A Report for Yanran Yang's Summer Research about Coding of information for reliable and low power communications

Yanran Yang

(Because the result of my research hasn't been published yet, this report will not elaborate too much on the project details and it will also not give out the concrete implementation of my design.)

Abstract

This report records my summer research project with Professor Zhengya Zhang about a FPGA implementation of a polar decoder. The report will show what polar code is, what I did and what the result of this research got.

I. Introduction of the project

Topic of my research project is Coding of information for reliable and low power communications. It is about a new type of decoding algorithms with high efficiency and low power consumption. Overall goal of this project is to make communication systems more energy-efficient.

My research involves channel polarization, which is proposed to construct capacity-achieving codes in binary-input discrete memoryless channel, and FPGA implementation architecture for polar codes.

I first studied and investigated novel decoding polar codes algorithms and high performance hardware decoder architectures. Then I mapped the existing C code of a polar decoder onto hardware implementation on FPGA board. I designed most part of the hardware and used different method to build the architecture.

II. Brief introduction of polar decoder

Polar coding is a new type of error-correction codes, which is proved to approach the best performance in energy and efficiency in communication so far. The theoretical part of polar codes involves a lot of mathematical proof and computation. (A. Pamuk, 2011)

I will mainly focus on the architecture of polar decoder. Here is a simple 4-node graph representation for polar codes.

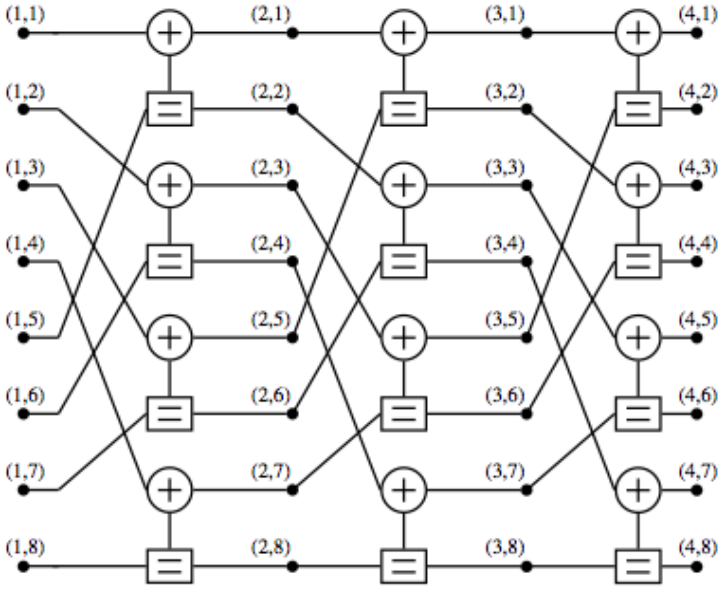
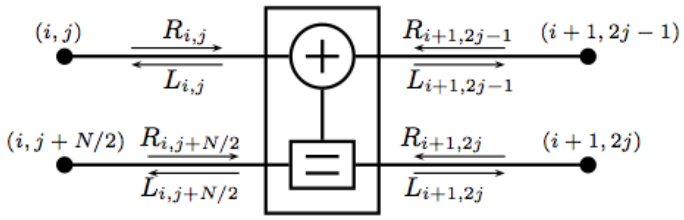


Fig. 1. Uniform factor graph representation for $\mathbf{G}_8$. [1]

We can see on the graph that there are certain patterns in connections between different data points, which is the special point of polar codes.



[2]

On the left is a detailed illustration of what a node in polar decoder means. A node consists of two kinds of operations. It will take in input from different positions and different stages of propagation. The node will compute data and create new output to next stage.

$$
\begin{aligned}
L_{i,j} &= g(L_{i+1,2j-1}, L_{i+1,2j} + R_{i,j+N/2}) \\
L_{i,j+N/2} &= g(R_{i,j}, L_{i+1,2j-1}) + L_{i+1,2j} \\
R_{i+1,2j-1} &= g(R_{i,j}, L_{i+1,2j} + R_{i,j+N/2}) \\
R_{i+1,2j} &= g(R_{i,j}, L_{i+1,2j-1}) + R_{i,j+N/2}
\end{aligned}
$$

[3] $g(x,y) = \ln((1+xy)/(x+y))$

How data is dealt is shown on the left. These four functions are mainly constructed by two main function, one is g function, which is shown on the right and the other one is

[1] A. Pamuk, *An FPGA Implementation Architecture for Decoding of Polar Codes*, 2011
[2] A. Pamuk, *An FPGA Implementation Architecture for Decoding of Polar Codes*, 2011
[3] A. Pamuk, *An FPGA Implementation Architecture for Decoding of Polar Codes*, 2011

addition. The addition function will just add the total of two inputs. However, g function will do more complicated computation. We didn't realize directly using this g function due to many reasons. Instead, we used a simpler one, but also achieved the same result.

Now we have some knowledge about how polar decoder functions. Then we can see how it can be constructed. (A. Pamuk, 2011)

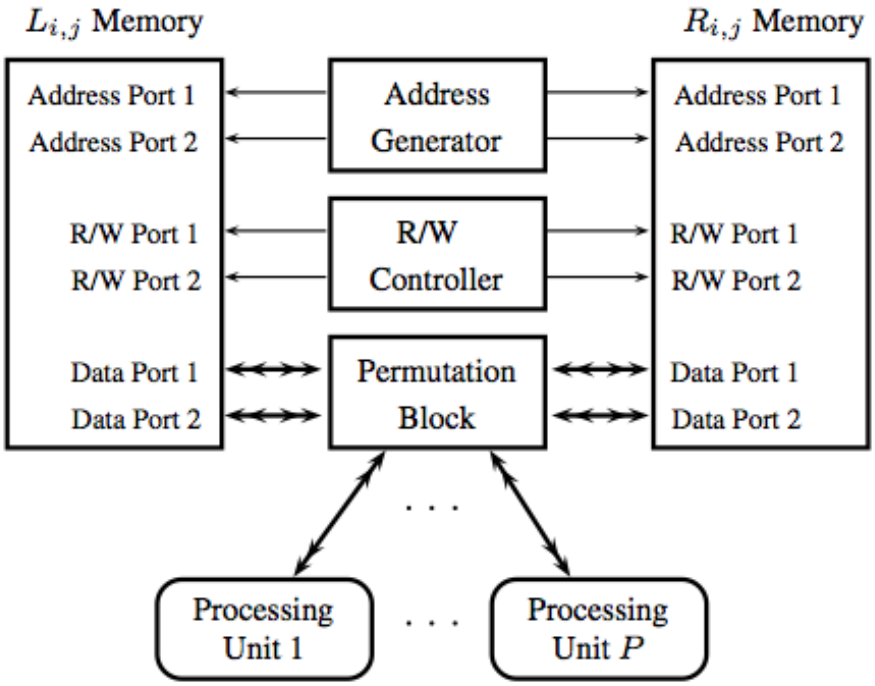


Fig. 3. A top level block diagram of the proposed BP decoder architecture. [4]

Two basic parts on the graph is permutation block, which contains processing unit and two large memories. Two columns of memory are used in different propagations, left propagation and right propagation. Left propagation will propagate from left node to right node and right propagation does the opposite. These two kinds of propagation will affect each other and help get the correct results. Two memories are used to store those intermediate value produced during iterations. Propagations in two directions won't affect the hardware architecture inside nodes because they have the same functions but different inputs (A. Pamuk, 2011).

III. Design thoughts and considerations

a) Tools

[4] A. Pamuk, *An FPGA Implementation Architecture for Decoding of Polar Codes*, 2011

I used Matlab and Simulink to construct the hardware model. Simulink allows users to directly design on hardware level by its Graphic User Interface. By using Simulink, I can simulate the architecture I designed in a faster speed and in an easier way. Also, it's more convenient to modify and redesign different hardware modules according to simulation outputs. After my design is verified, I can put it on a real FPGA board and then retest it.

I used Xilinx blocks to construct my design. I am quite familiar with Xilinx blocks because I studied it before. Also they coordinate with Simulink and Matlab very well.

b) General design process

Because I was new to the software, it took some time for me to get used to the design tool. Simulink is a kind of software used to simulate how hardware behaves. However, it sometimes has some strange restrictions, which cannot be avoided due to its software characteristics. For example, there is a module called "Concat", which will take in parallel data and send them out in series. In Simulink, "Concat" can only have unsigned inputs, which actually doesn't make much difference in hardware because hardware won't affect the way data are combined. I had to deal with this kind of difference between Simulink and a real hardware.

The goal of my hardware design is to build a 16-bit and 8-node polar decoder. I first started with a single node. I built it and then tested it. Because nodes are one of the most important parts in the whole design, I spent much time on it. Also the computation inside the nodes decided in general how data are handled. After single node was verified to work well, I made it into a subsystem and started to construct the general data path. I used scripts to make auto-generations and auto-routing, which had a clear advantage over other methods. After the whole frame was done, I wrote the Verilog code for controller and used a black box module to make it into hardware blocks.

Finally I combined all units together and connected them into a completed system and tested it. It was sure that there were bugs in the design, so I revised it and tested it again. After several rounds of this process, I got a successful hardware implementation of polar decoder.

IV. Introduction of the design

The whole system consists of several main parts. It is done in a hierarchy way so that it would be easier to debug and test. For example, a single node is a subsystem itself. There are also some other subsystems.

a) Nodes

The first part of the system is nodes. Nodes are of the amount of half of the data width. Each Node is identical, so I only need to build one node. After it was verified and tested, I can continue to copy and paste it into the number I want. Nodes decide in which way data is handled. All data in my design is in a format of 2's complement. It is easier in computation.

b) Controller

The second part is controller. Because we used message-passing method to propagate data and do iterations, we need a controller to control the input and memories. The system is pipelined, so we also need to control different parts to work in different time cycles. Controller consists several stages and nine output signals.

c) Control units

The third part is all other control units. Because nodes need to get different input and the system is pipelined, we need control units to decide what to be fed into each part in each cycle. Multiplexers and registers are efficient in this way.

d) Memory

The fourth part is memory. Memories are needed to store the intermediate value during the computation. In every stage of the computation, we need to use some intermediate value from last iteration as the input of this iteration according to the theory of polar codes. Thus we need to store all those values in memories for late use even though they are not needed in the final results. Memories are usually large compared to other hardware units.

e) Hard decision

Last part is the hard decision part. Hard decision unit is the essential unit of deciding whether the process has ended or not. It will get the bit error rate after each iteration is done. When the result reaches zero, it means that our whole process is done. It can be used to control how many iterations we need in the further model.

V. Design methods

There are basically three ways for me to design the whole decoder. I implemented different ways on different parts according to the characteristics and functions they have.

a) Verilog programming

The controller of the whole system was implemented in Verilog method. I used a finite state machine to control the system, so it would be complicated in hardware structure. Also it would be hard to debug and test if I hand construct it. Because our system is pipelined and bidirectional, there are more than twenty stages in the controller for the 1024-bit version. This greatly increased the complexity of the hardware architecture.

I programmed using Verilog to describe the function of the controller on behavior level and then used black box module in Simulink to automatically generate corresponding hardware design in the software. Simulink would give out the optimal solution to my code.

b) Manually design on hardware basis

For nodes, which are actually the core parts of the whole system, I designed them on hardware level directly. Nodes are used to process information and give out the information into next steps. There are two main parts in the nodes, which would be g function and add function. Each node contains two g functions and two add functions. There would be four inputs for the entire node and two outputs propagated into next stage.

Both functions will take two inputs. G function will get the smaller magnitude of two inputs and combines it with sign, which creates the new output. Decision of the output's sign also depends on two inputs. It's basically the product of two signs from two inputs.

$$g(x,y) \approx \operatorname{sign}(x)\operatorname{sign}(y)\min(|x|,|y|)$$ [5]

This is the g function we used. It is a simplified version from theoretic one. Because the original one is very complex in mathematics and also is very hard to implement in hardware. It needs sine function and product function, which will take much space on hardware scale. A single node needs to be repeated for 512 times and each single node contains two g functions. If g function took too much space on hardware, it won't be able to fit the whole system onto a FPGA board concerning other space-consuming part such as memories. So we want the hardware design of node to be as simple as possible.

(c) Scripts

Script is the third way I took to construct the frame of hardware design. I used script to generate blocks, assign different modules to their positions, and connect different units inside the system. The version I made contains only 8 nodes. However, eventually we want to make it into a 512-node version. In that case, routing would become a big issue if it totally depends on manually connection.

[5] A. Pamuk, *An FPGA Implementation Architecture for Decoding of Polar Codes*, 2011

I used scripts to do auto-routing to solve this problem. The script can run under Matlab. I wrote code to auto-generate nodes, multiplexers and some other connecting units from the library I constructed. I also wrote code to auto-connect between units.

The reason of doing this is not only that we aim at a design in a larger scale, but also that the design itself is very complex. Manually connection can easily go wrong.

The first problem occurred is that although we have several stages in the whole process, we won't be able to put the hardware units for all ten stages because of the limitation of space, power and money. What we do is to reuse the same hardware for each stage. Then there occurs feedback loops. Because we need the output from last stage to act as the input for next stage, we have to feedback for node.

Another difficulty is that I made the node bidirectional, so that we can save more space on hardware. When the node is bidirectional, then it will take in different input source during left propagation and right propagation. Then I need to consider the whole frame more carefully in order to ensure the right orders of data. Data needs to be shuffled in different ways in different propagation.

Also polar codes have great complexity itself in math and routing. We get a better performance through the movement of data. We need to shuffle the entire output to feed them into the memory and nodes. This kind of shuffle also took a lot of efforts. By scripting, we can assign precisely which output goes to which input. And because most of these routings are patterned, we can save much time by auto-generation.

One more advantage of scripting is that I can parameterize the data width as n in my script. That n can be changed late into any number, which will be an easy way to produce models of different scales.

VI. Result of the project

After testing and verifying, my hardware design gave out an outstanding result. We used the result from C code and compared it with what we got from the hardware architecture. Results from two sides matched perfectly. Also because of the use of scripts, we are able to expand the scale of the design to a larger data width in the near future.

The final system is a three-stage pipelined system. It is bidirectional, which means it can propagate from left to right and from right to left, which will save a lot of space. Also the whole system is designed in the simplest way and took the least space as

possible.

I learned a lot from this project in theoretical knowledge as well as in hardware design. I know more about what polar code is and how it functions. Also I learned to use Simulink and get more knowledge about how hardware design is like in real projects. I practiced more in Verilog language and debugging skills, which will definitely help me greatly in my further study.

Overall I get a better knowledge about communications and decoding algorithms. I will continue my research in testing different patterns of propagation in the coming winter.

References

E. Hof, I. Sason and S. Shamai, *Polar coding for reliable communications over parallel channels*, Proceedings of the 2010 IEEE Information Theory Workshop, Dublin, Ireland, Aug. 30 - Sept. 3, 2010.

C. Leroux, I. Tal, A. Vardy, W. J. Gross, *Hardware architectures for successive cancellation decoding of polar codes*, The 36th International Conference on Acoustics, Speech and Signal Processing (ICASSP, 2011), Prague, May 22-27, 2011.

A. Pamuk, *An FPGA Implementation Architecture for Decoding of Polar Codes*, 2011 8th International Symposium on Wireless Communication Systems, Aachen, 2011