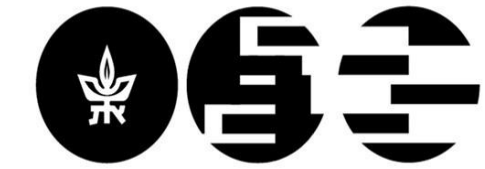

Iby and Aladar Fleischman
Faculty of Engineering
Tel Aviv University

הפקולטה להנדסה
ע"ש איבי ואלדר פליישמן
אוניברסיטת תל-אביב

# Project Documentation

# Advanced FPGA Based Waveform Generator

# 1. Hardware

## 1.1. Ctrl.v

### 1.1.1. Description

The Ctrl module is responsible for controlling the waveform generation process in the waveform generator system. It receives various inputs such as the clock signal (`clk`), reset signal (`rst_n`), BRAM read value (`bram_read`), and provides several outputs including BRAM address (`bram_addr`), frequency divisor reset signal (`freq_divisor_rstn`), BRAM reader reset signal (`bram_reader_rstn`), frequency divisor value (`freq_divisor_value`), number of samples (`num_of_samples`), phase accumulator constant (`phase_acc_const`), S data (`s_data`), and enable signal (`enb`).

### 1.1.2. Internal Constants

1.1.1.1. `SIZE`: Represents the size of internal registers.

1.1.1.2. `WAIT_FOR_NEW_REQ`, `READ_NUM_OF_SAMPLES`, `DONE`, `DUMMY_CYCLE`, `READ_PHASE_ACC_CONST`, `BRAM_READ_ADDR`: These are local parameters used to define different states within the state machine.

### 1.1.3. Internal Registers

The module utilizes several internal registers to store and control various signals throughout the waveform generation process. These registers include:

1.1.1.1. `state`: Represents the current state of the state machine.

1.1.1.2. `freq_divisor_rstn_reg`, `bram_reader_rstn_reg`: Store the frequency divisor reset and Bram reader reset signals, respectively.

1.1.1.3. `freq_divisor_value_reg`: Stores the value of the frequency divisor.

1.1.1.4. `bram_read_addr_reg`: Holds the current Bram read address.

1.1.1.5. `bram_read_reg`: Stores the Bram read value.

1.1.1.6. `num_of_samples_reg`: Stores the number of samples.

1.1.1.7. `phase_acc_const_reg`: Stores the phase accumulator constant.

1.1.1.8. `dummy_cycle_cnt_reg`: Keeps track of the dummy cycle count for synchronization purposes.

1.1.1.9. `s_reg`: Stores the S data signal.

1.1.1.10. `enb_reg`: Stores the enable signal.

1.1.4. State Machine

The module contains a state machine that controls the waveform generation process based on the current state. The state machine transitions through various states based on input signals and conditions. The different states include:

1.1.1.1. `WAIT_FOR_NEW_REQ` (initial state) (1): In this state, the CTRL module waits for a new request or trigger signal to initiate the data processing. It reads the frequency divisor value from the input data obtained from the BRAM. If it is not equal to zero – a new request is received thus its register is updated accordingly. After reading the frequency divisor value, the control module increments the BRAM address to point to the next data location and transitions to the next state, `READ_NUM_OF_SAMPLES`.

1.1.1.2. `READ_NUM_OF_SAMPLES` (2): In this state, the CTRL module reads the number of samples from the input data obtained from the BRAM and updating its register. After reading the number of samples, the control module increments the BRAM address to prepare for reading the next set of data and transitions to the next state, `READ_PHASE_ACC_CONST`.

1.1.1.3. `READ_PHASE_ACC_CONST` (3): Here, the CTRL module

reads the phase accumulator constant from the input data obtained from the BRAM and updates its register. Furthermore, it de-assert the BRAM READER and CLK DIV modules and sets the MUX's S to 1 causing the BRAM to be connected to the BRAM READER and disconnected from the CTRL. Finally, the control module transitions to the final state, `DONE`.

1.1.1.4. `DONE` (4): In the `DONE` state, the CTRL module has completed the necessary configuration and data retrieval tasks and there it stays until the next reset.

1.1.1.5. `DUMMY_CYCLE` (5): This state operates as a buffer between the states, giving the BRAM a cycle to output the read value of the read address.

1.1.5. Outputs

The module assigns values to the output signals based on the internal registers. These output signals include:

1.1.1.1. `bram_addr`: Represents the Bram address output.

1.1.1.2. `freq_divisor_rstn`: Provides the frequency divisor reset output.

1.1.1.3. `bram_reader_rstn`: Provides the Bram reader reset output.

1.1.1.4. `freq_divisor_value`: Outputs the value of the frequency divisor.

1.1.1.5. `num_of_samples`: Outputs the number of samples.

1.1.1.6. `phase_acc_const`: Provides the phase accumulator constant.

1.1.1.7. `s_data`: Outputs the S data signal.

1.1.1.8. `enb`: Outputs the enable signal.

## 1.2. bram_reader.v

### 1.2.1. Description

The bram_reader module is responsible for reading data from a BRAM memory in the waveform generator system. It receives various inputs such as the reference clock signal (`ref_clock`), reset signal (`rst_n`), Bram read value (`bram_read`), number of samples (`num_of_samples`), phase accumulator constant (`phase_acc_const`), and DAC ready signal (`dac_ready`). The module provides outputs including Bram read address (`bram_read_addr`), data (`data`), and Bram read validity (`bram_read_valid`).

### 1.2.2. Internal Constants

1.2.2.1. `BRAM_BASE_ADDR`: Represents the base address of the BRAM memory.

### 1.2.3. Internal Registers

1.2.3.1. `bram_read_addr_reg`: Stores the current Bram read address.

1.2.3.2. `data_reg`: Stores the data read from the Bram.

1.2.3.3. `state`: Represents the state of the module.

1.2.3.4. `bram_read_valid_reg`: Indicates the validity of the BRAM read operation.

1.2.3.5. `sample_counter`: Keeps track of the number of samples read.

### 1.2.4. State and Data Processing

The module operates on the rising edge of the reference clock (`ref_clock`). When the system is not in a reset state (`~rst_n`) or the DAC is not ready (`~dac_ready`), the module resets various internal registers and counters.

During normal operation, the module reads the Bram value (`bram_read`) and stores it in the `data_reg` register. It sets the

bram_read_valid_reg signal to indicate that a valid read operation has occurred. The module then checks if the sample_counter has reached the specified number of samples (num_of_samples). If the counter reaches the desired value, the Bram read address is reset to the base address, and the sample counter is reset to zero. Otherwise, the Bram read address and sample counter are incremented based on the phase accumulator constant (phase_acc_const) multiplied by 4.

#### 1.2.5. Outputs

The module assigns values to the following output signals based on the internal registers:

1.2.5.1. bram_read_addr: Represents the current Bram read address output.

1.2.5.2. data: Provides the data read from the Bram.

1.2.5.3. bram_read_valid: Indicates the validity of the Bram read operation.

### 1.3. Clk_div.v

#### 1.3.1. Description

The clk_div module is responsible for dividing the frequency of a reference clock signal (ref_clock) based on a specified divisor (divisor). It generates an output clock signal (out_clock) with a frequency that is a fraction of the reference clock frequency. The module also takes into account the reset signal (rst_n) to ensure proper initialization and synchronization.

#### 1.3.2. Internal Registers

1.3.2.1. count: Keeps track of the current count value for frequency division.

1.3.2.2. clk_track_even: Tracks the even division of the reference clock.

1.3.2.3. clk_track_odd: Tracks the odd division of the reference clock.

1.3.2.4. `pos_count`: Counts the positive edges of the reference clock.

1.3.2.5. `neg_count`: Counts the negative edges of the reference clock.

1.3.3. Even Divider

The module uses an even division approach to generate the output clock signal. On the rising edge of the reference clock (`ref_clock`), the module increments the count register by 1. If the count value exceeds or equals the specified divisor (`divisor-1`), it resets the count to zero. The `clk_track_even` signal is set to logic high (`1'b1`) when the count is less than half of the divisor value, and logic low (`1'b0`) otherwise.

1.3.4. Odd Divider

The module also employs an odd division technique to generate the output clock signal. On the positive edge of the reference clock, it checks the reset state and increments the `pos_count` register by 1 if the count is not at its maximum (`divisor-1`). On the negative edge of the reference clock, it performs a similar operation for the `neg_count` register. The `clk_track_odd` signal is assigned based on whether either `pos_count` or `neg_count` exceeds half of the divisor value (`divisor>>1`).

1.3.5. Output Clock Assignment

The `out_clock` signal is assigned based on the divisor value and the tracking signals. If the divisor is equal to 1, the output clock is directly set to the reference clock. Otherwise, if the divisor is an odd value (divisor % 2 evaluates to true), the `out_clock` is assigned the `clk_track_odd` signal. Otherwise, for even divisors, the `out_clock` is set to the `clk_track_even` signal.

## 2. Software

### 2.1. get_parameters()

The get_parameters function is responsible for obtaining all the necessary parameters for generating a waveform from the user. It interacts with the user through the console and prompts for various waveform parameters such as waveform implementation, waveform type, amplitude, frequency, offset, phase, duty cycle, and number of samples.

2.1.1. Parameters:

`wf_params` (pointer to `waveform_params` struct): A pointer to the waveform_params struct where the obtained parameters will be stored.

2.1.2. Return Type:

None

2.1.3. User Interface

The function presents a series of prompts to the user and receives their input to set the waveform parameters.

2.1.4. Prompt Sequence:

2.1.4.1. Choose waveform implementation:

User is prompted to select the waveform implementation by entering a number corresponding to the desired option: ARB or DDS (Direct Digital Synthesis).

The function verifies the validity of the input and prompts the user again if an invalid option is chosen.

2.1.4.2. Choose waveform type:

User is prompted to select the waveform type by entering a number corresponding to the desired option: Sine wave,

Square wave, Sawtooth wave, Triangle wave, or Arbitrary waveform using an external file.

The function verifies the validity of the input and prompts the user again if an invalid option is chosen.

2.1.4.3. Arbitrary waveform file (if waveform type is Arbitrary):

If the user selects the Arbitrary waveform type, they are prompted to enter the file path containing the waveform samples.

The function checks if the file exists and prompts the user again if the file is not found.

2.1.4.4. Amplitude:

User is prompted to enter the desired amplitude of the waveform in volts.

The function verifies the validity of the input and prompts the user again if an invalid amplitude is entered.

2.1.4.5. Frequency:

User is prompted to enter the desired frequency of the waveform in hertz.

The function verifies the validity of the input and prompts the user again if an invalid frequency is entered.

2.1.4.6. Offset:

User is prompted to enter the desired offset of the waveform in volts.

The function verifies the validity of the input and prompts the user again if an invalid offset is entered.

2.1.4.7. Phase (for Sine wave):

If the user selects the Sine wave type, they are prompted to enter the desired phase in radians. The phase represents the phase shift or starting phase of the sine wave.

2.1.4.8. Duty Cycle (for Square wave):

If the user selects the Square wave type, they are prompted to enter the desired duty cycle in percentage.

The duty cycle represents the ratio of the duration of the high state to the total period of the waveform.

2.1.4.9. Number of samples:

User is prompted to enter the desired number of samples for the waveform.

The function verifies the validity of the input and prompts the user again if an invalid number of samples is entered. For the Arbitrary waveform, the number of samples is obtained from the external file.

## 2.2. generate_waveform()

The generate_waveform function is responsible for storing the waveform parameters and samples in the BRAM. It takes the waveform parameters as input and writes the generated waveform samples to the BRAM.

2.2.1. Parameters:

wf_params (pointer to waveform_params struct): A pointer to the waveform_params struct containing the waveform parameters.

2.2.2. Return Type:

None

2.2.3. Function Flow:

2.2.3.1. Memory Mapping:

The function opens the /dev/mem device file to access physical memory. It maps a portion of memory (2 pages) starting at address 0x40000000 using the mmap system call. The mapped memory region is accessible for both reading and writing.

2.2.3.2. Waveform Generation and Storage:

The function calculates the frequency divisor and phase accumulator constant based on the waveform parameters and implementation type. It stores the frequency divisor at

the memory address 0x40000000, the number of samples at 0x40000004, and the phase accumulator constant at 0x40000008 in the BRAM. It iterates over each sample in the waveform. For each sample, the function calculates the current sample value using the get_curr_sample function based on the waveform parameters and the sample index. The calculated sample value is converted to fixed-point format using the float_to_fixed function. The fixed-point sample value is written to memory starting at the address 0x4000000C in the BRAM.

### 2.3. float_to_fixed()

The float_to_fixed function converts a floating-point value to a fixed-point representation. It performs the conversion by multiplying the input value by a scaling factor and clamping the result within a specific range.

2.3.1. Parameters:

`value` (float): The floating-point value to be converted to fixed-point.

2.3.2. Return Type:

int: The converted fixed-point value.

2.3.3. Fixed-Point Conversion:

The function multiplies the input value by $2^{13}$ to move the decimal point 13 places to the right. This scaling factor is used to convert the floating-point value to fixed-point representation. The result of the multiplication is cast to an integer, fixed, representing the fixed-point value. The function clamps the value of fixed to ensure it falls within the range of valid fixed-point values. If fixed is greater than the maximum representable fixed-point value (1 << 13) - 1, it is set to the maximum value. If fixed is less than the minimum

representable fixed-point value -(1 << 13), it is set to the minimum value.

### 2.4. get_curr_sample()

The get_curr_sample function calculates and returns the current sample value for a waveform based on the specified waveform type in the `waveform_params` structure. The waveform type determines which specific waveform calculation function or sample array is used to retrieve the sample value.

2.4.1. Parameters:

2.4.1.1. `wf_params` (waveform_params*): A pointer to the waveform_params structure that contains the waveform parameters and type.

2.4.1.2. `i` (int): The index of the current sample.

2.4.2. Return Type:

float: The current sample value of the waveform.

2.4.3. Waveform Calculation:

The function evaluates the waveform type specified in `wf_params->waveform_type` using a switch statement and returns the corresponding waveform sample value.

For each waveform type, the function calls the appropriate waveform calculation function or retrieves the sample value from the arbitrary sample array, based on the waveform type.

The calculated sample value is returned by the function.

2.4.4. Supported Waveform Types:

2.4.4.1. `SINE`: Returns the current sample value for a sine waveform by calling the `get_curr_sine_sample` function.

2.4.4.2. `SQUARE`: Returns the current sample value for a square

waveform by calling the `get_curr_square_sample` function.

2.4.4.3. `SAWTOOTH`: Returns the current sample value for a sawtooth waveform by calling the `get_curr_sawtooth_sample` function.

2.4.4.4. `TRIANGLE`: Returns the current sample value for a triangle waveform by calling the `get_curr_triangle_sample` function.

2.4.4.5. `ARBIT`: Returns the current sample value from the arbitrary sample array with the appropriate amplitude and offset.