Blinky LED on KV260 Using AXI-Lite and Python (Ubuntu)

1. Objective

To control the brightness of an LED connected to the PMOD of KV260 using:

- Custom Verilog PWM IP
- AXI-Lite interface
- Zynq PS–PL communication
- Python script running on Ubuntu

---

2. Planning and Design Flow

Overall Flow

```
Python (Ubuntu on KV260)
        ↓
AXI-Lite Write (/dev/mem)
        ↓
Custom AXI IP (PL)
        ↓
PWM Logic
        ↓
PMOD LED
```

---

3. Required Files and Tools

Tools

- Vivado 2023.1
- Ubuntu running on KV260
- Python3 (already available on Ubuntu)

RTL Files

- `<filename>.v` → PWM logic
- `myip_v1_0.v` → AXI IP top

- `myip_v1_0_S00_AXI.v` → AXI-Lite slave

---

4. Step 1: Create PWM Logic (PL RTL)

File to create

`<filename>.v`

Purpose

- Generate PWM signal based on brightness value
- Brightness range: 0–255

Key signals

- `clk`
- `reset`
- `enable`
- `brightness[7:0]`
- `pwm_out`

---

5. Step 2: Create AXI-Lite IP

5.1 Create IP Project

1. Open Vivado
2. **Tools → Create and Package New IP**
3. Select **AXI4-Lite Peripheral**
4. Set:
    - Registers: **4**
    - No interrupts

5.2 Edit AXI IP Files

File 1: `myip_v1_0_S00_AXI.v`

Purpose: AXI register logic

Add outputs in the output declration:

```
output wire enable;
output wire [7:0] brightness;
```

Connect registers:

```
assign enable     = slv_reg0[0];
assign brightness = slv_reg1[7:0];
```

File 2: `myip_v1_0.v`

Purpose: Glue AXI + PWM logic

Add output:

```
output wire pwm_out;
```

Instantiate PWM:

```
blinky_driver u_pwm (
    .clk        (s00_axi_aclk),
    .reset      (~s00_axi_aresetn),
    .enable     (enable),
    .brightness (brightness),
    .pwm_out    (pwm_out)
);
```

---

6. Step 3: Add PWM RTL to IP Packager (IMPORTANT)

File to add

`<filename>.v`

Exact steps

1. Open **IP project**
2. Click **Package IP**
3. Go to **File Groups**
4. Expand:

   `Advanced → Verilog Synthesis`
5. Select **Verilog Synthesis**
6. Click **Add (+)**
7. Change **Files of type** to:

   `Verilog Files (*.v) or All Files (*)`
8. Select <filename>`.v`
9. Repeat for **Verilog Simulation**

Re-package IP

- **Review and Package → Re-Package IP**

---

7. Step 4: Block Design Integration

7.1 Create Block Design

1. Open hardware project
2. **Create Block Design**

7.2 Add Blocks

- Zynq UltraScale+ MPSoC
- Custom AXI IP
- AXI Interconnect
- Processor System Reset

7.3 Run Connection Automation

- Use **FPD AXI**
- Accept all defaults

7.4 Make PWM Output External

- Right-click `pwm_out`
- **Make External**

---

8. Step 5: Address Assignment

Open **Address Editor**

You will see:

```
myip_0 @ 0xA000_0000
```

Register Map

Function	Address
Enable	0xA0000000
Brightness	0xA0000004

---

9. Step 6: XDC Constraints (PMOD)

Create constraint file:

`blinky_pmod.xdc`

Example:

```
set_property PACKAGE_PIN <PMOD_PIN> [get_ports pwm_out_0]
set_property IOSTANDARD LVCMOS33 [get_ports pwm_out_0]
```

Step-by-step method to check the pmod pin

1 Refer the Kria schematic

- Open this file:
  `038-05101-01_sck-kr_reva02_SKIT_20211015.pdf`
- In the schematic, locate:
  - **PMOD1**
  - **PMOD2**
  - **USER LEDs**
- For each PMOD or LED signal, **note the SOM ball name**
  Example: `SOM240_1`, `SOM240_2`, etc.

---

2 Check the K26 SOM datasheet

- Open this file:
  `ds987-k26-som.pdf`
- Search for the **SOM ball name** you noted from the schematic
  Example:
  - `SOM240_1 → D14`
- This step converts **SOM ball name → FPGA package pin**

---

3 Verify using the constraint file

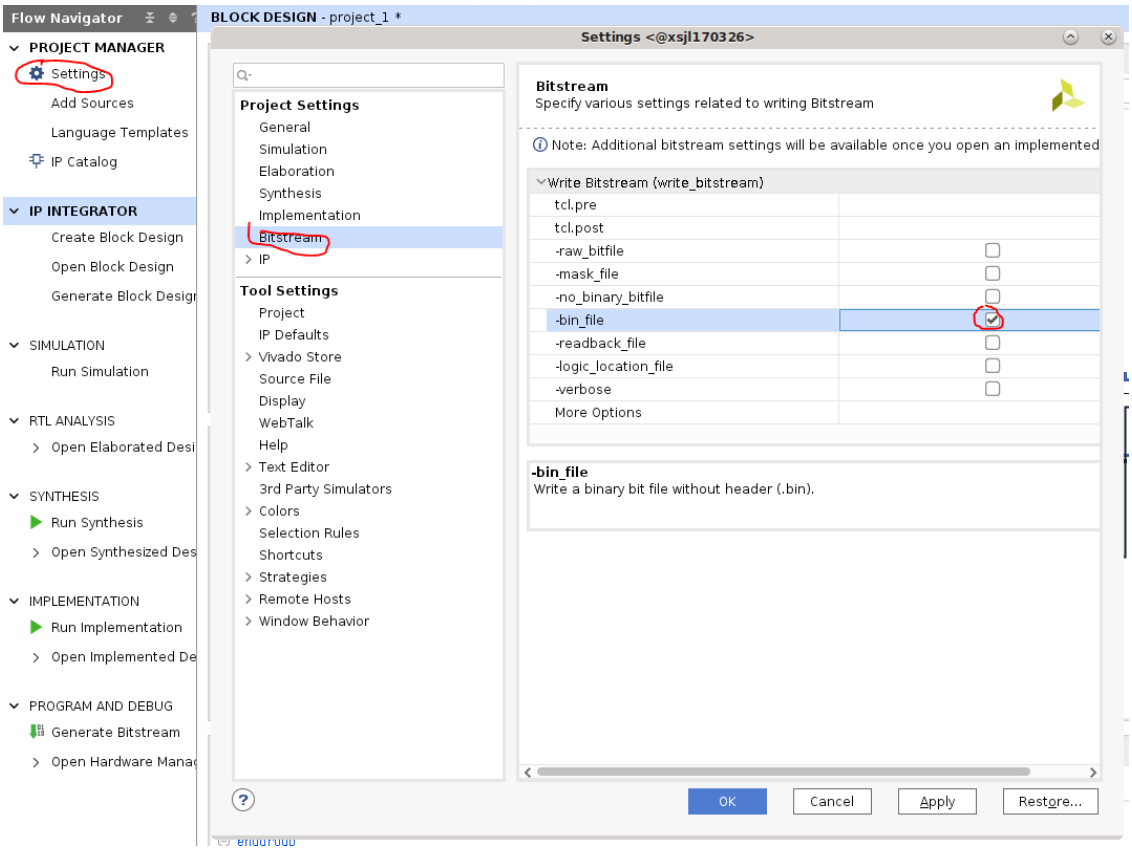
- Open the XDC file:
  `Kria_K26_SOM_Rev1.xdc`
- Search for the FPGA pin name (example: `D14`)
- You will find the exact constraint like:
  - `PACKAGE_PIN D14`
  - Corresponding I/O standard and signal name

Final conclusion

Schematic → gives **SOM ball name**
Datasheet → maps **SOM ball** → **FPGA pin**
XDC file → confirms **actual pin constraint**

follow the below screenshoot

Tools-->settings-->Bitstream-->chech the checkbox of *bin_file



---

10. Step 7: Generate Bitstream

1. **Validate Design**
2. **after enabling the checkbox as shown in above picture**
3. **Generate Bitstream**

---

11. Step 8: Export Hardware

1. **File → Export → Export Hardware**

2. Enable **Include Bitstream**
3. Finish

After generating the bitstream in Vivado, then rename the generated `.bin` file from <projectname>.`runs/impl_1` to **`.bit.bin`** and export the design as a `.xsa` file.

mv <design_name>.bin blinky_ip.bit.bin

12.Generate Device Tree Overlay (.dtbo)

Launch XSCT:

source /tools/Xilinx/Vitis/2023.1/settings64.sh

xsct

run the command one by one make necassary path changes according to your sysytem:

hsi open_hw_design blinky_ip.xsa

hsi set_repo_path /home/<user>/device-tree-xlnx

hsi create_sw_design device-tree -os device_tree -proc psu_cortexa53_0

hsi set_property CONFIG.dt_overlay true [hsi::get_os]

hsi generate_target -dir blinky_dts

hsi close_hw_design

exit

Compile the overlay:

cd blinky_dts

dtc -@ -O dtb -o blinky_ip.dtbo pl.dtsi

shell.json

nano shell.json

paste:

```
{
      "shell_type" : "XRT_FLAT",
      "num_slots" : "1"
}
```

13.Copy Files to KR260

Required files:

1.blinky_ip.bit.bin

2.blinky_ip.dtbo

3.shell.json

scp blinky_ip.bit.bin blinky_ip.dtbo shell.json ubuntu@<kr260-ip>:/home/ubuntu/

14.Create the firmware directory (ON KR260)

sudo mkdir -p /lib/firmware/xilinx/Blinky_driver

***sudo cp Blinky_driver.bit.bin Blinky_driver.dtbo shell.json ***

/lib/firmware/xilinx/Blinky_driver/

Your ***app name is*** ***`Blinky_driver`*** *so your files also has same name like above(except shell.json)*

Verify xmutil can see your app

sudo xmutil listapps

Load the application (FINAL STEP)

sudo xmutil unloadapp

sudo xmutil loadapp Blinky_driver

15 python script

nano blinky_test.py

paste :

```
  #!/usr/bin/env python3

import mmap
import os
import struct
import time

# AXI base address (from device tree)
AXI_BASE_ADDR = 0xA0000000
MAP_SIZE = 0x1000  # 4KB AXI-Lite space

# Register offsets (from RTL)
REG_ENABLE     = 0x00  # slv_reg0
REG_BRIGHTNESS = 0x04  # slv_reg1

# Open /dev/mem
fd = os.open("/dev/mem", os.O_RDWR | os.O_SYNC)

# Map AXI space
mem = mmap.mmap(
    fd,
    MAP_SIZE,
    mmap.MAP_SHARED,
    mmap.PROT_READ | mmap.PROT_WRITE,
    offset=AXI_BASE_ADDR
)
```

```
def write_reg(offset, value):
    mem.seek(offset)
    mem.write(struct.pack("<I", value))

try:
    # Enable PWM
    write_reg(REG_ENABLE, 1)
    print("PWM Enabled")
    time.sleep(1)

    # Sweep brightness
    for b in [0, 32, 64, 128, 192, 255]:
        write_reg(REG_BRIGHTNESS, b)
        print(f"Brightness = {b}")
        time.sleep(2)

    # Disable PWM
    write_reg(REG_ENABLE, 0)
    print("PWM Disabled")

finally:
    mem.close()
    os.close(fd)
```

Run script

chmod +x blinky_test.py

sudo python3 blinky_test.py

---

14. Test Setup & Output

14.1 Using DSO (Digital Storage Oscilloscope)

- Connect the **DSO ground probe** to the **PMOD ground pin** using a jumper wire.
- Connect the **DSO positive probe** to the **PMOD signal pin** (PWM output).
- Set the **voltage scale** and **time scale** properly on the DSO.
- Observe the **PWM waveform** and check the **pulse width (duty cycle)**.

14.2 Using Onboard User-Defined LED

- The **onboard user LED** will **glow according to the brightness value**.
- Higher brightness value → LED glows brighter
- Lower brightness value → LED glows dimmer

---

15. Common Mistakes and How to Avoid

Mistake	Solution
`.v` file not found	Add RTL in **IP Packager → File Groups**
PWM pin not visible	Re-package IP and upgrade in BD
LED not glowing	Check XDC PMOD pin
Python error	Always run with `sudo`
AXI not working	Verify base address

---

16. Conclusion

This project demonstrates:

- Custom RTL design
- AXI-Lite control
- PS–PL communication
- Python hardware control on KV260

This is a **complete and correct SoC design flow**.

END OF DOCUMENT

https://xilinx.github.io/kria-apps-docs/creating_applications/2022.1/build/html/index.html