Implementation of Input Data Buffering and Scheduling Methodology for 8 Parallel MDC FFT

Govinda Rao Locharla[1] Sudeendra Kumar K[2] Prof. K K Mahapatra[3] Prof. Samit Ari[4]
Department of Electronics and Communications Engineering,
National Institute of Technology, Rourkela.
Email: grlocharla09@gmail.com[1], kumar.sudeendra@gmail.com[2], kkm@nitrkl.ac.in[3], samit@nitrkl.ac.in[4]

Abstract— **The MIMO-OFDM improves the link budget, simplifies the receiver and decreases the peak to average power ratio (PAPR) of a transmitting terminal. It includes multiple antennas for transmission and reception. The buffering and scheduling methodology of the data arrived from multiple antennas is very critical in reducing memory size and power consumption. This paper presents the Data Buffering, Scheduling methodology and their implementation for an eight parallel variable length Multipath Delay Commutator (MDC) FFT processor suitable for IEEE 802.11ac compliant MIMO-OFDM systems. It starts with a mathematical model for an 8 parallel variable length MDC FFT to understand the requirement of input buffer. Input buffer is required to park eight streams of the samples received serially at baseband processor and to schedule the ordered data onto eight channels such that the multimode FFT processing can be done efficiently and correctly. Proposed buffer can handle the data in four different orders for FFT core while operating in 512/256/128/64 point mode respectively. Proposed design consists of register banks, which can simplify the routing and save power. Also, this design is free from address conflict that may occur with SDRAM. The design is synthesized using TSMC 65nm library. The area and dynamic power results are presented for the implementation.**

Keywords— MDC; OFDM; MIMO; IEEE 802.11ac; FFT

I. INTRODUCTION

Wireless communication technology today has greatly improved compared to the day Wi-Fi standards are introduced. However there is a break through with the introduction of IEEE 802.11ac standard. It is an upgrade standard to the IEEE 802.11n and offering higher speeds over wider bandwidths [1]. This can run at 5GHz and provides the data rate of 1.3Gbps and can also works on multi user, Multiple Inputs Multiple Outputs (MIMO). It also can support up to eight data streams [2]. MIMO devices increase the data throughput drastically. MIMO OFDM schemes provide high data rate with good reliability in wireless communications. In the applications like telepresence and voice, this standard is very suitable. In PHY layer design of such high rate wireless applications, the Orthogonal Frequency Division Multiplexing (OFDM) modulation has a crucial role, in which, the Fast Fourier transform (FFT) processor is a key component [3]. To handle multiple data streams of different lengths, processing blocks are duplicated to process the incoming data. To avoid duplication of processing blocks and redundancy, a proper design of system is need of a day. FFT plays a significant role in OFDM systems and the number of FFT processing elements grows linearly with the number of data streams. To avoid the redundancy, pipelined architectures are used widely in implementation of FFT/IFFT. Receiving, storing and scheduling the incoming data in an appropriate way is a challenging task. Without proper design or efficient implementation, FFT butterfly will be kept idle during memory read, write and data arrangement operations, which affects the performance of the total system. In pipelined schemes, single path delay feedback (SDF) and multipath delay commutator (MDC) are well known architectures in DSP design community [8]. Previous research works suggests that MDC has got several advantages like simple control path, efficient utilization of FFT cores, less area consumption in comparison with conventional designs [9][10]. It is evident from the literature that, FFT cores used in MIMO-OFDM duplicates the butterflies and memory with the number of data streams. To keep the control logic simple and for better utilization of resources, MDC is a preferred technique. The memory elements consume more area in MDC architecture. For example, 2048 point MDC FFT architecture requires 5112 words of memory [7].

MDC architecture offers simpler data flow control. For MIMO-OFDM FFT, MDC gives efficient utilization in terms of area and power when data streams are properly scheduled. For proper scheduling of data streams, designing an efficient input buffer is a research problem. The input buffer and scheduling block plays a crucial role in the success of MDC technique. The proper reception, saving and re-ordering of data is required for optimal utilization of resources. Hence, the buffering and scheduling methodology of the data coming from multiple antennas is very critical in reducing memory size and power consumption. In this paper, we proposed a mathematical model for an eight parallel multipath delay commutator based FFT in brief and it is mapped to the block diagram of the eight parallel FFT architecture which can support the variable lengths and throughput requirements of the FFT specified in IEEE 802.11ac standard [4].

From the above discussion, it is clear that the input buffer has a significant role in the MDC implementation. This work mainly focus on converting eight input streams into the format required to feed variable length pipelined MDC FFT core. The pipelined MDC FFT core is described in Section III. Earlier to our effort, a method for input buffering and scheduling is proposed in [5] and it is better suitable for MDC FFT. However, it is power hungry because, most of the registers in the bank are engaged in shifting. On the other hand, buffering and scheduling methodology proposed in [6] can handle the variable streams of the data, but it is only meant for fixed length of 256 point FFT processing.

The proposed buffering and scheduling technique is an improvement to the technique found in [7]. The proposed scheduling method in this paper is targeted for eight stream data rather four streams like in [7]. In addition, register banks are used in this work to handle the input data instead of SDRAM used in [7]. Therefore the proposed technique can simplify the routing and save power. Also, this design is free from address conflict, which may occur with SDRAM.

This paper is organized as follows. Section II briefly describes the mathematical model to derive the FFT architecture suitable for baseband processor in 8X8 MIMO OFDM applications. Section III introduced the idea of an eight parallel MDC FFT/IFFT architecture to address the need of proposed input buffering and the scheduling technique. Rest of this section focused on the methodology for buffering and scheduling of the input streams. In section IV, the design specifications are presented. Implementation is presented in Section V. Then, the conclusions are made in Section VI.

II. Algorithm for 8 Parallel MDC based FFT

Fourier Transform of a discrete sequence x(n) can be defined as,

$$X(k) = \sum_{n=0}^{N-1} x(n).W_N^{nk} \quad \text{where, } k = 0,1,2\ldots(N-1). \qquad (1)$$

In order to provide the FFT processing seamlessly for eight independent streams of the data in multiple modes, the basic FFT algorithm represented by equation (1) can be written as shown in equation (3).

Based on the common factor algorithm, variables ‘n’ & ‘k’ can be written as,

$$k = k_1 + 8 * k_2 + 64 * k_3 + 256 * k_4$$
$$\text{where}, 0 \le k_1, k_2 \le 7; 0 \le k_3 \le 3; 0 \le k_4 \le (N/256-1).$$

$$n = (N/8) * n_1 + (N/64) * n_2 + (N/256) * n_3 + n_4$$
$$\text{where, } 0 \le n_1, n_2 \le 7; 0 \le n_3 \le 3; 0 \le n_4 \le (N/256-1).$$

$$\therefore \quad W_N^{nk} = W_8^{n_1k_1}.W_{64}^{n_2k_1}.W_8^{n_2k_2}.W_{256}^{n_3(k_1+8k_2)}.W_4^{n_3k_3}$$
$$W_{512}^{n_4(k_1+8k_2+64k_3)}.W_2^{n_4k_4} \qquad (2)$$

Equation (2) can be written as,

$$X(k) = \sum_{n_1:0}^{7} \sum_{n_2:0}^{7} \sum_{n_3:0}^{3} \sum_{n_4:0}^{1} x(64n_1 + 8n_2 + 2n_3 + n_4).W_8^{n_1k_1}.W_{64}^{n_2k_1}$$
$$.W_8^{n_2k_2}.W_{256}^{n_3(k_1+8k_2)}.W_4^{n_3k_3}.W_{512}^{n_4(k_1+8k_2+64k_3)}.W_2^{n_4k_4}$$
$$(3)$$

Equation (3) can be re organized as,

$$X(K) = \sum_{n_4:0}^{1} \sum_{n_3:0}^{3} \overbrace{\sum_{n_2:0}^{7} \underbrace{\sum_{n1:0}^{7} x(64n_1 + 8n_2 + 2n_3 + n_4).W_8^{n_1k_1}}_{\text{BF1 (Radix-8)}} .\underbrace{W_{64}^{n_2k_1}}_{\text{TFS1}} .W_8^{n_2k_2}}^{\text{BF2 (Radix-8)}}$$

$$\underbrace{\underbrace{.\underbrace{W_{256}^{n_3(k_1+8k_2)}}_{\text{TFS2}} .W_4^{n_3k_3}}_{\text{BF3 (Radix-4)}} .\underbrace{W_{512}^{n_4(k_1+8k_2+64k_3)}}_{\text{TFS3}} .W_2^{n_4k_4}}_{\text{BF4 (Radix-2)}} \qquad (4)$$

Reformulation of the FFT algorithm shown in this section is similar to the modified radix 2^5 algorithm proposed in [11], but it is completely differs in the number of stages, radix of the butterflies at various stages, values of the intermediate twiddle factors to be stored, length of the FFT and targeted applications. A pipelined 8 parallel MDC FFT architecture can be derived from the equation (4). Radix of various butterflies at different stages is shown in Fig.1.

III. Data Buffering and Scheduling Methodology

The equation (4) can be mapped into an eight parallel FFT processor of four stages, where first two stages are radix-8 butterflies, third stage consist two radix-4 butterflies and fourth is of four radix-2 butterflies shown in Fig. 1. Switch boxes are used to re order the data from butterfly output before multiplying with intermediate twiddle factors. Output buffer arranges the final output into a sequence similar to the input. Cross symbols between the switch box and the butterfly stage indicates the complex multipliers, which performs the multiplications between reordered butterfly outputs and the twiddle factors. TFS1, TFS2 and TFS3 represent the intermediate twiddle factors. These are to be stored and fetched appropriately during the process. It can be observed that the order of the data shown in Fig. 2(b) is to feed the first stage of the FFT core, which is different from its order appears on input stream shown in Fig. 2(a).

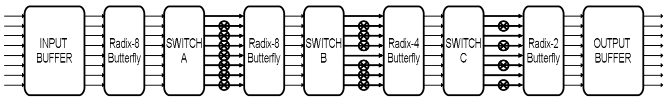
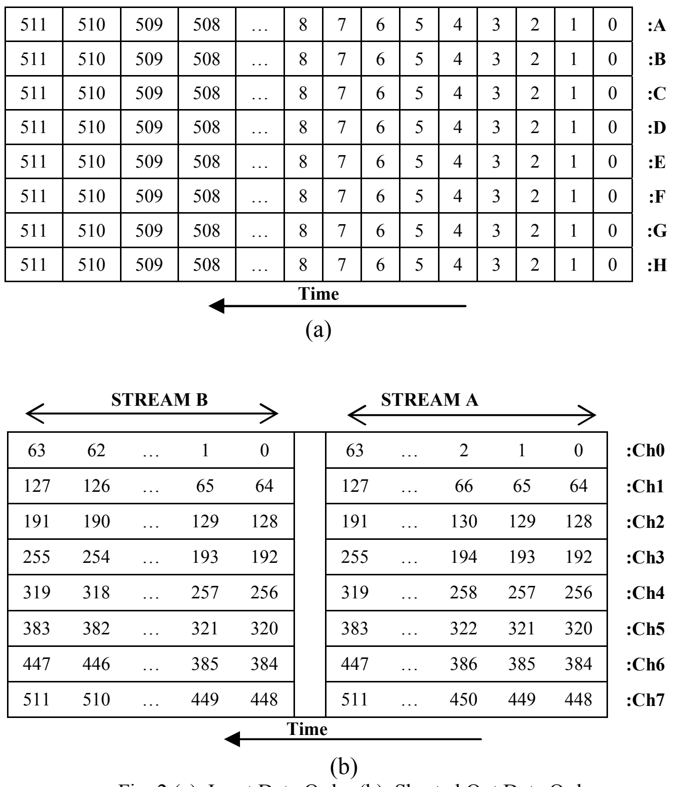


Fig. 1. Block Diagram of the FFT/IFFT processor for 8x8 MIMO Applications

Hence it is required to buffer and schedule the eight independent streams of the input samples and to feed the 8 parallel MDC FFT core in four modes (N = 64/128/256/512). Input data streams are assumed for N= 512 as an example, which are shown in Figure 2(a). In order to distinguish the shorted out streams easily, all sample streams are assumed as same and every sample is taken as its time sequence of occurrence. Scheduled and shorted data is shown in Figure 2(b), which is ready to be the input of MDC FFT.

For the description convenience, the following notations are followed: $T_1$1: 0 to (N/8-1), $T_1$2: N/8 to (2N/8-1), $T_1$3: 2N/8 to (3N/8-1), $T_1$4: 3N/8 to (4N/8-1), $T_1$5: 4N/8 to (5N/8-1), $T_1$6: 5N/8 to (6N/8-1), $T_1$7: 6N/8 to (7N/8-1). $T_1$8: 7N/8 to (N-1), $T_2$1: N to N+(N/8-1), $T_2$2: N+ N/8 to N+(2N/8-1),



511	510	509	508	…	8	7	6	5	4	3	2	1	0	:A
511	510	509	508	…	8	7	6	5	4	3	2	1	0	:B
511	510	509	508	…	8	7	6	5	4	3	2	1	0	:C
511	510	509	508	…	8	7	6	5	4	3	2	1	0	:D
511	510	509	508	…	8	7	6	5	4	3	2	1	0	:E
511	510	509	508	…	8	7	6	5	4	3	2	1	0	:F
511	510	509	508	…	8	7	6	5	4	3	2	1	0	:G
511	510	509	508	…	8	7	6	5	4	3	2	1	0	:H

(a)

STREAM B					STREAM A					
63	62	…	1	0	63	…	2	1	0	:Ch0
127	126	…	65	64	127	…	66	65	64	:Ch1
191	190	…	129	128	191	…	130	129	128	:Ch2
255	254	…	193	192	255	…	194	193	192	:Ch3
319	318	…	257	256	319	…	258	257	256	:Ch4
383	382	…	321	320	383	…	322	321	320	:Ch5
447	446	…	385	384	447	…	386	385	384	:Ch6
511	510	…	449	448	511	…	450	449	448	:Ch7

(b)

Fig. 2 (a). Input Data Order (b). Shorted Out Data Order

$T_2$3: N+2N/8 to N+(3N/8-1), $T_2$4: N+3N/8 to N+(4N/8-1), $T_2$5: N+4N/8 to N+(5N/8-1), $T_2$6: N+5N/8 to N+(6N/8-1), $T_2$7: N+6N/8 to N+(7N/8-1), $T_2$8: N+7N/8 to (2N-1) and so on.

All input streams are labelled as *A* to *H* respectively. Every *N/8* clocks are assumed as a time cycle, which is indicated by $T_i j$, where *i* indicate the part (symbol sequence number) of a given data stream and *j* stands for time cycle index. Each part consists of *N/8* samples. First *7N/8* samples of every stream are buffered during the first seven time cycles, it is shown in Table I. During the *(7N/8+1)*th clock to the *(N-1)*th clock (Time cycle: $T_1$8), the samples belongs to stream A are read-out from the BB0 and then vacated buffer locations are updated with the incoming samples from 8th part of stream *B* to *H* as shown in Table II.

TABLE I. DATA BUFFERED DURING THE FIRST 7N/8 CLOCKS

	BB0	BB1	BB2	BB3	BB4	BB5	BB6	BB7
$T_1$1:	A1N/8	B1N/8	C1N/8	D1N/8	E1N/8	F1N/8	G1N/8	H1N/8
$T_1$2:	A2N/8	B2N/8	C2N/8	D2N/8	E2N/8	F2N/8	G2N/8	H2N/8
$T_1$3:	A3N/8	B3N/8	C3N/8	D3N/8	E3N/8	F3N/8	G3N/8	H3N/8
$T_1$4:	A4N/8	B4N/8	C4N/8	D4N/8	E4N/8	F4N/8	G4N/8	H4N/8
$T_1$5:	A5N/8	B5N/8	C5N/8	D5N/8	E5N/8	F5N/8	G5N/8	H5N/8
$T_1$6:	A6N/8	B6N/8	C6N/8	D6N/8	E6N/8	F6N/8	G6N/8	H6N/8
$T_1$7:	A7N/8	B7N/8	C7N/8	D7N/8	E7N/8	F7N/8	G7N/8	H7N/8

TABLE II. DATA BUFFERED DURING 7N/8TH TO (N -1)TH CLOCKS

$T_1$8:

BB0	BB1	BB2	BB3	BB4	BB5	BB6	BB7
B8N/8 A1N/8	B1N/8	C1N/8	D1N/8	E1N/8	F1N/8	G1N/8	H1N/8
C8N/8 A2N/8	B2N/8	C2N/8	D2N/8	E2N/8	F2N/8	G2N/8	H2N/8
D8N/8 A3N/8	B3N/8	C3N/8	D3N/8	E3N/8	F3N/8	G3N/8	H3N/8
E8N/8 A4N/8	B4N/8	C4N/8	D4N/8	E4N/8	F4N/8	G4N/8	H4N/8
F8N/8 A5N/8	B5N/8	C5N/8	D5N/8	E5N/8	F5N/8	G5N/8	H5N/8
G8N/8 A6N/8	B6N/8	C6N/8	D6N/8	E6N/8	F6N/8	G6N/8	H6N/8
H8N/8 A7N/8	B7N/8	C7N/8	D7N/8	E7N/8	F7N/8	G7N/8	H7N/8

During this period, last *N/8* samples of the stream *A* are directly sent to the 8th input of radix-8 butterfly at first stage along with seven samples read out from the input buffer to feed other inputs of the butterfly in each clock. Hence, Radix 8 butterfly can process all the samples belongs to stream *A* during $T_1$8 (at the rate of eight samples per clock). By the end of $T_1$8th time cycle, Stream *A* is completely processed and register bank *BB0* is updated with last *N/8* samples of the streams *B* to *H* respectively as shown in Table II. Shaded entries in Table II, Table IV and Table V indicates the register locations being read, which are written on earlier time cycles, while un shaded entries are under write or no operation.

Data belongs to the stream *B* is read out from register banks *BB1* and *BB0* during the time cycle $T_2 1$ in shorted fashion to feed the Radix-8 butterfly, while writing the first N/8 samples of the stream *A* into *BB0* and first N/8 samples of stream *B* to *H* into BB1 (which are belongs to the second symbol)[1]. Similar procedure is repeated for streams *C* to *H* and is shown in Table IV. Register bank *BB0* is operated in all the time cycles, which is indicated by the dashed line. Scenario during the time cycle $T_2 8$ is shown in Table V, which is similar to it during the time cycle $T_1 8$, but this time, data belongs to the second symbol.

By the end of $T_2$8, buffer contains the complete 2nd symbol of streams *B* to *H* respectively and stream A is completely processed. During the period $T_3 1$ to $T_3 7$, these streams in buffer are read out and new data belongs to the third symbol on streams *A* to *H* is written into register banks. At the end of time cycle $T_3 7$, data entries on buffer looks like it is shown in Table I and the total procedure repeats.

[1] On each stream, FFT is to be calculated for a set of samples (of length 512/256/128/64 according to the mode) belongs to the respective symbol.

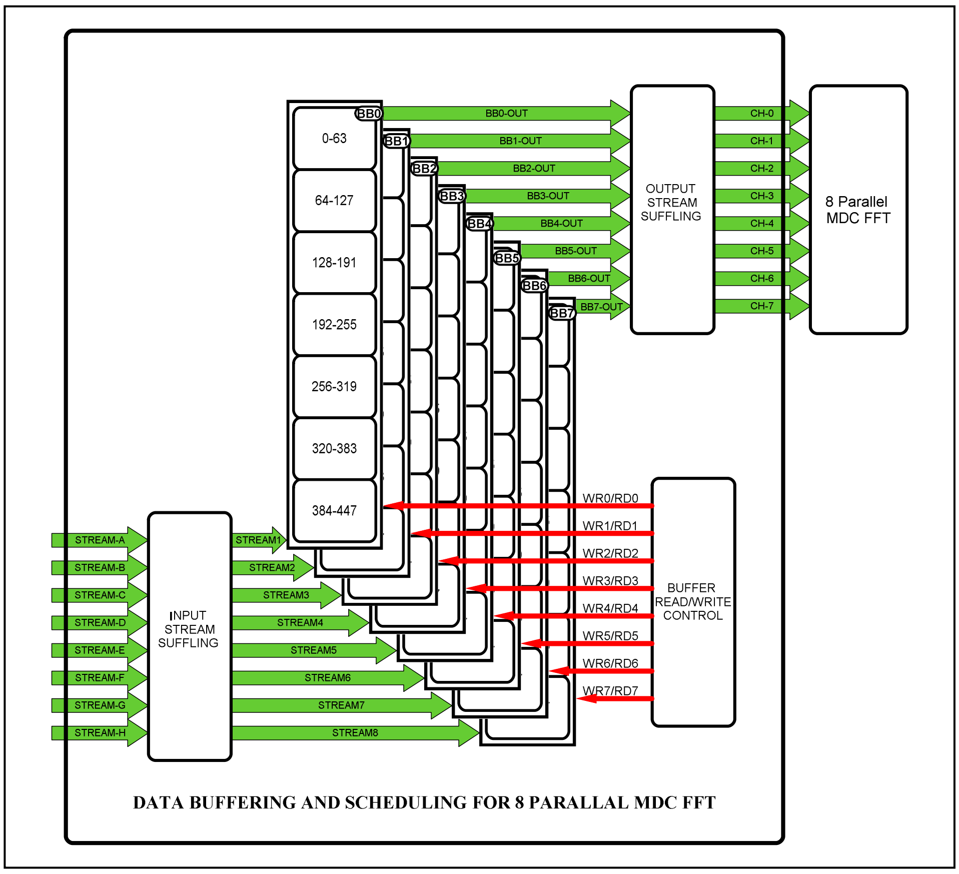


Fig. 3. Block diagram of the proposed input buffer for an 8 Parallel MDC FFT/IFFT Processor.

TABLE III. READ - WRITE CONTROL SIGNALS OF THE PROPOSED BUFFER

Cycle	OP	BB0	BB1	BB2	BB3	BB4	BB5	BB6	BB7
$T_1$1 :	w	1000000	1000000	1000000	1000000	1000000	1000000	1000000	1000000
$T_1$2 :	w	0100000	0100000	0100000	0100000	0100000	0100000	0100000	0100000
$T_1$3 :	w	0010000	0010000	0010000	0010000	0010000	0010000	0010000	0010000
$T_1$4 :	w	0001000	0001000	0001000	0001000	0001000	0001000	0001000	0001000
$T_1$5 :	w	0000100	0000100	0000100	0000100	0000100	0000100	0000100	0000100
$T_1$6 :	w	0000010	0000010	0000010	0000010	0000010	0000010	0000010	0000010
$T_1$7 :	w	0000001	0000001	0000001	0000001	0000001	0000001	0000001	0000001
$T_1$8 :	w/r	1111111	0000000	0000000	0000000	0000000	0000000	0000000	0000000
$T_2$1 :	w/r	100000	1111111	0000000	0000000	0000000	0000000	0000000	0000000
$T_2$2 :	w/r	0100000	0000000	1111111	0000000	0000000	0000000	0000000	0000000
$T_2$3 :	w/r	0010000	0000000	0000000	1111111	0000000	0000000	0000000	0000000
$T_2$4 :	w/r	0001000	0000000	0000000	0000000	1111111	0000000	0000000	0000000
$T_2$5 :	w/r	0000100	0000000	0000000	0000000	0000000	1111111	0000000	0000000
$T_2$6 :	w/r	0000010	0000000	0000000	0000000	0000000	0000000	1111111	0000000
$T_2$7 :	w/r	0000001	0000000	0000000	0000000	0000000	0000000	0000000	1111111
$T_2$8 :	w/r	1111111	0000000	0000000	0000000	0000000	0000000	0000000	0000000

TABLE IV. DATA BUFFERED DURING N^{TH} TO $(N + 7N/8-1)^{TH}$ CLOCKS

		T_21:	T_22:	T_23:	T_24:	T_25:	T_26:	T_27:
	BB0	**BB1**	**BB2**	**BB3**	**BB4**	**BB5**	**BB6**	**BB7**
T_21:	A1N/8	B1N/8	B2N/8	B3N/8	B4N/8	B5N/8	B6N/8	B7N/8
	B8N/8	B1N/8	C1N/8	D1N/8	E1N/8	F1N/8	G1N/8	H1N/8
T_22:	A2N/8	C1N/8	C2N/8	C3N/8	C4N/8	C5N/8	C6N/8	C7N/8
	C8N/8	B2N/8	C2N/8	D2N/8	E2N/8	F2N/8	G2N/8	H2N/8
T_23:	A3N/8	D1N/8	D2N/8	D3N/8	D4N/8	D5N/8	D6N/8	D7N/8
	D8N/8	B3N/8	C3N/8	D3N/8	E3N/8	F3N/8	G3N/8	H3N/8
T_24:	A4N/8	E1N/8	E2N/8	E3N/8	E4N/8	E5N/8	E6N/8	E7N/8
	E8N/8	B4N/8	C4N/8	D4N/8	E4N/8	F4N/8	G4N/8	H4N/8
T_25:	A5N/8	F1N/8	F2N/8	F3N/8	F4N/8	F5N/8	F6N/8	F7N/8
	F8N/8	B5N/8	C5N/8	D5N/8	E5N/8	F5N/8	G5N/8	H5N/8
T_26:	A6N/8	G1N/8	G2N/8	G3N/8	G4N/8	G5N/8	G6N/8	G7N/8
	G8N/8	B6N/8	C6N/8	D6N/8	E6N/8	F6N/8	G6N/8	H6N/8
T_27:	A7N/8	H1N/8	H2N/8	H3N/8	H4N/8	H5N/8	H6N/8	H7N/8
	H8N/8	B7N/8	C7N/8	D7N/8	E7N/8	F7N/8	G7N/8	H7N/8

TABLE V. DATA BUFFERED DURING$(N+7N/8)^{TH}$ TO $(2N -1)^{TH}$ CLOCKS

T_28:							
BB0	**BB1**	**BB2**	**BB3**	**BB4**	**BB5**	**BB6**	**BB7**
B8N/8 A1N/8	B1N/8	B2N/8	B3N/8	B4N/8	B5N/8	B6N/8	B7N/8
C8N/8 A2N/8	C1N/8	C2N/8	C3N/8	C4N/8	C5N/8	C6N/8	C7N/8
D8N/8 A3N/8	D1N/8	D2N/8	D3N/8	D4N/8	D5N/8	D6N/8	D7N/8
E8N/8 A4N/8	E1N/8	E2N/8	E3N/8	E4N/8	E5N/8	E6N/8	E7N/8
F8N/8 A5N/8	F1N/8	F2N/8	F3N/8	F4N/8	F5N/8	F6N/8	F7N/8
G8N/8 A6N/8	G1N/8	G2N/8	G3N/8	G4N/8	G5N/8	G6N/8	G7N/8
H8N/8 A7N/8	H1N/8	H2N/8	H3N/8	H4N/8	H5N/8	H6N/8	H7N/8

IV. ARCHITECTURE FOR DATA BUFFERING AND SCHEDULING

Proposed register bank is designed to operate in four modes (mode 0 to 3), where it supports the length of 64, 128, 256 and 512 samples on each stream per symbol. This proposed scheduling buffer is suitable for an 8 parallel pipelined MDC based FFT shown in Fig. 1 and it can be operated in variable lengths of 64/128/256/512.

A. Input Stream Shuffling Unit: This unit directs the input sample streams into the register banks: BB0 – BB7 in different patterns at different time cycles as the sequence of writing operations discussed in Section III. During T_11 to T_17 shown in Table I, stream A to stream H are directed to BB0 to BB7 respectively. The final N/8 samples of stream B to stream H are directed to BB0 during T_18 as shown in Table II. First N/8 samples of Stream B to stream H are directed to BB1 and first N/8 samples of stream A are directed to BB0 during T_21. It is shown in Table IV. Similarly remaining parts of the stream B to stream H are directed to BB2 to BB7 during T_22 to T_27 while 2N/8th to 7N/8th samples belongs to stream A are directed to BB0. The final part of stream B to stream H is directed to BB0 during T_28 as shown in Table V.

B. Buffer Read Write Control Unit: Input buffer unit consists of 8 register banks, each with the storage capability of 7N/8 words. Each register bank is organized into 7 parts, where each part of N/8 registers and every part has independent read - write access. Read and write access to various parts of the register bank can be established by the control unit and control signals are shown in Table III. Each entry in this table corresponds to a register bank and a time cycle. There are seven write/read control signals connected to seven parts of the respective register bank during the given time cycle. Only write operation takes place during the first seven time cycles (T_11 to T_17), where the content of the input streams are written as shown in Table I. Both read and write happens from the time cycle T_18 and onwards, by following a rule that write after read. Control signals from T_31 to T_37 are similar to those from T_11 to T_17 shown in Table III, but here onwards there will be both read and write. Also, those signals during T_38 to T_48 are similar to T_18 to T_28 and this is repeated for the consequent time cycles. It can be observed that, the control signals shown in Table III are used to direct the input streams into register banks *BB0* to *BB7* as shown in Table I, Table II, Table IV and Table V respectively. The control unit also dynamically maintains the length of time cycle T_ij as N/8 clocks. Hence, this value varies according to the mode of the targeted FFT core. It is 64 clocks for N=512 and 32, 16, 8 clocks for N= 256, 128, 64 respectively.

C. Output Stream Shuffling Unit: This unit shuffles the read out data from register banks into an order required by the FFT core for a given mode. In each clock, eight samples are read from the buffer, as it is described in Section III. These samples are shuffled and directed by this unit onto output channels CH0 to CH7. It is shown in figure 3.

V. IMPLEMENTATION

The proposed input buffer is verified for functionality using Synopsys VCS simulation tools.

TABLE VI. IMPLEMENTATION AREA RESULTS FOR N=512

Design	Area (μm^2)	Operating Frequency
Input Buffer	738446.055	40 MHz

TABLE VII. IMPLEMENTATION POWER RESULTS FOR N=512

Power	Input Buffer used in [7]	Proposed Input Buffer
Total Power (mW)	8.69 (15.12% of the total)	5.02

After rigorous verification using coverage driven functional verification, input buffer design is synthesized using TSMC 65nm standard cell library. Synopsys Design Compiler is used for synthesis. The gate level Netlist obtained from the synthesis is verified by performing the gate level simulations. The power analysis of the design is performed using Synopsys power compiler. The dynamic power is analysed on RTL code and synthesized gate level net list. The dynamic power is analysed by generating the SAIF (Switching Activity Interchange Format) file from the test bench simulations. Based on the switching activity inside the design, an accurate dynamic power calculation is done. To maximize the switching activity inside the design, we make use of ATPG (Automatic Test Pattern Generation) tool Synopsys Tetramax, which generates the stimulus patterns for simulations. SAF (Stuck at fault) patterns are generated in Tetramax to maximize the switching activity in the design. The area results are presented in Table VI. The total power is given in Table VII. The total power is 5.02 mW. The leakage power component is 3.58 mW and dynamic power is 1.44 mW.

VI. Conclusions

In this paper, the brief idea of a variable length and multimode MDC FFT suitable for baseband processor in 8x8 MIMO OFDM systems is proposed. A novel approach for buffering and scheduling of eight streams of samples is proposed. Input buffer is designed using 7N registers on eight register banks. The implementation results are also presented for N=512. The input buffer for 512 point FFT used in [7] consumes 8.62 mW (15.12% of total power for complete FFT implementation with N =512) of power. Our proposed design, with simple control and data flow using register files, consumes 5.02 mW of power at the same operating frequency (40MHz).

References

[1] Sneha Ambastha, "Wireless Communication: Evolving to Meet Ever-Changing Needs," *Electronics For You,* pp. 56-60, November 2014.

[2] CISCO, "802.11ac: The Fifth Generation of Wi-Fi," Technical White Paper, Cisco Public Information, 2012.

[3] J.-W. T. a. T.-Y. C. Song-Nien Tang, "A 2.4-GS/s FFT Processor for OFDM-Based WPAN Applications," *IEEE Trans. CIRCUITS AND SYSTEMS-II,* vol. 57, no. 6, pp. 451-455, 2010.

[4] Ian Poole, "Radio-Electronics.com," Adrio Communications Ltd, [Online].Available:http://www.radioelectronics.com/info/wireless/wi-fi/ieee-802-11ac-gigabit.php. [Accessed 25th February 2015].

[5] E. a. U. J. Koushik Maharatna, "A 64-Point Fourier Transform Chip for High-Speed wireless LAN Application Using OFDM," *IEEE J. SOLID STATE CIRCUITS,* vol. 39, no. 3, pp. 484-493, 2004.

[6] Y.-W. L. Y.-C. T. a. C.-Y. L. Yuan Chen, "A 2.4 -Gsample/s DVFS FFT Processor for MIMO OFDM Communication Systems," *IEEE J. SOLID STATE CIRCUITS,* vol. 43, no. 5, pp. 1260-1273, 2008.

[7] S.-H. T. G. C. H. C. Kai-Jiun Yang, "MDC FFT/IFFT Processor With Variable Length for MIMO-OFDM Systems," *IEEE Trans. VLSI SYSTEMS,* pp. 720-731, APRIL 2013.

[8] S. He and M. Torkelson, "A new approach to pipeline FFT processor," in *Proc. IEEE Int. Parallel Process. Symp.*, Apr. 1996, pp. 766–770.

[9] T. Sansaloni, A. Perex-Pascual, V. Torres, and J. Valls, "Efficient pipeline FFT processors for WLAN MIMO-OFDM systems," *Electron. Lett.*, vol. 41, no. 19, pp. 1043–1044, Sep. 2005.

[10] B. Fu and P. Ampadu, "An area efficient FFT/IFFT processor for MIMO OFDM WLAN 802.11n," *J. SIGNAL PROCESS. Syst.*, vol. 56, no. 1, pp. 59–68, Jul. 2009.

[11] T. C. a. H. Lee, "A High-Speed Low-Complexity Modified Radix-25 FFT Processor for high rate WPAN Applications," *IEEE Trans. VLSI SYSTEMS,* pp. 187-191, 2013.