

## *First Prize*

# SOPC Implementation of Software-Defined Radio

**Institution:** National Institute of Technology, Trichy  
**Participants:** A. Geethanath, Govinda Rao Locharla, V.S.N.K. Chaitanya  
**Instructor:** Dr. B. Venkataramani

## Design Introduction

Our project implements a software-defined radio (SDR) in a Nios® II processor. A software-defined radio is a radio that provides software control of a variety of modulation and demodulation techniques, wide-band or narrow-band operation, communications security functions (such as hopping), and waveform requirements of current and evolving standards over a broad frequency range. In this project, we used the Nios II processor—which supports easy reconfiguration and low development costs—to build a software-defined radio that supports many real-time applications. Altera®-based system-on-a-programmable-chip (SOPC) designs let designers implement real-time critical functions in hardware. Custom instructions make it easy to implement the whole system on an SOPC platform, with better software partitioning and hardware implementation of the software-defined radio.

## *Design Purpose*

SDR technology facilitates the software development of the radio system functional modules, such as modulation/demodulation, signal generation, coding, and link-layer protocols. This implementation helps designers build reconfigurable software radio systems in which parameters are selected dynamically. A complete hardware-based radio system has limited utility, because the functional module parameters are fixed. A radio system built using SDR technology extends the use of the system for a wide range of applications.

## *Application Scope*

SDR technology can be used to implement military, commercial, and civilian radio applications. Designers can implement a wide variety of radio applications using SDR technology, such as Bluetooth, WLAN, GPS, radar, wideband code division multiple access (W-CDMA), general packet radio services (GPRS), etc. SDR has generated tremendous interest in the wireless communications industry because of the wide-ranging economic and deployment benefits it offers.

## ***SOPC Builder Role***

The Nios II processor offers embedded design versatility and reconfigurability. It has built-in memory, peripherals, and interfaces, as well as a variety of intellectual property (IP) functions. Productivity tools—such as SOPC Builder, the Nios II Integrated Development Environment (IDE), and Nios II compiler—combined with an FPGA target device allowed us to reduce our turnaround time and costs. The Nios II processor is a perfect fit because we can choose the peripheral, performance, processor mix, and cost that best suits the our SOPC needs. The Nios II processor supports multi-processor systems, and using SOPC Builder, we can use multiple processor cores according to our design requirements.

The Nios II processor provides design flexibility because it is compatible with both software and hardware description languages. Efficient implementation of a complete SOPC is feasible when high-density FPGAs are integrated with high-capacity RAMs and the Nios II processor.

## ***Suitability of FPGAs***

Designers can use FPGAs efficiently for digital signal processing (DSP) and other computationally intensive tasks. High-bandwidth memories, embedded DSP blocks, phase-locked loops (PLLs), and high-speed interfaces can be programmed into the FPGA, which facilitates SDR implementation. Additionally, embedded processors with FPGA co-processors enable easy design reconfiguration.

## **Functional Description**

Our SDR design receives and demodulates realtime AM signals. The design can accomodate future enhancements, e.g., accommodating a transmitter in addition to the existing receiver, implementing different transmission modes, enhancing reception, and supporting the real-time requirements of civilian and military applications.

The SDR implementation involves frequency translation, analog-to-digital (A/D) conversion, and message recovery. In the frequency translation phase, the received signal frequency is translated to the intermediate frequency (IF) of 455 KHz. This part consists of an antenna, a radio frequency (RF) amplifier, a mixer, a local oscillator, a band-pass filter, and an A/D converter. We used the the MATLAB and Simulink software to implement the blocks and obtain AM signal sample values. Sampled values of the modulated signal are stored in RAM and are fed to the demodulator circuit, which is programmed into the FPGA. The demodulator circuit is based on a special sampling theorem, which makes using a mixer unnecessary. This signal then produces a demodulated output using the coordinate rotation digital computer (CORDIC) algorithm.

## ***CORDIC Algorithm***

CORDIC algorithms use shifts and adds to compute a wide range of functions, including trigonometric, hyperbolic, linear, and logarithmic functions. The CORDIC algorithm is used in diverse applications such as mathematical co-processor units, calculators, waveform generators, and digital modems.

The CORDIC algorithm uses shifts and adds to perform vector rotations iteratively. In rotation mode, CORDIC converts one vector in rectangular form to another vector in rectangular form. In vector mode, it converts a vector in rectangular form to polar form.

## ***CORDIC Rotation Mode***

The CORDIC algorithm for this mode is derived from the general rotation transform:

$$x_{\text{fin}} = x_{\text{in}}\cos\Theta - y_{\text{in}}\sin\Theta \quad (1)$$

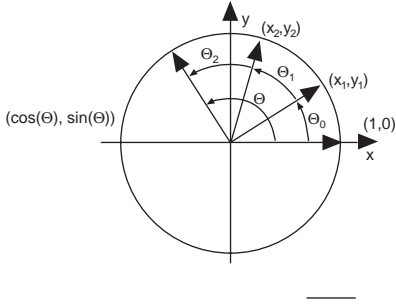
$$y_{\text{fin}} = y_{\text{in}}\cos\Theta + x_{\text{in}}\sin\Theta \quad (2)$$

The transform rotates a vector  $(x_{in}, y_{in})$  in a Cartesian plane by an angle  $\Theta$  to another vector with the coordinates  $(x_{fin}, y_{fin})$ . Rotation is achieved by performing a series of successively smaller elementary rotations  $\Theta_0, \Theta_1, \Theta_2 \dots \Theta_N$  such that:

$$\Theta = \sum_{i=0}^N \Theta_i$$

Figure 1 shows the case where rotation of a vector of magnitude 1 by an angle  $\Theta$  is achieved using three elementary rotations  $\Theta_0, \Theta_1$ , and  $\Theta_2$ .

**Figure 1. Vector Rotation by an Angle  $\Theta_i$  Using a Number of Steps**



Rotation of the vector by an angle  $\Theta_i$  can be rewritten as:

$$x_{i+1} = x_i \cos \Theta_i - y_i \sin \Theta_i \quad (3)$$

$$y_{i+1} = y_i \cos \Theta_i + x_i \sin \Theta_i \quad (4)$$

$$\frac{x_i + 1}{\cos \Theta_i} = x_i - y_i \tan \Theta_i \quad (5)$$

$$\frac{y_i + 1}{\cos \Theta_i} = y_i + x_i \tan \Theta_i \quad (6)$$

The computational complexity of equations 5 and 6 can be reduced by rewriting them as:

$$x_{i+1} = x_i - y_i \tan \Theta_i \quad (7)$$

$$y_{i+1} = y_i + x_i \tan \Theta_i \quad (8)$$

$$(x_{fin}, y_{fin}) = \left( \frac{x_n}{\prod_{i=1}^N \cos \Theta_i}, \frac{y_n}{\prod_{i=1}^N \cos \Theta_i} \right) \quad (9)$$

and performing the division by  $\cos \Theta$  together for all  $N$  iterations by dividing the value of  $(x_N, y_N)$  by:

N

$$\prod_{i=1}^N \cos \Theta_i$$

Further, the value of  $\Theta$  for  $i = 1, 2, \dots, N$  is chosen such that  $\tan \Theta$  is  $2^{-i}$ . Table 1 shows the values of the angles for  $i = 0$  to 9.

**Table 1. Values of  $\Theta = \tan^{-1}(2^{-i})$ s**

i	$\Theta$	$\tan \Theta$
0	45	1
1	26.5	0.5
2	14	0.25
3	7.1	0.125
4	3.57	0.0625
5	1.78	0.03125
6	0.895	0.015625
7	0.4476	0.0078125
8	0.2238	0.00390625
9	0.1119	0.001953125

This process reduces the multiplication by the  $\tan \Theta$  to a simple shift operation. As the iteration increases,  $\Theta$  becomes smaller and smaller. We terminate the iteration when the difference between  $\Theta$  and the sum of  $\Theta$  from 1 to N becomes very small for some value of N. The remaining angle by which the vector must be rotated after the completion of i iterations is indicated by the parameter  $z_{i+1}$  as defined by equation 10.

$$Z_{i+1} = Z_i - \Theta \quad (10)$$

$$Z_0 = \Theta \quad (11)$$

$\Theta$  is considered to be positive when the rotation required is counter-clockwise and negative otherwise. To approximate an arbitrary angle using  $\Theta$  of the form  $\tan^{-1}(2^{-i})$ ,  $\Theta$  may be negative for some values of i. For example, to approximate 50, we choose  $\Theta$  as 45, 26.5, -14 and, -7.1 in the first four iterations. (The actual sum of these angles is 50.4.) The sign (sgn) of  $z_i$  indicates whether, in the next iteration, the rotation should be counter-clockwise or clockwise. Because,  $\tan \Theta$  is  $+2^{-i}$  when  $\Theta$  is positive and  $-2^{-i}$  otherwise, the iterative equations may be rewritten as:

$$\delta_i = \text{sgn}(z_i) \quad (12)$$

$$x_{i+1} = x_i - \delta_i y_i 2^{-i} \quad (13)$$

$$y_{i+1} = y_i + \delta_i x_i 2^{-i} \quad (14)$$

$$z_{i+1} = z_i - \delta_i \tan^{-1}(2^{-i}) \quad (15)$$

The computation of

$$\prod_{i=1}^N \cos \Theta_i$$

can be simplified. Because  $\cos \Theta_i = 1$  for very small values of  $\Theta_i$ , the equation can be computed for  $N = 6$  (therefore  $K = 0.6073$ ), and can be used for any other value of  $N > 6$ .

### ***CORDIC Vector Mode***

In this mode, an initial vector with the  $x, y$  coordinates of  $(x_{in}, y_{in})$  is rotated such that its  $y$  coordinate becomes zero. The procedure used for rotation may be adopted for vector mode with the following modifications: rotation is carried out clockwise (so that the  $y$  coordinate can be made 0), and the total angle by which the vector has been rotated from the initial position after  $i$  rotation is indicated by the parameter  $Z_{i+1}$  and  $Z_0$  is defined as 0. See equations 16 through 19. To obtain equation 16, when  $y_i$  is positive, the rotation is clockwise in the next iteration. When it is negative, it is counter-clockwise.

$$\delta_i = -\text{sgn}(y_i) \quad (16)$$

$$x_{i+1} = x_i - \delta_i y_i 2^{-i} \quad (17)$$

$$y_{i+1} = y_i + \delta_i x_i 2^{-i} \quad (18)$$

$$z_{i+1} = z_i - \delta_i \tan^{-1}(2^{-i}) \quad (19)$$

As  $i$  becomes large,  $y_i$  goes to 0 and  $x_{fin}$ , the magnitude of the vector after  $N$  iterations and  $Z_{fin}$ , the angle of the vector are obtained as:

$$x_{fin} = \frac{x_n}{\prod_{i=1}^N \cos \theta_i} \quad (20)$$

$$z_{fin} = \tan\left(\frac{y_0}{x_0}\right)^{-1} \quad (21)$$

### ***CORDIC as a Universal Demodulator***

If the carrier frequency, amplitude, and phase of the received signal are  $f_i$ ,  $2b(t)$ , and  $(t)$  respectively, then the received signal  $r(t)$  is given by:

$$r(t) = 2b(t) \sin(f_i t + (t)) \quad (22)$$

One approach to demodulating the signal is generating the in-phase  $I(t)$  signal and quadrature signal  $Q(t)$  given by:

$$I(t) = b(t) \{ \sin[2(f_i - f_0)t + (t)] \} \quad (23)$$

$$Q(t) = b(t) \{ \cos[2(f_i - f_0)t + (t)] \} \quad (24)$$

using a local oscillator of frequency  $f_o$  with a known initial phase.

The in-phase and quadrature signals can be fed to the two inputs ( $x_{in}, y_{in}$ ) of the CORDIC operated in rotation mode. The magnitude of the vector  $x_{fin}$  gives the demodulated signal corresponding to amplitude modulation as:

$$x_{fin} = \sqrt{I^2(t) + Q^2(t)} = b(t) \quad (25)$$

Similarly,  $Z_{fin}$ , the angle of the vector, gives the phase shift introduced into the carrier and is given by:

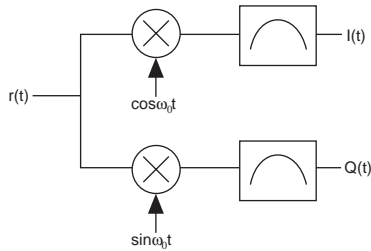
$$Z_{fin} = \tan\left(\frac{Q(t)}{I(t)}\right)^{-1} = \tan\left(\frac{\sin[2\Pi(f_i - f_o)t + \Theta(t)]}{\cos[2\Pi(f_i - f_o)t + \Theta(t)]}\right)^{-1} = (f_i - f_o)t + \Theta(t) \quad (26)$$

Differentiating equation 26, the instantaneous frequency of the carrier can be found.

## Generating In-Phase and Quadrature Signals Using Quadrature Mixers

Using the quadrature mixers shown in Figure 2, I (t) and Q (t) can be generated.

**Figure 2. Generating In-Phase and Quadrature Signals**



In this scheme, the input signal is divided into two in-phase paths and the local oscillator signals applied to the two mixers are 90-degree out of phase. The outputs of the two mixers are:

$$V_{if1} = 2b(t)\sin(2f_i t + (t)) \cos(2f_o t) \quad (27)$$

$$= b(t)\{\sin[2(f_i - f_o)t + (t)] + \sin[2(f_i + f_o)t + (t)]\}$$

$$v_{if2} = 2b(t) \sin(2f_i t + (t)) \sin 2f_o t \quad (28)$$

$$= b(t)\{\cos[2(f_i - f_o)t + (t)] - \cos[2(f_i + f_o)t + (t)]\} \quad (29)$$

The desired I and Q components are obtained using low-pass filters, which filter out the high-frequency signals represented by the  $f_i + f_o$  terms in equations 27 and 29.

A special sampling scheme generates the I and Q components without using quadrature mixers. In this case, if the local oscillator frequency is  $f_o$ ,  $r(t)$  is sampled at a rate of  $f_s = 4f_o$ . If  $V_{if1}$ ,  $V_{if2}$ , and  $r(t)$  are sampled at a rate of  $f_s = 4f_o$ , then:

$$t = nTs = n/4f_o \quad (30)$$

$$2tf_0 = 2(nTs)f_0 = 2n/4 = n/2 \quad (31)$$

If we substitute equation 31 in equations 1, 27, and 29, the output sequence at different sampling instants are as shown in Table 2. Table 2 shows the samples of the received signal, in-phase channel, and quadrature channel signals at different sampling instants for  $f_0 = f_i$ .

**Table 2. Signal Samples (Sample at  $t = t_n = nTs$ )**

Form=	0	1	2	3	4
R(t)	$b(0)\sin(\Theta)$	$b(t_1)\sin(2\Pi f_{it1} + \Theta)$	$-b(t_2)\sin(2\Pi f_{it2} + \Theta)$	$b(t_3)\sin(2\Pi f_{it3} + \Theta)$	$b(t_4)\sin(2\Pi f_{it4} + \Theta)$
V <sub>if1</sub>	$b(0)\sin(\Theta)$	0	$-b(t_2)\sin(2\Pi f_{it2} + \Theta)$	0	$b(t_4)\sin(2\Pi f_{it4} + \Theta)$
V <sub>if2</sub>	0	$b(t_1)\sin(2\Pi f_{it1} + \Theta)$	0	$-b(t_3)\sin(2\Pi f_{it3} + \Theta)$	0

From Table 1, we can verify that the in-phase and quadrature components can be generated by alternately passing  $r(t)$  to one of the two channels and inserting zeros alternately to each channel as shown in Figure 3. We obtain the desired I and Q components using low-pass filters that filter out the high-frequency signals represented by the  $f_i + f_0$  terms in equation 29. Let us denote the value of  $r(t)$ ,  $I(t)$ ,  $Q(t)$  at  $t = nTs$  as  $r(n)$ ,  $I(n)$ , and  $Q(n)$ , respectively. Then,  $I(n)$  and  $Q(n)$  can be written as:

$$I(n) = r(0), 0, -r(2), 0, r(4), \dots$$

$$Q(n) = 0, r(1), 0, -r(3), 0, r(5), \dots$$

What happens if the local oscillator frequency  $f_0$  is the same as the input frequency  $f_i$ ? In this case,  $f_s$  can be  $4f_0$ . Table 3 shows  $r(t)$ ,  $I(t)$ , and  $Q(t)$  at various sampling instants for  $f_0 = f_i$ .

**Table 3.  $r(t)$ ,  $I(t)$  and  $Q(t)$  ((Sample at  $t = t_n = nTs$ ))**

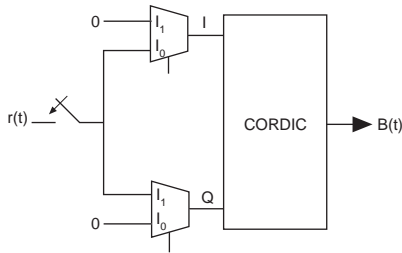
For n =	0	1	2	3	4
R(t)	$b(0)\sin(\Theta)$	$b(t_1)\cos(\Theta)$	$-b(t_2)\sin(\Theta)$	$-b(t_3)\cos(\Theta)$	$b(t_4)\sin(\Theta)$
V <sub>if1</sub>	$b(0)\sin(\Theta)$	0	$-b(t_2)\sin(\Theta)$	0	$b(t_4)\sin(\Theta)$
V <sub>if2</sub>	0	$b(t_1)\cos(\Theta)$	0	$-b(t_3)\cos(\Theta)$	0

Therefore, we can generate the in-phase and quadrature components without using mixers, at the expense of a higher sampling rate.

## Demodulator Implementation

Two mutually complimentary clocks are fed to the selection lines of both multiplexers operated at four times the IF signal as defined by our special sampling theorem. I and Q are given as the x and y inputs to the CORDIC, which is operating in vector mode. The demodulated signal can be fed to the PC and displayed. See Figure 3.

**Figure 3. Generating In-Phase and Quadrature Signals Using Special Sampling Scheme and CORDIC**



## Advantages of Using the Nios II Processor

We chose the Nios II processor for the following reasons:

- Based on the Nios II processor, we can enhance system performance by adjusting the Avalon® switch fabric.
- The customizable Altera Nios II processor has high performance and supports flexible product development at low cost.
- The Nios II processor's customizable instruction set can accommodate complicated arithmetic operations. Additionally, it can accelerate algorithm processing, which provides faster execution than implementing these operations in software.
- A configurable design enhances system performance.
- The  $\mu$ C/OS-II and RTOS with the Nios II IDE are very user friendly.
- Implementing the processor, peripherals, memory, and I/O interface in a single FPGA reduces the total system cost.
- We could implement the system rapidly. We could go from the original concept design to system implementation in a short time with the Nios II processor. Additionally, we could easily upgrade the hardware and software on site. This flexibility allows us to design products in-line with the latest specifications and equipped with new features.
- The Nios II processor can be customized and reconfigured. For example, it supports three processor cores, peripherals, the Avalon switch fabric, custom instructions, and hardware acceleration. All of these functions can be implemented using commonly available Altera FPGAs.
- Using IP optimized for the FPGA architecture, we can redesign standard functions easily, rapidly customize hardware peripherals, focus on design partitioning, and improve our design knowledge.
- Integrated development kits, the Quartus® II software, SOPC Builder, ModelSim®-Altera software, and SignalTap® II embedded logic analyzer provide a complete set of test and debug tools for hardware design. With the Nios II IDE, it is possible to simplify software design and all software development tasks, such as program editing and debugging.



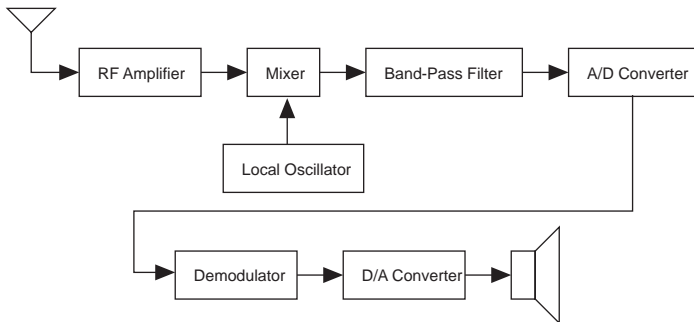
## Design Architecture

This section describes our design architecture.

### Hardware Design

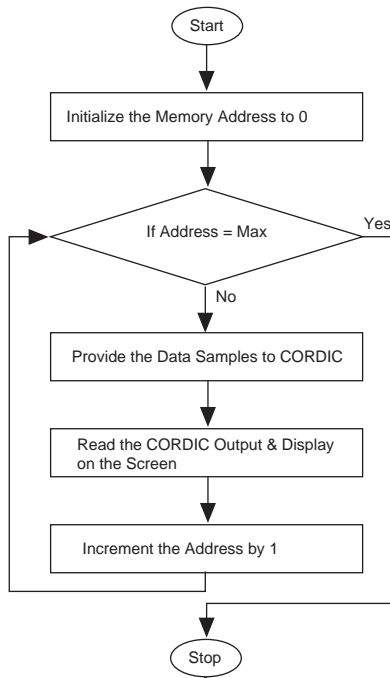
A signal is received from an antenna and is fed to the RF amplifier. Its output is mixed with the local oscillator signal in the mixer. All frequency components, except IF (455 KHz), are filtered out by the band-pass filter (BPF). The IF signal is digitized by the analog-to-digital converter (ADC) and is fed to the demodulator that is implemented on the FPGA. The demodulator reconstructs the message signal and feeds it to the digital-to-analog converter (DAC), which drives a loudspeaker. Figure 4 shows the SDR block diagram.

**Figure 4. SDR Block Diagram**



### Software Flow Chart

The demodulator output is called by a Nios II custom instruction. The demodulator output is available in the Nios II processor after it has been plotted in MATLAB. Figure 5 shows the software flow chart.

**Figure 5. Software Flow Chart**

## Design Description

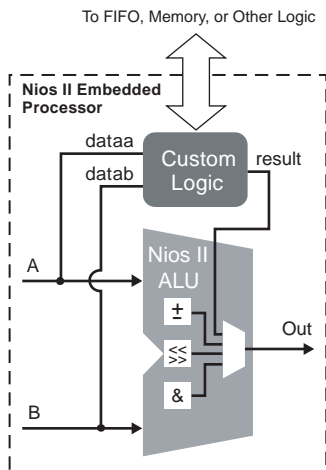
The SDR implementation involves two design phases:

- Frequency translation
- Message recovery

We implemented the first phase using the MATLAB software and the second phase in an FPGA using Altera tools. The amplitude modulated signal is generated and digitized in the Simulink (MATLAB) software before being stored in RAM. These data values are fed to the demodulator, which is called by the Nios II processor using a custom instruction. The RAM address is incremented for every call.

The custom instruction consists of two essential elements (see Figure 6):

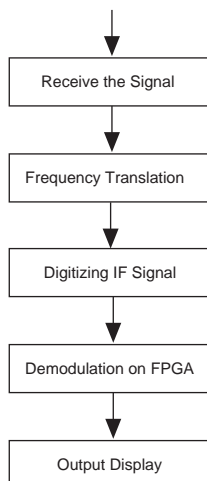
- *Custom logic block*—Hardware that performs the user-defined operation. The Nios II processor can include up to five user-defined custom logic blocks. These blocks become part of the Nios II microprocessor's arithmetic logic unit (ALU).
- *Software macro*—Allows the system designer to access the custom logic from the software code.

**Figure 6. Adding Custom Logic to the Nios II ALU**

The demodulator is based on a special sampling theorem and the CORDIC algorithm. Modulated samples are split into in-phase and quadrature components with multipliers. These components are fed as x and y inputs of the rectangular-to-polar coordinate converter (cordic hardware). The rectangular-to-polar converter, RAM, and multiplexers are implemented using Verilog HDL. The Verilog HDL demodulator implementation block, along with M4K blocks that contain the sampled AM signal values, is downloaded into the FPGA.

In a software-defined radio, the amplitude modulation (AM), frequency modulation (FM), phase modulation (PM), phase shift keying (PSK), and frequency shift keying (FSK) modulation types and the local oscillator frequency can be varied using software. The variable local oscillator frequency can be implemented using CORDIC and a phase accumulator; however, the CORDIC algorithm cannot be used to generate high-frequency signals. Instead, we use programmable logic (multiplexers) to generate the high-frequency signals. The stable signal frequencies, on the order of tens of MHz, can be generated efficiently using the Quartus II LogicLock™ feature and programmable logic.

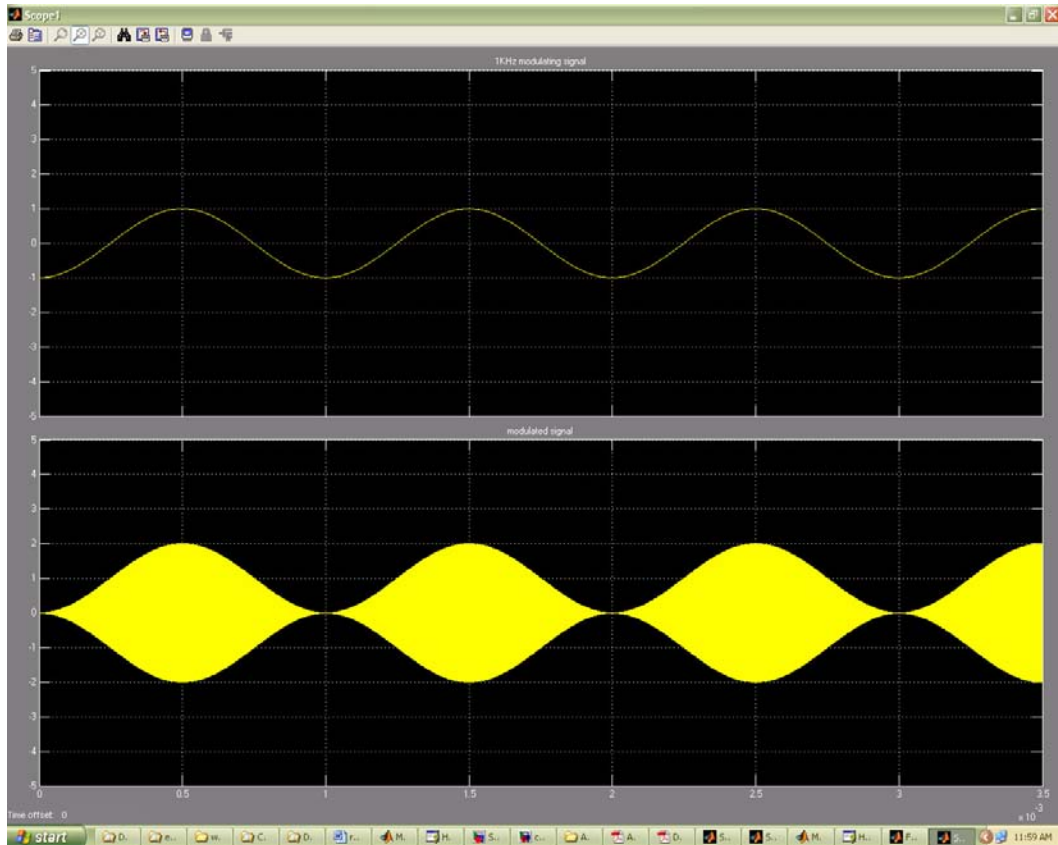
Figure 7 shows the design flow.

**Figure 7. Design Flow**

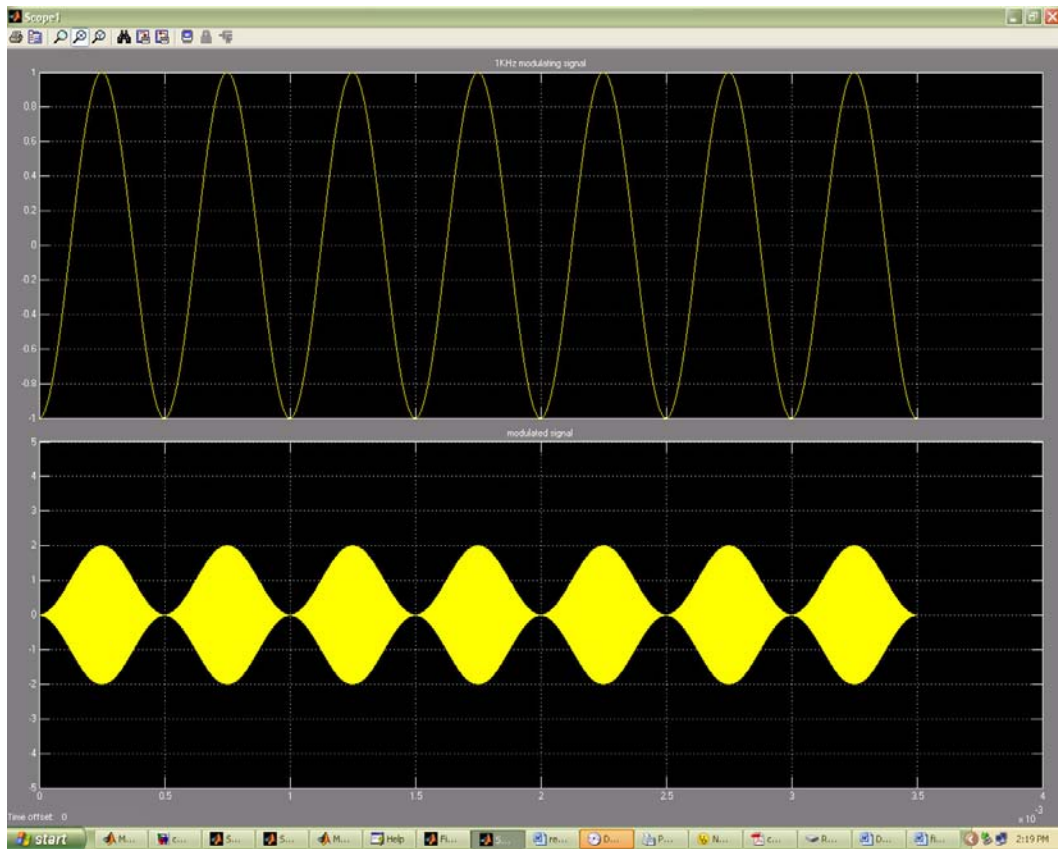
## Experiments and Results

To test the design, we applied two amplitude modulated signals to the design (a carrier of 1 MHz is modulated with two message signals of 1 KHz and 2 KHz, respectively). Figures 8 and 9 show the two modulated signals.

**Figure 8. Amplitude Modulated Waveforms ( $f_c = 1\text{ MHz}$ ,  $f_m = 1\text{ KHz}$ )**

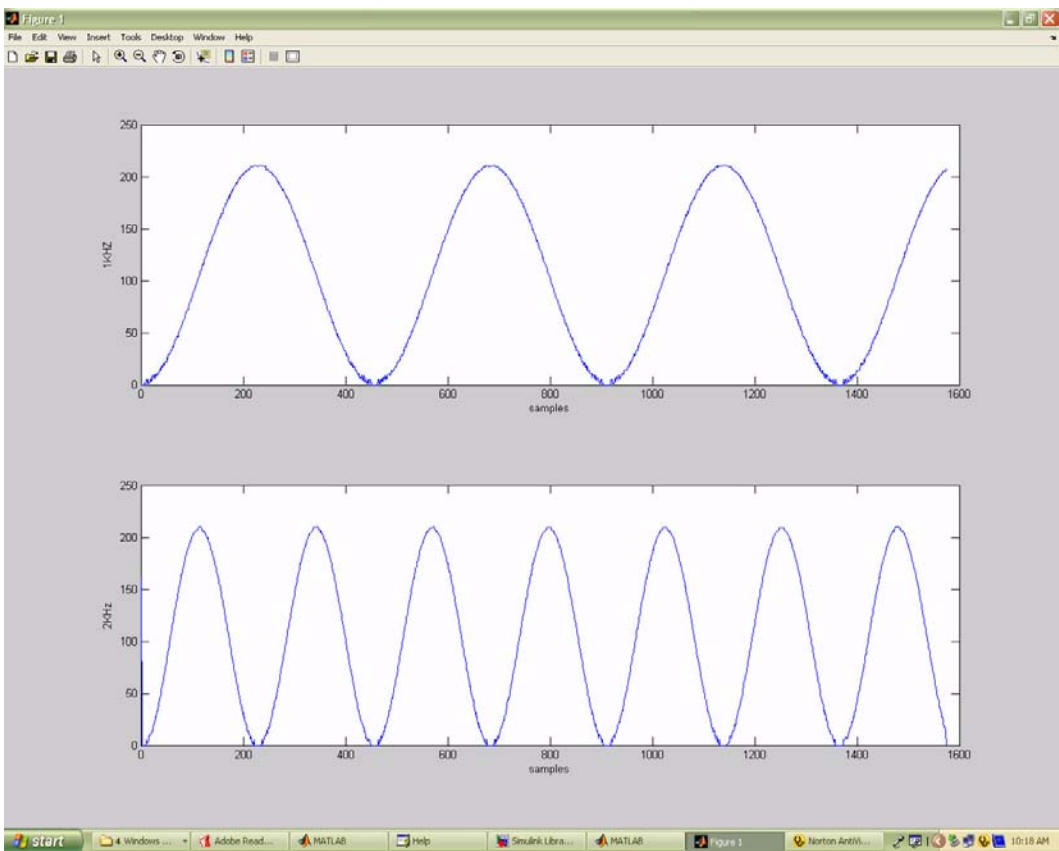


**Figure 9. Amplitude Modulated Waveforms ( $f_c = 1\text{ MHz}$ ,  $f_m = 2\text{ KHz}$ )**



The design generated perfectly reconstructed (i.e., demodulated) message signals from the two modulated signals, as shown in Figure 10.

Figure 10. 1 KHz and 2 KHz Demodulated Waveforms



Performance Parameters

Table 4 shows the design’s resource usage.

Table 4. Design Resource Usage

Parameter	Value
Device	EP2C35F672C6 (Cyclone II)
Tool	Quartus II version 5.1
Total logic elements (LEs)	3,154 of an available 33,216 (9%)
Total memory bits	181,760 of an available 483,840 (38%)
Total number of phase-locked loops (PLLs)	One quarter (25%)
Total registers	1,860

## Design Features

Our design has the following features:

- Implementing a demodulator for real-time applications involves mathematical operations instructed by the CORDIC algorithm. We efficiently implemented our design using SOPC concepts.
- Because the Nios II processor can be reconfigured, we can enhance the design in the future to meet the requirements of real-time applications with very low development costs.
- The Nios II processor enables optimum hardware/software development by executing more computationally intensive tasks in hardware and the remaining tasks in software.
- The Nios II embedded processor delivers good price/performance and lowers technical development hurdles. We use the Nios II processor to implement the processor, peripherals, memory, and I/O interface on a single FPGA, which helped to reduce the total system cost. Additionally, Altera's comprehensive development tools and optimized IP functions reduced our software development cost, letting us focus more attention on the design details of the software-defined radio.

## Future Development

We plan to develop our design in the following areas in the future:

- Add various modulation techniques, such as FM, amplitude shift keying (ASK), PSK, and FSK.
- Add support for variable frequency transmission, which is required for military applications.
- Enhance the design such that multiple software modules implementing different standards on the same system co-exist. This support will allow dynamic system reconfiguration by simply selecting the appropriate software module.

## Conclusion

We thank Altera for having the contest and acknowledge their support when we had design problems. During the design we learned that the Nios II processor complies with the emerging trend of industrial technology software such as hardware design. Using the Nios II processor, we reduced development costs, material costs, and turnaround times, improving our competitiveness. The Nios II processor and its development platform provide flexibility and an enhanced, robust system. Using custom instructions with the Nios II processor is an added advantage for design customization. On the whole, using SOPC concepts allowed us to create a more flexible, dynamically reconfigurable, and computationally intensive implementation.

## References

G. Seetharaman, B. Venkataramani, V. Amudha, Anurag Saundattikar, “System on chip implementation of 2D DWT using lifting scheme,” *Procedures of the International Asia and South Pacific Conference on Embedded SOC's (ASPICES 2005)*, (July 5 - 8, 2005), Bangalore.

Walter Tuttlebee, “Software Defined Radio,” John Wiley & Sons Ltd., (2004).

Jeffrey H. Reed, “Software Radio,” Pearson Education Pte Ltd., India, (2002).

James Tsui, “Digital Techniques for Wide Band Transmission,” Artech House Publishers, (1995).

Ray Andraka, “A survey of cordic algorithms for FPGA based computers,” *International Symposium on Field Programmable Gate Arrays 1998 Proceedings*, (1998): 191 - 200.