Design parameterizable filter using High Level Synthesis

TROELS MÆHL FOLKE & WENQIAN DING

MASTER´S THESIS

DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY

FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

Design parameterizable filter using High Level Synthesis

Troels Mæhl Folke
`tr2721fo-s@student.lu.se`
Wenqian Ding
`we6474di-s@student.lu.se`

Department of Electrical and Information Technology
Lund University

Supervisor: Liang Liu

Examiner: Erik Larsson

September 7, 2021

Abstract

As the ASIC designs continue to grow in complexity, traditional RTL level of abstraction is becoming a productivity bottleneck. The RTL design process requires extensive time and effort for verification of algorithmic correctness as well as correct timing and interface behavior. Furthermore, a non-trivial change to the algorithm often results in a complete rewrite of the RTL implementation. High Level Synthesis (HLS) solves this issue, by allowing the designer to focus on the functionality while the tool takes care of implementation details such as finite state machines and timing. HLS vendors promise considerable savings in development time.

In this master's project, we have implemented a template of parameterizable polyphase filters in C++. The design was then synthesized using Mentor Catapult HLS. The number of polyphases, bitwidth, number of taps, and coefficient binding were made parameterizable.

Simulation and verification results show the functional correctness of the design. Also, a thorough comparison of the RTL reference design and an equivalent HLS design, using the same parameter set, has been carried out. Results reveal that the HLS design achieves higher performance in both area and latency. Taking the symmetric FIR filter as an example, the latency is reduced by up to 5 clock cycles, and the area decreased 21% compared to the reference design. The main reason for reduction in latency and area is the ability of HLS tool to reduce and balance the pipeline stages more efficiently compared to the manual RTL design.

Popular Science Summary

In today's digital reality, computer chips are everywhere. From personal devices like our computers and smartphones, to cars, airplanes, refrigerators and even microwave ovens – all have at least one computer chip. Of these products, the smartphone is probably among the most advanced ones – at least from the perspective of the functionality of the chips inside it. A smartphone is required to have a wealth of diverse functionality, such as a processing unit, a graphics unit, a cell modem, WiFi, etc, all of which today are integrated on a single chip – a so-called System on a Chip (SoC). The same is true for the equipment at the other end of the radio-link, the base station, which is manufactured by the Swedish company Ericsson, among others.

The defining feature of the business environment in the contemporary technology sector, including the sub-sector of chip development, is the increasingly rapid pace of product development, paired with the customer expectation of an increasingly feature-rich and complex product. At the same time, the current chip development methodology and the related software tools have both not developed noticably since the late 1990s. A methodology, that requires the chip developer to describe his digital circuit at the so-called "register transfer level" (RTL) of abstraction. This involves specifying in detail what happens in every single clock cycle of the chip. As a result, a bottleneck has appeared in the implementation- and verification parts of the development cycle, due to the sheer complexity and size of digital systems today.

This thesis has investigated a relatively new method of development, namely High Level Synthesis (HLS). Here, an algorithm written in the regular computer programming languages of C or C++ can be synthesized directly to hardware, by the use of a special software tool. We have implemented a template library of FIR filters using this technology, and compared it with an existing RTL-based design provided by Ericsson. The results show improvements across the board, both in terms of area, latency and power consumption, while also adding design flexibility and shortening development time.

Acknowledgements

We would like to express our gratitude to our supervisors Liang Liu from LTH, and Naeem Abbas and Asad Jafri from Ericsson Lund who gave us the feedback and technical support throughout the thesis project. Also, we want acknowledge our manager Stefano Olivotto for the support and this opportunity to conduct our thesis project in Ericsson. We would also like to thank Eril Larsson to be our examiner.

Besides, we would like to thank all the great teachers who taught us in EIT especially in the course DSP design, you gave us the basic knowledge of filter design and also design flow in Catapult.

Finally, we would like to acknowledge our family and friends for their support and company.

Troels Mæhl Folke
Wenqian Ding

List of Abbreviations

ASIC	Application-Specific Integrated Circuit
DSP	Digital Signal Processing/Processor
DFG	Data FLow Graphics
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Arrays
HDL	Hardware Description Language
HLS	High-Level Synthesis
II	Initial Interval
IIR	Infinite Impulse Response
LTI	Linear Time-Invariant
MUX	Multiplexer
RTL	Register Transfer Language/Level
VHDL(VHSIC-HDL)	Very High-Speed Integrated Circuit HDL

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

The connectivity company Ericsson develops their own customized radio integrated circuits (ICs) for their Fifth Generation (5G) base stations. The digital predistortion (DPD) section of these ICs makes heavy use of various digital filters, all of which today are manually implemented and verified at the Register-Transfer (RTL) level of abstraction.

This approach has several drawbacks: First, the translation from a high-level algorithmic description in Matlab or C/C++ to a specific hardware micro-architecture in RTL is a manual endeavor, making it tedious and time-consuming to explore the architectural design-space and evaluate various hardware tradeoffs. Furthermore, the human nature of the translation process is highly susceptible to the introduction of bugs, requiring extensive verification efforts (not only for algorithmic correctness, but also for correct timing and interface behavior, etc.). Finally, if the design requirements of the overall project changes, the RTL implementation is often so customized that a total reimplementation is necessary – wasting precious time as a result.

High-Level Synthesis (HLS) technology has set out to change that. While mainly being a subject of academic interest in the 1990s, the technology has steadily progressed and is now mature enough for many digital design tasks – and is offered commercially by several EDA vendors[1]. It allows the designer to describe a circuit on an algorithmic level in C++ (with some restrictions), without concerning himself with implementation details such as timing, interface, state machines,datapath, pipelining, parallelism or the number of functional units.These details are all taken care of by the HLS tool, working under a set of constraints given by the designer.

1.2 Thesis Structure

The structure of this thesis is organized as follow:

- Chapter 1 is the introduction and motivation of this thesis.

- In Chapter 2, the theory of basic filters that implemented in this thesis is given, including lowpass fir filter and half-band fir filter.
- Chapter 3 introduces the theory of the polyphase filter, it consists of the multirate system knowledge and also the polyphase theoretical implementation.
- The Catapult design flow and the parameterizable polyphase library implementation details are presented in Chapter 4.
- In Chapter 5, the results of comparison between the reference RTL design and the HLS design. The analysis of the reason that causes the difference of these two design is illustrated.
- Chapter 6 gives the conclusion of this thesis and further work.

Chapter 2

FIR filter theory

FIR filter and IIR filter are two types of digital filters.[2] Generally, IIR filters have wide application since they have low computation complexity and latency and when the linear phase response is not necessary. And In this case, they are chosen to be employed in biomedical sensor signal processing, audio equalisation and telecommunication applications.

Despite the advantages of IIR filters, FIR filters become the first option when the linear phase response comes into a vital requirement. In contrast to an IIR filter, the FIR filter has no feedback loop, which brings a distinct advantage to the FIR filter as stability. With these two strengths, the downside of the FIR filter, like more computation complexity, can be disregarded. In this way, FIR filters are preferred over the IIR counterparts and applied in digital signal processing applications.[3]

The comparison between the FIR filter and IIR filter can be seen in the table below to make it clear.

2.1 FIR filters

Generally, the output of a discrete linear time-invariant(LTI) system in the time domain can be represented as:

$$y(n) = \sum_{i=0}^{m} b_i x(n-i) - \sum_{j=1}^{l} a_j y(n-j) \tag{2.1}$$

The following system is named Infinite Impulse Response(IIR). In this system, the output depends on input and previous result, which is so-called feedback. Such a structure causes instability. When the feedback part of this system is removed by setting l to be zero, the output then only depends on the input data. While the stability can be fulfilled, typically, the FIR filter requires a higher order for the same performance as IIR filters. At the same time, a higher delay is caused by a higher order. The FIR filter structure can be described below. The output of an FIR filter is the accumulation of the delayed input sample multiplied by the filter

	FIR filter	**IIR filter**
Structure	No feedback loop	With feedback loop
Stability	High stability because the output is not dependent on the previous input	Less numerically stable owing to the feedback loop
Computation complexity	More MACs (Multipliers and adders) are required for achieving the exact specifications	Less implementation cost (coefficients) to satisfy the cut-off frequency and stopband attenuation
Phase response	Simple to design a linear phase response filter, phase distortion can be prevented	Generally not linear, especially near the cut-off frequencies
Latency	Higher group latency due to more taps, not very suitable for high throughput applications	Low latency and can be applied in real-time process and high-speed applications

Figure 2.1: Comparison between FIR and IIR filters

coefficient.

$$y(n) = \sum_{i=0}^{m} b_i x(n-i) \equiv \sum_{i=0}^{m} h_i x(n-i) \tag{2.2}$$

The system has m order and non-recursive. The impulse response $h = h_0, h_1, h_2, ..., h_m$ is limited to $m + 1$ taps. There are many structures of FIR filter, the basic direct form architecture, and other types like direct form transposed; it can also be symmetrical or asymmetrical.

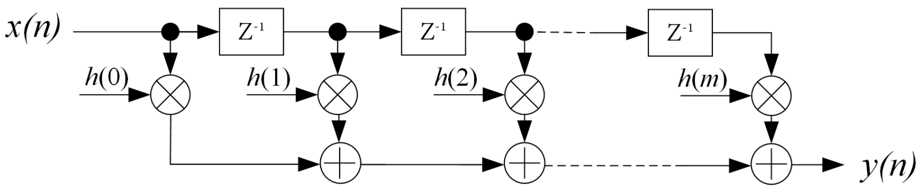


Figure 2.2: The structure of direct form FIR filter

2.1.1 Linear Phase FIR filter

As described before, the distinct advantage of the FIR filter is that the phase response is linear, which refers to the condition that the phase response of the filter is a linear function of the frequency. Correspondingly, the delay through the filter is the same at all frequencies. The delay of a linear phase FIR filter is:

$(N-1)/(2f_s)$, where N is the number of taps and f_s is the sampling frequency.

Therefore, the filter will not cause phase distortion or delay distortion. Lacking phase and delay distortion results in a critical advantage over IIR filters and analog filters, especially in specific systems such as digital data modems.

Apart from the characteristics of linear phase response and constant delay, the linear phase filters will preserve the waveshapes of the input signal, which causes the application to the communications area—for example, the coherent signal processing and demodulation. Preserving or recovering the waveshape without distortion in such applications is essential when the waveshape takes the thresholding decisions.

Designing a linear phase FIR filter can be straightforward since the filter is linear phase if and only if the impulse response is symmetrical. At the same time, the number of taps can be odd or even. This characteristic also causes the possibilities for the design optimisation to accumulate the data with the same coefficients first. In this way, half of the multiplication can be saved, which reduces area since a multiplier has much less area than an adder.

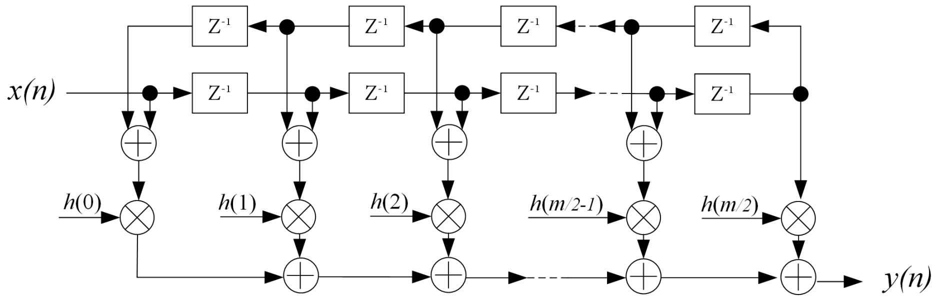


Figure 2.3: Block diagram of linear phase fir filter

The block diagram in figure 2.3 is the structure of a linear phase FIR filter[4]. The basic idea of this structure is very similar with the direct-form FIR filter, but the difference is that the taps with same coefficients are added together before the multiplication. The number of coefficients is reduced to half. When the number of taps is odd, the central tap will be multiplied by the last coefficient.

2.2 Low-pass FIR filter

A lowpass filter functions as passing low-frequency signals and attenuates signals with frequencies higher than the cutoff frequency. The cutoff frequency of an ideal low-pass filter is $w_c = (w_p + w_s)/2$, Where w_p is the passband and w_s is the stopband.

For an ideal low-pass FIR filter, the frequency response is rectangular. Theoretically, it can be realised by convolution with the impulse response in the time domain, which is a *sinc* function[5]. However, the ideal filter is impossible to implement in reality since the filter would need infinite order and delay. Real-life filters usually approximate ideal ones by truncating and windowing the infinite impulse response.

$$w[n] = \begin{cases} 0.54 - 0.46cos(\frac{2n\pi}{M}) & 0 \leq n \leq M \\ 0 & otherwise \end{cases} \tag{2.3}$$

Take the Hamming window as an example. The equation is described as 2.3, where M is the order of the filter. And to have a linear phase response for the filter, a time shift equal to $M/2$ should be applied by multiplying the result by w[n].

$$h[n] = [0.54 - 0.46cos(\frac{2n\pi}{M})][\frac{w_c}{\pi}sinc(\frac{w_c n}{\pi})] \quad 0 \leq n \leq M \tag{2.4}$$

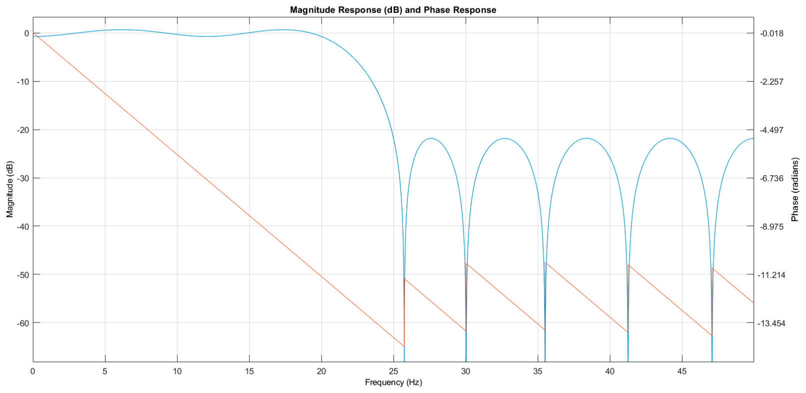


Figure 2.4: Frequency response of low pass filter

The impulse response contains the coefficients with a proper delay of the filter. When the input signal $x[n]$ equals $\sigma[n]$, the output will be the impulse response, which means $y[n]$ equals $h[n]$. It is evident that the impulse response fits the *sinc* function but windowed.

2.3 Halfband Filter

Halfband filters are one specific lowpass filter with the cut-off frequency of one-quarter of sampling frequency f_s and odd symmetry about $f_s/4$.[6] Halfband filters are widely applied in multirate systems, especially when interpolating or decimating by a factor of 2. Normally, the halfband filter is implemented in the form of polyphase structure to avoid unnecessary computation, since almost half of the filter coefficients are zero. When the number of taps is odd, the property of the halfband filter causes every other coefficient to be zero except the central one.

There are two main characteristics of halfband filters which causes it becomes the specific case of lowpass filter.

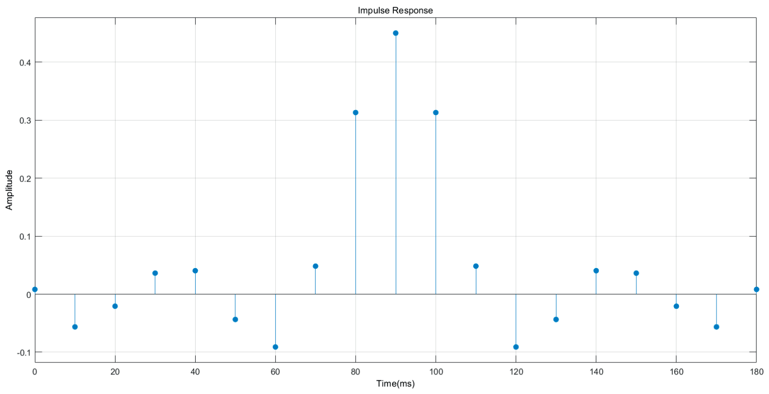


Figure 2.5: Impulse response of low pass filter

- The passband and stopband ripples must be the same
- The passband-edge and stopband-edge frequencies have equivalent distance from the halfband frequency, and they are both $f_s/4$. When normalized, the frequencies become $\frac{\pi}{2} rad/sample$.

The properties of halfband filter bring a restriction on the number of filter taps. When designing a linear-phase N-tap halfband filter, with alternating zero-coefficient to be zero, N+1 must be an integer multiple of four. If this condition is not met, then the first and last coefficient will be discarded.

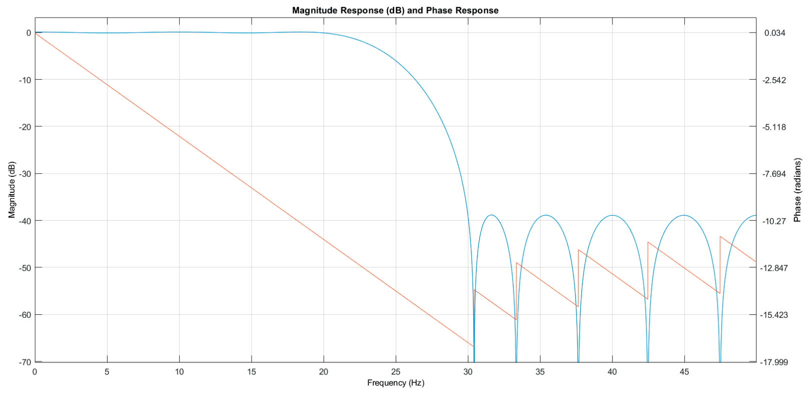


Figure 2.6: Frequency response of halfband filter

Figure 2.6 is the frequency response of the halfband filter with the f_s equals to 100KHz, the blue curve is the magnitude response and the red curve is the phase response respectively.
This characteristic is illustrated in the impulse response as figure 2.7 clearly. Since every other coefficient in the halfband filter is zero, the structure of the halfband

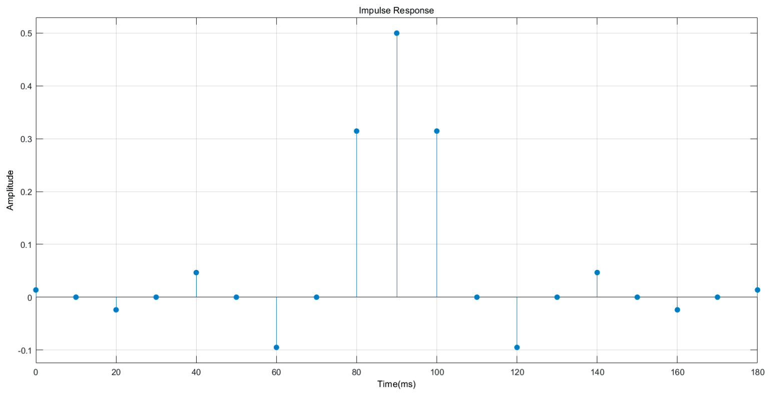


Figure 2.7: Impulse response of halfband filter

filter can thus be reduced to approximately half the area compared to the low-pass filter.

Chapter 3

Polyphase Filters Theory

In this chapter, the theoretical knowledge of polyphase filters is introduced. Firstly, the basic information of multirate systems is presented, including the application field and the concept of decimation and interpolation[7], which are the two significant processes in the multirate system. Then the polyphase is presented afterward, while polyphase is used to solve the problem of unnecessary computations in decimation and interpolation. A detailed description of how the filtering is executed and wherewith the polyphase filter be designed will be illustrated in this section.

3.1 Multirate Systems

The past decades have seen the rapid development of multirate digital filters and filter banks, which have many applications in the field of communication, spectrum analysis[8], radio systems, image and video processing. Compared to the single-rate system, the multirate system shows an obvious advantage in processing the signals efficiently since the sampling spaces can be changed internally. Although aliasing can happen due to this portrait, it can be discarded with careful design.

Multirate system research involves sample rate conversion, the concept of decimation and interpolation. The sample rate conversion is unavoidable in real-time signal processing when the two separate hardware units operating in two different sampling rates and must communicate with each other.[9] For certain narrow-band systems, sample rate conversion can reduce computation and complexity.

3.1.1 Decimation

Decimation is a non-time-invariant process that contains two steps, first low pass filtering and following as downsampling.

The concept of downsampling can be explained as retaining every M_{th} sample in the input signal and discard the rest data. Express the original sample frequency as $f_{s,in}$, then the current sample rate is:

$$f_{s,out} = f_{s,in}/M \tag{3.1}$$

The downsampled output is:

$$x'(n) = x(Mn), M = 0, 1, 2, ... \tag{3.2}$$

In the time-domain, the output can be written as:

$$y\,[n] = \sum_k h[k]x[(n-k)M] \tag{3.3}$$

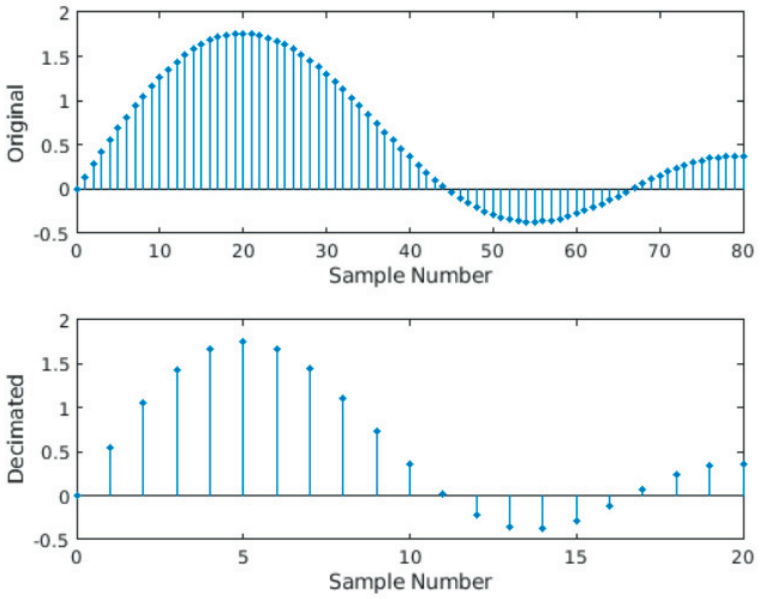


Figure 3.1: Demonstration of decimation by factor of 4

Based on Nyquist Theorem[10], the new sample rate should be larger than 2 times bandwidth to avoid overlapped spectral replications, also known as aliasing errors. To prevent this error, a low pass filter is required before downsampling. The structure of a typical decimator is :

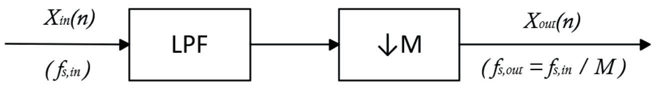


Figure 3.2: Process of decimation

Where $h[k]$ is the impluse response of each tap in the low pass filter, and M is the decimation factor.

The convolution process of the low pass filter can be expressed as:

$$V^z(z) = X^z(z)\,H^z\left(z^M\right) \tag{3.4}$$

After the filtering, the downsampled signal in z-transform is

$$\begin{aligned} Y^z(z) &= V^z(z)_{(\downarrow M)} \\ &= \frac{1}{M}\sum_{m=0}^{M-1} V^z(z^{1/M}W_M^{-m}) \\ &= \frac{1}{M}\sum_{m=0}^{M-1} X^z(z^{1/M}W_M^{-m})H^z((z^{1/M}W_M^{-m})^M) \end{aligned} \tag{3.5}$$

3.1.2 Interpolation

In contrast to decimation, the process of interpolation includes upsampling to increase the sample rate by inserting zero-valued data. Given the original sample rate as $f_{s,in}$, L-1 zero-value data must be inserted between each input sample to increase the sample rate by a factor of L.

$$f_{s,out} = L \cdot f_{s,in}$$

Similar to decimation, a low pass filter should be applied, but after the process of upsampling. It can reduce the distortion caused by the undesired mirroring spectrum images of the signal centred at the multiple of original signal.

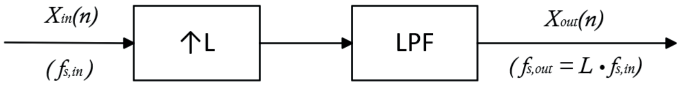


Figure 3.3: Process of interpolation

Given the input signal $x(m)$, the upsampled output is:

$$x'(m) = \begin{cases} x(\frac{m}{L}), & m = 0, L, 2L, ... \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

In the z-domain, the upsampled input signal can be expressed as:

$$V^z(z) = X^z_{(\uparrow L)}(z) \tag{3.7}$$

The undesired mirror images in the spectrum caused by upsampling can be reduced by a low pass filter, the z-transform of output is:

$$\begin{aligned} Y^z(z) &= V^z(z)\,H^z(z^L) \\ &= X^z(z^L)H^z(z^L) \end{aligned} \tag{3.8}$$

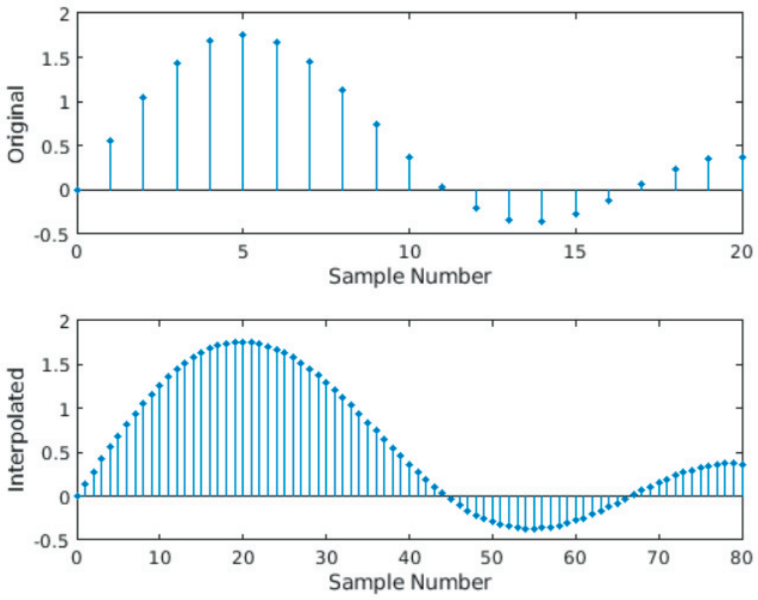


Figure 3.4: Demonstration of interpolation by factor of 4

In frequency domain, the final output is:

$$Y_L(w) = X_L(w)H_{LP}(w) \tag{3.9}$$

Where for a low pass filter, the frequency response is given by:

$$H_{LP}(w) = \begin{cases} 1, & \text{if } \mid w \mid \leq \frac{\pi}{L} \\ 0 & \text{if } \frac{\pi}{L} \leq \mid w \mid \leq \frac{\pi}{L} \end{cases} \tag{3.10}$$

3.2 Polyphase Filters

In the last section, the concept of decimation and interpolation are explained. The polyphase decomposition was originated from the work by Bellanger et al.[11], and is significant in the application of multirate DSP. Consider the process of interpolation, in every L operations,$L-1$ zeros multiplication are executed. Which causes unnecessary computation and memory usage. And for the decimation, M-1 computation results are discarded after the filtering. The digital polyphase filter design reveals this problem by splitting the filter taps into several subfilters, avoiding inefficiency.

The following is an example that describes the process of how the polyphase filter works. Figure 3.5(a) shows the input data samples after interpolation by 4, the squares in Figure 3.5(a) are the original input data and the dots are the interpolated zeros. Figure 3.5(b) is the impose response of a 12-tap FIR filter in reversed order, where triangles are the filter coefficients. Initially, the filter operates by the convolution of the input signal and the coefficient. Apparently, to get the result of x_{new}, 9 out of 12 computations are wasted because of interpolated zeros in each convolution.

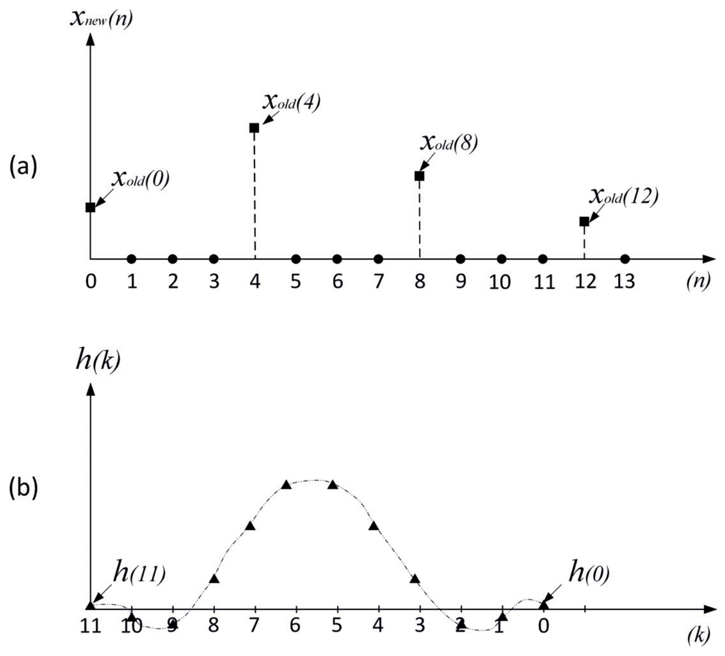


Figure 3.5: Interpolation by factor of 4 for a 12-tap filter. (a) Input data samples; (b) Impulse response of the filter

To make the convolution more efficient, Figure presents how to get the result of $x_{new}(11)$, $x_{new}(12)$, $x_{new}(13)$ and $x_{new}(14)$ to find the regularity of the polyphase filter since the interpolation factor is 4. In Figure 3.6 (a) ,the input data and impulse response are overlapped in 12 samples. As mentioned before, only four computation times are necessary, in which the input data is non-zero, and the corresponding filter coefficients are $h(11)$. $h(7)$ and $h(3)$. The next step is to slide one sample of the impulse response to the right to compute $x_{new}(12)$, and now the valid computation is where the non-valued input data and the coefficients overlapped. Similarly, Figure 3.6 (c) and (d) illustrate how to compute $x_{new}(13)$ and $x_{new}(14)$. To make it more visualized and for better understanding, the equations are presented as follows.

$$\begin{aligned} x_{new}(11) &= h(3)x_{old}(2) + h(7)x_{old}(1) + h(11)x_{old}(0) \\ x_{new}(12) &= h(0)x_{old}(3) + h(4)x_{old}(2) + h(8)x_{old}(1) \\ x_{new}(13) &= h(1)x_{old}(3) + h(5)x_{old}(2) + h(9)x_{old}(1) \\ x_{new}(14) &= h(2)x_{old}(3) + h(6)x_{old}(2) + h(10)x_{old}(1) \end{aligned} \tag{3.11}$$

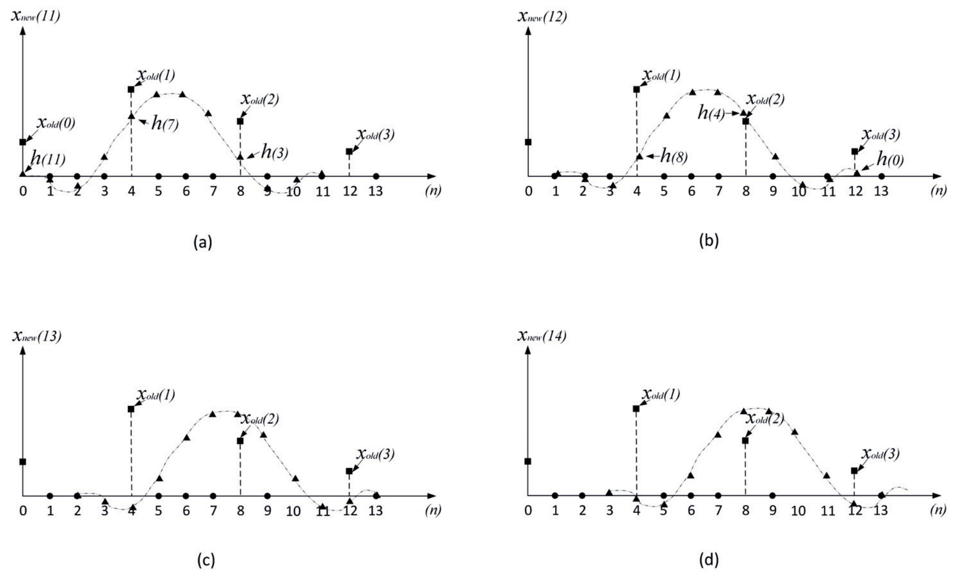


Figure 3.6: The process to calculate the filter output

The below steps not only present how the prototype filter works but also provide the method to implement a polyphase filter. Rather than 12 times multiplications, only three of them are needed in each clock cycle and the whole process to compute the output of the 12-taps filter can be divided into four clock cycles and then add them together to get the final result.

Figure 3.7 shows the block diagram of the polyphase filter with interpolation by 4. The filter bank consists of four subfilters, and each subfilter has a different coefficient set. For each input sample, four output samples are computed and accumulated. The number of subfilters is typically the same as the interpolation factor. Thus the number of taps N can be chosen as multiple of the interpolation factor for convenience. Due to the zeros inserted and the filtering process, there is a gain loss in the polyphase filter, and the gain is the same as interpolation factor L. There are two ways to reimburse for this amplitude loss. One is to increase the filter's coefficients by a factor of L, and the other one is to multiple the output $x_{new}(14)$ by L.

The decimation process in the polyphase filter is similar to the interpolation. In figure 3.8, the block diagram presents the decimation of a 12-tap prototype filter by a factor of 4. The number of subfilters is typically the same as the decimation factor M. In this way, four Four subfilters are applied with different coefficient sets, and the output of each subfilter will be accumulated, but four input samples will be computed in the subfilters for one result. In this way, the decimation process is before the filtering and no discarded computation, which avoiding unnecessary hardware usage.

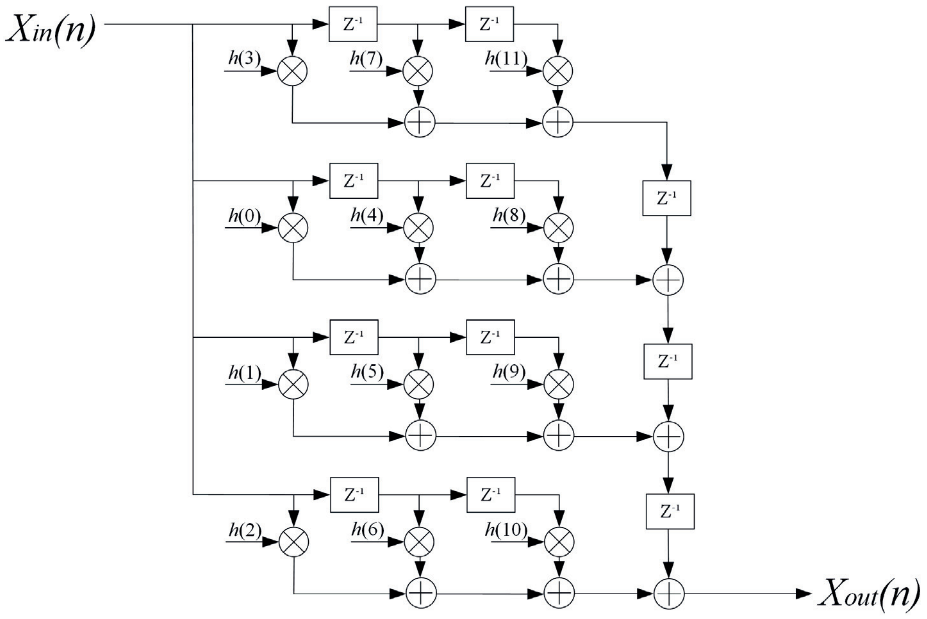


Figure 3.7: Interpolation block diagram of polyphase filter

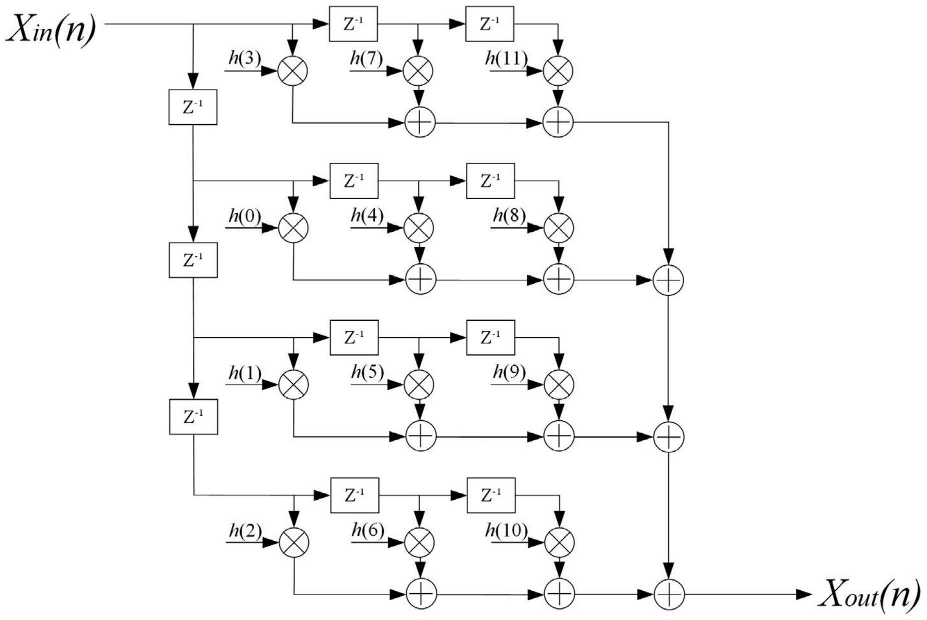


Figure 3.8: Decimation block diagram of polyphase filter

Chapter 4

Implementation

4.1 Catapult design flow

With the increase of complexity in electronics design(ASIC and FPGA), traditional manual RTL implementation can be time-consuming, especially when verifying the hardware-specific constraints like timing and interfaces. Furthermore, when the design specifications are changed, the whole design may need to be modified, which wastes enormous developing time and effort. In comparison, the High-Level Synthesis was introduced and revealed the problems mentioned before, allowing the designer to focus on the implementation of the algorithm itself.

Throughout the thesis, the tool used for the high-level synthesis design is Catapult HLS from Mentor Graphics. In this section, the introduction of Catapult and the design flow is given.

4.1.1 Introduction to Catapult

Catapult is a product of Mentor Graphics for high-level synthesis. Some other similar products that are widely used, are Vivado HLS of Xilinx and Cadence. Catapult allows designers to develop C/C++ code and generate RTL code automatically by the tool. Moreover, for the generated RTL files, the target hardware device can be both FPGA and ASIC. The design flow of traditional RTL is different from Catapult design flow and can be illustrated in the figure 4.1.

As presented in the figure 4.1, both of the designs should include algorithm description using C/C++, and manual written RTL or generated by the tool, then the RTL synthesis and lastly integrated to ASIC or FPGA. The main difference between RTL design flow and Catapult design flow is the Catapult synthesis part. Conventionally, manual RTL design requires manual verification, which the timing and other interfaces behavior could take numerical time. While for the catapult synthesis, the only thing is to set constraints in the tool and be verified at the algorithm level with the testbench written in C/C++.

Unlike standard behavioral C/C++, there are some constraints in the High-level language to make it hardware-friendly. As it is known, the RTL description con-

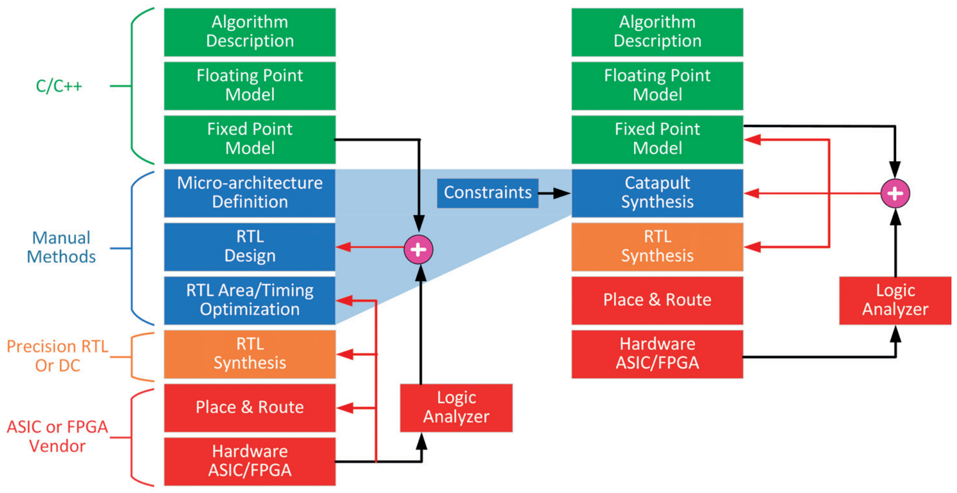


Figure 4.1: Traditional RTL design flow and Catapult design flow[1]

sists entity or module and always be a pipelining process. Since the High-level synthesis should generate corresponding RTL, the structure should be the same. In this way, the HLS should include functions as blocks, and each one can be called or call another function. Like the hierarchy in RTL, there should also be a top-level function in HLS to connect each block(function) and define the interfaces, such as port definitions of the complete design, bit widths and data type. Generally, the design that builds in high-level synthesis is synchronous. The output of the top-level design is usually stored in registers to ensure that the timing is matched when communicating with other designs.

Another point that worth mentioning is that the HLS design always uses bit-accurate data types[12] to make the design more efficient for synthesize. This data type makes it possible to have the same interfaces with RTL and reduces hardware resources since it avoids the default assigned bit width for the standard C++ data types, such as a 128-bit integer. Generally, apart from the conventional data types in C++, Catapult provides the *ac_int* <> and *ac_fixed* <> types for the bit accurate data type for integer and fixed-point numbers , both signed and unsigned. The declaration and configurable parameters are explained as follows.

- $ac\_int < W, S > x$: defines the signal x with W bit as the bit-width and signed when the boolean parameter is set as *true*. The numerical range for x is $[0, 2^W - 1]$ by increments of 1 when x is unsigned, and respectively $[-2^{W-1}, 2^{W-1} - 1]$ by increments of 1 for signed number.

- $ac\_fixed < W, I, S > x$: defines the signal x with W bit where I bit as integer part, and $(W - I)$ as fractional part. Same as *ac_int*, S means signed when the boolean value is *true*. For unsigned signal, the numerical range is $[0, 1 - 2^{-W}2^I]$ by increments of 2^{I-W}, and $[-0.5 * 2^I, 0.5 - 2^{-W}2^I]$ by increments of 2^{I-W} when signed.

- $ac\_fixed < W, I, S, Q, O > x$: defines the signal with quantization and overflow mode. The default mode is to truncate and overflow in $ac\_int < W, S >$ and $ac\_fixed < W, I, S >$. Truncate means throw the bits to the right LSB away, which does not cost extra area but might not be ideal for some applications that high accuracy is required. Instead of throwing the data away, the rounding mode will round up or down depending on the fractional value. To enable this mode, Q is set to ac_RND. There is another mode to detect overflow in the design. Although the overflow should never occur in some applications, especially in the control and image processing area. By enabling the overflow mode, O should be set as ac_ACT to prevent this situation.

4.1.2 Design Flow in Catapult

In this section, the design flow in Catapult will be introduced step by step, which starts from the project creation, significant setups and constraints throughout the design, following dataflow and resource allocation, architecture mapping, and lastly synthesis, and design verification. The design flow can be presented as the flow chart as below.

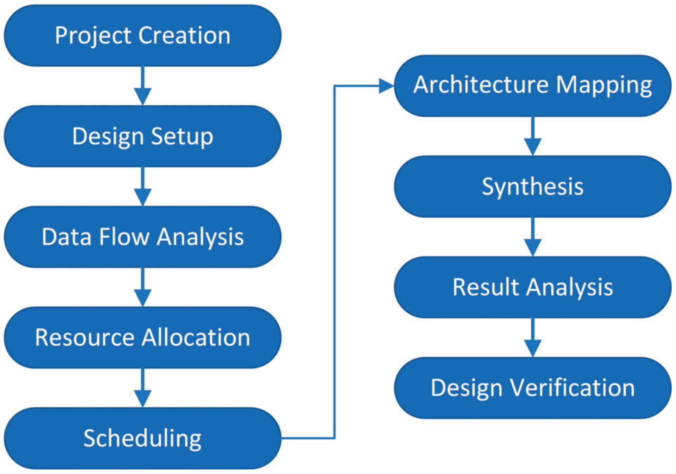


Figure 4.2: Design flow in Catapult

1. In the first step, to create a project, the input files should be specified. Typically, the input files are the C/C++ source files. Catapult will include them by the directory path for the header files, so the manual operation is unnecessary. The testbench is always excluded from design; otherwise, it will be implemented into the RTL and cause extra area. Catapult will automatically structure the design when the top-level function is assigned. There are also two types of functions called *block* and *inline*. Apart from the top-level function, other functions can be set as *block*, which is one hierarchy

level down than the top function, or *inline*, meaning that it is inside of the hierarchy block.

2. The design setup is mainly to set the libraries used for later synthesis. In this step, the synthesis tool, vendor and technology can be specified. There are also memory libraries that can be used. Apart from these libraries, previously synthesized blocks and designs can be included in the present generation. This is because Catapult allows bottom-top and top-bottom design methods. For the bottom-top way, each block will be generated into RTL, the performance such as latency and throughput for each block can be evaluated. While the top-bottom method(no previous design is included), all the functions will be in the same RTL and each as an entity. The performance is for the whole design.

3. After the design setup, the following are data flow analysis and resource allocation. The data flow analysis will generate DFG(data flow graphics) showing the data dependencies and operations order inside the function. Timing and area constraints are applied during resource allocation. In the extracted DFG, each operation has corresponding hardware implementations. Catapult provides a list of the components in the libraries determined during setup with different behavior of area and latency that the designer can assign.

4. In the scheduling step, the tool specifies the operations be executed in every clock cycle. Pipelining can be applied in this step to reduce combinational delays. The more detailed knowledge is explained in the later part Design Constraints in the last of this section.

5. Then the next step is architecture mapping. The hardware architecture depends on the previous setup of resource allocation and the number of loop unrolling. The top-level function is also interpreted as the main loop and can be unrolled. There is a detailed example in the constraints part for the loop unrolling in the typical loop, such as for loop. The number of overlapped loops is called Initiation Interval(II).

6. The following is the synthesis of the design. Catapult will synthesis the design automatically. The signals clock, enable and reset can be added in this step. In this step, the RTL files are generated but can be challenging to read and understand. The synthesis output also includes the schematic, and the data path and critical path can be retrieved. Meanwhile, the schedule of the design is presented in the Gantt chart. The designer can also get the hardware resource information such as area and timing in the summary table or more detailed information in the result reports. In this thesis, another synthesis tool called Design Vision is also used for the result of power consumption.

7. To verify the implementation, Catapult allows two types of testbench: for the C/C++ code and the other is for the RTL files. The testbench to verify the algorithm's correctness is at the behavioral level, written in C/C++. The functionality in the generated RTL can be tested and compared with

the source file in C/C++ in ScVerify. It generates verification infrastructure automatically, and the signals like input/output and internal signals can be seen in the wave and make it easier to solve errors.

Design Constraints

Pipelining

Pipelining is also known as pipeline processing, enabling multiple functional units to execute concurrently. By inserting registers between these combinational components, the instructions are conducted in an orderly process. The registers can store data and perform combinational logics. Pipelining can increase overall instruction throughput. Different from parallelism, pipelining will not use more hardware resources and can achieve speed enhancement.

This project mainly aims to design FIR filters. Thus the analyze of pipelining of FIR filters can be essential. Pipelining reduces the critical path and results in a higher sampling rate. The following is an example to explain how pipelining works.

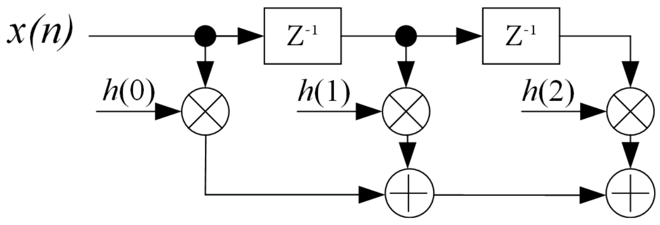


Figure 4.3: FIR filter without pipelining

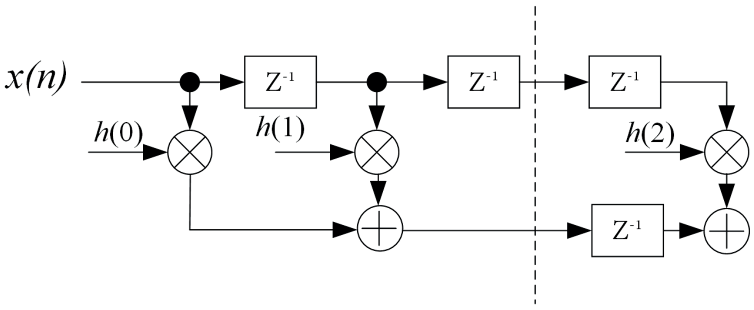


Figure 4.4: Pipelined FIR filter

For a 3-tap FIR filter, suppose the calculation time unit of multiplier and adder are T_m and T_a separately. Initially, the minimum sampling period should be $T_m + 2T_a$ without pipelining with the restriction of the critical path. When the pipelining is introduced, a register is inserted in the critical path, and the sampling period can be reduced to $T_m + T_a$ at the lowest.

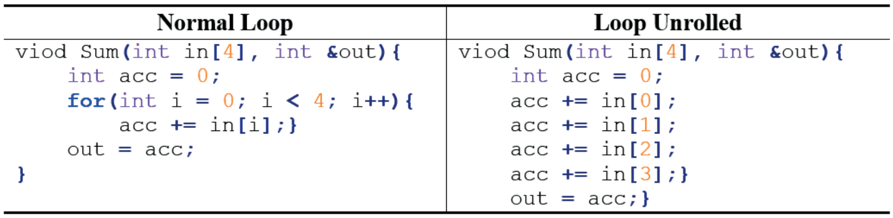
Pipelining can also help reduce power consumption. The power consumption in the original structure can be described as:$P = fCV_{DD}^2$. Reducing the critical path can cause a higher sampling frequency, resulting in double throughput, or with the same sampling frequency but a lower supply voltage βV_{DD}. Thus the power consumption is reduced to $P_{pipe} = fC_L(\beta V_{DD})^2 = \beta^2 P_{seq}$.

Loop Unrolling

Loop unrolling[13] is a technique to add parallelism to a design. Unlike loop pipelining, which permits loop iterations to start at every second clock cycle, loop unrolling can execute all current loop iterations in the design in one clock cycle when there are no dependencies in the successive loop iterations.

Partial loop unrolling means the unrolling takes place in the loop iteration internally. Take the example of unrolling by a factor of two, and it is equivalent to duplicating the loop body by two and cut down the loop iterations to half. In this way, the loop can be performed by half number of clock cycles.

The fully unrolling loop separates all iterations and permits them to execute in the same clock cycle, while assuming sufficient time to compute the dependencies among the iterations. The following example presents the fully unrolled loop iteration. When the unrolling is applied, the original loop is divided into four sequential accumulations. Without the control logic for the loop iteration, the unrolled function can be executed within one clock cycle, causing less latency than the standard loop.



Normal Loop

```
viod Sum(int in[4], int &out){
    int acc = 0;
    for(int i = 0; i < 4; i++){
        acc += in[i];}
    out = acc;
}
```

Loop Unrolled

```
viod Sum(int in[4], int &out){
    int acc = 0;
    acc += in[0];
    acc += in[1];
    acc += in[2];
    acc += in[3];}
    out = acc;}
```

Figure 4.5: Normal Loop and fully unrolled Loop

4.2 Design of polyphase filter template library

The main goal of this thesis project has been to design a library of parametrizable polyphase filters - or a template library, in other words. The library implements a generic polyphase filter structure, which by parametrization can be customized into a specific realization. The structure is shown in figure 4.6. It contains a

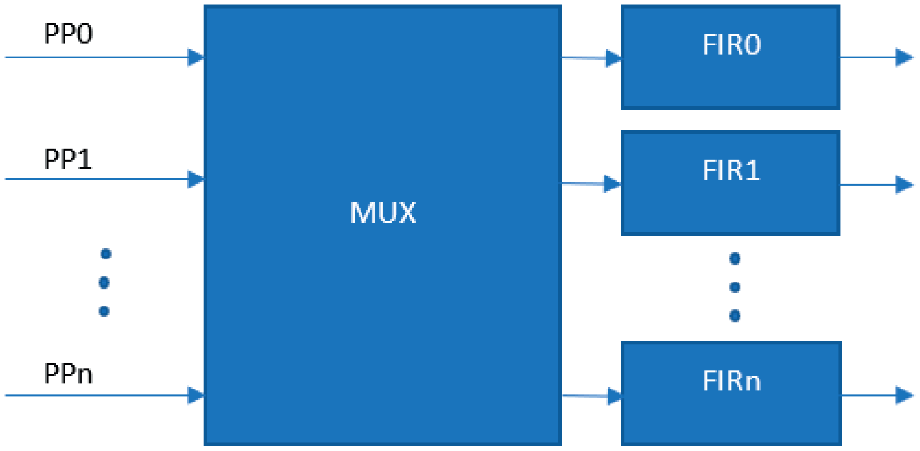


Figure 4.6: Generic polyphase filter structure

number of parallel input and output streams (the polyphases), as well as FIR subfilters, to and from which data can be routed in different ways. For example, in a configuration where we have 4 polyphases, they could all be assigned to one data source, which would then be sending 4 samples in parallel per clock cycle. In another configuration, there could be two data sources, each sending 2 samples per clock cycle. Depending on the desired configuration, the MUX will route data differently.
"Polyphase mode" is not the only configurable parameter, however. We have added support for tuning a lot of different properties of the filter structure, including, but not limited to:

- FIR subfilter type: Generic, Symmetric or Half-band
- Number of polyphases: 1, 2, 4, or 8
- Bitwidths of input, output and coefficient data types
- Option to use runtime-changeable coefficients, or constant compiled-in ones
- etc...

Although we call our creation a template library, it is not a template in the C++ sense of the word. Rather, it is a piece of generic code written in a mix of C and preprocessor macros. By including a central file, the user will trigger the instantiation of the template, which is accomplished by the expansion of macros. The macros, in turn, get their values from constants that the user has defined. Checks have been added to insure that mandatory parameters are defined by the user, and that their values are all legal.

One of the benefits of using preprocessor macros in the library code, is that we can tightly control what hardware will be generated or not. For example, depending on how the filter type parameter has been specified, we use different subfilter implementations. As a result, no "dead" hardware will be sitting around. The different subfilter implementations - namely symmetric and half-band FIR filters - can help save area for cases when the impulse response is symmetric, and in the half-band case, where every other sample/coefficient furthermore is zero.

4.3 Verification

Verification can be a significant part of the whole design since the design will be considered valid after verifying the functional correctness and evaluating the hardware performance if it meets the design specification. The implementation throughout this project is based on C++ development, and naturally the testbenches are written in C++ for convenience. The testbench can be executed in the built-in Catapult verification flow or the GCC compiler. After Catapult generating RTL files automatically, the simulation for the RTL can also be executed by SCVerify(SystemC Verify), which is a powerful tool for hardware verification without writing the VHDL or Verilog testbench manually.

Functional verification is a process that tests all the expected behavior of the design. Typically, it is performed by comparing the expected results with the output of the function. The verification method used in this project is from bottom to top-level. Firstly, the individual blocks are tested, such as the MUX and subfilters, including the symmetrical FIR filter and the halfband filter. The MUX will be tested for the input data distribution to each subfilter. And the subfilters will be verified if the impulse response corresponds with the filter coefficients with a unity input signal. After all the basic modules are tested correctly, the top-level verification can be performed. The modifiable parameters in the top-level, such as the number of polyphase and number of taps, can be tested if the proper hardware structure will be generated by the tool after synthesizing.

To check the generated RTL netlist, SCVerify is applied. It can be executed by modifying the written C++ testbench and automatically simulating the design with Questa, VCS or NCSim. Figure 4.7 illustrates the block diagram of the SCVerify flow.

Verification with SCVerify is accomplished by the following: test stimuli provided in the C++ testbench will be both applied in the original C++ source code and the RTL netlist wrapped with SystemC. The testbench generated by SCVerify will compare the result of the RTL netlist placed by a SystemC wrapper and the C++ testbench to check if they are identical. Meanwhile, Catapult also generates SystemC code that includes the interconnect of the design and interface translator objects. The interface connections of the RTL blocks will be provided by the interface translator objects. Also, the data type will be translated to logic vectors of the RTL netlist. The timing behavior can also be given with built-in functions.

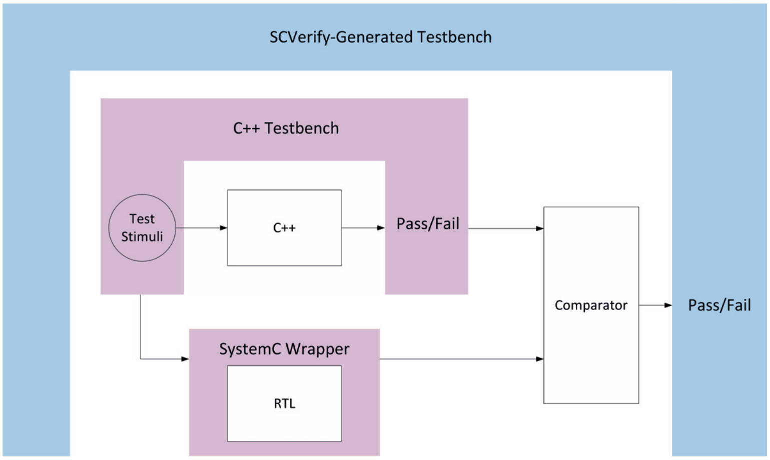


Figure 4.7: Block diagram of SCVerify

4.4 Synthesis

The synthesis in Catapult is done with the tool DesignComplier, which is the last step in the design flow of Catapult, and running synthesis can extract the RTL netlist and generate VHDL/Verilog files. The data path and critical path in the schematics folder are given to review the hardware architecture. The netlist can also be used to calculate the power consumption by Catapult. At the same time, the designer can also get other performance information, including the area and delay of each component. Since Catapult always overestimates the score value of the area and underestimates the slack when generating RTL, running synthesis avoids this problem and can check if there are any timing violations in the design.[14] By analyzing the generated VHDL/Verilog files, the interfaces of the implementation can be examined and compared with the reference design in this project.

The comparison between the HLS design and the reference RTL design is a significant part of this project. Since Catapult can only synthesis the design by C/C++, another tool to evaluate the reference design should be applied. DesignVision can make this possible by compiling and synthesizing the RTL files to generate the area and power consumption of the design. The following are the steps to synthesis RTL files in DesignVision.

- The first step is to include the input RTL files, which should be in the depending sequence(from bottom to top), so that the tool can understand the hierarchy of the design and analyze them properly. Also, the corresponding libraries utilized in the design should be specified in the setup step, which define the technology and components in the hardware architecture.
- After analyzing the RTL files, the next step should be to elaborate the design. In this step, the top-level module should be filled in the design blank,

and all the parameters should be defined. This step makes the schematics available once the design is elaborated successfully. As mentioned before, the schematics of the design can be reviewed to investigate if the hardware architecture is correct as expected.

- Then, to synthesis the design, the clock signal should be specified in a certain frequency, while the port name of the clock signal must be the same port in the RTL to ensure it will synthesis the design properly.
- The last step is to run the synthesis and generate the reports. The timing, area, and power consumption reports are available to compare for the RTL design and HLS design performance. The results are recorded and analyzed detailedly in the next chapter.

Chapter 5

Results

In this chapter, the verification and simulation result of the parameterizable polyphase filter designed by HLS is given, then the comparison between the reference design and equivalent HLS design is presented and the reason for the differences is analyzed. Besides, the arguments in the HLS design are also varied to investigate their effect on performance in latency and area.

5.1 Simulation Result

The simulation result of the reference design and equivalent HLS design of symmetric FIR filter is shown in figure 5.1. Both of the two design have the same parameters. The number of taps is 19 and 10 coefficients. The coefficients are set as linear constant(1.0, 0.9, 0.8, ... , 0.1). And the input is a constant value 1. Thus the output of the two design are impulse response.

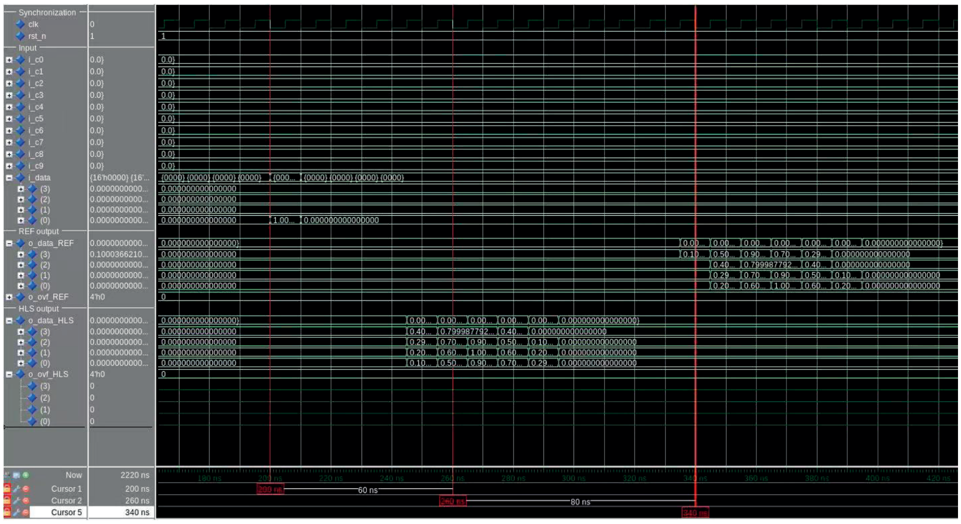


Figure 5.1: Simulation of the reference RTL design and equivalent HLS design

To highlight the output, figure 5.2 is given. The right part is the output of the reference design and the left is the output of the equivalent HLS design. From the figure, the values of the reference design and HLS design are same. But there are fractional delay in the reference design but not in the HLS design. This can be fixed in the future work,by modifying the MUX.

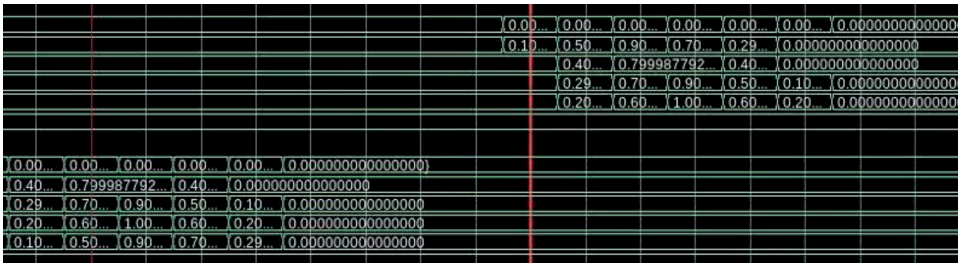


Figure 5.2: Simulation output of the two design

5.2 Comparison between HLS and RTL

The comparison was carried out in the ASIC flow and the methodology in the comparison of the reference RTL design and equivalent HLS design is as follows. First, synthesize the reference design, which RTL includes symmetric and half-band FIR polyphase filters written manually in VHDL, with the same parameter settings including the number of polyphase, number of taps, input/output width, coefficient width and frequency. Then, synthesize the equivalent HLS design with the same arguments. At last, vary one parameter in the HLS design while keeping the others steady to investigate how the varied parameter influences the performance of the design. There are mainly three parameters we examined: the branch mode (variable, 1TX, 2TX and 4TX),the number of polyphases (1,2,4 and 8), and the coefficients setting (variable, constant with linear numbers and constant with all the numbers are the power of 2).

Table 5.1: Parameter setting for the comparison in Table 5.2

No. of PP	No. of Taps	BR_Mode	Data Width	Coeff Width
4	19	Variable	16 bits	18 bits

Table 5.1 presents the parameters for the comparison between reference design and HLS design. Both of the two design are synthesized with 4 polyphases, 19 taps, and variable br_mode. The data width of input/output and coefficient are 16 bits and 18 bits separately.

Table 5.2: Comparison of performance in different frequency

Filter Type	Performance	Frequency	RTL	HLS	Diff (%)
Symmetric	Latency (clock cycles)	1GHz	8	3	-
		2GHz	8	5	-
	Area (no. of gates)	1GHz	5126.3	4043.3	-21.1
		2GHz	5456.3	4690.2	-14.0
Half-Band	Latency	1GHz	8	3	-
		2GHz	8	4	-
	Area	1GHz	2843.4	2471.6	-13.1
		2GHz	3004.1	2946.7	-1.9
Generic	Latency	1GHz	-	3	-
		2GHz	-	5	-
	Area	1GHz	-	6371.8	-
		2GHz	-	7989.4	-

Table 5.2 shows the overall results of the performance in the reference RTL design and equivalent HLS design. The HLS design has reduced up to 5 clock cycles compared to the RTL design and spends less time in each clock cycle. The latency for both RTL design and the equivalent HLS design is both from input to output, which means the delay in MUX in the HLS design is also included. The area of HLS design is reduced 21.1% and 13.1% for symmetric filter and half-band filter separately with the clock frequency at 1 GHz. When the sampling frequency increases to 2 GHz, the improvement of HLS has decreased especially for the half-band filter. This is because in the reference RTL design, the central tap was hard-coded and the multiplication was replaced by shifting. In this way, the area is reduced at the cost of flexibility. The generic FIR filter in HLS has more area than the RTL design due to the fact that the generic filter is the direct form of the FIR filter without pre-adders and the multipliers are close to two times the symmetric FIR filter.

Table 5.3: Parameter setting for the comparison in Table 5.4

Filter type	No. of PP	No. of Taps	BR_Mode	Data Width	Coeff Width	Freq
Symmetric	4	19	Variable	16 bits	18 bits	1 GHz

In Table 5.4, it shows more details of the results displayed previously. Not only

Table 5.4: Comparison between reference RTL design and HLS design

Performance		**RTL**	**HLS**	**Diff (%)**
Latnecy (cc)		8	3	-
WNS (ns)		0.29	0.33	-
Area	Total	5126.3	4043.3	-21.1
	Comb	3775.9	3409.3	-9.7
	Non-comb	1350.3	634	-53.0
	Buf/Inv	200.3	151.8	-24.2
No. of Cells	Total	53444	43342	-18.9
	Comb	46979	40179	-14.5
	Seq	6304	2960	-53.0
MISC	No. of ports	17356	15718	-9.4
	No. of nets	81655	69277	-15.2
	No. of buf/inv	8462	6433	-24.0

the total area, the combinational and non-combinational logic area are also given which assist the analysis of the result. Also, the number of cells, number of ports can be extracted from the result report. From the non-combinational area and the number of sequential cells, it reveals that the main reduction of the area comes from registers, which contributed to a 53% reduction in the area.

To investigate the difference between the reference RTL design and equivalent HLS design, the schedule of both design are given. Figure 5.3 and figure 5.3 show the schedule of HLS design working in 1 GHz and 2 GHz separately. With a lower frequency, the schedule will be more compact and the design is more efficient. Compared to 2 GHz, the read operation, mux, and pre-adder are finished within one clock cycle when the frequency is 1 GHz. And the adder trees are in parallel to reduce latency. In conclusion, the HLS design can achieve a higher frequency but in the cost of more latency and area.

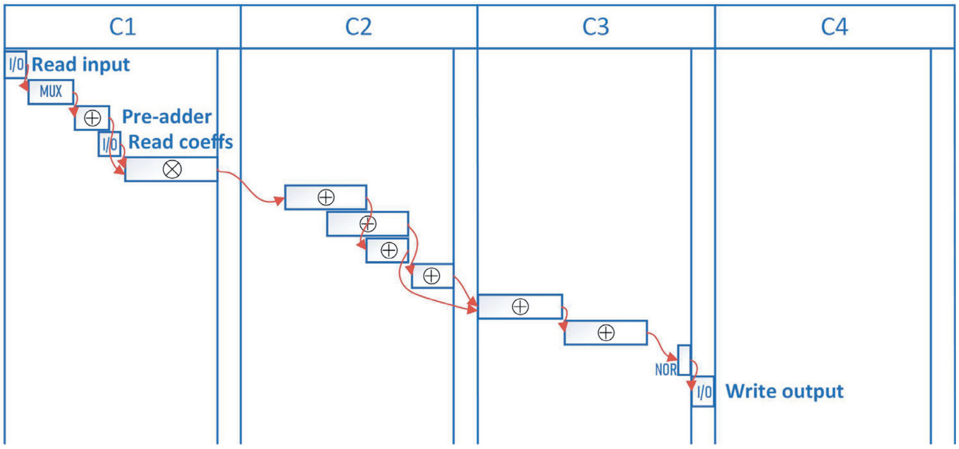


Figure 5.3: Schedule of the HLS design with clock frequency 1 GHz

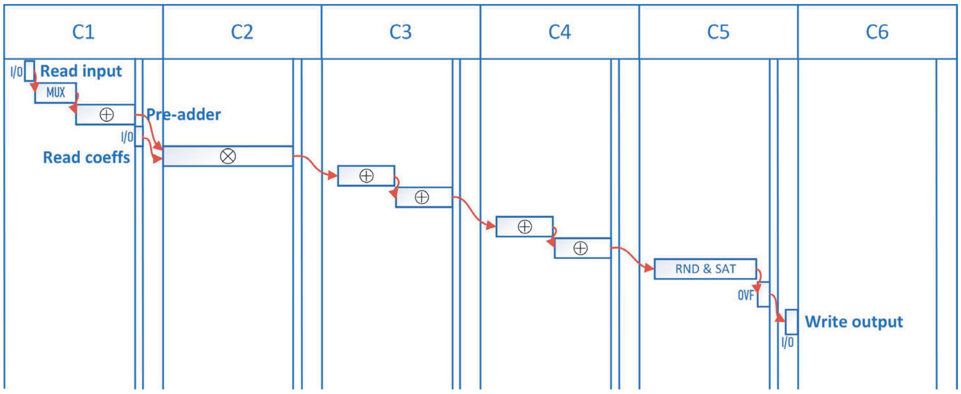


Figure 5.4: Schedule of the HLS design with clock frequency 2 GHz

To make the analysis of the difference between RTL and HLS design more intuitive, the schedule of symmetric FIR reference design is also derived from the synthesis tool Design Vision and as shown in figure 5.5. The RTL design has more slack in each clock cycle. And it also has an empty clock cycle after the MAC. The main difference exists in the addition algorithm in the two designs. While actually, the reference design can be intelligent in the MACs, an adder chain is used in RTL, which makes sure only one clock cycle is spent. And the HLS design is a more balanced implementation. This leads to the potential for further latency reduction in HLS design at expense of more area. In conclusion, the reduction in the area is mainly caused by a reduction of pipeline stages.

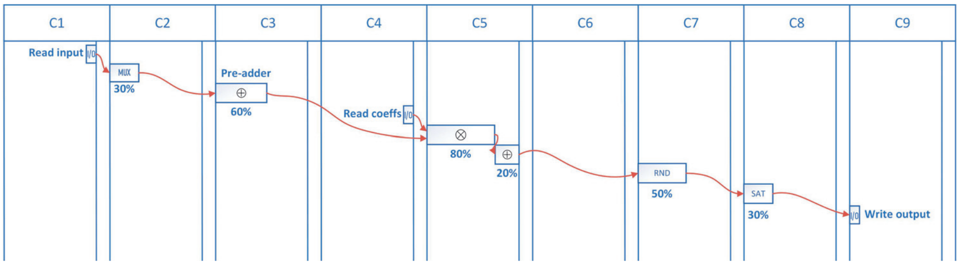


Figure 5.5: Schedule of the reference RTL design with clock frequency 2 GHz

Figure 5.6 presents the hardware structure of symmetric FIR reference design in RTL, which is drawn after analyzing the VHDL codes of the reference RTL design. The stages are separated by the registers. The architecture of the reference design is similar to the HLS design besides more registers, the main difference is the MACs. The adder chain in this structure shortens the latency for the addition to 1 clock cycle but in cost of more inserted registers between the adders and thus causes more area.

After the comparison between the reference design and HLS design, the following part is to investigate the effect of different arguments in HLS design. There are mainly three arguments we have examined. The branch mode, coefficients setting and the number of polyphases. While other parameters can also be varied, such as the bitwidth of input/output and coefficient, these are not so interesting.

Table 5.5: Parameter setting for the comparison in figure 5.7

Filter type	No. of PP	No. of Taps	Data Width	Coeff Width	Freq
Symmetric	4	19	16 bits	18 bits	1 GHz

The bar charts in figure 5.7 show the result of different branch mode in the symmetric filter in HLS. From the figure, it is clear that the combinational logic takes the most area. And for different branch modes, the variable one costs more area,

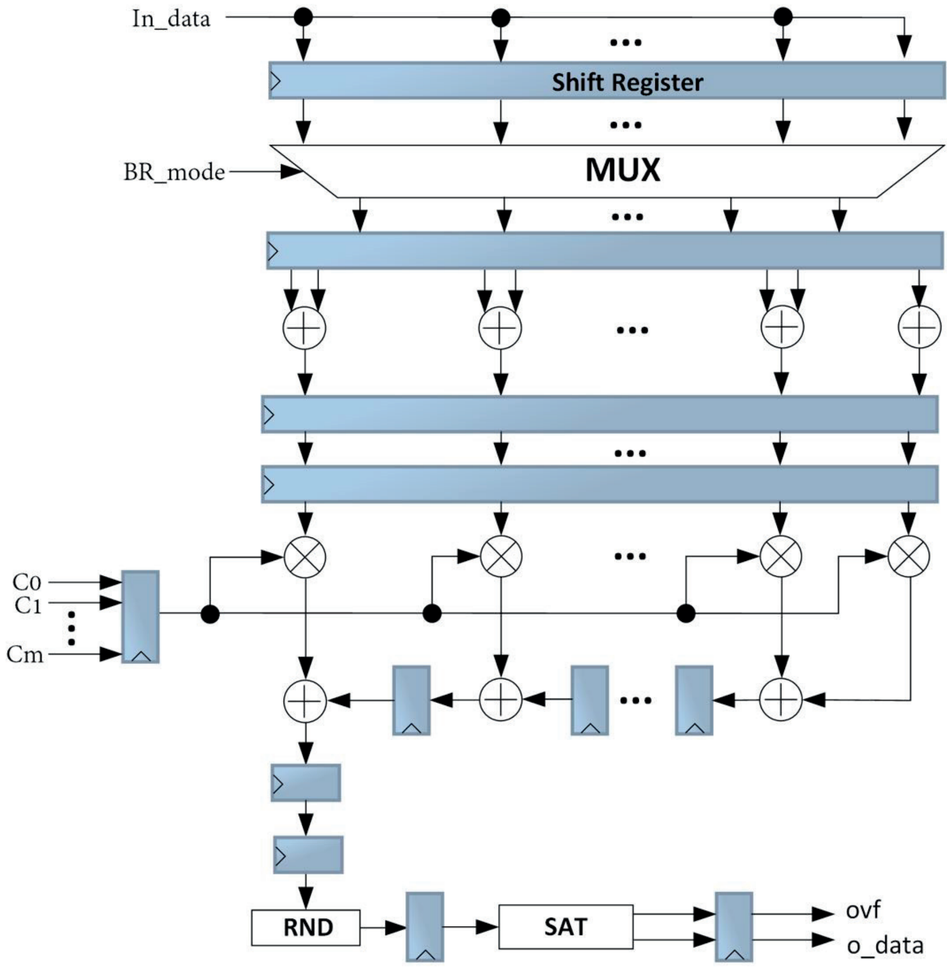


Figure 5.6: Hardware architecture of the reference RTL design

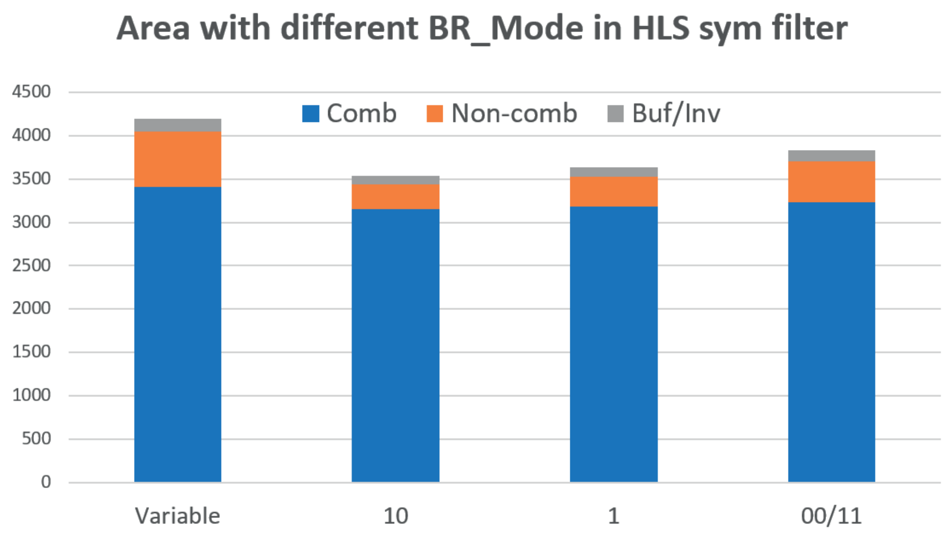


Figure 5.7: Area of different branch mode in HLS symmetrical filter

especially in the non-combinational logic. For the constant branch mode, the multiple TX means they need more room in the shift registers to process more input data, and hence the non-combinational logic area increases.

The next argument varied is the coefficients setting. When the coefficients are constant and power of 2 (1.0, 0.5, 0.25, ... during the synthesis), the area can be reduced to a large proportion. That is because the multipliers are replaced by shifts. Things are different when the coefficients are chosen to be linear constants(1.0, 0.9, 0.8, ...). It will cost more area than the variable ones. That might be account of the fact that Catapult implements multiplication with constants as a series of shifts and adds but no multiplications. To find out if this can be changed will be the in the future work.

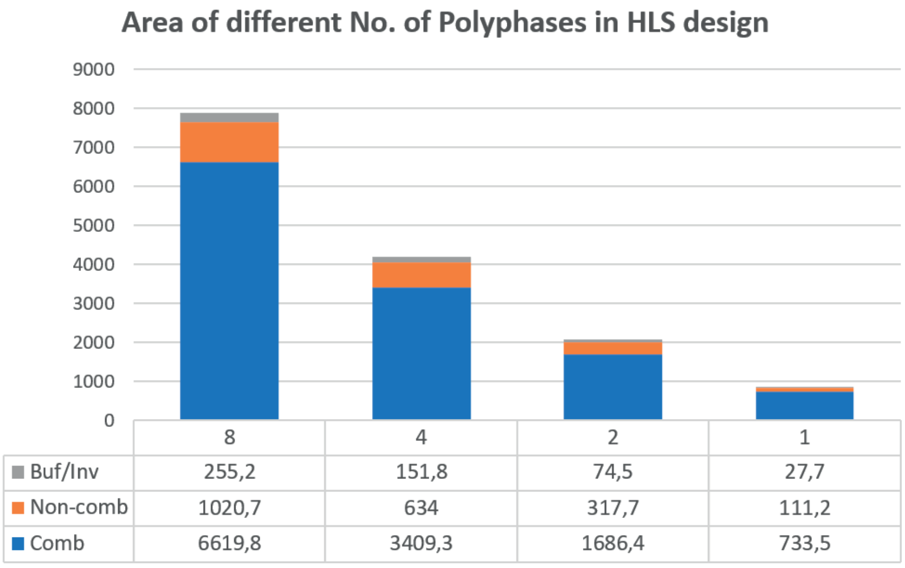
Table 5.6: Comparison of performance with various coefficient sets

Coeff Set	Performance	HLS	Diff (%)
Variable	Latency	5	-
	Area	4690.2	0
Constant, power of 2	Latency	5	-
	Area	1184.3	-74.7
Constant, linear	Latency	10	-
	Area	5509	17.5

Last but not least parameter is the number of polyphase. The number of polyphase means the number of subfilters. And from the bar chart in figure 5.8, it is clear that the area is a linear function of number of polyphase, which corresponds with the theoretical knowledge since the subfilters are all same. And also, the ratio of sequential to combinational logic is approximately constant.

Table 5.7: Parameter setting for the comparison in figure 5.8

Filter type	No. of Taps	BR_Mode	Data Width	Coeff Width	Freq
Symmetric	19	Variable	16 bits	18 bits	1 GHz



	8	4	2	1
■ Buf/Inv	255,2	151,8	74,5	27,7
■ Non-comb	1020,7	634	317,7	111,2
■ Comb	6619,8	3409,3	1686,4	733,5

Figure 5.8: Area of different No. of Polyphase in HLS symmetrical filter

Chapter 6

Conclusion and Future work

In this chapter, a short conclusion of this thesis project is given. Also, there are some imperfection in the design, and will be carried out in the future work.

6.1 Conclusion

During the 5-month thesis project, we have implemented a template of parameterizable polyphase in HLS, the design includes the MUX, three filter types(symmetric, half-band and generic). The parameterizable arguments are number of polyphase, bitwidth, number of taps and coefficient binding. We verified the functional correctness by the C++ compiler. And simulated both design to check their timing behaviour. Then we carried out a thorough comparison between the reference RTL design and HLS design by synthesizing them separately and get the results from the report. Finally, the analyse of the reason that causes the difference among the two design is given.

From the results in Chapter 5, it seems that the HLS design has less latency and area compared to the reference RTL design. But it is worth pointing out that the results are based on the fact that the reference RTL design is not optimized and can be improved by a more compact implementation and reducing unnecessary pipeline stages. The main advantage of the HLS implementation is the more efficient design flow and less redundant efforts for designers.

6.2 Future work

There are some possible improvement that found out in the analysing part and can be carried out in the future. First is the addition algorithm in the HLS design, it can be modified as the adder chain as the reference design to reduce latency. The second is the fractional delay in the HLS design should be same as the reference design, which can be done by modifying the MUX.

References

[1] Philippe Coussy and Adam Morawiec. *High-level synthesis*, volume 1. Springer, 2010.

[2] Lawrence R Rabiner, Bernard Gold, and CK Yuen. *Theory and application of digital signal processing.* Prentice-Hall, 2016.

[3] T Saramäki. Finite impulse response filter design, chapter 4 in handbook for digital signal processing, edited by sk mitra and jf kaiser. *John Wiley & Sons, New York*, 14:155–277, 1993.

[4] Bo-You Yu, Peng-Hua Wang, and Po-Ning Chen. A general structure of linear-phase fir filters with derivative constraints. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 64(7):1839–1852, 2017.

[5] Pavel Zahradnik. Equiripple approximation of low-pass fir filters. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 65(4):526–530, 2017.

[6] Sanjit Kumar Mitra and Yonghong Kuo. *Digital signal processing: a computer-based approach*, volume 2. McGraw-Hill New York, 2006.

[7] Eugene Hogenauer. An economical class of digital filters for decimation and interpolation. *IEEE transactions on acoustics, speech, and signal processing*, 29(2):155–162, 1981.

[8] Michael Portnoff. Time-frequency representation of digital signals and systems based on short-time fourier analysis. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(1):55–69, 1980.

[9] Richard G Lyons. *Understanding digital signal processing, 3/E.* Pearson Education India, 2004.

[10] HJ Landau. Sampling, data transmission, and the nyquist rate. *Proceedings of the IEEE*, 55(10):1701–1706, 1967.

[11] Maurice Bellanger, Georges Bonnerot, and Michel Coudreuse. Digital filtering by polyphase network: Application to sample-rate alteration and filter banks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 24(2):109–114, 1976.

[12] Michael Fingeroff. *High-level synthesis: blue book.* Xlibris Corporation, 2010.

[13] Michael Fingeroff. *High-level synthesis: blue book*. Xlibris Corporation, 2010.

[14] Ayla Chabouk Jokhadar and Carlos Gomez Gonzalez. High level synthesis for design of video processing blocks. page 31, 2015.

Printed by Tryckeriet i E-huset, Lund 2021

Series of Master's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2021-845
http://www.eit.lth.se