UNIVERSITÀ DI PISA

Department of Information Engineering
MSc Computer Engineering

Electronics and Communications Systems

Amplitude Equalizer

Federico Cristofani

Academic Year 2022/2023

Contents

List of Figures

List of Tables

1. Introduction

1.1 Specifications

Design a digital circuit that performs amplitude equalisation of a triangle wave signal. The wave is sampled by an ADC, which outputs signed samples represented on 8 bits. The sampling frequency of the ADC is 8 times that of the input signal. Suppose that sampling is carried out in such a way as to have for each period of the input signal also the sample on the peaks of the triangular wave.

Equalisation must be done by multiplying the samples output by the ADC by a correction factor represented with sign on 8 bits.
This factor is obtained recursively according to the equation:

$$\rho_{k+1} = \rho_k + (R - |P_k \cdot \rho_k|) \tag{1.1}$$

Where ρ_k is the value of the correction factor at step k, P_k is the sample on the peak at step k and R is the reference at which the amplitude on the peak of the output triangular wave must reach. Keep in mind that for each period of the input signal there are two peaks (one positive and one negative), both of which can be exploited to perform the updating of the factor ρ for which the index k is updated with a frequency equal to twice that of the input signal.

1.2 Amplitude Equalizer

An *amplitude equalizer* for a triangle wave is a digital circuit that receives as input the samples of a signal with such a waveform and produces as output the same signal whose amplitude[1] is raised to a reference value. The operation performed internally is a multiplication of each input sample by a correction factor proportional to the difference between sample on last received peak and the desired amplitude value, computed dynamically according to the the recursive formula (1.1).

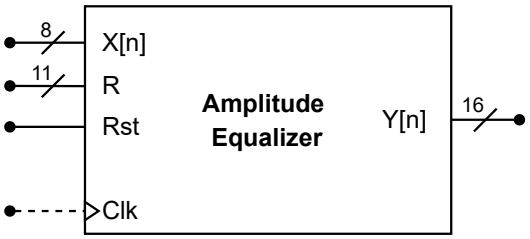


Figure 1.1: Digital circuit interface

.

1.2.1 Triangle wave

A triangular waveform is a periodic signal, each period consisting of a pair of consecutive straight ramps moving in opposite directions. The ramps have the same rise and fall slopes, producing a 50% duty cycle. The triangle wave has many applications in the electronic field:

[1]The definition adopted is "peak amplitude", it indicates to the maximum distance from zero

- **Signal processing**, a triangle wave can be used as the carrier signal of a modulation schema.
- **Testing**, electronic devices can be tested by observing the output produced when a signal of known shape is given as input. In particular, the triangle wave can be used when linearity is required.
- **Pulse Width Modulation**, the PWM is a technique used to control the power delivered to a device, such as a motor in order to control its speed. The generation of a PWM signal can be performed by comparing a DC voltage with a triangle wave.
- **Audio synthesis**, triangle wave is one of the basic wave-forms applied to generate sound electronically, it contains odd harmonics that decrease in amplitude as the frequency increases.

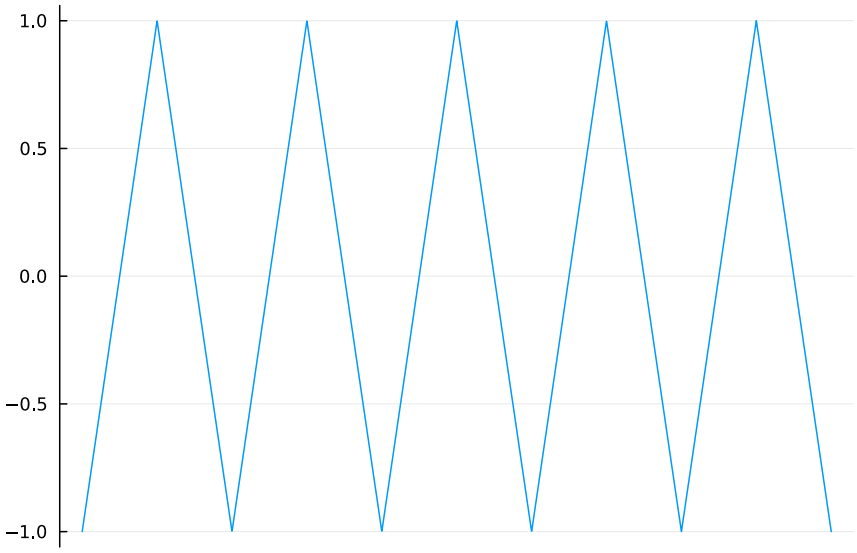


Figure 1.2: Triangle wave

Basically, the triangle wave is the most suitable signal when linear variations are required.

All these applications are sensitive to some properties of the chosen triangle wave signal, such as frequency or amplitude. In the case of amplitude, the use of an *amplitude equalizer* allows to obtain an arbitrary amplitude signal from a single signal reference.

2. Architecture design

2.1 Preliminary phase

The first step was to clearly identify what the interface of the circuit is, in order to understand how the circuit interacts with the outside world. In this case, the interface was already provided by specifications. Next, the components and sub-components required to realize the circuit and their respective interconnections were identified. Finally, all the signals and components involved were correctly sized to meet the initial specifications.

Moreover, during the design phase following choices were taken:

- All signed integers adopts 2's complement representation.
- Unless specifications enforce a precise length, the sizing of signals and components is done in such a way that no overflow occurs.
- The circuit is based on a single clock domain, all operation are completed in one clock cycle.
- Architecture is more oriented towards ease of design than speed and power/area consumption.

2.2 Interface

The interface of the amplitude equalizer, as shown in figure 1.1, is composed by 4 inputs and 1 output. More in details, the inputs are:

- **X[n]**, signed sample of the triangle wave signal whose amplitude is arbitrary.
- **R**, unsigned value representing the desired amplitude of the output triangular wave.
- **Rst**, high active reset signal, used to initialize all the internal components of the circuit in order to reset any spurious values, e.g. when the circuit is powered on.
- **Clk**, clock signal used by internal synchronous components, such as memory elements. Every clock cycle a new sample is presented as input and the output is update[1].

Regarding the output:

- **Y[n]**, signed sample multiplied by the correction factor, in order to obtain the triangle wave whose amplitude is raised to the reference value.

2.3 Components

The architecture of the *amplitude equalizer* was defined following a hierarchical approach, in such a way to obtain the final circuit as a composition of several components, each dedicated to a specific function. In a similar manner, each component was designed following the same approach, defining sub-components. The procedure was iterated until sub-components composed only of basic logic gates were obtained.
The higher-level components were directly derived by observing the formula (1.1) that the circuit has to implement:

[1] Actually, as will be described in the following, the output requires a initial warm-up of few clock cycle, during which each output sample is zero

- P_k Peak detector (PD)
- $P_k \cdot \rho_k$ Signed multiplier (MUL)
- $|P_k \cdot \rho_k|$ Absolute value (ABS)
- $R - |P_k \cdot \rho_k|$ Signed subtractor(SUB)
- $\rho_{k+1} = \rho_k + (R - |P_k \cdot \rho_k|)$ Accumulator(ACC)

The above statements can be summarized:

$$\rho_{k+1} = \underbrace{\rho_k + (\overbrace{R - |\underbrace{P_k}_{\text{PD}} \cdot \rho_k|}^{\text{SUB}})}_{\text{ACC}} \quad \overbrace{}^{\text{MUL}} \quad \overbrace{}^{\text{ABS}} \tag{2.1}$$

2.3.1 Peak Detector

The task of the *peak detector* is to emit the samples corresponding to the peaks (positive and negative) of the input signal, discarding the others samples.

The interface is composed by:

- **Input**, signed sample of the triangle wave signal.
- **En**, active high enable signal, when the component is disabled the output retains the last value memorized and the possible changes of input value are ignored.
- **Rstn**, active low reset signal, used to initialize the output of the circuit and internal memory elements to zero.
- **Clk**, clock signal, necessary for the internal memory elements.
- **Output**, sample corresponding to a peak of the input signal, it may be the upper or lower peak.
- **NewPeak**, active high output control signal that indicates the presence of a new peak in output in the current clock cycle.

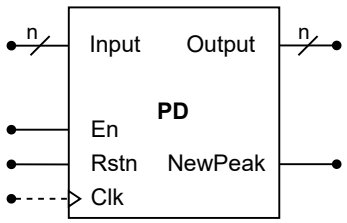


Figure 2.1: Peak detector interface

This component exploits the signal sampling assumption made by the specifications, according to which for each period the samples on the peaks are always captured. This assumption allows to obtain the desired functionality easily designing a down-sampler. From the specifications, each signal period consists of 8 samples, 2 of which are associated with the positive and negative peaks. This results in a down-sampling factor of 4, i.e. every 4 clock cycles the component emits a new value (the one on the peak). The down-sampler is composed by a counter that keeps track of the number of clock cycles and a memory register that stores the value of the current peak. Each time the counter goes to zero, a simple logic made up by multiplexers selects the appropriate line, causing the outputs to be updated, both the sample on the peak and the control signal *newPeak*. At the same time, the peak value is stored in the internal

register, so that the output can hold the value of the last peak encountered until the counter output returns to zero again. The output of the down-sampler could be connected directly to the output of the internal register, but in this case the output of the component would be delayed by one clock cycle. This problem is solved by using a multiplexer controlled by the counter output, which chooses between the input sample and the register output. So, the output takes the input value when the counter goes to zero, otherwise it takes the value stored in the register.

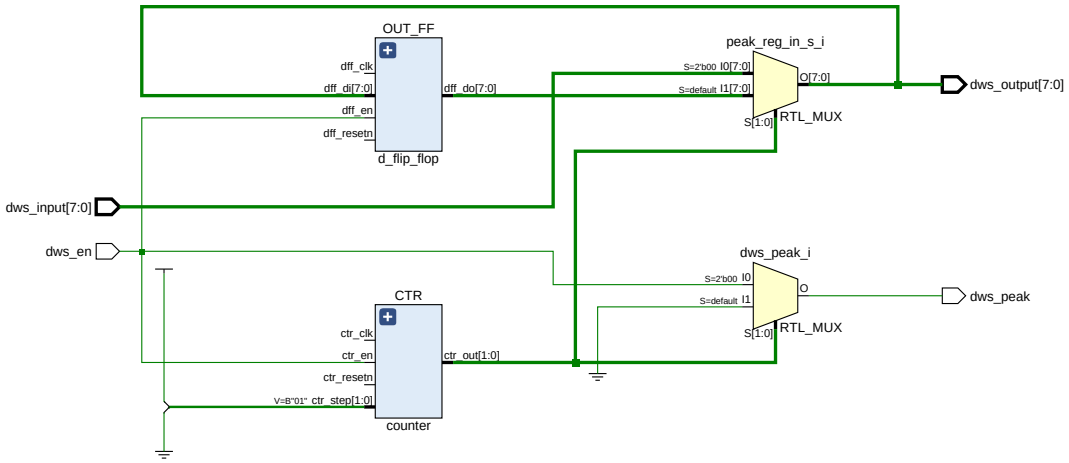


Figure 2.2: Down-sampler schematic

Note that the signals clk and rstn are not shown in the figure to simplify the diagram

2.3.2 Signed multiplier

The signed integer multiplier is the most tricky component to be designed, taking also into account that the circuit must respect the initial assumption to complete the operation in one clock cycle and must manipulate signed integers.

The interface is the following:

- **A**, first factor of the multiplication, requires 2's complement integer representation.
- **B**, second factor of the multiplication, requires 2's complement integer representation.
- **Prod**, result of the product represented on 2N bits in order to represent any possible result.

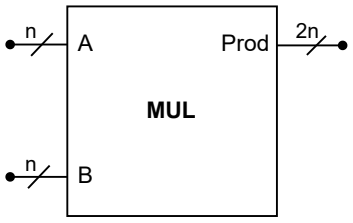


Figure 2.3: Signed multiplier interface

Taking in mind the ease of design over others properties the selected architecture is the standard "parallel multiplier", in which figures out AND logic gates to perform the one-bit multiplication and full/half adders organized as a matrix to perform the addition of partial products, resembling the common hand-made algorithm.

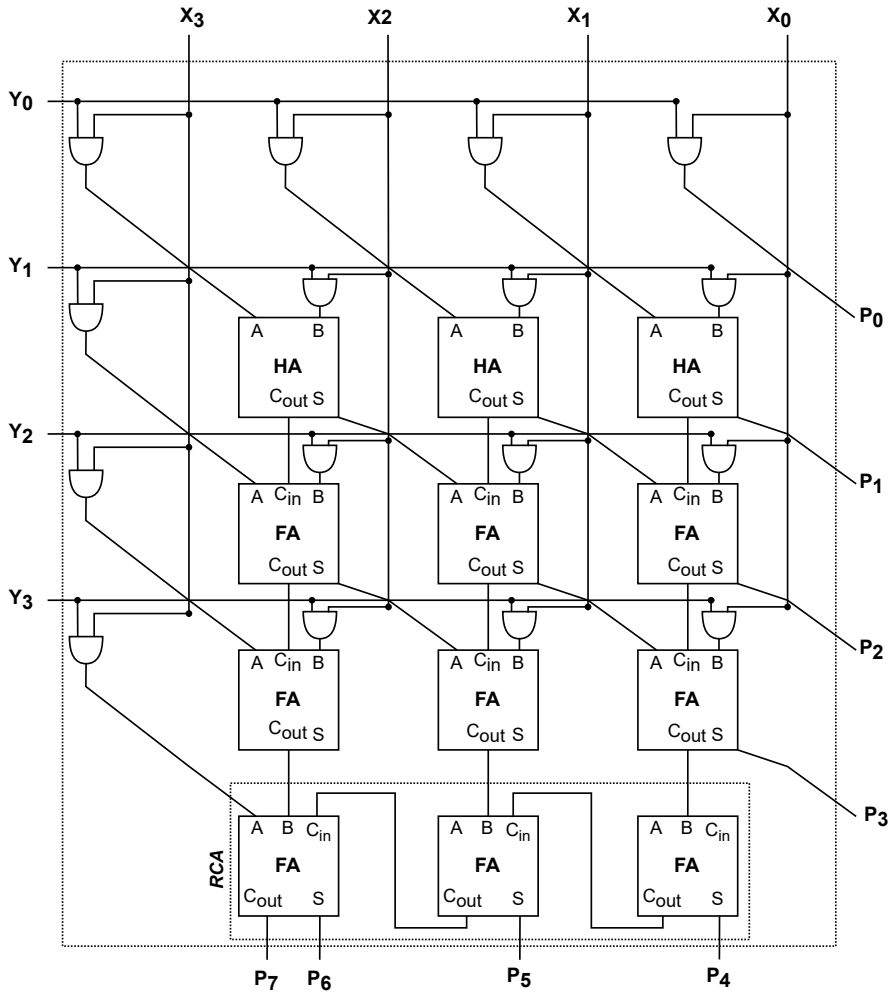


Figure 2.4: Standard parallel multiplier (4x4) schematic

Actually, the standard parallel multiplier is capable of performing multiplication of unsigned integers, but isn't able to handle 2's complement integers properly. This limitation has been overcome with a not-so-performing trick: the multiplication considers the absolute value of the two factors, then the sign of the result is adjusted via a simple logic that only negates the result if the two inputs have opposite signs. All these operations are implemented internally by the multiplier and are transparent to the outside world. The two input factors go through the absolute value component before being multiplied, then the result goes through additional logic to handle the sign.
The design of the absolute value is the same of the one that will be presented in the following subsection. Regarding the handling of the result sign, a multiplexer chooses the correct line between the positive and negative result. The control signal of the multiplexer is obtained via a XOR logic gate between the MSB bits of the two inputs. The negative branch of the result is designed implementing the well-known operation of negation for 2's complement integer.

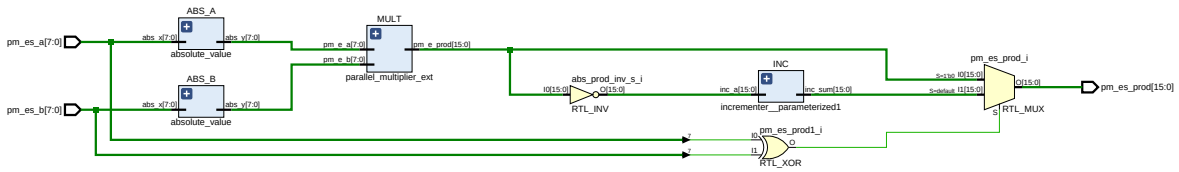


Figure 2.5: Signed multiplier schematic

2.3.3 Absolute value

The *absolute value* realizes the classical math operation of abs, it takes a signed integer number as input and outputs a positive integer that corresponds to the magnitude of the input value. Since the adopted representation is the 2's complement, the design circuit can be easily derived from literature.

The interface is the following:

- **X**, input value, requires 2's complement integer representation.
- **Y**, output value, positive integer that adopts 2's complement representation.
- **Ow**, overflow flag signal that raises when the number of bits of the output are not sufficient to represents the absolute value of the input value.

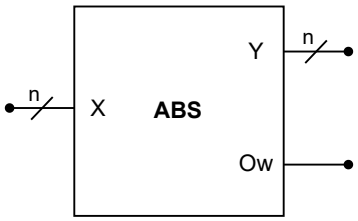


Figure 2.6: Absolute value interface

The component internally is made up by two parallel branches. If the input value is already a positive integer, no logic is needed and the output directly follows the input. Otherwise if the input value is a negative integer the value is complemented and the result is incremented by a unit. This operation requires a barrier of *NOT* logic gates and a sub-component that realizes the increment.

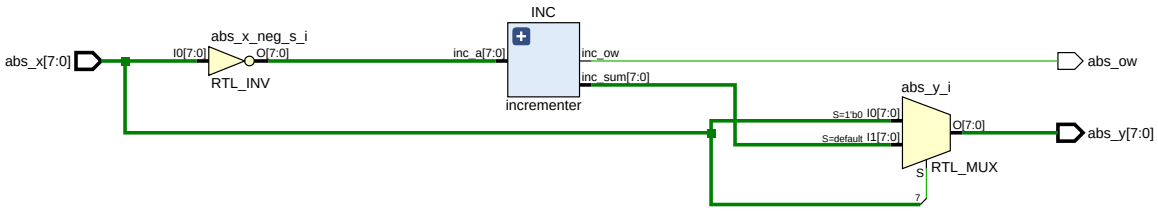


Figure 2.7: Absolute value schematic

2.3.4 Signed integer subtractor

The *signed integer subtractor* is designed leveraging another component yet designed for other components, i.e. the ripple-carry-adder. One of the 2's complement strength is the possibility to reuse the adder to perform the subtraction, so to save design time this solution has been adopted.

The interface is composed by:

- **A**, subtrahend, requires 2's complement representation.
- **B**, minuend, requires 2's complement representation.
- **Diff**, result of the subtraction, represented on N + 1 bits in order to represent any possible result.

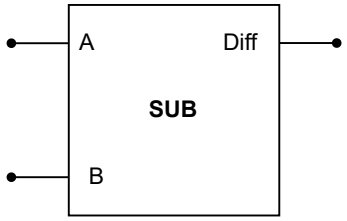


Figure 2.8: Subtractor interface

The component haven't the external borrow-in signal because no other components of the circuit require such signal. Internally there is very little additional logic required to achieve the subtraction operation via the adder. It's sufficient to negate the subtrahend through an inverter and extends the inputs on N + 1 bits. The extension is the one valid for the 2's complement representation, i.e. replication of the MSB bit. The adder must be sized to accept inputs on N + 1 bits, so that the result is on the same number of bits. This simple trick is sufficient to obtain a component that is able to represent any result considering the possible inputs.

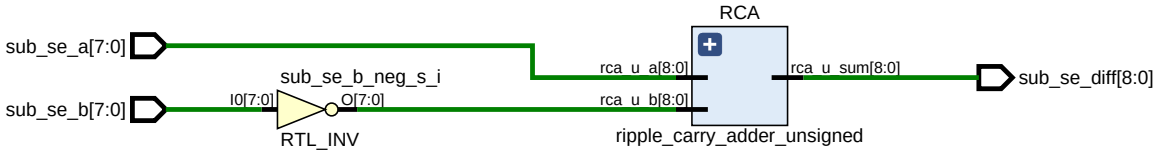


Figure 2.9: Subtractor schematic

2.3.5 Accumulator

The *accumulator* is the component that allows to calculate recursively the correction factor. At each step, i.e. when peak is encountered, it stores the correction factor in its internal memory and outputs that value until the next step when the value will be updated. Since the update is an addition between the old correction factor and an input quantity, the operation performed is an accumulation, hence the name of the component.

The interface is the following:

- **Step**, input quantity to be accumulated, i.e. the value that is added to the current stored value.
- **Keepn**, active low control signal, if in low state the accumulator hold the value that has stored internally, any modification of the *step* input will be ignored.
- **Rstn**, active low reset signal, used to initialize the output of the circuit and the internal memory elements to zero.
- **Clk**, clock signal needed by the internal components. Each clock cycle, if the control signal *Keepn* is in high state, the accumulator update its internal value and the output.
- **C_fact**, memorized internal value, truncated to the number of bits as required by specifications.

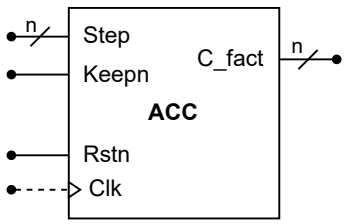


Figure 2.10: Accumulator interface

From the description of the component, it might seem that a design from the scratch is required, however, with a few tricks it's possible to achieve exactly the desired function by means of a counter with an enabler. Usually a counter increases its output value at each clock cycle, but if the *keepn* control signal is connected to the enabler of the counter, it's possible to force the latter to hold a value until the next update, i.e. when the *keepn* signal switches back to 1.
From specification the correction factor is represented on 8 bits, while the accumulator must be designed to accommodate 20 bits. Since the output of the accumulator is considered as the correction factor from the other components the reduction is performed internally. The reduction cannot be performed guaranteeing that no information is lost, however, taking into account that the components deal with 2's complement integers, the reduction is performed by truncating the LSB bits, saving at least the original sign. The effect of this will be explained in more detail in the following chapters.

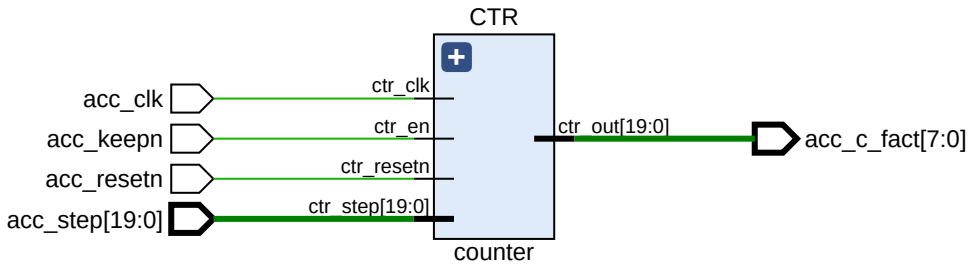


Figure 2.11: Accumulator schematic

2.3.6 Common sub-components

During the presentation of the main components, some sub-components popped up. Their design don't deserve a dedicated in-depth study because they are basic circuits easily found in the literature. However, a few words will be spent to complete the discussion:

- **Memory element**, designed as a simple *D flip-flip positive edge triggered* with low active reset signal.
- **Incrementer**, designed chaining some standard half-adders, fixing the first addendum to a hard-coded bit '1'.
- **Adder**, designed chaining some standard full-adders, obtaining a simple ripple-carry-adder. The number of connected sub-components depends on the size of the addends that the adder is designed to handle.
- **Counter**, designed as composition of a adder plus a memory element that stores the reached value. The adder takes the step and the content of the memory, via a feedback loop, as input. The output of the counter is taken from the memory element.

2.4 Interconnections

Once all the components have been designed, the most difficult part has been completed. In the following it will be showed how to interconnect them to obtain the final circuit.

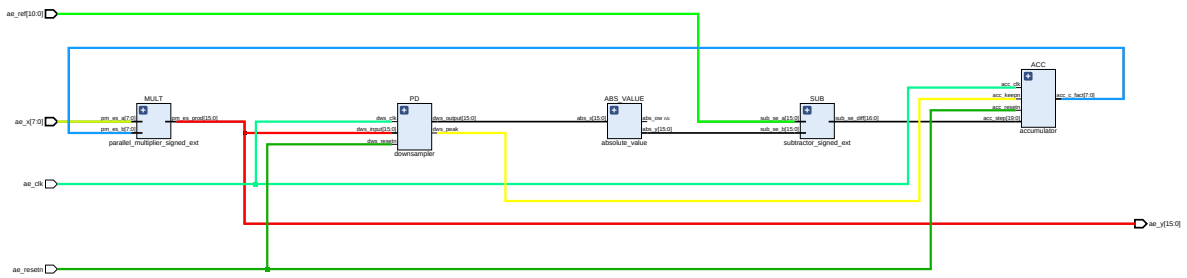


Figure 2.12: Amplitude equalizer schematic

In the schematic diagram, figure 2.12, it's possible to visualize the chain realized from input to output, as well as the feedback loop necessary to implement the recursion of the formula (1.1). The input signal passes through the various component in the following order:

1. Multiplier
2. Peak detector
3. Absolute value
4. Subtractor
5. Accumulator

Some aspects deserve to be better highlighted:

- The multiplier takes the input sample and the correction factor from the feedback loop (blue path). The latter it's initialized to 0 by means of the reset signal, so few steps are required to reach the value at steady-state that allows to obtain the desired output. The speed of convergence will be better analyzed in the chapter dedicated to test phase.
- The result of the multiplication represents the output of the circuit (red path), the remaining components are all dedicated to the update of the correction factor.
- The peak detector takes the sample as input after multiplication by the correction factor, as in subsequent stages is the required quantity. The peak detector is based on a sub-sampling operation, so it completely ignores the value of the input value. However, it can be observed that multiplication does not change the relationship between samples of the same half-period, because in that interval the correction factor is assumed to be constant.
- The reference input signal, from the external view is an unsigned integer, but internally is extended padding with zeros to fit 16 bits. After the extension the representation is equivalent of the one of 2's complement, thus can be correctly interpreted as signed integer by the subtractor.
- The perfect chaining of the components is broken by the control signal flowing from the peak detector to the accumulator (yellow path). That jump is necessary because the logic circuits in between (absolute value and subtractor) have no the information (and output signal) to control the accumulator, i.e. whether to accumulate the input quantity or hold the current value.

2.5 Signal sizing

The sizing of the signal and circuit components was carried out with the aim of avoiding errors due to overflow on internal operations. Some signal sizes were fixed by the requirements:

- Correction factor is on 8 bits
- Accumulator has a memory of 20 bits

The other component sizes were inferred considering the operation that realize:

- **Multiplication**, the input factors are both on 8 bits, so 16 bits in output will represents every possible results. With regard to the internal operation of the absolute value, it should be noted that the component will not be able to generate the absolute value of the smallest negative number (i.e. -128 out of 8 bits). However, the result of the operation will be considered an unsigned integer. It can therefore be seen that the representation of -128 for the 2's complement is the same as 128 for unsigned integers, resulting in the correct result.
- **Peak detector**, both input and output are on 16 bits because the component receives the multiplication result as input and doesn't changes the values.
- **Absolute value**, both input and output are on 16 bits, the extension of the output was not taken into account although a possible overflow may occur. The reason for this is the assumption that the least negative number will not be present as an input to the component, due to the fact that the reference input signal is less than 16 bits. Thus, if the reference is less than 16 bits, there is no reason why the input to the absolute value component should be the least negative number.
- **Subtractor**, the inputs are on 16 bits, so 17 bits for the output will be enough to represent every possible output.
- **Accumulator**, the accumulator is sized to 20 bits as specified by requirements, so it's necessary an extension from the output of the subtractor to the input of the accumulator. The output is truncated from 20 to 8 bits.
- **Reference signal**, the reference is on 14 bits because given the fact that both the input sample and correction factor are considered as integers on 8 bits, the maximum reachable value[2]can be represented on 14 bits, so there is no reason to allow a larger value to be specified as the reference.

Note that the given values are the default one and those that were used during the synthesis and implementation phase. As will be shown below the hardware description make extensive use the *generic* parametrization in order to guarantee the maximum flexibility in the case of requirements modification or reuse of the component in a larger project.

[2]The integers representation is 2's complement, so the range of possible values in case of 8 bit signal is $[-128, +127]$, so the maximum value that can be reached by the multiplication is $127 * 127$ for positive peaks and $-128 * 127$ for negative peaks, both of them can be represented on 14 bits (unsigned representation of the peak magnitude).

3. Hardware description and Testing

3.1 Description style and code organization

The digital circuit has been fully described using VHDL language, mainly adopting the structural approach resembling the followed design approach and in somewhere the data-flow style. Rather then it may be not the optimal choice to produce the best description in terms of performance and resource utilization, it allows an easy reuse of code and simplification of the description. The behavioral approach has been limited as much as possible to keep the description of the circuit at the lowest possible level. The organisation of the source code directory follows the structure of the components: the description of the final circuit is located in a dedicated folder, while the various components are located in a second folder, which also contains a further folder with sub-components. The source directory present also a wrapper sub-directory, containing wrappers useful during the synthesis phase. All test-bench code is located in a dedicated directory without any further division.

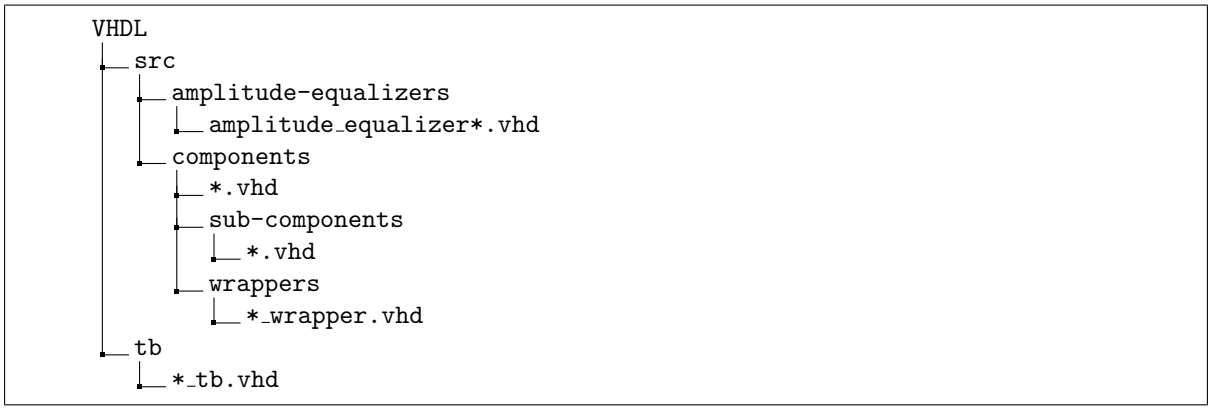


Figure 3.1: VHDL code organization

The VHDL code merely translates what has been designed, so it should not require much effort to understand it. However, certain conventions adopted are made explicit to further simplify the reading of the code:

- The interface signals start with an acronym for the full name, e.g. interface signals for amplitude equalizer are in the form *ae_**, while the ones for accumulator are *acc_*.
- All internal signals used to interconnect components or logic gates have the suffix *_s*.
- All components description provides a generic section which can be used to instantiate the component with the desired size (e.g. number of bits in case of memory element). This allows flexible reuse of the components throughout the others component descriptions. If the instance doesn't specify a generic map the default value will be used.

3.2 Testing phase

The components were described following a bottom-up approach, so that each component could be tested during the development of its description, catching any errors as soon as possible. For each component a

test-bench has been realized, in order to test the expected behavior through simulation. The test-bench file, the one that determines the evolution of the simulation, simply consists of an instance of the device under test (DUT), and some external signals connected to the interface of the DUT to determine the inputs and verify the output produced. For each component the test, although may be not exhaustive, consists in:

- Checks the reset signal, if present.
- Checks the output in presence of "normal" input values, e.g. checks the result of the multiplication of 10 times 5.
- Checks the output in presence of edge input value that may cause overflow, e.g. least negative number for the absolute value component.

Due to the simplicity of the implemented functionalities, the results were manually validated by observing the wave drawn by the simulator. The results of intermediate components are not so interesting, so only the tests for the final circuit are considered below. Unless otherwise specified, each period of the input triangle wave is composed by the samples:

$$[-10, -5, 0, +5, +10, +5, 0, -5] \tag{3.1}$$

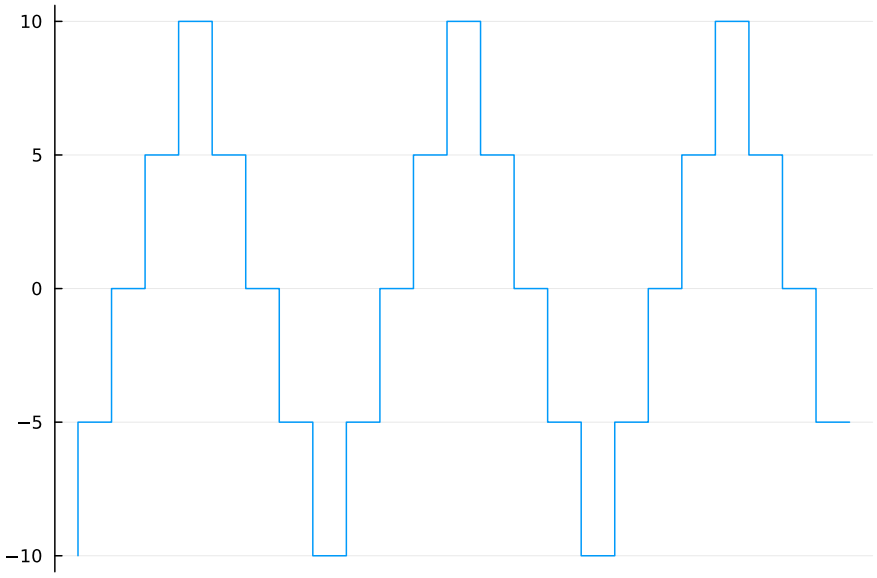


Figure 3.2: Sampled triangle wave

3.2.1 Output wave

The first test consists of observing the output wave obtained by running the simulation by fixing the input reference at the value of 1023 ($2^{10} - 1$). Unfortunately, in order to obtain a complete view of the waves and to appreciate the convergence to the final value of the amplitude, it is necessary to zoom out the wave graph, so that the triangle waves are very compressed and is not possible to distinguish their periods. However, the test wants to demonstrate the the output wave amplitude reaches the reference value, so this view fit for the purpose of test.
Note that the amplitude of input and output waves in the following figures are not in scale.

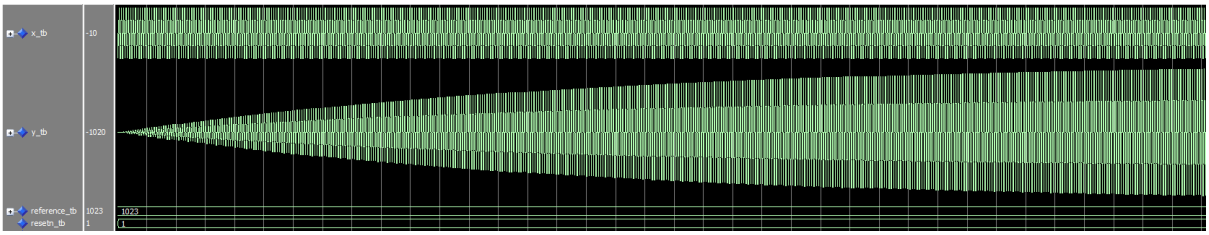


Figure 3.3: Amplitude equalizer - input and output waves

The first wave represents the input wave, whose amplitude is constant throughout the simulation, the second is the output wave. It can be seen that the amplitude equalizer requires an initial transient phase to reach the desired amplitude, in which the correction factor converges to the final value. Once the steady-state value of the correction factor has been reached, this is maintained as long as there is no change in the input values. During this phase, the amplitude of the output wave is stable at the reference value. The clock input has been removed from the figure because the applied zoom doesn't allow to distinguish any clock cycle.

3.2.2 Phase shifting

During the design of the *peak detector* it has been assumed that input triangle wave phase is always the same, so that a simple operation of down-sampling is sufficient to get the peaks of the wave. This assumption is not so realistic because there are no guarantees on the phase of the wave, the initial sample is arbitrary. It's worth to observe what happens when the phase of the input wave is not the expected one and the down-sampling fails to sample the peaks. Once again the focus is on the amplitude of the signal, so the zoom-out compress the waves but allows the amplitude to be observed at a glance.

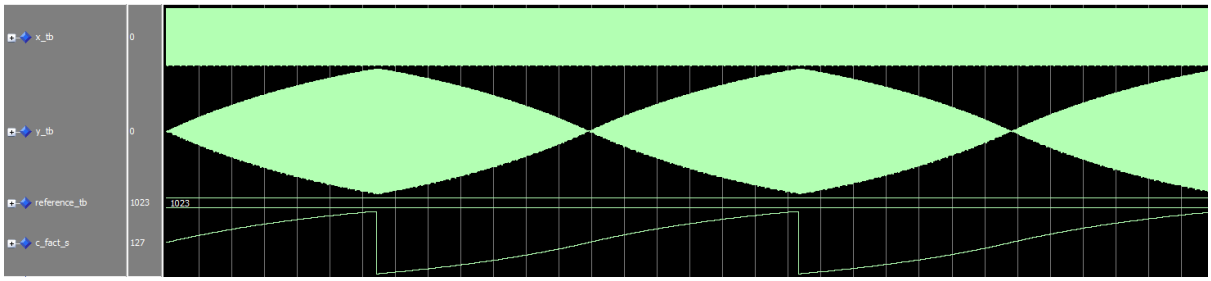


Figure 3.4: Amplitude equalizer - output wave and shifted input signal

The result obtained is not even close to the one desired, but assumes a periodic trend in which the amplitude raises to a maximum value (larger than the reference) and then falls to zero. This trend is due to correction factor, the last wave in the figure, that reaches the upper positive limit (+127) and then falls to the lowest negative number (-128) due to the wrap around of 2's complement integer representation and finally it increases again up to zero. The wrap around happens because the peak detector emits an arbitrary sample that will be considered as the peak of the input signal so the circuit will increase the correction factor until that arbitrary sample multiplied by correction factor will reach the reference. In this test the phase shifting is one, so the value *-5* will be considered as the peak, however even the larger possible correction factor, considering 8 bits-length, is not sufficient to reach the reference, so the logic, following the implemented formula (1.1), keep increasing the correction factor resulting in a wrap around.

The phase shift problem was solved by attaching a new component on the top of the amplitude equalizer component chain, which is able to remove the phase shift of the input signal in a transparent manner, in such a way to avoid any changes to the other parts of the circuit. The new component indicated as *phase cancellation* provides the following interface:

- **Input**, input wave sample, possibly of a phase shifted signal, which will be buffered internally until the desired phase is obtained.

- **Ref**, reference value that is compared with the buffered samples to spot when the signal reaches the desired phase.
- **Rstn**, low active reset signal, used to initialize internal memory elements.
- **Clk**, clock signal used by internal memory elements.
- **Output**, output sample delayed respect to the input sample due to the internal buffer.
- **Ready**, active high output control signal that indicates when the output signal has the desired phase.

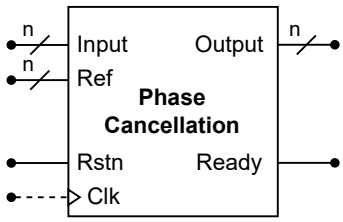


Figure 3.5: Phase cancellation interface

The circuit is very simple, the main idea is to fill a shift register composed by as many register as the samples per period with the incoming input samples. The buffered samples are compared with a reference via a XNOR logic gate, used as a simple comparator that outputs '1' only when the two inputs are identical. When the comparator goes to '1' for the first time, the value is stored in a memory element that will hold it until the next reset signal. More in details, the memory element takes as input the result of a logic OR between the memory output and the comparator result, in such a way when a high value is stored in the memory element the OR will produce '1' regardless the comparator output. The memory output represents the control signal *ready*, while the output sample is taken from the last register, i.e. the one that contains the oldest value. The control signal is necessary to inform subsequent components when the samples are ready to be processed. No additional logic has been added because it's sufficient to connect the control signal to the *peak detector* enabler. When the control signal is '0', the peak detector is not enabled, the internal counter is frozen and the control signal *keepn* is fixed at '0', so that the accumulator retains its initial value and the correction factor remains constant at the reset value. When the control signal *ready* rises to '1', the *peak detector* starts sub-sampling the input samples and all other components react accordingly. Regarding the reference used to spot when input signal is ready, the entire sequence of the wave period may be used, but to minimize the number of bits needed it has been observed that the MSB bits of each sample in a period form a unique sequence that can be used as the reference. This observation allows to use a reference composed by only one bit per sample in the period. For example, considering a period as (3.1) the reference sequence will be: *'10000011'*.

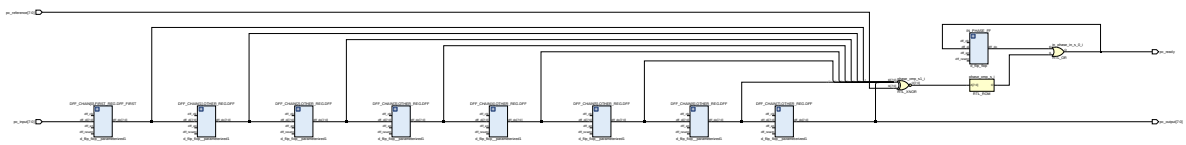


Figure 3.6: Phase cancellation schematic

Note that the signals clk, rstn and internal memory element enablers are not shown in the figure to simplify the diagram

The same test has been repeated after connecting the new component, obtaining the expected result. The main drawback related to the introduction of this component is the initial warm-up time needed to fill the internal buffer. The warm-up has a duration of few clock cycles, depending by the phase of the

signal and the cost must be paid even when the phase is the one expected by the circuit because there is no way to get the information in advance.

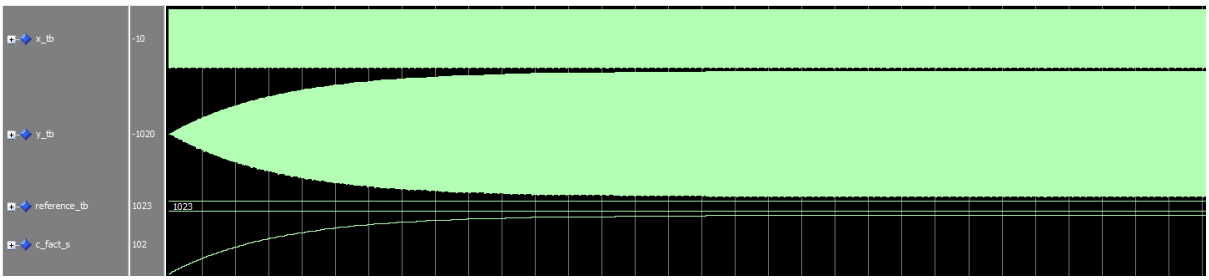
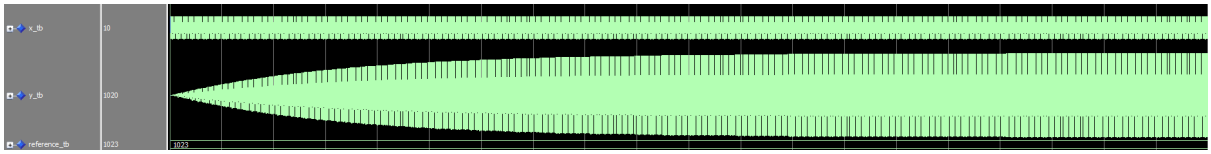


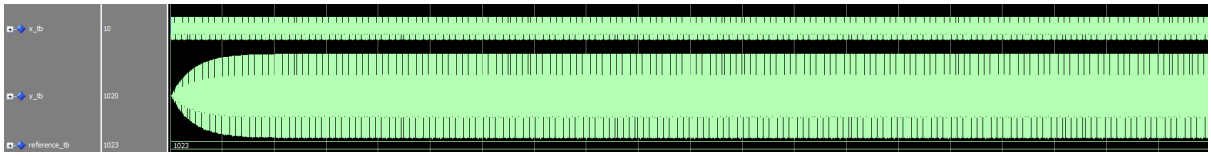
Figure 3.7: Amplitude equalizer - phase cancellation components

3.2.3 Convergence speed and Reference limit

As already highlighted in the previous sections the circuit needs a transitory to produce the final wave signal. The convergence speed can be tuned operating on the accumulator, in particular modifying the truncation of the correction factor. The truncation keeps the bits starting from the MSB to preserve the sign of the factor, however the accumulator requires some clock cycles to modify that bits, hence the transient phase occurs. Deciding to truncate the correction factor starting from an inner bit would result in a higher convergence speed. In the following are reported the result of two different truncation policies, the first keeps 8 bits starting from the MSB, the second skips the two MSBs and then keeps the next 8 bits.



(a) MSB bits



(b) Inner bits

Figure 3.8: Convergence speed comparison

As anticipated truncation starting from inner bits guarantee a faster convergence, but it's worth to mention that to avoids errors due to sign modification the maximum reference should be decreased. The decrease is due to the fact that if the correction factor increases and the accumulator reaches a value that requires the first bit retained by the truncation to be represented, then the sign of the correction factor will fluctuate from step to step, producing unexpected behaviour. If this limit is not taken into account the circuit will produce an output signal totally different from the one expected.

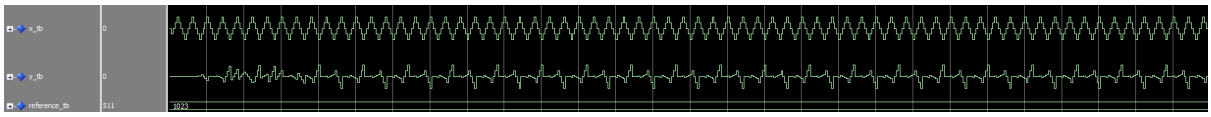


Figure 3.9: Amplitude equalizer - error due to sign modification

3.2.4 Dynamic input modifications

The above tests were performed under the assumption that the input signals are fixed and maintain their value or properties during the circuit elaboration. However, it can be shown that the circuit produces

the expected result even if the inputs are modified during the processing, without the need for a reset.

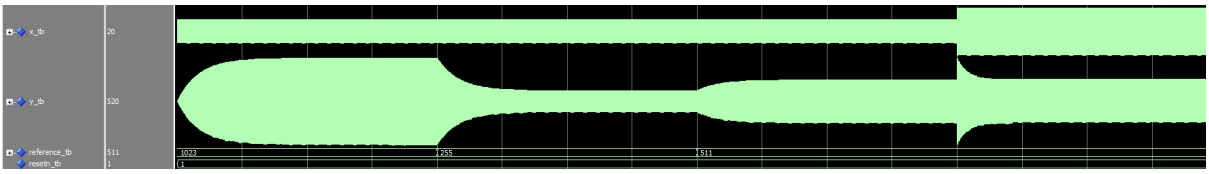


Figure 3.10: Amplitude equalizer - output wave with inputs modifications

Each time the reference is modified the output of the circuit converges to the new value, the correction factor is updated as usual, regardless of whether the initial value is 0 (as after reset) or an arbitrary value. If the amplitude of the input triangle wave changes, the output is affected by the sudden change, but after a few clock cycles the output converges back to the reference value.

4. Synthesis and Implementation

The final step is to synthesise and implement the VHDL hardware description to evaluate the performance of the designed circuit and possibly generate a bitstream capable of programming the selected FPGA. The performance are evaluated in terms of clock speed, area utilization and power consumption, referring to the FPGA identified by the code "xc7z010clg400-1".
The FPGA provides many resources, but only the basic ones relevant to amplitude equalizer evaluation are listed below:

- 4,400 logic slices
  - 4 6-input LUT
  - 8 flip-flops
- 80 DSP slices, including multiplier and adders

The original description doesn't use the VHDL libraries to realize the functionalities, however, it's interesting to compare the results obtained from the fully detailed description with one or more versions in which certain operations are specified at a high level using the libraries. For this reason, several versions of the final amplitude equalizer have been defined, each with the same architecture, but characterised by the substitution of one or more components fully described by the functionally equivalent library operator. The defined versions are indicated as follows:

- **Amplitude equalizer**, no library operator is used.
- **Amplitude equalizer library full**, the library operators substitute the absolute value, subtractor and the multiplier.
- **Amplitude equalizer multiplier\subtractor**, the library operators substitute only the multiplier and the subtractor.
- **Amplitude equalizer library multiplier**, the library operators substitute only the multiplier.

All final description of the amplitude equalizer were synthesized and implemented within a wrapper, i.e. a register barrier connected to the inputs and output of the circuit in order to get a more precise timing evaluation. The synthesis and implementation tool reported the following warnings after the implementation phase, which must be considered when assessing the confidence of the results to be reported later:

- **Pin planning**, the out_of_context mode was not exploited and not pin planning was performed, so two warnings pop up to inform the user that pins are selected by the tool and this may cause problems on the physical device. The generation of bitstream and programming is not taken into account, so these warning have been ignored, even though it may cause a poor evaluation of the implemented design.
  - *40 out of 40 logical ports use I/O standard (IOSTANDARD) value 'DEFAULT', instead of a user assigned specific value*
  - *40 out of 40 logical ports have no user assigned specific location constraint (LOC)*
- **PS7**, the implementation warns the user to about the PS7 cell, i.e. a component on the Zynq-7000 platform that refers to the ARM core processing system available on the board (other than the FPGA). This warning should be better inspected before going on with the device programming, but as already pointed out this step is not covered and furthermore seems to be platform specific.
  - *The PS7 cell must be used in this Zynq design in order to enable correct default configuration*

4.1 Timing

The timing performance is the one more investigated by the analysis made on the synthesis and implementation of the amplitude equalizer. The metrics considered is the the *worst negative slack (WNS)* and *worst hold slack (WHS)* that determine the achievable clock frequency. The implementation phase was performed specifying a constraint of 20 ns for the clock signal, i.e. 50 MHz clock frequency.

Version	**WNS [ns]**	**WHS [ns]**	**Max Freq. [MHz]**
Amplitude Equalizer	1.133	0.187	53.00
Amplitude Equalizer Lib. Mult.	4.495	0.141	64.50
Amplitude Equalizer Lib. Mult.\Sub.	4.485	0.173	64.45
Amplitude Equalizer Lib. Full	4.523	0.166	64.61

Table 4.1: Timing results

The results show that the use of libraries guarantees a better implementation in terms of achievable speed. This makes perfect sense because in the implementation phase, the function can be mapped smoothly on the physical hardware resources instead of recreating the specific architecture described via the VHDL code, that in many case is not very optimized. However, it can be seen that best improvement is given by the substitution of the fully described multiplier, that represent the most critical component. The three library versions, are very close in terms of maximum frequency, but the low confidence on results doesn't allow to given any further conclusion. The critical path, i.e. the one with the longest register-to-register delay, as might be expected is the logic chain composed by the multiplier, peak detector and absolute value components. The fact that the peak detector is part of the critical path, despite the internal memory element, might seem strange, but it's worth remembering the optimization designed to avoid the update delay of an output clock cycle (see subsection 2.3.1). That optimization notably increases the length of the path, reducing the maximum clock frequency supported by the circuit. If a circuit with a higher clock frequency is required, it's possible to modify the design and memorize both outputs inside registers in order to break the logic chain and obtain a higher WNS margin to increase the speed of the circuit, without affecting the final results. Considering the best version of amplitude equalizer, in terms of speed, the new timing report is:

Version	**WNS [ns]**	**WHS [ns]**	**Max Freq. [MHz]**
Amplitude Equalizer Lib. Full	9.763	0.198	97.96

Table 4.2: Modified design timing results

Obviously, the logic chain could be broken down several times, creating a sort of pipeline, reducing the length of the critical path and further increasing the maximum achievable clock frequency.

4.2 Area utilization

Another important metric to consider is the area occupied by the circuit, which can be expressed as the amount of resources required to map the design onto the selected FPGA.

Version	**Slices (4400)**	**LUTs (17600)**	**Flip-Flops (35200)**
Amplitude Equalizer	52	218	86
Amplitude Equalizer Lib. Mult.	52	163	86
Amplitude Equalizer Lib. Mult.\Sub.	49	144	86
Amplitude Equalizer Lib. Full	52	149	86

Table 4.3: Resource utilization

The results shown by the implementation utilization report mirror those of the timing report. The fully described version is the one requiring the most resources, while the others require a comparable amount

of resources. This confirms that without tying the components to a specific architecture, especially if not truly optimized, the implementation process is capable of producing a better result. The number of memory elements used is the same in each version because the components replaced by the library counterpart don't require any memory capacity. Finally, the table doesn't show the DSP blocks because the implementation phase doesn't infer any during the implementation.

4.3 Power consumption

The last metric taken into account is the estimated power consumption for the circuit designed on the selected FPGA. The results obtained are not very significant due to the low confidence of the report produced by the synthesis tool, however the results are state below.
The implementation of the circuit versions are not so different to appreciate variations in energy consumption from one version to another. Therefore, a summary table containing details for each implementation is not shown.

The power estimated is around **0.1 W**, distributed as follows:

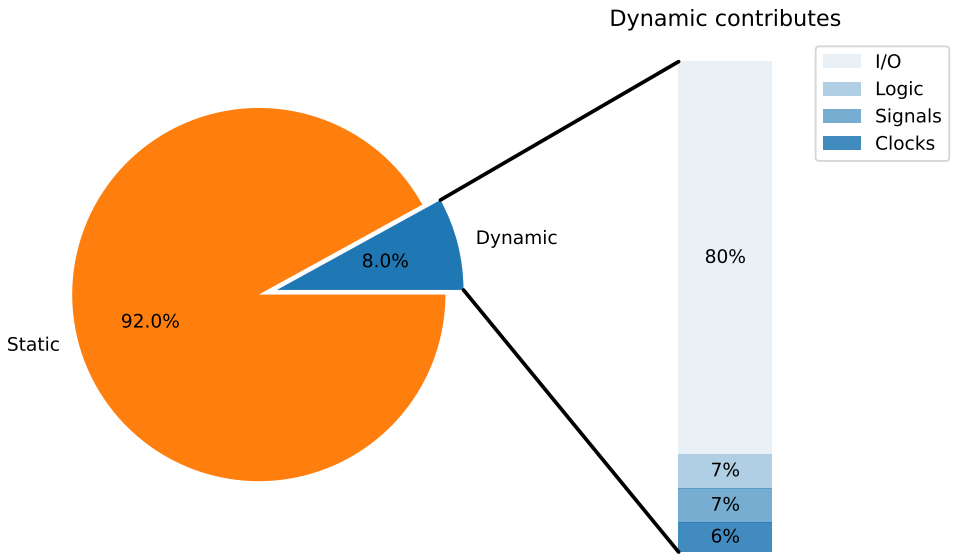


Figure 4.1: Power consumption

The data refers to the full library version, the other versions report minor modifications that are not worth reporting.

5. Conclusions

The designed amplitude equalizer, even if has a very simple architecture, is able to performs its function. The introduction of the *phase cancellation* allows to smoothly handle the problem related to a signal having an arbitrary phase, enhancing the capabilities of the circuit, while the possibility to tune the convergence speed, at the expense of the maximum achievable amplitude, allows to use the same architecture in different contexts. The flexibility of the circuit could be improved allowing a dynamic tuning, not fixing it during the design phase, exposing some pins dedicated to such functionality.
Regarding the performance, the possibility to interleave the logic blocks with registers allows to increase the maximum clock frequency keeping the initial design. The DPS slices capable of performing some advanced operation may be exploited to achieve even better performance, however the implementation phase is not able to infer any DSP block, neither if some components are specified by library operator to let the maximum flexibility to the implementation tool. To understand why these resources are not exploited, a deeper analysis of both the implementation process and the available physical resources is required.