ELEC143 C1

INDIVIDUAL COURSEWORK C1

Dr. Nicholas Outram | Real Time Systems | 2018-19

Key Information

Module Leader Dr Nicholas Outram
Summative Deadline: See DLE
Coursework Weighting: 50% of module mark

FORMATIVE ASSESSMENT

If you demonstrate your solution to the tutor at least one week in advance, the tutor will be able to provide formative feedback intended to guide you towards improving aspects of your work and thus, increasing the likelihood of obtaining a better mark.

MODE

You may only work **individually**

ASSESSMENT

Work must be submitted online by the due date and time provided on the DLE.

The assessment scheme is given at the end of this document.

In some cases, the tutor may require you to attend a verbal examination (viva). This may be done for many reasons, which include:

- Where students request it
- The tutor needs further clarification to assign a grade
- It is judged that student would benefit from verbal feedback
- To ensure that the work you submit is your own and that you understand all aspects of the work.

Failure to attend a verbal examination when requested could result in your marks being reduced or even withdrawn.

Objective

You are to **create, test and evidence** a fully commented set of VHDL components that can be re-used by other engineers with confidence. You shall evidence functionality by writing testbenches (also in VHDL).

Tasks

Using VHDL, you are to create, test and evidence the full functionality of the following:

Component	***Functional and fully commented VHDL Code***	***Functional and fully commented VHDL test bench code*****
A component to interleave the bits of two N-bit inputs	10	10
N-Bit adder with carry and overflow outputs.	15	10
A N-bit program counter register that can be set, reset or incremented	15	10
SPI Master (16-bit)	15	15

N is a generic constant that is specified when a component is instantiated. Details are on the following pages.

Common attributes for all tasks

- For ALL tasks, inputs and outputs in each entity are of type `std_logic` or `std_logic_vector`.
- Where input and/or output vector sizes are **generic**, you should assume a maximum size of 16 bits for any given input or output.
- For full marks, you must **evidence** the functionality of your design using fully commented **VHDL test benches**.
- Where possible, use exhaustive testing and test edge cases as appropriate.

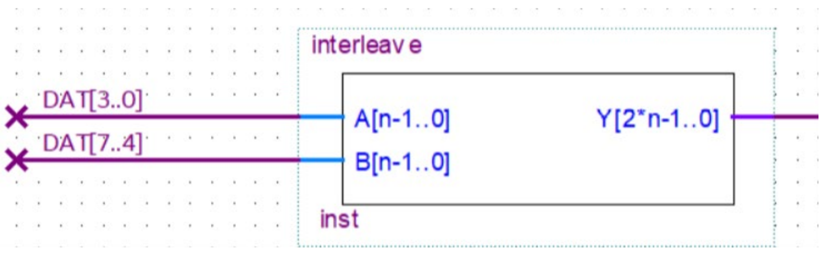
A single Quartus project file is provided including all vhdl files / entities. Your task is (i) to write the architecture in the space provided and write test benches for all components. Do NOT edit the entities or change the project structure.

N-bit Interleave

This is a simple component that manipulates two `std_logic_vector` inputs into one.

ENTITY

Filename: `interleave.vhd`



```
entity interleave is
      generic (
            N : natural := 4
      );

      port(
            A      : in  std_logic_vector((N-1) downto 0);
            B      : in  std_logic_vector((N-1) downto 0);
            Y      : out std_logic_vector((2*N-1) downto 0)
      );

end entity;
```

BEHAVIOUR

Consider the case where N=4. The output Y is constructed by interleaving the bits of A and B together as shown in the table below.

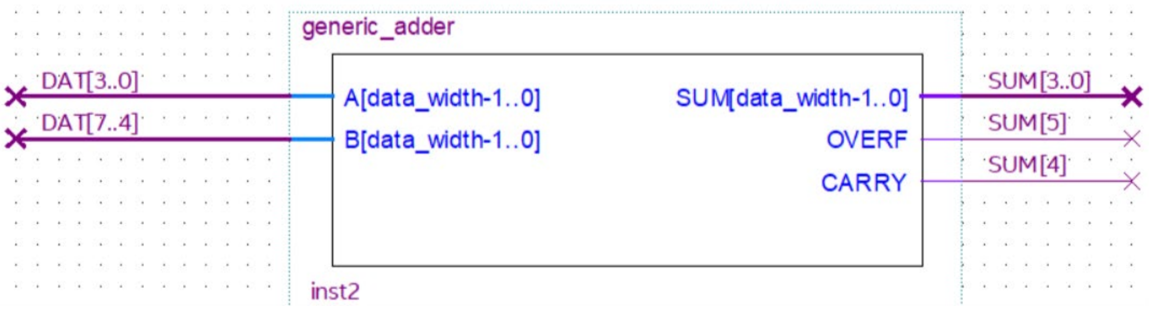
Y(7)	Y(6)	Y(5)	Y(4)	Y(3)	Y(2)	Y(1)	Y(0)
A(3)	B(3)	A(2)	B(2)	A(1)	B(1)	A(0)	B(0)

N-Bit adder

The adder has two N-bit inputs A and B, an N-bit output SUM and two single-bit outputs OVERF and CARRY.

ENTITY

Filename: `generic_adder.vhd`



```
entity generic_adder is

 generic (DATA_WIDTH : natural := 4);

 port (
      A      : in  std_logic_vector ((DATA_WIDTH-1) downto 0);
      B      : in  std_logic_vector ((DATA_WIDTH-1) downto 0);
      SUM : out std_logic_vector    ((DATA_WIDTH-1) downto 0);
      OVERF : out std_logic;
      CARRY : out std_logic
 );

end entity;
```

BEHAVIOUR

The operation is as follows:

- The device should calculate SUM=A+B.
- The OVERF bit should be HIGH when two signed values produce a numerical overflow
- The CARRY bit should be HIGH when two unsigned values produce a value that does not fit in SUM.

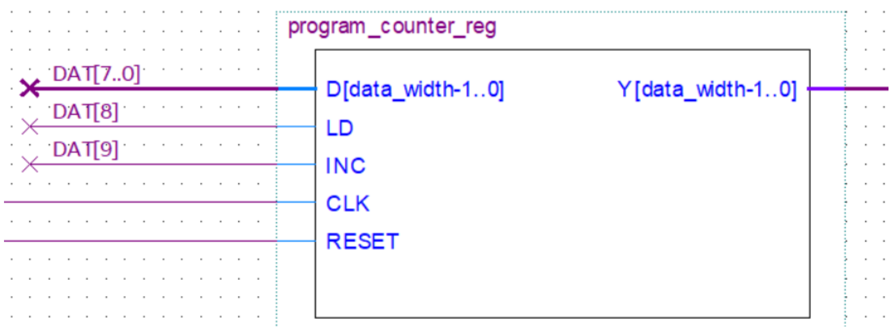
You should research the difference between overflow and carry – they are NOT the same thing.

A N-bit program counter register

This is a specialist register. A value can be loaded (as with any normal register). It has an additional feature increment its current value. Load always takes priority over increment.

ENTITY

Filename: `program_counter_reg.vhd`



```
entity program_counter_reg is

 generic (DATA_WIDTH : natural := 8);

 port (
      D     : in  std_logic_vector ((DATA_WIDTH-1) downto 0);
      LD    : in  std_logic;
      INC   : in  std_logic;
      CLK   : in  std_logic;
      RESET : in  std_logic;
      Y     : out std_logic_vector((DATA_WIDTH-1) downto 0)
      );

end entity;
```

D is the input data

LD is the LOAD input signal

INC is the INCREMENT input signal

CLK is the clock

RESET is an asynchronous reset input.

Y is the register value

BEHAVIOUR

The behaviour is described by the table below.

RESET	LD	INC	Y
HIGH	HIGH	LOW	Assigned to D on the rising edge of the clock (LOAD)
HIGH	HIGH	HIGH	Assigned to D on the rising edge of the clock (LOAD)
HIGH	LOW	HIGH	Should be incremented by 1 (Y = Y + 1)
HIGH	LOW	LOW	Latched to last known value
LOW	X	X	Zeros (asynchronous)

Where X indicates a don't care.

Note the following:

The reset always takes priority over all other functions

Y is only updated on the rising edge of the clock (except for the reset condition).

The LD input takes priority over the INC

ADC / SPI Master

This is a more advanced component designed to push those who may be confident in VHDL.

The DE0 Nano board has an Analogue to Digital Convert (ADC) connected to the FPGA using an SPI interface.

Your task is to complete the SPI master component that generates the SPI signals. See Appendix A for full details.

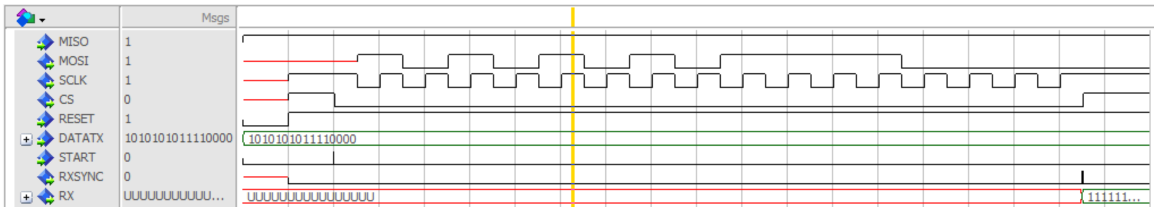
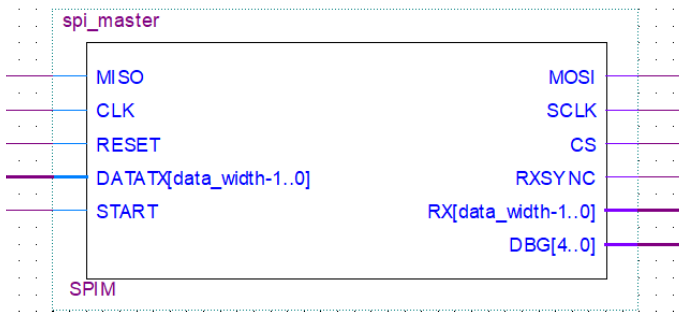


Figure 1. A complete SPI transaction between the FPGA (Master) and the ADC (Slave).

ENTITY

Filename: `spi_master.vhd`



```
entity spi_master is

generic (
      DATA_WIDTH : natural := 16;
      CLK_DIV    : natural := 25
);
port (
      MISO  : in   std_logic;     --Master In Slave Out
      MOSI  : out  std_logic;     --Master Out Slave In
      SCLK  : out  std_logic;     --SPI Serial CLock (1MHz)
      CS     : out  std_logic;    --Chip select
      CLK   : in   std_logic;     --FPGA Clock
      RESET  : in   std_logic;    --Async Reset
      DATATX : in   std_logic_vector((DATA_WIDTH-1) downto 0);
      START : in   std_logic;     --Start Conversion
      RXSYNC  : out  std_logic;   --HIGH when RX is valid
      RX     : out  std_logic_vector((DATA_WIDTH-1) downto 0);
      DBG    : out  natural range 0 to 31  --used for debug
      );
end entity;
```

Note - *the use of the generic values CLK_DIV and DATA_WIDTH are not mandatory in your architecture.*

BEHAVIOUR

This is a SPI Master Component that is interfaced to the ADC converter on the DEo-Nano Board. See Appendix A for the technical details.

Although it is the more advanced aspect of this coursework element, it is not as difficult to build as you might first think.

READ THIS CAREFULLY - what to edit / change

You are provided with both Quartus and Keil uVision projects. These include code used to automate testing on real hardware. Therefore, you need to read the following carefully!

Quartus project – This project contains multiple files. **You must only modify the following:**

`generic_adder.vhd`	THERE ARE THE **ONLY** FILES YOU SHOULD EDIT.
`interleave.vhd`	
`program_counter.vhd`	ONLY MODIFY THE ARCHITECTURE AND **NOT** THE
`spi_master.vhd`	ENTITY

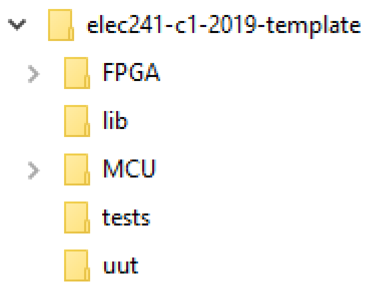
All these files are contained in a folder named `uut`

- These are located in the sub-folder `uut`[1] but should only be edited from within Quartus or ModelSim
- Do NOT modify any other files in the Quartus project.
- Only use Quartus v16.1

A Keil uVision Project is provided in the `MCU` folder– this can be used to test your components on real hardware. It sends a series of values over the SPI interface and echoes some results back in a PuTTY window. Hardware testing is only a requirement for the SPI Master.

FOLDER STRUCTURE

When you obtain the project files, they should have the following structure.



`FPGA` - Quartus project folder. Open the .qpf file using Quartus 16.1 to test the FPGA components on actual hardware. *You should not directly modify anything in this folder*

`lib` – VHDL source files shared with other projects. Again, do NOT modify anything in this folder.

[1] UUT stands for Unit Under Test

`MCU` - the Microcontroller code used to test your hardware. Inside there is a Keil uVision project that performs some very basic tasks. You do not need this unless you wish to attempt and test the SPI Master

`tests` – *This is where you should save your testbench files*. It is currently empty.

`uut` - Units Under Test. This is where the 4 components are to be found. *Your task is solely to modify the architecture of these files*. Do NOT change any filenames or edit the entity of any component.

What to submit

You are to submit a single ZIP file containing the following:

REQUIRED:

- The `uut` folder containing your modified VHDL files.
- The `tests` folder containing all your test bench files. These should be fully commented.
    - Full evidence of functionality should be demonstrated using VHDL test benches.
    - You should use the **assert** command to automate testing where possible
    - Test benches should be fully commented.
- `TASKS.docx` - complete the table within this document (see below)

Task	**Completed (F/P/N)****	**Test bench filename**	**Notes**
Interleave			
Adder			
Program Counter			
SPI Master			

** **F**ull, **P**artial or **N**one

Failing to list an achievement could result loosing marks for that element.

Note – there is no requirement to write a report. If your VHDL functions and test benches are properly written + commented sufficiently, then no report should be needed for the purposes of this assessment.

It should be possible to launch ModelSim from Quartus and compile+run your test benches.

OPTIONAL

- Any vector waveform files or ModelSim scripts (.do) used for testing may also be included. In the absence of a testbench, these approaches may be used but are capped at 50%
- PDF documentation containing details of further simulation and test results (see assessment scheme)

Appendix A – SPI Master Protocol

Below is a timing diagram illustrating the timing of the SPI Master component

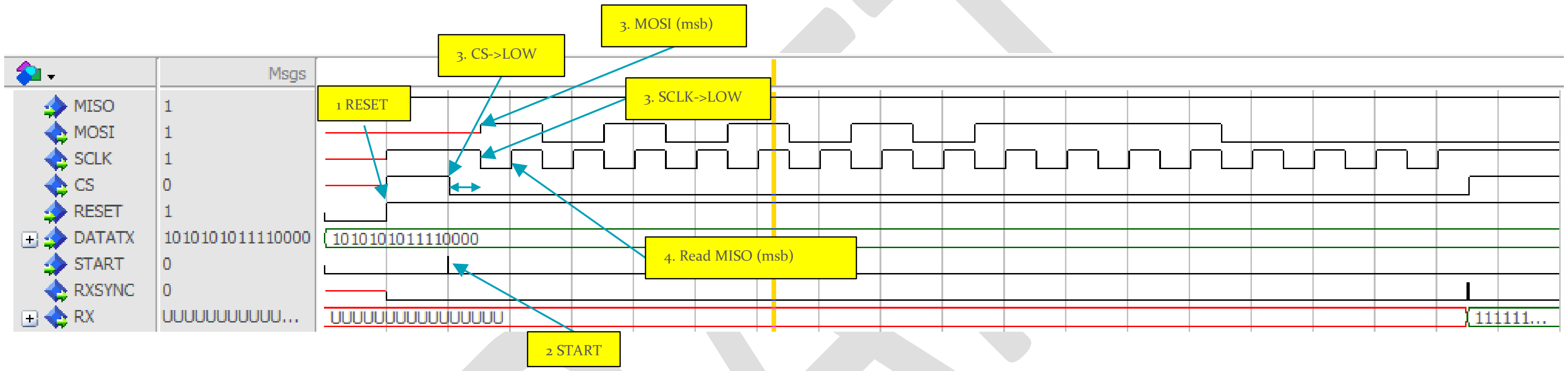


Figure 2 Showing a complete SPI transaction between the FPGA and the ADC. The FPGA is the SPI Master.

MISO – Generated by the ADC device. It should be sampled on the rising edge of SCLK. In this example it is shown to always output a 1 (3.3V input on the ADC).

MOSI – Generated by the SPI Master. It should be asserted on the falling edge of SCLK (the ADC will read it on the following rising edge).

SLKC – generated by the SPI Master, this should be a 1Mhz clock signal, derived from the 50MHz clock input. It is HIGH during the idle states.

CS – Generated by the SPI Master. It is HIGH during the idle states. Upon receiving a START input, the CS should be asserted low for the duration of the 16-bit transaction. It should be LOW at least 500ns before the first SCLK edge / after the last SCLK edge.

RESET – this is an asynchronous reset that resets the SPI master to its idle state.

DATATX – This is the data that is sent by the FPGA to the ADC (msb first) to select the analogue channel. As we are using channel 0, this can be permanently asserted low for the.

START – this input will go HIGH to request an ADC conversion. It is the input signal to start the SPI transaction. See

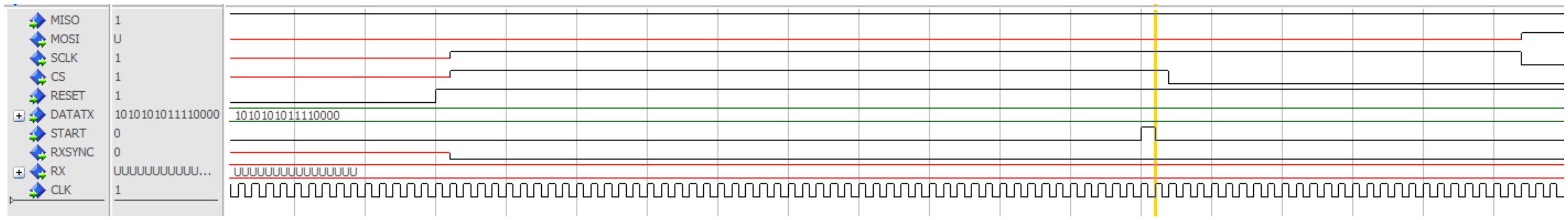


Figure 3. The SPI transaction begins when the START input is asserted HIGH. Note that CLK is the 50MHz clock and not the 1MHz SCLK.

RXSYNC – This should go briefly HIGH once RX holds a valid 16-bit result.

RX – The 16-bit data received from the ADC and should be latched. Note that the first 4 bits are always zero (this is a 12-bit ADC).

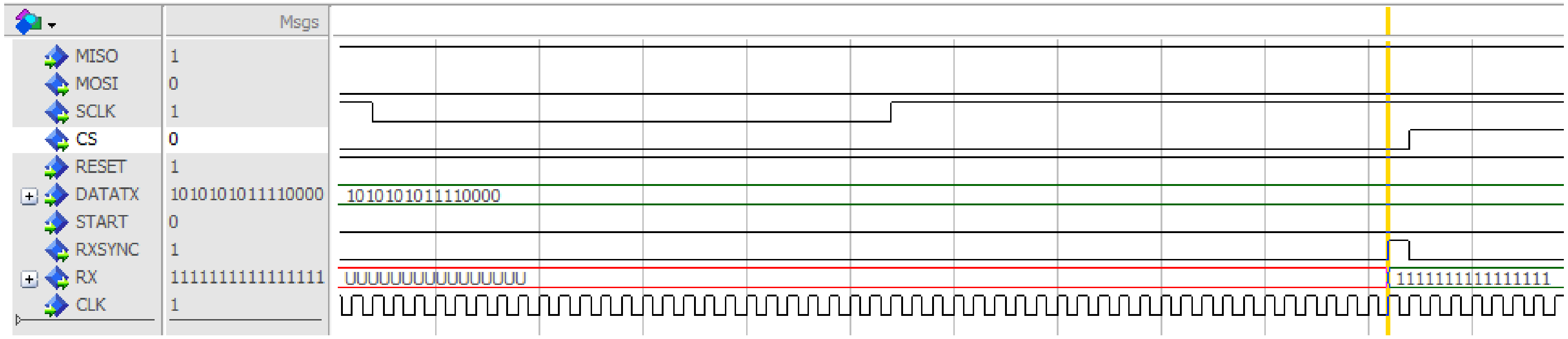


Figure 4. The RXSYNC signal should go HIGH to indicate that the RX signal holds a value 16-bit output.

TESTING

In addition to a testbench, you should demonstrate correct functionality using real hardware.

Will need an FPGA board, a microcontroller and ribbon cable.

- The analogue input is PIN24 of JP3 (see Figure 2 and Figure 3)
- You can use the ground and 3.3V pins on JP3 as test inputs (edge cases!)

A Keil uVision project is provided to help you test your design. Use PuTTY to see the output.

Appendix B – Key figures from the DE-0 Nano User Guide

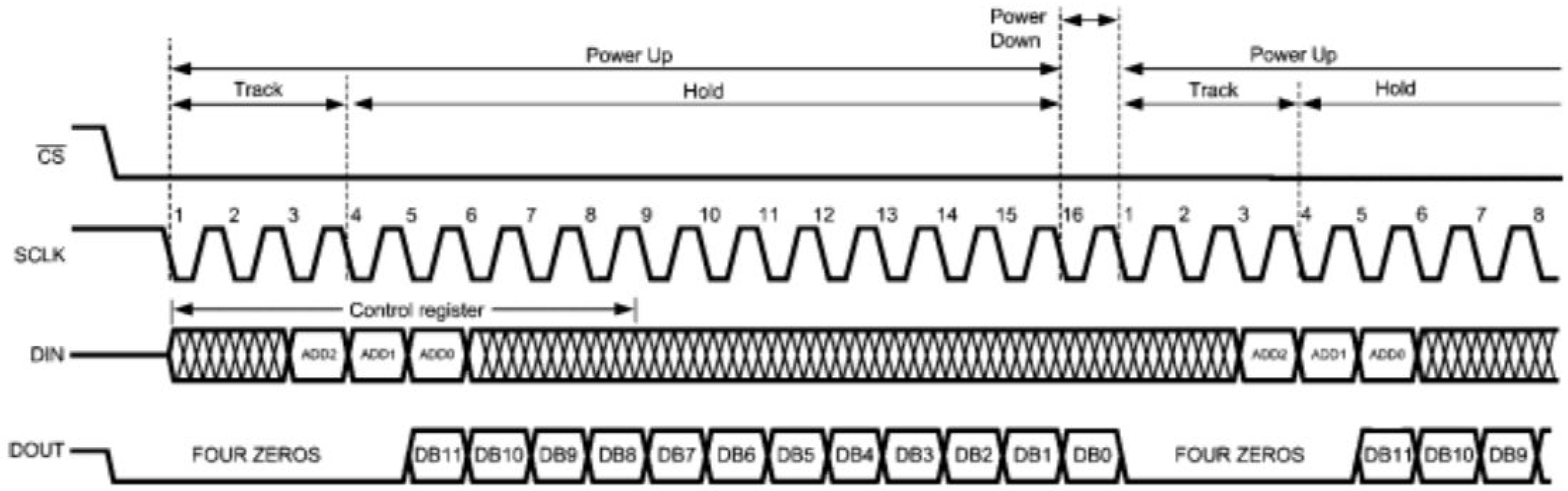


Figure 5. The SPI timing diagram for the ADC - source: DE0-Nano User Guide

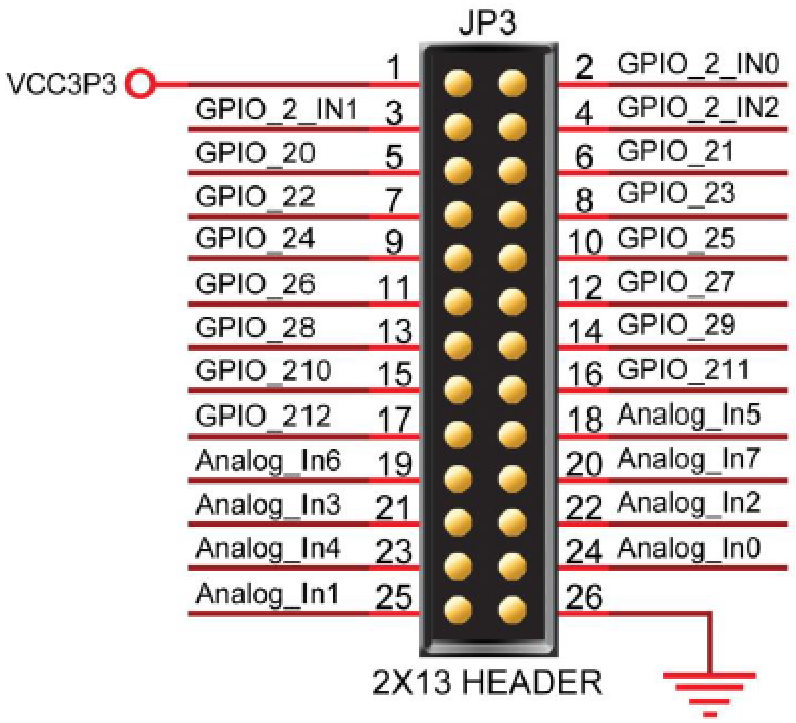


Figure 6. JP3 Connections

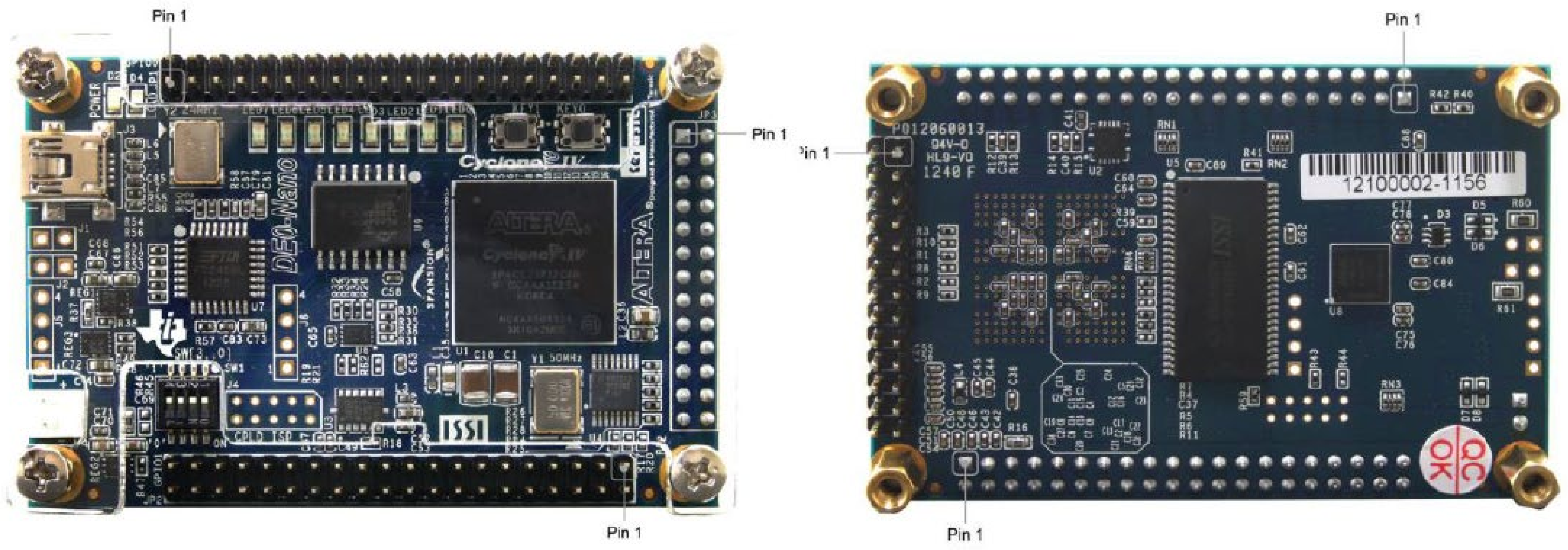


Figure 7. Locating Pin 1 on JP3