HW/SW-Codesign Lab
SS 2024

Dr. Emil Matúš emil.matus@tu-dresden.de
Viktor Razilov viktor.razilov@tu-dresden.de

April 2024

1 Introduction

1.1 Tensilica Customizable Processor

The HW/SW-Codesign Lab uses the reconfigurable processor flow from the company Tensilica to demonstrate the potential of HW/SW-Codesign. The Tensilica Instruction Extension (TIE) is a description language that allows to modify the standard RISC/VLIW pipelines of the different baseline processors. Out of the TIE description, a modified compiler, assembler, linker, debugger, and a cycle accurate processor simulator are generated. Thereby architecture modifications and their impact to the software (i.e. C/C++) can be analyzed with short turn-around times and manageable complexity. An energy and area estimator can be used to infer some estimates on the hardware complexity of the modified architecture, without the need for setting up a complete RTL synthesis flow.

The complete flow provides an eclipse GUI frontend for visual debugging. All the tools are based on the standard GNU binutils [1] or use a similar command line frontend. The experience you gain with the Tensilica tools are beneficial for any other embedded projects based on Linux. Other vendors like Altera NIOS [2] or ARM Realview [3] use the same structure for their software build processes, even though their flows provide not such kind of flexibility or maintainability from hardware perspectives.

1.2 Task Description

Within the scope of this lab, you will apply the TIE in order to improve the performance of an FIR filter. Further, you draw on this knowledge to optimize an FFT implementation in regards to the runtime. In summary, your tasks are:

1. Implement a Finite Impulse Response filter (FIR) design (direct form) by using extended and customized TIE instructions. Apply optimization techniques such as Fusion, SIMD, and FLIX. Analyze performance in regards to runtime. As an optional task, compare different filter implementations, e.g., transposed form. Notice, this task need not take part in your final report.

2. Implement a Fast Fourier Transform (FFT) and an inverse FFT (IFFT) design by using extended and customized TIE instructions. Due to a similar HW structure as the FIR filter, apply your knowledge from task 1. Also, compare different implementations such as the Decimation in Time (DIT) and Decimation in Frequency (DIF) algorithms. Analyze your performance results in terms of cycle count and throughput. Notice, this task is the main part of your final report and will be included in the valuation.

2 Setting up the environment

Open the following URL with a browser:
`https://entmrdwa.ifn.et.tu-dresden.de/RDWeb`

You will be asked for your credentials to connect to the Xtensa Xplorer. Choose a location to store the Xplorer workspace in your local path:
`U:\workspace`

At first you should notice our special Xtensa processor configuration called "coreLX5_hwswcd" in the "System Overview" (lower left part of the GUI) which is used for all tasks of the lab. To get some information about the processor just double click on the configuration. Under `Configuration Installed Build` → `View Details` you can obtain various information about the current processor. Just go through the configuration and make yourself familiar with it. It is a nice occasion to repeat some fundamentals from computer architecture/OS lectures. As the Xtensa architecture is quite flexible, most parameters can be tuned for the application.

In the "Project Explorer" you will already find a small hello world project example. Please do the following steps:

- Compile the project.
- Run the program.
- Profile the program.
- Set compiler optimization (`Right click on project` $\rightarrow$ `Build Properties` $\rightarrow$ `Optimization`) to -O3 (highest level).
- Compare the runtime to -O0.
- Have a look a the instruction pipeline in the `main` routine.

Please use the Documentation under `Help` $\rightarrow$ `PDF documentation` as reference and to get some more insight into the tools. Please answer following questions:

- How many cycles takes the `main`-function (-O0/-O3)?
- Which sub-functions are called?
- What do they do?
- What is the most time consuming instruction in `main`?
- What's the purpose of this instruction?
  See `Xtensa Instruction Set Architecture (ISA) Reference Manual` for reference.

3 Tasks

3.1 FIR Filter

First, take the files for the FIR filter from the network directory. Then, create a new Xtensa C/C++ Project in the Xtensa Xplorer. The input values are read from an input file and the output is written to a separate file. The filter has 8 coefficients: $-3, 4, -3, 2, 5, -1, 4, 8$

Particularly, the following subtasks should be accomplished:

- Development of a TIE instruction with fused multiplication and addition operations.
- Vectorization technique of FIR filter algorithm (SIMD).
- Verification of results using Matlab reference.

The 8-tap real FIR filter is already implemented with pure C code. The following part gives some mathematical background about the FIR filter and its possible parallelization techniques. A formal equation of an FIR filter is given by

$$y_n = \sum_{i=0}^{N-1} b_i x_{n-i} \tag{1}$$

with $N = 8$. b_i are filter coefficients, y_n and x_n are output and input signals, respectively.

There exist various vectorization strategies for efficient implementation of FIR filters on vector processor. An example is described by the following dot product, where the 8 multiplications and subsequent additions are executed in parallel to compute y_n for a specific n from x_n up to x_{n-7}.

$$y_n = b \cdot x_{n-i} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \cdot \begin{bmatrix} x_n \\ x_{n-1} \\ x_{n-2} \\ x_{n-3} \\ x_{n-4} \\ x_{n-5} \\ x_{n-6} \\ x_{n-7} \end{bmatrix} \tag{2}$$

This dot product can be executed in one clock cycle by one TIE instruction. In every computation of y_n the coefficients b_i stay the same. When computing y_{n+1}, the input signal vector x_n is only shifted by one element and filled up with one new element to obtain x_{n+1}.

Use the TIE `User's Guide` and `Reference Manual` to make yourself familiar with the basic TIE concepts. So far concentrate on the basic ones (`Reference Manual` chapter 1 to 5.7). At the end answer the following questions:

- What is the logic gate count for the instructions?
- How many cycles does it take to process one sample?
- What is the maximum filter throughput?

3.2 Fast Fourier Transform (FFT)

Now, the Discrete Fourier Transform (DFT) is a FIR filter with input values x_k where the coefficients are replaced by complex values. These complex values are dependent on the frequency index m, the time index k, and the size of the DFT N. The DFT transforms a series of discrete data points from time domain into frequency domain. Assuming, N is a power of 2, the DFT is called FFT and can be executed much faster. The following paragraph provides more information about the FFT and its parallelization methods. Please also refer to the HW/SW-Codesign exercise script and to other resources such as [4][5] to get more information on this topic.

The formal equation of the DFT and the IDFT is given by

$$X_m = \sum_{k=0}^{N-1} x_k e^{-j2\pi \frac{km}{N}}, \tag{3}$$

$$x_k = \sum_{m=0}^{N-1} X_m e^{+j2\pi \frac{km}{N}} \tag{4}$$

The Cooley-Tukey algorithm for a Decimation in Time (DIT) radix-2 FFT is shown in the following equations.

$$\begin{aligned} X_m &= \sum_{k=0}^{N-1} x_k e^{-j2\pi \frac{km}{N}} \\ X_m &= \sum_{k=0}^{\frac{N}{2}-1} x_{2k} e^{-j2\pi \frac{(2k)m}{N}} + \sum_{k=0}^{\frac{N}{2}-1} x_{2k+1} e^{-j2\pi \frac{(2k+1)m}{N}} \\ X_m &= \sum_{k=0}^{n-1} x_{even,k} e^{-j2\pi \frac{km}{n}} + e^{-j\pi \frac{m}{n}} \sum_{k=0}^{n-1} x_{odd,k} e^{-j2\pi \frac{km}{n}} \end{aligned} \tag{5}$$

This algorithm divides the N-point FFT into an n-point FFT which computes the even indexed input values and an n-point FFT that computes the odd indexed input values (with $N = 2n$, N is power of 2). The remaining factor $W_n^m = e^{-j\pi\frac{m}{n}}$ is called twiddle factor. Now, the FFT can be computed much faster by applying the butterfly network depicted in Figure 1.

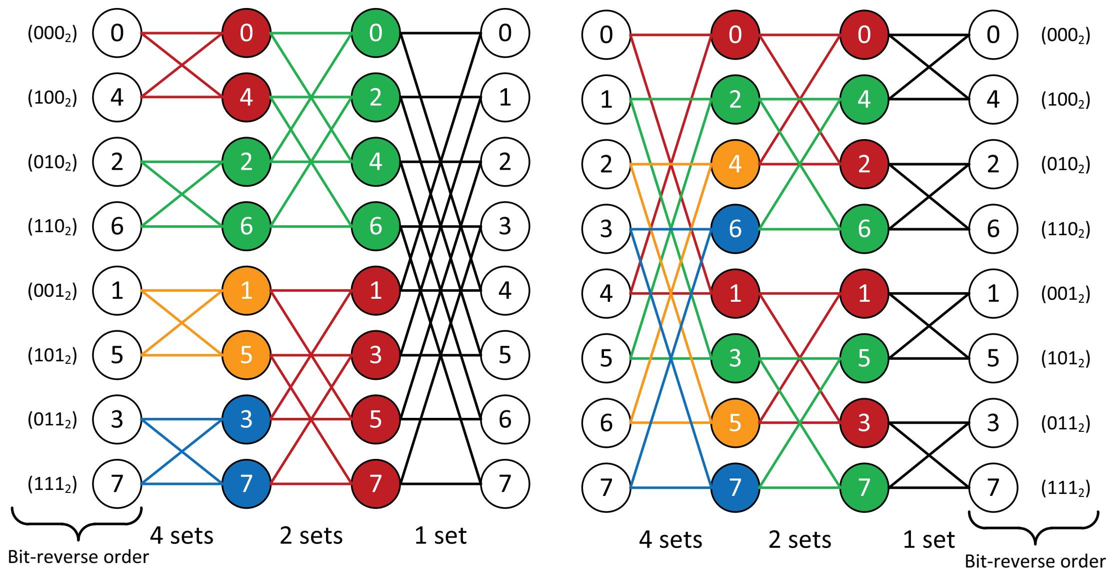

Figure 1: Full Network for an 8-point FFT DIT (left) and an 8-point FFT DIF (right)

The numbers in all nodes indicate the order of the input values. To achieve the right ordered result, the input values of the first stage have to be exchanged by an additional network which is not shown in the Figure. Each node consists of an adder and a multiplier. An even, an odd input value, and the twiddle factor are connected as depicted in Figure 2.

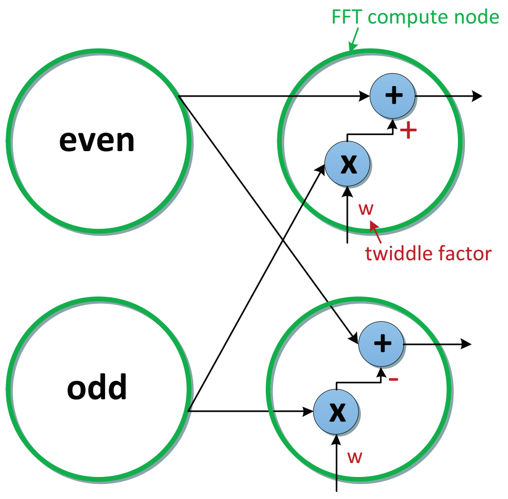

Figure 2: FFT DIT Compute Node

The Sande-Tukey algorithm for a Decimation in Frequency (DIF) radix-2 FFT uses the distribution into even and odd indexed values as well [6]. Now, the partitioning happens in the frequency domain as the following equations show.

$$X_m = \sum_{k=0}^{N-1} x_k e^{-j2\pi\frac{km}{N}}$$
$$X_m = \sum_{k=0}^{\frac{N}{2}-1} x_k e^{-j2\pi\frac{km}{N}} + \sum_{k=\frac{N}{2}}^{N-1} x_k e^{-j2\pi\frac{km}{N}}$$
$$X_m = \sum_{k=0}^{\frac{N}{2}-1} [x_k + x_{k+\frac{N}{2}}(-1)^m] e^{-j2\pi\frac{km}{N}} \tag{6}$$

By using Eq.(6) the frequency values for even and odd indices m (with $N = 2n$, N is power of 2), are given by

$$X_{2m} = \sum_{k=0}^{n-1} [x_k + x_{k+n}] e^{-j2\pi\frac{km}{n}} \tag{7}$$
$$X_{2m+1} = \sum_{k=0}^{n-1} [x_k - x_{k+n}] e^{-j\pi\frac{k}{n}} e^{-j2\pi\frac{km}{n}} \tag{8}$$

The remaining factor $W_n^k = e^{-j\pi\frac{k}{n}}$ is the twiddle factor for the DIF algorithm.

In the source code folder which you can download from the lab's website, you will find the pure C code of the FFT/IFFT DIT algorithm. When implementing the improved FFT network with TIE, try to consider the following aspects:

- Find a scheme to implement the preliminary sorting network.
- Reuse HW of compute nodes.
- Perform on independent input values of each network stage simultaneously.
- Make use of features such as SIMD and FLIX, i.e., compute multiple values within one instruction and execute multiple instructions within the same cycle, respectively.
- Create lookup table to prevent multiple loads of constants, e.g., for twiddle factors.

Present your performance results including the number of cycles, area consumption, and optimization level in the final report and indicate whether you did any changes or improvements in the actual given algorithms. Please do this as already stated for the FFT and IFFT each with DIT and DIF. It is recommended to develop the FFT for $N = 8$ points. Afterwards, the algorithm should work for selected N, e.g., $N = 256$ and $N = 1024$.

References

[1] Wikipedia, "GNU Binutils." http://en.wikipedia.org/wiki/GNU_Binutils, 2014.

[2] Altera Corporation, "Nios II." http://www.altera.com/devices/processor/nios2/ni2-index.html, 2014.

[3] ARM Ltd, "RealView Development Suite Standard." http://www.arm.com/products/tools/software-tools/rvds/index.php, 2014.

[4] Wikipedia, "Butterfly Diagram." http://en.wikipedia.org/wiki/Butterfly_diagram, 2014.

[5] Always Learn, “FFT network with Twiddle factors.” http://www.alwayslearn.com/default.htm, 2014.

[6] Openstax, “Decimation in Frequency Radix-2 FFT.” http://www.oercommons.org/courses/decimation-in-frequency-dif-radix-2-fft/view, 2014.