

Integrating a soft modem

By Thomas F. Herbert
Systems Scientist
Digital Video Express

In today's embedded systems with today's faster CPUs, things that were unthinkable a few years ago are now well within the realm of possibility. Moore's law says that computing power increases exponentially with time. This fact continues to govern computer and software engineering for the desktop and as prices drop, it has a trickle-down effect on embedded systems. Engineers designing embedded systems tend to design in ever increasing computer power. With each design iteration, ever faster processors find their way into smaller and more mundane products. It would have been unthinkable to put processors having the power of today's CPUs in the low-cost embedded systems of just a few years ago. Every time the technology advances we have to rethink our design decisions.

In many embedded systems designs, it is now practical to consider implementing a modem as software that shares the CPU with the rest of the embedded application. Traditionally, modems have been implemented as specialised software running in a separate DSP core, usually packaged as a separate chip; but if sufficient CPU performance is available in the main embedded CPU core, considerable hardware costs can be saved by implementing the modem in software.

In this article, I discuss some of the issues and gotchas associated with selecting soft modem IP (intellectual property) and integrating it with a software application in an embedded system.

Designing with soft modem IP

Traditionally, engineers have balanced trade-offs in deciding whether to implement a feature

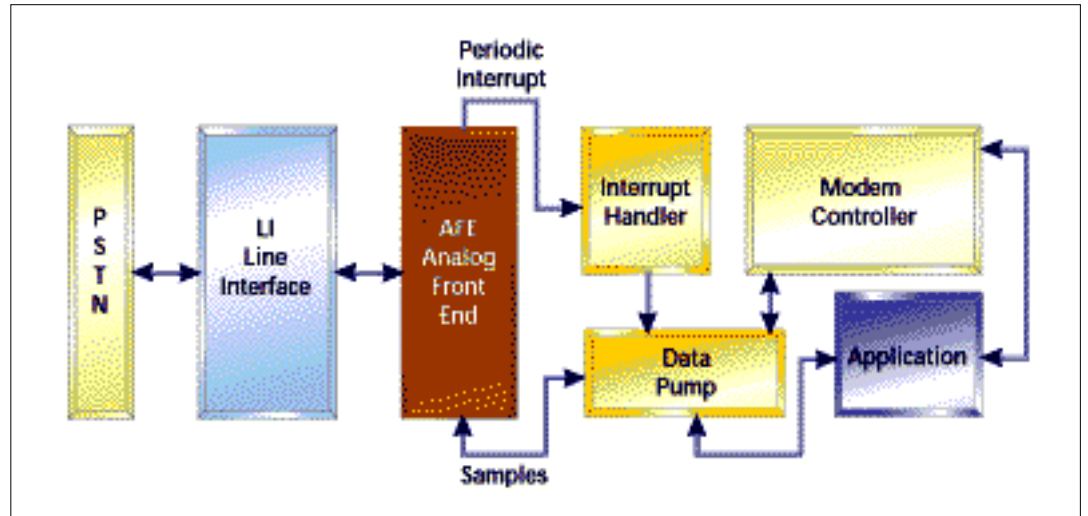


Figure 1

in software or hardware. Today's embedded system designers can exploit ever higher ASIC densities. These higher densities also give designers the choice of using higher CPU performance. Design decisions about whether to implement a feature in software or hardware are made on a feature-by-feature basis. Many of these features can often be implemented in either hardware or software and can be made available in either software or HDL (hardware description language) code.

A more complex feature can be implemented with an embedded DSP core sharing the same die with a general purpose CPU; or the same feature can often be implemented in software running on the same CPU as the rest of the embedded application. The design effort for an embedded system using a modern ASIC consists largely of integrating vendor-supplied features provided as IP with customer-supplied logic. Just a year or two ago it would have been necessary to use either a separate chip or DSP core within an ASIC to implement a modem. Many vendors can make the soft modem available in either DSP-executable code or code for a general-purpose embedded CPU.

It is a tough decision to decide whether to use a modem chip, a DSP core integrated with another ASIC, or a soft modem on the ASIC's general purpose CPU.

Often the modem is offered by a silicon vendor as a plum to encourage a designer to use a particular processor architecture. Soft modem IP is available from several independent vendors as well. Choosing a vendor for the soft modem is a tough decision. I will present some criteria that can help you make this choice.

Also, consider whether your CPU architecture is adequate to run the modem concurrently with the rest of the application. I will show that this is a complex question, but with a few basic rules and a fundamental understanding of a few key facts, it's possible to reduce the risk of a decision to implement the modem in software.

Modem basics

Many applications in set-top boxes and consumer electronics need to make occasional phone calls to exchange information with a central server. This information is required to exchange billing, financial, or authentication information between a central information system (IS) server

and end-consumer equipment. Modems are used because they are still the lowest cost connectivity option.

In this article, I limit the discussion to software implementations of "traditional" modems that are the lower bit-rate modems used to interface to voice-grade analog phone systems. We won't discuss some of the higher-speed devices of today such as ISDN (integrated services digital network), cable modems, DSL (digital subscriber loop), or RF modems. These traditional modems convert digital data to and from a modulated audio frequency within the bandwidth of a POTS (plain old telephone system) analog phone line. Although in the modern world, the PSTN (public switched telephone network) is digital, most homes are still connected to the telephone company's switching equipment with a simple analog loop and therefore are limited in available bandwidth and must use these traditional modems.

Until now, most software engineers interfacing to a modem have viewed the modem as a discrete hardware component. When integrating the modem as a software component, it will be necessary to become somewhat

more familiar with communications technology. A soft modem can't be plugged into your design as easily as a discrete modem. It will be important to look at some aspects of modem design in light of the performance requirements of your overall system. The data speed of the modem is a significant factor in system performance and most of the modems that would be required in these types of low-end embedded designs will probably be in the range of 2,400 bits per second (bps) to 28,800bps.

Modem standards

Most modem standards are governed by the International Telecommunication Union (ITU), formally known as CCITT. ITU standards for modems carry the "V" designation. *Bis* means second and *Ter* means third. Some of the pertinent standards for asynchronous transmission over an analog POTS line are listed in **Table 1**.

Modem transmission speeds are more accurately expressed in bps than baud. (*Baud* is actually short for Baudot, who was a 19th century French inventor.) The term *baud* originally applied to the number of clicks per second that a telegraph operator could send but is more recently understood to mean the number of 1-to-0 transitions per second. Baud is

not technically accurate because most modern modem modulation schemes involve a symbol or constellation that can contain several 1-to-0 transitions.

System building blocks

Any system containing a modem has some common design elements, whether the modem is implemented in software or hardware. If the design includes a soft modem, the components in your system will need to be friendlier with the modem than if it was implemented in a separate chip. Viewing the whole embedded system from the modem's point of view is helpful. This section contains a description of the major building blocks for an embedded system containing an integrated soft modem.

Figure 1 shows how the component parts fit together and how the data and control flow through the system. An analog signal is passed between the telephone network and the AFE through the line. The data pump is interrupted periodically when samples are ready at the A to D. When an input sample is read by the data pump, a sample is also output for transmission. A list of system building blocks follows.

- PSTN. Public switched telephone network
- LI (line interface) to PSTN. The LI is external to the chip and con-

tains a transformer for impedance matching, along with any other resistors and capacitors, to present the correct line characteristics to the phone line

- AFE (analog front end). The AFE contains the analog-to-digital conversion. This may be a simple linear D to A or it may be a codec (coder decoder). A codec provides compression and decompression in addition to analog-to-digital conversion, allowing it to maximise available digital signal bandwidth. The AFE also contains a timer to generate a periodic sample interrupt at the appropriate frequency
- Data pump. This is the actual DSP code that maintains the carrier frequency, modulating and demodulating the carrier with the transmitted digital data. The data pump for a system containing a codec is somewhat more complicated than a data pump for a linear D to A because of the additional processing required for the compression algorithm.

The data pump collects samples from the AFE into a buffer. When each sample is read, another is placed for transmission. The data pump also presents circular buffers for interfacing to the modem controller and the application. The data pump

runs in batch mode, processing a set of samples in each iteration. The data pump must run and complete before the next set of samples is ready

- Modem controller. The modem control code is responsible for carrier detection, call progress monitoring, and command interpretation. It is also responsible for call termination, whether initiated by the other modem or by local intervention. The controller implements the AT command set, probably the best choice for the modem's application programming interface (API). In a stand-alone modem, the AT command set is the user interface, but in an embedded modem, it is the basis of the API. It is better to implement a standard AT command set to maintain compatibility with implementations using a discrete chip.

The modem controller interprets the AT commands and controls the state of the data pump. It also inserts the result codes of the commands into the buffer for interpretation of the application

- Application. The application includes any necessary background processing that has to run while a phone call is in progress. This, of course, is the customer-added value in your particular embedded system. It includes any data filtering or processing, as well as writing and reading the data to and from permanent storage.

The application places the data for transmission into the data pump's transmit buffer. It also reads any converted data from the data pump's receive buffer

- Real-time multitasking OS. Most systems containing a real-world soft modem application will also have one or more other tasks running concurrently with the modem. An OS is important to manage these multiple tasks running at multiple priorities. The OS should be capable of arbitrating CPU bandwidth between the

TABLE 1	Pertinent standards for asynchronous transmission over an analog POTS line
V.21	300 bps modulation
V.22	1200 bps modulation
V.22bis	2400 bps modulation
V.32	9600 bps modulation
V.32bis	14.4 Kbps modulation
V.34	28.8 Kbps modulation
V.42	Error Correction
V.42bis	Data Compression
V.56bis	ITU modem testing standard
V.25	Answer tone generation and echo cancellation
Hayes 2400B	"AT" command set pseudo standard
TSB 37a, TSB 38	EIA/TIA Modem testing standards

various tasks, including the data pump, the modem controller, and any other application tasks in the system

Selecting soft modem IP

Modem design is highly specialised. Many of the soft modem IP providers are linked to the semiconductor and design houses. The modem IP is offered as an incentive to customers to tie them to a particular vendor's processor architecture. It's important to look at the soft modem as an important and complex component in an overall system design. The software integration process is complicated and soft modem IP has to be selected very carefully when it is to be used as an integrated component in your software. You should consider the following criteria when selecting a soft modem vendor.

Soft modem design stability. The stability of the soft modem vendor's product is an important consideration. The number of design wins can be an indicator of software stability. Try to select modem IP that has been implemented in a real-world system. Get your vendor to give you names of other customers that have successfully integrated a system or delivered a product containing the soft modem. Have the vendor demonstrate the soft modem. Check to see if the vendor has a reference design available, and scrutinise the reference design for similarities to your own proposed design. Try to borrow an implementation of the reference design for testing.

Processor architecture. Try to select a soft modem that has been implemented in the processor architecture you're using. Also, soft modems are usually written to take advantage of special architectural features in your CPU. Check to make sure your processor has the same features, as this will be important to ensure you can reach the same levels of performance as the vendor's reference design. For example, a guaranteed level of performance may be dependent on having fast

multiplier hardware to accelerate the fixed-point multiplies in the data pump.

Code optimisation is important, and optimisation has architectural dependencies. Even if the data pump code is written in C, it will need to be optimised to the particular processor architecture used in your design. Ensure that your soft modem IP vendor has sufficient experience with your processor architecture.

Performance. Your vendor will provide some performance numbers. Often performance is expressed in CPU MIPS (millions of CPU instructions per second) that the modem will consume. Of course, as we discuss elsewhere in this article, modems are a hard real-time application, and therefore sufficient modem performance can be a determining factor in the success or failure of your design. The vendor should express the performance in terms of the minimum and maximum times necessary to process a given number of samples through the data pump. These numbers should be stated and verifiable on a particular CPU architecture.

Cache and memory architecture. Determining how the modem will perform in your particular hardware architecture is important. CPU speed and architecture are the first things to consider when predicting system performance, but cache and memory architecture are equally significant. Of course, accurately gauging the required level of modem performance is very important. You'll need a few answers to a few questions about the memory architecture in your embedded design.

Determine whether the soft modem code will run in ROM or RAM and the number of wait states for each, as these will be large factors in gauging modem performance. Also, the bus width of your system will determine whether data will be multiplexed on the external bus. The performance requirements for the data pump will generally require a 32-bit processor. If the 32-bit processor is running with an external 16-bit bus, the number of

wait states per memory access is effectively doubled.

Cache is probably the largest factor in performance. Instruction cache is probably more important than data cache as a factor in increasing soft modem performance. Determine the penalty in number of wait states charged to a cache miss. The cache will need to be adequately sized for the modem data pump in order to minimise the cache hit/miss ratio, so the main loop in the data pump code can almost always be cache resident.

Standards compliance and reliability. The modem should faithfully implement the applicable V specifications. The V.56 standard, as well as the TSB 37a and the TSB 38 standards, specify loop tests to verify modem compliance. Your soft modem vendor should be able to supply testing results run against one or more of these standards to verify compliance. Test equipment is available that can run these loop tests, adding varying amounts of attenuation and noise.

The best way to determine the reliability is to pass truckloads of data through the modem for a long period of time using degraded line conditions. This will help expose any buffer overrun conditions, loose pointers, or data sensitivity.

You'll want to retest your system for compliance after the soft modem is integrated with your ASIC and AFE. There are a number of good contract modem test houses that can test your modem implementation against the applicable standards. Generally, the test houses are accustomed to work with stand-alone modems with a UART and a DTE (data terminal equipment) interface. You should probably plan for the development effort necessary to supply a standard UART-based DTE interface if this isn't already part of your design.

Software interface. Modems traditionally implemented the "AT Command Set," to be compatible with the Hayes modems that were dominant in the market in those ancient times. This tradition still holds true

today. Most modern software using modems has for many years been written to use the AT commands for modem control. Even discrete modem chips typically implement the AT commands. For compatibility, it's best if the soft modems also implement these commands. The modem can then be more easily tested by a standard test suite. A UART driver can be fitted relatively easily to convert your application to look like a stand-alone modem to facilitate testing.

Another aspect of the software interface to consider is the buffering. Implementing circular buffers for interface to and from the modem data pump is best. This approach will insulate the modem running at a synchronous rate from other asynchronous software. The buffer sizes should be tweakable.

Analog section. Design differences exist among various AFEs. The vendor's code should provide adjustable constants so the modem can be tuned to interface with different AFEs. For example, the vendor may advertise that the soft modem will work with either a codec or a linear AFE. Even if the AFE design is known, its impedance characteristics may change as the design goes into production. The data pump should be easily tuned to compensate for any AFE characteristics that may change.

Once your embedded design goes into production, there may be some component value variation that will require re-tweaking the data pump to compensate for differences between the production analog components and the design.

Configurability. Traditional modems using the AT command set have provided "S registers" for setting modem configuration parameters and establishing modem default behaviour. The S registers include settings for the number of answer rings, the number of seconds to wait for dial tone, whether result codes are numeric or verbal, and whether dial tone or pulse is the default.

The S registers may be used

to provide adjustability while avoiding unnecessary coding changes. Also, the S registers can be extended to provide additional configuration parameters to support specific features in a customised modem design. The constants governing the default values for the S registers should be settable at compile time. Any gains and values needed to adjust the modem to changes in the AFE should be mirrored in the S registers. The S registers are used to adjust modem performance by issuing AT commands instead of by making code changes.

Tools and library configuration. Knowing what compilers and linkers are supported by your soft modem IP vendor is important. The vendor's code should be provided in library form and of course, the library file format should be compatible with your linker. If all or part of the modem is in source, you'll want to make sure that the modem has been built and tested with the compiler you plan to use in your system. If the soft modem is written completely or partially in C, have the vendor specify what compiler optimisations are required.

Will there be enough system performance?

Before you can seriously consider implementing the modem in software, you must guarantee that the system design allows for adequate performance. Unfortunately, this guarantee is more complicated than you might think. If you make the assumption that your CPU is a RISC processor, making performance comparisons is a little easier because the MIPS rating usually scales linearly with CPU clock speed. The hard part is factoring in allowances for memory wait states and memory bandwidth limitations, and other concurrent CPU activity.

It is important to distinguish between hard real-time and pseudo real-time requirements. Pseudo real-time systems require latencies to be within a certain range, but if the latency is exceeded, system performance merely degrades. In hard real-

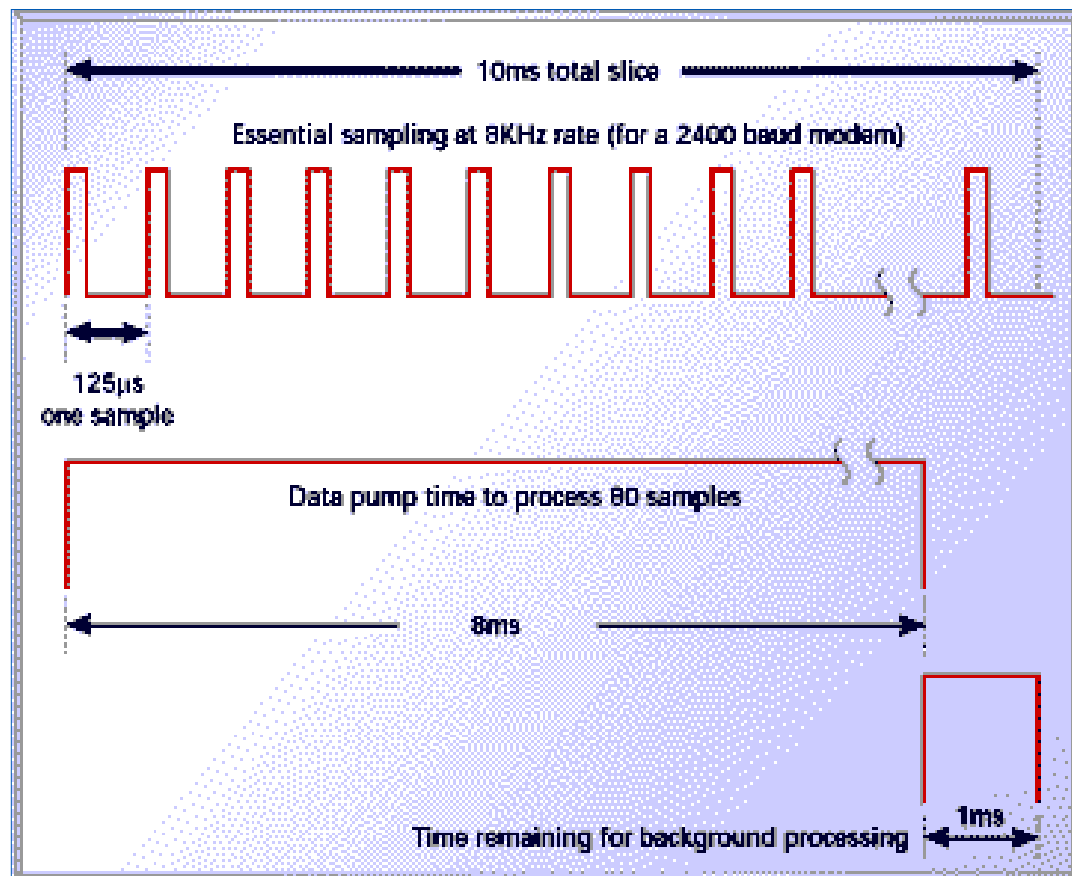


Figure 2

time systems, if latency requirements are exceeded, the result is a catastrophic failure. A modem is definitely a hard real-time application. Modems require a sample rate sufficient to digitally construct the carrier and modulation tones. Also, the data pump must complete processing a set of samples before the next set of samples is ready. The carrier tone is constructed digitally; therefore, if the sample rate isn't met, the modem will fail to sustain the carrier and drop the call. This is true even if no data is being transmitted.

Complicating the picture even more is the fact that overall system activity and overhead must be taken into account when calculating the CPU bandwidth required during the call. In a hardware or discrete modem implementation, the modem can gobble the entire CPU, but in a soft modem implementation the modem must share the CPU with other activity. Although other activity can often be assumed to be minimal, the modem is such a CPU hog that knowing the required performance for all system

activity is very important. You can thereby ensure that there are enough CPU cycles left for necessary background application software. You must also carefully calculate interrupt servicing and any other operating system and event servicing overhead.

The guts of the modem consist of the data pump or modem process. The data pump needs to process a fixed number of samples in a given time period, usually about 10ms or 20ms. During each time period, the next set of samples is being collected while the previous set of samples is processed to and from the modulated analog signal. In other terms, during period n , the samples are collected for period $n + 1$. For a 2400 baud modem, an 8KHz sample rate is required. In order to collect the next set of samples, the system must be able to service 80 interrupts during a 10ms period.

The system must be designed with the ability to assign task priorities. The highest priority is the reading and writing of analog samples to and from the D to A. This servicing should be done

at the highest priority interrupt level available in your CPU architecture, guaranteeing the lowest possible interrupt latency.

The next higher priority is the data pump. In the previous example, if the data pump code wasn't able to complete processing all 80 samples in 10ms, the modem wouldn't be able to sustain the carrier. In most designs, the modem will accumulate some number of samples in a buffer and process those samples while gathering the next set of samples. If the data pump uses all of the 10ms to process the samples, there won't be any additional available CPU bandwidth for any necessary background tasks. Also, any other interrupt activity in the system gets subtracted from the time available to the data pump.

Any system overhead for sampling interrupts, task context switches, and other interrupts must be minimised. The hardware and software interrupt latencies can be the worst consumer of CPU bandwidth. The most significant of these factors is the sampling interrupt. For

example, let's assume that the sample interrupt occurs at an 8KHz rate. If the interrupt service routine (ISR) takes only 12µs, a total of 0.96ms is consumed for every 80 samples. This would take almost 10% of the available CPU away from the essential data pump task. You must be able to guarantee that each sample will be acquired from the AFE well before the next one is ready.

Having a circular buffer between the AFE and the data pump is preferable. This buffering allows for a little slop in the timing of the data pump. Even if the data pump is a little late getting started due to other system activity, a few samples can be held, allowing the data pump to overrun its time slice by a little bit.

Providing a FIFO on the D-to-A input and output is also a good idea. Without buffering, 90% or more of processing required to read a sample is overhead generated by the ISR. This buffering will reduce the required number of interrupts needed to maintain the sampling rate. For example, a 16 deep FIFO would reduce the interrupt rate from 8,000 per second to 500 per second.

In order to sustain the call, a carrier tone must be generated and sustained in software. A certain minimum amount of CPU bandwidth will be required, whether or not any data is being

transmitted or received by the modem. The need to maintain the sampling is one of the most significant factors in contributing to this overhead. You should consider two aspects of sampling. The first is allowable latency and the second is adequate CPU bandwidth.

Figure 2 illustrates the timing considerations. It shows an approximation of the timing (not to scale) of a system containing a 2400 baud V.22bis modem with a 8KHz sample rate. This causes 80 sample interrupts in a 10ms period. The 80th interrupt triggers the start of the data pump. In this figure you can see the allocation of available CPU bandwidth between the sampling interrupts, the data pump, and background processing.

What other activities must occur while the modem is off-hook? When calculating if there is adequate CPU bandwidth to handle the modem process, don't forget to include the overhead associated with any other background processing that must occur. For example, if the modem is transmitting or receiving data (and it will be most of the time it is off-hook), there is a need to read or write the data to temporary or permanent storage, such as flash. Understand what the CPU bandwidth requirements are for this data reading and process-

ing. If the modem process and the sampling interrupt consume most of the CPU, there may not be enough CPU bandwidth to adequately process the data.

OS considerations

What do operating systems have to do with systems containing soft modems? CPU allocation must be arbitrated so the modem process will have execution priority while ensuring that sampling interrupts aren't missed. Also, any leftover CPU bandwidth needs to be allocated to at least one background task that monitors call progress. There will probably be other background tasks to store and retrieve the data being transmitted through the modem. If your design includes a real-time multitasking OS, it will be easier to arbitrate these varying needs.

Because the D-to-A input must be read before the next sample or samples are ready, the modem processing must be done at a lower priority than the interrupt level so the processor is ready to receive the next interrupt.

A good multitasking OS has the ability to assign task priorities to help allocate the CPU loading. The OS should stay out of the way of the sampling interrupts so they can be done at the highest hardware priority level to ensure minimum latency. The challenge for the OS is to ensure that the data

pump task will get enough CPU resources to finish processing its set of samples before the next set of samples arrives. At the same time, the OS should allocate any excess CPU cycles for background processing. The background processing includes one or more tasks that may be monitoring call progress, storing and receiving the call transmission data, or performing any other jobs unrelated to the modem. The OS should provide semaphores for task synchronisation and queues for task intercommunication.

A significant decision

A soft modem is a good choice for embedded systems implemented for cost-sensitive applications. The soft modem can reduce chip count or chip complexity and therefore can provide a significant cost savings for embedded system designers. Selecting a soft modem is a decision that must be taken seriously. It can have a profound impact on system and software design and performance. Nevertheless, a soft modem is clearly the right answer for inexpensive embedded systems with sufficient CPU bandwidth and an architecture that is designed to optimise soft modem performance.



[Email](#)



[Send inquiry](#)