Avalon MM Slave : WS2812 Driver

Introduction:

This Intellectual Property Core is designed to drive WS2812b or SK6812 800 kHz addressable LEDs. It exposes a friendly interface on Avalon Intel FPGA MM bus. With 3 control registers and a LED color register bank. This 100% hardware component will drive your LEDs with 0% load on the CPU.
Can work with both HPS and NIOS based systems. Register data addressing is 32bits wide.
In Qsys, you can select Avalon width to accommodate with your maximum LED design number.
This driver should work with any 800 kHz addressable LED with only 1 data pin. It was tested on WS2812b LEDs and successfully interfaced with NIOS2 and HPS on DE0-Nano-Soc.

Driver HDL files:

1. **WS2812_Avalon.v :** Top-Level Avalon Verilog File. Useful for synthesis of the driver in Qsys.
2. **WS2812b_Driver.v :** Low-level Verilog IP core producing the actual driving signals for the LEDs
3. **DynCntModN.v :** Low-level utility IP core. Dynamic modulo counter with asynchronous reset and set functionality. Used to control LED data bus of the **WS2812b_Driver.v IP Core.**
4. **WS2812_Driver_hw.tcl** : TCL script, component description of the file for Qsys platform design software

Input and Outputs

Inputs	Outputs
Avalon MM Slave Bus : Avalon standard bus for Qsys components **Clk:** This signal should be a high speed clock to generate precise WS2812 timings **Reset:** Avalon reset signal, check Avalon specifications for more details	**DOUT :** Output pin to drive the LEDs, corresponds to a 800kHz PWM signal to drive WS2812b LEDs

Qsys - Platform Designer instantiation

Before instantiating the WS2812 driver under Qsys platform designer software, make sure you copy all ***.v** and ***.tcl** files to you *Quartus Prime Lite 18.1* project directory.

Place all 4 files under **$PROJECTDIR/hw/*all 4 files go here*.**

Now add all the ***.v** files to you Quartus project with the menu **Project-> Add/Remove files in project…** .
You can now open **Qsys** Platform designer software to create you system.
The **ws2812_driver** should appear under **IP Catalog**. From there, add the IP core to your design and connect the Avalon signals to the driver. Then, export the ws2812_dout signal to use on an external GPIO.

An example bellow has been created for HPS system on DE0-Nano-Soc with only 24 LEDs on the LED chain.

Component parameters:

- LED_MAX_NUMBER: defines the maximum number of LEDs that the driver can drive on the output. Internally, defines the size of the register bank that will hold your LED colors
- LED_ADDR_W: Defines the width of Avalon address bus. **Caution** you must set this to a value greater or equal to **CEIL (LOG2(LED_MAX_NUMBER+3))**. This will let Qsys minimize the address range needed by the driver.
- LED_DATA_W: Defines the width of LED color data registers. For WS2812 GRB LED, it is 24 bits. For RGBW LEDs like SK6812W, it is 32bits. **But beware that the driver has only been tested with GRB WS2812b/SK6812 regular 24 bits LEDs.**
- CLK_FREQUENCY: Avalon bus CLK frequency to let driver generate the correct LED driving timings based on this Clock value. Internally, it is used to count cycles and generate the correct timings by the WS2812b_driver.v IP core.

Qsys Configuration examples	
WS2812b LEDs **LED_MAX_NUMBER = 200** LOG2(200) = 7.64 CEIL(7.64) = 8 **LED_ADDR_W = 8** **LED_DATA_W = 24** **CLK_FREQUENCY = 50000000**	*WS2812b LEDs* **LED_MAX_NUMBER = 24 leds** LOG2(200) = 4.58 CEIL(4.58) = 5 **LED_ADDR_W = 5** **LED_DATA_W = 24** **CLK_FREQUENCY = 50000000**

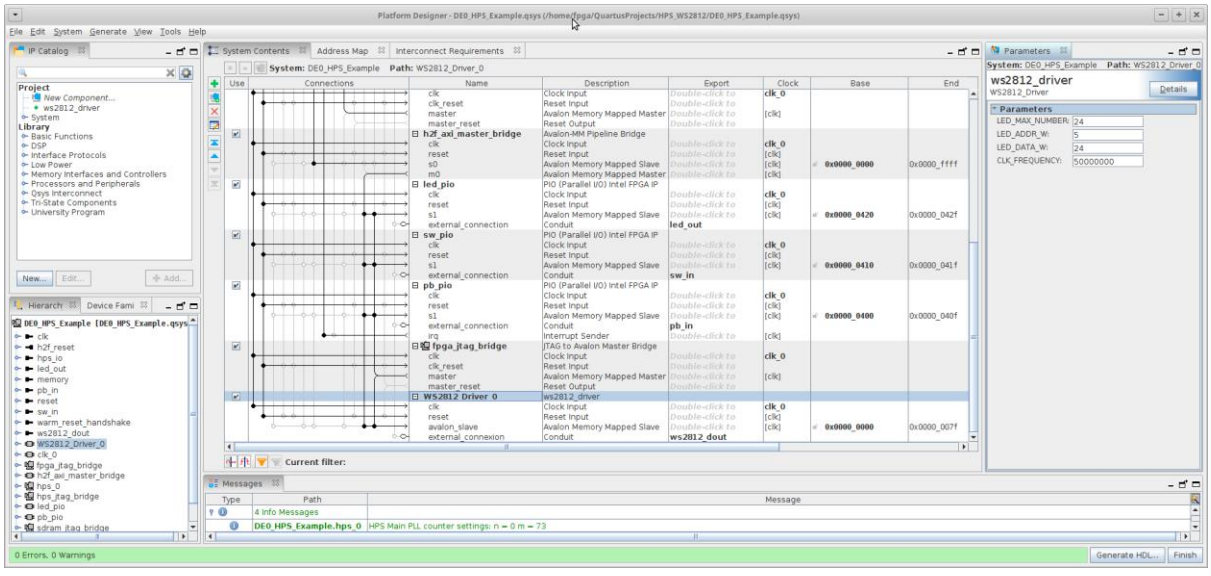

Figure 1: WS2812 Qsys instanciation in Quartus Prime Lite 18.1

Registers memory map:

The following abbreviations correspond to the following parameters in Qsys.

- **LW = LED_DATA_W**
- **N = LED_MAX_NUMBER**

Register assignment table

Register Address	Name	Access Mode	b31 Register content b0			
0	STATUS	R	X			b[0]=IDLE
1	CONTROL	RW	X		b[1]=SYNC	b[0]=RESET
2	LED_NUMBER	RW	X	b[LW-1:0]=LED_COUNT		
3	LED_0_DATA	RW	X	b[LW-1:0]=LED_DATA[0]		
3 + i	LED_i_DATA	RW	X	b[LW-1:0]=LED_DATA[i]		
3 + (N-1)	LED_(N-1)_DATA	RW	X	b[LW-1:0]=LED_DATA[N-1]		

Register	Description
STATUS	You can only read the **IDLE** bit of this register. When **IDLE** is 1, then the driver is ready to receive a **LED_NUMBER** configuration or a **SYNC** order. When **IDLE = 0** then the driver is busy latching data to the output pin **DOUT**. Usually, you will use this bit to wait for the driver before updating the **LED_DATA** registers content and requesting a new **SYNC** request when you are done setting the **LED_DATA** registers.
CONTROL	When writing 1 to **RESET** bit of this register, the WS2812 signal generation circuitry gets reset. It means that after that, you will have to reconfigure the LED_NUMBER of you led chain. When writing to 1 to **SYNC** bit to this register, the WS2812 signal generations starts latching the LED_DATA to the **DOUT** pin of the driver. Reading CONTROL register **RESET** bit corresponds to reading the actual **RESET** signal of the WS2812 signal generation circuitry. Same goes for **SYNC** bit.
LED_NUMBER	Writing to this register sets the actual LED count of the LED chain. This should be less or equal to LED_MAX_NUMBER. **You need to check that the driver is in IDLE state before writing to the register.** **This is mandatory to make sure parameters are applied.** For this, you can either wait for **STATUS register IDLE bit to become 1** or issued a reset request by writing 1 to the **RESET bit which will force the driver into IDLE state.** And then write your led counts in the chain to this LED_NUMBER register.
LED_i_DATA	For **i** between 0 and N-1 (included) corresponds to the actual data set to the **ith LED** in the WS2812 chain. Beware that for WS2812, this data is 24 bits **Green Red Blue** order MSB to LSB respectively.

Ressources:

- Avalon MM bus specifications: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/manual/mnl_avalon_spec.pdf