Interfacing the Nucleo F429ZI Microcontroller to the DE-0 Cyclone IV FPGA using SPI

Bill of Materials:

DE-0 Nano FPGA Board

Ribbon connector + Angle Bracket (not in your kit) – you may need to construct this

STM32F429FZ Nucleo Board

Software Required

Quartus Prime v18.1

Keil uVision 5.28 or later

PuTTY

The sources in the SPI_SLAVE_EXAMPLE folder

Document Revision

V1.0 - 01/12/2017 – Initial version for ELEC240

V1.01 – 01/12/2017 – Added a label to Figure 9 indicating the position of Pin 1

V1.02 – 14/11/2018 – Table of pin connections now on the same page as the figures to aid readability.

V1.03 - 29/01/2019 - Updated to use new ribbon cable

V1.04 – 24/01/2020 – Updated software versions

Task 01-01 Connecting the FPGA to the Nucleo F429ZI Board

In this task, you will connect the FPGA to the Microcontroller, and run some demo code.

1. Connect the ribbon cable to the FPGA connector JP1 as shown in Figure 1. If you do not have this cable, please ask the technicians (you may need to construct one yourself).

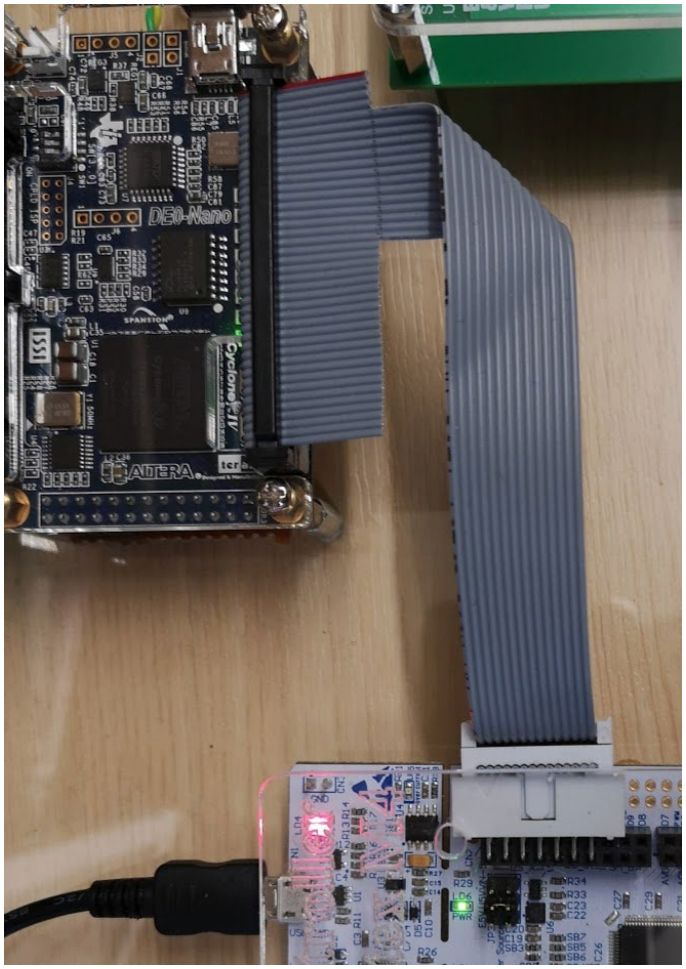


Figure 1. Showing the connections between the FPGA and the Nucleo Board

2. Browse to the SPI_SLAVE_EXAMPLE folder[1].
3. In Quartus, open the project in the FPGA folder. Build and deploy to the FPGA board
4. In Keil uVision, open the project inside the MCU Folder. Again, build and deploy to the Nucleo Board
5. Run device manager and note the COM port number used by that the “STMicroelectronics STLink Virtual COM Port” (see below)

[1] In general, it is advisable to put your files in a folder that has a path with no spaces and that is not too long. This is particularly important when we use ModelSim.

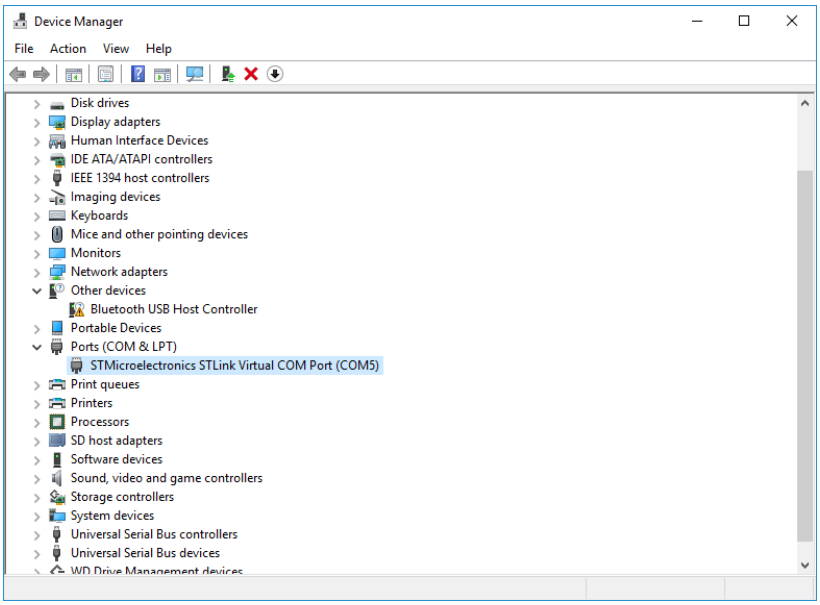


Figure 2. On Microsoft Windows, use the Device Manager to identify the COM port enumeration

You should now see the row of LEDs animate approximately once a second. This is known as the knight-rider pattern (from the 1970's TV series). You should also see the DE-0 Nano DIP switch positions reported in PuTTY.

TASK - Carefully change the DIP switch settings using a pointed tool and note the output in the PuTTY terminal.

Question: Where and how in the MCU software do we read the switch positions?

Question: Briefly read the Wikipedia article on SPI. How is it possible we can read and write a value to a device at the same time?

Task 01-02

This is a programming task to familiarize you with interfacing over SPI. Modify the C code to produce the following LED pattern:

All LEDs on

Wait for 1s

All LEDs off

Wait for 1s

All the even LEDs on

Wait for 1s

All the odd LEDs on

Wait for 1s

Repeat

Task 01-03: Software Decoder

Modify the C code to use three DIP switches to **select** which LED is on. Choose three DIP switches to represent a 3-bit binary select value.

Your main loop should read back the DIP switch settings from the FPGA (unsigned integer), and send a command to light the corresponding LED. Note that SPI reads and writes a byte at the same time.

For example:

SW3, SW2 and SW1 set to 111 all on would result in LED7 being illuminated (sends binary 10000000)

SW3, SW2 and SW1 set to 000 all on would result in LED0 being illuminated (sends binary 00000001)

SW3, SW2 and SW1 set to 010 all on would result in LED2 being illuminated (sends binary 00000100)

In effect, you are building a **software decoder**.

Task 01-04: Hardware Decoder

Currently we are sending 8-bits of data over the SPI interface, and this allows us to specify any of the 2^8combinations of 8-LEDs we choose. However, where we only require a subset of N combinations, then we only need $\lceil log_2(N) \rceil$ bits (where $\lceil x \rceil$ is the mathematical operator for rounding to the higher integer).

Design some logic in VHDL to perform hardware decoding. The MCU shall send data over SPI using the following format. The bits S2 down to S0 make up a 3-bit value, which determines which LED should be lit (000 for LED0, 111 for LED7 etc..).

msb							Lab
S2	S1	S0	Not used	Not used	Not used	Not used	Not used

Write some C code to generate some values to test your design – show the tutor when you've done this.

Note that we only have 8 different LED combinations, so we only need $log_2 8 = 3$ bits to specify. The remaining bits could be used for other purposes.

Task 01-04 Progressive Display

This is similar to task 01-04, only the logic will display a progressive display instead of a single LED. You will need to modify your VHDL to achieve this. There are still only 8 unique patterns, so again, we only need 3 bits to set the output state.

S2	S1	S0	LED7	LED6	LED5	LED4	LED3	LED2	LED1	LED0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	1
0	1	0	0	0	0	0	0	1	1	1
0	1	1	0	0	0	0	1	1	1	1
1	0	0	0	0	0	1	1	1	1	1
1	0	1	0	0	1	1	1	1	1	1
1	1	0	0	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1

Question. How might you compare the "output logic" of a Moore machine to a decoder? Might you classify this logic as a decoder?

Task 02 – Debugging Hardware with Signal Tap

In this task we are going to perform some on-chip FPGA hardware debugging. Quartus has a facility to add some additional logic to your design. This logic acts as a **logic analyser**. It samples internal signals, and uses on-chip[2] RAM to store them until the RAM is full. It then sends them back to Quartus (over USB) for display in a timing diagram.

- Go back to the original Quartus and Keil project files in the SPI_SLAVE_EXAMPLE (you might want to extract them again)
- Build and deploy both.

This should produce the simple knight-rider[3] LED pattern. We are now going to capture some of the SPI data in real-time and observe this in Quartus.

TASKS

In the project navigator, panel switch to the Files view. Then double click top open the stp1.stp file (stp is short for Signal TaP). A window similar to that in Figure 3 should appear.

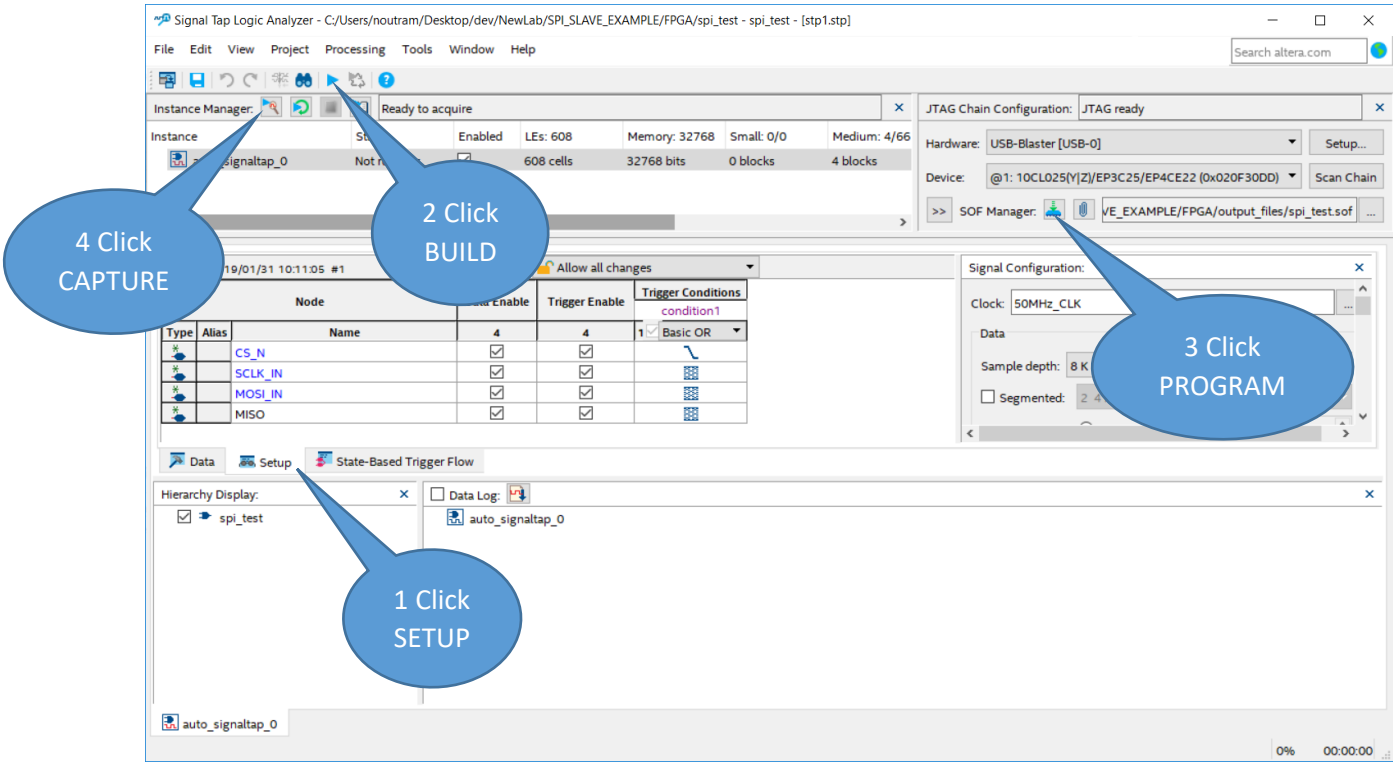


Figure 3. Showing the Signal Tap interface

1. Ensure you are in the SETUP view (1)
2. Click the build button (2) to build your design + the signal tap logic analyser block. Wait for the build to complete. You can get back to the signal tap window via the Window menu in Quartus.

[2] FPGA

[3] https://www.youtube.com/watch?v=Mo8Qls0HnWo

3. Click the program button (3) to deploy the design
4. Click the capture button (4) or press F5 to capture the data. You should see a timing diagram similar to that in Figure 4 appear. Note the data on MOSI and MISO will vary depending on timing and the DIP switch positions.
5. If you press F5 again, you will recapture data in real time. This is synchronized (triggered) to start each time the CS signal falls. The data you see on MOSI depends on what value the MCU sent to the SPI interface at that time.
6. The data on the MISO signal is the data being sent from the FPGA to the MCU. This reflects the settings on the DIP switches. Try changing the DIP switches on the FPGA board and press F5 to see the change in the MISO signal.

You can cross reference the signals shown with the following article
https://eewiki.net/pages/viewpage.action?pageId=7569477

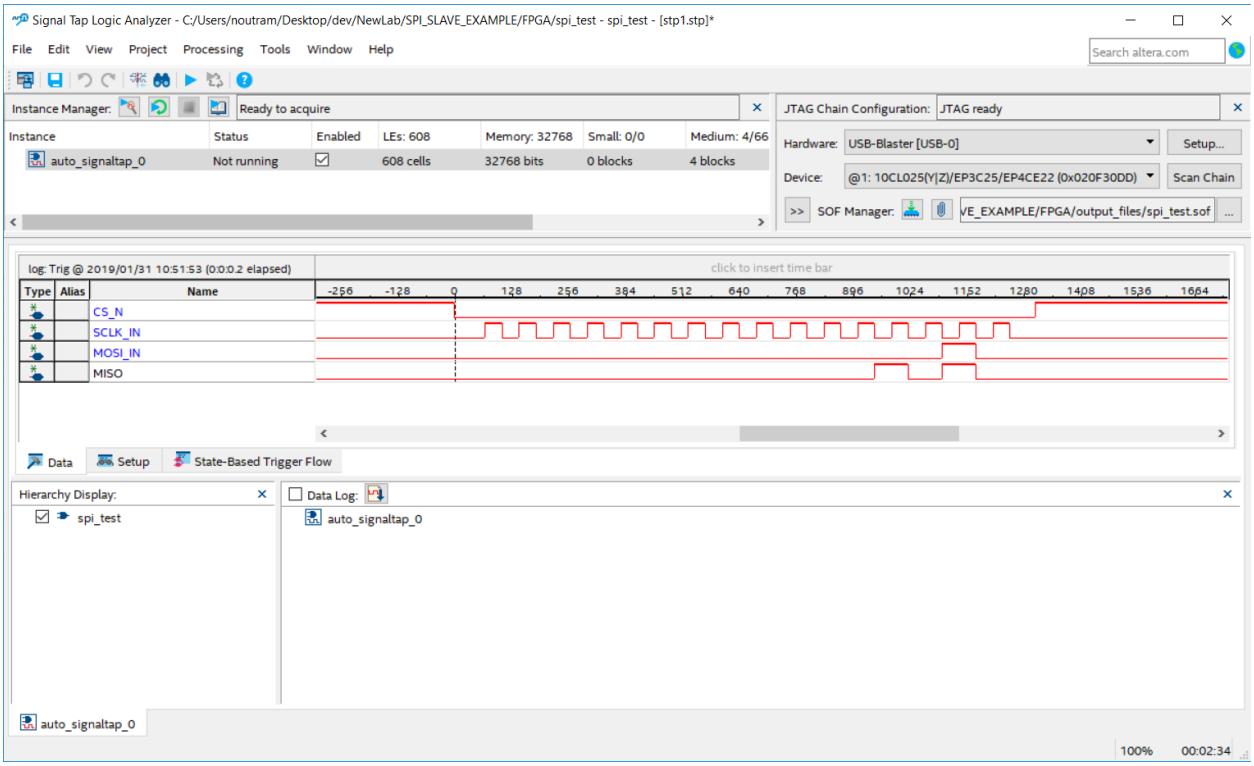


Figure 4. Example Signal Tap output for the SPI example

Questions / research tasks:

Which device is the Master and which is the Slave?

What do MISO and MOSI mean?

Which bit is sent first by the master?

Which bit is received first by the slave?

What is the purpose of the chip select (CS)?

Appendix A – Nucleo and DE0-Nano Connections

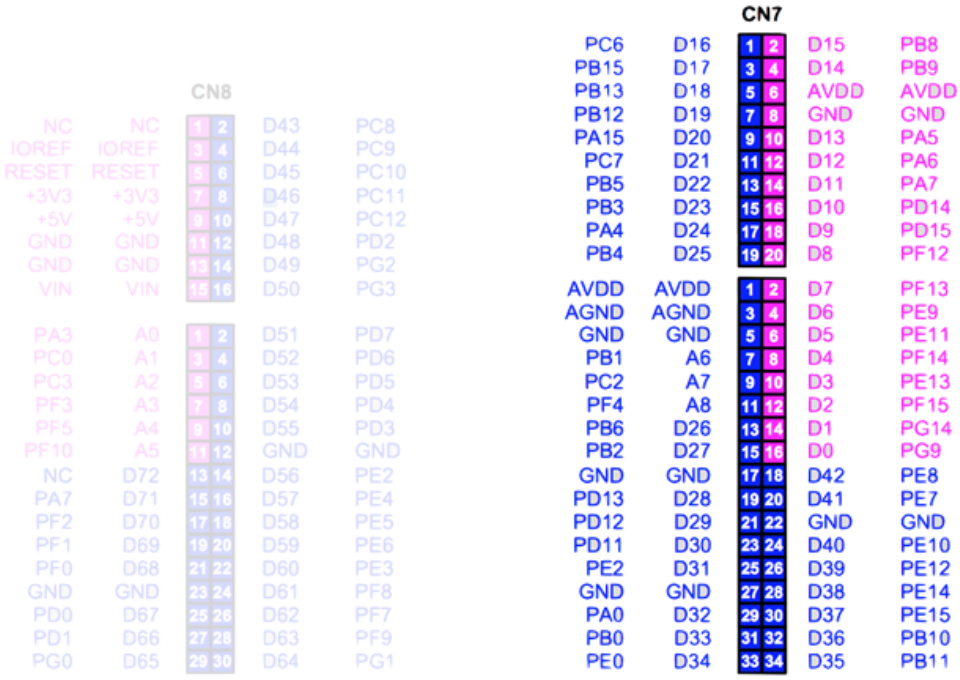


Figure 5 Nucleo Pinouts

Signal	MCU Pin (CN7)	FPGA Pin (JP1)
CS	PC_6 (D16) – Pin1	A2 (JP1-Pin 5) GPIO_2
GND	GND - Pin8/Pin22	Pin12
SCLK	PA_5 (D13) - pin10	D5 (JP1-Pin14) GPIO_9
MISO	PA_6 (D12) - pin12	A6 (JP1-Pin16) GPIO_11
MOSI	PA_7 (D11) – pin14	D6 (JP1-Pin18) GPIO_13

Table 1. Pin Assignments.

Note that only connector CN7 is used on the Nucleo Board and JP1 on the DE0-Nano FPGA board. Pin 8 on CN7 is strictly analogue ground, however this is connected to digital ground on this board. Note also that the pin number of the FPGA = pin number of the MCU + 4, making it easy to construct a ribbon cable.

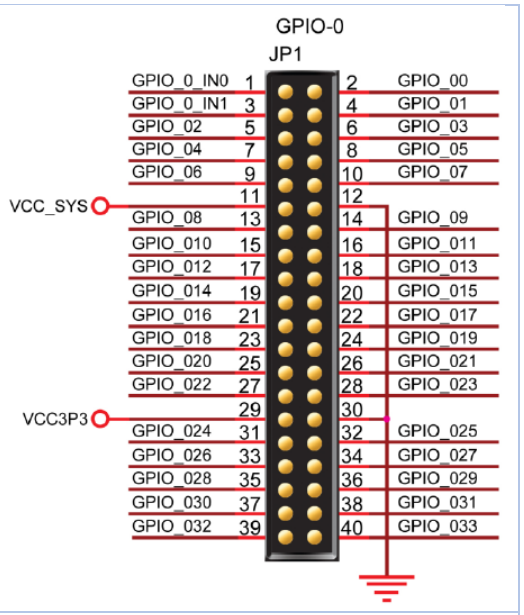


Figure 6. DE0-Nano Pinouts for JP1

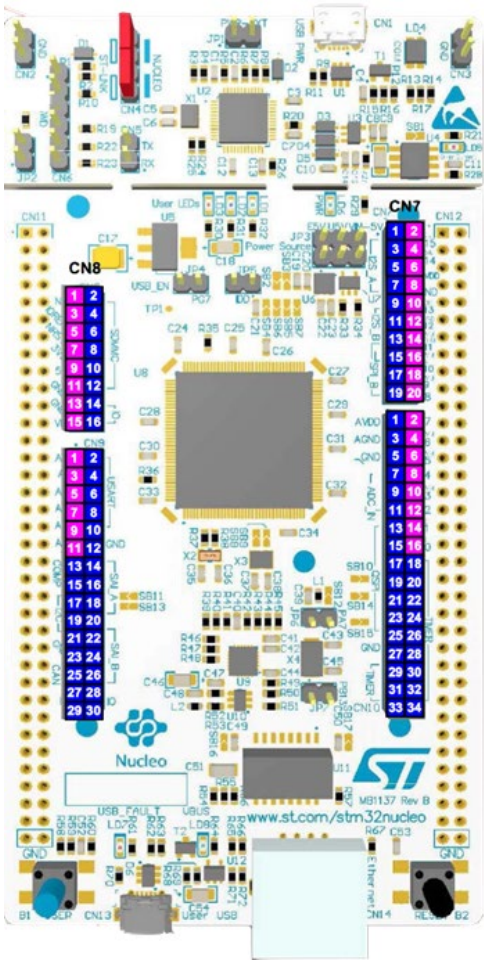


Figure 7. Nucleo FZ429ZI

REMOVED

Figure 8. Showing the connections on the Nucleo board CN7 connector.

REMOVED

Figure 9. Showing the connection to the FPGA DE-0 Nano Board

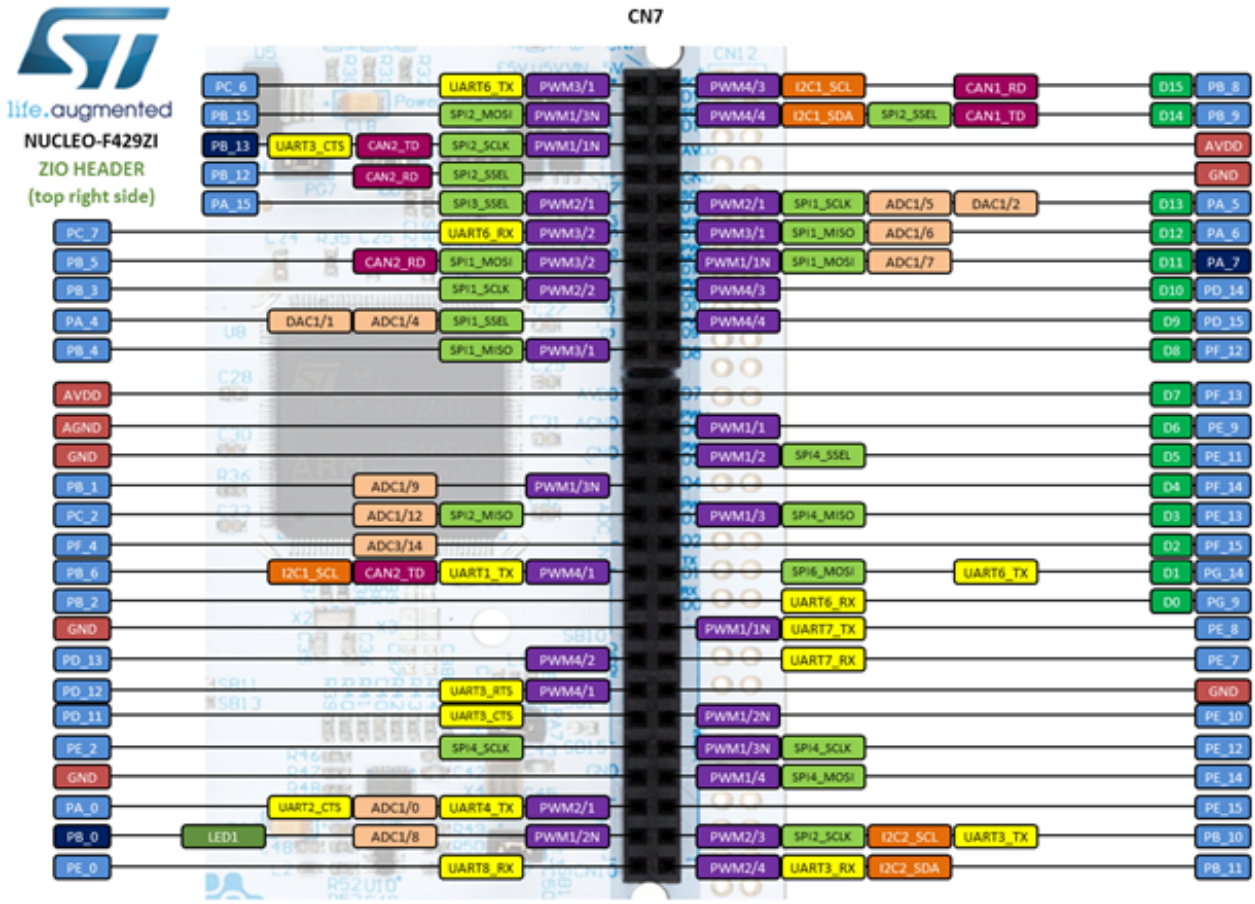
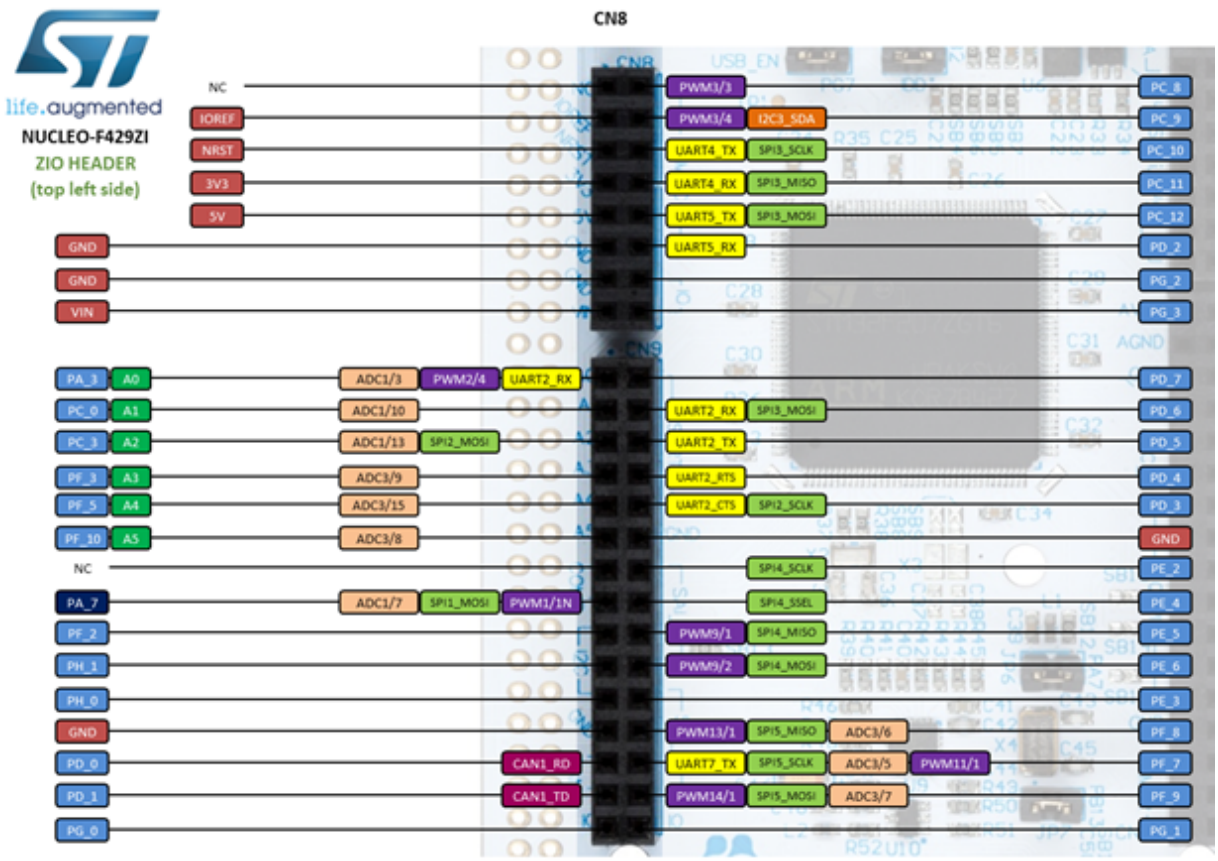


Figure 10. Showing the alternative functions of the pins on the Nucleo F429ZI CN7 connector


life.augmented
NUCLEO-F429ZI
ZIO HEADER
(top left side)
CN8
CN8
NC
IOREF
NRST
3V3
5V
GND
GND
VIN
PA_3
A0
PC_0
A1
PC_3
A2
PF_3
A3
PF_5
A4
PF_10
A5
NC
PA_7
PF_2
PH_1
PH_0
GND
PD_0
PD_1
PG_0
ADC1/3
PWM2/4
UART2_RX
ADC1/10
ADC1/13
SPI2_MOSI
ADC3/9
ADC3/15
ADC3/8
ADC1/7
SPI1_MOSI
PWM1/1N
CAN1_RD
CAN1_TD
PWM3/3
PWM3/4
I2C3_SDA
UART4_TX
SPI3_SCLK
UART4_RX
SPI3_MISO
UART5_TX
SPI3_MOSI
UART5_RX
UART2_RX
SPI3_MOSI
UART2_TX
UART2_RTS
UART2_CTS
SPI2_SCLK
SPI4_SCLK
SPI4_SSEL
PWM9/1
SPI4_MISO
PWM9/2
SPI4_MOSI
PWM13/1
SPI5_MISO
ADC3/6
UART7_TX
SPI5_SCLK
ADC3/5
PWM11/1
PWM14/1
SPI5_MOSI
ADC3/7
PC_8
PC_9
PC_10
PC_11
PC_12
PD_2
PG_2
PG_3
PD_7
PD_6
PD_5
PD_4
PD_3
GND
PE_2
PE_4
PE_5
PE_6
PE_3
PF_8
PF_7
PF_9
PG_1