

ECE 441/543 Lab 4 Report

Soft Core Processors and IP Blocks

1st Khephren Gould
V01012827

*Dept. of Electrical and Computer Engineering
University of Victoria
khephrengould@uvic.ca*

2nd Zarina Guzman
V01075558

*Dept. of Electrical and Computer Engineering
University of Victoria
zguzman@uvic.ca*

Abstract—This lab examined the use of the Vivado IP Library to synthesize soft core microcontrollers and microprocessors on an FPGA. Both the MicroBlaze processor and microcontroller were synthesized. The Vivado SDK was used to develop compile and load C programs onto the synthesized Microblaze systems. An ILA was used to probe signals internal to an 8-bit counter synthesized on the FPGA.

Index Terms—Microprocessor, Microcontroller, Vivado, FPGA, ILA, UART, GPIO, C, AXI, MicroBlaze

I. INTRODUCTION

Soft Core Processors have many applications including SoC based CPU-FPGA systems, ASIC prototyping, and embedded applications which require custom computing hardware (intensive DSP computations etc.). The Xilinx MicroBlaze Soft Core processor is available within the Vivado IP library and was used for the experiments in this lab. The MicroBlaze is a Harvard architecture CPU that implements the RISC ISA. In this lab, both the stand-alone MicroBlaze Processor as well as the complete Microblaze Microprocessor were synthesized on an Artix-7 FPGA. Additional features of the Vivado IP library were used including peripheral IP blocks for UART and GPIO. Furthermore, an industry standard debugging tool called an ILA (integrated logic analyzer) was synthesized and used to analyze a simple 8-bit counter circuit.

II. IN LAB STEPS

A. Stand alone MicroBlaze Microcontroller Instantiation - hello world

In this step, a complete microcontroller was synthesized and programmed on the lab's Artix 7 FPGA. The MicroBlaze microcontroller contains a RISC soft core processor that interfaces with many built in peripherals (UART, GPIO interfaces, Block RAM memory etc.).

First, the built-in IP Library from Vivado was used to configure and instantiate an entity for the microcontroller. Eight GPI and GPO pins were each enabled, the UART interface was enabled at 9600 baud and a clock frequency of 100MHz was selected.

After instantiating the microcontroller IP block, the Vivado SDK was used to write a small C Program. The C code was compiled and included in the bitstream used for programming the FPGA. The C-code used for this step is shown in Figure 1. As shown in the figure, the code prints the text "Hello" to the screen 5 times using a loop.

Next, the bitstream was generated and used to program the FPGA. The results from connecting to the microcontroller's UART interface were viewed using the PuTTY serial console. This is shown in Figure 2.

Finally, a C program for interacting with the GPIO interface of the microcontroller was compiled and the FPGA was programmed. The program simply reads the state of the GPI inputs (connected to the switches) and writes this state to the GPO outputs (connected to the LEDs) in a loop. The results are shown in Figure 3.

B. MicroBlaze system using IP blocks - AXI UART helloworld

In this step, a MicroBlaze Processor was instantiated and programmed onto the FPGA. As opposed to a Microcontroller, a Processor does not contain any built in communication peripherals.

This step used the Vivado IP Block Design tool. This tool allows the designer to configure their system graphically. It also contains automations for routing the interconnections between the IP blocks.

First, a block for the system clock was added to the design. Next, a Microblaze processor block and a UART interface block were added. In this case, the UART interface communicates with the processor using the AXI protocol.

As in step 1, the Vivado SDK was used to write a simple C program. For this step, the program consists of a loop that prints the loop index at each iteration. The code along with the results from programming the FPGA and monitoring the serial console are shown in Figures 5 and 6 respectively.

C. Extending the MicroBlaze system to include AXI GPIO

In step 3, a GPIO peripheral was added to the processor. As with the UART peripheral in step 2, the GPIO interface communicates with the processor using the AXI protocol. The

```

/*
 * helloworld.c: simple test application
 *
 * This application configures UART 16550 to baud rate 9600.
 * PS7 UART (Zynq) is not initialized by this application, since
 * bootrom/bsp configures it to baud rate 115200
 *
 * -----
 * | UART TYPE   BAUD RATE |
 * -----
 * * uartns550   9600
 * * uartrlite   Configurable only in HW design
 * * ps7_uart    115200 (configured by bootrom/bsp)
 */

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

int main()
{
    init_platform();
    for(int i = 0; i < 5; ++i){
        print("Hello");
    }

    cleanup_platform();
    return 0;
}

```

Fig. 1: "Hello World" C Program

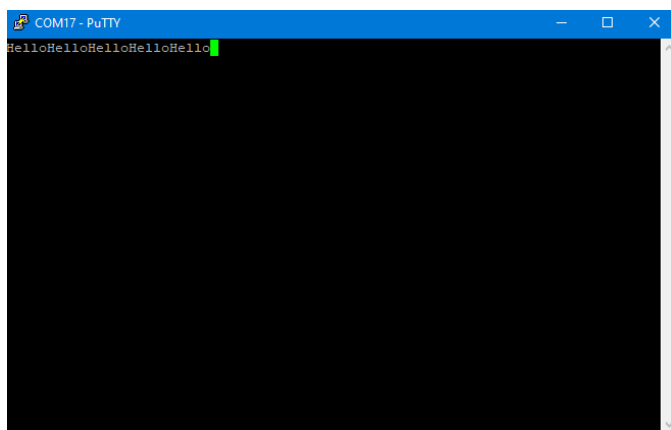


Fig. 2: "Hello World" UART Reception

GPIO peripheral was added to the design using the Vivado IP Block Design Tool. Because the project was created using the Nexys 7 Development board template, the Block Design Tool could be used to connect the GPIO peripheral to the LEDs and slide switches directly. The completed schematic for the system is shown in Figure ??.

After adding the GPIO peripheral to the IP block design, the Vivado SDK was used to create a C Program to interact with it. The program monitors the state of the slide switches and writes a string over the UART interface. The string contains the hexadecimal representation of the current value contained on the switches. Additionally, the program writes the state of the switches to the LEDs on the Nexys development board. The serial console output as a result of running the C program

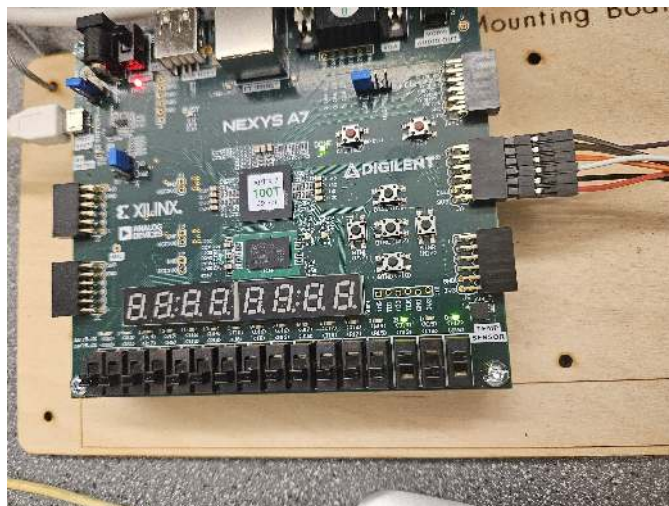


Fig. 3: Microcontroller GPIO

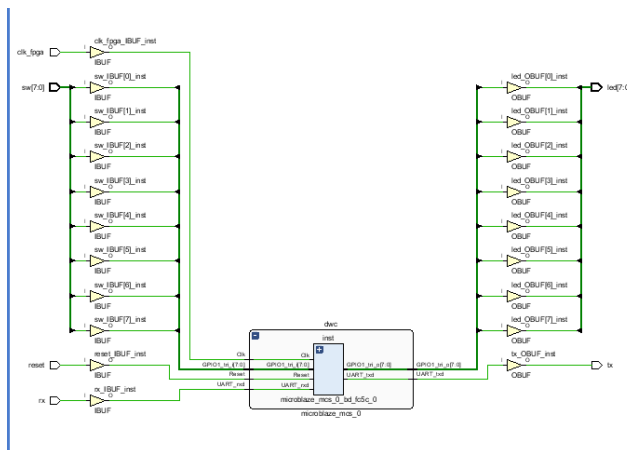


Fig. 4: Step 1 System Schematic

```

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

int main()
{
    init_platform();
    for(int i = 0; i < 5; i++){
        xil_printf("i: %d\n\r", i);
    }

    cleanup_platform();
    return 0;
}

```

Fig. 5: Step 2 C Program

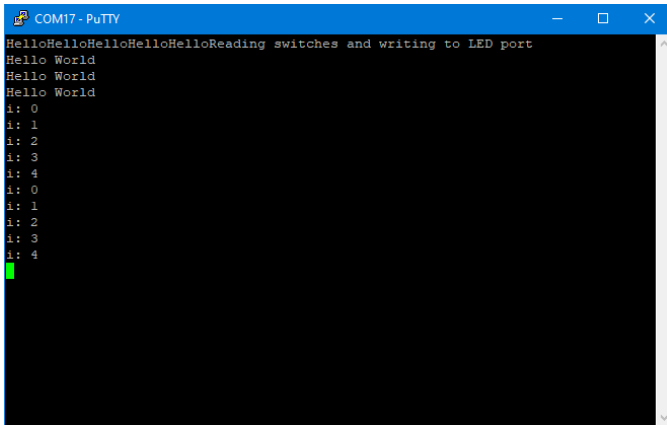


Fig. 6: Step 2 Serial Console

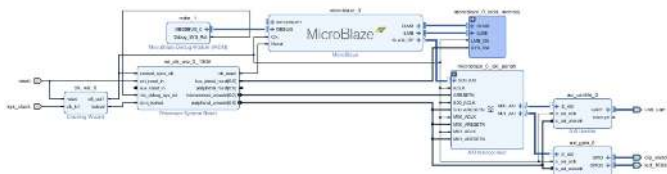


Fig. 7: Step 3 IP Block Design

on the processor is shown in Figure 8.

D. Create and add a custom IP block to the AXI system

In Step 4, a custom 16×16 signed multiplier IP block was created using Vivado's "Create and Package New IP" wizard, which generated an AXI4-Lite peripheral scaffold with four 32-bit registers. The provided multiplier VHDL code was added as a design source, and the peripheral's VHDL was manually edited to instantiate the multiplier, connecting the upper and lower 16 bits of register 0 as inputs and routing the 32-bit product to the register 1 read path. After re-packaging the IP, it appeared in the Vivado IP catalog and was added to the existing MicroBlaze block design from Step 3, where

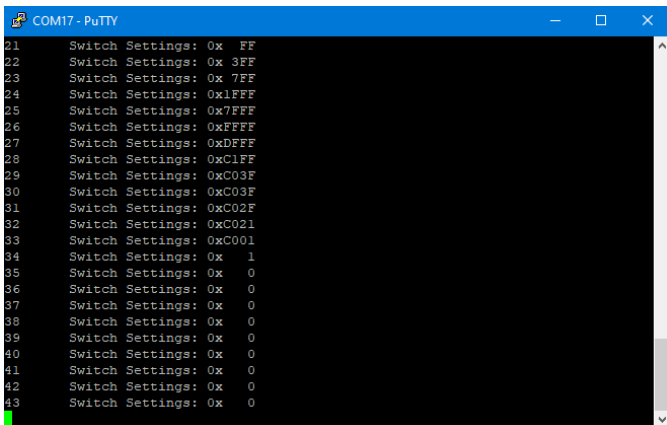


Fig. 8: Step 3 Serial Console

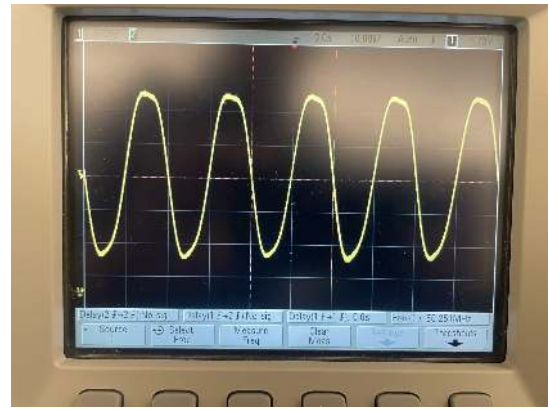


Fig. 9: Speed of Counter LSB - 50MHz

Connection Automation handled the AXI bus wiring. A new bitstream was generated and programmed onto the FPGA, and a short C program in the SDK was written to test the multiplier by writing two packed 16-bit operands as a single 32-bit write and reading back the resulting product. Several test cases were run to verify correct signed multiplication behavior.

E. Binary Counter and ILA demo using IP blocks

In Step 5, a new Vivado project was created to demonstrate the use of the Integrated Logic Analyzer (ILA) IP block, a powerful on-chip debugging tool that allows internal FPGA signals to be captured and examined in real time. A block design was built using a 32-bit binary counter, the System ILA core, two Slice IP blocks, and a Clocking Wizard, with the counter's enable input tied to slide switch SW0 on the NEXYS A7 board. The Slice blocks were configured to extract specific bit ranges from the counter output, routing bits 26-23 to the onboard LEDs and bits 7-0 to the JB PMOD connector. After generating and programming the bitstream, the ILA waveform viewer in Vivado's Hardware Manager was used to capture and display all 32 bits of the running counter, with a sample depth of 1024. Toggling SW0 allowed the counter to be started and stalled, and the LSB frequency of the counter was measured on the PMOD output. The output of the LSB of the counter is shown in Figure 9. The frequency corresponds to $\frac{100MHz}{2^1} = 50MHz$.

The toggling frequency of bit 26 can be calculated in a similar way:

$$f = \frac{100MHz}{2^{26}} = 1.49Hz$$

The result from displaying the ILA signals using Vivado are shown in Figure 10

III. QUESTIONS

A. What is a microcontroller? How is it different from a general purpose microprocessor?

To explain what a microcontroller is, the microprocessor must be explained first. This is because microcontrollers contain a microprocessor. A microprocessor is a digital circuit that executes instructions formatted according to the ISA (instruction set architecture) the processor was designed for.

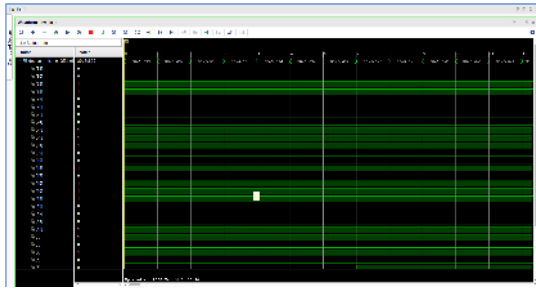


Fig. 10: 32 bit Counter ILA Results

Typically a microprocessor is contained within a single silicon chip. A microcontroller is a larger single chip system that, in addition to the microprocessor, contains peripheral circuits that can be used to control circuits external to the chip. For example, the STM32 line of microcontroller is used in many ECE courses at UVic. This microcontroller contains many peripherals such as timers, UART, SPI, and ADC. The STM32 is built off the ARM Cortex M line of microprocessors. In this case, the microcontroller and microprocessor are designed by completely separate companies and integration is possible because both companies adhere to standardized protocols (such as AMBA).

B. What is a UART

UART is a serial communication protocol that stands for Universal Asynchronous Receiver-Transmitter. It is used for communication between devices, such as sensors and microcontrollers. The UART IC contains a transmitter and receiver register as well as two shift registers for each direction of data flow (TX and RX), and finally a clock generator is provided that is then divided down to a chosen baud rate for the system.

The UART data frame consists of a start bit, followed by the data bits, parity bit (optional), followed by the stop bit.

C. What is an ILA?

The acronym ILA stands for Integrated Logic Analyzer. An ILA allows the designer to directly probe individual wires or registers of interest within their FPGA. Vivado provides an IP block that can be used to synthesize an ILA onto the FPGA. The ILA functions like a complete oscilloscope and logic analyzer. It can be used to view waveforms from any gate or wire of interest. It also allows configuration of the trigger condition just like a real oscilloscope. It contains capture memory which is typically implemented using the Block RAM on the FPGA. Every clock cycle, samples from all the probes are stored in the block RAM and can be viewed from within the Vivado ILA viewer.

IV. CONCLUSION

This lab demonstrated the design and implementation of soft-core processor systems using the Vivado IP library on an FPGA platform. Both a standalone MicroBlaze microcontroller and a customizable MicroBlaze processor system were successfully synthesized and programmed, highlighting

the flexibility of soft-core architectures compared to fixed hardware solutions. The integration of peripheral IP blocks such as UART and GPIO illustrated how complex embedded systems can be rapidly developed using standardized interfaces like AXI.

Additionally, the lab provided experience in extending processor functionality through the creation and integration of a custom AXI IP block, reinforcing the concept of hardware/software co-design. The use of the Integrated Logic Analyzer (ILA) enabled real-time observation of internal FPGA signals, demonstrating an effective method for debugging and validating digital designs.

Overall, this lab emphasized the advantages of FPGA-based system design, including modularity, scalability, and the ability to tailor hardware to specific application requirements. The combination of software development and hardware configuration provides a powerful approach for implementing embedded systems.

V. APPENDIX A: PROGRAMMING ASSIGNMENT CODE LISTINGS

Modify the constraints file so that the microcontroller outputs are directed to the JA PMOD on the NEXYS board

The constraints file modifications are shown in Figure 11.

Write a C program to output an 8 bit binary counter on the GPIO outputs. Measure the speed of the counter.

The 8-bit binary counter was implemented in C by using a for loop that iterated an integer from 0 to 255, with each value written to the GPIO port. The constraints file was modified to route the output signals to the JA port, as shown in Figure 11, allowing measurement of the least significant bit (LSB) using an oscilloscope (Figure 12). The measured frequency of the LSB was approximately 820 kHz, which was significantly lower than the 50 MHz observed for the 8-bit hardware counter. This difference is due to the MicroBlaze executing instructions sequentially and incurring overhead from looping and GPIO writes over the bus, whereas the hardware counter increments every clock cycle using dedicated parallel logic.

```

53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

Fig. 11: Constraints File Modification

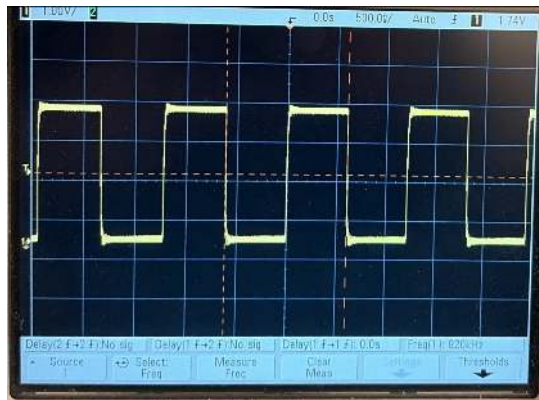


Fig. 12: 8-bit counter in C: LSB Frequency

Listing 1: 8-bit Counter C program

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

/*
 * The following constants map to the XPAR parameters created in the
 * xparameters.h file. They are defined here such that a user can easily
 * change all the needed parameters in one place.
 */
#define GPIO_OUTPUT_DEVICE_ID XPAR_GPIO_0_DEVICE_ID
#define GPIO_INPUT_DEVICE_ID XPAR_GPIO_1_DEVICE_ID
#define GPIO_BITWIDTH 16 /* This is the width of the GPIO */

/* The following constant is used to determine which channel of the GPIO is
 * used if there are 2 channels supported in the GPIO. Must be either 1 or 2.
 */
#define INPUT_SWITCH_CHANNEL 1 //in block design defined as GPIO
#define OUTPUT_LED_CHANNEL 2 //in block design defined as GPIO_2

/***** Variable Definitions *****/

/*
 * The following are declared globally so they are zeroed and so they are
 * easily accessible from a debugger
 */
XGpio GpioOutput; /* The driver instance for GPIO Device configured as O/P */
XGpio GpioInput; /* The driver instance for GPIO Device configured as I/P */

/*****

int main()
{
    u8 i;

    u32 DataRead;
    int Status;

    init_platform();
    xil_printf("Initialize\n\r");

    /*
     * Initialize the GPIO driver so that it's ready to use,
     * specify the device ID that is generated in xparameters.h
     * also set its data direction
     */
    Status = XGpio_Initialize(&GpioOutput, GPIO_OUTPUT_DEVICE_ID); //output
    if (Status != XST_SUCCESS) {
        return XST_FAILURE;
    }
    /* Set the direction for all signals to be outputs */
    XGpio_SetDataDirection(&GpioOutput, OUTPUT_LED_CHANNEL, 0x0);

    Status = XGpio_Initialize(&GpioInput, GPIO_INPUT_DEVICE_ID); //input
    if (Status != XST_SUCCESS) {
        return XST_FAILURE;
    }
    /* Set the direction for all signals to be inputs */
    XGpio_SetDataDirection(&GpioInput, INPUT_SWITCH_CHANNEL, 0xFFFFFFFF);

    for(i = 0; i < 256; i++)
    {
        XGpio_DiscreteWrite(&GpioOutput, OUTPUT_LED_CHANNEL, i);
    }

    cleanup_platform();
    return 0;
}
```