ARM Implementation of FFT on the Zynq-7000 All Programmable SoC

Neeraj Kumar Cheryala[1], Sai Vamshi Jagarapu[2], Tharun Reddy Dodla[3]

Department of Electronics and Communication Engineering, 4th Semester
Vasavi College of Engineering
Hyderabad 500031, India

[1]neerajcher@gmail.com

Abstract— **Nowadays, the development of the Fast Fourier Transform (FFT) remains of a great importance due to its substantial role in the field of signal processing. In this paper, a radix-2 algorithm[1] of the FFT, 8 point is proposed and its software implementation on the Zynq 7000 all programmable SoC is discussed. Complete project is carried out in the Xilinx SDK and the code is cross-compiled on the ARM cortex A9 processor present in the Zynq SoC. The Xilinx Software Development Kit (SDK) is an Integrated Development Environment (IDE) for development of embedded software applications targeted towards Xilinx embedded processors. The SDK provides feature-rich C/C++ code editor and compilation environment which is exactly needed in this case since the algorithm is brought to life in the form of C code. The SDK also provides for System level performance analysis which is also depicted in this paper for the FFT algorithm. Zynq 7000 all programmable soc is chosen for the project since it comes with both the processing system and programmable logic. Hence it is crucial to compare the performance of PS only with that of PL in the picture since the integration of PL with PS on Zynq 7000 gives rise to higher bandwidth. Hence, this paper also analyses the performance of the ARM (PS) implementation of FFT algorithm on the Zynq 7000 All programmable soc.**

Keywords— **Fast Fourier Transform (FFT), Cooley-Tukey, Zynq-7000 APSoC, ARM Cortex A9, Xilinx SDK, Zedboard.**

I. INTRODUCTION

The fast Fourier transform (FFT)[2] is a mathematical algorithm for reducing the computational complexity of the Discrete Fourier transform (DFT) and is widely used for frequency analysis. The DFT is obtained by decomposing a sequence of values into components of different frequencies. This operation is useful in many fields, but computing it directly from the definition is often too slow. An FFT rapidly computes such transformations by factorizing the DFT matrix into a product of sparse (mostly zero) factors. As a result, it manages to reduce the complexity of computing the DFT from $O(N^2)$ to $O(N\log N)$ where N is the data size. The difference in speed can be enormous, especially for long data sets where N may be in the thousands or millions. By far the most commonly used FFT is the Cooley–Tukey algorithm. This is a divide and conquer algorithm that recursively breaks down a DFT of any composite size N into many smaller DFTs of sizes N1 and N2, along with O(N) multiplications by complex roots of unity traditionally called 'twiddle factors'. The best known use of the Cooley–Tukey algorithm is to divide the transform into two pieces of size N/2 at each step, and is therefore limited to power-of-two sizes but also can be extended to other sizes by appending zeros at the end. Hence this variant is called as radix-2 FFT.

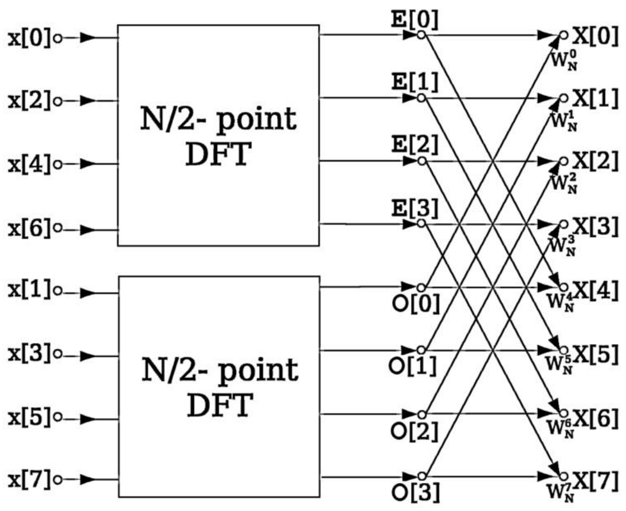

Figure 1: An example FFT algorithm structure, using a decomposition into half-size FFTs.

II. ZEDBOARD, ZYNQ-7000 AND XILINX SDK

The ZedBoard[3] is an evaluation and development board based on the Xilinx Zynq-7000 All Programmable SoC (AP SoC). Combining a dual Corex-A9 Processing System (PS) with 85,000 Series-7 Programmable Logic (PL) cells, the Zynq-7000 AP SoC can be targeted for broad use in many applications. The ZedBoard's robust mix of on-board peripherals and expansion capabilities make it an ideal platform for both novice and experienced designers. Zynq-7000 devices are equipped with dual-core ARM Cortex-A9 processors integrated with 28nm Artix-7 or Kintex-7 based programmable logic for excellent performance-per-watt and maximum design flexibility.

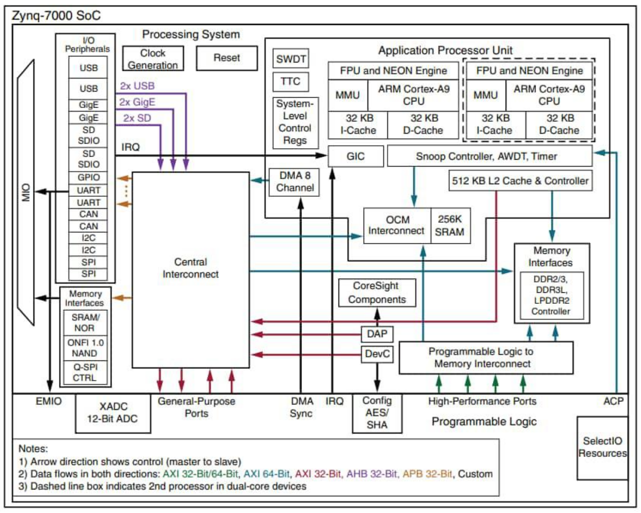

Figure 2: Zynq-7000 SoC architectural overview

With up to 6.6M logic cells and offered with transceivers ranging from 6.25Gb/s to 12.5Gb/s, Zynq-7000 devices enable system level performance through optimized architecture and deliver lowest system power. The Xilinx Software Development Kit (SDK)[5] provides a complete environment for creating software applications targeted for Xilinx embedded processors. It includes a GNU-based compiler toolchain, JTAG debugger, flash programmer, middleware libraries, bare-metal BSPs and drivers for Xilinx IP. Xilinx SDK also includes a robust IDE for C/C++ bare-metal and Linux application development and debugging. Based upon the open source Eclipse platform, SDK incorporates the C/C++ Development Toolkit (CDT). It is an all-in-one suite that serves the entire software team from concept to deployment.

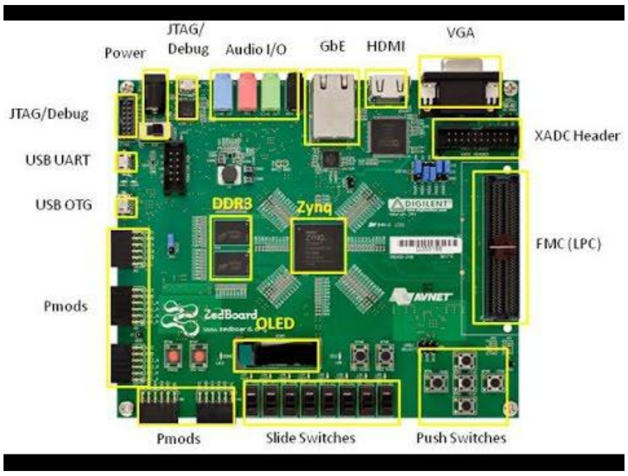

Figure 3: Zedboard and I/O peripherals

III. EIGHT POINT RADIX-2 DIT FFT

The algorithm repeatedly divides the time-domain sequence into two parts. In the first stage of decimation, the input sequence say x[n] of size 8 is divided into two parts f1[n] and f2[n] where f1[n] contains even-numbered samples and f2[n] contains odd-numbered samples i.e. f1[n]=x[2n] and f2[n]=x[2n+1] for n=0,1,2,3. Now, we are left with performing two 4-point DFTs for the obtained two parts. In the second stage of decimation, f1[n] and f2[n] are in turn divided into two parts each (even and odd). As a result, each 4-point DFT is calculated by taking two 2-point DFTs. Finally, each 2-point DFT is calculated simply as per the formula of DFT which is just a linear combination of the corresponding samples present in the even and odd parts of f1[n] and f2[n]. This algorithm can be described mathematically by the following equations:

The equation for N-point DFT is:

$$X[k] = \sum_{n=0}^{N-1} X[n]e^{-\frac{jk2\pi n}{N}} \qquad \textit{where } k=0,1,2,...N-1$$

After decimation, we get

$$X[k] = \sum_{m=0}^{N-1} f1[m] * \frac{Wn^{km}}{2} + Wn^{k} \sum_{m=0}^{\frac{N}{2}-1} \frac{Wn^{km}}{2} * f2[m]$$

where Wn is Twiddle factor

$$i.e. \quad X[k] = F1[k] + Wn^{k} * F2[k]$$

$$and \quad X\left[k+\frac{N}{2}\right] = F1[k] - Wn^{k} * F2[k]$$

After the second stage of decimation, we get

$$F1[k] = V_{11}[k] + \frac{Wn^{k}}{2} * V_{12}[k] \ \textit{where } k=0,1,2,...,(N/4-1)$$

$$and \quad F1\left[K+\frac{N}{4}\right] = V_{11}[k] - \frac{Wn^{k}}{2} * V_{12}[k]$$

The odd numbered samples after second stage of decimation are transformed as

$$F2[k] = V_{21}[k] + \frac{Wn^{k}}{2} * V_{22}[k]$$

$$and \quad F2\left[K+\frac{N}{4}\right] = V_{21}[k] - \frac{Wn^{k}}{2} * V_{22}[k]$$

where k=0,1,2,...,(N/4-1)

The above process is similarly carried out for all the samples in the time-domain sequence to get the corresponding 8-point DFT. The data flow diagram for the above 8-point DFT using the Cooley-Tukey FFT algorithm is shown below:

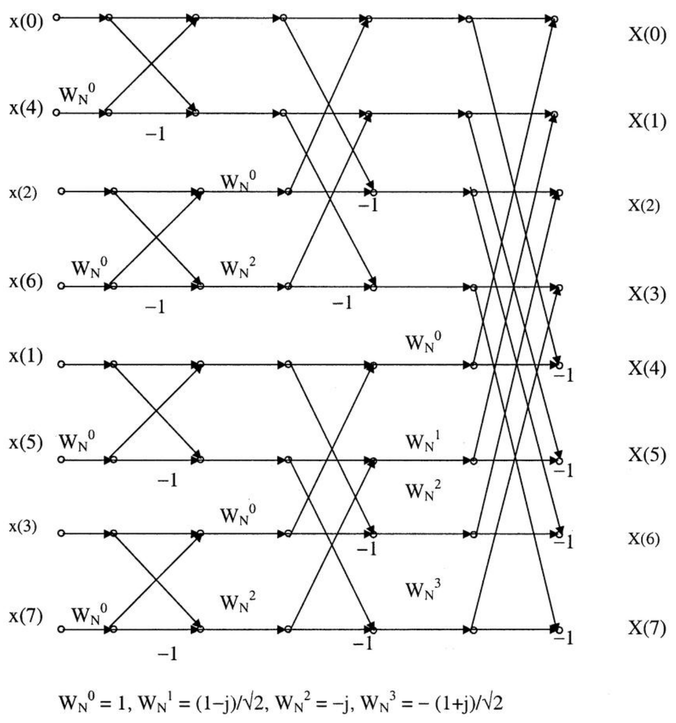
x(0)
x(4)
x(2)
x(6)
x(1)
x(5)
x(3)
x(7)
X(0)
X(1)
X(2)
X(3)
X(4)
X(5)
X(6)
X(7)
$W_N^0 = 1, W_N^1 = (1-j)/\sqrt{2}, W_N^2 = -j, W_N^3 = -(1+j)/\sqrt{2}$

Figure 4: Data flow chart of 8-point radix-2 DIT FFT

IV. PERFORMANCE ANALYSIS

To evaluate the performance of the algorithm, we consider the execution time of the code on the ARM processor in terms of the number of clock pulses elapsed. For that, we make use of 'time.h' library of the C standard library. Hence by knowing the frequency at which the processor is running, we can compute the execution time of the code. Other alternative to this method of calculating the execution time is to utilize the processor's internal timers since all the versions of Xilinx SDK may not support the 'time.h' library. Zynq SoC contains many timers called the global timers (64-bit) and Snoop Control Unit (SCU) timers. Each processor core has seperate SCU timers which are of 32-bit width. There are also two triple timer counting blocks each of which has 16-bit timer/counter. However, the easiest one to use is the global timer which does not require any explicit initialization because it is common for both processor cores present in the Zynq SoC. The SDK includes <xtime_l.h> for this purpose under the project explorer section. This library has two functions, one for setting the time in the Global timer i.e. XTime_SetTime() and the other to get the current time i.e. XTime_GetTime(). Note that the global timer runs at half the frequeny of the ARM cortex A9 processor at around 333.5 MHz. In this way, we can comprehend the performance of the software implementation for any given application. The execution time for the code is shown in the output section.

V. OUTPUT

Mostly, software implementation takes 10 times more time for execution when compared to hardware implementation on the Zynq-7000 SoC. Below is the output for an example sequence x[n]=[1, 1, 1, 1, -1, -1, -1, -1] feeded to the Zedboard via Xilinx SDK after debugging. It should be noted that the execution time shown below only considers the FFT code section but does not consider the time taken for data transmission between the windows machine (on which Xilinx SDK is installed) and the Zedboard.

```
Data: 1 1 1 1 -1 -1 -1 -1
FFT: 0   (2, -4.82843)   0   (2, -0.828427)   0
(2, 0.828427)   0   (2, 4.82843)
Execution Time: 258833.891249 usec
```

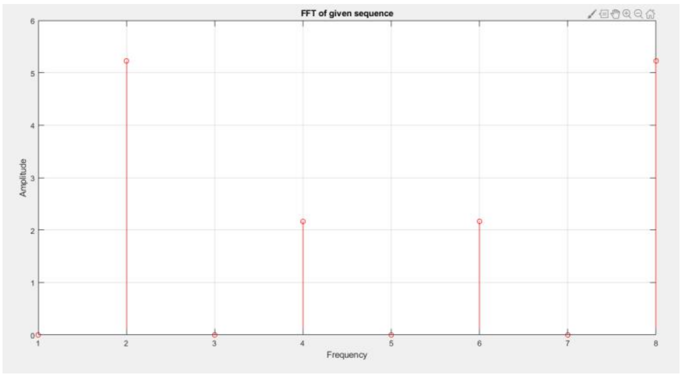

Figure 5: Stem of magnitude of FFT of the example sequence obtained using MATLAB coder.

VI. THEORETICAL VERIFICATION AND CODE

The Google drive links given below take you to the hand-written notes on the entire methodology adopted for the algorithm and theoretical verification at last. The latter contains the C code in a text file.

- https://drive.google.com/open?id=1BUHY5mlFNU5w9gR1PGSnFIwZCvXydTrE
- https://drive.google.com/open?id=1zhqG8j4G_NqbktNfUuOcuC2mZZghXY3K

VII. CONCLUSIONS

In this paper, we proposed an algorithm for radix-2 Decimation in Time (DIT) FFT by taking an example of an 8-point DFT. Software implementation of the algorithm on the ARM Cortex A9 processor 00 present in the Zynq-7000 APSoC is also briefly explained. Performance of the implementation is also depicted in terms of execution time. In addition to this, we can also perform hardware implementation of the same algorithm on the PL part of Zynq SoC and compare the results obtained with those of the software implementation to decide which is of better performance in terms of execution time and power consumed.

VIII. ACKNOWLEDGEMENT

We sincerely thank our faculty V. Krishna Mohan sir, Assistant professor, Department of Electronics and Communication Engineering, Vasavi College of Engineering for giving us a golden opportunity to study the software implementation of such a ubiquitous algorithm on Zynq-7000 APSoC. We are now a bit more accustomed to the Xilinx SDK and are looking for more such projects in the future.

IX. REFERENCES

[1] William H. Press, Brian P. Flannery, Saul A. Teukolsky, William T. Vetterling, "Numerical Recipes in C: The Art of Scientific Computing, Second Edition", Cambridge University Press.

[2] Wikipedia contributors, "Fast Fourier transform," Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Fast_Fourier_transform&oldid=951652065.

[3] Zedboard (Zynq Evaluation and Development) Hardware User's Guide, Version 2.2, 27 January 2014.

[4] Xilinx Community, Zynq-7000 SoC Data Sheet: Overview, DS190 (v1.11.1) July 2, 2018.

[5] https://www.xilinx.com/products/design-tools/embedded-software/sdk.html

[6] Kizheppatt Vipin, https://scholar.google.co.in/citations?user=F1bMG5AAAAAJ&hl=en.