Digital Beam Forming Using Software Defined Radio Based Adaptive Algorithm

P. Ragasudha, B. R. Vikram, K. Sridhar

***Abstract*— This paper presents the development of an Adaptive Digital Beam Forming (ADBF) application on Software Defined Radio Platform using Open Source GNU Radio software. Adaptive beam formers for sensor arrays are widely used in RADAR, SONAR and communications applications. This is to increase the directivity of the sensor system to the target, while suppressing the interfering signals having a direction of arrival different from that of a desired signal. Array beam forming techniques can yield multiple beams that are simultaneously available. The beams can be made to have controlled beam width or high gain and low side lobe levels. Beam-forming techniques dynamically adjust the array pattern to optimize some characteristic of the received signal. Antenna arrays using beam-forming techniques can reject interfering signals having a direction of arrival different from that of desired signal. The principal reason of interest is their ability to automatically steer nulls into undesired sources of interferences, thereby reducing output noise and enhancing the detection of desired signal. Beam forming and beam scanning are generally accomplished by phasing the feed to each element of an array so that signals received from all the elements will be in phase in particular direction. Digital beam forming is thus a powerful technique for boosting the antenna performance. The work reported in this paper is purely a software based approach where all the waveform-specific processing is implemented on host CPU. The results supporting the presented work are furnished in this paper.**

***Index Terms*—Adaptive Digital Beam Forming, Field Programmable Gate Array, GNU Radio Software, Software Defined Radio.**

I. INTRODUCTION

Beam forming technique utilizes an array of sensor elements to focus a receiver channel on a specific Signal Of Interest (SOI). Historically, it has been at the heart of RADAR and SONAR signal processing. RADAR beam-forming techniques are used for target detection in the presence of ground clutter and jamming signals. It is used to steer the beam to a particular direction without having to mechanical steer the antenna element. The main goal of the beam forming technique is directional transmission and reception. This can be achieved by combining the elements of the array in two ways. One of the way is the signal arrives from single and particular direction combat with effective interferences. The other way is the signal arrives from other direction compromising with the destructive interferences. So, one can receive power from (direct the radiation towards) the desired direction (with less interference) and nullifying the interfering signal. It improves the Signal to Noise Ratio (SNR) and leads a better signal estimation.

Many wireless communication applications and the other signal intelligences can get benefit from beam forming techniques. A beam forming approach is regarded as a vital solution to the challenge of increasing spectral efficiency and improving the performance of wireless communication system. The advantage includes, allowing for an increased capacity of a communications network through the use of Space Division Multiple Access (SDMA) techniques. Since a beam former can steer the look direction toward the SOI, this frees the carrier frequency for use by other resources. Also, as the beam former is focused in a particular direction, the antenna sensitivity can be increased for a better SNR, a factor that is especially important when receiving weak signals. Both reception and transmission ranges can be significantly increased with beam forming. Additionally, beam forming techniques provide reduced probability of interception of secure transmissions. Finally, signal interference is reduced due to the ability to reject interfering signals.

The basic beam forming system is as shown in Fig. 1. The beam forming is achieved by the use of an array of sensors such as antenna, hydrophones and so on. The signal originating far away from the sensor array can be modeled as a plane wave. The signal received by each sensor element is a phase shifted version of the signal received by the other sensor elements. Finally an N-element beam forming system is capable of forming up to N beams. For antenna array beam forming the Least Mean Square (LMS) algorithm, one of the most popular adaptive signal processing techniques is adopted, because of its simplicity and robustness.

In this paper, Adaptive Digital Beam Forming (ADBF) has been implemented using GNU Radio, which is a free collection of signal processing blocks that can be used for Radio Frequency (RF) real-time applications. It can act as a stand-alone software package or as a backend to a hardware device. Applications can be developed by using either Python or C++. As the proposed application is time sensitive and involves implementation of feedback, it is implemented in C++ and connected in Python. Currently, GNU Radio is the primary software platform supporting the drivers for the USRP on a personal computer. The USRP's software defined parameters (e.g. center frequency, PGA gain, interpolation factor, decimation factor, and some transmit and receive path multiplex options) can only be controlled using Python. The host device can be any kind of signal processing device that can be connected via USB 2.0 (e.g., any kind of signal

P. Ragasudha, M.Tech. student, Department of Electronics and Communication Engineering, Vijay Rural Engineering College, Nizamabad, Telangana, INDIA. Mobile No. 9030490670.

Dr. B. R. Vikram, Principal, Vijay Rural Engineering College, Nizamabad, Telangana, INDIA. Mobile No. 9848884300.

K. Sridhar, Assistant Professor, Department of Electronics and Communication Engineering, Vijay Rural Engineering College, Nizamabad, Telangana, INDIA. Mobile No. 9848955100,.

processing system that includes components like General Purpose Processors (GPP), Digital Signal Processors (DSP), Field Programmable Gate Arrays (FPGA), or Application Specific Integrated Circuits (ASIC), etc.). But when the C++ block is tested on host computer of speed 3GHz and 3GB RAM the time for beam formation is 400 µsec. As RADAR applications needs the beam to be formed with in hundreds of nano-seconds, the beam former is implemented on FPGA (cyclone II).

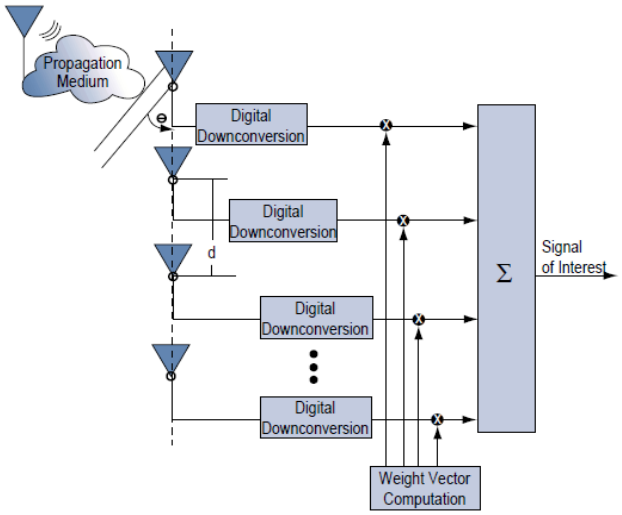

Fig. 1. Basic beam forming system.

Rest of the paper is divided in to four sections. Second section consists of the explanation for Least Mean Square algorithm. Third section introduces the software defined radio platform. Results are presented and discussed in the fourth section and the conclusions are presented in the last section.

II. LEAST MEAN SQUARE ALGORITHM

A. *LMS Algorithm formulation*

As shown in Fig. 2 the outputs of the individual sensors are linearly combined after being scaled using corresponding weights. This is done such that the antenna array pattern is optimized to have maximum possible gain in the direction of the desired signal and nulls in the direction of the interferers. The weights here will be computed using LMS algorithm based on Minimum Squared Error (MSE) criterion. Therefore the spatial filtering problem involves estimation of signal from the received signal (i.e. the array output) by minimizing the error between the reference signal, which closely matches or has some extent of correlation with the desired signal estimate and the beam former output y(t) (equal to w(t)). This is a classical Weiner filtering problem for which the solution can be iteratively found using the LMS algorithm.

From the method of steepest descent, the weight vector equation is given by

$$w(n+1) = w(n) + \frac{1}{2}\mu[-\nabla(E\{e^2(n)\})] \quad (1)$$

where μ is the step-size parameter and controls the convergence characteristics of the LMS algorithm; $e^2(n)$ is the mean square error between the beam former output y(n) and the reference signal which is given by,

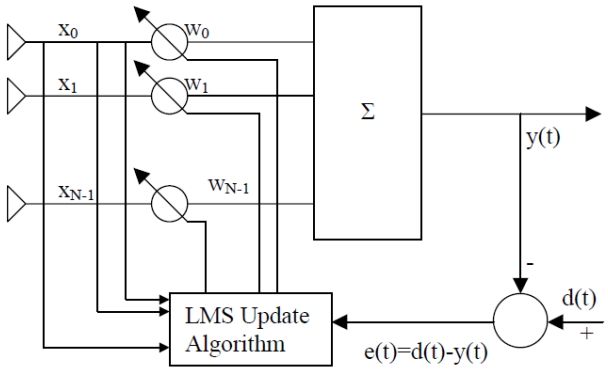

Fig. 2. Adaptive beam forming using LMS algorithm.

$$e^2(n) = [d^*(n) - w^h x(n)]^2 \quad (2)$$

The gradient vector in the above weight update equation can be computed as

$$\nabla_w(E\{e^2(n)\}) = -2r + 2Rw(n) \quad (3)$$

In the method of steepest descent the biggest problem is the computation involved in finding the values r and R matrices in real time. The LMS algorithm on the other hand simplifies this by using the instantaneous values of covariance matrices r and R instead of their actual values i.e.,

$$R(n) = x(n)x^h(n) \quad (4)$$

$$R(n) = d^*(n)x(n) \quad (5)$$

Therefore the weight update can be given by the following equation,

$$w(n+1) = w(n) + \mu x(n)[d^*(n) - x^h(n)w(n)]$$
$$\Rightarrow w(n+1) = w(n) + \mu x(n)e^*(n) \quad (6)$$

The LMS algorithm is initiated with an arbitrary value w(0) for the weight vector at n=0. The successive corrections of the weight vector eventually leads to the minimum value of the mean squared error.

Therefore the LMS algorithm can be summarized in following equations;

Output, $y(n) = w^h x(n)$ (7)

Error, $e(n) = d^*(n) - y(n)$ (8)

Weight, $w(n+1) = w(n) + \mu x(n)e^*(n)$ (9)

B. *Convergence and stability of LMS algorithm*

The LMS algorithm initiated with some arbitrary value for the weight vector is seen to converge and stay stable for

$$0 < \mu < 1/\lambda_{max} \quad (10)$$

where λ_{max} is the largest Eigen value of the correlation matrix R.

The convergence of the algorithm is inversely proportional to the Eigen value spread of the correlation matrix R. When the Eigen values of R are widespread, convergence may be slow. The Eigen value spread of the correlation matrix is estimated by computing the ratio of the largest Eigen value to the smallest Eigen value of the matrix. If µ is chosen to be

very small then the algorithm converges very slowly. A large value of μ may lead to a faster convergence but may be less stable around the minimum value. One of the literatures also provides an upper bound for μ based on several approximations as μ <= 1/(3trace(R)). Also, the signal flow and architecture for implementing LMS algorithm is shown in Figs. 3 and 4 respectively.

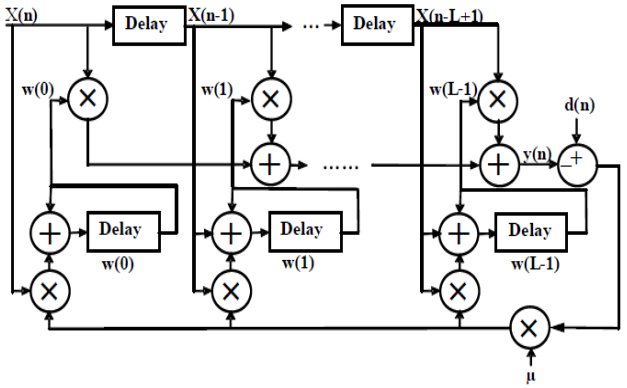

Fig. 3. Signal flow for implementation of LMS algorithm.

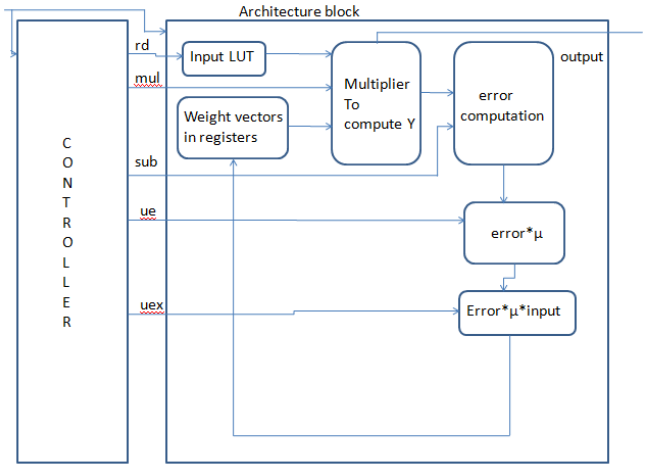

Fig. 3. Architecture for implementing LMS algorithm.

III. SOFTWARE DEFINED RADIO PLATFORM

Software radio is the technique of getting code as close to the antenna as possible. It turns radio hardware problems into software problems. In principle, a universal applicable hardware serves as interface between the baseband and the RF. The waveform of a transmitted signal is fully generated through software, as well as a received signal is fully processed and demodulated within software algorithms. In SDR, the processing power required for signal processing is sourced out to a universal host (PC). An important benefit for industrial usage is the possibility to change complete processing stacks or modulation schemes just with a software update. This also saves costs and time for new hardware developments.

A universal SDR structure with the specific software (GNU Radio) and hardware (USRP) is shown in Fig. 3.

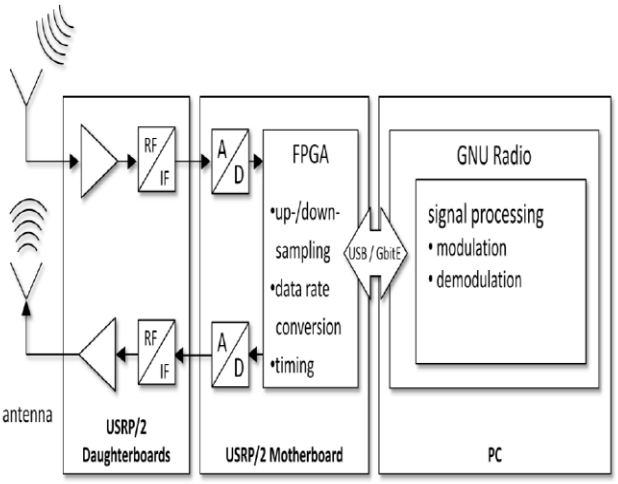

Fig. 3. Structure of SDR.

In Fig. 3, the Software-Defined Radio (SDR) structure is divided into three blocks. The left one builds the RF frontend of the hardware which serves as interface to the analog RF domain. In the second block, the intelligence of the hardware part is implemented, forming the interface between the digital and the analog world. In the third block, the whole signal processing is done - fully designed in software.

Getting more detailed, the interface to the analog world is given as mentioned on left side of Fig. 3 An analog RF signal can be received or transmitted over antennas, or can also be directly connected via SMA connectors to the SMA ports of RF frontend called daughter boards. The upper path (arrow towards the daughterboard) marks the receive path (Rx), the lower path describes the transmit path (Tx). Both paths can operate autonomously. The possible operation frequency range is very modular (from DC to 5.9 GHz), depending on the available daughter boards for USRP/2. Daughter boards form the RF frontend of USRP/2 and are connected to the USRP/2 motherboard.

In USRP2 motherboard, an Analog to Digital Converter (ADC) samples the received signal and converts it to digital values depending on the ADCs dynamic range of 14 bit. The digital sample values are transferred to the FPGA and processed with Digital Down Converters (DDC) to meet exactly the requested output frequency and sample rate. The schematic of a DDC is shown in Fig. 4.

The digitized samples from ADC are mixed down to the desired IF by being multiplied with a sine respectively cosine function resulting in the 'I' and 'Q' path. The frequency is generated with a Numerically-Controlled Oscillator (NCO) which synthesizes a discrete-time, discrete-amplitude waveform within the FPGA. Via the used NCO, very rapid frequency hopping is feasible.

Afterwards a decimation of the sampling rate is performed by an arbitrary decimation factor N. The sampling rate (fs) divided by N results in the output sample rate, sent to host. In transmit path, the same procedure is done vice versa using Digital Up Converters (DUC) and Digital to Analog Converters (DAC).

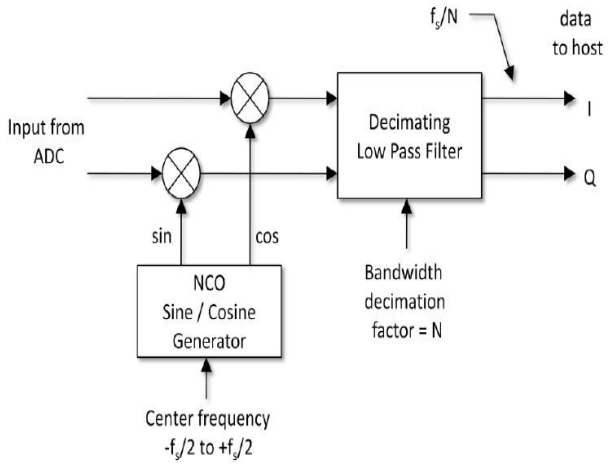

Fig. 4. Schematic of DDC.

The FPGA also supports time dependent applications which e.g. use TDMA. A free running internal counter allows incoming samples to be sent in strictly definable timestamps. Regarding Fig. 4, data sampled by the FPGA are sent to the host by USB or Gigabit Ethernet respectively what is used – USRP or URSP2. Connected to the host computer, the GNU Radio framework controls the further signal processing capabilities. GNU Radio is an open source framework, providing various pre-assembled signal processing blocks for waveform creation and analysis in software radio development. In the GNU Radio environment, Python and C++ are used as main programming languages as well as a signal flow application called GNU Radio Companion (GRC).

A. Implementation of ADBF using GNU Radio

The Software-defined radios (SDRs) provide researchers with a powerful and flexible wireless communications experimentation platform. GNU Radio is the most popular open-source software toolkit for deploying. Every SDR is comprised of software and hardware. In this project GNU Radio software coupled with Universal Software Radio Peripheral (USRP) hardware is used. In GNU Radio, C++ blocks perform specific signal processing tasks, while Python applications connect the blocks together to form a functional software radio. For example, a basic transmitter can be implemented by using Python to connect the following C++ blocks (which already exist in the GNU Radio software library) together: modulator, mixer, and amplifier.

Each block specifies its input and output requirements, both in number and type. For example, the gr_add_cc block adds two complex input streams and copies the results onto one complex output stream. Blocks are generally implemented in C++ for computational efficiency.

After writing a new block, a process is needed to expose these C++ blocks for use by Python scripts. GNU Radio uses the Simplified Wrapper and Interface Generator (SWIG), to generate the necessary components to make C++ blocks accessible from Python. From the standpoint of Python applications, each block consumes its input stream(s), performs a specific task, and generates output stream(s). As long as the connections between blocks are compatible, there is no restriction to how many blocks can be chained together. A single output stream can connect to multiple input streams, but multiple outputs cannot connect to a single input due to ambiguity. A multiplexer can be used in such a situation by interleaving many inputs onto a single output.

In summary, the stages of block creation in GNU Radio are the following:

1. Implementation of blocks in C++, (.h) and (.cc) files.
2. Creation of SWIG interfaces between C++/Python, the (.i) file.
3. Installation of blocks into a shared library.
4. Usage of blocks in an application (Python), the (.py) file.

In order to select a sufficient word length and a range for the various variables in the system, we should design the proposed adaptive beam former accordingly. So, a MATLAB simulation model of fixed-point adaptive beam former is used in order to get the variables' representation with minimum possible word length and sufficient partitioning between integer part and decimal fraction, which results from model tests with varying word lengths and decimal fractions. A bad choice of a decimal fraction produces more quantization noise. Table 1 shows the numerical ranges of the input signals, the output signal, and the coefficients obtained by simulation. Based on the ranges obtained, the locations of the bits for the integer parts and those for the decimal fractions of each variable are obtained.

Table 1: Numerical ranges of inputs and signals.

Variable name	x	d	error	w
Data Range	(-10,10)	(-10,10)	(-10,10)	(-0.3,0.3)

For the input signal in the range from -10 to 10, its whole scale may use 4 bits, so the decimal fraction is located between the 10^{th} bit and the 1^{st} bit. According to this location method, the best precision can be obtained within the given dynamic range. The selected number system is shown in Tables 2, 3, 4 and 5.

Table 2: Data format of x and μ.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Sign	Integer				Decimal fraction										

Table 3: Data format of d.

31	30-24	23-0
Sign	Integer	Decimal fraction

Table 4: Data format of y and e.

47	46-39	28-0
Sign	Integer	Decimal fraction

Table 5: Data format of w.

31	30-26	27-0
Sign	Integer	Decimal fraction

IV. RESULTS DISCUSSION

Fixed beam forming has been implemented for different angles. Fixed weight vectors are computed in MATLAB and presented in Table 6. Figs. 5 to 7 show the plots from MATLAB for signal arriving angles of 10º, 30 º and 50 º.

GNU Radio Companion has all the necessary modules to implement LMS algorithm, but adaptive algorithms cannot be implemented with existing modules because python (used to create flow graphs) doesn't support feedback loops. The alternative to create loops in GNU Radio is to include loops in C++ signal processing block. So, LMS algorithm has been implemented in C++ as a module. Inputs to the module and outputs from the module are vector format. Output is written to a text file and plotted using GNU octave tool.

Table 6: weight vectors used for fixed beam forming.

Direction of arrival angle	Direction of Nulls	Weight vectors
10	**40, 60**	W0=1.0000 W1=0.6086 + 0.1406i W2= 0.7602 + 0.5114i W3=0.2374 + 0.8241i W4=-0.0209 + 0.9436i
30	**60, 90**	W0=1.0000 W1= 0.1284 + 1.0178i W2= -1.1418 + 0.1338i W3= -0.0235 - 1.1666i W4= 1.0766 + 0.0134i
50	**10, 90**	W0=1.0000 W1=-0.5610 + 1.0491i W2=-0.6987 - 0.7409i W3=0.8185 + 0.2250i W4=-1.3704 + 0.3990i

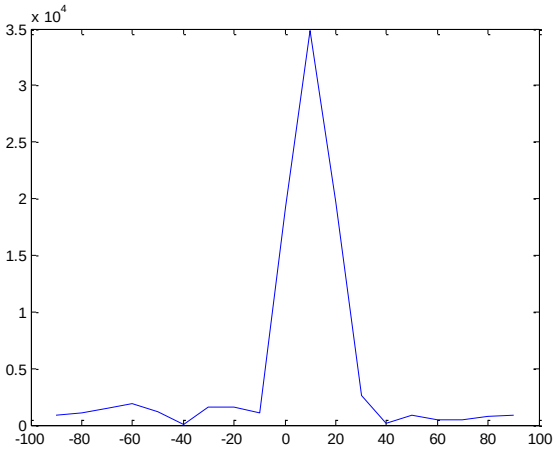

Fig. 5. MATLAB plot for signal arriving angle of 10º

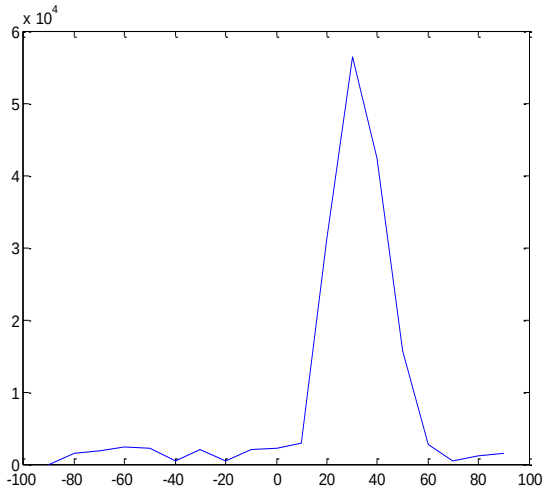

Fig. 6. MATLAB plot for signal arriving angle of 30º

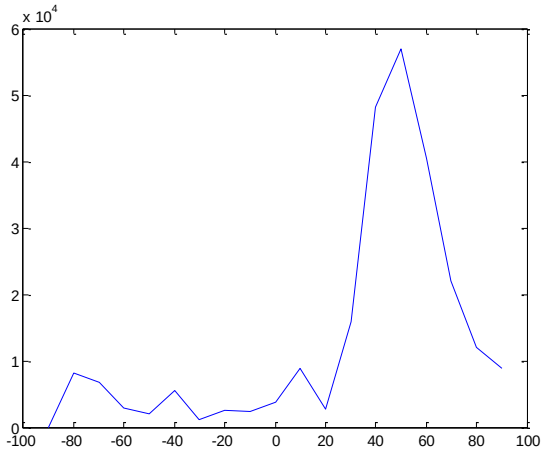

Fig. 7. MATLAB plot for signal arriving angle of 50º

Below (Fig. 8) is the screenshot after running "make check". Make check will run qa_howto.py (top level python script) which interconnects the inputs to signal processing blocks and outputs from signal processing block to a text file.

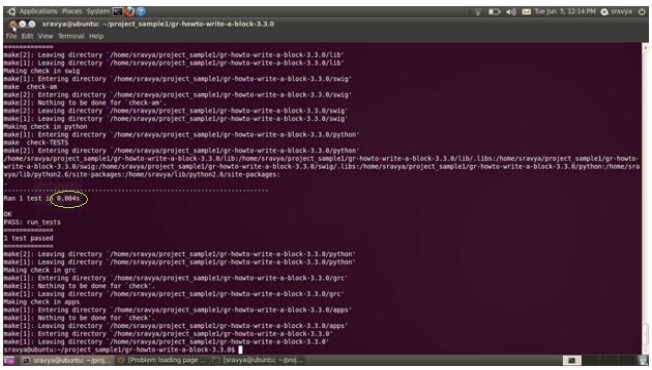

Fig. 8. Screen shot after running make_check (GNU).

The time taken to run the test is 0.004s. Here number of samples in input vector file are 100. So it is taking 4 ms to run 100 samples. LMS algorithm takes 10 iterations to converge. The total time for convergence of algorithm is 400 µs. Here the execution is being carried out by CPU (2 GHz processor and 3 GB RAM), so the convergence time varies depending on the number of process handling by CPU.

GNU Octave is a high-level interpreted language, primarily intended for numerical computations. It provides capabilities for the numerical solution of linear and nonlinear problems, and for performing other numerical experiments. Octave has extensive tools for solving common numerical linear algebra problems, finding the roots of nonlinear equations, integrating ordinary functions, manipulating polynomials, and integrating ordinary differential and differential-algebraic equations. It is easily extensible and customizable via user-defined functions written in Octave's own language, or using dynamically loaded modules written in C++, C, Fortran, or other languages. It also provides extensive graphics capabilities for data visualization and manipulation. Octave is normally used through its interactive command line interface, but it can also be used to write non-interactive programs. The Octave language is quite similar to MATLAB so that most programs are easily portable.

The weight vectors are routed to a text file and beam is plotted using GNU Octave and is presented in Fig. 10.

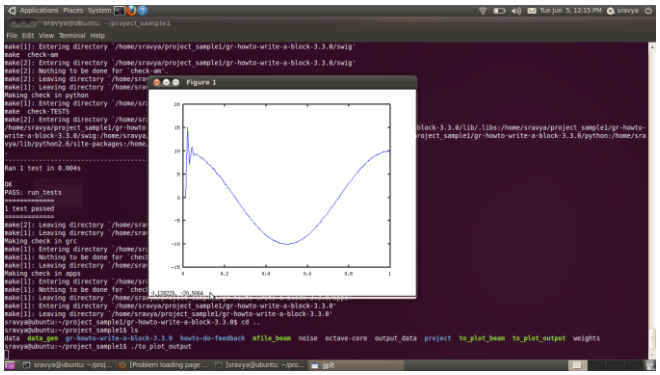

Fig. 9. Output signal plotted using GNU octave.

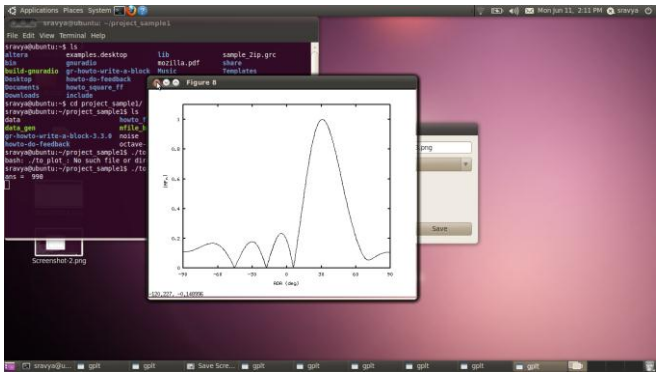

Fig. 10. Output beam plotted using GNU Octave.

V. CONCLUSION

Adaptive digital beam forming has been implemented in GNU Radio, which is free software defined radio. The languages used for programming in GNU Radio are C++ and Python. Flow graphs are generally created using python but as python doesn't support feedback/loops, Adaptive beam forming is implemented using C++. For time critical applications, this whole process can be implemented on USRP FPGA after Down Conversion and resulting samples can be sent to PC for further processing. The results supporting the above discussed work are also presented.

REFERENCES

[1] Taniza Roy , Meena D. and LGM Prakasam, "FPGA based Digital Beam Forming for Radars", Radar conference, 2009 IEEE, pp.1-5, May 2009.

[2] Asit Kumar Subudhi, Biswajit Mishra , Mihir Narayan Mohanty, "VLSI Design and implementation of Adaptive filter using LMS Algorithm", International Journal of Computer & Communication Technology (IJCCT), Volume-2, Issue-VI, 2011.

[3] K. R. Rekha , Dr B. S. Nagabushan and Dr K.R..Nataraj, "FPGA Implementation of NLMS Algorithm for Identification of unknown system", International Journal of Engineering Science and Technology, Vol. 2(11), 2010, 6391-6407.

[4] Tian Lan, Jinlin Zhang, "FPGA Implementation of an Adaptive Noise Canceller", 2008 International Symposiums on Information Processing

[5] Mohamed Salah, Abdel-Halim Zekry, Mohammed Kamel, "FPGA Implementation of LMS Adaptive Filter", 28th National Radio Science Conference (NRSC 2011).

[6] Usha Mallaparapu, K. Nalini and P. Ganesh, "Non-blind adaptive beam forming algorithms for smart antennas", IJRRAS 6 (4), March 2011.

[7] Lilja P., Saarnisaari H, "Robust adaptive beam forming in software defined radio with adaptive diagonal loading", Military communications Conference, 2005, MILCOM 2005, IEEE, 2596 - 2601 Vol. 4.

[8] Fertig L B, "Statistical performance of the MVDR beam forming in the presence of diagonal loading", Proceedings of 2010 IEEE Sensor Array and Multichannel Signal Processing workshop, pp. 77-81, 2010.

[9] Kogon S M, "Eigen vectors, diagonal loading and white noise gain constraints for robust adaptive beam forming", proceedings of 37th Asilomar conference on Signals, Systems and Computers, pp-1853-1857, 2003.

[10] Gershman A B, "Robust adaptive beam forming: An overview of recent trends and advances in the field", proceedings of international conference on Antenna Theory and techniques, pp. 30-35, 2003.

[11] GNU Radio User Manual by Firas Abbas.

[12] Matt Ettus, Ettus Research LLC, USRP User's and developers' guide.

[13] G. Liang, W. B. Gong, H. J. Liu, and J. P. Yu, "Development of 61-channel digital beam-forming (DBF) transmitter array for mobile satellite communication", Progress In Electro-magnetics Research, PIER 97, 177-195, 2009.