We've processed your Assignment extension request FORM-AEX-280294

no-reply@qut.edu.au <no-reply@qut.edu.au>
Thu 27/10/2022 20:25
To: Alexander Barnwell <alexander.barnwell@connect.qut.edu.au>

Hi Alexander,

Thank you for your assignment extension request **(FORM-AEX-280294)**.

We have approved your request and the due date for your assignment **Project Final Report (Written)**, for unit EGH400-2 has been extended by 48 hours from the original due date. If your unit outline does not specify that your assignment is eligible for an extension, this confirmation email is not valid and unless you submit by the original due date, the late assessment policy will apply.

You are responsible for ensuring that this assignment is eligible for extension before submitting it after the original due date. Check your unit outline for eligibility.

Be aware that a copy of this email is kept on file. You should not alter this email in any way. Email notifications that have been altered or differ in any way from the original may result in an allegation of student misconduct as set out in the Student Code of Conduct.

Need extra support? You can access free, confidential counselling with qualified professionals. We also offer planning and support if you have a disability, injury or health condition that affects your ability to study. If you are struggling to meet your university commitments, academic support is also available.

Have a question? You can contact us by email or phone. We're also available to help you on campus or via online chat. Visit the HiQ website for our contact details and opening hours.

Email us | +61 7 3138 2000

HiQ is your place to go for **student services, support** and **general enquiries:** qut.to/abouthiq

hi, how can we help you?

Ref ID: 10264248 FORM-AEX-280294

Faculty of Engineering

Hardware Efficient Implementations of FFTs

Final Research Report submitted for EGH400-2 Semester 2 2022

Student Name:	[Alexander , Barnwell]
Supervisor:	[Dr Mark Broadmeadow]
Submission Date:	[28 October 2022]

1 Abstract

This report details the design and implementation of a hardware efficient FFT for delta-sigma modulation. The requirement arises from the need for real time spectrogram analysis, which has been shown to be useful for bat echolocation and other monitoring methods and can be a useful tool in wider machine learning systems. Previous research has demonstrated ways to reduce in-operation computation of the FFT and to reduce the overall computation required when only a select amount of bins are desired. However, current limitations on utilising these methods mean that required FFT timing for real time operation are not met. The methodology of the proposed implementation follows a design science methodology at which methods pertaining to the creation of a proof of concept, an FPGA (Field Programmable Gate Array) implementation and experimental validation are conducted. the resulting application's precision is 17 bits, the reduction in precision resulting from data stored within the FPGA not being of full precision. The hardware utilisation of this strategy is dependent on the number of parallel components used, however, the strategy can be implemented on an ARTIX-7 FPGA while conforming to real time constraints for computing the FFT over the range necessary for bat echolocation.

Contents

2 Introduction

Fundamentally this thesis aims to answer the question of 'How to implement a hardware efficient FFT for delta-sigma modulated inputs?'. The significance of this implementation is that by being able to compute the FFT efficiently and in real time allows other real time elements, such as a machine learning element, to identify the bat echolocation calls[1]. This in turn allows the bat calls to be identified and catalogued in real time, improving conservation efforts. Real time spectrogram analysis is possible when the FFT can be computed at the same speed as, or faster than, the microphone can supply it with data. The reduction in hardware used by a hardware efficient approach would allow for more affordable solutions along with the potential for more exhaustive and hardware intensive bat echolocation categorisation techniques. This is because a reduction in hardware usage by the FFT allows a larger amount to be used by the machine learning element. Outside of this specific use, a hardware efficient FFT solution can serve to benefit any implementation that requires real time spectrogram analysis with single bit encoding / delta-sigma modulation. The resulting implementation strategy revolves around utilising the beneficial nature of the fine grained parallelism on FPGAs (Field Programmable Gate Arrays) to deliver a solution that meets these constraints.

2.1 Aim

In summary, the aim of the project is to develop a hardware efficient solution to computing an FFT given a sigma-delta bitstream input with an appropriate level of documentation detailing its precision and hardware usage. The FFT implementation is required to be implementable at a maximum of $580\mu s$ computing time (as the bit input time is ideally $580\mu s$) in accordance with the machine learning element while utilising a 75% reuse model on a PDM (Pulse Density Modulation) microphone with a maximum clock rate of 4.8MHz, while attempting to be as precise as possible with as little hardware utilisation as possible. From these aims comes four main objectives:

- the first objective is to create the algorithm that will compute the FFT
- secondly, the creation of rudimentary MATLAB model to verify feasibility of the algorithm.
- thirdly, to design in VHDL an FPGA ready model, then test its precision and implementation results.
- fourthly, to implement the VHDL model physically on an FPGA with appropriate supporting functions to allow testing of precision for final verification.

3 Literature Review

3.1 Introduction/Motivation

Traditional animal call detection and classification systems involve human labour and have resulted in a slow classification rate. With the advent of modern computing, resulting in machine learning systems, amongst other techniques, it is possible to improve efficiency and increase classification rate. Such a system can be used for animal population monitoring by utilising signal detection to classify certain animal noises. Specifically when using sound based detection there is a need to produce hardware efficient fast Fourier transforms to produce the necessary spectrograms. Such sound based approach is utilised in [1], where the goal was to be able to monitor bat population density utilising recorded calls from a sigma-delta modulated microphone. Sigma-delta is a modulation scheme that single bit encodes the system such that the FFT is the same as the encoded one. However, as the learning algorithm required the use of the spectrogram to be able to identify the bats the implementation was not able to run in real time. The spectrogram consumed a considerable amount of computation resources [1].Therefore as an FFT is needed for a spectrogram there exists a need for a Hardware efficient FFT for real time spectrogram analysis.

3.2 Sigma-delta

As the thesis aims to propose an efficient implementation of the fast Fourier transform (FFT) using the results from an onboard microphone, the analogue to digital converter outputs need to be considered. The decision of the specific signal form is outside the scope, the choice resulting in sigma-delta ADCs (Analogue Digital Converter). Sigma-delta ADCs modulation schemes have a high resolution and operate over smaller bandwidths, optimal for low frequency sound signals. [2]. Sigma-delta systems also have lower quantisation noise due to higher sampling or the use of lower frequency input signals [3]. However, the single bit encoding nature of sigma-delta forms the basis for the methods used to compute the FFT.

3.3 The FFT and DFT

As all computing will be done using discrete components, there is a need to compute the discrete Fourier transform. The discrete Fourier transform is a method to compute the frequency information of a sampled signal, the information is sorted in bins of a given frequency range defined by input parameters. The first algorithm considered to compute this operation was the Cooley Tukey Fast Fourier transform [4], [5] This implementation takes advantage of previous calculation to be able to reduce computational complexity. Fundamentally, the Stanley Tukey consists of two strategies: the decimation in time in which the inputs signals are interleaved, or decimation in frequency where the output bins are interleaved. This method can be described by a butterfly diagram where each section promotes a high degree of parallel-ability. This promotes the idea of an FPGA (Field Programmable Gate Array) implementation as such devices have innate fine grained parallelism. However, the the requirement to compute sections for every DFT when not every DFT is required, and the computation time for each transform block, bar the use of this implementation. The major bottleneck for effective Fourier transforms is hence in the effective computation of these sub blocks [6], [7], [8]. Therefore, investigation into different methods was undertaken (for brevity only the chosen ones are shown) resulting in the *bitstream decomposition* [6], *transform decomposition* [9]. These methods ultimately formed the basis of the implementation strategy.

3.4 Bitstream Decomposition

The bitstream decomposition is an algorithm that takes advantage of the nature of sigma-delta modulation schemes where the outputs are constrained to ’0’ or ’1’, this allows storing parts of the FFT process before-hand, reducing computation time. This is the principle that is used to formulate the bitstream decomposition, which primarily involves splitting up the bitstream into smaller sections to operate on the subsections of the stream, at which point, the results can then be recombined to get the overall DFT for that bitstream. This process allows the twiddle factors (the twiddle factor is an multiplicative augmentation at a given stage in an FFT algorithm) for these computations to be stored in LUT (Look Up Tables) allowing for evaluation and optimisation. This process is advantageous for the FPGA implementation as it is possible to parallelize this method and take advantage of the inherent parallel properties within FPGAs. Due to the shift away from in-operation computation of the twiddle factors, the LUT size can be unfeasibly large; however, decomposing into sets of two bits allows the LUT size to follow N^2 instead of 2^N which allows a feasible implementation [5], [6], [8]. This process is outlined below:

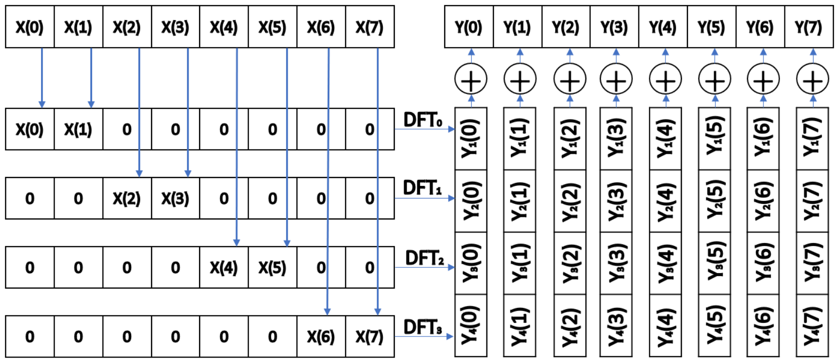
X(0) X(1) X(2) X(3) X(4) X(5) X(6) X(7)
X(0) X(1) 0 0 0 0 0 0
0 0 X(2) X(3) 0 0 0 0
0 0 0 0 X(4) X(5) 0 0
0 0 0 0 0 0 X(6) X(7)
DFT₀
DFT₁
DFT₂
DFT₃
Y(0) Y(1) Y(2) Y(3) Y(4) Y(5) Y(6) Y(7)
Y₁(0) Y₁(1) Y₁(2) Y₁(3) Y₁(4) Y₁(5) Y₁(6) Y₁(7)
Y₂(0) Y₂(1) Y₂(2) Y₂(3) Y₂(4) Y₂(5) Y₂(6) Y₂(7)
Y₃(0) Y₃(1) Y₃(2) Y₃(3) Y₃(4) Y₃(5) Y₃(6) Y₃(7)
Y₄(0) Y₄(1) Y₄(2) Y₄(3) Y₄(4) Y₄(5) Y₄(6) Y₄(7)

Figure 3.4.1: The access process and summation for bitstream decomposition, adapted from [6]

The input bitstream is piped to a different bitstream decomposition channel which outputs the partial DFT for each possible DFT from the bitstream. This is then summed together for each DFT. Thus, this computation results in all the DFTs that correspond to the section of the bitstream passed through being computed. The radix P controls the size of the bitstream section and B is the number of bits fed into each bitstream decomposition channel. Decreasing the number of bits per channel increases the RAM needed for the bitstream decomposition while increasing the number of additions [6]. The method for computing the DFT with this method is shown below

$$Y[l] = \sum_{k=0}^{K-1} Y_k[l], \quad l = 0, \dots, N-1 \tag{1}$$

$$Y_k[l] = \sum_{n=0}^{N-1} x_k[n] W_N^{nl}, \quad l = 0, \dots, N-1 \tag{2}$$

where the input byte is defined as

$$x_k[n] = \begin{cases} x[n], & \text{if } kB \leq n \leq (k+1)B - 1 \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

and:

$$W_N = e^{-j(2\pi/N)} \tag{4}$$

where N is the number of bits in each bin, this is the same as the radix ’P’, ’l’ indexes the specific DFT, ’k’ is the specific sub segment and ’K’ is P/B (the amount of segments)[10].

3.5 Transform Decomposition

Another key implementation strategy that is utilised in this project is the Transform decomposition [9] for delta-sigma modulated inputs. The transform decomposition, unlike many other FFT implementations, does not compute the full frequency spectrum to be able to compute a given bin. This is particularly advantageous as many implementations do not need the full spectrum and thus can reduce computation intensity and complexity. With respect to bat echolocation, this can be utilised to only compute the range from 10KHz to 120KHz[1], the relevant range over which bat echolocation is signalled.

The transform decomposition method has three main segments. The first is to reformat the input bitstream then compute the DFTs and recombine back into the FFT for the original bitstream. The process to reordering the bitstream involves initially grouping bytes of an arbitrary size such that the number of bytes is equal to a defined amount. After reordering, the byte size becomes the number of bytes and the byte length, the byte size. An example of this is shown below in figure 3.5.1.

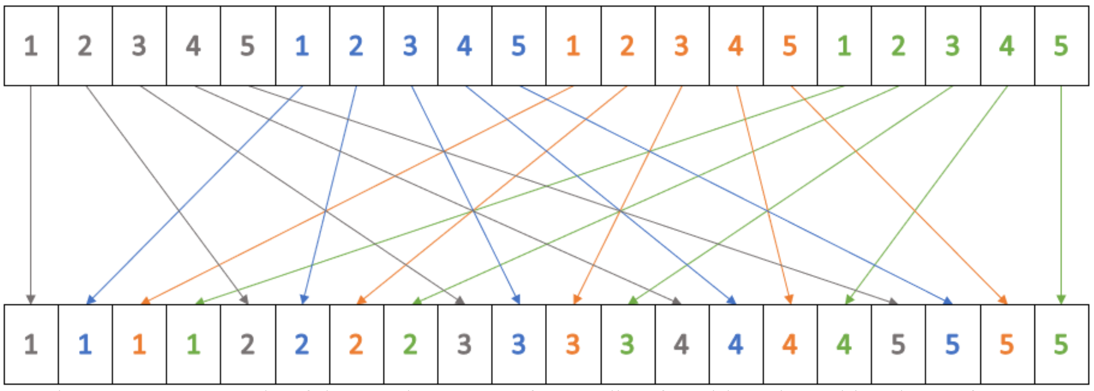

Figure 3.5.1: Example of the reorder process for a radix of 4 with 20 input bits, theory from [9]

where the above example outlines the reordering process for 4 bytes of 5 to 5 bytes of 4. The process to compute the DFT in the context of the transform decomposition is arbitrary but the final combination process for the real part of the FFT is shown below for a 4 byte 4 byte length bitstream.

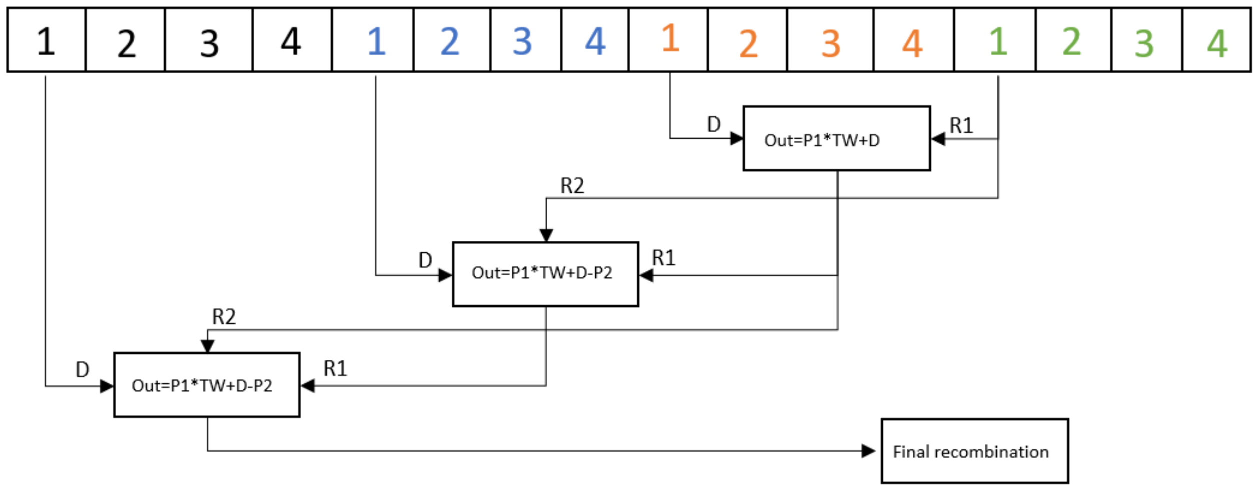

Figure 3.5.2: example final combination process (Real component, imaginary process is functional identical),theory from [9]

The process for transform decomposition combination is an accumulation algorithm that multiplies the previous value and performs addition before the next accumulation cycle. Note, however, that this only displays the process for the the real component, but the imaginary component is computed in an identical way. 'R1' is the first previous, 'R2' the second previous, and 'D' the real component of the DFT. 'TW' is the twiddle factor of the given series cycle which is found by computing; $2\cos(2\pi l/N)$, where 'N' is the number of total bits and the current DFT being computed is 'l'. The final combination combines the real and imaginary loops together by performing the following[10]:

$$\begin{aligned}\text{Real out} &= R1 - \cos(2\pi l/N) * R2 + \sin(2\pi l/N) * I2\\ \text{Imaginary out} &= I1 - \sin(2\pi l/N) * R2 - \cos(2\pi l/N) * I2\end{aligned}$$

'R1' and 'I1' are the last values of the recombination series for the real and imaginary components respectively [9], [10].

3.6 Pre-existing Implementation

Given the establishment of the previously mentioned algorithms, there is an implementation that combines the two methods to compute an FFT. The method by [5] uses both methods to combine the benefits for both implementations, the pre-computed nature of the bitstream and the selective output selection that is offered by the Transform decomposition. However, the nature of this implementation meant that the computation time of 580μs would not be met but, as evidence in [5] suggests, the two algorithm can be used to compute an FFT.

3.7 Gaps Identified

From literature review, a few key gaps were identified. The first is the lack of a hardware efficient solution able to be implemented for spectrogram analysis of delta-sigma input bitstreams. There is also a gap in this implementation specific to bat echolocation detection as it benefits from real time spectrogram analysis. Although some solutions are available they do not meet the real time timing requirements for implementation with the proposed system and thus another gap exists.

4 Methodology

As this thesis aims to provide a solution for a hardware efficient FFT for delta-sigma modulation with a focus on design, a design science methodology was followed. A literature review into current FFT algorithms to form the basis for the proposed model was conducted. Construction and development of the proposed algorithm to be implemented in hardware followed. To verify the the feasibility of the algorithm, a MATLAB model was constructed. To verify feasibility with this model, the results from the MATLAB model produced quantitative simulation data to asses against the known FFT. To verify the quality of the algorithms in hardware it was paramount to develop a VHDl model (using VIVADO IDE) to determine the implemented precision against the ideal model and the hardware resource usage using implementation usage results(using VIVADO IDE). To confirm the precision, simulations within the VIVADO IDE were conducted and subsequently exported to a MATLAB environment where the data could be assessed against the ideal one more directly. For the implementation, the results from the VIVADO implementation results tab were utilised. To further verify the design approach, a physical implementation was developed to gather real world results. This implementation to verify the algorithms feasibility computes a given FFT from a bitstream to be exported to MATLAB where simulations can be assessed against the ideal FFT. These mentioned methods make up the cornerstone of the design approach conducted throughout this project. The underlying simulation approach was to simulate all relevant signals in the FPGA with a variety of bit inputs to then be able to analyse the precision, accuracy and correctness. This is relevant as there is a risk that the FFT implementation will have too low a precision to be utilised and thus high quality verification is needed to detect and therefore address if any such situation occurs. Furthermore it is imperative to consider the ethics and sustainability of using certain FPGA devices as the final result may be implemented in the environment for the monitoring.

4.1 Methods

4.1.1 Algorithm

Firstly the construction of the algorithm was undertaken. The algorithm is an amalgamation of the bitstream decomposition method and transform decomposition method. The equations pertain to those found in the literature review (1) and figure 3.5.2 .

The implementation strategy was to have the transform decomposition form the basis of the algorithm but use the bitstream decomposition for computing the DFTs after the bitstream reordering. This strategy allows the advantage from both the bitstream decomposition (in pre-computing values) and the transform decomposition (in selection of relevant bins from 10KHz to 120KHz for the bat echolocation). An outline of how these methods are combined is shown below.

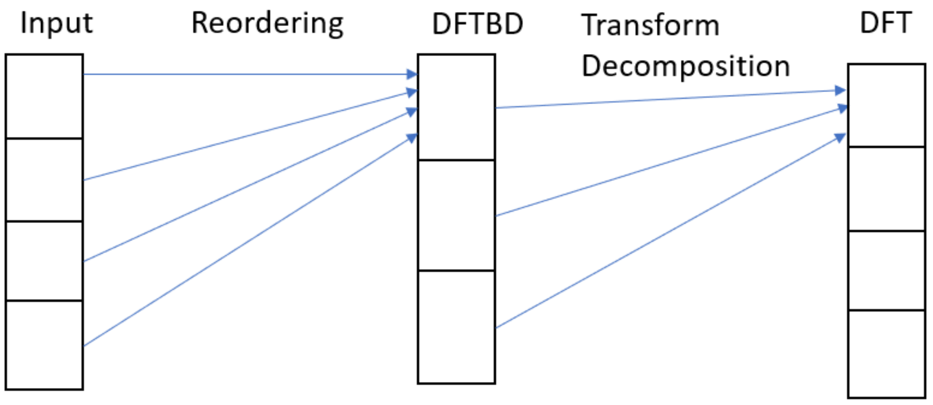

Figure 4.1.1: overall algorithm structure (high level)

From 4.1.1 the resulting implementation of these algorithms is similar to what was conducted in [5] but differs as it combines the reorder and DFT via bitstream steps rather than leaving them separate as this removes the need to store the reordered step. The choice of radix was chosen to be 16 (P=16) as any greater increases RAM considerably and any lower requires too many iterations of the transform decomposition. Also the number of bits used to perform the decomposed sequence was nominated as 2 since this reduces the RAM usage the most, bar B=1, which was not used as the limited RAM reduction is outweighed by the required adders doubling [6]. However this does limit the input bitstream size as it has to be divisible by the radix.

This algorithm can be parallelized as the DFT required for each bin repeats every radix (16 in this case) amount of bins. As the same DFT from the bitstream decomposition is used the algorithm can run simultaneously with the different Twiddle factor, only requiring the one DFT at any given time, thus forming the principle for the parallelization of the algorithm.

4.1.2 MATLAB Implementation

Upon completion of the algorithm structure, a MATLAB implementation was developed, designed to verify the feasibility of the algorithm for computing the FFT of the sigma-delta bitstream. The MATLAB method was used to be able to confirm that the proposed algorithm produces an accurate FFT and that the algorithm may be used to answer the question for how to construct a hardware efficient FFT. The MATLAB method, to ensure its relevancy for the later VHDL model to be implemented on the FPGA, was constructed as close as possible to the later implementation. This involved a later modification to allow the precision of the Twiddle factors in the MATLAB to match that present in the VHDL. This entails the use of a look up table-like structure to index the values from the bitstream decomposition, similar to that which was implemented later within RAM on the FPGA. This implementation was compared to the benchmark, the inbuilt MATLAB *fft()* function. For the MATLAB method to be useful in identifying the feasibility of the algorithm, it is assumed that it correctly emulates the process that would be performed in the FPGA by the hardware implementation. The MATLAB code can be found in the following git repository [11].

4.1.3 VHDL Implementation

This section details the design process to create the VHDL implementation of the FFT algorithm. The main three design areas were the input handling to get the microphone output, the storage mechanism for the twiddle factors and outputs from the bitstream decomposition and the method for implementing the algorithm loop efficiently. The fundamental state/flow structure of the implementation is shown below, and the implementation is available at [11].

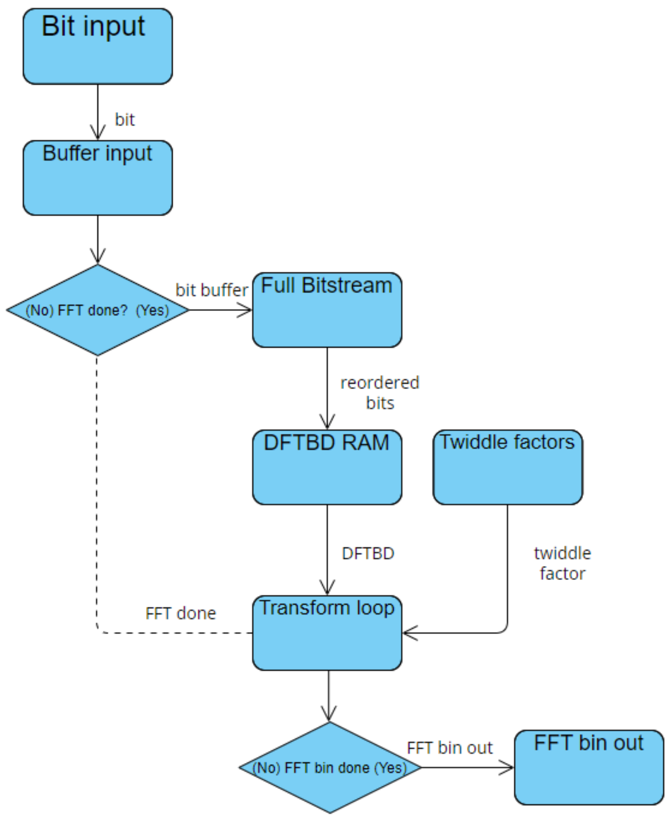

Figure 4.1.2: Overall state/flow structure of the VHDL implementation

Figure 4.1.2 depicts the high level flow of data from the bit input to the FFT out where every state tries at every possible instance to push its data to the next state unless stopped by a stated condition not being met. The data starts off from the microphone where it is sampled and stored in a buffer which, when full and the previous FFT having been completed, is pushed to the full bitstream. The full bitstream then sends the needed, reordered bits to the bitstream decomposition DFT, which are then combined with the necessary twiddle factors and sent to the transform decomposition loop. The FFT is then ported out.

4.1.3.1 RAM

The decision on how to store the bitstream decomposition and twiddle factors was addressed first. Two main ways for storing information on an FPGA exist; in look up tables or within the 'fabric' of block RAMs. For the purpose of reducing the amount of the FPGA utilised, a RAM approach was used as the reduction in FPGA usage was more important than allowing a higher degree of multiple access to the RAM data, as the design approach makes this unnecessary. To be able to use the RAM effectively and efficiently the inbuilt VIVADO IP cores were used to create the block RAMs with reference from [12]. From this single port ROM (Read Only Memory) style RAMs were chosen as the values for the bitstream decomposition and twiddle factor are read only. These are read only as the storing of these in the ROMs is akin to having them pre-computed and thus only accessing is required.

The accessing of the bitstream decomposition ROM is done so with an address that consists of which DFT of the 16 within the radix is needed and the relevant reordered bitstream byte. However, the RAMs were split up, so instead of one, eight were used, one for each set of 2 (as B=2) bits within the given decomposed sequence. A diagram of how the split up ROMs interact to form the DFT is shown below

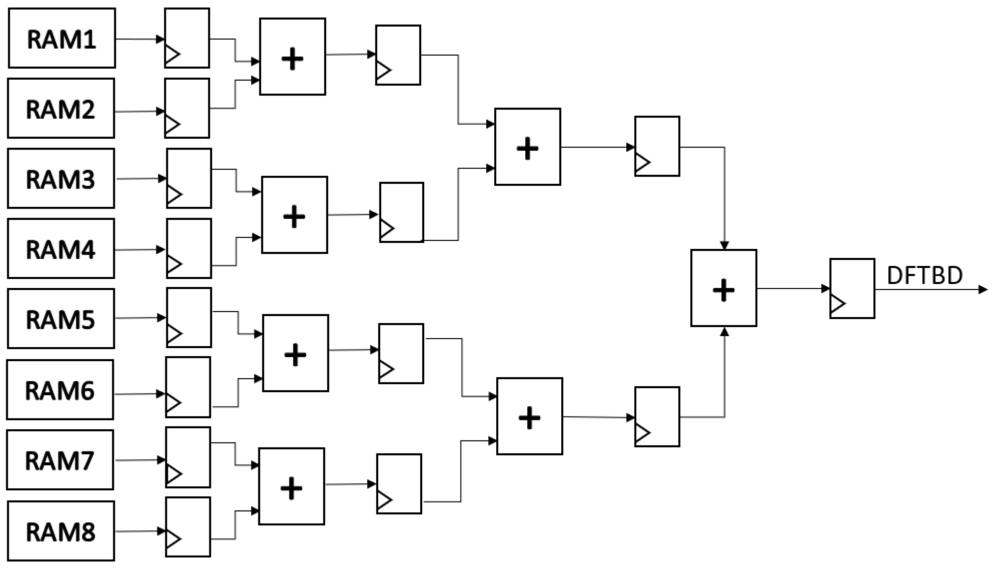

Figure 4.1.3: Example RAM structure for a radix of 16

From 4.1.3, by implementing a pipe lined additive structure it is possible to get a clock cycle per bitstream decomposition DFT instead of using the traditional method of one RAM which needs P/B clock cycles. This is possible because the access happens in parallel instead of in series. The ROM resources are also not affected as it is the same amount of data being stored.

4.1.3.2 Input Handling

To be able to interface with the microphone and subsequently compute the FFT from its output, appropriate input handling is required. This input handling predominately consists of a synchronized clock that is sent out to the microphone to clock it and to also allow the synchronised data store of its outputs. As the microphone samples on the rising edge of its clock it is then optimal to sample those bits within the FPGA on the falling edge of that same clock. However, to store the data a look up table approach was used. This approach is beneficial over a FIFO (First In First Out) or standard RAM approach as the amount of information stored numbers a likely maximum of 10210 bits (the input buffer and bitstream storage). This is because the most hardware efficient implementations are 2^n input size and 8192 is the largest possible given the microphone and real time constraints. So therefore this storage size is is small enough so as to not consume an onerous amount of FPGA resources and, from the implementation results, was able to fit on the FPGA. The benefit of using the LUT based approach is the ease of implementation for a shift register based design. This shift register based design consists of storing the new bit from the microphone on its falling edge to then shift it into the buffer on the FPGA system clock. The buffer then updates the bitstream every FFT cycle by shifting in the buffer as the most recent quarter of the held bitstream (the bitstream that the FFT will be performed over) thus maintaining the 75% overlap.

The input handling also reorders the bitstream from the full bitstream storage in the fashion outlined in 3.5.1 however only pulls the required bits for the current computation at any given time so as to reduce unnecessary storage and streamline data flow.

4.1.3.3 DSP and Overflow

To implement the transform decomposition algorithm, direct instantiation DSP (Digital Signal Processor) slices were used, with guidance from [13]. The advantage of using this is the DSP is optimised for pre-adder, multiplication and post-adder arithmetic such that it can compute these operations in a short window. By implementing the transform decomposition algorithm to conform to these operations, a fast clock cycle can be reached. From 3.5.2 the transform decomposition operation inputs are connected as shown in 4.1.4.

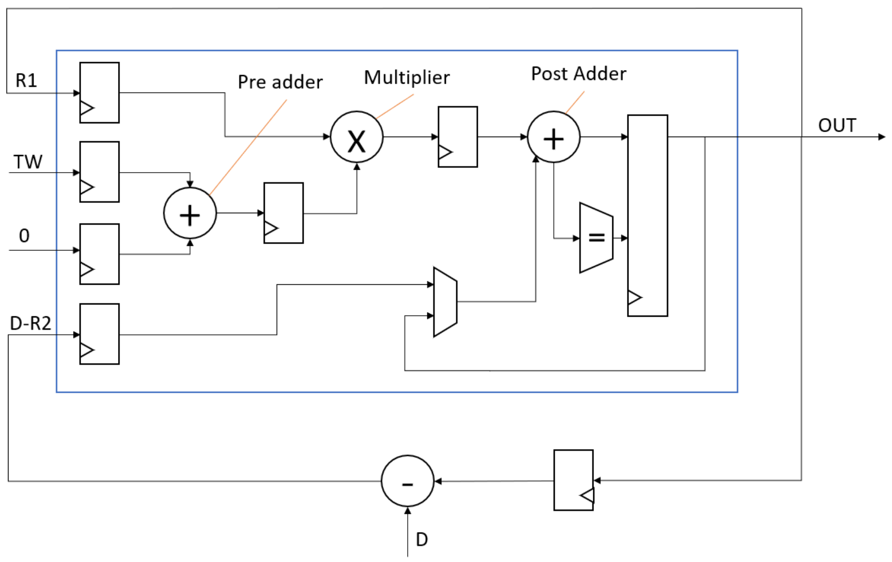

Figure 4.1.4: Diagram showing how the transform decomposition is mapped to the DSP for the real FFT (imaginary section is functionally identical

From 4.1.4 the previous iteration of the transform loop is fed into the input of the DSP and latched for the delay (to become the second previous iteration), subsequently combined with the DFT from the bitstream decomposition and into the DSP. Implementing the algorithm in this nature allows for a single clock cycle per cycle of the loop allowing the highest throughput. Pipelining is not possible as the internal steps of the DSP are not possible as the input requires the previous output so throughput is equivalent to latency in this instance.

Once the initial DSP loop was constructed the overflow needed to be considered as under certain conditions an FFT bin can exceed the maximum storable value with the give in precision. To alleviate this there is a component that detects if an overflow is likely soon and shifts the data right while incrementing a counter. This counter coupled with the data can then be combined to create the actual FFT value. The resulting FFT data then is similar to floating point in that there is now a mantissa and exponent as the overflow counter. Additionally this transform decomposition implementation can be parallelized such that each instantiation/component of it performs a set of 16 (radix) FFT bins in a row independently of another instantiation/component which will be performing another set of 16. The advantage of this is the same bitstream decomposition values can be used for each instantiation/component so that only the twiddle RAM has to be modified as it needs to pipe out all twiddle factors for each of the parallel instantiation/components per clock cycle, the size of twiddle RAM does remain unchanged, however the width increases and depth decreases by the same proportion as the number of instantiations/components. Therefore this means that the FFT bin count is limited such that it needs to be divisible by the radix multiplied by the number of parallel instantiation/components. The FFT computation time then is defined as:

$$T = \frac{16Mt^2 \times F}{PP_aC} \tag{5}$$

Where ’M’ is microphone clock, ’t’ the bitstream input time, ’F’ the accumulated frequency range of the bins needed, ’P’ is the radix, P_a the number of parallel components and C is the FPGA clock. Then for an FFT to be real time the computing time must be less than ’t’ (bit input time).

4.1.3.4 Timing Considerations

To be able to increase the operable clock rate and be able to operate the FPGA at the highest usable clock rate, certain design considerations needed to be made. Firstly, the most critical loop that controls the rate of FFT generation is the internal loop that computes each step of the algorithm. This loop has an apparent restriction of 160MHz (further optimisation may be performed to reduce net delay) as this is the minimal time to compute the additive and multiplicative steps required for the algorithm within the DSP. However, the whole design experiences a limit of 138MHz as the carry chains from the overflow calculations and its associated shifting operations appear to have timing limits of 138MHz.

4.1.4 Experimental Model

To be able to conduct more conclusive results on the effectiveness and viability of the design a physical implementation was conducted to be subsequently tested with external sources that simulate the behaviour of the microphone. This formed a closed loop where a script is set up to transmit the bitstream to the FPGA to then subsequently receive and evaluate. The experimental model is found in [11].

4.1.4.1 UART (Universal Asynchronous Receiver/Transmitter)

To be able to analyse the FFT externally, a means of exporting the result out the FPGA was required. As the FPGA communicated with a computer and to utilise the existing serial to USB interface on the Arty board a UART based communication system was utilised, the UART component was adapted from [14]. This communication scheme takes the resulting FFT bin that has previously been computed and stores it in a holding RAM. When parallel components are used the FFT bins are not computed in chronological order, so it also functions as a reordering step. Upon completion of the last bin of the FFT the data from this RAM is pushed to a UART buffer which is chipped out at the standard 115200 baud rate. This system allows a MATLAB script to then interpret the UART data for verification.

4.1.4.2 Arduino

Another consideration was the device for simulating the behaviour of the microphone. This device is needed as the data from the microphone can not be directly controlled and thus the precision of the result directly measured. A device to feed in a controllable and knowable output is therefore required. With respect to the data stream, the microphone is essentially a component that is externally clock driven and that samples on the rising edge of the clock. To maintain this behaviour an external Arduino device was used in SPI (Serial Peripheral Interface) slave mode as SPI slave mode devices are clock driven and can output data in the rising edge, like the microphone. To be able to sync up the Arduino to the state of the FFT in the FPGA so the the start of the FPGA incoming bitstream is the start of the Arduino SPI transmission, a secondary sync pulse was utilised. Although benefits are gained by undertaking this physical implementation pin on the assumption that the microphone can be accurately simulated by the SPI communication, this does pose a risk to the accuracy of the result. This is mitigated as much as possible by leaving the bitstream sampling network within the FPGA untouched when compared to the standard implementation, allowing for the most accurate substitution of the microphone.

4.2 Validation

To validate the various implementations at any given stage known inputs, such as full random bit-streams, an oscillation of '1's and '0's and the sigma-delta modulated sin wave input were passed through. This allowed testing the FFT output against a known output. For the FPGA implementation test benches were developed to monitor all stages of the implementation to assess all areas for any discrepancies. Once operation was as expected, validation of the precision and accuracy of the implementation began by comparing known solutions to the methods outputs by simulating the results. In extension, the physical implementation was also used to assist in validating the authenticity of the simulation/test-bench results to confirm if the predicted results are indicative of what they would be in hardware. The results from this final stage of validation are depicted in the results section below.

4.3 Stakeholder Engagement

The primary stakeholder is the end user for the hardware efficient FFT, the bat echolocation research team. However direct engagement with the stakeholder was limited and the initial criteria pertaining to the microphone and desired timing constraints for real time analysis were delivered via the supervisor. No further interaction was conducted as access to the stakeholder was not possible. In the later half of the project, engagement was made with another stakeholder (Fraser Williams, QUT research) on potentially using the developed FFT implementation for software defined radio applications, but only preliminary discussions were conducted and no concrete requirements established.

5 Results

This section details the results obtained for the various outcomes of the project intended to inform on the precision and hardware efficiency obtained from the given implementation. This section analyses results from the initial MATLAB full precision model, the FPGA implementation precision and hardware usage results and the physical verification of the simulations. All error depicted in this section is absolute as the error is not strongly related to the final value of the FFT bin, so relative error is disregarded. Note that a random bitstream is used for the results as it depicts average error more accurately. Additionally the parameters for the FFT outputs shown in this section are 8192 bits as this is the number of bits the microphone will output at 4MHz for a 75% overlap with a 512μs FFT. This time window reduction from 580μs is because, the range from 0 to 125KHz is only 256 bins (allows easier simulation) and includes the bat echolocation range and low frequency (overflow region). It is assumed that all relevant findings with the random bitstream are at minimum relevant for bat echolocation inputs, as bat echolocation inputs are essentially a frequency weighted random bitstream. Also See appendix D for FFT outputs with constant frequency (98KHz) bitstream input.

5.1 MATLAB Implementation

The MATLAB model was tested using the full precision model and with the twiddle factor precision the same as found in the FPGA implementation, and input parameters the same as mentioned above.

5.1.1 Results

To verify the precision of the proposed algorithm the MATLAB implementation was compared to the known inbuilt MATLAB function *fft()*.

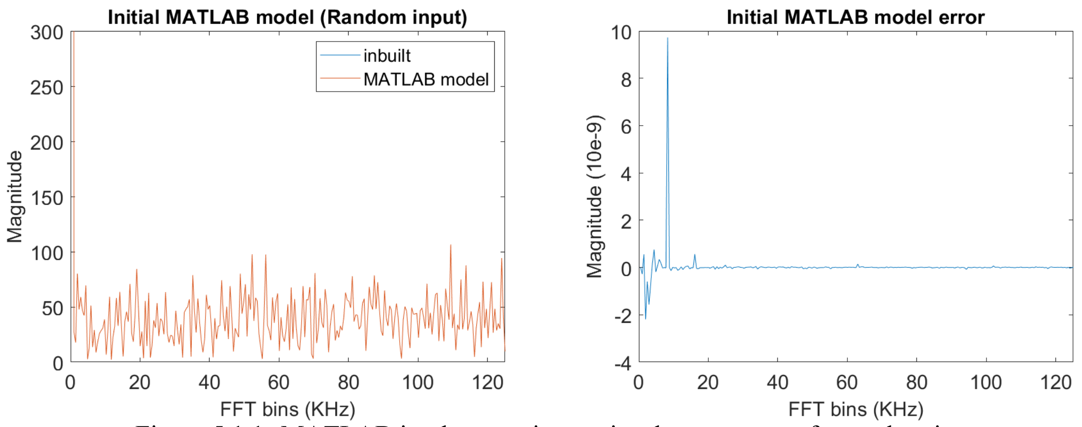

Figure 5.1.1: MATLAB implementation against known output for random input

5.1.1 shows the FFT results for the inbuilt MATLAB against the proposed algorithm implementation for the range of 0KHz to 125KHz.

5.1.2 Analysis

From 5.1.1 it can be seen that the known FFT very closely matched with the MATLAB full precision implementation of the algorithm as the error is very minimal. The minimal error in turn verifies that the algorithm is sound as an FFT implementation

5.2 VHDL Implementation

Once the VHDL implementation was created, simulations were used to verify the precision of the implementations. The simulation strategy for this is to pass a known bitstream from a text file, incrementing on the rising edge of the microphone clock as if it was the microphone. The resulting FFT is then stored in another text file adding each bin to it as they are processed, to be extracted then viewed using MATLAB. The simulations were run with the same settings as the MATLAB for the aforementioned reasons.

5.2.1 Results

To verify the FPGA implementation, the simulation results were compared against the initial MATLAB model with full precision and the twiddle factor precision set to 18bits, the same as is stored in the FPGA implementation twiddle factor RAMs.

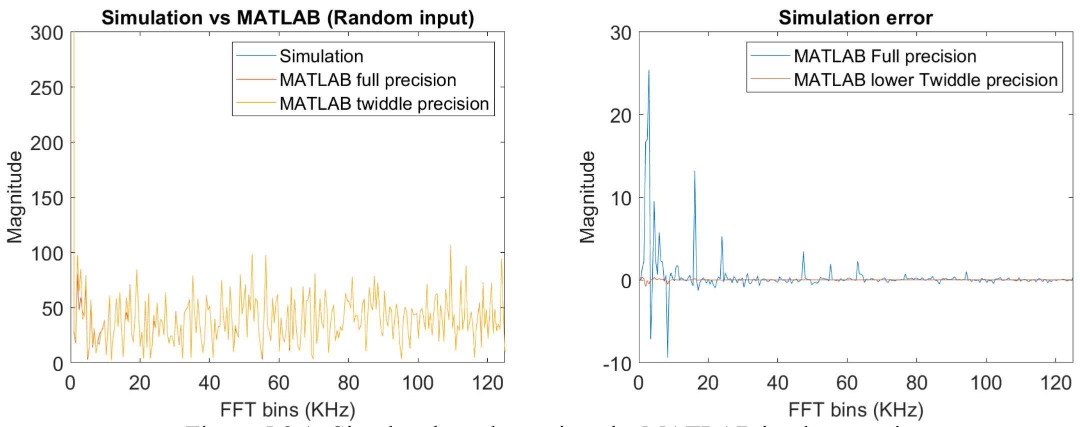

Figure 5.2.1: Simulated results against the MATLAB implementations

5.2.2 Analysis

From 5.2.1 is is evident that the simulation closely follows the MATLAB implementations. However, the simulation more closely follows the MATLAB implementation with the reduced twiddle factor precision as the error is noticeably reduced but still present. The resulting average precision of the simulation is then 17 bits and 22 bits respectively, more than adequate for a machine learning model as it can be trained with this error in mind. Additionally as there is no error for the DC bin there is no overflow issue, thus is is adequate.

The difference in error between the simulation and the full and reduced precision model is due to the accumulated error from the twiddle factors only being 18 bit. This error is accumulated due to having an 8192 bit input that requires 512 cycles of the algorithm per FFT bin. As the twiddle factor value gets multiplied with the previous value in the loop the error from only storing it at 18 bits gets multiplied. This explanation of the error is proven by the fact that in 5.2.1 the error between the simulation and the MATLAB implementation is significantly reduced when the twiddle precision is the same. As this resulting error is negligible, the implementation is as precise as the current algorithm implementation can be.

To further increase precision the algorithm implementation would have to be modified. Firstly, a possible change would be to change how the DSP is interwoven into each loop lifting the maximum requirement of the 18 bits for the second multiplier, allowing for the twiddle factor to be stored at a higher precision and thus in its error. Secondly, by increasing the radix there would be less loops required for each and and thus a smaller accumulation of error. However the increase in radix would require more RAM to store the outputs from the bitstream decomposition and this may be undesirable given the resource utilisation of the coupled machine learning element or other components on the same chip.

5.3 Hardware Utilisation

To be able to comment on the hardware efficiency of the of the FFT implementation, an adequate test bench for calculating the resource elements was required. The construction for testing the resources was to run the FPGA implementation through an out-of-context implementation strategy. An out-of-context strategy is a ’black box’ implementation strategy that mimics as if the FPGA implementation was a component in a larger design, which is how it will be implemented in the larger bat echolocation detection system.

5.3.1 Results

The following section shows the resource utilisation results for using different amounts of parallel instantiations/components with a 4MHz clock rate, 8192 bit input size computing 256 bins. This, along with the required amount of parallel components for given microphone clock speeds to conform to real time constraints are detailed. Both assume that the FPGA is run at 100MHz and the later has the number of input bits so as to maintain the 580μs input time, this along with 10KHz to 120KHz (256 bins) range of the FFT being computed in accordance with the range for the bat echolocation. Additionally An example of the implementation results for certain microphone clock frequency was also conducted to ascertain the accuracy of appearing trends. Only the FF (Flip Flops), DSP (Digital Signal Processor), BRAM (Block RAM), LUT (Look Up Tables) and LUTRAM (Look UP Table RAM) were monitored as these are the resources that vary and consume a meaningful amount in the implementation.

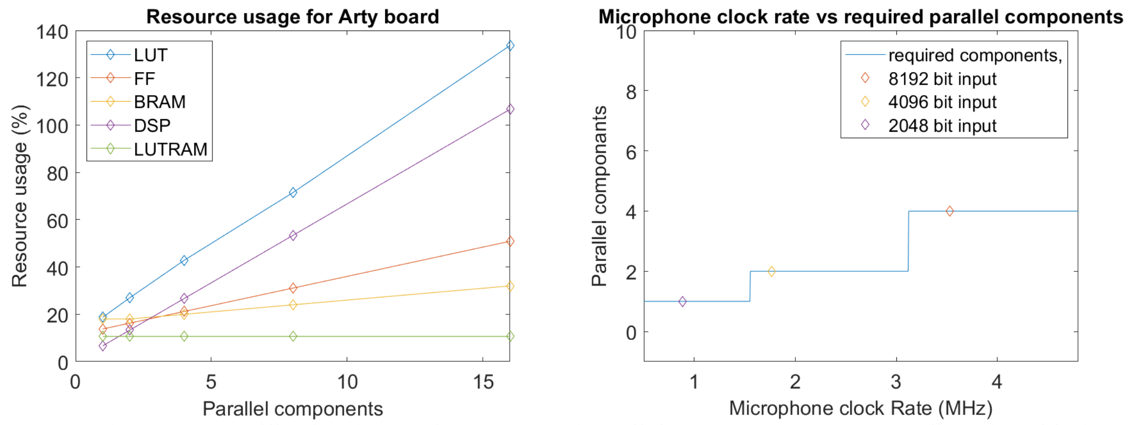

Figure 5.3.1: (Left) Resource utilisation for varying amount of parallel components (see Appendix B for table format) with an 8192 bit bitstream, (Right) Required minimum amount of parallel components for different microphone clock rates (for equation see appendix C).

Table 5.3.1: Table of the resource utilisation on the ARTIX-7 for different microphone clock rates with required bitstream lengths for 580μs input time, showing required amount of parallel components. The most efficient input sizes where chosen

Resources / Mic Clock rate | parallel components	LUT	LUTRAM	DSP	FF	RAM
883 KHz (2048 bits) | 1	2381 (11.4%)	0	6 (6.67%)	4207 (10.1%)	9 (18%)
1.77 MHz (4096 bits) | 2	5618 (27%)	1024 (10.7%)	12 (13.3%)	5746 (13.8%)	9 (18%)
3.54 MHz (8192 bits) | 4	8909 (42.8%)	1024 (10.7%)	24 (26.7%)	8815 (21.2%)	10 (20%)

From figure 5.3.1 it is evident that increasing the amount of parallel components increases the utilisation requirements. Other than utilising 16 parallel components, all conditions are currently implementable on the Artix-7 FPGA. From figure 5.3.1 the amount of required parallel components increases with an increase in the microphone clock rate. Table 5.3.1 generally follows the trend established in figure 5.3.1 for all resources except the LUTs and FFs and LUTRAMs which vary due to the changing microphone clock and subsequent bitstream size.

5.3.2 Analysis

A further analysis of the implementation hardware utilisation in figure 5.3.1 results in the following: The most significant resource types from a utilisation standpoint are the LUTs such that implementing 16 parallel loops uses more than is available on the ARTIX-7. The factors of least concern are the BRAM and LUTRAM utilisation as they are utilised the least. As DSP utilisation is important for potential machine learning implementations, increasing the amount of parallel components in the FFT implementation may restrict its implementation size. Furthermore, as the implementation results assumed the bitstream was of size 8192 the LUT results may differ for a specific implementation with a different bitstream size. A larger bitstream will result in slightly high usage and vice versa. However, 8192 bits is a reasonable approximation as it corresponds to a clock rate of 3.53MHz, not so high that it is unlikely to be used and not so low that the precision over the bat echolocation range is too reduced and not indicative of likely use cases. However, figure 5.3.1 was run at 4MHz to differ from table 5.3.1 on all runs and to be able to determine if the microphone clock rate affects resource usage. As table 5.3.1 shows, the DSPs and RAM usage remain unaffected when varying the bitstream size as it is identical to figure 5.3.1. FF usage does vary slightly depending on bitstream size, due to higher flip flops required for the shift register input as the bitstream size increases. The LUT count does also vary as mentioned earlier, caused because the myriad of logic depends on bitstream size so that low bitstream sizes do deviate below the trend. The lack of LUTRAMs for the lower bitstream size is due to the inefficiency of storing the input bits in LUTRAMs at lower sizes.

From figure 5.3.1, increasing the microphone clock rate does increase the minimum required amount of parallel components. Consideration of this and of the resources allowed for a machine learning implementation will need to be conducted, as even though the FFT does fit on the ARTIX-7 it may consume too many resources when using a high amount of parallel components at the higher microphone clock speeds. However, if other designs on FPGA chip remain small, it is possible to increase the clock rate to a maximum of 138MHz. This increase in the clock rate allows for a higher microphone clock rate with a smaller parallel component amount. This, coupled with the resource utilisation is intend to inform the restrictions for utilisation in the bat echolocation detection.

5.4 Experimental Implementation

To confirm that the results from the simulation are realistic and report accurately on what the actual implementation will yield, the physical implementation was tested against the simulation and the MATLAB model. The following tests conducted all run with the same input bitstream. However, upon investigation, a key limitation of the experimental implementation was found. The limitation is that the input bitstream must have a period so that it is repeated 8 times. Considering this constraint passing through a a valid bitstream allows the simulation results to be verified.

5.4.1 Results

The specific test conducted with the physical implementation was a random bitstream complying with the aforementioned limitations. This bitstream was passed through the external Arduino and monitored using MATLAB as well as passed through both the simulation and MATLAB models.

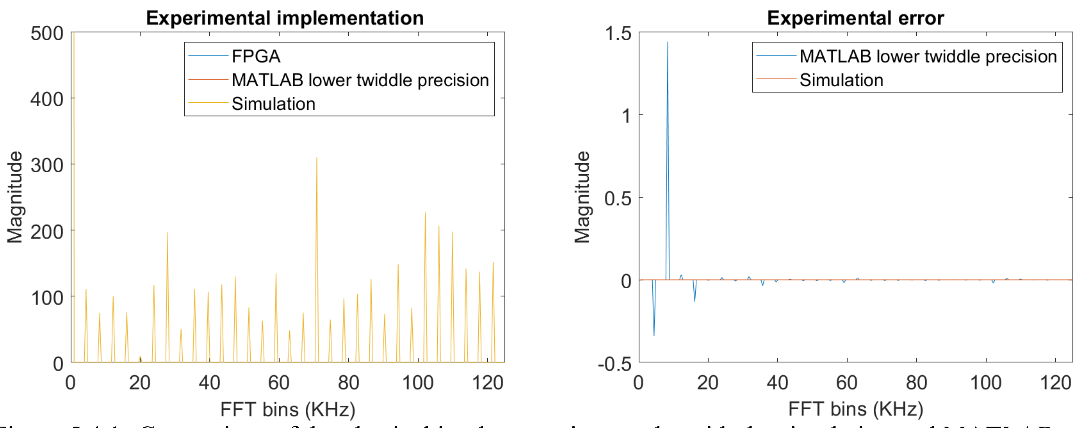

Figure 5.4.1: Comparison of the physical implementation results with the simulation and MATLAB results

5.4.2 Analysis

Evident from figure 5.4.1, the experimental implementation is the exact same FFT as in the simulation as there is zero error between them. As both the experimental and physical results are the same this verifies the simulation results. Also from figure 5.4.1 the MATLAB model with the reduced twiddle precision does differ from the FPGA result this is due to the small unmodeled rounding errors occurring in the FPGA (as data is not full precision) and not in the MATLAB model. This is to be expected and is the same small error seen in section 5.2. Even though there exists the limitation of the bitstream structure, the limited bitstream requires the same level of computation complexity as if no error exists no error in it it can be assumed that a completely random bitstream would be successfully computed.

6 Risks

Throughout the project there was key consideration of risk ethics and sustainability. The nature of the risks associated with this project are predominantly related to the delivery of the deliverables. These risks include the risk of algorithm being fundamentally flawed such that an implementation would not yield usable results and thus an ineffectual implementation would arise, this however was address by the preliminary implementation for the MATLAB model to verify that the algorithm can produce the correct FFT. Furthermore, the other main risk to project outcomes was that there would not be any verification able to be performed on the final implementation. Ineffective precautions to this risk may result in no verifiable results and such no significant project outcomes. To mitigate this risk the development of the FPGA implementation underwent a iterative process such that a rudimentary model was created initially so a minimal testing was at least attainable. In addition to this there is a risk that the precision of the FFT is too low to be useful, proper testing and verification of the simulation and experimental implementations alleviated the risk as the precision could be verified. Although minimal the health and safety risks pertain to this project there is a risk of static shock that may damage that FPGA and Arduino device, to mitigate this precaution for anti-static shoes were worn and the devices kept away from static producing devices.

7 Ethics

The primary ethical consideration of the project was to avoid the development of a device of prohibitive cost. This was addressed by having the design implementable on a cost effective FPGA such as the ARTIX-7 as this is a powerful, inexpensive model that will form the basis of a low cost field device. On completion of the project, the resulting hardware results show that within acceptable limitations the algorithm is implementable on an ARTIX-7 and thus the physical cost is reduced maintaining the ethical consideration [15]. As the final result from the project may be implemented in the environment using the method identified in [1], reducing maintenance is key, thus another reason the ARTIX-7 was chosen as it has lower power consumption and can be kept in the environment without disturbance or adverse environmental impact. All project data and findings are genuine and reproducible with no attempt to mislead by omission.

8 Sustainability

Sustainability was a key consideration throughout the development of the project. Sustainability in the traditional sense was not a primary factor of the project as the project is predominantly code focused. However, to sustain the use of the code and project findings, and allow the code to be utilised well into the future, certain features were included. The result of the code sustainability was the conclusion of the overflow system. This inherently isn't required for the special case of bat echolocation as that data does not need to include the DC condition and is spread over a large enough region of the spectrum so that large values for given bins are unlikely and thus so is overflow. However, overflow allows the implementation to be used for other purposes where overflow may occur far more often. Overflow was therefore included within the design, improving social sustainability [16]. The environmental aspect mentioned in the ethics section is pertinent also to sustainability [17].

9 Conclusion

9.1 Key Findings

The goal of the project was to develop a solution for a hardware efficient FFT for sigma-delta modulated inputs with maximum computation time of 580μs with a 75% overlap for the purpose of bat echolocation. This project consisted of the development of an appropriate algorithm to compute the FFT, a MATLAB proof of concept and an FPGA implementation with an appropriate experimental variant. These objectives were achieved and resulted in the following key findings. Firstly, the final implementation was found to suffer from an accruing of error due to the restricted data depth of the twiddle factors resulting in an average bit precision of 17. The FPGA implementation can allow the full micro[phone clock rate of 4.8MHz with an increasing number of parallel components. Running the microphone at speeds at or less than 3.4MHz only utilises 2 or less parallel components so there is minimal hardware usage at these speeds with enough space for potential machine learning elements. However, running at the maximum microphone clock speed is likely too large for a machine learning implementation (with the LUT count being excessive at that speed). The implementation itself was tested physically and, under the limitation of the testing system, operates at expected levels, confirming precision. These findings can form the basis of real time spectrogram analysis as the FFT is computed before the data for the new FFT is recorded. This opens up research avenues as it allows better data collection techniques since the processing can be done at the same time as the collection. Thus, the the aim of the project was achieved.

9.2 Future Work

To further investigate the creation of a Hardware efficient FFT for delta-sigma modulation there are a few key elements which should be investigated. Firstly, investigate the affect of changing the radix of the FFT has on the error. Secondly, to design a more rigid physical test to run a wider range of potential bitstreams for further validation. Thirdly, investigate reducing the twiddle factor RAM storage size along with more efficient storage of twiddle factors and bitstream DFT RAMs for a lower resource utilisation. Finally, frame domain windowing and log magnitude computing should be addressed to allow for easier use of the final FFT product.

10 References

[1] O. Mac Aodha, R. Gibb, K. E. Barlow, *et al.*, "Bat detectiveâ€"Deep learning tools for bat acoustic signal detection," en, *PLOS Computational Biology*, vol. 14, no. 3, B. Fenton, Ed., e1005995, Mar. 2018, ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1005995. [Online]. Available: https://dx.plos.org/10.1371/journal.pcbi.1005995 (visited on 04/13/2022).

[2] R. Stewart, "An overview of sigma delta ADCs and DAC devices," en, in *IEE Colloquium on Oversampling and Sigma-Delta Strategies for DSP (Digital Signal Processing)*, vol. 1995, London, UK: IEE, 1995, pp. 1–1. DOI: 10.1049/ic:19951371. [Online]. Available: https://digital-library.theiet.org/content/conferences/10.1049/ic_19951371 (visited on 04/18/2022).

[3] M. Shao, W.-K. Ma, Q. Li, and A. L. Swindlehurst, "One-Bit Sigma-Delta MIMO Precoding," *IEEE Journal of Selected Topics in Signal Processing*, vol. 13, no. 5, pp. 1046–1061, Sep. 2019, ISSN: 1932-4553, 1941-0484. DOI: 10.1109/JSTSP.2019.2938687. [Online]. Available: https://ieeexplore.ieee.org/document/8821391/ (visited on 04/18/2022).

[4] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of Computation*, vol. 19, pp. 297–301, 1965.

[5] F. Williams, "The Design and Implementation of a Hardware-Efficient Discrete Fourier Transform Algorithm for Field-Programmable Gate Arrays," English, Ph.D. dissertation, Queensland University of Technology, Brisbane, Nov. 2021.

[6] I. D. d. S. Miranda and A. C. d. C. Lima, "An FPGA-Oriented FFT Algorithm for Sigma-Delta Signals," en, *Circuits, Systems, and Signal Processing*, vol. 39, no. 5, pp. 2459–2472, May 2020, ISSN: 0278-081X, 1531-5878. DOI: 10.1007/s00034-019-01265-0. [Online]. Available: http://link.springer.com/10.1007/s00034-019-01265-0 (visited on 04/13/2022).

[7] C. Ingemarsson and O. Gustafsson, "SFFâ€"The Single-Stream FPGA-Optimized Feedforward FFT Hardware Architecture," en, *Journal of Signal Processing Systems*, vol. 90, no. 11, pp. 1583–1592, Nov. 2018, ISSN: 1939-8018, 1939-8115. DOI: 10.1007/s11265-018-1370-y. [Online]. Available: http://link.springer.com/10.1007/s11265-018-1370-y (visited on 04/13/2022).

[8] A. Barnwell, "Hardware Efficient Implementation of FFT (EGH400-1 Project Scope)," Project Propsal, Queensland University of Technology, Brisbane, May 2022.

[9] H. Sorensen and C. Burrus, "Efficient computation of the DFT with only a subset of input or output points," *IEEE Transactions on Signal Processing*, vol. 41, no. 3, pp. 1184–1200, Mar. 1993, ISSN: 1053587X. DOI: 10.1109/78.205723. [Online]. Available: http://ieeexplore.ieee.org/document/205723/ (visited on 04/23/2022).

[10] A. Barnwell, "Hardware Efficient implementations of FFTs (EGH400-1 Progress Report)," Progress Report, Queensland University of technology, Brisbane, Jun. 2022.

[11] A. Barnwell, *AlexBarnwell/EGH400_1_thesis*, git repository, publisher: github, Jun. 2022. [Online]. Available: https://github.com/AlexBarnwell/EGH400_1_Thesis.

[12] Xilinx, *Block Memory Generator*, en, Jun. 2022. [Online]. Available: https://www.xilinx.com/products/intellectual-property/block_memory_generator.html (visited on 06/09/2022).

[13] Xilinx, *DSP48 Macro*, en, Jun. 2022. [Online]. Available: https://www.xilinx.com/products/intellectual-property/dsp48_macro.html (visited on 06/09/2022).

[14] L. Voudouris, *Arty-uart*, 2016. [Online]. Available: https://github.com/lvoudour/arty-uart/blob/master/LICENSE.

[15] Engineers Australia, *Code of Ethics and Guidelines on Professional Conduct*, En, Nov. 2019. [Online]. Available: https://www.engineersaustralia.org.au/sites/default/files/2022-08/code-ethics-guidelines-professional-conduct-2022.pdf.

[16] Engineers Australia, *Implementing Sustainability: Principles and Practice*, 2017. [Online]. Available: https://www.engineersaustralia.org.au/sites/default/files/Learned%20Society/Resources-Guidelines%26Practice%20notes/Implementing%20Sustainability-Principles%20and%20Practice.pdf.

[17] Engineers Australia, *Sustainability Policy*, Nov. 2014. [Online]. Available: https://www.engineersaustralia.org.au/sites/default/files/2018-11/Sustainability-Policy.pdf.

[18] E. Ogier, *Delta Sigma modulator*, Mar. 2016. [Online]. Available: https://au.mathworks.com/matlabcentral/fileexchange/56166-delta-sigma-modulator.

A Appendix 1: Timeline

The project was completed by first designing the algorithm to be implemented along with any key considerations to be addressed in the implementation. Subsequently, the initial proof of concept utilising MATLAB was developed to determine feasibility. The FPGA model was developed using VHDL in VIVADO and tested through simulation and an experimental model. The experimental model test bench was designed in Matlab and Arduino for the peripheral, along with any necessary VHDL to modify the FPGA accordingly. The interim milestones were the delivery of the above listed elements along with an interim progress report. Overall the project involved a heavy amount of coding in MATLAB, C and VHDL, with the design of required tests/validation taking up a large part.

Frame domain windowing and log magnitude computing were removed from the project as time constraints and modification to the project timeline left them unachievable under the time frame.

Table A.0.1: The Finalised timeline showing changes from previous iteration required dependencies and start and end dates

NUMBER	DELIVERABLES	DESCRIPTION	START	END	DEPENDENCIES	PROGRESS	CHANGES
1	Literature Review	Conduct literature review of relevant background research	3/4/2022	1/5/2022	-	100%	
2	Conceptual Design	Form the overall data flow and preliminary algorithm structure	10/4/2022	6/5/2022	1	100%	
3	Detailed Design	Design the detailed data flow and key elements of the algorithm	10/4/2022	13/5/2022	2	100%	
4	Algorithm	overall design of the algorithm to implement the FFT	10/4/2022	18/5/2022	3	100%	
5	Proof of Concept	MATLAB implementation to prove the algorithm can produce a correct FFT	18/5/2022	31/7/2022	4	100%	
6	Interim Report	Report detailing the progress at the mid way of the project	28/5/2022	Written 12/6/2022 Oral 19/6/2022	3	100%	

7	First Protototype	Initial prototype for basic FPGA implementation	4/6/2022	7/8/2022	3,4	100%	removed the inclusion of frame domain windowing as was not achievable under the the frame
8	Second prototype & Final design	Second prototype that has more functionality and better conforms to bitstream size and latency requirements	8/8/2022	28/9/2022	7	100%	removed the inclusion of log magnitude computing as was not achievable under the time frame
9	Experimental Implementation	Design the testing rig and associate script and peripherals to validate the FPGA implementation experimentally	21/9/2022	15/10/2022	-	90%	This was added to the timeline as to provide a way to verify the authenticity of the simulations
10	Simulation/Results	gather simulation and results for the final FPGA implementation	16/10/2022	21/10/2022	5,8,9	100%	This deliverable was added to distinguish the simulation and results as a key element of the Project
11	Final Report Draft	Draft for the Final Report	30/9/2022	22/10/2022	4,5,8,9,10	100%	

12	Final Report & Presentation	Final Report and presentation detailing design and results in full.	30/9/2022	Written 26/10/2022 Oral 11/11/2022	11	50%	

B Appendix 2: Resource Utilisation Table

Table B.0.1: resource utilisation from figure 5.3.1 in table format

Resources (ARTIX-7 %) parallel components	LUT	LUTRAM	DSP	FF	BRAM
1	3900 (18.8%)	1024 (10.7%)	6 (6.67%)	5739 (13.8%)	9 (18%)
2	5617 (27%)	1024 (10.7%)	12 (13.3%)	6763 (13.8%)	9 (18%)
4	8909 (43.2%)	1024 (10.7%)	24 (26.7)	8815 (21.2%)	10 (20%)
8	14855 (71.4%)	1024 (10.7%)	48 (53.3%)	12923 (31.1%)	12 (24%)
16	27793 (133.6%)	1024 (10.7%)	96 (106%)	21166 (50.9%)	16 (32%)

C Appendix 3: Microphone Parallel Component Equation

the underlying equation for the graph in figure 5.3.1.

$$P_a = \frac{(4t \times 110k)\,(2 + \left(\frac{M \times 4L}{P}\right)}{t \times C}$$

Where P_a is the number of parallel components t is the bit input time P is the radix M is the microphone clock rate and C is the chip clock rate (nominally 100MHz). This equation was formed from the analysis of the behaviour within the FPGA. Note the 4 is from the 3 quarter reuse and the 110k is because 110k worth of FFT bins are computed over the range of 10KHz to 120KHz.

D Appendix 4: Results for Constant Frequency Input

This section is the same is the FFT outputs from the results section however with a constant frequency (approximately 98KHz) input bitstream instead of the full random bitstream. To delta-sigma modulate for the constant frequency bitstream input the toolbox from [18] was used.

D.1 MATLAB Implementation

MATLAB proof of concept against inbuilt *fft()* function with a single frequency input.

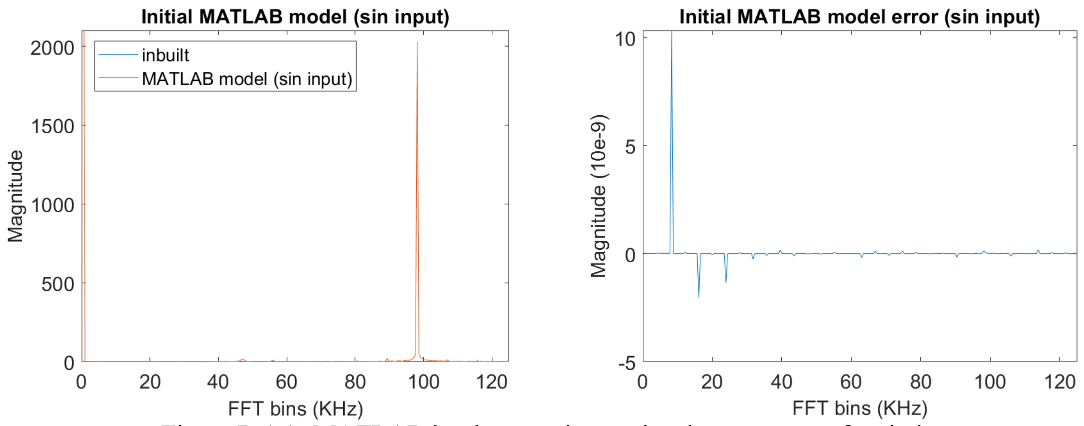

Figure D.1.1: MATLAB implementation against known output for sin input

Evident from D.1.1 the proof of concept works with sin inputs thus the same conclusion found in the result section.

D.2 VHDL Implementation

VHDL simulation output against MATLAB proof of concept with a single frequency input.

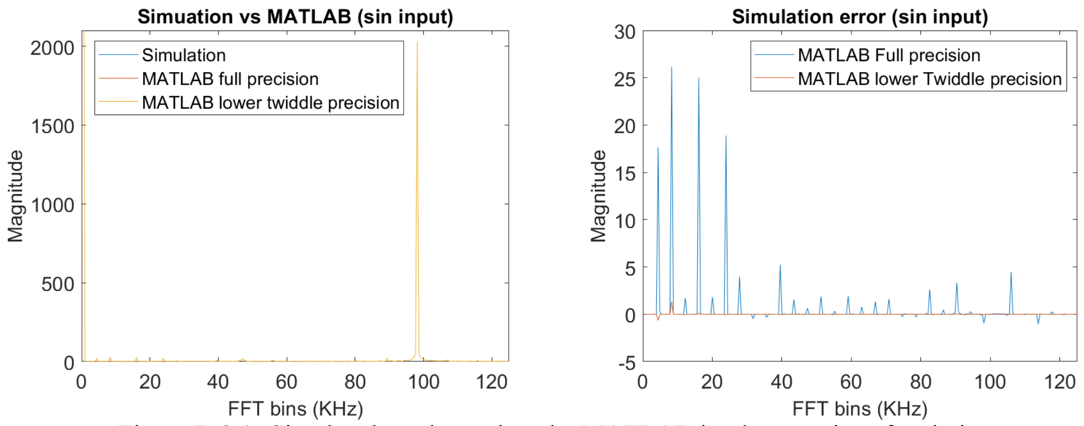

Figure D.2.1: Simulated results against the MATLAB implementations for sin input

Evident from D.2.1 the implementation has a 20 bit average precision this is higher than the full random bitstream due to the unique nature of this bitstream. However in general then same conclusions from the results section applies.
in inputs thus the same conclusion found in the result section.

D.3 Experimental Implementation

Experimental FPGA output against VHDL simulation with a single frequency input.

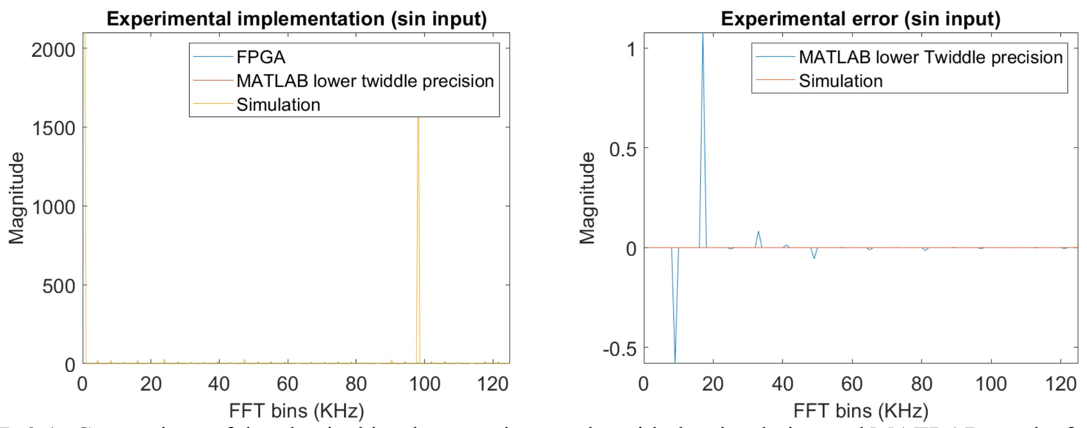

Figure D.3.1: Comparison of the physical implementation results with the simulation and MATLAB results for a sin input

Evident from D.3.1 the Experimental result is the same as simulation validating the simulation, thus the same outcome as in the results section.