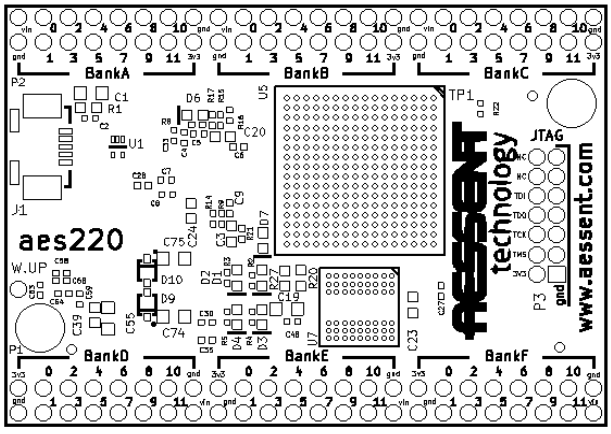
Aessent Technology Ltd

aes220 High-Speed USB FPGA Mini-Module

User Manual Version 1.4.2



Change History

Version	Changes	Author
V1.2	Original Version	Sébastien Saury
V1.4	Update pipes interface (introduction of PAUSE_in input)	Sébastien Saury
V1.4.1	Added note to clarify pipe direction convention and diagram of test bench overview	Sébastien Saury
V1.4.2	Corrected typo in table 6. Modified documentation pertaining to pipe in and out signals and behaviour since moving to VHDL library version 1.3 which uses a valid/ready interface.	Sébastien Saury

Table of Contents

Illustration Index

Index of Tables

1 Overview

1.1 Description

The aes220 is a high-speed USB 2.0 (480Mb/s) FPGA module for rapid prototyping and incorporation in other systems. It combines a Cypress CYC7C68013A FX2LP High-Speed USB 2.0 micro-controller with a Xilinx Spartan 3AN device (XC3S200AN) connected to a 128Mb SDRAM. The module includes its own DC/DC converter and crystal oscillator. All it requires is a 5Vdc power supply if disconnected from the USB. Its tiny 43x61mm size allows fitting in confined spaces and thanks to its stackable interface it can easily be combined with other modules.

The device small size does not prevent it from offering a total of 72 General Purpose Inputs Outputs (GPIO) 12 of which can be used as clock inputs fed directly to the FPGA Global Clocks Buffers (GCLK). The 36 GPIO on connector P1, including the clocks, can be used single ended or as 18 differential pairs.

The two 48 pin 0.1" pitch through hole connectors footprints on either side of the module allow for it to be stacked up either on top, bottom or both sides.

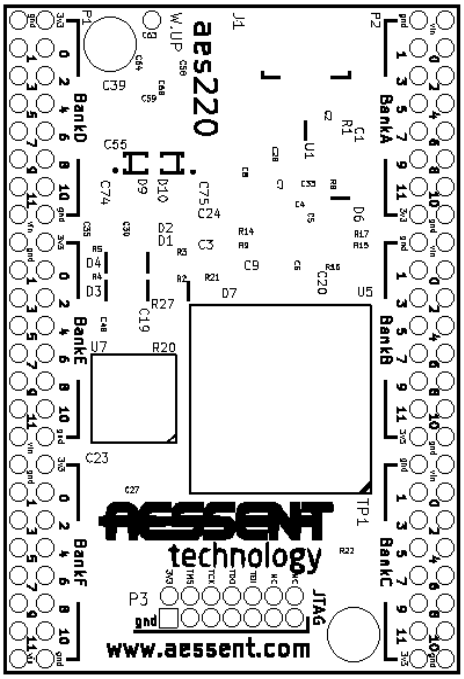


Figure 1: Module Top View

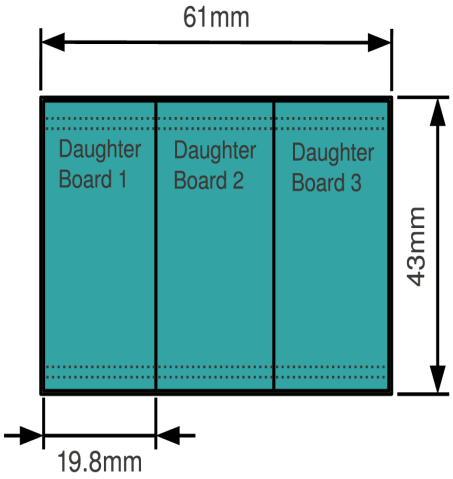


Figure 2: Daughter Boards Stack Up

To add to the flexible GPIO scheme power and grounds pins are spread symmetrically among the two main connectors. This feature allows for modules a third of its size to be directly interfaced to the aes220. Each slot hence connecting to power supply pins and interfacing with 24 GPIO, 4 of which usable as clock signals to the FPGA.

The module offers a flexible clock implementation with the micro-controller clock connected to the FPGA and 16 GCLK inputs present on the external connectors allowing for multiple clock inputs (for a 10MHz external reference for example, but also for running buses or serial links requiring an external clock). The FPGA possesses four Digital Frequency Synthesiser (DFS) to multiply any clock provided up to 334MHz.

Thanks to the FPGA 4Mb of internal Flash memory no external device is required for the FPGA to keep its configuration over power cycles. The integrated flash memory can be used for device configuration, with enough space for two different configurations, or data storage or a mix of both. In addition the SDRAM device provides 128Mb of RAM memory with fast access time (16bits parallel data bus and 100MHz clock). See the Xilinx web site for more documentation on the FPGA itself.

The Cypress FX2LP micro-controller provides the communication between the host computer and the FPGA. The RAM and EEPROM of the micro-controller are fully accessible allowing customisation of the communication link. However the communication channel between the PC, micro-controller and FPGA is fully set-up requiring no intervention from the user.

Both the micro-controller and the FPGA can be configured using a simple USB cable (or also using a JTAG programmer via standard Xilinx JTAG connector in the case of the FPGA). VHDL or Verilog code synthesising for the FPGA is via Xilinx free ISE WebPACK or ISE Design Suite both downloadable from the Xilinx website.

There is no need for learning about USB communication protocols as libraries are provided for dealing with the communication between the host PC, the micro-controller and the FPGA. All programs and libraries are free and Open Source so as not to be tied up with any proprietary tools and permitting customisation if required. Examples, tools and libraries are all available on both Windows and Linux platforms.

1.2Block Diagram

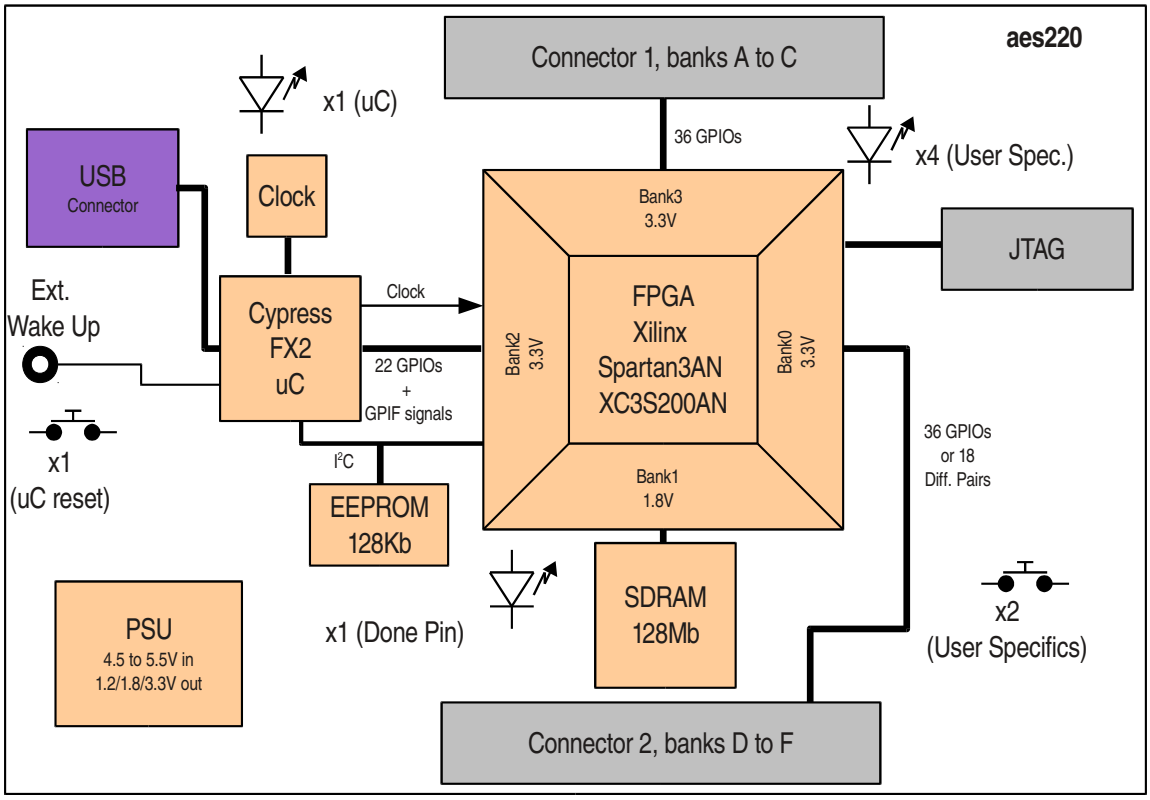


Figure 3: aes220 Block Diagram

2 Operation

2.1 Note to the user

The operation described in this chapter are for the software, firmware and VHDL code provided with the module. As these are open source and freely modifiable by the user some part of this manual could stop being relevant and the onus would be on the user to document its own work. The code and compiled modules required to set the module in its original condition are available on the website at www.aessent.com .

2.2 Power Supplies

The module can be powered up directly from the USB power supply or from an external 4.5 to 5.5V supply provided via one or more of the six Vin pins available on the two main connectors P1 and P2.

The two power supply paths (i.e. USB and external supply paths) can be swapped from one to the other without disrupting the functioning of the module. When an external supply is detected the supply from the USB port is turned off.

By specification a USB 2.0 host can provide up to 500mA of current (1A if using a double connector cable). As depending on the application a module can draw more than that it is the responsibility of the user to ensure its application won't exceed the USB specification (using Xilinx power estimator tools for example). If it is the case a 5V external power supply via P1 or P2 must be used.

The on-board power supply converter can provide up to 1.5A on the 3.3V rail used by the inputs/outputs of the FPGA. On the unlikely event that more power is required it is possible to turn off the on-board power supply and provide the 3.3V directly to the FPGA, please refer to the API on how to turn off the 3.3V rail. Note, however, that for this amount of power the FPGA will require some form of cooling to be provided. See table 1 below for the different supplies capabilities.

Note: it might also be necessary to turn of the 3.3V rail if stacking up boards together. Only one of the boards should provide the 3.3V to the rail (or none if an external 3.3V power supply is used).

Power Supply	Capability	Demand	Client Device
Switching 3.3V	1.5A	Design dependent* (max 4A!)	FPGA I/O Bank 0,2,3
Switching 1.2V	1.5A	Design dependent* (max 600mA)	FPGA Core
Linear 3.3V	300mA	Design dependent* (max 150mA FPGA + 85mA uC)	FPGA Aux, micro-controller
Linear 1.8V	300mA	70mA SDRAM + FPGA Bank 1 activity	SDRAM, FPGA Bank 1 I/O

Table 1: Power Supply Capabilities

* Power dissipation will limit how much current can be drawn to something lower than the power supply limits.

2.3 Start-Up and Shut-Down

The module powers up automatically when any of the power supply is present. No special step is required. The micro-controller will wake up first and only then allow the FPGA to configure from its internal flash (if programmed).

Shutting-down the module is a matter of cutting off the power supplies to it.

2.4 Reset

There are two ways of resetting the module: via a USB command or by pressing the reset switch on the board. See the API documentation on how to perform an USB reset. The reset, whether software via USB or hardware via the reset switch, resets the micro-controller which in turns resets the FPGA.

2.5 Sleep Mode

Both the micro-controller and FPGA can be put into low-power mode independently of each other. The micro-controller commands the SUSPEND pin of the FPGA. An API function is used to send the relevant command to the micro-controller. Note however that to wake up the micro-controller the W.UP pin on the board has to be pulled low as USB communication is lost when the micro-controller is in low-power mode.

2.6 Programming the FPGA and Micro-controller

Programming both the FPGA and the micro-controller of the aes220 has been made very simple thanks to the aes220 Programmer. It is a GUI based software that makes use of the functions present in the API to program both circuits.

The micro-controller does not require to be programmed as it can be used directly with already loaded firmware. However it is worth keeping an eye for new versions of the firmware published on the website (www.aessent.com). When programming the micro-controller it is possible to either overwrite the program present in the RAM of device or in the EEPROM. If loaded into the RAM the program will revert to the original firmware (or whatever firmware is present at that time in the EEPROM) on a reset or power cycle. If loaded into the EEPROM the program become permanent. Note that if the program loaded in the EEPROM is faulty or does not set the USB communication properly then there is the possibility to “brick” the device. That is the device looses USB communication and hence cannot be reprogrammed. However there is a simple solution to this which is to remove the cap on jumper one and reset or power cycle the device. See the chapter on recovering from a wrongly-programmed EEPROM in the installation manual pertaining to your operating system.

Most Xilinx FPGA do not have intrinsic memory and need to be configured on or after power-up. The device

used on the aes220 however incorporates some Flash memory that can be programmed with the FPGA configuration file. At power on the FPGA will automatically fetch this configuration file from the memory and boot up from it. There is no particular requirement from the configuration file in order to achieve this. The boot up process is handled by the micro-controller. It is not a requirement to program the Flash memory. The device can also be configured and simply run from this configuration until it is powered off or reset at which point the configuration will be lost.

2.7 Communicating with the PC

Communicating between a FPGA and a PC via a USB link can be a tricky matter. However thanks to the API and VHDL interface provided it needs not be. The micro-controller which handles the whole USB communication channel is programmed in such a way as to create a simple yet versatile interface between the FPGA and the PC. No in-depth knowledge of USB communication is required to transfer data back and forth between the two.

The interface allows high-speed data channels between the PC and the FPGA as well as six user general purpose pins which can be used for example as control or status pins for the user application if required.

2.7.1 Overview

A USB communication channel is usually called a pipe. On the PC side sending data out from the PC to the FPGA will be done using the function pipe_Out() and receiving data from the FPGA will be done via the function pipe_In(). The exact syntax for both functions is described in the API documentation but each function requires a buffer of 8 bit unsigned integers, the size of the buffer and a pipe address.

Note: in a USB communication the host is the PC and is the master. There is no interrupt process for the slave to signal to the master it wants to communicate. The host decides when to send and receive data (using pipe_Out and pipe_In functions in this case) as well as how much data is transferred.

On the micro-controller side the data sent by the PC or about to be sent to the PC is stored into dedicated FIFO memories. The FPGA interface provided handles the FIFO communication simplifying further the communication with the PC. The interface is constituted of a main interface module (FX2_Interface) and as many as 128 pipes, although in practice much fewer will be required. Each pipe offers an 8 bit wide data bus with VALID/READY control signals. The user instantiates as many pipes as required in its application and gives them all a unique address. This is depicted in figure 4 below.

Two sort of pipes are provided, PipeIn and PipeOut. PipeIn is used when receiving data into the FPGA and PipeOut when sending data to the host PC.

Note 1: on the FPGA side the pipeIn_ent entity is used to receive data from the PC and the pipeOut_ent entity to send data to the PC. So pipe_out() function of the PC connects to the pipeIn_ent entity of the FPGA and

pipe_in() function of the PC connects to pipeOut_ent entity of the FPGA. The in/out direction is always the one relating to the device being programmed (PC or aes220).

Keep in mind however that the simulation functions provided to write test benches, although written in VHDL and used in the ISE environment, are emulating the PC side of the chain and hence adhere to the PC convention (in is FPGA → PC, out is PC → FPGA). See below.

Note 2: since version 1.3 of the library the pipes are using a VALID/READY type interface. The previous library is still available in the Archives folder for legacy designs.

To simplify the simulation two functions PipeIn and PipeOut are available in the Simulation package provided. The user application can hence be developed and simulated using the signals coming in and out of the pipes and does not have to contend with the handling of the FX2 micro-controller FIFOs. See Figures 4, 5 and 6 for a graphical overview of the different modules.

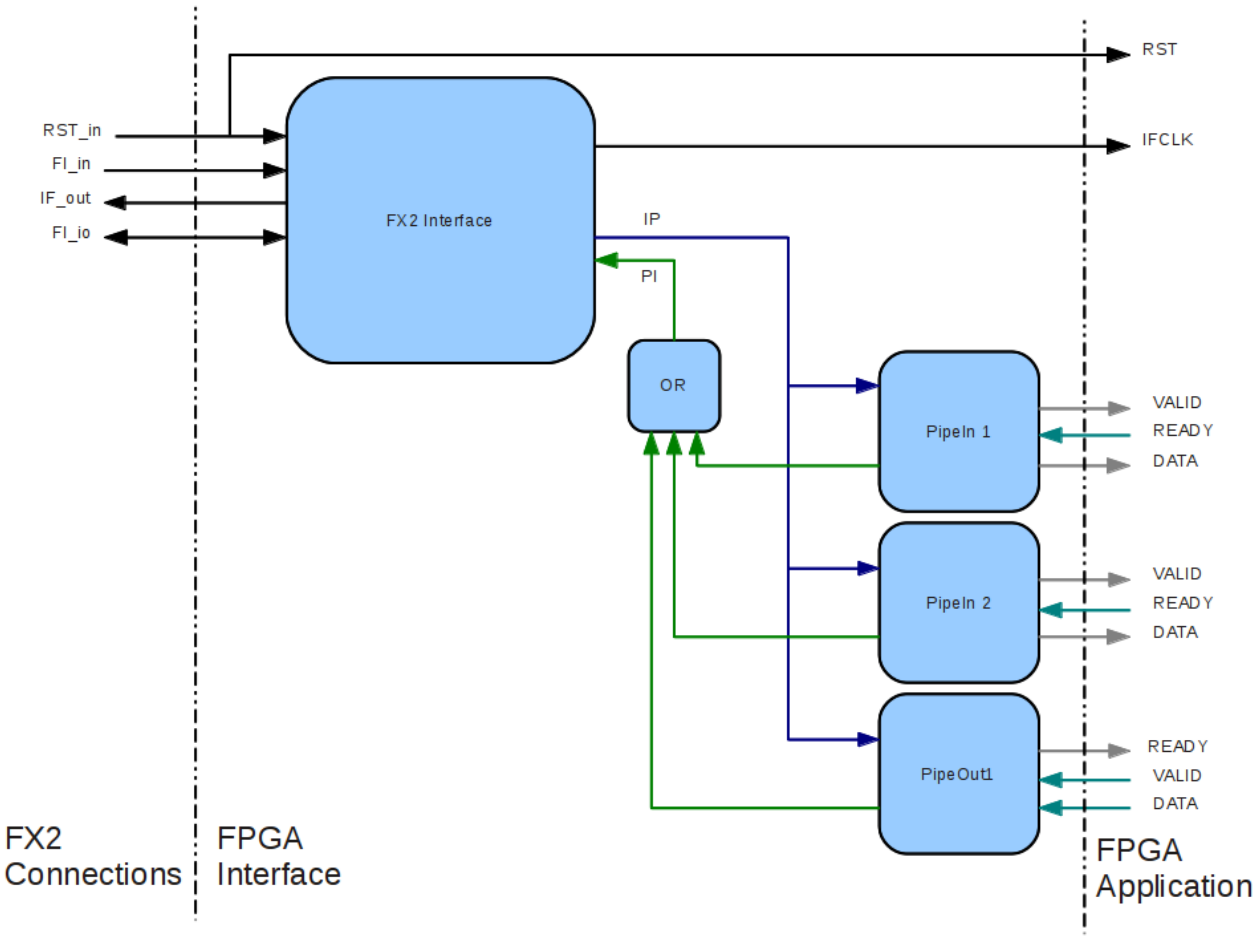


Figure 4: FX2 Interface and Pipes Block Diagram.;

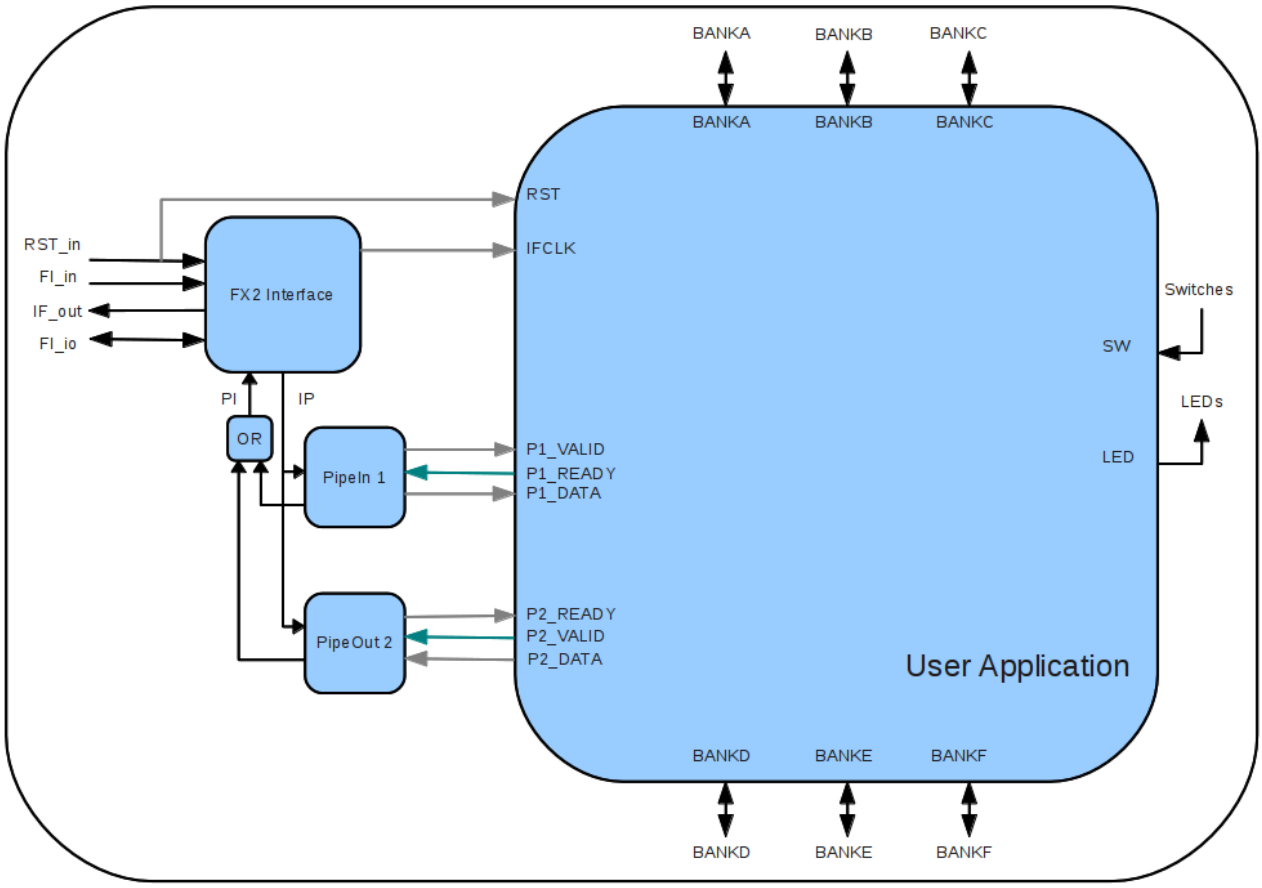


Figure 5: Interface, Pipes and User Application

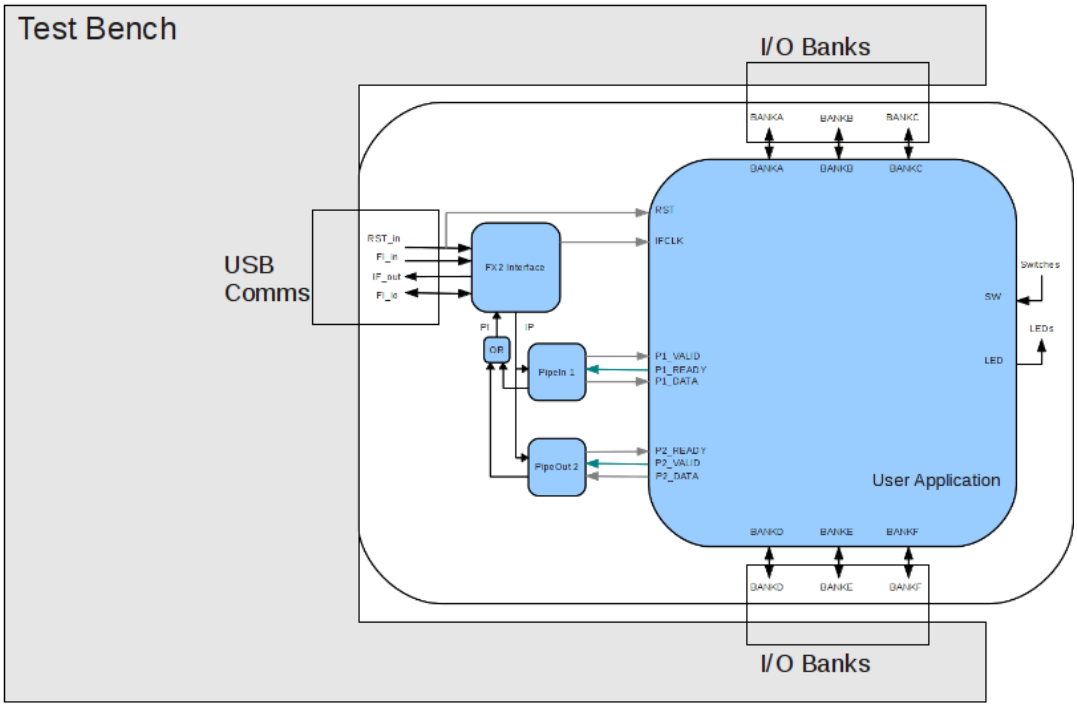


Figure 6: User Application and Test Bench

2.7.2 FX2 Interface

The micro-controller deals with all the USB communications and offers a simplified FIFO interface directly connected to the FPGA. The FX2_Interface joins this interface to the various pipes used in the application. Only one FX2_Interface should be instantiated in the application and all pipes linked to it.

The FX2_interface should be instantiated and mapped as follow:

```
fx2_Interface : aes220_FX2_Interface_ent
  port map (RST_in => RST_in, ifCLK_out => ifClk_s,  -- User app
      FI_in  => FI_in, IF_out => IF_out, FI_io  => FI_io,
      PI_in  => pi_s, IP_out => ip_s);
```

Table 2 describes the various parameters of the instantiation.

Instantiation Parameters	Parameters Description
`fx2_interface : aes220_FX2_Interface_ent`	fx2_interface is an instance of the aes_220_FX2_Interface_ent component (the only one FX2 Interface to be instantiated).
`RST_in => RST_in`	Reset signal coming from the micro-controller (or the host PC). Mapped straight to the top level port of the application. The port is defined in the aes220_RevA1_FX2_Interface.ucf constraint file.
`IFCLK_out => ifclk_s`	The 48MHz clock coming from the micro-controller and transiting via the FX2 Interface. This is the clock signal to use with the rest of the application when dealing with the pipes.
`FI_in => FI_in`	Signal bus going from the FX2 to the FX2 Interface. Mapped straight to the top level port of the application. The port is defined in the aes220_RevA1_FX2_Interface.ucf constraint file.
`IF_out => IF_out`	Signal bus going from the FX2 Interface to the FX2. Mapped straight to the top level port of the application. The port is defined in the aes220_RevA1_FX2_Interface.ucf constraint file.
`FI_io => FI_io`	Bi-directional signal bus between the FX2 and the FX2 Interface. Mapped straight to the top level port of the application. The port is defined in the aes220_RevA1_FX2_Interface.ucf constraint file.
`IP_out => ip_s`	ip_s is the signal linking IP_in port of the pipes to the IP_out port of the interface. ip_s is declared in the aes220_Package.vhd file. ip_s is declared in the aes220_Package.vhd file.
`PI_in => pi_s`	pi_s is the signal linking the PI_out port of the pipes to the PI_in port of the interface. If more than one pipe is used it is the signal coming out of a logical OR between the similar signals coming out of the pipes. pi_s is declared in the aes220_Package.vhd file.

Table 2: aes220_FX2_Interface Parameters

2.7.3 Logical OR

When more than one pipe is used the signals from the pipes to the FX2 Interface need to be tied together using a logical OR before being fed to the interface.

This is done very simply as shown in the following example:

```
-- Logical OR all the signals from the different pipes to the interface:
pi_s <= (pi1_s or pi2_s or pi3_s or pi4_s);
```

Where pi_s is linked directly to the port PI_in of the FX2_Interface instance while pi1_s to pi4_s are linked each to a separate pipe on port PI_out. Note that pi1_s to pi4_s have to be declared in the user application.

Note: Unless there is only one pipe in the whole user application, in which case pi_s can be used directly without a logical OR, there should be one piX_s signal per pipe.

2.7.4 PipeIn: receiving data

On the user application side of the FPGA incoming data is signalled on each pipe by the assertion of the VALID signal. When this signal is asserted the user application must immediately start reading the data synchronously with the provided interface clock (IFCLK) on the DATA bus until the VALID signal is de-asserted. However the user application can pause the transfer by de-asserting the READY signal at any time, including before the transfer has started. The data is considered read by the pipe when both the VALID and READY signal are asserted together. See Figure 7 for a graphical example.

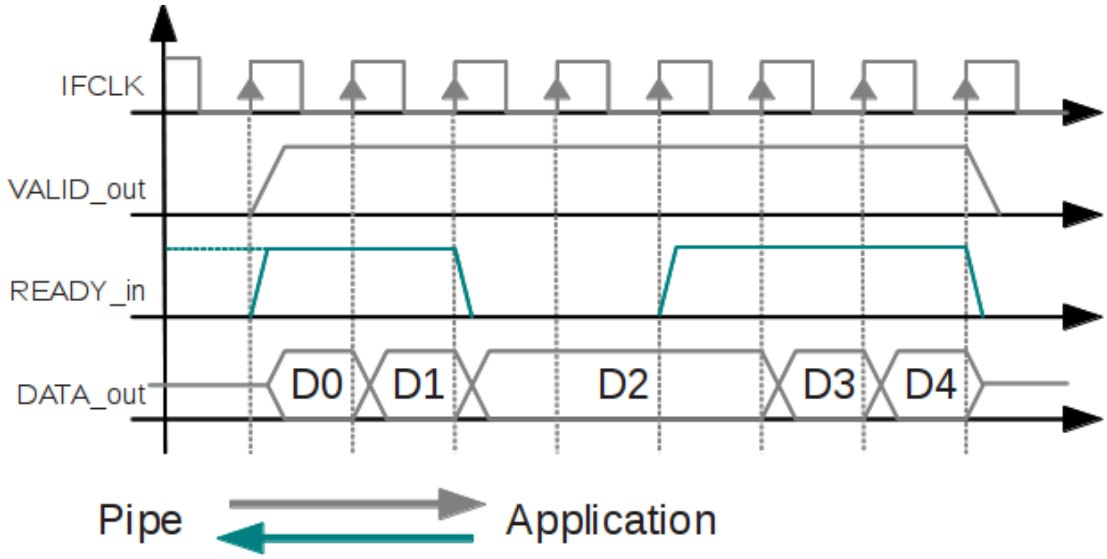


Figure 7: Reading from a pipe

A PipeIn is instantiated using the following declaration (using an instance called Pipe_3_Data as an example):

```
Pipe_3_Data : aes220_PipeIn_ent
  port map (IP_in => ip_s, PI_out => pi3_s, P_ADDR_in => "0000011",
            READY_in => readyP3_s, VALID_out => validP3_s,
            DATA_out => pipe3Data_s);
```

Table 3 describes the various parameters of the instantiation.

Instantiation Parameters	Parameters Description
`Pipe_3_Data : aes220_PipeIn_ent`	Pipe_3_Data is an instance of the aes_220_PipeIn_ent component.
`IP_in => ip_s`	ip_s is the signal linking IP_in port of the pipe to the IP_out port of the interface. ip_s is declared in the aes220_Package.vhd file and hence does not need to be declared in the user application.
`PI_out => pi3_s`	pi3_s is the signal linking the PI_out port of the pipe to the PI_in port of the interface via a logical OR. If only one pipe is used then pi_s, which is declared in the aes220_Package.vhd file, can be used instead. Otherwise pi3_s need to be declared in the user application (10 bits wide signal).
`P_ADDR_in => "0000011"`	Used to set Pipe 3 address (address 3 chosen arbitrarily as an example)
`READY_in => readyP3_s`	The application uses the readyP3_s signal to signal the pipe that the data is accepted. Note that pausing the transfer by not asserting the readyP3_s signal cannot go on forever as it will eventually time out the USB communication with the PC. The time out is currently set to 5s.
`VALID_out => validP3_s`	validP3_s is the signal warning the rest of the application that valid data is present on Pipe_3_Data data bus.
`DATA_out => pipe3Data_s`	Pipe_3_Data data bus (8 bits wide)

Table 3: aes220_PipeIn Parameters

The entity declaration for aes220_PipeIn is as follow:

```
entity aes220_PipeIn_ent is
     Port(
    -- Connections to the FX2 interface
    IP_in     : in  std_logic_vector(18 downto 0);
    PI_out    : out std_logic_vector(9 downto 0);
    -- User application connections
    P_ADDR_in : in std_logic_vector(6 downto 0);  -- The pipe address
                                                  -- set by user app
    READY_in  : in std_logic;      -- User app ready to receive data
    VALID_out : out std_logic;     -- Valid data available to user app
    DATA_out  : out std_logic_vector(7 downto 0)  -- data to user app
    );
end aes220_PipeIn_ent;
```

During simulation the data can be sent to the pipe using the function pipe_out provided in the aes220_SimulationPackage.vhd:

```
pipe_out(outChannel3_v, dataSize_v, dataOut_ar,
         rst_s, fi_s, if_s, ifData_s);
```

Where the signals on the bottom row are linking the function to the interface and are declared in the simulation package. There is no need to change these. The other three parameters in the function are in order:

The pipe address (outChannel3_v)

The number of bytes to be sent (dataSize_v)

The data pre-arranged in a byte array (dataOut_ar)

Note that the types byte and byte_array are also declared in the simulation package as:

```
subtype byte is std_logic_vector(7 downto 0);
type    byte_array is array (natural range <>) of byte;
```

Note: although it is the PipeIn component of the FPGA described here it is linked to the simulation function pipe_out as this function emulates the function used on the host PC to send data to the FPGA.

Note: please have look at the aes220_Loopback_Example for a demonstration on how all these elements interact with each other.

2.7.5 PipeOut: Writing data

When the host (PC) initiates a pipe out event the READY signal is asserted on the addressed pipe. The FPGA user application must start providing the data on the DATA_in bus and signal it is doing so by asserting the VALID signal. Once the data has been provided the pipe releases the READY signal. If the application is providing more data than the FIFO can hold (2048 bytes) the pipe will de-assert the READY signal. When this signal is de-asserted the application must stop providing data until the signal is re-asserted.

The transfer can also be paused from the user application side by de-asserting the VALID input of the pipe. The VALID signal can be asserted before receiving a READY signal. Figure 8 shows an example of writing to a pipe and pausing while the READY signal remains asserted.

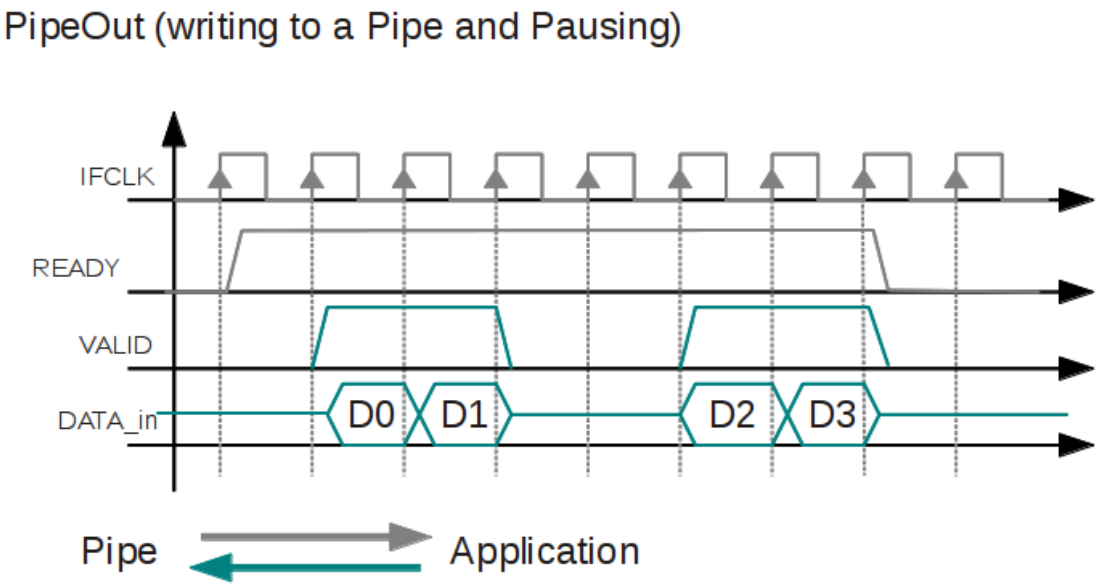


Figure 8: Writing to a pipe

Note: the FX2 micro-controller is using a quad buffered FIFO. What it means is that the FIFO is constituted of four 512 bytes FIFO. When one fills up its contents are automatically sent to the host while the user data is being switched to the second FIFO and so on. If the first FIFO hasn't been emptied by the time the last FIFO is full the READY signal will be de-asserted until the first FIFO becomes available again.

A PipeOut is instantiated using the following declaration (using an instance called Pipe_4_Data as an example):

```
Pipe_4_Data : aes220_PipeOut_ent
  port map (IP_in => ip_s, PI_out => pi4_s, P_ADDR_in => "0000100",
            VALID_in => validP4_s, READY_out => readyP4_s,
            DATA_in => pipe4Data_s);
```

Table 4 describes the various parameters of the instantiation.

Instantiation Parameters	Parameters Description
`Pipe_4_Data : aes220_PipeOut_ent`	Pipe_4_Data is an instance of the aes_220_PipeOut_ent component.
`IP_in => ip_s`	ip_s is the signal linking IP_in port of the pipe to the IP_out port of the interface. ip_s is declared in the aes220_Package.vhd file.
`PI_out => pi4_s`	pi4_s is the signal linking the PI_out port of the pipe to the PI_in port of the interface via a logical OR if more than one pipe is used. If only one pipe is used then pi_s, which is declared in the aes220_Package.vhd file, can be used instead. Otherwise pi4_s need to be declared in the user application (10 bits wide signal).
`P_ADDR_in => "0000100"`	Pipe_4_Data address (address 4 chosen arbitrarily)
`VALID_in => validP4_s`	The application uses the validP4_s signal to let the pipe know when the data on DATA_in is valid. Note that pausing the transfer by de-asserting the validP4_s signal cannot go on forever as it will eventually time out the USB communication with the PC. The time out is currently set to 5s.
`READY_out => readyP4_s`	readyP4_s is the signal warning the rest of the application that Pipe_4_Data is expecting data on its data bus.
`DATA_in => pipe4Data_s`	Pipe_4_Data data bus (8 bits wide)

Table 4: aes220_PipeOut Parameters

The entity declaration for aes220_PipeOut is as follow:

```
entity aes220_PipeOut_ent is
     Port(
    -- Connections to the FX2 interface
    IP_in    : in  std_logic_vector(18 downto 0);
    PI_out   : out std_logic_vector(9 downto 0);
    -- User application connections
    P_ADDR_in   : in std_logic_vector(6 downto 0);  -- The pipe address
                                                    -- set by user app
    VALID_in    : in std_logic;      -- valid input validating the data
    READY_out   : out std_logic;     -- Data ready from user app
    DATA_in     : in  std_logic_vector(7 downto 0)  -- data from user app
    );
end aes220_PipeOut_ent;
```

During simulation the data can be received from the pipe using the function pipe_in provided in the aes220_SimulationPackage.vhd:

```
pipe_in(inChannel_v, dataSize_v, dataIn_ar,
         rst_s, fi_s, if_s, ifData_s);
```

Where the signals on the bottom row are linking the function to the interface and are declared in the simulation package. There is no need to change these. The other three parameters in the function are in order:

The pipe address (inChannel_v)

The number of bytes to be received (dataSize_v)

A pre-arranged byte array to store the data coming in (dataIn_ar)

Note that the types byte and byte_array are also declared in the simulation package as:

```
subtype byte is std_logic_vector(7 downto 0);
type    byte_array is array (natural range <>) of byte;
```

Note: although it is the PipeOut component of the FPGA described here it is linked to the simulation function pipe_in as this function emulates the function used on the host PC to receive data from the FPGA.

Note: please have look at the aes220_Fifo_Example for a demonstration on how all these elements interact with each other.

3 Features

3.1 LED and switches

There are four LED and two switches available to the user connected directly to the FPGA.

The LED are active low (setting the FPGA pins to 0 turns them on), while the switches are normally open and the corresponding FPGA pin reading them tied to ground via a pull down resistor. See Figure 9 for the related schematics.

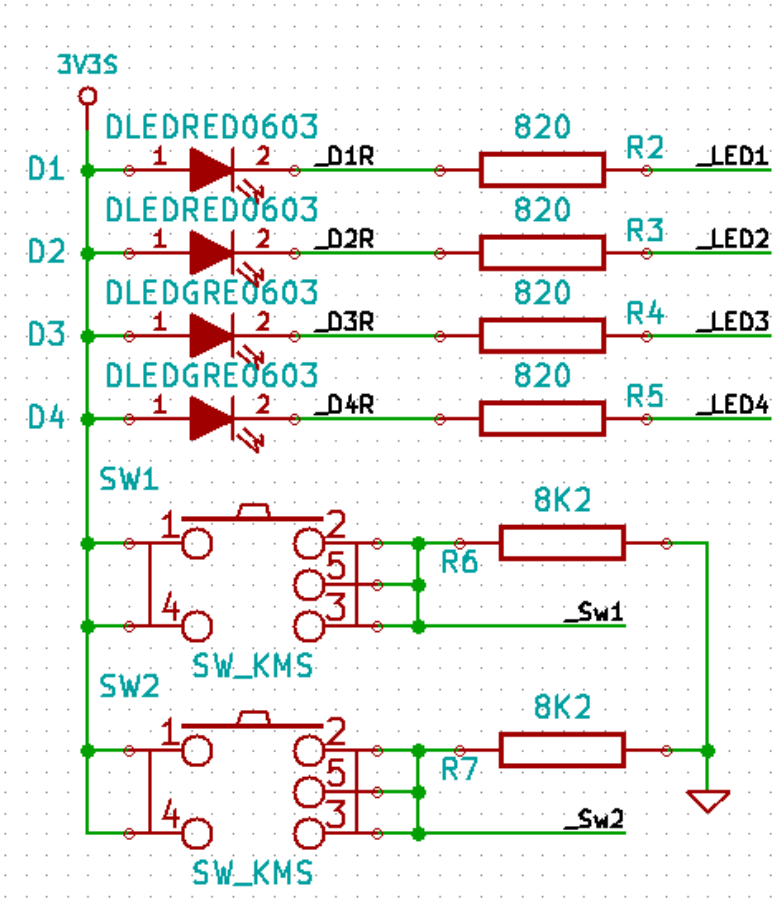


Figure 9: LED and switches schematics

Available on the website (www.aesent.com) is a constraint file provided for the FPGA ISE environment (aes_220_Revxx_LED.ucf) that can be used directly by the user.

3.2

3.3 I²C

The I2C functions of the micro-controller are available to the user. The clock and data line of the bus are connected to the FPGA and can be used by the application within the FPGA or outputted to the outside world. The micro-controller acts as an I2C master. There are three functions provided to the user via the API to communicate with resources attached to the bus: readI2C(), writeI2C() and combinedI2C(). Please see the API documentation for more information.

Note however that both the EEPROM storing the micro-controller program and the on-board DC/DC converter are connected to the bus. Their addresses are:

- EEPROM I2C address: 0x51
- DC/DC converter I2C address: 0x60

These addresses should not be used by any other peripherals connecting to the bus.

3.4 Connectors pin-out

The main user connectors are standard two rows 48 pin 0.1" pitch connectors. Connections to the FPGA have been arranged in six banks of 12 GPIO each including at least two GCLK inputs. They are not to be confused with the FPGA own banks. All the banks on connector P1 belong to the same FPGA bank (bank 3) and all the banks on connector P2 belong to FPGA bank0. The banking arrangement on both connectors is only a naming convention to facilitate board stack-up (see paragraph 3.5 Stack-up format) and does not prevent from using all the banks on one connector as one.

Connector P1

	Funct.	FPGA pin	legend	Connector pin		legend	FPGA pin	Funct.	
BankD		N/A	GND	1	2	3V3	N/A		BankD
		IO_L01N_0	BankD1	3	4	BankD0	IO_L01P_0		
		IO_L03N_0	BankD3	5	6	BankD2	IO_L03P_0		
		IO_L02N_0	BankD5	7	8	BankD4	IO_L02P_0		
		IO_L04N_0	BankD7	9	10	BankD6	IO_L04P_0		
		IO_L06N_0	BankD9	11	12	BankD8	IO_L06P_0		
	GCLK5	IO_L09N_0	BankD11	13	14	BankD10	IO_L09P_0	GCLK4	
		N/A	Vin	15	16	GND	N/A		
BankE		N/A	GND	17	18	3V3	N/A		BankE
		IO_L05N_0	BankE1	19	20	BankE0	IO_L05P_0		
		IO_L07N_0	BankE3	21	22	BankE2	IO_L07P_0		
		IO_L08N_0	BankE5	23	24	BankE4	IO_L08P_0		
		IO_L13N_0	BankE7	25	26	BankE6	IO_L13P_0		
		IO_L16N_0	BankE9	27	28	BankE8	IO_L16P_0		
	GCLK11	IO_L12N_0	BankE11	29	30	BankE10	IO_L12P_0	GCLK10	
		N/A	Vin	31	32	GND	N/A		
BankF		N/A	GND	33	34	3V3	N/A		BankF
	GCLK7	IO_L10N_0	BankF1	35	36	BankF0	IO_L10P_0	GCLK6	
		IO_L15N_0	BankF3	37	38	BankF2	IO_L15P_0		
		IO_L17N_0	BankF5	39	40	BankF4	IO_L17P_0		
		IO_L18N_0	BankF7	41	42	BankF6	IO_L18P_0		
		IO_L19N_0	BankF9	43	44	BankF8	IO_L19P_0		
	GCLK9	IO_L11N_0	BankF11	45	46	BankF10	IO_L11P_0	GCLK8	
		N/A	Vin	47	48	GND	N/A		

Differential pairs

Connectors are standard 0.1" through hole footprint

Figure 10: Connector P1 Mapping

Connector P2

	Funct.	FPGA pin	legend	Connector pin		legend	FPGA pin	Funct.	
BankA		N/A	GND	1	2	Vin	N/A		BankA
	LHCLK7	IO_L15N_3	BankA1	3	4	BankA0	IO_L15P_3	LHCLK6	
		IO_L23N_3	BankA3	5	6	BankA2	IO_L23P_3		
		IO_L22N_3	BankA5	7	8	BankA4	IO_L22P_3		
		IO_L24N_3	BankA7	9	10	BankA6	IO_L24P_3		
		IO_L19N_3	BankA9	11	12	BankA8	IO_L19P_3		
		IO_L18N_3	BankA11	13	14	BankA10	IO_L18P_3		
		N/A	3V3	15	16	GND	N/A		
BankB		N/A	GND	17	18	Vin	N/A		BankB
	LHCLK3	IO_L12N_3	BankB1	19	20	BankB0	IO_L12P_3	LHCLK2	
		IO_L20N_3	BankB3	21	22	BankB2	IO_L20P_3		
		IO_L16N_3	BankB5	23	24	BankB4	IO_L16P_3		
	LHCLK5	IO_L14N_3	BankB7	25	26	BankB6	IO_L14P_3	LHCLK4	
		IO_L09N_3	BankB9	27	28	BankB8	IO_L09P_3		
		IO_L08N_3	BankB11	29	30	BankB10	IO_L08P_3		
		N/A	3V3	31	32	GND	N/A		
BankC		N/A	GND	33	34	Vin	N/A		BankC
	LHCLK1	IO_L11N_3	BankC1	35	36	BankC0	IO_L11P_3	LHCLK0	
		IO_L07N_3	BankC3	37	38	BankC2	IO_L07P_3		
		IO_L05N_3	BankC5	39	40	BankC4	IO_L05P_3		
		IO_L02N_3	BankC7	41	42	BankC6	IO_L02P_3		
		IO_L03N_3	BankC9	43	44	BankC8	IO_L03P_3		
		IO_L01N_3	BankC11	45	46	BankC10	IO_L01P_3		
		N/A	3V3	47	48	GND	N/A		

Connectors are standard 0.1” through hole footprint

Figure 11: Connector P2 Mapping

To facilitate the use of the banks a constraint file (aes220_Revxx_Banks.ucf) for the ISE environment provided on the website (www.aessent.com)

3.5 Stack-up format

The module can be stacked up with other boards/modules very easily thanks to its versatile pin arrangement.

In addition to the 72 GPIO present on the connectors six 3.3Vpins, six Vin pins and twelve ground pins are also provided at regular intervals.

All these pins are grouped into banks allowing an easy division of the board across the way into three smaller sections each able to accept a different daughter board. Each sub-section is therefore provided with 24 GPIO of which at least four can be used as clock inputs, two Vin pins, two 3.3V pins and four ground pins.

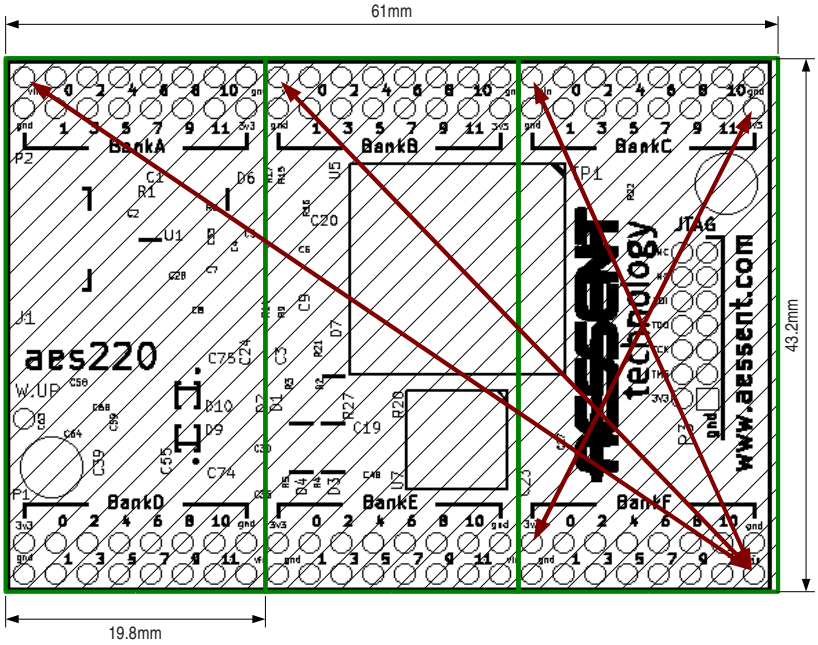


Figure 12: Daughter Boards Stack-Up

Note also that the power and ground pins as well as the clock inputs are symmetrical with regard to the centre of the board or sub-sections. This allows for a daughter board to be plug in either direction without damaging the modules.

4 Characteristics

4.1 FPGA Power Dissipation

4.1.1 FPGA Thermal Characteristics

The following values are for the FTG256 XC3S200AN package (From Xilinx DS557 document).

Thermal resistances	Value	Unit
Junction to case	7.4	°C/Watt
Junction to board	23.3	°C/Watt
Junction to ambient (still air)	29.0	°C/Watt
Junction to ambient (250LFM*)	23.8	°C/Watt
Junction to ambient (500LFM*)	23.0	°C/Watt
Junction to ambient (750LFM*)	22.3	°C/Watt

Table 5: FPGA Thermal Characteristics

* LFM: air velocity in Linear Feet per Minute

4.1.2 FPGA Power Limits

The aes220 is fitted with the industrial grade package option for which the recommended max junction temperature is 100C while its max temperature is 125C. This gives a maximum power dissipation of:

Pmax = (Tj-Ta)/Rja

In still air at 25°C:
Pmax recommended = (100-25)/29=2.75W
Pmax absolute = (125-25)/29 = 3.3W

Note that these figures do not take into account the heat dispersion through the PCB itself so in practice the power dissipation figures should be slightly better.

	Junction to ambient thermal resistance (° C/W)			
	Still air	250LFM	500LFM	750LFM
	29	23.8	23	22.3
Ambient Temp (° C)	Max Recommended Power (W)			
20	2.76	3.36	3.48	3.59
25	2.59	3.15	3.26	3.36
30	2.41	2.94	3.04	3.14
35	2.24	2.73	2.83	2.91
40	2.07	2.52	2.61	2.69
45	1.90	2.31	2.39	2.47
50	1.72	2.10	2.17	2.24
55	1.55	1.89	1.96	2.02
60	1.38	1.68	1.74	1.79
65	1.21	1.47	1.52	1.57
70	1.03	1.26	1.30	1.35
75	0.86	1.05	1.09	1.12
80	0.69	0.84	0.87	0.90
85	0.52	0.63	0.65	0.67
90	0.34	0.42	0.43	0.45
95	0.17	0.21	0.22	0.22
100	0.00	0.00	0.00	0.00

Table 6: Max Recommended Power

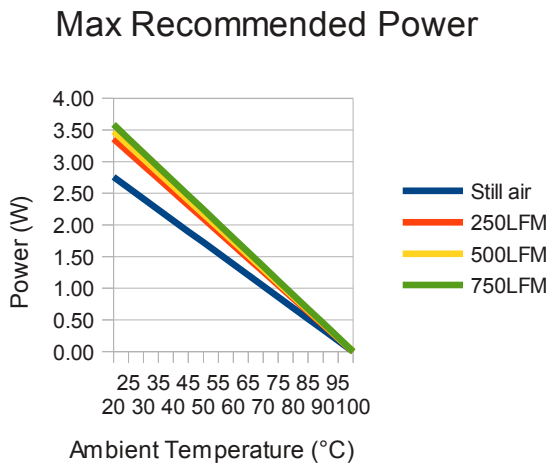


Figure 13: Max Recommended Power

	Junction to ambient thermal resistance (° C/W)			
	Still air	250LFM	500LFM	750LFM
	29	23.8	23	22.3
Ambient Temp (° C)	Absolute Max Power (W)			
20	3.62	4.41	4.57	4.71
25	3.45	4.20	4.35	4.48
30	3.28	3.99	4.13	4.26
35	3.10	3.78	3.91	4.04
40	2.93	3.57	3.70	3.81
45	2.76	3.36	3.48	3.59
50	2.59	3.15	3.26	3.36
55	2.41	2.94	3.04	3.14
60	2.24	2.73	2.83	2.91
65	2.07	2.52	2.61	2.69
70	1.90	2.31	2.39	2.47
75	1.72	2.10	2.17	2.24
80	1.55	1.89	1.96	2.02
85	1.38	1.68	1.74	1.79
90	1.21	1.47	1.52	1.57
95	1.03	1.26	1.30	1.35
100	0.86	1.05	1.09	1.12
105	0.69	0.84	0.87	0.90
110	0.52	0.63	0.65	0.67
115	0.34	0.42	0.43	0.45
120	0.17	0.21	0.22	0.22
125	0.00	0.00	0.00	0.00

Table 7: Absolute Maximum Power

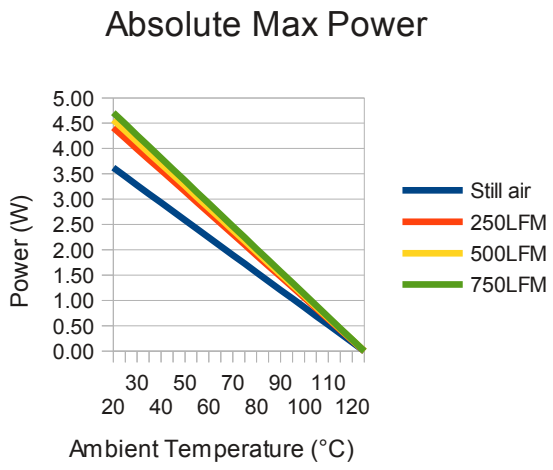


Figure 14: Absolute Maximum Power

4.2 Board Dimensions

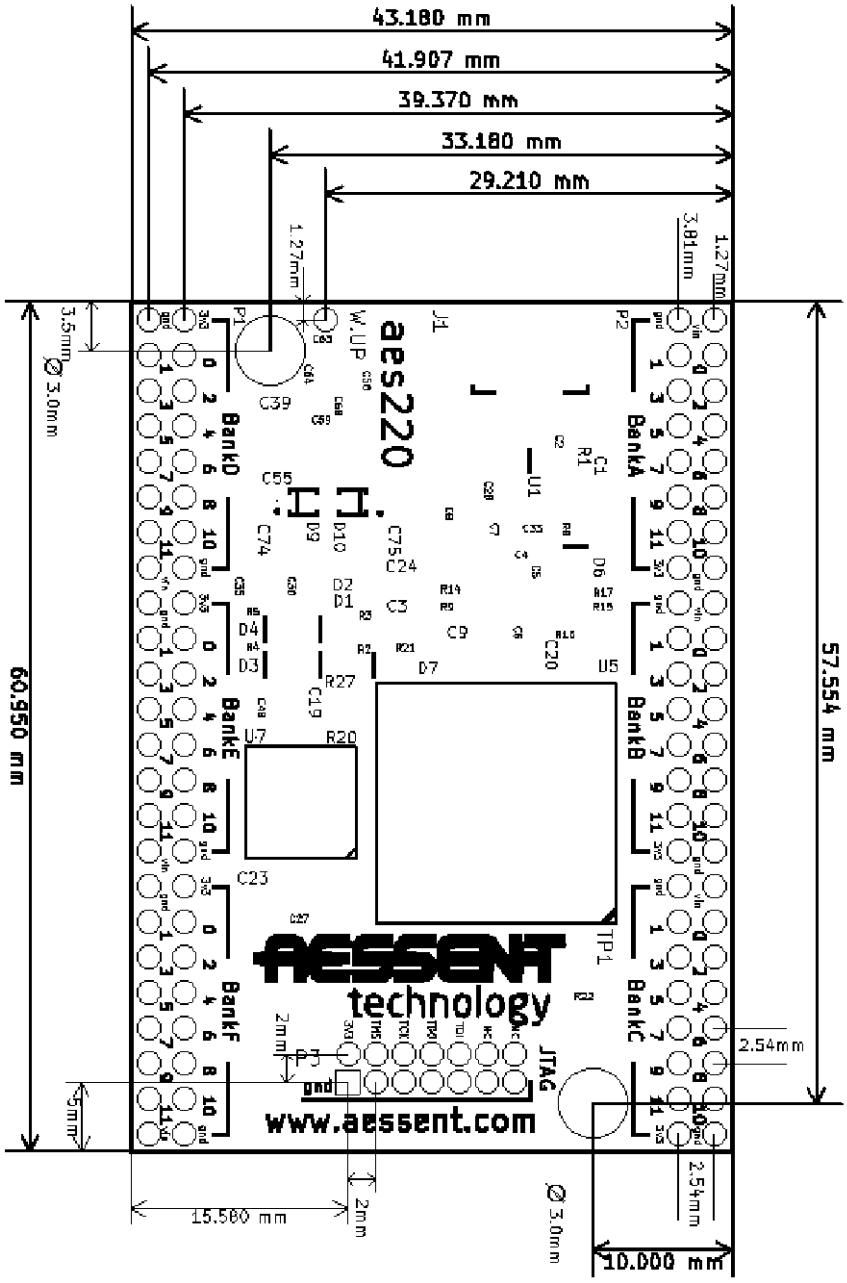


Figure 15: aes220 Board Dimensions