

# **V.22 bis Modem on Fixed-Point TMS320C2xx DSPs**

*Digital Signal Processing Solutions*

## **Abstract**

This document describes a V.22 bis modem implemented on the Texas Instruments (TI™) TMS320C2xx digital signal processor (DSP). The V.22 bis modem standard is a QAM type modem at 2400 bps, implemented in full duplex. The V.22 bis modem implementation includes transmitter, receiver with scrambling and V.14 as well as the handshake for connection in originate and answer mode. Modulation and demodulation are carried out at a sampling frequency of 8 kHz.

## **Contents**

V.22 bis/V.22 Modulator .....	2
Theoretical Considerations .....	2
Description of the V.22 bis/V.22 S/W Module .....	5
V.22 bis/V.22 Demodulator .....	7
Global Structure .....	7
Interface .....	8
Symbol Demodulator .....	9
V.14 and Descrambling.....	10
V.22 bis/V.22 Interface .....	11
Interrupt Interface.....	11
User Interface .....	11
Interface Functions and Variables.....	12
Carrier(void) .....	12
CID .....	12
Fdisconnect(void).....	12
GetData(void).....	12
V14Send(Data) .....	13
V14Stat(void) .....	13

## **Figures**

Figure 1. Main Transmitter Stages of a Baseband Modulator .....	2
Figure 2. QAM Modulator .....	3
Figure 3. V.22 bis Constellation .....	4
Figure 4. Generation of the Modulated Sine Wave .....	5
Figure 5. Flow Diagram of the Modulation Routine .....	7
Figure 6. Receiver.....	8
Figure 7. Symbol Demodulator .....	10

## Tables

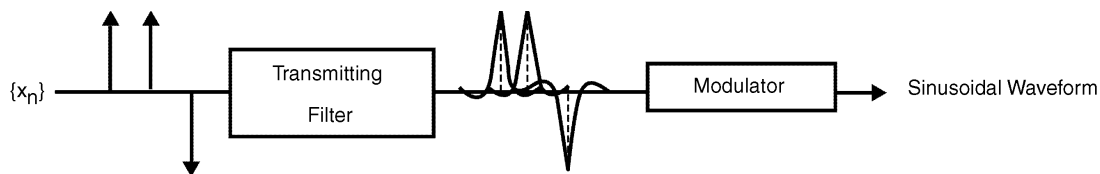
Table 1. Phase Changes for a Given Combination of Two Bits in v.22 bis .....	4
Table 2. Variables Used for Sample Management in the Modulation Sequence .....	6
Table 3. Values Taken by Variables in the Modulation Sequence.....	6
Table 4. Commands.....	9

## V.22 bis/V.22 Modulator

### Theoretical Considerations

In this paragraph, the basic aspects of baseband modulation that apply to V.22 bis standard are summarized. The main transmitter stages are shown in Figure 1, below:

Figure 1. Main Transmitter Stages of a Baseband Modulator



The transmitting filter aims at reducing to the minimum the bandwidth required to transmit a specified signal. At the filter output a given input bit or symbol is represented by the filter shape. The filter has to be designed in such a way as to avoid inter-symbol interference (ISI). The optimum bandwidth without ISI is achieved with a rectangular filter of a bandwidth  $W = R_s / 2$  (where  $R_s$  denotes the symbol rate in symbols/s). This relation is called the Nyquist bandwidth constraint. As ideal rectangular filtering is not physically realizable, a frequently used filter is the so-called raised-cosine filter. The filtered signal is perfectly band limited without introducing ISI. However some "excess bandwidth" is needed beyond the theoretical minimum. The filter transfer function is given by:

$$H(f) = 1 \quad \text{for } |f| < 2W_0 - W$$

$$H(f) = \cos^2 \left[ \frac{\pi(|f| + W - 2W_0)}{2(W - W_0)} \right] \quad \text{for } 2W_0 - W < |f| < W$$

$$H(f) = 0 \quad \text{for } |f| > W$$

where  $W$  is the absolute bandwidth and  $W_0 = R_s / 2$  the minimum bandwidth. Thus the difference  $W - W_0$  is the excess bandwidth. The roll-off factor is defined as:

$$r = (W - W_0) / W_0$$

The roll-off factor specifies the excess bandwidth as a fraction of a given minimum required bandwidth  $W_0$  (where  $r = 1.0$  -> excess bandwidth = 100%). The required bandwidth  $W$  can also be written as a function of the symbol rate,  $R_s$  (symbols/sec):

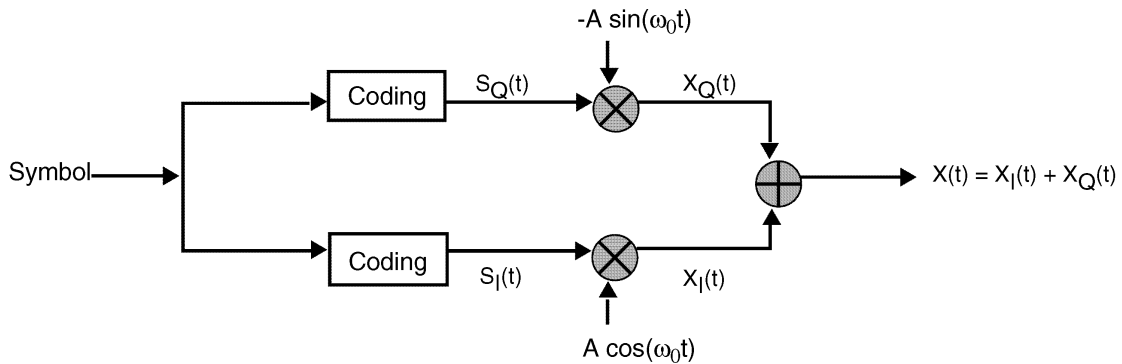
$$W = \frac{1}{2} (1 + r) R_s$$

These considerations apply to baseband signals. Baseband modulated signals require twice the bandwidth of equivalent baseband signals. Thus the minimum required bandwidth for frequency translated signals is given by:

$$W = (1 + r) R_S$$

Digital baseband modulation converts a symbol into a sinusoidal waveform of duration  $T$ . The V.22 bis modem standard uses QAM (Quadrature Amplitude Modulation) which consists of two independently amplitude-modulated carriers in quadrature. The amplitude and phase are varied according to the symbol transmitted. The principle of QAM modulation is illustrated in Figure 2.

Figure 2. QAM Modulator



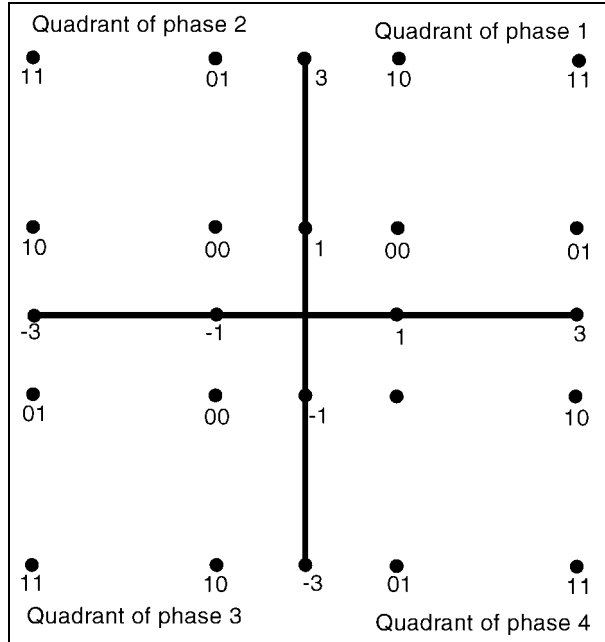
The V.22 bis standard uses symbols that consist of 4 bits (quadrubits): 2 bits are used for amplitude coding and 2 for phase coding:



Table 1. Phase Changes for a Given Combination of Two Bits in v.22 bis

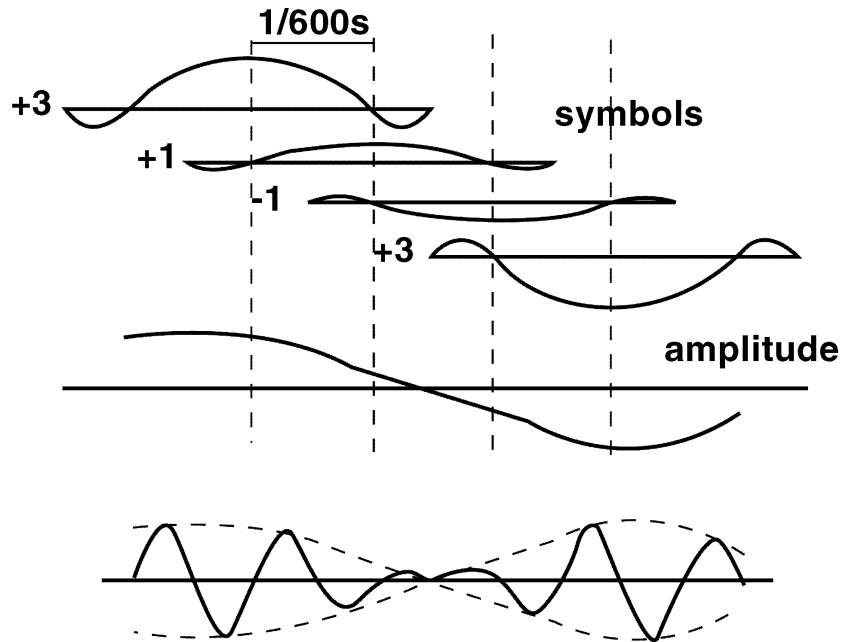
First Two Bits of Quadribits (2400 bps)	Quadrant Change	Phase Change
00	1 → 2 2 → 3 3 → 4 4 → 1	90°
01	1 → 1 2 → 2 3 → 3 4 → 4	0°
11	1 → 4 2 → 1 3 → 2 4 → 3	270°
10	1 → 3 2 → 4 3 → 1 4 → 2	180°

Figure 3. V.22 bis Constellation



Amplitude modulation is carried out with the following amplitudes: -3, -1, 1, 3. Each of these values corresponds to a combination of 2 bits. A symbol is represented by a pulse shape given by the square root of the raised cosine with a roll-off factor of 75% ( $r = 0.75$ ). The square root raised cosine filter is used to maximize the signal to noise ratio at the output (matched filter). A new symbol is sent every  $1/600^{\text{th}}$  of a second. However, a symbol spreads out about  $3/600^{\text{th}}$  of a second, so that the sum of three superimposed symbols has to be computed. This is illustrated in Figure 4.

Figure 4. Generation of the Modulated Sine Wave



## Description of the V.22 bis/V.22 S/W Module

### Sample Management

The V.22 bis/V.22 software module works at an 8-kHz sample rate. This means that a symbol consists of  $13 \frac{1}{3}$  samples. As only entire sample values can be processed, the following modulation sequence is implemented: 14 samples, 13 samples, and 13 samples. At the end of each modulation sequence, three complete symbols have been sent. One complete symbol is processed at a time. For this purpose, a buffer of  $2 \times 14$  words is used. One half of the buffer is used to output the modulated signal to the line (output buffer) and the other half is used to compute the next symbol representation (modulation buffer).

Three different root raised cosine pulse shapes are used for each step in the modulation sequence (14-13-13). The shape for a symbol consisting of 14 samples is symmetrical about the y-axis, whereas the shape for a symbol containing 13 samples is left or right shifted by  $\frac{1}{3}$  of a sample. Table 2 summarizes variables used for sample management in the modulation sequence described above.



Table 2. Variables Used for Sample Management in the Modulation Sequence

Variable Name	Description
FifoOut	Buffer for modulation (output and computation), 2x14 words
ModulSeq	Step in modulation sequence 14-13-13 samples
ModemOut	Pointer to modulation buffer
Shape	Pointer to table containing the shape for 14-13-13 samples/symbol
NbSamp	Number of samples that are to be generated
FifoOutCnt	Number of samples for output
P0Tmp1	Temporary pointer to output buffer

For each step in the modulation sequence the pointers ModemOut and P0Tmp1 can point either to the beginning or to the end of the buffer FifoOut. In the case of 13 samples per symbol, the pointer at the beginning of the buffer points to the second buffer element. The values of the variables for each step in the modulation sequence are given in Table 3.

Table 3. Values Taken by Variables in the Modulation Sequence

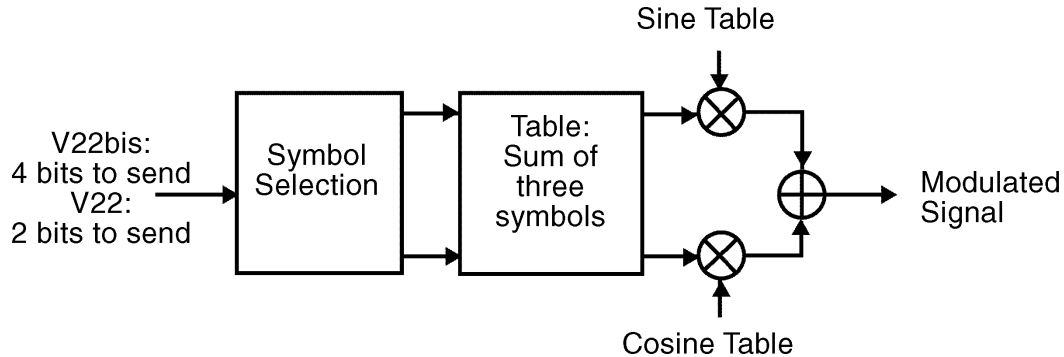
Variable	Step1		Step2		Step3	
ModulSeq	Current:2	Next:1	Current:1	Next:0	Current:0	Next:2
ModemOut	Previous	Current	Previous	Current	Previous	Current
	FifoOut+1	FifoOut+14	FifoOut	FifoOut+14	FifoOut+1	FifoOut+14
	FifoOut+14	FifoOut	FifoOut+14	FifoOut+1	FifoOut+14	FifoOut+1
P0Tmp1	Previous	Current	Previous	Current	Previous	Current
	FifoOut+14	FifoOut+1	FifoOut+14	FifoOut	FifoOut+14	FifoOut+1
	FifoOut+1	FifoOut+14	FifoOut+1	FifoOut+14	FifoOut	FifoOut+14
XShape	Xshape1		Xshape0		Xshape2	
NcSamp	14		13		13	
FifoOutCnt	12		13		12	
	Generate 14 samples		Generate 13 samples		Generate 13 samples	
	Output 13 samples		Output 14 samples		Output 13 samples	

The 8-kHz sample interrupt for V.22 bis/V.22 is contained in the file V22IT.ASM.

## Modulation Routine

The modulation routine is contained in the file MODUL.ASM. It is called every 14 or 13 samples (one complete symbol). The function Modul carries out QAM modulation. The flow diagram of the modulation routine is shown in Figure 5.

Figure 5. Flow Diagram of the Modulation Routine



The same modulation routine is used for V.22 bis and V.22. First the impulse response representing the current symbol is determined. This is performed by the routine ENCODE contained in the file ENCODE.ASM. Therefore the symbol (constellation point) is determined according to the bits to be sent. The bits to be sent are contained in the variable XBITS.

Bits 3 and 2 determine the phase increment according to Table 1. Bits 1 and 0 determine the amplitude as shown in Figure 3. Each entry of the constellation table (label: *NewConst*) contains the real part in the 8 LSBs (least significant bits) and the imaginary part in the 8 MSBs (most significant bits). We can now determine the pointer to the raised cosine table from the new constellation in combination with the two previous constellations. *PtSin* points to the real part and *PtCos* to the imaginary part. The root raised cosine tables are labeled *XShape0*, *XShape1* and *XShape2* depending on the step in the modulation sequence (see Table 3).

After the routine ENCODE has been carried out, the shape which represents the current symbol is multiplied by the carrier. In Originate mode the carrier frequency equals 1200 Hz, in Answer mode the carrier frequency equals 2400 Hz. The real part of the pulse shape is multiplied by sine whereas the imaginary part is multiplied by cosine. The sine and cosine tables of the corresponding carrier frequencies are contained in the file *cos12\_24.tbl*. The results of the multiplication are added up and multiplied by a gain (variable *TVGain*). The gain is set so as to obtain an output level of -10 dBm on the line.

### Guard Tone

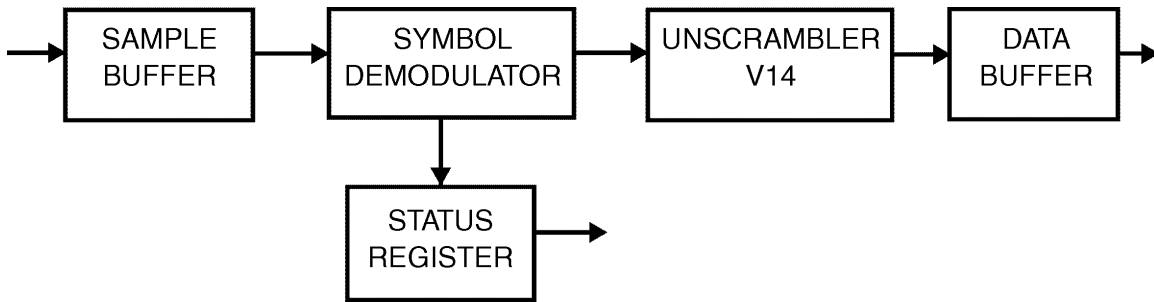
For Answer mode, a guard tone of 1800 Hz is implemented. The decision whether or not to send the guard tone is made by the AT command "at&gX" (X=0: no guard tone, X=2: guard tone at 1800 Hz, see the AT command set). If the command "at&g2" has been executed, the guard tone is added to the modulated signal in answer mode. The guard tone table is labeled *Tab1800*.

## V.22 bis/V.22 Demodulator

### Global Structure

The demodulator structure is as follows:

Figure 6. Receiver



The sample interrupt loads the sample buffer. When there are enough samples (cf. task switch) the task switch calls the demodulator routine. The demodulator routine will work on one symbol. Once the symbol has been demodulated the routine checks if there are enough samples to start a new symbol. If so, the routine starts again; if not, it returns to the task switch.

At the end of the symbol demodulator the routine gives 4 (V.22 bis) or 2 (V.22) demodulated bits. It then calls the unscrambler and the V.14 routine. This routine is a UART function that will reconstruct a block of 8 bits of data, eliminating start and stop bits. When a block of data is created it is put in the data buffer.

## Interface

The interface of the demodulator with the upper layer is made through variables `mdm_sta`, `V22TR1`, `V22TR2`, `ALPHA`, `CID`, `speed` and functions `SendCmd()`, `StartUart()`, `GetData()`, `modemini`.

`mdm_sta` is a status register (read only). The 4 LSB of `mdm_sta` are the last 4 demodulated bits. The next 4 bits come from the `STWRD` defined in `DSP20.I8X`. These bits are: `S0` detector, `S1` detector and energy detector. When the gain of the AGC is lower than `V22TR1` the energy detector bit is set to 1 (carrier detected), when `ALPHA` is higher than `V22TR2` the energy detector bit is set to 0 (carrier lost).

Variable “speed” defines the modem speed (2400 or 1200) for the V.14 function. Some commands send with `SendCmd()` specify V.22 or V.22 bis but have no influence on the variable “speed”. “Speed” must be set before using V.14 (`StartUart()`) to allow it to use the right demodulated bit.

Variable `CID` controls the way sample interrupt (`main.asm`) works. It is set to `CIDMode` after reset, set to 0 (V.22/V.22 bis mode) by `modemini()` (`V22c.c`) and set back to `CIDMode` by `FdisConnect()` (`mainc.c` and `main.asm`) when the modem loses carrier. `CID` must be set to 0 by `modemini()` because, when its value is 0, the sample interrupt routine calls the V.22/V.22 bis routine which is initialized by `modemini()`. Because the V.22/V.22 bis function uses the same memory as the `CPTD` function, the lack of initialization or a call to the `CPTD` function will hang the software.

Function `modemini()` initializes the V.22/V.22 bis modem function. It must be called before any use of the interface of V.22/V.22 bis. The V.22/V.22 bis function uses the same memory as the `CPTD` function, so the memory has to be cleaned before starting.



Function SendCmd() is used to configure the V.22/V.22 bis modem. The commands are words of 8 bits. The 4 MSBs give the command, the 4 LSBs the parameter. The available commands are:

Table 4. Commands

Code	Action	Parameter
1X	Send X and force 2400bit/s	Data send
2X	Send X and force 1200bit/s	Data send
3X	Reset IRCNT (synchro)	No
4X	Enable time recovery	3 (good value)
7X	Receiver mode	0: receive idle 2: receive data
8X	Transmit mode	0: transmit idle 2: transmit enable
FX	Miscellaneous	1xxx : answer mode 0xxx : calling mode xx01 : 2400bit/sec xx11 : 1200bit/sec Bit 2 is not used

SendCmd() puts the command into a FIFO, so the command is not immediately executed. SendCmd() signifies whether there is enough space in the FIFO. Only one command is taken every transmitter symbol. When there is no command available the last command is repeated. The speed of the modem can be fixed by command 1x, 2x or Fx. When StartUart() has been called, the last activated command is used by V.14 to transmit data. Two command sendmarks (S1200) are sent before calling StartUart() in file V22b.c line 459 to be sure that the modem will start in V.22 (not in V.22 bis). This guarantees the activation of the command.

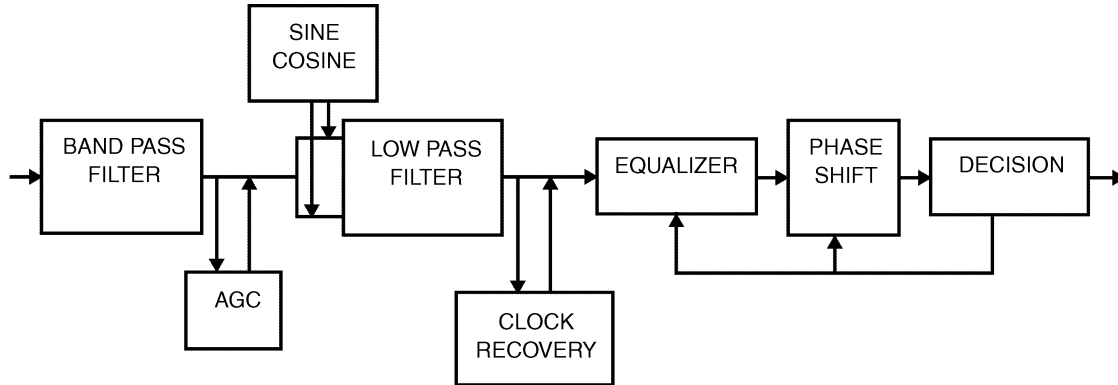
From a state where the user must give to the modulation the scrambled bit to be sent, function StartUart() switches to a state where the V.14 routine creates the bit to be sent from what it reads in its buffer. StartUart() also starts the interpretation of the received bits as V.14 bits and put them into a buffer. The decoded data in the buffer are available through function GetData().

Function GetData() returns information from the V.14 receive buffer. If there is data available, it returns the 8 bits of the data as MSB and sets FLAG20 to 1. FLAG10 set to 1 indicates an overflow in the buffer and is reset after calling GetData(). If there is no data available, FLAG20 is set to zero.

## Symbol Demodulator

The organization of the symbol demodulator is as follows:

Figure 7. Symbol Demodulator



The symbol demodulator demodulates samples of a symbol before the unscrambling function. A symbol is made of the samples for transmitting 2 (1200 bps) or 4 (2400 bps) bits. The first operation is the bandpass filter to eliminate the noise from the band and the echo of the transmission. After filtering, the signal is normalized by the AGC.

The AGC is also used for carrier detection. The gain of the AGC (ALPHA) is compared to two thresholds to determine whether there is a carrier. Demodulation follows the AGC and consists of multiplication of the signal by the sine and cosine of the carrier. This creates two paths (I and Q), which carry the complex number—the signal manipulated from the low-pass block up to the decision block.

A low-pass filter follows multiplication by sine and cosine. The output signal is now the amplitude of the carrier. This complex amplitude is undersampled with an interpolator to have four samples for each symbol (600 Hz). Two of these four samples are used for the clock recovery, which control the interpolator. The equalizer uses the other two samples. After equalization and a phase shift, the decision box determines the bit received by means the complex number. It also computes the error vector that is used by the equalizer and the phase shift to converge to the best value.

To have correct convergence of the equalizer, the AGC and clock recovery must have performed their own convergence. Once this is done the equalizer can converge if the incoming signal is sufficiently random. This condition exists in the sequence of receiving the scrambled one. So the equalizer must remain blocked up to that moment. The command allowing the convergence of the equalizer to start is `SendCmd(0x43)`.

## V.14 and Descrambling

The V.14 is a function that converts synchronous data to asynchronous data and vice-versa. On the transmit side, the incoming data is asynchronous, with start bits and stop bits. The modulator itself works synchronously, transmitting  $600 \times 4$  bits per second, so the V.14 function is placed between these two environments. The interface we have with our V.14 function is based on the 8 useful bits (without the start and stop bits).



These 8 bits are sent to the V.14 with the function V14Send(), which puts the 8 bits in a buffer. The V.14 function is called 600 times per second by the modulator to get the 4 bits it needs. So the V.14 checks that data is available in the buffer. If not, it sends a stop bit; if so, it sends a start bit and cuts the 8 bits of data to fit in the 4-bit blocks. When the 8 bits of data are sent, it fills with stop bits.

The V.14 has also an overspeed function that allows the transmission of an asynchronous rate that is a little faster than 2400 bps. To do this, it cancels the transmission of the stop bit at the end of a word from time to time. This overspeed function is controlled by variable V14NbStop which define the number of words to wait between the removals of stops.

## V.22 bis/V.22 Interface

The V.22 bis/V.22 software mode has two interfaces. One interface is towards the telephone line (interrupt), the other is towards the user.

### Interrupt Interface

The interface of the software module towards the line is made through the sample interrupt routine. This routine is Sigmalt in main.asm. The sample interrupt routine reacts at 8 kHz to receive and send samples to and from the AD/DA converter. How these samples are managed depends on the status of the modem. The status of the modem is defined by the variable \_CID. When the \_CID variable is set to 0, the modem status is V.22 bis/V.22 mode. The program goes in file V22it.asm, which contains all of the management of samples and tasks for V.22 bis/V.22.

V22it.asm transfers samples between the AD/DA converter and the buffer for modulation and demodulation. If there is not an adequate sample for a symbol (1/600 second) the interruption is simply terminated. If there is an adequate sample, modulation or demodulation is activated as a task and generates (modulation) or demodulates (demodulation) the current symbol.

### User Interface

An example of the user interface is implemented in the AT command set routine in mainc.c.

The first element of the interface is function V22b(), which starts the connection of the modem after the dialing or off-hook setting. This function uses global variable Modul to define the modulation speed and direction to use and sends back the result of the connection (speed or error).

Once the connection is made, the user interface is made through functions GetData(), V14Send(), V14Stat() and Carrier(). GetData() returns demodulated data from a buffer if available. V14Send() puts data to be sent into a buffer when possible, V14Stat() shows whether there is room in the transmit buffer. Carrier() checks the carrier detect function. The data used by GetData() and V14Send() are 8-bits wide without start and stop bits. GetData() has suppressed start and stop bit. V14Send() will add start and stop bits.



## Interface Functions and Variables

### Carrier(void)

Carrier() is a C callable function. It returns 0 if no carrier, a non-zero value if carrier is detected.

### CID

CID is a variable specifying which mode the processor is in. The available modes are defined in file comon.inc :

- ModemMode = 0: in this mode, the processor operates in V.22/V.22 bis mode. (The value is 0 so that it is simpler to test (less MIPS). CID is set in ModemMode by calling the routine modemini() (in V22b.c).)
- NoMode: no signal processing
- CIDMode: caller ID mode
- CPTMode: call progress mode
- FSKV23: V.23 mode
- FSKV21: V.21 mode

#### CAUTION:

This variable must be handled with care because it is used by the interrupt and could hang the software if used incorrectly. The best way to use it is through function modemini() to initiate V.22/V.22 bis mode and function Fdisconnect() to return to waiting mode.

### Fdisconnect(void)

Fdisconnect is a C callable routine defined in main.asm. It disconnects the line and sets CID to CIDMode with initialization of that mode.

### GetData(void)

GetData() is a C callable function that needs no parameter. It checks and returns data from demodulation. The returned data is put in the high-order bits 8 to 15. In the lower bits are two flags, FLAG20 and FLAG10. FLAG20 is set to 1 if a data is available. FLAG10 is set to 1 if an overflow has occurred. That means that data was demodulated and lost because the user has not taken them fast enough. Calling GetData() clears FLAG10 so that FLAG10 is read only once when overflow occurs.

The data words are 8-bits wide with start and stop bits removed according to V.14 standard. The buffer size (FIRST\_OUT) is set to 16 words of 8 bits.



## **V14Send(Data)**

V14Send() is a C callable routine, it allows the sending of data to the modulator. It returns a negative value if the transmit buffer is full and does not accept the data. The data is an 8-bit word put in the lower position of the word. The V14Send() function will add to Data start and stop bits as defined by V.14 standard. The function V14Stat() can be used to see whether or not the buffer is full.

## **V14Stat(void)**

V14Stat() is a C callable routine. It returns 0 if the transmit buffer is not full, a non-zero value if the transmit buffer is full.



## TI Contact Numbers

---

### INTERNET

*TI Semiconductor Home Page*

[www.ti.com/sc](http://www.ti.com/sc)

*TI Distributors*

[www.ti.com/sc/docs/distmenu.htm](http://www.ti.com/sc/docs/distmenu.htm)

### PRODUCT INFORMATION CENTERS

#### *Americas*

Phone +1(972) 644-5580

Fax +1(972) 480-7800

Email [sc-infomaster@ti.com](mailto:sc-infomaster@ti.com)

#### *Europe, Middle East, and Africa*

Phone

Deutsch +49-(0) 8161 80 3311

English +44-(0) 1604 66 3399

Español +34-(0) 90 23 54 0 28

Français +33-(0) 1-30 70 11 64

Italiano +33-(0) 1-30 70 11 67

Fax +44-(0) 1604 66 33 34

Email [epic@ti.com](mailto:epic@ti.com)

#### *Japan*

Phone

International +81-3-3457-0972

Domestic 0120-81-0026

Fax

International +81-3-3457-1259

Domestic 0120-81-0036

Email [pic-japan@ti.com](mailto:pic-japan@ti.com)

#### *Asia*

Phone

International +886-2-23786800

Domestic

Australia 1-800-881-011

TI Number -800-800-1450

China 10810

TI Number -800-800-1450

Hong Kong 800-96-1111

TI Number -800-800-1450

India 000-117

TI Number -800-800-1450

Indonesia 001-801-10

TI Number -800-800-1450

Korea 080-551-2804

Malaysia 1-800-800-011

TI Number -800-800-1450

New Zealand 000-911

TI Number -800-800-1450

Philippines 105-11

TI Number -800-800-1450

Singapore 800-0111-111

TI Number -800-800-1450

Taiwan 080-006800

Thailand 0019-991-1111

TI Number -800-800-1450

Fax 886-2-2378-6808

Email [tiasia@ti.com](mailto:tiasia@ti.com)

TI is a trademark of Texas Instruments Incorporated.

Other brands and names are the property of their respective owners.



## IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

CERTAIN APPLICATIONS USING SEMICONDUCTOR PRODUCTS MAY INVOLVE POTENTIAL RISKS OF DEATH, PERSONAL INJURY, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE ("CRITICAL APPLICATIONS"). TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF TI PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE FULLY AT THE CUSTOMER'S RISK.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty, or endorsement thereof.

Copyright © 1999 Texas Instruments Incorporated