Direct Sequence Spread Spectrum (DSSS) Modem Reference Design

September 2001, ver. 1.0 Functional Specification 14

Introduction

Much of the signal processing performed in modern wireless communications systems—such as digital modulator/demodulator applications—takes place in the digital domain and requires high throughput. Dedicated hardware can provide the processing capability to meet this requirement. The parallel processing capability of Altera® programmable logic devices (PLDs) makes them ideal for baseband/intermediate frequency (IF) digital signal processing applications.

The direct sequence spread spectrum (DSSS) digital modem reference design is a hardware design that has been optimized for the Altera APEX™ DSP development board (starter version), which features an APEX EP20K200E device. The reference design is a spread spectrum modulator/demodulator subsystem that can be used as a starting point for a complete 3G or fixed wireless modem. The design highlights the ease of use, performance, and efficiency of implementing a design using Altera devices and DSP intellectual property (IP) cores.

The design uses the correlator, FIR compiler, and NCO compiler MegaCore® functions, and uses 7,183 logic cells and achieves 100-MHz performance. A parameterizable UART and a Windows application enables communication between the APEX DSP development board and a PC via the board's RS-232 interface, yielding a true development platform for intermediate frequency (IF) modem designs. The reference design includes VHDL source code (except for the IP cores), and a simulation library and test bench for simulation in ModelSim simulators.

Allocating channel resources using spread spectrum techniques are a favorable alternative to allocating channels using frequency division multiple access (FDMA) or time division multiple access (TDMA) schemes. By spreading the spectrum of the data to be transmitted with orthogonal codes, the receiver can decode each user's data uniquely, even though many users share the same spectral and temporal channel resources.

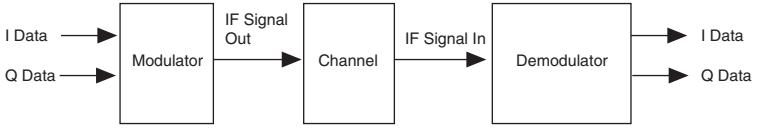
The two primary methods used to spread the baseband data spectrum are:

- *Frequency hopping spread spectrum (FHSS)*—The frequency of the modulating carrier varies in a pseudo-random manner unique to each user.
- *Direct sequence spread spectrum (DSSS)*—The data is spread by multiplication with a channelization code prior to up-conversion to an intermediate frequency.

Each method has its advantages and disadvantages. For example, in a DSSS systems, the designer can reduce narrowband interference in the same channel by increasing the processing gain, but this technique cannot be used in FHSS schemes. However, FHSS schemes are less susceptible to jamming and the receiver is easier to implement, usually requiring only a simple analog limiter/discriminator.

This functional specification discusses the DSSS digital modem reference design. The design modulates direct sequence spread data onto an IF carrier. The modulated data is then input to a channel model and is passed to a digital receiver, which demodulates and recovers the data from the received IF signal. See Figure 1 on page 2.

Figure 1. DSSS Modem Block Diagram



The design is implemented using Altera DSP MegaCore® functions, functions from the library of parameterized modules (LPM), and custom logic. The designer can download the application into the Altera APEX™ EP20K200E or EP20K1500E device on the APEX DSP development board (starter or professional version, respectively).

DSSS Overview

Each user in a DSSS system has a channelization code that is orthogonal to the other codes in the system. The input data symbol d_i is spread by an N-element codeword c_i to yield the spread data s_i as shown in the following equation:

$$s_i = c_i \times d_i$$

The resulting spread sequence rate, referred to as the chip rate, is increased to N times the original data rate. The code length parameter N is called the spreading factor. The receiver despreads the demodulated sequence by calculating the inner product of the received spread signal with a local copy of the channelization code that was used to spread it. The following discussion demonstrates the importance of using orthogonal codes.

The received demodulated spread sequence is $\underline{r}_i$ where

$$\underline{r}_i = \underline{s}_i + \underline{n}$$

and $\underline{n}$ is noise channel induced noise.

The received signal is despread to yield $\tilde{d}_i$ by the following calculations:

$$\begin{aligned}
\tilde{d}_i &= \langle \underline{r}_i, \underline{c}_i \rangle \\
&= \langle \underline{c}_i d_i + \underline{n}, \underline{c}_j \rangle \\
&= d_i \langle \underline{c}_i, \underline{c}_j \rangle + \langle \underline{n}, \underline{c}_j \rangle \\
&= \left(d_i \sum_{k=0}^{N-1} c_i[k] c_j[k-m] \right) + \tilde{n} \\
&= \begin{cases} k_1 d_i + \tilde{n} & i = j,\ m = 0 \\ \pm k_2 d_i + \tilde{n} & i = j,\ m \neq 0 \\ \tilde{n} & i \neq j \end{cases}
\end{aligned}$$

where:

- m is a relative offset between the received down-converted sequence and the local copy of the despreading code
- k_1 and k_2 are constants dependent on the code sequences concerned and the offset m

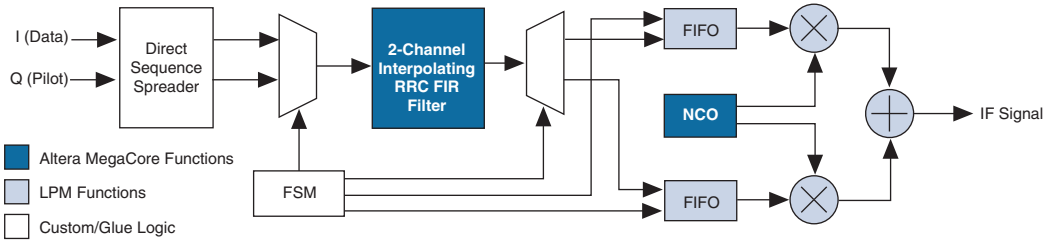
If i = j (the spreading code and the despreading code are identical) the receiver can recover the data from the demodulated spread sequence. The index of the local copy of the channelization codes must be aligned with the received spread sequence for the receiver to despread the data correctly. A temporal offset can lead to an incorrectly despread sequence. In reality, the receiver chooses the lag m such that the received signal and despreading codes are aligned.

The sign of $k_1 d_i$ uniquely indicates the original binary value of d_i.

DSSS Modulator

Figure 2 shows a block diagram of the DSSS modulator. The modulator consists of a complex sequence spreader, a Nyquist pulse-shaping FIR filter, a digital oscillator, and a mixer.

Figure 2. DSSS Modulator Block Diagram



Two serial binary data channels form the inputs to the modulator. The I channel carries the data to be transmitted, i[n]. The Q channel carries a known repeating pilot sequence q[n], which the receiver uses to perform I-Q derotation and symbol recovery. (Generally, the Q channel is a control channel; it can also carry power control commands and link monitoring information to aid in hand-off procedures.)

Direct Sequence Spreader

The input sequences are combined to form the complex sequence i + jq and are spread by a complex spreader. The complex spreader has a spreading factor of 16 and codewords c_i and c_q. The resulting spread sequence quadrature components $\{s_i,s_q\}$ have a ternary symbol alphabet {-2,02} and are formed as shown in the following equations:

$$s[n] = s_i[n] + js_q[n]$$

$$= (i[n] + jq[n]) \times (c_i[mod(n,16)] + jc_q[mod(n,16)])$$

$$s_i[n] = i[n] \times c_i[mod(n,16)] - q[n] \times c_q[mod(n,16)]$$

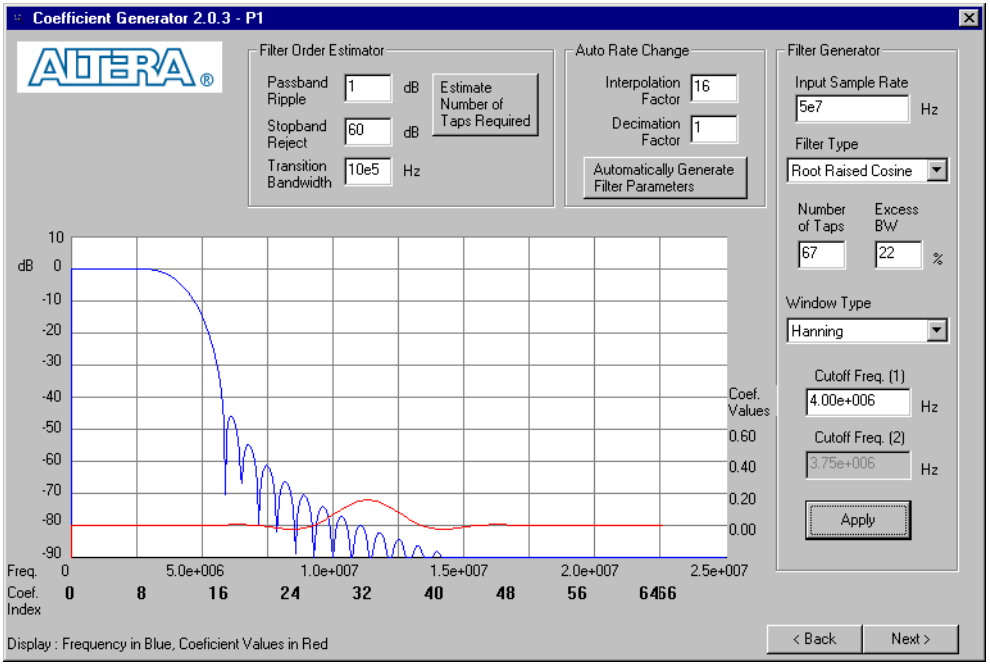
$$s_q[n] = i[n] \times c_q[mod(n,16)] + q[n] \times c_i[mod(n,16)]$$

The sequences $s_i[n]$ and $s_q[n]$ have been spread to the chip rate—which is N times the input data rate, where N = 16. (This discussion does not consider scrambling the sequences using long or short scrambling codes, and this technique was not implemented in the DSSS modem reference design).

FIR Filter

Next, the design filters the spread sequences using a pulse-shaping interpolating FIR filter. The FIR filter was created using the Altera FIR compiler MegaCore function. The filter is a 2-channel, 67-tap root-raised-cosine filter with an excess bandwidth of 22%. The design uses a serial structure, which uses PLD resources efficiently. The serial architecture trades off throughput (clock cycles) for resources (PLD logic cells and embedded system blocks). Figure 3 shows the filter frequency response.

Figure 3. FIR Filter Parameters & Frequency Response



The filter takes the chip-rate spread data streams as input. First, the I and Q channel sequences are multiplexed into a single stream. The filter interpolates the data by a factor of 16 and is clocked at 100 MHz. The filtered data is output at the interpolated rate with 16 successive samples for each channel. The FIR compiler automatically generates a polyphase decomposed interpolation filter. This type of filter is usually smaller and runs faster than the conventional zero-stuffed FIR filters.

The filter output data is demultiplexed into two FIFO buffers to yield two baseband filtered pulse streams at 50 MSPS. A simple state machine implements the necessary control and multiplexing/de-multiplexing functions. An additional block delays the I-channel output data after the demultiplexing and buffering stages to align the I- and Q-channel data.

Figure 4 shows a plot of the filter input spread sequence and the corresponding pulse-shaped data output.

Figure 4. Input Filter Spread Sequence Plot

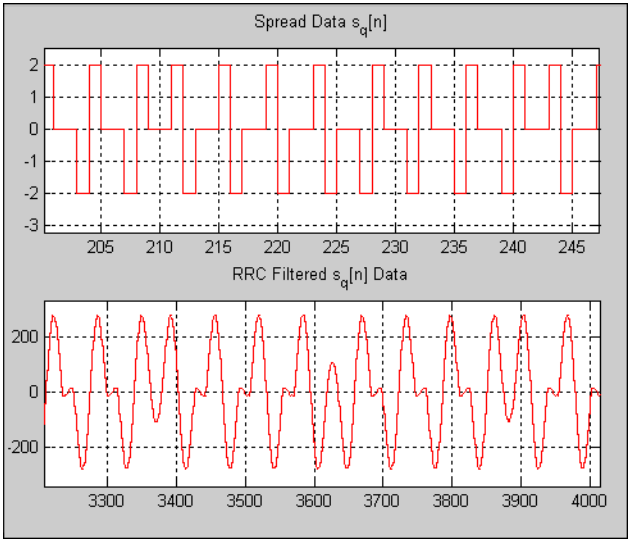


Table 1 shows the resource usage of the FIR filters.

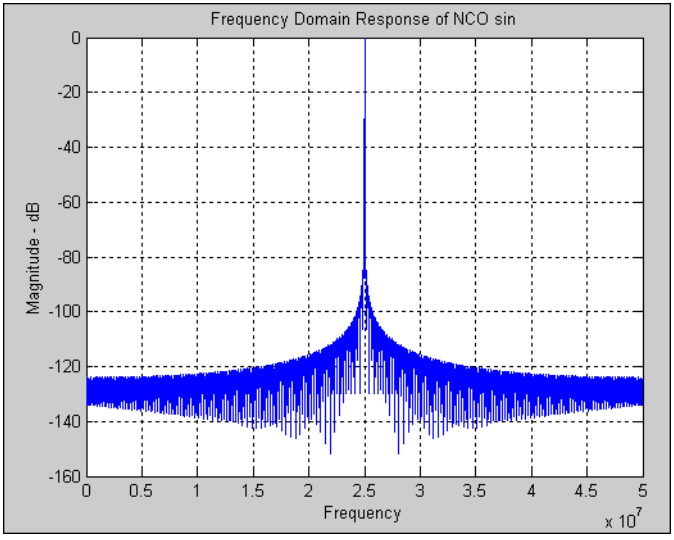
Table 1. FIR Filter Resource Usage

Design	Logic Elements	Embedded System Blocks
2-Channel Interpolating FIR Filter	1,365	1

Numerically Controlled Oscillator

After filtering, the sequences are modulated onto quadrature carrier oscillators. The design's numerically controlled oscillator (NCO) was created using the Altera NCO compiler MegaCore function. The NCO is a ROM-based implementation because of the high data rates required by the design. The NCO compiler small ROM option generates both sine and cosine waveforms while storing only 45 degrees of the waveform. Therefore, the NCO provides a high-quality oscillator using less APEX 20KE device embedded system blocks. The NCO compiler automatically generates a MATLAB simulation, which shows the spurious free dynamic range (SFDR) of the carrier waveform. Figure 5 on page 7 shows the frequency domain response of the NCO sine wave.

Figure 5. NCO Frequency Domain Response



A 24-bit phase accumulator generates the 12-bit NCO output precision, yielding an SFDR of greater than 75 dB using few PLD logic resources. The design requires a 25-MHz IF, which implies that there are four samples of the sinusoid per clock cycle; each clock cycle advances the carrier phase 90 degrees. The multipliers perform the mixing operation modulating the quadrature carriers with the full 13-bit precision outputs of the FIR filter. The resulting signals are truncated to 10-bit resolution before being added to form the IF signal.

Table 2 shows the resource usage of the NCO.

Table 2. NCO Resource Usage

Design	Logic Elements	Embedded System Blocks
NCO	178	4

Modulator Design Files

The modulator uses 2,490 logic cells, which is 29% of available logic cell resources in an APEX EP20K200E device, and 6 embedded system blocks. Table 3 shows the modulator design files, which are located in the *<path>***dsp_development_kit_v1.0.0****reference_design****dsss****quartus****source** directory.

Table 3. Modulator Design Files

Module	Function
modulator.vhd	Top-level modulator entity. Instantiates the modulator sub-entities.
complex_spreader.vhd	Direct sequence complex spreader. Spreads the input data and pilot sequences by a complex code.
rrc_fir_16.tdf	2-channel interpolating RRC FIR filter. Implements the pulse-shaping filter.
fsm_mod.vhd	State machine.Controls the FIR channel multiplexing and demultiplexing.
ififo.vhd	FIFO. Buffers the dechannelized, interpolated filter output data.
delay_i_channel.vhd	Delay register. Realigns the dechannelized I and Q sequences.
nco_mod.vhd	Numerically controlled oscillator. Generates the quadrature IF carrier.
mult.vhd	Multiplier. Performs mixing.
iqadd.vhd	Adder. Adds the I and Q channel data to form the IF signal.

Channel Model

The modem's channel model exercises the receiver's symbol recovery circuitry and the despreading code synchronization. In spread spectrum communications, synchronization is arguably the most important factor when correctly recovering the transmitted data.

The channel model lets the user introduce random data by pressing push-button SW2 on the APEX DSP development board. While the user presses the push-button, the random data causes the receiver to lose both symbol and PN synchronization. When the user releases the push-button, the modulated IF signal is fed into the receiver again, which must reacquire I-Q derotation and PN synchronization. Table 4 shows the channel model design file, which is located in the *<path>*\dsp_development_kit_v1.0.0\reference_design\dsss\quartus\source directory.

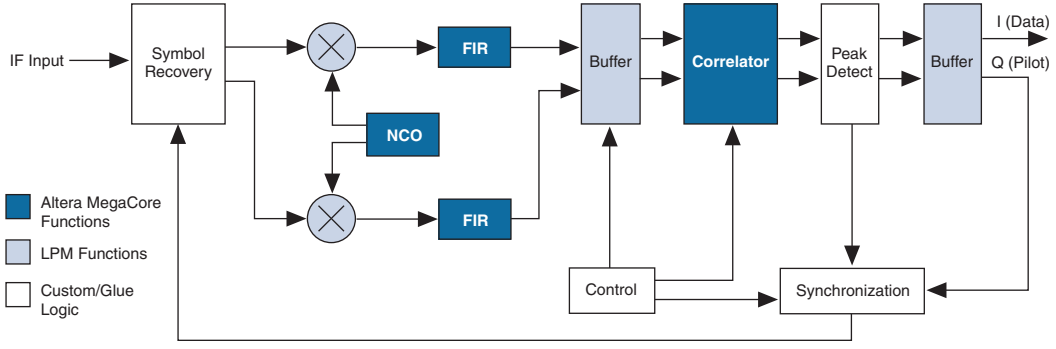
Table 4. Channel Model Design File

Module	Function
channel.vhd	Channel model. Selects between the IF signal and random data to the receiver given user input on the APEX DSP development board.

Demodulator

The demodulator down-converts the signal from IF to baseband and recovers the original data by despreading the demodulated sequences. Figure 6 shows the demodulator block diagram.

Figure 6. Demodulator Block Diagram



NCO

The receiver converts the 10-bit 25-MHz IF signal by mixing the incoming signal with the quadrature carrier output of a dual-output NCO using multipliers (refer to Figure 6 above). The NCO was created with the NCO compiler, using a small ROM architecture and the same parameters as the modulating NCO. Table 5 shows the resource usage of the NCO.

Table 5. NCO Resource Usage

Design	Logic Elements	Embedded System Blocks
NCO	178	4

FIR Filters

Two root-raised-cosine FIR filters remove the higher frequency components created by the signal mixing. These 15-tap root-raised cosine filters have an excess bandwidth of 22% and were designed using the FIR compiler. The filters use a fully parallel architecture with extended pipelining. Extended pipeline architectures allow the user to achieve the desired performance quickly with automatic place-and route (in this case over 100 MSPS).

Table 6 shows the resource usage of each FIR filter.

Table 6. FIR Filter Resource Usage

Design	Logic Elements	Embedded System Blocks
Parallel FIR FIlter	904	0

The combined filtering effect of the modulator and demodulator pulse-shaping results in an overall raised-cosine filter response. This implementation reduces ISI, while allowing sufficient bandwidth to permit very small timing offsets in the sampling instances.

Despreader

Next, the receiver despreads the demodulated sequences. The heart of the despreading and synchronization circuitry is the Altera correlator MegaCore function. The core provides aids in the alignment—to the granularity of 1/4 chip interval—of the local copy of the c_i and c_q codes with the incoming demodulated spread signal, and simultaneously despreads the sequence.

The correlator calculates the complex correlation of the incoming signal, $\underline{r} = r_i + jr_q$ with the conjugate complex code $\underline{c} = c_i - jc_q$ over N lags m, to yield the complex sequence $a_i[m] + ja_q[m]$ by the following equation:

$$a[m] = \sum_{k=0}^{N-1} \underline{r}[k-m] \times \underline{c}[k]$$

$$a_i[m] + ja_q[m] = \sum_{k=0}^{N-1} (r_i[k-m] + jr_q[k-m])(c_i[k] - jc_q[k])$$

Selecting the lag m—for which this correlation is greatest—and assuming that the design compensates for all other phase offsets, the correlation maxima uniquely specify the original unspread binary data.

Table 7 shows the resource usage of the correlator.

Table 7. Correlator Resource Usage

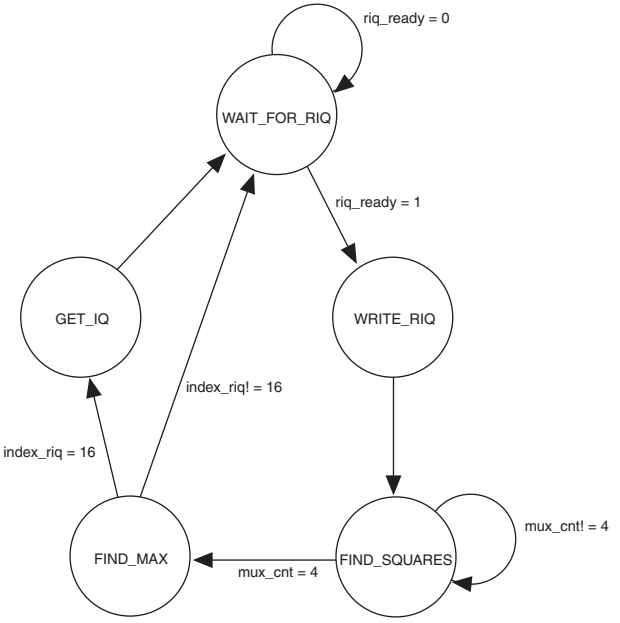
Design	Logic Elements	Embedded System Blocks
Correlator	861	2

The filter output data for each channel (I and Q) is saturated and truncated to 4 bits. Then, the two channels are concatenated to form an 8-bit word. The concatenated output is passed to the correlator interface and buffered. A four-state Mealy state machine feeds the buffered data to the correlator's internal buffer where it is decimated. For increased processing gain, the correlator has an oversampling factor of four (specified in the wizard). The correlator computes the 16 complex lag values, $a_i[m]$ and $a_q[m]$, on the I and Q channels, eruptively, for each of the 8 sections of a 128-length block.

A second three-state Mealy-type state machine controls which channel lag data is computed and switches between codes c_i and c_q accordingly. While the correlator processes data for the next 16-chip section of the 128-length block, this state-machine goes into an idle state (WAIT_FOR_RESULTS), signaling to a peak-detection circuit that a new set of lag values are ready and that location of the correlation maximum for the current section should begin.

The peak detection circuit operation is controlled by a 5-state state-machine (see Figure 7). The peak detection state transitions from WAIT_FOR_RIQ to WRITE_RIQ when the correlator indicates via the `riq_ready` signal that new lag data is available.

Figure 7. Peak Detection Circuit Operation



For each lag index m, a pair of samples $a_i[m]$ and $a_q[m]$ is written into an array. The complex magnitude of the sample pair, $R_{iq}[m]$ is calculated as shown in the following equation:

$$R_{iq}[m] = a_i[m]^2 + a_q[m]^2 \qquad m = 0 \ldots 15$$

The squaring function is implemented by a look-up-table to conserve logic cell resources. The table is shared between the I and Q lag data, and requires four clock cycles to compute $R_{iq}[m]$. A counter, `mux_cnt`, indicates when the operation is complete and the state transitions from FIND_SQUARES to FIND_MAX. $R_{iq}[m]$ is compared to the previously stored maximum. If it is greater, its lag index and magnitude are stored, and the next available lag values, $a_i[m]$ and $a_q[m]$, are stored to an incremented array position. This process is repeated for each of the 16 lags, after which the state transitions to GET_IQ. Once the index of the correlation maximum, m_{max} has been located, the sign of the values $a_i[m_{max}]$ and $a_q[m_{max}]$ in the stored array indicate the despread sequences i[n] and q[n]. The despread pilot sequence q[n] is passed to the synchronization monitor while the despread data sequence *i*[*n*] is output to the UART interface.

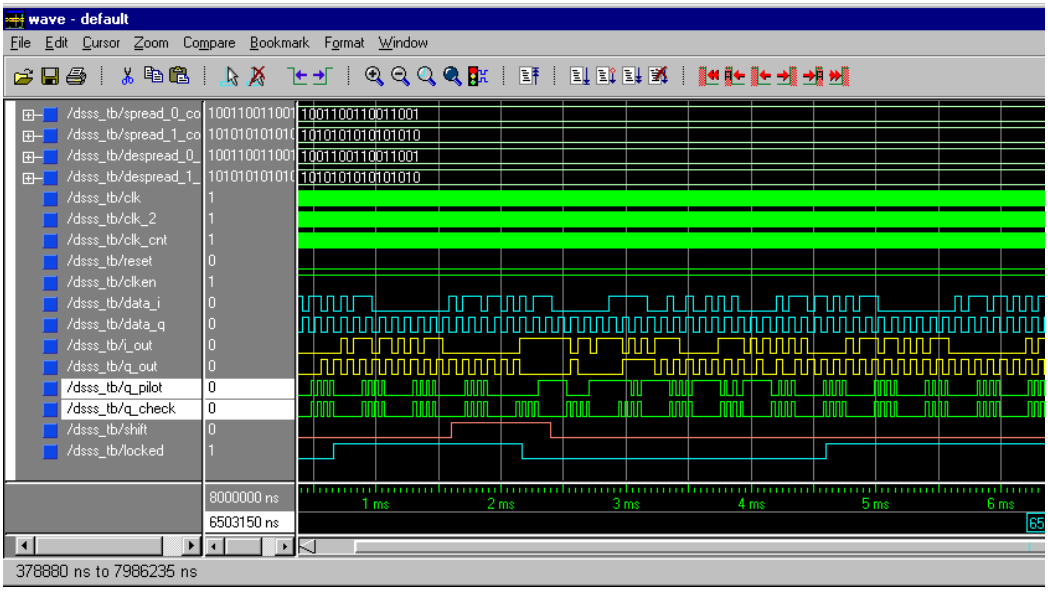
The peak detection logic increments a block counter to keep track of how many symbols of the block of 8 symbols (128 chips) have been despread. Then it returns to the idle state, WAIT_FOR_RIQ, until the correlator indicates that it has the next lag data available. After the full block is processed, the peak detection circuit sends a processing complete indication to the correlator interface. The correlator then begins processing the data read in from the input-side buffer for the next block of 128 chips.

The system clocks the despreading portion of the demodulator at half the rate of the down-conversion circuitry. This rate is twice what is needed to compute the required 128 complex lags for each block because the correlator decimates the FIR filter output rate by four. This redundancy allows the peak detection circuit to work on each set of 16 complex lag values before the next set of lag values for the next section is available. After the correlator reads the last data for the 128-chip block from its input buffer, it initiates a reads of the next block. Therefore, the block correlator operates in a continuous manner. The APEX 20KE device's on-chip PLL generates the required 100- and 50-MHz clocks from the 40-MHz on-board crystal clock.

The sign of the $a_i[m_{max}]$ and $a_q[m_{max}]$ values always yield the original data sequence, i[n] and the pilot sequence q[n] if the phase rotation induced by the system latency and channel is an integer multiple of 2π radians. However, this situation may not occur all of the time. Any rotations of the signal before demodulation can cause the I-Q channels to interchange, be inverted, or a combination of these effects. Because the pilot sequence is known prior to transmission, the system can compensate for rotations or offsets that do occur using a symbol recovery circuit.

The synchronization block shown in Figure 6 on page 9 monitors the resulting despread pilot sequence for errors. After the number of errors in the 8-bit repeating pilot sequence reaches a pre-defined threshold, the system can assume that the phase alignment of the incoming signal is offset. The system can then shift the position of the sampling point of the incoming signal relative to the rest of the demodulator in time until errors are not observed in the pilot sequence. A locked condition is assumed and the input sampling instance remains fixed. Upon detecting pilot errors, the system re-initiates this process to reacquire synchronization. The waveform in Figure 8 illustrates the operation of the pilot monitor and the symbol recovery blocks.

Figure 8. Pilot Monitor & Symbol Recover Blocks



Initially, the receiver is synchronized to the incoming data and the received despread pilot signal `q_pilot` matches the locally generated pilot sequence `q_check`. Hence `locked` remains high and the output despread sequences `i_out` and `q_out` match the corresponding input sequences `data_i` and `data_q`. After 1.6 ms the input signal `shift` is asserted, introducing random data into the demodulator. The receiver is seen to lose synchronization shortly thereafter. While the signals `q_pilot` and `q_check` continue to show discrepancies, the sampling instance of the demodulator input signal is shifted as described above

until synchronization is reacquired. After the pilot monitor detects no errors for a defined duration the monitor assumes that the receiver has regained synchronization and reasserts the `locked` signal.

Demodulator Design Files

The demodulator design uses 4,353 logic cells (which is 53% of the available logic cell resources in an APEX EP20K200E device) and 13 embedded system blocks. Table 8 gives a description of the demodulator design files.

Table 8. Demodulator Design Files

Module	Function
demodulator.vhd	Top-level demodulator entity. Contains a netlist of all the demodulator sub-entities.
Mult_d.vhd	Multiplier. Performs down-conversion multiplication.
Nco_demod.vhd	Altera NCO Compiler MegaCore function. Generates the quadrature IF carrier.
rrc_fir_par.v	Altera FIR Compiler MegaCore function. Implements the parallel demodulator pulse-shaping root-raised cosine filter.
Ram_in.vhd	Down-converter output buffer. Stores the data from down-converter to be used by the correlator.
Corr_interface_syn.vhd	Correlator interface. Implements the data control interface to the correlator.
dsss_corr.vhd	Altera correlator MegaCore function. Correlates the incoming sequences with the locally stored channelization codes.
peak_detect.vhd	Peak detection circuit. Locates the lag index of the correlation maxima.
riq_rom.vhd	Internal ROM. Performs the squaring function used to calculate the polar magnitude of the correlator output lag data.
Ram_out.vhd	Peak detection output buffer. Holds the buffer despread I and Q channel data.
sync_monitor.vhd	Pilot sequence monitor. Monitors the pilot channel output for errors.
Sym_recovery.vhd	Symbol recovery. Shifts the sample instance of the demodulator input signal.

Simulation in ModelSim

Altera provides a simulation library and testbench to exercise the design in the ModelSim simulators version 5.5b or later. The testbench design is named **dsss_tb.vhd** and is located in the *<path to DSSS reference design>***\examples\dsss\simulation** directory. Altera also provides a macro file, **run_dsss_tb.do**. To run the design, perform the following steps at the ModelSim command prompt:

1. Change to the DSSS simulation directory:

   `cd` *<path to DSSS reference design>*`/examples/dsss/simulation`

2. Choose **Execute Macro** (Macro menu).

3. Select the file **run_dsss_tb.do** and click **Open**. The macro runs the testbench and outputs the results to the ModelSim waveform viewer.

UART Interface

Altera also provides a UART and an interface between it and the DSSS modem with the reference design to allow communication between the APEX DSP development board and a PC. The designer can send input data from a PC's serial port to the UART implemented in the APEX DSP development board via an RS-232 cable using an Altera-provided Windows application. The application provides a graphical plot of the input serial data and the resulting output. The designer can view the effects of synchronization loss by pressing the switch SW3 on the board. Figure 9 shows a block diagram of the UART and interface to the DSSS modem.

Figure 9. UART/ DSSS Modem Interface

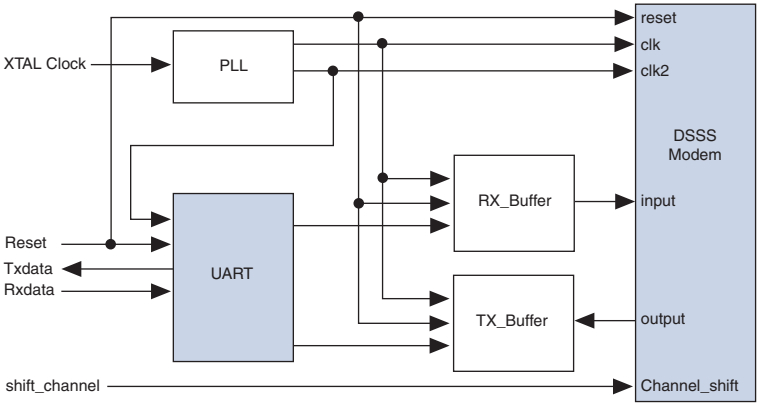
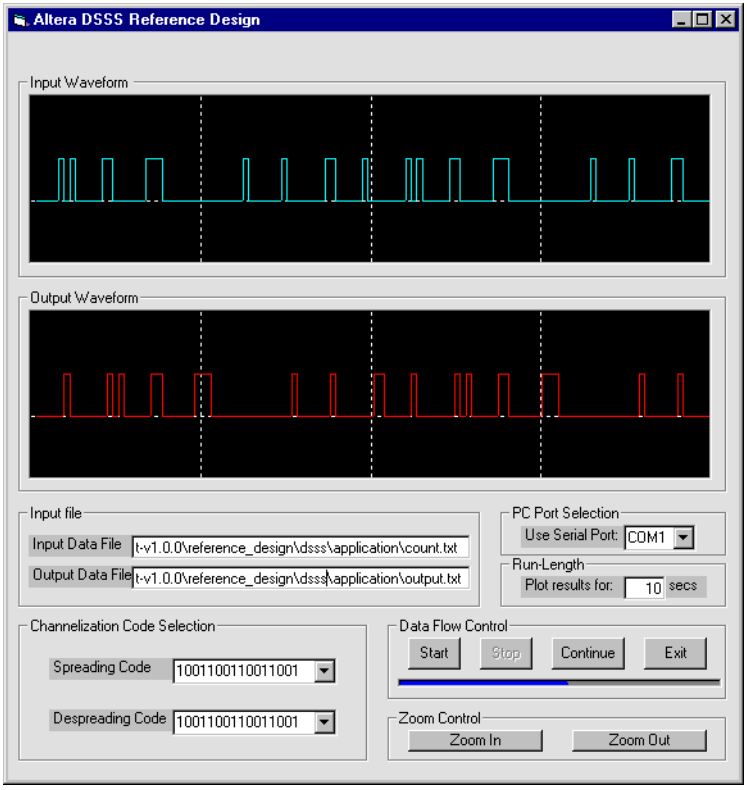


Figure 10 shows a screen shot of the Windows application.

Figure 10. Windows Application



Refer to the *DSSS Modem Lab White Paper* for instructions on how to use the Windows application with the APEX DSP development board and a PC.

Table 9 gives a description of the UART/DSSS modem interface design files

Table 9. UART/DSSS Modem Interface Design Files

Module	Function
top_level_dsss.vhd	Top-level file. Instantiates the phase-locked loop, DSSS modem, and the UART.
dsss.vhd	Modem netlist. Instantiates the modulator, channel, and demodulator entities.
altera_uart.vhd	UART. Allows communication between the APEX DSP development board and the PC.
receiver.vhd	UART receiver. Receives data from the PC and outputs it to the on-chip receive buffer.
transmitter.vhd	UART transmitter. Transmits serial data from the on-chip transmit buffer to the PC.
div_clk.vhd	Clock divider. Generates the internal UART baud clock.
clkpll.vhd	Altera ClockLock megafunction. Generates the 100-MHz and 50-MHZ clocks from the on-board 40-MHz crystal oscillator.
Uart_rec_buf.vhd	On-chip receive buffer. Bridges between the UART and the input to the DSSS modem.
Uart_tx_buf.vhd	On-chip transmit buffer. Bridges between the DSSS modem output and the UART.
uart_interface.vhd	Provides the UART/modem data flow control.

Table 10 gives the resource usage for the complete design.

Table 10. UART Interface & DSSS Modem Design Resource Usage

Logic Cells	ESBs	% Utilization	Performance
7,183	22	86	100 MHz

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
http://www.altera.com
Applications Hotline:
(800) 800-EPLD
Literature Services:
lit_req@altera.com