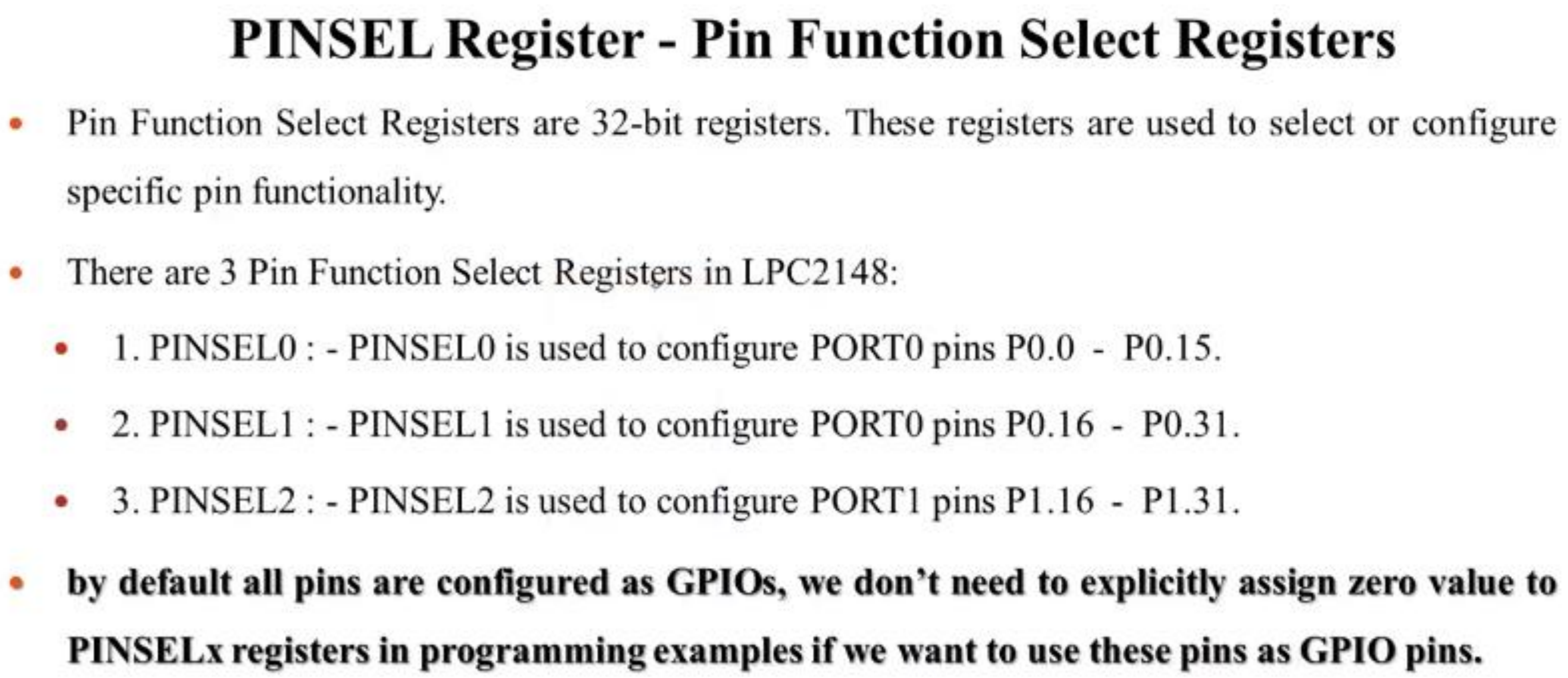
MODULE - 5

ARM Processors Interfacing & Applications



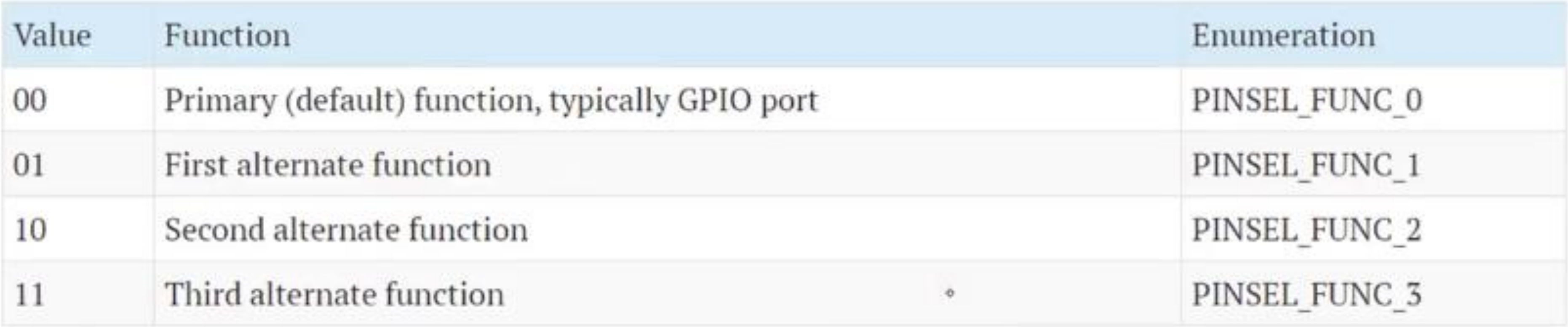
PINSEL Register - Pin Function Select Registers

- Pin Function Select Registers are 32-bit registers. These registers are used to select or configure specific pin functionality.
- There are 3 Pin Function Select Registers in LPC2148:
  - 1. PINSEL0 : - PINSEL0 is used to configure PORT0 pins P0.0 - P0.15.
  - 2. PINSEL1 : - PINSEL1 is used to configure PORT0 pins P0.16 - P0.31.
  - 3. PINSEL2 : - PINSEL2 is used to configure PORT1 pins P1.16 - P1.31.
- **by default all pins are configured as GPIOs, we don't need to explicitly assign zero value to PINSELx registers in programming examples if we want to use these pins as GPIO pins.**

PINSEL Register - Pin Function Select Registers

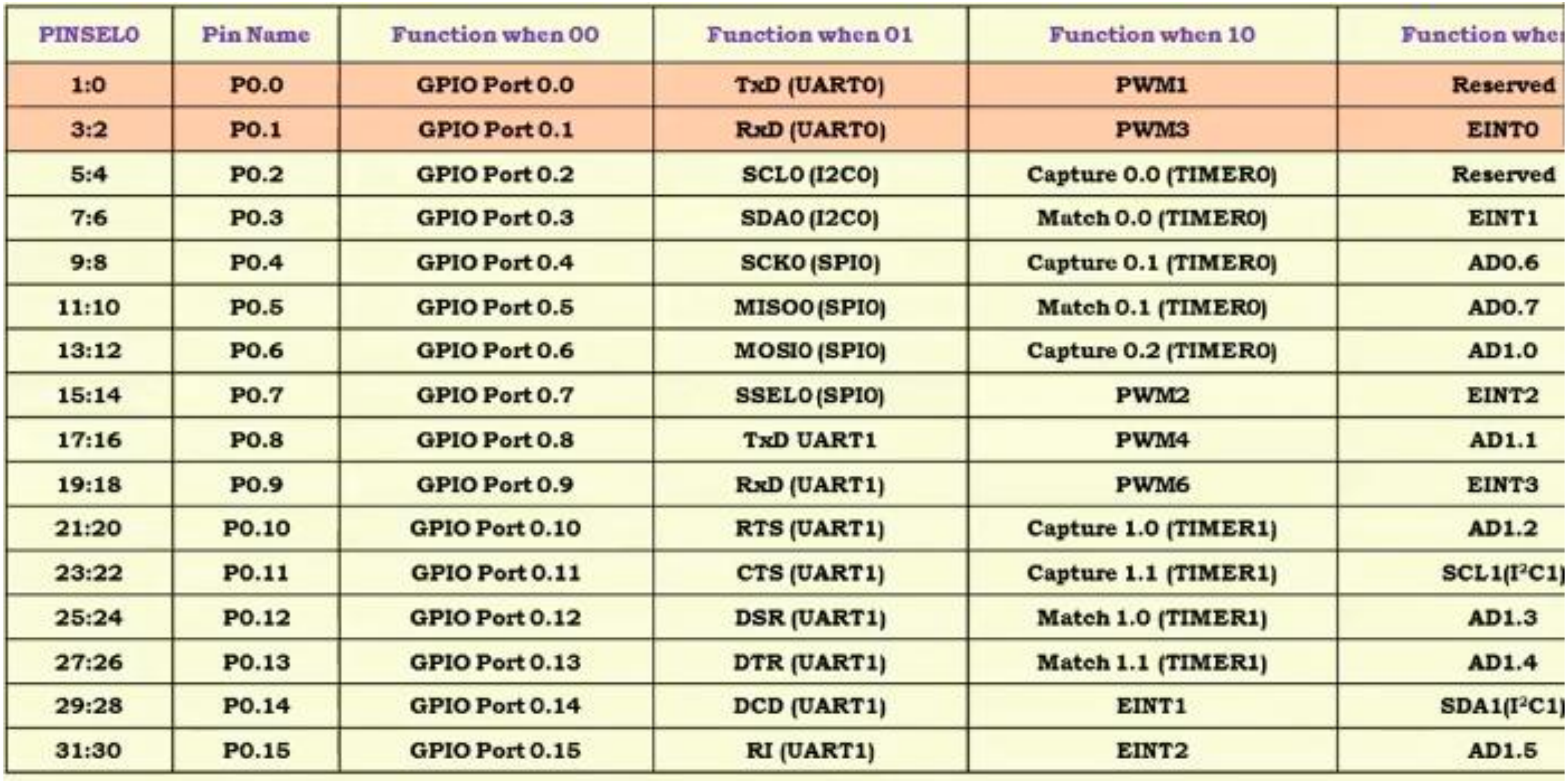
- Almost all the LPC2148 pins are multiplexed to support more than 1 function. Every GPIO pin has a minimum of one function and max of four functions.
- The required function can be selected by configuring the PINSEL register.
- As there can be up to 4 functions associated with a GPIO pin, two bits for each pin are available to select the function.
- This implies that we need **two PINSEL** registers to configure a ONE PORT pins, and three for two ports.
- By this, the first 16(P0.0-P0.16) pin functions of PORT0 can be selected by 32 bits of **PINSEL0** register.
- ⋄The remaining 16 bits(P0.16-P0.32) are configured using 32bits of **PINSEL1** register.
- The **PINSEL2** is used for PORT 1 to select PORT1 pins P1.16 to P1.31.

PINSEL Register - Pin Function Select Registers

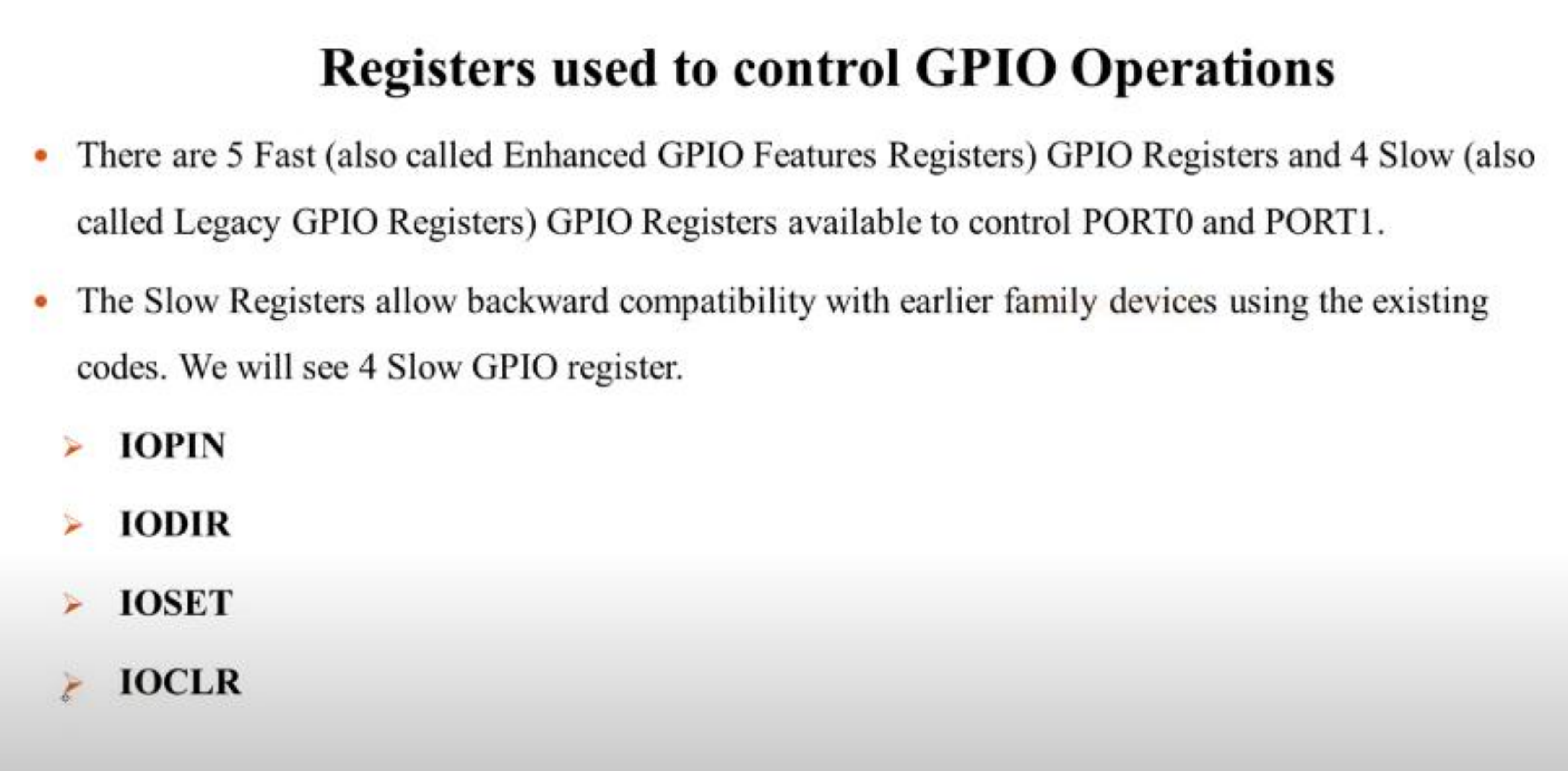


Value	Function	Enumeration
00	Primary (default) function, typically GPIO port	PINSEL_FUNC_0
01	First alternate function	PINSEL_FUNC_1
10	Second alternate function	PINSEL_FUNC_2
11	Third alternate function	PINSEL_FUNC_3

PINSEL0 and corresponding functions on Port0

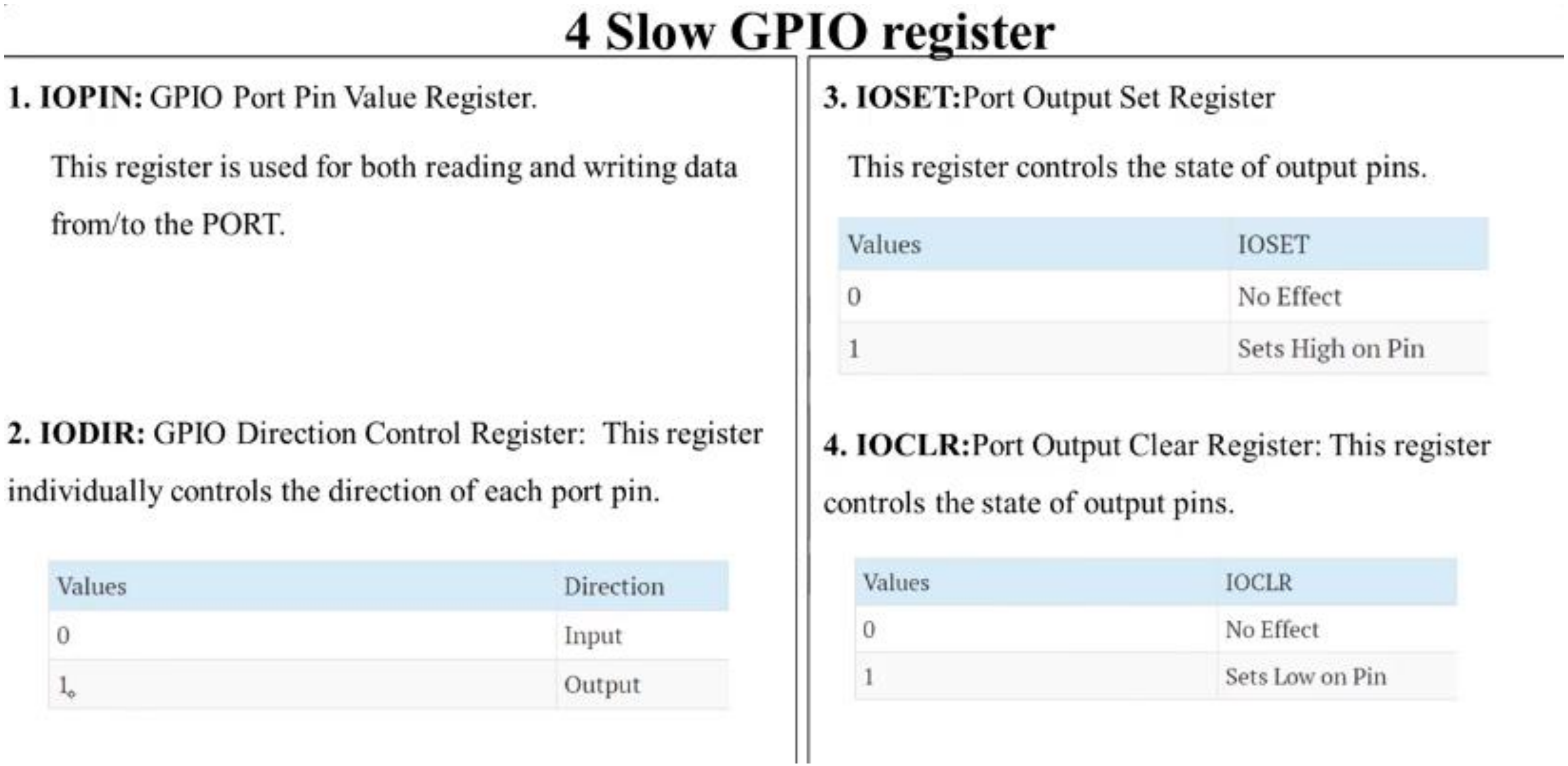


PINSEL0	Pin Name	Function when 00	Function when 01	Function when 10	Function whe
1:0	P0.0	GPIO Port 0.0	TxD (UART0)	PWM1	Reserved
3:2	P0.1	GPIO Port 0.1	RxD (UART0)	PWM3	EINT0
5:4	P0.2	GPIO Port 0.2	SCL0 (I2C0)	Capture 0.0 (TIMER0)	Reserved
7:6	P0.3	GPIO Port 0.3	SDA0 (I2C0)	Match 0.0 (TIMER0)	EINT1
9:8	P0.4	GPIO Port 0.4	SCK0 (SPI0)	Capture 0.1 (TIMER0)	AD0.6
11:10	P0.5	GPIO Port 0.5	MISO0 (SPI0)	Match 0.1 (TIMER0)	AD0.7
13:12	P0.6	GPIO Port 0.6	MOSI0 (SPI0)	Capture 0.2 (TIMER0)	AD1.0
15:14	P0.7	GPIO Port 0.7	SSEL0 (SPI0)	PWM2	EINT2
17:16	P0.8	GPIO Port 0.8	TxD UART1	PWM4	AD1.1
19:18	P0.9	GPIO Port 0.9	RxD (UART1)	PWM6	EINT3
21:20	P0.10	GPIO Port 0.10	RTS (UART1)	Capture 1.0 (TIMER1)	AD1.2
23:22	P0.11	GPIO Port 0.11	CTS (UART1)	Capture 1.1 (TIMER1)	SCL1(I^2C1)
25:24	P0.12	GPIO Port 0.12	DSR (UART1)	Match 1.0 (TIMER1)	AD1.3
27:26	P0.13	GPIO Port 0.13	DTR (UART1)	Match 1.1 (TIMER1)	AD1.4
29:28	P0.14	GPIO Port 0.14	DCD (UART1)	EINT1	SDA1(I^2C1)
31:30	P0.15	GPIO Port 0.15	RI (UART1)	EINT2	AD1.5



Registers used to control GPIO Operations

- There are 5 Fast (also called Enhanced GPIO Features Registers) GPIO Registers and 4 Slow (also called Legacy GPIO Registers) GPIO Registers available to control PORT0 and PORT1.
- The Slow Registers allow backward compatibility with earlier family devices using the existing codes. We will see 4 Slow GPIO register.
  - **IOPIN**
  - **IODIR**
  - **IOSET**
  - **IOCLR**



4 Slow GPIO register

1. IOPIN: GPIO Port Pin Value Register.

This register is used for both reading and writing data from/to the PORT.

2. IODIR: GPIO Direction Control Register: This register individually controls the direction of each port pin.

Values	Direction
0	Input
1	Output

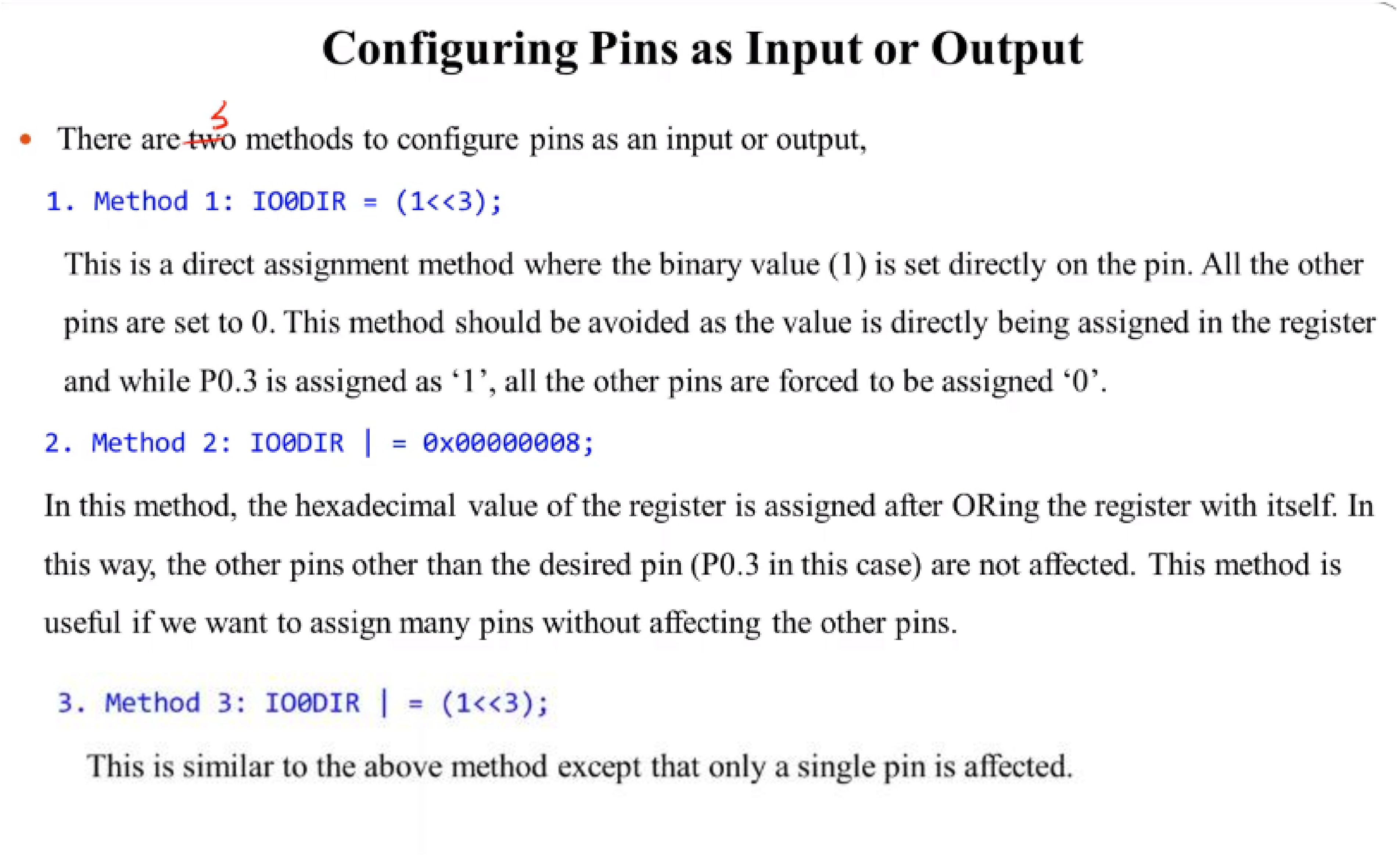
3. IOSET: Port Output Set Register

This register controls the state of output pins.

Values	IOSET
0	No Effect
1	Sets High on Pin

4. IOCLR: Port Output Clear Register: This register controls the state of output pins.

Values	IOCLR
0	No Effect
1	Sets Low on Pin



Configuring Pins as Input or Output

- There are ~~two~~ 3 methods to configure pins as an input or output,

```
1. Method 1: IO0DIR = (1<<3);
```

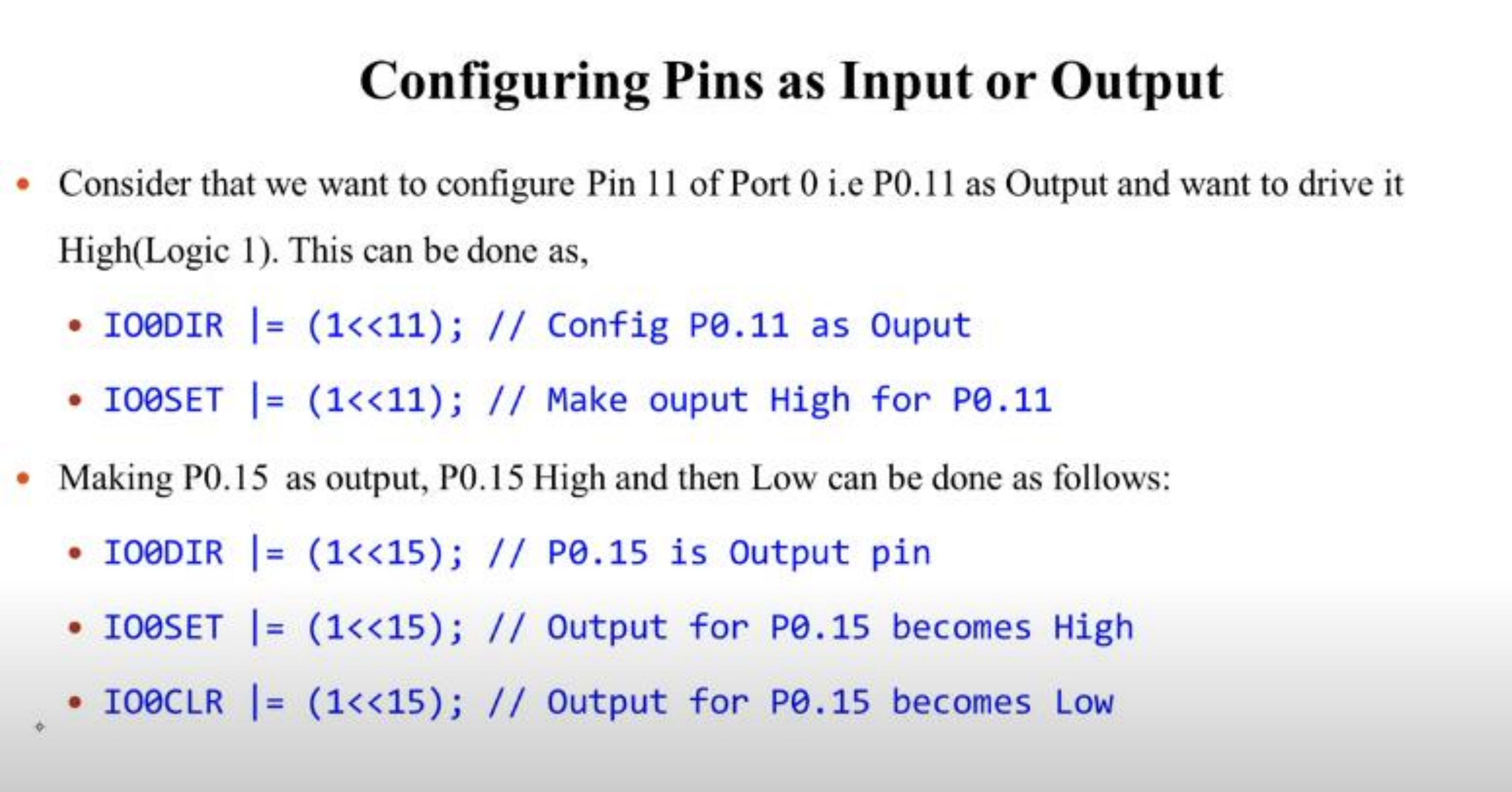
This is a direct assignment method where the binary value (1) is set directly on the pin. All the other pins are set to 0. This method should be avoided as the value is directly being assigned in the register and while P0.3 is assigned as ‘1’, all the other pins are forced to be assigned ‘0’.

```
2. Method 2: IO0DIR | = 0x00000008;
```

In this method, the hexadecimal value of the register is assigned after ORing the register with itself. In this way, the other pins other than the desired pin (P0.3 in this case) are not affected. This method is useful if we want to assign many pins without affecting the other pins.

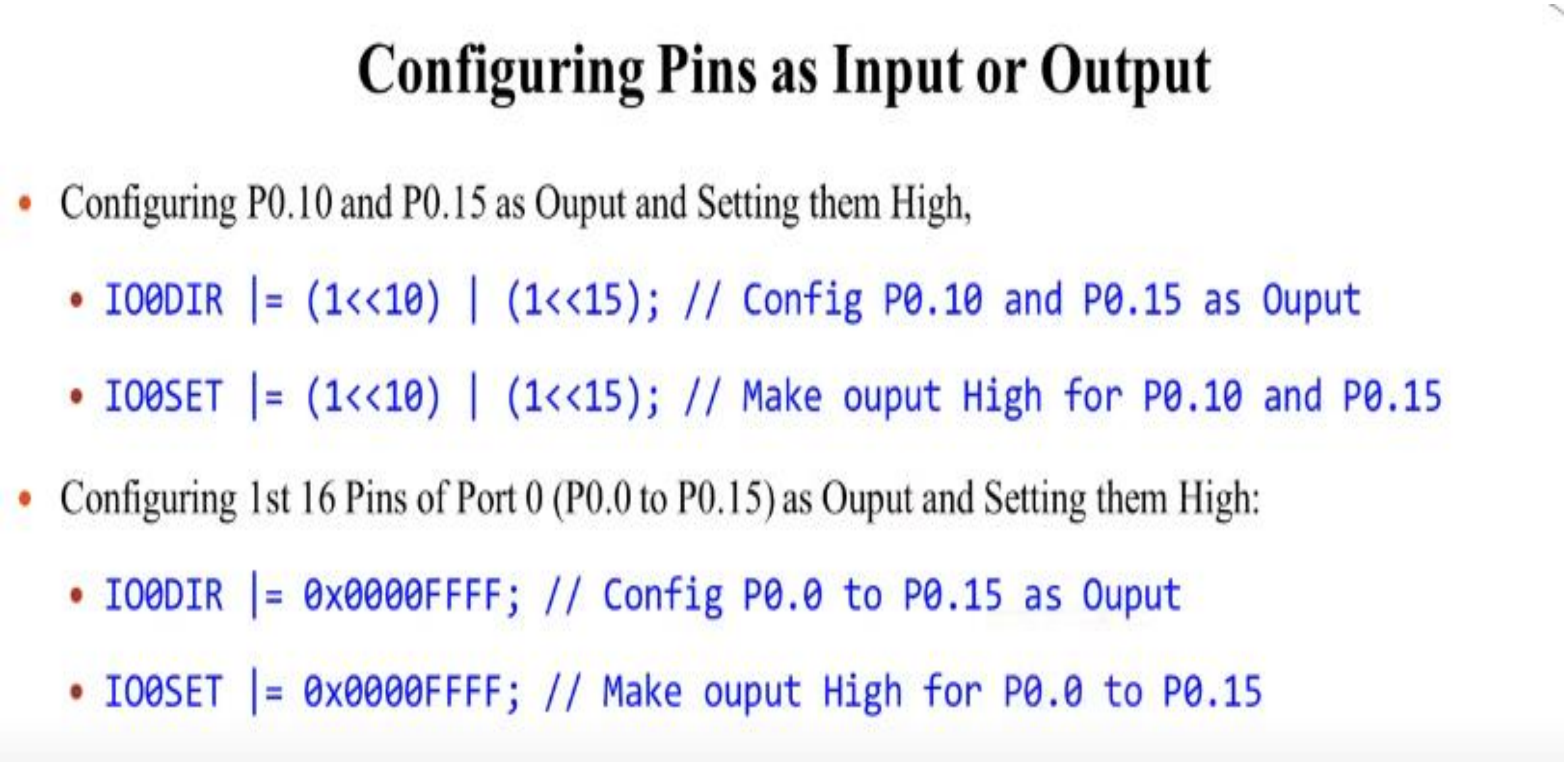
```
3. Method 3: IO0DIR | = (1<<3);
```

This is similar to the above method except that only a single pin is affected.



Configuring Pins as Input or Output

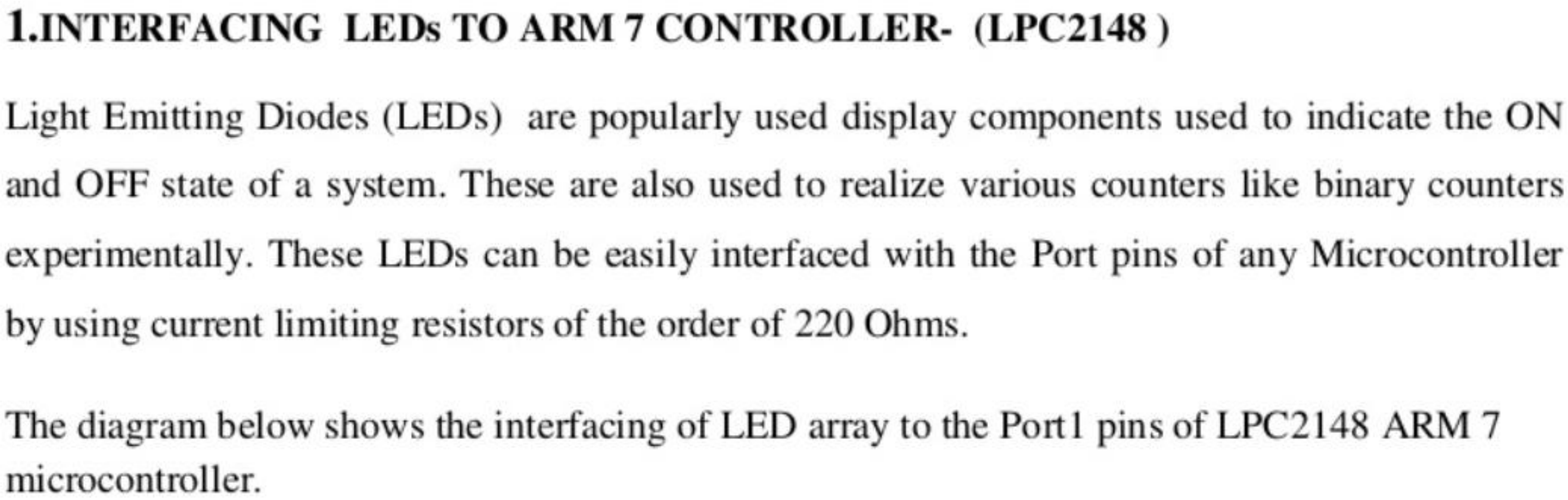
- Consider that we want to configure Pin 11 of Port 0 i.e P0.11 as Output and want to drive it High(Logic 1). This can be done as,
  - IO0DIR |= (1<<11); // Config P0.11 as Ouput
  - IO0SET |= (1<<11); // Make ouput High for P0.11
- Making P0.15 as output, P0.15 High and then Low can be done as follows:
  - IO0DIR |= (1<<15); // P0.15 is Output pin
  - IO0SET |= (1<<15); // Output for P0.15 becomes High
  - IO0CLR |= (1<<15); // Output for P0.15 becomes Low



Configuring Pins as Input or Output

- Configuring P0.10 and P0.15 as Ouput and Setting them High,
  - `IO0DIR |= (1<<10) | (1<<15); // Config P0.10 and P0.15 as Ouput`
  - `IO0SET |= (1<<10) | (1<<15); // Make ouput High for P0.10 and P0.15`
- Configuring 1st 16 Pins of Port 0 (P0.0 to P0.15) as Ouput and Setting them High:
  - `IO0DIR |= 0x0000FFFF; // Config P0.0 to P0.15 as Ouput`
  - `IO0SET |= 0x0000FFFF; // Make ouput High for P0.0 to P0.15`

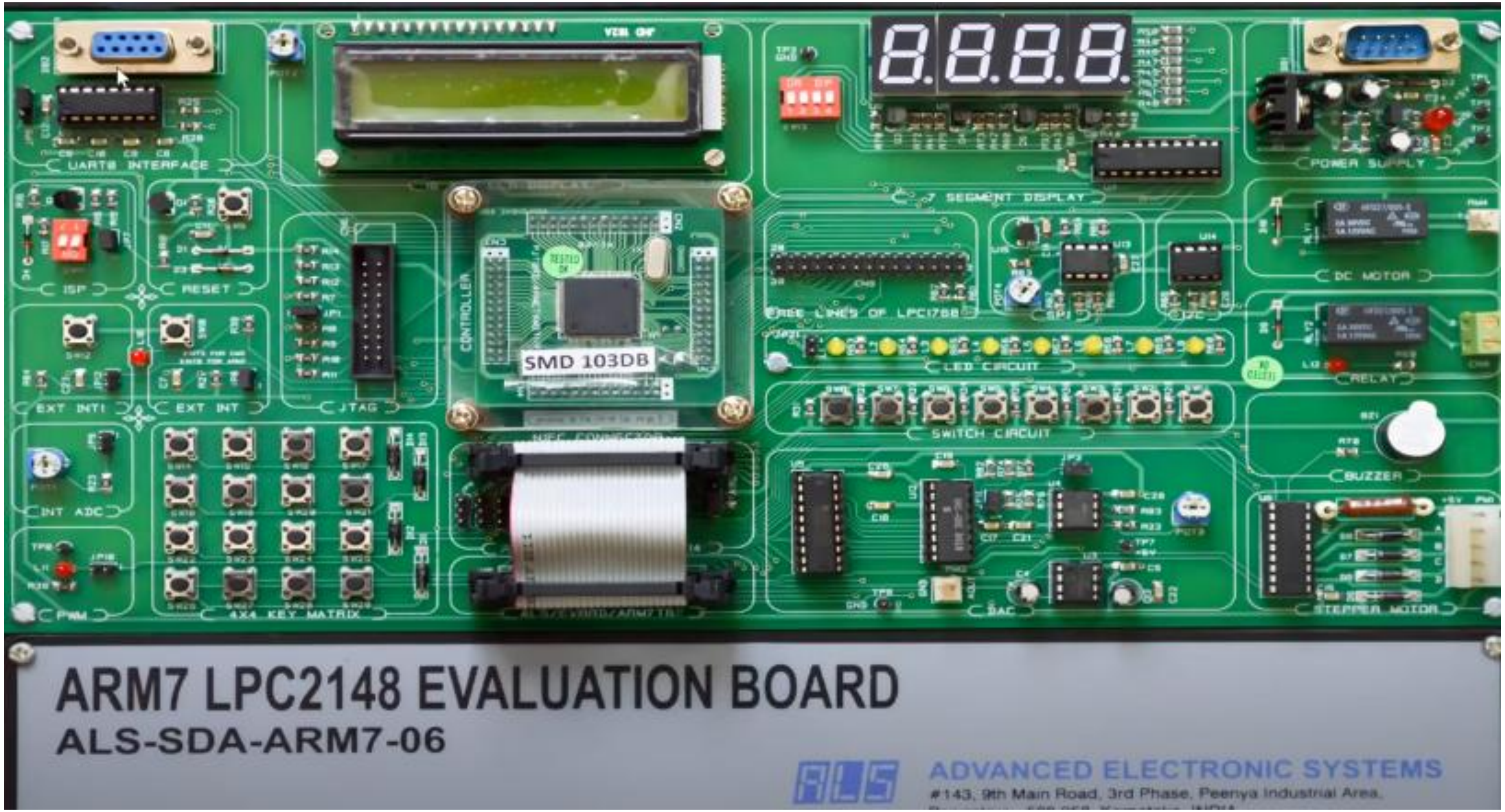
ARM LPC2148 Interfacing

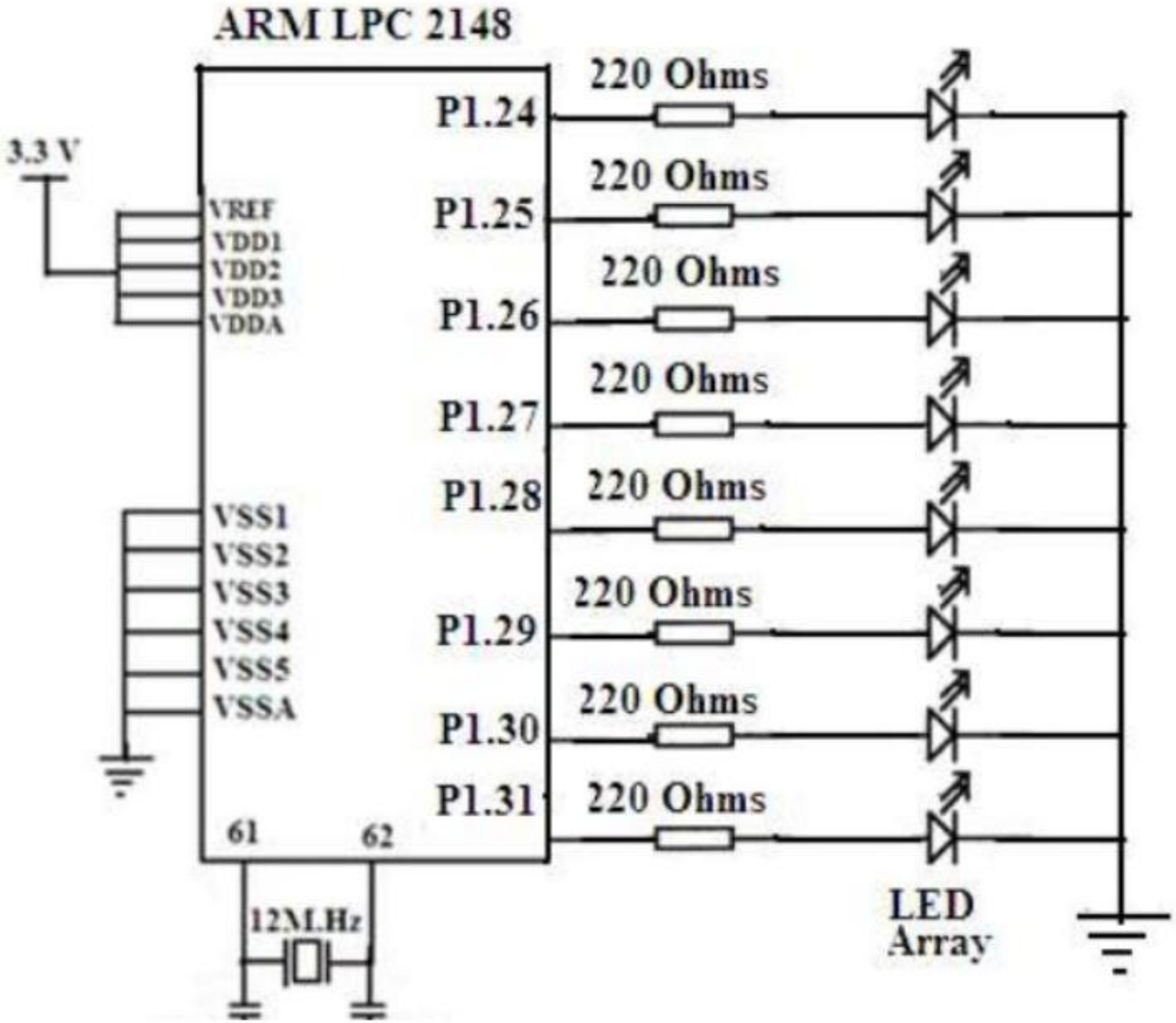


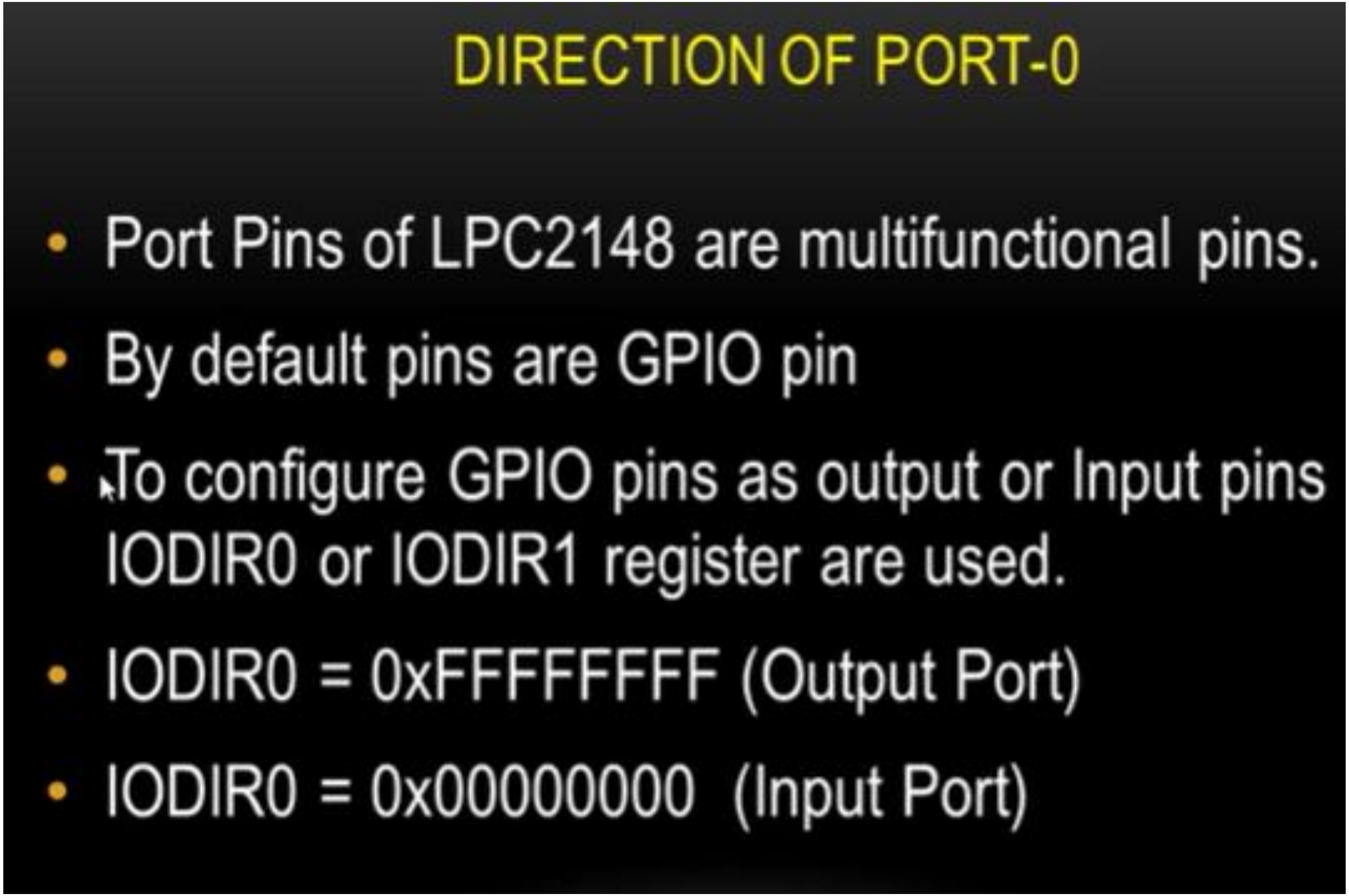
1.INTERFACING LEDs TO ARM 7 CONTROLLER- (LPC2148)

Light Emitting Diodes (LEDs) are popularly used display components used to indicate the ON and OFF state of a system. These are also used to realize various counters like binary counters experimentally. These LEDs can be easily interfaced with the Port pins of any Microcontroller by using current limiting resistors of the order of 220 Ohms.

The diagram below shows the interfacing of LED array to the Port1 pins of LPC2148 ARM 7 microcontroller.

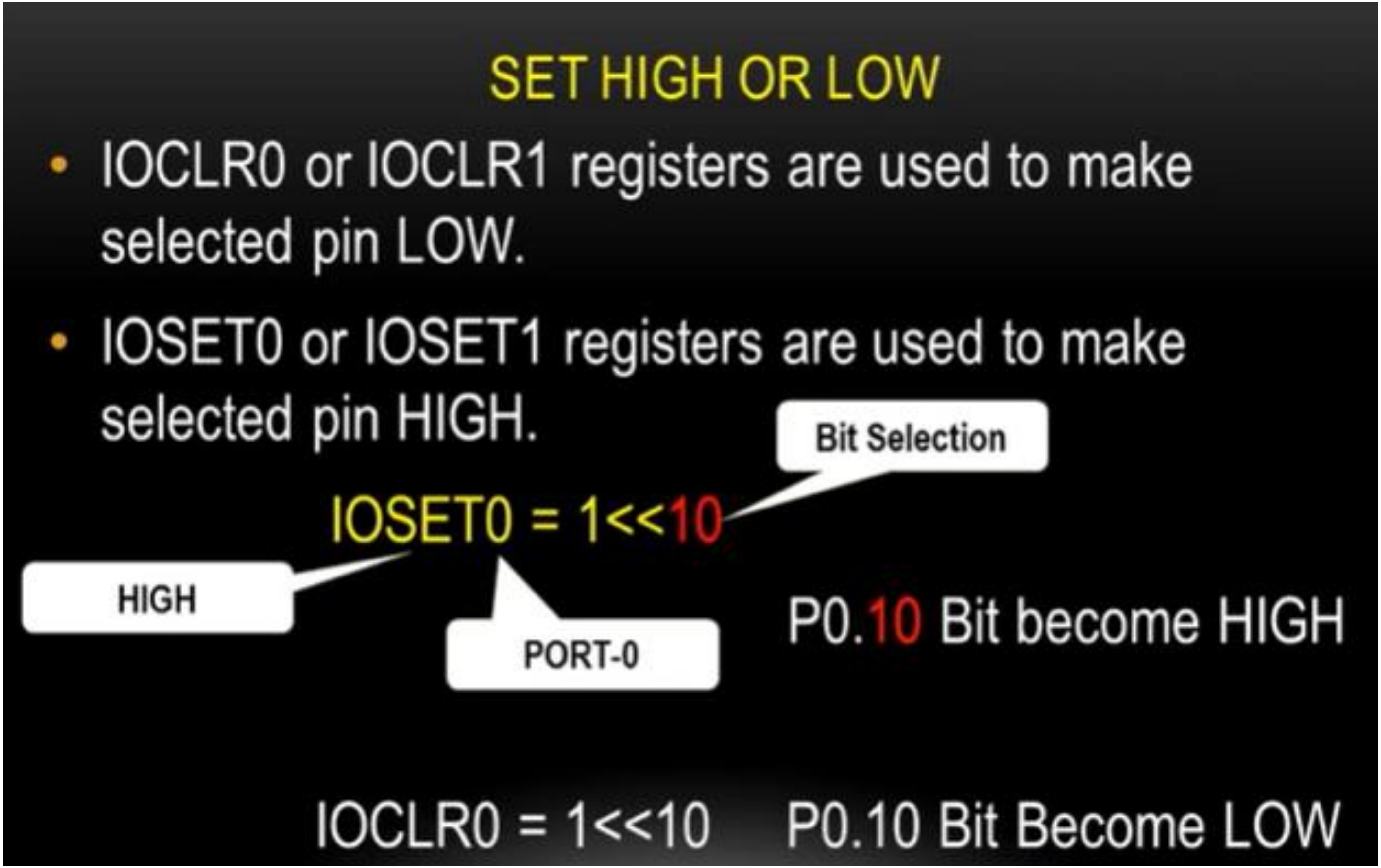

UART0 INTERFACE
ISP
RESET
EXT INT1
EXT INT
INT ADC
PWM
4X4 KEY MATRIX
JTAG
CONTROLLER
SMD 103DB
7 SEGMENT DISPLAY
POWER SUPPLY
DC MOTOR
FREE LINES OF LPC1768
SPI
I2C
LED CIRCUIT
RELAY
SWITCH CIRCUIT
BUZZER
DAC
STEPPER MOTOR
ARM7 LPC2148 EVALUATION BOARD
ALS-SDA-ARM7-06
ALS
ADVANCED ELECTRONIC SYSTEMS
#143, 9th Main Road, 3rd Phase, Peenya Industrial Area,


ARM LPC 2148
3.3 V
VREF
VDD1
VDD2
VDD3
VDDA
VSS1
VSS2
VSS3
VSS4
VSS5
VSSA
61
62
12M.Hz
P1.24
P1.25
P1.26
P1.27
P1.28
P1.29
P1.30
P1.31
220 Ohms
220 Ohms
220 Ohms
220 Ohms
220 Ohms
220 Ohms
220 Ohms
220 Ohms
LED Array



DIRECTION OF PORT-0

- Port Pins of LPC2148 are multifunctional pins.
- By default pins are GPIO pin
- To configure GPIO pins as output or Input pins IODIR0 or IODIR1 register are used.
- IODIR0 = 0xFFFFFFFF (Output Port)
- IODIR0 = 0x00000000 (Input Port)



SET HIGH OR LOW

- IOCLR0 or IOCLR1 registers are used to make selected pin LOW.
- IOSET0 or IOSET1 registers are used to make selected pin HIGH.

Bit Selection

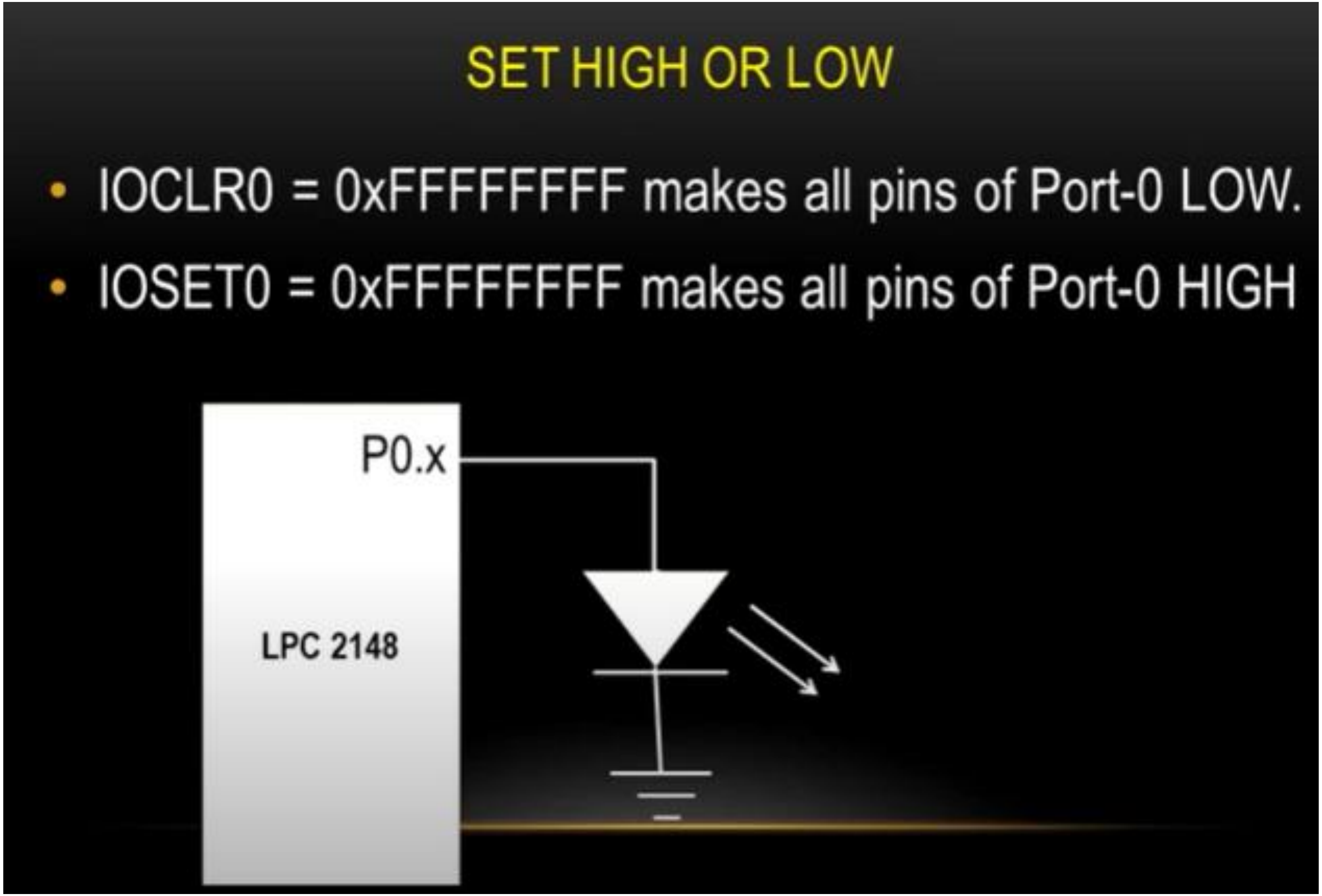
IOSET0 = 1<<10

HIGH

PORT-0

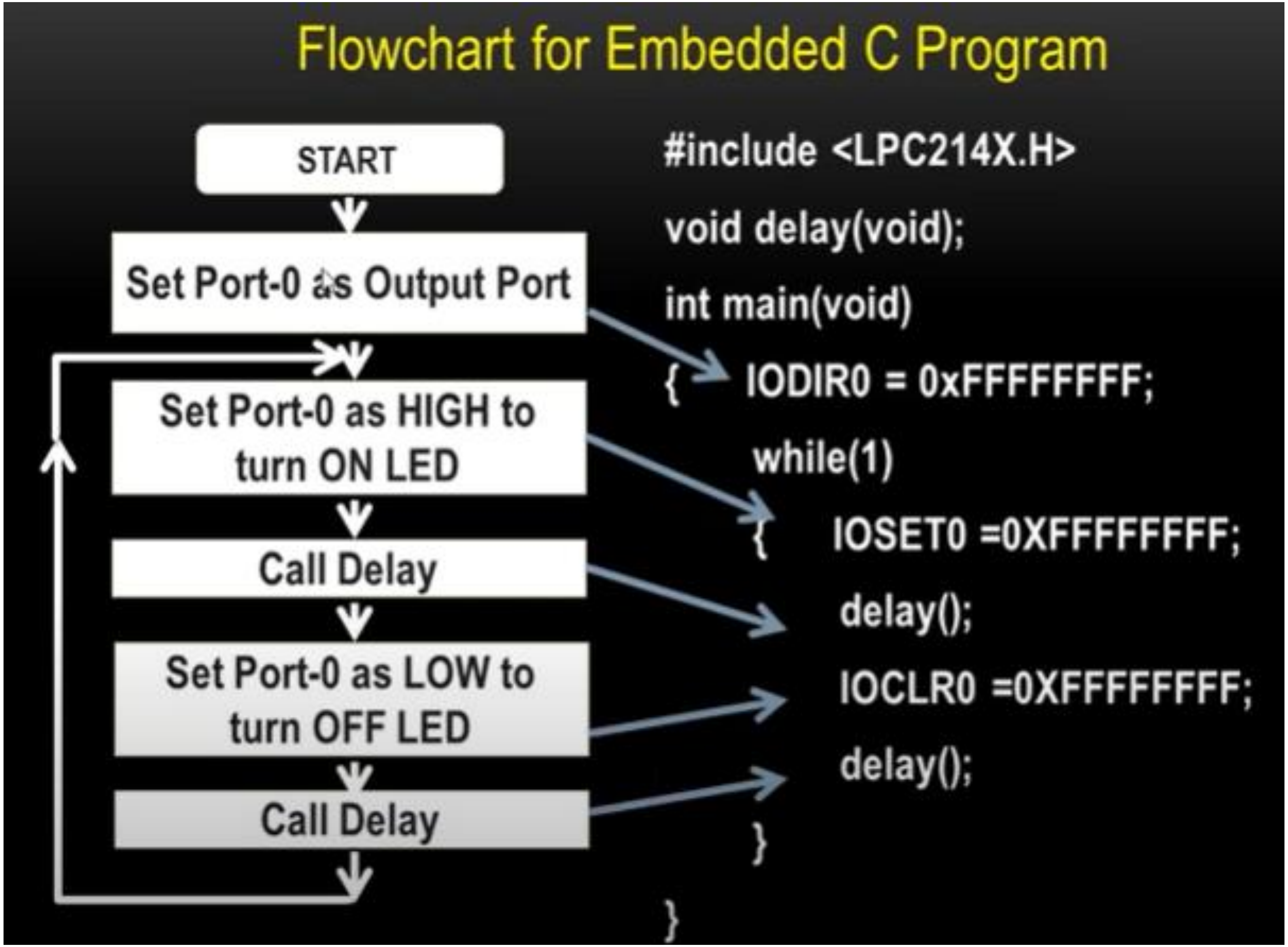
P0.10 Bit become HIGH

IOCLR0 = 1<<10 P0.10 Bit Become LOW



SET HIGH OR LOW

- IOCLR0 = 0xFFFFFFFF makes all pins of Port-0 LOW.
- IOSET0 = 0xFFFFFFFF makes all pins of Port-0 HIGH

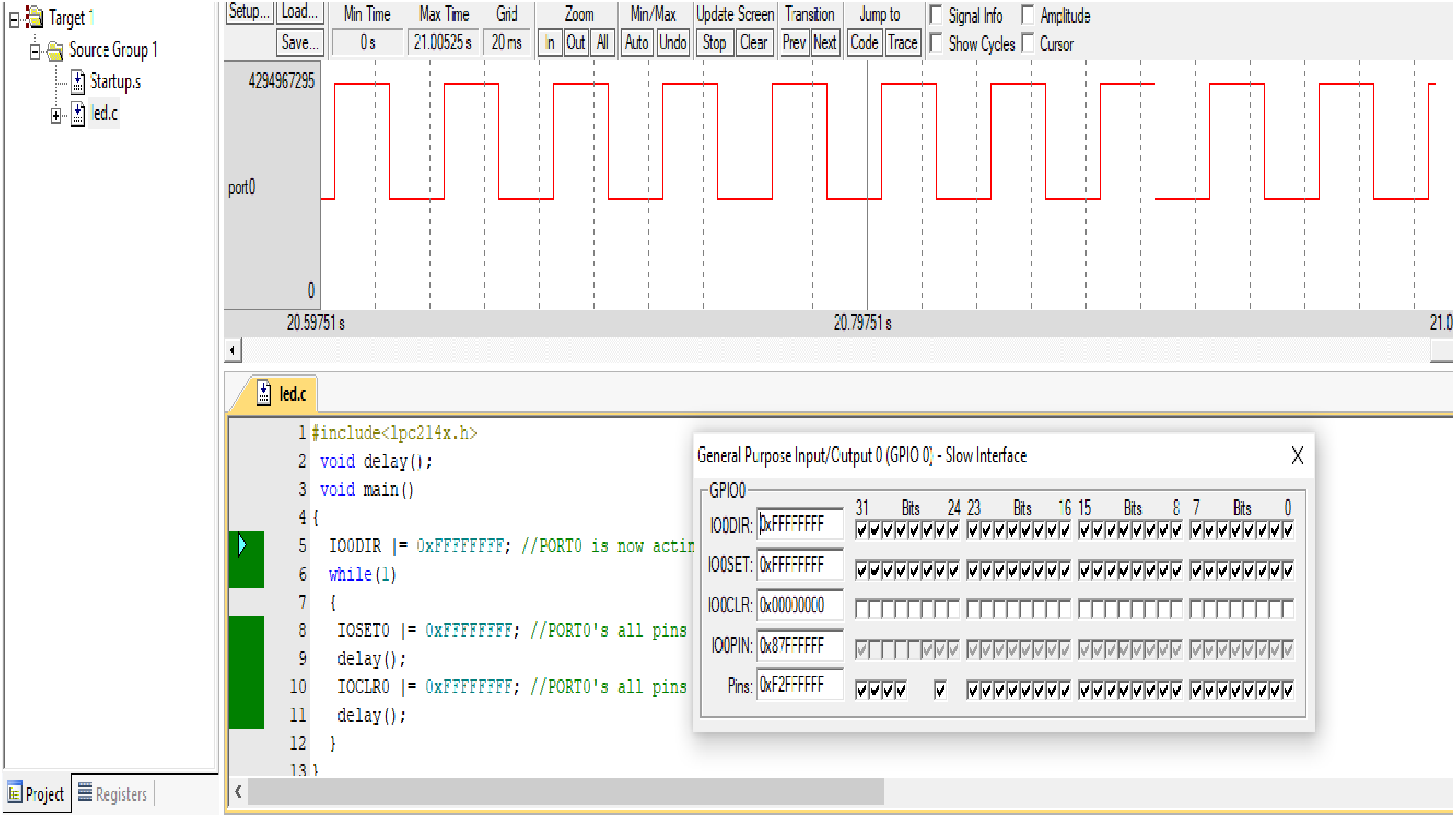


Flowchart for Embedded C Program

START
Set Port-0 as Output Port
Set Port-0 as HIGH to turn ON LED
Call Delay
Set Port-0 as LOW to turn OFF LED
Call Delay

```
#include <LPC214X.H>
void delay(void);
int main(void)
{    IODIR0 = 0xFFFFFFFF;
     while(1)
     {    IOSET0 =0XFFFFFFFF;
          delay();
          IOCLR0 =0XFFFFFFFF;
          delay();
     }
}
```

```
#include<lpc214x.h>
 void delay();
 void main()
{
  IO0DIR |= 0xFFFFFFFF; //PORT0 is now acting as a output pin
  while(1)
  {
   IOSET0 |= 0xFFFFFFFF; //PORT0's all pins are HIGH now(LED is glowing)
   delay();
   IOCLR0 |= 0xFFFFFFFF; //PORT0's all pins are LOW (LED is OFF)
   delay();
  }
}
void delay()
{
unsigned int i;
for(i=0;i<30000;i++);
}
```

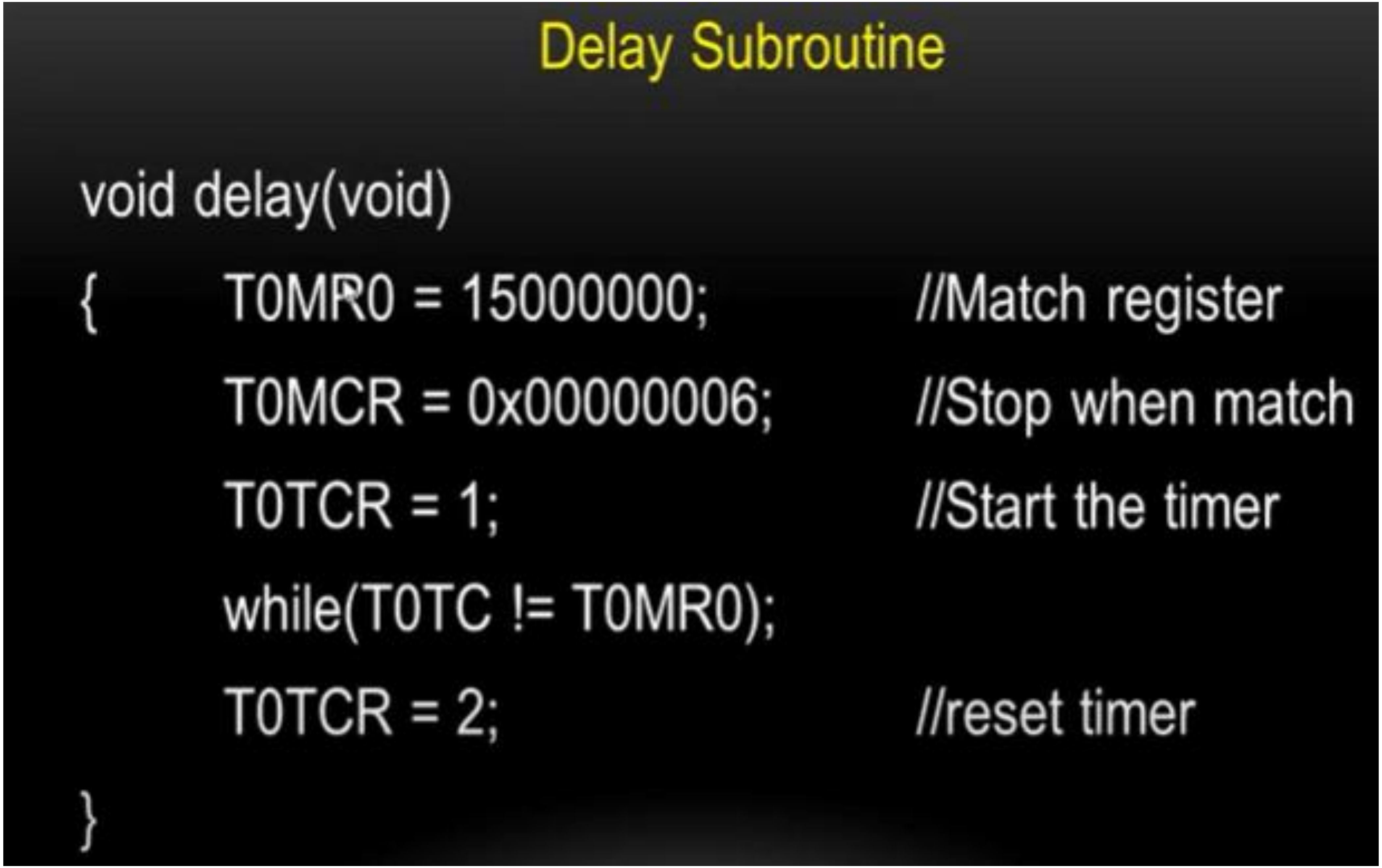


Min Time: 0 s
Max Time: 21.00525 s
Grid: 20 ms

```
#include<lpc214x.h>
void delay();
void main()
{
 IO0DIR |= 0xFFFFFFFF; //PORT0 is now actin
 while(1)
 {
  IOSET0 |= 0xFFFFFFFF; //PORT0's all pins
  delay();
  IOCLR0 |= 0xFFFFFFFF; //PORT0's all pins
  delay();
 }
}
```

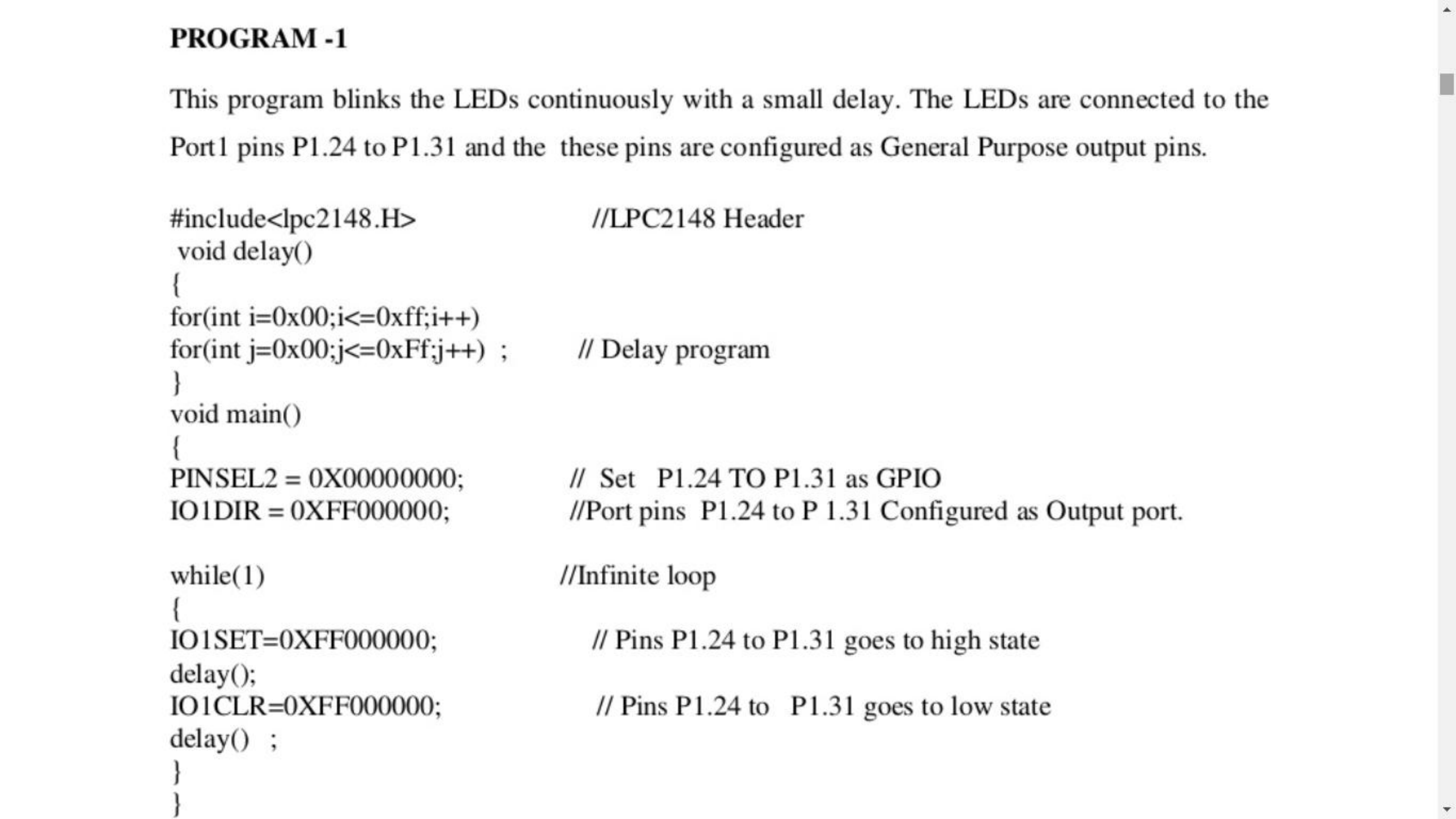
General Purpose Input/Output 0 (GPIO 0) - Slow Interface

IO0DIR: 0xFFFFFFFF
IO0SET: 0xFFFFFFFF
IO0CLR: 0x00000000
IO0PIN: 0x87FFFFFF
Pins: 0xF2FFFFFF



Delay Subroutine

```
void delay(void)
{       T0MR0 = 15000000;               //Match register
        T0MCR = 0x00000006;             //Stop when match
        T0TCR = 1;                      //Start the timer
        while(T0TC != T0MR0);
        T0TCR = 2;                      //reset timer
}
```



PROGRAM -1

This program blinks the LEDs continuously with a small delay. The LEDs are connected to the Port1 pins P1.24 to P1.31 and the these pins are configured as General Purpose output pins.

```
#include<lpc2148.H>                         //LPC2148 Header
 void delay()
{
for(int i=0x00;i<=0xff;i++)
for(int j=0x00;j<=0xFf;j++)  ;          // Delay program
}
void main()
{
PINSEL2 = 0X00000000;                  //  Set   P1.24 TO P1.31 as GPIO
IO1DIR = 0XFF000000;                   //Port pins  P1.24 to P 1.31 Configured as Output port.

while(1)                               //Infinite loop
{
IO1SET=0XFF000000;                          // Pins P1.24 to P1.31 goes to high state
delay();
IO1CLR=0XFF000000;                          // Pins P1.24 to   P1.31 goes to low state
delay()   ;
}
}
```

PROGRAM – 2

This program glows LEDs alternately by sending 55H and AAH through the port1 Pins.

```
# include  <LPC214X.H>                 //LPC2148 HEADER
void delay(void)                       // Delay Program
{
unsigned  int i;
i=0xffffff;
 while(i--);
}
int main(void)
{
PINSEL2=0x0000;                            // Port 1 is I/O

IODIR1 = 0XFF <<24   ;               // Port Pins P1.24 to P1.31 as Output Pins
while(1)                      // Infinite loop
 {


IOSET1=0X55<<25          ;     //  P1.25,P1.27,P1.29  & P1.31 LEDs will  Glow
delay()                  ;     // Call delay function
 IOCLR1=  0X55 <<25      ;     // P1.25,P1.27,P1.29 &P1.31 LEDs will be  off
IOSET1=0XAA<<24          ;   //P1.24,P1.26,P1.28 &P1.30 LEDs are Glow
delay ()                 ; // Call delay function
IOCLR1=0XAA<<24          ; // P1.24,P1.26,P1.28 &P1.30 LEDs are off
}
}
```

Interface switch with LPC2148

Switch

A **switch** is an electrical component that can break an electrical circuit, interrupting the current or diverting it from one conductor to another. A switch may be directly manipulated by a human as a control signal to a system, or to control power flow in a circuit.

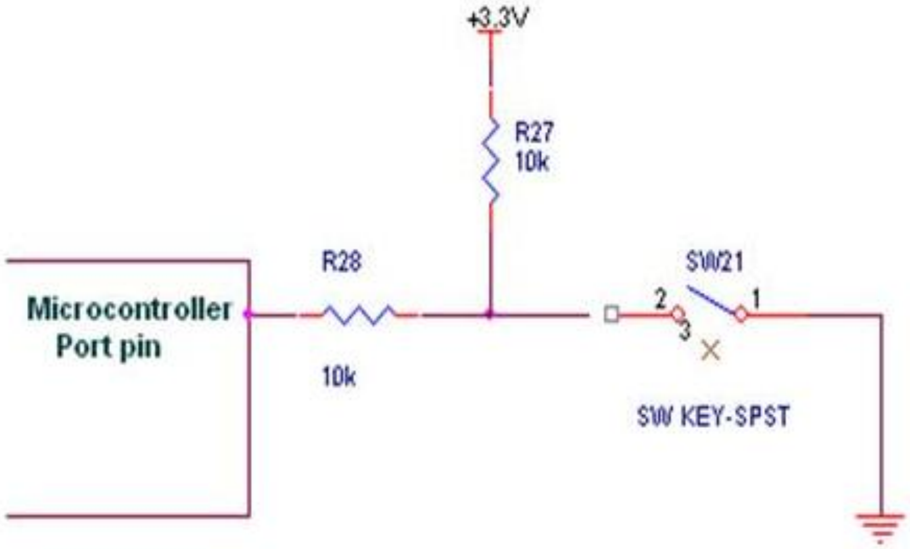
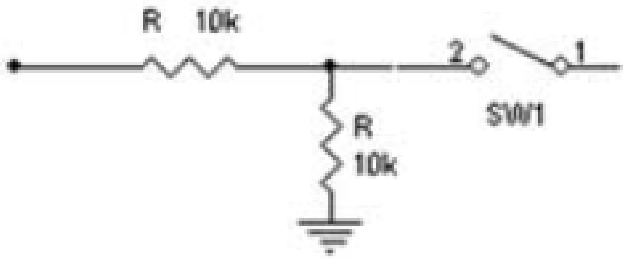
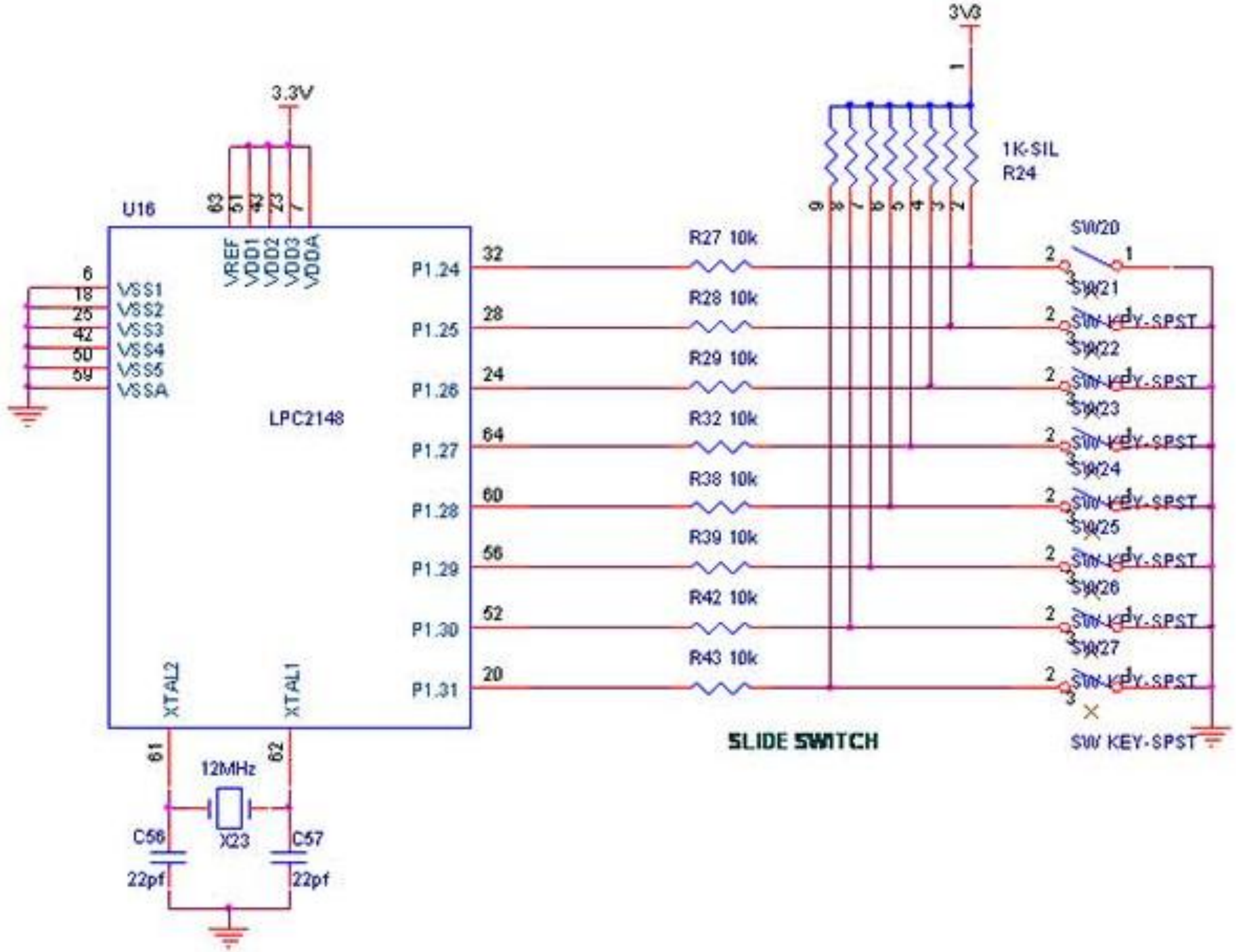


Fig. 1 Interfacing switch to Microcontroller

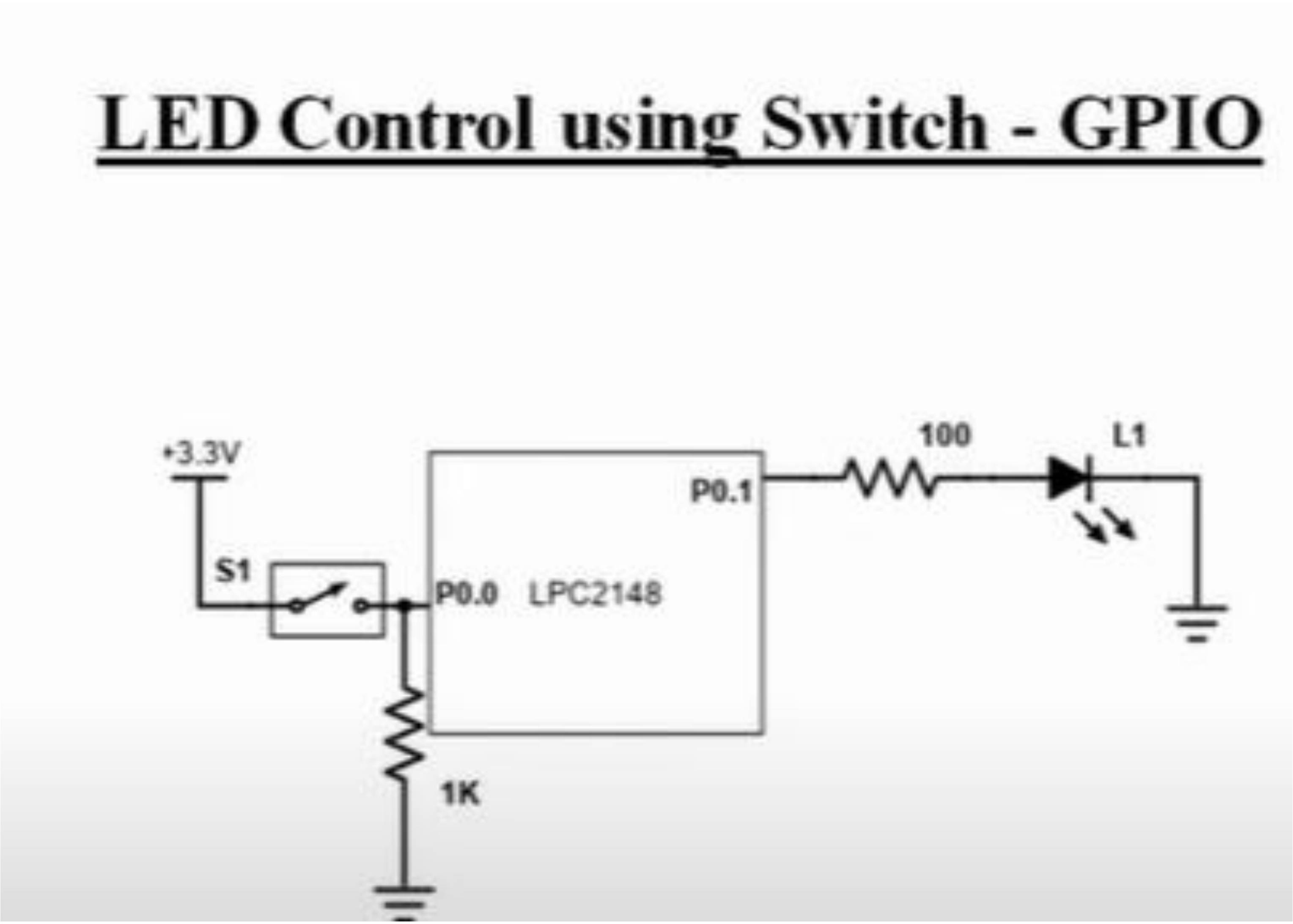
Pin Assignment with LPC2148

	Slide Switch	LPC2148 Lines	Input Logic Selection
DIGITAL INPUTS	SW20	P1.24	 Make Switch Close – Low Make Switch Open – High
	SW21	P1.25	
	SW22	P1.26	
	SW23	P1.27	
	SW24	P1.28	
	SW25	P1.29	
	SW26	P1.30	
	SW27	P1.31	

Circuit Diagram to Interface Switch with LPC2148



LED & Switch Interfacing to LPC2148



LED Control using Switch - GPIO

PINSEL Registers

- PINSEL0 – P0.0 to P0.15
- PINSEL1 – P0.16 to P0.31
- PINSEL2 – P1.16 to P1.31

PINSEL0 = 0x00000000;

PINSEL0 REGISTER

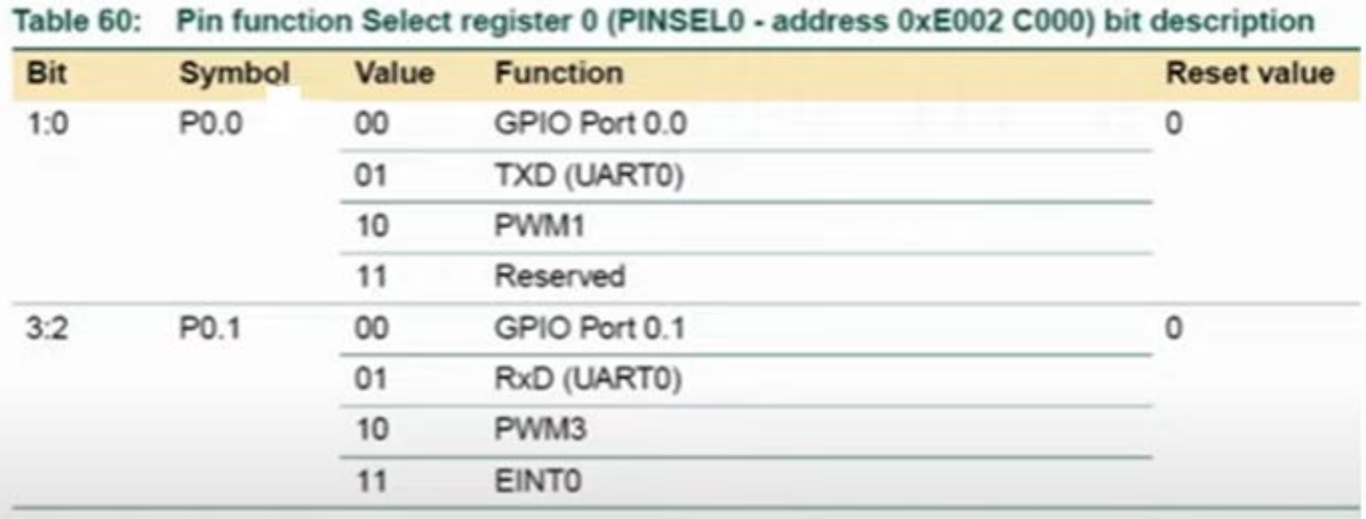


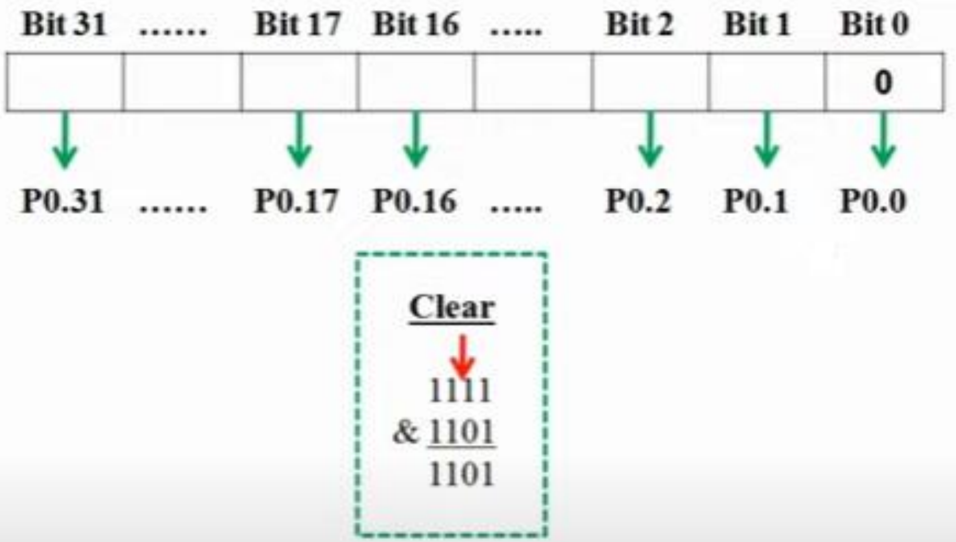
Table 60: Pin function Select register 0 (PINSEL0 - address 0xE002 C000) bit description

Bit	Symbol	Value	Function	Reset value
1:0	P0.0	00	GPIO Port 0.0	0
		01	TXD (UART0)	
		10	PWM1	
		11	Reserved	
3:2	P0.1	00	GPIO Port 0.1	0
		01	RxD (UART0)	
		10	PWM3	
		11	EINT0	

2) Configure Direction of P0.0 as Input Pin

IO0DIR &= 0xFFFFFFFE;

IODIR0 (Direction Register)



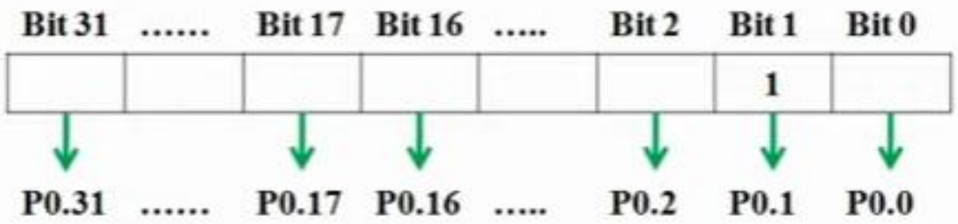
Bit 31	……	Bit 17	Bit 16	…..	Bit 2	Bit 1	Bit 0
							0
P0.31	……	P0.17	P0.16	…..	P0.2	P0.1	P0.0

STEPS FOR LED CONTROL USING SWITCH

3) Configure Direction of P0.1 as Output Pin

```
IO0DIR |= 0x00000002;
```

IODIR0 (Direction Register)



Set

```
  1101
| 0010
  1111
```

IO0DIR |= 0x00000002;

STEPS FOR LED CONTROL USING SWITCH

4) Check Switch connected to P0.0 is Pressed

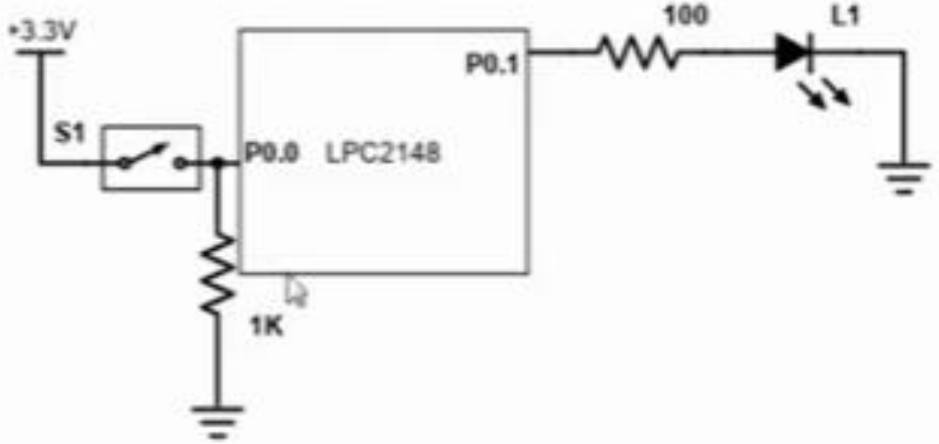
```
if((IOPIN0 & 0x01)==1)
```

IOPIN0 (Data Register)

Bit 31	……	Bit 17	Bit 16	…..	Bit 2	Bit 1	Bit 0
							RD
P0.31	……	P0.17	P0.16	…..	P0.2	P0.1	P0.0

Split a Bit

```
  1111
& 0010
  0010
```



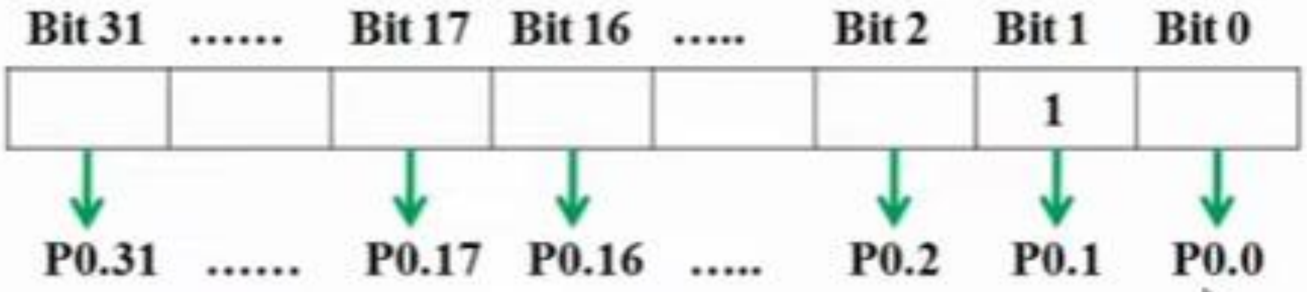
if((IOPIN0 & 0x01)==1)

STEPS FOR LED CONTROL USING SWITCH

5) If Pressed, Switch ON LED connected to P0.1

```
IOSET0 |= 0x00000002;
```

IOSET0 (Data Register - Set)

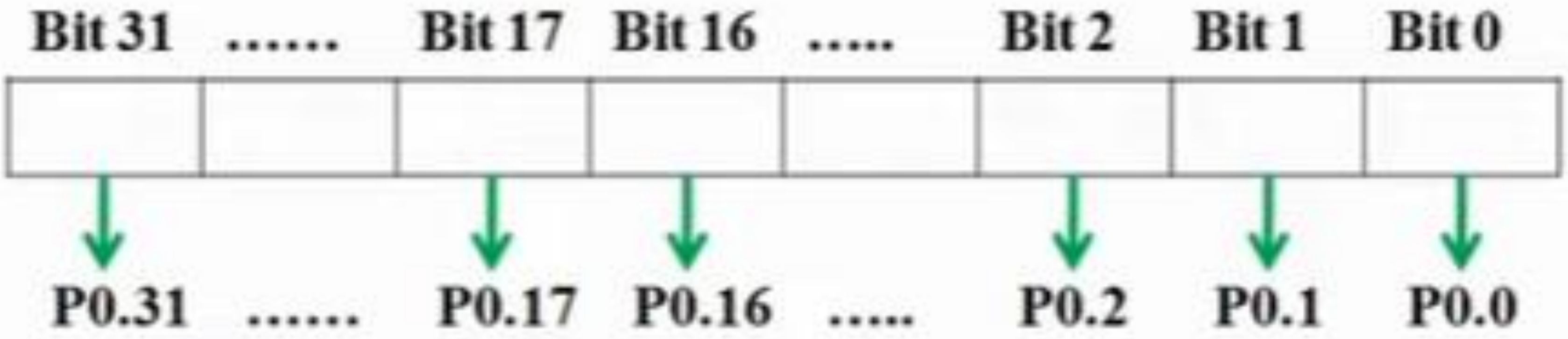


Bit 31	……	Bit 17	Bit 16	…..	Bit 2	Bit 1	Bit 0
						1	
P0.31	……	P0.17	P0.16	…..	P0.2	P0.1	P0.0

1- Set

```
IOSET0 |= 0x00000002;
```

IOCLR0 (Data Register - Clear)



1- Clear

```
IOCLR0 |= 0x00000002;
```

STEPS FOR LED CONTROL USING SWITCH

6) Else Switch OFF LED connected to P0.1

```
        IOCLR0 |= 0x00000002;
```

STEPS FOR LED CONTROL USING SWITCH

1) Configure functionality of P0.0 and P0.1 as GPIO Port

   ```
   PINSEL0 = 0x00000000;
   ```

2) Configure Direction of P0.0 as Input Pin

   ```
   IO0DIR &= 0xFFFFFFFE;
   ```

3) Configure Direction of P0.1 as Output Pin

   ```
   IO0DIR |= 0x00000002;
   ```

4) Check if Switch connected to P0.0 is Pressed

   ```
   if((IOPIN0 & 0x01)==1)
   ```

5) If Pressed, Switch ON LED connected to P0.1

   ```
   IOSET0 |= 0x00000002;
   ```

6) Else Switch OFF LED connected to P0.1

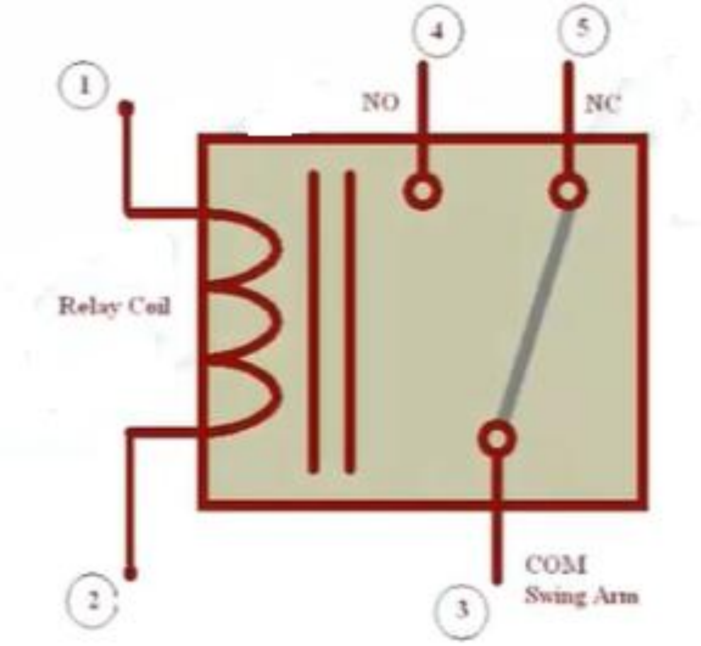
   ```
   IOCLR0 |= 0x00000002;
   ```

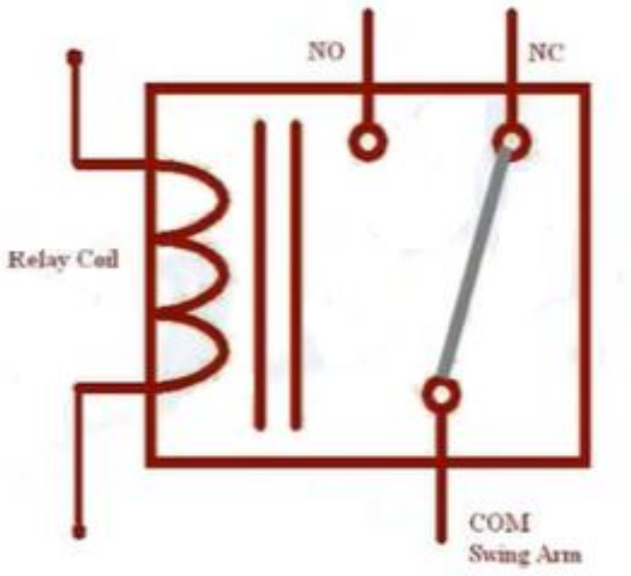
SWITCH & LED Interfacing with LPC 2148

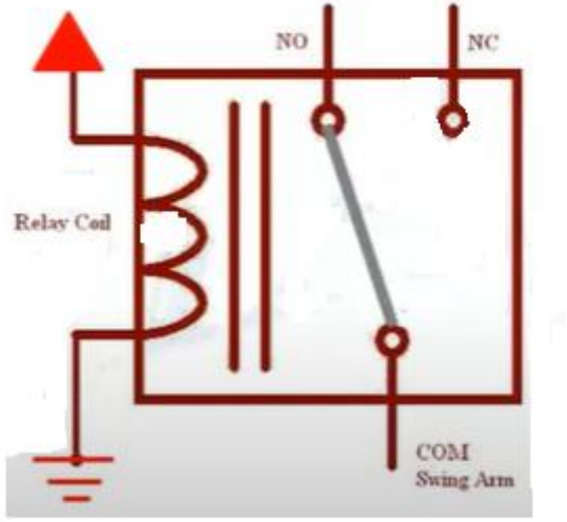
```
#include<21xx.h>
Char switchstatus
Int main (void)
{  PINSEL0 = 0x00000000;
IO0DIR0 &= 0xFFFFFFFE; //P0.0 Switch
IO0DIR0  |= 0x00000002; //P0.1 Led
  while(1)
{
    switchstatus =( IOPIN0 & 0x01)
    if(switchstatus == 1)
  {
      IOSET0 |= 0x00000002;
}  else
   {
  IOCLR0  |= 0x 00000002;
}
}
}
```

Relay Interfacing with LPC 2148

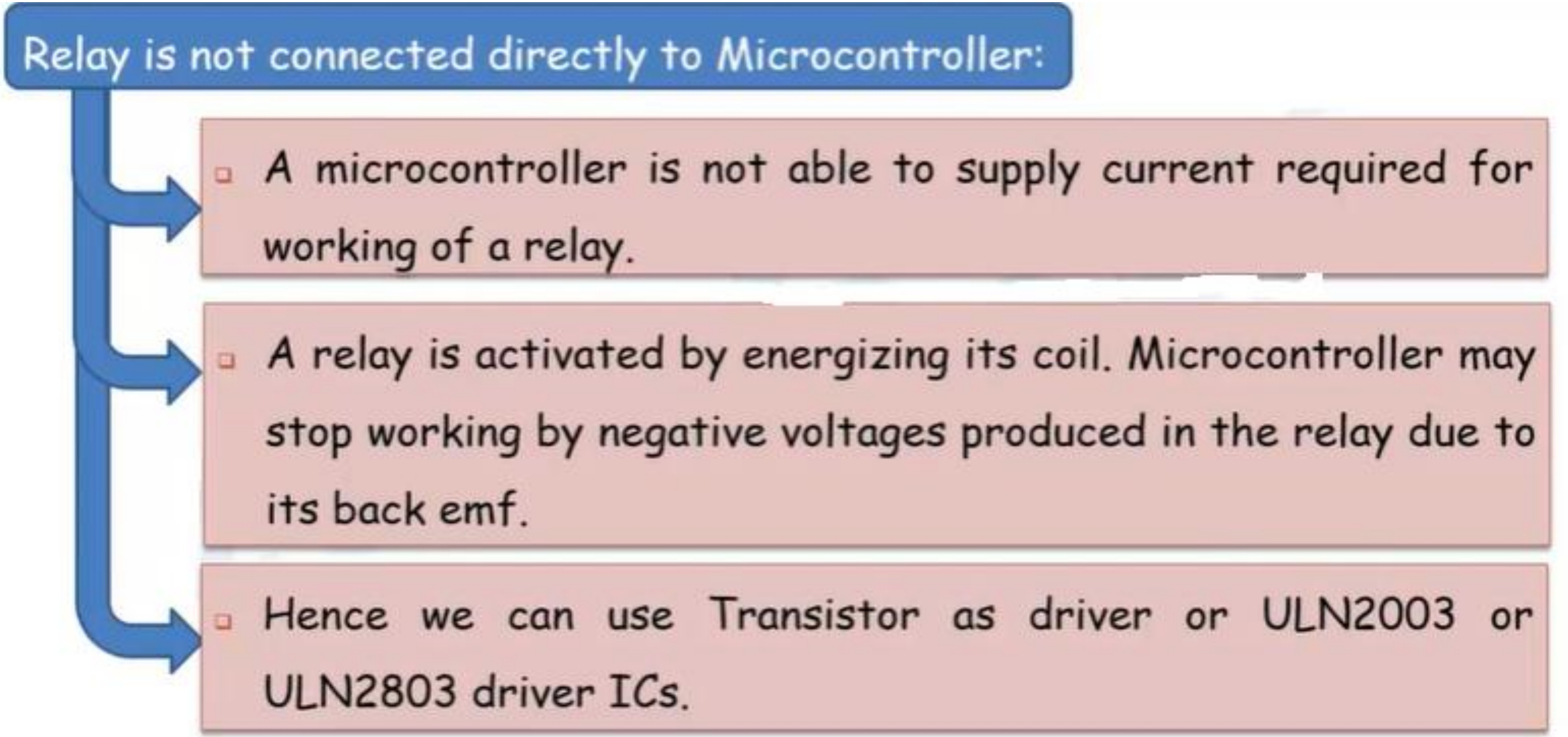
- A relay is an electromagnetic switch , which performs ON & OFF operations.







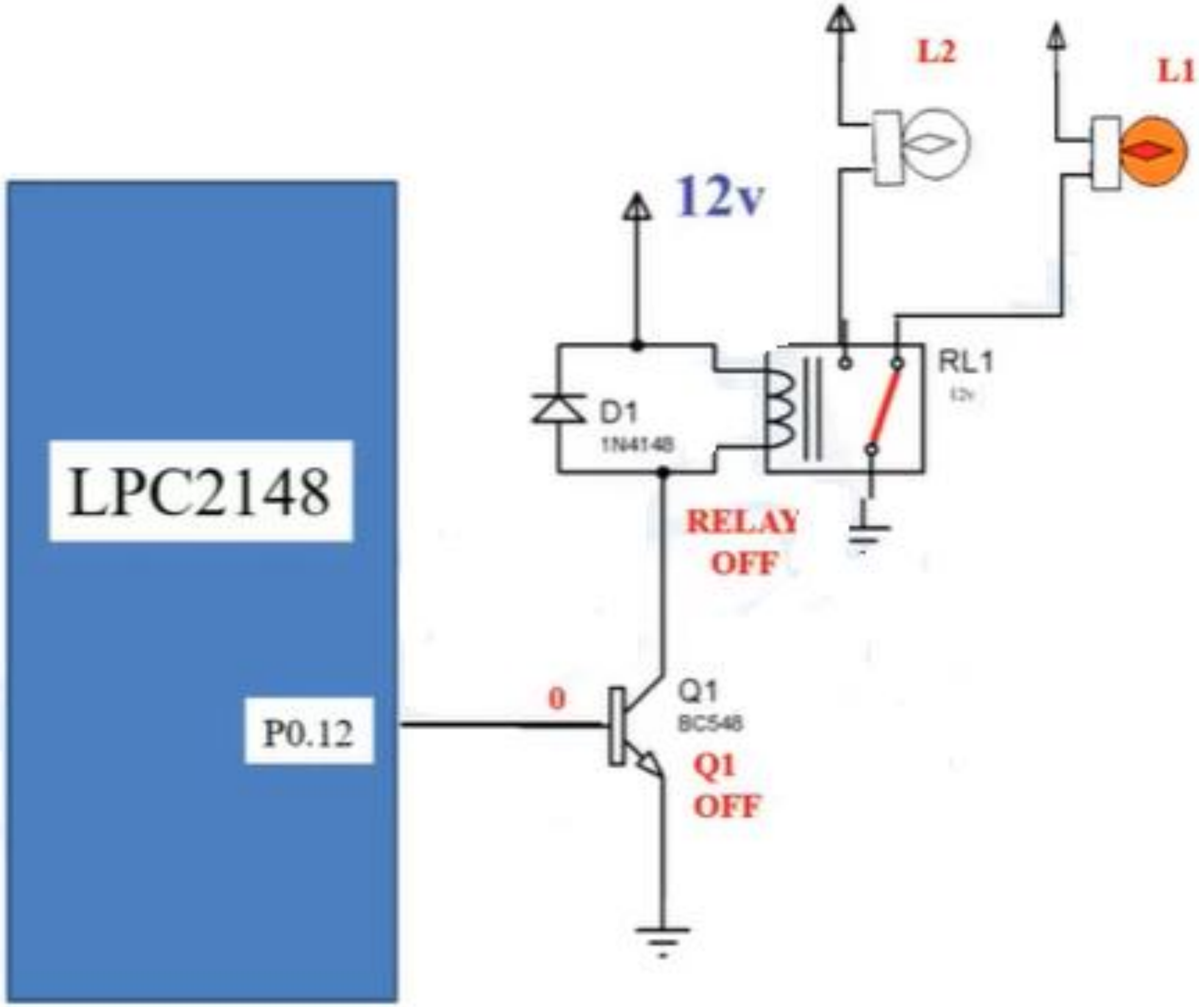
Relay Interfacing with LPC 2148



Relay is not connected directly to Microcontroller:

- A microcontroller is not able to supply current required for working of a relay.
- A relay is activated by energizing its coil. Microcontroller may stop working by negative voltages produced in the relay due to its back emf.
- Hence we can use Transistor as driver or ULN2003 or ULN2803 driver ICs.

Relay Interfacing with LPC 2148

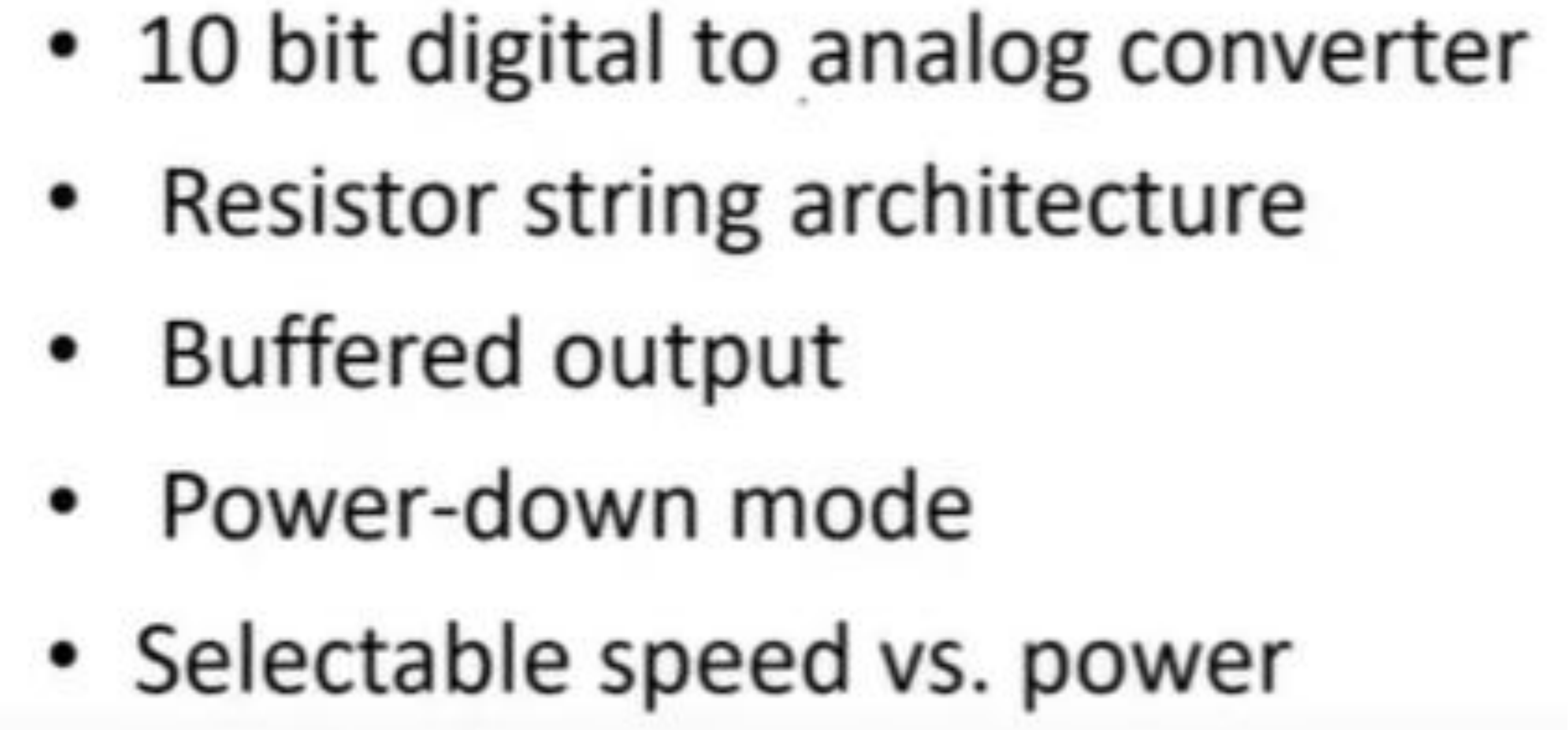


Relay Program

```
 #include<LPC214x.h>
 Void delay()
{
 int i ,j;
 for(i=0; i<1000; i++)
 for(j=0; j<2000; j++);
 }
 int main(void)
{    PINSEL0 = 0;
     IO0DIR 1<<12;  //set P0.12 as output pin
    while(1)
   {
       IOSET0 = 1<<12;    //send Logic 1 on P0.12 to make RELAY ON
        delay();
       IOCLR0 = 1<<12;   // send Logic 0 on P0.12 to make RELAY OFF
     }
  }
```

DAC interfacing with LPC2148

- DAC Features



- 10 bit digital to analog converter
- Resistor string architecture
- Buffered output
- Power-down mode
- Selectable speed vs. power

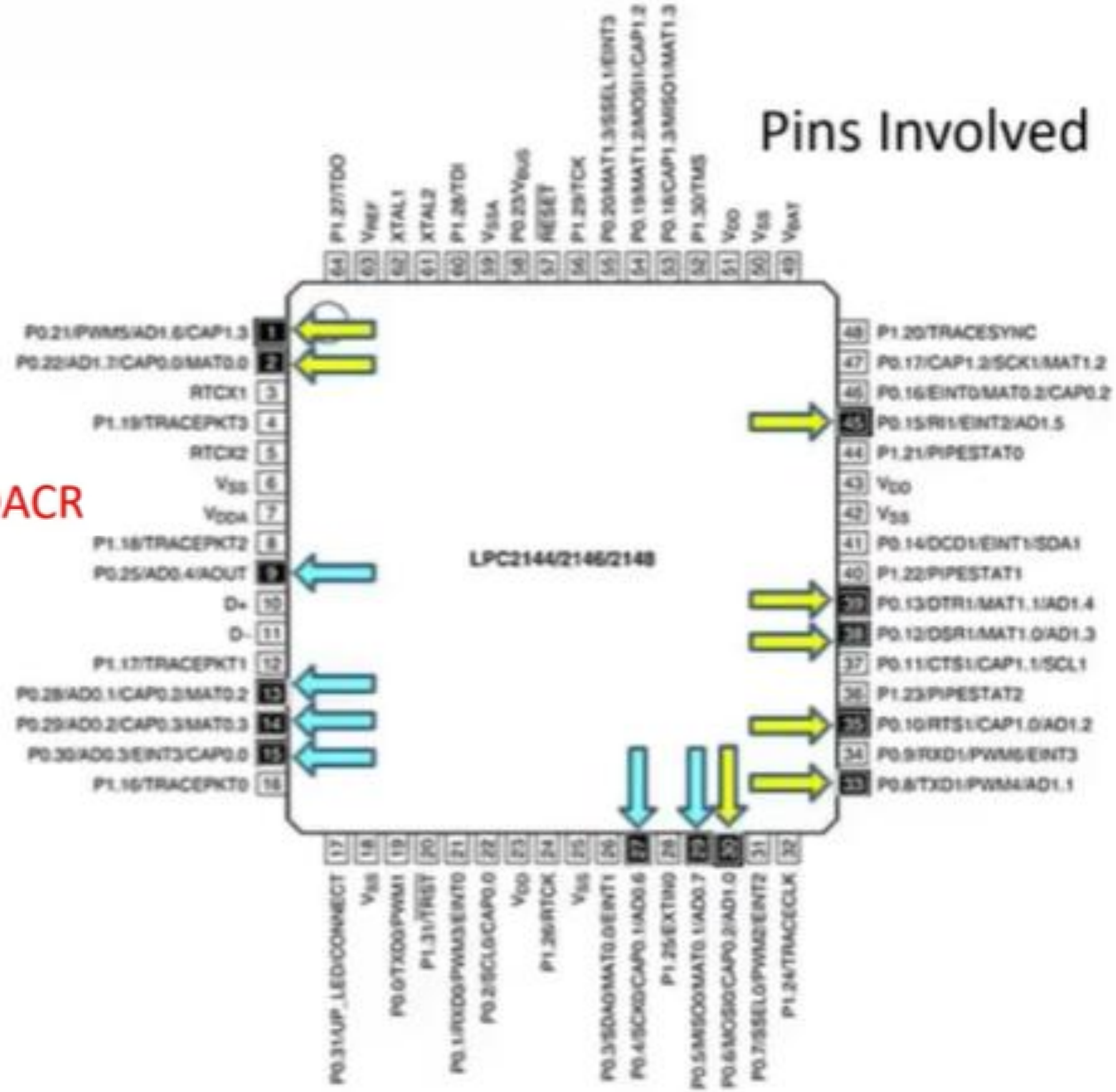
DAC

ADC0.4 [Pin 9] P0.25

SFRs Involved

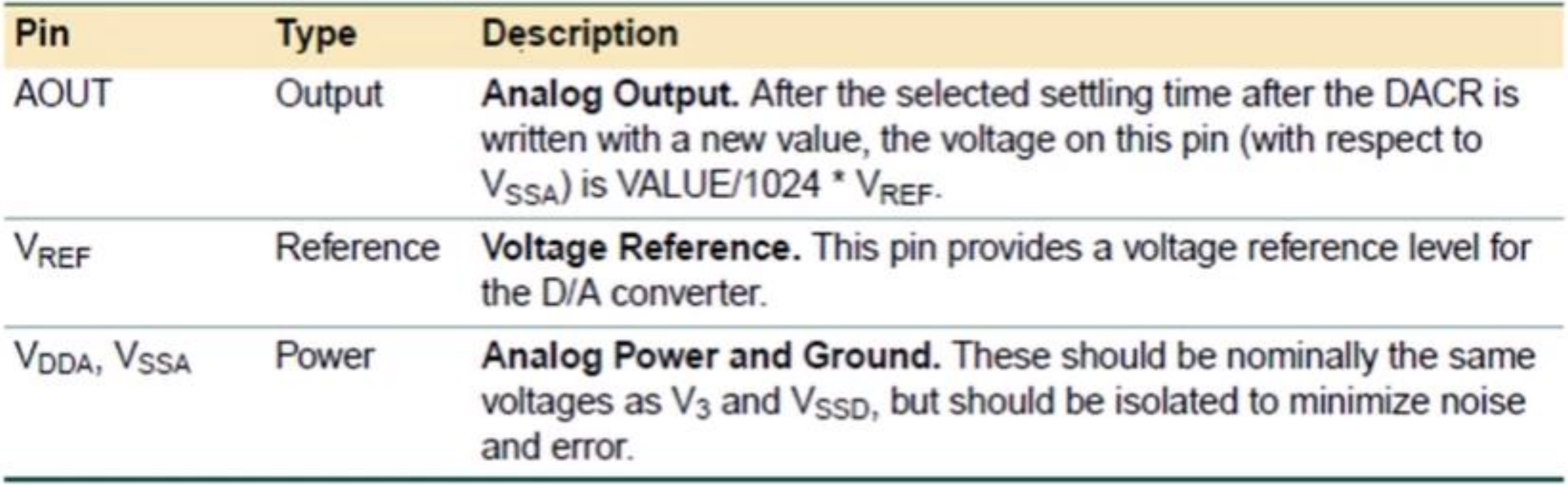
- DAC Control Register - DACR

Pins Involved



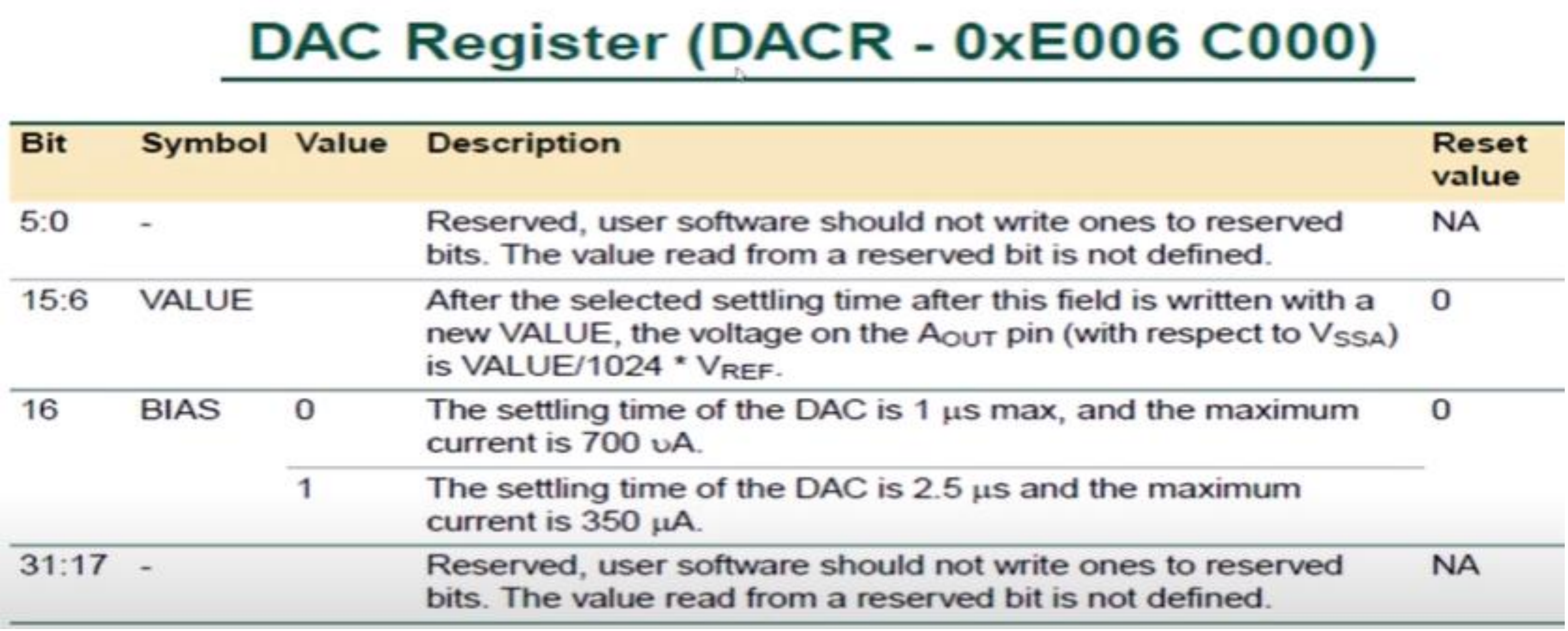
PINSEL1= 0000 0000 0000 1000 0000 0000 0000 0000

DAC pin description



Pin	Type	Description
AOUT	Output	**Analog Output.** After the selected settling time after the DACR is written with a new value, the voltage on this pin (with respect to V_{SSA}) is VALUE/1024 * V_{REF}.
V_{REF}	Reference	**Voltage Reference.** This pin provides a voltage reference level for the D/A converter.
V_{DDA}, V_{SSA}	Power	**Analog Power and Ground.** These should be nominally the same voltages as V_3 and V_{SSD}, but should be isolated to minimize noise and error.

DAC Register (DACR - 0xE006 C000)



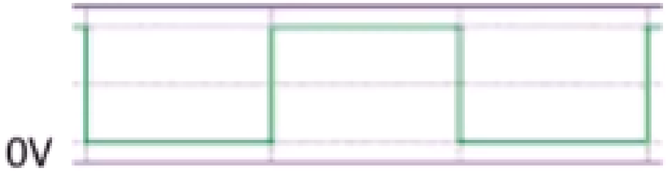
Bit	Symbol	Value	Description	Reset value
5:0	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
15:6	VALUE		After the selected settling time after this field is written with a new VALUE, the voltage on the A_{OUT} pin (with respect to V_{SSA}) is VALUE/1024 * V_{REF}.	0
16	BIAS	0	The settling time of the DAC is 1 μs max, and the maximum current is 700 υA.	0
		1	The settling time of the DAC is 2.5 μs and the maximum current is 350 μA.	
31:17	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

D16	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1

1 1 1 1 1 1 1 1 1 1

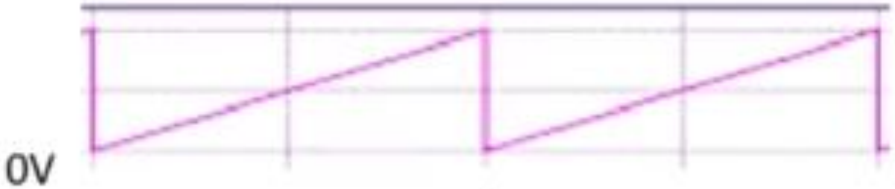
3 F F

```
/****************************** DAC:  SQUARE WAVE *********************/
#include "lpc214x.h"

/* Main Program */

void DELAY()
{
        int i;
        for(i=0;i<30000;i++);
}


 int main(void)
 {
          PINSEL1=0x80000; // 0000 0000 0000 1000 0000 0000 0000 0000
          IODIR0=0X80000;
          DELAY();
```



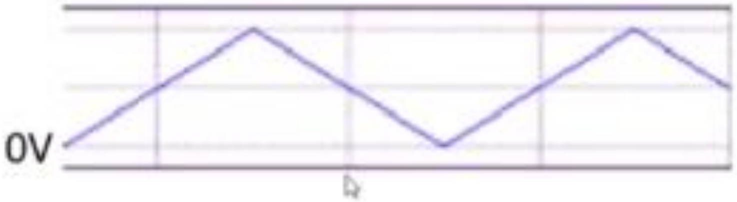
```
while(1)
        {
                DACR=(0X3FF<<6);
                DELAY();
                DACR=(0X0000<<6);
                DELAY();


        }
        DELAY();

        return 0;
}
```

```
/****************************** RAMP SIGNAL********************/
#include "lpc214x.h"

/* Main Program */

int main(void)
{        int j;
         PINSEL1=0x80000; // 0000 0000 0000 1000 0000 0000 0000 0000
         IODIR0=0X80000;
         DELAY();
```



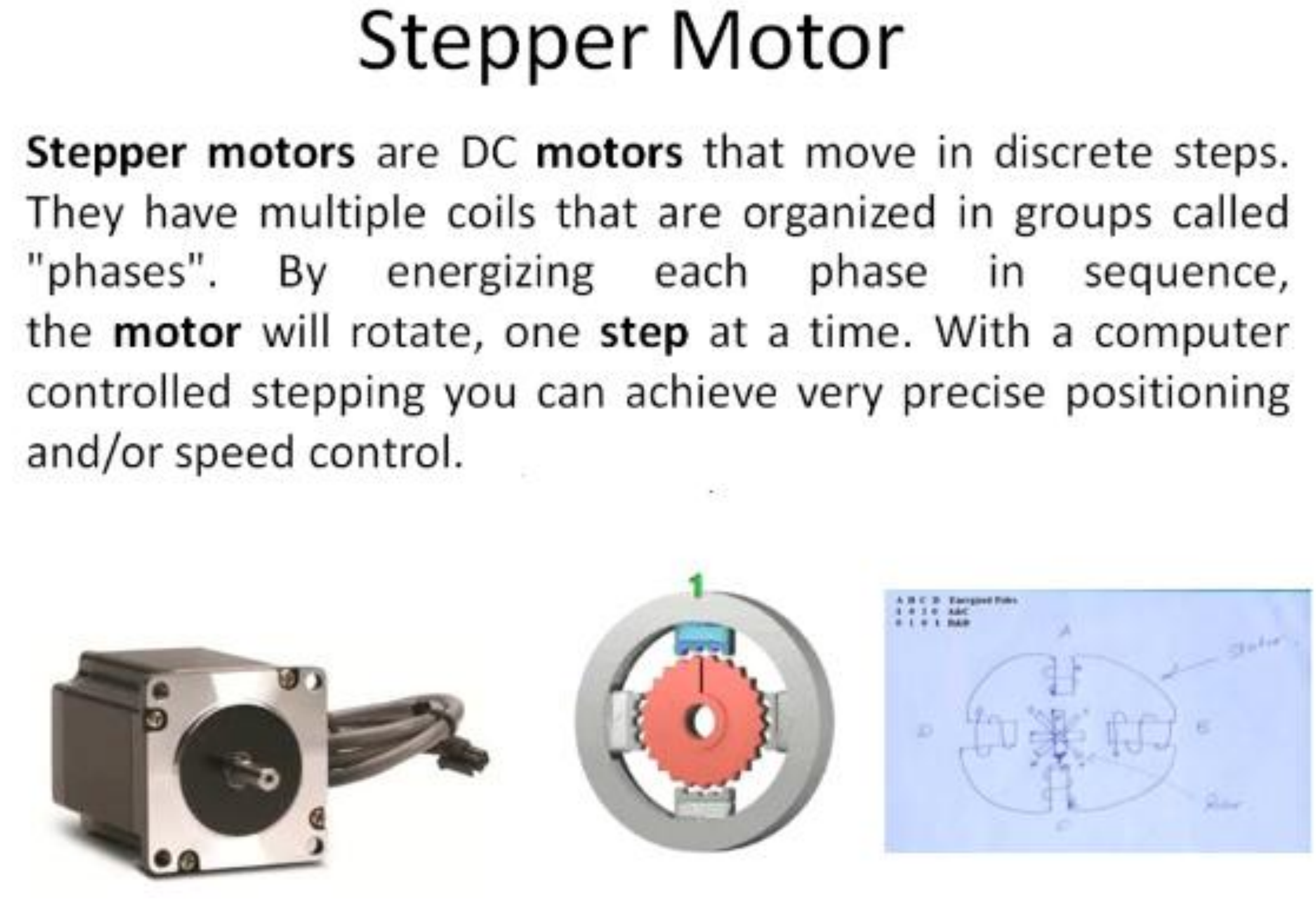
```
while(1)
        {
          for (j=0x00; j<=0x3FF;j ++)
           {
               DACR=(j<<6);

           }

        return 0;
}
```

```
/******************************TRAINGULAR SIGNAL*********************/
#include "lpc214x.h"

/* Main Program */

int main(void)
{       int j;
        PINSEL1=0x80000; // 0000 0000 0000 1000 0000 0000 0000 0000
        IODIR0=0X80000;
        DELAY();
```



```
while(1)
          {
     for(j=0x00;j<=0x3FF;j++)
            {
               DACR=(j<<6);
             }

     for(j=0x3FF;j>=0x00;j--)

             {
               DACR=(j<<6);
             }
         }
       delay();
     return 0;
     }
```

Interfacing Stepper motor with LPC2148



Stepper Motor

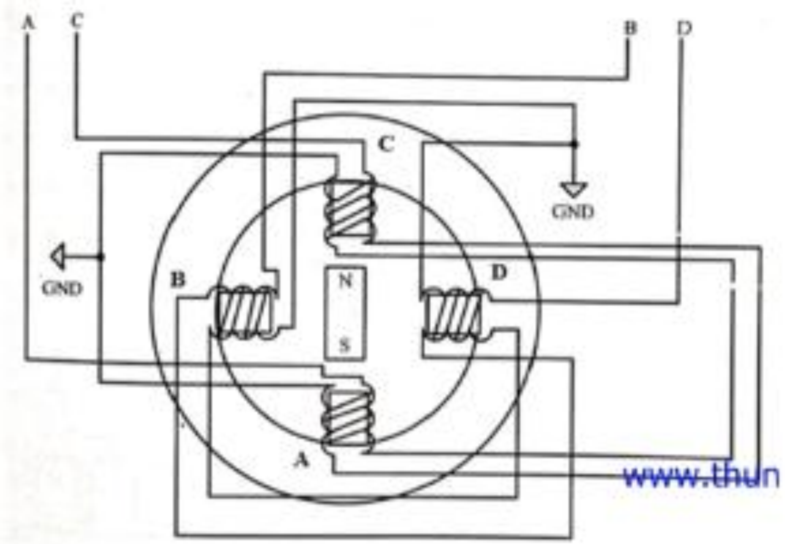
- **Stepper motors** are DC **motors** that move in discrete steps. They have multiple coils that are organized in groups called "phases". By energizing each phase in sequence, the **motor** will rotate, one **step** at a time. With a computer controlled stepping you can achieve very precise positioning and/or speed control.

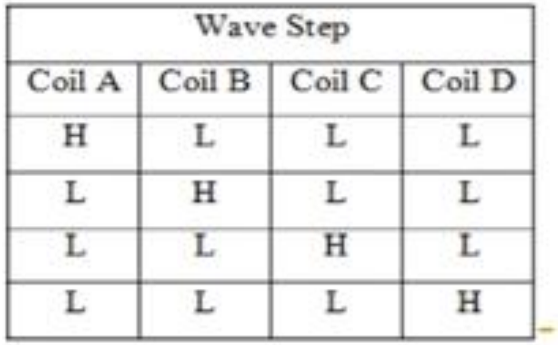
A	B	C	D	Energized Poles
1	0	1	0	A&C
0	1	0	1	B&D

1
A
B
C
D
Stator
Rotor

Interfacing Stepper motor with LPC2148

Stepper Motor Working





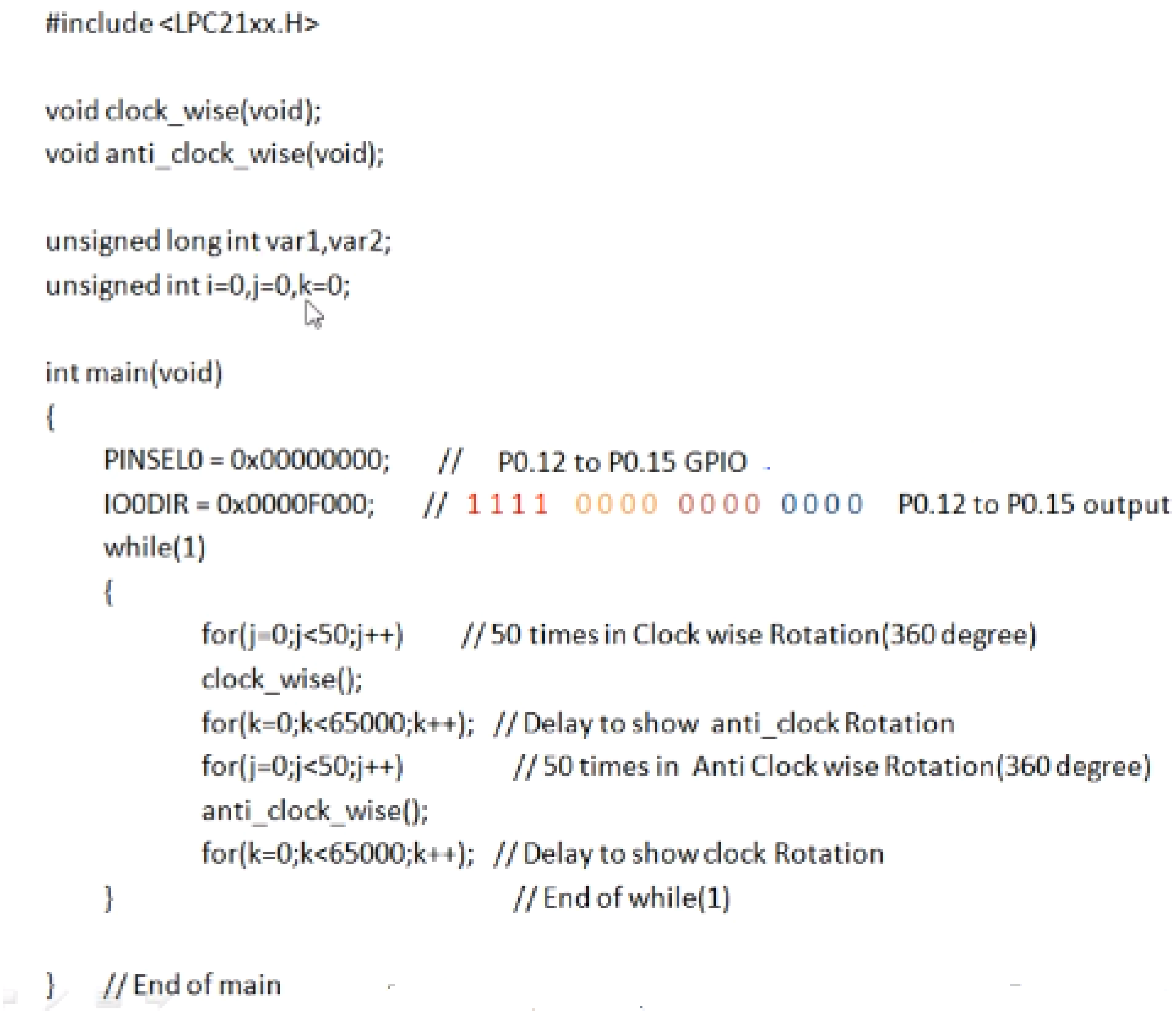
Wave Step			
Coil A	Coil B	Coil C	Coil D
H	L	L	L
L	H	L	L
L	L	H	L
L	L	L	H

Prot Pins	Poles
P0.12	A
P0.13	B
P0.14	C
P0.15	D

Energizing				
	P0.12	P0.13	P0.14	P0.15
Pole A	1	0	0	0
Pole B	0	1	0	0
Pole C	0	0	1	0
Pole D	0	0	0	1

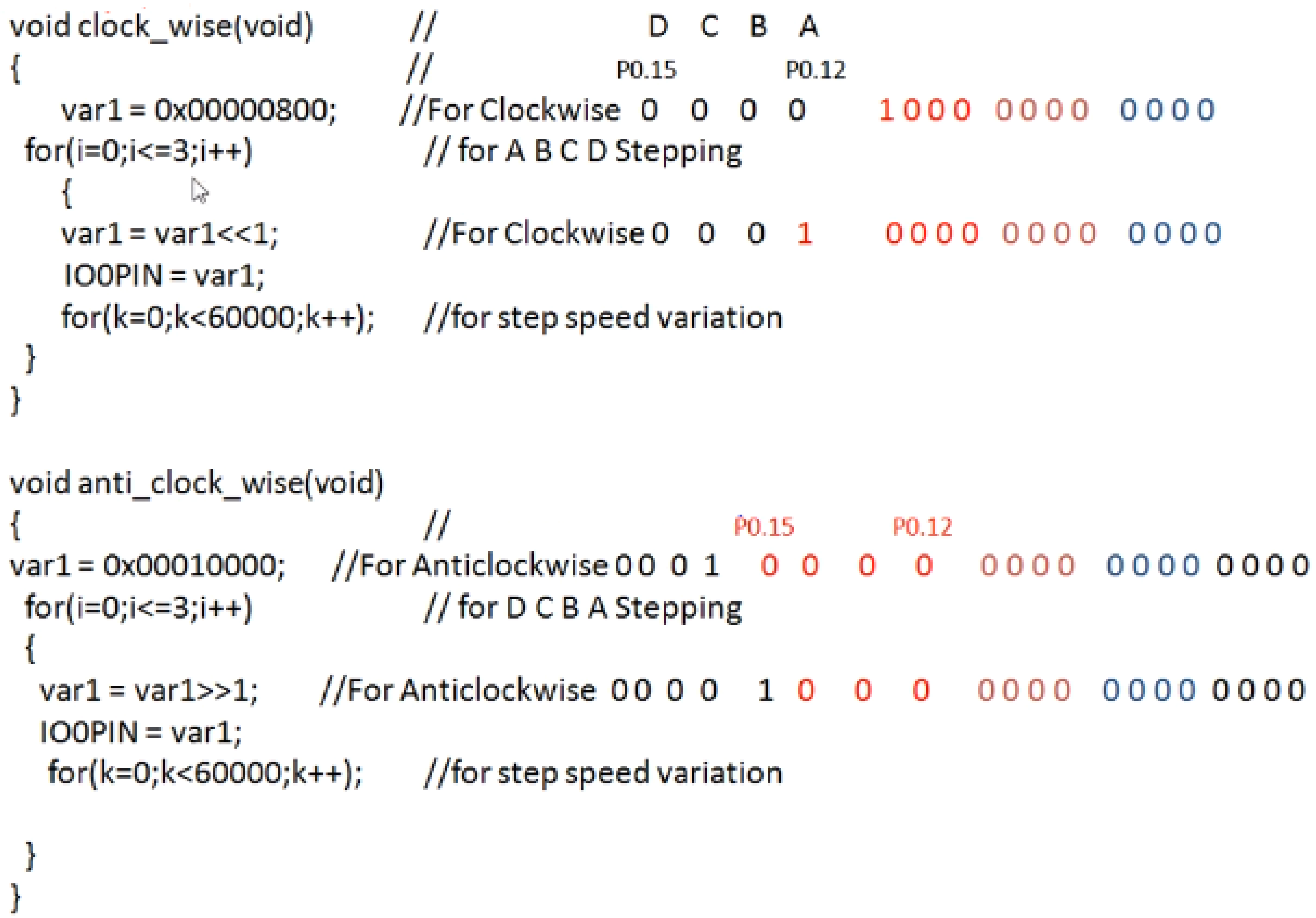
Interfacing Stepper motor with LPC2148

Stepper Motor Embedded C Program



```
#include <LPC21xx.H>

void clock_wise(void);
void anti_clock_wise(void);

unsigned long int var1,var2;
unsigned int i=0,j=0,k=0;

int main(void)
{
    PINSEL0 = 0x00000000;     //   P0.12 to P0.15 GPIO
    IO0DIR = 0x0000F000;     //  1111  0000  0000  0000    P0.12 to P0.15 output
    while(1)
    {
            for(j=0;j<50;j++)       // 50 times in Clock wise Rotation(360 degree)
            clock_wise();
            for(k=0;k<65000;k++);  // Delay to show  anti_clock Rotation
            for(j=0;j<50;j++)           // 50 times in  Anti Clock wise Rotation(360 degree)
            anti_clock_wise();
            for(k=0;k<65000;k++);  // Delay to show clock Rotation
    }                                     // End of while(1)

}    // End of main
```

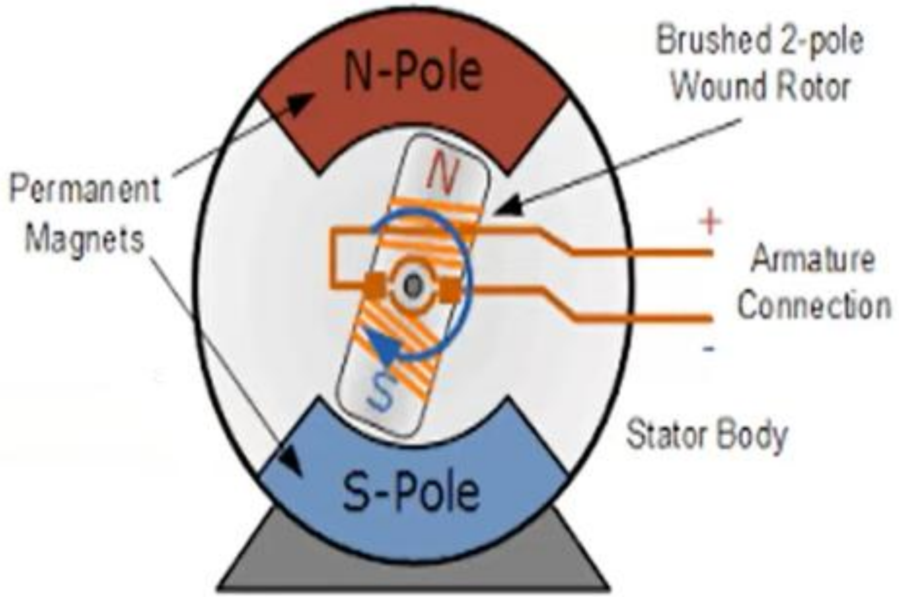
Interfacing Stepper motor with LPC2148



```
void clock_wise(void)        //                    D    C    B    A
{                            //              P0.15          P0.12
    var1 = 0x00000800;      //For Clockwise  0    0    0    0      1000  0000   0000
 for(i=0;i<=3;i++)            // for A B C D Stepping
    {
    var1 = var1<<1;          //For Clockwise 0    0    0    1      0000  0000   0000
    IO0PIN = var1;
    for(k=0;k<60000;k++);    //for step speed variation
 }
}

void anti_clock_wise(void)
{                              //                  P0.15          P0.12
var1 = 0x00010000;     //For Anticlockwise 0 0  0  1   0  0   0   0   0000   0000 0000
 for(i=0;i<=3;i++)             // for D C B A Stepping
 {
   var1 = var1>>1;     //For Anticlockwise  0 0  0  0   1  0   0   0   0000   0000 0000
   IO0PIN = var1;
    for(k=0;k<60000;k++);      //for step speed variation

 }
}
```

DC MOTOR

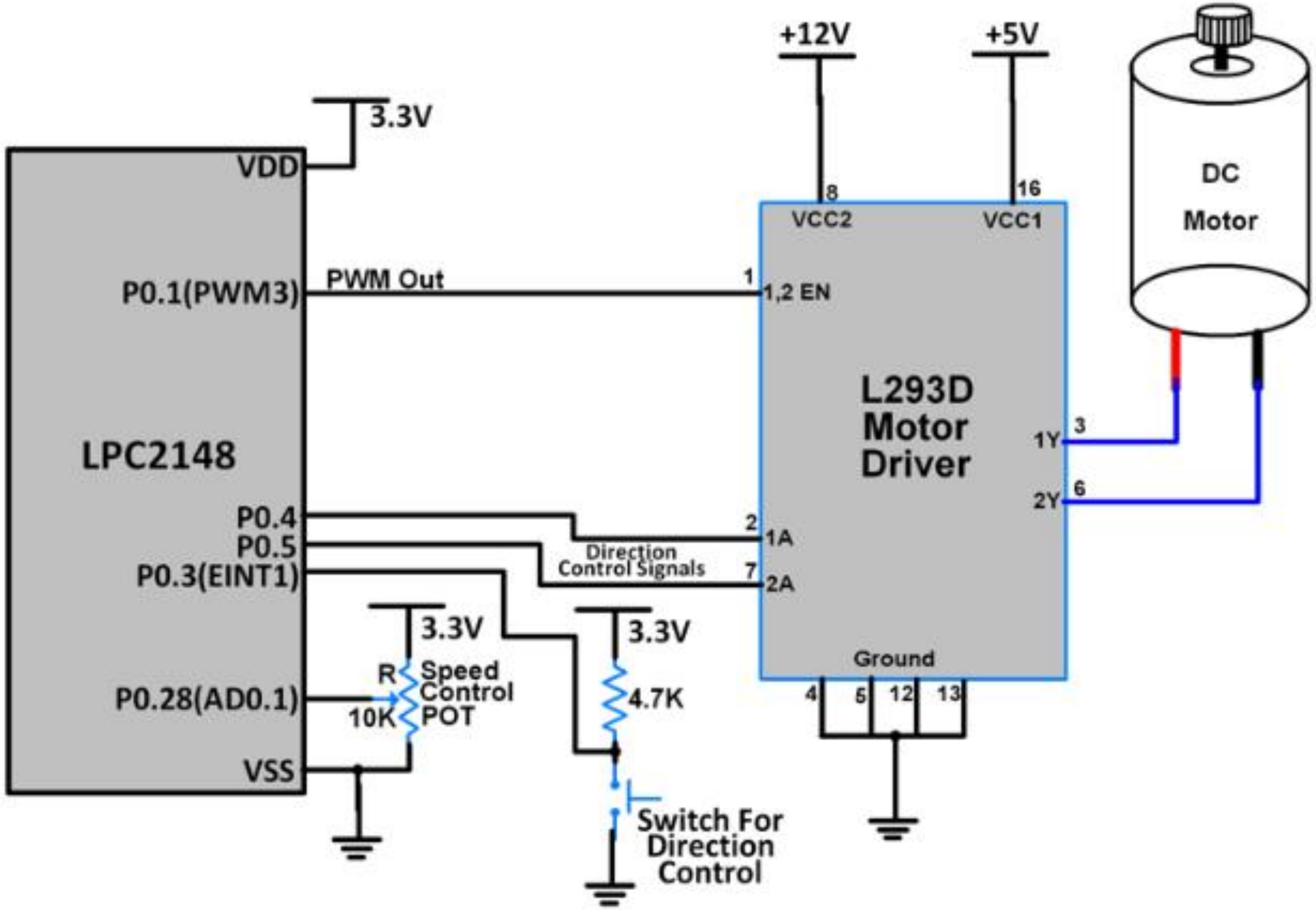
A **DC motor** is any of a class of rotary electrical motors that converts direct current electrical energy into mechanical energy. The most common types rely on the forces produced by magnetic fields.



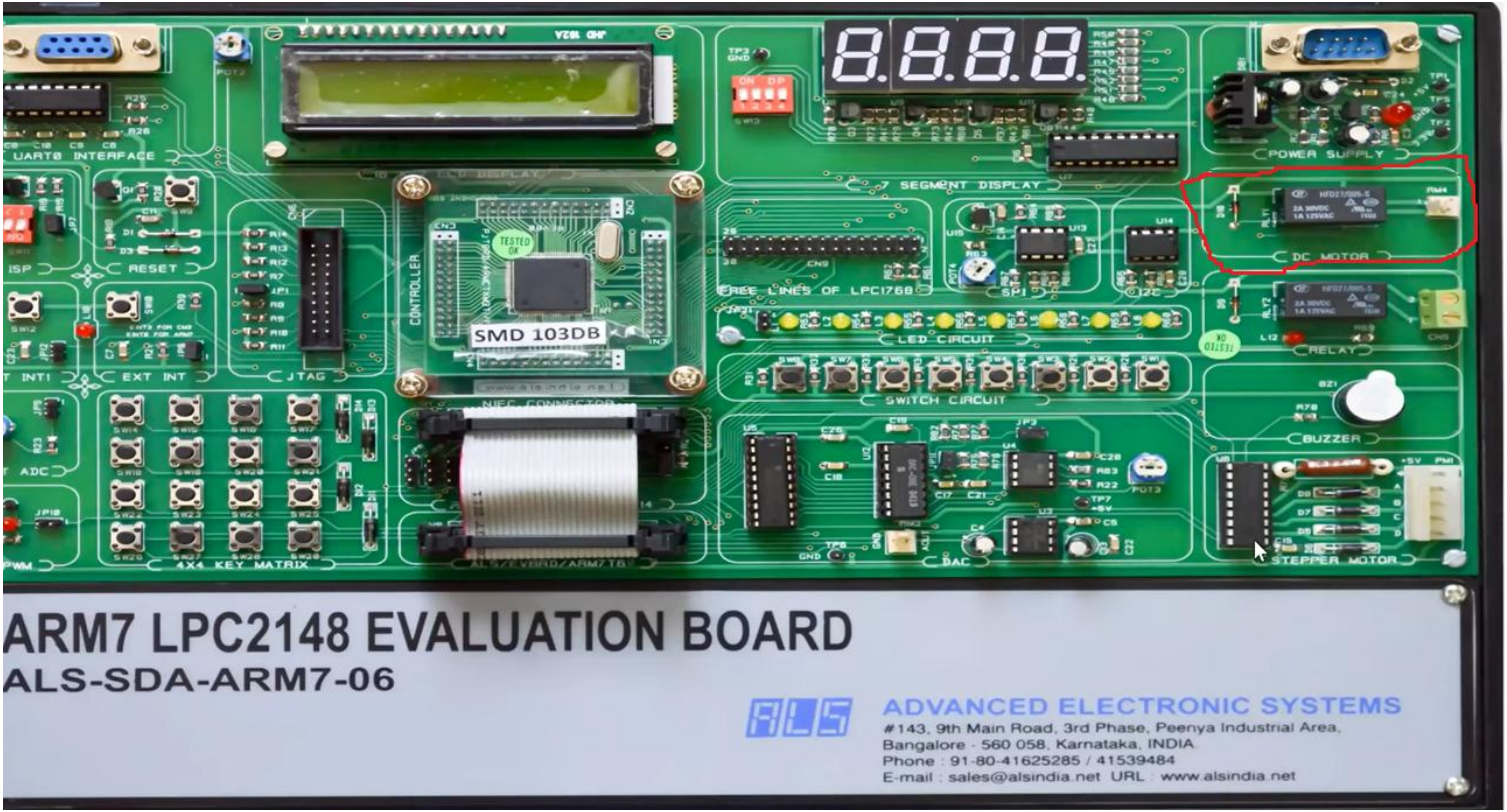
Interfacing DC motor with LPC2148

- DC motor converts electrical energy in the form of Direct Current into mechanical energy.
- In case of a motor, the mechanical energy produced is in the form of rotational movement of the motor shaft.
- The direction of rotation of the shaft of the motor can be reversed by reversing the direction of Direct Current through the motor.
- The motor can be rotated at a certain speed by applying a fixed voltage to it. If the voltage varies, the speed of the motor varies.
- Thus, the DC motor speed can be controlled by applying varying DC voltage; whereas the direction of rotation of the motor can be changed by reversing the direction of current through it.
- For applying varying voltage, we can make use of PWM technique.

Interfacing DC motor with LPC2148



Interfacing DC Motor with LPC2148


UART0 INTERFACE
ISP
RESET
INT1
EXT INT
JTAG
CONTROLLER
TESTED OK
SMD 103DB
LCD DISPLAY
7 SEGMENT DISPLAY
FREE LINES OF LPC1768
SPI
I2C
LED CIRCUIT
SWITCH CIRCUIT
POWER SUPPLY
DC MOTOR
RELAY
BUZZER
ADC
PWM
4X4 KEY MATRIX
DAC
STEPPER MOTOR
ARM7 LPC2148 EVALUATION BOARD
ALS-SDA-ARM7-06
ALS
ADVANCED ELECTRONIC SYSTEMS
#143, 9th Main Road, 3rd Phase, Peenya Industrial Area,
Bangalore - 560 058, Karnataka, INDIA
Phone : 91-80-41625285 / 41539484
E-mail : sales@alsindia.net URL : www.alsindia.net

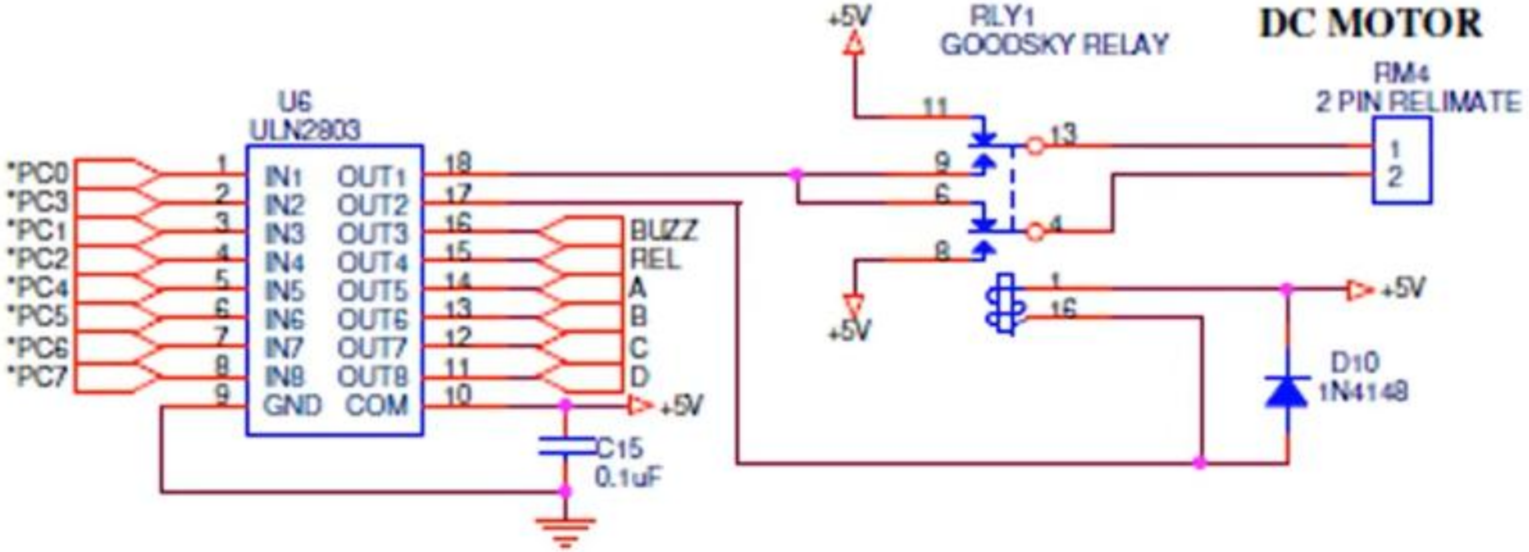
DC Motor Working

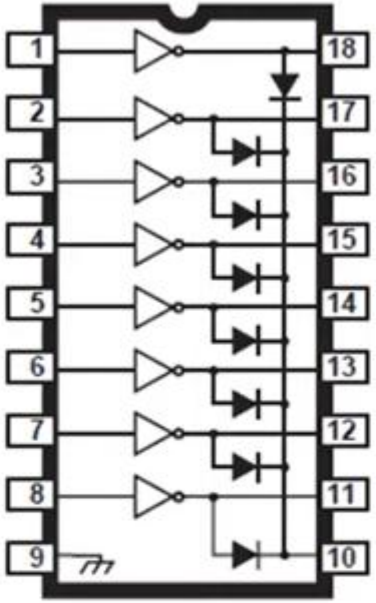
- Connected to port pins P0.11 P0.8

if	1	0	Runs in Clock Dir
If	0	0	Motor off
If	0	1	Runs in Anti Clock Dir

P0.8 ->PC0

P0.11->PC3





Program

```
///////Program to test Working of DC Motor /////////
#include<lpc214x.h>
unsigned int j=0;
int main()
{
PINSEL0=0x00000000;
IO0DIR= 0X00000900; //   1 0 0 1 0 0 0 0 0 0 0 0 P0.11 and P0.8 are made as output pins
```

```
    while(1)
{
//clock_wise
    IO0CLR = 0x00000900;          //stop motor and also turn off relay
    for(j=0;j<10000;j++);         //small delay to allow motor to turn off
    IO0SET = 0X00000800;  //Selecting the P0.11 line for clockwise and turn on motor
for(j=0;j<400000;j++);        //delay
```

```
          //anti_clock_wise
          IO0CLR = 0X00000900;          //stop motor and also turn off relay
     for(j=0;j<10000;j++);              //small delay to allow motor to turn off
     IO0SET = 0X00000100;               //selecting the P0.08 line for Anti clockwise
     for(j=0;j<400000;j++);             //delay
      }                                 //End of while(1)
  }                                     //End of Main
```

Liquid Crystal Display

1. Character LCD
2. Graphical LCD

16 Pin device

Vee –Adjust the contrast of the device

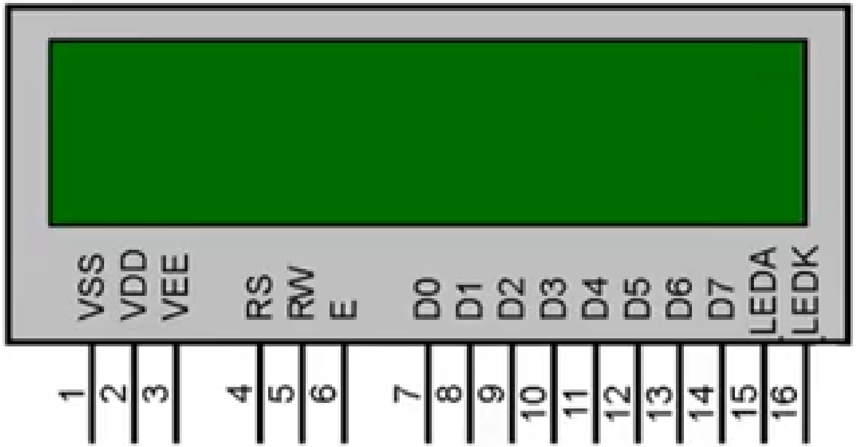
Control lines used to control the LCD

RS- Register select : 1- Data & 0 - Command

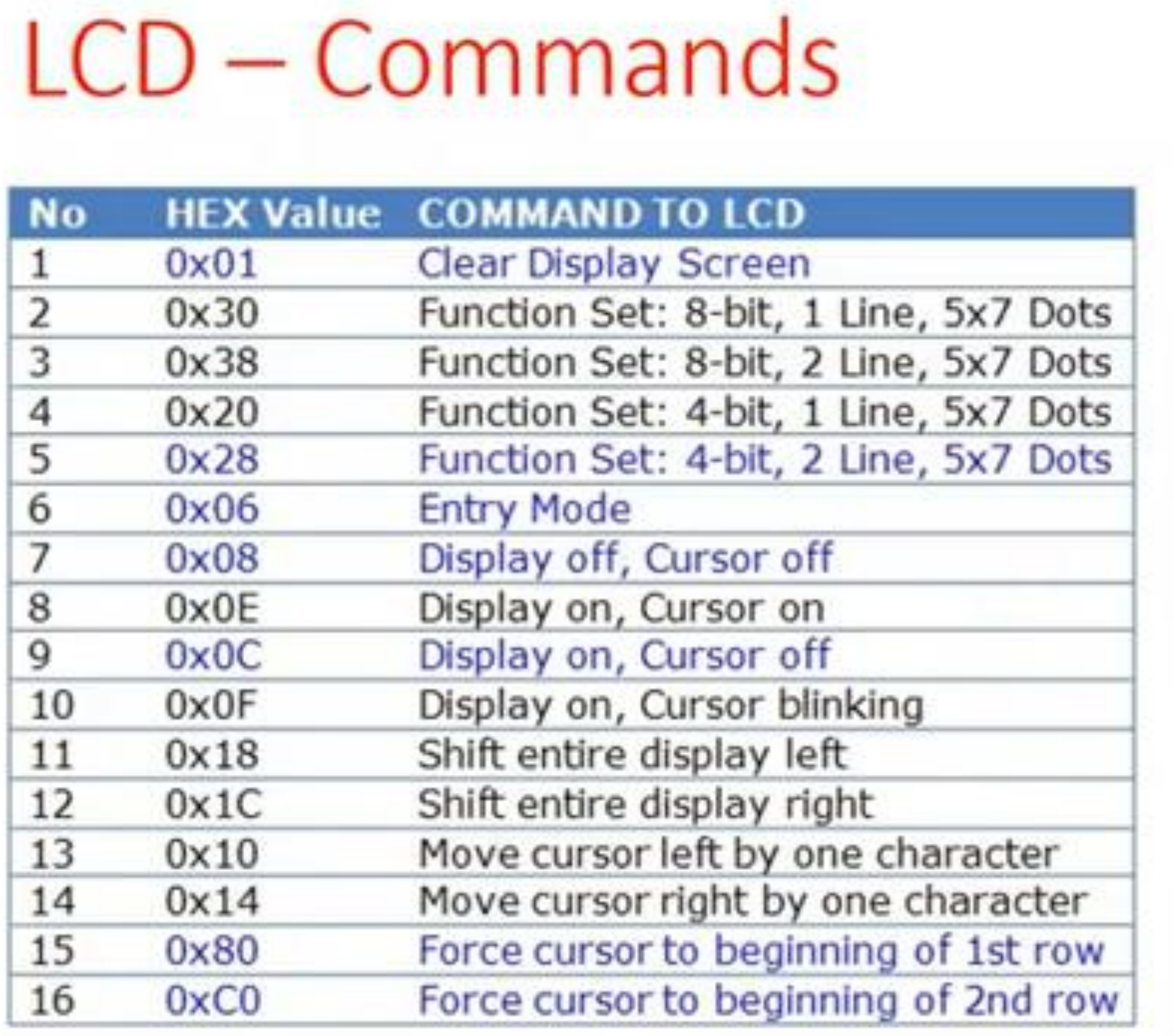
Read write pin : 0- writing data to LCD & 1- Read

Enable : Strobe line : To initiate operations of LCD

Command/ Data transmission – D0 to D7
Backlight + & - : Illuminate LCD



HD 44780 – LCD Controller



LCD – Commands

No	HEX Value	COMMAND TO LCD
1	0x01	Clear Display Screen
2	0x30	Function Set: 8-bit, 1 Line, 5x7 Dots
3	0x38	Function Set: 8-bit, 2 Line, 5x7 Dots
4	0x20	Function Set: 4-bit, 1 Line, 5x7 Dots
5	0x28	Function Set: 4-bit, 2 Line, 5x7 Dots
6	0x06	Entry Mode
7	0x08	Display off, Cursor off
8	0x0E	Display on, Cursor on
9	0x0C	Display on, Cursor off
10	0x0F	Display on, Cursor blinking
11	0x18	Shift entire display left
12	0x1C	Shift entire display right
13	0x10	Move cursor left by one character
14	0x14	Move cursor right by one character
15	0x80	Force cursor to beginning of 1st row
16	0xC0	Force cursor to beginning of 2nd row

0x38 -2 lines (5x7 Matrix)

0x0C – Display ON, Cursor OFF

0x06 – Write data

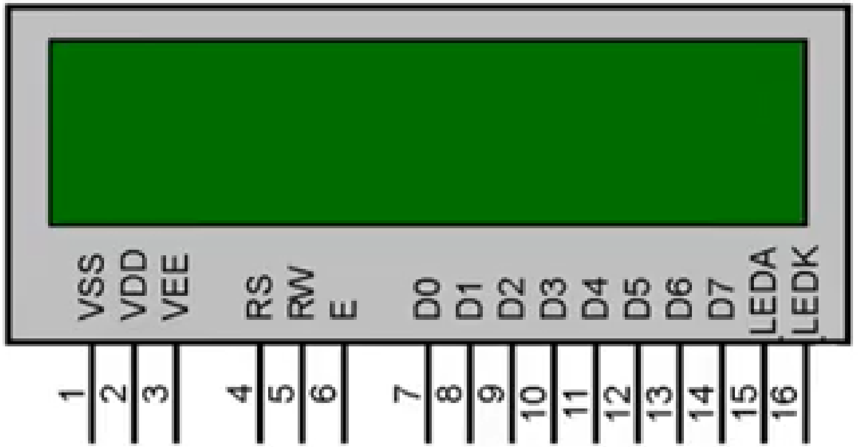
0x01 – Clear LCD display

0x0E - Display ON, Cursor ON

0x80 –Cursor Position

Interfacing 4 X 4 Hex keypad and displaying result on LCD

- How to Initialize
- How to Send Commands
- How to Send Data



Send Command : RS=0 ; RW=0; EN = High After delay LOW

Send Data : RS=1; RW=0; Data to be displayed
EN = High After delay LOW

```
//////  "LCD DISPLAY" To display the predefined data            //////
#include<lpc214x.h>
#include<stdio.h>
//Function prototypes
void lcd_init(void);
void wr_cn(void);
void clr_disp(void);
void delay(unsigned int);
void lcd_com(void);
void wr_dn(void);
void lcd_data(void);

unsigned char temp1;
unsigned long int temp,r=0;
unsigned char *ptr,disp[] = "DSCE ,EIE DEPT.,",disp1[] = "BENGALURU-58";
```

```
int main()
{
            IO0DIR = 0x000000FC;                    //configure o/p lines for lcd

            IO0PIN = 0X00000000;

            delay(3200);                                    //delay

            lcd_init();             //lcd intialisation

            delay(3200);                                    //delay

   clr_disp();                                              //clear display

            delay(3200);         //delay


//........LCD DISPLAY TEST.........//

            temp1 = 0x80;                                   //Display starting address      of

first line 1 th pos

            lcd_com();

            ptr = disp;
while(*ptr!='\0')
  {
                    temp1 = *ptr;
          lcd_data();
                    ptr ++;
  }

          temp1 = 0xC0;                             // Display starting address of second line 4 th
pos
          lcd_com();

          ptr = disp1;
          while(*ptr!='\0')
  {
          temp1 = *ptr;
          lcd_data();
                    ptr ++;
  }
          while(1);
}  //end of main()
```

```
// lcd initialisation routine.
void lcd_init(void)
{
        temp = 0x30;
        wr_cn();
        delay(3200);

        temp = 0x30;
        wr_cn();
        delay(3200);

        temp = 0x30;
        wr_cn();
        delay(3200);

        temp = 0x20;        // change to 4 bit mode from default 8 bit mode
        wr_cn();
        delay(3200);
// load command for lcd function setting with lcd in 4 bit mode,
// 2 line and 5x7 matrix display

        temp = 0x28;
        lcd_com();
        delay(3200);

// load a command for display on, cursor on and blinking off
        temp1 = 0x0C;
        lcd_com();
        delay(800);

// command for cursor increment after data dump
        temp1 = 0x06;
        lcd_com();
        delay(800);

        temp1 = 0x80;  // set the cursor to beginning of line 1
        lcd_com();
        delay(800);
}
```

```
void lcd_com(void)
{
            temp = temp1 & 0xf0;
   wr_cn();
   temp = temp1 & 0x0f;
   temp = temp << 4;
   wr_cn();
   delay(500);
}

// command nibble o/p routine
void wr_cn(void)            //write command reg
{
            IO0CLR  = 0x000000FC;                   // clear the port lines.
            IO0SET     = temp;                                  // Assign the value to the PORT
lines
            IO0CLR  = 0x00000004;                   // clear bit  RS = 0
            IO0SET     = 0x00000008;      // E=1
            delay(10);
            IO0CLR  = 0x00000008;
}
```

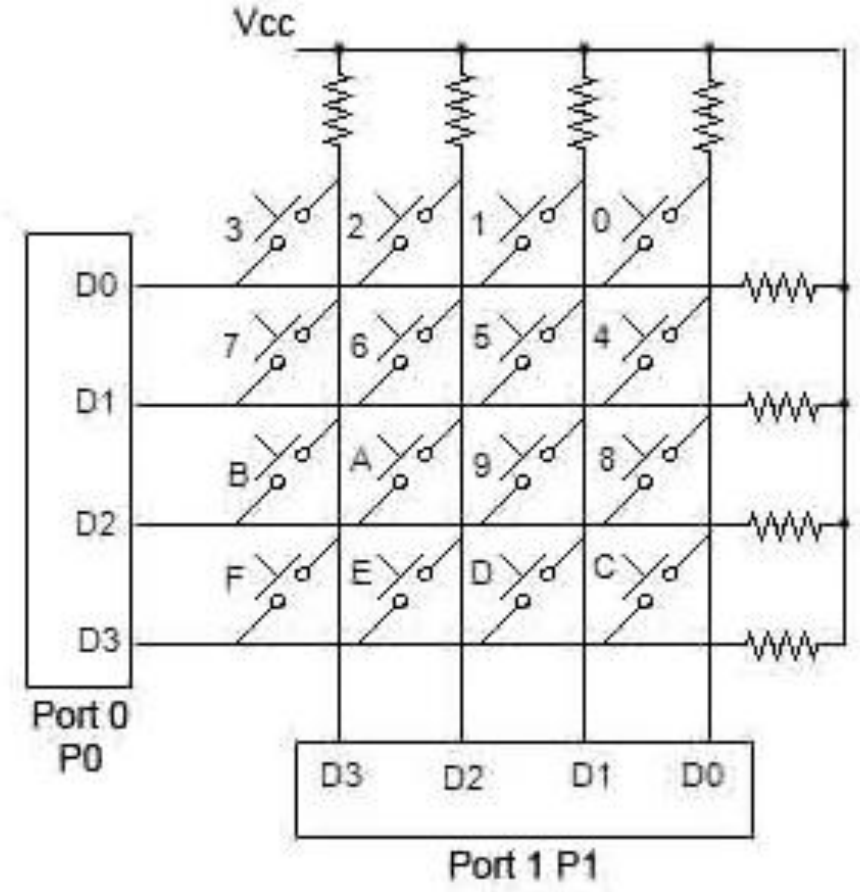
```
// data nibble o/p routine
void wr_dn(void)                        ////write data reg
{
        IO0CLR = 0x000000FC;            // clear the port lines.
        IO0SET = temp;                          // Assign the value to the PORT lines
        IO0SET = 0x00000004;            // set bit  RS = 1
        IO0SET = 0x00000008;            // E=1
        delay(10);
        IO0CLR = 0x00000008;
}

// data o/p routine which also outputs high nibble first
// and lower nibble next
void lcd_data(void)
{
        temp = temp1 & 0xf0;
   temp = temp ;//<< 6;
   wr_dn();
   temp= temp1 & 0x0f;
   temp= temp << 4;
   wr_dn();
   delay(100);
```

```
void clr_disp(void)
{
// command to clear lcd display
   temp1 = 0x01;
   lcd_com();
   delay(500);

}

void delay(unsigned int r1)
{
        for(r=0;r<r1;r++);
}
```

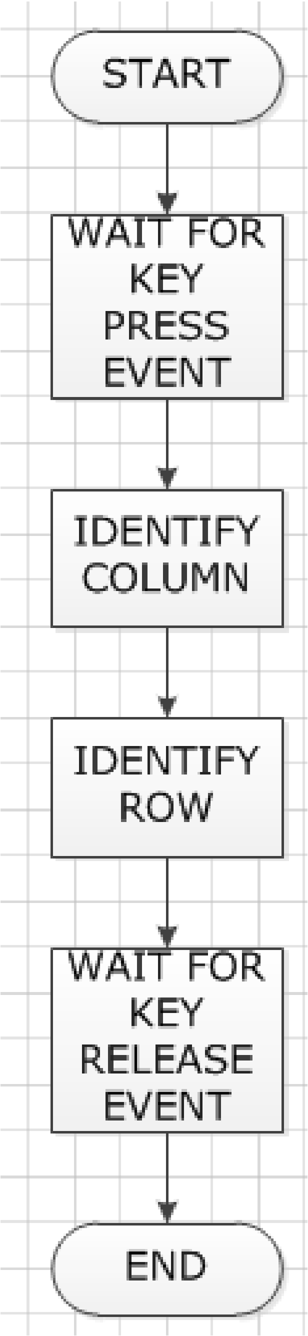
Interfacing 4 X 4 Hex keypad

Matrix keypad is one of the widely used input devices. Some of the application includes Mobile keypad, Telephone dial pad, calculator, ATM etc. Keypad provides an easy way to allow user to provide input to any system. In this article, we will explain **how to interface 4x4 matrix keypad with LPC2148**. The pressed key will be displayed on LCD.

In matrix keypad, keys are connected in rows and columns. When any switch is pressed, rows and columns come into contact which is detected by controller to identify which key has been pressed.



```
/*Program to demonstrate keyboard operation
Takes a key from key board and displays it on LCD screen*/
#include<lpc21xx.h>
#include<stdio.h>
/******* FUNCTION PROTOTYPE*******/
void lcd_init(void);
void clr_disp(void);
void lcd_com(void);
void lcd_data(void);
void wr_cn(void);
void wr_dn(void);
void scan(void);
void get_key(void);
void display(void);
void delay(unsigned int);
void init_port(void);
```



```
unsigned long int scan_code[16]= {0x00EE0000,0x00ED0000,0x00EB0000,0x00E70000,
                                  0x00DE0000,0x00DD0000,0x00DB0000,0x00D70000,
                                  0x00BE0000,0x00BD0000,0x00BB0000,0x00B70000,
                                  0x007E0000,0x007D0000,0x007B0000,0x00770000};


unsigned char ASCII_CODE[16]= {'0','1','2','3',
                               '4','5','6','7',
                               '8','9','A','B',
                               'C','D','E','F'};

   unsigned char  row,col;

   unsigned char temp,flag,i,result,temp1;

   unsigned int r,r1;

   unsigned long int var,var1,var2,res1,temp2,temp3,temp4;

   unsigned char *ptr,disp[] = "4X4 KEYPAD";

   unsigned char disp0[] = "KEYPAD TESTING";

   unsigned char disp1[] = "KEY = ";
```

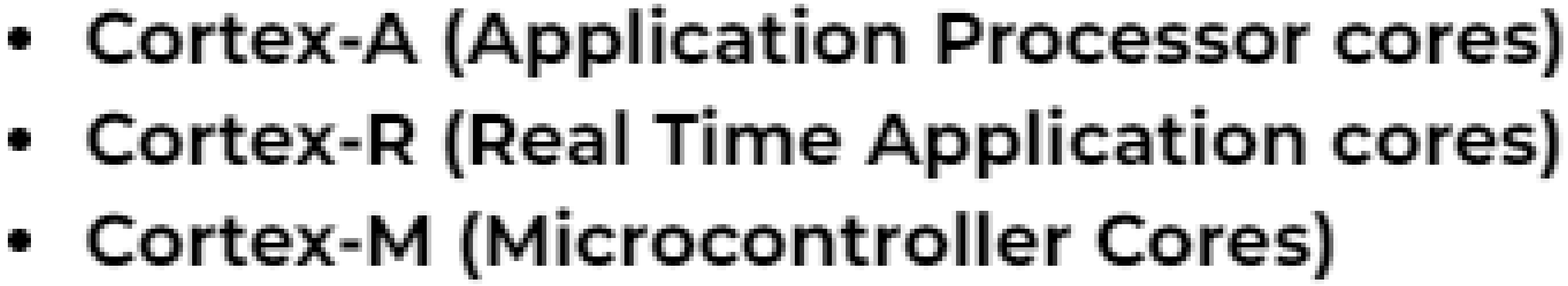
```
int main()
{
  //   __ARMLIB_enableIRQ();
        init_port();         //port intialisation
        delay(3200);                              //delay
        lcd_init();          //lcd intialisation
        delay(3200);                              //delay
  clr_disp();                                     //clear display
        delay(500);           //delay

        //........LCD DISPLAY TEST........./ /
        ptr = disp;
        temp1 = 0x81;                             // Display starting address
        lcd_com();
        delay(800);
```

```
while(*ptr!='\0')
  {
         temp1 = *ptr;
    lcd_data();
           ptr ++;
  }

         //........KEYPAD Working........//
         while(1)
         {
         get_key();
    display();
         }

}  //end of main()

void get_key(void)              //get the key from the keyboard
{
         unsigned int  i;
         flag = 0x00;
  IO1PIN=0x000f0000;
```

```
while(1)
            {
            for(row=0X00;row<0X04;row++)           //Writing one for col's
      {
            if( row == 0X00)
        {
            temp3=0x00700000;
                           }
        else if(row == 0X01)
        {
            temp3=0x00B00000;
                           }
                           else if(row == 0X02)
                           {
            temp3=0x00D00000;
                           }
        else if(row == 0X03)
                           {
            temp3=0x00E00000;
                           }
```

```
var1 = temp3;
            IO1PIN = var1;                  // each time var1 value
is put to port1
            IO1CLR =~var1;                  // Once again
Conforming (clearing all other bits)
            scan();
            delay(100);                                 //delay
            if(flag == 0xff)
            break;
            } // end of for
                        if(flag == 0xff)
                        break;
            }  // end of while
```

CORTEX Series Processors



- Cortex-A (Application Processor cores)
- Cortex-R (Real Time Application cores)
- Cortex-M (Microcontroller Cores)

ARM Architecture road map

4T

ARM7TDMI
ARM922T

Thumb instruction set

5TE

ARM926EJ-S
ARM946E-S
ARM966E-S

Improved ARM/Thumb Interworking

DSP instructions

Extensions:

Jazelle (5TEJ)

6

ARM1136JF-S
ARM1176JZF-S
ARM11 MPCore

SIMD Instructions

Unaligned data support

Extensions:

Thumb-2 (6T2)

TrustZone (6Z)

Multicore (6K)

7

Cortex-A8/R4/M3/M1

Thumb-2

Extensions:

v7A (applications) – NEON

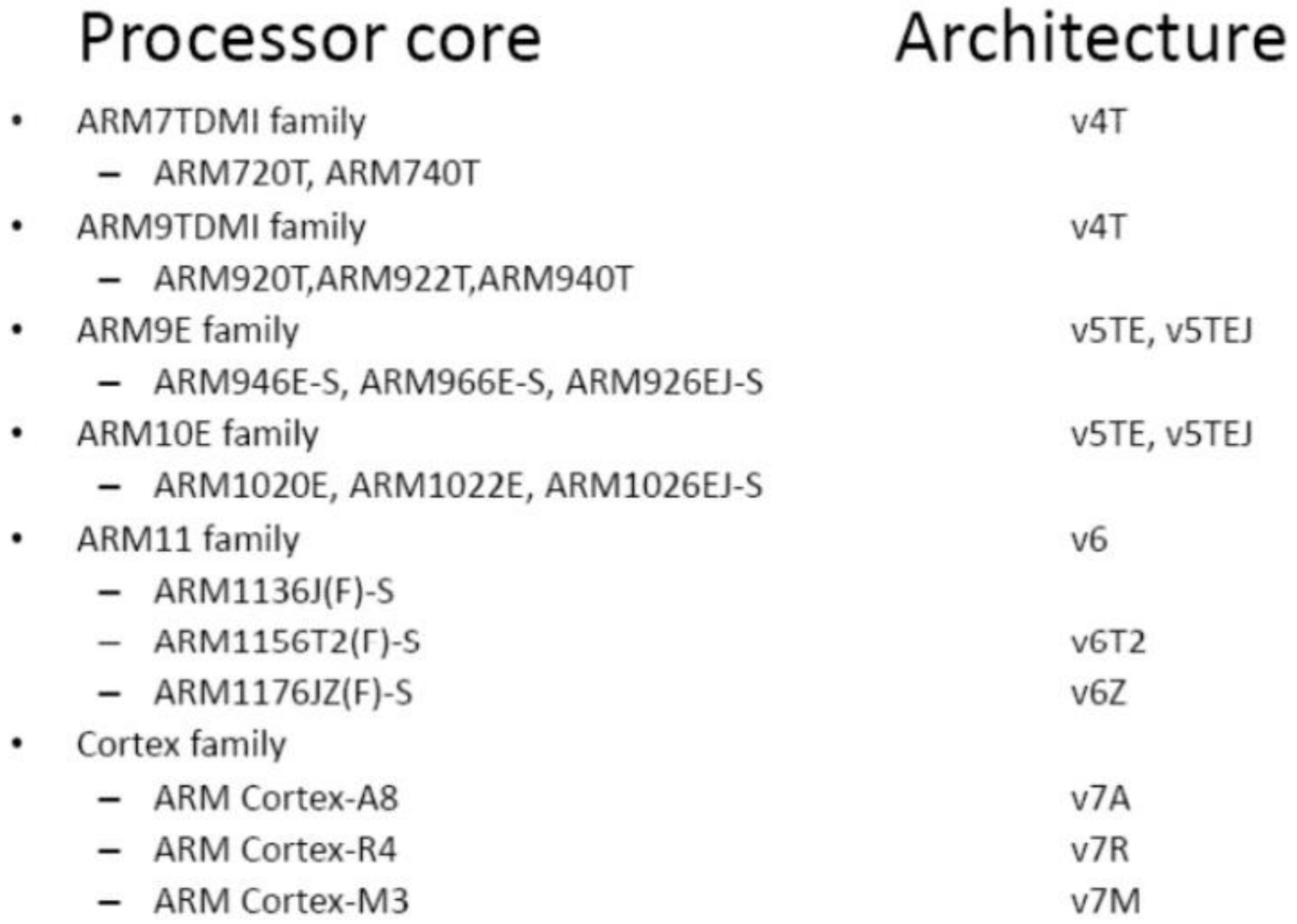
v7R (real time) – HW Divide

V7M (microcontroller) – HW Divide and Thumb-2 only

WHY CORTEX?

- *Greater performance efficiency*: allowing more work to be done without increasing the frequency or power requirements.
- *Low power consumption*: enabling longer battery life, especially critical in portable products including wireless networking applications.
- E*nhanced determinism*: guaranteeing that critical tasks and interrupts are serviced as quickly as possible and in a known number of cycles.
- *Ease of use*: providing easier programmability and debugging for the growing number of 8-bit and 16-bit users migrating to 32 bits.
- *Lower cost solutions*: reducing 32-bit-based system costs close to those of legacy 8-bit and 16-bit devices and enabling low-end, 32-bit microcontrollers to be priced at less than US$1 for the first time.

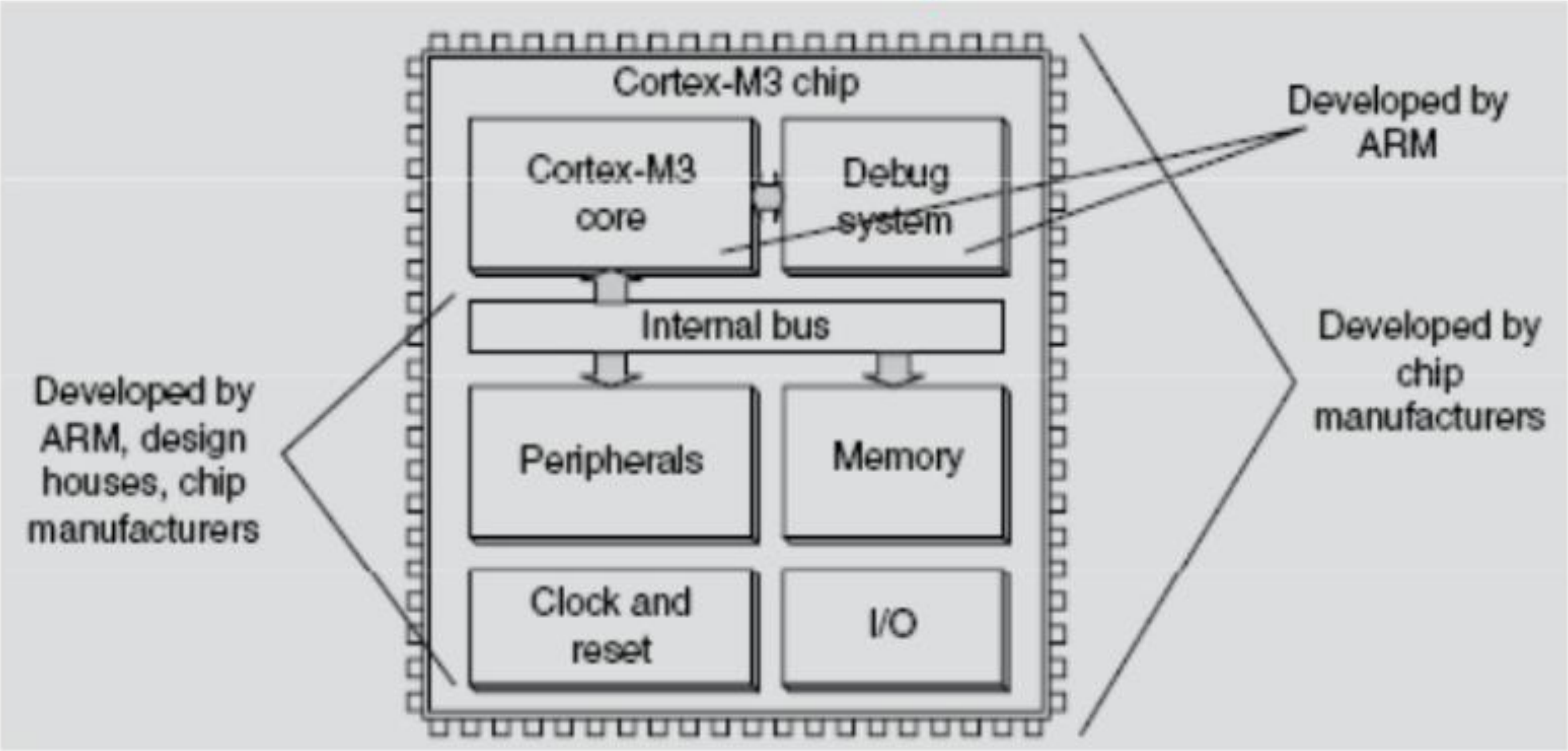
- *Wide choice of development tools*: from low-cost or free compilers to full-featured development suites from many development tool vendors



Processor core	Architecture
• ARM7TDMI family	v4T
– ARM720T, ARM740T	
• ARM9TDMI family	v4T
– ARM920T,ARM922T,ARM940T	
• ARM9E family	v5TE, v5TEJ
– ARM946E-S, ARM966E-S, ARM926EJ-S	
• ARM10E family	v5TE, v5TEJ
– ARM1020E, ARM1022E, ARM1026EJ-S	
• ARM11 family	v6
– ARM1136J(F)-S	
– ARM1156T2(Γ)-S	v6T2
– ARM1176JZ(F)-S	v6Z
• Cortex family	
– ARM Cortex-A8	v7A
– ARM Cortex-R4	v7R
– ARM Cortex-M3	v7M

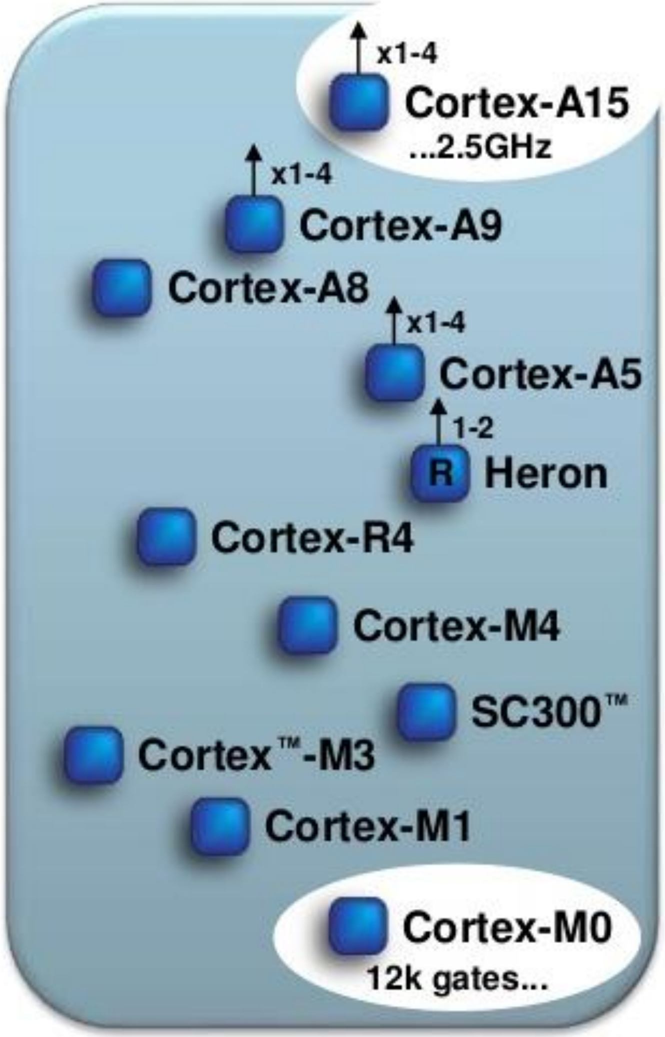
Processor vs MCU

Focus today



ARM Cortex Processors (v7)

- ARM Cortex-**A** family (v7-A):
  - Applications processors for full OS and 3rd party applications
- ARM Cortex-**R** family (v7-R):
  - Embedded processors for real-time signal processing, control applications
- ARM Cortex-**M** family (v7-M):
  - Microcontroller-oriented processors for MCU and SoC applications



The Cortex-A series applications start from handsets, smartphones, computers to high-end broadcasting and networking equipment, the cores behind this magic all are started with a series of processors A5, A7, A8, A9, A12, A15 and later ARM has renamed Cortex-A12 to Cortex-A17 based on the ARM v7 architecture. When used with a perfect combination these cores provide peak-performance and high power efficiency.

The cortex-R7 comes with an increased performance clock ticking at the frequency of 1GHz. This series has come up with multiple options for the multi-core configurations as well as lock-step. It also has a fully integrated generic interrupt controller (GIC) supporting complex priority-based interrupt handling. Even though there are endless possibilities in this domain, this series of processors are not suitable for the rich operating systems such as android and Linux.

The Cortex M is the famous microcontroller series which is ruling the world with its flexibility in mending with the designer’s applications.

Cortex M is built on the ARMv7 architecture and the smallest microcontroller Cortex M0+ is built on the ARMv6 architecture.

The first Cortex M was released in 2004 namely Cortex M3 and the latest one in the series is Cortex M35P(2018) in which P stands for Physically secure

Cortex-M based devices are growing exponentially

+45 billion Cortex-M based chips shipped*

*Arm (as of Sept 2018)

CORTEX A stands for application which help in performance intensive applications such as Android, Linux and many other applications related to handsets, tablets, laptops and desktops.

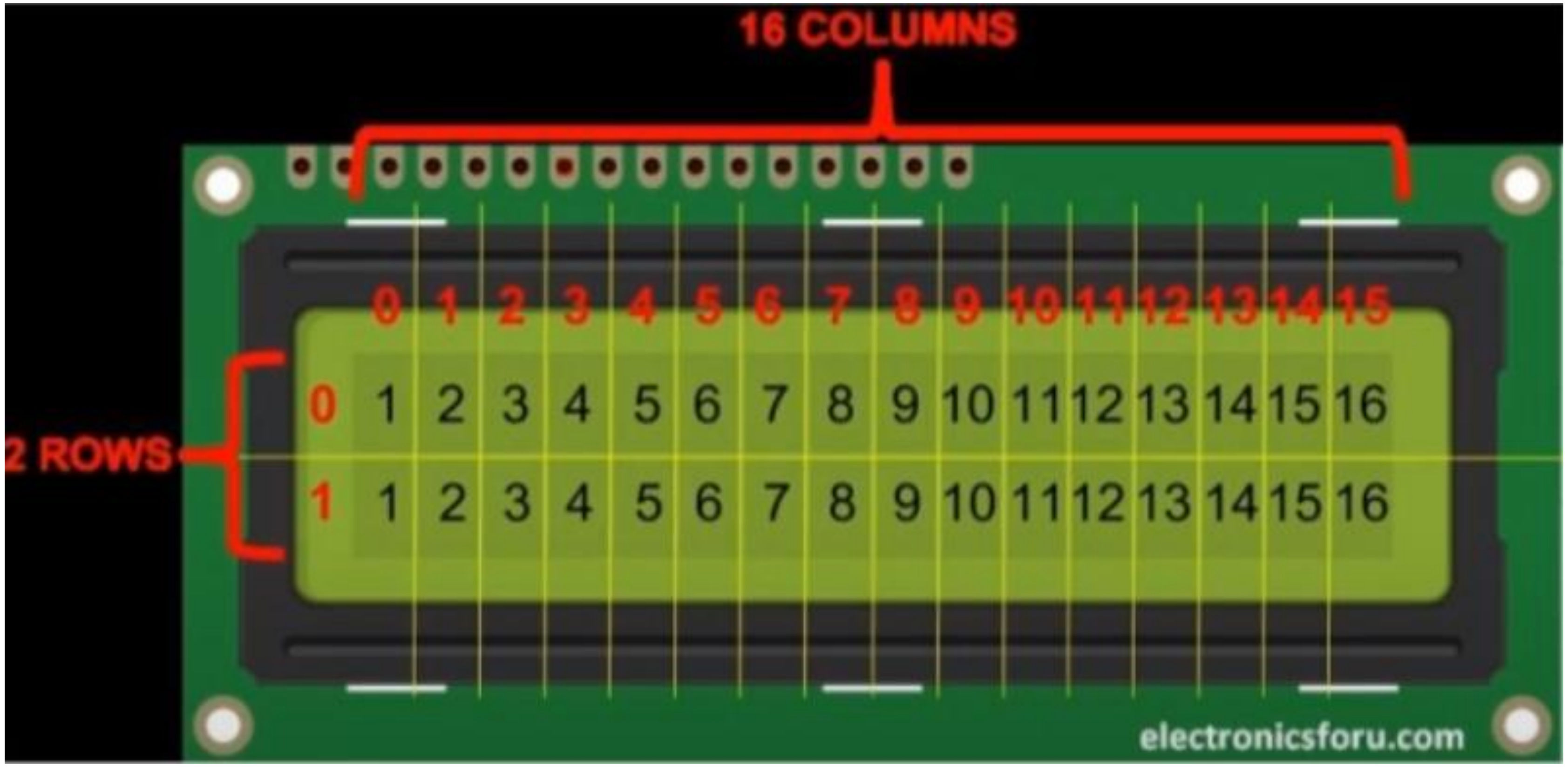
CORTEX R stands for real time application which is used in the safety critical applications and where we need real time responses of the system such as avionics, automotive, medical, defense and server side technologies where data related applications are executed.

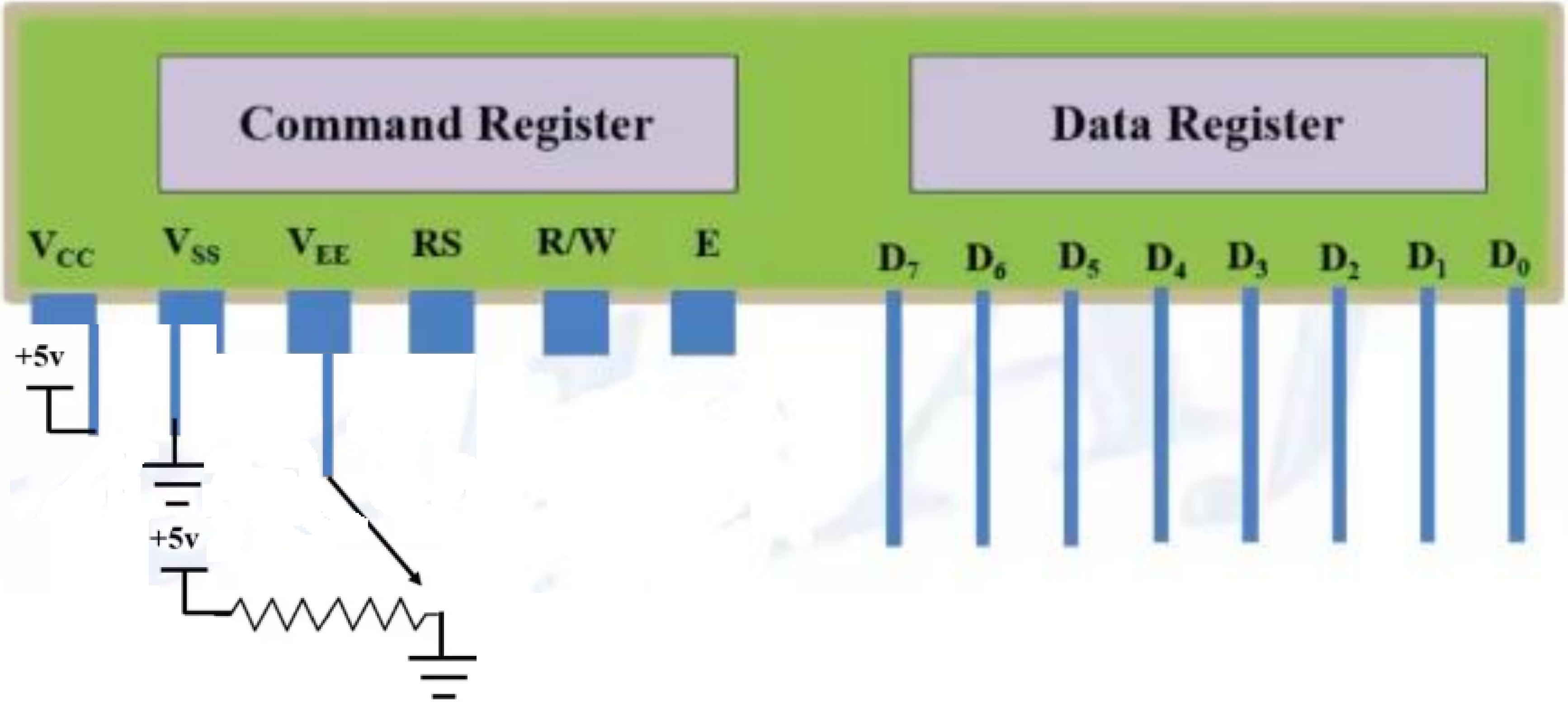
CORTEX M stands for microcontroller which is used in most of our daily life applications also starting from the automation to DSP applications , sensors , smart displays , IoT applications and many more.
The CORTEX M series is an ocean of possibilities with a large number of probabilities and configuration.

LCD DISPLAY

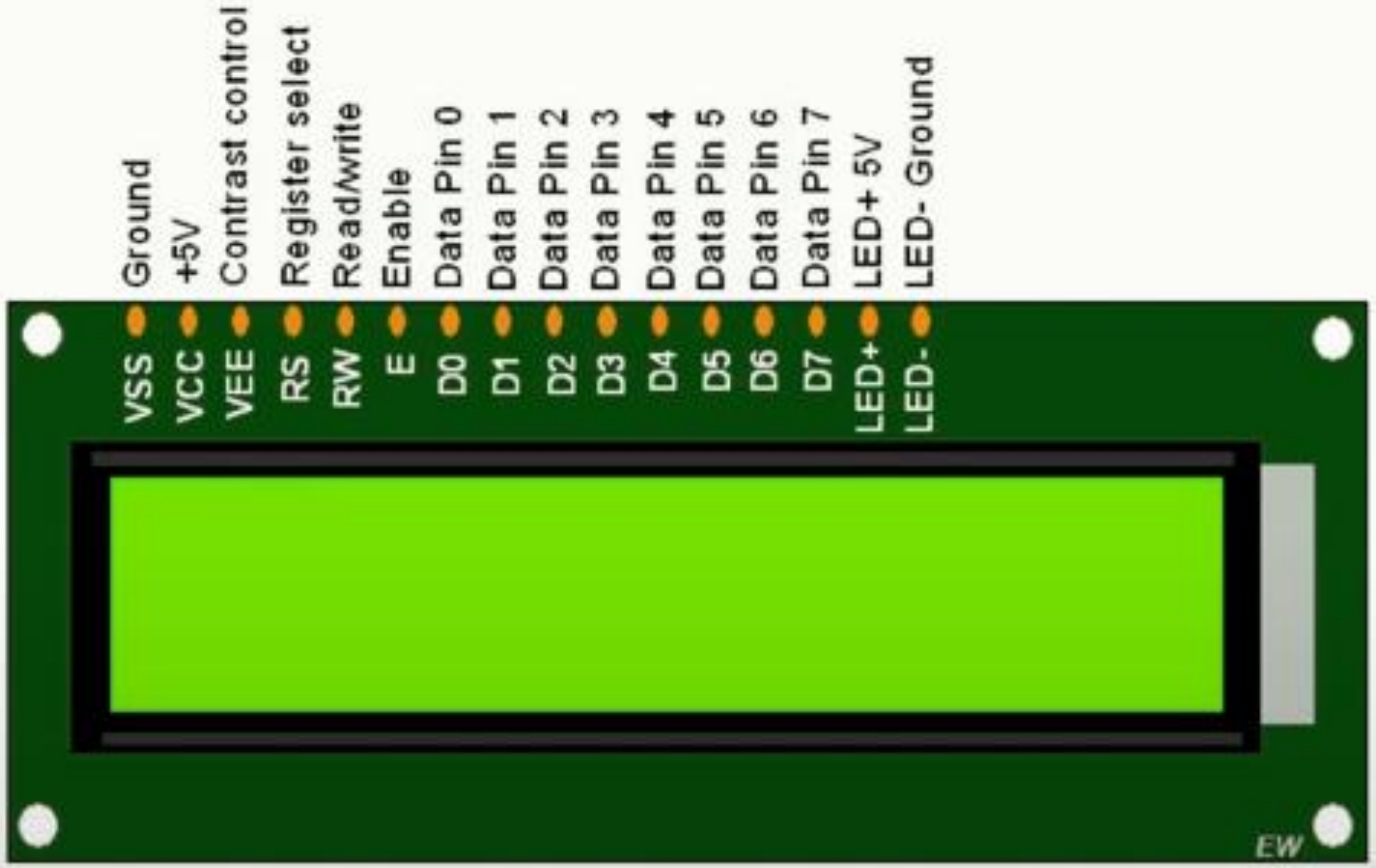
- □ Liquid Crystal Display (LCD) :
- □ 16×2 character LCD display is very basic module
- □ It can display 2 lines of 16 characters.
- □ Each character is displayed using 5×7 or 5×10 pixel matrix.

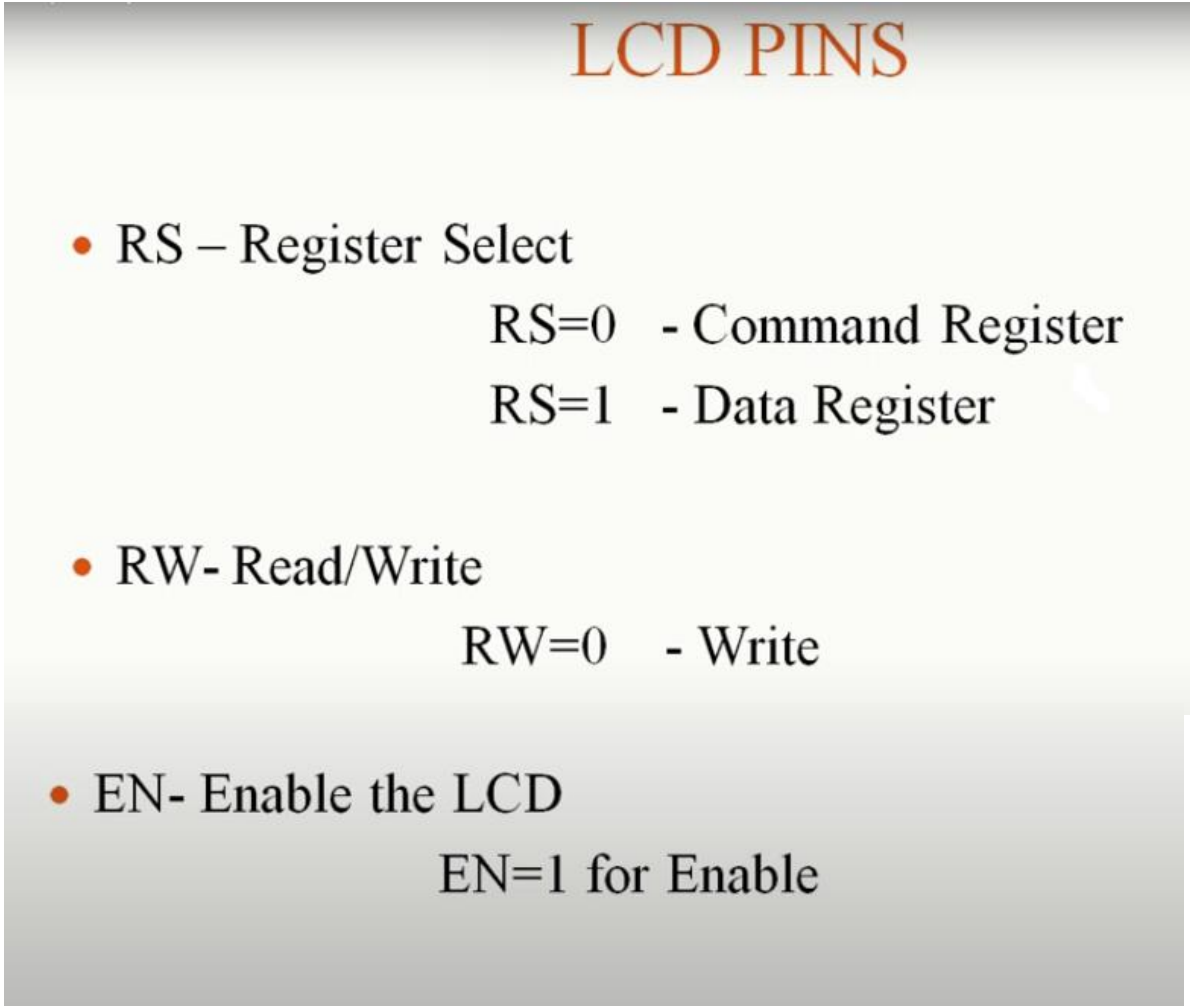
LCD Interfacing with LPC2148 Processor




Command Register
Data Register
V_{CC}
V_{SS}
V_{EE}
RS
R/W
E
D_7
D_6
D_5
D_4
D_3
D_2
D_1
D_0
+5v
+5v

2 line and 16 Character LCD





LCD PINS

- RS – Register Select

  RS=0 - Command Register

  RS=1 - Data Register

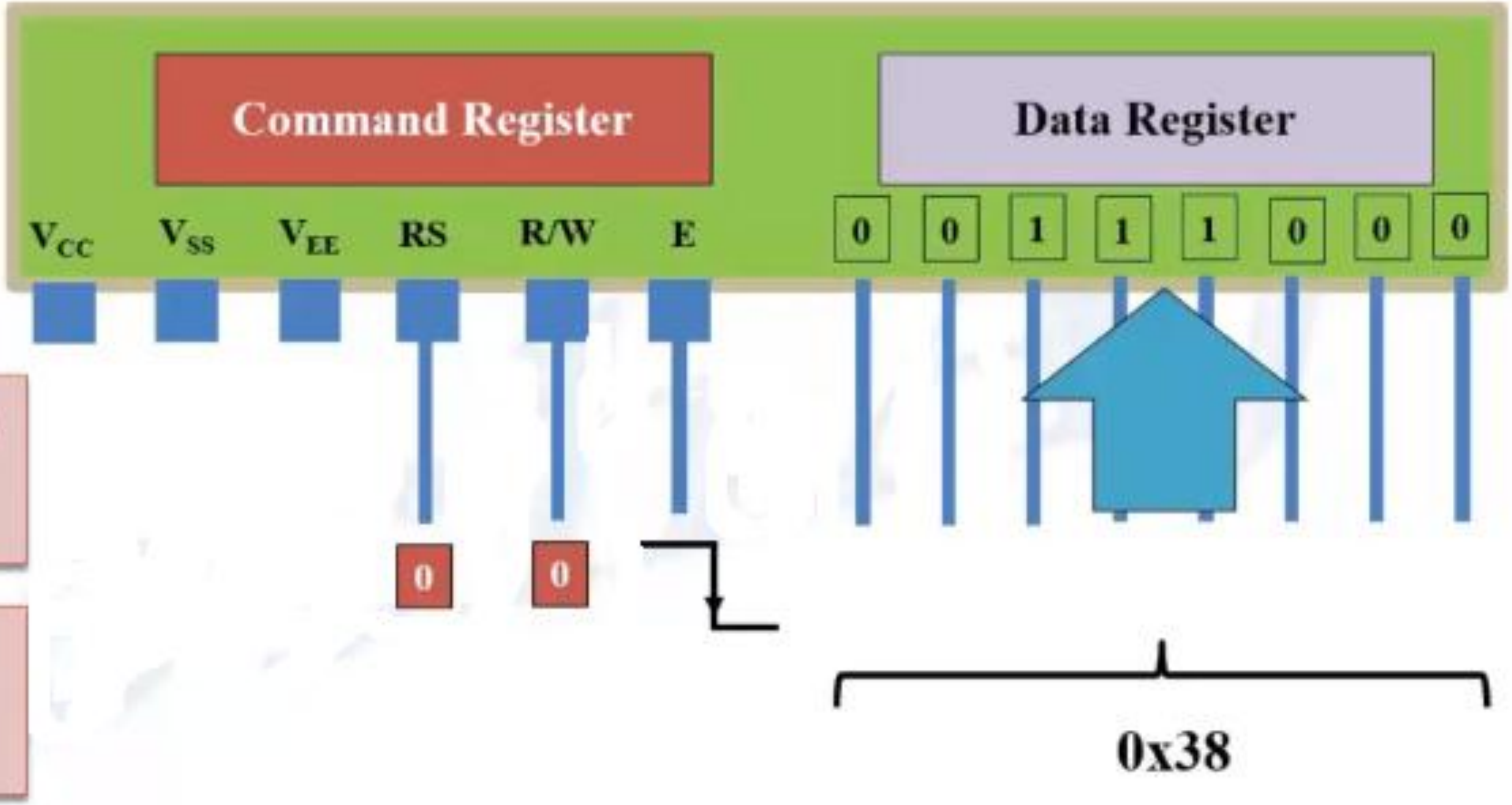
- RW- Read/Write

  RW=0 - Write

- EN- Enable the LCD

  EN=1 for Enable

Send Command to LCD

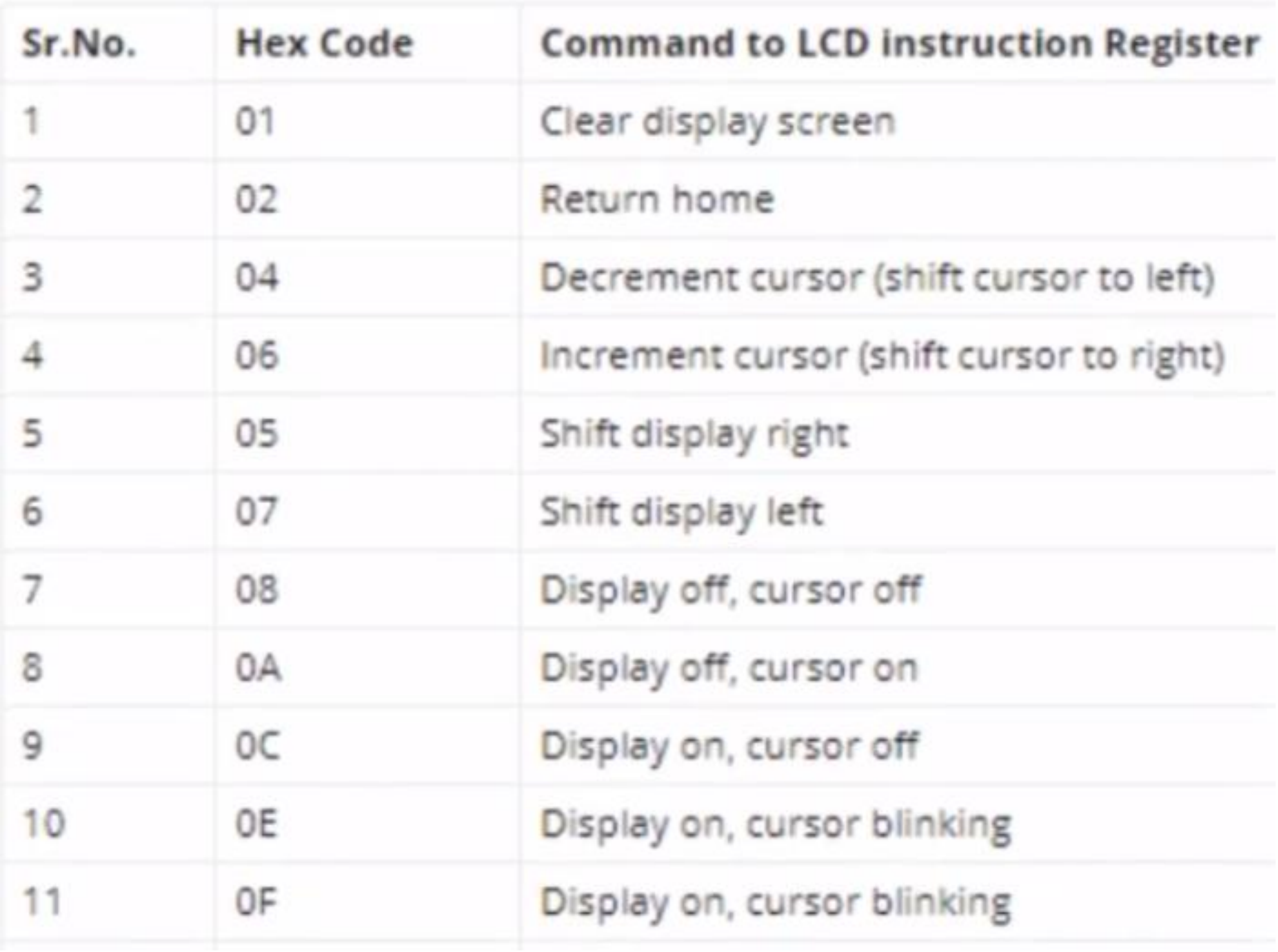


- RS=0 Command
- RS=1 Data

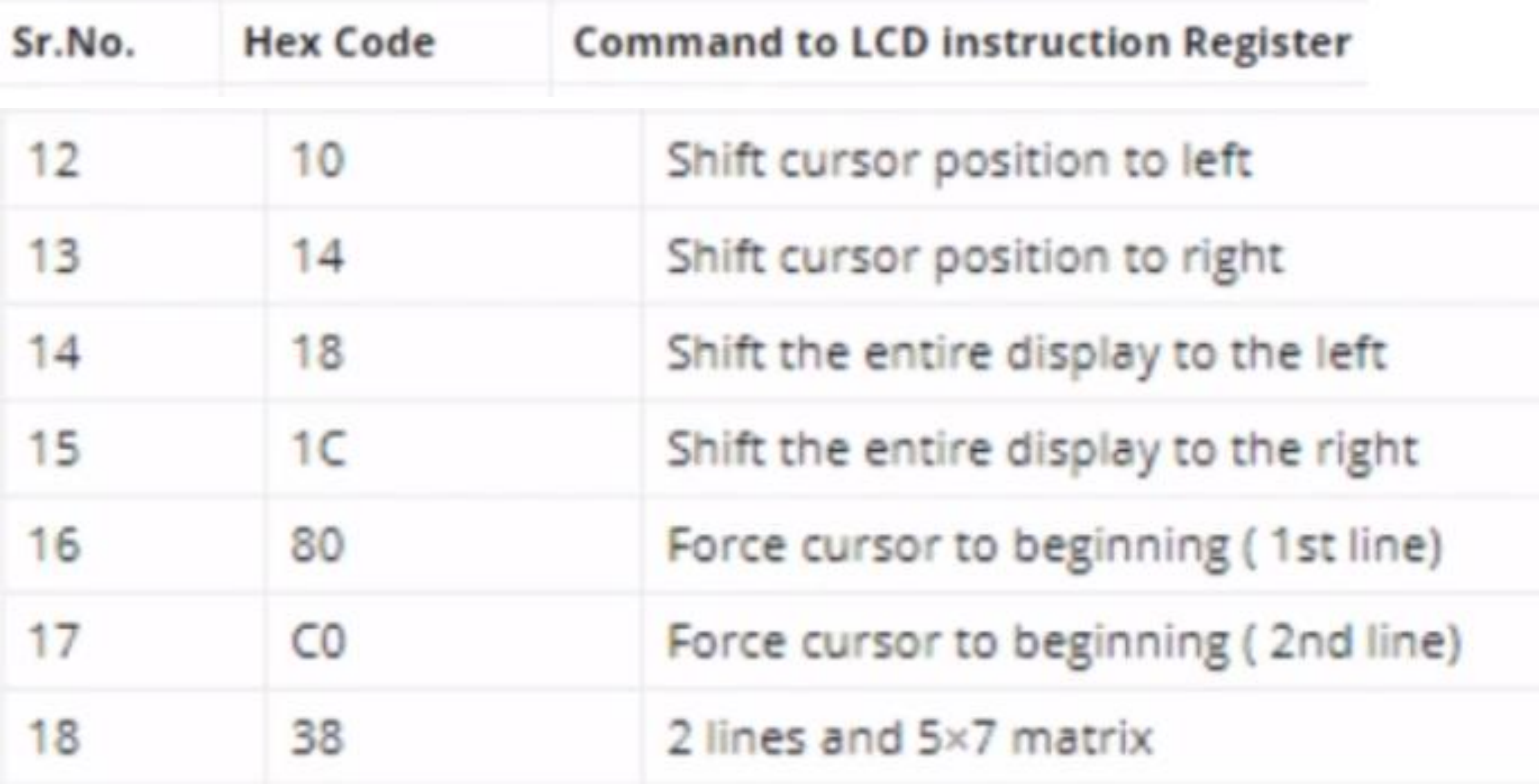
- R/W=0 Write
- R/W=1 Read

- E = high to low

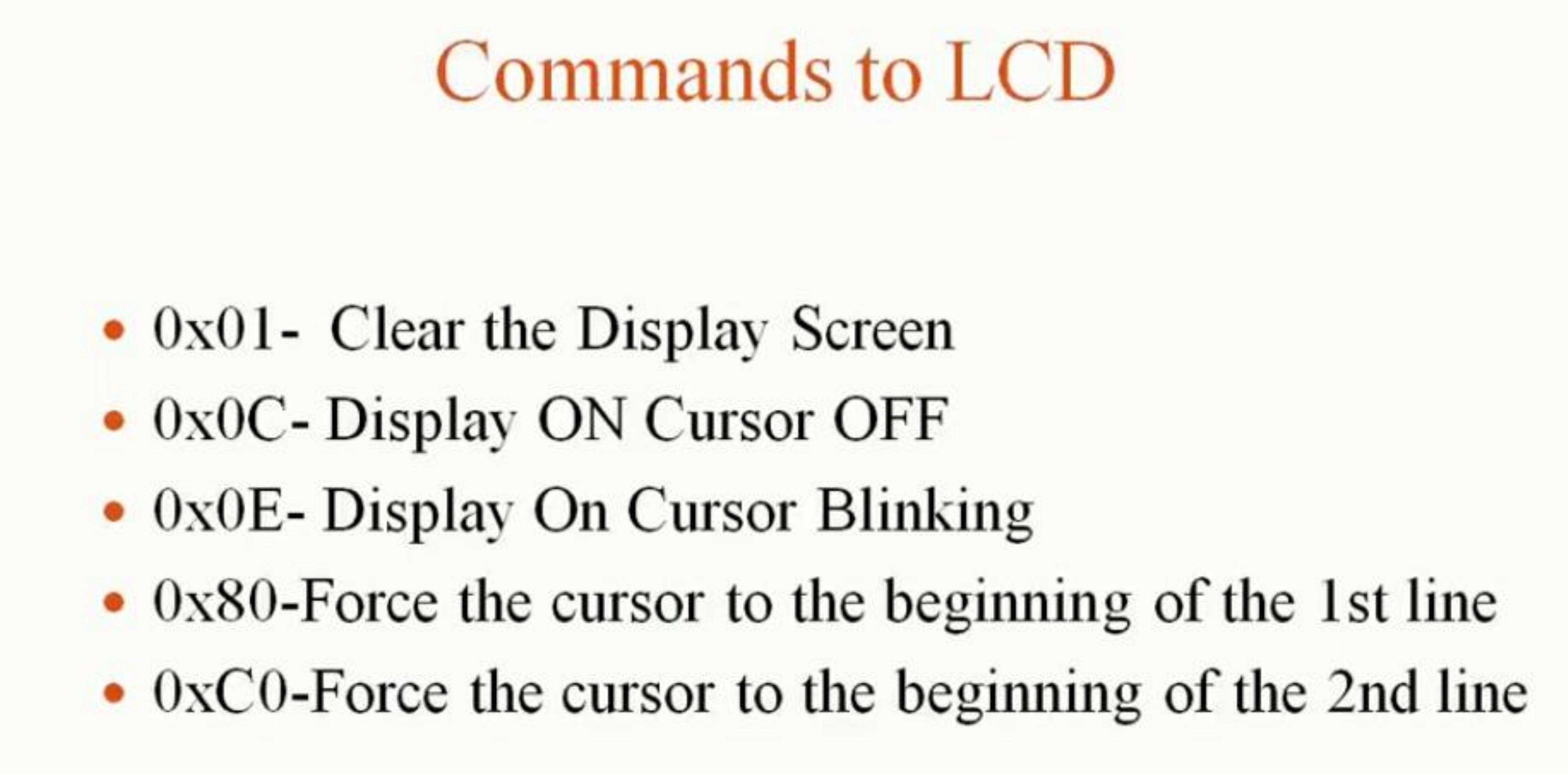
- Send Command 0x38



Sr.No.	Hex Code	Command to LCD instruction Register
1	01	Clear display screen
2	02	Return home
3	04	Decrement cursor (shift cursor to left)
4	06	Increment cursor (shift cursor to right)
5	05	Shift display right
6	07	Shift display left
7	08	Display off, cursor off
8	0A	Display off, cursor on
9	0C	Display on, cursor off
10	0E	Display on, cursor blinking
11	0F	Display on, cursor blinking

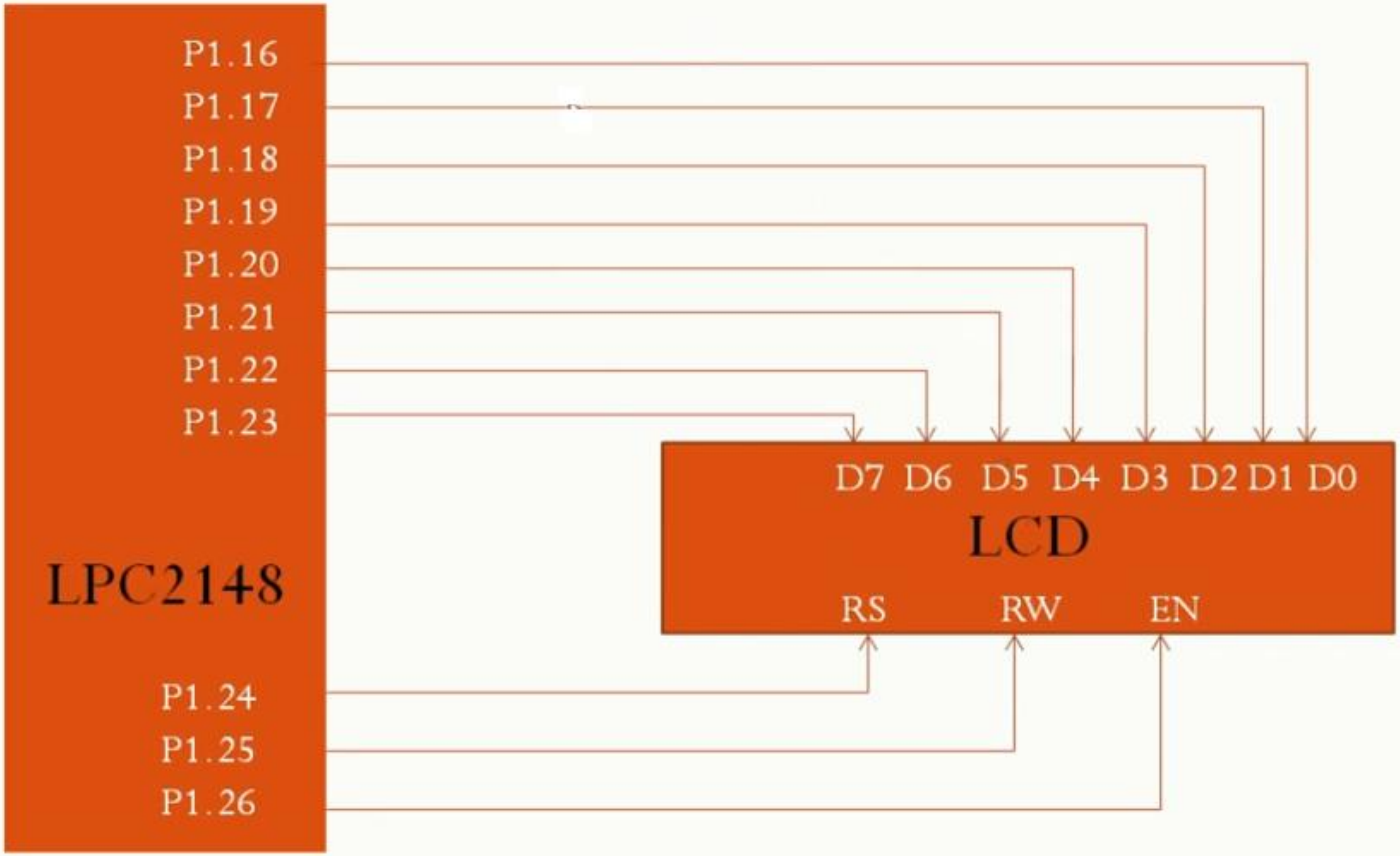


Sr.No.	Hex Code	Command to LCD instruction Register
12	10	Shift cursor position to left
13	14	Shift cursor position to right
14	18	Shift the entire display to the left
15	1C	Shift the entire display to the right
16	80	Force cursor to beginning (1st line)
17	C0	Force cursor to beginning (2nd line)
18	38	2 lines and 5×7 matrix



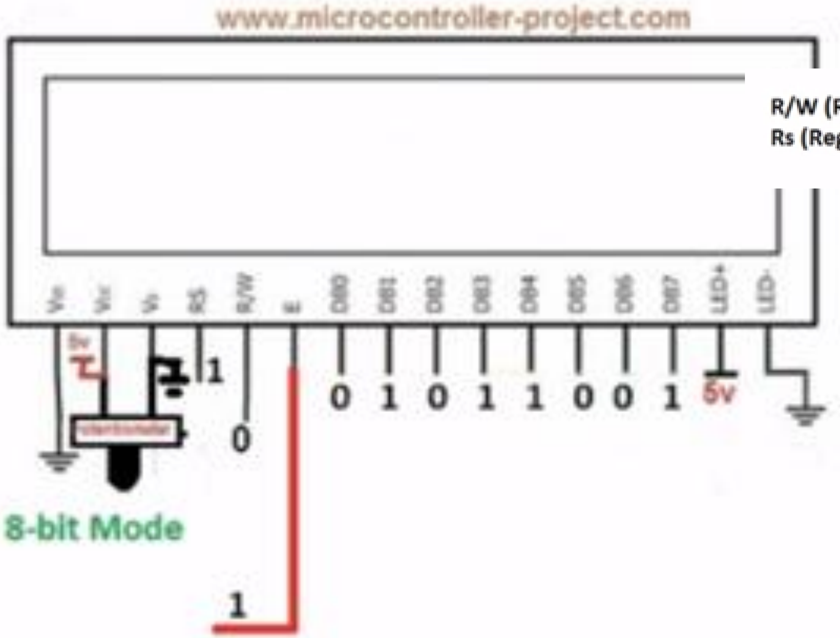
Commands to LCD

- 0x01- Clear the Display Screen
- 0x0C- Display ON Cursor OFF
- 0x0E- Display On Cursor Blinking
- 0x80-Force the cursor to the beginning of the 1st line
- 0xC0-Force the cursor to the beginning of the 2nd line


P1.16
P1.17
P1.18
P1.19
P1.20
P1.21
P1.22
P1.23
LPC2148
P1.24
P1.25
P1.26
D7
D6
D5
D4
D3
D2
D1
D0
LCD
RS
RW
EN

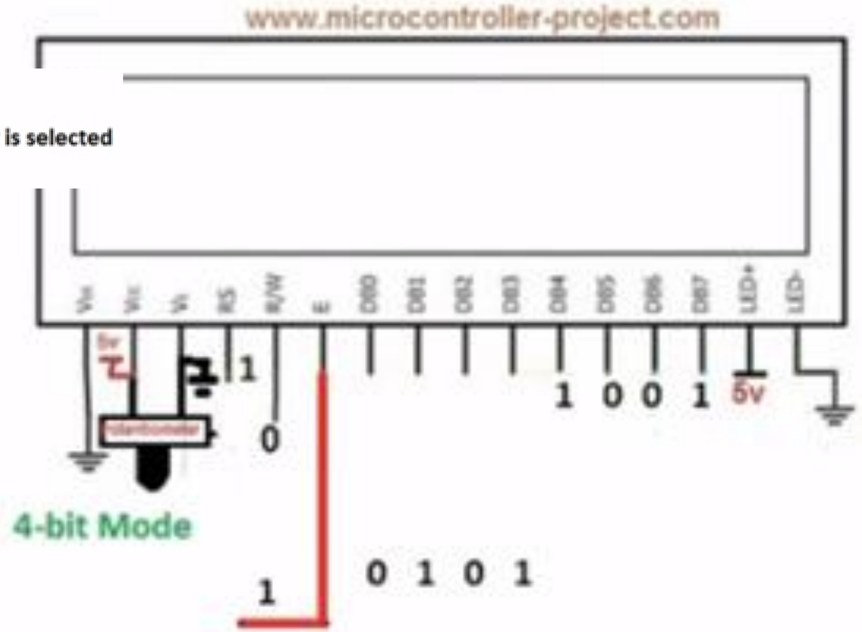
R/W (Read/Write) =0 Write to Lcd

Rs (Register Select)=1 Data Register is selected



8-bit Mode

R/W (Read/Write) =0 Write to Lcd
Rs (Register Select)=1 Data Register is selected



4-bit Mode