LabVIEW-Based Software-Defined Radio: 4-QAM Modem

N. KIM, N. KEHTARNAVAZ, and M. TORLAK
Department of Electrical Engineering, University of Texas at Dallas
Richardson, TX 75080, USA

Abstract

A software-defined radio consists of a programmable communication system where functional changes can be made by merely updating software. In this paper, a software-defined radio 4-QAM (Quadrature Amplitude Modulation) modem system is implemented in LabVIEW. LabVIEW is a widely used graphical programming environment which allows designing systems in an intuitive block-based manner in shorter times as compared to the commonly used text-based programming languages. Basically, this paper demonstrates the ease with which a software-defined radio system can be built and analyzed via the LabVIEW graphical programming environment. Examples are provided to demonstrate the phase and frequency tracking capability of the system.

Keywords: Software-defined radio, QAM modem, LabVIEW graphical programming, Phase and frequency tracking.

1. INTRODUCTION

This paper discusses a software-defined radio (SDR) system built using LabVIEW. A software-defined radio consists of a programmable communication system where functional changes can be made by merely updating software. Similar to other digital communication systems, the transmitter of a SDR system converts digital signals to analog waveforms. These waveforms are then transmitted to the receiver. The received waveforms are downconverted, sampled, and demodulated using software on a reconfigurable baseband processor. Normally, high-performance digital signal processors and/or FPGAs are used to serve as the baseband processor. Basically, the programmability and flexibility of a SDR system makes it possible for it to be used in ubiquitous network environments [1-5].

In this paper, 4-QAM (Quadrature Amplitude Modulation) is chosen to be the modulation scheme of the designed software-defined radio system noting that this modulation is widely used for data transmission applications over bandpass channels such as FAX modem, high speed cable, multi-tone wireless, and satellite channels. In particular, digital cable television and cable modem utilize 64-QAM and 256-QAM [6-8].

A number of hardware platforms are available in the market for development and testing of SDR systems. For example, the Sundance's SMT8096 development platform is equipped with an ADC/DAC (analog-to-digital converter), a Texas Instruments TMS320C6416 DSP as a baseband processor, and an FPGA for pre- and post signal processing [9]. Knowledge of VHDL and DSP programming is required to use the SMT8096 platform.

Here, the software implementation of the QAM modem system is accomplished using LabVIEW as a time-efficient and cost-effective solution. LabVIEW is a graphical programming environment developed by National Instruments which allows high-level or system-level design via its flow-chart intuitive block-based programming as compared to the commonly used text-based programming languages. A design using LabVIEW is achieved by integrating different blocks, components or subsystems, called Virtual Instruments (VI), within a graphical framework [10-11].

The paper is organized as follows. First, an overview of the 4-QAM transmitter and receiver is mentioned in section 2. In section 3, the LabVIEW implementation details are then discussed. The simulation results are presented in section 4. Finally, the conclusions are stated in section 5.

2. SOFTWARE DEFINED RADIO: 4-QAM MODEM

The building blocks of the 4-QAM modem system are stated in this section. This system has two parts: transmitter and receiver. The first three modules (message source, pulse shape filter, and QAM modulator) make up the transmitter part and the other modules make up the receiver part. A brief description of each block follows.

Message source
Pseudo Noise (PN) sequences are used for this purpose. A PN sequence is generated with a 5-stage linear feedback shift register structure, see Figure 1, whose connection polynomial is given by

$$h(D) = 1 + D^2 + D^5 \tag{1}$$

where D denotes delay and the summations represent modulo 2 additions.

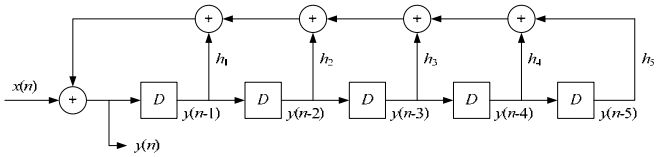


Figure 1: PN generation with linear feedback shift register.

The sequence generated via Eq. (1) has a period of $31(=2^5-1)$. Two PN sequence generators are used in order to create the message sequences for both the in-phase and quadrature phase components. In the constellation of 4-QAM, the reference signals are located at each quadrant.

Frame marker bits are inserted in front of the generated PN sequences. This is done for frame synchronization which is

discussed later. A known bit sequence of length 10 is used as the frame marker. This frame marker is chosen to carry low correlation with PN sequences.

Pulse shape filter
The generated message sequences are passed through a raised-cosine FIR filter to create a band-limited baseband signal. The excess bandwidth beyond the Nyquist frequency is specified by a roll-off factor of the filter. In our implementation, a roll-off factor of 0.5 is used.

QAM modulator
The output of the raised cosine filter is then used to build a complex envelope, $\tilde{s}(t)$, of a QAM signal expressed by

$$\tilde{s}(t) = \sum_{k=-\infty}^{\infty} c_k g_T(t - kT) \tag{2}$$

where c_k indicates a complex message, made up of two real messages a_k and b_k, $c_k = a_k + jb_k$.

After modulating $\tilde{s}(t)$ with $e^{j\omega_c t}$, the transmitted QAM signal, $s(t)$, can be expressed as

$$\begin{aligned} s(t) &= \Re e\left[\tilde{s}(t)e^{j\omega_c t}\right] \\ &= a(t)\cos(\omega_c t) - b(t)\sin(\omega_c t) \end{aligned} \tag{3}$$

where $\Re e[\cdot]$ corresponds to the real part of the complex value inside the brackets.

Hilbert transformer
A Hilbert transformer builds the analytic signal for demodulation from the transmitted QAM signal. That is, if $r(nT)$ is considered to be the sampled received signal, the analytic signal $r_+(nT)$ is given by

$$r_+(nT) = r(nT) + j\hat{r}(nT) \tag{4}$$

where $\hat{r}(\cdot)$ indicates the Hilbert transform of $r(\cdot)$. An FIR filter is used for its implementation.

QAM demodulation
This block involves the multiplication of a complex carrier having a negative frequency with the analytic signal obtained from the Hilbert transformer block.

The complex envelope of the received QAM signal $\tilde{r}(nT)$ can be expressed as

$$\begin{aligned} \tilde{r}(nT) &= r_+(nT)e^{-j\omega_c nT} \\ &= a(nT) + jb(nT) \end{aligned} \tag{5}$$

Frame Synchronization
Frame synchronization is required for properly grouping transmitted bits into an alphabet. To achieve this synchronization, a similarity measure, consisting of cross-correlation, is computed between the known marker bits and received samples. The cross-correlation of two complex values v and w is given by

$$R_{wv}[j] = \sum_{n=-\infty}^{\infty} \overline{w}[n]v[n+j] \tag{6}$$

where the bar denotes complex conjugate.

Decision Based Carrier Tracking
Let us now examine the phase offset, denoted by θ, between the transmitter and the receiver. Based on this offset, the received signal can be written as

$$\begin{aligned} \tilde{r}(nT) &= r_+(nT)e^{-j(\omega_c nT+\theta)} \\ &= \hat{c}_n e^{-j\theta} \end{aligned} \tag{7}$$

where $\hat{c}_n$ indicates the output of a slicer mapping a received sample to the nearest ideal reference in the signal constellation. As a result, the baseband error at the receiver is given by

$$\tilde{e}(nT) = \hat{c}_n - \tilde{r}(nT) \tag{8}$$

Next, the LMS (least mean square) update method is used to minimize the phase error $\Delta\theta(n)$ given by

$$\Delta\theta(n) = \frac{\Im m\left\{\overline{\tilde{e}(nT)}\tilde{r}(nT)\right\}}{|c_n|^2} \tag{9}$$

When both phase and frequency tracking are considered, the carrier phase of the receiver becomes the phase update $\Delta\varphi(n)$ and is given by

$$\Delta\varphi(n) = k_1\Delta\theta(n) + \psi(n) \tag{10}$$

where $\psi(n)$ denotes the contribution of frequency tracking, which can be expressed as

$$\psi(n) = \psi(n-1) + k_2\Delta\theta(n) \tag{11}$$

The scale factors k_1 and k_2 are configured to be small here and usually $k_1/k_2 \geq 100$ is required for phase convergence. The reader is referred to [2] for more theoretical details of the modem mentioned here.

3. LABVIEW IMPLEMENTATION OF QAM MODEM

This section presents the LabVIEW software implementation of the 4-QAM modem system. LabVIEW is a graphical programming environment which allows one to design complex DSP systems in a relatively time-efficient manner as compared to textual programming.

A LabVIEW program consists of two major components: Front Panel (FP) and Block Diagram (BD). A Front Panel provides a graphical user interface while a Block Diagram contains building blocks of a system resembling a flowchart. LabVIEW

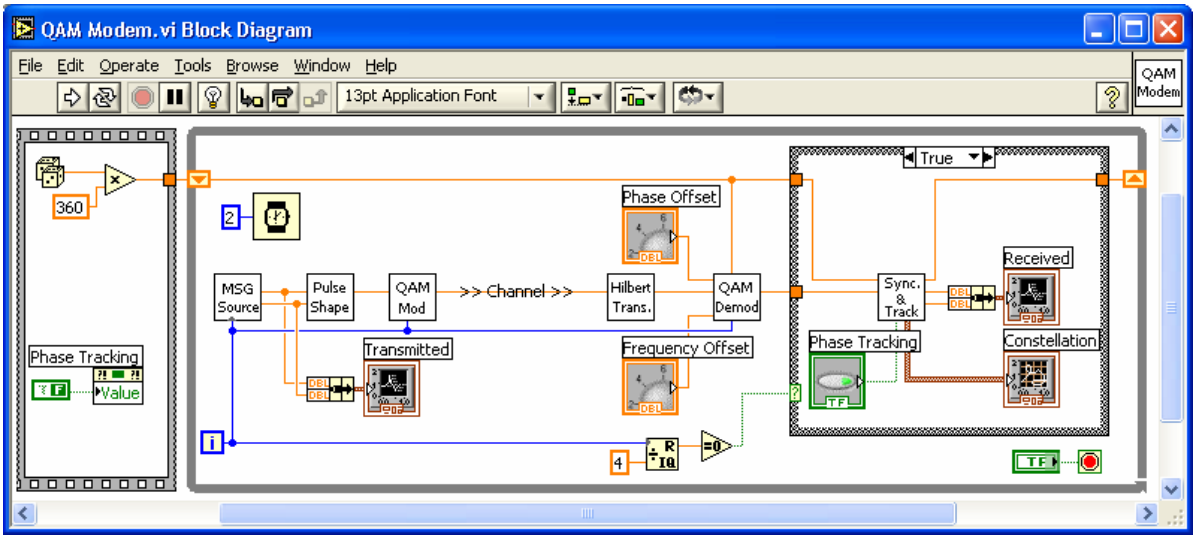


Figure 2: System-level BD of 4-QAM modem system.

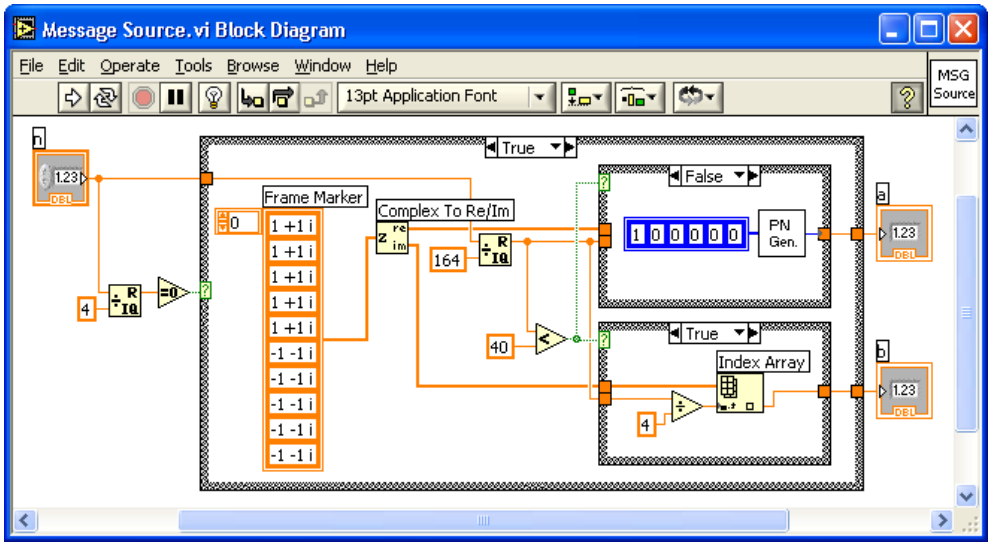


Figure 3: Message Source VI.

blocks are called Virtual Instruments, or VIs. The interested reader is referred to [10-11] for details on LabVIEW programming.

A system-level BD of the QAM modem is shown in Figure 2. An overview description regarding the implementation of each block follows.

QAM Transmitter

Message source: The first component of the QAM modem is the message source. Here, PN sequences are used for this purpose. Frame marker bits are inserted in front of these sequences to achieve frame synchronization. The BD of the Message Source VI is shown in Figure 3**Error! Reference source not found.**.

Note that the generated samples are oversampled 4 times according to the specification of the pulse shape filter. This is done by comparing with 0 the remainder of a global counter. Thus, out of four executions of this VI, one message sample (frame marker bit or PN sample) is generated. For the remaining three executions of the VI, zero samples get generated. The total length of the message for one period of a PN sequence and frame marker bits is 164, which is obtained by 4 (oversampling rate) × [10 (frame marker bits) + 31 (period of PN sequence)]. A constant array of 10 complex numbers is used to specify the marker bits. The real parts of the complex values are used as the frame marker bits of the in-phase samples and the imaginary parts as the frame marker bits of the quadrature-phase samples.

The PN Generator VI, shown as an icon in Figure 3, generates a pseudo-noise sequence of length 31 by XORing the values of the second and fifth shift registers.

Pulse shape filter: Next, the generated samples are passed through a pulse shape filter that is shown in Figure 4. A raised cosine filter is used to serve as the pulse shape filter. An FIR filter (FIR Filter PtByPt VI) is utilized for this purpose. The two outputs of the pulse shape filter are combined to construct the complex value pulse shaped message signal. The filter coefficients can be obtained by any filter design tool such as LabVIEW DFD toolkit and stored in an array of constants.

The coefficients are designed based on the specified oversampling rate, i.e. 4 in our case.

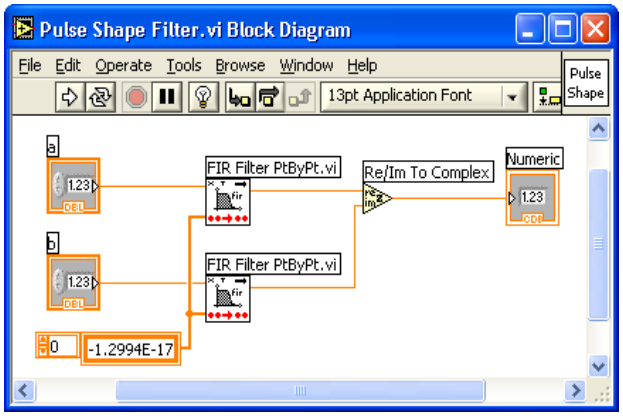
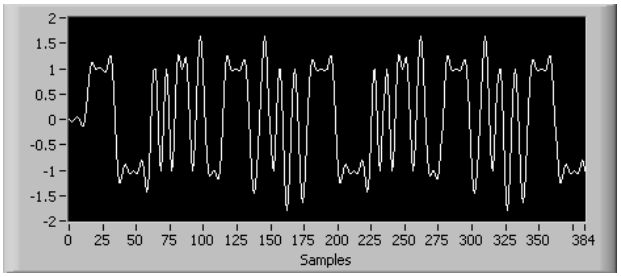
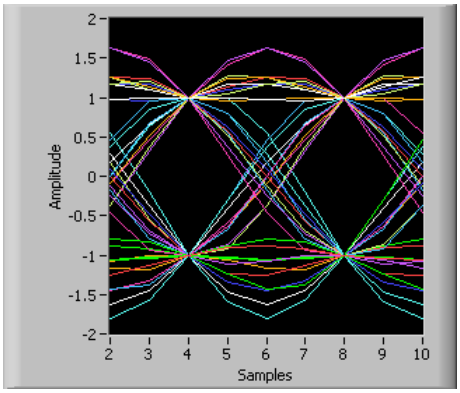


Figure 4: Pulse shape filter.

The output of the pulse shape filter is illustrated in Figure 5. As shown in the figure, the digital sequence is smoothed or filtered to minimize any intersymbol interference (ISI). The eye diagram of this signal is also shown in Figure 5.



(a)



(b)

Figure 5: Pulse shape filter output and eye diagram.

Modulator: The signal passed through the pulse shape filter is then connected to the QAM modulator shown in Figure 6. The QAM modulated signal $s(t)$ is obtained by taking the real part of the pre-envelope signal $s_+(t)$. This is achieved by performing a complex multiplication between the complex input and a complex carrier consisting of a cosine and a sine waveform.

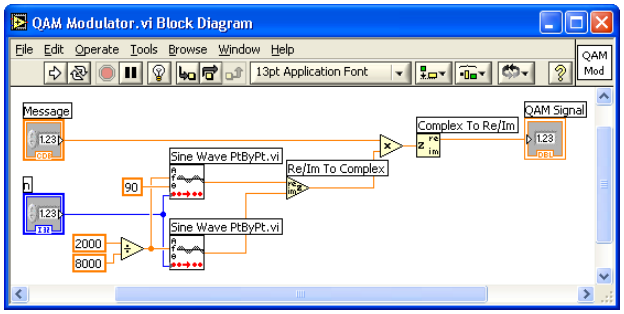


Figure 6: QAM modulator.

This completes the transmitter. Next, the receiver blocks are mentioned.

QAM Receiver

Hilbert transformer: The first module on the receiver side is the Hilbert transformer. This module builds the required analytic signal for demodulation based on the transmitted QAM signal. The Hilbert transformer is implemented as a bandpass filter with the specification indicated in Figure 7. To have an integer group delay, an even number, such as 32, is specified as the filter order.

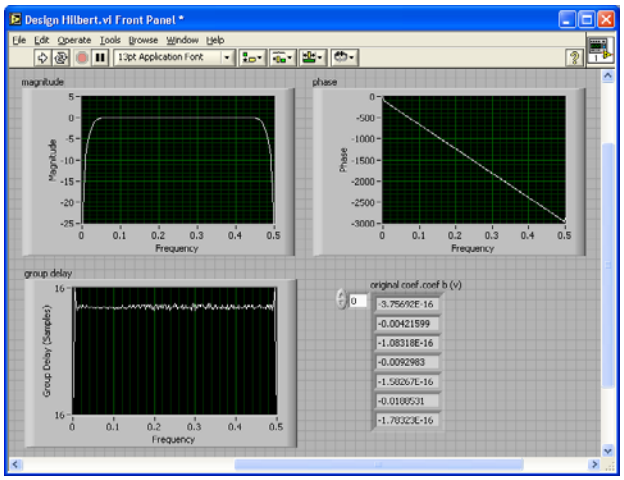


Figure 7: Analysis of Hilbert transformer.

Once the coefficient set of the Hilbert transformer is acquired based on the supplied specifications, the Hilbert transformer is implemented using an FIR filter with the coefficients previously obtained. This VI is shown in Figure 8.

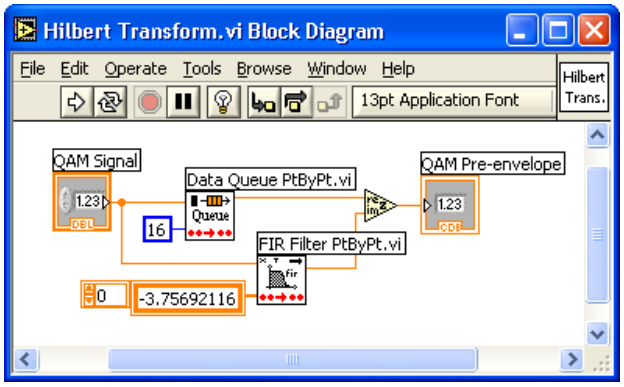


Figure 8: Hilbert Transform VI.

A data queue (Data Queue PtByPt VI) is employed in order to synchronize the input and output of the Hilbert transformer. In other words, the input samples are delayed until the corresponding output samples become available. This is needed due to the group delay associated with the filtering operation. For an FIR filter of 33 taps or order of 32, the group delay is 16. An array of numeric constants corresponding to the filter coefficients is set up based on the designed Hilbert transformer shown in Figure 7.

Demodulator: The analytic signal obtained from the Hilbert transformer is demodulated by the QAM demodulator as illustrated in Figure 9. The demodulation process is similar to the modulation process except for the negative frequency part.

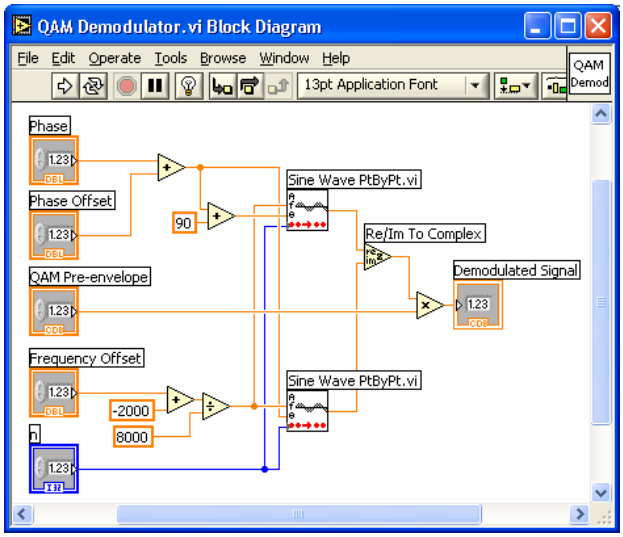


Figure 9: QAM demodulator.

Frame Synchronization: Next, the QAM demodulated signal is decimated by 4. To achieve this, a Case Structure is used so that every fourth sample is selected for processing, as illustrated in Figure 2. The decimated signal is sent to the Sync & Tracking VI for frame synchronization and phase/frequency tracking. The Sync & Tracking VI is an intermediate level subVI incorporating several subVIs/functions and operating in two different modes: frame synchronization and phase/frequency tracking.

Let us examine the BD of this VI, which is displayed in Figure 10. The input samples are passed into the receiver queue, implemented via the Complex Queue PtByPt VI, in order to obtain the beginning of a frame by cross-correlating the frame marker bits and received samples in the queue. Filling the queue is continued until the queue is completely filled. Extra iterations are done to avoid including any transient samples due to the delays associated with the filtering operations in the transmitter.

The length of the queue is configured to be 51 in order to include the entire marker bits in the queue. This length is decided based on this calculation: 31 [(one period of PN sequence) + 2 × 10 (frame marker bits)]. Also, 16 extra samples are taken to flush out any possible transient output of the filter as mentioned previously. Bear in mind that the length of the queue or the number of extra reads varies based upon the specification of the transmitted signal such as the length of the frame marker bits and the number of taps of the phase shape filter. A counter, denoted by the Loop Count VI in Figure 10, is used to count the number of samples filling the queue. Once the queue is completely filled and extra reads are done, the frame synchronization module is initiated.

The VI for frame synchronization is shown in Figure 11. In this VI, the cross-correlation of the frame marker bits and the samples in the receiver queue are computed. The absolute value of the complex output is used to obtain the cross-correlation peak since the location of this peak coincides with the beginning of the frame.

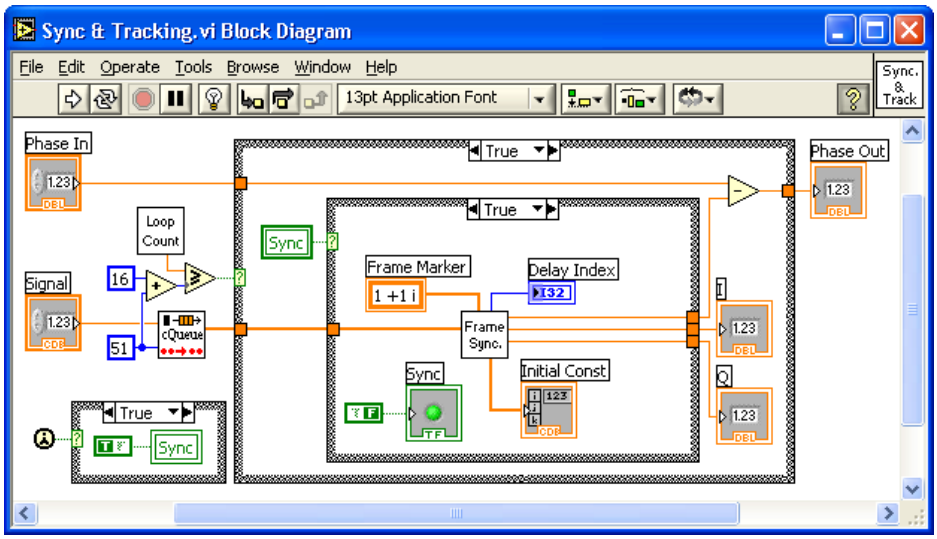


Figure 10: Sync & Tracking VI – frame synchronization mode.

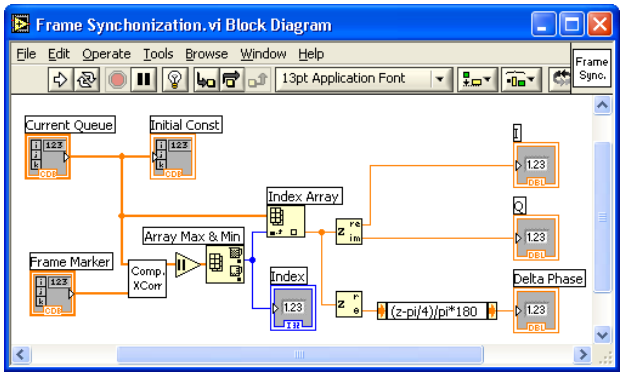


Figure 11: Frame Synchronization VI.

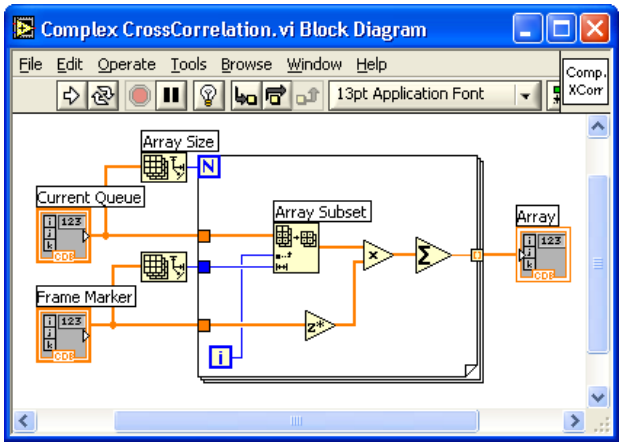


Figure 12: Complex CrossCorrelation VI.

In Figure 12, the Complex CrossCorrelation VI is shown. This VI accomplishes the complex cross-correlation operation by evaluating Eq. (6).

Once the index of the maximum cross-correlation value is obtained, all data samples are taken at this location of the queue. Consequently, data bits are synchronized.

Phase and frequency tracking: The initial phase estimation is achieved using the phase of the complex data at the beginning of the marker bits. Considering that the ideal reference is known for the first bit of the frame marker, $1+i$ in our case. This allows us to obtain the phase difference between the ideal reference and the received frame marker bits. The real and imaginary parts of data at the beginning of the marker bits are also passed to the Phase and Frequency Tracking VI to provide the initial constellation.

The subVI of the frame synchronization is now complete. Note that in order to control the flow of data for the frame synchronization, local variables, shown as a label with two border lines, are used in the BD. More details on using local and global variables can be found in [10].

The initial value of the local variable, denoted by Sync, is set to true to execute the frame synchronization. Then, it is changed to false within the case structure so that it is not invoked again. The other two local variables, Initial Const and Delay Index, are used as the inputs of the phase and frequency tracking module, see Figure 13.

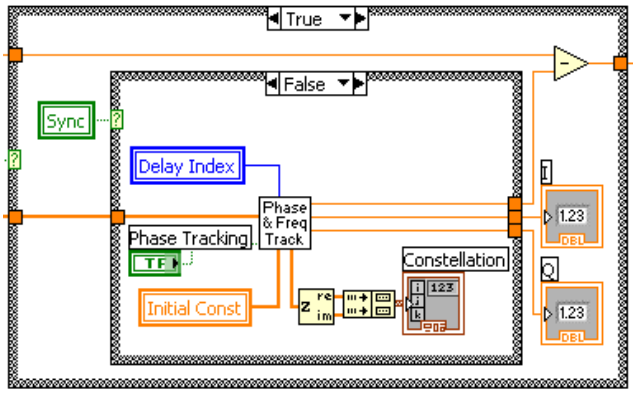


Figure 13: Sync & Tracking VI – phase and frequency tracking mode.

Next, let us describe the Phase and Frequency Tracking VI illustrated in Figure 14. A Formula Node, shown as a box with thick gray border, is capable of evaluating a script including a C or a MATLAB text-based code. The formula node shown in the upper part of the BD acts as a slicer to determine the nearest ideal reference based on the quadrant on the I-Q plane. There are numerous built-in mathematical functions and variables in LabVIEW which can be used in a formula node. For example, pi represents π in the formula node script shown in Figure 13. Further details on formula node appear in [11].

The phase error, see the BD in Figure 14, is computed from Eq. (9). This error is multiplied by a small scale factor to determine the phase update $\Delta\varphi(n)$ in a second formula node corresponding to Eq. (11).

In what follows, we provide key features of LabVIEW that are utilized for our implementation. The same features can be utilized for implementing other types of SDR.

Formula node: Formula node allows one to use C-like code in LabVIEW. In Figure 13, logical comparisons and simple mathematical equations are written in the form of C-language within the gray enclosure called Formula Node.

Digital Filtering: The digital filtering operations used in pulse shaping and Hilbert transformation can be achieved through a LabVIEW toolkit called LabVIEW DFD.

Modulation-Demodulation: Though not utilized in our implementation, the LabVIEW Modulation toolkit can be used to implement the widely used modulation and demodulation schemes.

Point-by-point processing: LabVIEW supports point-by-point processing in addition to vector/matrix processing. Thus, all the filtering and modulation VIs can be carried out based on point-by-point processing.

Other features: LabVIEW DSP Integration & LabVIEW FPGA toolkits require no textual coding. These toolkits generate the object code for specific target platforms, e.g. TI TMS320C6713 DSK. Hence, the development time can be shortened by using these tools. Also, Mathscripting is a newly added feature of LabVIEW allowing one to incorporate MATLAB compatible scripts into the LabVIEW graphical programming environment.

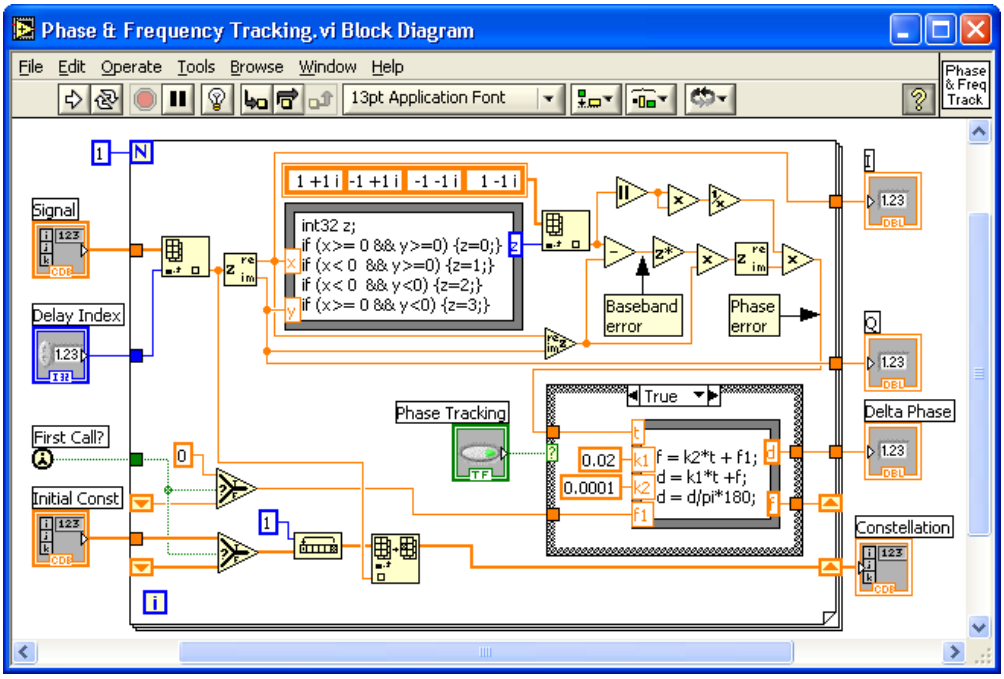


Figure 14: Phase & Frequency Tracking VI.

It is worth mentioning that other graphical tools such as Simulink can be used for the implementation of SDR. LabVIEW is used here since in a comparative study reported in [12], it is shown that LabVIEW provides preferred interactivity and graphical-user-interface capabilities.

In summary, as compared to a hardware-based solution, the LabVIEW-based SDR implementation presented in this work has enabled a rapid prototyped software solution. Moreover, in this solution, C and MATLAB textual codes can be integrated in a seamless manner. The built SDR VI can be easily converted to object codes for a number of specific target platforms.

4. SIMULATION RESULTS

This section provides the simulation study done to test the performance of the SDR modem system. To watch the simulation outcome, a waveform chart and an XY graph are added to the system-level BD shown in Figure 2.

If there exist a phase and a frequency offset with no tracking, the received signal appears as shown in Figure 15. As displayed in this figure, the constellation of the received signal is rotated, and the amplitudes of some of the received samples become too small. Obviously, the received signal will change by introducing channel noise.

By executing the phase tracking routine, the phase error, affected by the initial phase and frequency difference, is minimized and the received signal becomes a perfect reproduction of the transmitted signal except for the time delay. This is illustrated in Figure 16.

More specifically, the change in the constellation via the phase and frequency tracking is shown in Figure 16. The constellation of the samples in the I-Q plane becomes that of the ideal reference as the tracking operation progresses. Figure 16 illustrates a typical compensation achieved for the phase and/or frequency offset between the transmitter and receiver. The results show that the developed software-defined radio system provides successful decoding of the transmitted message samples.

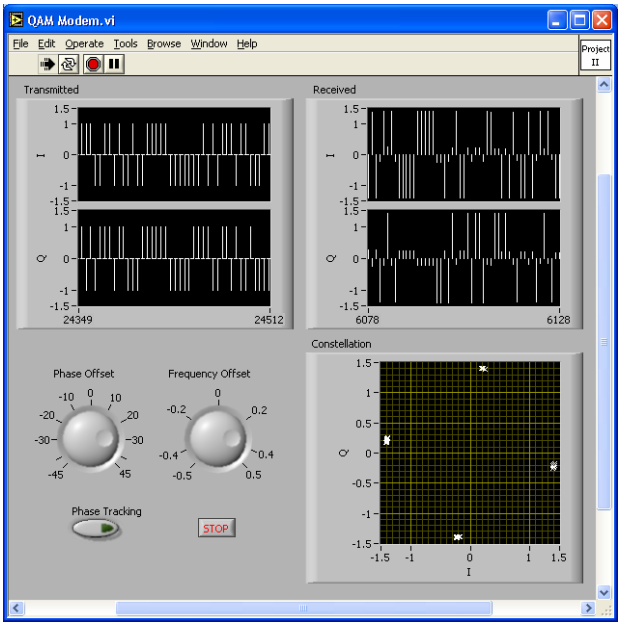


Figure 15: Received signal with no phase & frequency tracking.

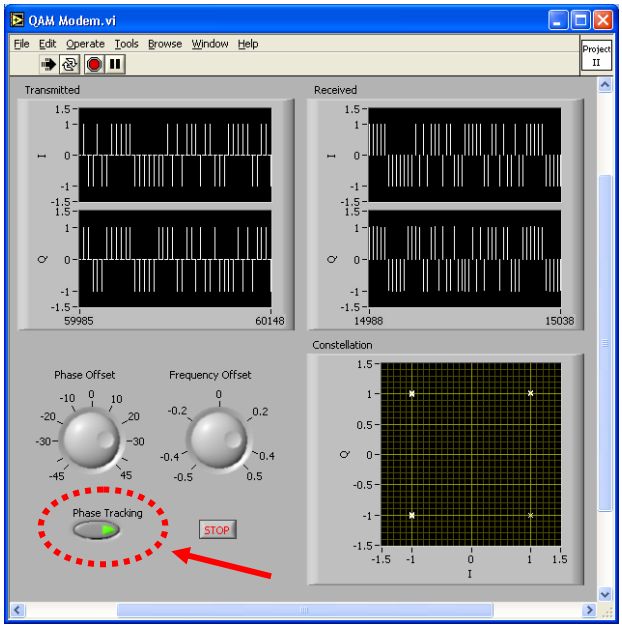
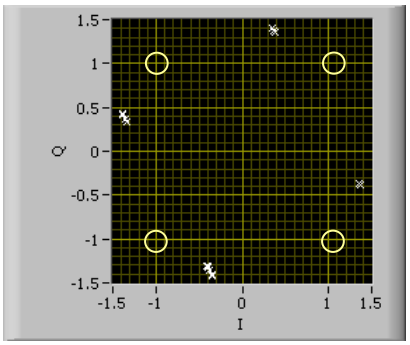
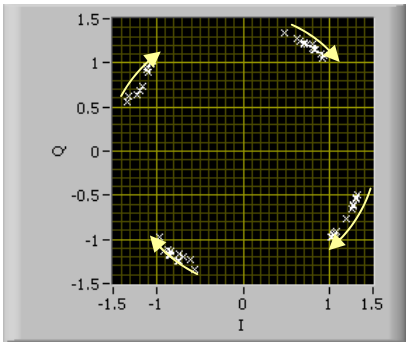


Figure 16: Perfect reproduction of transmitted signal with phase & frequency tracking.



(a)



(b)

Figure 16: 4-QAM constellation: (a) phase offset exists between transmitter/receiver (ideal references are indicated by 'o'), and (b) outcome of phase and frequency tracking process.

5. CONCLUSION

In this paper, it is shown how LabVIEW can be used to build a software-defined radio system. In particular, a 4-QAM modem system consisting of a message source, a pulse shape filter, a modulator, an automatic gain controller, a Hilbert transformer, demodulator, a frame synchronizer, and a phase and frequency tracker was built in the graphical programming environment of LabVIEW. Basically, the use of LabVIEW allowed this interactive software-defined radio system to be built in a shorter time as compared to text-based programming languages.

6. ACKNOWLEDGEMENT

This work was supported by a grant from National Instruments to University of Texas at Dallas.

7. REFERENCES

[1] W. Tuttlebee, **Software Defined Radio: Baseband Technologies for 3G Handsets and Basestations**, John Wiley & Sons , 2004.

[2] C. Johnson, Jr., and W. Sethares, **Telecommunication Breakdown: Concepts of Communication Transmitted via Software-Defined Radio**, Prentice-Hall, 2004.

[3] S. Tretter, **Communication System Design Using DSP Algorithms**, Klumer Academic/Plenum Publishers, 2003.

[4] J. Mitola, "The Software Radio Architecture," **IEEE Communication Magazine**, vol. 33, no. 5, May 1995, pp. 26–38.

[5] GNU Radio - The GNU Software Radio Project. http://www.gnu.org/software/gnuradio/

[6] D. Bryan, "QAM for Terrestrial and Cable Transmission", **IEEE Trans. Consumer Electronics**, vol. 41, no. 3, pp. 383-391, 1995.

[7] G. Karam, K. Maalej, V. Paxal, and H. Sari, "Variable Symbol-rate Demodulators for Cable and Satellite TV Broadcasting," **IEEE Trans. Broadcasting**, vol. 42, no. 2, 1996, pp. 102-109.

[8] L. D'Luna et al., "A Single-chip Universal Cable Set-top Box/Modem Transceiver," **IEEE Journal of Solid-State Circuits**, vol. 34, no. 11, 1999, pp. 1647-1660.

[9] Sundance, Software Defined Radio Development Platform. http://focus.ti.com/lit/ml/sprt348a/sprt348a.pdf

[10] N. Kehtarnavaz and N. Kim, **Digital Signal Processing System-Level Design Using LabVIEW**, Elsevier, 2005.

[11] National Instruments, **LabVIEW User Manual**, Part Number 320999E-01, 2003.

[12] N. Kehtarnavaz and C. Gope, "DSP System Design Using Labview and Simulink: A Comparative Evaluation," **Proceedings of ICASSP**, vol. 2, 2006, pp. 985-988.