U8G2 library on the PYNQ-Z2

|  |
| --- |
| Summary:   * U8G2 works with I2C, SPI and parallel interfaces. * Using the I2C hardware library of the PYNQ-Z2 with the U8G2 library * Set up and ready to use for a cheap I2C OLED screen. * Can support a wide range of screens with some modification to the code * **PYNQ-Z2 has no pull up resistors** |

# Revision history

|  |  |  |
| --- | --- | --- |
| Date | Author | Revision |
| 07/02/2024 | Mees Trietsch | Created the document; Described the general organization of the u8g2 library and the combination with the I2C hardware library. |

table of contents

[Revision history 1](#_Toc158311221)

[1 introduction 2](#_Toc158311222)

[2 Included functions 2](#_Toc158311223)

[3 The screen 2](#_Toc158311224)

[4 U8G2 library 3](#_Toc158311225)

[5 Hardware I2C 3](#_Toc158311226)

[6 Initializing display 5](#_Toc158311227)

[7 Porting to new MCU 5](#_Toc158311228)

[8 Using different screens and fonts 8](#_Toc158311229)

[9 references 9](#_Toc158311230)

[10 Appendix 9](#_Toc158311231)

# introduction

PYNQ-Z2 contains two I2C outputs, which are connected to the hardware I2C chip of the PS. This document will cover the use of this chip in combination with olikraus’ U8G2 library for driving a cheap OLED screen [1]. The OLED screen is driven by a SH1106 chip which is supported by the U8G2 library. The library supports a range of different screens. For the full list of the supported screen it is recommended to go to the homepage of the U8G2 library. <https://github.com/olikraus/u8g2>

The 4th and 5th chapter will dive into the working of the U8G2 and hardware I2C library for people who want to build their own code around the U8G2 library. While the 6th chapter will only cover the integration of the hardware I2C into the U8G2 library and how the SH1106\_screen library works.

# Included functions

* initDisplay(): initializes the U8G2 library, I2C chip and the screen. It as no inputs or outputs.
* printDisplay(): prints a string on the display on any location desired. It starts writing from the top left of the first letter. It does not remove the old data of the screen.
* printNew(): prints a new display. It first clears the display so the old data is gone. Furthermore it functions the same are the printDisplay() function.
* printCentreX(): prints on the horizontal mid line. It does not require an x location. It does not remove the old data of the screen.
* printNewMiddle(): prints in the middle of the screen. It does not require an x or y location. Removes the old data of the screen.
* txtWthFrameCentreX(): prints the data/text with a frame around it. Prints on the horizontal mid line. Does not remove the old data of the screen. A different font may not work with this function.

Possible input parameters:

* x the x location where should be printed
* y the y location where should be printed
* \*str pointer to a string. It can also be an array of characters. This will be printed on the screen.

# The screen

The screen is an 1.3 inch OLED screen that has a resolution of 128 by 64 pixels. The screen is powered by either 3V3 or 5V and the IO pins need the same power. The PYNQ-Z2’s logic works on 3V3, so the screen will need to powered with 3V3 to work. The screen also has a good viewing angle of 160 degrees. The PYNQ-Z2 has no pull up resistors but the screen itself does thus the screen is compatible with the PYNQ-Z2. If your screen has no pull up resistors it is necessary to add these.

Afbeelding met tekst, Lettertype, nummer

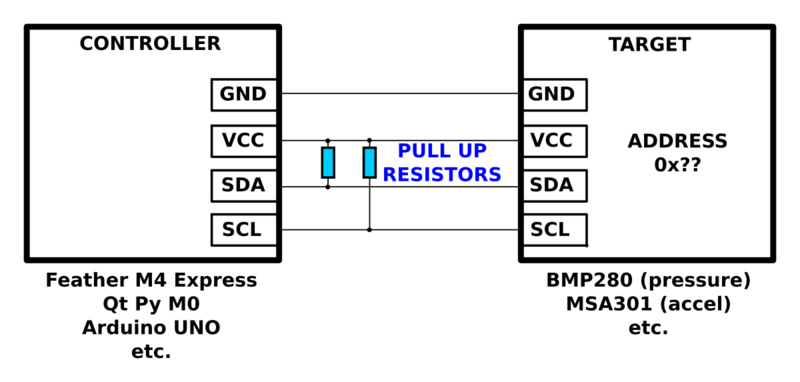
Automatisch gegenereerde beschrijving

Figure ) picture of the screen Figure 2) How the pull up resistors should be connected

# U8G2 library

The U8G2 library is a library that translates the data that needs to be send to a format that the screen can interpret. The library has a wide range of supported screens and communication protocols. It supports I2C, SPI and parallel interfaces and is widely used with Arduinos, but is also has support for C platforms. However, the library needs to be altered a bit to port it to the new MCU platform. This, and the setup procedure, is further explained in chapter 6 and 7.

# Hardware I2C

The PYNQ-Z2 has a hardware I2C chip that can be used to interface with devices. This HW I2C chip can be used by the PS by connecting the I2C bus of the PS with the Arduino I2C pins on the board as is shown in the following figure. The .xsa file is the bitstream of this block design and can be used to test the screen. By enabling and connecting the I2C bus, the xiicps library is enabled, meaning that the I2C code can be written. The following code enables the I2C chip.

The HW I2C library has multiple functions to send data. The function that is needed to drive the screen is the XIicPs\_MasterSendPolled(XIicPs \*InstancePtr, u8 \*MsgPtr, s32 ByteCount, u16 SlaveAddr). This function initiates a polled mode send in master mode, meaning that the PYNQ-Z2 is the master and will poll if the slave has received the data. The function expects the following parameters [2]:

* \*InstancePtr Pointer to the I2C instance
* \*MsgPtr Pointer to the data that needs to be send
* ByteCount The amount of bytes that need to be send
* SlaveAddr The address of the slave where the data needs to be written to

If the slave address is not correct the slave will not be able to read the data and the HW I2C library will give an error.

This function will later be used to send the data from the PYNQ-Z2 to the screen.

// Look up the config of the IIC

Config = XIicPs\_LookupConfig(XPAR\_XIICPS\_0\_DEVICE\_ID);

if (NULL == Config) {

printf("XIicPs\_LookupConfig failure\r\n");

return XST\_FAILURE;

}

// Initialize the IIC using the config

Int status = XIicPs\_CfgInitialize(&Iic, Config, Config->BaseAddress);

if (status != XST\_SUCCESS) {

printf("XIicPs\_CfgInitialize failure\r\n");

return XST\_FAILURE;

}

// Do a selftest on the IIC struct to ensure it is working

status = XIicPs\_SelfTest(&Iic);

if (status!= XST\_SUCCESS) {

printf("IIC selftest FAILED \r\n");

return XST\_FAILURE;

}

// Set the clock speed of the IIC bus

status = XIicPs\_SetSClk(&Iic, IIC\_SCLK\_RATE)

if (status != XST\_SUCCESS) {

printf("IIC setClock FAILED \r\n");

return XST\_FAILURE;

}

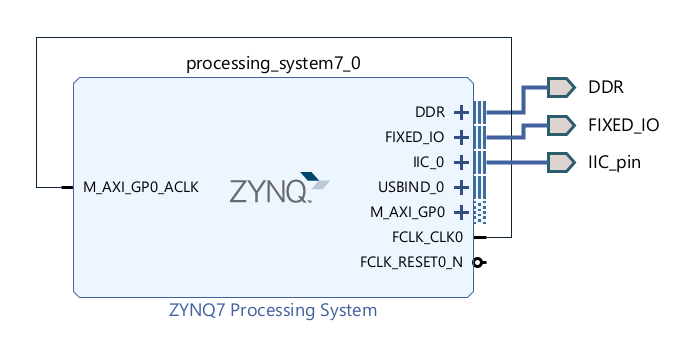


Figure ) block diagram of the PL in vivado

# Initializing display

Each setup procedure name has the following naming convention [3]:

u8g2\_Setup\_<display>\_<i2c>\_<memory>

* <display>: Describes the display name and subtype (dimension, configuration options).
* <i2c>: I2C protocol requires special care. If the display supports I2C communication interface, then there is a special setup procedure for I2C, which includes the string "i2c" in the procedure name. For SPI or parallel communication, use the setup procedure without "i2c" in the procedure name.
* <memory>: This is 1, 2 or f for one page, two page or full page mode.

The naming convention is just a rough idea for the function names. Function names and their possible arguments are listed below.

All available setup procedure are listed in the rest of the document. Each setup procedure requires four arguments:

1. **U8g2**: Pointer to an empty u8g2 structure (see the example above)
2. **Rotation**: Rotation procedure, see appendix
3. **Byte** **communication** **procedure**: Either one of the existing procedures or a custom procedure for the target controller.
4. **Lowlevel** **delay** **and GPIO procedure**: This must be a custom procedure.

The 3rd and 4th argument are further explained in chapter 6.

# Porting to new MCU

Because we want to use the hardware I2C chip of the PYNQ-Z2, the 3rd and 4th setup procedure arguments need to be custom callback functions. This chapter will give a detailed view of the callback functions required.

The byte communication procedure callback function is the link between the U8G2 and HW I2C library. This custom function rounds up all the data that needs to be send to the display and uses the I2C library functions to send the data.

Prototype of the function:

uint8\_t cb\_HW\_I2C\_send(u8x8\_t \*u8x8, uint8\_t msg, uint8\_t arg\_int, void \*arg\_ptr)

This function is not used by the user of the library self. It only is passed on to the U8G2 library so it can use it to send data to the screen. This function will receive a “message” of the U8G2 library that is related to a certain function the function has to do. The following list is a list of all the messages that can be send.

1. **U8X8\_MSG\_BYTE\_INIT:** This message is send only once when the I2C needs to be initialized.
2. **U8X8\_MSG\_BYTE\_SEND:** This message is send when data has to be rounded up into a buffer.
3. **U8X8\_MSG\_BYTE\_SET\_DC:** Is ignored for I2C communication
4. **U8X8\_MSG\_BYTE\_START\_TRANSFER:** This message is send when sending of data needs to start.
5. **U8X8\_MSG\_BYTE\_END\_TRANSFER:** This message is send when the sending of data needs to stop.

The actual purpose of the messages will change a little bit to work better with the I2C hardware. The HW I2C function needs to receive a string or a char array that works as a buffer so the data needs to be stored into a buffer.

By looking at the original purpose of the messages the buffer can be loaded and send correctly. This means that the new functions of the messages are as follows:

1. **U8X8\_MSG\_BYTE\_INIT:** Initializes the I2C. It is only called once during the init process.
2. **U8X8\_MSG\_BYTE\_SEND:** Loads the data into the buffer.
3. **U8X8\_MSG\_BYTE\_SET\_DC:** Is ignored for I2C communication
4. **U8X8\_MSG\_BYTE\_START\_TRANSFER:** Resets the buffer so data can be loaded.
5. **U8X8\_MSG\_BYTE\_END\_TRANSFER:** Send the buffer using the HW I2C chip.

In figure 4 the code that does these functions is shown. For a copiable version check the repository or the appendix.

<https://github.com/FuzzyUmbrella2/U8G2-for-PYNQ_Z2>

The second callback function that is used by the U8G2 library is used to specify the GPIO and the delays. In this callback function the GPIO pins could be initiated and/or configured and the delays that may be required by the U8G2 library are done. However, the use of the HW I2C library means that there is no GPIO that needs to be initiated (this is done by the HW I2C library) and there is no need for delays. This means that this callback function will not have any useful information. But it is necessary to include this callback function because the U8G2 library calls it during the setup phase, it will just do nothing. Because this function will do nothing we will not go into detail about this callback function. However, it this function may be necessary for the use of SPI or parallel interfaces.

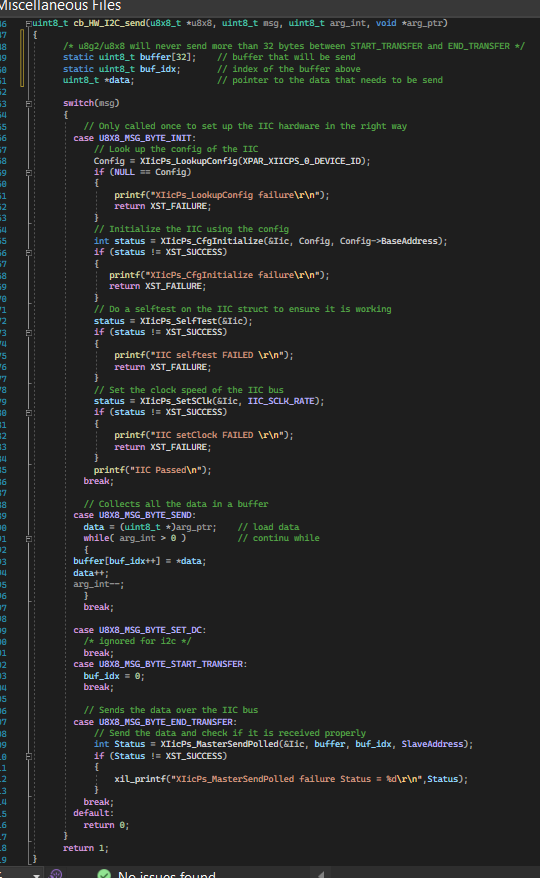


Figure ) Code of the function callback function that sends the data to the screen

# Using different screens and fonts

Since the U8G2 library supports a wide variety of screen the SH1106\_screen library is made in such a way that using the library with different screen is rather easy.

1. Uncomment the setup code of the screen that is going to be used in the u8g2\_d\_setup.c file.
2. Changing the setup function call in the initDisplay() function in the SH1106\_screen.c file.  
   **Don’t change the parameters**

These are the only two steps that have to be taken to be able to use different screens.

The U8G2 library also has a lot of different fonts included but the font that is used is the only one that is enabled. To use other fonts the following steps have to be taken.

1. Uncomment the font code that is desired to use in the u8g2\_fonts.c or u8x8\_fonts.c file.
2. Change the font name in the define.h file.

Because the u8g2\_d\_setup.c, u8g2\_fonts.c and u8x8\_fonts.c files are very big the unused code is commented out. This makes the program quicker to build and upload. Without this the code size that is going to be uploaded is around 10MB.

# references

[1]

Tiny Tronics, “1.3 inch OLED Display 128\*64 pixels wit - I2C,” *Tiny Tronics*. https://www.tinytronics.nl/shop/nl/displays/oled/1.3-inch-oled-display-128\*64-pixels-wit-i2c (accessed Feb. 08, 2024).

[2]

Xilinx, “embeddedsw/XilinxProcessorIPLib/drivers/iicps/src/xiicps.h at master · Xilinx/embeddedsw,” *GitHub*. https://github.com/Xilinx/embeddedsw/blob/master/XilinxProcessorIPLib/drivers/iicps/src/xiicps.h (accessed Feb. 08, 2024).

[3]

olikraus, “u8g2setupc,” *GitHub*. https://github.com/olikraus/u8g2/wiki/u8g2setupc (accessed Feb. 08, 2024).

# Appendix

|  |  |
| --- | --- |
| Rotation/Mirror | Description |
| U8G2\_R0 | No rotation, landscape |
| U8G2\_R1 | 90 degree clockwise rotation |
| U8G2\_R2 | 180 degree clockwise rotation |
| U8G2\_R3 | 270 degree clockwise rotation |
| U8G2\_MIRROR | No rotation, landscape, display content is mirrored (v2.6.x) |

|  |  |
| --- | --- |
| Byte Procedure | Description |
| u8x8\_byte\_4wire\_sw\_spi | Standard 8-bit SPI communication with "four pins" (SCK, MOSI, DC, CS) |
| u8x8\_byte\_3wire\_sw\_spi | 9-bit communication with "three pins" (SCK, MOSI, CS) |
| u8x8\_byte\_8bit\_6800mode | Parallel interface, 6800 format |
| u8x8\_byte\_8bit\_8080mode | Parallel interface, 8080 format |
| u8x8\_byte\_sw\_i2c | Two wire, I2C communication |
| u8x8\_byte\_ks0108 | Special interface for KS0108 controller |

uint8\_t cb\_HW\_I2C\_send(u8x8\_t \*u8x8, uint8\_t msg, uint8\_t arg\_int, void \*arg\_ptr)

{

/\* u8g2/u8x8 will never send more than 32 bytes between START\_TRANSFER and END\_TRANSFER \*/

static uint8\_t buffer[32]; // buffer that will be send

static uint8\_t buf\_idx; // index of the buffer above

uint8\_t \*data; // pointer to the data that needs to be send

switch(msg){

// Only called once to set up the IIC hardware in the right way

case U8X8\_MSG\_BYTE\_INIT:

XIicPs\_Config \*Config;

// Look up the config of the IIC

Config = XIicPs\_LookupConfig(XPAR\_XIICPS\_0\_DEVICE\_ID);

if (NULL == Config){

printf("XIicPs\_LookupConfig failure\r\n");

return XST\_FAILURE;

}

// Initialize the IIC using the config

int status = XIicPs\_CfgInitialize(&Iic, Config, Config->BaseAddress);

if (status != XST\_SUCCESS){

printf("XIicPs\_CfgInitialize failure\r\n");

return XST\_FAILURE;

}

// Do a selftest on the IIC struct to ensure it is working

status = XIicPs\_SelfTest(&Iic);

if (status != XST\_SUCCESS){

printf("IIC selftest FAILED \r\n");

return XST\_FAILURE;

}

// Set the clock speed of the IIC bus

status = XIicPs\_SetSClk(&Iic, IIC\_SCLK\_RATE);

if (status != XST\_SUCCESS){

printf("IIC setClock FAILED \r\n");

return XST\_FAILURE;

}

printf("IIC Passed\n");

break;

// Collects all the data in a buffer

case U8X8\_MSG\_BYTE\_SEND:

data = (uint8\_t \*)arg\_ptr; // load data

while( arg\_int > 0 ) // continue while there is data

{

buffer[buf\_idx++] = \*data;

data++;

arg\_int--;

}

break;

case U8X8\_MSG\_BYTE\_SET\_DC:

/\* ignored for i2c \*/

break;

case U8X8\_MSG\_BYTE\_START\_TRANSFER:

buf\_idx = 0;

break;

// Sends the data over the IIC bus

case U8X8\_MSG\_BYTE\_END\_TRANSFER:

// Send the data and check if it is received properly

int Status = XIicPs\_MasterSendPolled(&Iic, buffer, buf\_idx, SlaveAddress);

if (Status != XST\_SUCCESS){

xil\_printf("XIicPs\_MasterSendPolled failure Status = %d\r\n",Status);

}

break;

default:

return 0;

}

return 1;

}