Unit-2

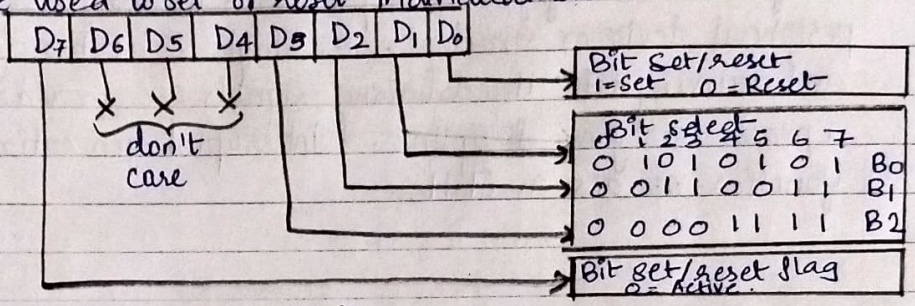
(1) Illustrate the operating modes of 8255 PPI. (5) Explain the different operating modes of 8255

→ The different modes of 8255 PPI are

1) BSR mode
2) I/O mode
   - mode 0 : Simple I/O mode
   - mode 1 : I/O with handshaking mode
   - mode 2 : Bidirectional data transfer mode

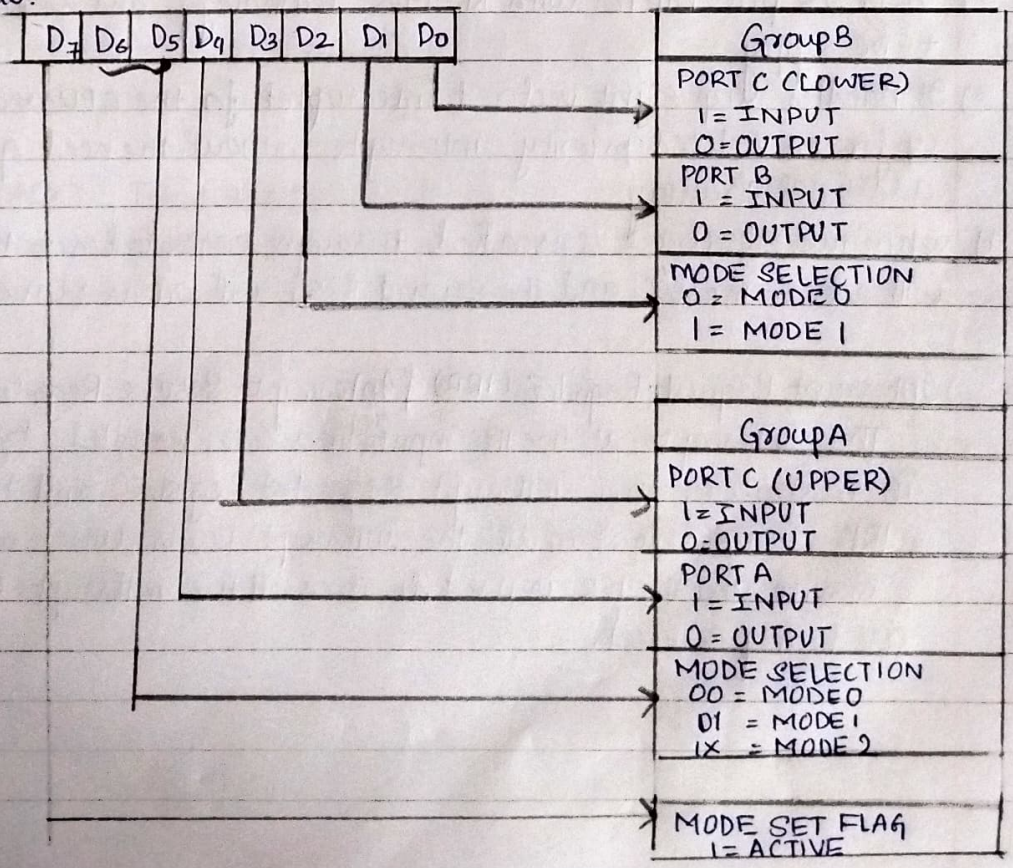
1) BSR (Bit Set/Reset) mode

Individual bit of port C can be set or reset by sending out single OUT instruction to the control signal. When port C is used for control/status operation, this feature can be used to set or reset individual bit.



A BSR word is to be written for each bit that is to be set or reset. The BSR word can also be used for enabling or disabling the interrupt signals generated by port C when 8255 is programmed for mode1 or mode2 operation.

2) I/O mode:



(i) mode 0:
This mode provides simple output and input operations for each of the three ports. Data is simply written to or read from a specified port

(ii) Mode 1:
It provides means for transfering I/O data to or from a specified port in conjunction with strobes or hand shaking signals. Port A and Port B use the lines on port C for handshaking signals.

(iii) Mode 2:-
This functional configuration provides a means for communicating with a peripheral device or structure. on a single 8-bit bus for both transmitting and receiving data Handshaking signals are provides to maintain a proper bus flow discipline. Interrupted generation and enable/disable function are also available.

2) List the salient features and explain interrupt servie register interrupt request register of 8259 interrupt controller.

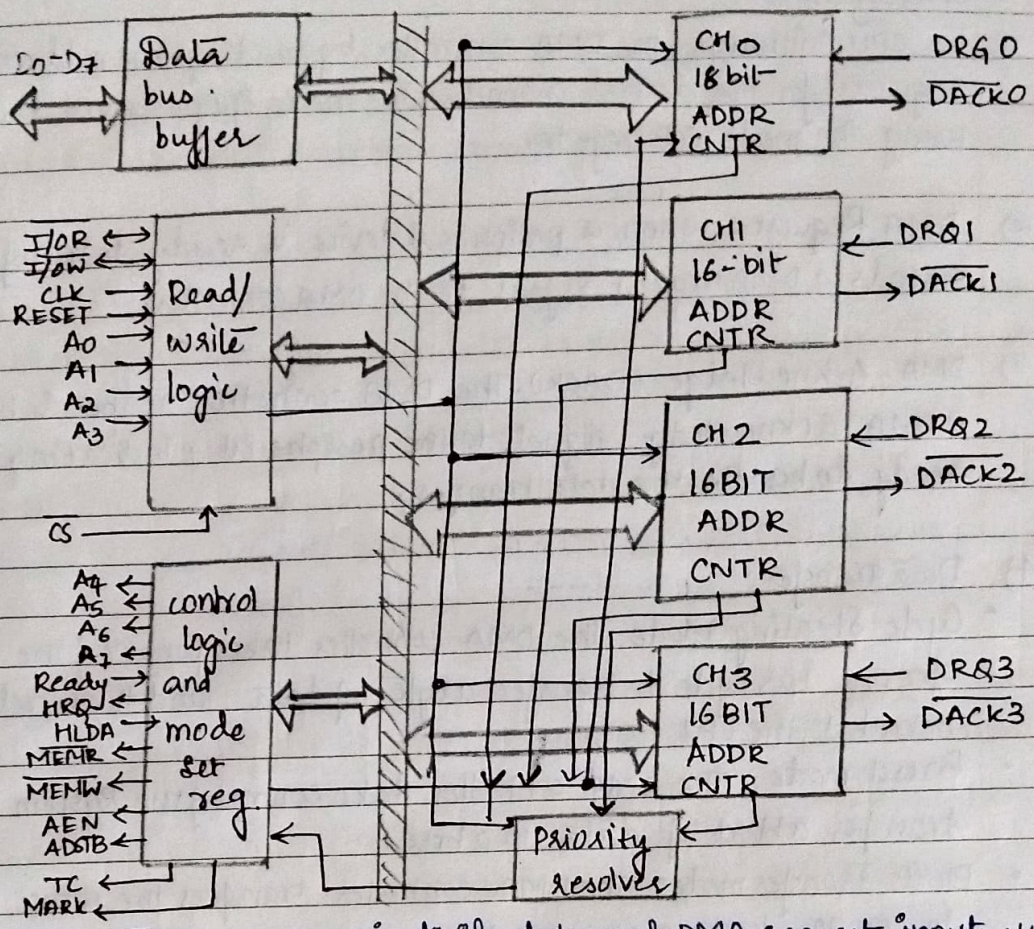
→ The salient features of 8259 interrupt controller

1) INTEL 8259 is a single chip programmable interrupt controller which is compatiable with 8085, 8086 and 8088 processors.
2) It is 28 pin DIPIC with N-Mos technology and require a single +5DC supply
3) It handles upeight vectred interrupted for the CPU and cascadable for upfor 64 vectred priority interrupts without the need of any additional circuitry
4) when two 8259s are cascaded through cascade lines the first 8259 will act as master and the second 8259 will act as slave.

Interrupt Request Register (IRR) & Interrupt Servie Register (ISR)

The Interrupts at the IRs input lines are handled by two registers in cascade, the interrupt Request Register and In Service. The IRR is used to store all the interrupt levels which are requesting Service and the ISR is used to store all the interrupt levels which are being serviced.

3) Illustrate the working of 8257 DMA with neat diagram.



DRQ0 - DRQ3 : These are the 4 individual channel DMA request inputs, which are used by the peripheral devices for using DMA services. When the fixed priority mode is selected, the DRQ0 has first highest priority and DRQ3 has lowest priority among them.

DACK0 - DACK3 : These are the active low DMA acknowledge lines, which updates the requesting peripherial about the status of their request by the CPU. These lines can also act as strobe lines for the requesting devices.

D0-D7 : These are bidirectional, data lines which is used to interface the system bus with internal databus of DMA controller. In the slave mode, it carries command words to 8257 and status word from 8257.
On the master mode, these lines are used to send higher bytes of the generated address to the latch.

Working of 8257 DMA controller

1) Initialization:-
The CPU initializes the DMA controller by loading the address and count registers for the desired channels. The mode of operations is also set using the mode set register.

2) DMA Request: when a peripheral device is ready to transfer data, it sends a DMA request signal to the DMA controller.

3) DMA Acknowledge (DACK): The DMA controller responds by a sending a DMA acknowledge signal to the peripheral, indicating that it is ready to handle the data transfer.

4) Data transfer:
- Cycle Stealing Mode: The DMA controller takes control of the system bus for one bus cycle to transfer a byte of data and then relinquishes control to the CPU.
- Burst mode: The DMA controller take control of the system bus and transfer a block of data in a burst
- Block Transfer mode: The DMA controller transfers the entire block of data in one go, keeping control of bus throughout the transfer.

5) Memory access: Depending on the operation mode set, the DMA controller either read data from memory.

6) Completion: Once the transfer count reaches zero, the DMA controller deactivate the DACK signal, indicating the completion of the transfer.

4) Discuss the block diagram of 8255 and explain the function of each block.
⇒ PC_0-PC_7: These 8-bit bidirectional I/O pins are divided into two groups PC_1 and PC_2. These group can indivually transfer data in or out when programmed I/O. when programmed in bidirectional or handshake modes these bits used as handshake signals.

RD' - MPU or CPU reads data in the ports or the status words through data buffer.

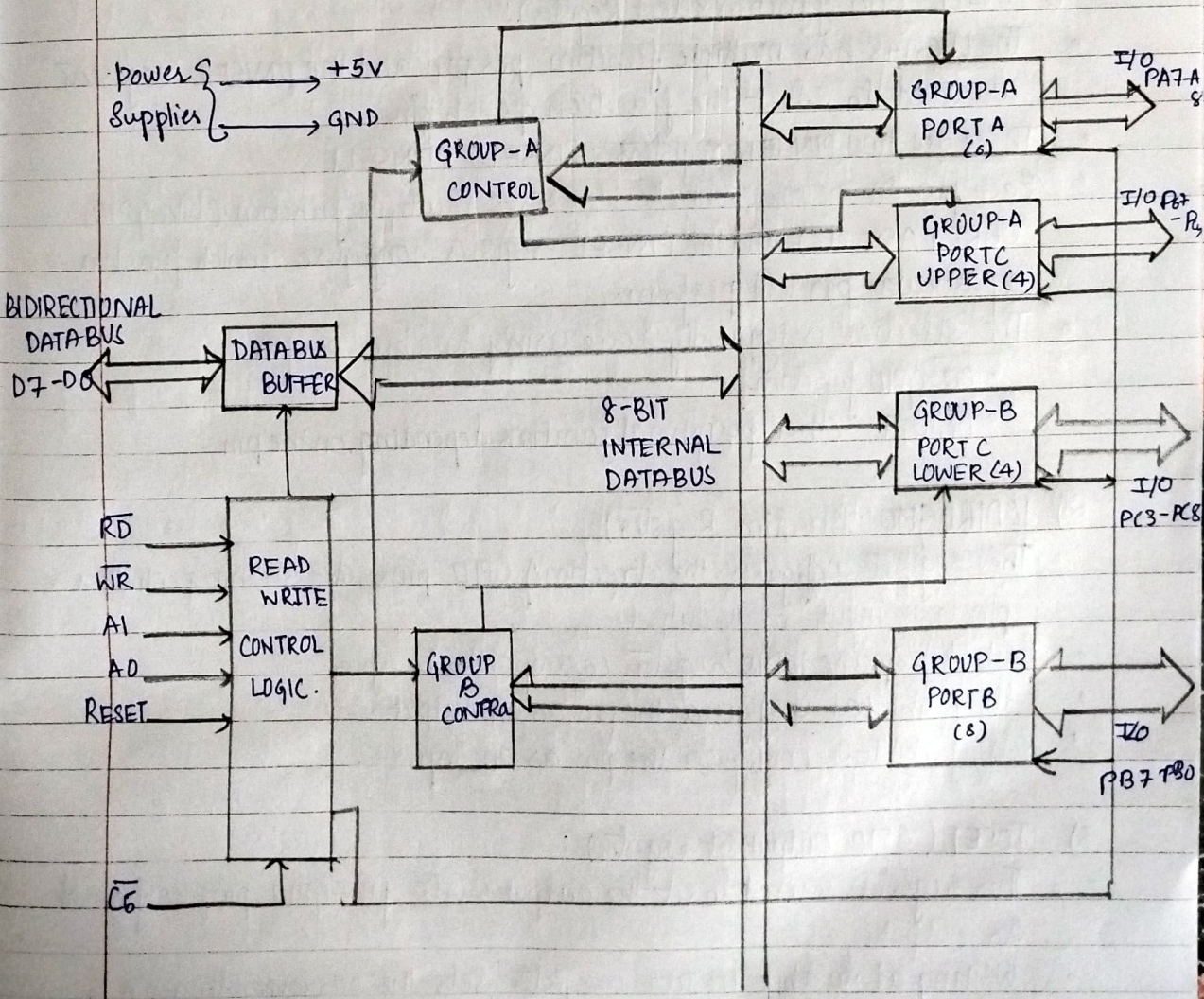
WR' : MPU or CPU writes data in the ports or the status words through data buffer.

CS' (chip select) : It is an active below input which can be used to enable 8255 for data transfer operation b/w CPU and 8255

RESET : It is an active high input used to reset 8255. When reset input is high, the control register is cleared and all the port are set to the input mode. Usually RESETOUT signal from 8085 is used to reset 8255.

Data Bus Buffer :- Tri state bidirectional buffer is used to interface the internal data bus of 8255 of the system data bus. Output data from the MPU to the ports or control register and the input data to the MPU from the ports or status register are all pushed through the buffer.

Control Logic :-
This block accepts controls bus signals as well as input from the address bus and issues commands to the individual group control blocks.



Unit-5

1) Explain registers associated in programming GPIO's in LPC2148 ARM7 processor (PINSEL, IOSET, IOCLR, IODIR)

⇒ Device pins which are not connected to a specific peripheral function are controlled by GPIO registers. Pins may be dynamically configured as i/p or o/p. Seperate registers allow setting or clearing any no of o/p simultaneously.

The value of o/p registers may be read back, as well as the current state of the port pins.

GPIO's registers are relocated to the ARM local bus for the fastest possible I/O timing.

All GPIO registers are byte addressable.

In LPC2148 ARM7 microcontrollers, GPIO pins are controlled and configurated through various registers

1) PINSEL (Pin function select Register):

- The LPC2148 has multiple function per pin and the PINSEL registers are used to select the function of each pin
- There are two PINSEL registers: PINSEL0, PINSEL1
- Each pin on the microcontroller can serve multiple function (like GPIO, UART, ADC etc) and the PINSEL register configure which function a particular pin will perform.
- The selection is typically done using two bits per pin.
  - 00 - GPIO function
  - 01, 10, 11 - other peripheral functions depending on the pin.

2) IODIR (GPIO Direction Register)

- This register configures the direction of GPIO pins, determining whether a pin is an input or an output.
- Each bit in the IODIR register corresponds to a pin
- Setting a bit to 1 configures the pin as an output.
- Setting a bit to 0 configures the pin as an input.

3) IOSET (GPIO Output Set register):

- This register is used to set the output state of the GPIO pins configured as outputs.
- Writing a 1 to a bit in IOSET register sets the corresponding pin to high.

- writing a 0 has no effect

4) IOCLR (GPIO Output clear register):
- This register is used to clear the output state of the GPIO pins configured as output
- Writing a 1 to a bit in the IOCLR register clears the corresponding pin to low
- writing a 0 has no effect

2) Explain PCB in LPC2148

⇒ The Pin connection Block in LPC2148 microcontroller is a verstile system that allows each pin on the microcontroller to be configured for multiple function such as GPIO, ADC, UART, PWM, SPI or I2C. This flexibility is managed through the use of specific configuration registers, primarily PINSEL0 and PINSEL1. These registers control the function of the microcontroller's pins, P0.0 to P0.31 and P1.16 to P1.31 by setting specific bit to select the desired function for each pin. For instance, setting pin bit in PINSEL0 can configure pin P0.0 as a UART transmit pin, PWM output, or a standard GPIO. The multiplexing capability allows efficient utilization of the limited pin count, enabling the LPC2148 to support a wide array of peripheral and interfaces. Proper configuration of these registers is crucial for ensuring that each pin perform its intended function in a given application.

- 64 pins are attached to two 32 bit I/O ports, Port0 & Port1
- Port-0, Port-1 pins are designed as P0.0 - P0.31 & P1.0 - P1.31
- Pins P0.24, P0.26, P0.27, P1.0 - P1.15 are unavailable
  Pin functions are multiplexed up to 4 function assigned to each pin
- Port-0 pins multiplex peripheral pin, comm interface pin function
- Port-1 pins multiplex - JTAG interface, trace function
- Disadvantage: If function not carefully seleted, some can't be availed
- advantage: keep size small, adds more functionalities to desvices

An function selects Registers: PINSEL0, PINSEL1, PINSEL2
- PINSEL0 selects functions of pin P0.0 to P0.15.
- PINSEL1 selects function of pin P0.16 to P0.31
- PINSEL2 selects functions of pin P1.16 to P1.31

3) Write the steps for programming of PLL

→ Programming a phase locked loop (PLL) involves several key steps from understanding the system requirements to writing the code to configure the PLL.

steps for programming PLL

1) Understanding PLL component and specification

- Gain a through understanding of PLL's key components : The phase Detector (PD), Low pass filter (LPF), Voltage controlled Oscillator (VCO) and the dividers. This foundational knowledge is essential for effective PLL.

2) Define System Requirement : Establish the critical parameters such as the desired output frequency (Fout), the reference frequency (Fref), the VCO's operational frequency range, and the acceptable level of phase noise and jitter. This step ensures PLL is desired to meet the system's specification.

3) Calculate Divider Values : perform calculation to determine the appropriate divider values. This involves :

- N divider : The feedback divider that scales the VCO output
- M divider : The reference divides that scales the input frequency
- P divider : The output divider.

4) Configure PLL register : Identify and set the necessary registers in the PLL These typically includes.

- Control Register : for general configuration settings.
- N-divider Register : To set the feedback divider
- M-divider Register : To set the reference divider
- P-divider Register : For the output divider.
- VCO control Register : To configure VCO parameter

5) Initialize the PLL: Proceed with initializing the PLL by:
- Powering up the PLL: Ensure the PLL is powered and enabled.
- Setting divider values: Input the calculated N and M and P values into their respective register
- Enabling the PLL: Activate the PLL and allow it to lock onto the desired frequency

6) Monitor PLL Lock status: Continuously check the PLL's lock status through a status register or lock detect pin. This step is crucial to confirm the PLL has achieved phase lock.

7) Fine Tuning: If the PLL does not lock or the performance does not meet specification.

8) Test and validate: Conduct comprehensive testing to ensure the PLL meets all desired specification. These includes verifying frequency accuracy, measuring phase noise and jitter and accessing stability across the expected temperature range.

4) Write the steps for programming of Timer 0/ Timer 1

⇒ Programming Timer 0 or Timer 1 on a microcontroller typically involves several steps to set up and configure the timer for specific application such as generating delays, creating PWM signals or counting events. Here are the general steps for programming Timer 0 or Timer 1

1) Understanding the Timer Basics: Familiarize yourself with the microcontroller's timer features by reading the datasheet. Understand the timer's modes available prescalar options and the timer registers.

2) Define Application Requirement: Determine the specific requirement for your application such as the desired time delay, frequency for PWM, or event counting.

3) Select the Timer and mode: choose the appropriate timer and the operating mode that best suits your application.

Configuration steps.

Step1: Calculate Timer values
Calculate the timer's initial count values and the prescalar setting based on the desired time delay or frequency.

For a desired delayed time T with a clock frequency Fclk and prescalar value Pi

Count = Fclk x T/P

Step2: Set the Prescalar
Select the appropriate prescalar value to achieve the desired timing resolution and range. Prescalar values are typically available as powers of two multiples.

Step3: Configure the time registers
Set the timers control register to configure the timers mode and prescalar
Step4: Load the timer counter
Load the initial count value into the timer's counter register
Step5: Enable timer interrupts
Enable timer interrupts by setting the appropriate interrupt mask register bit. for Ensure the global interrupt enable bit is set in the status register
Step6: Start the timers by setting the appropriate bits in the timer control register.

5) Write a program to interface LPC2148 ARM7 processor to blink the LED's one after the other with delay of 50000000.

⇒
```
#include <LPC21XX.h>
unsigned int delay;
int main()
{
    PINSEL0 = 0x00000000; // configure P0.0 to P0.15 as GPIO
    PINSEL1 = 0x00000000; // configure P0.16 to P0.31 as GPIO
    IO0DIR = 0x00FF0000; // configure P0.16 to P0.23 as output
while(1)
 { IO0CLR = 0x00FF0000; // CLEAR(0) P0.10 to P0.13 and
   // P018 to P0.21 , LEDs ON
```

```
for(delay=0; delay<50000000 ; delay++); // delay
IO0SET = 0x00FF0000; // set (1) P0.10 to P0.13, P0.18 to P0.21 , LED's OFF
for(delay=0; delay<50000000; delay++); // delay
}
}
```

6) Write a program to interface LPC2148 ARM7 processor to rotate DC motor thrice in clockwise and five times in anti-clockwise.

⇒
```
#include<lpc214x.h>
void clock_wise(void);
Void anti_clock_wise(void);
Void delay(unsigned int count);

int main(void)
{
  IO0DIR |= (1<<8) | (1<<11);
  IO0SET = (1<<8);

  while(1)
  {
    for(unsigned int i=0; i<3; i++)
    { clockwise();
    delay(50000000);
    IO0CLR = (1<<11);
    delay(50000000);
    }
    delay(10000000);

  for(unsigned int i=0; i<5; i++)
    { anti_clockwise();
      delay(50000000);
      IO0CLR = (1<<11);
      delay(50000000);
    }
  delay(10000000);
}
```

```
void clock_wise (void)
 {
    IO0SET = (1<<11);
 }

void anti_clockwise (void)
  {
      IO0CLR = (1<<11);
      IO0SET = (1<<8);
  }

void delay (unsigned int count).
  {
    for (unsigned int i=0; i<count; i++);
  }
```