Raytheon Marine GmbH is one of the world’s leading manufacturers of integrated bridge systems and nautical equipment such as gyro compasses, autopilots, steering control systems, monitoring systems, radar systems, electronic sea chart equipment and communication (GMDSS) for ships. In 2003 the idea was born to complete the product range of gyros with a satellite compass.

Such a GPS based compass mathematically calculates position information from two GPS receivers, as well as location information from a triaxial gyro system and a biaxial accelerometer that in affect provide compass information. The overall system should be so reliable that it can be used as a compass device in merchant shipping.

The application software for calculating compass information from the planned input values is available as Windows software. A demo is available and functional for evaluation based on a standard PC.

The goal of this paper is to demonstrate through the practical example of a project how it has been possible to cost effectively bring a saleable product to market by design, implementation and testing. It should be mentioned that the series production of the project represented here was initiated in January 2005. Theoretical backgrounds are taken into consideration only as needed.

Hardware requirements

The standard PC should be replaced in the series device with a customer-specific embedded device. The kernel must possess sufficient computing power to handle the mathematical formulas for the compass calculation in particular. At the same time, it must also enable a simple, encapsulated device design. The following interfaces must be appropriated for attaching the necessary sensors and peripheral devices:

- 2 RS-485 interfaces for the GPS receiver
- 2 RS-485 interfaces for other peripheral devices (display, input unit, etc.)
- 2 CAN interfaces for communicating with other devices on board
- 3 synchronous serial interfaces for the gyro
- 2 synchronous serial interfaces for the accelerometers

All interfaces must be able to be interrupt-driven. The data rates are between 19.2 kBits/s (RS-485) and 2.5Mbit/s (synchronous serial).

The overall design should be compact, highly shock and vibration resistant, and functional in extended temperature ranges. Since there is no display, the computer does not require graphics capabilities.

This computing unit is the so-called Connection Unit of the compass system – see figure 2. It samples the measurement data, executes the calculations and transfers the compass information and other data to the connected peripherals.

Software requirements

System software should be available on the embedded device that meets the following conditions:

- Preemptive multitasking operation
- Software drivers for all interfaces in accordance with a standard driver model
- Process synchronization
- Manipulation of input measurement data at a clock frequency of 50 Hz
- System monitoring and error handling

The system software should provide an application programming interface for the application layer to which the existing Windows application can easily be ported. No graphical user interface is necessary;
the device works without any direct user input. Communication with other devices and incoming measured values influence the operational process.

**Hardware solution design**

The CPU must be looked for in the area of the embedded controller based on the requirements from the environmental conditions for the system. The requirements on computing power also necessitate that the CPU has a floating point unit. Only the Renesas SH4 Microcontroller currently meets these requirements.

The SH4 – figure 1 – is a RISC controller with multiscalar design (2 execution units in parallel). In addition to a high-efficiency integer unit, the SH4 offers a floating point unit that has a vector extension as a feature (FPU commands for vector / matrix calculation). The SH4 also offers additional on-chip modules, separate cache memory for data and instructions, an MMU for managing virtual memory and a bus controller for easily connecting external peripherals and memory.

The SH4 is optimized for program execution from an SDRAM memory. SDRAM can be connected to the CPU with a 32-bit or 64-bit data bus. From a price/performance ratio perspective, the 32-bit data bus is the better solution.

At the time of inquiry, Garz & Frick had a fully developed SH4 CPU module in its range of products, with which all requirements on the hardware were met. The module uses the 32-bit data bus. The module would have required a redesign for the 64-bit bus, something that would have demanded additional development costs and time. The 64-bit bus would have also significantly increased the series price because double the number of RAM chips is required.

The existing SH4 CPU module was thus selected as the kernel of the compass system. The module makes all CPU signals available via industry-suited connectors, so that the customized hardware application can be connected by means of a baseboard next to the power supply. Two asynchronous and the synchronous serial interfaces, as well as one CAN interface are already implemented on the CPU module.

The interface to the gyros and accelerometers is implemented on the baseboard next to the still missing serial and CAN interfaces. Both sensor types deliver analog signals that are converted to digital values by A/D converters. The circuit is completely copied from the evaluation setup. The A/D converters provide an operating mode in which they convert cyclical data with an adjustable clock frequency. This clock frequency is led to the CPU as an interrupt and forms the basic clock frequency of the entire device.

Two ports each are combined into one interrupt for asynchronous serial interfaces. When allocating the peripherals on the ports, the GPS receivers are attached so that each receiver has its own interrupt. The second port in each case is then for the additional peripheral that can be handled at a somewhat lower priority and – depending on the end device – is also only optional.

Both CAN interfaces receive their own interrupt. They are redundant, i.e. the same data is exchanged via both interfaces independently.

Figure 2 shows the block diagram of the satellite compass system. This document refers to the connection unit only.

The hardware setup thus forms the first layer of the overall desired real-time behavior. Crucial for later overall function was that software and hardware developers were involved from the beginning and the priority module of the system was established together.

In particular, it was enormously important to also consider the application layer from the start.

Without knowledge of later functionality, decisions are otherwise made that can later lead to malfunctions of the overall system, even though the individual components by themselves are alright.

Such a thing can then only be corrected with substantial additional costs or can even lead to the failure of a project.

When developing a device, it was also of significant advantage that a development system be available for the CPU module, with which the customer could first perform trials on a hardware component in very close proximity to the target device and, in particular, could evaluate CPU performance. The development system also made it possible for application developers to port the existing software parallel to the development of the series device.

**Software solution design**

In the first step, the customer had a proprietary system software in mind much like an operating system that should be a component of development and that should carry out the required tasks.

From a technical perspective, a customized, optimal solution can thus be achieved in any case. In contrast, however, the costs of such development must be taken into account and must contain a clear risk factor, since the development risk of such a task is not insignificant.

To make the decision process easier, the view was once again directed to the hardware at this point in the discussion. Even then, the more optimal, but more expensive individual solution was not selected, but a mix of existing components with customer-specific enhancements. This solution method is even today the most efficient for the system software of an embedded device.

---

*Image 252x503 to 592x779*

**Fig 2: Block diagram Satellite Compass 21**

![Block diagram Satellite Compass 21](image-url)
Today there are a number of capable operating systems for 32-bit kernels that can be customized for the chosen hardware via a customization layer and thus can be optimally enhanced with custom drivers and program interfaces to the customer task. Every operating system manufacturer offers a number of development tools that make customization and even application programming more or less comfortable from a PC.

**Operating system choice**

At this point, the discussion about the advantages or disadvantages of individual operating systems will not be repeated. Ultimately, Microsoft Windows CE version 4.2 was selected as the operating system for this project. Only a few of the reasons that speak for Windows CE are listed here:

- A completely ordinary reason to work with Windows CE was: It was available on the development system for the CPU module. First of all, it was cost effective to start with the existing solution and to perform the evaluation.

- However, it should be clearly expressed that this did not yet constitute a decision for an operating system. In this phase, proprietary software was still a requirement.

- With Windows, PCs today are always associated with multimedia capabilities. People also like to assert that Windows is unstable and has security flaws. Windows CE itself is more recognizable under the brand name PocketPC on devices of the same name. Where did we get the idea to use Windows on an embedded device with such rigorous requirements?

  - The question answers itself by looking at some of the options Windows CE offers:
    - Preemptive multitasking
    - Scalability through modular design
    - Minimal hardware requirements, thus operational on several different platforms
    - Kernel objects such as events, semaphores, mutex for synchronization of processes
    - Thread-based priority model with 256 priority levels
    - Hard real-time capability for many operating system components

  - Win32-compatible programming interface for applications
  - A large part of the source code is available for debugging.

  On paper, Windows CE satisfies the significant requirements for system software for the compass device. Its modular design in particular makes it possible to target only those components that are needed for the device, and everything else can be omitted. Therefore, Windows CE is particularly well suited for use on systems without graphics and without a user interface. Another appealing advantage is the Win32 API. The API makes it possible to elegantly fulfill the requirement of portability of the existing application code.

  But how does the whole thing behave in practice? What is in the brochures is not crucial for the product. Crucial is, whether it is possible to make a stable, functional whole from the individual components that satisfies the requirements.

  Therefore, Windows CE was closely examined during the evaluation phase to determine if it was able to fulfill the real-time requirements. For this purpose, the customer ported to the development system during a critical routine in the application and measured performance. Conversely, Garz & Fricke tested for interrupt latency on the CPU module.

- The affected components of the software were enhanced, so that I/O signals were toggled when measuring events occurred, thus generating a signal for the measuring device. This type of measurement has the least influence on the real-time behavior of the system; other solutions that use software profilers and performance tools have in part substantial influence on the timetable of such embedded devices and should only be used for preliminary tests.

  All tests listed were performed according to this procedure.

  Windows CE performs a two-stage interrupt processing. Measured was how the timetable of processing a CAN message appears upon arrival in the CAN controller (triggers the interrupt) until transfer to the application (end IST) on the SH4 development system.

  As a result of the evaluation, it was determined that a system with an SH4 microcontroller under Windows CE appears to be capable of meeting the requirements. This result now flowed into discussion of the system software; it was initially formulated more generally: The investigation shows that the requirements placed on the selected hardware with an operating system can be achieved. That makes the development of a proprietary system software a non-issue.

  Compatibility with the desktop PC tipped the scales in favor of Windows CE and the simple portability of the existing software.

**Implementation**

The selected design made it possible to work in parallel during implementation. The application developers could perform a large part of the work on the development system. Operating system customization was also started on the development system, while hardware development manufactured the customized baseboard.

- Productive tools from Microsoft and third parties are available for software development on all levels of development. Microsoft in particular provides a complete set of tools for Windows CE free of charge for the most part.

  As soon as the baseboard was available, operating system customization was completed and verified on the target system. Application development then received the target system and completed the application programming. System testing and
optimization were then performed at the same time.

The focus of software development was on the priority model and the implementation of the required real-time behavior. For this type of system, it is not so much of interest at what maximum speed an individual data value goes through the system. It is essential that constantly incoming data and events are processed with uniform speed at predictable times (deterministic behavior). For this purpose, the required processing steps for a data interface were listed and compared with the other interfaces. The result was then the distribution of the tasks on priority levels; same / similar tasks in all interfaces always run with the same priority.

Priority model
The Windows CE priority model is somewhat different than that of the desktop Windows. The priority is uniformly assigned to a thread in 256 graduations as an execution path; the Task Scheduler distributes the computing time on the respective priority level in round robin procedure. Threads of higher priority interrupt the current thread at any time, if they become executable (preemptive). The hardware interrupts are embedded into this system.

The synchronous interfaces are implemented as a bus. All participants lie on the same data and clock lines and have an individual select signal. All synchronous interfaces are seen as one interface for reading in the data. Each time, a data packet with 5 values is read in and forwarded. The data ready signal of the A/D converter is ANDed via hardware logic and an interrupt is generated, if all converters have data.

During initialization, all converters are provided with the same settings and are started at the same time.

In operation, the interrupt must always come at the clock frequency of 50Hz. The jitter should be smaller than 1ms. This is monitored by a monitoring service running in the customization layer of the Windows CE kernel; this service derives its time base from one of the hardware timers of the CPU.

During initialization, all converters are provided with the same settings and are started at the same time.

The most important individual test was the determination of the timetable to pass a data packet from the A/D converter to the application layer. The operation includes the following steps:

- DataReady interrupt requests the next processing with the CPU (1)
- The assigned ISR in the Windows CE kernel cancels the interrupt and forwards the logical interrupt number to the kernel (2)

Not all results should now be listed here. Some exemplary data that shows what is possible in the implemented design should be shown instead. In two places should also be demonstrated that errors can occur despite careful planning and consultation between all development groups and how the know-how of the system integrator is then questioned to find a solution.

Individual interface testing
In this case, only one interface is subjected to test data. A test program running on the device receives the incoming data, logs the receipt and responds, if necessary. With this test, the time response should correspond to the series device, if possible. All performance measurements are therefore taken by external measurement devices. The individual software layers toggle when reaching specific test points I/O ports; the time response can then be charted for individual tests with an oscilloscope.

The flow of a data value is tested, several data values until the internal data buffer size is exceeded (hardware FIFO in serial blocks, transfer buffer in the individual software layers), continuous data flow with the specified data rate and with maximum data rate. Some of the results were already introduced further above for a CAN interface.

The test data are generated for the RS485 and Can interfaces by a PC test program. This program then also receives the response from the device under test and logs the test. The test data are generated with a frequency generator and a voltage source. With this interface, the accuracy of the results of the conversion is also tested.

The basis of the data processing in the compass device is the clock frequency of 50Hz, with which the data is read from the gyro. The overall system must also be capable of performing all calculation steps for processing a data packet (incl. compass calculation) in 20ms at the most. Since there are still higher-level processes with lower clock frequency, as well as system tasks that require computing time, this part should take realistically ca. 10-15ms.

www.embedded.com/europe
The kernel activates the assigned event object and starts the scheduler.

The scheduler checks the list of threads and determines that the service thread responsible for this interrupt is now executable (the thread waits for the event object that is now active).

All interrupt service threads run at the same, highest priority; if no other service thread is currently running, the scheduler interrupts the running thread and allocates the CPU to the ADC IST. If a service thread is currently running and/or other service threads are still executable, they are processed in the round robin procedure.

The ADC IST reads in the new data value from the five interfaces and transfers it to a buffer; it then activates a DataReady event object. The ADC IST can based on an event object test, whether the next higher layer has finished processing the previous packet already or not. If the previous operation is not terminated, a packet is rejected. The error correction built into algorithm recognizes and reports this.

A callback thread that waits for the DataReady event runs in the API layer. This thread receives upon initialization a callback thread from the customer application to which the data packets are transferred for calculation. This callback is the interface between application and operating system. The callback thread reads the data from the buffer and launches the callback function with the packet as a parameter.

The compass algorithm is executed. In the individual test, its time is not of interest; instead, a test function is executed that verifies the data and signals end of the cycle.

The individual times are obvious from the oscilloscope diagrams. The tests for all interfaces were performed just as is represented in this example. All time measurements relate to the level change on the interrupt lines and the point in time.

Interface interaction testing
In this step, the individual tests of the particular interfaces were carried out parallel at the same time. Because of this, the system finds itself in a condition, which actually approaches the later run in the field, but must nevertheless designated as artificial, as only dedicated test programs are running.

With the individual tests, the vertical design of the software is demonstrated to be in order and functioning.

With these tests, the system must demonstrate that also the horizontal design must additionally behave as desired and that the cooperative behavior functions on the individual layers.

Even here, all results were very satisfactory and showed that the device manages the high demands very well. As a summary from both test steps, the heavy duty real-time capability of Windows CE can be certified in interaction with the SH4 hardware.

Stress testing
During stress testing, the main issue was to determine how the system works under the highest possible continuous load, while – in addition – ‘disruptions’ are continually generated. For this purpose, test programs carried out data communication on all interfaces with the highest possible speed. Errors arising in the data or within the communication are logged. To disrupt the entire run, unusual commands are carried out over the debug connections and the generation of measurement data is impaired. The test runs for at least 24 hours.

The test confirms the previous results and shows, that the system is very robust. With this, performance tests can now be performed with the application.

The application developer naturally had to be able to fit the application to the...
customer-specific program interfaces during the implementation and the first test phases of the beta versions of the operating system and to be able to optimize the run time of the application. With the release of a fully tested version of the operating system, it must now be demonstrated, that the entire system meets the demands under field conditions.

Testing with the customer application

For this test, the commands for the placement of the hardware trigger are also included in the drivers and APIs, otherwise the operating system corresponds to the serial version. The application logs its run by means of log-outputs to the serial debug interface; the log reports are recorded with terminal software. The design of the hardware fully conforms the serial device to all peripheral devices connected according to standards.

A first test of the entire system showed namely, that the system is capable of running and actually also meets the demands; a more exact analysis of the log data, as well as the assessment of the compass algorithm showed two serious defects in the time response.

At one time, there was a gap in the data flow on one of the serial interfaces, every once in a while. The compass device sends data to the display terminal over this interface. Nothing is received over this interface.

The gap in the data flow was the clearly visible sign that something was not in order; in addition, delays arose in the clock frequency of the application, whereby the computation of errors became inaccurate. This behavior did not allow itself to be reproduced in interface testing by the operating system developer – the classical starting point for a solid dispute.

In this case, the parties did not wallow in contracts and specification manuals, as is often common, in order to establish their legal positions. The demand existed to get the product ready for the market, in cooperation with the customer. The application developer was supported by the developer of the driver and they collectively analyzed the problem. The solution then was ultimately very easy, but it also demonstrated, that not everything allows itself to be planned and arranged.

The application developer was previously familiar with the internal, proprietary system program with-in part– very specialized hardware in the embedded programming. The non-blocking, if necessary, DMA-supporting send procedure is in these systems strictly with the serial interfaces. On a standard Windows system, this is completely different. The Win32 API for the serial communication namely recognizes the non-blocking call, which is actually only implemented on a NT-based system – Windows CE supports only the blocking call of the send function.

In the application however, the programmer did not know this and had therefore unknowingly provided for a problem in the run from time to time. The specification manual did not cover this problem – the delivered system complied with the agreement, but not the expectation.

With the know-how of the system integrator, the driver programmers were in the position to upgrade the missing functionality as a customized enhancement and to provide the application with the expected behavior. At the same time, this situation also shows that Windows CE can be adapted and tailored to the current task. As the operating system producer, Microsoft makes construction kit available in Windows CE, with which the system integrator then cuts the solution to fit the need.

Correcting problems

A second problem is found in the 50Hz basic clock frequency. The control regularly notified every few seconds, that the processing of the A/D converters was slowing down and a buffer overflow was occurring. The problem could be metrologically documented – see figure 11, blue curve – however there was initially no indication as to the cause. With help of the methodology applied for the individual tests, the data flow was verified layer upon layer through continuous application. By doing so, it could be determined that the blockage was located in the application, namely in the already mentioned callback function.

However, the application programmer could not provide an indication of what was possibly going wrong; the individual algorithms all appeared to be in order. The system developer then debated all of the functions implemented in the system for the possibility whether each could induce a blockage of the system.

A function for writing error and status messages in a log book arranged in a Flash memory was carved out as a possible cause. The customer was informed during general discussions that such functions are fundamentally not real-time capable, but it was not known just how much the log book functions even used the application.

In order to clarify the issue, a test version of the operating system will be produced, where the functionality of the log book functions is commented out. The replication of the measurements demonstrates, that the problem has now been resolved.

Consultation with the application programmer confirms that he makes an entry in the log book at considerable intervals within the callback. The problem was solved by means of a reorganization of the callback.

With this example, we see that gaps can occur time and time again. A violation of the rules of real-time programming is surely the cause. Not all programmers were present during the familiarization with the application programming in the operating system and hardware and therefore, the information did not reach them all. This situation once again underscores the fact, that it is elementary for the development of such a system that all groups communicate with each other and know what each other is doing.

Conclusion

The flow of this project clearly shows how a complex development can be successfully brought to market through optimal use of resources within a relatively short period of time. Crucial for success was certainly that all parties have focused on their respective know-how and have contributed.

In terms of hardware, it was important to build the solution around existing, proven components and to not reinvent the wheel. The same thing also applies to the use of an established operating system.

Windows CE was able to demonstrate in the project that it is a real-time operating system and is also an outstanding choice for such non-graphical solutions.

Garz & Fricke has been able to contribute its competence as a Microsoft Embedded Partner, as well as Solution Provider for Renesas Microcontrollers to this project, and thus provide a customized solution to the customer.

www.embedded.com/europe