Reconfigurable Modem

For

Ocean Observatories

By
Ethem Mutlu Sözer
MIT Sea Grant College Program

Table of Contents

1 Introduction

Acoustic communications is an important part of underwater research. The mass amount of data collected by sub-sea devices can be made available to the scientific community in real-time with the utilization of acoustic modems.

Sensor data collected by the sub-sea devices, including but not limited to pictures, depth, currents, sonar images, traditionally are stored in the observation post until the end of the mission. By employing a high speed acoustic link between the observation post and a gateway, which is connected to the command and control station through a radio link, we can reach the data in real-time.

Currently, off-the-shelf modems are employed for these applications. These modems are limited in their capabilities and are not flexible enough to experiment with different communication schemes. Especially in the case of underwater acoustic networks, it is almost impossible to utilize off-the-shelf modems.

We are developing an acoustic modem that will be flexible enough to test different communication algorithms including networking protocols. Due to its highly flexible structure, we will call this modem the Reconfigurable Modem. The main purpose of the Reconfigurable Modem will be to bring simulation and development environments together. By this way, algorithms developed by researchers and tested using simulation can be rapidly prototyped and proved in real world scenarios. Algorithm that are found to be effective can be implemented in hardware with more stringent requirements, such as low power consumption for extended operation time and low production cost.

The development of the modem is carried out using Mathworks tools, such as Matlab, Simulink, and Real-Time Workshop. Matlab has been the choice of the scientific community for developing new algorithms. We will create a common simulation environment using Matlab and Simulink. Once the algorithms are tested using the simulation environment, we will generate real-time code using Real-Time Workshop. The generated real-time code can be run on a digital signal processor (DSP). Once the algorithm is transferred into a DSP, we can test it in real world.

In the Simulink environment, algorithms are defined using functional blocks. We can exploit this property and design a highly modular acoustic modem. A researcher can only focus on one of the functional block, say the equalizer, and develop a new algorithm. By simply changing the equalizer block in the reconfigurable modem, we can test this new algorithm and generate real-time code.

The reconfigurable modem hardware has three parts: the main board, power amplifier – preamplifier board, and the transducer(s).

2 Reconfigurable Modem Software

We developed the Reconfigurable Modem (rModem) software using the Simulink platform by Mathworks. Simulink provides a simulation environment where the rModem can be modeled in a block diagram fashion. Each block defines a separate task of the rModem, such as filtering, synchronization, and equalization. The rModem functional block can be tested by running Simulink simulation. In addition to simulations, using the Real-Time Workshop tool, we can convert the Simulink block diagram into real-time C code. This generated code can be compiled and downloaded to our hardware using Code Composer Studio (Texas Instruments Development Environment).

Figure 1 shows the highest level block diagram of the rModem. We followed the ISO layered network definitions [1]. This version of the rModem software defines the Pyhsical Layer (or Layer 1), the Transport Layer (Layer 4), and the UART interface for serial communications with the modem. In the future versions, we will include the Network Layer (Layer 3) and the Data Link Control Layer (Layer 2). Each layer is connected to its higher level through two queues (or FIFO buffers), one for downstream communications and one for upstream communications. The rest of this section explains the individual blocks in more detail.

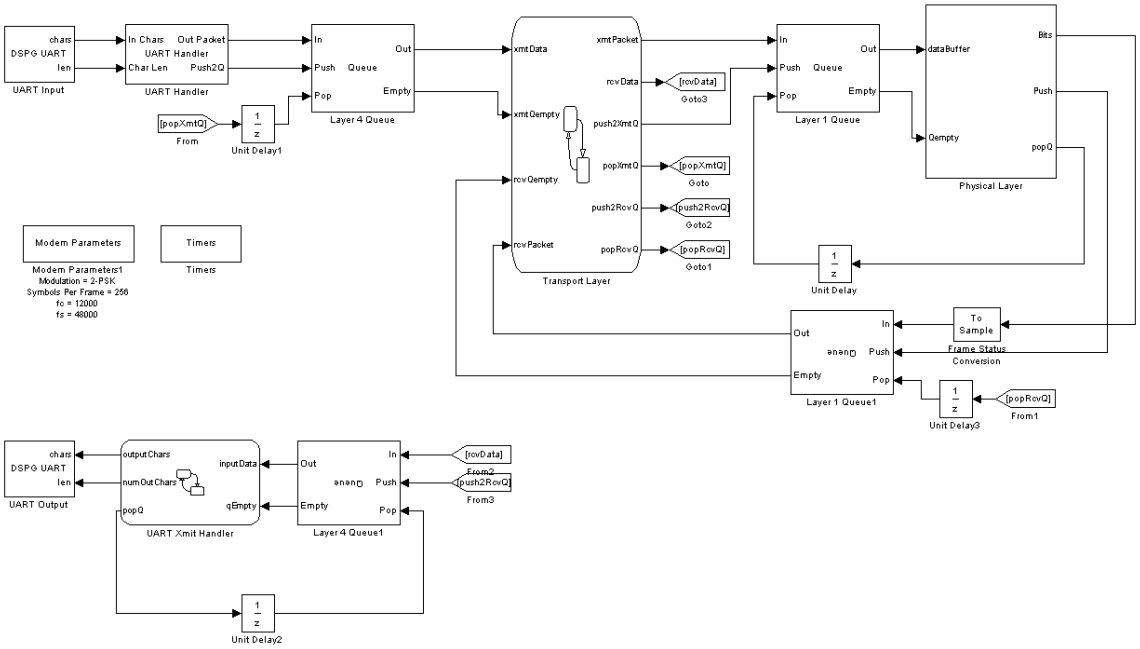

Figure 1 The highest level block diagram for rModem defined in Simulink.

Frame, packet, frequency, sampling rate, ….

2.1 Some Simulink Requirements

Simulink is originally intended to be used with dynamic control systems. It can resolve loops in a system as long as there is a delay block in the loop. Therefore, you will find

some delay blocks in our implementation. This becomes a problem especially in case of queues. In a conventional communication system, if a process pushes an element into a queue, the next process can pop that element out of the queue even though they executed in the same clock tick. However, in Simulink the next process first has to send a pop signal to the queue. And this signal can only be processes by the queue in the next clock tick, which causes an unnecessary delay in the system.

In the upcoming versions, we are planning to include the queues into the state machine definitions. In other words, each state machine will have a block of memory to implement an input queue. By this way, we can access the queue multiple times in one time instance.

2.2 Physical Layer

Physical layer converts data bits into acoustic signals before transmitting them through the underwater channel and converts received acoustic signals into data bits. The block diagram of the physical layer is given in Figure 2.

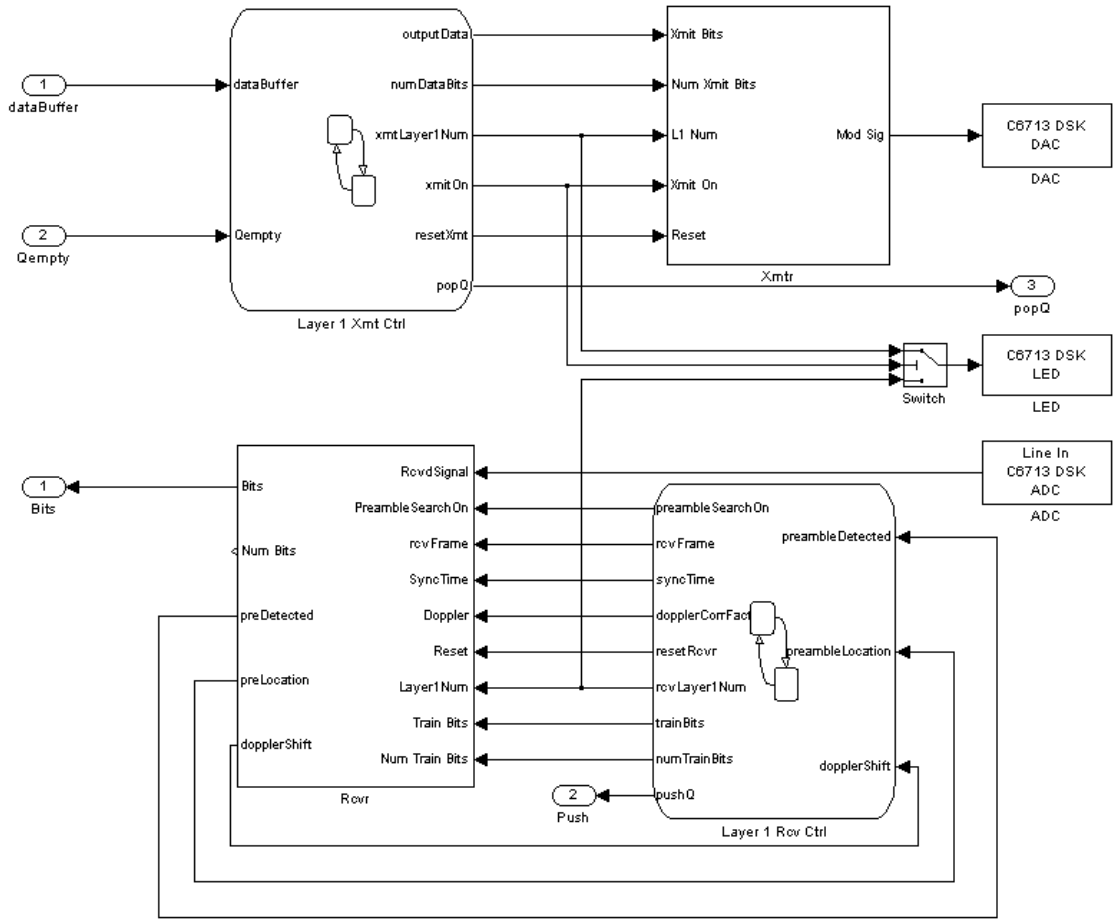

Figure 2 Physical layer (Layer 1) of rModem consists of the transmitter, the receiver, layer 1 controllers, and AD/DA converters.

2.2.1 Layer 1 Transmitter Controller

Physical layer transmitter controller (Layer 1 Xmit Ctrl) handles the creation of acoustic packets. The state diagram for the transmit controller is given in Figure 3. When the controller detects that the Layer 1 downstream queue is not empty, it generates a signal to pop the packet from the queue. Due to the implementation of the queue, the packet appears at the input port of the controller with one unit delay (or at the next clock tick).

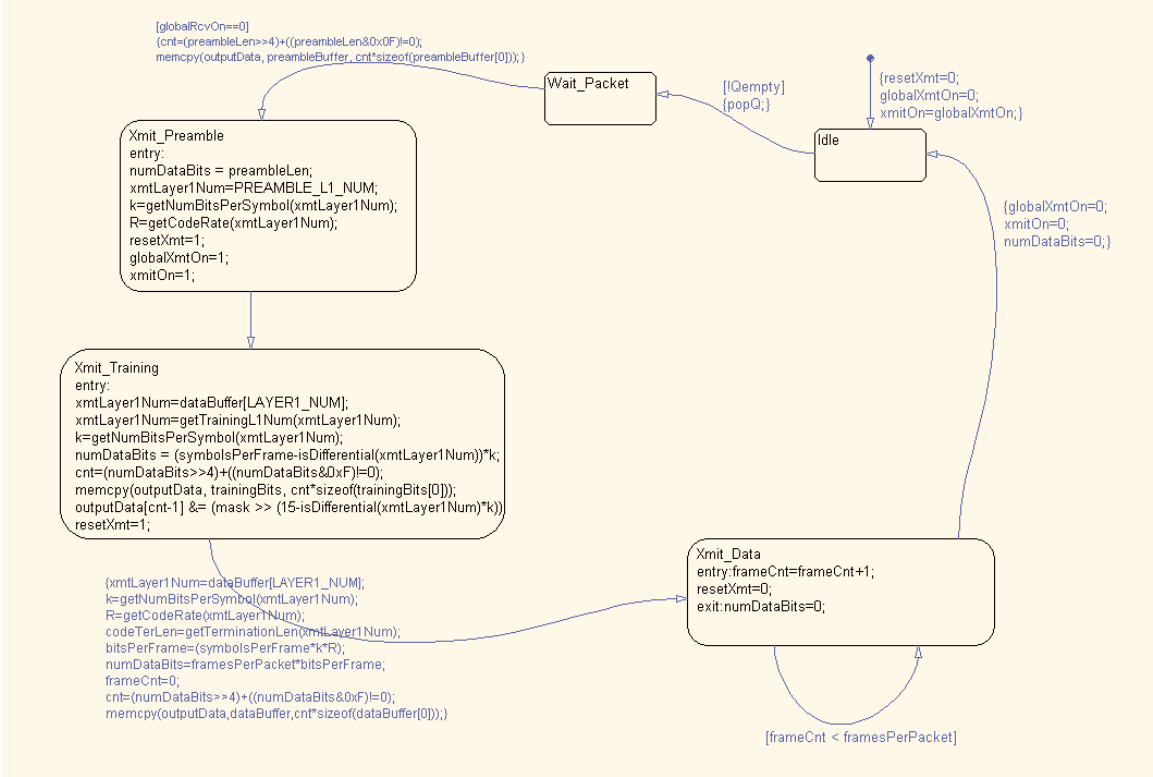

Figure 3 Layer 1 transmitter controller controls the sequence of events to create acoutic packets.

When the controller receives the packet from the queue, it checks if the receiver is on. If there is no reception going on, it turns on the transmitter and generates a preamble. The preamble is a pulse used to detect and synchronize to an acoustic packet at the receiver. The receiver also uses the preamble to estimate the Doppler shift presented by the channel. The preamble occupies one frame duration together with the silence period. We inserted a silence period to ensure that the multipath arrivals due to the preamble will die before the start of the data signals.

After the preamble, the controller generates a training sequence, whose length depends on the modulation constellation size. Training symbols occupy one frame duration. Then, the controller passes all the data bits to the transmitter and enters a loop to wait for all the bits to be sent out to the channel. The loop size is determined by the framesPerPacket

parameter. The controller expects enough number of bits from the upper layer to fill create one packet.

Upon completion of the transmission of the packet, the controller returns to the Idle state. Returning to the Idle state without checking for a new packet in the queue assures that there will always be one frame duration between consecutive packets.

2.2.2 Transmitter

The transmitter handles the actual conversion of the bits into acoustic signals. Figure 4 and Figure 5 show the block diagrams for the transmitter. Figure 5 represents the block labeled Xmit in the previous figure. The data bits are first passed through the convolutional encoder and encoded according to the codeNum provided by the transmitter controller. The codeNum may also indicate no coding, in which case the data bits pass through the encoder block without any modification. The encoded bits are then interleaved and passed to a circular buffer. The circular buffer outputs one frame duration of bits every tick. These bits are passed through an interpolation filter. Finally, the signals are carrier modulated and sent to the D/A converter.

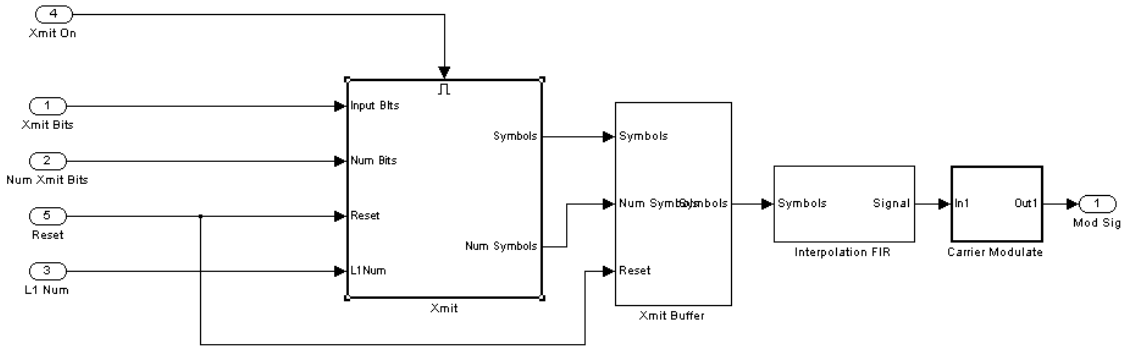

Figure 4 The modulated symbols are passed through an interpolation filter and carrier modulated before sent to the channel.

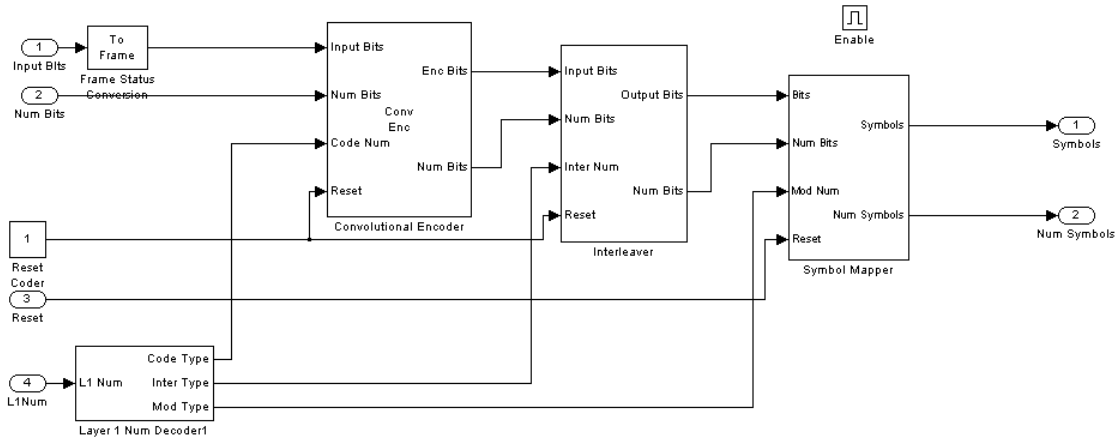

Figure 5 The transmitter encodes, interleaves, and modulates the data bits.

2.2.3 Physical Layer Receiver Controller

The physical layer receiver controller handles the operation of the acoustic receivers. Figure 6 shows the state machine for the receiver controller. We keep the receiver in the PreambleSearch state as long as the transmitter is not turned on. If the transmitter is turned on by the transmitter controller, then the receiver switches to the Idle state. (Note that the transmitter can only be turned on if the receiver is in the PreambleSearch state.)

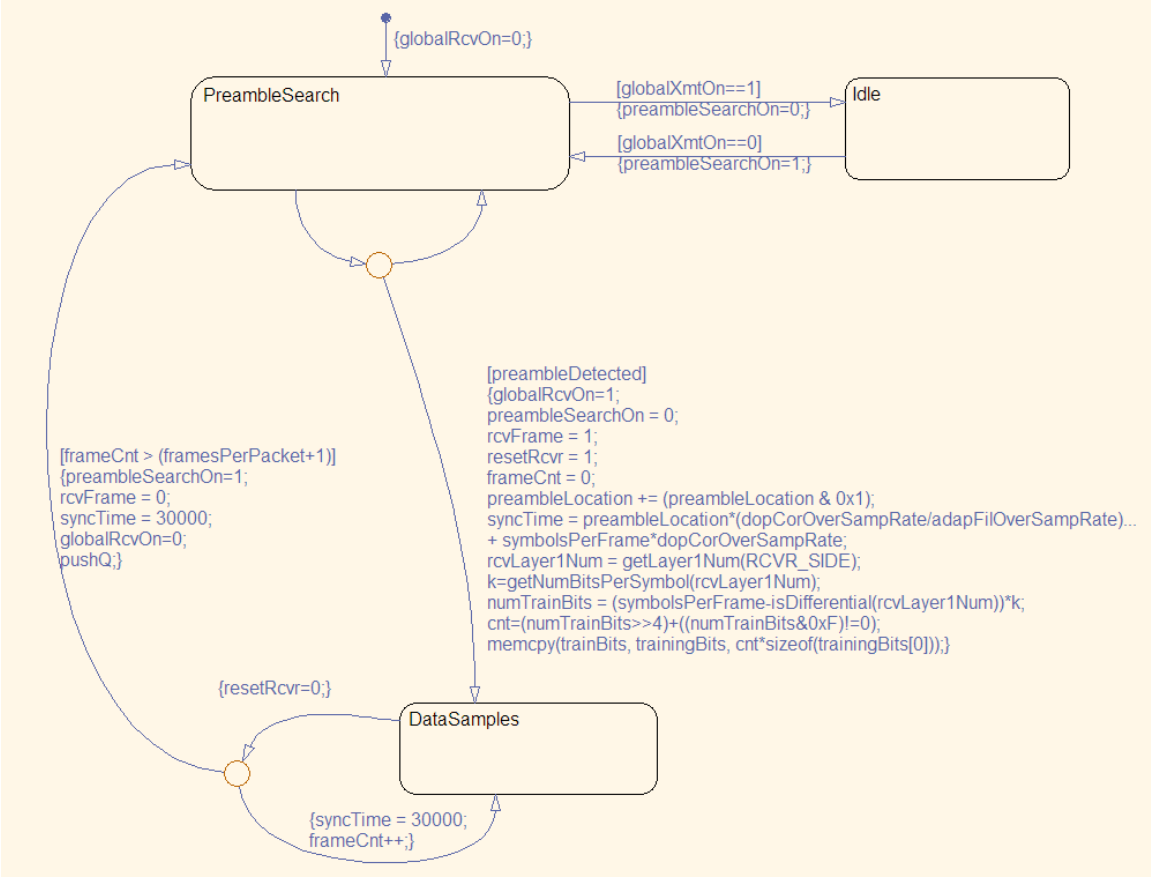

Figure 6 State machine of the physical layer receiver controller handles the operations of the acoustic receiver.

When the transmitter is turned off, the receiver controller switches to the PreambleSearch state. In this state, the samples received from the A/D converter are correlated with the known preamble. If the controller detects a preamble, it turns on the receiver, initializes the receiver, and enters a loop of length (framesPerPacket+1) to receive the packet. Upon completion of the reception of the packet, the controller turns off the receiver and returns to the PreambleSearch state.

The preamble detector returns the location of the preamble (preambleLocation) in the last received frame of samples. The controller uses this information to compute the starting point of the training symbols (syncTime). The syncTime reported by the controller is a multiple of the adaptive filters oversampling rate (adapFilOverSampRate). By this way, we make sure that we will always sample at the right instance without the need of a state variable to remember the number of the location of the last sample.

The receiver controller also provides the training bits used by the adaptive equalizer.

The current version of the receiver controller requires the user to set the modulation type of the receiver signal. In the future versions, we are planning to add a header to each

packet that will carry the modulation type information of the following packet. The controller will initialize the receiver according to this information.

2.2.4 Physical Layer Receiver

The acoustic receiver consists of two major parts: the *Preamble Process* block and the *Demodulator* block (Figure 7). The samples received from the A/D converter are first down converted to baseband. The baseband samples are then passed through a decimator filter. The output of the decimator filter is fed to both the *Preamble Process* block and the *Demodulator* block.

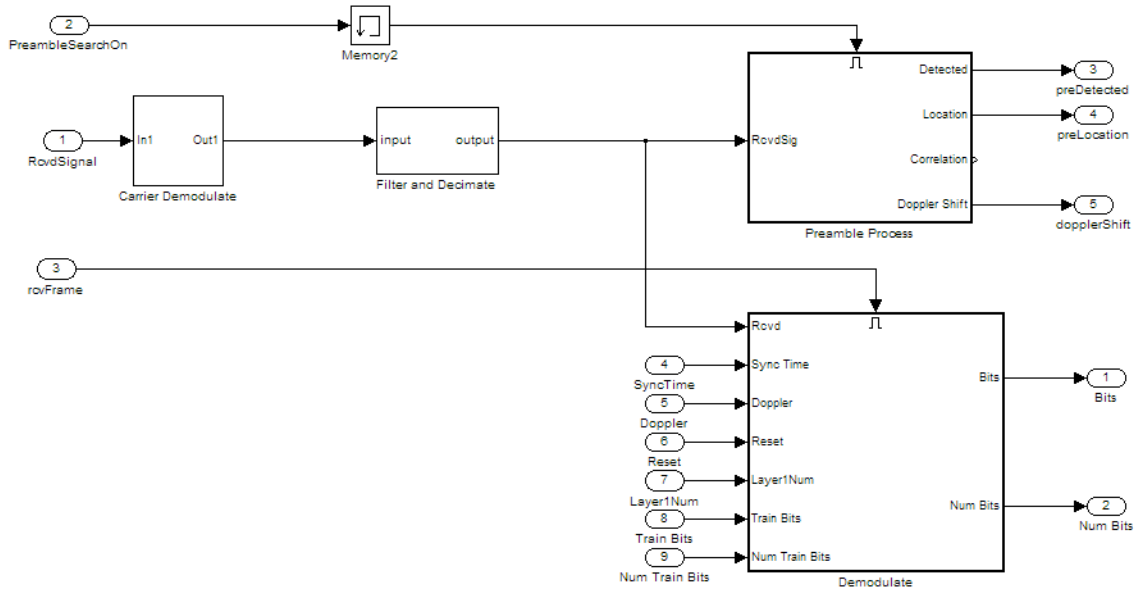

Figure 7 Block diagram of the acoustic receiver.

In the PreambleSearch state, the receiver controller enables the *Preamble Process* block, which is shown in Figure 8 **The preamble search block correlates the received samples with the known preamble.**. The received samples are further downsampled before correlating with the known preamble to reduce the computational cost. This block is also responsible for providing an estimate of the Doppler shift present in the received signal. If the correlation value exceeds a threshold, this block issues a detection signal together with the position of the preamble and the Doppler shift estimate.

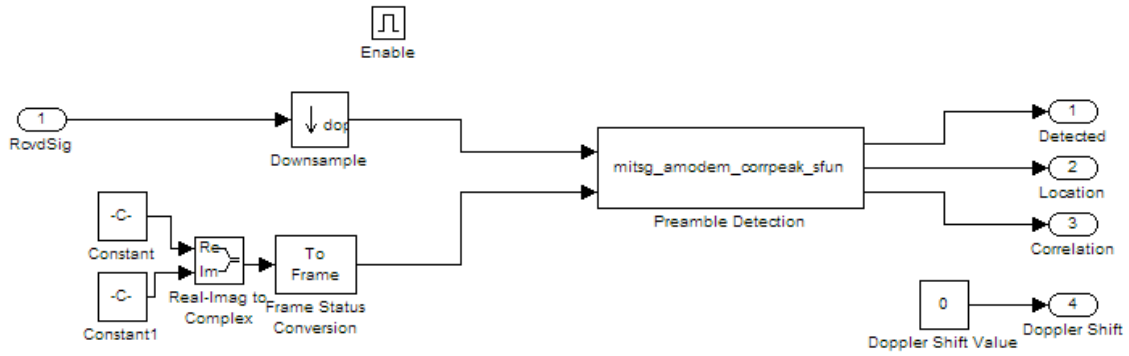

Figure 8 The preamble search block correlates the received samples with the known preamble.

The Demodulator block is shown in Figure 9. Following the detection of a preamble, the controller disables the *Preamble Processor* block and enables the *Demodulator* block. The received samples are first passed through a Doppler compensator, synchronizer, and decimator. The output of the *Sync Doppler Decim* block is fed into the equalizer. The equalized symbols are demapped into soft bit values. The *Deinterleaver* block has an internal buffer where the soft bits of a data packet are buffered until the whole packet is received. Then the soft bits are deinterleaved and decoded in the *Viterbi Decoder* block.

By placing the buffer just before the deinterleaver, we can distribute the computationally intensive equalization process over multiple frame durations and timing requirements for real-time operation. Due to the presence of a deinterleaver, we cannot employ the same approach to the Viterbi decoder.

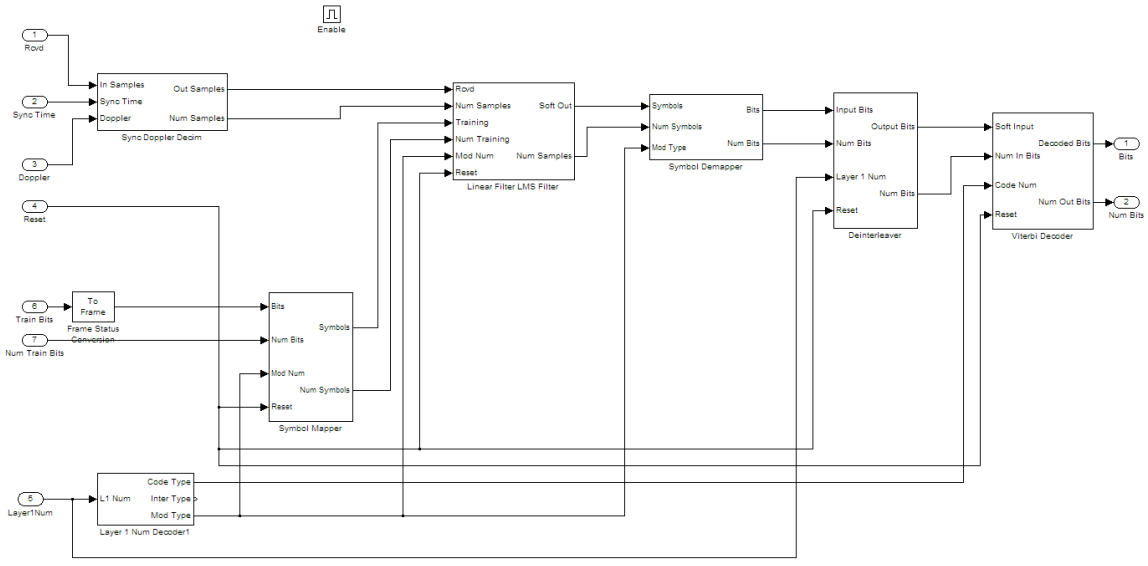

Figure 9 The demodulator block converts the received signals into a bit stream.

2.3 Transport Layer

The transport layer is responsible for dividing the data to be transmitted into packets and assembling the received packets. We modeled the transport layer with two parallel state machines (see Figure 10): *Transport_Xmt_Ctrl* and *Transport_Rcv_Ctrl.*

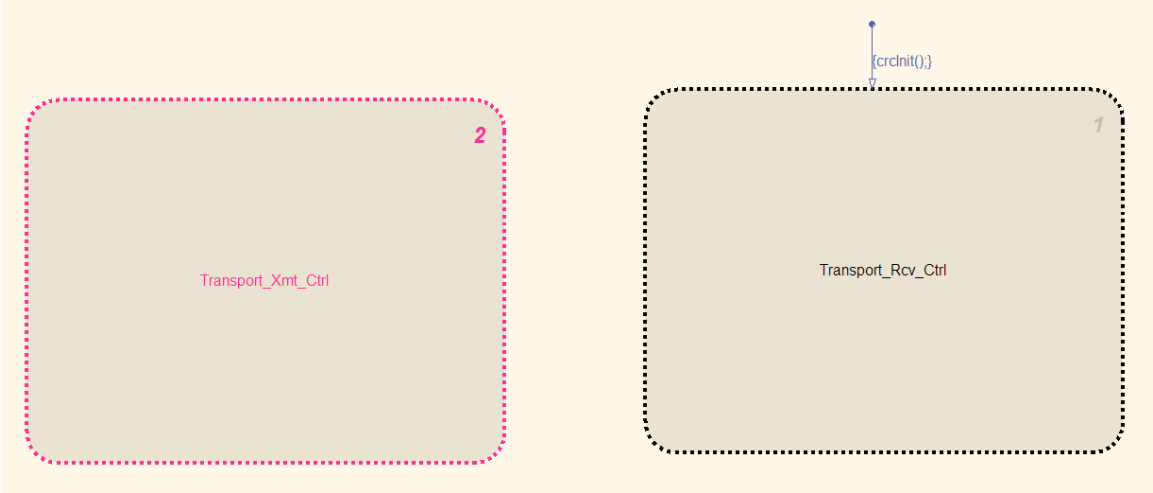

Figure 10 The transport layer has two parallel state machines.

The *Transport_Xmt_Ctrl* state machine represents the controller for the transmitter side. The details of the state machine are shown in Figure 11. When the controller detects a packet in the queue, it issues a popXmtQ signal and initializes a session. The initialization involves assigning a session number, determining the number of bits in a Layer 3 packet based on the physical layer setting. The physical layer settings that affect the Layer 3 packet size are the modulation and coding types.

Once the packet is received from the queue, the controller determines the number of Layer 3 packets needed to carry the information. Then the controller enters a loop of length xmtNumPackets. At each execution of the loop, the controller creates a new Layer 3 packet, enters the header information, and copies the payload bits. The created Layer 3 packets are pushed into the lower layer's queue.

Upon completion of the loop, the controller checks for a new packet in its queue. If there is a new packet, it issues a popXmtQ signal and initializes a new session. Otherwise, the controller returns to the Idle state.

The current version of the transport layer transmitter controller does not check for queue overflows. Therefore, if the rate of new data arrival to the transport layer is more then the rate of the lower layers, packets may be lost. For now, it is the upper layers' responsibility to ensure that no queue overflow will occur.

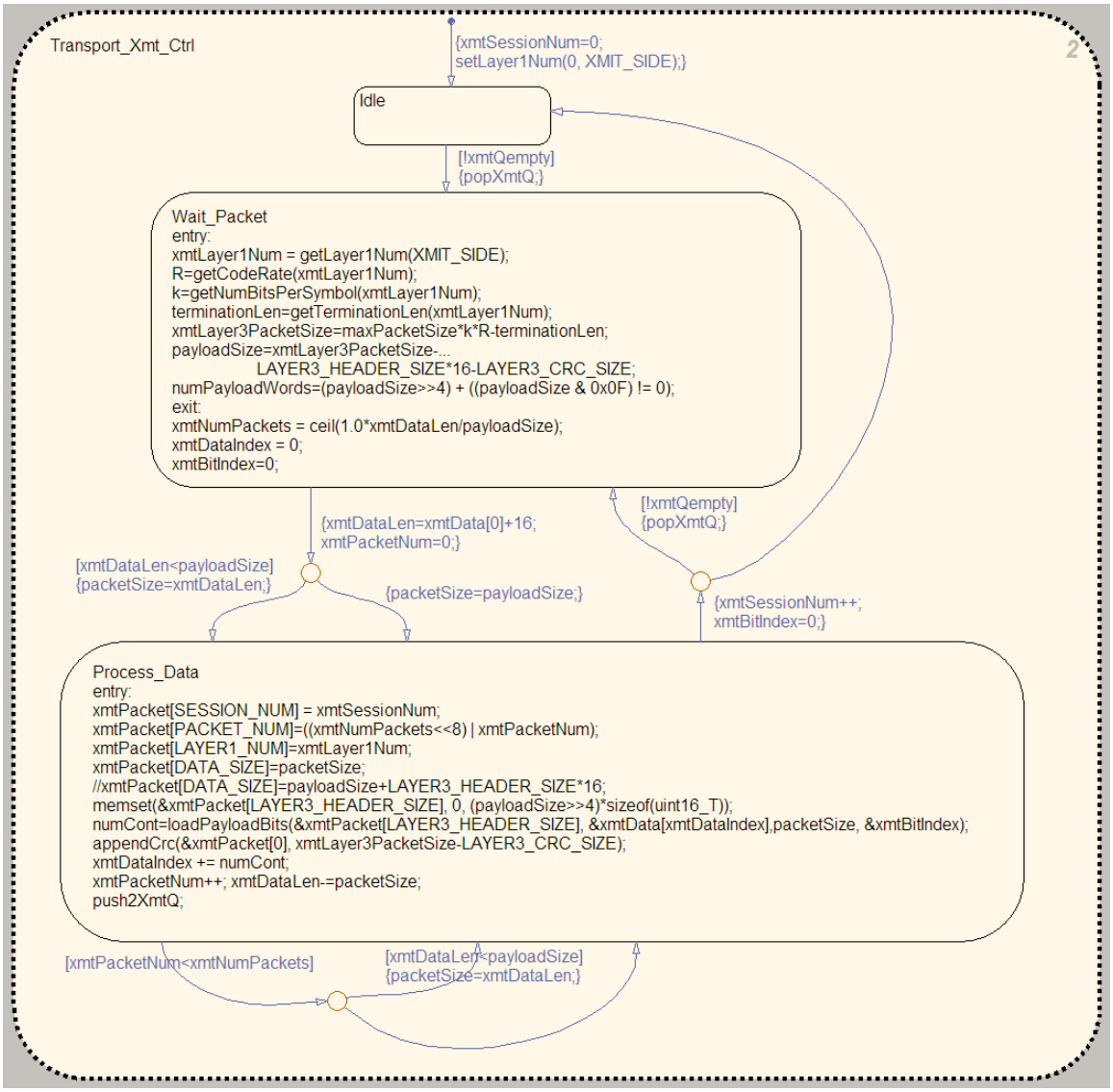

Figure 11 The details of the state machine for the transport layer transmitter controller.

The Transport_Rcv_Ctrl state machine represents the controller for the transmitter side. The details of the state machine are shown in Figure 12. The controller waits in the Idle state until a packet appears in its queue. Upon detection of the packet, the controller issues a popRcvQ signal and determines the expected payload size of the received Layer 3 packet based on the current physical layer settings. When the controller receives the packet from the queue, it first checks the CRC and determines if the packet is valid. If it is a valid packet, then the controller reads the header to determine the session number, number of packets in this session, packet number, and the size of the data in this packet. If this is the first packet of a session and there is no open session, the controller starts a new session. If there is an open session, the packet is ignored. The current version of the transport layer can handle one session at a time. If the packet is accepted by the transport layer, it is placed into the reassembly buffer.

If the session requires more packets, the controller checks the queue for a new packet. If there is a new packet, then the process is repeated for the new packet. Otherwise, the controller sets a timer and waits for a new packet from the queue. If the timer expires before the arrival of a packet, then the session is closed before completion. If all the packets of a session are received successfully, then the reassembled Layer 4 packet is sent to the higher level.

This version of the transport layer receiver uses the Layer 3 payload length to determine how to reassemble the received packets. However, this may cause a problem if the physical layer receiver settings are changed before all the packets of a session are processed or if the packets are transmitted using different physical layer settings. We will address this possible source of problem in the future versions.

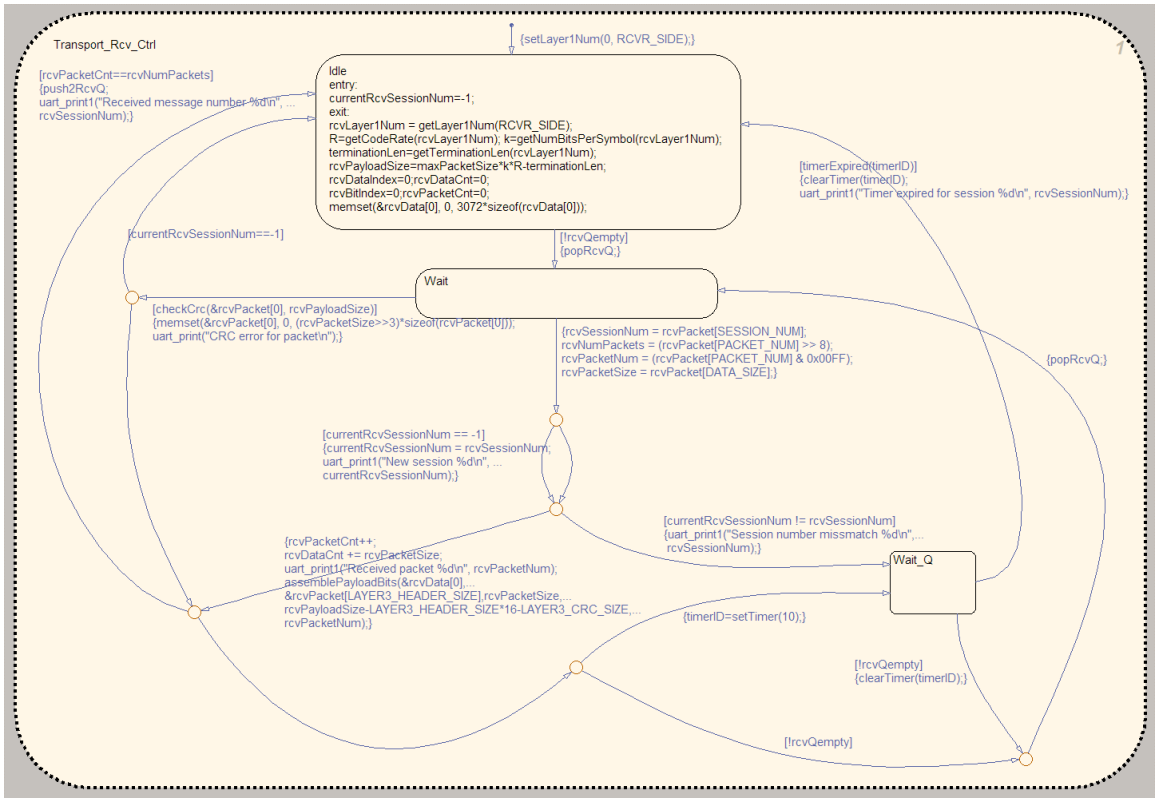

Figure 12 The state machine for the transport layer receiver controller.

2.4 UART Receiver Controller

The UART receiver controller parses the information received from the UART (or the serial port). The information received from the UART can be commands or data. In the ASCII mode, the UART receiver handler assumes that the received information is 8-bit ASCII command. These commands can be used to set the receiver type, transmitter type, and to send data. We also implemented special commands to debug or send large canned files.

Some of the commands implemented for the preliminary version of the rModem are as follows:

1) rmsnnd: send data d, which is a series of hexadecimal numbers represented in ASCII, to modem nn.
2) rmvn: set the verbose mode to n.
3) rmr: get the state register
4) rmd[nn]: set or get the default recipient ID
5) rmct: get the clock tick value
6) rmcf: get the clock tick frequency

2.5 UART Transmitter Controller

UART transmitter controller handles the transmission of data to the user through the serial port. The data can be either received bits or status information. This block ensures that the buffer in the UART driver does not overflow.

3 Reconfigurable Modem Hardware

The reconfigurable modem hardware consists of 3 parts: main board, power amplifier – preamplifier board, and transducer(s). In the following we describe these hardware modules.

3.1 Main Board

Main board of the second version of the rModem hardware combines the power supply, DSP board, and the analog digital interface.

The power supply is designed to provide several digital and analog voltage levels with a wide range of input voltages, specifically 8 to 40 Volts. We paid special attention to prevent the switching power supplies to generate noise that can effect other devices connected to the same source.

We choose Texas Instruments TMS320C6713 DSP together with an Altera Cyclone II 33k gate FPGA. The TMS320C6713 is a MHz floating-point DSP processor with a theoretical maximum performance of 2400 MFLOPS. We decided to utilize a floating-point processor to minimize the time required to convert simulation software into real-time code. The processing power of this DSP is enough to minimize the hand optimization effort for rapid prototyping.

The price we pay for high performance with floating point functionality is high power consumption as compared to the C5000 series low power DSP chips. As we intend to employ the rModem as a research-based rapid prototyping environment, we decided to choose ease of programming over low power consumption. We assume that these modems will not be employed for extended periods without maintenance or will be deployed within a system which does not have strict power consumption requirements for its peripherals, such as an AUV.

The analog-digital interface features four analog-to-digital and digital-analog (AD/DA) channels. By employing multiple input and output channels, we will be able to develop and test multi-input-multi-output (MIMO) modems.

Each channel on this board can sample at 240 kHz and has a built in anti-aliasing filter with cut-off frequency at 100 kHz. However, we can program the board to provide us with a lower sampling rate, by decimating the signals in the on board FPGA. We can program the sampling rate of the A/D converter, the decimation rate, and the decimation filter coefficients. The same coefficients are used to interpolate the signals going to the D/A channels. The coefficients can be programmed during start up.

The main board also provides a daughter card expansion where external peripherals can be connected to the main board through the FPGA. We designed this expansion slot mainly to host the power amplifier – preamplifier board.

The main board also features 32 Mbyte on board SDRAM and 32 Mbyte FLASH ram. Since rModem may be deployed in an observation station where space is limitted, we paid special attention to the size of the system. The main board dimensions are 3" x 7". The peripheral boards stack on the DSP board. The final size of the system depends on the number of peripherals.

3.2 Power Amplifier Board

Power amplifier – preamplifier board is connected to the main board through the daughter card interface. The power amplifier is a modified Class B amplifier that can deliver about 15 W of transmission power. The preamplifier is employed to provide about 25 dB of constant gain in addition to the variable gain amplifier on the main board.

4 References

[1] Bertsekas D. and Gallager R., *Data Networks*, 2nd Edition, Prentice-Hall Inc., 1992