ENV: Envelope Block for SynTech

© 2019 Applied ASIC Design Cornell Tech
Alana Crognale (arc232@cornell.edu)
Eva Esteban (epe26@cornell.edu)
Christine Ku (ck753@cornell.edu)

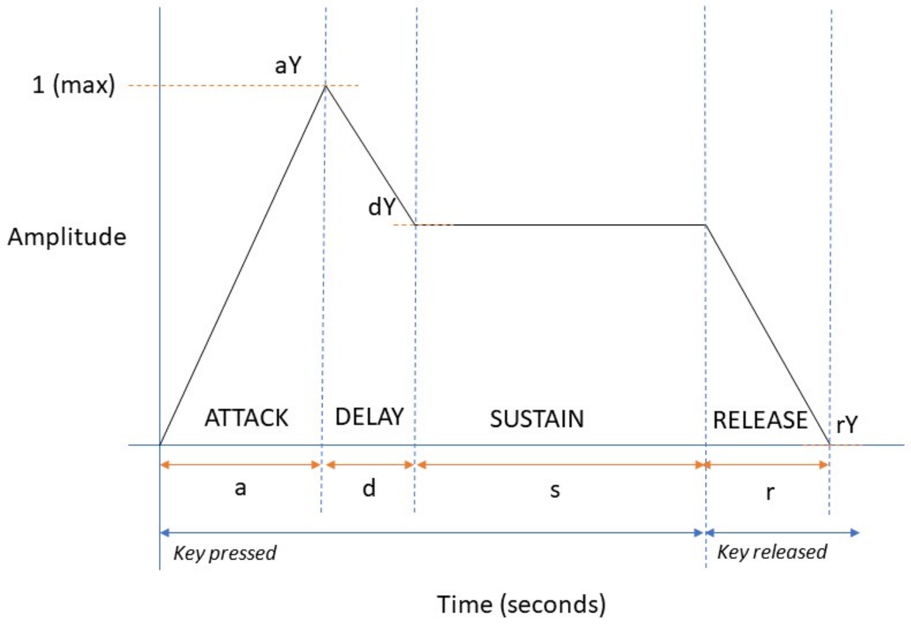


Figure 1: ADSR Envelope Block Shape

Abstract

The purpose of this report is to provide an in-depth description and analysis of the amplitude envelope block (ENV) of the synthesizer. This block controls the sound's amplitude change over time, which is what allows us to differentiate sounds from each other. The enveloped used for this project is an Attack, Decay, Sustain, Release (ADSR) envelope which takes a sample from the oscillator block (OSC) and modifies it according to which ADSR phase of the envelope the sample corresponds to. This sample then gets sent to the amplifier block (AMP), which then sends it to the Nyquist filter block (NYQ). The ENV block requires a minimum period of T = 40 ns, has a total area of 201004.65 μm^2, and consumes 1370 mW. After synthesizing and running verification through Synopsis DC and ModelSim, Cadence Innovus was used to place and route the netlist components. A timing constraint of 40 ns was used. The final design has a total area of 260174.13 μm^2, cell density of 68%, and consumes 6.04 W of power. The place and route was successful with no timing or DRC violations and no violating paths.

1 Introduction

The envelope controls the sound's amplitude change over time. This is what, as humans, allows us to distinguish sounds from each other. In this project we will be using an ADSR envelope before the Amplifier and Nyquist Filter block to create different sounds based on how long a key is pressed on the keyboard.

The envelope block of the system evaluates whether a keyboard key has been pressed. If a key is pressed, it starts a counter to evaluate how long the key has been pressed for. Based on the counter, it then modifies the amplitude with some specified gain depending on the ADSR parameters. This amplitude is then fed into the amplifier block

Specifically, our function checks to see if a key has been pressed to control to envelope. If it has, then the function goes into the ASDR regions. For the attack portion, the output is the amplitude of the envelope at a certain position (which is calculated by multiplying the slope by the position). Then the position is incremented based on some constant a that we established, multiplied by the internal sampling frequency. Once the output amplitude has reached the max, the function moves into the decay region. For this part, the same idea is applied but the slope is negative. Once the output has hit a certain amplitude, the function moves into the sustain region. The sustain region outputs a constant amplitude until the key is released. When the function enters the release region, the same calculations as the decay are applied.

Inputs:

- Key press to indicate the start of the envelope cycle counter. Once the cycle is complete, the counter resets.
- Volume to indicate the amount by which we need to scale our final amplitude output.

Outputs:

- One amplitude sample of the modified signal based on the gain related to one of the four ADSR states.

Parameters (shown in Figure 1):

- Attack: The time it takes for the amplitude to reach the maximum level of the chosen wave.
- Decay: The time it takes for the amplitude to drop to the sustain level from the Attack stage.
- Sustain: The amount of time the amplitude is constant at the sustain level.
- Release: The time it takes for the amplitude to reach the minimum level/silence (0) after the release of the key.

2 MATLAB Golden Model

Our MATLAB golden model evaluates in which state we are currently in based on tracking the current position value and whether a key is pressed to appropriately scale the output.

2.1 Files

- `ENV.m` takes in par and sta as parameters, where par describes the internally defined parameters such as ENV_a_DI and ENV_aY_DI, and sta describes the externally defined parameters such as current position and key press. ENV.m converts all parameters to fixed point, performs logic checks to determine in which state we are, and performs the correct mathematical operations to scale the output based on the state.
- `ENV_init.m` defines all defined parameters passed to ENV.m and converts them to fixed point.

2.2 Parameters and States

Table 1: MATLAB input, internal, and output parameters including fixed-point format and unit.

MATLAB variable name	Fixed-point format	Unit
`sta.INP.Key_DO`	{1,30,'s'}	Sample value (unitless)
`sta.ENV.pos`	{6,25,'s'}	Seconds
`sta.ENV.inc`	{6,25,'s'}	Seconds
`sta.ENV.prevKey`	{31,0,'s'}	Sample value (unitless)
`sta.ENV.out_DO`	{0,31,'s'}	Sample value (unitless)
`par.ENV.a`	{0,31,'s'}	Seconds
`par.ENV.d`	{0,31,'s'}	Seconds
`par.ENV.r`	{0,31,'s'}	Seconds
`par.ENV.aY`	{31,0,'s'}	Sample value (unitless)
`par.ENV.dY`	{0,31,'s'}	Sample value (unitless)
`par.ENV.rY`	{0,31,'s'}	Sample value (unitless)
`par.ENV.aRec`	{7,24,'s'}	1/Seconds
`par.ENV.dRec`	{7,24,'s'}	1/Seconds
`par.ENV.rRec`	{7,24,'s'}	1/Seconds
`par.ENV.glofsi`	{6,25,'s'}	Seconds
`sta.INP.Vol_DO`	{1,30,'s'}	Sample value (unitless)

- `sta.INP.Key_DO` is the input to the ENV module which is a boolean value representing whether a key is pressed.
- `sta.ENV.pos` defines the current position in seconds.
- `sta.ENV.Out_DO` is the output generated by the ENV module which defines the correctly scaled amplitude to be fed to the amplifier block.
- `sta.ENV.inc` is a parameter that defines how how much position increments every clock cycle in seconds.

- `sta.ENV.prevKey` is a parameter that defines whether a key was pressed in the previous clock cycle.
- `par.ENV.a` is a parameter that defines the time interval for the attack stage in seconds.
- `par.ENV.d` is a parameter that defines the time interval for the decay stage in seconds.
- `par.ENV.r` is a parameter that defines the time interval for the release stage in seconds.
- `par.ENV.aY` is a parameter that defines the maximum amplitude for the attack stage.
- `par.ENV.dY` is a parameter that defines the minimum amplitude for the decay stage
- `par.ENV.rY` is a parameter that defines the minimum amplitude for the release stage.
- `par.ENV.aRecip` is a parameter that defines the reciprocal of par.ENV.a to allow us to use multiplier blocks as opposed to dividers in our amplitude calculations.
- `par.ENV.dRecip` is a parameter that defines the reciprocal of par.ENV.d to allow us to use multiplier blocks as opposed to dividers in our amplitude calculations.
- `par.ENV.rRecip` is a parameter that defines the reciprocal of par.ENV.r to allow us to use multiplier blocks as opposed to dividers in our amplitude calculations.
- `par.ENV.glofsi` is a parameter that defines the reciprocal of the sampling frequency to allow us to use multiplier blocks as opposed to dividers in our amplitude calculations.
- `sta.INP.Vol_DO` is an input parameter that defines the desired volume which we use for scaling the amplitude.

2.3 Detailed Functionality of MATLAB Code

In our MATLAB script, the parameters a, d, s and r represent the time intervals between the Attack, Decay, Release and Sustain stages, respectively. In this way, a is the time interval from 0 to the end of the Attack stage, d is the time interval between the end of the Attack stage, or beginning of the Decay stage, and the end of the Decay stage, and so on. The parameters aY, dY and rY represent the amplitude the signal reaches at the end of each stage and beginning of the next one. Therefore, aY corresponds to the value of the signal reached at the point between Attack and Decay, and is the maximum value reached by the signal. The parameter dY indicates the signal amplitude at the point between the Decay and Sustain phases, and rY is the amplitude at the point between Sustain and Release.

After converting parameters to fixed point, we first check whether we need to increment position by sta.ENV.inc - we do so if a key has been pressed and position is less than sta.ENV.r. Then, we check if a key has been pressed. If it has, we check if a key was pressed in the previous cycle - if it was not, we set position to 0 to indicate the start of the attack stage. Then, we check if we are in the attack stage or the decay stage by comparing the position value to sta.ENV.a and sta.ENV.d. Based on which stage we determine, we then calculate the amplitude output based on the equation of the line for the corresponding attack or decay stage. If a key has instead not been pressed, then we again check if a key was pressed in the previous cycle - if it was, then we set position to 0 to indicate the start of the release stage. We then check if we are in the release stage by comparing position to sta.ENV.r - if so, we calculate the amplitude based on the equation of the line for the release stage. We then scale the output with the volume parameter as our final output.

2.3.1 An example

Figure 2 shows an example of the shape of the final output produced from our MATLAB model, produced from the tune Old MacDonald. We can see clearly defined attack, decay, sustain, and release states based on the key presses and volume of the notes.

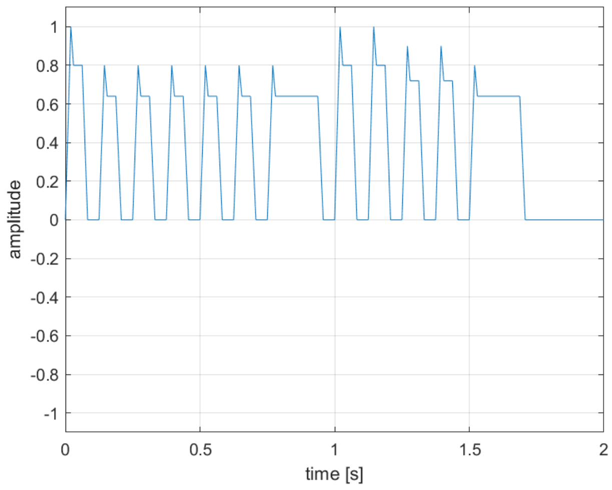


Figure 2: Example MATLAB-produced output

2.4 State of MATLAB Code

Currently, everything in MATLAB is working as intended. To verify, we ran several different test cases with varying parameters. We visually inspected the output to ensure that the envelope shape was appropriately maintained, as well as audially inspected the .wav file outputs to ensure that the sound produced matched with what we expected. We typically simulated only for 1 second to reduce the wait time.

To test different values we ran:

	test	test1	test2	test3	test4
a	0.02	0.01	0.01	0.01	0
d	0.02	0.03	0.02	0.02	0.01
r	0.03	0.03	0.02	0.02	0.03
max (aY)	1	1	1	1	1
decayto (dY)	0.8	0.8	0.8	0.8	0.8
rY	0	0	0	0.5	0

3 Verilog Code

The Verilog code describes an ADSR amplitude envelope (ENV) block that can be integrated with the other components of the sythesizer. The outputs produced by the Verilog code match the MATLAB Golden Model. The purpose of ENV block is to controls the sound's amplitude change over time, which is what, as humans, allows us to distinguish sounds from each other. The block diagram can be found on Figure 5.

3.1 Files

The files required by the Verilog code are:

- `Add.v`: Adds two 32-bit numbers, outputting a 32-bit number. It does not take into account carry in or carry out.
- `COMP.v`: Compares two 32-bit numbers and outputs three 1-bit numbers (for equal, less than or greater than). Each of these corresponds to a wire and will equal 1 when true.
- `FF.v`: Implements a positive-edge flip-flop that stores a 32-bit number. This flip-flop has asynchronous reset.
- `MULT.v`: Multiplies two 32-bit numbers, outputting a 64-bit number.
- `MUX2.v`: Implements a MUX which takes in two 32-bit numbers and a 1-bit selector. It outputs the first 32-bit number if the selector is 0 and the second 32-bit number if the selector is 1.
- `Nand.v`: Implements a NAND gate with two 1-bit numbers as inputs. The block outputs 1 when both inputs are 0, and 0 otherwise.
- `Subtract.v`: Subtracts two 32-bit numbers, outputting a 32-bit number. It does not take into account carry in or carry out.
- `XOR.v`: Implements a XOR gate with two 1-bit numbers as inputs. The block outputs 1 when only one of the inputs is 1, and 0 otherwise.

The final Verilog folder was submitted along with the final project.

3.2 Inputs and Outputs

The input and output signals of the ENV block are:

- `CLK_CI`: Master clock signal.
- `RST_RBI`: Asynchronous reset signal; the module is reset if `RST_RBI=0`.
- `ENV_vol_DI`: Volume of the signal.
- `ENV_a_DI`: Time measured in seconds of the attack phase of the envelope.
- `ENV_d_DI`: Time measured in seconds of the decay phase of the envelope.
- `ENV_r_DI`: Time measured in seconds of the release phase of the envelope.
- `ENV_aY_DI`: Maximum amplitude of the attack phase of the envelope.
- `ENV_dY_DI`: Maximum amplitude of the decay phase of the envelope.

Table 2: Verilog code inputs, outputs, and parameters including fixed-point format and unit.

Verilog code name	Fixed-point format	Unit
`CLK_CI`	{1,0,'u'}	Clock input signal
`RST_RBI`	{1,0,'u'}	Reset input signal
`ENV_vol_DI`	{1,30,'s'}	Volume (unitless)
`ENV_a_DI`	{0,31,'s'}	Attack x-axis value (seconds)
`ENV_d_DI`	{0,31,'s'}	Decay x-axis value (seconds)
`ENV_r_DI`	{0,31,'s'}	Release x-axis value (seconds)
`ENV_aY_DI`	{31,0,'s'}	Attack y-axis value (unitless)
`ENV_dY_DI`	{0,31,'s'}	Decay y-axis value (unitless)
`ENV_rY_DI`	{0,31,'s'}	Release y-axis value (unitless)
`ENV_inc_DI`	{6,25,'s'}	Increment (seconds)
`ENV_recip_a_DI`	{7,24,'s'}	Reciprocal of ENV_a_DI (seconds^{-1})
`ENV_recip_d_DI`	{7,24,'s'}	Reciprocal of ENV_d_DI (seconds^{-1})
`ENV_recip_r_DI`	{7,24,'s'}	Reciprocal of ENV_r_DI (seconds^{-1})

- `ENV_rY_DI`: Maximum amplitude of the release phase of the envelope.
- `ENV_inc_DI`: Increment added to position every clock cycle.
- `ENV_recip_a_DI`: Reciprocal of the time of the attack phase of the envelope.
- `ENV_recip_d_DI`: Reciprocal of the time of the decay phase of the envelope.
- `ENV_recip_r_DI`: Reciprocal of the time of the release phase of the envelope.
- `Out_DO`: Output sample generated by the ENV block.

The names of the parameters in the Verilog code match the names in the MATLAB Golden Model code, so no mapping is needed. The assumption made in this code is that no parameters should be greater than 1. Additionally, the timing relationship of the signals obtained with ModelSim [1] when the parameters are being written to memory can be found on Figure 3, and the timing relationship of the signals at a different time can be found on Figure 4.

3.3 Block Diagrams and Functional Description

COMP_i

The purpose of this block is to check whether a key is currently being pressed, which will help determine the current state (A, S, D, or R), as well as whether we increment position (ENV_mux_j_D) by ENV_inc_D or increment position by 0. This comparator takes two inputs - ENV_Key_DI (1 if a key is pressed and 0 if a key is not pressed) and the constant 0 - and checks whether their values are equal. All COMP blocks that we use have three corresponding 1-bit outputs ('greater', 'less', and 'equal') that are generated as follows:

if the first input is greater than the second: 'greater' is 1, 'less' is 0, and 'equal' is 0

if the first input is less than the second: 'greater' is 0, 'less' is 1, and 'equal' is 0

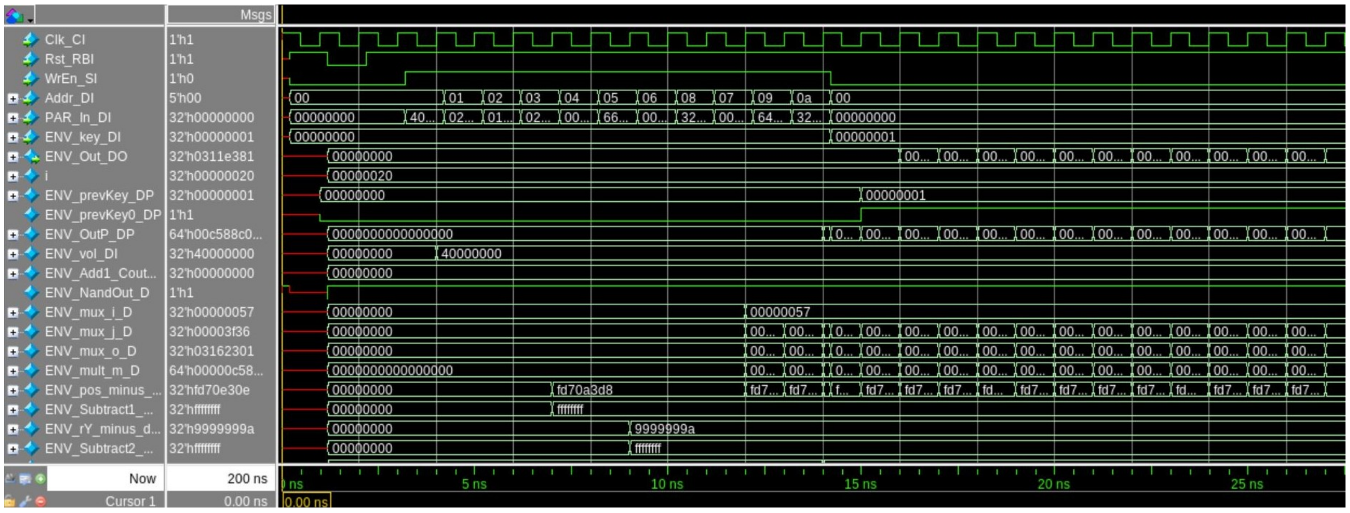


Figure 3: Timing diagram when parameters are being written to memory.

if the inputs are equal: 'greater' is 0, 'less' is 0, and 'equal' is 1.

The 'equal' output of this block gets fed into one of the inputs of our NAND gate such that our output is 1 if ENV_Key_DI is 0 (a key has not been pressed), and 0 if ENV_Key_DI is 1 (a key has been pressed).

COMP_j

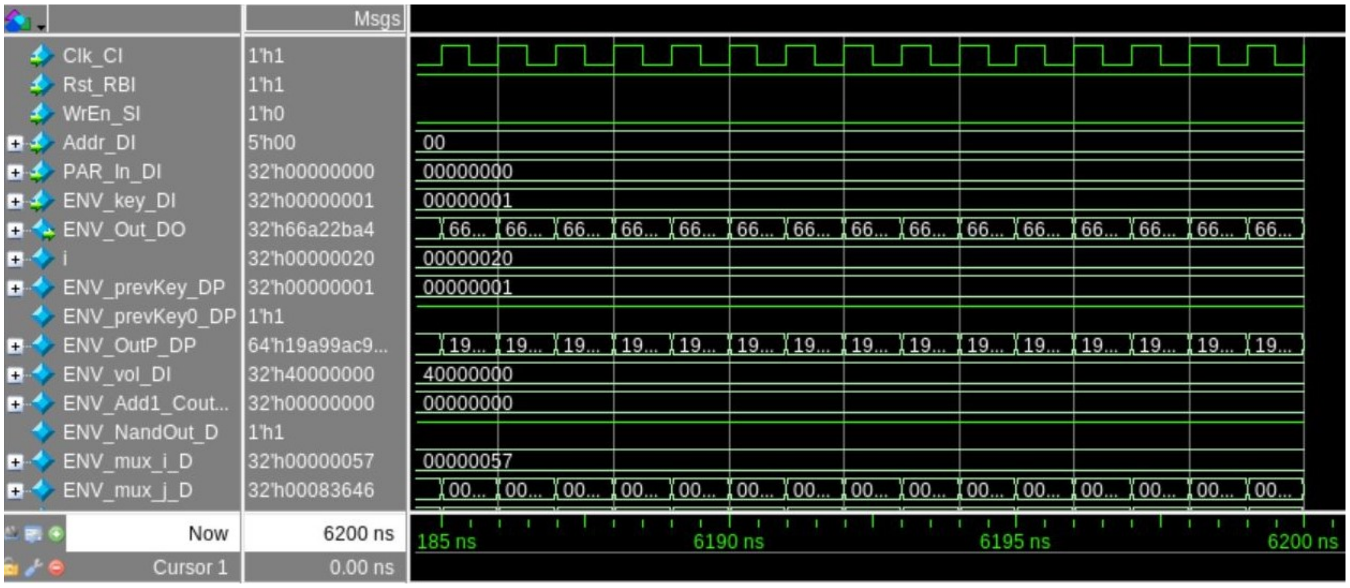
The purpose of this block is to help check whether we are in the release (R) state, which will help determine whether we increment position by ENV_inc_D or increment position by 0, as well as the amount by which to scale the overall output to the ENV block. This comparator takes two inputs - ENV_mux_j_D (the current position in seconds) from FF_i and ENV_r_DI (the R state's time interval in seconds) - and checks whether position is greater than ENV_r_DI. The 'greater' output of this block gets fed into one of the inputs of our NAND gate such that our output is 1 if position is greater than ENV_r_DI (the position is not in R) and 0 if position is less than or equal to ENV_r_DI (the position is inside R assuming no key is currently pressed).

NAND

The purpose of this block is to combine the outputs from COMP_i and COMP_j to help determine which state we are in and thus the amount by which to increment position. This NAND block takes two inputs - the output from COMP_i and the output from COMP_j. The output of this block gets fed into the selector of MUX2_i which selects with which value we increment position. The output is 0 which tells MUX2_i to increment position by 0 if no key is pressed and we are not inside R, i.e. when a note has already been pressed, sustained, and fully released, and no new key has yet been pressed to signal the start of the attack state, in essence an idle state. Otherwise, the output is 1 and we increment position with ENV_inc_D.

MUX2_i

The purpose of this block is to help select by which amount position will increment. This multiplexer takes three inputs: the constant 0, ENV_inc_D, and the output from the NAND block as the selector. The output of the multiplexer is fed into Add_i and is 0 if the output from the NAND gate is 0, i.e. if no key is currently pressed and we are not in the release state, in which case we do not want to increment our position. The output of the multiplexer is ENV_inv_D if the output from the NAND gate is 1, in which case



```
                     Msgs
Clk_CI               1'h1
Rst_RBI              1'h1
WrEn_SI              1'h0
Addr_DI              5'h00
PAR_In_DI            32'h00000000
ENV_key_DI           32'h00000001
ENV_Out_DO           32'h66a22ba4
i                    32'h00000020
ENV_prevKey_DP       32'h00000001
ENV_prevKey0_DP      1'h1
ENV_OutP_DP          64'h19a99ac9...
ENV_vol_DI           32'h40000000
ENV_Add1_Cout...     32'h00000000
ENV_NandOut_D        1'h1
ENV_mux_i_D          32'h00000057
ENV_mux_j_D          32'h00083646
Now                  6200 ns
Cursor 1             0.00 ns
```

Figure 4: Timing diagram of ENV signals.

we will increment position by ENV_inc_D.

Add_i
The purpose of this block is to add a chosen value (either 0 or ENV_inc_D) to the most current position value. This adder takes two inputs: the output from MUX2_i and the output from FF_i (ENV_old_pos_D), adds them, and feeds the output to one of the inputs of MUX2_j.

MUX2_j
The purpose of this block is to update the position value based on the logic from the above blocks which corresponds to which state of the envelope that we are in. This multiplexer takes three inputs: the output from Add_i, the constant 0, and the output from XOR_i as the selector. If the selector is 1 (the state of the key press is different than the state of the key press in the previous clock cycle), then the output of this block is 0. Otherwise, the output of this block is the output of Add_i. The output of this block gets fed into FF_i, such that the position value will either be overwritten/reset to 0, or it will be overwritten with the output of Add_i.

FF_i
The purpose of this block is to store our position variable (ENV_mux_j_D) which we use to help determine which state we are in in order to produce the appropriate output for the ENV block. The input of this flip-flop is the output of MUX2_j, and the output of this block (ENV_old_pos_D) gets fed into one of the inputs of Add_i and one of the inputs of COMP_j. Depending on what state the envelope is currently in, we need to increment position accordingly: when a key is initially pressed, we want to first reset our position to 0, once we are in state A and going through state D and S, we want position to be updated with position plus increment, as soon as the key is released and we are in state R, we want to reset the position to 0 and then start incrementing position by ENV_inc_D, and if we are no state such that no key has been pressed and the R state is not ongoing, we do not want to increment position.

Envelope Block Diagram

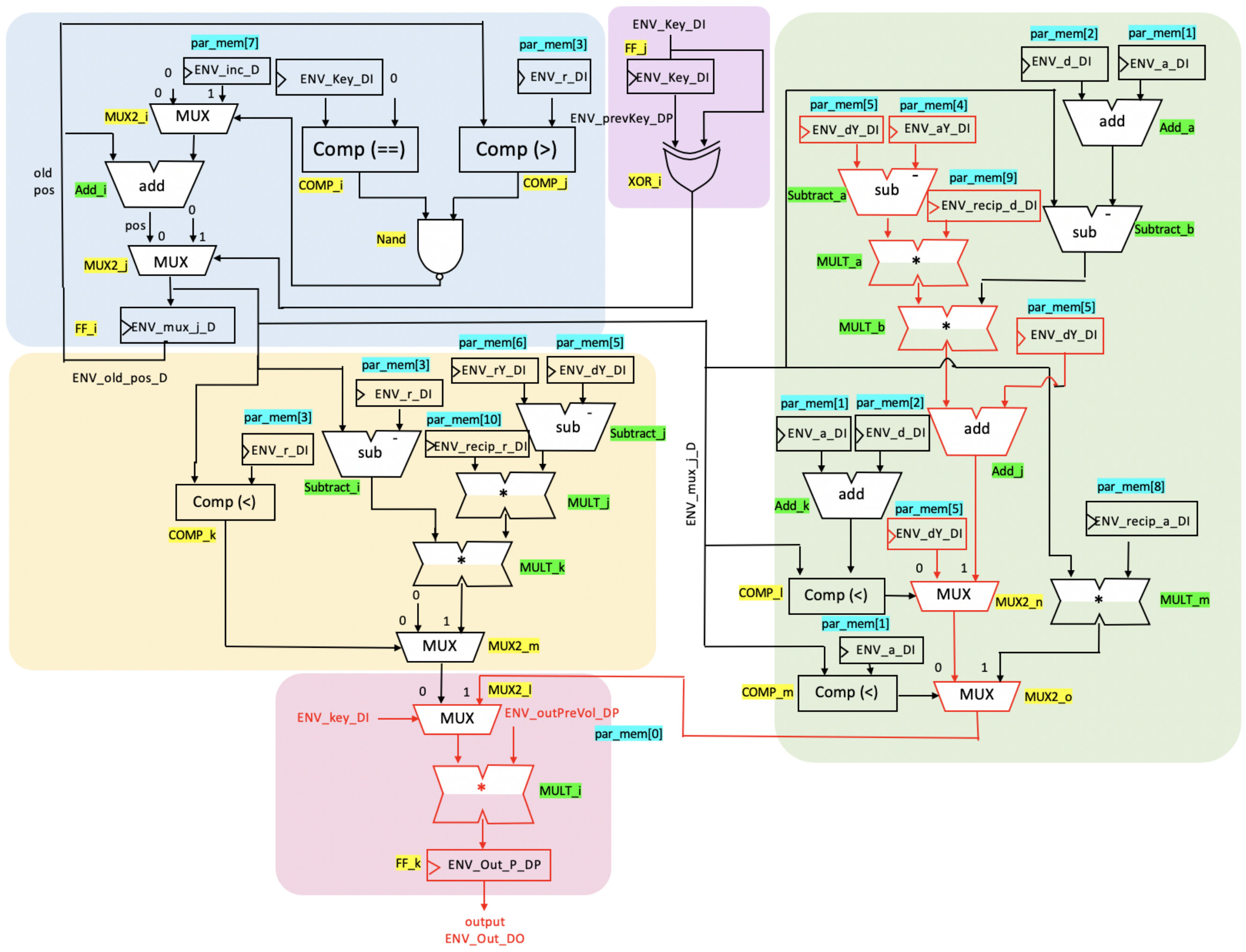


Figure 5: ENV Block Diagram

FF_j

The purpose of this block is to store ENV_Key_DI which is 1 if a key is currently being pressed and 0 otherwise - this information will be used to help determine in what state the envelope is in. The input of this flip-flop is ENV_Key_DI and the output is ENV_prevKey_DP, which gets fed into one of the inputs of XOR_i.

XOR_i

the purpose of this block is to compare the state of key press. The block takes two inputs: ENV_Key_DI and ENV_prevKey_DP, outputting their XOR operation. If a key is currently pressed and a key was not pressed in the previous clock cycle, the output will be 1 - this corresponds to the beginning of the A state. If a key is not currently pressed and a key was pressed in the previous cycle, the output will be 1 - this corresponds to the beginning of the R state. If a key is currently pressed and a key was also pressed in the previous cycle, the output will be 0 - this corresponds to any time within any of the A, S, D, or R states. If a key is not currently pressed and a key was not pressed in the previous clock cycle, the output will be 0 - this corresponds to an idle state. The output of this block gets fed into the selector of MUX2_j.

COMP_k
The purpose of this block is to help determine which state we are in for determining the amount by which to scale the overall output to the ENV block. This comparator takes two inputs - ENV_mux_j_D from FF_i and ENV_r_DI - and checks whether position is less than ENV_r_DI. The 'less' output of this block gets fed into the selector of MUX2_m such that the output is 1 if position is less than ENV_r_DI (the position is in R) and 0 if position is greater than or equal to ENV_r_DI (the position is inside R). This selector determines whether the output will be scaled according to the R state.

Subtract_i
The purpose of this block is to subtract the R-position value from the current position value which will be used to determine how far within the R state we currently are in order to scale the volume output accordingly. This subtracter takes two inputs: the output from MUX2_j (ENV_mux_j_D i.e. current position) and ENV_r_DI. The block subtracts ENV_r_DI from ENV_mux_j_D and outputs the result to one of the inputs of MULT_k.

Subtract_j
The purpose of this block is to help determine the slope of the R state's line equation by calculating the change in y-value from the beginning of the R state to the end of the R state. This subtracter takes two inputs: ENV_rY_DI, the y-position (amplitude) of the final point in the R stage, and ENV_dY_DI,the y-position (amplitude) of the final point in the D stage. The block subtracts ENV_dY_DI from ENV_rY_DI and outputs the result to one of the inputs of MULT_j.

MULT_j
The purpose of this block is to calculate the slope of the R state which will be used to appropriately scale the final output of the ENV block. This multiplier takes two inputs: ENV_recip_r_DI (the reciprocal of ENV_r_DI) and the output from Subtract_j (the change in y-value). The block multiplies the two inputs - i.e. multiplies the change in the y-coordinates with the reciprocal of the change in x-coordinates - to determine the slope and outputs the result to one of the inputs of MULT_k.

MULT_k
The purpose of this block is to calculate the current amplitude within the R state given the slope and current position. This multiplier takes two inputs: the output from Subtract_i (the current x-axis position within the R state's line equation) and the output from MULT_j (the slope of the R state's line equation). The block multiplies the two inputs to determine the current amplitude and outputs the result to one of the inputs of MUX2_m.

MUX2_m
The purpose of this block is to select which amplitude value gets selected as the intended current amplitude based on whether we are in state R or in an idle state. This multiplexer takes three inputs: the constant 0, the output from MULT_k, and the output from COMP_k as its selector. If the selector is 0 (we are currently in an idle state), then the output of this block is 0. Otherwise, if the selector is 1 (we are currently in the R state), the output of this block is the output of MULT_k, which is the current amplitude within the R state. The output of this block gets fed into MUX2_l.

Add_a

The purpose of this block is to add the D-state x-axis position interval to the A-state x-axis position interval which will be used to to help determine where within the D-state the current position is. This adder takes two inputs: ENV_d_DI and ENV_a_DI. This block adds the two inputs and sends the output to one of the inputs of Subtract_b.

Subtract_b
The purpose of this block is to subtract the position corresponding to the end of the D-state from the current position value to determine where within the D-state the current position is. This subtracter takes two inputs: ENV_mux_j_D and the output from Add_a. This block subtracts the two inputs and sends the output to one of the inputs of MULT_b.

Subtract_a
The purpose of this block is to help determine the slope of the D state's line equation by calculating the change in y-value from the beginning of the D state to the end of the D state. This subtracter takes two inputs: ENV_dY_DI and ENV_aY_DI. This block subtracts the two inputs and sends the output to one of the inputs of MULT_a.

MULT_a
The purpose of this block is to calculate the slope of the D state which will be used to appropriately scale the final output of the ENV block. This multiplier takes two inputs: the output from Subtract_a (the change in y-value) and ENV_recip_d_DI (the reciprocal of ENV_d_DI). The block multiplies the two inputs - i.e. multiplies the change in the y-coordinates with the reciprocal of the change in x-coordinates - to determine the slope and outputs the result to one of the inputs of MULT_b.

MULT_b
The purpose of this block is to help calculate the current amplitude within the D state given the slope and current position. This multiplier takes two inputs: the output from MULT_a (the slope of the D state's line equation) and the output from Subtract_b (the current x-axis position within the D state's line equation). The block multiplies the two inputs to determine the current amplitude and outputs the result to one of the inputs of Add_j.

Add_j
The purpose of this block is to finish calculating the current amplitude within the D state. This adder takes two inputs: the output from MULT_b and ENV_dY_DI (the y-position i.e. amplitude of the final point in the D state). The block adds the two inputs, shifting the output from MULT_b up by ENV_dY_DI following the equation of a line, to determine the current amplitude and sends the output to one of the inputs of MUX2_n.

Add_k
The purpose of this block is to help determine whether we are within the D state. This adder takes two inputs: ENV_a_DI and ENV_d_DI. The block adds the two inputs to determine the final x-position corresponding to the end of the D state, and sends the output to one of the inputs of COMP_l.

COMP_l
The purpose of this block is to help determine whether we are in state S or state D. This comparator takes two inputs - ENV_mux_j_D from FF_i and the output from Add_k - and checks whether position is less than the output from Add_k (ENV_a_DI + ENV_d_DI). The 'less' output of this block gets fed into the selector of

MUX2_n such that the output is 1 if position is less than the output from Add_k (the position is within state D) and 0 if position is greater than or equal to the output from Add_k (the position is not within state D). This selector determines whether the output will be scaled according to the D state or the S state.

MUX2_n
The purpose of this block is to select which amplitude value gets selected as the intended current amplitude based on whether we are in state R or in an idle state. This multiplexer takes three inputs: ENV_dY_DI, the output from Add_j, and the output from COMP_l as its selector. If the selector is 0 (we are currently in the S state), then the output of this block is ENV_dY_DI, the constant amplitude for the S state. Otherwise, if the selector is 1 (we are currently in the D state), the output of this block is the output of Add_j, the current amplitude within the R state. The output of this block gets fed into MUX2_o.

MULT_m
The purpose of this block is to calculate the current amplitude within the A state given the slope and current position. This multiplier takes two inputs: ENV_mux_j_D (the current position) and ENV_recip_a_DI (the reciprocal of ENV_a_DI, i.e. the slope of the A state). The block multiplies the two inputs to determine the current amplitude and outputs the result to one of the inputs of MUX2_o.

COMP_m
The purpose of this block is to help determine whether we are in state A. This comparator takes two inputs - ENV_mux_j_D from FF_i and ENV_A_DI. The 'less' output of this block gets fed into the selector of MUX2_o such that the output is 1 if position is less than the ENV_a_DI (the position is within state A) and 0 if position is greater than or equal ENV_a_DI (the position is not within state A and is in either state S or D). This selector determines whether the output will be scaled according to the A state or either state S or D (MUX2_n determines whether we are in state S or D).

MUX2_o
The purpose of this block is to select which amplitude value gets selected as the intended current amplitude based on whether we are in state A or either of state S or D. This multiplexer takes three inputs: the output from MUX2_n (the amplitude corresponding to either state S or D based on its selector), the output from MULT_m (the amplitude corresponding to state A), and the output from COMP_m as its selector. If the selector is 0 (we are currently in state S or D), then the output of this block is the output from MUX2_n. Otherwise, if the selector is 1 (we are currently in state A), the output of this block is the output of MULT_m, the current amplitude within the R state. The output of this block gets fed into MUX2_l.

MUX2_l
The purpose of this block is to select which amplitude value gets selected as the intended current amplitude based on which state the previous multiplexers have determined we are in. This multiplexer takes three inputs: the output from MUX2_m (the amplitude corresponding to either state R or an idle state), the output from MUX2_0 (the amplitude corresponding to state A, S, or D), and ENV_key_DI as its selector. If the selector is 0 (no key is pressed so we are currently in state R or an idle state), then the output of this block is the output from MUX2_m. Otherwise, if the selector is 1 (a key is pressed so we are currently in state A, S, or D), the output of this block is the output of MUX2_o. The output of this block which represents the amplitude based on the correct state we have determined we are in gets fed into MULT_i to then be scaled based on volume.

MULT_i

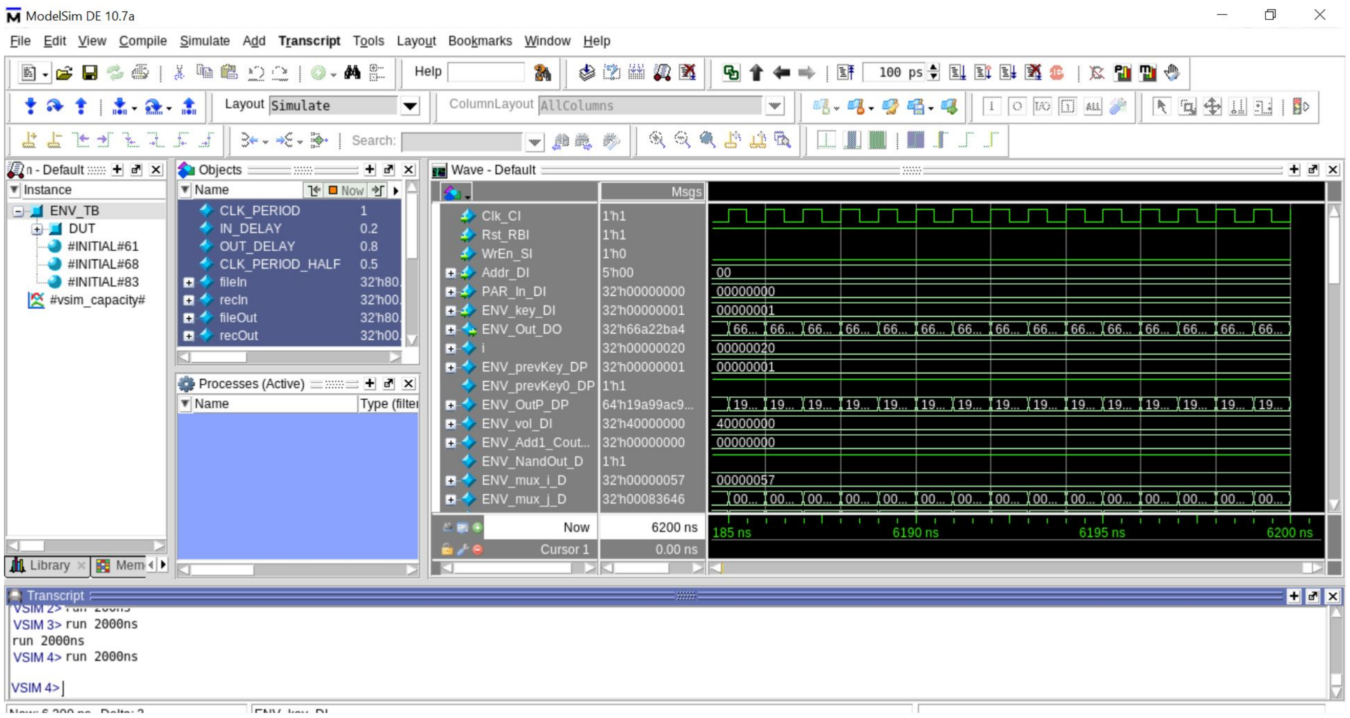
The purpose of this block is to scale the current amplitude with the current volume to output the correct final amplitude of the ENV block. This multiplier takes two inputs: the output from MUX2_l (the un-scaled amplitude) and ENV_outPreVol_DP (the current volume). The block multiplies the two inputs to determine the current scaled amplitude and outputs the result to FF_k.

FF_k

The purpose of this block is to store ENV_OUT_P_DP which is the final amplitude and output of the ENV block. The input of this flip-flop is the output from MULT_i and the output is ENV_Out_DO, which gets fed into the AMP block.

3.4 Verification

We approached the verification by testing through the different modes that our block goes through: attack, decay, sustain, and release. We were running into issues with the attack mode because the values suddenly turn negative. We think there is an issue regarding the signed bit and how we are truncating for the output. This is the only test vector we have tested (a = 0.02, d = 0.01, r = 0.02), but if the values for the different parameters (a,d,r) are below 1, then any combination of values should work once the flipping from 0 to 1 of the signed bit is resolved for the attack mode at least. The Verilog currently does not match the MATLAB code's expected output.


ModelSim DE 10.7a
File Edit View Compile Simulate Add Transcript Tools Layout Bookmarks Window Help
Layout Simulate
ColumnLayout AllColumns
Search:
Wave - Default
Msgs

```
Instance
ENV_TB
  DUT
  #INITIAL#61
  #INITIAL#68
  #INITIAL#83
#vsim_capacity#
```

```
Objects
Name
CLK_PERIOD        1
IN_DELAY          0.2
OUT_DELAY         0.8
CLK_PERIOD_HALF   0.5
fileIn            32'h80.
recIn             32'h00.
fileOut           32'h80.
recOut            32'h00.
```

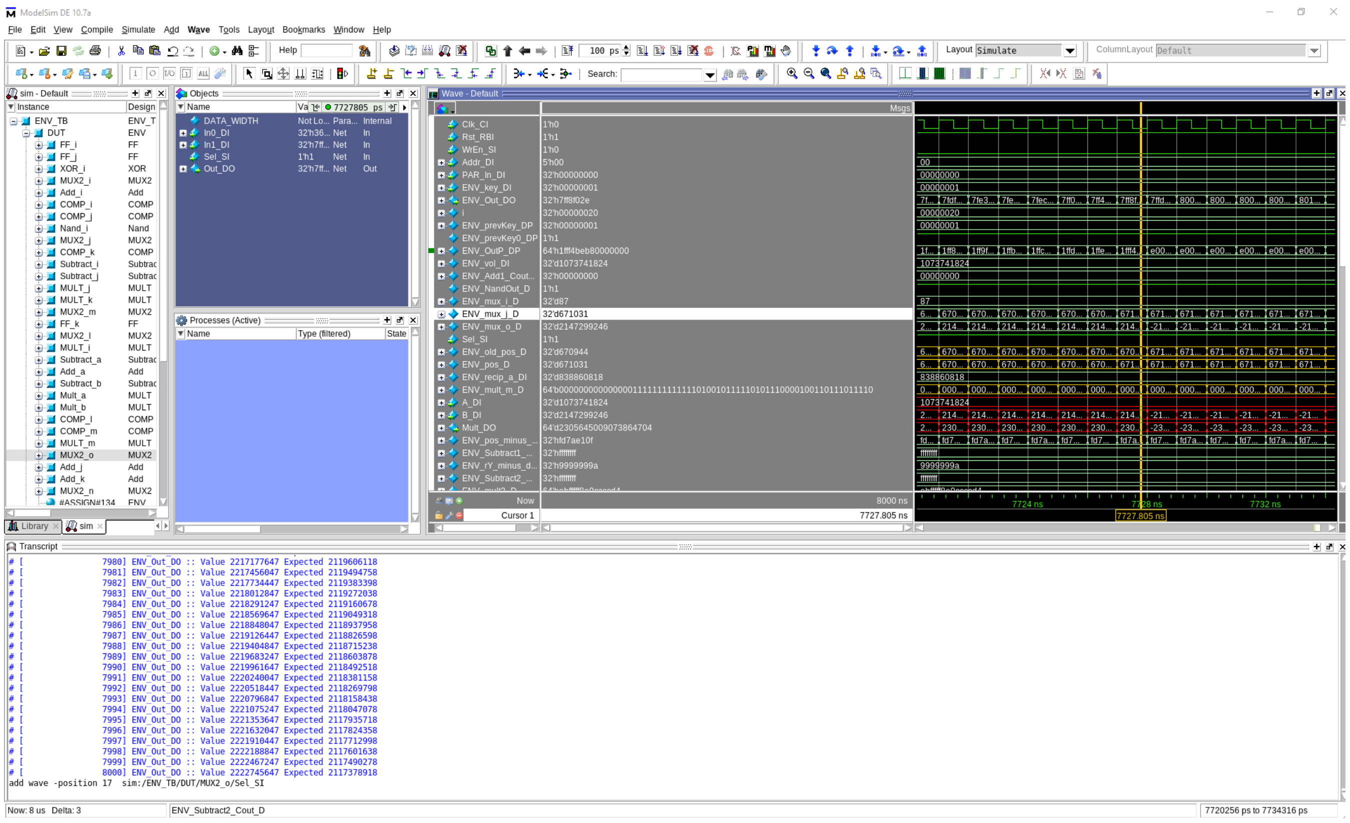
```
Clk_CI            1'h1
Rst_RBI           1'h1
WrEn_SI           1'h0
Addr_DI           5'h00
PAR_In_DI         32'h00000000
ENV_key_DI        32'h00000001
ENV_Out_DO        32'h66a22ba4
i                 32'h00000020
ENV_prevKey_DP    32'h00000001
ENV_prevKey0_DP   1'h1
ENV_OutP_DP       64'h19a99ac9...
ENV_vol_DI        32'h40000000
ENV_Add1_Cout...  32'h00000000
ENV_NandOut_D     1'h1
ENV_mux_i_D       32'h00000057
ENV_mux_j_D       32'h00083646
Now               6200 ns
Cursor 1          0.00 ns
```

```
VSIM 3> run 2000ns
run 2000ns
VSIM 4> run 2000ns

VSIM 4>
```

Figure 6: Running ModelSim

- *Loading Correct Parameters* makes sure that we can write the values of our 11 parameters correctly into the test bench. We can into some issues with making this compatible with MATLAB at first



	Msgs
Clk_CI	1'h0
Rst_RBI	1'h1
WrEn_SI	1'h0
Addr_DI	5'h00
PAR_In_DI	32'h00000000
ENV_key_DI	32'h00000001
ENV_Out_DO	32'h7ff8f02e
i	32'h00000020
ENV_prevKey_DP	32'h00000001
ENV_prevKey0_DP	1'h1
ENV_OutP_DP	64'h1fff4beb80000000
ENV_vol_DI	32'd1073741824
ENV_Add1_Cout...	32'h00000000
ENV_NandOut_D	1'h1
ENV_mux_i_D	32'd87
ENV_mux_j_D	32'd671031
ENV_mux_o_D	32'd2147299246
Sel_SI	1'h1
ENV_old_pos_D	32'd670944
ENV_pos_D	32'd671031
ENV_recip_a_DI	32'd838860818
ENV_mult_m_D	64'b0000000000000000111111111111010010111110101110000100110111011110
A_DI	32'd1073741824
B_DI	32'd2147299246
Mult_DO	64'd2305645009073864704
ENV_pos_minus_...	32'hfd7ae10f
ENV_Subtract1_...	32'hffffffff
ENV_rY_minus_d...	32'h9999999a
ENV_Subtract2_...	32'hffffffff
Now	8000 ns
Cursor 1	7727.805 ns

```
# [                 7980] ENV_Out_DO :: Value 2217177647 Expected 2119606118
# [                 7981] ENV_Out_DO :: Value 2217456047 Expected 2119494758
# [                 7982] ENV_Out_DO :: Value 2217734447 Expected 2119383398
# [                 7983] ENV_Out_DO :: Value 2218012847 Expected 2119272038
# [                 7984] ENV_Out_DO :: Value 2218291247 Expected 2119160678
# [                 7985] ENV_Out_DO :: Value 2218569647 Expected 2119049318
# [                 7986] ENV_Out_DO :: Value 2218848047 Expected 2118937958
# [                 7987] ENV_Out_DO :: Value 2219126447 Expected 2118826598
# [                 7988] ENV_Out_DO :: Value 2219404847 Expected 2118715238
# [                 7989] ENV_Out_DO :: Value 2219683247 Expected 2118603878
# [                 7990] ENV_Out_DO :: Value 2219961647 Expected 2118492518
# [                 7991] ENV_Out_DO :: Value 2220240047 Expected 2118381158
# [                 7992] ENV_Out_DO :: Value 2220518447 Expected 2118269798
# [                 7993] ENV_Out_DO :: Value 2220796847 Expected 2118158438
# [                 7994] ENV_Out_DO :: Value 2221075247 Expected 2118047078
# [                 7995] ENV_Out_DO :: Value 2221353647 Expected 2117935718
# [                 7996] ENV_Out_DO :: Value 2221632047 Expected 2117824358
# [                 7997] ENV_Out_DO :: Value 2221910447 Expected 2117712998
# [                 7998] ENV_Out_DO :: Value 2222188847 Expected 2117601638
# [                 7999] ENV_Out_DO :: Value 2222467247 Expected 2117490278
# [                 8000] ENV_Out_DO :: Value 2222745647 Expected 2117378918
add wave -position 17  sim:/ENV_TB/DUT/MUX2_o/Sel_SI
```

Now: 8 us Delta: 3   ENV_Subtract2_Cout_D   7720256 ps to 7734316 ps

Figure 7: Where our simulation fails (7728 ns)

Table 3: Summary of Verification Tests.

Test	Brief description	# vectors
Loading Correct Parameters	Load parameters that match with MATLAB	5
Attack Mode	Confirming correct mode behavior	1

because of different fixed-point formatting and because MATLAB calculated some of our values differently than expected. We put in 5 different test vectors, changing the a, d, r, aY, and dY values.

- *Attack Mode* makes sure that we are outputting the correct values compared to the expected output in MATLAB. We can the MATLAB code to run for only 0.1s which is in the attack mode so that we were isolating this region to compare to the output. We were struggling with this verification test because of some fixed-point truncation issues. We can the simulation for 1 test case and 10000 ns, but ran into the issue mentioned above.

3.5 Synthesis Commands

Yes, the synthesis commands were the ones that had been provided with the syndc tutorial from our discussion class. For compile_ultra, we followed the noautoungroup command. The script is in ENV_block_template/ENV/Synopsis_Script. The script for 40 ns can be found below. The second part of the script to set a different clock period and generate the reports was run for each value in the range 10 to 120 ns in jumps of 10 ns.

```
analyze -library work -format verilog ../src/FF.v
analyze -library work -format verilog ../src/Add.v
analyze -library work -format verilog ../src/COMP.v
analyze -library work -format verilog ../src/MUX2.v
analyze -library work -format verilog ../src/MULT.v
analyze -library work -format verilog ../src/Subtract.v
analyze -library work -format verilog ../src/Nand.v
analyze -library work -format verilog ../src/ENV.v
elaborate ENV -library work

create_clock Clk_CI -period 10.0
set_clock_transition 0.2 [get_clocks Clk_CI]
set_input_delay 0.2 -clock Clk_CI [remove_from_collection [all_inputs] [get_ports Clk_CI]]
set_driving_cell -library saed90nm_typ -lib_cell INVX4 -pin ZN [all_inputs]
set_load 0.01 [all_outputs]
compile_ultra -no_autoungroup check_design
report_timing -max_paths 5 >./reports/timing_max_40ns.rpt
report_timing -delay min -max_paths 5 >./reports/timing_min_10ns.rpt
report_area -hierarchy >./reports/area_40ns.rpt
```

3.6 State of Verilog Code

The Verilog code matches the MATLAB code through the attack mode until 7728 ns. Then, there is a sign change which causes a mismatch in the output values. We ran into some issues with establishing the values for our parameters being loaded in and there being a different with how MATLAB calculated certain values, but we resolved those and have everthing correct in that regard. However, as of now, because of this mismatch caused by the signed bit, it does not 100% match our MATLAB code.

4 Synthesis Results

For this part of the report, the design was synthesized using Synopsis Design Compiler (Synopsis DC) [2] and sweeping through the period (T) range T = 10 ns to T = 120 ns in jumps of 10 ns. The resulting synthesis reports obtained with Synopsis DC together with the `dc_syn.log` file were included in the zip-folder that contains this report for the final submission.

The period for the smallest AT product for which there are no hold violations was found to be T = 40 ns. This value was then used to generate the final gate-level design.

4.1 Synchronous Design

The design includes three flip-flops to store the position, the key value and the output sample. The three flip-flops are created from the FF.v template and are therefore positive-edge and with asynchronous reset. A screenshot of the elaborate results can be found on Figure 8.

The AT product for each combination was calculated and can be found on Table 4. The smallest AT value for which the hold conditions were not violated is for A = 224068.72 and T = 40 ns. Thus, T = 40 ns was used to generate the final gate-level Verilog code to place and route.

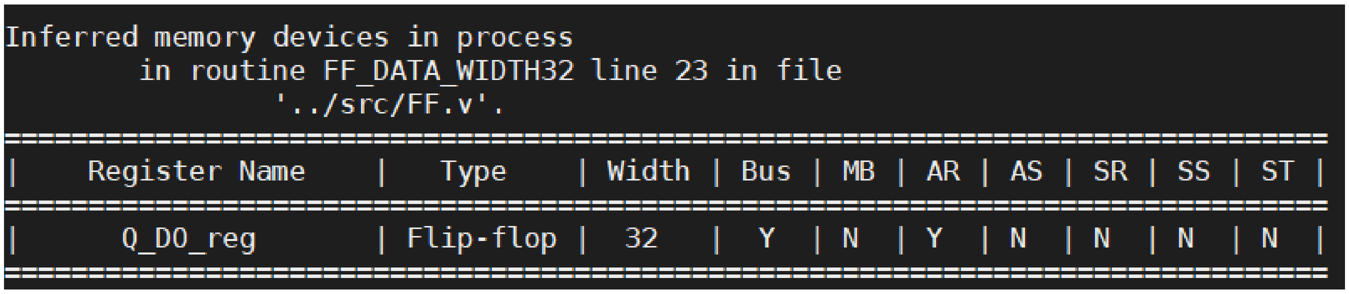
4.2 AT-Tradeoff Plot

The Area-Delay (AT) diagram was created using the values from the reports output by Synopsis DC. The design was synthesized sweeping through the period (T) range T = 10 ns to T = 120 ns in jumps of 10 ns. The tool used to generate the graph was MATLAB, and the code can be found together with the final submission. The AT product was calculated from the total area (A) and the delay (T), which was calculated as the clock period minus the slack. The final AT plot can be found on Figure 9.

4.3 Area, Delay, and Power Results

For the selected period T = 40 ns, Table 5 shows the area, delay and power results of the ENV block. The period of T = 40 ns the smallest value for which the hold conditions are not violated. Figure 10 provides a screenshot from the report generated in Synopsis DC that proves the slack was not violated.

Since the minimum clock period required for the block to run is T = 40 ns, the maximum frequency



```
Inferred memory devices in process
        in routine FF_DATA_WIDTH32 line 23 in file
                '../src/FF.v'.
===============================================================================
|    Register Name    |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
===============================================================================
|      Q_DO_reg       | Flip-flop |  32   |  Y  | N  | Y  | N  | N  | N  | N  |
===============================================================================
```

Figure 8: Synopsis DC elaborated Flip-Flop instance.

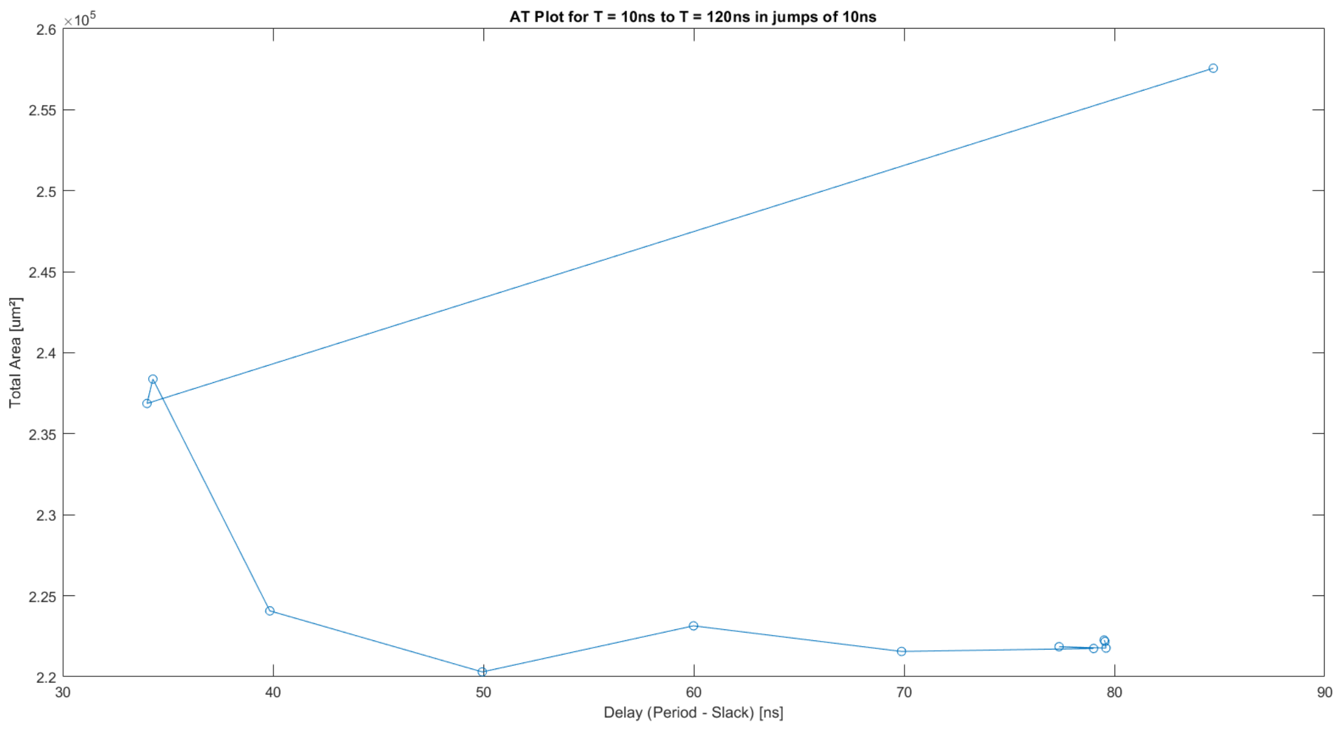
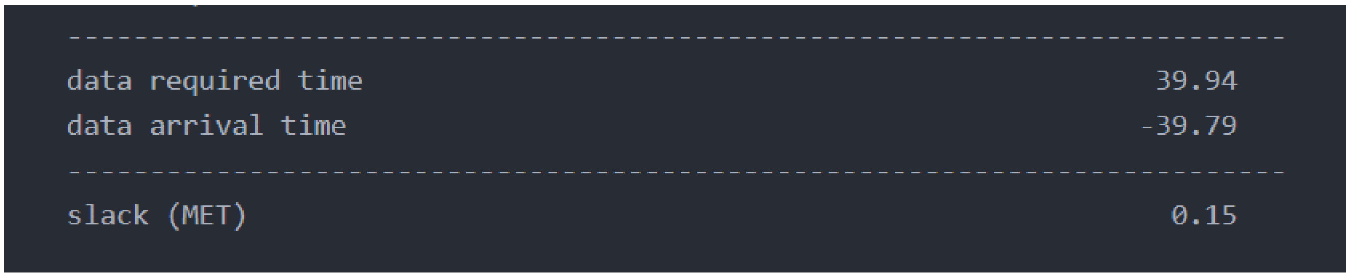


Figure 9: AT Diagram.

is given by $fmax = \frac{1}{T} = \frac{1}{40x10^{-9}} = 250MHz$. In addition, the critical path fort he design was found by using the maximum timing reports generated by Synopsis DC and highlighted in Figure 11.

4.4 State of Synthesis

From the results above we can conclude that the design is synthesizable. The minimum clock period required for the block to run is T = 40 ns, and the associated maximum frequency is 250 MHz. The total area of the block under these conditions is 201004.65 μm^2, and the block consumes 1370 mW.



```
---------------------------------------------------------------------
data required time                                          39.94
data arrival time                                          -39.79
---------------------------------------------------------------------
slack (MET)                                                  0.15
```

Figure 10: Maximum Timing for T = 40 ns Report Screenshot.

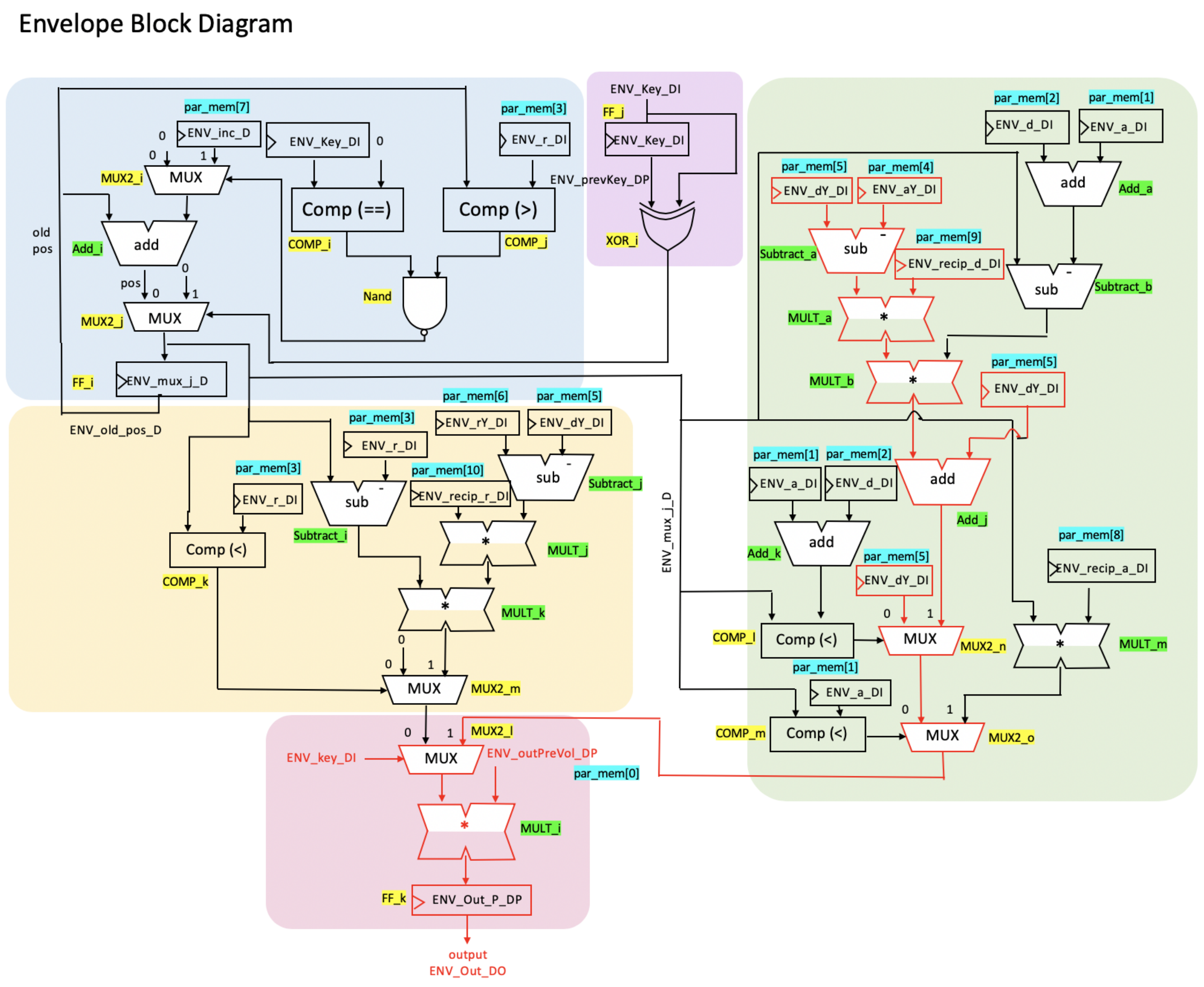


Figure 11: ENV Block Diagram with Synopsis Crtical Path in Red

Table 4: Verilog code inputs, outputs, and parameters including fixed-point format and unit.

Clock Period (ns)	Area A (μm^2)	Delay T (ns)	AT Product
10	257565.71	84.70	21816000
20	236872.46	34.01	8056000
30	238375.68	34.29	8173900
40	224068.72	39.85	8929100
50	220300.16	49.95	11004000
60	223145.56	59.99	13387000
70	221564.41	69.88	15483000
80	221746.81	79.01	17520000
90	221857.00	77.37	17165000
100	221772.91	79.60	17653000
110	222178.42	79.55	17674000
120	222269.74	79.51	17673000

Table 5: Summary of Synthesis with Synopsys DC.

Metric	Result
Total cell area	201004.65 μm^2
Total core area	224068.72 μm^2
Clock frequency	250 MHz
Power consumption	1370 mW

5 Backend: Place and Route

After place and route, the final design output had no DRC or hold violations and 0 violating paths. The design was placed and routed successfully, but the post-layout simulation does not work.

5.1 Floorplanning

Table 6: Summary of Floorplanning

Metric	Value
Number of Input Pins	72
Number of Output Pins	32 %
Area	650 x 650 with 14.08 margins μm^2
Pre-Placement Density	66.051 %

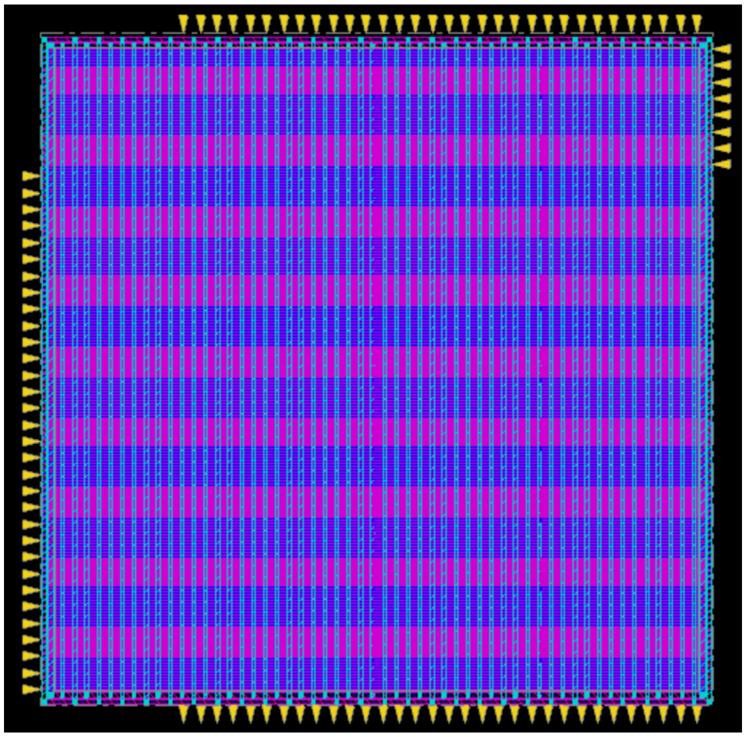
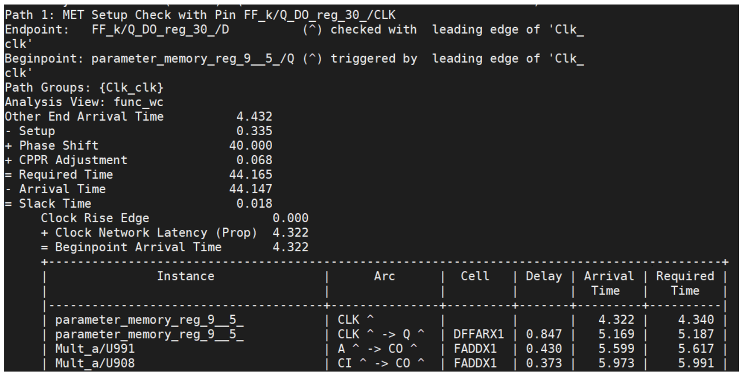


Figure 12: Floorplan Capture

5.2 Area, Delay, and Power Results

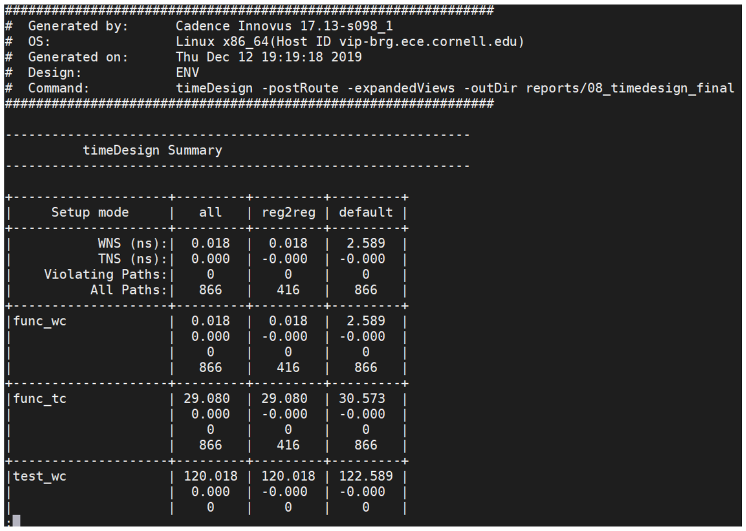
The clock period constraint was 40 ns and there were no timing violations post routing.



```
Path 1: MET Setup Check with Pin FF_k/Q_DO_reg_30_/CLK
Endpoint:   FF_k/Q_DO_reg_30_/D          (^) checked with  leading edge of 'Clk_
clk'
Beginpoint: parameter_memory_reg_9__5_/Q (^) triggered by  leading edge of 'Clk_
clk'
Path Groups: {Clk_clk}
Analysis View: func_wc
Other End Arrival Time          4.432
- Setup                         0.335
+ Phase Shift                  40.000
+ CPPR Adjustment               0.068
= Required Time                44.165
- Arrival Time                 44.147
= Slack Time                    0.018
     Clock Rise Edge                 0.000
     + Clock Network Latency (Prop)  4.322
     = Beginpoint Arrival Time       4.322
     +-----------------------------------------------------------------------------------------+
     |             Instance                |      Arc       |  Cell   | Delay | Arrival | Required |
     |                                     |                |         |       |  Time   |   Time   |
     |-------------------------------------+----------------+---------+-------+---------+----------|
     | parameter_memory_reg_9__5_          | CLK ^          |         |       |   4.322 |    4.340 |
     | parameter_memory_reg_9__5_          | CLK ^ -> Q ^   | DFFARX1 | 0.847 |   5.169 |    5.187 |
     | Mult_a/U991                         | A ^ -> CO ^    | FADDX1  | 0.430 |   5.599 |    5.617 |
     | Mult_a/U908                         | CI ^ -> CO ^   | FADDX1  | 0.373 |   5.973 |    5.991 |
```

Figure 13: Timing Met

There were no violating paths or hold/setup violations:



```
###############################################################
#  Generated by:      Cadence Innovus 17.13-s098_1
#  OS:                Linux x86_64(Host ID vip-brg.ece.cornell.edu)
#  Generated on:      Thu Dec 12 19:19:18 2019
#  Design:            ENV
#  Command:           timeDesign -postRoute -expandedViews -outDir reports/08_timedesign_final
###############################################################

------------------------------------------------------------
          timeDesign Summary
------------------------------------------------------------

+--------------------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg | default |
+--------------------+---------+---------+---------+
|           WNS (ns):|  0.018  |  0.018  |  2.589  |
|           TNS (ns):|  0.000  | -0.000  | -0.000  |
|    Violating Paths:|    0    |    0    |    0    |
|          All Paths:|   866   |   416   |   866   |
+--------------------+---------+---------+---------+
|func_wc             |  0.018  |  0.018  |  2.589  |
|                    |  0.000  | -0.000  | -0.000  |
|                    |    0    |    0    |    0    |
|                    |   866   |   416   |   866   |
+--------------------+---------+---------+---------+
|func_tc             | 29.080  | 29.080  | 30.573  |
|                    |  0.000  | -0.000  | -0.000  |
|                    |    0    |    0    |    0    |
|                    |   866   |   416   |   866   |
+--------------------+---------+---------+---------+
|test_wc             | 120.018 | 120.018 | 122.589 |
|                    |  0.000  | -0.000  | -0.000  |
|                    |    0    |    0    |    0    |
```

Figure 14: No Violating Paths

Critical Path:

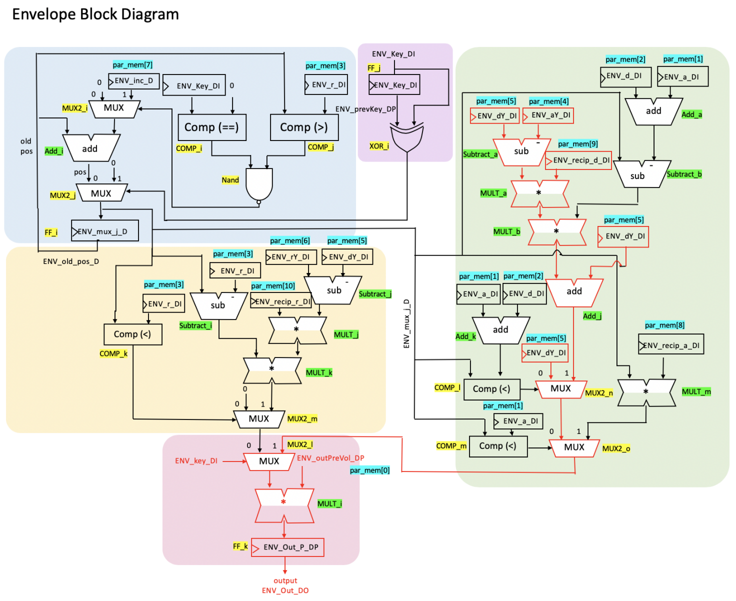


Figure 15: Innovus Critical Path

Table 7: Summary of Place and Route with Cadence Innovus.

Metric	Result
Total area	260174.1312 μm^2
Cell density	67.999 %
Delay (Clock Period - Slack)	39.982 ns
Clock frequency (1/T)	25 MHz
Power consumption	6.04 W

5.3 Backend Screenshot

Screenshot of your final chip:

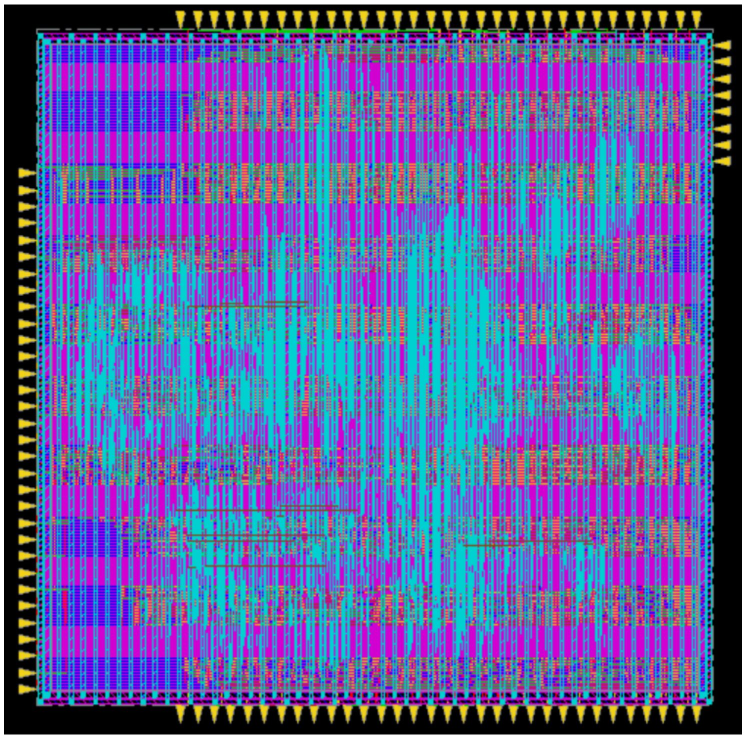


Figure 16: Post Routing Final Chip

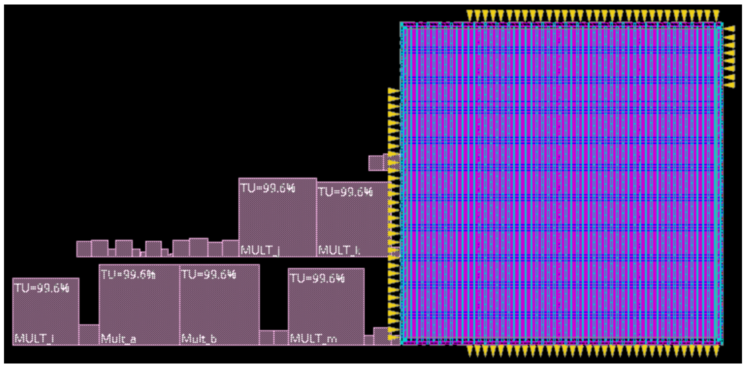


Figure 17: Post Routing Floorplan View.

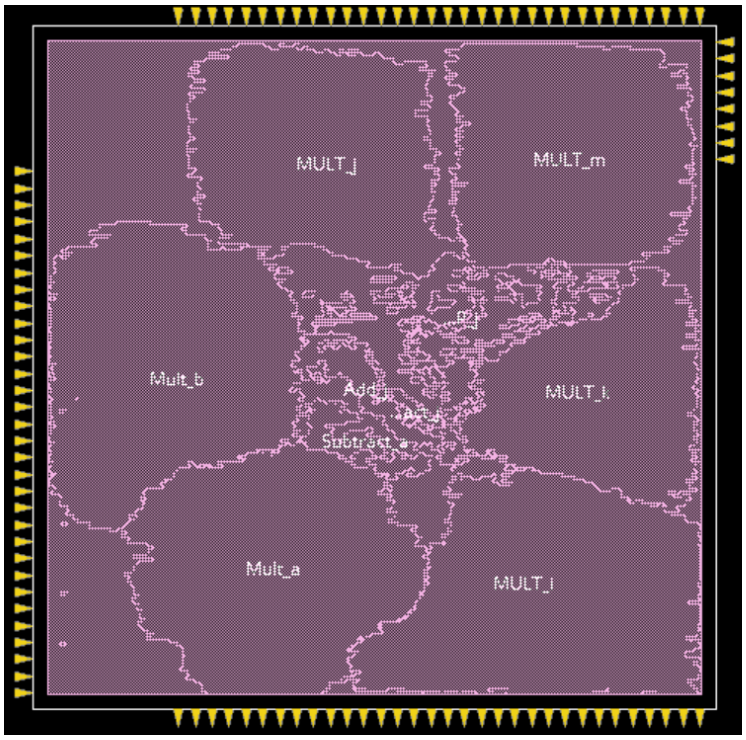
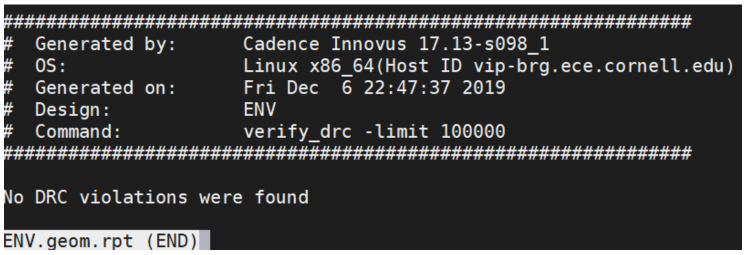


Figure 18: Post Routing Amoeba View.

5.4 Design Rule Check (DRC)

Our design has no DRC violations.



```
################################################################
#  Generated by:      Cadence Innovus 17.13-s098_1
#  OS:                Linux x86_64(Host ID vip-brg.ece.cornell.edu)
#  Generated on:      Fri Dec  6 22:47:37 2019
#  Design:            ENV
#  Command:           verify_drc -limit 100000
################################################################

No DRC violations were found

ENV.geom.rpt (END)
```

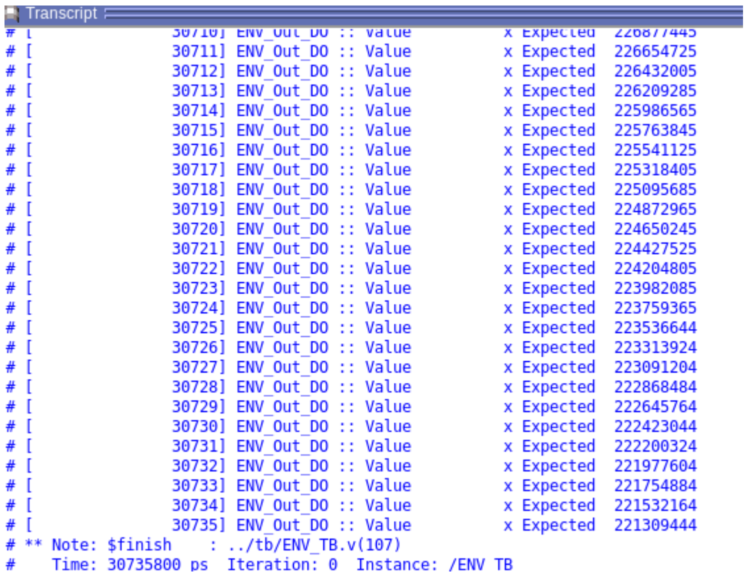
Figure 19: No DRC Violations.

5.5 Place and Route Commands

We followed the commands given in class. There were no modifications and decided not to make the commands into a script so that the reports could be checked at each checkpoint.

5.6 Post-Layout Simulation

The post-layout simulation does not work. For some reason the output is all undefined for our first test vector and did not have time to troubleshoot this. The main focus was on getting the MATLAB to match the Verilog so the MATLAB does not still match after place and route at this point.



```
# [                30710] ENV_Out_DO :: Value          x Expected  226877445
# [                30711] ENV_Out_DO :: Value          x Expected  226654725
# [                30712] ENV_Out_DO :: Value          x Expected  226432005
# [                30713] ENV_Out_DO :: Value          x Expected  226209285
# [                30714] ENV_Out_DO :: Value          x Expected  225986565
# [                30715] ENV_Out_DO :: Value          x Expected  225763845
# [                30716] ENV_Out_DO :: Value          x Expected  225541125
# [                30717] ENV_Out_DO :: Value          x Expected  225318405
# [                30718] ENV_Out_DO :: Value          x Expected  225095685
# [                30719] ENV_Out_DO :: Value          x Expected  224872965
# [                30720] ENV_Out_DO :: Value          x Expected  224650245
# [                30721] ENV_Out_DO :: Value          x Expected  224427525
# [                30722] ENV_Out_DO :: Value          x Expected  224204805
# [                30723] ENV_Out_DO :: Value          x Expected  223982085
# [                30724] ENV_Out_DO :: Value          x Expected  223759365
# [                30725] ENV_Out_DO :: Value          x Expected  223536644
# [                30726] ENV_Out_DO :: Value          x Expected  223313924
# [                30727] ENV_Out_DO :: Value          x Expected  223091204
# [                30728] ENV_Out_DO :: Value          x Expected  222868484
# [                30729] ENV_Out_DO :: Value          x Expected  222645764
# [                30730] ENV_Out_DO :: Value          x Expected  222423044
# [                30731] ENV_Out_DO :: Value          x Expected  222200324
# [                30732] ENV_Out_DO :: Value          x Expected  221977604
# [                30733] ENV_Out_DO :: Value          x Expected  221754884
# [                30734] ENV_Out_DO :: Value          x Expected  221532164
# [                30735] ENV_Out_DO :: Value          x Expected  221309444
# ** Note: $finish    : ../tb/ENV_TB.v(107)
#    Time: 30735800 ps  Iteration: 0  Instance: /ENV_TB
```

Figure 20: Post Layout ModelSim Run.

5.7 State of Place and Route

The design placed and routed successfully, but the post-layout simulation does not work. There are also a few hitches in the ModelSim simulations, so these are all bound to change.

References

[1] M. Graphics. (2019) Modelsim. ”https://www.mentor.com/company/higher_ed/modelsim-student-edition”[Accessed 12-16-2019].

[2] S. Inc. (2019) Synopsis design compiler. ”https://www.synopsis.com/”[Accessed 12-16-2019].

A Included Files in Zip-Folder

```
Group-2-ENV/
+- SynTechV4
+- ENV_block_template
+-- ENV
+-- tb
+-- Deliverable_11_Reports
+- par_reports
```

B Design Review Slides

The design review slides are provided in the zipped folder under the main Group-2-ENV folder.

C MATLAB Code

Below is the initializations of the parameters and states for our block:

```
% ==========================================================================
% ECE 5746 - Simple Amplitude Envelope Model (INITIALIZATION)
% (c) 2019 studer@cornell.edu
% ==========================================================================

function [par,sta] = ENV_init(par,sta)

% ENV parameters
%resizing
%FixP_out = {0,31,'s'}; % {I,F,'s'} where 's' is signed
FixP_out = {6,25,'s'};
FixP_par = {0,31,'s'};
FixP_recip = {7,24,'s'};
QType_out = 'WrpTrc_NoWarn';
% all state variables used by this block must be initialized
sta.ENV.pos = RealRESIZE(0, FixP_out,QType_out);
sta.ENV.inc = RealRESIZE(0,FixP_out,QType_out);
sta.ENV.prevKey = RealRESIZE(0,{31,0,'s'}, QType_out);

sta.ENV.Out_DO = RealRESIZE(0,FixP_par, QType_out); % ENV block output

par.ENV.a = RealRESIZE(0.02, FixP_par, QType_out);
par.ENV.d = RealRESIZE(0.01,FixP_par, QType_out);
par.ENV.r = RealRESIZE(0.02,FixP_par, QType_out);
par.ENV.aY = RealRESIZE(1,{31,0,'s'}, 'SatTrc_NoWarn');
par.ENV.dY = RealRESIZE(0.8,FixP_par, QType_out);
par.ENV.rY = RealRESIZE(0,FixP_par, QType_out);

par.ENV.aRec = RealRESIZE(1/par.ENV.a,FixP_recip, QType_out);
par.ENV.dRec = RealRESIZE(1/par.ENV.d,FixP_recip, QType_out);
par.ENV.rRec = RealRESIZE(1/par.ENV.r,FixP_recip, QType_out);

par.ENV.glofsi = RealRESIZE(1/par.GLO.FSInt_DI,FixP_out, QType_out);

%write the in text file
end
```

Main block's MATLAB code:

```
% ==========================================================================
% ECE 5746 - Simple Amplitude Envelope Model
% (c) 2019 studer@cornell.edu
% ==========================================================================

function sta = ENV(par,sta)
%FixP_out = {0,31,'s'}; % {I,F,'s'} where 's' is signed
FixP_out = {6,25,'s'};
FixP_par = {0,31,'s'};

% Avoid warnings when wrapping and truncating
QType_out = 'WrpTrc_NoWarn';

in = fopen('C:\Users\ChristineKu\Documents\GitHub\Applied-Digital-ASIC-Design\ENV_block_template\
fprintf(in, '%d %d %d %d %d\n', [1,0,0,0,sta.INP.Key_DO]);
fclose(in);

% Apply naming convention
ENV_key_DI = sta.INP.Key_DO;
ENV_pos_D = sta.ENV.pos;
ENV_inc_D = sta.ENV.inc;
ENV_prevKey_DP = sta.ENV.prevKey;
ENV_vol_DI = sta.INP.Vol_DO;

ENV_outPrev_DP = RealRESIZE(0, FixP_out,QType_out);

ENV_a_DI = par.ENV.a; %trial 0 (test.wav): a = 0.02, d = 0.02, r = 0.03
ENV_d_DI = par.ENV.d; %trial 1 (test1.wav): a = 0.01, d = 0.03, r = 0.03
ENV_r_DI = par.ENV.r; %trial 2 (test2.wav): a = 0.01, d = 0.02, r = 0.02
ENV_aY_DI = par.ENV.aY;
ENV_dY_DI = par.ENV.dY;
ENV_rY_DI = par.ENV.rY;

% convert to 32-bit signed fixed point
%in = RealRESIZE(in, FixP_out,QType_out);
ENV_vol_DI = RealRESIZE(ENV_vol_DI,{1,30,'s'},QType_out);
ENV_pos_D = RealRESIZE(ENV_pos_D, FixP_out,QType_out);
ENV_key_DI = RealRESIZE(ENV_key_DI, {1,30,'s'},QType_out);

ENV_dYaY_D = RealRESIZE(ENV_dY_DI - ENV_aY_DI, FixP_par, 'SatTrc_NoWarn');
ENV_rYdY_D = RealRESIZE(ENV_rY_DI - ENV_dY_DI, FixP_par, QType_out);

ENV_aSlope_D = par.ENV.aRec;
ENV_adSlope_D = RealRESIZE((ENV_dYaY_D*par.ENV.dRec), {8,23,'s'},QType_out);
ENV_drSlope_D = RealRESIZE(ENV_rYdY_D*par.ENV.rRec, {7,24,'s'},QType_out);

ENV_inc_D = RealRESIZE(par.ENV.glofsi, FixP_out, QType_out);

% increment position within one period of the sine wave
if(~((ENV_key_DI == 0) && (ENV_pos_D > ENV_r_DI)))
```

```
        ENV_pos_D = RealRESIZE(ENV_pos_D + ENV_inc_D, FixP_out,QType_out);
end

if(ENV_key_DI) % if key is pressed
    if(ENV_prevKey_DP == 0)
        ENV_pos_D = RealRESIZE(0,FixP_out,QType_out);
    else
        if(ENV_pos_D <= ENV_a_DI) % if within a range
            ENV_outPrev_DP = RealRESIZE(ENV_aSlope_D*ENV_pos_D,{0,31,'s'},'SatTrc_NoWarn');
        elseif(ENV_pos_D <= (ENV_d_DI+ENV_a_DI)) % if within d range
            ENV_da_D = RealRESIZE(ENV_d_DI+ENV_a_DI, FixP_par, QType_out);
            ENV_posda_D = RealRESIZE(ENV_pos_D - ENV_da_D, FixP_out, QType_out);
            ENV_adposda_D = RealRESIZE(ENV_posda_D*ENV_adSlope_D, FixP_out, QType_out);
            ENV_outPrev_DP = RealRESIZE(ENV_adposda_D+ENV_dY_DI,{0,31,'s'},QType_out);
        else
            ENV_outPrev_DP = RealRESIZE(ENV_dY_DI, {0,31,'s'}, QType_out);
        end
    end
else % if key is not pressed
    if(ENV_prevKey_DP)
        ENV_pos_D = RealRESIZE(0,FixP_out,QType_out);
    end
    if (ENV_pos_D <= ENV_r_DI)
        ENV_posr_D = RealRESIZE(ENV_pos_D-ENV_r_DI,FixP_out, QType_out);
        ENV_outPrev_DP=RealRESIZE(ENV_drSlope_D*ENV_posr_D,{0,31,'s'},QType_out);
    end
end

%disp(ENV_outPrev_DP);
ENV_outPrev_DP = RealRESIZE(ENV_outPrev_DP, {0,31,'s'}, QType_out); %only test this output
%run matlab simulatin to make sure it still works
%rewrute testenv to compare this value
%run in modelsim without multiplying by the vol (change env.v)

ENV_prevKey_DP = ENV_key_DI;
sta.ENV.Out_DO = RealRESIZE(ENV_outPrev_DP*ENV_vol_DI, FixP_par, QType_out); % send out sample va
if (sta.testcount == 7716)
    disp("cry");
end
sta.testcount=sta.testcount+1;
sta.ENV.pos=ENV_pos_D;
sta.ENV.prevKey = ENV_key_DI;

out = fopen('C:\Users\ChristineKu\Documents\GitHub\Applied-Digital-ASIC-Design\ENV_block_template
fprintf(out, '%d\n', round(sta.ENV.Out_DO*2^(31)));
fclose(out);
```

```
end
```

D Verilog Code

Main ENV code:

```
/* ==============================================================================
// ENV: Example block for ECE 5746 Applied Digital ASIC Design
// Author: ENV Team
// Date: 10/26/2019
// Project: SynTech
// ---- Description -----------------------------------------------------------
// Please describe your block here:
// The ENV block provides the envelope (ADSR) portion of the SynTech project for ECE 5746
// ---- Input/Output specifications ------------------------------------------
// Please clearly describe any specific aspect about the input-output ports of your
// module that other modules should be aware of for proper operation
// EXA_In_DI        sample input         [24bit]
// EXA_Out_DO        sample output         [24bit]
// ============================================================================== */


/* -----------------------------------------------------------------------------
// Module ports declaration
// ----------------------------------------------------------------------------- */
module ENV
#(
  // ---- The following parameters are the same for all blocks
  parameter ADDR_WIDTH = 5,  // number of entries in the parameter memory is equal to 2^(ADDR_WID
  parameter MEM_WIDTH = 32,  // word length of each memory entry to store parameters
  parameter DOUBLE_MEM_WIDTH = 64,
  // ---- the following parameters are ENV block specific (you can add or remove inputs and outpu
  parameter IN_WIDTH = 32,   // width of block inputs
  parameter OUT_WIDTH = 32,   // width of block outputs
  parameter PARAM_WIDTH = 24, //width of internal parameters
  parameter DATA_WIDTH = 24 // data width
)
(
  // ---- The following signals are the same for all blocks (no touchy!)
  input                        Clk_CI,     // Clock signal
  input                        Rst_RBI,    // Asynchronous active low reset
  input                        WrEn_SI,    // Active high write enable
  input           [ADDR_WIDTH-1:0] Addr_DI,    // Address of the parameter in the memory
  input                  [MEM_WIDTH-1:0]  PAR_In_DI,  // Parameter input (written from external int

  // ---- The following signals are ENV block specific (you can add or remove inputs and outputs)

  input signed      [IN_WIDTH-1:0]   ENV_key_DI,  // Input to the block (block can read from othe
  output signed [MEM_WIDTH-1:0]  ENV_Out_DO  // Output of the block (state). Use signed, whenever
);

/* ---------------------------------------------------------------------------------
// Common parameters for all blocks (do not change)
// --------------------------------------------------------------------------------- */

localparam MEM_DEPTH = (1 << ADDR_WIDTH);    // You can have at most 2^ADD_WIDTH parameters per bl
integer i;                                   // temporary variable
// ---- Defines block parameter memory that can be written from the outside of your module
// ---- Creates a register array (memory) to hold the parameter of the block
reg [MEM_WIDTH-1:0] parameter_memory [0:MEM_DEPTH-1];

/* ---------------------------------------------------------------------------------
// Internal parameters and signals/variables declaration that are block specific
// Define the internal wires, regs and all other variables of your module
// --------------------------------------------------------------------------------- */

// ---- Assign wires with meaningful names to the memory entries that actually hold used paramete
// ---- For example, if your block uses 4 parameters, only the 4 first entries of the memory shou
// ---- Assign a wire with descriptive names to each of these memory entries that hold a paramete
wire signed [MEM_WIDTH-1:0] ENV_prevKey_DP;
wire signed ENV_prevKey0_DP;
wire [DOUBLE_MEM_WIDTH-1:0] ENV_OutP_DP;
wire signed [MEM_WIDTH-1:0] ENV_vol_DI;
wire signed [MEM_WIDTH-1:0] ENV_Add1_Cout_D;
wire ENV_NandOut_D;
wire signed [MEM_WIDTH-1:0] ENV_mux_i_D;
wire signed [MEM_WIDTH-1:0] ENV_mux_j_D;
wire signed [MEM_WIDTH-1:0] ENV_mux_o_D;
//wire signed [DOUBLE_MEM_WIDTH-1:0] ENV_mult_l_D;
wire signed [DOUBLE_MEM_WIDTH-1:0] ENV_mult_m_D;
wire signed [MEM_WIDTH-1:0] ENV_pos_minus_r_D;
wire signed [MEM_WIDTH-1:0] ENV_Subtract1_Cout_D;
wire signed [MEM_WIDTH-1:0] ENV_rY_minus_dY_D;
wire signed [MEM_WIDTH-1:0] ENV_Subtract2_Cout_D;
wire signed [DOUBLE_MEM_WIDTH-1:0] ENV_mult2_D; // need to change to variable
wire signed [DOUBLE_MEM_WIDTH-1:0] ENV_mult3_D;
wire signed [MEM_WIDTH-1:0] ENV_mux_m_D;

wire signed [MEM_WIDTH-1:0] ENV_AddSumj_D;
wire signed [MEM_WIDTH-1:0] ENV_Addj_Cout_D;

wire signed [MEM_WIDTH-1:0] ENV_AddSumk_D;
wire signed [MEM_WIDTH-1:0] ENV_Addk_Cout_D;
wire signed [MEM_WIDTH-1:0] ENV_mux_n_D;

wire signed [MEM_WIDTH-1:0] ENV_dY_minus_aY_D;
wire signed [MEM_WIDTH-1:0] ENV_Subtracta_Cout_D;

wire signed [MEM_WIDTH-1:0] ENV_d_plus_a_D;
wire signed[MEM_WIDTH-1:0] ENV_Adda_Cout_D;

wire signed [MEM_WIDTH-1:0] ENV_pos_minus_daAdd_D;
wire signed [MEM_WIDTH-1:0] ENV_Subtractb_Cout_D;

wire signed [DOUBLE_MEM_WIDTH-1:0] ENV_recip_sub_D;
wire signed [DOUBLE_MEM_WIDTH-1:0] ENV_recip_sub_pos_D;

wire signed[MEM_WIDTH-1:0] ENV_old_pos_D;
wire signed[MEM_WIDTH-1:0] ENV_pos_D;

wire ENV_CompEq1_D;
wire ENV_CompLs1_D;
wire ENV_CompGr1_D;
wire ENV_CompEq2_D;
wire ENV_CompLs2_D;
wire ENV_CompGr2_D;
wire ENV_CompEq3_D;
wire ENV_CompLs3_D;
wire ENV_CompGr3_D;
wire ENV_CompEq4_D;
wire ENV_CompLs4_D;
wire ENV_CompGr4_D;
wire ENV_CompEq5_D;
wire ENV_CompLs5_D;
wire ENV_CompGr5_D;

wire signed[MEM_WIDTH-1:0] ENV_a_DI;
wire signed[MEM_WIDTH-1:0] ENV_d_DI;
wire signed[MEM_WIDTH-1:0] ENV_r_DI;
wire signed[MEM_WIDTH-1:0] ENV_aY_DI;
wire signed[MEM_WIDTH-1:0] ENV_rY_DI;
wire signed[MEM_WIDTH-1:0] ENV_dY_DI;
wire signed[MEM_WIDTH-1:0] ENV_inc_D;
wire signed[MEM_WIDTH-1:0] ENV_recip_a_DI;
wire signed[MEM_WIDTH-1:0] ENV_recip_d_DI;
wire signed[MEM_WIDTH-1:0] ENV_recip_r_DI;
wire signed[MEM_WIDTH-1:0] ENV_outPreVol_DP;
wire signed[MEM_WIDTH-1:0] ZERO;

assign ENV_vol_DI = parameter_memory[0];
assign ENV_a_DI = parameter_memory[1];
assign ENV_d_DI = parameter_memory[2];
assign ENV_r_DI = parameter_memory[3];
assign ENV_aY_DI = parameter_memory[4];
assign ENV_dY_DI = parameter_memory[5];
assign ENV_rY_DI = parameter_memory[6];
assign ENV_inc_D = parameter_memory[7];
assign ENV_recip_a_DI = parameter_memory[8];
assign ENV_recip_d_DI = parameter_memory[9];
assign ENV_recip_r_DI = parameter_memory[10];

assign ZERO = 32'b0;
assign ENV_prevKey0_DP = ENV_prevKey_DP[0];


/* ------------------------------------------------------------------------------
// Register Transfer Level (RTL) description of parameter memory (do not change)
// ------------------------------------------------------------------------------ */

// ---- Description of the parameter memory (same for all blocks)
always @(posedge Clk_CI or negedge Rst_RBI)
begin                                          // begin the memory control block
  if (!Rst_RBI) begin
    for (i = 0; i < MEM_DEPTH; i = i + 1)  // Reset all the parameters upon Rst_RBI being deasser

          parameter_memory[0] = 32'sb01_000000000000000000000000000000; //1
          parameter_memory[1] = 32'sb0_0011001100000000000000000000000; //0.2
          parameter_memory[2] = 32'sb0_0001101000000000000000000000000; //0.1
          parameter_memory[3] = 32'sb0_0011001100000000000000000000000; //0.2
          parameter_memory[4] = 32'sb01_000000000000000000000000000000; //1
          parameter_memory[5] = 32'sb0_1100110100000000000000000000000; //0.8
          parameter_memory[6] = 32'sb0_0000000000000000000000000000000;  //0
          parameter_memory[7] = 32'sb0_0000000000000000000000000000000; //0
          parameter_memory[8] = 32'sb0000101_000000000000000000000000; //5
          parameter_memory[9] = 32'sb0001010_000000000000000000000000; //10
          parameter_memory[10] = 32'sb0000101_000000000000000000000000; //5
  end else if (WrEn_SI) begin
    parameter_memory[Addr_DI] = PAR_In_DI; // Write parameter to memory if WrEn_SI is asserted hi
  end
end                                     // end the memory control block

/* ------------------------------------------------------------------------------
// ------------------------------------------------------------------------------
// Register Transfer Level (RTL) description of block specific code
// Here goes the actual RTL description of your own module
// ------------------------------------------------------------------------------
// ------------------------------------------------------------------------------ */
```

```

FF #(
  .DATA_WIDTH ( MEM_WIDTH )
)
FF_i
(
  .Clk_CI   ( Clk_CI        ),
  .Rst_RBI  ( Rst_RBI       ),
  .WrEn_SI  ( 1'b1          ),
  .D_DI     ( ENV_mux_j_D ),
  .Q_DO     ( ENV_old_pos_D )
);

FF #(
  .DATA_WIDTH ( MEM_WIDTH )
)
FF_j
(
  .Clk_CI   ( Clk_CI        ),
  .Rst_RBI  ( Rst_RBI       ),
  .WrEn_SI  ( 1'b1          ),
  .D_DI     ( ENV_key_DI ),
  .Q_DO     ( ENV_prevKey_DP )
);

XOR XOR_i
(
        .A_DI( ENV_key_DI[0] ),
        .B_DI( ENV_prevKey_DP[0] ),
        .Out_DO( SELECTOR )
);

//blue portion of block diagram

// 2-input MUX module
MUX2 #(
  .DATA_WIDTH ( MEM_WIDTH )
)
MUX2_i
(
  .In0_DI ( ZERO),
  .In1_DI ( ENV_inc_D),
  .Sel_SI (ENV_NandOut_D),
  .Out_DO ( ENV_mux_i_D )
);

```

```

// Add1
Add #(
  .DATA_WIDTH ( MEM_WIDTH )
)
Add_i
(
  .A_DI    ( ENV_mux_i_D ),
  .B_DI    ( ENV_old_pos_D ),
  .Sum_DO  ( ENV_pos_D),
  .Cout_DO ( ENV_Add1_Cout_D )
);

//comparator
COMP #(
   .DATA_WIDTH ( MEM_WIDTH )
)
COMP_i(
   .a(ENV_key_DI),
   .b(ZERO),
   .equal(ENV_CompEq1_D),
   .less(ENV_CompLs1_D),
   .greater(ENV_CompGr1_D)
);

COMP #(
   .DATA_WIDTH ( MEM_WIDTH )
)
COMP_j(
   .a(ENV_old_pos_D),
   .b(ENV_r_DI),
   .equal(ENV_CompEq2_D),
   .less(ENV_CompLs2_D),
   .greater(ENV_CompGr2_D)
);

Nand #(
   .DATA_WIDTH ( MEM_WIDTH )
)
Nand_i(
   .A_DI(ENV_CompEq1_D),
   .B_DI(ENV_CompGr2_D),
   .Out_DO(ENV_NandOut_D)
);

//----------------------------------------------
```

```
MUX2 #(
  .DATA_WIDTH ( MEM_WIDTH )
)
MUX2_j
(
  .In0_DI ( ENV_pos_D ),
  .In1_DI ( ZERO ),
  .Sel_SI (SELECTOR),
  .Out_DO ( ENV_mux_j_D )
);
//---------------------------------------------------

//yellow portion of block diagram
COMP #(
   .DATA_WIDTH ( MEM_WIDTH )
)
COMP_k(
   .a(ENV_old_pos_D),
   .b(ENV_r_DI),
   .equal(ENV_CompEq3_D),
   .less(ENV_CompLs3_D),
   .greater(ENV_CompGr3_D)
);

Subtract #(
  .DATA_WIDTH ( MEM_WIDTH )
)
Subtract_i
(
  .A_DI    ( ENV_mux_j_D ),
  .B_DI    ( ENV_r_DI    ),
  .Sum_DO  ( ENV_pos_minus_r_D),
  .Cout_DO ( ENV_Subtract1_Cout_D )
);

Subtract #(
  .DATA_WIDTH ( MEM_WIDTH )
)
Subtract_j
(
  .A_DI    ( ENV_rY_DI    ),
  .B_DI    ( ENV_dY_DI    ),
  //
  //.Cin_DI  ( ENV_Subtract2_Cin_D),
  .Sum_DO  ( ENV_rY_minus_dY_D),
  .Cout_DO ( ENV_Subtract2_Cout_D )
);

MULT #(
   .DATA_WIDTH ( MEM_WIDTH )
)
MULT_j
(
   .A_DI (ENV_recip_r_DI), //7,25
   .B_DI (ENV_rY_minus_dY_D), //1,31
   .Mult_DO(ENV_mult2_D) //8,24
);

MULT #(
   .DATA_WIDTH (MEM_WIDTH)
)
MULT_k
(
   .A_DI (ENV_pos_minus_r_D), //1,31
   .B_DI (ENV_mult2_D[63:32]), //8,24
   .Mult_DO(ENV_mult3_D) //9,23
);

MUX2 #(
  .DATA_WIDTH (MEM_WIDTH)
)
MUX2_m
(
  .In0_DI (ZERO), //1,31
  .In1_DI (ENV_mult3_D[49:18]), //9,23
  .Sel_SI (ENV_CompLs3_D),
  .Out_DO (ENV_mux_m_D) //1,31 or 9,23
);
// Pink part of the block diagram

FF #(
  .DATA_WIDTH ( MEM_WIDTH )
)
FF_k
(
  .Clk_CI   ( Clk_CI       ),
  .Rst_RBI  ( Rst_RBI      ),
  .WrEn_SI  ( 1'b1         ),
  .D_DI     ( ENV_OutP_DP[61:30] ),
  .Q_DO     ( ENV_Out_DO )
);

```

```
MUX2 #(
  .DATA_WIDTH ( MEM_WIDTH )
)
MUX2_l
(
  .In0_DI (ENV_mux_m_D), //1,31 or 9,23
  .In1_DI ( ENV_mux_o_D),
  .Sel_SI (ENV_key_DI[0]),
  .Out_DO ( ENV_outPreVol_DP) //
);

MULT #(
  .DATA_WIDTH ( MEM_WIDTH )
)
MULT_i
(
  .A_DI    ( ENV_vol_DI ), //2,30
  .B_DI    ( ENV_outPreVol_DP ), //7,25
  .Mult_DO ( ENV_OutP_DP ) //9,23
);

// Green part of the block diagram
Subtract #(
  .DATA_WIDTH (MEM_WIDTH)
)
Subtract_a
(
  .A_DI    ( ENV_dY_DI ),
  .B_DI    ( ENV_aY_DI ),
  //.Cin_DI  ( ENV_Subtracta_Cin_D ),
  .Sum_DO  ( ENV_dY_minus_aY_D ),
  .Cout_DO ( ENV_Subtracta_Cout_D )
);

Add #(
  .DATA_WIDTH ( MEM_WIDTH )
)
Add_a
(
  .A_DI    ( ENV_d_DI    ),
  .B_DI    ( ENV_a_DI    ),
  //.Cin_DI  ( ENV_Adda_Cin_D ),
  .Sum_DO  ( ENV_d_plus_a_D),
  .Cout_DO ( ENV_Adda_Cout_D )

);
```

```

Subtract #(
  .DATA_WIDTH (MEM_WIDTH)
)
Subtract_b
(
  .A_DI    ( ENV_old_pos_D ), //7 or 1
  .B_DI    ( ENV_d_plus_a_D ), //1,31
  //.Cin_DI  ( ENV_Subtractb_Cin_D ),
  .Sum_DO  ( ENV_pos_minus_daAdd_D ), //7 or 1
  .Cout_DO ( ENV_Subtractb_Cout_D )
);

MULT #(
   .DATA_WIDTH (MEM_WIDTH)
)
Mult_a
(
   .A_DI (ENV_dY_minus_aY_D ), //2,30
   .B_DI (ENV_recip_d_DI), //7,25
   .Mult_DO(ENV_recip_sub_D) //9,23
);

MULT #(
   .DATA_WIDTH (MEM_WIDTH)
)
Mult_b
(
   .A_DI (ENV_recip_sub_D[63:32]), //9,23
   .B_DI (ENV_pos_minus_daAdd_D), //1 or 7
   .Mult_DO(ENV_recip_sub_pos_D) //10 or 16
);

COMP #(
   .DATA_WIDTH ( MEM_WIDTH )
)
COMP_1
(
   .a(ENV_old_pos_D),
   .b(ENV_AddSumk_D),
   .equal(ENV_CompEq4_D),
   .less(ENV_CompLs4_D),
   .greater(ENV_CompGr4_D)
);

COMP #(
   .DATA_WIDTH ( MEM_WIDTH )
)
COMP_m
(
   .a(ENV_old_pos_D),
   .b(ENV_a_DI),
   .equal(ENV_CompEq5_D),
   .less(ENV_CompLs5_D),
   .greater(ENV_CompGr5_D)
);

/*
MULT #(
  .DATA_WIDTH ( MEM_WIDTH )
)
MULT_l
(
  .A_DI    ( ENV_aY_DI ), //0 fractional
  .B_DI    ( ENV_recip_a_DI ),//24 fractional
  .Mult_DO ( ENV_mult_l_D ) //8 int, 23 fractional, signed
);
*/

MULT #(
  .DATA_WIDTH ( MEM_WIDTH )
)
MULT_m
(
  .A_DI    ( ENV_mux_j_D ), //6int, 25 factional
  .B_DI    ( ENV_recip_a_DI ), //7 int, 24 fractional
  .Mult_DO ( ENV_mult_m_D ) //13int, 49 fractional bits
);

MUX2 #(
  .DATA_WIDTH ( MEM_WIDTH )
)
MUX2_o
(
  .In0_DI ( ENV_mux_n_D ),
  .In1_DI (ENV_mult_m_D[49:18] ),
  .Sel_SI (ENV_CompLs5_D),
  .Out_DO ( ENV_mux_o_D)
);

Add #(
  .DATA_WIDTH ( MEM_WIDTH )
```

```
)
Add_j
(
  .A_DI    ( ENV_recip_sub_pos_D[54:23] ), //10 or 16
  .B_DI    ( ENV_dY_DI    ),
  //.Cin_DI  ( ENV_Addj_Cin_D ),
  .Sum_DO  ( ENV_AddSumj_D),
  .Cout_DO ( ENV_Addj_Cout_D )

);

Add #(
  .DATA_WIDTH ( MEM_WIDTH )
)
Add_k
(
  .A_DI    ( ENV_a_DI    ),
  .B_DI    ( ENV_d_DI    ),
  //.Cin_DI  ( ENV_Addk_Cin_D ),
  .Sum_DO  ( ENV_AddSumk_D),
  .Cout_DO ( ENV_Addk_Cout_D )

);

MUX2 #(
  .DATA_WIDTH ( MEM_WIDTH )
)
MUX2_n
(
  .In0_DI (ENV_dY_DI),
  .In1_DI (ENV_AddSumj_D),
  .Sel_SI (ENV_CompLs4_D),
  .Out_DO (ENV_mux_n_D)
);
/* --------------------------------------------------------------------------------
// Done!
// -------------------------------------------------------------------------------- */

endmodule
```

Testbench code:
Below are the modules that we used in the ENV code: Add.v

```
/*
 * Add.v
 * Adder for real numbers
 *
 * Author: Oscar Castaneda
 * Last update: April 17, 2019
 */

module Add
#(
   parameter DATA_WIDTH = 32   //Number of bits for inputs and outputs
 )
 (
   input signed [DATA_WIDTH-1:0] A_DI,     // Numbers to
   input signed [DATA_WIDTH-1:0] B_DI,     // be added
   //input signed [DATA_WIDTH-1:0] Cin_DI,  // Carry-in

   output signed [DATA_WIDTH-1:0] Sum_DO, // Result of addition
   output signed [DATA_WIDTH-1:0] Cout_DO // Carry-out
 );

 assign {Cout_DO, Sum_DO} = A_DI + B_DI;

endmodule
```

COMP.v

```
/* ================================================================================
// 2-input Comparator
// Author: Christine Ku
// Date: 10/26/2019
// Project: SynTech
// ---- Description ----------------------------------------------------------------
// Comparator
// ---- Input/Output specifications ------------------------------------------------
// In0_DI                Oth input data signal
// In1_DI                1st input data signal
// Out_equal                Output data signal
// Out_less                        Output data signal
// Out_greater                Output data signal
// ================================================================================ */

module COMP
#(
  parameter DATA_WIDTH = 32
)
(
  input  wire signed [DATA_WIDTH-1:0] a,
  input  wire signed [DATA_WIDTH-1:0] b,
  output reg equal,
  output reg less,
  output reg greater
);

 always @* begin
    if (a<b) begin
           equal = 0;
           less = 1;
           greater = 0;
        end
        else if (a==b) begin
                equal = 1;
                less = 0;
                greater = 0;
        end
        else begin
                equal = 0;
                less = 0;
                greater = 1;
        end
  end
endmodule
```

FF.v

```
/*
 * FF.v
 * Flip-flop with asynchronous reset and write enable.
 *
 * Author: Oscar Castaneda
 * Last update: October 11, 2017
 */

module FF
#(
   parameter DATA_WIDTH = 32   // Number of bits for data
 )
 (
   input                        Clk_CI,  // Clock signal
   input                        Rst_RBI, // Asynchronous, active low reset
   input                        WrEn_SI, // Write enable

   input signed [DATA_WIDTH-1:0] D_DI,    // Input data to FF

   output reg signed [DATA_WIDTH-1:0] Q_DO     // Output data of FF
 );

 always@(posedge Clk_CI or negedge Rst_RBI)
   if(~Rst_RBI)
     Q_DO <= 0;
   else if(WrEn_SI)
     Q_DO <= D_DI;

endmodule
```

Nand.v

```
/*
 * Nand.v
 * Nand gate for real numbers
 *
 * Author: Group 2 (ENV)
 */

module Nand
#(
   parameter DATA_WIDTH = 32   //Number of bits for inputs and outputs
 )
 (
   input A_DI,     // Numbers to
   input B_DI,     // apply nand to

   output Out_DO // Output of nand gate
 );

 assign Out_DO = ~ (A_DI&B_DI);

endmodule
```

MUX2.v

```
/* ==================================================================================
// 2-input MUX
// Author: Seyed Hadi Mirfarshbafan
// Date: 10/22/2019
// Project: SynTech
// ---- Description ----------------------------------------------------------------
// Multibit 2-input multiplexer (MUX) template
// ---- Input/Output specifications --------------------------------------------------
// In0_DI                  0th input data signal
// In1_DI                  1st input data signal
// Sel_SI                    select bit (0 picks In0, 1 picks In1)
// Out_DO                  Output data signal
// ================================================================================== */
```

```

module MUX2
#(
  parameter DATA_WIDTH = 32
)
(
  input signed [DATA_WIDTH-1:0] In0_DI,
  input signed [DATA_WIDTH-1:0] In1_DI,
  input Sel_SI,
  output signed [DATA_WIDTH-1:0] Out_DO
);

  assign Out_DO = Sel_SI ? In1_DI: In0_DI;

endmodule
```

MULT.v

```
/* ================================================================================
// Multiplier
// Author: Group 2 (ENV)
// ================================================================================ */

module MULT
#(
  parameter DATA_WIDTH = 32,    // Number of bits for data
  parameter DATA_WIDTH_DOUBLE = 64
)
(
      input signed [DATA_WIDTH-1:0] A_DI,     // Numbers to multiply
      input signed [DATA_WIDTH-1:0] B_DI,

      output signed [DATA_WIDTH_DOUBLE-1:0] Mult_DO // Result of addition, output is twice the am
);

 assign Mult_DO = A_DI * B_DI;

endmodule
```

XOR.v

```
module XOR
 (
   input A_DI,     // Numbers to
   input B_DI,     //

   output Out_DO //
 );

 assign Out_DO = A_DI^B_DI;

endmodule
```

Subtract.v

```
/*
 * Subtract.v
 * Subtracter for real numbers
 *
 */

module Subtract
#(
   parameter DATA_WIDTH = 32   //Number of bits for inputs and outputs
 )
 (
   input signed [DATA_WIDTH-1:0] A_DI,    // Numbers to
   input signed [DATA_WIDTH-1:0] B_DI,    // be added
   //input signed [DATA_WIDTH-1:0] Cin_DI,  // Carry-in

   output signed [DATA_WIDTH-1:0] Sum_DO, // Result of addition
   output signed [DATA_WIDTH-1:0] Cout_DO // Carry-out
 );

 assign {Cout_DO, Sum_DO} = A_DI - B_DI;

endmodule
```