Implement of Many-Core System Midterm Project

109061564 馬婕芸

Introduction

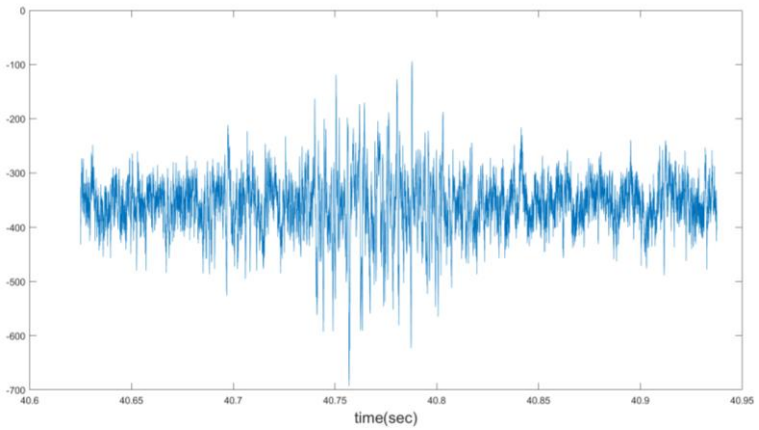
Spike-sorting DSPs are necessary to allow for the real-time processing of wireless and implantable neural recordings. Due to classification algorithm and high sampling rate, the original data size is too large to do wireless data transmission. The method to compress data is necessary. Since time and frequency information are both important when processing spikes data, discrete wavelet transformation (DWT) outperform fast Fourier transformation (FFT). In this project, I design a 2-level symlet-4 DWT kernel due to symmetric property of symlet-4 wavelets.

With high-level synthesis tool (Cadence® Stratus™), we can quickly design and verify RTL implementation from abstract IEEE 1666 synthesizable SystemC®, C, or C++ models. In this project, I build a synthesizable RTL of DWT kernel and do simulation for real data recorded from a mouse brain (10000 data points, with 32000Hz sampling rate).

Implementation details

1. 2-level symlet-4 DWT:

The figure on the right-hand side is a real spike data I used to do simulation. It is sampled at 32,000 Hz and has 10000 data points.

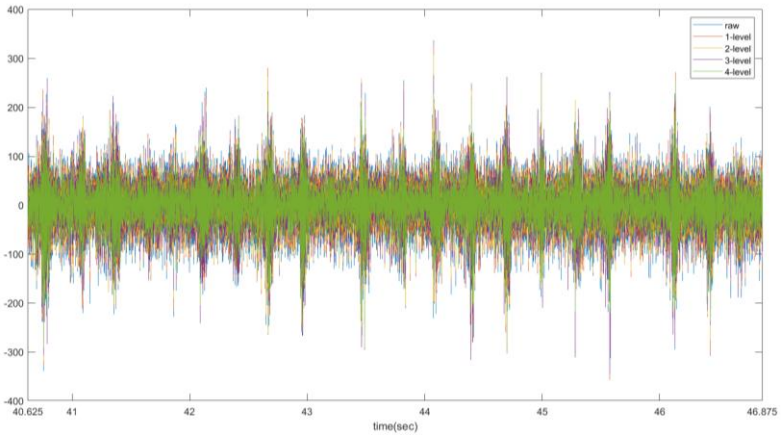


In terms of level number, I do simulation on MATLAB to find the best way to implement hardware design. The tradeoff when choosing level number is between classification accuracy, data-rate, and also area hardware.

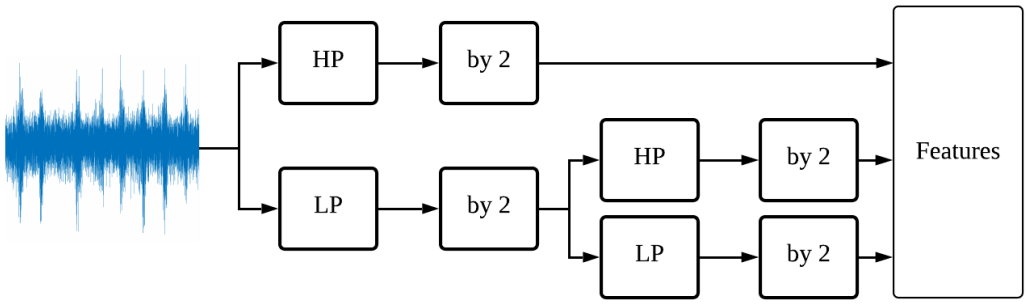
The following table is shown below. Since the area overhead is the most important factor, and accuracy in each experiment are all similar, I choose 2-level DWT to do further hardware implementation.

	Original signal	1-level	2-level	3-level	4-level
Accuracy	100 %	99.952%	99.860%	99.729%	99.792%
Data Rate	100 %	50%	25%	12.5%	6.25%
Area Overhead	N/A	1x	2x	3x	4x

The figure shown on the righ-hand side is comparison between raw signal and using different number of level to reconstruct spike data.

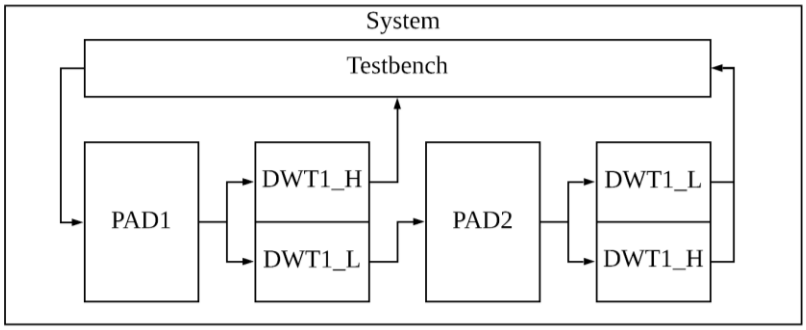


2. Common schematic to implement multi-level DWT:



Filter bank schematic is a common way to implement multi-level DWT. In this project, I choose to implement 2-level DWT, so the flow chart is shown above. In each layer, the signal will pass high-pass filter, low-pass filter, and also be down sampled by two. Then, the signal which pass low-pass filter (approximated information) will be transmitted into the second level.

3. Architecture of the whole system in high-level synthesis (HLS):



The system has three main parts, including non-synthesizable module Testbench, and synthesizable modules (PAD1, DWT1, PAD2, DWT2). In PAD1 and PAD2 module, since symlet-4 wavelet is near symmetric, the signals need to be padded. The input of the whole system is 100 data points, and symlet-4 wavelet has 8 data points, so PAD1 and PAD2 module needs to pad 7 elements on the boundary. The schematic diagram is shown below.

Pad 7 elements	100 input elements	Pad 7 elements

In DWT1 and DWT2, I do 1-D convolution. Take DWT1 for example, since each signal need to be down sampled by 2, I replace convolution done by each element with convolution done by each 2 elements as shown in the following schematic diagram.

Signal		1	2	3	4	5	6	7	8	9	10	11	…	113	114
Filter	cycle = 0		8	7	6	5	4	3	2	1					
Filter	cycle = 1				8	7	6	5	4	3	2	1			

The logic used to implement 1-D convolution is mainly multiplication, and my spike signals are all floating number. As a result, I use 36-bit fixed-point multiplication for the whole system to make sure that data resolution is enough for signal reconstruction and accuracy. The bit representation is show below, and it can represent from +262143 to -262143.

Sign (1 bit)	Integer part (18 bit)	Fractional part (7 bit)

Besides, since both high-pass and low-pass filter need to be used for each data points, I parallelize both convolution in my kernel design.

Testbench will feed data point one by one to the kernel, and get three features in each step, including detailed feature from DWT1, detailed feature from DWT2, and also approximated feature from DWT2. Additionally, since the kernel can only do 100 data points process, a shell script is also written for processing more data points (100 * 100 data points is provided).

In HLS, I put timing constraints and loop unroll constraints to optimize the whole design.

Results

1. HLS result using 100 data points:

	Behavior	BASIC	DPA
PAD1	N/A	1516.6	1383.9
DWT1	N/A	5465.1	11496.5
PAD2	N/A	1489.0	1332.8
DWT2	N/A	5340.8	11397.7
Total run time (ns)	5610	58210	53590

DPA optimize the latency of the kernel, and it is important when processing large amount of data; otherwise, the area overhead is less important.

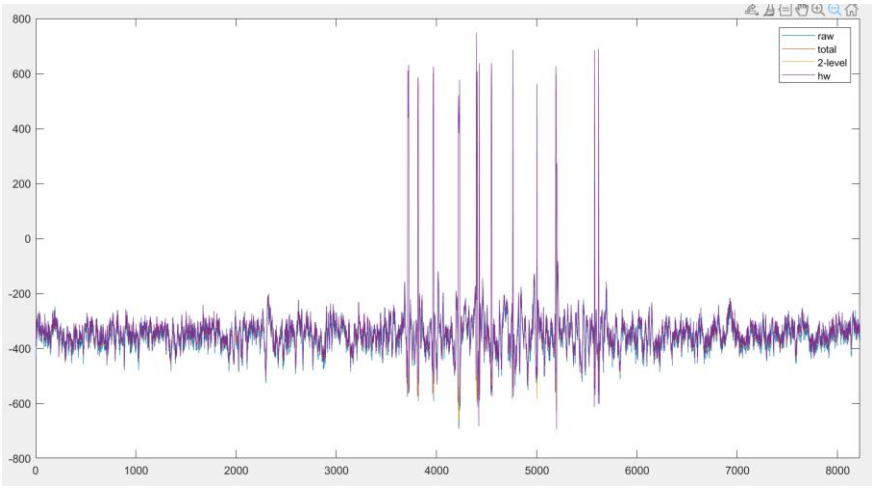
2. Accuracy comparison between hardware and software (MATLAB) using 100 * 100 data points:

	Raw Data	Software	Hardware
Accuracy	100 %	99.52%	99.53%

The accuracy is calculated by setting a suitable threshold to the signal since the peak location is the most important information in spike data processing. It is interesting that the accuracy of hardware is slightly better than that of software, and it may because the resolution in MATLAB is smaller than that in the hardware.

Discussion

In the final version, I implement the whole system using 36-bit fixed points. The main reason is that in the previous version with 32-bit fixed point, I found that there will be overflow in some data points and lead to bad accuracy. The results are shown below. Those abnormal peak is generated by the overflow.



Conclusion

In this midterm project, I use HLS to implement the software algorithm into hardware design without changing the testbench into hardware description language (HDL). Also, the rough latency and area are given by HLS which can give me more insight to figure out how to optimize my kernel design. Though implementing a new algorithm is hard, HLS and SystemC really helps me to make my algorithm close to a real hardware design, and reduce a large amount of time to write a HDL, like SystemVerilog or Verilog.