Low Cost FPGA based Implementation of a DRFM System

M.B. Mesarcik University of Cape Town
South Africa
Email: msrmic004@myuct.ac.za

***Abstract*—Digital Radio Frequency Memory (DRFM) is a technique used to record an incoming Radio Frequency (RF) signal, in turn applying a series of time-delays, amplitude scalings and frequency shifts and retransmitting the signal. This technique is used widely in the electronic defense industry as a form of radar jamming, in that it allows for the synthesis of artificial targets. This paper discusses the design and implementation of a DRFM system on a low cost FPGA system and documents this system's performance.**

I. Introduction

A. Digital Radio Frequency Memory

Digital Radio Frequency Memory (DRFM) systems digitize and store incoming RF input signals at specific frequencies and bandwidths. The captured signals undergo time delays, frequency shifts and amplitude scalings, after which they are retransmitted. DRFM is used in order to synthesize false targets for radar systems by exploiting the radar assumptions of target identification. This is achieved by retransmitting a time delayed, amplitude scaled and frequency shifted coherent replica of the inputted signal, thereby making it, from the radar's perspective, indistinguishable from other genuine signals [1].

The reason why a radar system is deceived is due to its fundamental operation. For example, a pulsed radar operates by transmitting pulses on a target and measuring the time delay and frequency shift of the return pulse from the target in order to infer information about a target's location and velocity. The implications of this are such that if we simulate echos with certain frequency shifts and time delays it is possible for a radar to identify a target incorrectly.

An illustration of a DRFM system may be seen in Fig. 1. It can be seen that the input signal is mixed down to intermediate frequency and then is sampled by an Analog to Digital Converter (ADC). The sample rate of the ADC is equal to the bandwidth of the incoming RF signal as the ADC is sampling complex I/Q data. The digitized signal is then stored in memory and may be manipulated by means of the control interface. The control interface is used to designate the various time delays, frequency shifts and amplitude scalings.

B. FPGA based Architecture

Due to the high data rates and the Digital Signal Processing (DSP) requirements the implementation of DRFM systems are highly suitable to Field Programmable Gate Arrays (FPGA). Therefore this paper describes the design and implementation of a DRFM system on a low cost FPGA, namely the Altera MAX 10 [2].

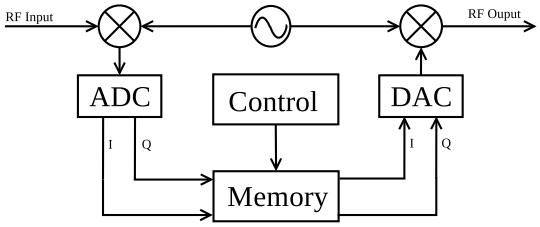


Fig. 1: Illustration of DRFM System

C. Plan of Development

This paper begins by discussing the conceptual design of the proposed DRFM architecture by giving an overview of its implementation. After which, the details of each subsystem is described. Finally, results are shown and conclusions are drawn based on the prototype design.

II. Overview

For means of explanation, the architecture of the FPGA based implemented DRFM system can be described by three subsystems. These being, the interfacing and control system, the external peripherals and the DSP system. Whereby each of the constituent systems are monitored and controlled by the top-level controller module. This simplified architecture may be seen in Fig. 2.

It must be noted that the architecture described in Fig. 2 has not yet been integrated into a RF front end and therefore this paper is a review of the work in progress. The implications of this are that although real time signal processing and data storage are realizable at the speeds dictated by the RF front-end, it has not been proven that it is possible using this design.

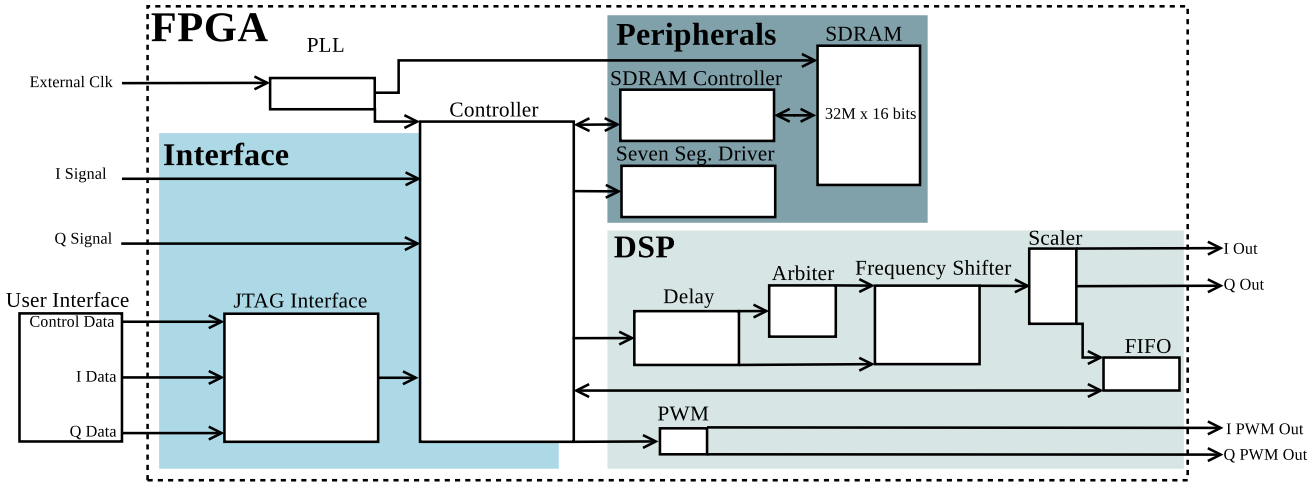


Fig. 2: Illustration of the implementation of the FPGA based DRFM System

A. Interface

1) JTAG Interface: Joint Action Test Group (JTAG) was originally used as a simplified standard for interconnectivity testing for PCB manufacturing. It was created as a result of Integrated Circuits (IC) pins becoming more densely spaced and therefore classical interconnectivity test approaches were infeasible. The JTAG protocol mitigated the need for physical access to an IC's pins through the use of shift register chains.

In the JTAG standard (IEEE 1149.1) the collections of shift registers are referred to as Boundary Scan Cells (BSCs) which sample and hold data from and to the pin interface. As a result the BSCs can be configured into a serial shift chain thereby allowing to be driven by a serial interface.

The JTAG interface implemented on the DRFM system allows for software based monitoring, updating and controlling of the DRFM system through the JTAG port. Through the use of the Altera Virtual JTAG Intellectual Property (IP) core it is possible to access to the JTAG control signals that are routed to the FPGA core. This allows for a fine control over the JTAG resources thereby granting real-time general purpose serial communication [3].

2) User Interface: Through the use of a Python based User Interface (UI) that runs a TCL JTAG server it is possible to transfer both I/Q data as well as control information to the FPGA as may be seen in Fig. 2. The control word consists of 64 bits with flags and data contained within its length. The control word composition may be seen in Table. I given that the most significant bit is at position 61.

TABLE I

CONTROL WORD COMPOSITION

Control Information	Control Flag Position	Control Data Length
Doppler Shift	33	32
Time Delay	44	10
Amplitude Scale	61	16

In addition to this, the UI was capable of transferring 32 bits of I/Q data via JTAG to the FPGA system on demand. Where the I/Q data were 16 bits wide respectively.

B. External Peripherals

In order to account for the requirement of I/Q data injection, an external SDRAM memory was incorporated into the DRFM system. It was used as a means to store the injected I/Q data so that it could be recalled for the DSP operations of the DRFM system. The SDRAM device is a 512Mb high speed CMOS dynamic access memory with 32M of address space that address 16 bit words [4].

Due to the fact that each I/Q channel were 16 bits wide, it was necessary to interleave the I/Q data samples. Where the first address contained I data and the next Q data.

A SDRAM controller was implemented on the FPGA in order to facilitate communication with the external SDRAM peripheral. The controller made use of the Altera SDRAM Controller IP Core which uses the Avalon Interfacing standard to communicate with the SDRAM device. This was integrated into the system through the use of the Altera Qsys tool, that allowed for pin allocations to the external SDRAM device [5].

In addition to this it was required to clock the external SDRAM device at different clock rate. Therefore it was necessary to implement a Phase Locked Loop (PLL) in order to synchronously step down the clock speed.

Finally, for debugging and user interfacing purposes a group seven segment displays were used. The seven segment displays gave visual feedback to the operator of the system by displaying the current state of the DRFM system.

C. Digital Signal Processing

As previously discussed, the digital signal processing aspect of a DRFM system was critical to its operation. As time delays, frequency shifts and amplitude scalings were needed in order to emulate a target from a radar system's perspective.

The major consideration in the implementation of this signal processing scheme was the order in which the signal processing operations were performed. This was due to the real-time processing requirements of a DRFM system, such when performing variable time delays there would be no loss of coherence between the input and output signals.

This meant that if a time delay was set by the user and a frequency shift was then applied, the system should not have to repopulate the delay buffer. For this reason the order of the signal processing operations were to first perform the time-delay, then frequency shifts and finally amplitude scaling, as can be seen in Fig. 2.

1) Signal Processing Model: In order to describe the signal processing model, the signal, $s_{rx}(n)$, is considered to be the input to the signal processing chain. As the DRFM system receives I/Q data, $s_{rx}(n)$ may be expressed mathematically as

$$s_{rx}(n) = v_{rx}(n) + j\mathbb{H}\{v_{rx}(n)\} = I(n) + jQ(n) \quad (1)$$

where $v_{rx}(n)$ is the digitized received RF signal and $\mathbb{H}\{v_{rx}(n)\}$ is the Hilbert Transform of $v_{rx}(n)$. Such that, $v_{rx}(n)$ is equal to I and $j\mathbb{H}\{v_{rx}(n)\}$ is equal to jQ.

This model is used as it lends itself to the analytic signal representation of the incoming RF signal. This being that the I/Q data does not contain any negative frequency components. As the I component is a replica of the incoming RF signal and the Q component is a 90° phase shifted version of the RF signal, thereby resulting in their sum yielding no negative frequency components. The benefit of this is that it allows one to sample at the RF frequency, rather than at the Nyquist Rate and that it simplifies the arithmetic operations applied to these signals. [6]

2) Time Delay: The time delay operation may be expressed as a method of altering the index of the input signal $s(t)$. This process may be summarized mathematically by

$$s_{delay}(k) = s_{rx}(n-k), \quad n \geq k. \quad (2)$$

This model may be implemented on a digital system by injecting a portion of the input signal $s_{rx}(n)$ into a RAM. After which, indexing previous values of $s_{rx}(n)$ by means of the RAM's address pointers. Which then allows for the outputting of previous values of the inputted signal given that capacity criteria has been met. This being that the delay address index, k, is less than or equal to current RAM address index, n. In the implementation of the DRFM system, a dual port, 1024 address long, 16 bit wide RAM is implemented internally on the FPGA.

3) Frequency shifting: The frequency shifting model implemented on the DRFM system can be easily summarised mathematically by the following expression

$$s_s(n) = s_{rx}(n)e^{j2\pi f_s n} \quad (3)$$

where $s_s(n)$ is the shifted version of $s_{rx}(n)$ and f_s is the shift frequency. By expanding the complex exponential term into its constituent sinusoids and using the previously explained I/Q signal model it can be seen that

$$\begin{aligned} s_s(n) &= [I(n) + jQ(n)][cos(2\pi f_s n) + jsin(2\pi f_s n)] \\ &= [I(n)cos(2\pi f_s n) - Q(n)sin(2\pi f_s n)] \\ &\quad + j[I(n)sin(2\pi f_s n) + Q(n)cos(2\pi f_s n)] \\ &= I_s(n) + jQ_s(n) \end{aligned} \quad (4)$$

where $I_s(n)$ can be considered to be the real frequency shifted component and $jQ_s(n)$ is the complex frequency shifted component.

The implications of Equation. 4 are that the frequency shifter can be implemented through the use of four multipliers, an addition and a subtraction.

4) Amplitude Scaling: As control word allocates 16 bits to the amplitude scaling information, a model was implemented to translate the control data to a value between 1 and 0. This was achieved by applying a non circular bit shift to the amplitude scaling portion of the control word. Such that the amplitude scaling information would be shifted 16 bits to the right in order to scale it between 1 and close to zero. This operation may be expressed mathematically by

$$s_{scaled}(n) = 2^{-p} A \; s_{rx}(n), \quad 0 \leq p \leq 15 \quad (5)$$

where p is the number of shifted positions the amplitude scaling section of the control word is shifted by and A is the magnitude of the amplitude control word. This is possible due to the properties of the binary number system as shifting a bit sequence right by one position implies a division by 2.

The implications of this are that the smallest number that can be represented is 30.52×10^{-6}, which for the practicalities of this system is sufficient as a DRFM should not have to attenuate the outputted signal completely.

5) Complete Signal Model: From the above derivations it is possible to conclude that the outputted signal can be expressed mathematically by

$$s_{tx}(k) = 2^{-p} A \; s_{rx}(n-k)e^{j2\pi f_s n}. \quad (6)$$

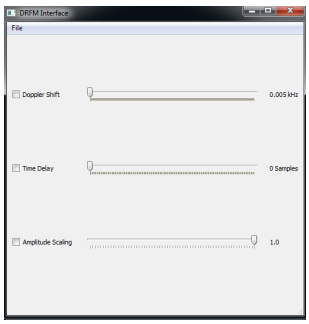
III. Detailed Design

A. Interface

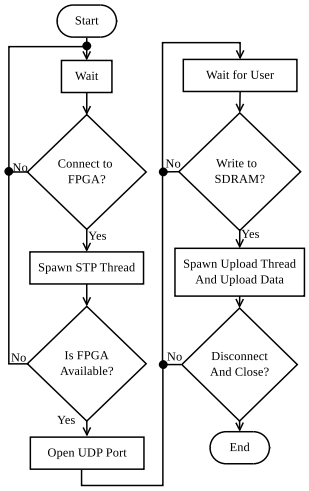
1) User Interface: In order to give the user intuitive control over the DRFM system a python based UI was implemented. It is a multi-threaded cross-platform interface built on the Python PyQt framework that called the Quartus TCL API [7] [8]. It must be noted, that all TCL interfacing was done using the adapted Altera example TCL server code [9].

The UI is made up of a menu bar, 3 sliders, and three check buttons. The menu bar allowed the user to connect to the DRFM based FPGA system or to upload data to

SDRAM. The buttons enabled the user to select each of the DSP operations to be performed and the sliders give the user a fine grained control over these operations, as can be seen in Fig. 3a.



(a) Screen shot of the DRFM User Interface



(b) Flow diagram documenting the operation of the DRFM interface

Fig. 3: Figures documenting the DRFM interface

The way in which the interface works can be seen in Fig. 3b. The first stage of operation is initializing the Graphical User Interface (GUI), and then waiting for the user's input. At this point, if the user connects to the FPGA a STP thread is spawned. This thread opens up a TCL server that allows for UDP traffic to be directed to the FPGA given that it is connected and available.

From this point, the user can send control information via the check boxes and sliders through the Python socket API. At any moment after connection has been made, the user may upload data to the FPGA via the menu option and by setting switch zero (SW0) high on the FPGA development board. The data is then transferred by calling another TCL script that polls data from a pre-generated MATLAB `.dat` file and shifting the data into to the FPGA through the JTAG interface.

2) JTAG Interface: As previously explained, the JTAG interface was implemented using the Altera Virtual JTAG IP core for it enabled communication between the TCL server and the FPGA. The implementation thereof, required the use of the standardised JTAG interface. The pins and their descriptions can be seen in Table. II.

TABLE II

JTAG INTERFACE PIN DESCRIPTIONS

Pin Name	Description
tck	Test Clock Input
tdi	Test Data Input
tdo	Test Data Output
ir_in	Instruction Input
cdr	Capture Data Register
sdr	Shift Data Register
udr	Update Data Register

The `tck` input is an external clock that is supplied by the TCL server that synchronizes the internal state machine operations. The `tdi` and `tdo` pins are parallel input and output data lines to and from the TCL server. The `ir_in` output port of the Virtual JTAG IP core is the resultant output of the data that has been shifted into the instruction register of the JTAG interface. Finally `cdr`, `sdr`, and `udr` each indicate which state the finite state machine is currently in, these being, capture, shift and update states respectively [3].

These pins are then routed according to which mode of operation in DRFM system is in, either write mode or data capture mode.

In write mode, the `tdi` line is routed to the input of an SDRAM controller and the addressing information is incremented according to the current state the virtual JTAG controller. In data capture mode, the control information is shifted out of the `tdi` line and its contents is captured. This saved information then determines the values that are applied to the frequency shift, amplitude scaling and time delay.

These two modes of operation are determined by the user setting SW0 on the development board. When the switch is high, the system is in write mode and when low the system is in data capture mode.

3) Controller: The controller module enabled the communication and routing between each of the various subsystems, for this reason it is described as part of the interface subsystem.

The controller receives and transmits the external peripheral data signals from the SDRAM and the PLL and routes them to and from the SDRAM Controller, JTAG interface and the DSP subsystems. In addition to this, it serves as a means to ensure the data being outputted from SDRAM is correctly handled as the SDRAM controller can not output data on every clock cycle.

The reason for this is due to the physical properties of the SDRAM device which needs to refresh its cells regularly in order to maintain coherency of its data. During these refresh cycles, which are managed by the SDRAM controller, the SDRAM cannot ensure valid output data. Therefore during the duration of the refresh cycles the SDRAM cannot output data, and so the data output must be arbitrated, which is performed by the controller module.

The arbitration employed by the controller module worked by writing the SDRAM data into a 2048 address wide RAM module. When the data was needed, it was read back from the RAM at a significantly slower speed than the writing operation was performed. This meant that when the SDRAM was unable to read data during its refresh cycles, there was sufficient data in the RAM module so that its output appeared unchanged.

Furthermore, data was only written from the SDRAM to the RAM block when the `Read_DataValid` line from the SDRAM was high and was only read from the SDRAM to the RAM block when the `Read_WaitRequest` was low.

Finally, the controller module also performed the changes in assignment for when the SDRAM was being written to or read from. The way this was done was by having duplicates of the inputs to the SDRAM controller, and reassigning them when the switch on the FPGA development board was toggled.

B. Peripheral Interfacing

1) SDRAM Subsystem: As already mentioned, the Altera SDRAM Controller IP Core [5] was implemented through the use of Altera's Qsys integration tool. This was done so to abstract the internal details of the SDRAM system. A functional block diagram of the SDRAM Controller core can be seen in Fig. 4.

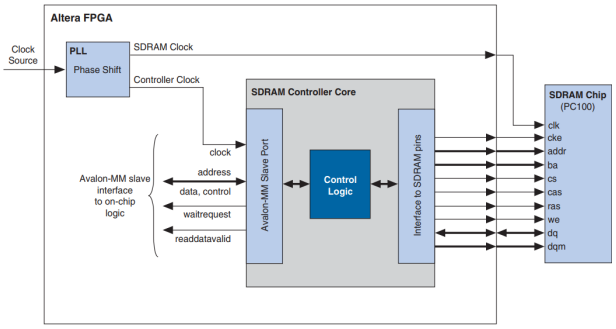


Fig. 4: Block Diagram Demonstrating Interfacing SDRAM [5]

The Avalon-MM interface is the part of the SDRAM controller that is visible to the higher level controller block. It abstracted the external SDRAM ports to a simpler interface that enabled its integration into the DRFM system.

Practically, the interface abstracted most of the lines so that only the data, data valid, wait request, write address, read address and read/write lines had to be considered when reading and writing from SDRAM. This meant that the top level controller module was able to drive Avalon-MM Slave interface as described above.

This was because the Avalon-MM slave port supports external SDRAM peripheral controlled wait states for when the external SDRAM device is not capable of receiving or sending data. This is as a result of the external SDRAM peripheral periodically refreshing its cells. This means, that in order to overcome these problems, the slave port interface employed the use of wait request and read data valid lines. So that when the external SDRAM peripheral was in its refresh state, the top level controller module could wait until it was capable of receiving data again.

2) PLL: As shown in Fig. 4, the implementation of the DRFM system required for the fundamental clock frequency to be altered in order to effectively communicate with the external SDRAM chip. In order, to do this the Altera Phase Locked Loop (PLL) IP Core was incorporated into the design [10].

A PLL is a closed loop frequency control system that works by comparing phase difference between the reference input clock signal and the feedback clock signal from the voltage controlled oscillator. This information about the error in phase of the two signals is then used to control the frequency of the loop. The main components of a PLL can be seen in Fig. 5.

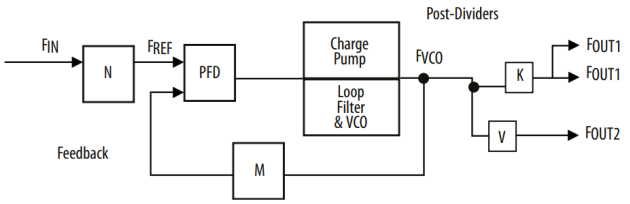


Fig. 5: Block Diagram of the Internal Structure of the PLL IP Core [10]

The fundamental elements of the Altera PLL are the Phase Frequency Detector (PFD), Charge Pump, Loop Filter, Voltage Controlled Oscillator (VCO), and Counters and Dividers, such as a Feedback Counter (M), a Pre-Scale Divider (N), and Post Dividers (K and V).

A feature of the Altera PLL IP Core was that it offers a locked output signal that goes high when the output is in phase with the input. Therefore, in order to ensure the system was synchronous with the PLL the bitwise inverse of locked line was used as a reset port of each module. Such that when the PLL was not locked to the input signal's phase, the system would not output and all output lines would be pulled to ground.

3) PWM Module: The Pulse Width Modulation (PWM) module was used as a debugging mechanism in the DRFM system to verify that the frequency shifter module was changing the frequency correctly. The PWM output was probed with an oscilloscope with FFT capabilities so that the output frequency could be measured accurately.

As the system clock was 100MHz and the PWM output had an 8-bit resolution the maximum bandwidth that could be outputted through the PWM module was $100 \times 10^6/(10)(2^8) \approx 40kHz$. This is because the PWM frequency is required to have at least 10 times the signal's bandwidth.

Furthermore, due to fact that the output from the DSP subsystem was 32-bits wide meant that the output had to be concatenated down from 32 bits to 8 bits. However, this resolution was sufficient for system testing and verification, but lead to various harmonics being present within the

output wave form. Nonetheless, the fundamental frequency component was still clearly present in the probed output.

C. Digital Signal Processing Subsystems

1) Delay: As already explained, the delay module was implemented through the use of a dual port RAM block. Practically, the time delay was achieved by injecting a portion of the input signal into RAM and indexing previous values of the input signal. In the implementation of the DRFM system, a dual port, 1024 address long, 16 bit wide RAM was implemented internally on the FPGA.

The limitations of this were that 1024 samples had to be maintained in the RAM, thereby inhibiting the DRFM system from being able to output data for first 1024 clock cycles of operation. Naturally, this is an overhead of any DRFM system as real-time data must be sampled and stored. However, once the capacity criteria of the RAM had been met a single sample may be outputted every clock cycle.

For this reason, the signal processing chain was structured as it is. This was because had a user defined variable frequency shift occurred before the delay, the RAM would have had to be refilled as previous values of the frequency shifted signal would need to be outputted.

Finally, the system also took into account that when the user changed the delay input, the subtraction from the current address should only occur once. For example, if the current address pointer of the delay RAM was 1024 and the user changed the delay to 256, then the RAM address should change to 768 and the address pointer should address sequentially from that point. This means that the previous delay input was kept track of, so that only when the new user input delay was different to the old delay would the RAM change its address.

2) Arbiter: As the SDRAM was only 16 bits wide and each I/Q sample was 16 bits wide, the I/Q data had to be interlaced into the RAM when being injected. This meant that as the frequency shifting operations required both I/Q samples simultaneously the data had to be arbitrated to perform the addition and multiplication operations of Equation. 4.

This process was achieved through the use of a simple finite state machine that had 2 states, the read I data state and the read Q data state. In the read I state the first address from RAM was stored in the arbiter as a temporary variable. In the read Q state, the subsequent address was outputted from the arbiter to the frequency shifter and a ready flag was pulled high. This finite state machine may be seen in Fig. 6.

In addition to this, the arbiter module outputted a full scale 2's compliment version of the inputted I/Q data injected to SDRAM. The way in which this is done was by inverting the first bit of the I/Q Data word and appending it to the rest of the word. So in the case of the 16 bit data word, the 15th bit was inverted and bits 14 down to 0 remained unchanged.

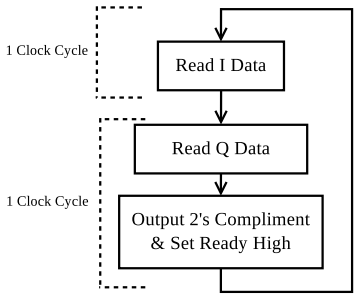


Fig. 6: Flow chart of the Arbiter Finite State Machine

3) Frequency Shifter: The frequency shifter module used in this DRFM system followed the exact mathematical model described by Equation. 4. As per the derivation, the implementation required four multipliers, an addition and a subtraction, in order to result in the frequency shifted I/Q data. For this reason, the frequency shifter subsystem can be described in two parts. These being the Numerically Controlled Oscillator (NCO) and the arithmetic units.

The NCO is an oscillator that uses a look up table, an adder and a memory to output periodic waveforms as shown in Fig. 7. In this case, the periodic waveforms are sin and cos, as they are required to map the complex exponential frequency shift to a realisable model. The way the NCO worked was by receiving an input frequency word that was integrated to obtain a phase that could be addressed on a Look Up Table (LUT). The LUT was implemented through the use of a 4096 address long 16-bit wide dual port ROM that was pre-initialized with a MATLAB generated Memory Initialization File (MIF). The MIF file's contents was a 100Hz sinusoid that was sampled at 2^{12}Hz.

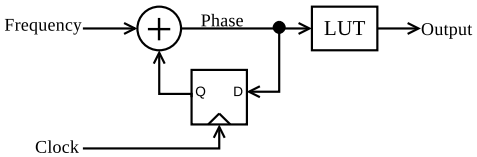


Fig. 7: Functional Block Diagram of an NCO

Furthermore, in order to generate both the cosine and sine signals that were needed for the frequency shift, two 90° out of phase pointers were used. Digitally, the 90° phase shift translated into the pointers being 1024 addresses apart.

Finally, in order to scale the user inputted frequency shift into a frequency shift a translation had to be performed. This translation can be expressed mathematically as

$$f_{shift} = f_{slider}\frac{2^{32}}{100MHz} \tag{7}$$

where f_{shift} is the instantaneous frequency that the NCO must output, f_{slider} is the user input in kHz, 2^{32} is the frequency word resolution and 100MHz is the clock frequency.

The second part of the frequency shifter was the arithmetic subsystem. A block diagram of the arithmetic subsystem may be seen in Fig. 8. It can be seen that the arithmetic operations mirror exactly what is described in Equation. 4. The intricacies of these arithmetic operations lie within the binary representations of the outputs of each arithmetic operation. As the input data is 2's complement the system had to keep track of the signed-ness of each of the results. These being that the multiplication operation had to check that the 32 bit result overflowed correctly, thereby maintaining the resultants sign.

This was achieved by outputting the result to a 33 bit temporary vector and checking its Most Significant Bit (MSB). If the MSB was high it meant that the output was negative, then it checked if the subsequent bit was high, if it was, it meant that the result had not overflowed and the bits 31 down to zero were outputted. However when the subsequent bit was low, meant that the result had overflowed and `0x8000 0000` was outputted as it was the biggest negative 32 bit number. Conversely, if the MSB bit was low, implied that the result was positive, which meant that the subsequent bit had to be inspected. If it was high, it meant that result had overflowed and `0x7FFF FFFF` was outputted as it was the largest positive 32 bit number in 2's compliment notation. However if the subsequent bit was low meant that the result had not overflowed and that bits 31 down to zero could be outputted.

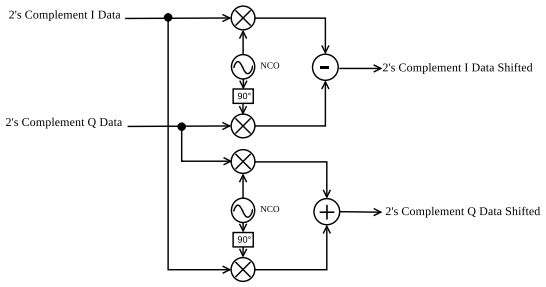


Fig. 8: Diagram of the Arithmetic Elements of the Frequency Shifter

Naturally as all the arithmetic operations performed in the frequency shifter chain were 2's complement, the same method of ensuring that no overflows had occurred were done in the multipliers, the subtracter and the adder.

Finally, the only difference between the addition and subtraction operations was that the subtraction operation applied a 2's complement inversion of the Q Data before performing the addition. This is a result of the properties of 2's complement notation, in that if you do a bitwise inverse of a 2's complement number and then add 1, the result is a negated version of the original bit stream.

4) Scaler: As already explained the scaler module worked by applying a bit shifted multiplication on the incoming I/Q data and the amplitude scaling control word. This was implemented by applying a non circular shift to the amplitude scaling control word by 32 bits to the right and the I/Q data by 16 bits to the left. So that the resultant vectors were 48 bits wide. This was done to translate the scaling operation into a multiplication of the I/Q data with the amplitude scaling control word that had been translated between one and close to zero.

Finally, the I/Q data that was received by the amplitude scaling module was the unsigned, delayed and frequency shifted version of the original data. This meant that an unsigned multiplication had to be done, such that the negative and positive overflows never had to be considered.

IV. Results

Using the implementation described the following results were obtained:

A. Delay

Through the use of the Quartus Signal Tap II Logic Analyser [11], it was possible to probe registers while the implementation was running on the FPGA in real time. As a result screen shots were taken documenting the achievements of the DRFM system.

In Fig. 9, the time delay functionality of the system can be seen. The the topmost value of the screen shot, `rdaddress`, is the address pointer to the RAM delay block. It can be seen that when the value `time_delay` changes from 853 to 938 the `rdaddress` register changes from 1016 to 78 and increments by one for each subsequent clock cycle. It can be also seen that on the cycle after the `time_delay` delay line changes, the `old_time_delay` line takes the `time_delay`'s value.

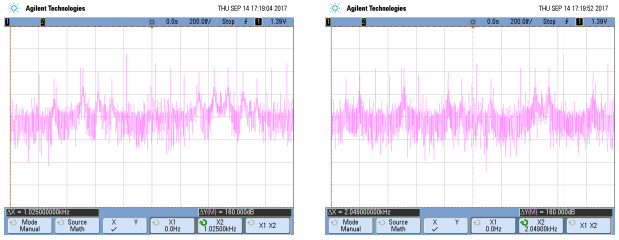
Fig. 9: Simulation Output Documenting a Time Delay

B. Frequency Shift

As already mentioned, the PWM module was used to output the waveforms through a GPIO pin on the FPGA development board. Therefore in order to obtain results about the frequency shift information, an Oscilloscope with FFT functionality was used to measure the frequency information of the PWM outputted signal. The results of the frequency shift were obtained by injecting $y(t)$ into SDRAM which can be expressed mathematically by

$$y = cos(2\pi 10t) + cos(2\pi 20t) + cos(2\pi 30t) + cos(2\pi 40t) + cos(2\pi 50t). \quad (8)$$

It can be seen in Fig. 10a that when the frequency shift on the UI was set to 1kHz, the output from the 8 bit PWM had a frequency of 1kHz. Furthermore, when the frequency was changed to 2kHz, the output also changed to 2kHz as can be seen in Fig. 10b.



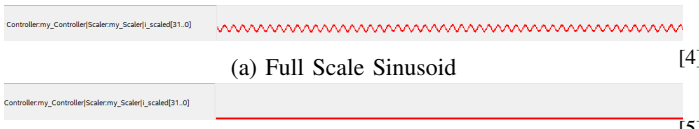
(a) Oscilloscope Print of a 1kHz frequency shift

(b) Oscilloscope Print of a 2kHz frequency shift

Fig. 10: Figures Showing the Measured Frequency Shift

C. Amplitude Scaling

By using Quartus Signal Tap II Logic Analyser [11], it was possible to verify the amplitude scaling functionality of the DRFM system. In Fig. 11a, a full scale outputted sinusoid can be seen. In Fig. 11b, a fully attenuated sinusoid can be seen. This is as a result of the user sliding the amplitude scaling slider down to zero.



(a) Full Scale Sinusoid

(b) Fully Attenuated Sinusoid

Fig. 11: Figures Showing the Measured Amplitude Scaling

V. Conclusions

From the results, the following conclusions can be drawn.

A. Increased Delay Functionality

As shown in the results, the delay function of the DRFM system was clearly working. However, it would be preferable to allow for a longer delay so that the output of the delay could be seen visually when probing with a oscilloscope. This could be achieved by creating a larger RAM block and have a longer delay control word.

B. Improved Frequency Measurement

It can be seen that the output of Fig. 10 is incredibly noisy. This may be attributable to both the fact that the PWM output has some harmonic components and in order to enable a 8 bit output, the 32 bit output vector had to be concatenated. This meant that the concatenation operation, may have resulted in unforeseen harmonic components being captured.

This problem could be solved by either performing noise shaping to get a higher resolution PWM output, or by allowing for 2 way JTAG communication so that data could be streamed back to a PC to be analysed more accurately.

C. Integration into a RF Front-end

In order to effectively prove that the DRFM system is functional it should be properly integrated into a RF front-end such as the one shown in Fig. 1. This would allow for the system to be tested more thoroughly, thereby giving a practical measure of its feasibility.

D. Extension to Multiple Targets

Finally, a simple way to extend the DRFM system would be to accommodate for multiple targets. This could be a achieved by duplicating the DSP subsystem so that false targets could be synthesized at multiple time delays and frequency shifts.

References

[1] S. J. Roome, "Digital radio frequency memory," *Electronic & Communication Engineering Journal*, pp. 147–153, August 1990. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=101420

[2] TerasIC , "DE10-Lite User Manual," January 2007. [Online]. Available: https://www.altera.com/content/dam/altera-www/global/en_US/portal/dsn/42/doc-us-dsnbk-42-2912030810549-de10-lite-user-manual.pdf

[3] Altera FPGA , "Altera Virtual JTAG (altera_virtual_jtag) IP Core User Guide," October 2016. [Online]. Available: https://www.altera.com/en_US/pdfs/literature/ug/ug_virtualjtag.pdf

[4] Integrated Silicon Solution, Inc. , "http://www.issi.com/WW/pdf/42-45R-S_86400D-16320D-32160D.pdf," June 2011. [Online]. Available: ISSISDRAMDataSheet

[5] Altera FPGA, "SDRAM Controller Core User Manual," November 2009. [Online]. Available: https://www.altera.com/zh_CN/pdfs/literature/hb/nios2/n2cpu_nii51005.pdf

[6] AJ Wilkinson, "Notes on Radar Signal Processing Fundamentals." [Online]. Available: https://wenku.baidu.com/view/e195261655270722192ef797.html

[7] Riverbank Computing, "About PyQt," June 2015. [Online]. Available: https://wiki.python.org/moin/PyQt

[8] Altera FPGA, "Quartus II Scripting Reference Manual," July 2013. [Online]. Available: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/manual/tclscriptrefmnl.pdf

[9] C. Zeh, "Talking to the DE0-Nano using the Virtual JTAG interface." April 2012. [Online]. Available: http://idlelogiclabs.com/2012/04/15/talking-to-the-de0-nano-using-the-virtual-jtag-interface/

[10] Altera FPGA, "ALTPLL (Phase-Locked Loop) IP Core User Guide," July 2016. [Online]. Available: https://www.altera.com/en_US/pdfs/literature/ug/ug_altpll.pdf

[11] ——, "Signaltap ii with verilog designs," October 2012. [Online]. Available: ftp://ftp.altera.com/up/pub/Altera_Material/12.1/Tutorials/Verilog/SignalTap.pdf