D.1 Design Unit Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Design Unit Index

D.2 Design Unit List

Here is a list of all design unit members with links to the Entities they belong to:

File Index

D.3 File List

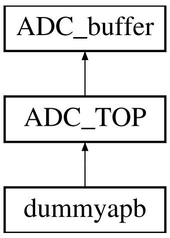
Here is a list of all documented files with brief descriptions:

Class Documentation

D.4 ADC_buffer Entity Reference

This component buffers samples when buff_write goes from low to high, until the buffer is full. When this happens Buffer full is driven high for one clock cycle. The softcore will the read the values from the buffer by changing the address. The buffer will then instantly output the value on this address on the buffout port.

Inheritance diagram for ADC_buffer:



Entities

- Buffer_ADC architecture

  Architecture of the ADC buffer.

Libraries

- IEEE

  Use of standard library.

Use Clauses

- IEEE.STD_LOGIC_1164.all

  Use of standard logic arguments.
- IEEE.NUMERIC_STD.all

  Use of standard numerical arguments.

Generics

- bufferwidth **integer:=** 7

  Generic indicating the width of the buffer address.

Ports

- clk **in STD_LOGIC**

  clock for buffer registers.
- rst **in STD_LOGIC**

  Global reset, active low.
- buff_write **in STD_LOGIC**

  Controls when values are to be stored, also changes the buffer memory pointer.
- Buffin **in STD_LOGIC_VECTOR(** 15 **downto** 0)

  Input value to be stored.
- Buffout **out STD_LOGIC_VECTOR(** 15 **downto** 0)

  Output value from the current address slot.
- Bufferfull **out STD_LOGIC**

  high for one clock cycle when the buffer is full, low otherwise. This is due to Leon 3 only accepting interrupts being high for one clock cycle.
- Addr **in STD_LOGIC_VECTOR(bufferwidth -** 1 **downto** 0)

  Address currently asked for.

D.4.1 Detailed Description

This component buffers samples when buff_write goes from low to high, until the buffer is full. When this happens Buffer full is driven high for one clock cycle. The softcore will the read the values from the buffer by changing the address. The buffer will then instantly output the value on this address on the buffout port.

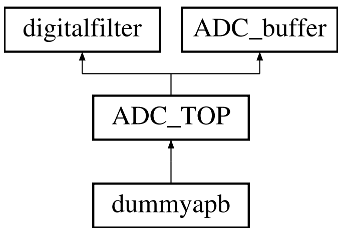
The documentation for this class was generated from the following file:

- ADC_buffer.vhd

D.5 ADC_TOP Entity Reference

Use of standard logic arguments.

Inheritance diagram for ADC_TOP:



Entities

- TOP_ADC architecture

  ADC_TOP.

Libraries

- IEEE

  This line outputs the value of the buffer of the current address to the out port.

Use Clauses

- IEEE.STD_LOGIC_1164.all

  Use of standard library.

Ports

- CLK **in STD_LOGIC**

  This component is a wrapper for the ADC, a buffer and the components needed to complete the decimation. Its main functionality is to provide for internal connections between the components. Global clock running at 50 MHz.
- CLK100 **in STD_LOGIC**

  A clock on 100MHz to let the filter have more taps.
- RST **in STD_LOGIC**

  Global reset active low.
- sampleclk **in STD_LOGIC**

  Sample enable running at ∼44100 Hz.
- vauxp3 **in STD_LOGIC**

Positive analogue signal.
- vauxn3 **in STD_LOGIC**

  Negative analogue signal.
- addr **in STD_LOGIC_vector(6 downto 0)**

  Address from the softcore.
- buff_full **out STD_LOGIC**

  Signal indicating the buffer is full.
- ADC_buff_write **in STD_LOGIC**

  Signal indicating the buffer should be written.
- ADC_buff_out **out STD_LOGIC_VECTOR(15 downto 0)**

  Sampled value after decimation.

D.5.1 Detailed Description

Use of standard logic arguments.

The documentation for this class was generated from the following file:

- ADC_TOP.vhd

D.6 APB_interface Architecture Reference

Architecture of the Dummy_apb.

Processes

- regs(**clk** , **rstn**)

Components

- ADC_TOP
- DacTop

Constants

- pconfig **apb_config_type:=(0 =>ahb_device_reg(VENDOR_GROUP,OWN_ADC, 0 , 0 , 0), 1 =>apb_-iobar(paddr ,pmask))**

Signals

- sLED **std_logic_vector(31 downto 0)**

  Buffered signals from the APB bus.
- sampledvalue **STD_LOGIC_VECTOR(15 downto 0)**

  Sampled value from the ADC_buffer.
- sampleclk **std_logic**

  Sample clock for the ADC.
- ADDR **STD_LOGIC_VECTOR(6 downto 0)**

  Address for the ADC_buffer and DAC_buffers.
- buffer_interupt **STD_LOGIC**

  Interrupt from the ADC_buffer.

- sampleena44kHz **STD_LOGIC**

  Sample clock for the ADC buffer.
- dac_buff_write **STD_LOGIC**

  This signal comes from the right address combination indicating the DAC_buffer is to read.
- irq **STD_LOGIC**

  Buffered interrupt to prevent the interrupt if being high for more than one clock cycle.

Instantiations

- inst_top **dactop**

  This vector configs the address ranging and start address.
- inst_adc_top **ADC_TOP**
- bootmsg **report_version**

D.6.1 Detailed Description

Architecture of the Dummy_apb.

The Dummy APB creates an interface between the APB and the ADC/DAC. This is done in the simplest way possible. The address to the ADC and DAC buffer is part of the APB address to this component.

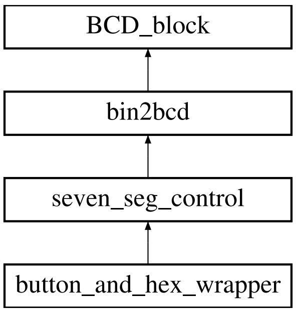
The documentation for this class was generated from the following files:

- dummyapb.vhd

D.7 BCD_block Entity Reference

This module is used in the conversion of binary to binary coded decimals.

Inheritance diagram for BCD_block:



Entities

- LUT architecture

  Architecture of the BCD_block.

Libraries

- IEEE

  Use of standard library.

Use Clauses

- IEEE.STD_LOGIC_1164.all

  Use of standard logic arguments.

Ports

- in_vector **in STD_LOGIC_VECTOR(3 downto 0)**

  Input binary vector.
- out_vector **out STD_LOGIC_VECTOR(3 downto 0)**

  output BCD vector

D.7.1 Detailed Description

This module is used in the conversion of binary to binary coded decimals.

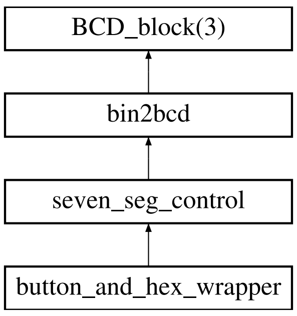
The documentation for this class was generated from the following file:

- BCD_block.vhd

D.8 bin2bcd Entity Reference

This component calculates the binary coded decimal equivalent of a binary number. This module is not completely generic yet but it is verified to work at the size used in the implementation. For updates check `https://github.com/Jaxc/bin2bcd`.

Inheritance diagram for bin2bcd:



Entities

- converter architecture

  bin2BCD component is a generic binary to binary coded decimal. It does this by using the BCD_block according to the method used in `http://www.johnloomis.org/ece314/notes/devices/binary_to_BCD/bin_to_bcd.html`*. This module is not completely generic yet but it is verified to work at the size used in the implementation. For updates check* `https://github.com/Jaxc/bin2bcd`

Libraries

- IEEE

  Use of standard library.

Use Clauses

- IEEE.STD_LOGIC_1164.all

  Use of standard logic arguments.
- IEEE.NUMERIC_STD.all

  Use of standard numerical arguments.
- IEEE.MATH_REAL.all

  Use of real math arguments to calculate generics.

Generics

- bits **integer**:= 8

  binary bit width

Ports

- bin **in STD_LOGIC_VECTOR(bits - 1 downto 0)**

  Binary input.
- BCD **out STD_LOGIC_VECTOR(bits $*$ 2 - 1 downto 0)**

  BCD output.

D.8.1 Detailed Description

This component calculates the binary coded decimal equivalent of a binary number. This module is not completely generic yet but it is verified to work at the size used in the implementation. For updates check `https://github.com/Jaxc/bin2bcd`.

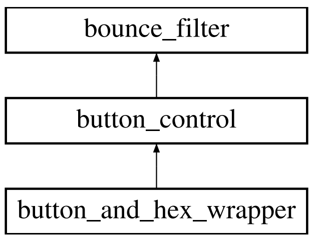
The documentation for this class was generated from the following file:

- bin2bcd.vhd

D.9 bounce_filter Entity Reference

This component stabilized signals by waiting until a signal have been high or low for a set about of time.

Inheritance diagram for bounce_filter:



Entities

- filter_bounce architecture

  Architecture of the bounce_filter.

Libraries

- IEEE

  Use of standard library.

Use Clauses

- IEEE.STD_LOGIC_1164.all

  Use of standard logic arguments.
- IEEE.NUMERIC_STD.all

  Use of standard numerical arguments.

Generics

- counterbits **integer:=** 8

  Counter bits controls how many bits the counter will count before it changes value.

Ports

- Button_in **in STD_LOGIC**

  Button_in is the input to the filter.
- clk **in STD_LOGIC**

  Clock for counter and registers.
- rstn **in STD_LOGIC**

  Global reset, active low.
- Button_out **out STD_LOGIC**

  Stabilized signal out.

D.9.1 Detailed Description

This component stabilized signals by waiting until a signal have been high or low for a set about of time.

The documentation for this class was generated from the following file:

- bounce_filter.vhd

D.10 Buffer_ADC Architecture Reference

Architecture of the ADC buffer.

Processes

- PROCESS_0(**clk** , **rst**)

  The main process of the module. In this process handles the writing to the buffer. This process is Dependant on the clock and the reset.

Types

- **Memory_array_typeisarray(0 to 2 ∗∗bufferwidth - 1)ofSTD_LOGIC_VECTOR(15 downto 0**

  The memory storage type. The type creates an array of a size of 2^ bufferwidth∗16.

Signals

- Memory_array **Memory_array_type**

  The actual memory storage element. This signal holds all of the stored elements.
- lastwrite **STD_LOGIC**

  Lastwrite holds the last value of buff_write to be able to detect a rising edge without using the signal as a clock.
- Write_index **integerrange 0 to 2 ∗∗bufferwidth - 1**

  Write_index is a signal to indicate where the next value in the buffer is to be written. When this happens the write_-index is incremented by one. This makes the buffer write in a circular fashion.

D.10.1 Detailed Description

Architecture of the ADC buffer.

The architecture containing the main body of the component.

The documentation for this class was generated from the following file:

- ADC_buffer.vhd

D.11 Buffer_dac Architecture Reference

Architecture of the DAC buffer.

Processes

- PROCESS_6(**clk** , **rst**)

Types

- **Memory_array_typeisarray(0 to 2 ∗∗bufferwidth - 1)ofSTD_LOGIC_VECTOR(15 downto 0**

  the type for the memory array, the length changes size due to a generic

Signals

- Memory_array **Memory_array_type**

  This signal is containing the buffered values.
- lastread **STD_LOGIC**

  stored value of the last read
- read_index **integerrange 0 to 2 ∗∗bufferwidth - 1**

  this signal keeps track of where to write the next value in the memory.

D.11.1 Detailed Description

Architecture of the DAC buffer.

The architecture containing the main body of the component.

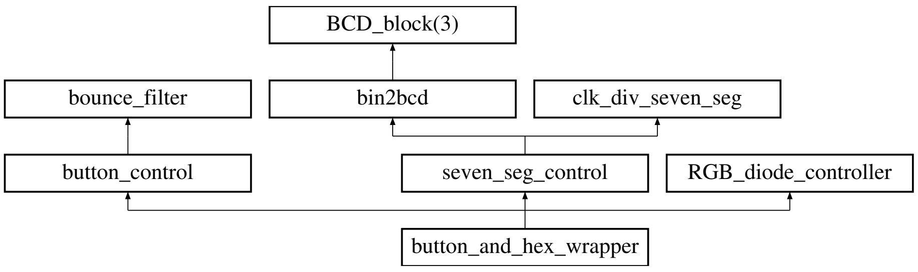
The documentation for this class was generated from the following file:

- DAC_BUFFER.vhd

D.12 button_and_hex_wrapper Entity Reference

This component gathers all the sub-modules needed for HID. It also supplies an interface to the APB bus.

Inheritance diagram for button_and_hex_wrapper:



Entities

- HID_wrapper architecture

  Architecture of the Dummy_apb.

Libraries

- IEEE

  Use of standard library.
- grlib

  use of the GRLIB

Use Clauses

- IEEE.STD_LOGIC_1164.all

  Use of standard logic arguments.
- grlib.amba.all

  Use of the AMBA bus signals and constants.
- grlib.stdlib.all

  Use of standard GRLIB signals and constants.
- grlib.devices.all

  use of GRLIB devices signals and constants

Generics

- pindex **integer:=** **0**

  Slave index.
- paddr **integer:=** **0**

  Address of the APB bank.
- pmask **integer:=** **16#002#**

  Address range.

Ports

- rstn **in std_ulogic**

  global reset, active low
- clk **in std_ulogic**

  Clock at the bus speed of 50 MHz.
- apbi **in apb_slv_in_type**

  APB slave inputs.
- apbo **out apb_slv_out_type**

  APB slave outputs.
- Buttons_in **in STD_LOGIC_VECTOR(4 downto 0)**

  Buttons in from the board.
- seven_seg_out **out STD_LOGIC_VECTOR(6 downto 0)**

  Seven_sel_out marks the seven segment display of the current selected seven segment display segment.
- seven_seg_sel **out STD_LOGIC_VECTOR(7 downto 0)**

  Seven_seg_sel marks the current selected seven segment display.
- diode_out **out STD_LOGIC_vector(2 downto 0)**

  diode_out is a vector containing the state of the red, green and blue diode

D.12.1 Detailed Description

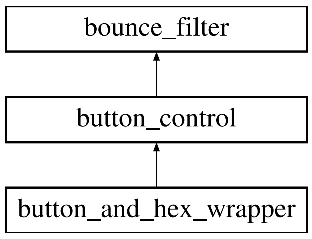
This component gathers all the sub-modules needed for HID. It also supplies an interface to the APB bus.

The documentation for this class was generated from the following file:

- Button_and_hex_wrapper.vhd

D.13 button_control Entity Reference

Inheritance diagram for button_control:



Entities

- control_button architecture

  Architecture of the button_control.

Libraries

- IEEE

  Use of standard library.

Use Clauses

- IEEE.STD_LOGIC_1164.all

  Use of standard logic arguments.
- IEEE.NUMERIC_STD.all

  Use of standard numerical arguments.

Ports

- clk **in STD_LOGIC**

  Clock in for registers.
- rstn **in STD_LOGIC**

  Global reset, active low.
- buttons_in **in STD_LOGIC_VECTOR(4 downto 0)**

  Buttons in.
- current_preset **out STD_LOGIC_VECTOR(7 downto 0)**

  Current value out.
- selected_preset **out STD_LOGIC_VECTOR(7 downto 0)**

  Selected value out.
- read_interupt **out STD_LOGIC**

  Interrupt indicating a read from flash is to be done.
- write_interupt **out STD_LOGIC**

  Interrupt indicating a write to flash is to be done.

The documentation for this class was generated from the following file:

- button_control.vhd

D.14 clk_div_DAC Architecture Reference

Architecture of the clk_divider.

Processes

- PROCESS_5**(clk , rst)**

  When the system is reseted all values are set to 0.

Constants

- cnt44kHz_max **integer:=integer(round(real(systemclock)/real((sampleclock ∗OSR))))∗OSR -**

  The cnt44kHz max calculates the number the counter has to reach to reset. The calculation is as follows-: $round({systemclock}{sampleclock∗OSR})∗ORS-1. The OSR is in the equation to make sure the rate between sample clock and OSR will be correct.

Signals

- cnt44kHz **integerrange 0 tocnt44kHz_max**

  Counter signal with the required range.
- clk50MHzbuf **STD_LOGIC**

  Buffered value for the 50 MHz clock to be able to use their states for calculation.
- clk25MHzbuf **STD_LOGIC**

  Buffered value for the 25 MHz clock to be able to use their states for calculation.
- clk44kHzbuf **STD_LOGIC**

  Buffered value for the 44 kHz clock to be able to use their states for calculation.
- clk705kHzbuf **STD_LOGIC**

  Buffered value for the 705 kHz clock to be able to use their states for calculation.

D.14.1 Detailed Description

Architecture of the clk_divider.

The architecture containing the main body of the component.

The documentation for this class was generated from the following file:

- CLK_divide.vhd

D.15 clk_div_seven Architecture Reference

Architecture of the clk_div_seven_seg.

Processes

- PROCESS_4(**rstn** , **clk**)

Signals

- cnt **integerrange 0 to 2** $**$**counterbits - 1**

  The counter.

D.15.1 Detailed Description

Architecture of the clk_div_seven_seg.

This clock divider is need to make sure seven segments of the seven segment displays in the correct speed. To fast and numbers will "float" to its neighbors due to slow transients, to slow and the numbers will appear as flashing instead of solid.

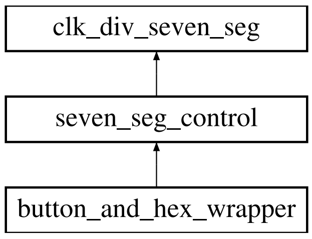
The documentation for this class was generated from the following file:

- clk_div_seven_seg.vhd

D.16 clk_div_seven_seg Entity Reference

This component takes the system clock divides it for Seven segment displays. This is done by implementing counters.

Inheritance diagram for clk_div_seven_seg:



Entities

- clk_div_seven architecture

  Architecture of the clk_div_seven_seg.

Libraries

- IEEE

  Use of standard library.

Use Clauses

- IEEE.STD_LOGIC_1164.all

  Use of standard logic arguments.
- IEEE.NUMERIC_STD.all

  Use of standard numerical arguments.

Generics

- counterbits **integer:=** 17

  A generic for setting the bits of the counter.

Ports

- clk **in STD_LOGIC**

  Clock for counter and registers.
- rstn **in STD_LOGIC**

  Global reset, active low.
- slow_clk **out STD_LOGIC**

  The output slower clock.

D.16.1 Detailed Description

This component takes the system clock divides it for Seven segment displays. This is done by implementing counters.

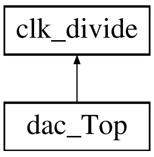
The documentation for this class was generated from the following file:

- clk_div_seven_seg.vhd

D.17 clk_divide Entity Reference

This component takes the system clock divides it for ADC/DAC components using lower clocks. This is done by implementing counters. When a certain counter reached a set number it will change the output and reset the counter.

Inheritance diagram for clk_divide:



Entities

- clk_div_DAC architecture

  Architecture of the clk_divider.

Libraries

- IEEE

  Use of standard library.

Use Clauses

- IEEE.STD_LOGIC_1164.all

  Use of standard logic arguments.
- IEEE.NUMERIC_STD.all

  Use of standard numerical arguments.
- IEEE.MATH_REAL.all

  Use of real math arguments to calculate generic divisions.

Generics

- systemclock **integer:= 100000000**

  Generic setting the system clock speed.
- sampleclock **integer:= 44100**

  Generic describing the intended sampling frequency.
- OSR **integer:= 16**

  Generic describing the Over Sampling Ratio.

Ports

- rst **in STD_LOGIC**

  Global reset, active low.
- clk **in STD_LOGIC**

  Clock in, in our case the 100MHz clock to improve the accuracy of the sample clock.
- clk50MHz **out STD_LOGIC**

  Clock out at half of the input clock frequency.
- clk25MHz **out STD_LOGIC**

  Clock out at a fourth of the input clock frequency.
- clk705kHz **out STD_LOGIC**

  Clock out at the over sampling rate (sampling rate ∗ OSR)
- clk44kHz **out STD_LOGIC**

  Clock out at the sampling frequency.

D.17.1 Detailed Description

This component takes the system clock divides it for ADC/DAC components using lower clocks. This is done by implementing counters. When a certain counter reached a set number it will change the output and reset the counter.

The documentation for this class was generated from the following file:

- CLK_divide.vhd

D.18 control_button Architecture Reference

Architecture of the button_control.

Processes

- PROCESS_3(**rstn** , **clk**)

Components

- bounce_filter

Signals

- current_counter **unsigned(7 downto 0)**

  This signal stores the current value of the counter.
- selected_counter **unsigned(7 downto 0)**

  This signal stores the selected value.
- stable_buttons **STD_LOGIC_VECTOR(4 downto 0)**

  This vector holds the stabilized values of the buttons.

Instantiations

- bounce_fix **bounce_filter**

D.18.1 Detailed Description

Architecture of the button_control.

The button control takes the inputs from the buttons and stabilizes them with the bounce_filter. Then, depending on what button different commands are done. If the left or right button is pushed the current counter is incremented or decremented by one. If the middle button is pushed the current value is copied to the selected value and a read interrupt is sent. If the down button is pushed the current value is copied to the selected value and write interrupt is sent. If the up button is pressed nothing happens.

The documentation for this class was generated from the following files:

- button_control.vhd

D.19 converter Architecture Reference

bin2BCD component is a generic binary to binary coded decimal. It does this by using the BCD_block according to the method used in `http://www.johnloomis.org/ece314/notes/devices/binary_to_BCD/bin_to_bcd.html`. This module is not completely generic yet but it is verified to work at the size used in the implementation. For updates check `https://github.com/Jaxc/bin2bcd`

Components

- BCD_block

  Input binary vector.

Constants

- array_length **integer:=bits +integer(floor(real(bits)/real(4)))**

  output BCD vector Constant calculating the needed length of the array.

Types

- **array_typeisarray(0 to 5)ofSTD_LOGIC_VECTOR(array_length - 1 downto 0)**

  An array at the size of array_length and length of 6. The length is not get generic.

Signals

- temp_vector **array_type**

  A signal of the array_type for storing temporary values in the converter.

Instantiations

- inst_bcd **BCD_block**
- inst_bcd **BCD_block**
- inst_bcd **BCD_block**

D.19.1 Detailed Description

bin2BCD component is a generic binary to binary coded decimal. It does this by using the BCD_block according to the method used in `http://www.johnloomis.org/ece314/notes/devices/binary_to_BCD/bin_to_bcd.html`. This module is not completely generic yet but it is verified to work at the size used in the implementation. For updates check `https://github.com/Jaxc/bin2bcd`

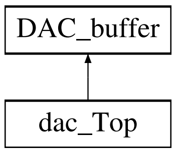
The documentation for this class was generated from the following file:

- bin2bcd.vhd

D.20 DAC_buffer Entity Reference

This component buffers samples from the softcore at the current address when buffwrite is high. The buffer will then output the values on when the buffRead goes from low to high.

Inheritance diagram for DAC_buffer:



Entities

- Buffer_dac architecture

  Architecture of the DAC buffer.

Libraries

- IEEE

  Use of standard library.

Use Clauses

- IEEE.STD_LOGIC_1164.all

  Use of standard logic arguments.
- IEEE.NUMERIC_STD.all

  Use of standard numerical arguments.

Generics

- bufferwidth **integer:=** 7

  Generic indicating the width of the buffer address.

Ports

- clk **in STD_LOGIC**

  Generic indicating the width of the buffer address.
- rst **in STD_LOGIC**

Global reset, active low.

- buffRead **in STD_LOGIC**

  signal to change the read value from the buffer
- indexReset **in STD_LOGIC**

  Controls when values are to be stored.
- buffWrite **in STD_LOGIC**

  controls when to change the output value
- buffIn **in STD_LOGIC_VECTOR(15 downto 0)**

  Input value to be stored.
- buffOut **out STD_LOGIC_VECTOR(15 downto 0)**

  Output value from the memory.
- addr **in STD_LOGIC_VECTOR(bufferwidth - 1 downto 0)**

  Address currently written to.

D.20.1 Detailed Description

This component buffers samples from the softcore at the current address when buffwrite is high. The buffer will then output the values on when the buffRead goes from low to high.

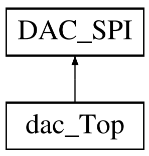
The documentation for this class was generated from the following file:

- DAC_BUFFER.vhd

D.21 DAC_SPI Entity Reference

The DAC SPI interface converts parallel data from the data port and transforms it to a SPI to be sent to the external DAC chip on the din port. The module also adds flag bits to this signal, as well as a chip select signal called n_sync. The interface listens to the sample clock and only transmits a new message when this clock goes from low to high.

Inheritance diagram for DAC_SPI:



Entities

- SPI architecture

  Architecture of the DAC_SPI.

Libraries

- IEEE

  Use of standard library.

Use Clauses

- IEEE.STD_LOGIC_1164.all

  Use of standard logic arguments.

Ports

- rstn **in STD_LOGIC**

  Global reset active low.
- clk **in STD_LOGIC**

  Clock in, in this case the 25 MHz SPI clock.
- data **in STD_LOGIC_VECTOR(15 downto 0)**

  Data vector containing the parallel sample.
- sampleclk **in STD_LOGIC**

  The sampleclock indicating a new sample is available, this triggers a new transmission.
- din **out std_logic**

  din is the serial connected to the DAC IC
- nSync **out STD_LOGIC**

  nsync is the chip select for the DAC IC

D.21.1 Detailed Description

The DAC SPI interface converts parallel data from the data port and transforms it to a SPI to be sent to the external DAC chip on the din port. The module also adds flag bits to this signal, as well as a chip select signal called n_sync. The interface listens to the sample clock and only transmits a new message when this clock goes from low to high.

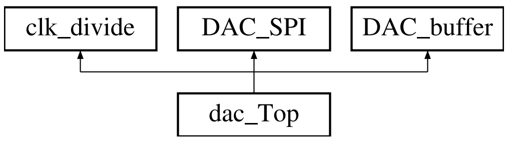
The documentation for this class was generated from the following file:

- DAC_SPI.vhd

D.22 dac_Top Entity Reference

This component is a wrapper for the parts needed for the digital to analog converter. Its main functionality is to provide for internal connections between the components and to throughput any signals to the next level in the hierarchy.

Inheritance diagram for dac_Top:



Entities

- TOP_DAC architecture

  Architecture of the DACTOP.

Libraries

- IEEE

  Use of standard library.

Use Clauses

- IEEE.STD_LOGIC_1164.all

  Use of standard logic arguments.

Ports

- rstn **in STD_LOGIC**

  Global reset, active low.
- clk **in STD_LOGIC**

  Global clock running on 100 MHz to provide maximum accuracy for the sampling clock.
- clk50MHz **in STD_LOGIC**

  Global LEON clock running at 50 MHz for everything else.
- data **in STD_LOGIC_VECTOR(** 15 **downto** 0 **)**

  Data in from the softcore output.
- addr **in STD_LOGIC_VECTOR(** 6 **downto** 0 **)**

  Address from the softcore.
- write **in STD_LOGIC**

  Signal indicating the buffers to read values.
- sampleclk **out STD_LOGIC**

  The sampleclock running at 705.6 kHz, used by ADC for oversampling.
- sampleclk44khz **out STD_LOGIC**

  A sampleclock indicating a new sample were to be read to/from buffers.
- sclk **out STD_LOGIC**

  A clock for the off chip DAC chip.
- din **out std_logic**

  The output sample in serial to the off chip DAC.
- nSync **out STD_LOGIC**

  The output sync to the off chip DAC.
- index_reset **in STD_logic**

  A signal to reset the memory counter.

D.22.1 Detailed Description

This component is a wrapper for the parts needed for the digital to analog converter. Its main functionality is to provide for internal connections between the components and to throughput any signals to the next level in the hierarchy.

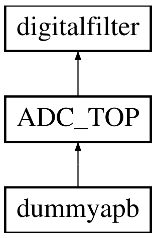
The documentation for this class was generated from the following file:

- DAC_TOP.vhd

D.23 digitalfilter Entity Reference

This module implements a digital FIR filter. When the start port goes from low to high the filter will shift a storage vector and sample the current input value. The filter than multiplies and accumulates once every clock cycle until the calculations are done. When this happened the calculated value is outputted and the finished port will be set.

Inheritance diagram for digitalfilter:



Entities

- FIR_filter architecture

  digitalfilter

Libraries

- IEEE

  Use of standard library.

Use Clauses

- IEEE.STD_LOGIC_1164.all

  Use of standard logic arguments.
- IEEE.NUMERIC_STD.all

  Use of standard numerical arguments.

Generics

- WIDTH **INTEGER:=** 8

  Width decides the bit width of the filter.
- N **INTEGER:=** 4

  N decides the number of taps of the filter.

Ports

- reset **in STD_LOGIC**

  reset, active low
- start **in STD_LOGIC**

  start indicates a new value is available, starting the calculations in the filter
- clk **in STD_LOGIC**

  clock for the filter operations
- x **in STD_LOGIC_VECTOR(WIDTH - 1 downto 0)**

  x is the input of the filter
- y **out STD_LOGIC_VECTOR(31 downto 0)**

  y is the output of the filter
- finished **out STD_LOGIC**

  finished indicates the filter calculations are done

D.23.1 Detailed Description

This module implements a digital FIR filter. When the start port goes from low to high the filter will shift a storage vector and sample the current input value. The filter than multiplies and accumulates once every clock cycle until the calculations are done. When this happened the calculated value is outputted and the finished port will be set.

The documentation for this class was generated from the following file:

- digitalfilter.vhd

D.24 diode Architecture Reference

Architecture of the RGB_diode.

Processes

- PROCESS_8(**rstn** , **clk**)

Signals

- diode_duty_counter **integerrange 0 toN**

  A counter for the duty cycle.
- diode_enable **STD_LOGIC_vector(2 downto 0)**

  A enable to indicate if a diode should be lit.

D.24.1 Detailed Description

Architecture of the RGB_diode.

The RGB_control changed the RGB diodes color depending on the state of the input is_working. Depending of the generic N the brightness of the diode can also be controlled as an N of 0 proved to bright

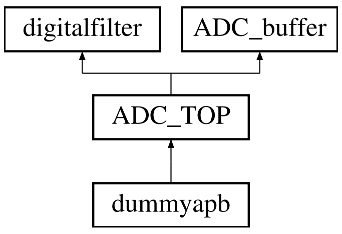
The documentation for this class was generated from the following file:

- RGB_diode_controller.vhd

D.25 dummyapb Entity Reference

This component gathers all the sub-modules needed for communication with ADC and DAC. It also supplies an interface to the APB bus.

Inheritance diagram for dummyapb:



Entities

- APB_interface architecture

  Architecture of the Dummy_apb.

Libraries

- IEEE

  Use of standard library.
- grlib

  use of the GRLIB

Use Clauses

- IEEE.STD_LOGIC_1164.all

  Use of standard logic arguments.
- grlib.amba.all

  Use of the AMBA bus signals and constants.
- grlib.stdlib.all

  Use of standard GRLIB signals and constants.
- grlib.devices.all

  use of GRLIB devices signals and constants

Generics

- pindex **integer:= 0**

  Slave index.
- paddr **integer:= 0**

  Address of the APB bank.
- pmask **integer:= 16#fff#**

  Address range.

Ports

- rstn **in std_ulogic**

  Global reset, active low.
- clk **in std_ulogic**

  Clock at the bus speed of 50 MHz.
- clk100 **in std_ulogic**

  Clock at 100 MHz for certain speed dependent modules.
- vauxp3 **in STD_LOGIC**

  XADC related signal.
- vauxn3 **in STD_LOGIC**

  XADC related signal.
- apbi **in apb_slv_in_type**

  APB slave inputs.
- apbo **out apb_slv_out_type**

  APB slave outputs.
- spiSclk **out std_logic**

  Serial clock for DAC SPI.

- spiDin **out std_logic**

  Data signal for DAC SPI.
- spiNsync **out std_logic**

  Sync signal for DAC SPI.

D.25.1 Detailed Description

This component gathers all the sub-modules needed for communication with ADC and DAC. It also supplies an interface to the APB bus.

The documentation for this class was generated from the following file:

- dummyapb.vhd

D.26 filter_bounce Architecture Reference

Architecture of the bounce_filter.

Processes

- PROCESS_2(**rstn** , **clk**)

Signals

- cnt **unsigned(counterbits - 1 downto 0)**

  A counter signal, the size is defined by the generic.
- last_button_in **std_logic**

  A signal to indicate the last state of the button in.
- button_out_buff **STD_LOGIC**

  A buffered value of the output to determine if the output changes from low to high or high to low.

D.26.1 Detailed Description

Architecture of the bounce_filter.

This component stabilizes an input signal using a counter. This counter can be set with the generic. If the signal changed during the counting the counter resets to 0.

The documentation for this class was generated from the following file:

- bounce_filter.vhd

D.27 FIR_filter Architecture Reference

digitalfilter

Processes

- PROCESS_7(**reset** , **clk**)

  The filter parameters.

Constants

- FILTER_PARAMETERS **parameter_array_type:=(x " 0041b1f5 ",x " 00462ce8 ",x " 00693e0c ",x " 00958f2e ",x " 00cbd646 ",x " 010c8add ",x " 0157b719 ",x " 01acf299 ",x " 020b3c83 ",x " 0270fe9f ",x " 02dbf43d ",x " 03492ca7 ",x " 03b4f47c ",x " 041af631 ",x " 047641dc ",x " 04c16e65 ",x " 04f69e6e ",x " 050fc1f9 ",x " 0506d3c5 ",x " 04d60dd4 ",x " 0477ff3a ",x " 03e81aa5 ",x " 0322e43a ",x " 0225ed51 ",x " 00f0a718 ",x " ff841059 ",x " fde34b3b ",x " fc13990e ",x " fa1c76b5 ",x " f807b848 ",x " f5e16719 ",x " f3b7b517 ",x " f19ab76c ",x " ef9c3525 ",x " edcf338b ",x " ec479930 ",x " eb199d89 ",x " ea5952f6 ",x " ea19f88c ",x " ea6d6b27 ",x " eb6387d1 ",x " ed099a31 ",x " ef69b5fa ",x " f28a5e48 ",x " f66df637 ",x " fb127d21 ",x " 00715beb ",x " 067f373a ",x " 0d2c0bab ",x " 1463500e ",x " 1c0c3948 ",x " 240a3bae ",x " 2c3d832f ",x " 3483b0e0 ",x " 3cb88747 ",x " 44b6d486 ",x " 4c593e4d ",x " 537b3d7c ",x " 59f9f153 ",x " 5fb51708 ",x " 648fc927 ",x " 68714a0b ",x " 6b45a707 ",x " 6cfe3e28 ",x " 6d9218ce ",x " 6cfe3e28 ",x " 6b45a707 ",x " 68714a0b ",x " 648fc927 ",x " 5fb51708 ",x " 59f9f153 ",x " 537b3d7c ",x " 4c593e4d ",x " 44b6d486 ",x " 3cb88747 ",x " 3483b0e0 ",x " 2c3d832f ",x " 240a3bae ",x " 1c0c3948 ",x " 1463500e ",x " 0d2c0bab ",x " 067f373a ",x " 00715beb ",x " fb127d21 ",x " f66df637 ",x " f28a5e48 ",x " ef69b5fa ",x " ed099a31 ",x " eb6387d1 ",x " ea6d6b27 ",x " ea19f88c ",x " ea5952f6 ",x " eb199d89 ",x " ec479930 ",x " edcf338b ",x " ef9c3525 ",x " f19ab76c ",x " f3b7b517 ",x " f5e16719 ",x " f807b848 ",x " fa1c76b5 ",x " fc13990e ",x " fde34b3b ",x " ff841059 ",x " 00f0a718 ",x " 0225ed51 ",x " 0322e43a ",x " 03e81aa5 ",x " 0477ff3a ",x " 04d60dd4 ",x " 0506d3c5 ",x " 050fc1f9 ",x " 04f69e6e ",x " 04c16e65 ",x " 047641dc ",x " 041af631 ",x " 03b4f47c ",x " 03492ca7 ",x " 02dbf43d ",x " 0270fe9f ",x " 020b3c83 ",x " 01acf299 ",x " 0157b719 ",x " 010c8add ",x " 00cbd646 ",x " 00958f2e ",x " 00693e0c ",x " 00462ce8 ",x " 0041b1f5 ")**

Types

- **signal_array_typeisarray(0 toN - 1)ofstd_logic_vector(WIDTH - 1 downto 0)**

  An array type for storing the latest values of the input.
- **multi_outisarray(0 toN - 1)ofstd_logic_vector(2 ∗WIDTH - 1 downto 0**

  An array type for storing the values during calculation.
- **parameter_array_typeisarray(0 toN - 1)ofsigned(WIDTH - 1 downto 0)**

Signals

- i **naturalrange 0 toN**

  This signals indicates where in the calculation we are.
- last_start **std_logic**

  This signal indicates the state of start one clock cycle ago.
- x_array **signal_array_type**

  A signal storing previous values of the input.
- y_array **std_logic_vector(2 ∗WIDTH - 1 downto 0)**

  A signal for storing the values during calculation.

D.27.1 Detailed Description

digitalfilter

The architecture containing the main body of the component.

The documentation for this class was generated from the following file:

- digitalfilter.vhd

D.28 HID_wrapper Architecture Reference

Architecture of the Dummy_apb.

Processes

- regs(**clk** , **rstn**)

Components

- button_control
- seven_seg_control
- RGB_diode_controller

Constants

- pconfig **apb_config_type:=(0 =>ahb_device_reg(VENDOR_GROUP,OWN_BTN, 0 , 0 , 0), 1 =>apb_-iobar(paddr ,pmask))**

Signals

- current_preset **STD_LOGIC_VECTOR(7 downto 0)**

  The current and selected preset hold the value of these signals.
- selected_preset **STD_LOGIC_VECTOR(7 downto 0)**

  The current and selected preset hold the value of these signals.
- irq_read **STD_LOGIC**

  The irq_read and irq_write is the generated interrupt for read and write respectively. This signal comes from the button_control.
- irq_write **STD_LOGIC**

  The irq_read and irq_write is the generated interrupt for read and write respectively. This signal comes from the button_control.
- read_interupt **STD_LOGIC**

  Buffered interrupts to prevent the interrupts from being high for more than one clock cycle.
- write_interupt **STD_LOGIC**

  Buffered interrupts to prevent the interrupts from being high for more than one clock cycle.
- is_working **STD_LOGIC**

  Bit indicating if the diode should be red or green.

Instantiations

- inst_button **button_control**

  This vector configs the address ranging and start address.
- inst_seven_seg **seven_seg_control**
- inst_rgb_diode_controller **RGB_diode_controller**
- bootmsg **report_version**

D.28.1 Detailed Description

Architecture of the Dummy_apb.

The Dummy APB creates an interface between the APB and the HID. This is done in the simplest way possible. A read from any address to this module will result in the 8 LSB being the selected preset. A write to any address will result in the RGB diode getting a command to either show green or red depending on the value of the LSB.

The documentation for this class was generated from the following files:

- Button_and_hex_wrapper.vhd

D.29 LUT Architecture Reference

Architecture of the BCD_block.

D.29.1 Detailed Description

Architecture of the BCD_block.

This component uses a lookup used in the conversion of binary to binary coded decimal. An input value above 9 is invalid and returns don't care.

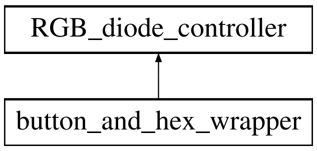
The documentation for this class was generated from the following file:

- BCD_block.vhd

D.30 RGB_diode_controller Entity Reference

The RGB_control controls on of the onboard diodes. Depending on the input of the is_working the diode will either be green or red.

Inheritance diagram for RGB_diode_controller:



Entities

- diode architecture

  Architecture of the RGB_diode.

Libraries

- IEEE

  Use of standard library.

Use Clauses

- IEEE.STD_LOGIC_1164.all

  Use of standard logic arguments.
- IEEE.NUMERIC_STD.all

  Use of standard numerical arguments.

Generics

- N **integer**:= 1

  Generic to decide the duty cycle of the diode. Duty cycle = 1/(N+1)

Ports

- clk **in STD_LOGIC**

  Clock in for calculation of duty cycle.
- rstn **in STD_LOGIC**

  Global reset, active low.
- diode_out **out STD_LOGIC_VECTOR(2 downto 0)**

  Diode_out is a vector containing the state of the red, green and blue diode.
- is_working **in STD_LOGIC**

  Is_working decides if the diode is to be green or red.

D.30.1 Detailed Description

The RGB_control controls on of the onboard diodes. Depending on the input of the is_working the diode will either be green or red.

The documentation for this class was generated from the following file:

- RGB_diode_controller.vhd

D.31 seven_crtl Architecture Reference

Architecture of the seven_seg_control.

Processes

- PROCESS_9(**rstn** , **clk**)

Components

- clk_div_seven_seg
- bin2bcd

Signals

- cnt_hex_display **integerrange 0 to 7**

  A counter to decide what seven segment to be active.
- slow_clk **STD_LOGIC**

  A clock for toggling between the seven segment displays.
- value_vector **STD_LOGIC_VECTOR(31 downto 0)**

  This vector concatenates the current and selected value.
- current_number **integerrange 0 to 9**

  This signal indicated the current number to be outputted on the seven segment selected.

Instantiations

- bin_2_bcd_inst_current **bin2bcd**
- bin_2_bcd_inst_selected **bin2bcd**
- clk_div **clk_div_seven_seg**

D.31.1 Detailed Description

Architecture of the seven_seg_control.

The seven_seg_control controls the output of the seven segment displays. To do this the modules uses clk_div_-seven_seg to divide the clock to a suitable speed to the seven_seg_sel to switch. This module also uses bin2bcd to convert the numbers from binary to BCD. The BCD is then converted to seven segment numbers and outputted on the seven_seg_out.

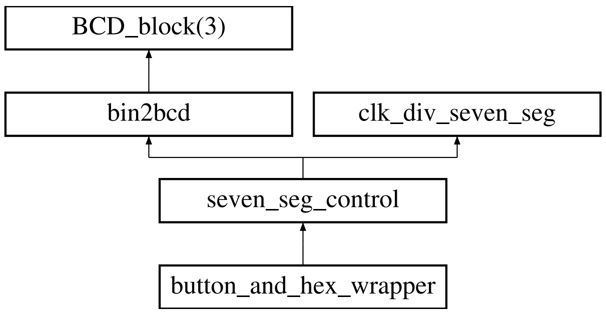
The documentation for this class was generated from the following files:

- seven_seg_control.vhd

D.32 seven_seg_control Entity Reference

The seven segment display takes two 8 bit integers in and outputs these on an 8 digit seven segment display.

Inheritance diagram for seven_seg_control:



Entities

- seven_crtl architecture

  Architecture of the seven_seg_control.

Libraries

- IEEE

  Use of standard library.

Use Clauses

- IEEE.STD_LOGIC_1164.all

  Use of standard logic arguments.
- IEEE.NUMERIC_STD.all

  Use of standard numerical arguments.

Ports

- clk **in STD_LOGIC**

  Input clock for registers and the clock divider.
- rstn **in STD_LOGIC**

  Global reset, active low.

- current_preset **in STD_LOGIC_VECTOR(7 downto 0)**

  Current_preset marks the preset to be selected.
- selected_preset **in STD_LOGIC_VECTOR(7 downto 0)**

  Selected_preset marks the current selected preset.
- seven_seg_out **out STD_LOGIC_VECTOR(6 downto 0)**

  Seven_sel_out marks the seven segment display of the current selected seven segment display segment.
- seven_seg_sel **out STD_LOGIC_VECTOR(7 downto 0)**

  Seven_seg_sel marks the current selected seven segment display.

D.32.1 Detailed Description

The seven segment display takes two 8 bit integers in and outputs these on an 8 digit seven segment display.

The documentation for this class was generated from the following file:

- seven_seg_control.vhd

D.33 SPI Architecture Reference

Architecture of the DAC_SPI.

Processes

- dataOut(**rstn** , **clk**)

Signals

- dataCounter **integerrange 0 to 25**

  This signal keeps track on what bit to send over the SPI.
- configBits **STD_LOGIC_VECTOR(7 downto 0)**

  Bits containing configuration information for the DAC.
- lastsampleclk **STD_LOGIC**

  Bit storing the last value of the sampleclk.
- databuff **STD_LOGIC_vector(15 downto 0)**

  Bit sampling the input data to make sure it is not changed during transmission.

D.33.1 Detailed Description

Architecture of the DAC_SPI.

The architecture containing the main body of the component.

The documentation for this class was generated from the following file:

- DAC_SPI.vhd

D.34 TOP_ADC Architecture Reference

ADC_TOP.

Processes

- PROCESS_1(**CLK100** , **RST**)

Components

- digitalfilter

  Digital FIR filter.
- ADC

  finished indicates the filter calculations are done
- ADC_buffer

  Buffer for samples.

Signals

- den_in **STD_LOGIC**

  Address indicates what address bufferout should be read from Signal for den_in in the XADC.
- dwe_in **std_logic**

  Signal for the write enable in for the XADC.
- di_in **STD_LOGIC_VECTOR(15 downto 0)**

  Signal for the input vector for the XADC.
- daddr_in **std_LOGIC_vector(6 downto 0)**

  Address in registers.
- inv_rst **std_logic**

  Inversed reset for XADC.
- sampledvalue **STD_LOGIC_VECTOR(15 downto 0)**

  Sampled value from XADC.
- busy **STD_LOGIC**

  Busy signal from XADC.
- lastsampleclk **STD_LOGIC**

  The sample clock delayed one CLK.
- filterin **STD_LOGIC_VECTOR(31 downto 0)**

  The sampled value extended to 32 bits as input to the digital filter.
- filterout **STD_LOGIC_VECTOR(31 downto 0)**

  The output of the digital filter and input of the buffer.

Attributes

- SYN_BLACK_BOX_ADC **BOOLEAN**

  Signaling the ADC is busy sampling.
- SYN_BLACK_BOX_ADC **ADC :componentisTRUE**
- BLACK_BOX_PAD_PIN_ADC **STRING**
- BLACK_BOX_PAD_PIN_ADC **ADC :componentis" di_in [15 : 0] , daddr_in [6 : 0] , den_in , dwe_in , drdy_out , do_out [15 : 0] , dclk_in , reset_in , convst_in , vp_in , vn_in , vauxp3 , vauxn3 , user-_temp_alarm_out , vccint_alarm_out , vccaux_alarm_out , ot_out , channel_out [4 : 0] , eoc_out , alarm_out , eos_out , busy_out "**

Instantiations

- isnt_filter **digitalfilter**

  Address for the XADC register is set to 0x13.
- inst_adc **adc**

  Instantiation of the XADC.
- inst_buffer **ADC_buffer**

D.34.1 Detailed Description

ADC_TOP.

The architecture containing the main body of the component.

D.34.2 Member Data Documentation

D.34.2.1 isnt_filter digitalfilter `[Instantiation]`

Address for the XADC register is set to 0x13.

Input vector is set to 0 as we will never write to the XADC.

The documentation for this class was generated from the following file:

- ADC_TOP.vhd

D.35 TOP_DAC Architecture Reference

Architecture of the DACTOP.

Components

- clk_divide
- DAC_SPI

  The DAC SPI interface takes parallel data and a clock and converts it to serial according to the DAC.
- DAC_buffer

  The DAC buffer is a buffer to store the current processed window.

Signals

- DACin **std_logic_vector(15 downto 0)**

  Signal for the unsigned sample used by the DAC_SPI.
- sBuffOut **std_logic_vector(15 downto 0)**

  Signal for the signed sample outputted by the buffer.
- readBuffer **std_logic**

  Signal indicating the next sample is to be read.
- clk25MHz **STD_LOGIC**

  clock running at 25 MHz used as input for the DAC_SPI;

Instantiations

- inst_clk_divider **clk_divide**
- inst_dac_spi **DAC_SPI**
- inst_dac_buffer **DAC_buffer**

D.35.1 Detailed Description

Architecture of the DACTOP.

The DACtops main purpose is to connect the different sub-blocks. It does also converts the samples from signed to unsigned during the transfer from the buffer.

D.35.2 Member Data Documentation

D.35.2.1 clk_divide `[Component]`

The clock divide components takes a clock and divides it in to: clock/2 clock/4 sample clock sample clock $*$ Oversampling rate

The documentation for this class was generated from the following file:

- DAC_TOP.vhd

File Documentation

D.36 ADC_buffer.vhd File Reference

A buffer for storing samples before they are ready by the softcore.

Entities

- ADC_buffer entity

  This component buffers samples when buff_write goes from low to high, until the buffer is full. When this happens Buffer full is driven high for one clock cycle. The softcore will the read the values from the buffer by changing the address. The buffer will then instantly output the value on this address on the buffout port.
- Buffer_ADC architecture

  Architecture of the ADC buffer.

D.36.1 Detailed Description

A buffer for storing samples before they are ready by the softcore.

D.37 BCD_block.vhd File Reference

A 4bit BIN to BCD lookup table.

Entities

- BCD_block entity

  This module is used in the conversion of binary to binary coded decimals.
- LUT architecture

  Architecture of the BCD_block.

D.37.1 Detailed Description

A 4bit BIN to BCD lookup table.

D.38 bin2bcd.vhd File Reference

An almost generic binary to BCD.

Entities

- bin2bcd entity

  This component calculates the binary coded decimal equivalent of a binary number. This module is not completely generic yet but it is verified to work at the size used in the implementation. For updates check `https://github.com/Jaxc/bin2bcd`.
- converter architecture

  bin2BCD component is a generic binary to binary coded decimal. It does this by using the BCD_block according to the method used in `http://www.johnloomis.org/ece314/notes/devices/binary_to_BCD/bin_to_bcd.html`*. This module is not completely generic yet but it is verified to work at the size used in the implementation. For updates check* `https://github.com/Jaxc/bin2bcd`

D.38.1 Detailed Description

An almost generic binary to BCD.

D.39 bounce_filter.vhd File Reference

The bounce filter stabilizes a bouncing input.

Entities

- bounce_filter entity

  This component stabilized signals by waiting until a signal have been high or low for a set about of time.
- filter_bounce architecture

  Architecture of the bounce_filter.

D.39.1 Detailed Description

The bounce filter stabilizes a bouncing input.

D.40 Button_and_hex_wrapper.vhd File Reference

A wrapper to solve the interfacing between the APB bus and HID.

Entities

- button_and_hex_wrapper entity

  This component gathers all the sub-modules needed for HID. It also supplies an interface to the APB bus.
- HID_wrapper architecture

  Architecture of the Dummy_apb.

D.40.1 Detailed Description

A wrapper to solve the interfacing between the APB bus and HID.

D.41 button_control.vhd File Reference

This module controls the current and selected number, using the input button.

Entities

- button_control entity
- control_button architecture

  Architecture of the button_control.

D.41.1 Detailed Description

This module controls the current and selected number, using the input button.

D.42 clk_div_seven_seg.vhd File Reference

A clock divider for the seven segment display.

Entities

- clk_div_seven_seg entity

  This component takes the system clock divides it for Seven segment displays. This is done by implementing counters.
- clk_div_seven architecture

  Architecture of the clk_div_seven_seg.

D.42.1 Detailed Description

A clock divider for the seven segment display.

D.43 CLK_divide.vhd File Reference

A simple clock divider using counters.

Entities

- clk_divide entity

  This component takes the system clock divides it for ADC/DAC components using lower clocks. This is done by implementing counters. When a certain counter reached a set number it will change the output and reset the counter.
- clk_div_DAC architecture

  Architecture of the clk_divider.

D.43.1 Detailed Description

A simple clock divider using counters.

D.44 DAC_BUFFER.vhd File Reference

A buffer for storing samples from the softcore before they are sent to the DAC.

Entities

- DAC_buffer entity

  This component buffers samples from the softcore at the current address when buffwrite is high. The buffer will then output the values on when the buffRead goes from low to high.
- Buffer_dac architecture

  Architecture of the DAC buffer.

D.44.1 Detailed Description

A buffer for storing samples from the softcore before they are sent to the DAC.

D.45 DAC_SPI.vhd File Reference

A buffer for storing samples from the softcore before they are sent to the DAC.

Entities

- DAC_SPI entity

  The DAC SPI interface converts parallel data from the data port and transforms it to a SPI to be sent to the external DAC chip on the din port. The module also adds flag bits to this signal, as well as a chip select signal called n_sync. The interface listens to the sample clock and only transmits a new message when this clock goes from low to high.
- SPI architecture

  Architecture of the DAC_SPI.

D.45.1 Detailed Description

A buffer for storing samples from the softcore before they are sent to the DAC.

D.46 DAC_TOP.vhd File Reference

A top file to instantiate the modules used to the DAC. The components instantiated in this file are the the clk_divide, DAC_SPI and the DAC_buffer.

Entities

- dac_Top entity

  This component is a wrapper for the parts needed for the digital to analog converter. Its main functionality is to provide for internal connections between the components and to throughput any signals to the next level in the hierarchy.
- TOP_DAC architecture

  Architecture of the DACTOP.

D.46.1 Detailed Description

A top file to instantiate the modules used to the DAC. The components instantiated in this file are the the clk_divide, DAC_SPI and the DAC_buffer.

D.47 digitalfilter.vhd File Reference

A buffer for storing samples before they are ready by the softcore.

Entities

- digitalfilter entity

  This module implements a digital FIR filter. When the start port goes from low to high the filter will shift a storage vector and sample the current input value. The filter than multiplies and accumulates once every clock cycle until the calculations are done. When this happened the calculated value is outputted and the finished port will be set.
- FIR_filter architecture

  digitalfilter

D.47.1 Detailed Description

A buffer for storing samples before they are ready by the softcore.

D.48 dummyapb.vhd File Reference

A wrapper to solve the interfacing between the APB bus and the ADC_TOP and DACTOP.

Entities

- dummyapb entity

  This component gathers all the sub-modules needed for communication with ADC and DAC. It also supplies an interface to the APB bus.
- APB_interface architecture

  Architecture of the Dummy_apb.

D.48.1 Detailed Description

A wrapper to solve the interfacing between the APB bus and the ADC_TOP and DACTOP.

D.49 RGB_diode_controller.vhd File Reference

This unit controls the color and strength of the RGB.

Entities

- RGB_diode_controller entity

  The RGB_control controls on of the onboard diodes. Depending on the input of the is_working the diode will either be green or red.
- diode architecture

  Architecture of the RGB_diode.

D.49.1 Detailed Description

This unit controls the color and strength of the RGB.

D.50 seven_seg_control.vhd File Reference

An output interface for the seven segment displays.

Entities

- seven_seg_control entity

  The seven segment display takes two 8 bit integers in and outputs these on an 8 digit seven segment display.
- seven_crtl architecture

  Architecture of the seven_seg_control.

D.50.1 Detailed Description

An output interface for the seven segment displays.

Index