENTER();

    if (GetHead() == NULL)
    {
        bStart = 1;
    }

    pclListNode_->ClearNode();
    DoubleLinkList::Add(pclListNode_);

    // Set the initial timer value
    pclListNode_->m_ulTimeLeft = pclListNode_->m_ulInterval
;

    if (!bStart)
    {
        // If the new interval is less than the amount of time remaining...
        lDelta = KernelTimer::TimeToExpiry() -
pclListNode_->m_ulInterval;

        if (lDelta > 0)
        {
            // Set the new expiry time on the timer.
            m_ulNextWakeup = KernelTimer::SubtractExpiry
((K_ULONG)lDelta);
        }
    }
    else
    {
        m_ulNextWakeup = pclListNode_->m_ulInterval;
        KernelTimer::SetExpiry(m_ulNextWakeup
);
        KernelTimer::Start();
    }
    // Set the timer as active.
    pclListNode_->m_ucFlags |= TIMERLIST_FLAG_ACTIVE
;
    CS_EXIT();
}

```

```cpp
//---------------------------------------------------------------------------
void TimerList::Remove(Timer *pclLinkListNode_)
{
    CS_ENTER();

    DoubleLinkList::Remove(pclLinkListNode_);

    if (this->GetHead() == NULL)
    {
        KernelTimer::Stop();
    }

    CS_EXIT();
}

//---------------------------------------------------------------------------
void TimerList::Process(void)
{
    K_ULONG ulNewExpiry;
    K_ULONG ulOvertime;
    K_UCHAR bContinue;

    Timer *pclNode;
    Timer *pclPrev;

    // Clear the timer and its expiry time - keep it running though
    KernelTimer::ClearExpiry();

    do
    {
        ulNewExpiry = MAX_TIMER_TICKS;
        pclNode = static_cast<Timer*>(GetHead());
        pclPrev = NULL;
        bContinue = 0;

        // Subtract the elapsed time interval from each active timer.
        while (pclNode)
        {
            // Active timers only...
            if (pclNode->m_ucFlags & TIMERLIST_FLAG_ACTIVE)
            {
                // Did the timer expire?
                if (pclNode->m_ulTimeLeft <= m_ulNextWakeup)
                {
                    // Yes - set the "callback" flag - we'll execute the callbacks later
                    pclNode->m_ucFlags |= TIMERLIST_FLAG_CALLBACK;

                    if (pclNode->m_ucFlags & TIMERLIST_FLAG_ONE_SHOT)
                    {
                        // If this was a one-shot timer, deactivate the timer.
                        pclNode->m_ucFlags |= TIMERLIST_FLAG_EXPIRED;
                        pclNode->m_ucFlags &= ~TIMERLIST_FLAG_ACTIVE;
                    }
                    else
                    {
                        // Reset the interval timer.
                        // I think we're good though...
                        pclNode->m_ulTimeLeft = pclNode->m_ulInterval;

                        // If the time remaining is less than the expiry, set the new expiry.
                        if (pclNode->m_ulTimeLeft < ulNewExpiry)
                        {
                            ulNewExpiry = pclNode->m_ulTimeLeft;
                        }
                    }
                }
                else
                {
                    // Not expiring, but determine how K_LONG to run the next timer interval for.
                    pclNode->m_ulTimeLeft -= m_ulNextWakeup;
                    if (pclNode->m_ulTimeLeft < ulNewExpiry)
                    {
                        ulNewExpiry = pclNode->m_ulTimeLeft;
                    }
                }

            }
            pclNode = static_cast<Timer*>(pclNode->GetNext());

        }

        // Process the expired timers callbacks.
        pclNode = static_cast<Timer*>(GetHead());
        while (pclNode)
        {
            pclPrev = NULL;

            // If the timer expired, run the callbacks now.
            if (pclNode->m_ucFlags & TIMERLIST_FLAG_CALLBACK)
            {
                // Run the callback. these callbacks must be very fast...
                pclNode->m_pfCallback( pclNode->m_pclOwner, pclNode->m_pvData );
                pclNode->m_ucFlags &= ~TIMERLIST_FLAG_CALLBACK;

                // If this was a one-shot timer, let's remove it.
                if (pclNode->m_ucFlags & TIMERLIST_FLAG_ONE_SHOT)
                {
                    pclPrev = pclNode;
                }
            }
            pclNode = static_cast<Timer*>(pclNode->GetNext());

            // Remove one-shot-timers
            if (pclPrev)
            {
                Remove(pclPrev);
            }
        }

        // Check to see how much time has elapsed since the time we
        // acknowledged the interrupt...
        ulOvertime = KernelTimer::GetOvertime();

        if( ulOvertime >= ulNewExpiry ) {
            m_ulNextWakeup = ulOvertime;
            bContinue = 1;
        }

    // If it's taken longer to go through this loop than would take us to
    // the next expiry, re-run the timing loop
    } while (bContinue);


    // This timer elapsed, but there's nothing more to do...
    // Turn the timer off.
    if (ulNewExpiry >= MAX_TIMER_TICKS)
    {
        KernelTimer::Stop();
    }
    else
    {
        // Update the timer with the new "Next Wakeup" value, plus whatever
        // overtime has accumulated since the last time we called this handler
        m_ulNextWakeup = KernelTimer::SetExpiry(ulNewExpiry + ulOvertime);
    }
}

//---------------------------------------------------------------------------
void Timer::Start( K_UCHAR bRepeat_, K_ULONG ulIntervalMs_, TimerCallback_t pfCallback_, void *pvData_ )
{
    SetIntervalMSeconds(ulIntervalMs_);
    m_pfCallback = pfCallback_;
    m_pvData = pvData_;
    if (!bRepeat_)
    {
        m_ucFlags = TIMERLIST_FLAG_ONE_SHOT;
    }
    else
    {
        m_ucFlags = 0;
    }
    m_pclOwner = Scheduler::GetCurrentThread();
    TimerScheduler::Add(this);
}

//---------------------------------------------------------------------------
```

```
void Timer::Stop()
{
    TimerScheduler::Remove(this);
}

//---------------------------------------------------------------------------
void Timer::SetIntervalTicks( K_ULONG ulTicks_ )
{
    m_ulInterval = ulTicks_;
}

//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
void Timer::SetIntervalSeconds( K_ULONG ulSeconds_)
{
    m_ulInterval = SECONDS_TO_TICKS(ulSeconds_) - TL_FUDGE_FACTOR
;
}

//---------------------------------------------------------------------------
void Timer::SetIntervalMSeconds( K_ULONG ulMSeconds_)
{
    m_ulInterval = MSECONDS_TO_TICKS(ulMSeconds_) - TL_FUDGE_FACTOR
;
}

//---------------------------------------------------------------------------
void Timer::SetIntervalUSeconds( K_ULONG ulUSeconds_)
{
    m_ulInterval = USECONDS_TO_TICKS(ulUSeconds_) - TL_FUDGE_FACTOR
;
}

#endif //KERNEL_USE_TIMERS
```

14.147 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/timerlist.h File Reference

Timer list and timer-scheduling declarations.

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "ll.h"
#include "thread.h"
```

Classes

- class Timer

  Timer - an event-driven execution context based on a specified time interval.
- class TimerList

  TimerList class - a doubly-linked-list of timer objects.
- class TimerScheduler

  "Static" Class used to interface a global TimerList with the rest of the kernel.

Macros

- #define TIMERLIST_FLAG_ONE_SHOT (0x01)

  Timer is one-shot.
- #define TIMERLIST_FLAG_ACTIVE (0x02)

  Timer is currently active.
- #define TIMERLIST_FLAG_CALLBACK (0x04)

  Timer is pending a callback.
- #define TIMERLIST_FLAG_EXPIRED (0x08)

Timer is actually expired.

- #define MAX_TIMER_TICKS (0x7FFFFFFF)

    Maximum value to set.
- #define **SECONDS_TO_TICKS**(x) ((((K_ULONG)x) ∗ TIMER_FREQ))
- #define **MSECONDS_TO_TICKS**(x) ((((((K_ULONG)x) ∗ (TIMER_FREQ/100)) + 5) / 10))
- #define **USECONDS_TO_TICKS**(x) ((((((K_ULONG)x) ∗ TIMER_FREQ) + 50000) / 1000000))
- #define MIN_TICKS (3)

    The minimum tick value to set.

Typedefs

- typedef void(∗ **TimerCallback_t**)(Thread ∗pclOwner_, void ∗pvData_)

14.147.1 Detailed Description

Timer list and timer-scheduling declarations. These classes implements a linked list of timer objects attached to the global kernel timer. Unlike other kernels which use a fully-synchronous "tick-based" timing mechanism, where the OS timing facilities are based on a fixed-frequency timer (which causes regular timer interrupts), Mark3 uses a "tickless" timer implementation, which only triggers interrupts when absolutely required. This is much more efficient in most cases - timer interrupts occur less frequently, allowing the kernel to stay in sleep much longer than it would otherwise.

Definition in file timerlist.h.

14.147.2 Macro Definition Documentation

14.147.2.1 #define TIMERLIST_FLAG_EXPIRED (0x08)

Timer is actually expired.

Definition at line 45 of file timerlist.h.

14.148 timerlist.h

```
00001 /*===========================================================================
00002      _____        _____        _____        _____
00003  ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
00004 |    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
00005 |     \/   | ||     \     ||     \     ||     \     ||___   |
00006 |__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
00007     |_____|      |_____|      |_____|      |_____|
00008 
00009 --[Mark3 Realtime Platform]--------------------------------------------------
00010 
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 ===========================================================================*/
00030 #ifndef __TIMERLIST_H__
00031 #define __TIMERLIST_H__
00032 
00033 #include "kerneltypes.h"
00034 #include "mark3cfg.h"
00035 
00036 #include "ll.h"
00037 #include "thread.h"
00038 
00039 #if KERNEL_USE_TIMERS
00040 
00041 //---------------------------------------------------------------------------
00042 #define TIMERLIST_FLAG_ONE_SHOT         (0x01)
00043 #define TIMERLIST_FLAG_ACTIVE           (0x02)
00044 #define TIMERLIST_FLAG_CALLBACK         (0x04)
00045 #define TIMERLIST_FLAG_EXPIRED          (0x08)
00046 
```

```
//---------------------------------------------------------------------------
#define MAX_TIMER_TICKS                 (0x7FFFFFFF)

//---------------------------------------------------------------------------
/*
    Ugly macros to support a wide resolution of delays.
    Given a 16-bit timer @ 16MHz & 256 cycle prescaler, this gives us...
    Max time, SECONDS_TO_TICKS:  68719s
    Max time, MSECONDS_TO_TICKS: 6871.9s
    Max time, USECONDS_TO_TICKS: 6.8719s
     With a 16us tick resolution.
*/
//---------------------------------------------------------------------------
#define SECONDS_TO_TICKS(x)             (((K_ULONG)x) * TIMER_FREQ))
#define MSECONDS_TO_TICKS(x)            ((((((K_ULONG)x) * (TIMER_FREQ/100)) + 5) / 10))
#define USECONDS_TO_TICKS(x)            ((((((K_ULONG)x) * TIMER_FREQ) + 50000) / 1000000))

//---------------------------------------------------------------------------
#define MIN_TICKS                       (3)
//---------------------------------------------------------------------------
typedef void (*TimerCallback_t)(Thread *pclOwner_, void *pvData_);

//---------------------------------------------------------------------------
class TimerList;
class TimerScheduler;
class Quantum;
class Timer : public LinkListNode
{
public:
    Timer(){ m_ulInterval = 0; m_ulTimeLeft = 0; m_ucFlags = 0; }

    void Start( K_UCHAR bRepeat_, K_ULONG ulIntervalMs_, TimerCallback_t pfCallback_, void *pvData_ );

    void Stop();

    void SetFlags (K_UCHAR ucFlags_) { m_ucFlags = ucFlags_; };

    void SetCallback( TimerCallback_t pfCallback_){ m_pfCallback = pfCallback_; };

    void SetData( void *pvData_ ){ m_pvData = pvData_; };

    void SetOwner( Thread *pclOwner_){ m_pclOwner = pclOwner_; };

    void SetIntervalTicks(K_ULONG ulTicks_);

    void SetIntervalSeconds(K_ULONG ulSeconds_);

    void SetIntervalMSeconds(K_ULONG ulMSeconds_);

    void SetIntervalUSeconds(K_ULONG ulUSeconds_);

private:

    friend class TimerList;

    K_UCHAR m_ucFlags;

    TimerCallback_t m_pfCallback;

    K_ULONG m_ulInterval;

    K_ULONG m_ulTimeLeft;

    Thread  *m_pclOwner;

    void    *m_pvData;
};

//---------------------------------------------------------------------------
class TimerList : public DoubleLinkList
{
public:
    void Init();

    void Add(Timer *pclListNode_);

    void Remove(Timer *pclListNode_);

    void Process();

```

```
private:
    K_ULONG m_ulNextWakeup;

    K_UCHAR m_bTimerActive;
};

//---------------------------------------------------------------------------
class TimerScheduler
{
public:
    static void Init() { m_clTimerList.Init(); }

    static void Add(Timer *pclListNode_)
        {m_clTimerList.Add(pclListNode_); }

    static void Remove(Timer *pclListNode_)
        {m_clTimerList.Remove(pclListNode_); }

    static void Process() {m_clTimerList.Process();}
private:

    static TimerList m_clTimerList;
};

#endif // KERNEL_USE_TIMERS

#endif
```

14.149 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/tracebuffer.cpp File Reference

Kernel trace buffer class definition.

```
#include "kerneltypes.h"
#include "tracebuffer.h"
#include "mark3cfg.h"
#include "writebuf16.h"
#include "kernel_debug.h"
```

14.149.1 Detailed Description

Kernel trace buffer class definition.

Definition in file tracebuffer.cpp.

14.150 tracebuffer.cpp

```
/*===========================================================================
     _____     _____     _____     _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  _____
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||_____|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "kerneltypes.h"
#include "tracebuffer.h"
#include "mark3cfg.h"
#include "writebuf16.h"
#include "kernel_debug.h"

#if KERNEL_USE_DEBUG

//---------------------------------------------------------------------------
WriteBuffer16 TraceBuffer::m_clBuffer;
```

```
volatile K_USHORT TraceBuffer::m_usIndex;
K_USHORT TraceBuffer::m_ausBuffer[ (TRACE_BUFFER_SIZE/sizeof(K_USHORT)) ];

//---------------------------------------------------------------------------
void TraceBuffer::Init()
{
    m_clBuffer.SetBuffers(m_ausBuffer, TRACE_BUFFER_SIZE/sizeof(K_USHORT));
    m_usIndex = 0;
}

//---------------------------------------------------------------------------
K_USHORT TraceBuffer::Increment()
{
    return m_usIndex++;
}

//---------------------------------------------------------------------------
void TraceBuffer::Write( K_USHORT *pusData_, K_USHORT usSize_ )
{
    // Pipe the data directly to the circular buffer
    m_clBuffer.WriteData(pusData_, usSize_);
}

#endif

```

14.151 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/tracebuffer.h File Reference

Kernel trace buffer class declaration.

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "writebuf16.h"
```

14.151.1 Detailed Description

Kernel trace buffer class declaration. Global kernel trace-buffer. Used to instrument the kernel with lightweight encoded print statements. If something goes wrong, the tracebuffer can be examined for debugging purposes. Also, subsets of kernel trace information can be extracted and analyzed to provide information about runtime performance, thread-scheduling, and other nifty things in real-time.

Definition in file tracebuffer.h.

14.152 tracebuffer.h

```
/*===========================================================================
     _____     _____     _____     _____
 ___|    _|__ __|_    |__ __|__   |__ __| __  |__  ______
|    \  /  | ||    \     ||     |     ||  |/ /     ||___   |
|     \/   | ||     \    ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|     |_____|     |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#ifndef __TRACEBUFFER_H__
#define __TRACEBUFFER_H__

#include "kerneltypes.h"
#include "mark3cfg.h"
#include "writebuf16.h"

#if KERNEL_USE_DEBUG

#define TRACE_BUFFER_SIZE           (16)
```

```cpp

class TraceBuffer
{
public:
    static void Init();

    static K_USHORT Increment();

    static void Write( K_USHORT *pusData_, K_USHORT usSize_ );

    void SetCallback( WriteBufferCallback pfCallback_ )
        { m_clBuffer.SetCallback( pfCallback_ ); }
private:

    static WriteBuffer16 m_clBuffer;
    static volatile K_USHORT m_usIndex;
    static K_USHORT m_ausBuffer[ (TRACE_BUFFER_SIZE / sizeof( K_USHORT )) ];
};

#endif //KERNEL_USE_DEBUG

#endif
```

14.153 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/unit_test.cpp File Reference

Unit test class definition.

```
#include "kerneltypes.h"
#include "unit_test.h"
```

14.153.1 Detailed Description

Unit test class definition.

Definition in file unit_test.cpp.

14.154 unit_test.cpp

```cpp
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "kerneltypes.h"
#include "unit_test.h"

//---------------------------------------------------------------------------
UnitTest::UnitTest()
{
    m_bIsActive = false;
    m_usIterations = 0;
    m_usPassed = 0;
    m_bComplete = false;
}

//---------------------------------------------------------------------------
void UnitTest::Pass()
{
    if (m_bComplete)
    {
        return;
    }

    if (m_bIsActive)
    {
        m_bIsActive = false;
        m_usIterations++;
        m_usPassed++;
        m_bStatus = true;
    }
}

//---------------------------------------------------------------------------
void UnitTest::Fail()
{
    if (m_bComplete)
    {
        return;
    }

    if (m_bIsActive)
    {
        m_bIsActive = false;
        m_usIterations++;
        m_bStatus = false;
    }
}
```

14.155 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/unit_test.h File Reference

Unit test class declarations.

`#include "kerneltypes.h"`

Classes

- class UnitTest

  Class used to implement a simple unit-testing framework.

14.155.1 Detailed Description

Unit test class declarations.

Definition in file unit_test.h.

14.156 unit_test.h

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||____  |
|     \/   | ||     \     ||     \     ||     \     ||____  |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#ifndef __UNIT_TEST_H__
#define __UNIT_TEST_H__


#include "kerneltypes.h"

//---------------------------------------------------------------------------
class UnitTest
{
```

```
public:
    UnitTest();

    void SetName( const K_CHAR *szName_ ) { m_szName = szName_;
}

    void Start() { m_bIsActive = 1; }

    void Pass();

    void Fail();

    void Complete() { m_bComplete = 1; }

    const K_CHAR *GetName(){ return m_szName; }

    K_BOOL GetResult() { return m_bStatus; }

    K_USHORT GetPassed() { return m_usPassed; }

    K_USHORT GetFailed() { return m_usIterations -
m_usPassed; }

    K_USHORT GetTotal() { return m_usIterations; }

private:
    const K_CHAR *m_szName;
    K_BOOL m_bIsActive;
    K_UCHAR m_bComplete;
    K_BOOL m_bStatus;
    K_USHORT m_usIterations;
    K_USHORT m_usPassed;
};

#endif
```

14.157 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/writebuf16.cpp File Reference

16 bit circular buffer implementation with callbacks.

```
#include "kerneltypes.h"
#include "writebuf16.h"
#include "kernel_debug.h"
#include "threadport.h"
```

14.157.1 Detailed Description

16 bit circular buffer implementation with callbacks.

Definition in file writebuf16.cpp.

14.158 writebuf16.cpp

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#include "kerneltypes.h"
#include "writebuf16.h"
#include "kernel_debug.h"
```

```
#include "threadport.h"
//---------------------------------------------------------------------------
void WriteBuffer16::WriteData( K_USHORT *pusBuf_,
K_USHORT usLen_ )
{
    K_USHORT *apusBuf[1];
    K_USHORT ausLen[1];

    apusBuf[0] = pusBuf_;
    ausLen[0] = usLen_;

    WriteVector( apusBuf, ausLen, 1 );
}

//---------------------------------------------------------------------------
void WriteBuffer16::WriteVector( K_USHORT **ppusBuf_,
 K_USHORT *pusLen_, K_UCHAR ucCount_ )
{
    K_USHORT usTempHead;
    K_UCHAR i;
    K_UCHAR j;
    K_USHORT usTotalLen = 0;
    bool bCallback = false;
    bool bRollover = false;
    // Update the head pointer synchronously, using a small
    // critical section in order to provide thread safety without
    // compromising on responsiveness by adding lots of extra
    // interrupt latency.

    CS_ENTER();

    usTempHead = m_usHead;
    {
        for (i = 0; i < ucCount_; i++)
        {
            usTotalLen += pusLen_[i];
        }
        m_usHead = (usTempHead + usTotalLen) % m_usSize;
    }
    CS_EXIT();

    // Call the callback if we cross the 50% mark or rollover
    if (m_usHead < usTempHead)
    {
        if (m_pfCallback)
        {
            bCallback = true;
            bRollover = true;
        }
    }
    else if ((usTempHead < (m_usSize >> 1)) && (m_usHead >= (
m_usSize >> 1)))
    {
        // Only trigger the callback if it's non-null
        if (m_pfCallback)
        {
            bCallback = true;
        }
    }

    // Are we going to roll-over?
    for (j = 0; j < ucCount_; j++)
    {
        K_USHORT usSegmentLength = pusLen_[j];
        if (usSegmentLength + usTempHead >= m_usSize)
        {
            // We need to two-part this... First part: before the rollover
            K_USHORT usTempLen;
            K_USHORT *pusTmp = &m_pusData[ usTempHead ];
            K_USHORT *pusSrc = ppusBuf_[j];
            usTempLen = m_usSize - usTempHead;
            for (i = 0; i < usTempLen; i++)
            {
                *pusTmp++ = *pusSrc++;
            }

            // Second part: after the rollover
            usTempLen = usSegmentLength - usTempLen;
            pusTmp = m_pusData;
            for (i = 0; i < usTempLen; i++)
            {
                *pusTmp++ = *pusSrc++;
            }
        }
        else
        {
            // No rollover - do the copy all at once.
            K_USHORT *pusSrc = ppusBuf_[j];
            K_USHORT *pusTmp = &m_pusData[ usTempHead ];
            for (K_USHORT i = 0; i < usSegmentLength; i++)
            {
                *pusTmp++ = *pusSrc++;
            }
        }
    }


    // Call the callback if necessary
    if (bCallback)
    {
        if (bRollover)
        {
            // Rollover - process the back-half of the buffer
            m_pfCallback( &m_pusData[ m_usSize >> 1], m_usSize >> 1 );
        }
        else
        {
            // 50% point - process the front-half of the buffer
            m_pfCallback( m_pusData, m_usSize >> 1 );
        }
    }
}
```

14.159 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/writebuf16.h File Reference

Thread-safe circular buffer implementation with 16-bit elements.

`#include "kerneltypes.h"`

Classes

- class WriteBuffer16

  This class is used to provide a general-purpose, fully thread-safe circular buffer implementation which can be used for creating tracebuffers, data logging queues, transaction queues, etc.

Typedefs

- typedef void(∗ WriteBufferCallback)(K_USHORT ∗pusData_, K_USHORT usSize_)

  Function pointer type used to define a callback handler for when the circular buffer reaches 50% capacity,.

14.159.1 Detailed Description

Thread-safe circular buffer implementation with 16-bit elements.

Definition in file writebuf16.h.

14.160 writebuf16.h

```
/*===========================================================================
     _____        _____        _____        _____
 ___|    _|__  __|_    |__  __|__   |__  __| __  |__  ______
|    \  /  | ||    \      ||     |     ||  |/ /     ||___   |
|     \/   | ||     \     ||     \     ||     \     ||___   |
|__/\__/|__|_||__|\__\  __||__|\__\  __||__|\__\  __||______|
    |_____|      |_____|      |_____|      |_____|

--[Mark3 Realtime Platform]--------------------------------------------------
```

```

Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information
===========================================================================*/
#ifndef __WRITEBUF16_H__
#define __WRITEBUF16_H__

#include "kerneltypes.h"

typedef void (*WriteBufferCallback)( K_USHORT *pusData_,
K_USHORT usSize_ );

class WriteBuffer16
{
public:
    void SetBuffers( K_USHORT *pusData_, K_USHORT usSize_ )
    {
        m_pusData = pusData_;
        m_usSize = usSize_;
        m_usHead = 0;
        m_usTail = 0;
    }

    void SetCallback( WriteBufferCallback
pfCallback_ )
        { m_pfCallback = pfCallback_; }

    void WriteData( K_USHORT *pusBuf_, K_USHORT usLen_ );

    void WriteVector( K_USHORT **ppusBuf_, K_USHORT *pusLen_,
K_UCHAR ucCount_);

private:
    K_USHORT *m_pusData;

    volatile K_USHORT m_usSize;
    volatile K_USHORT m_usHead;
    volatile K_USHORT m_usTail;

    WriteBufferCallback m_pfCallback;
};

#endif
```

Index