[Microprocessor Applications]
Lab 4: I/O Peripherals

Chester Sungchung Park
SoC Design Lab, Konkuk University
Webpage: http://soclab.konkuk.ac.kr

Outline

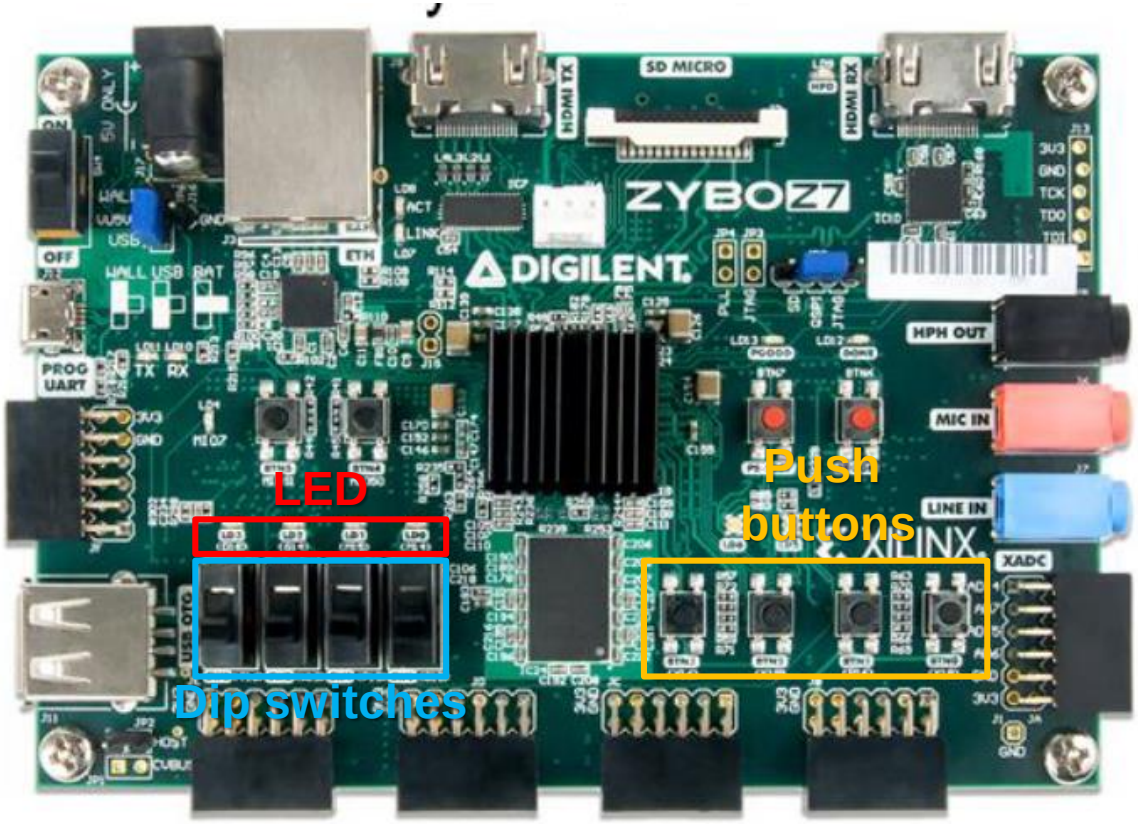
- Objectives
- Description
- Block diagram
- Address map
- Section map
- Source codes
- Evaluation

Objectives

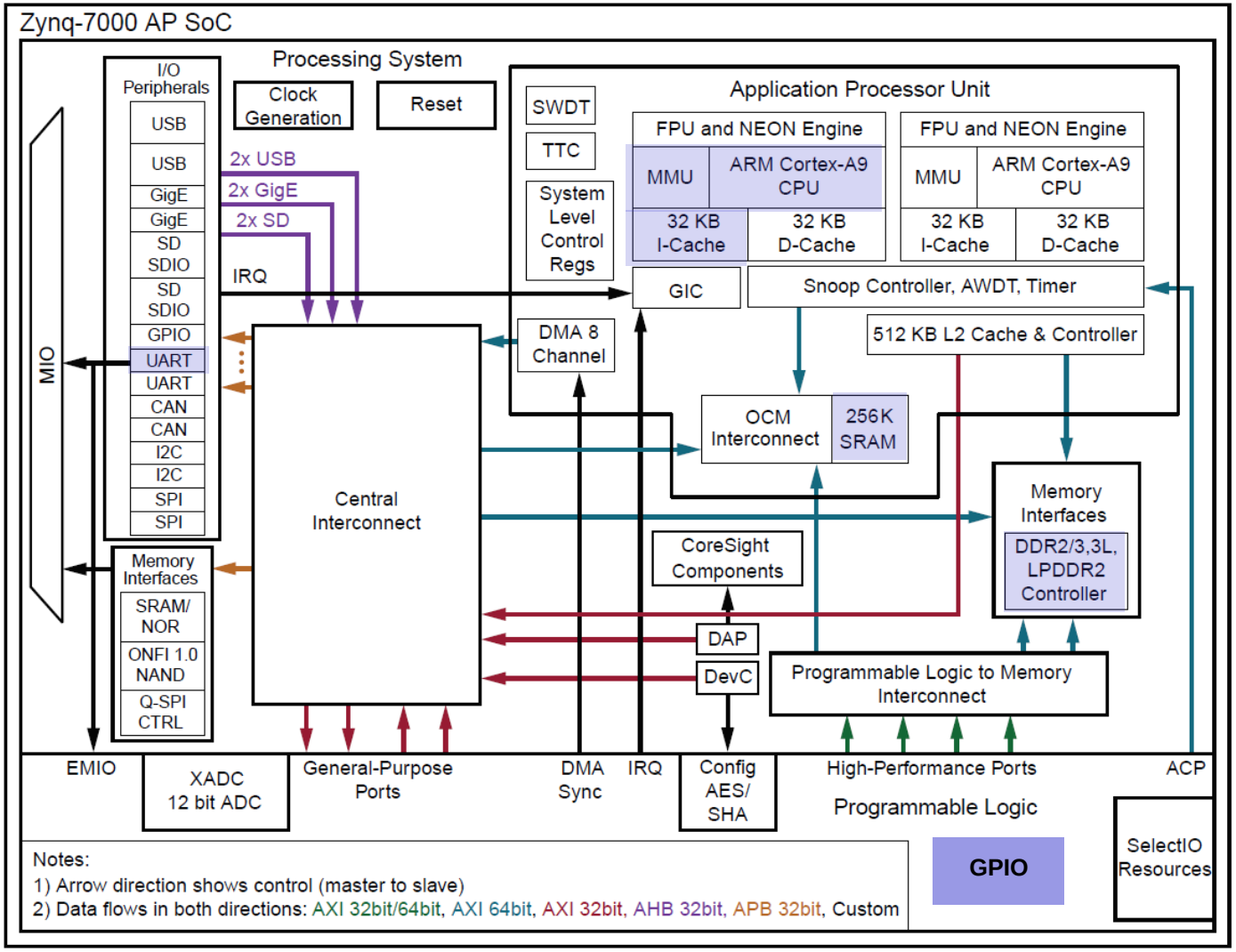
- Running a C application that writes and reads different on-board I/O components through GPIOs
- Programming a C application that implements a stopwatch

Description

- On-board I/O components

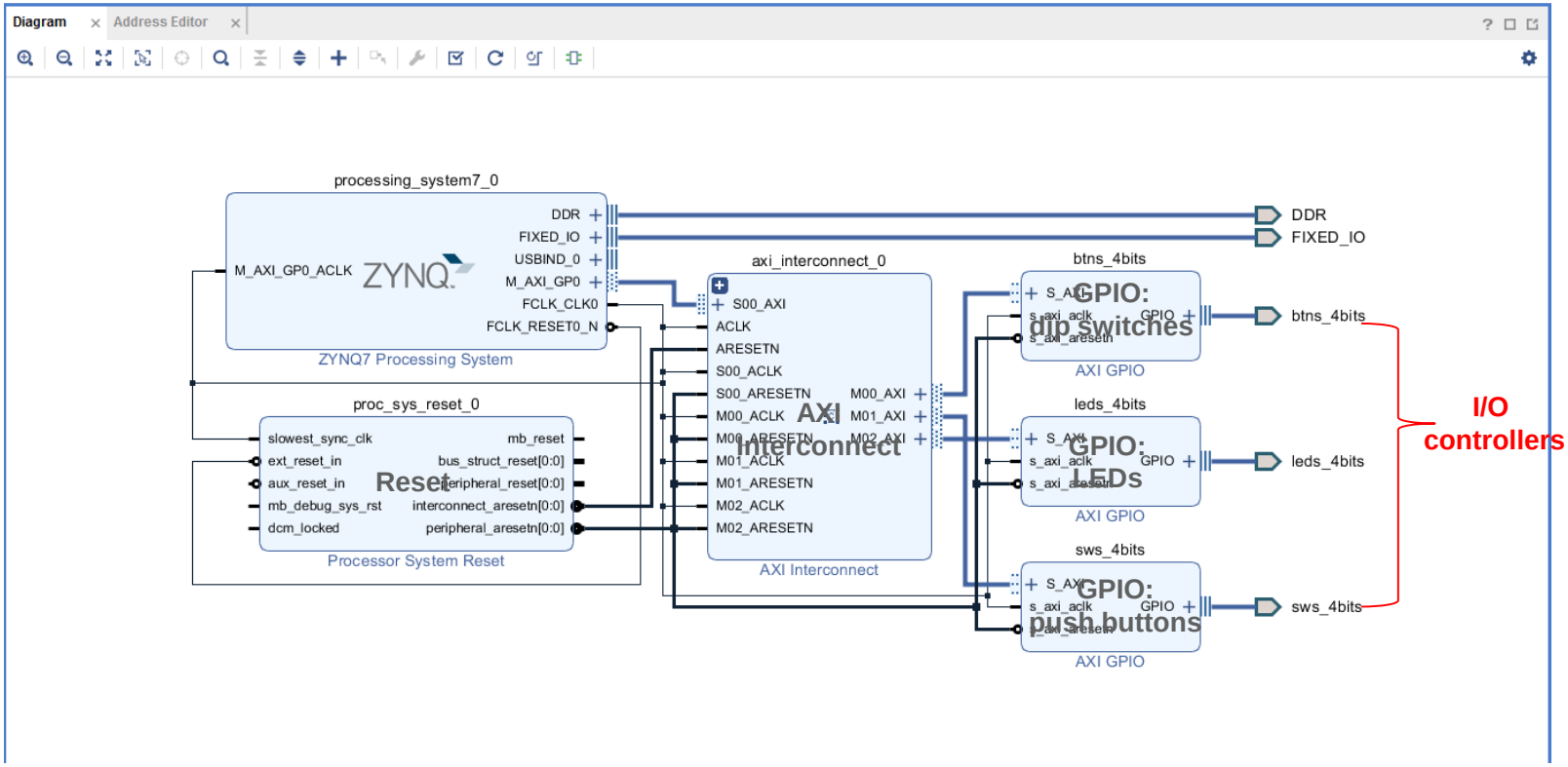


Block Diagram



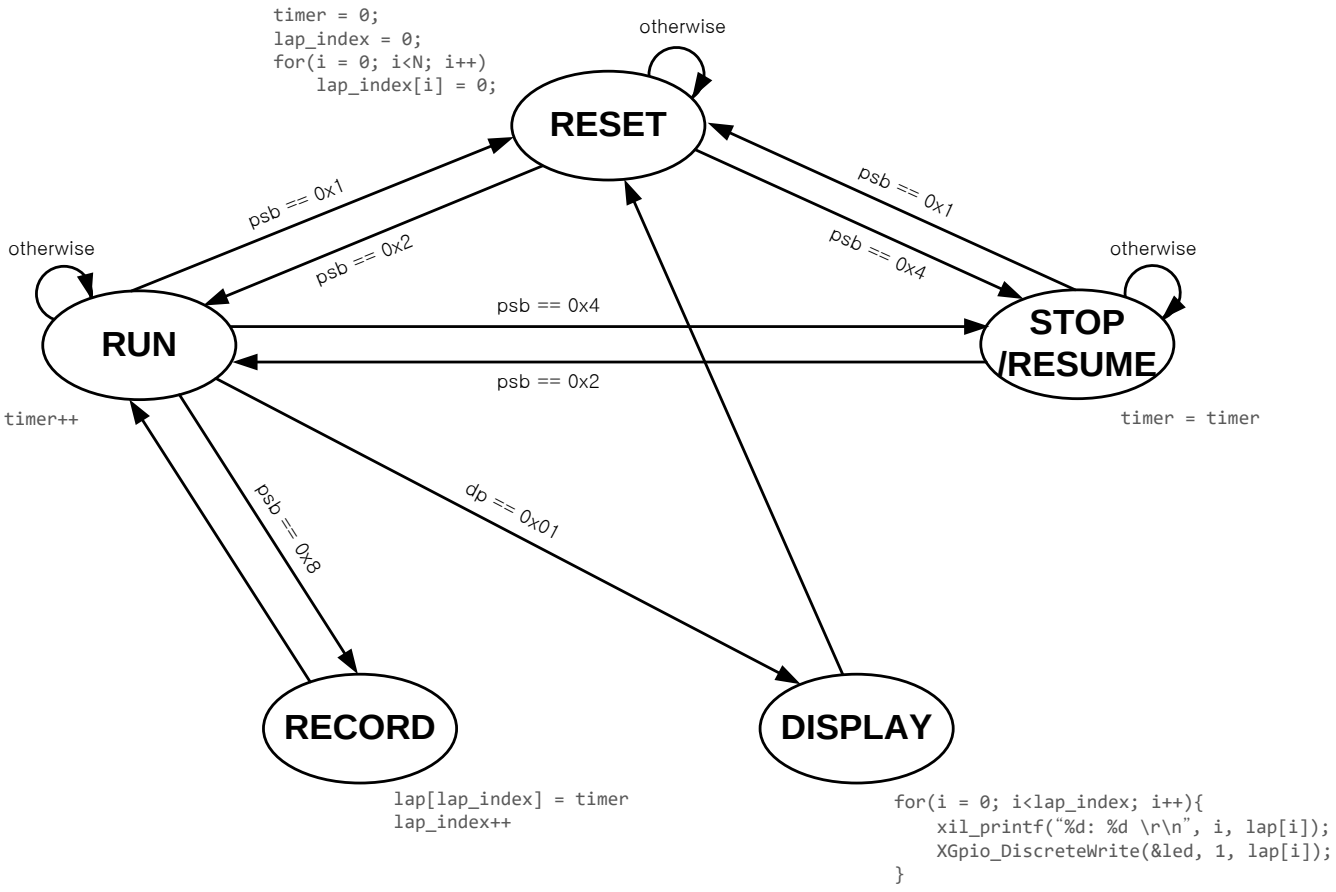
Block Diagram

- Vivado block diagram



Stopwatch

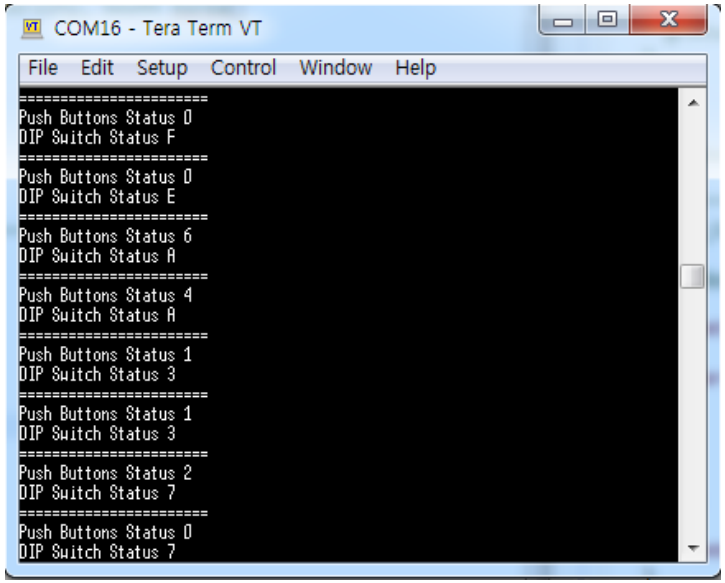
- State diagram



Example Codes

- gpiotest.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <xtime_l.h>
#include "xparameters.h"
#include "xgpio.h"
//#include "xutil.h"

//====================================================

void delay()
{
    int  u = 0, c = 0, p = 0;
    for (u=0; u<9999999; u++);
        for (c=0; c<9999999; c++);
            for (p=0; p<9999999; p++);
}

int main (void)
{
    XGpio dip, push, led;
    int i = 0;
    int psb_check = 0, dip_check = 0, led_cnt = 0;

    xil_printf("-- Start of the Program --\r\n");

    XGpio_Initialize(&dip, XPAR_SWS_4BITS_DEVICE_ID);
    XGpio_SetDataDirection(&dip, 1, 0xffffffff);

    XGpio_Initialize(&push, XPAR_BTNS_4BITS_DEVICE_ID);
    XGpio_SetDataDirection(&push, 1, 0xffffffff);

    XGpio_Initialize(&led, XPAR_LEDS_4BITS_DEVICE_ID);
    XGpio_SetDataDirection(&led, 1, 0x00000000);

    while(1)
    {
      psb_check = XGpio_DiscreteRead(&push, 1);
      xil_printf("Push Buttons Status %x\r\n", psb_check);

      dip_check = XGpio_DiscreteRead(&dip, 1);
      xil_printf("DIP Switch Status %x\r\n", dip_check);

      led_cnt = dip_check;
      for (i = dip_check; i>=0; --i){
          XGpio_DiscreteWrite(&led, 1, led_cnt);
          --led_cnt;
          delay();
      }

      xil_printf("====================== \r\n");
    }
}
```



Source Codes

- xgpio_sinit.c

xgpio_tapp_example.c | gpiotest.c | xgpio_sinit.c | *xgpio.c | xgpio.h

```
            CfgPtr = &XGpio_ConfigTable[Index];
            break;
        }
    }

    return CfgPtr;
}


/*****************************************************************************/
/**
* Initialize the XGpio instance provided by the caller based on the
* given DeviceID.
*
* Nothing is done except to initialize the InstancePtr.
*
* @param    InstancePtr is a pointer to an XGpio instance. The memory the
*       pointer references must be pre-allocated by the caller. Further
*       calls to manipulate the instance/driver through the XGpio API
*       must be made with this pointer.
* @param    DeviceId is the unique id of the device controlled by this XGpio
*       instance. Passing in a device id associates the generic XGpio
*       instance to a specific device, as chosen by the caller or
*       application developer.
*
* @return
*       - XST_SUCCESS if the initialization was successfull.
*       - XST_DEVICE_NOT_FOUND  if the device configuration data was not
*       found for a device with the supplied device ID.
*
* @note     None.
*
******************************************************************************/
int XGpio_Initialize(XGpio * InstancePtr, u16 DeviceId)
{
    XGpio_Config *ConfigPtr;

    /*
     * Assert arguments
     */
    Xil_AssertNonvoid(InstancePtr != NULL);

    /*
     * Lookup configuration data in the device configuration table.
     * Use this configuration info down below when initializing this
     * driver.
     */
    ConfigPtr = XGpio_LookupConfig(DeviceId);
    if (ConfigPtr == (XGpio_Config *) NULL) {
        InstancePtr->IsReady = 0;
        return (XST_DEVICE_NOT_FOUND);
    }

    return XGpio_CfgInitialize(InstancePtr, ConfigPtr,
                ConfigPtr->BaseAddress);
}
```

Source Codes

- xgpio.h

xgpio_tapp_example.c | gpiotest.c | xgpio_sinit.c | xgpio.c | xgpio.h

```
/************************** Type Definitions *****************************/

/**
 * This typedef contains configuration information for the device.
 */
typedef struct {
    u16 DeviceId;       /* Unique ID  of device */
    u32 BaseAddress;    /* Device base address */
    int InterruptPresent;   /* Are interrupts supported in h/w */
    int IsDual;     /* Are 2 channels supported in h/w */
} XGpio_Config;

/**
 * The XGpio driver instance data. The user is required to allocate a
 * variable of this type for every GPIO device in the system. A pointer
 * to a variable of this type is then passed to the driver API functions.
 */
typedef struct {
    u32 BaseAddress;    /* Device base address */
    u32 IsReady;        /* Device is initialized and ready */
    int InterruptPresent;   /* Are interrupts supported in h/w */
    int IsDual;     /* Are 2 channels supported in h/w */
} XGpio;

/****************** Macros (Inline Functions) Definitions *********************/


/************************** Function Prototypes ******************************/

/*
 * Initialization functions in xgpio_sinit.c
 */
int XGpio_Initialize(XGpio *InstancePtr, u16 DeviceId);
XGpio_Config *XGpio_LookupConfig(u16 DeviceId);

/*
 * API Basic functions implemented in xgpio.c
 */
int XGpio_CfgInitialize(XGpio *InstancePtr, XGpio_Config * Config,
                u32 EffectiveAddr);
void XGpio_SetDataDirection(XGpio *InstancePtr, unsigned Channel,
                u32 DirectionMask);
u32 XGpio_GetDataDirection(XGpio *InstancePtr, unsigned Channel);
u32 XGpio_DiscreteRead(XGpio *InstancePtr, unsigned Channel);
void XGpio_DiscreteWrite(XGpio *InstancePtr, unsigned Channel, u32 Mask);


/*
 * API Functions implemented in xgpio_extra.c
 */
void XGpio_DiscreteSet(XGpio *InstancePtr, unsigned Channel, u32 Mask);
void XGpio_DiscreteClear(XGpio *InstancePtr, unsigned Channel, u32 Mask);
```

Source Codes

- xgpio.c

xgpio_tapp_example.c | gpiotest.c | xgpio_sinit.c | *xgpio.c | xgpio.h

```
int XGpio_CfgInitialize(XGpio * InstancePtr, XGpio_Config * Config,
                        u32 EffectiveAddr)
{
    /*
     * Assert arguments
     */
    Xil_AssertNonvoid(InstancePtr != NULL);

    /*
     * Set some default values.
     */
#if (XPAR_XGPIO_USE_DCR_BRIDGE != 0)
    InstancePtr->BaseAddress = ((EffectiveAddr >> 2)) & 0xFFF;
#else
    InstancePtr->BaseAddress = EffectiveAddr;
#endif

    InstancePtr->InterruptPresent = Config->InterruptPresent;
    InstancePtr->IsDual = Config->IsDual;

    /*
     * Indicate the instance is now ready to use, initialized without error
     */
    InstancePtr->IsReady = XIL_COMPONENT_IS_READY;
    return (XST_SUCCESS);
}

/*****************************************************************************/
/**
* Set the input/output direction of all discrete signals for the specified
* GPIO channel.
*
* @param    InstancePtr is a pointer to an XGpio instance to be worked on.
* @param    Channel contains the channel of the GPIO (1 or 2) to operate on.
* @param    DirectionMask is a bitmask specifying which discretes are input
*       and which are output. Bits set to 0 are output and bits set to 1
*       are input.
*
* @return   None.
*
* @note     The hardware must be built for dual channels if this function
*       is used with any channel other than 1.  If it is not, this
*       function will assert.
*
******************************************************************************/
void XGpio_SetDataDirection(XGpio * InstancePtr, unsigned Channel,
                            u32 DirectionMask)
{
    Xil_AssertVoid(InstancePtr != NULL);
    Xil_AssertVoid(InstancePtr->IsReady == XIL_COMPONENT_IS_READY);
    Xil_AssertVoid((Channel == 1) ||
                   ((Channel == 2) && (InstancePtr->IsDual == TRUE)));

    XGpio_WriteReg(InstancePtr->BaseAddress,
                   ((Channel - 1) * XGPIO_CHAN_OFFSET) + XGPIO_TRI_OFFSET,
                   DirectionMask);
}
```

Source Codes

- xgpio_l.h

xgpio_tapp_example.c | gpiotest.c | xgpio_sinit.c | *xgpio.c | xgpio.h | xgpio_l.h

```
#define XGpio_In32  Xil_In32
#define XGpio_Out32 Xil_Out32

#endif


/****************************************************************************/
/**
*
* Write a value to a GPIO register. A 32 bit write is performed. If the
* GPIO core is implemented in a smaller width, only the least significant data
* is written.
*
* @param    BaseAddress is the base address of the GPIO device.
* @param    RegOffset is the register offset from the base to write to.
* @param    Data is the data written to the register.
*
* @return   None.
*
* @note     C-style signature:
*       void XGpio_WriteReg(u32 BaseAddress, u32 RegOffset, u32 Data)
*
*****************************************************************************/
#define XGpio_WriteReg(BaseAddress, RegOffset, Data) \
    XGpio_Out32((BaseAddress) + (RegOffset), (u32)(Data))

/****************************************************************************/
/**
*
* Read a value from a GPIO register. A 32 bit read is performed. If the
* GPIO core is implemented in a smaller width, only the least
* significant data is read from the register. The most significant data
* will be read as 0.
*
* @param    BaseAddress is the base address of the GPIO device.
* @param    RegOffset is the register offset from the base to read from.
*
* @return   Data read from the register.
*
* @note     C-style signature:
*       u32 XGpio_ReadReg(u32 BaseAddress, u32 RegOffset)
*
*****************************************************************************/
#define XGpio_ReadReg(BaseAddress, RegOffset) \
    XGpio_In32((BaseAddress) + (RegOffset))

/************************** Function Prototypes ******************************/

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

References

- Zynq-7000 All Programmable, technical reference manual, Xilinx UG585