2.6 KEY INTERFACING WITH LPC1768 – IMPLEMENTING POLLING TECHNIQUE

Learning Outcomes:

After studying this interfacing project, students will be able to:

Learn to configure port pin as input.

Understand the concept of polling and its drawback.

2.6.1 INTERFACING DIAGRAM

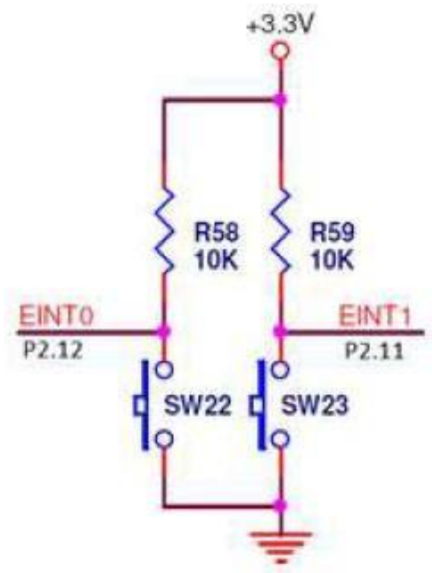

Fig.: Keys connected to P2.12($\overline{EINT0}$) and P2.11 ($\overline{EINT1}$)

- **Connection of Keys:**
  - Each key is connected to a GPIO pin (P2.11 and P2.12) of the LPC1768. ○ The other terminal of the key is usually connected to ground (GND).
  - The microcontroller pins are configured with an internal pull-up resistor, ensuring that when the key is not pressed, the pin remains HIGH (logic 1).

- **Detection of Key Press:**
  - Pressing the key creates a direct connection to GND.
  - This pulls the voltage on the corresponding pin LOW (logic 0). The microcontroller detects this change.

- **Choosing the PULL-UP Resistor Value:**

  For a pull-up configuration:

  - The GPIO pin is normally HIGH due to the pull-up resistor.
  - Pressing the button connects the pin to GND (LOW state).

  The resistor value (R) is determined based on:

a) **Input Leakage Current of the GPIO Pin:**
The LPC1768 has an input leakage current in the range of microamperes (µA), so a pull-up resistor must ensure the voltage level remains within logic HIGH.

b) **Voltage Divider Effect**
When the button is pressed, the voltage across the resistor should drop to a level recognized as logic LOW by the microcontroller.

c) **Standard Resistor Range Considerations**
A high resistor value (~100kΩ) may result in a slower transition due to stray capacitances. A low resistor value (~1kΩ) will cause unnecessary power dissipation. A typical pull-up resistor value is 4.7kΩ to 10kΩ.

Calculation Example for a 10kΩ Resistor:

- Assume Vcc = 3.3V, and GPIO pin draws negligible current.
- When the button is not pressed, the voltage at the pin remains Vcc.
- When the button is pressed, the pin is connected to GND, and current flows through the 10kΩ resistor:

$$I = \frac{V}{R} = \frac{3.3}{10K} = 0.33\, mA$$

This small current ensures minimal power consumption while still allowing a stable logic transition.

- **Key Debouncing:**
When a mechanical switch (like a push button) is pressed or released, the metal contacts inside the switch do not make or break the connection instantly. Instead, they physically bounce for a few milliseconds before settling into a stable state. This causes multiple unwanted transitions, which can be misinterpreted as multiple presses by the microcontroller. This phenomenon is called **switch bouncing or key bouncing**. To avoid this, a technique called debouncing is used to ensure that only a single, clean transition is detected per key press.

Delay-Based Debouncing:
When a button press is detected, introduce a small delay (e.g., 10–50 ms) before reading the button state again. If the button state remains the same after the delay, it is considered a valid press.

2.6.2 EXAMPLE CODE – POLLING & BLOCKING DELAY

1)Write a C program for the LPC1768 microcontroller that reads the status of a switch connected to P2.11. When the switch is pressed, turn on an LED connected to P1.19. Additionally, implement a blinking LED on P1.26 that toggles its state at regular intervals. Include a debounce mechanism for the switch input.

```
#include <lpc17xx.h>
#include <stdbool.h>

#define KEY_PIN     11 // Switch is connected to P2.11
#define LED_PIN     19  // LED1 is connected to P1.19
#define BLINK_LED   26  // LED8 is connected to P1.26
 void delay(unsigned int
count);

int main (void)
{
     bool swstatus;

     //Configure the pin connected to                 key as input
     LPC_GPIO2->FIODIR &= ~(1<<KEY_PIN);

     //Configure LED pins as output
     LPC_GPIO1->FIODIR|= (1<<LED_PIN)|(1<<BLINK_LED);

     // Clears bits 22-23 (P2.11) to enable pull-up
  LPC_PINCON->PINMODE4
&= 0xFF3FFFFF;

     while(1)
     {
          /*Read switch status. Provide debounce and confirm
          the switch status */
          swstatus = (LPC_GPIO2->FIOPIN & (1 << KEY_PIN));
          delay(5000);
          swstatus = (LPC_GPIO2->FIOPIN & (1 << KEY_PIN));

          if(swstatus == 0)
     LPC_GPIO1->FIOSET = (1<<LED_PIN);
          else
               LPC_GPIO1->FIOCLR = (1<<LED_PIN);
```

Learn to clear a bit

Refer last sheet for PINMODE register.

```
          //Blink LED connected to P1.26 continuously
          LPC_GPIO1->FIOSET = (1<<BLINK_LED);
          delay(10000);
          LPC_GPIO1->FIOCLR = (1<<BLINK_LED);
delay(10000);
     }
} void delay(unsigned int
count)
{    unsigned int
i,j;

     for(i=0; i<count; i++)
for(j=0; j<1275; j++);
}
```

Observation: The LED is not ON every time the Key is pressed.

Issue: The key press is not detected while the LED is blinking because the program is stuck inside delay() function in the LED blink logic. This is the issue with polling.

Solution: Use non-blocking delays / Timers in interrupt mode.

2)Repeat the above program using non-blocking delays. Observe the difference.

```
#include <lpc17xx.h>
#include <stdbool.h>

#define KEY_PIN      11    // Switch is connected to P2.11
#define LED_PIN      19    // LED1 is connected to P1.19
#define BLINK_LED    26    // LED8 is connected to P1.26

volatile unsigned int blinkcounter = 0; volatile
unsigned int debouncetimer = 0;
 int main
(void)
{
     bool swstatus;
```

```
      //Configure the pin connected to key as input
      LPC_GPIO2->FIODIR &= ~(1<<KEY_PIN);

      //Configure LED pins as output
      LPC_GPIO1->FIODIR|= (1<<LED_PIN)|(1<<BLINK_LED);

      // Clears bits 22-23 (P2.11) to enable pull-up   LPC_PINCON-
>PINMODE4 &= 0xFF3FFFFF;

      while(1)
      {
            /*Read switch status. Provide debounce and
   confirm  the switch status */             swstatus =
(LPC_GPIO2->FIOPIN & (1 << KEY_PIN));

            // Increment the debounce timer
debouncetimer++;

            // Wait for debounce delay to expire
if(debouncetimer > 100000)
            {
                  swstatus = (LPC_GPIO2->FIOPIN & (1 << KEY_PIN));

                  // Reset debounce timer after checking switch
      debouncetimer = 0;

                  if(swstatus == 0)
      LPC_GPIO1->FIOSET = (1<<LED_PIN);
                  else
                        LPC_GPIO1->FIOCLR = (1<<LED_PIN);
            }

            // Non-blocking blink logic using a counter
blinkcounter++;

          // If counter reaches the threshold, toggle the blink LED
if (blinkcounter >= 1000000)
            {
                  LPC_GPIO1->FIOPIN ^= (1 << BLINK_LED);
blinkcounter = 0;
            }
      }
}
```

Further exploration:

Write the circuit for pull down configuration.

Try checking the status of both the keys and design an application for the same.

DATASHEET REFERENCE

Pin mode select register values

The PINMODE registers control the input mode of all ports. This includes the use of the on-chip pull-up/pull-down resistor feature and a special open drain operating mode. The on-chip pull-up/pull-down resistor can be selected for every port pin regardless of the function on this pin with the exception of the I2C pins for the I2C0 interface and the USB pins (see Section 8.5.10). Three bits are used to control the mode of a port pin, two in a PINMODE register, and an additional one in a PINMODE_OD register. Bits are reserved for unused pins as in the PINSEL registers.

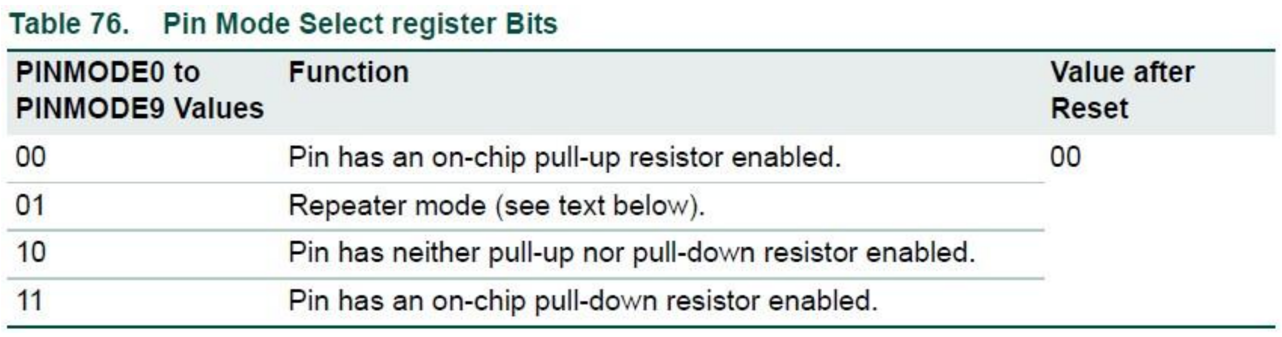

Table 76. Pin Mode Select register Bits

PINMODE0 to PINMODE9 Values	Function	Value after Reset
00	Pin has an on-chip pull-up resistor enabled.	00
01	Repeater mode (see text below).	
10	Pin has neither pull-up nor pull-down resistor enabled.	
11	Pin has an on-chip pull-down resistor enabled.	

Pin Mode select register 4 (PINMODE4 - 0x4002 C050)

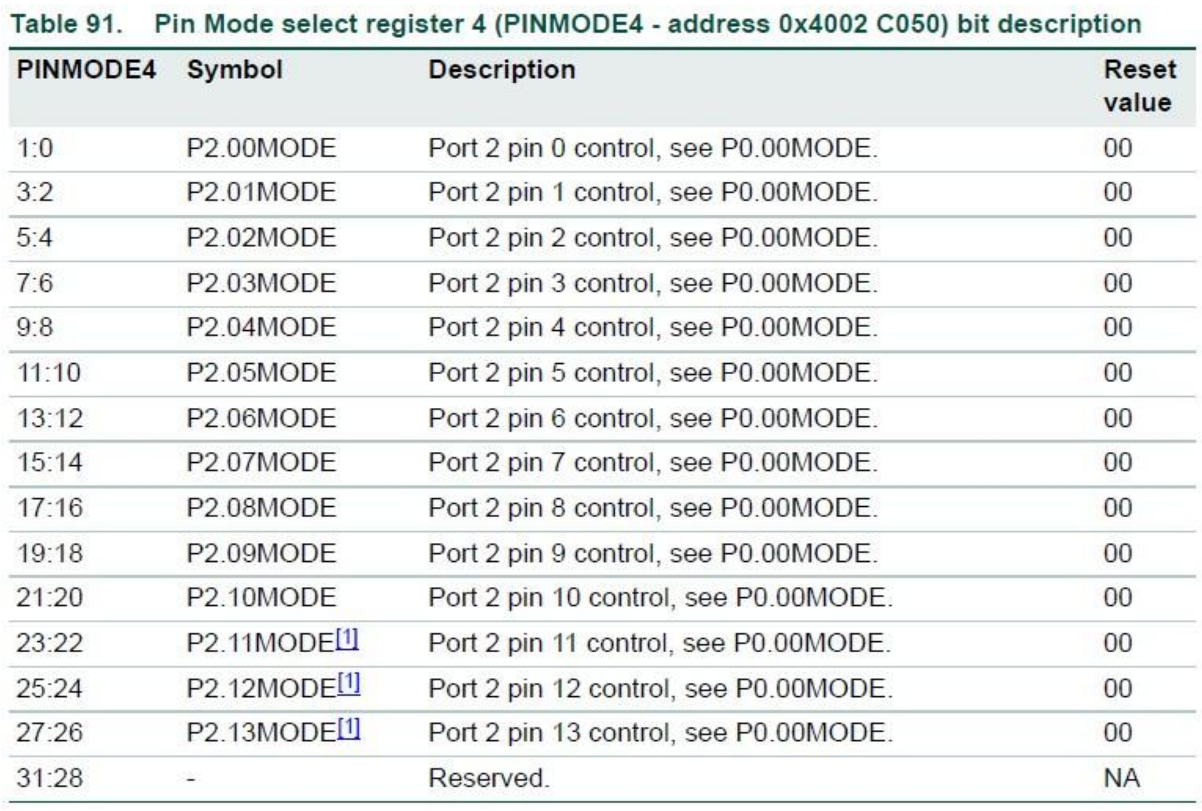
Table 91. Pin Mode select register 4 (PINMODE4 - address 0x4002 C050) bit description

PINMODE4	Symbol	Description	Reset value
1:0	P2.00MODE	Port 2 pin 0 control, see P0.00MODE.	00
3:2	P2.01MODE	Port 2 pin 1 control, see P0.00MODE.	00
5:4	P2.02MODE	Port 2 pin 2 control, see P0.00MODE.	00
7:6	P2.03MODE	Port 2 pin 3 control, see P0.00MODE.	00
9:8	P2.04MODE	Port 2 pin 4 control, see P0.00MODE.	00
11:10	P2.05MODE	Port 2 pin 5 control, see P0.00MODE.	00
13:12	P2.06MODE	Port 2 pin 6 control, see P0.00MODE.	00
15:14	P2.07MODE	Port 2 pin 7 control, see P0.00MODE.	00
17:16	P2.08MODE	Port 2 pin 8 control, see P0.00MODE.	00
19:18	P2.09MODE	Port 2 pin 9 control, see P0.00MODE.	00
21:20	P2.10MODE	Port 2 pin 10 control, see P0.00MODE.	00
23:22	P2.11MODE[1]	Port 2 pin 11 control, see P0.00MODE.	00
25:24	P2.12MODE[1]	Port 2 pin 12 control, see P0.00MODE.	00
27:26	P2.13MODE[1]	Port 2 pin 13 control, see P0.00MODE.	00
31:28	-	Reserved.	NA