Flexible M-QAM Modulator and ScalableFFT/IFFT: Design and Implementation for a SDRMulti-carrier Transmitter with Link Adaptation

Lal, Shradha; Kaur Warar, Shabanpreet; Popp, Andreas; Le Moullec, Yannick

Published in:
Proceedings of the 5th Karlsruhe Workshop on Software Radios

Publication date:
2008

Document Version
Peer reviewed version

Flexible M-QAM Modulator and Scalable FFT/IFFT: Design and Implementation for a SDR Multi-carrier Transmitter with Link Adaptation

Shradha Lal, Shabanpreet Kaur Warar, Andreas Popp, Yannick Le Moullec
Center for Software Defined Radio, Fr. Bajers Vej 7, A3, Aalborg University, DK-9220 Aalborg Ø, Denmark
{shradha|shaban|anp|ylm}@es.aau.dk

Abstract-- **In this work a flexible Orthogonal Frequency Division Multiplexing (OFDM) transmitter chain is considered with the main focus on three blocks namely: link adaptation, modulation and Inverse Fast Fourier Transform (IFFT) complying with the two communication standards IEEE 802.11a (WiFi) and 802.16d (WiMax). The Xtreme Development Kit-IV with Xilinx Virtex-4 FPGA is used as the target hardware platform. Since both IEEE 802.11a and 802.16d build upon OFDM-modulation, Multi-level Quadrature Amplitude Modulation (M-QAM) modulator and IFFT operation were chosen as the common blocks for the implementation. Based on the parameterization of these standards, flexibility and scalability are introduced in the modulator and IFFT, respectively, so that the same structure may be switched by parameters according to the requirements of both the standards.**

Index Terms—**Scalable, flexible, parameterizable, M-QAM, modulation, IFFT.**

I. Introduction

The worldwide growth in wireless communication technologies and equipments has fueled the existence of multiple standards which has led to the growing interest in creating a single architecture for wireless devices. With the advancement in the digital signal processing solutions, the software driven flexibility and ability to change air-interface baseband subsystems through software began to appear [1]. Thus, the term Software Defined Radio (SDR) was coined as one solution to converge the diverse communication standards into one terminal to enable the end users to move freely anywhere and anytime while using seamless services [2].

The existing multi-carrier technologies can bring the vision of a common hardware platform to reality [3]. This is because the recent trends in wireless communications are high throughput, seamless mobility and a wide variety of multimedia applications. These require significant increase in spectral efficiency and very high data rates and as an answer, OFDM as a multi-carrier modulation technique has been recognized especially appropriate to achieve high data rates in delay-dispersive environments. OFDM is being adopted by most of the existing and next generation wireless communication standards. M-QAM modulation and IFFT being the core blocks of an OFDM transmitter in IEEE 802.16d and IEEE 802.11a, are considered for implementation in this work. The system parameters considered are modulation format, number of subcarriers, system bandwidth and carrier frequency as shown in Table 1.

Table 1: System specifications of the design to be implemented.

Parameters	**Specifications**
System bandwidth	20 MHz (common)
Carrier frequency	2.4 GHz (common)
Number of subcarriers (FFT points)	64 (802.11a) 256 (802.16d)
Modulation schemes	BPSK, QPSK, 16-QAM, 64-QAM (802.11a) BPSK, QPSK, 16-QAM, 64-QAM,256-QAM (802.16d)

System reconfiguration may be implemented utilizing three different techniques, namely; (i) Exchange of complete radio module, (ii) Exchange of (a) single component(s) within a module, (iii) Using parameterized radio modules [1]. In these cases (as in this paper), where similarities among communication standards are predominant (as can be seen in Table 1), parameterized radio modules can create a structure that may be switched by parameters to realize the different standards. Methods for implementing parameterizable Baseband modulator and IFFT architecture are proposed here.

The M-QAM modulator is flexible to accommodate the modulation schemes specified by both standards. The inherent similarity between QAM constellations is exploited to avoid redundant hardware usage. The implemented architecture utilizes a single LUT for all the modulation schemes. A simple concept for minimizing the redundant hardware by exploiting the commonality in the M-QAM constellations is proposed. This implementation is named as *flexible implementation* of modulation schemes. A single LUT serves as a common hardware for all QAMs in this implementation.

The two standards use the modulation schemes: BPSK, QPSK, 16-QAM, 64-QAM and 256-QAM. Gray code is used for the mapping of symbols so that the bits in error are as small as possible. A new flexible modulator is implemented and compared with the modulator where each modulation scheme has a dedicated LUT. We mention the latter one as *conventional implementation* throughout the paper. In the implementation process, each subcarrier is assigned a different modulation scheme according to the link adaptation algorithms. The data bits are given as input to the modulator block serially.

It is a general practice to implement the modulator of OFDM transmitter using as many LUTs as there are modulation schemes. Although the constellation diagram of different level modulation is represented in hierarchal fashion, no implementation which exploits a single LUT for all the constellation points has been discussed in the literature, despite the fact that illustration has been given of a smaller-level QAM forming a subset of a higher-level QAM [4-8]. However, and to the best of our knowledge, no actual implementation of M-QAM modulator in terms of hierarchy has been presented. Figure 1 shows conventional implementation of the modulator using five modulation schemes and five LUTs.

An OFDM transmitter requires the implementation of an Inverse Discrete Fourier Transform (IDFT). IDFT converts the signal from frequency domain to time domain and is calculated by the following equation:

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k) W_N^{-kn}, \qquad 0 \le n \le N-1$$

where, N is number of samples, the size of IFFT, *x(n)* is vector of N-real time samples, *X(k)* is size N complex vector and W_N is the twiddle factor or the phase factor. The computation of an IDFT is usually performed using an IFFT algorithm due to its low complexity and thereby easier hardware implementation.

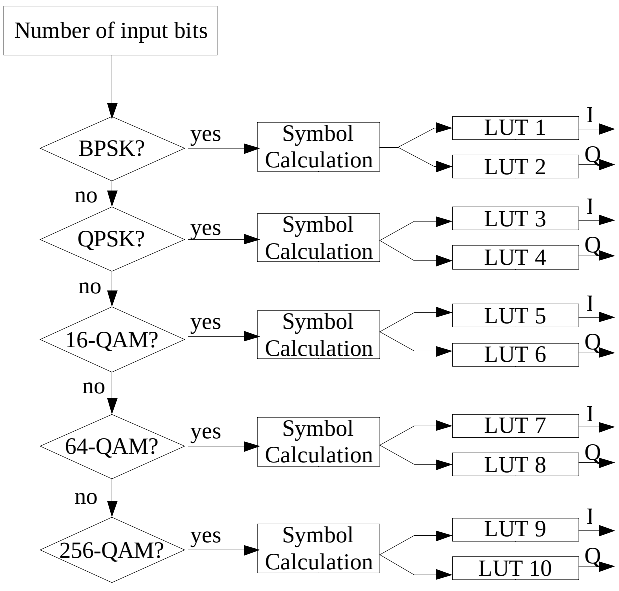


Figure 1: The conventional method for the implementation of M-QAM modulator utilizing ten LUTs for five modulation schemes.

This paper presents the implementation of a IDFT with radix-4 FFT algorithm in the decimation-in-frequency (DIF) for 64-point and 256-point IFFT used for OFDM modulation in WLAN (IEEE 802.11a) and WiMAX (IEEE 802.16d), respectively. The radix-4 IFFT algorithm is selected based on the trade-off between complexity, resource requirement and computational delay. However, the choice of radix-4 limits the IFFT size to a power of 4 (i.e., $N = 4^v$).

The IFFT block is designed in such a way that the same structure can be used by both standards for the generation of subcarriers. A modular structure of IFFT is developed based on radix-4 FFT algorithm. The implemented IFFT is made scalable to accommodate any number of IFFT points provided that the number is a power of 4.

The implementation of blocks is done using the Handel-C language. However, the design is simulated using MATLAB. The paper is organized as follows: Section II presents the design and implementation details of the modulator and the IFFT blocks. The results obtained are included in Section III. The paper concludes in Section IV.

II. II. DESIGN AND IMPLEMENTATION OF THE MODULATOR AND THE IFFT

A. *M-QAM Modulator*

Modulation scheme plays a very important role in hardware implementation, so the choice of modulation scheme should have a balanced trade-off between its design complexity and signal performance. This paper claims to devise a new form of implementation for the M-QAM modulator. With the help of BER plot, it has been shown that *flexible implementation* gives almost the same performance as the conventional way of implementing the modulator but at the benefit of less hardware usage and simple computations.

In conventional implementation, individual LUTs are used for different modulation schemes. The binary bit stream of length *n* passed through a *n*-bit serial-to-parallel converter to determine a symbol. Each symbol point in the constellation has an individual bit code. By an observation, the symbols for different modulation schemes are determined as follows:

- In QPSK, each symbol is determined as *b1**2 + *b0* and then mapping is done using a LUT.
- Symbol for 16-QAM is calculated as *b3**8 + *b2**4 + *b1**2 + *b0*.
- Symbol for 64-QAM is calculated as *b5**32 + *b4**16 + *b3**8 + *b2**4 +*b1**2 + b0.
- Symbol for 256-QAM is calculated as *b7**128 + *b6**64 + *b5**32 + *b4**16 + *b3**8 + *b2**4 + *b1**2 + *b0*.

From the above calculations, it can be generalized that for M-QAM, ($\log_2$M - 1) multiplications and ($\log_2$M - 1) additions are required for determining each symbol.

Normalization factor: For each modulation scheme, I and Q values are scaled by normalization factor. It is done so that the average power of all points in the constellation equals unity. Table 2 lists the normalization factor for BPSK, QPSK, 16-QAM, 64-QAM and 256-QAM [5, 9].

Table 2: Normalization factor for all modulation schemes.

Modulation Scheme	Normalization factor
BPSK	1
QPSK	1/√2
16-QAM	1/√10
64-QAM	1/√42
256-QAM	1/√170

During the implementation of modulator blocks, it can be noticed that different modulation schemes can be represented as sub-set of higher modulation schemes as shown in Figure 2. Using two points from QPSK constellations, i.e. {1+j, -1+j}, can produce BPSK. Thus, the BPSK can also be theoretically designed in an I-Q modulation format [4]. QPSK can also be called 4-QAM, because it uses four constellation points in I-Q modulation. In Figure 2, BPSK, QPSK, 16-QAM are represented as a sub-set of 64-QAM. In turn 64-QAM forms a sub-set of even higher level QAM. It is quite obvious from the figure that same set of constellation points can be used for lower-level modulation schemes.

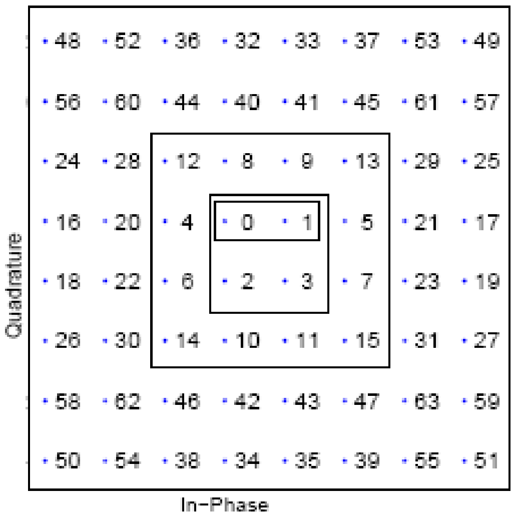


Figure 2: The constellation diagram for the flexible M-QAM showing that the lower-order modulation schemes are a subset of 64-QAM.

For example, 16-QAM can use the constellation points of 64-QAM and of higher level QAMs. Thus, unlike conventional implementation which uses different LUTs for every modulation scheme, this implementation uses one LUT for all the modulation schemes. A LUT stores all the possible values which a particular modulation scheme may require to map the bits into symbols to be transmitted. After the mapping from LUT, I and Q are scaled by normalization factor as in conventional modulation. To further reduce the computation, I and Q calculations are treated separately. The values of I is determined by even incoming bits and Q is determined by odd bits. For example:

- If there are two input bits for QPSK *b1* and *b0*. The value of I is directly determined by *b0* and Q is determined by *b1*.
- For 16-QAM, I can be determined by *b2**2 + *b0* and Q can be determined by *b3**2 + *b1*.

Generalizing from the determination of I and Q for 4-QAM and 16-QAM, the total number of multiplications/additions required for mapping of each symbol is $\log_2 M/2$ -1.

The number of multiplications/additions required for flexible implementation when compared with the conventional implementation is always less. For 16-QAM, the number of multiplications/additions required in conventional implementation is $\log_2 M$ - 1= $\log_2 16$ -1 = 3, however, in this implementation only $\log_2$ (M/2) -1 = log2 (16/2) -1 = 2 multiplications and 2 additions are required. When dealing with a large number of subcarriers, this reduction in calculations saves (1 multiply and 1 addition operation for 16-QAM) significant hardware resources being utilized at the time of computations.

Figure 3 shows the block diagram for flexible implementation. Five modulation schemes have been implemented. The rhombuses show the modulation schemes. The 'No. of bits' define the modulation scheme being selected. For example, if modulation scheme for a subcarrier is 16-QAM, the I and Q signals are calculated by taking two odd and two even bits. I for 16-QAM is obtained by shifting the bit *b2* by one positions left and then adding it to *b0* and Q for 16-QAM is obtained by shifting the bit *b3* by one positions left and then adding it to *b1*. Whichever a scheme is selected, it accesses the same LUT for the required symbol mapping. A left shift by 1 multiplies the value by 2. Thus, the multiply operation is avoided by the use of shift operation in flexible M-QAM implementation.

Table 3 (a) and (b) show the common LUT used for all modulation schemes where the even bits determine the I signal and the odd bits determine the Q signal, the only difference between the LUT for Q signal and I signal is the negative sign. So, LUT for I can be used for Q and can be obtained by negating the LUT for I. Thus, a single LUT stores the values for all modulation schemes in flexible implementation. For flexible modulation, the normalization factor of BPSK modulation is $1/\sqrt{2}$ and there is no change in normalization factor for other modulation schemes.

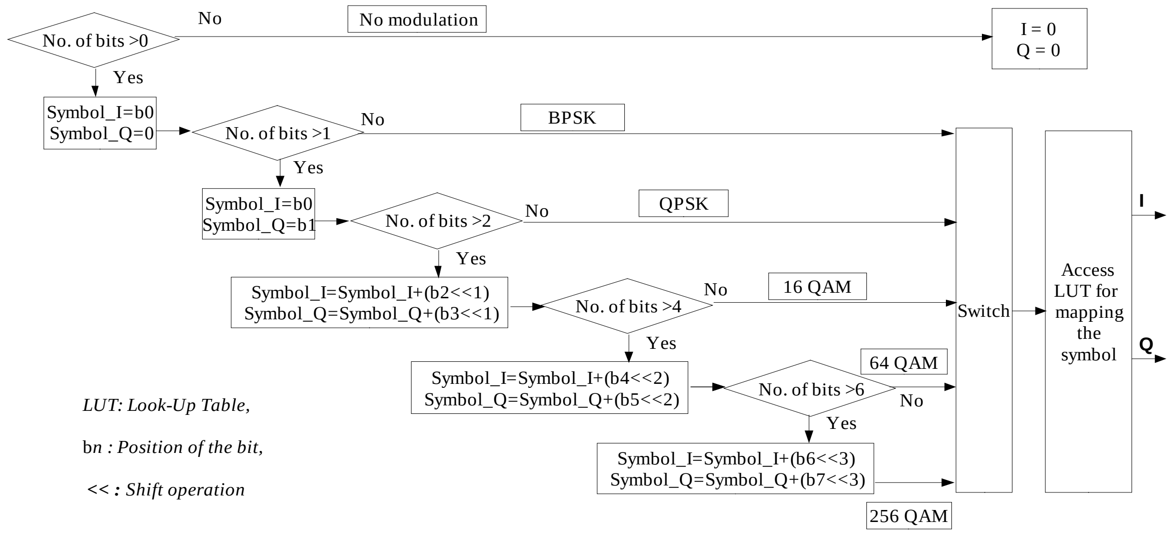


Figure 3: The logic for the implementation of flexible M-QAM modulator which utilizes one LUT for five modulation schemes.

Table 3: LUT for In-phase and Quadrature-phase signals in flexible M-QAM modulator.

(a)

b6	b4	b2	b0	I
0	0	0	0	-1
0	0	0	1	1
0	0	1	0	-3
0	0	1	1	3
0	1	0	0	-7
0	1	0	1	7
0	1	1	0	-5
0	1	1	1	5
1	0	0	0	-11
1	0	0	1	11
1	0	1	0	-13
1	0	1	1	13
1	1	0	0	-9
1	1	0	1	9
1	1	1	0	-15
1	1	1	1	15

(b)

b7	b5	b3	b1	Q
0	0	0	0	1
0	0	0	1	-1
0	0	1	0	3
0	0	1	1	-3
0	1	0	0	7
0	1	0	1	-7
0	1	1	0	5
0	1	1	1	-5
1	0	0	0	11
1	0	0	1	-11
1	0	1	0	13
1	0	1	1	-13
1	1	0	0	9
1	1	0	1	-9
1	1	1	0	15
1	1	1	1	-15

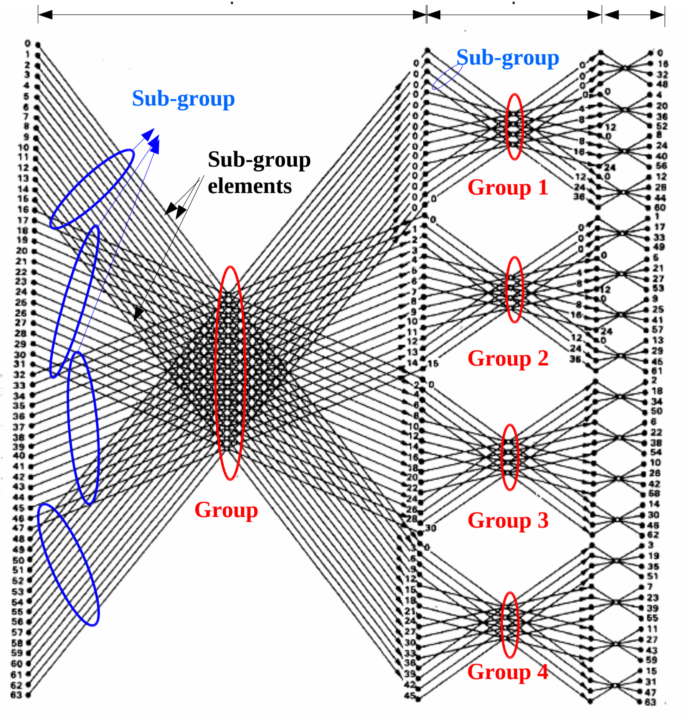


Figure 4: 64-point decimation-in-frequency radix-4 IFFT showing stages, groups, subgroups and elements.

B. IFFT

IFFT consists of butterflies organized in different stages. In each stage, the butterflies are divided into groups. Each group is further divided into 4 sub-groups. The terms stage, group, sub-group and element are illustrated in Figure 4, where a complete 64-point radix- 4 DIF IFFT is shown. The number of stages in N-point IFFT is log_4N.

64-point IFFT computation is done in $log_4 64 = 3$ stages. The twiddle factors, the number of groups in each stage and the butterflies in each group are summarized in Table 4.

Table 4: Initializations required for N-point radix-4 IFFT

Stage	Twiddle factor for 64-point IFFT	Twiddle factor for 256-point IFFT
1	$W_{64}^{0n,\,1n,\,2n,\,3n}$; n=0 to 15	$W_{256}^{0n,\,1n,\,2n,\,3n}$; n=0 to 63
2	$W_{64}^{0n,\,4n,\,8n,\,12n}$; n=0 to 3	$W_{256}^{0n,\,4n,\,8n,\,12n}$; n=0 to 15
3	$W_{64}^{0n,\,16n,\,32n,\,48n}$; n=0	$W_{256}^{0n,\,16n,\,32n,\,48n}$; n=0 to 3
4	-	$W_{256}^{0n,64n,128n,192n}$; n=0

It can be seen in Figure 4 that IFFT has an iterative nature. This provides scalability to IFFT which is exploited in this work. Table 5 shows the twiddle factors for each stage for IFFT-64 and IFFT-256. The twiddle factors for IFFT-64 are a sub-set of twiddle factors for IFFT-256 so, same twiddle factor table can be used for both the IFFTs.

Table 5: Twiddle factor table for IFFT-64 and IFFT-256.

Stage	1	2	3	(log2N)/2
Twiddle factor	$W_N^{0n,\,1n,\,2n,\,3n}$ n=0 to (N/4)-1	$W_N^{0n,\,4n,\,8n,\,12n}$ n=0 to (N/16)-1	$W_N^{0n,\,16n,\,32n,\,48n}$ n=0 to (N/64)-1	$W_N^{0n,\,(N/4)n,\,(N/2)n,\,(3N/4)n}$ n=0
Groups	1	4	16	N/4
Butterflies in each group	N/4	N/16	N/64	1

Also, all the stages can be generalized as one block due to their similarity of operation. This block is named as “IFFT stage” in the paper. The “IFFT stage” computes the IFFT calculations for a stage and the Look Up Tables (LUTs) for twiddle factors are initialized in IFFT stage. So, instead of implementing all the stages for both the IFFTs (256 and 64), only one “IFFT stage” is recursively used to calculate IFFT. Figure 5 illustrates the block diagram developed to implement “IFFT stage” which is enclosed in the dashed box in the figure.

Since, the twiddle factors for 64-IFFT are subset of 256-IFFT so, only one LUT for 256 point IFFT is initialized. The input to the stage is the IFFT size. Then, the inputs/elements are divided into groups and sub-groups. The radix-4 computation is done by taking the i^{th} element of each sub-group.

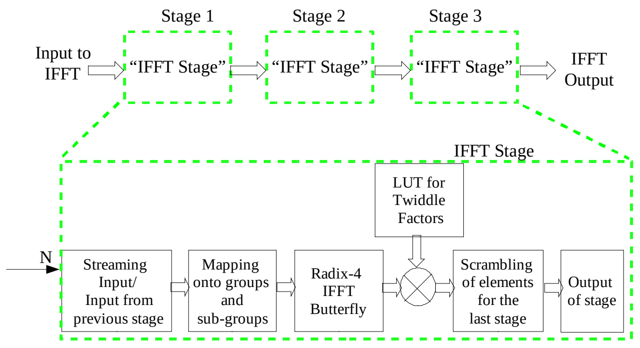
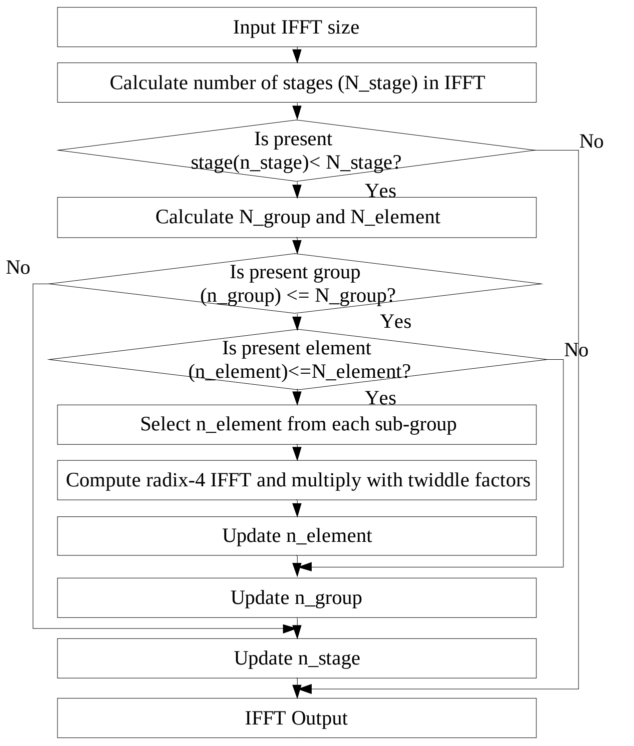


Figure 5: Representation of three stages of a 64-point IFFT in the terms of IFFT stage and the block diagram of IFFT stage.

This procedure is repeated for all the elements in sub-group and all the groups in a stage. The results from the radix-4 computation are multiplied with the twiddle factors and the output is given to the next stage.

An analogy of Figure 4 is shown in the term of "IFFT stages" in Figure 5. It is shown in Figure 5 that the same "IFFT stage" can be used for all the three stages in 64 point IFFT. This concept can be generalized and can be extended to all IFFT sizes. The algorithm used for IFFT computation is shown in Figure 6. In the algorithm, the size of the IFFT is the only input that is required. The twiddle factors for different stages are stored in LUTs. The algorithm consists of one radix-4 IFFT block and one multiplier for twiddle factor multiplication. With this the same implemented IFFT can be used for 64/256 point IFFT and any higher order IFFT, provided that the IFFT size is a power of 4.

The calculations as shown in Figure 6 are given as; Number of stages, N_stage = $\log_4 N$; Number of groups in each stage, N_group = $4^{\text{n\_stage}-1}$ and Number of elements in sub-group, N_element= N/(4*N_group). The algorithm iterates once for each stage in IFFT-64. Further, the iterations in each stage depend on the number of groups and the number of elements in each sub-group. The calculation for a group is done by taking an element from each sub-group, computing the radix-4 IFFT and multiplying with the twiddle factor. Similar is repeated for all the groups in a stage. The output from the first stage goes as input to the second stage and the algorithm iterates three times for IFFT-64. For 256 point IFFT, the algorithm will iterate four times as it has four stages.


Input IFFT size
Calculate number of stages (N_stage) in IFFT
Is present stage(n_stage)< N_stage?
No
Yes
Calculate N_group and N_element
No
Is present group (n_group) <= N_group?
Yes
Is present element (n_element)<=N_element?
No
Yes
Select n_element from each sub-group
Compute radix-4 IFFT and multiply with twiddle factors
Update n_element
Update n_group
Update n_stage
IFFT Output

N_stage=Number of stages in IFFT, n_stage=Present stage, N_group=Number of groups in the stage, n_group=Present group, N_element=Number of elements in sub-group, n_element=Present element

Figure 6: Flow diagram for the algorithm developed for the IFFT implementation.

During the implementation of the IFFT block following considerations are done:

- Fixed point implementation in Handel-C language is used to design the IFFT architecture.
- Shared hardware is used for fixed point operations such as multiplication, addition and subtraction. The sharing of hardware is done by making a function for each operation.
- The real and imaginary parts in FFT calculation are handled separately because in many cases they are swapped to obtain a complex number. Storing them separately saves the extra computation in generating a complex number.
- All the twiddle factors that are required at various stages for being multiplied to the data points are computed once and then stored in a look up table.

III. III. RESULTS

A. *M-QAM Modulator*

The results obtained by Handel-C implementation of conventional and flexible implementation are shown in Table 6. The table lists the average number of NAND gates, flip-flops and memory bits required in the implementations for 16 sub-carriers.

Table 6: Hardware required by LUTs in the conventional and the flexible implementations of M-QAM for 16 sub-carriers.

Implementation	**No. of LUTs**	**NAND Gates**	**Flip-Flops**	**Memory bits**	**Clock Cycles**
Conventional	10	17077	358	1704	353
Flexible	1	2571	149	80	406

Flexible implementation takes 12.5% more clock cycles than conventional implementation due to the extra steps used to calculate a symbol for higher order modulation schemes. Even when a modulation scheme used is higher than 256-QAM, the size of the LUT does not increase significantly. This is because for conventional M-QAM, two LUTs (one for I and one for Q) are required whose size is M*1 while for flexible M-QAM, one LUT of $\sqrt{M}$ is required. If the highest modulation scheme is 256-QAM, the size of LUTs for I and Q is 256*1 for conventional implementation while for flexible implementation of 256-QAM one 16*1 LUT is required. Similarly, the size of LUT is 1024*1 and 32*1 for conventional and flexible 1024-QAM respectively. The results obtained by conventional and flexible implementation of modulator using ISE 8.1i are shown in Table 7.

Table 7: ISE results for conventional and flexible M-QAM modulator

Logic Utilization	**Conventional**	**Flexible**
Number of 4 input LUTs	37	5
Number of occupied XtremeDSP slices	38	16
Total equivalent gate count for the design	694	238

In order to validate the signal quality of the modulator, a curve is plotted to see the bit error rate for both the types of implementation in MATLAB. Figure 7 shows BER versus SNR plot for 16-QAM modulators. The Additive White Gaussian Noise (AWGN) noise is added to the modulated signal and then the signal is demodulated to obtain the BER for the modulation scheme. A comparison is made between the flexible, conventional modulators along with the 16-QAM MATLAB modulator which is both gray-coded and non gray coded. The theoretical plot is a curve obtained from the in-built MATLAB instruction for the BER of QAM in AWGN channel is used as a reference. The BER plots for conventional and flexible implementation gives similar results to MATLAB 16-QAM modulator.

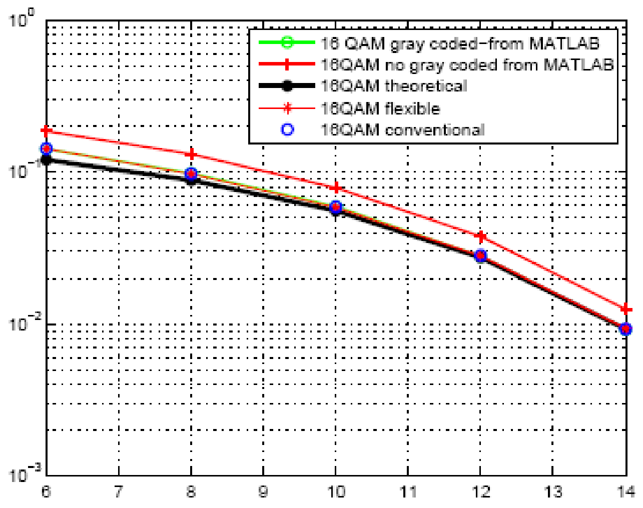


Figure 7: BER vs SNR curves comparing performance of conventional and flexible implementations of 16-QAM modulator with the theoretical 16-QAM.

B. *IFFT*

The implementation of 256-point IFFT takes 5.51 million NAND gates while the scalable implementation of 64 point and 256 point IFFT takes 1.35 million NAND gates. This results in 75% reduction in hardware usage by the use of scalable IFFT. The estimation of NAND gates required for the implementation of IFFT with 16 sub-carriers in DK4 Design Suite is 61297 and the estimated Flip Flops are 1490. The ISE implementation of 16-point IFFT occupies only 16% of the XtremeDSP slices in Virtex-4 FPGA.

In order to validate the design of IFFT, it is simulated in MATLAB. The output of the algorithm simulated in MATLAB is compared to the output of the in-built 'ifft' function in MATLAB. The real and imaginary parts of the output from algorithm for 256 point IFFT and the output of in-built 'ifft' function are compared. The difference between the two outputs is of the order of 10^{-16}. Also, validation of the implemented design of IFFT in Handel-C is done. Table 8 shows the result obtained from the testing of IFFT block for 4 inputs in Handel-C compared to the algorithm in MATLAB.

Table 8: Testing results of IFFT block by comparing outputs from Handel-C and MATLAB.

Input	MATLAB Output	Handel-C Output
1+j	0.25-0.25j	0.25-0.25j
1-j	0.25+0.75j	0.25+0.75j
0	0.25+0.75j	0.25+0.75j
-1-j	0.25-0.25j	0.25-0.25j

The testing is done by black box testing method. Same inputs are given to the algorithm in Handel-C and algorithm in MATLAB and the outputs are compared. It can be seen from the table, that the outputs are same which validates the design of scalable IFFT in Handel-C.

IV. CONCLUSIONS

This paper proves that the exploitation of inherent aspects of communication standards is a promising approach towards an efficient SDR implementation.

First of all, two implementation approaches (conventional and flexible) for a M-QAM modulator in a transmitter are discussed. It can be noticed that same performance is achieved at the benefit of less hardware resources (the flexible implementation takes 35 % of the conventional implementation). The proposed flexible implementation is advantageous in the sense that it requires less computation and also uses less hardware than the conventional implementation. Generally, in conventional implementation for BPSK, QPSK, 16-QAM, 64-QAM and 256-QAM modulation modes, ten LUTs (two for each mode, one LUT for I and one for Q) are used, whereas for the flexible implementation and for the same set of modulation modes, only one LUT (for both I and Q) for 256-QAM serves as the LUT for all lower modulation schemes.

Moreover, the implementation of a scalable IFFT for a flexible SDR platform is discussed for building a parameterizable IFFT/FFT architecture that implements the IFFT/FFT block in such a way that it can be used for any OFDM-based standard. A case-study illustrates our approach and shows that the hardware usage decreases by almost 25% (from 5.51 million to 1.35 million) NAND gates for the scalable, FPGA-based, implementation of a 64/256 points IFFT.

IV. REFERENCES:

[1]. Mitola J., "Cognitive Radio: An Integrated Agent Architecture for Software Defined Radio", Ph.D. dissertation, Royal Institute of Technology (KTH), 2000.

[2]. Tuttlebee W., Software Defined Radio Baseband Technology for 3G Handsets and Basestations, John Wiley and Sons, LTD, 2004, pp.338, ISBN: 0-470-86770-1.

[3]. Nassar C. R., Natarajan B., Wu Z., Wiegandt D., Zekavat S. A. and Shattil S., Multi-carrier Technologies for Wireless Communication, Kluwer Academic Publishers, 2002, ISBN: 0-7923-7618-8.

[4]. Muhammed I. R.,"Design and Implementation of an OFDM modem in FPGA for WLAN/WPAN",MSc Thesis, Aalborg University, January 2003.

[5]. "Constellation Mapper and Demapper for WiMAX", Altera Corporation, Application Notes 439, September 2006, version 1.0

[6]. Vitthaladevuni P. K. and Alouini M. S., "BER Computation of 4/M-QAM Hierarchical Constellations", IEEE Transactions on Broadcasting, Vol. 47, No. 3, September 2001.

[7]. Howald R. L., "QAM Bulks Up Once Again-Modulation to the Power of Ten", In Proceeding Manual and Collected Technical Papers, SCTE Cable-Tec Expo 2002, San Antonio, Tex.:http://broadband.motorola.com/ips/pdf/QAM.pdf

[8]. Soltanian A., "Coexistence of Ultra-Wideband System and IEEE 802.11a WLAN", NIST WCTG Report, April 2003 (Revised October 2003).

[9]. Li X., Liu N., Pei C., Yi K. and KouW., "Design and Analysis of High Speed WLAN Systems with Adaptive Margin Technology", International Conference on Parallel and Distributed Computing, Applications and Technologies, pp. 284-287, 2005.