TLDR explanation

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 |
| 09/02/2024 | 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](#_Toc158311344)

[1 General 2](#_Toc158311345)

[2 Included functions 2](#_Toc158311346)

[3 Hardware I2C 3](#_Toc158311347)

[4 Initializing display 4](#_Toc158311348)

[5 Porting to new MCU 4](#_Toc158311349)

[6 Using different screens and fonts 5](#_Toc158311350)

[7 Appendix 5](#_Toc158311351)

# General

PYNQ-Z2 contains two I2C outputs, which are connected to the hardware I2C chip of the PS. This document is a short version that explains how to use the library. The library is based on Olikraus’ U8G2 library. 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.

U8G2 library: <https://github.com/olikraus/u8g2>

OLED screen: [https://www.tinytronics.nl/shop/nl/displays/oled/1.3-inch-oled-display-128\*64-pixels-wit-i2c](https://www.tinytronics.nl/shop/nl/displays/oled/1.3-inch-oled-display-128*64-pixels-wit-i2c)

The library is ready to use and has built-in functions for initiating the screen or printing on the screen. The following chapter will give some information about the included functions while the other chapters will give a more detailed description of the library and how it 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.

# Hardware I2C

The PYNQ-Z2 has a hardware I2C chip that can be used to interface with devices. This HW I2C chip is connected to the PS of the PYNQ-Z2 and is connected to the IO pins as shown in figure 1. The following code configures the I2C chip.

// 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

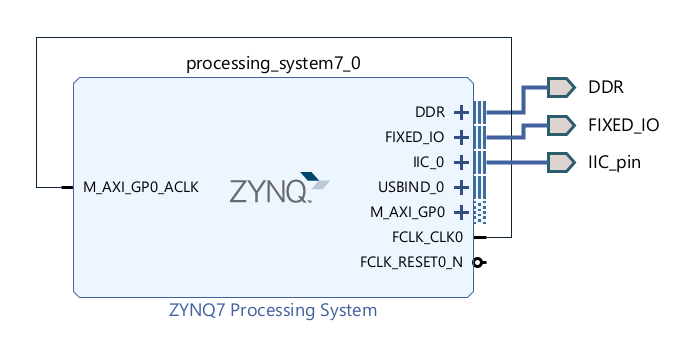
status = XIicPs\_SetSClk(&Iic, IIC\_SCLK\_RATE)

if (status != XST\_SUCCESS) {

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

return XST\_FAILURE;

}



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:

* \*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.

# Initializing display

Each setup procedure name has the following naming convention:

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

Two callback functions are required by the setup function. This chapter will give some information about thes function.

Prototype of the function:

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

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.

The code that does these functions is shown in the link below or in the appendix.

<https://github.com/FuzzyUmbrella2/U8G2-for-PYNQ_Z2/blob/main/src/SH1106_Screen.c>

The second callback function that is used by the U8G2 library is used to specify the GPIO and the delays. Since we use the HW I2C library this callback function does not need any functionality, thus it has no actual code that does something.

# Using different screens and fonts

Change the screen:

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**

Change the font.

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.

# 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;

}