Experiment 2
PS based Led-Switch Controller

Kaushik Balaji M S *M.Tech in System-on-chip Design*
Indian Institute of Technology, Palakkad
152502010@smail.iitpkd.ac.in

I. Introduction

The ZYNQ7 Processing system is a dual core ZynQ-7000 ARM Cortex-A9 processor with memory controllers, and peripherals. In this experiment, the PS runs a standalone application developed using Vitis. The application executes on the processor and performs memory-mapped read and write operations to control hardware peripherals implemented in the Programmable Logic (PL).

The Advanced eXtensible Interface (AXI) is a high-performance on-chip communication protocol used for data transfer between the PS and PL. In this design, the AXI-Lite interface is used as a slave module, to be controlled by the inputs from the PS(master).

II. Steps used and Observations made

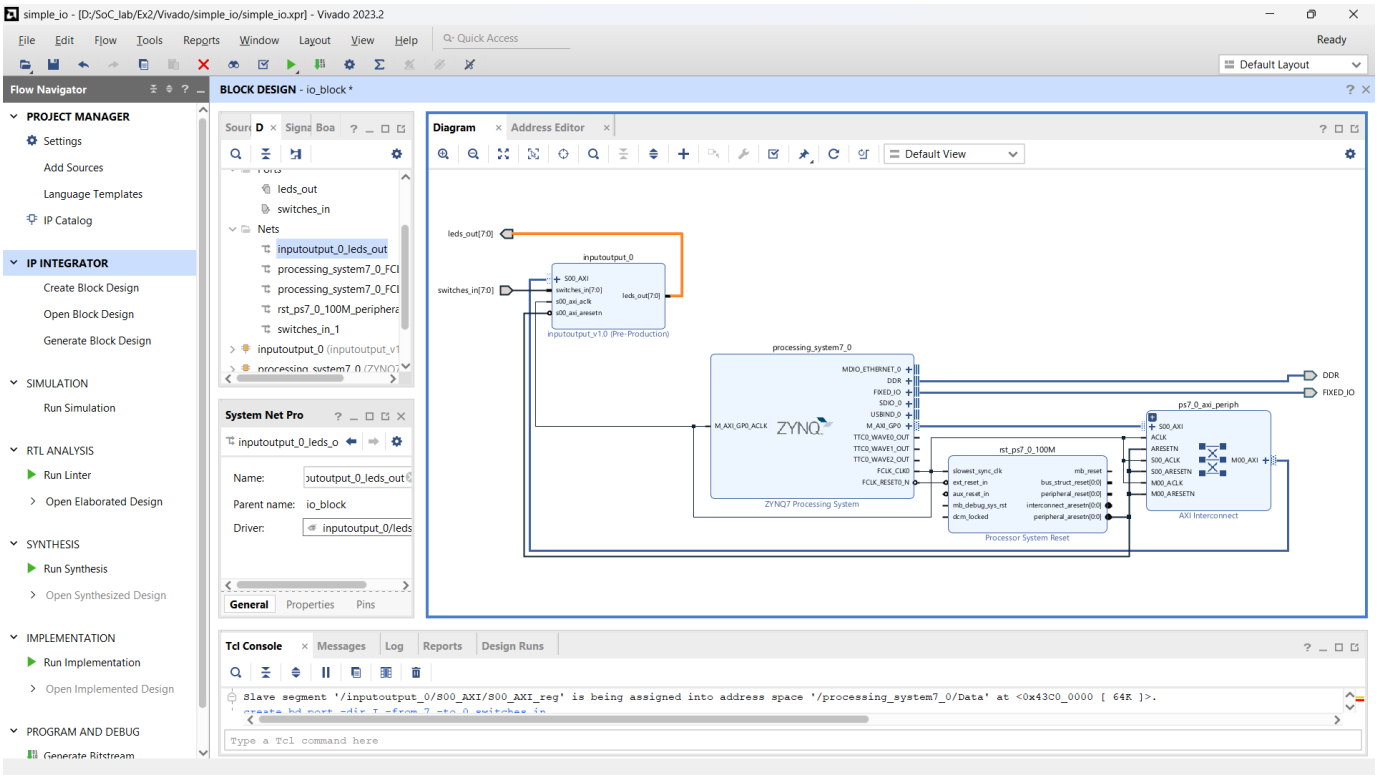

Fig. 1. Processing system after connecting to the AXI4 IP

1) **Slice LUTs**: A small number of LUTs are used to implement the AXI4-Lite interface and simple control logic, resulting in very low utilization.

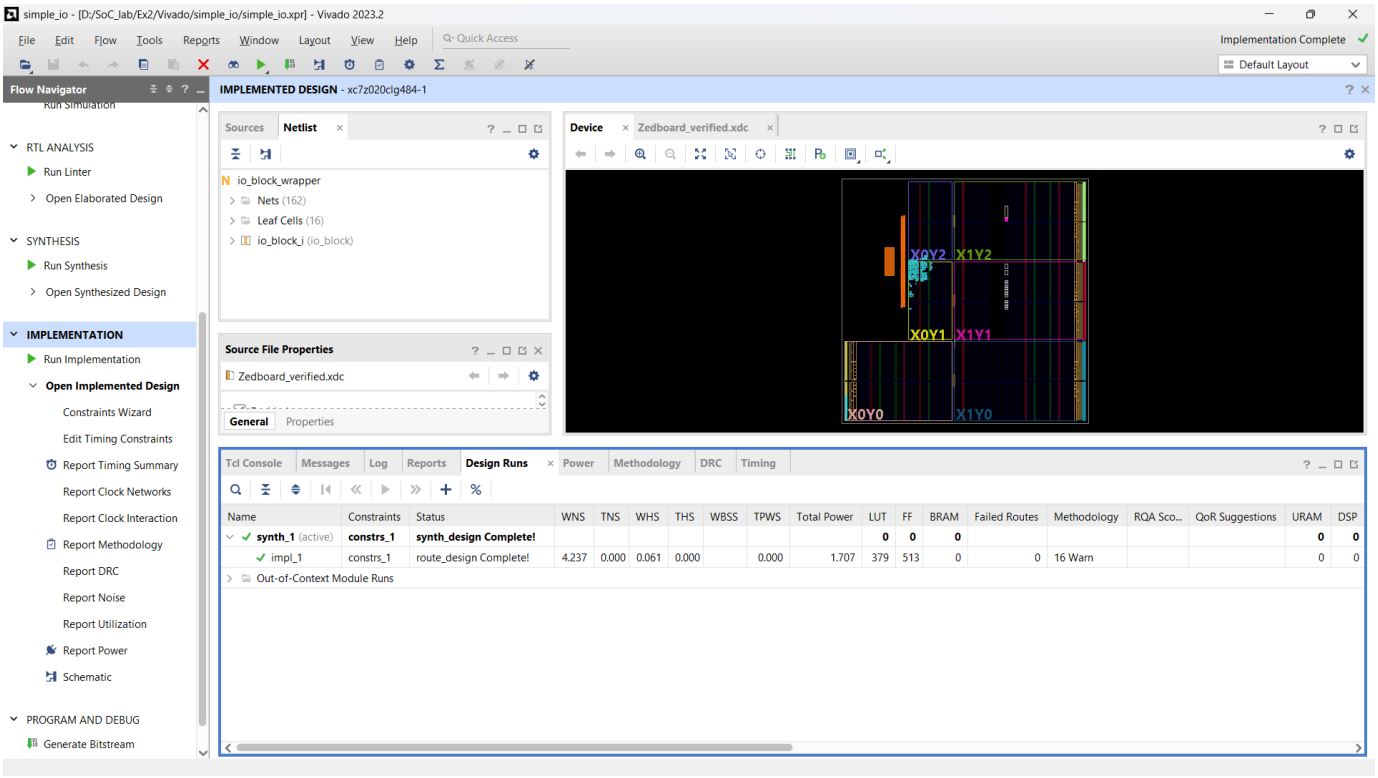

Fig. 2. Implemented design

TABLE I
FPGA RESOURCE UTILIZATION SUMMARY

Resource	Used	Total	Utilization (%)
Slice LUTs	25	53200	0.05
Slice Registers	0	106400	0.00
F7 Muxes	0	26600	0.00
F8 Muxes	0	13300	0.00

2) **Slice Registers**: No registers are required since the design contains minimal sequential logic and no pipelined or state-heavy modules.

Once the Implementation is done as in Figure 2, modify the Constraint voltage in the Constraints wizard and run the implementation again to see the difference.

The constraints file used for synthesis is given as:

```
# ------------------------------------------------------------
# User LEDs - Bank 33
# ------------------------------------------------------------
set_property PACKAGE_PIN T22 [get_ports {leds_out[0]}];  # "LD0"
set_property PACKAGE_PIN T21 [get_ports {leds_out[1]}];  # "LD1"
set_property PACKAGE_PIN U22 [get_ports {leds_out[2]}];  # "LD2"
set_property PACKAGE_PIN U21 [get_ports {leds_out[3]}];  # "LD3"
set_property PACKAGE_PIN V22 [get_ports {leds_out[4]}];  # "LD4"
set_property PACKAGE_PIN W22 [get_ports {leds_out[5]}];  # "LD5"
set_property PACKAGE_PIN U19 [get_ports {leds_out[6]}];  # "LD6"
set_property PACKAGE_PIN U14 [get_ports {leds_out[7]}];  # "LD7"

# ------------------------------------------------------------
# User DIP Switches - Bank 35
# ------------------------------------------------------------
```

```
set_property PACKAGE_PIN F22 [get_ports {switches_in[0]}];  # "SW0"
set_property PACKAGE_PIN G22 [get_ports {switches_in[1]}];  # "SW1"
set_property PACKAGE_PIN H22 [get_ports {switches_in[2]}];  # "SW2"
set_property PACKAGE_PIN F21 [get_ports {switches_in[3]}];  # "SW3"
set_property PACKAGE_PIN H19 [get_ports {switches_in[4]}];  # "SW4"
set_property PACKAGE_PIN H18 [get_ports {switches_in[5]}];  # "SW5"
set_property PACKAGE_PIN H17 [get_ports {switches_in[6]}];  # "SW6"
set_property PACKAGE_PIN M15 [get_ports {switches_in[7]}];  # "SW7"

# Note that the bank voltage for IO Bank 13 is fixed to 3.3V on ZedBoard.

set_property IOSTANDARD LVCMOS33 [get_ports {leds_out[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {leds_out[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {leds_out[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {leds_out[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {leds_out[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {leds_out[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {leds_out[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {leds_out[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {switches_in[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {switches_in[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {switches_in[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {switches_in[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {switches_in[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {switches_in[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {switches_in[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {switches_in[0]}]
```

Listing 1. Constraints file

Once the Synthesis ⟶ Implementation ⟶ Bitstream Generation is done, export the Hardware.

In Vitis, use the previously exported hardware (.xsa file) to generate a platform for the application to run.

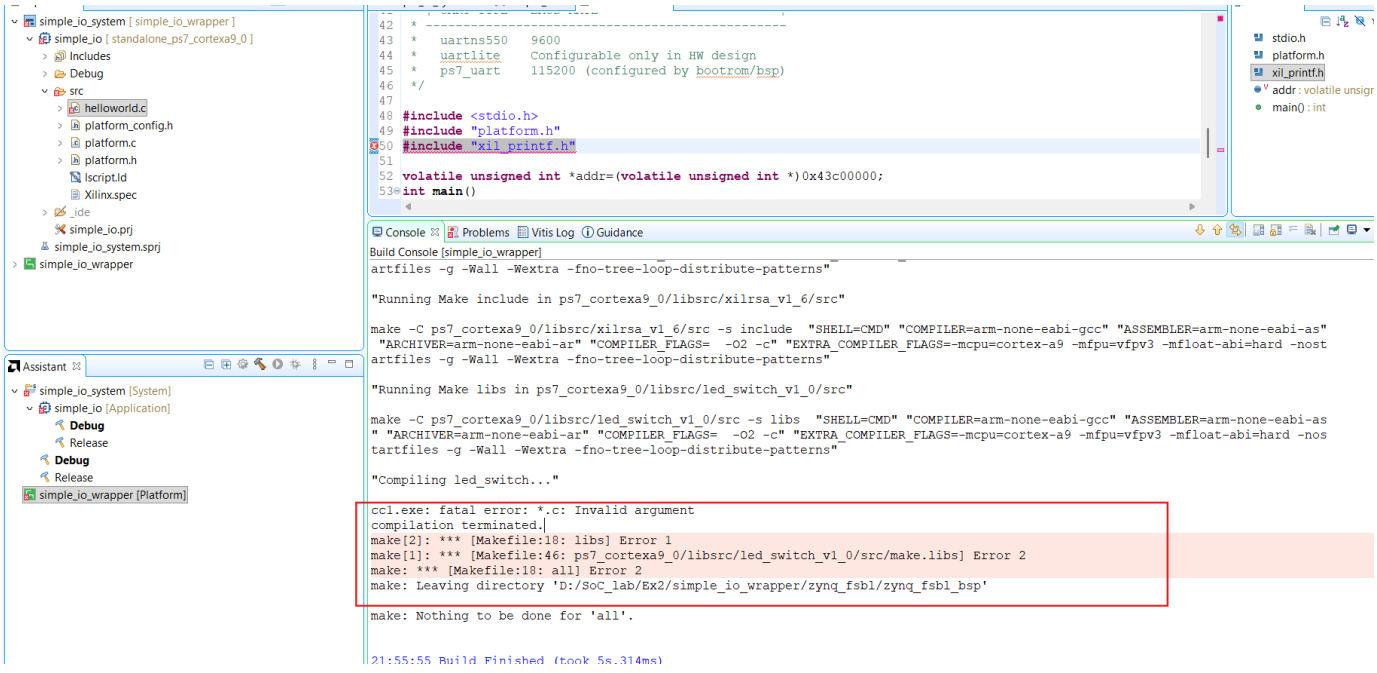

Fig. 3. Error when building

When building the imported platform, errors will be found on several Makefiles generated by Vivado. The errors shown in Figure 3 can be fixed by modifying the, Makefile located in

```
zynq_fsbl\zynq_fsbl_bsp\ps7_cortexa9_0\libsrc\led_switch_v1_0\src

hw\drivers\led_switch_v1_0\src

ps7_cortexa9_0\standalone_ps7_cortexa9_0\bsp\ps7_cortexa9_0\libsrc\led_switch_v1_0\src
```

Modifying the Makefiles in these paths from lines:

```
LIBSOURCES= *.c
OUTS = *.o
```

to the lines:

```
LIBSOURCES=$(wildcard *.c)
OUTS = $(wildcard *.o)
```

```
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"

volatile unsigned int *addr=(volatile unsigned int *)0x43c00000;
int main()
{
    init_platform();
    if(addr[2]==0xaabbccdd)print("success");
    print("Hello World\n\r");
    print("Successfully ran Hello World application");
    while(1)addr[0]=addr[1];
    cleanup_platform();
    return 0;
}
```

Listing 2. Hello world

The lines *.c works on Linux sometimes, because linux shell expands *.c $\longrightarrow$ to file1.c, file2.c, etc.

This fails on Windows system because the GCC compiler passes *.c to cc1.exe and cc1.exe tries to find a file named literally as "*.c", thus throwing a fatal error:

cc1.exe: fatal error: *.c: Invalid argument

After modifying the Makefile as mentioned above, the build runs perfectly.

There might be an error occurring due to *qemu_args.txt*. This is due to the Vitis trying to build the hardware for x86 processor (CISC Architecture) while the Hardware we have is Zedboard, which has Dual-core ARM COrtex9 processor, which are of RISC architecture. So simply adding an empty *qemu_args.txt* to the location is a temporary fix for this.

After Platform Build is successful, build the Application, then the System (in the same order).

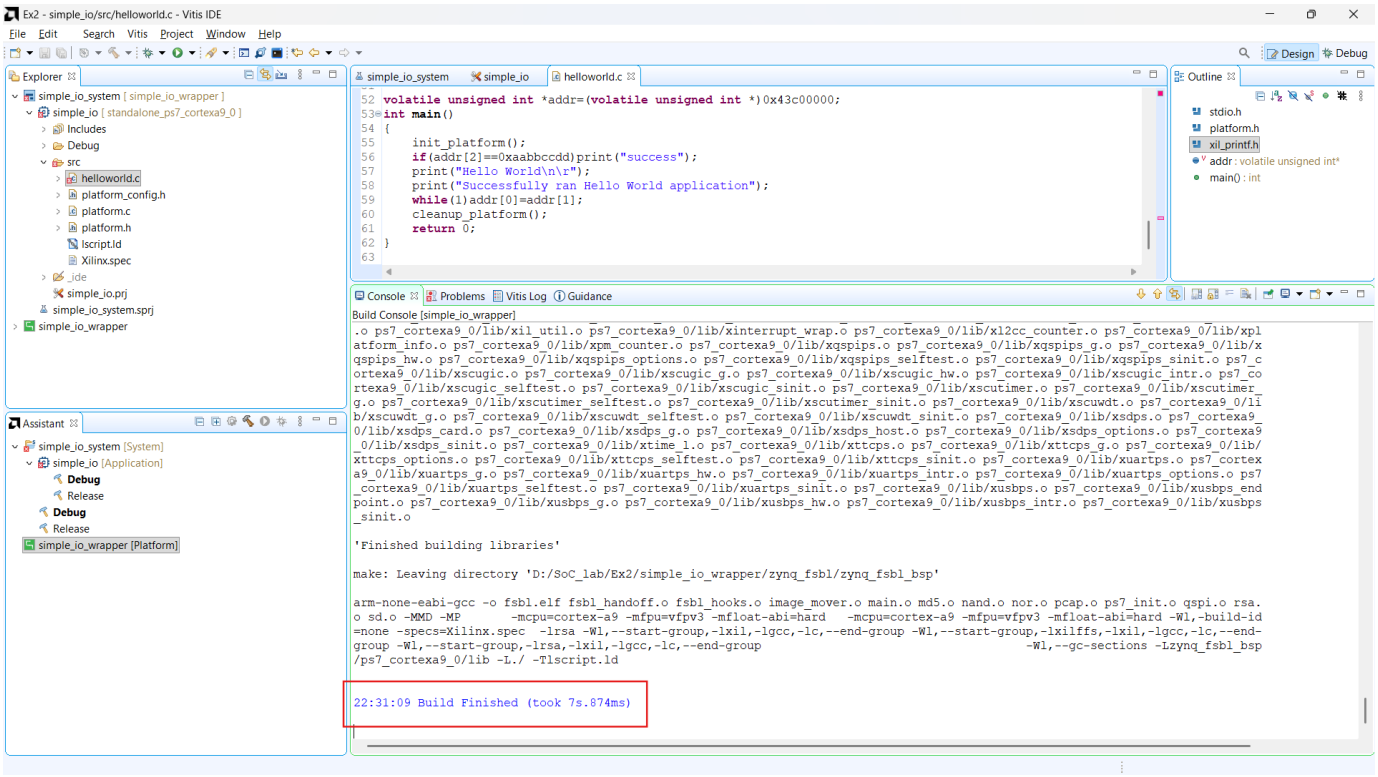

Fig. 4. Platform build done successfully

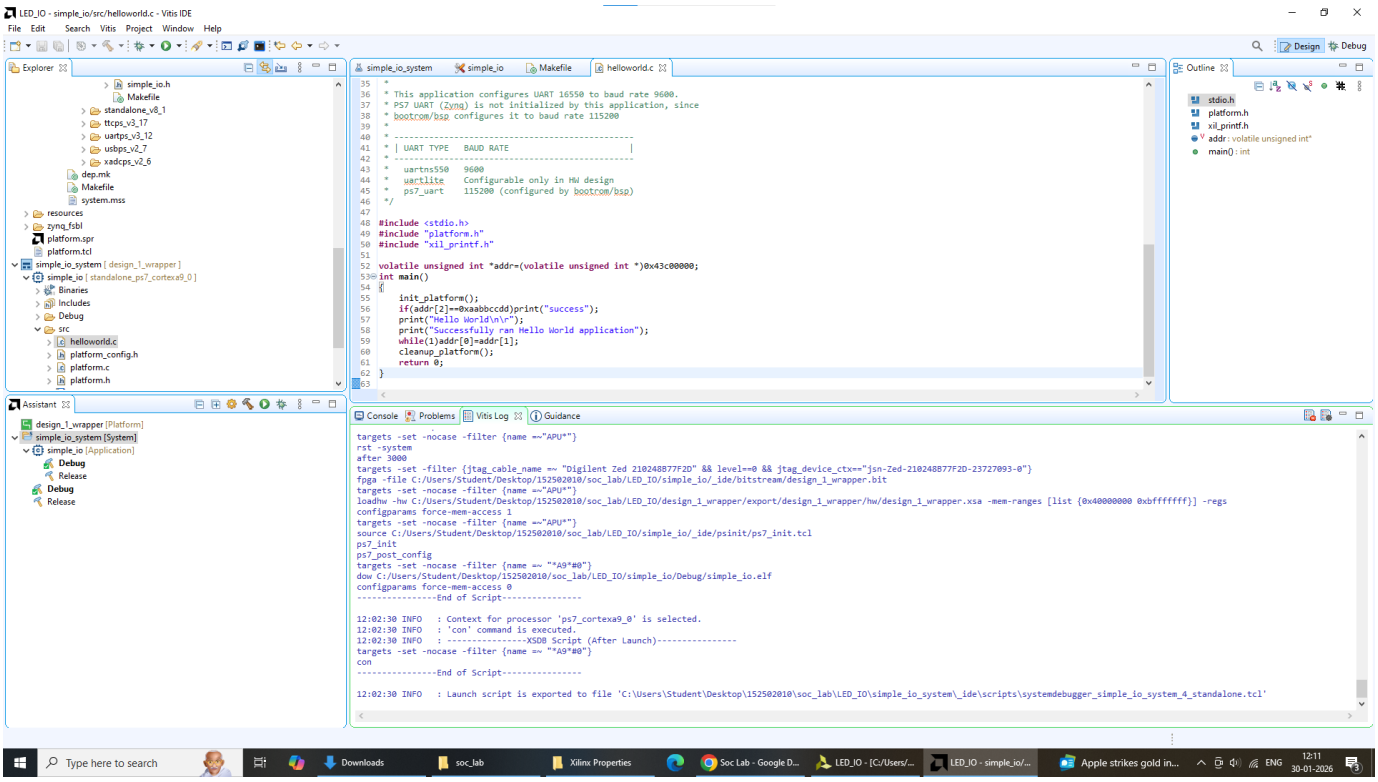

Fig. 5. Simple IO operation running on Zedboard

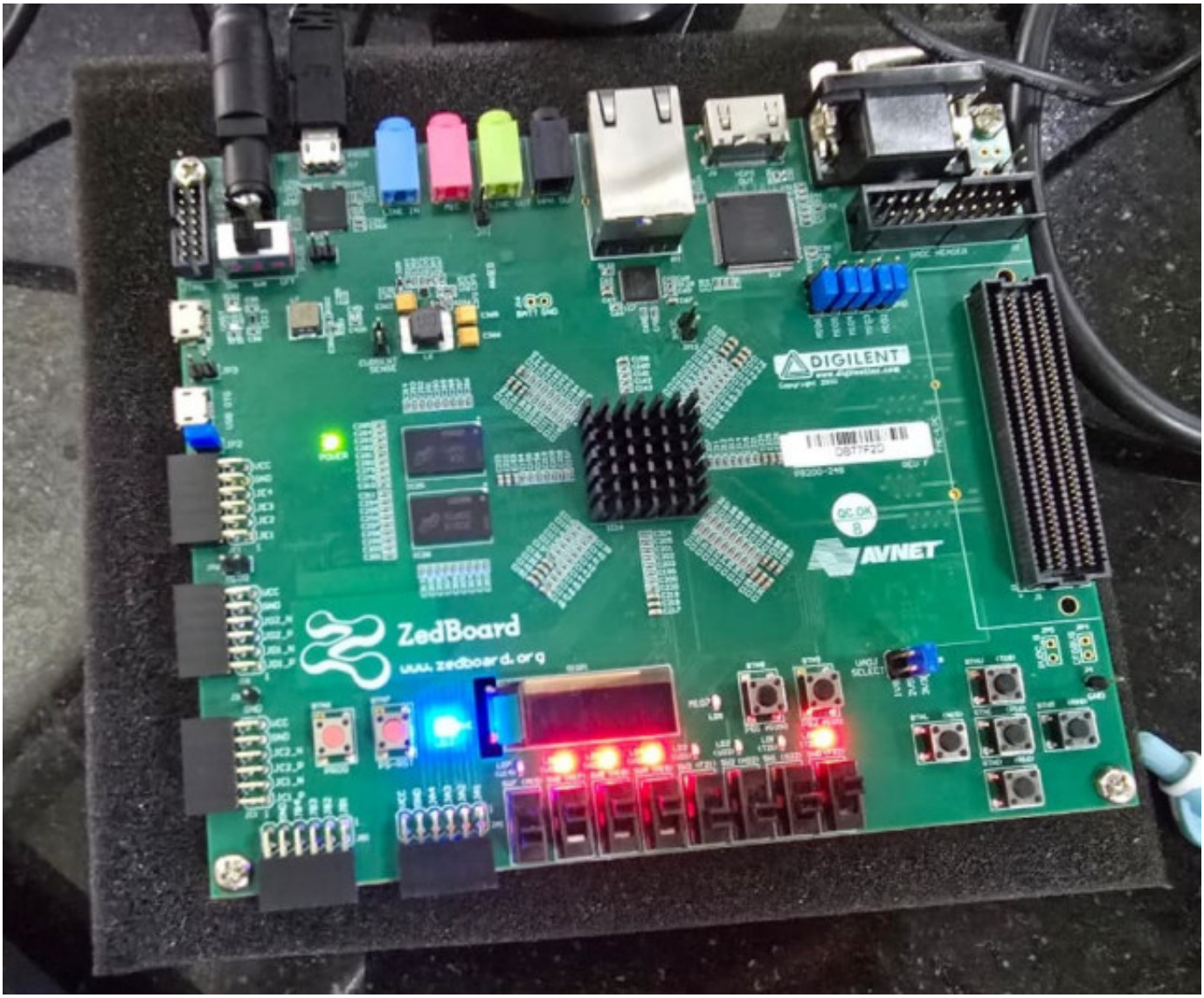

Fig. 6. SWITCH ⟶ LED interaction on Zedboard

III. Conclusion

A PS-based LED–Switch controller was successfully implemented on the Zynq-7000 platform by integrating a custom AXI4-Lite peripheral with the Processing System. The correct operation of memory-mapped I/O was verified by reading the inputs via Dip switches, and driving outputs throught LEDs through software running on the the ARM Cortex processor of Zedboard.