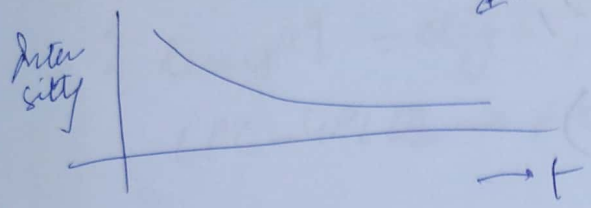
* We exploit the characteristics of LEDs that when powered off their intensity decreases exponentially (not instantaneously)



18.2.19

Display a 4 digit no on the multiplexed 7 segment display

P0.11 – P0.4 → Seven segment code (h-g)

P1.26 – P1.23 → decoder i/p

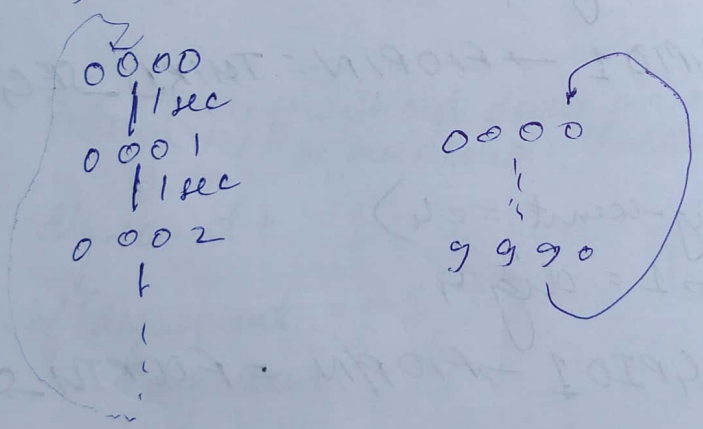
dig1 dig2 dig3 dig4 delay (time b/w 2 digits)

```
#include <lpc17xx.h>
#define FIRST_SEG  0<<23 //bc starts from 23.
#define SECOND_SEG 1<<23
#define THIRD_SEG  2<<23
#define FOURTH_SEG 3<<23
unsigned int dig_count, dig1=4, dig2=3, dig3=2,
 dig4 = 1.
unsigned char array_dec[] = {0x3F, 0x06 ...
// The array_dec is the 7seg lookup table
```

void delay (void)
{ for (i=0; i<500; i++);
}
// if delay is too large, LEDs can switch off if they decay.

Q simulate a 4 digit decimal up counter using multiplexed 7 seg display.

Ans



#include <lpc17xx.h>
#define
#define
#define
#define
#define
unsigned int count=0;
unsigned char one_sec_flag='0';

All these same as prev ques.

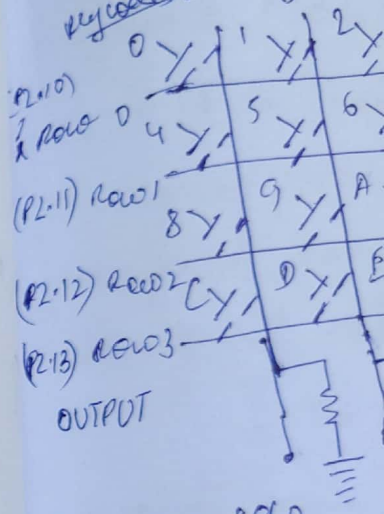
void main
{
LPC_GP
LPC_GP
while(
{ delay
dig_co
if (d
if (co
{

```
void main (void)
{
LPC_GPIO0->FIODIR = 0x00000FF0;
LPC_GPIO1->FIODIR= 0x07800000;
while(1)
{ delay (1)
  dig_count += 1
  if (dig_count == 5)
      dig_count = 0;
  if (one_sec_flag == '1')
    { one_sec_flag = 0;
      dig1 += 1;
      if (dig1 == 0x0A)
       { dig1 = 0;
         dig2 += 1;
         if (dig2 == 0x0A)
           { dig2=0;
             dig3+ = 1;
             if (dig3 == 0x0A)
               { dig3=0
                 dig4 += 1;
                 if (dig4 == 0x0A)
                   { dig4=0}}}}}
```

```
Display();
}
}
/* Rest same as prev */
/* updation of delay to keep track of
   1 second */
void Delay(void)
{
   for(i=0; i<500; i++); // assume it gives 1ms delay
   if(count==1000)
     { count=0;
       one_sec_flag = '1';
     }
   count+=1;
}
```

21-2-19

Interfacing 4



* If CNC or C

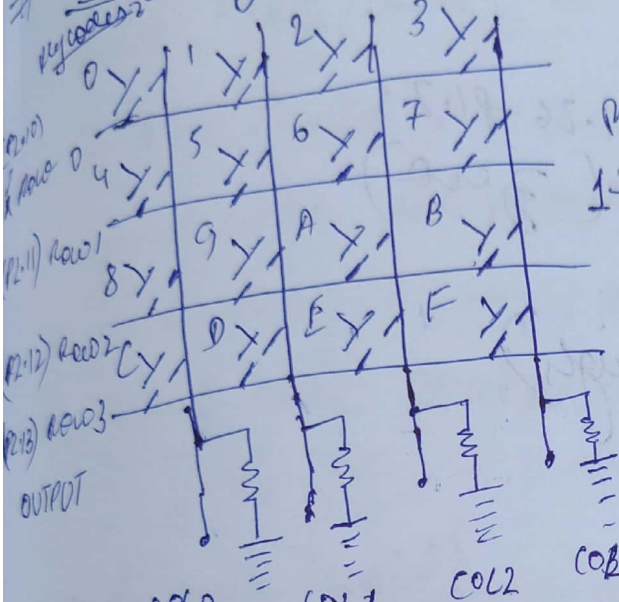
* keycodes → u
  ↳ (row, 2bits
  00
  01
  10
  11

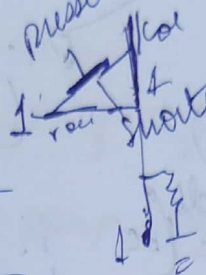
* To know not →

* Logical

21.2.19

Interfacing 4×4 matrix keyboard





all pins connector B.

CN3

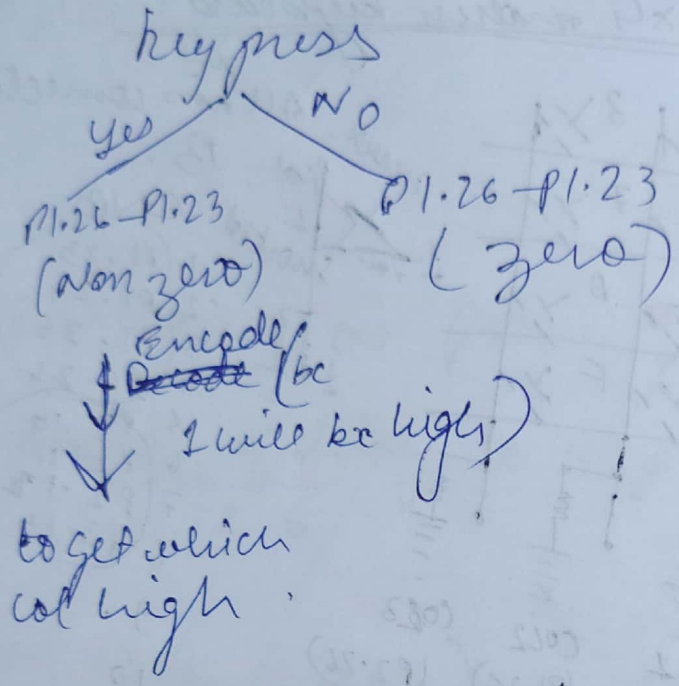
1	P1.23	COL
2	P1.24	
3	P1.25	
4	P1.26	
5	P2.10	ROW
6	P2.11	
7	P2.12	
8	P2.13	
9		
10		

* If CN3 or CN4 → then 1st 4 are cols last 4 are rows.

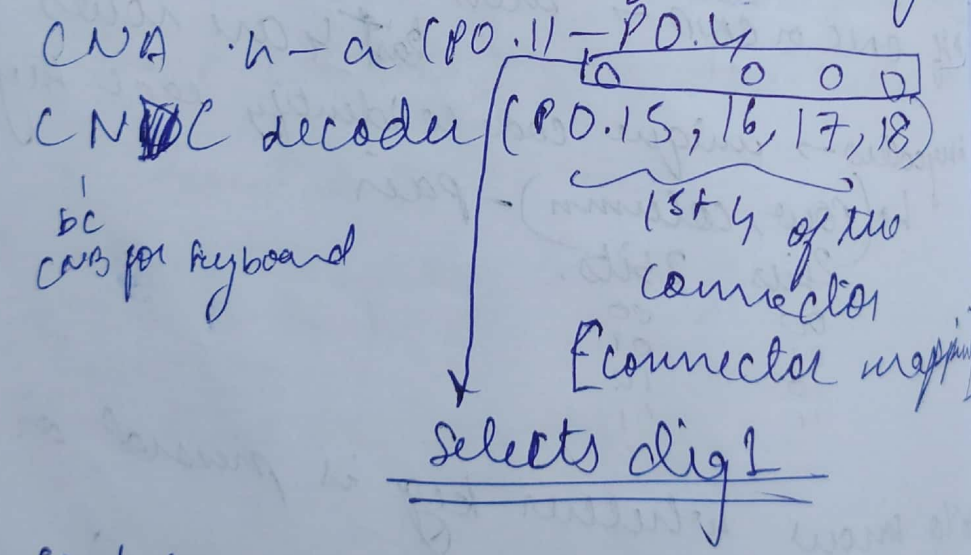
* keycodes → unique code to identify each key.
  ↳ (row, column) - pair
  2bits 2bits.
  00 00
  01 01
  10 10
  11 11

* To know whether key is pressed or not → keyscan
  basically scanning the rows.

* logical 1 → send to row which means you are scanning row others send logical 0.



Q write a program to display the keycode of the key pressed on the seven segment disp.



look carefully at CNA and CNC ~~ports~~ pins. We have same port P0. Therefore we have to apply MASKS.

#include <lpc
#define FIRS
void scan(u
unsigned cha
unsigned lon
unsigned ch
= { 0x
void main
{
LPC_
LPC_
LPC_
while(
{
for(r
{ if(
el
el

```
#include <lpc17xx.h>
#define FIRST_SEG 0<<15
void scan(void);
unsigned char col, row, flag;
unsigned long i, var1, temp, temp2, temp3;
unsigned char SEVEN_CODE[4][4]
  = { 0x 3F, . . . . . };

void main(void)
{
  _____
  _____

  LPC_GPIO0->FIODIR = 0xFFFFFFFF;
                          //output.

  LPC_GPIO1->FIODIR = 0; //Input.
  LPC_GPIO2->FIODIR = 0x00003C00;
                  // P2.10-13 output.

  while(1)
  {
  for(row = 0; row < 4; row++)
  {
    if(row == 0)
      temp = 1<<10;
    else if(row == 1)
      temp = 1<<11;
    else if(row == 2)
      temp = 1<<12;
```

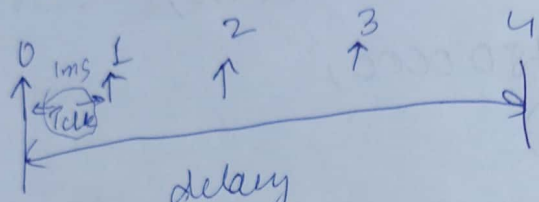
```
else if (row == 3)
    temp = 1 << 13;
LPC_GPIO2 -> FIOPIN = temp;
flag = 0;
Scan();
if(flag == 1)
{ temp2 = SEVEN_CODE[row][col];
  LPC_GPIO0 -> FIOMASK = 0xFF87FFFF;
                    // unblock P0.15-8 decoder
  LPC_GPIO0 -> FIOPIN = FIRST_SEG;
                    /// 0000 on P0.18-P0.15 digit
  temp2 = temp2 << 4;
  LPC_GPIO0 -> FIOMASK = 0xFFFFFF00F;
                    // P0.11-4 unblock
  LPC_GPIO0 -> FIOPIN = temp2;
  break;
} // if
} // for
} // while
```

```
void scan(void)
{
unsigned long temp3;
temp3 = LPC_GPIO1->FIOPIN;
temp3 &= 0x07800000;
if(temp3 != 0)
{ flag = 1;
  if (temp3 == 1<<23)
    col = 0;
  else if (temp3 == 1<<24)
    col = 1;
  else if (temp3 == 1<<25)
    col = 2;
  else if (temp3 == 1<<26)
    col = 3;
}
```

Timer / counter programming

delay — event count.

$4 T_{clk} = T_d$



* when you use ~~timer to count~~ counter to count time delay, → timer
  ↳ system clock is periodic

* If we have to use counters →
  counters used is event clock.
  ↳ aperiodic
  ↳ depends on external event

25.2.19

Timer 0, 1, 2, 3
(32-bit) count $0 - (2^{32} - 1)$

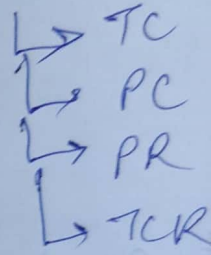
t_{PCLK} → period of peripheral clock

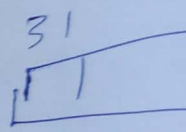
CCLK → PCLK → 3 MHz (default) derived from circuit clock.
↳ from internal oscillator.

f_{PCLK} — 3 M

$T_{PCLK} = \frac{1}{f_{PC}}$

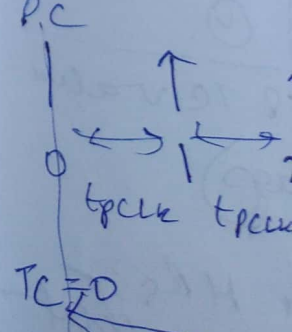
* each timer
registers).





PR → scalin
counter t

PC → +



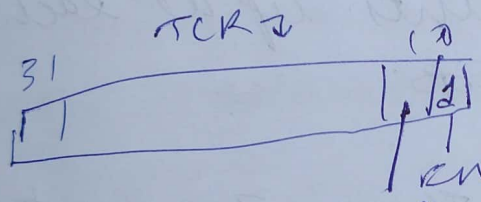
(PR +

f_{PCLK} — 3 MHz

$T_{PCLK} = \frac{1}{f_{PCLK}} \quad \frac{1}{3\times10^6} = \frac{1}{3}$ µs.

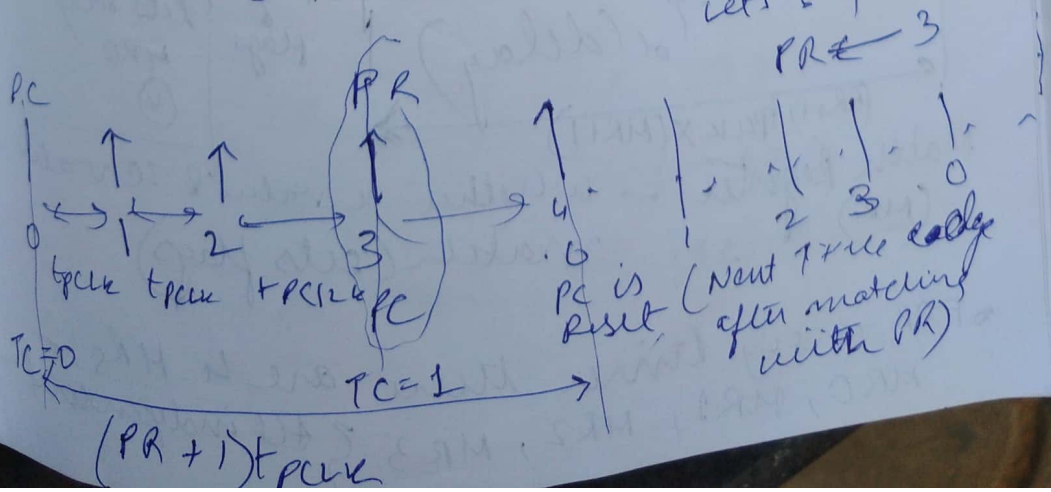
each timer has SFRs (special functional registers).

- TC (Timer Counter Register)
- PC (Prescale Counter)
- PR (Prescale Register)
- TCR (Timer Control Register)



PR → scaling the rate at which timer or counter counts up.

PC → time ↑ PC counts up.

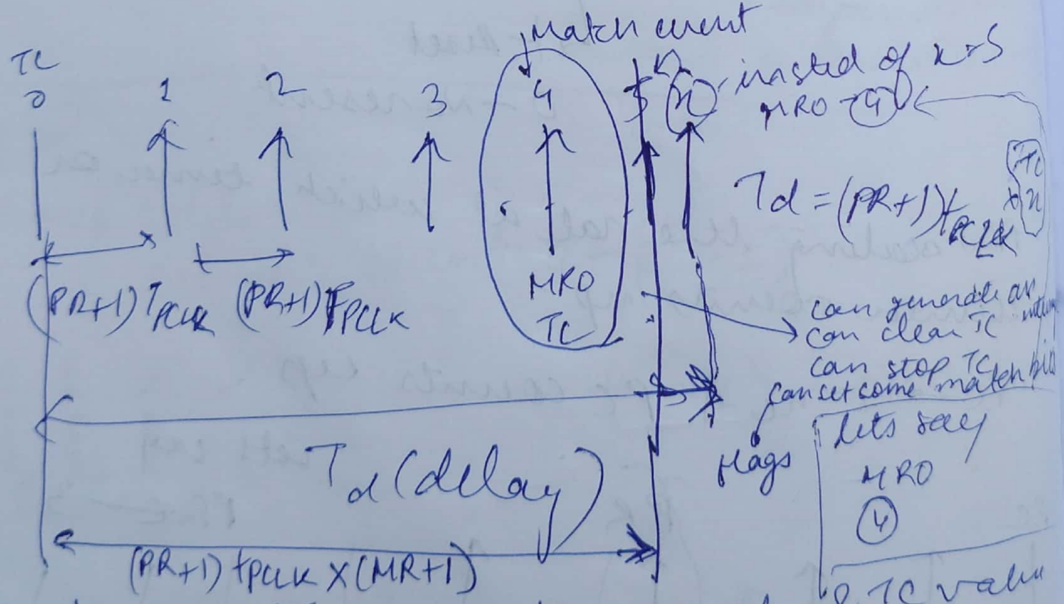


* TC is incremented when PC matches with PR (PC is reset)
* PC & PC are for prescaling purpose
* TC is the main counter

Q What is the rate at which our TC counts up?

Ans: $(PR+1)\,T_{PCLK}$

* If PR=0; TC counts up at each positive edge ~~of~~.
* Rate at which TC counts up ~~de~~ increases with increase in PR.



Match Register (MR) → whether x value & TC value match (sets flags)

* FOR every timer there are 4 MRs: MRO, MR1, MR2, MR3 } All independent

Td = (PR+1) tp...
+ time taken for match event

* keep polling t... it is captur... the delay...

* ~~For a given~~ MR has to... edge is cap...

Eg: f_{PCLK} = 3MH...

1 sec

T_d = (PR+1)...

1 sec = 1...

$T_{del} = (PR+1)\, t_{PCLK}\, (MR+1)$.

↳ time taken per match enter

* keep polling the flag (match bit as it is capturable). to calculate the delay.

* ~~To get a given delay~~ If $x = 5$, MR has to be 4 (as next rise edge is capturable)

Eg: $t_{PCLK} = 3\,MHz$

1 sec

$$T_{del} = \frac{(PR+1)\, t_{PCLK}\, (MR+1))}{t_{PCLK}}$$

$$T_{sec} = \frac{1000 \times 3000}{3 \times 10^6}$$

PR = 999 MR = 2995

PR = 2999 MR = 999

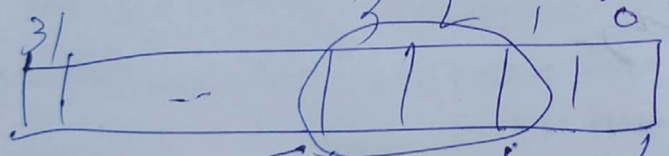
PR = 0 MR (3000000-1)

* Don't waste time in polling. Configure the system to call an interrupt after match event & then get flags.

* Timer as timer → source peripheral clock
* Timer as counter → external source
  ↳ from pins.

To differentiate b/w
2
↳ One SFR
↳ CTCR (Counter/Timer Control Register)

CTCR



Bits 1–0 (e.g.):

- 00 → Timer — on chip osc for PCLK of timer (3MHz)
- 01 → count at ↑
- 10 → count at ↓
- 11 → count at both edges

01, 10, 11 → counter → which pin? → as defined in 2,3

Bits 3–2:

→ 00 → CAP n.0 pin
01 → CAP n.1 pin

n = Timer no.

CAP → not GPIO function (Not GPIO)

eg: For:
timer 0 → CAP0.0, CAP0.1
timer 1 → CAP1.0, CAP1.1

Timer 0 : CAP 0.0 (P1.26)
CAP 0.1 (P1.27)

Timer 1 : CAP 1.0
CAP 1.1

Timer 2 : CAP 2.0
CAP 2.1

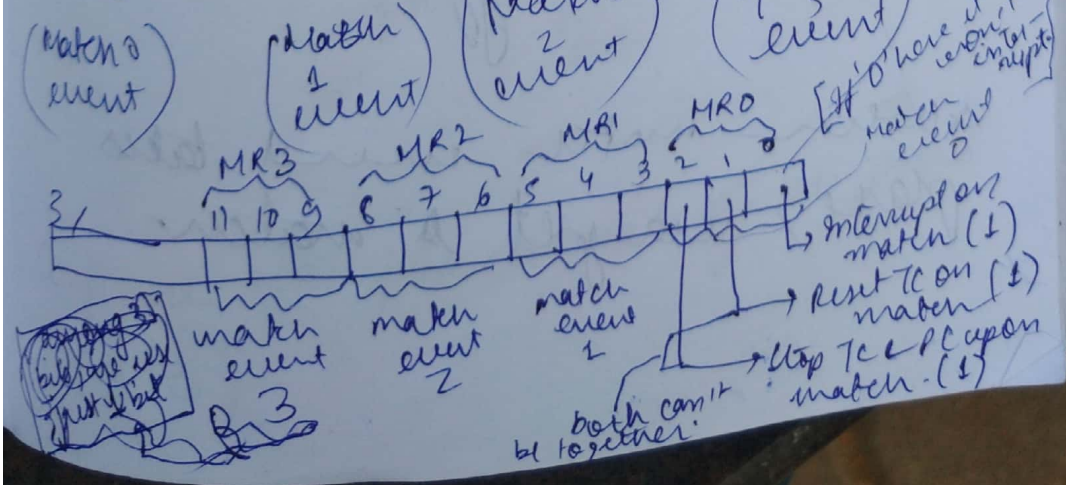
Timer 3 : CAP 3.0
CAP 3.1

CTCR
3 2 1 0
0 1 1 0
↳ Negative

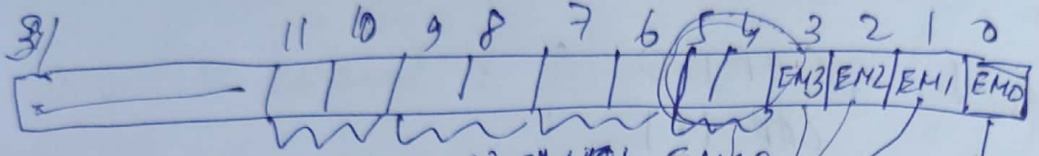
CAP n.1
Timer 1

⇒ MCR (Match Control Register)

MR0 MR1 MR2 MR3
TC TC TC TC
(match 0 event) (match 1 event) (match 2 event) (match 3 event)



⇒ External Match Register (EMR)



EMR3 EMR2 EMR1 EMR0

Match output for match 0

External match control bits for match 0.

match 2

match 3

what output to check delay in f^n after delay occurs

00 → Do nothing

01 → 0

10 → 1

11 → Toggle.

when match event takes place, we get a match.

EM bits can be pinned out using MAT pins. made available in I/O pins.

MAT ⊗ ⊗ — O/P pins — match output

Timer no.:

Timer 0

TC ⟷ MR0 MAT0.0 — P1.28 (fn 3)

TC ⟷ MR1 MAT0.1 — P1.29 (fn 3)

TC ⟷ MR2 MAT0.2

TC ⟷ MR3 MAT0.3

Timer 1

TC ⟷ MR0 MAT1.0

1.1

1.2

1.3

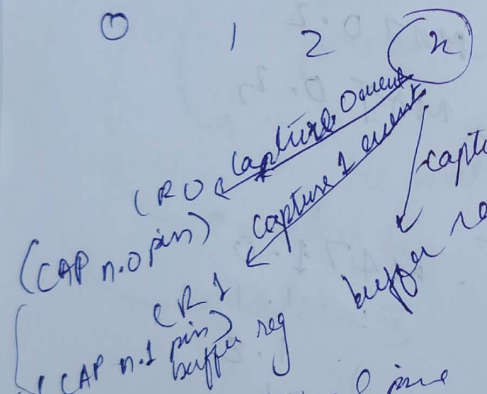
Match available at: EM0 — can be gotten at P1.28

EM1 → pin out. P1.29

Configure MAT functionality on pins (fn 3)

* when timer is counting up 0, 1, 2, ... (n) if we have to capture the value of n 2 store in a register (by default it is T0)

↳ capture Event:

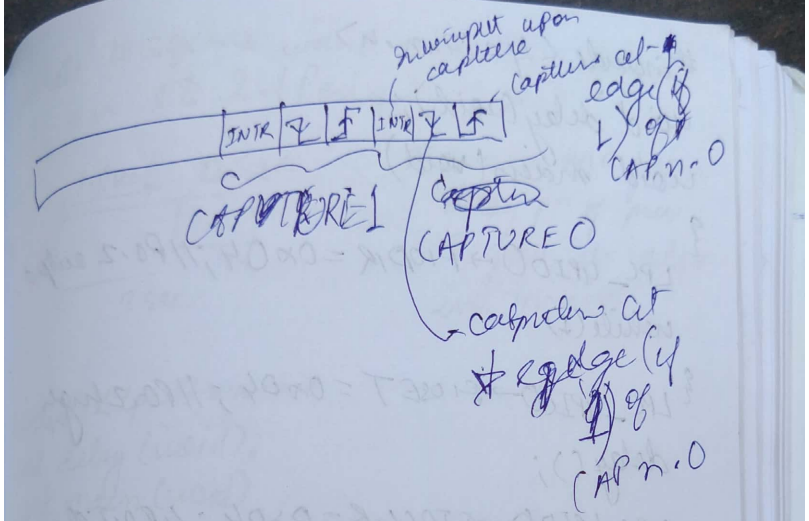


2 capture registers / Timer

CR0
CR1

→ CCR (capture control Reg)

How to tell in which (+ve or -ve edge) to capture

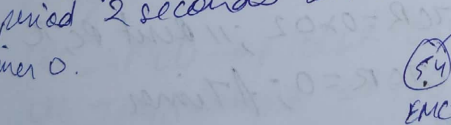


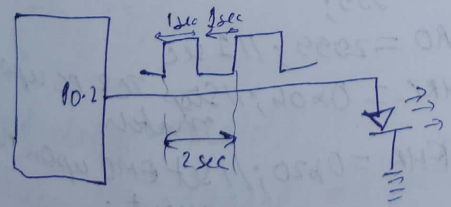
Q.19

Write an embedded C program to turn ON & OFF an LED connected to P0.2 for every second. use timer 0 for delay generation

OR

write a program to generate a square wave of period 2 seconds on P0.2 using timer 0.





Assume $P_{CLK} = 3$ MHz

$$T_d = \frac{(PR+1)(MR+1)}{f_{CLK}}$$

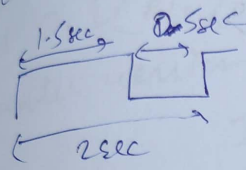
1000 3000

3×10^6 gives 1 sec

PR = 999, MR = 2999

```
#include <lpc17xx.h>
void delay(void);
void main(void)
{
  LPC_GPIO0->FIODIR = 0x04; //P0.2 output.
  while(1)
  {
    LPC_GPIO0->FIOSET = 0x04; //P0.2 high
    delay();
    LPC_GPIO0->FIOCLR = 0x04; //P0.2 clear
    delay();
  }
}
void delay(void)
{
  LPC_TIM0->TCR = 0x02; // Reset PC, TC
  LPC_TIM0->CTCR = 0; // Timer
  LPC_TIM0->PR = 999;
  LPC_TIM0->MR0 = 2999; //1 sec
  LPC_TIM0->MRC = 0x04; //Stop TC & PC upon match
  LPC_TIM0->EMR = 0x20; //Set EM0 upon Match0 event.
  LPC_TIM0->TCR = 0x01; // Start TC, PC
  while(!(LPC_TIM0->EMR & 0x01)); // To check if EM0 becomes '1'
  return;
}
```

Generate a square wave with 75% duty cycle on P0.2 w.r.t (Period = 2 sec)



* Remove MR from delay fn of prev 2 give its value in main.

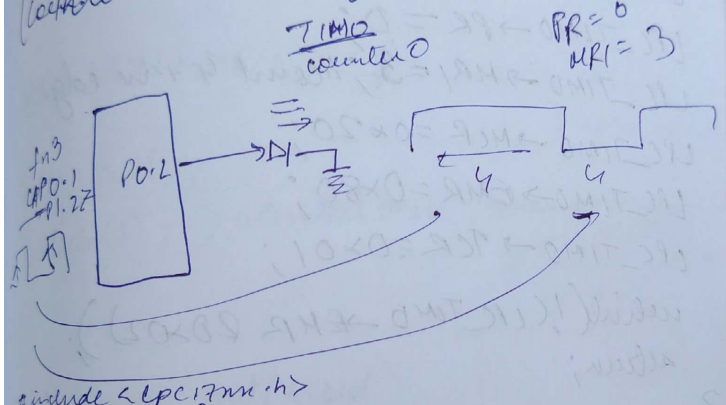
```
#include<lpc17xx.h>
void delay (void);
void main (void)
{ LPC_GPIO->FIODIR = 0x04; //P0.2 output.

  while(1)
  { LPC_GPIO0->FIOSET = 0x04;
    delay();
    LPC_TIM0->MR0 = 4499; // 1.5 = (1000 x n)/(3 x 10^6)
    delay(); //sec
    LPC_GPIO0->FIOCLR = 0x04
    LPC_TIM0->MR0 = 1499; // 0.5 = (1000 x 2)/(3 x 10^6)
    delay();
  }
}
void delay (void)
{ //Everything same as prev given delay.
  //Just remove
  LPC_TIM0->MR0 = 2999
}
```

→ this line remove

2. For the 50% ques (pwm to pwm)
use FIOPIN;

```
#include <lpc17xx.h>
void delay(void)
void main(void)
{
LPC_GPIO0->FIODIR = 0x04;
while(1)
{
LPC_GPIO0->FIOPIN = ~(LPC_GPIO0
                      ->FIOPIN & 0x04);
delay();
}
}
void delay()
{
//same pwm to pwm (50% duty
                   cycle)
}
```

Write a program to toggle P0.2 for every 4 clock pulses received at CAP 0.1 (P1.27, fn3)
[counter version of same prog]




```
#include <lpc17xx.h>
void delay(void);
void main(void)
{
LPC_PINCON->PINSEL3 = 3<<22 ; //P1.27 CAP0.1 fn3
LPC_GPIO0->FIODIR = 0x04 ; //P0.2 O/P
while(1)
{
LPC_GPIO0->FIOPIN = ~(LPC_GPIO0->FIOPIN & 0x04);
delay(); //wait for 4 ↓ on CAP0.1 (P1.27)
}
}
```

P1.0 .

```
void delay(void)
{
LPC_TIM0 -> TCR = 0x02
LPC_TIM0 -> CTCR = 0x05
LPC_TIM0 -> PR = 0;
LPC_TIM0 -> MR1 = 3; //count 4 rising edges.
LPC_TIM0 -> MCR = 0x20;
LPC_TIM0 -> EMR = 0x80;
LPC_TIM0 -> TCR = 0x01;
while(!(LPC_TIM0 -> EMR & 0x02));
return;
}
```

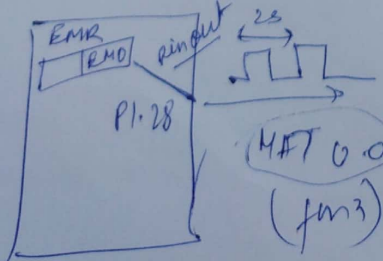
8.3.19

VIVA VOCE

32) Ich esse ____ ____ ____ und
ich trinke ____ ____ Tee

8.3.19

Q. write a program to generate a square wave of period 2 seconds on P1.28 (MAT 0.0 - fm3)

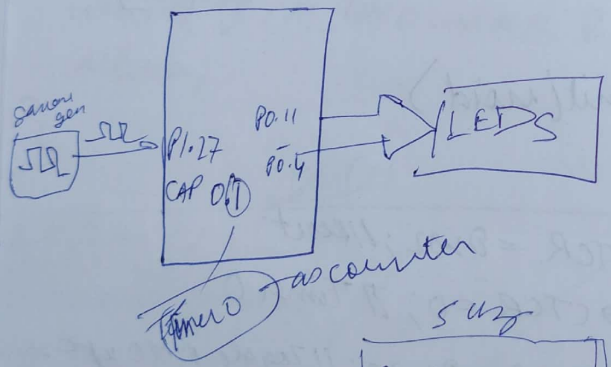


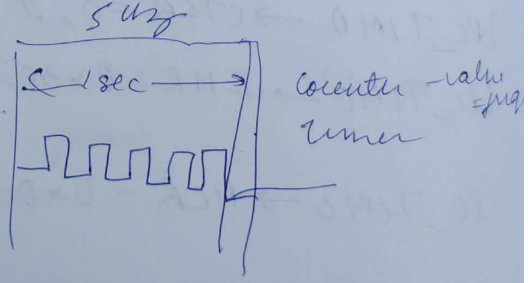
EM0 toggle for every second.

```
#include <lpc17xx.h>
void timer_init(void);
void main(void)
{
LPC_PINCON->PINSEL3 = 3<<24; //P1.28
                             // MAT0.0 fn3
timer_init();
while(1); // Parallel task [square wave gen by timer module so processor needn't do-what].
}
void timer_init(void)
{
~~LPC_TIM0~~
LPC_TIM0->TCR = 0x02; //Reset
LPC_TIM0->CTCR = 0; // Timer 0
LPC_TIM0->EMR = 0x30; // Toggle EM0 upon match 0.
LPC_TIM0->MCR = 0x02; // Reset TC upon match.
LPC_TIM0->PR = 0;
LPC_TIM0->MR0 = 299999; // 1 sec
LPC_TIM0->TCR = 0x01; // Start.
} // polling not required
```

Q Assume that output of a square wave generator is connected to P1.27 (CAP 0.1 - pin 3). Write a program to display the frequency of this square wave on the LEDs connected to ~~P0-P11~~ P0.4-11. Freq range (0 — 0>FF Hz) 0 — 255 Hz

1 Hz = cycles/sec





Timer 1 for 1 sec delay -

```
#include <LPC17xx.h>
void delay (void);
void counter0_init(void);

void main (void)
{
LPC_PINCON -> PINSEL3 = 3<<22; // P1.27 CAP 0.1 as 3
LPC_GPIO0 -> FIODIR = 0x00000FF0; // P0.11 - P0.4 output.

counter0_init();
while(1)
{
LPC_TIM0 -> TCR = 0x02; // Reset counter0
LPC_TIM0 -> TCR = 0x01; // start the counter 0.

delay(); // 1 sec.
n = LPC_TIM0 -> TC;
LPC_GPIO -> FIOPIN = n << 4; // display.
}
}
```

```
void counter0_init(void)
{
  LPC_TIM0->TC;
  LPC_TIM0->CTCR = 0x05; // counter at
                         //rise edge.
  LPC_TIM0->PR = 0; // At each rise edge
                    count
}
void delay(void) provides 1 sec delay.
{
  LPC_TIM1->TCR = 0x01;
  LPC_TIM1->CTCR = 0;
  LPC_TIM1->PR = 999;
  LPC_TIM1->MR0 = 2999; // 1 sec delay
  LPC_TIM1->MCR = 0x04;
  LPC_TIM1->EMR = 0x20;
  while(!(LPC_TIM1->EMR & 0x01));
}
```

CAP n.0

CAP n.1

CR0 ← TC

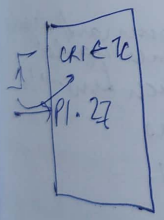
CR1 ← TC

CAP 0.1 P1.27 → fn3 ↓

LPC_PINCON → PINSEL3 |= (3 << 22); // CAP 0.1.

LPC_TIM0 → CCR = 0x08; // Capture TC to CR1 at +ve edge of CAP 0.1.

Instantaneous value of TC can be captured in one of CR registers by configuring the CAP n.0/1 pin & CCR register.



Eg: CAP 0.1 P1.27 ↓

CAP 0.0 P1.26 ↓

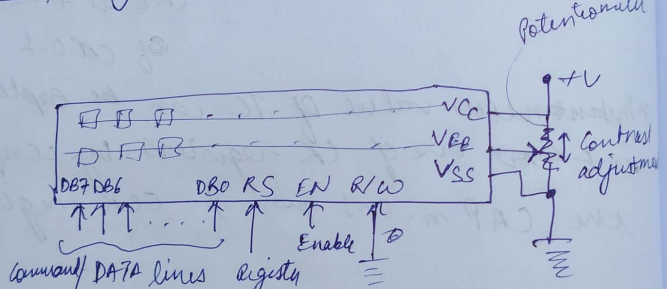
CCR	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																													1	0	0	0

LPC_PINCON → PINSEL3 |= (3 << 22) | (3 << 20)

LPC_TIM0 → CCR = 0x0A; // Capture TC to CR1 at +ve edge of CAP0.1 & CR0 at -ve edge of CAP 0.0

Q what's the use of the new?
diff difference b/w 2 capture registers is the pulse width (can be used to measure pulse width). (Or two if connected to switch - now long the switch is pressed.

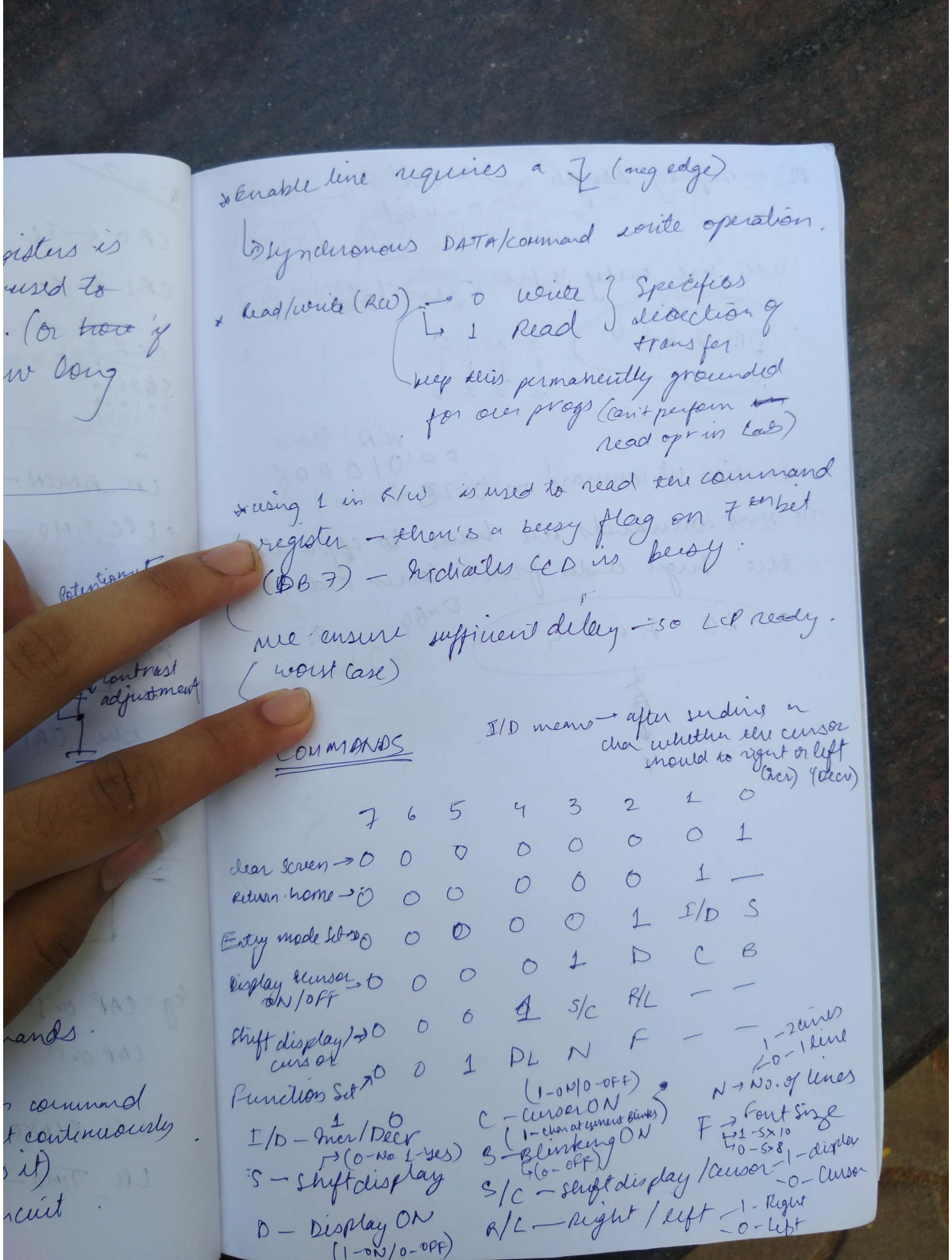
#LCD interfacing



DB7 DB0 (ASCII of chars)
↳ also configuration commands.

*LCD has its own processor. (variation in command register → it continuously executes it)
↳ also has its own refreshing circuit.

*Enable line
Synchrono
Read/write (RC
*using 1 in
register
(DB 7)
we ensu
(worst Case
⇒ COMMAN
clear screen → 0
return home → 0
Entry mode Set → 0
Display sensor ON/OFF → 0
Shift display/ curs or
Function Set
I/D - 1 incr/
S - (0
D - Displ



* Enable line requires a ↓ (neg edge)
  ↳ Synchronous DATA/command write operation.
* Read/write (R/W) → 0 write, 1 Read } Specifies direction of transfer
  ↳ keep this permanently grounded for our progs (can't perform read op in Lab)

* using 1 in R/W is used to read the command register → there's a busy flag on 7th bit (DB 7) — Indicates LCD is busy.
  ↳ we ensure sufficient delay → so LCD ready. (worst case)

COMMANDS

I/D means → after writing a character whether the cursor should go right or left (Incr) (Decr)

	7	6	5	4	3	2	1	0
Clear screen →	0	0	0	0	0	0	0	1
Return home →	0	0	0	0	0	0	1	—
Entry mode Set →	0	0	0	0	0	1	I/D	S
Display Cursor ON/OFF →	0	0	0	0	1	D	C	B
Shift display/cursor →	0	0	0	1	S/C	R/L	—	—
Function Set →	0	0	1	DL	N	F	—	—

I/D — Incr/Decr (1/0)
S — shift display (0 - No, 1 - yes)
D — Display ON (1-ON/0-OFF)
C — Cursor ON (1-ON/0-OFF)
B — Blinking ON (1 - char at cursor blinks, 0 - OFF)
S/C — shift display/cursor — 1 - display, 0 - Cursor
R/L — Right/left — 1 - Right, 0 - Left
DL — 1 - 8 bits, 0 - 4 bits
N — No. of lines (1 - 2 lines, 0 - 1 line)
F — Font size (1 - 5×10, 0 - 5×8)

DL — display length — 1 - 8 bit
0 - 4 bit

We use only 4 bits in this lab.

DB 7 6 5 4

3 2 1 0
1 1 1 1

0 0 1 0 1 0 0 0
DL N

function set command — 0x28

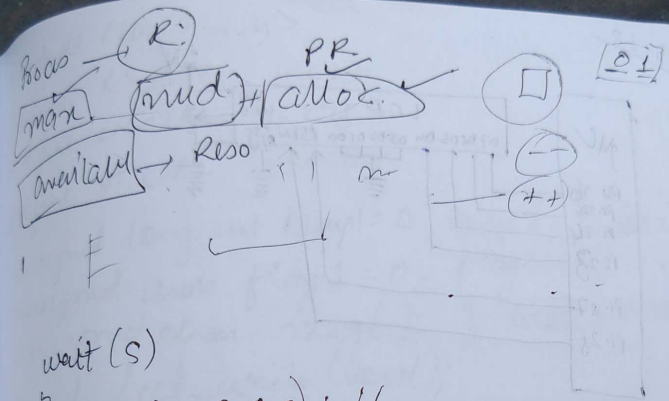
To send commands we have to separate the higher order from lower order

0x64

DB 7 6 5 4

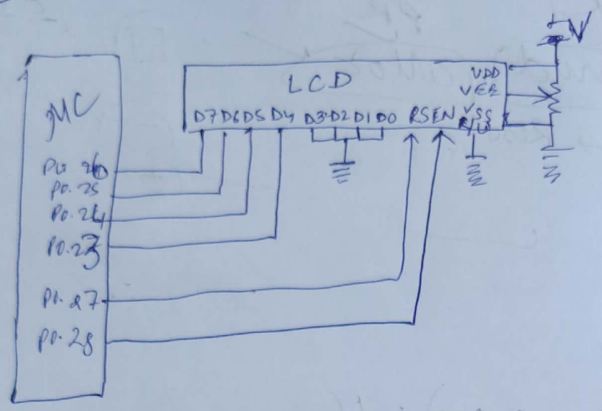
6

4

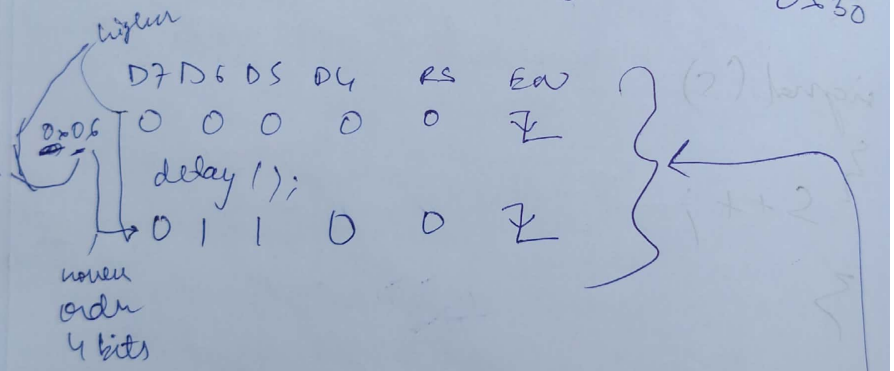



```
wait (S)
{ (while S<=0); //
    S--;
}

signal (S)
{
S++;
}
```

12.3.19



* By default LCD works in 8 bit mode.
  ↳ 8 bit command to send thrice: 0x30
  with delay. 0x30
  0x30



* Send command 0x20 to LCD to work in 4 bit mode.

∴ 0x30 (8 bit)
0x30 (8 bit)
0x30 (8 bit)
0x20 (4 bit) configured into 4 bit now
~~0x28 (function set)~~
already grounded in Lab

* then send commands like higher order then lower order.

```
#include <lpc17xx.h>
#define RS_CTRL 1<<27
#define EN_CTRL 1<<28
#define DT_CTRL 15<<23
unsigned long int temp1=0, temp2=0, i, j;
unsigned char flag1=0, flag2=0;
unsigned char msg[]= {"WELCOME"};
void lcd_write (void);
void port_write (void);
void delay_lcd (unsigned int);
unsigned long int init_command []

= {0x30, 0x30, 0x30, 0x20, 0x28, 0x0C,
0x06, 0x01, 0x08};
                                   2 lines
                                   small font
  clear   after clear back to
          1st line 1st char.

int main (void)
{ SystemInit();
SystemCoreClockUpdate();
LPC_GPIO0->FIODIR = DT_CTRL | RS_CTRL |
                    EN_CTRL;

flag1 = 0; //Command

for(i=0; i<9; i++)
{ temp1 = init_command[i];
```

```
lcd_write();
}
flag1=1; // Data
i=0;
while(msg[i]!='\0')
{
temp1 = msg[i];
lcd_write();
i+=1;
}
while(1); //Infinite
}

void lcd_write(void)
{
flag2 = (flag1==1)?0:((temp1==0x300)||(temp1==0x20)
                      ?1:0; // For commands 0x30 & 0x20 flag will be 1.
temp2 = temp1 & 0xf0;
temp2 = temp2 << 19; //23-26 (higher)
port_write();
if(!flag2)
{
temp2 = temp1 & 0x0f;
temp2 = temp2 << 23; //23-26 (lower)
```

```
port_write();
}

}

void port_write (void)
{
LPC_GPIO0->FIOPIN = 0; // clear IO lines
LPC_GPIO0->FIOPIN = temp2;
if(flag1 == 0)                          // P0.27=0
    LPC_GPIO0->FIOCLR = RS_CTRL; // Command
else
    LPC_GPIO0->FIOSET = RS_CTRL; // for data
                                 // P0.27=1

LPC_GPIO->FIOSET = EN_CTRL;
delay_lcd(25);
LPC_GPIO0->FIOCLR = EN_CTRL;
delay_lcd(5000); // Because not monitoring
                 // Busy flag, lcd needs
                 // time to respond.
}
delay_lcd (unsigned int r1)
{ unsigned int r;
for(r=0; r<r1; r++);
}
```

on P0.28

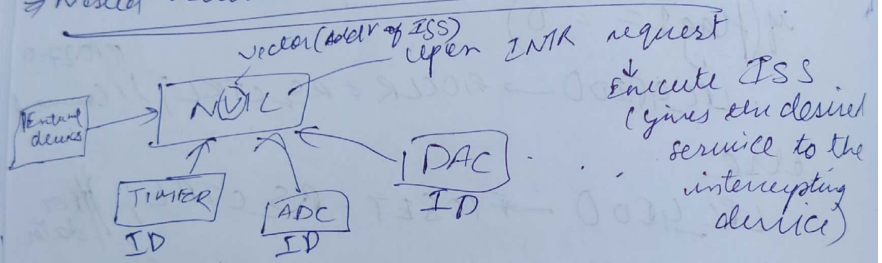
14.3.19

Interrupts

Request signal from on chip peripheral devices or external devices.

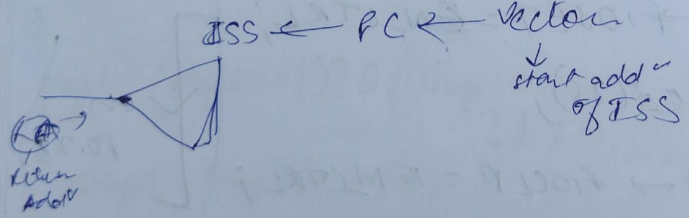
* ISS → Interrupt Service Subroutine → whenever the request is sent (interrupt signal) to CPU, the process executes a fn called ISS to perform the desired service for the interrupt device

⇒ Nested Vectored Interrupt controller (NVIC)

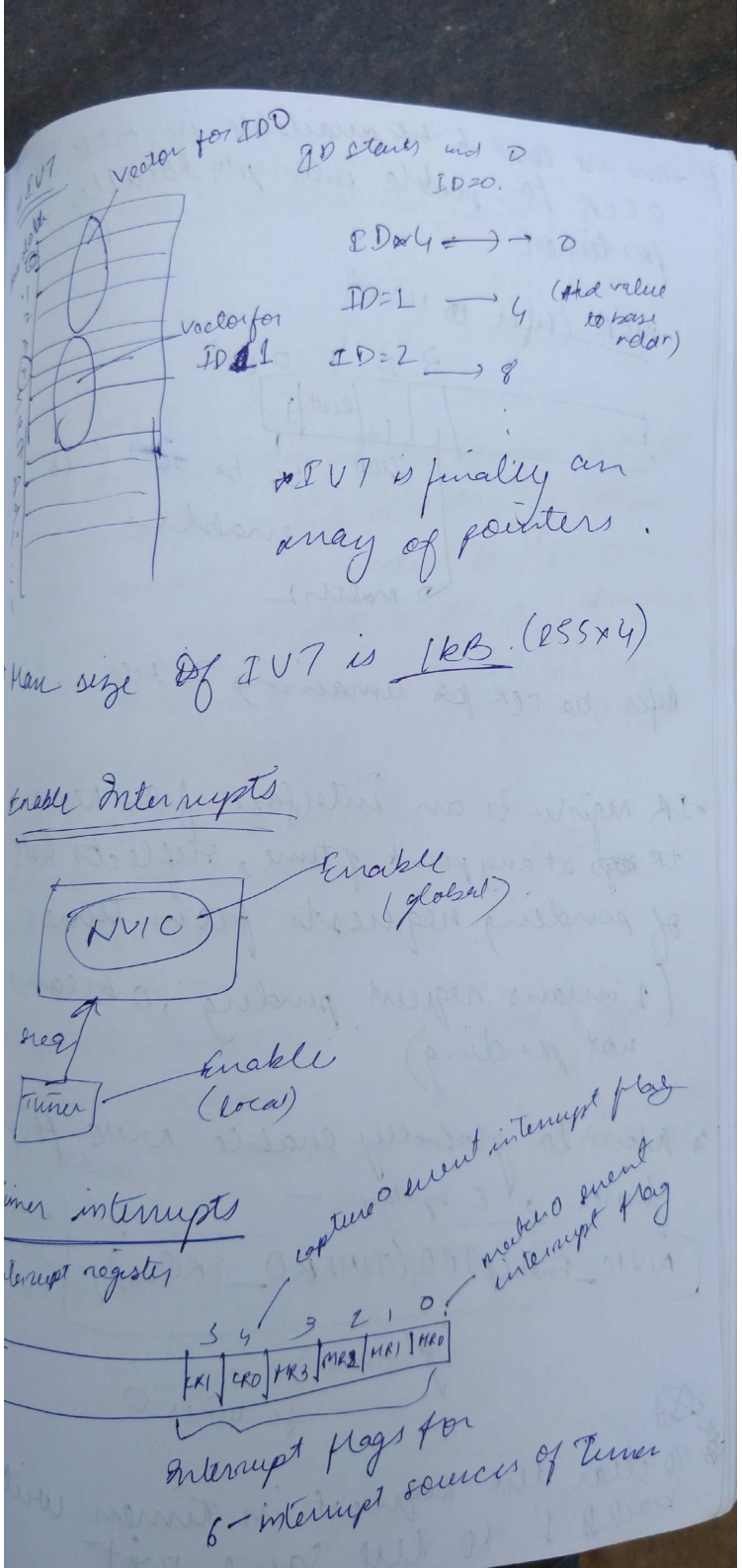


reg counter

* PC maintains the next instruction to be executed

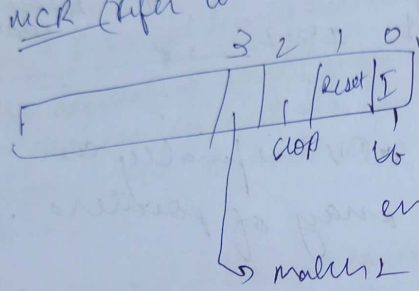


* can have upto 255 ISS (255 IDs)
* when interrupt is called, NVIC takes the ID & multiply ID by 4 (ID * 4)
After getting the value, the system maintains a table called IVT (Interrupt Vector table)



there are total 6 bits available in MCR to CCR to enable interrupts locally for timer

MCR (refer to MCR)



Refer to CCR for remaining 2 bits.

* IR register is an interface for NVIC. IR at any point of time, reflects bits of pending requests from timer (1 means request pending, 0 means not pending)

* How to globally enable NVIC for timer, (syntax :—

```
NVIC_EnableIRQ(TIMER0_IRQn);
```

(global)
for timer 0

* To clear the request in timer write back '1' to the same Bit

Timer 0 ISS.

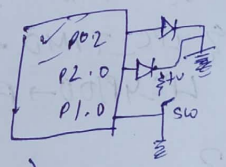
void TIMER0_IRQHandler(void)

{

}

Q.3.19

Write a program to toggle an LED connected ~~P0.2~~ to P0.2 every second, while displaying states of the switch connected to P1.0 on the LEDs connected to P2.0. Assume $P_{clk} = 3$ MHz



#include <lpc17xx.h>

void TIMER0_IRQHandler(void);

void timer0_init(void); // will locally enable in this

void main(void)

{

LPC_GPIO0 → FIODIR = 1<<2; // P0.2 O/P

LPC_GPIO2 → FIODIR = 1<<0; P2.0 O/P.

timer0_init();

NVIC_EnableIRQ(TIMER0_IRQn); // Timer0 INTR enable in NVIC

while(1)

{

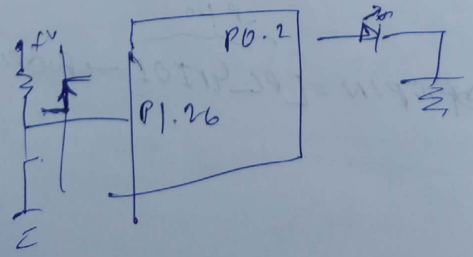
LPC_GPIO2 → FIOPIN = LPC_GPIO1 → FIOPIN[0x01]; P1.0 P2.0

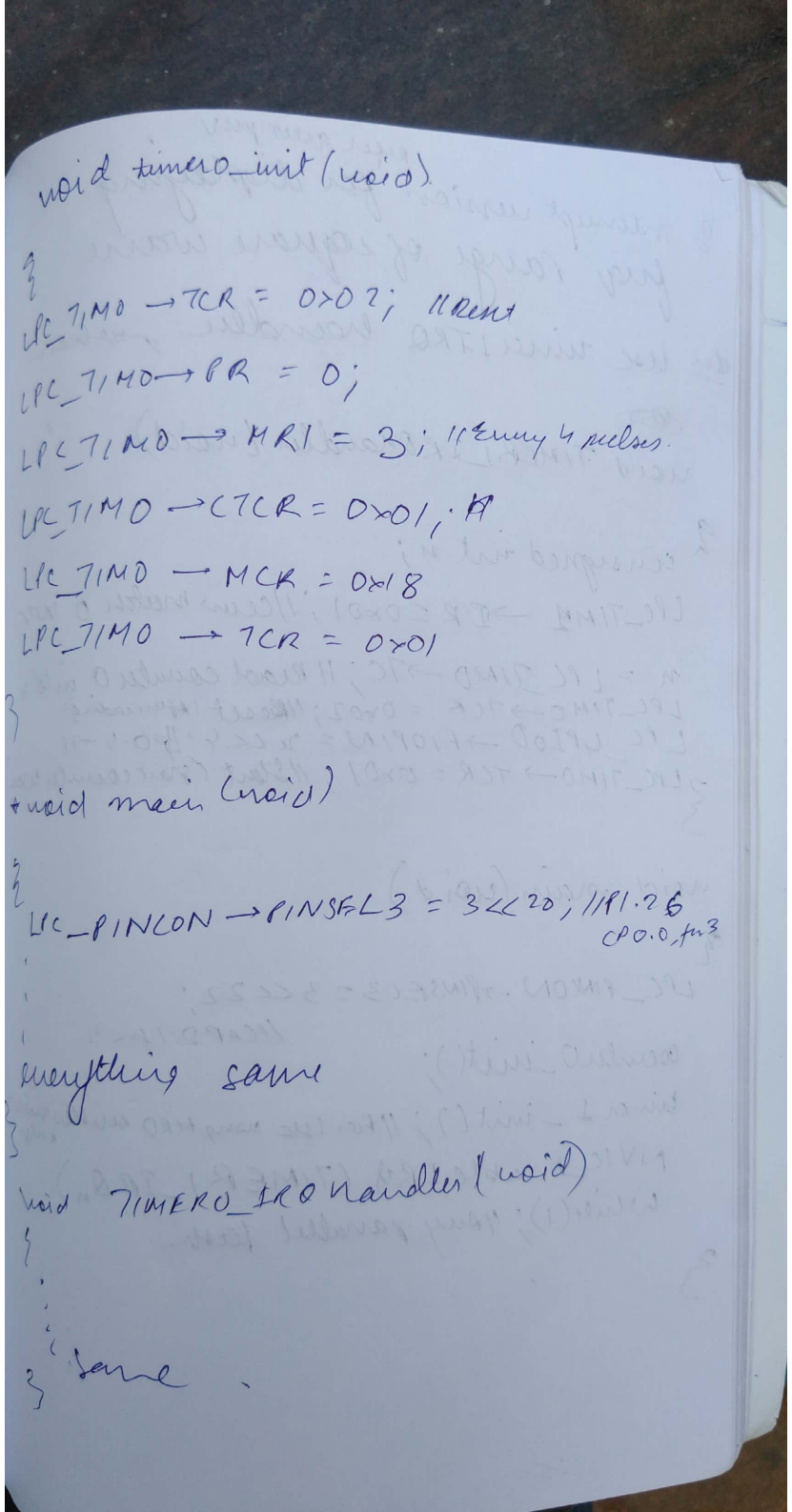
}

}

```
void timer0_init(void)
{ //EMR not required as it is for pooling (No pooling here)
  LPC_TIM0 → TCR = 0x02; //Reset
  LPC_TIM0 → PR = 0;
  LPC_TIM0 → MR1 = 3000000 - 1; //1sec
  LPC_TIM0 → CTCR = 0; //Timer
  LPC_TIM0 → MCR = 0x18; //Generate INTR upon match and reset
                         //current locally enabling the interrupt
  LPC_TIM0 → TCR = 0x01; //Start the timer
}

void TIMER0_IRQHandler (void)
{ LPC_TIM0 → IR = 0x02; //Clear INTR
  LPC_GPIO0 → FIOPIN = ~(LPC_GPIO0 → FIOPIN & 0x04);
}
```

Q. write a program to Toggle P0.2 for every 4 pulses applied on CAP0.0 (P1.26 fn 3) while displaying the states of the switch connected P2.0 on the LEDs connected to P2.0.





```
void timer0_init(void)

{
LPC_TIM0 -> TCR = 0x02;  // Reset
LPC_TIM0 -> PR = 0;
LPC_TIM0 -> MR1 = 3;  // Every 4 pulses.
LPC_TIM0 -> CTCR = 0x01;  // T
LPC_TIM0 -> MCR = 0x18
LPC_TIM0 -> TCR = 0x01

}

void main(void)

{
LPC_PINCON -> PINSEL3 = 3<<20;  // P1.26
                                   CP0.0, for 3
.
.
.
everything same
}

void TIMER0_IRQHandler(void)
{
.
.
.
same .
}
```

Q: Interrupt version for displaying freq range of square wave

Q: Use timer IRQ handler

```
void TIMER1_IRQHandler(void)
{
unsigned int n;
LPC_TIM1 -> IR = 0x01; // Clear match 0 INTR
n = LPC_TIM0 -> TC; // Read counter 0 into n
LPC_TIM0 -> TCR = 0x02; // Reset (after reading)
LPC_GPIO0 -> FIOPIN = n << 4; // P0.4 - 11
LPC_TIM0 -> TCR = 0x01; // Start (start counter again)
}

void main(void)
{
LPC_PINCON -> PINSEL3 = 3 << 22;
                    // CAP0.1 fn 3
counter0_init();
timer1_init(); // For 1 sec using MR0 enables match interrupt
NVIC_EnableIRQ(TIMER1_IRQn)
while(1); // Any parallel task.
}
```