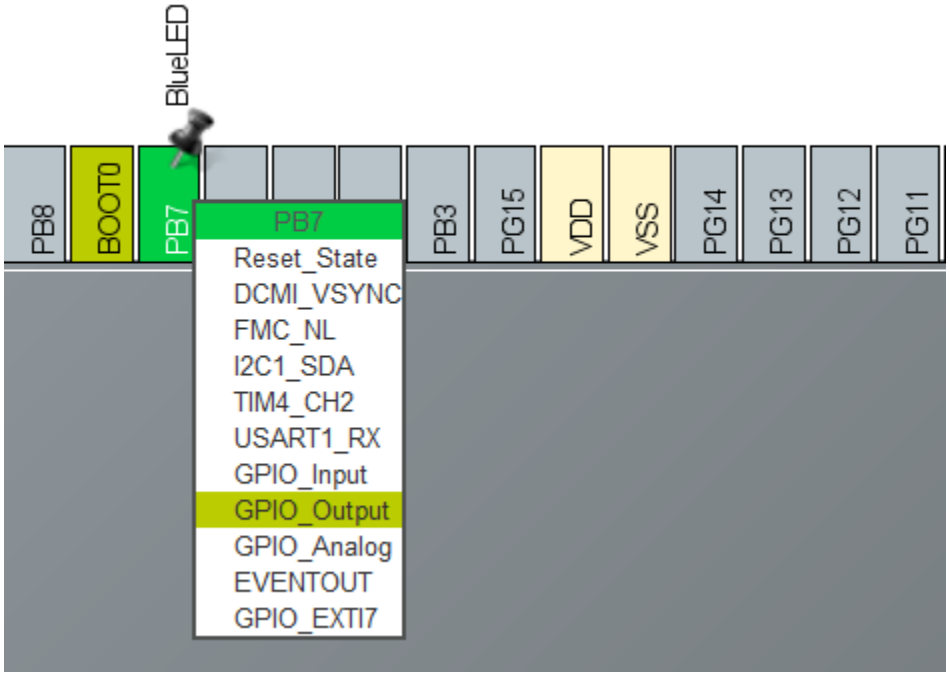
1. Follow the document named ‘Environment_setup.pdf’ to set up the IDE and create a project.
2. Download PuTTY from https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html
3. Initialize the peripherals for the LEDs that will be blinked
    a. Open up the project IOC file, and navigate to the ‘pinout’ view
    b. Select pin PB7 and set it to ‘GPIO_Output’:



This will allow the pin to be set HIGH or LOW. On the board, there is a blue LED attached to this pin which we will be able to toggle. You may also set pins PB0 (green LED) and PB14 (red LED) if you would like.

    c. Right click the pin PB7 and give it a custom label, e.g. ‘BlueLED’.
    d. Save the IOC file (CRTL+S) and generate new code. This will now create all of the initialization code and call it in main.c.
4. In the project Explorer, navigate to ‘Core/Src/’ and open main.c to view the generated code and modify it to blink an LED.
    a. Can now see that the initialization code has been generated and called in main():

```
/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();

/* USER CODE BEGIN 2 */
```

b. In the MX_GPIO_Init() function, the following line was also generated to initialize the LED to an 'Off' state:

```
/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(BlueLED_GPIO_Port, BlueLED_Pin, GPIO_PIN_RESET);
```

Passing the value 'GPIO_PIN_RESET' to the 'HAL_GPIO_WritePin' function will set the GPIO pin low, or in this case it will turn the LED off. If the value 'GPIO_PIN_SET' is passed it will set the GPIO pin high (or turn the LED on):

```
HAL_GPIO_WritePin(BlueLED_GPIO_Port, BlueLED_Pin, GPIO_PIN_SET);
```

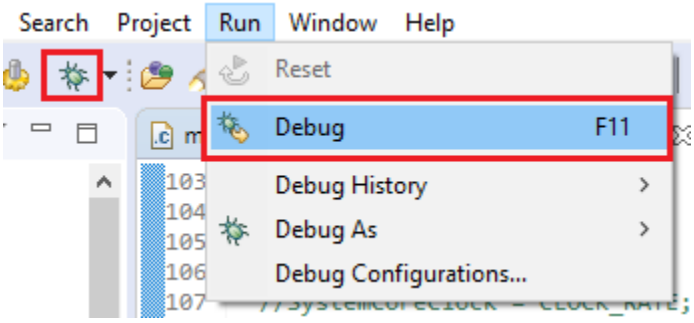
c. These two lines of code may then be used to blink the blue LED. Under main(), the SET and RESET code may be placed along with a HAL_Delay() as shown:

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  HAL_GPIO_WritePin(BlueLED_GPIO_Port, BlueLED_Pin, GPIO_PIN_RESET);
  HAL_Delay(500);
  HAL_GPIO_WritePin(BlueLED_GPIO_Port, BlueLED_Pin, GPIO_PIN_SET);
  HAL_Delay(500);
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
```

This will toggle the LED on and off, with a delay of 500 milliseconds between each toggle. If you added the red or green LEDs, you may add code for them similarly.

5. A debug configuration may now be created, and the code can be run

a. Run the debug with either button or F11:



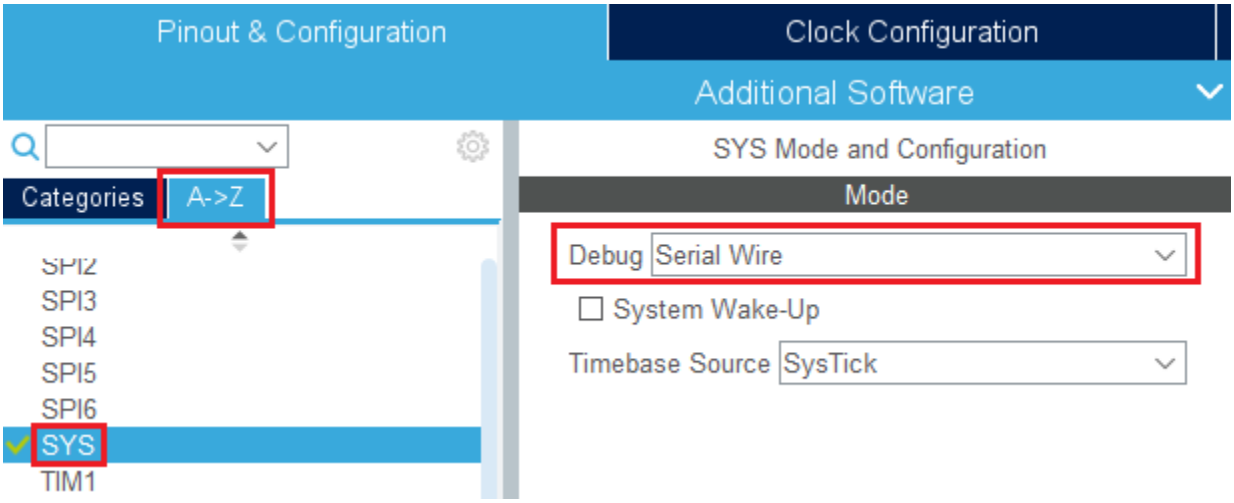
b. This will prompt for a debug configuration to be created. Select 'STM32 Cortex-M C/C++ Application'. On the following screen, nothing should need to be changed so just click the 'Ok' button to continue.

c. The code will be compiled and stored on the board's flash memory. Once finished, the program will then pause on the HAL_Init() line, and may be run by clicking the Resume (F8) button:

d. The code should now be running, and the LEDs blinking.

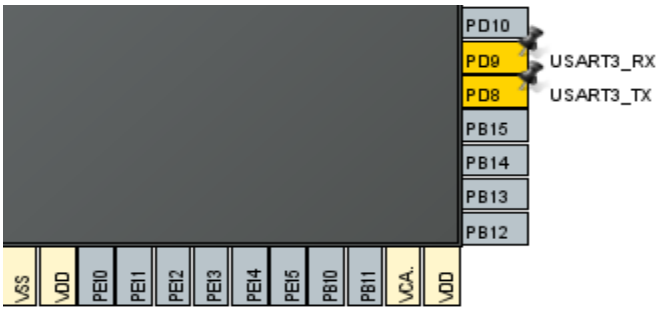
6. Now we can configure the board to output print statements which may be captured by the PuTTY console.

a. Navigate to the IOC file, and find the 'SYS' peripheral. Set the 'Debug' option to 'Serial Wire':



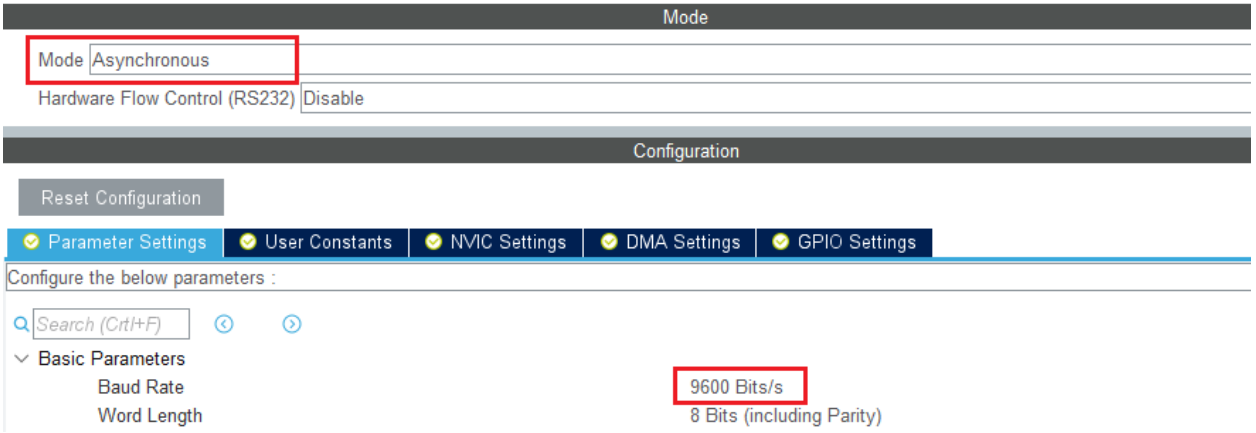
This will configure the board to send data transmitted to the UART3 peripheral through the debugger (but the pins and UART3 must be configured as well).

b. Under the 'Pinout' view, pin PD9 must be set to 'USART3_RX', and pin PD8 to 'USART3_TX'.



c. Back in the list of peripherals, navigate to ‘USART3’:

Set the mode to ‘Asynchronous’ to enable use of the peripheral. Additionally, set the ‘Baud Rate’ parameter to 9600. This value essentially specifies the rate of communication using this USART peripheral, and the PuTTY terminal will need to match it later on.



Save and regenerate the code and the USART initialization code will appear in main.c.

d. Add the following block of code (recommended between the ‘USER CODE BEGIN 4’ tags):

```
/* USER CODE BEGIN 4 */

void debugPrint(const uint8_t* _out){
    HAL_UART_Transmit(&huart3, (uint8_t*)_out, strlen(_out), 0xffffff);
}

/* USER CODE END 4 */
```

Additionally, a function prototype may be needed near the top of the file (recommended between ‘USER CODE BEGIN PFP’ tags):

```
/* USER CODE BEGIN PFP */

void debugPrint(const uint8_t* _out);

/* USER CODE END PFP */
```

As we configured the USART3 peripheral (referenced to by ‘huart3’ in the code), we built the ‘debugPrint’ function to output to the PuTTY terminal. Note it is also possible to configure the USART peripheral to take input from the PuTTY terminal as well (not covered here).

e. In the while loop in main() (or wherever else you like as long as it is after USART initialization), place debugPrint statements with your message of choice:

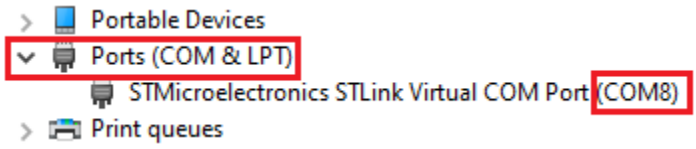
```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  debugPrint("This IDE is awful\r\n");

  HAL_GPIO_WritePin(BlueLED_GPIO_Port, BlueLED_Pin, GPIO_PIN_RESET);
  HAL_Delay(500);
  HAL_GPIO_WritePin(BlueLED_GPIO_Port, BlueLED_Pin, GPIO_PIN_SET);
  HAL_Delay(500);
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
```

f. Save and run the debug mode, and ensure the code blinks LEDs as before (we will capture the output next).

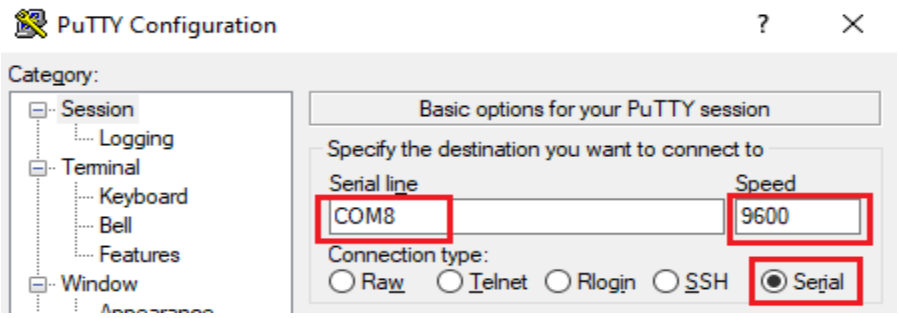
7. Now, the PuTTY terminal may be configured in order to capture the output.

a. With the Nucleo board still attached to your computer, on Windows, open up 'Device Manager' either through the Windows search bar or Control panel.

b. Look for the 'Ports' section, and expand it to find the attached Nucleo board. Take note of the COM port that is listed:



c. Open up PuTTY, and configure it as shown:



Ensure that the Speed option matches the baud rate that was previously set in the USART3 configuration. The Serial line option should also match the COM port that was found in the Device Manager.

d. Click the 'Open' button, and it should begin communication with the Nucleo board on USART3.

e. Run your blinking LED program on the board if it is not already running, and the debugPrint output should appear in the PuTTY terminal.