MicroController

single Chip (SOC) -> System on Chip

RISC - Reduced Instruction Set Computer

CISC - Complex Instruction Set Computer

RISC	CISC
1) Fixed Instruction size size always same	x86 can be 1, 2, 3 or even 5 bytes
2) Large number of registers All RISC CPUs have 8 or 16 registers.	
3) It employs load store architecture.	
4) Small Instruction set	Complex instructions can be performed directly
5) 99 % instructions in one clock cycle	More Variance
6) Harvard Architecture	Von Neumann Architecture

$IO/\overline{M} = 1$ } Memory will decide
$\overline{IO}/M = 0$ } IO will decide

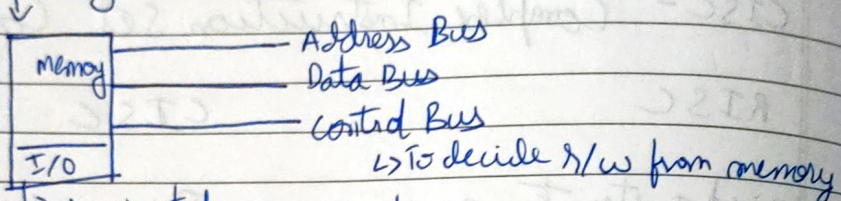
papergrid
Date: / /

→ CISC

IO mapped IO (Isolated I/O) - In which we have common data bus for I/O and memory but separate read and write control lines for I/O

→ RISC

Memory Mapped I/O - Every bus in common due to which the same set of instructions work for memory and I/O.



↳ Uses part of memory space (Exclusive Address) so $IO/\overline{M}$ not required

Drawback : Memory space reduced

ARM (Advanced Risc Machine)

Bits - 31 30 29 28

N Z C V

↓ ↓ ↓ ↓ → For signed operation to identify false - ve.

Sign Zero carry Overflow flag

CPSR - Current program status register (Flag in 8086)

- Program Counter - Points to next instruction in memory
- Stack Pointer - Points at top of stack.

Return Address of function stored in link register

~~When~~

Addressing Modes

1) ~~Register~~ The way in which an operand is specified in an instruction called Addressing Mode.
↓
☆ Defined for operand

1) Register - MOV R0, R1
2) Immediate - MOV R0, #1 → constant embedded in instruction
3) Register Indirect - LDR R0, [R1] → Load
STR R0, [R1] → store
4) Indexed → extended version of Register indirect

→ R0 gets value at address (R1) → Pointer
STR - Store Register → store R0 at [R1] location
LDR - Load Register

	Indexed Addressing Mode	Syntax	Pointing location in memory	Rm content after exec
1)	Preindex	LDR Rd, [Rm, #K]	Rm + #K	Rm
2)	preindex with write back	LDR Rd, [Rm, #K]!	Rm + #K	Rm + #K
3)	Postindex	LDR Rd, [Rm], #K	Rm	Rm + #K

#K can be signed value b/w -4095 to 4095

LPC1768

papergrid

Date: / /

Endianess - The order of bytes within a word of digital data stored in computer memory

Two types:

i) Little-Endian (LE) → Byte (smallest) - System stores LSB bit in lower memory address

Both LE and BE

Ex: Intel, ARM (Exception - M3, M4, default - LE)

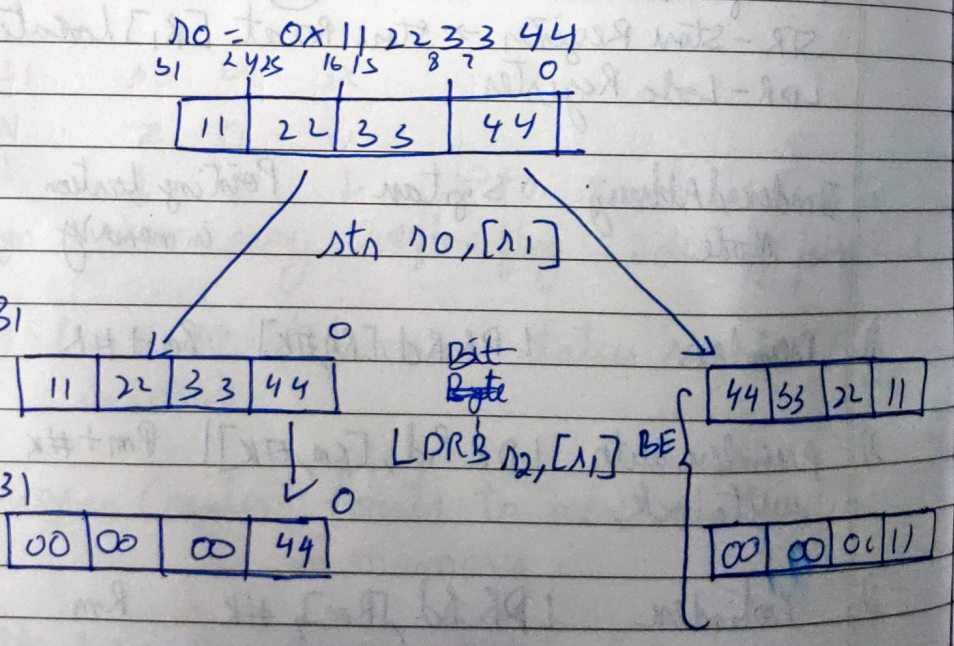
ii) Big Endian - stores LSB in greatest memory address

MSB in smallest → Byte

Ex: Motorola

Endianness is relevant when data is stored as word and accessed in smaller quantities

Ex:



Move Instructions:

- MOV: Move Data - only N and Z flag affected
- MOVW: word data to lower 16 bits
- MOVT: word to upper 16 bits
- MVN: Move NOT of the value to register
- MSR: Move data from General Purpose to Special function
- MRS: Move special function to general purpose

Eg:- MOV R1, #0xFA05 → Denotes constant value, flags not updated
(Hexa, Decimal)

- MOV R3, #23
- MOV R8, SP → stack pointer value
- MOVS R11, #0x000B → Updates flag only N & Z, C & V unaffected
- MVNS R2, #0xF → 0 0000000F negate it → 0xFFFFFFF0 (Negate)
- MOVW R7 #0x1234

R7 = | 00 | 00 | 12 | 34 |

- MOVT R5, #0xF123

R5 = | ~~00~~ | F1 | 23 | 00 | 00 |

3 types of load-store-

1) Word : LDR/STR
2) Half-Word: LDRH/STRH
3) Byte : LDRB/STRB

- For signed add S
- All LDR, STR instructions can be executed conditionally

★ <LDR/STR>{<condition>}{<size>} Rd, <address>

Multiple Register Data Transfer (LDM/STM)

★ <LDM/STM>{<condition>} Rd{!}, <register>

Note:
LDM R0, {R1-R3} · R0 ← R0 after execution
LDM R0!, {R1-R3} · R0 ← R0+12 after execution
↳ Each register 4 blocks

ARM - Instruction Set

Arithmetic Instructions

Syntax : <Operation>{S} Rd, Rn, Operand2
↳ Optional to update flags

Operations:

1) ADD
2) ~~ADDC~~ ADC - with carry
3) SUB
4) SBC - Sub with carry
5) RSB - Reverse Sub
6) MUL
7) UDIV - Unsigned div

Eg: ADD $R_1, R_0, R_2 \Rightarrow R_1 \Rightarrow R_0 + R_2$
ADD $R_1, R_0 \rightarrow R_1 \Rightarrow R_1 + R_0$
ADD $R_1, \#56 \rightarrow R_1 = R_1 + 36$
ADD Rd, Rn, Rm ∴ ~~No carry~~

ADD also uses carry when adding 2 numbers however ADC takes carry at start itself

ADC $S_1, S_2,$ (Cy) $\cdot S_1 = S_1 + S_2 + Cy$

'S' → N, Z, C, V will be updated

→ 0X1234
ADD $R_3, R_0, \#0X111$ $R_3 = 0X1234 + 0X0111$
$= 0X1345$
C flag
→ 0010

LDR $R_4, \#0X20000000$
MSR xPSR, R_4 → C = 1

LDR R8, #0X7FFFFFFF
LDR R9, #0X70110011
ADCS R10, R8, R9 ; N=1, V=1 as we have add yields -ve result
so we need to take 2's complement for proper result only in case of signed operation.

- BNE → Branch on not equal
- NE condition
  - ↳ $R_1 = 0X1234$
    $R_2 = 0X1111$
    $R_1 - R_2 = 0$
    ~~Eq~~ → EQ → Z = 1
    $R_1 - R_2 \neq 0$
    NE → Z = 0
- LDR R_0, =value (Location) value = 0x00000001
  $R_0 = 0XFFFFFFFF$ ↳ Loc = 0XFFFFFFFF
- LDR R0, [value]
  $R_0 = 0X00000001$

Subtraction Instructions

SUB Rd, Rn, Rm Rd = Rn - Rm
RSB Rd, Rn, Rm Rd = Rm - Rn

Logic for cy flag

→ Cy=1 ⇒ Bigger - Smaller

```
  3
- 2
-----
Cy=1 N=0
```

→ cy=0 ⇒ Smaller - bigger

```
  2
- 3
-----
Cy=0, N=1
```

→ cy=1 ⇒ when we subs 2 equal nos

SUB → 2's complement Add

SBC dest, S1, S2 → In case of SBC

DST = S1 - S2 - 1 + Cy

```
  3  4  2
- 3  3  3
---------
  0  0  9
```

1)
```
  2 → 12        4
- 3           - 3
---------     -----
  9 cy=0        4-3
```

Cy=0

R0 = 0x9 , R1 = 0x2 , SBC R4, R0, R1

R4 = 0x6

as R4 = 9 - 2 - 1 + cy

CMP R0, R1 , if R0 > R1 ⇒ Cy=1

Multiplication Instructions

- Integer Multiplication (32 bit)
- Long integer mul (64bit)
- Multiplication Accumulate:
  Add product to running total

MUL

~~MULA~~ MLA - Multiply Accumulate x+ runtotal
UMULL - Unsigned Long multiply
UMLAL - Unsigned Long multiply with accumulate
SMULL - Signed long
SMLAL - Signed long with accumulate

Multiplying 2 32-bit nos gives 64 bit result
but MUL only stores lower 32 bits

MLS Rd, Rm, Rs, Rn ⇒ Rd = Rn - (Rs × Rm)
MLA Rd, Rs1, Rs2, Rs3 ⇒ Rd = (Rs1 × Rs2) + Rs3

UMULL RdLO, RdHi, Rn, Rm ; RdHi (32 bit) ; RdLO (32 bit) = Rm × Rn

SMLAL RdLO, RdHi, Rn, Rm . RdHi : RdLO =
(Rm × Rn) + (RdHi : RdLO)
↳ Signed ~~mult~~ mul

Flags not affected

LDR R_2, =-4 (allows 32 bit to be copied)

papergrid

Date: / /

UMULL R1, R0, R2, R3 (R1 = RdLo, R0 = RdHi; R2 = 3, R3 = 4)

$R_2 \leftarrow$ 0X00000003

$R_3 \leftarrow$ 0X00000004

12 → 0C (in Hex)

0X0000 0000, 0000 000C

R0 | R1

SMULL R0, R1, R2, R3 = 5 (R2 = -4)

R_2 when stored as 2's complement ⇒ 4 = 0100 (last 4 bits) → 1011 → 1100 → 2's complement

R_2 = 0XFFFFFFFC

- 4×5 = 20

20 hex = - 0X14

↓

- 0X0000 0000 0000 0014

0001 0100

1110 1100

After 2's → 0XFFFF FFFF, FFFF FFEC

R1 | R0

SMLAL R2, R1, R3, R4 - 3 (R2 = -2 / 7FE, R1 = -1 / 8F, R3 = -4 / 7FC, R4 = 3)

0010 0001   -12 =   7F   C 1100

~~F...E~~ (R1 : R2) + R3×R4 = -14 (18)

(-12 + -2 = ↓ Store as 64)

0XFFFF FFFF FFFF FFFE

↓ 2's complement

= -2

-14 = 2's complement of - 0X0E

0X0000 0000 0000 000E

1's → 1110 → 0010

0XFFFF FFFF FFFF FFF2 = -14

R_1 R_2

Logical Instruction

reg reg reg/immediate

- AND Rd, Rn, OP2 ; Rd = Rn ANDed OP2
- ORR Rd, Rn, OP2 ; Rd = Rn ORed OP2
- ORN Rd, Rn, OP2 ; Rd = Rn ORed with 1's comp of OP2
- EOR Rd, Rn, OP2 ; Rd = Rn XORed OP2

Flags not affected
Use S suffix to change flags

TEQ Rn, OP2 ; Performs Rn XOR OP2
↳ Result is nowhere stored only flags NKZ affected

TST Rn, OP2 ; Performs Rn AND OP2

Ans: i) $R_0 = 0 \times 46$
$R_1 = 0 \times 46$

TEQ R_0, R_1 $\quad R_0 \oplus R_1$

0100	0110
0100	0110
0000	0000

Zflag = 1

TEQ $R_2, \#0$ → used to check if $R_2 = 0$
for loops as an exit condition

ii) $R_0 = 0x05$ → checks if bit 0 is 1 or not, Zflag = 1
TST $R_0, \#1$ → 0000 0101
0000 0001
0000 0001 → Nonzero

If bit 0 = 0 then Zflag = 1

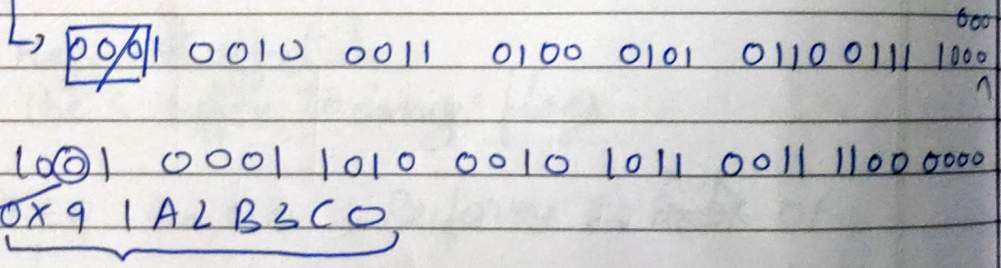
TST used to check if a bit position is 1 or 0.

Shift and Rotate

- LSL Rd, Rm, Op2 → Logical shift left
- LSR Rd, Rm, Op2 → MSB is shifted in from right
- ASR Rd, Rm, Op2 → Arithmetic shift right
- ROR Rd, Rm, OP2 → No 3rd operand
- RRX Rd, Rm → Rotate right extended through carry
  - ↳ Rm by 1 bit position

Flags not affected
Use S suffix to update flags

eg: if R2 = 0X12345678
LSL R3, R2, #3
↑ shift left R2 #Times



Most recent shift goes to cy flag if S suffix used

ii) $R_2 = 0X12345678$

LSR $R_2, \#5$

↳ R_2 shifted 5 times from right

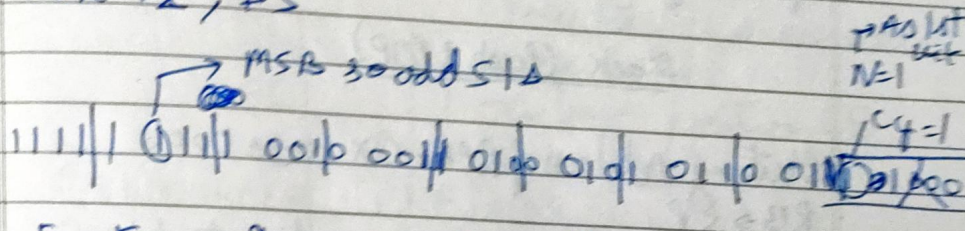
$R_2 = 0X00091A2B3$

$Cy = 1$, 5th bit from ~~left~~ right

iii) $R_2 = 0XF2345678$

ASR $R_2, \#5$

MSB 30 add 5 1s

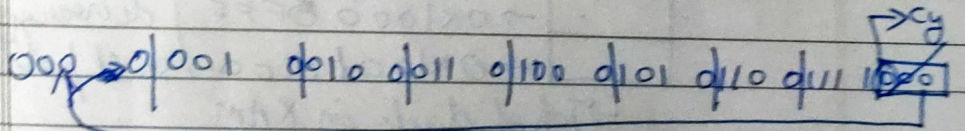


F F 9 1 A 2 B 3

$\Rightarrow 0XFF91A2B3$

iii) $R_2 = 0X12345678$

ROR $R_3, R_2, \#3$



$= 0X02468ACF$

iv) ~~$R_2 = 0X12345678$~~ $R_2 = 0X39864F33$

RRX R_3, R_2 Rotate right 1 bit

To set Cy flag as 1

LDR $R_0, \#20000000$ → Cy flag

MSR $XPSR, R_0 \rightarrow 29\,00\,10$

↳ Register to special function

MRS - special function to register

Now Cy=1

RRX R3, R2

1 0011 1001 1000 0110 0100 1111 0011 001X → 0 = Cy

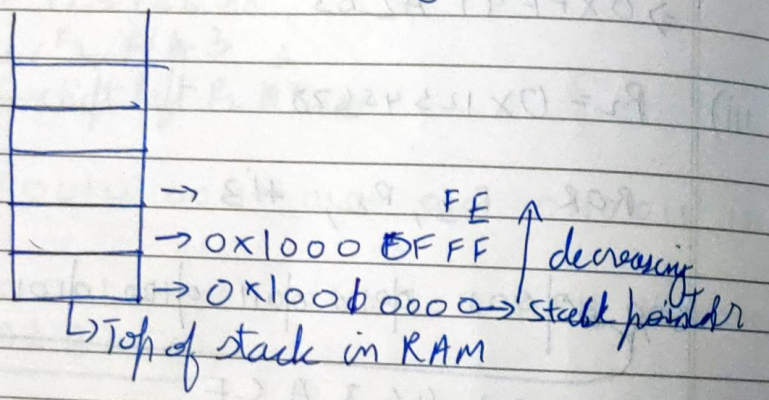
Now cy again becomes 1

If S suffix used logic to cy otherwise cy=0

Stack (Push & POP)

Starting Address of RAM - 0X1000 0000
R13 → SP
stack is stored in ram (descending stack)

eg:



- Push registers onto and pop registers off a full descending stack

- Push and Pop are synonyms for STMDB and LDM

Eg: PUSH {R0, R4-R7}; Push R0, R4, R5, R6, R7 onto stack

PUSH {R2, LR}; Push R2 and link register onto stack

- Storage is Little Endian

ii) R1 = 1, PUSH R1 → Each register is 32 bit
↓ memory → 8 bit
Therefore 4 shares occupied

FC	1 → In lower position LE
FE	0 as 1 ⇒ 0001
0X10000FFF	0
0X10001000	0

To learn:

- POP {R6, R7}

Branching Instructions (N, Z, C, V flags used to check)

- LR (Link Register) used for branching
- Program counter loaded to label on using branch instructions
- LR = Stores address of next instruction after branch
- Then instructions change order of execution
- Types of conditional flow:
  i) Conditional BEQ, BNE
  ii) Unconditional
  iii) Branch with Link

Bubble Sort Program

- Initialize numbers with 4 byte boundary
- Itialize a counter in one of the registers
- Initialize a counter for inner loop by subtracting 1.
- Load 2 numbers into registers for comparison
- CMP >>, 50 → C=0 as 25<50
- If C=1 Skip
- Otherwise swap using STR R2 to [R4], #4 (NO; position in memory)
- Decrement inloop counter by 1 till not 0.
- Then subtract out loop by 1. till out loop is not 0.

Hex Conversions

0-9 → 30-39 } ASCII Code
A-F → 41-46 }

0-9 Subtract 30 to get Hex Value for numbers.
A-F Subtract 37 to get Hex value for A-F
10-15

Hex to ASCII conversion

1 = 31

$D = (D)_{16} + (37)_{16} = (13)_{10} + 2(55)_{10} = (68)_{10} = (44)_{16}$

ASCII to hex

$36 \Rightarrow 36 - 30 = 6$

$45 \Rightarrow (45)_{16} - (37)_{16} \Rightarrow (67)_{10} - (55)_{10} = (14)_{10} = E$

$\hookrightarrow 16 > 37_{16}$

GCD of 2 numbers - Encryption Algos

Pseudo

```
while (a != b) { if (a > b) a = a - b
    else if (b > a) b = b - a }
```

1) Load both nos to 2 registers
2) Loop till both not equal
3) SUBHI to subtract if higher C=1
4) SUBLO to subtract if lower C=0
5) CMP R0, R1 -> sets carry flag

Fibonacci Series - 0, 1, 1, 2, 3, 5, 8, 13 ...

1) Initialize 0 & 1 in memory / Initialize counter
2) Load both numbers to registers
3) Add and store sum in 2nd register
4) Continue and perform SUBS signed so that on reaching 0, zero flag becomes 1
5) Exit loop on zero flag = 1.

Date: / /

BCD Addition

Hexadecimal Subtraction

$$
\begin{array}{cccc}
9 & 2+16 & 3+16 & 16+11 \\
9 & 13 & 14 & 15 \\
- & 5 & 8 & 7 & C \\
\hline
3 & A & C & F
\end{array}
$$

LDM - Load Multiple registers from memory ✓ flags unaffected
LDM Rm, {Rx, Ry, ...}

STM - Store Multiple copy on memory ✓ flags unaffected
STM Rm, {Rx, Ry, ...}
↳ Pointer to memory (array)

Each number is 32 bits

Use ! for updating pointer address after copying

eg: LDM R1!, {R3, R4, R6}
↳ R1 = R1 + 12 because each number 32 bits or 8 digits ⇒ 4 blocks / bytes

If R1 = 0x1000 0000
LDM R1, {R2, R4, R6}
R1 = 0x1000 0000

LDM R1!, {R2, R4, R6} ~~(only R2)~~
R1 = 0x1000 000C

STM R1, {R2, R4, R6} → stores values in R2, R4, R6 to memory location pointed by R1 incremented in 4s. Use ! after R1 to update R1 address.

R2-R4 notation also allowed - R2, R3, R4

- LDMDB - Load multiple registers and decrement before each access, flags unaffected

LDMDB R_1, {R_2, R_4, R_6} → It starts R_1 from $R_1 + 3 \times 4 = R_1 + 12$ (3)

R_1! ⇒ R_1 value at start R_1 will ~~remain~~ be $R_1 + 12$

- STMDB - Store multiple registers and decrement before each access, flags unaffected

Ex: R_1 = 0x10000000

LDMDB R_1, {R_2, R_4, R_6}

R_1 after execution = 0x10000000

LDMDB R_1!, {R_2, R_4, R_6}

R_1 after exec = 0x10000000

Same in case of STMDB

Fully Ascending Stack → SP increments on push

STM R11!, {R2, R4, R6} → Push 3 times

LDM R11!, {R2, R4, R6} → Pop 3 times

Fully Descending Stack → SP dec PUSH
→ SP incre POP

STMDB R13!, {R_2, R_4, R_6}

↳ PUSH

LDM R13!, {R_2, R_4, R_6}

↳ POP

Recursion Function:

Types:

1) Function calling itself (Direct way)

2) Recursion using mutual function call (indirect way):
Here func A can call B which in turn calls A

Applications: Encryption Algos.
LR => Return Address

Factorial Program

fact(n) = n × fact(n-1)

bl → branch with link
bl factorial → branch to factorial funct.

Reset Handler
v_1 ldr r0, =05
v_2 bl factorial
v_3 stop b stop

factorial → initially lr = v_3
push {r4, lr}
mov r4, r0
cmp r0, #0
bne not_zero
mov r0, #1
b last

not _ zero
sub r0, r0, #1
bl factorial
mov r1, r4
mul r0, r0, r1

last
last hop r4, lr
bx lr.

Division Instructions

- SDIV SDIV Rd, Rn, Rm ⇒ Rd = Rn/Rm

- UDIV Rd, Rn, Rm ⇒ Rd = Rn/Rm
  Rn = 0x10, Rm = 0x5 ⇒ Rd = 0x2

- To get remainder → Subsequent subtraction

BCD to hexa

21 = 0010 1001
↓ ↓
2 9

BCD = 6 5 5 3 5

$6 \times (2710)_{16} + 5 \times (03E8)_{16} + 5 \times (0064)_{16} + 3 \times (000A)_{16} + 5 \times (0001)_{16} = (1FFF)$

$(10000)_{10}$ $(1000)_{10}$ $(100)_{10}$ $(10)_{10}$ $(1)_{10}$

Hexa to BCD

Divide by 10^0, 10^1 ... OA ~~OA~~ → OA ... some way.

Linear Search

sequential search is made overall items

* BNE Next instruction after it loaded to link register

* Division only quotient stored no remainder

Hexa to BCD example

```
OA | FF
OA | 19 Q  -5 R
OA | 2  Q
   OA
```

Perform division until quotient = or less than divisor

Date: / /

Embedded C Programming

Ports → Ext devices (I/O pins) — LED, LCD

5 ports in LPC 1768 (Ports are available as part of microprocessor)

☆ P X . Y → Port x Pin Y

I/O Ports P0[0] to P0[31] / P1[0] to P1[31] / . . . P4[0] to P4[31]

Pin Connect Block - Hardware module used to connect pins to I/O block.

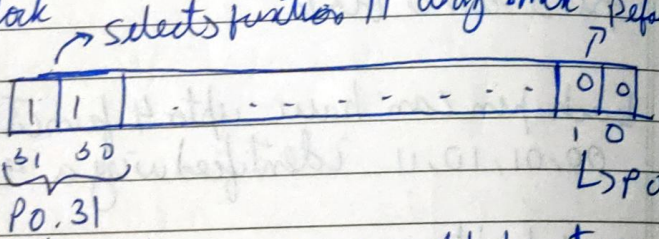
☆ Each pin can have upto 4 functions to select function 00, 01, 10, 11 identified using a 4:1 Mux.

By default 00 primary function, typically GPIO port

- The block has 10 Pinselect registers which are special function registers. (PINSEL0-9)
- These SFRs are used to select 1 of 4 functions
- 2 selection inputs needed to identify for each pin.
- Each Pinsel has 32 bits → 16 x 2 bits to configure all 16 Pins in a port PINSEL

	Register		Controls
Port0	PINSEL	0	P0[15:0]
	"	1	P0[31:16]
Port1	"	2	P1[15:0] (Ethernet)
	"	3	P1[31:16]
Port2	"	4	P2[15:0]
	"	5	P2[31:16]
Port3	"	6	P3[15:0]
	"	7	P3[31:16]
Port4	"	8	P4[15:0]
	"	9	P4[31:16]

PINSEL block



eg: For example to configure P0.1 with function-01 and P0.15 with function-02. for pinsel P(31:0)

The binary value to be loaded will be
$(10000000000000000000000000000100)_2$
$= (0x80000004)_{16}$ → Loaded to PINSEL

Simpler Method → (2<<30)|(1<<2)

→ To solve by simpler method place LSB at 0th position

To bring to P0.1 from P0.0.
1<<2 by shifting left 1 by 2 bits it comes to the right position.

01
P0.0

To obtain 10 at P0.31 assume 10 to be at P0.0 in order to reach correct position needs to be shifted left 30 position.

Final O/P = (2<<30) | (1<<2)

As both have 1s at different positions OR function creates the right final output.

= 2<<(15-0)x2 | (1<<(1-0)x2

buffer bit → LSB pin no.

↓

2 bits for each

In program → I/P Pinsel ← (2<<30) | (1<<2)

Eg: Configure P0.17 with function 3 and P0.30 with function -01 in P0[31:1]

PINSEL1 ← (1<<30-16)x2 | 3<<(17-16)x2

= (1<<28) | (3<<2)

papergrid

Date: / /

GPIO (General Purpose Input / Output) block - OD

Registers Present (SFR) → Determines port ~~no~~ no.

→ FIOx DIR

1) FIODIR - Fast GPIO Port Direction control register. This register individually control direction of each port pin.

P0.0 -> I/P FIOxDIR3 FIOxDIR4 Upper 16 bits → Lower → Lower 16 → Octets

P0.1 → O/P

It can be split into FIODIRH and FIO DIRL < FIOxDIR0, FIOxDIR1

~~Px.0 is input~~ By default it is configured as input as all 0s.

FIODIR is a SFR with 32 bits whenever pin value is set to 1 - it is output for each bit.

Px.0, Px.1 . . . . . it determines pin is ~~an~~ I/P or O/P

Each port has a FIODIR

2) FIOxSET - Fast port output register using FIOMASK

This register is used to O/P logic 1 to the pin.

0 → O/P is unchanged

1 → O/P is set to high

★ ~~On the~~ Only bits enabled by 0 in FIOMASK can be altered.

3) FIOCLR → Same function of FIOSET (Opposite)

0 → Controlled pin unchanged
1 → Pin O/P is set to low.

4) ~~FIOPIN~~
FIOXPIN – FastPort Pin Value registes. Any value can be sent as per choice to pin.

★ It can also be used to I/P data by reading value from pin.

★ If an FIOPIN register is read and its bit is masked with 1 in in FIOMASK then it will be read as 0 regardless.

5) FIOX MASK –

- 1 → Indicates Mask present
- 0 → Mask not present
- It prevents signal from reaching device
- Blocks signal sent by FIOXPIN

Write into FIOSET, FIOCLR, FIOPIN
Read from FIOPIN.

In Send 0xA5 to P0.15-P0.8 without affecting value of ~~lower~~ remaining pins

papergrid

Date: / /

Sol^n

1) FIO0MASK = 0XFFFF00FF
FIO0PIN = 0X0000A500

2) FIO0MASKL = 0X00FF
FIO0PINL = 0XA500
↳ Quarter denoted

3) FIO0PIN1 = 0XA5

☆ FIO0DIR
↳ ARM ⇒ LPC_GIO0 → FIODIR

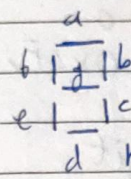
FIO1DIR
↳ LPC_GIO1 → FIODIR

Embedded C Programming

Q1) Write C program to turn ON and OFF LEDs connected to P0.11-P0.4

```
#include <LPC17xx.h>
unsigned int j;
unsigned long LED = 0x00000FF0

int main (void)
{
        SystemInit();                          } core-sensitive
        SystemCoreClockUpdate();               }
                                  → Port 0
  LPC_PINCON → PINSEL0 = 0x00000000; // P0.15-P0.0

  LPC_GPIO0 → FIODIR = 0x00000FF0; P0.11-P0.4

  while(1){

  ON LED { LPC_GPIO0 → FIOSET = LED; // Set P0.11-P0.4
           for(j=0; j<10000; j++); // delay

  OF  { LPC_GPIO0 → FIOCLR = LED; // clear P0.11-P0.4
        for(j=0; j<10000; j++); // Delay

Can also
  →  LPC_GPIO0 → FIOPIN = ~(LPC_GPIO0 → FIOPIN
                          & 0x00000FF0);
  to write this    for(j=0; j<100000; j++); // Delay
        }

}
```

Seven Segment Display



hgfedcba
A-0|1|00|11

h → To activate display

☆ Check a-h for MSB and LSB.

Common Cathode:

'0' - a b c d e f g h →
1 1 1 1 1 1 0 0 → 0xFC

a-MSB
h-LSB

'1' - 0 1 1 0 0 0 0 0 → 0x60

Common Anode:

a b c d e f g h
'0' - 0 0 0 0 0 0 1 1 → 0x03

'1' - 1 0 0 1 1 1 1 1 → 0x9F

- In ES if higher pin number is connected from mc to LED then h is MSB.
- P0.4 - P0.11 are a-h values
- FRC cable is connected

Code:

```
#include <lpc17xx.h>
unsigned char seven_seg[10] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F};
                                0    1     2     3     4     5     6     7     8     9
        L> Hex code for each digit
unsigned int i, j;
void delay(void);
int main(void){
    SystemInit();
    SystemCoreClockUpdate();

LPC_PINCON->PINSEL0 = 0; // P0.4-P0.11->0
                          -> To retain old values
LPC_GPIO0->FIODIR |= 0x00000FF0;
  //P0.4-P0.11 to Output

while(1)
 {
   for(i=0; i<10; i++)
     {
       LPC_GPIO0->FIOPIN = seven_seg[i] << 4;
          L> 32bit wride to store in P0.4-P0.11
        delay();                          To ~~from~~ through array
     }
 }
}
void delay(void)
{
  for(j=0; j<10000; j++);
}
```

P1.23 P1.27 for Segment Selection

papergrid

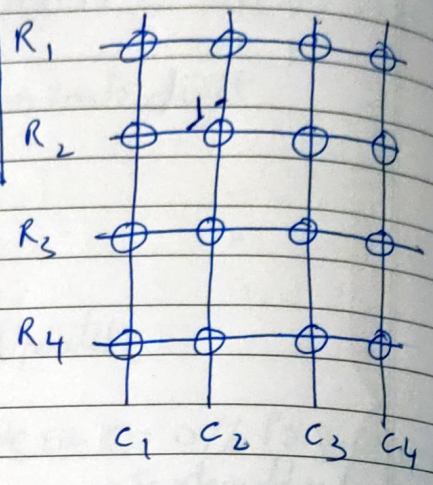
Date: / /

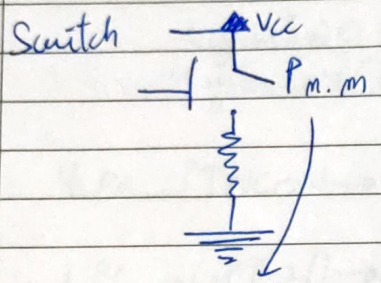
Multiple seven segment LEDs to display 1 2 3 4 on loop one by one.

Hex Keypad Interfacing

Rows - P2.10 - P2.13

Columns = P1.23 - P1.26





When pressed logic-1 (Vcc)
Not ↓ → 0

Using buttons in matrix over 16 individual advantage:

- 16 switches will require more pins.

Individually 16 port pins, here 8 pins

To identify a particular key pressed:

To activate a button make that row and column logic 1.

To find key ~~all columns are height~~ the pressed correct column is made high and then each of the rows are made high individually then checked for high output ~~key~~ key.

Formatrix P1.0 P0.3 . . . P0.0

:

$P_{1.3}$

papergrid

Date: / /

int main(void) {

corresponds to act as I/P

LPC_PINCON → PINSEL0 = 0x FFFFFF0;

LPC-PINCON → PINSEL2 = 0XFFFFFFF0;

corresponds to port 1

configures as GPIO

LPC_GPIO → FIODIR = 0x00

LPC_GPIO1 → FIODIR = 0x 0000000F; ← O/P

~~LPC~~ int flag, other Variable;

while(1) {

// Checking Row

for(row = 0; row < 4; row++)

{ // We make ~~port~~ row as high

LPC_GPIO → FIOPIN = 1 << Row

flag = 0; // f = 1 key is pressed

// f = 0 No key pressed

Scan(); // check collumns one by one for that row

if (flag == 1)

break;

}

if (flag == 1) {

key = 4 x row + colm;

display();

~~switch(~~

~~key~~ case 1 : key 1 => 4x0+2 = 2

case 2 : key A => 4x2+2 = A

case 3 : key D => 4x3+1 = D

:

}

```
void sum()
{
n = LPC_GPIO0->FIOPIN;   -> Port 0 values assigned to n
n = n & 0xF;   -> only first 4 pins used
if(n!=0){
    flag=1
switch(n){
case1 : col1=0
        break;
.
.
.
}
}
```

To change pin assigned n = n & 0xF << Pinno.

Add n >> Pinno

then Dumne code works

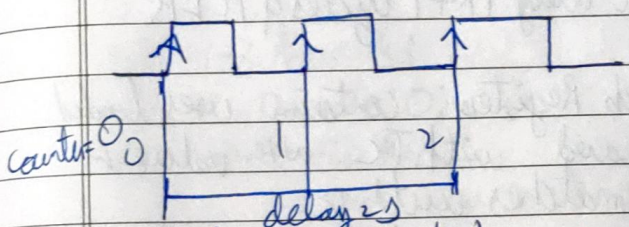
Timer / Counter Programming

Timer - Used to generate intended delay.

Counter - Counting internal events or external clock
↳ Using pulse +ve or -ve edge

There are four 32 bit Timers in LPC1768
Timer0, Timer1, Timer2, Timer3 → All 32 bit
↳ These are modules

☆ How same block can be used as timer and counter



counter increments for every positive edge

- Using frequency of clock and determining counter value
  ↳ T = 1/f → Time period of each positive edge.
  ↓
  Same as counter value

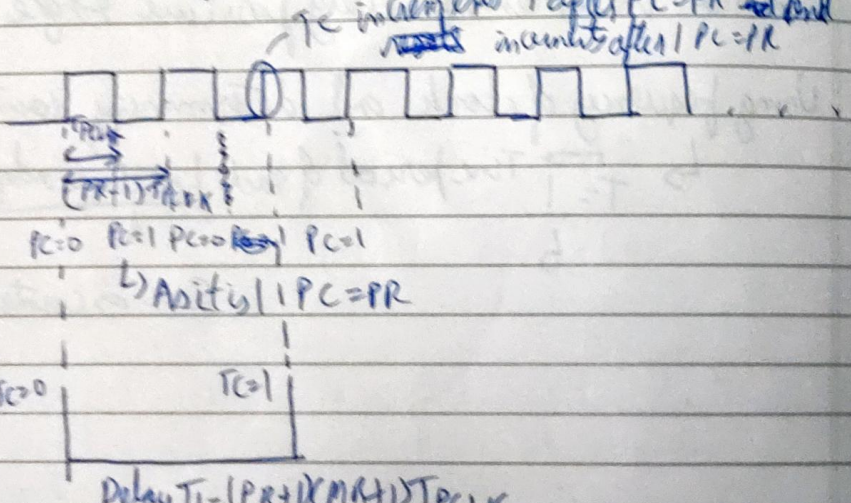
SFR Registers for each timer

PC - Prescale Counter Register → The value of in PC is incremented on every PCLK cycle and when its value matches in PR, the TC is incremented at next PCLK cycle.

PR - Prescale Register → User loaded value : 32 bit register and specifies maximum value for the prescale counter (PC)

TC - Timer Counter register → Incremented when PC = PR
is a 32 bit register and incremented at every PR+1 cycles of PCLK

MR0 - MR3 - Match Registers : contains user loaded value to be compared with TC. When value in match register matches with TC.



From every (PR+1) TPCLK TC increments

MR is used to match with TC.
TC = MR_m is the match event.
Flag is set when match event takes place.

Delay => Total time till match event takes place
$T_d = (PR+1)(MR+1) T_{PCLK}$

~~For 3MHz~~
To get 1s delay, 3MHz T_{PCLK} PCLK $\frac{3\times10^6}{3\times10^6}$
PR = 999, MR = 2999

TCR

Bit 0 → Enable 1
Disable 0
Bit 1 → Counter Reset
1 → Timer value resets TC & PC

CTCR (Counter or Timer control Register)

It is used to select between if Timer and Counter mode and in Counter mode to select the pin and edges for counting

Bit 1:0 Used to select timer or counter
00: Timer mode
01: Counter mode : TC incremented on rising edge
10: Counter Mode: TC incremented on falling edge
11: Counter Mode: TC incremented on both edges

Bit 3:2 Counter clock source select

Clock source is decided using these bits to select b/w CAPn.0/CAPn.1
For every timer 2 CAP lines
00 CAPn.0 for Timer n → Clock Source
01 CAPn.1 for Timer n
Timer - CLK source will be - PCLK (config) - 3 MHz
If used as counter PCLK no longer acts as CLK the CLK acts as external event counter

Match Registers

- The match register values are continuously compared to Timer Counter value.
- The action possibilities are to generate an interrupt, reset the timer counter, or stop timer. Actions are controlled by settings in the MCR register.

Action to be performed when TC = MR

MCR - Match Control Register

Bit	Symbol	Value	
MR0			
0	MR0I (I)	1	Interrupt on MR0
		0	Interrupt disabled
1	MR0R (R)	1	Reset on MR0. TC will be reset
		0	Feature disabled
2	MR0S (S)	1	TC and PC will be stopped and TCR[0] will be 0
		0	Feature disabled

Similarly till 11 bit for MR 3

Bit 12 - Reserved

External Match Register (EMR)

Bit	Symbol	Descr
0	EM0	TC matches (MR0+1)
1	EM1	" MR1
2	EM2	→ Decides output needed at EM0
3	EM3	→ EMC used to determine type of output needed
5:4	EMC0	
7:6	EMC1	
9:8	EMC2	
11:10	EMC3	

EMC bits
- 00 Do nothing
- 01 clear corresponding value at EM 0-3
- 10 set corresponding value at EM 0-3
- 11 Toggle the corresponding bit. complement.

Polling → checking continuously EM0 - EM3 bits

- To get value of EM0-3 to a GPIO pin.
- MAT lines are available for every match register
  - MAT X.Y → Line number
    - ↳ Timer
    - ↳ Used to get EM value to GPIO

MAT0.0
MAT0.1
:

Only for 2

MAT2.0	P0.6 / P4.28
MAT2.1	P0.7 / P4.29
MAT2.2	P0.8
MAT2.3	P0.9
	:

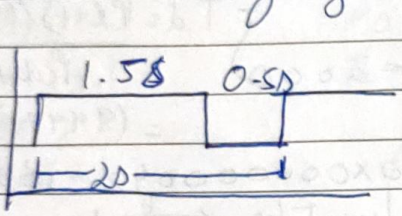
Timer/Counter Program

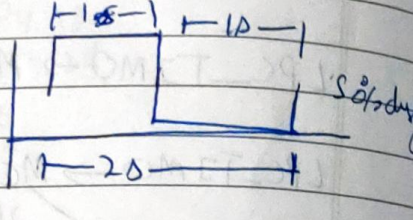
Q. Generate Square wave with period 2s / Toggle LED connected to P0.2 every second.

```
void delay (void)                          → Bit 1 as 1 for reset
{ LPC_TIM0 → TCR = 0X00000002; Timer 0 reset
  LPC_TIM0 → EMR = 0X20 // Set match bit
  LPC                       ↳ EMC0 bits set as 00(10)
                                                  ↓
                            It acts as set function

LPC_TIM0 → PR = 1000;  }  Td = (PR+1)(MR+1) / PCLK
LPC_TIM0 → MR0 = 3000  }     = (999+1)(2999+1) / 3×10^6
                             = 1s
LPC_TIM0 → MCR = 0X00000004;
        3 bits present for match event
Match to be checked for Bits[3:0] => 0(100)  → Stop TC and PC on match

LPC_TIM0 → TCR = 0X00000001; → enable the timer
                  ↳ Bit 0 used to enable timer.
                  Reset bit to be cleared its checked here

while( !(LPC_TIM0 → EMR & 0X01));
            ↳ EMR bit is checked EM0 for 1 value
return;
}
```

```
int main(void){
  LPC_GPIO0->FIODIR = 0x00000004;
  while(1){
    LPC_GPIO0->FIOSET = 0x4;
    delay();
    LPC_GPIO0->FIOCLR = 0x4
    delay();
}
```

Q. Generate square wave of period of 2 seconds with 75% duty cycle & 0.2 s





$\%PC = \frac{TH}{(TH+TC)}$ $\%PC = \frac{1}{2} = 50\%$

```
void delay(){
  LPC_TIM0->TCR = 0x00000002;
  LPC_TIM0->EMR = 0x20;
  LPC_TIM0->PR = 2499
```

$1.5 = \frac{3000 \times (MR0+1)}{5 \times 10^6}$

$MR0 = 1.5 \times 1000 - 1$

$= 1499$

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

For 0.5 = 499 MR values

```
LPC_TIM0->MCR = 0x00000004;
```

```
LPC_TIM->TCR=0x00000001;
while(1(...));
return;

int main(void)
{
    LPC_GPIO0->FIOPIR=0x00000004;
    LPC_TIM0->MR=1499;
     delay()
    LPC=GPIO0->FIOPIN=0x00000000;
    LPC_TIM0->MR0=499 //0.5 seconds
     delay();
}
```

Square Wave form on MAT0.0 → Timer0, 0 match so MR0
EM0 or output pin ↳ O/P line ley tahay ↳ P1.28 with func 3

```
void delay(void) {

    LPC_GPIO0
    LPC_TIM0->TCR=0x00000002;
    LPC_TIM0->CTCR=0x00000000;
    LPC_TIM0->EMR=0x30 //Toggle bit on match
    LPC_TIM0->PR=0; as PC can't be reset
    LPC_TIM0->MR0=3000000;
    LPC_TIM0->MCR=0x00000002 //Reset TC
    LPC_TIM0->TCR=0x00000001;
    return;
}
```

Timer enable ↵

↓ Function only called once

papergrid

Date: / /

```
int main (Void)
{
    LPC_PINCON->PINSEL3 |= (3<<24)
    delay();
    while(1);
)
```

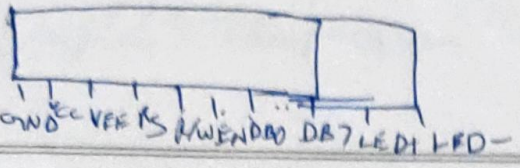
3 is for port/ Get from P1.26 ...

↳ 4th pin 23-16 (12x2)

↳ 2 bits per pin

Q. Square waveform on MAT0.0 by taking EM0 on o/p line

```
void delay(void)
{
    LPC_TIM0->TCR=0X00000002; Timer0 reset
    LPC_TIM0->CTCR=0X00000000;
    LPC_TIM0->EMR= 0X30 //Toggle bit on match
    LPC_TIM0->PR=0;
    LPC_TIM0->MR0=3000000;
    LPC_TIM0->MCR= 0X00000002;
    LPC_TIM0->TCR= 0X00000001;
    return;
}
int main(Void){
    LPC_PINCON->PINSEL3 |=(3<<24);
    delay();
    while(1);
}
```



papergrid

Date: / /

LCD interfacing

- It has 2 rows which can produce 16 characters in each row.
- 1mA needed with no backlight
- Each character made up of 5x8 pixel box.
- It can work in 2 modes 4-bit & 8-bit
- Printed IC in the display to show characters
- Operating Voltage 4.7 - 5.3V
- 16 pins present for interfacing

Pin No	Symbol	Function
1	V_{SS}	Ground
2	Vdd	+5V
3	V_O	LCD contrast adjust
4	RS	Register Select
5	RW	Read / write
6	E	Enable
7	DB0	Data Bit 0
8	DB1	Data Bit 1
9	DB2	Data Bit 2
10	DB3	Data Bit 3
11	DB4	Data bit 4
12	DB5	Data bit 5
13	DB6	Data bit 6
14	DB7	Data bit 7

RS → 0 = Data copied to command Register
1 = Data copied to data register

R/W → 0 = Write to LCD
1 = Read to data
↳ Read operation is used to understand state of LCD
MC waits until R/W becomes 0 inorder to write

After every command MC should wait 37 μs

E → 1 = Enable
0 = Disable

8 lines of data
8 bit D0-D7
4 bit D4-D7
↳ First send MSB then LSB

D5=D7 P0.23 - P0.26
RS = P0.27
E = P0.28
R/W = ⏚

// Komplete this program

papergrid

Date: / /

```
# define RS 27 //P0.27
# define EN 28 // P0.28
# define DT 23 // P0.23 - P0.26 data

int temp1, temp2=0, i, j;
char flag1 = 0, flag2 = 0;   => flag1 = check which LCD
                                register to be used
                       flag2 = 8bit or 4bit
char msg[] = {"Department of ICT Manipal"};
                                        Sleeh mode
                                        aware the
                                        board
long int init_command[] = {0x30, 0x30, 0x30, 0x20, 0x28,
                           0x0C, 0x06, 0x01, 0x80}
        No of data lines 8 or 4 to be selected    func select    1st line
                  Display on/off    mode set    clear display

int main(void) {
    SystemInit();
    SystemCoreClockUpdate();
    LPC_GPIO0 -> FIODIR = 1<<RS | 1<<EN | 0xF<DT
    flag1 = 0; // flag0 all are command

    for(i=0; i<9; i++)
        {
          temp1 = init_command[i]; // All 9 commands
          lcd_write();                to be executed
        }
        flag = 1; // Now data to be transferred

i=0;
while (msg[i] != '\0') {
        temp1 = msg[i]; char by char
```

```
lcd_write();
i++;
if(i==16) { flag1=0; // check for 1 character in first line
   temp1=0xc0; // goto your 2nd line
   lcd_write();
   flag1=1; // Data can be displayed
   Delay16(1); }
```

First 16 characters displayed

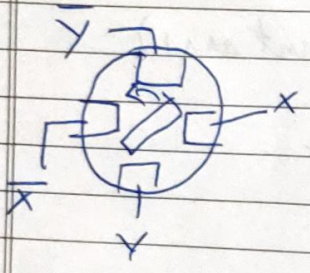
```
void LCD_write(void) {
   flag2=(flag1==1)?0:((temp1==0x30)||(temp1==0x20))?1:0;
   temp2=temp1&0xf0 // move data (26-8+1) times : 26-HN
         Move 4-bits to extract MSB and then LSB
      as needed to send 4 bits at a time
   temp2=temp2>>4; // Move to LSB
port_write();
if(!flag2) {
   temp2=temp1&0x0f; // 26-4+1
   temp2=temp2<<D7
   port_write();
   }
}
```

```
void lcd_write(void)
{
  LPC_GPIO0->FIOPIN=0;
  LPC_GPIO0->FIOPIN= temp2; //MSB
     if(flag==0)
        LPC_GPIO0->FIOCLR=1<<RS;
     else
        LPC_GPIO0->FIOSET=1<<RS;

  LPC_GPIO0->FIOSET=1<<EN;
  delay_lcd(25);
  LPC_GPIO0->FIOCLR=1<<EN;
  delay_lcd(30000); // 3ms highest delay
}
void delay_lcd(unsigned int n1)
{ int n;
   for(n=0;n<n1;n++);
   return;
}
```

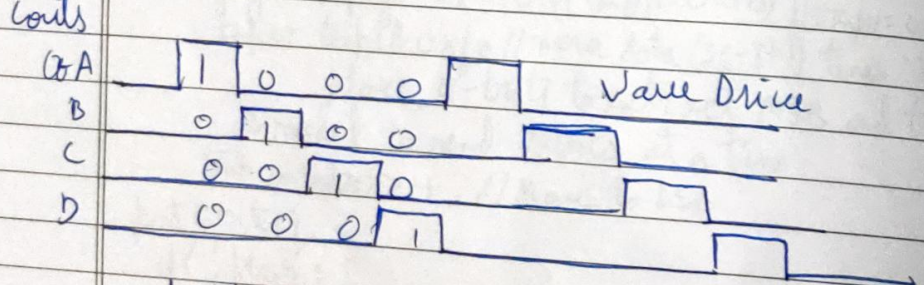
Stepper Motor

- Electromechanical device that converts electrical energy to mechanical energy.
- They are driven by digital pulses rather than continuous applied voltage.
- They rotate in fixed angular increments
- Five wire stepper is widely used Darlington driver

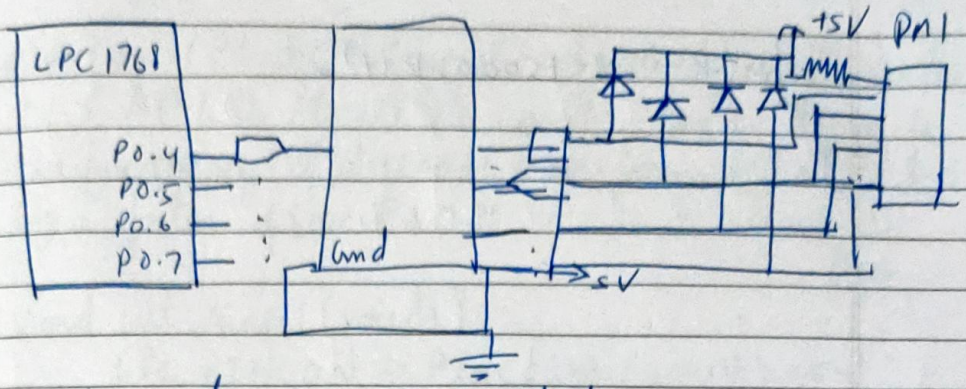
For 360°



X	X̄	Y	Ȳ	
0	1	0	1	0°
1	0	0	1	90°
1	0	1	0	180°
0	1	1	0	270°



1 step = 1.8° rotation
4 push step = 7.2° rotation

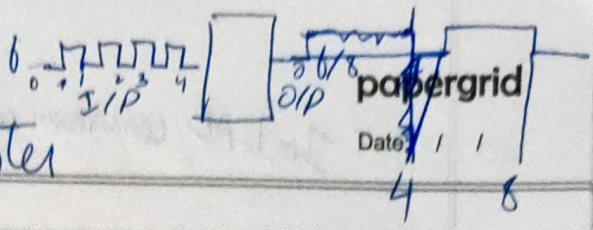


Stepper motor direction control

Clockwise:

```
int val;
int i=0, k=0;
int main(void){
    SystemInit();
    SystemCoreClockUpdate();                    P0.4-P0.7 as GPIO
    LPC_PINCON->PINSEL0 = 0x00000000;
    LPC_GPIO0->FIODIR = 0x000000F0;
                                    //P0.4 to P0.7 as output
    while(1){
      clock_wise();
    }
}
void clock_wise(void) //Exciting A->B->C->D
{
val = 0x00000008; //For clockwise
for(i=0; i<=3; i++) //for ABCD stepping
{
    LPC_GPIO0->FIOCLR = 0x000000F0;
    LPC_GPIO0->FIOSET = val;
    val = val << 1;
```

```
for(k=0;k<15000;k++);
```



Timer / Counter

fn 3

Q. MAT1.1 (P1.25) toggles whenever count reaches 3. CAP 1.0 (P1.28) is counter clock. Divide the frequency of square wave form input at P1.28 by a factor of 8 on P1.25

↓ fn 3

```
void init_timer1(void){
    LPC_PINCON -> PINSEL3 |= (3<<13)|3<<4);
    //MAT1.1(P1.25) and CAP1.0(P1.18)
    LPC_TIM1->TCR = 2 //Reset counter 1
    LPC_TIM1->CTCR = 0x2; //Counter at -ve edge of
    CAP1.0 ← 0010 → count -ve edge CAP1.0
//Toggles MAT1.1 for every 4 CAP1.0 cycles
    LPC_TIM1->MR1 = 0x03 //To count 4 clock pulses
```

MR1 MR0 SRI 0 000

↳ matches at 4 so increments at 4

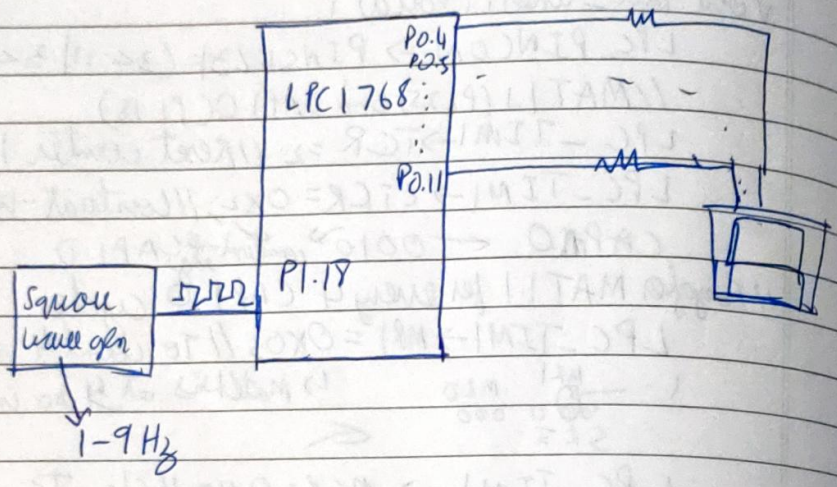
```
    LPC_TIM1 -> MCR = 0x10 //Clear TC when Match1
```

EMR 11 00 000 EM0

```
    LPC_TIM1->EMR = 0xC0; // Toggle FM1 when
                          match
    LPC_TIM1->TCR=1; //Start counter1
}
int main(void){
    init_timer1();
    while(1);
}
```

In LAB common cathode

Q. Assume that o/p square wave generation (f < 10Hz) connected to P1.18 (CAP1.0/fm3) write a program to display the frequency of square wave form on seven segment to P0.11 - P0.4



3Hz → 3 cycles per second ⇒ 3 falling and 3 rising edges.

In this question display O/P of no. of pulses after every 1s.

```
unsigned char seven_seg[10] = {0x3F, 0x06, 0x5B,
0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F};

Void delay (void) {
    LPC_TIM0 → TCR = 0x00000002; //Timer0 reset
    LPC_TIM0 → EMR = 0x20; //Set match bit upon match
    LPC_TIM0 → PR = 3000; // for 1ms.
    LPC_TIM0 → MR0 = 1000; // for 1s
    LPC_TIM0 → MCR = 0x00000004; //Stop PC and
                                  TC on MR0.
```

```
LPC_TIM0->TCR= 0x00000001; Timer0 Enable
while(!(LPC_TIM0->EMR&0x01)); // Wait until match.
}

void init_counter1(void)
{
    LPC_PINCON->PINSEL3 |=(3<<4); //Cap1.0
                                  (P1.25)
    LPC_TIM1->CTCR=0x01; // Counter at rise edge
                         of CAP1.0
                  00 01 - rise edge counting
}

int main(void){
    LPC_PINCON->PINSEL0=0;
    LPC_GPIO0->FIODIR= 0x00000FF0;
    init_counter1();
  while(1){
    LPC_TIM1->TCR=2; //Reset counter TC=0
    LPC_TIM1->TCR=1; //Start counter
    Delay(); wait 1 second
    LPC_GPIO->FIOPIN=seven_segment[LPC_TIM1->TC]<<4;
    //counter on seven seg
  }
}
```

★ Not that important

1) Capture registers (CR0-CR1) - Used to store instantaneous value of times TC value

2) Capture control register (CCR) Decides how capture event occurs (TC to CR0 copy must).

CCR

Bit Symbol

0 CAPRISE → 1 - Capture Rising edge

1 CAPFALL → 1 → capture Falling edge

2 CAPOINTE → 1 → Interrupt event on CAP0.0

3 CAP0.1

4 CAP0.2

5 CAP0.3

Q. Capture TC into TC when +ve edge applied to CO.0 (P1.26) or CO.1 (P1.27).

```
void delay(void){
    LPC_SC->PCONP |= (1<<1); //Power up TIMO
    LPC_TIM0->CCR = 4; // capture on positive edge
                       // CO.1 pin
                       // 0010 0001 -> CO.0 pin
    LPC_TIM0->TCR = 0X00000002; //TC and PC reset
    LPC_TIM0->EMR = 0x20; //set match bit upon match
    LPC_TIM0->PR = 3000; //1ms
    LP:
    :

}
int main(void)
{
    LPC_GPIO->FIODIR = 0X00000004;
    LPC_PINCON->PINSEL3 = (3<<20)|(3<<22);
    while(1)
    {
        LPC_GPIO->FIOPIN = ~(LPC_GPIO->FIOPIN
                             & 0X00000004);
        delay();
    }
}
```

Interrupts

- A type hardware signal from device to request for a service.
- In LPC 176X, the NVIC (Nested Vector Interrupt Controller) supports 35 vectored interrupts.
- Interrupt vector - Address of interrupt service subroutine (ISR)
- Interrupt vector of Interrupt Type N is stored at an offset of N*4 from base address of interrupt vector Table (IVT).
- ★ NVIC must be enabled to service INTR request
- ★ The peripheral device must be enabled to generate interrupt when some event occurs, the INTR request is sent to NVIC.
- In case of Timer, there are 6 INTR enable flags (4 in MCR, 2 in CCR)
- When the event occurs the corresponding bit is set automatically in IR register for NVIC to find out occurrence of an event.
  If NVIC is enabled to service timer interrupt, it executes the corresponding ISR and gives the desired service to the timer.

Interrupt Registers.

Bit	Symbol	
0	MR0 Interrupt	0/1 = Event occurred
1	MR1 "	
2	MR2 "	
3	MR3 "	
4	CR0 "	
5	CR1 "	

☆ writing logic 1 to IR ~~the~~ to reset interrupt. zero has no effect when value present in IR

Q. Toggle LED connected to P0.2 every second while displaying states of switch connected to P1.0 on the LED connected to P2.0.

↳ Executed directly on NVIC

```
Void TIMER0_IRQHandler(Void){
    LPC_TIM0->IR=1; //clearing interrupt
    ticks++;  // To wait for 1 ms
    if(ticks==1000){
        ticks=0;
        LPC_GPIO0->FIOPIN=~(
        LPC_GPIO0->FIOPIN&
        0x00000004);
```

01 → x → increment and make PR=999 then is delay y0 → y

```
void init_timer0(void)
{
  LPC_TIM0->TCR = 0x00000002; // Timer 0 reset
  LPC_TIM0->CTCR = 0x00; // Timer
  LPC_TIM0->MR0 = 2999; // For 1s
  LPC_TIM0->EMR = 0x00;                    SCI
  LPC_TIM0->PR = 0;                        reset and generate inter.
  LPC_TIM0->MCR = 0x00000003; Reset
// TC upon match0 and generate ~~INT~~ INTR.
  LPC_TIM0->TCR = 0x00000001; // Timer0 enable.
  return;
}

int main(void){                    P0.2 o/p(led)
  LPC_GPIO0->FIODIR = 0x00000004; P2.0
  LPC_GPIO2->FIODIR = 0x00000001; o/p(square)
  init_timer();
  NVIC_EnableIRQ(TIMER0_IRQn); // Timer 0 int
                               enabled in NVIC
  while(1){
    LPC_GPIO2->FIOPIN = (LPC_GPIO1->FIOPIN &
                         0x01);
  }
}
```

External Hardware Interrupt

- System Control block of ARM has SFR's to handle external hardware interrupts
- LPC1768 has four external hardware interrupts: EINT0 - EINT3 in general represented as EINTx pins

P2.10 EINT0
P2.11 " 1
P2.12 " 2
P2.13 " 3

There are 2 types of interrupt triggers:

- Level Triggered: Level 0 or Level 1
- Edge Triggered: Rising or Falling edge

Registers associated with External Hardware Interrupts (EHI)

1) PINSEL - To configure pins as External Interrupts
2) EXTINT - EXT interrupt flag register contains interrupt flags for EINT0, EINT1, ... EINT3
3) EXTMODE - External Interrupt Mode register (Level / Edge triggered)
   0 1
4) EXTPOLAR - External Interrupt Polarity (Falling / Rising edge, Active low / high)
   0 1

☆ Writing 1 to a specific bit will clear corresponding interrupt.

EXTMODE	EXTPOLAR	EINTX
0	0	Level 0
0	1	Level 1
1	0	Falling edge
1	1	Rising edge

Steps to configure External Hardware Interrupts

1) Configure pins as external interrupts in PINSELX register.

2) Configure the EINTn as Edge / Level triggered in EXTMODE register.

3) Select the polarity (Falling / Rising Edge, Active Low / High) of the interrupt in EXTPOLAR register.

4) Enable the interrupts by calling NVIC_EnableIRQ (EINTx_IRQn)

5) Clear any pending interrupts in EXTINT on entering ISR.

Q. Toggle LED connected to P1.23 (O/P) at each -ve edge of the input applied at P2.12 (EINT2) fm 1)

```
void EINT2_IRQHandler(void);
int main(void)
{
    SystemInit();
    SystemClockUpdate();
    LPC_PINCON->PINSEL4 |=(1<<24)
    //P2.12 as EINT2 fm-1
    LPC_GPIO1->FIODIR=0x00800000;
    //P1.23 assigned output
    LPC_SC->EXTMODE=0x00000004;
    LPC_SC//EINT2 is initiated as edge sensitive o/p level
    System Control Block
    LPC_SC->EXTPOLAR=0x00000000;
    NVIC_EnableIRQ(EINT2_IRQn);
    while(1);
}

Void EINT2_IRQHandler(void){
    LPC_SC->EXTINT=0x00000004;
    //clear the interrupt
    LPC_GPIO1->FIOPIN=~(LPC_GPIO1->FIOPIN);
}
```

Toggling Operation

Q. Toggle LED connected to P1.23 whenever switch connected to P2.12 (EINT2, pin-01) is pressed. LED remains ON as long as the switch is pressed. Logic 0 is input.
↳ Continuous press - LED on.

```
int main(void)
{
    LPC_PINCON->PINSEL4 |= (1<<24); //P2.12 as EINT2
    LPC_GPIO1->FIODIR = 0x00800000;
    //P1.23 as O/P
    LPC_SC->EXTMODE = 0x00000000; //EINT2 as
    // level-0 sensitive
    NVIC_EnableIRQ(EINT2_IRQn); → Automatically executes EINT2_IRQHandler
    while(1);
}
```
↳ As long as button is pressed it keeps executing

```
Void EINT2_IRQHandler(void)
{
    LPC_SC->EXTINT = 0x00000004; //clear the interrupt
    LPC_GPIO1->FIOSET = 1<<23; //LED ON
    for(i=0; i<<255; i++);
    LPC_GPIO1->FIOCLR = 1<<23; //LED OFF
```

Q. Program to stop displaying Hex digits on 7 segment on button press

1 button to stop → IRQ
1 button to restart → Main program

Date: / /

GPIO Interrupts

Only supported in Port 0 & Port 2

They are edge triggered.

Falling Edge Interrupt

Enables falling edge detected interrupt in which 'x' the group number, either 0 or 2. To turn on falling edge interrupt of pin, set corresponding pin to '1'.

Ex:

To enable P0.3 as falling edge

LPC_GPIOINT->IO0IntEnf |= (1<<3);

Read only

Rising Edge Interrupt

Enables rising edge detected interrupt. In which 'x' is the group number to turn on rising edge interrupt of pins set the corresponding pin to 1

Ex:

To enable P0.5 and 0.5 as rising edge interrupt

LPC_GPIOINT->
IO0IntEnR |= ((1<<3) | (1<<5));

All GPIO interrupt are connected to EINT3 interrupt source.

You need to turn EINT3_IRQn ON in order to use GPIO interrupt

NVIC_EnableIRQ(EINT3_IRQn);

GPIO registers — Sep reg for Port 0, Port 1

IntStatus → GPIO overall interrupt status

IntEnR → GPIO interrupt enable rising edges 1 - Port 0, 2 → Port 2

IntEnF → GPIO interrupt for falling edge

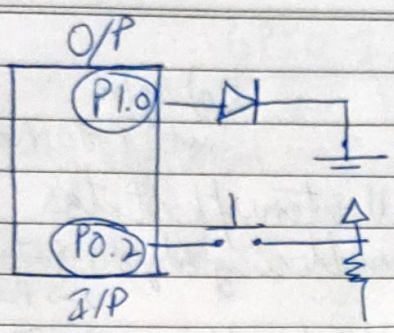
IntStatR → GPIO interrupt status for rising edg
↳ sep reg for 0x2

IntStatF → falling

IntClr → GPIO Interrupt Clear → ~~Sep reg for 0x2~~
↳ only single register
↳ only one will be serviced when both falling and rising.

IO Int Status Register

Bit	Symbol	Value	Descrip
0	P0Int	0	No pending interrupts
		1	Pending interrupts
1	-		
2	P2Int (Port)	0	There are no pending interrupts on Port 2
		1	There is atleast one pending interrupt
~~31:3~~ 31:2			Reserved




```
main()
{
LPC_PINCON->PINSEL0 = Value
LPC_PINCON->PINSEL2 = Value
                      ↳ input
LPC_GPIO0->FIODIR = 0<<2 I/P

LPC_GPIO1->FIODIR = 1<<0  O/P

//3rd interrupt handler for GPIO
NVIC_EnableIRQ(EINT3_IRQn);
LPC_GPIOINT->IO0INTEnR = 1<<2;

while(1){ //other task/fn"/operations
LPC_GPIOINT->IO0INTENF = 1<<2;
}
}

Void EINT3_IRQHandler(void){

n=LPC_GPIOINT->IO0INTStatR;
n=(n & 0x1<<2);
if(n){
LPC_GPIO1->FIOSET = 1<<0;
}
y=LPC_GPIOINT->IO0INTStatF;
y=(y & 0x1<<2);
```

```
LPC_GPIOInt -> IO0INTCLR = 1<<2;
if(y){
    LPC_GPIO1 -> FIOCLR = 1<<0;
}
```

Analog to Digital Converter

- The ADC block in LPC1768MC is based on Successive Approximation Register (SAR) conversion method.

- The ADC module uses 12 bit SAR.
  Analog I/P voltage 0-3.3V

- Maximum 8 ~~multi~~ analog input can be converted to digital (Multi-Channing).

★ $V_A = (V_{REFP}/2^N)$

$V_{REF} = 3.3V$ reference voltage of ADC.
$V_A \propto$ Dig O/P $\times K$

$\frac{3.3}{2^{12}} = 0.805mV$

$V_A = 0.805$ (Dec Rq)
$V_{A_1} = 0.805 \times (1) \rightarrow$ Dig O/P Voltage
$V_{A_2} = 0.805 \times 2$
⋮
$V_{A_{4095}} = \underbrace{0805 \times 4095}_{\text{close to } (3.3-0.805)} = 0XFFF$

Whenever I/P changes by 0.805 mV O/P Changes by +1

channels
In Labs 5 & 6 used

ADCR - A/D Control Register

Bits	Symbol	Value	Description
7:0	SEL		Used to select which channels are to be converted
15:8	CLKDIV		Division factor for clock 00 to be used by default
16	Burst	1	Only when burst bit = 1 then concurrent conversion will take place
		0	Conversions software controlled and require 65 clocks. Start bits must be 000 when burst = 1 or conversions will not start. If Burst is set to 1, the ADCINTEN bit in ADCINTEN must be 0
20:17			Reserved
21	PDN	1	Power ON
		0	Off
23:22	Reserved		
26:24	Start	000	Not start (for Burst) — HW automatically does
		001	Start conversion now by SW
		Others	irrelevant
27	Edge	1	Start conversion falling edge
		0	Start conversion on rising edge.
31:28	Reserved		

AD0.0 – P0.23 FNO1
AD0.1 – P0.24 FNO1
AD0.2 – P0.25 FNO1
AD0.3 – P0.26 FNO1
AD0.4 – P0.27 FNO3
AD0.5 – P0.31 FNO3
AD0.6 – P0.3 FNO2
AD0.7 – P0.2 FNO2

A/D Global Data Reegister (ADGDR)

- It holds the result of most recent A/D conversion that has completed

Bit	Symbol	Description
3:0	-	Reserved
15:4	Result	12 bit output value
23:16	Reserved	
26:24	CHN (Channel ID)	These bits contain the channel from which result bit were converted
29:27	Reserved	
30	overrun	The result is 1 in burst mode if results of one or more conv. conversion was lost and overwritten before the conversion
31	Done	Bit is set to 1 when A/D conversion completes.

A/D Data Register (ADDR0 to ADDR7)

Each channel has a data register

3:0	Reserved	
15:4	Result	
29:16	Reserved	
30	Overrun	Bit is 1 if one or more conversions were lost in burst
31	Done	Bit is set to 1 when A/D conversion completes.

A/D Interrupt Enable Register (ADINTEN): This register allows control over which A/D channel generate an interrupt.

Bit	Symbol	Value	Descr.
0	ADINTEN0	0	Completion of conversion on ADC channel 0 will not generate
		1	Will generate interrupt on completion of conversion
1	ADINTEN1		
8	ADGINTEN	0	Only individual ADC channels enabled by ADINTEN7:0 will generate interrupts. This bit must be set to 0 in burst mode.
		1	Only global done flag in ADDR is enabled to gen interrupt
31:17	Reserved		

A/D Status Register

Contains all done & overrun flags

Bit	Symbol	Descri
0	Done 0	→ Mirrors done flag status from result register for A/D channel 0
1	Done 1	
⋮	⋮	
7	Done 7	
8	Overrun 0	
⋮	⋮	
15	Overrun 7	
16	ADINT	This bit is the A/D interrupt flag. It is one when any of the individual A/D channel done flag is asserted and enabled to contribute to the A/D interrupt via the ADINTEN register

Q. ADC software mode for 2-channel concurrent conversion.

```
main(void){
   // P1.30 as AD0.4, P1.31 as AD0.5
   LPC_PINCON->PINSEL3 = (3<<28) | (3<<30);
   LPC_ADC->ADINTEN = 0
   while(1)
   {
      LPC_ADC->ADCR = (1<<4) | (1<<21) | (1<<24);   // select channel 4, power on, 001 start conversion
      while(((temp4 = LPC_ADC->ADDR4) & (1<<31)) == 0);
      // wait till done bit becomes 1.
      temp4 = LPC_ADC->ADDR4;   // at 15-4
      temp4 >>= 4
      temp4 &= 0x0000FFFF
   }   // Channel 4

   {
      LPC_ADC->ADCR = (1<<5) | (1<<21) | (1<<24);
      for(i=0; i<2000; i++);
      while(((temp5 = LPC_ADC->ADDR5) &
             (1<<31)) == 0);
      temp5 = LPC_ADC->ADDR5;
      temp5 >>= 4;
      temp5 &= 0x0000FFFF; // 12 bit ADC
   }   // Channel 5
}
```

Q.2) ADC burst mode for 2 channel concurrent conversion.

```
int main(Void) {                                  P1.25 as AD0.5
                                                  P1.30 as AD0.4 and
    LPC_PINCON->PINSEL3 = (3<<18)|(3<<30);
    LPC_ADC->ADCR = (1<<4)|(1<<5)|(1<<16)|(1<<21);
    //Enable CH4 and 5 in burst with ADC power on
                                        Burst bit
    LPC_ADC->ADINTEN = (1<<4)|(1<<5);
    //Enable done for INTR.
    NVIC_EnableIRQ(ADC_IRQn);
    while(1);
}

Void ADC_IRQHandler(Void)
{ int channel, temp, result;          Global data reg
    channel = (LPC_ADC->ADGDR>>24) & 0x07;  -> channel ID
    Result = (LPC_ADC->ADGDR>>4) & 0XFFF;   -> within
                                              15-4 result.
    if (channel==4)
    {
        temp4 = (LPC_ADC->ADDR4>>4) & 0XFFF;
        //Read to clear done flag
    }
    else if (channel==5)
    {
        temp5 = (LPC_ADC->ADDR5>>4) & 0XFFF;
        //Read to clear done flag
}
```

Date: / /

★★★ To learn from slides / Lab manual

Q. Display input analog voltage and display on LCD.

O/P Analog I/P 1.1V → 0-3.3V
ADC output 0x555

DAC

- DAC used in audio equipment like music players.

Types of DAC:

i) Switched resistor
ii) R-2R Ladder DAC → Resistor String
iii) Successive Approximation DAC

- Simplest method to implement PWM Functionality of a timer/peripheral.

- Resolution – No. of possible values DAC unit can produce – 1024 values

DAC channel	Port pin	Pin Func^n	Associated PINSEL
AOUT	P0.26	0 – GPIO, 1 – AD0[3], 2 – AOUT, 3 – RXD3	20, 21 bits of PINSEL1

, 3.3V , 0V

$$V_{AOUT} = \frac{VALUE * (V_{REFP} - V_{REFN})}{1024} + V_{REFN}$$

$$\Rightarrow V_{AOUT} = \frac{VALUE * V_{REFP}}{1024}$$

- LPC1768 has only a single DAC channel. The remaining pins related to power.

Registers in DAC

DACR → Analog to digital conversion

DACCTRL & DACCNTVAL → Irrelevant

DACR - Contains digital value that must be converted into analog.

Bit [5:0] : Reserved

Bit [15:6] - value after a new value is written to this field, given settling time selected using BIAS has elapsed, we get converted Analog voltage at output.

Bit [16] - Bias - setting this bit to 0 settling time of 1 µs max with max current of 700 µA at max 1 MHz.
Setting it to 1 will select settling time of 2.5 µs but reduce max current of 300 µA at max 400 kHz

Bit [31:17] - Reserved.

Square Wave

```
#define Voltage 1024
#define freq 120000

int main(void){
uint32_t m;
LPC_PINCON->PINSEL1 |= (1<<21);   // for P0.26 = AOUT
while(1){

  LPC_DAC->DACR = (Voltage/2 <<4);
  for(m=freq; m>1; m--);

  LPC_DAC->DACR = (Voltage <<4);
  for(m=freq; m>1; m--);
}
}
```

Triangular Wave

```
#define Voltage 1024

int main(void){
uint32_t i=0;

LPC_PINCON->PINSEL1 |= (1<<21);
while(1)
{
  for(i=0; i<Voltage; i++)        } +ve half
    LPC_DAC->DACR = (i<<6);
  for(i=Voltage; i>0; i--)        } -ve
    LPC_DAC->DACR = (i<<6);
}
}
```

Sin Wave of voltage swing 0V to 2.5V

$Y = A + A\sin\theta$

where A = 1.25 V (mid voltage value)

For $\theta = 7.5°$

$Y = A + A\sin\theta$

$= 1.25 + 1.25 \sin 7.5$

$= 1.41315 V$

For 2.5 → 0xff

then 1.41315 →

00001, 0100, 0100

$X = (255 \times 1.41315) / 2.5$

$= 143.57 = 144 = 90H$

```
#include<LPC17xx.h>
int count=0, sinevalue, value;
unsigned char sine_tab[49]={0x80, 0x90, 0xA1,
0xB1, 0xC0, 0xCD, 0xDA, 0xE5, 0xEE, 0xF6,
0xFB, 0xFE, 0xFF . . . . .
. . . . . 0x80};
int main(void){

    LPC_PINCON->PINSEL1 |= (1<<21);
count=0;
while(1){
  for(count=0; count<=48; count++){
    sinevalue = sine_tab[count];
    value = sinevalue<<8;
  LPC_DAC->DACR = value
  }
}
```

Date: / /

Serial Communication

- The process of sending data sequentially over a computer bus is called as serial communication, which means the data will be transmitted bit by bit.

- Serial communication is slower than parallel but used for long ~~dis~~ data transmission due to lower cost.

Characteristics:

1) Baud Rate - Bits transmitted per second.

2) Stop Bits - Used for a single packet to stop the transmission which is denoted as "T".

3) Parity Bit - Simplest form of checking the errors.

Eg: if 011 is a number parity bit=0; even parity as 1's (wireless)

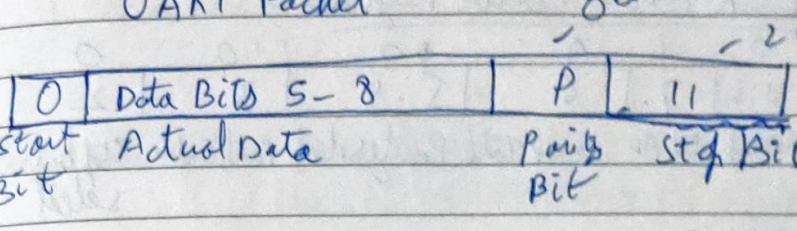
- RS232 is a standard protocol used for serial communication

- Universal Asynchronous Data Receiver & Transmitter (UART) used in communication with RS232 for transferring data.

UART typically used for communication b/w Microcontroller and a computer

- UART is a full duplex communication hence it needs 2 wires (1 in b/w the communicating device. They are called Tx and Rx.

UART Packet



Start Bit = 0 => Indicates data about to be transmitted.

UART1 adds full modem control and handshaking support

- UART0 - used for in-system programming
- UART0 & 1 are enabled after reset

- UART2 & 3 also available.

- RXD0 - RXD3 → Used for transmitting
- TXD0 - TXD3 → Transmission

Registers

RBR – Contains recently received data
THR – contains data to be transmitted
FCR – FIFO control register
LCR – Controls UART frame format
DLL, DLM – MHT/LHT of UART's baud rate generator

LCR (Line Control Register)

0	1	0	00	0	0	11 → 0x3
31:8	7	6	5:4	3	2	1:0
Reserved	DLAB	Break control	Parity select	Parity Enable	Stop bit select	Word length select

- Bit 1:0
  - 00 - 5 bit
  - 01 - 6 bit
  - :
  - 11 - 8 bit
- Bit 2 Stop, number of bits
  - 0 – 1 stop bit
  - 1 – 2 stop bits
- Bit 3
  - 0 - Disable parity enable and checking
  - 1 - Enable parity "
- Bits 5:4 : To select type of parity
  - 00 - odd parity
  - 01 - even parity
  - 10 - Forced '1' stick parity
  - 11 - Forced '0' stick parity

Bit6: Break control
0 - Disable Break transmission
1 - Enable break " . O/P pin UART TXD is forced to logic 0

Bit7 - DLA; Divisor latch access bit
0 - Disable access to divisor latch
1 - Enable "

$$\text{Baudrate} = \frac{\text{Pclk}}{(16 \times ()5(\times DLM + DLL) + \left(1 + \frac{\text{DivAddVal}}{\text{MulVal}}\right)}$$

PCLKSELx register contains PCLK clock info for all clock dependent peripherals in which bit 6, Bit no is for UART clock.

UART_PCLK	PCLK
0	System freq/4
1	System freq
2	System freq/2
3	System freq/8

DivAddVal/MulVal == 0.

For

UxDLL = $\frac{\text{PCLK in Hertz}}{16 \times \text{Desired - BaudRate}}$

16 byte FIFO for Receiver/Transmitter

- Thus it can store 16 bytes of data received on UART without overwriting. If the data is not read before the queue (FIFO) is filled filled then the next data will be lost and the Overrun error bit will be set.

Steps for configuring UART0

1) Configure GPIO pin for UART0 function ~~as~~ using PINSEL registers.

2) Configure FCR for enabling the FIFO and reset both the R_x / T_x FIFO.

3) Configure LCR for 8-data bits, 1 stop bit, disable parity and enable DLAB.

4) Get PCLK for from PCLKSELx register 7-6 bits

5) Calculate DLM and DLL values for required baudrate from PCLK.

→ 1600

6) Update the DLM, DLL with the calculated values

7) Finally clear DLAB to disable the access to DLM, DLL

Now UART will be ready to transmit/receive data at Specified baudrate.

Date: / /

Only receiver this will be 1

For both receive and transmit TX=0

31:9	7:1	5:4	3	2	1	0
Reserved	RX Trigger	Reserved	DMA mode	TX FIFO reset →	RX "	FIFO enable
0		0		0	1	1

⇒ 0x07

```
void UART0_Init(void)
{
    LPC_SC->PCONP |= 0x00000008; // Uart peripheral device
    LPC_PINCON->PINSEL0 |= 0x00000050;
    LPC_UART0->LCR = 0x00000083; //Enable divisor
    LPC_UART0->DLM = 0x00; //latch, parity disable, 1 stop
    LPC_UART0->DLL = 0x13 //bit, 8 bit word len
    LPC_UART0->LCR = 0x00000003; // Disabling access
    LPC_UART0->FCR = 0x07; //Transmit and receive both cleared
    LPC_UART0->IER = 0x03; //Transmit and receive interrupt
```

→ For selecting TX0[P0.2 → 5:4] and RX[P0.3 → 7:1] of UART0.

→ selects baud rate as 9600 bps.

```
NVIC_EnableIRQ(UART0_IRQn
}
```

Pulse Width Modulation (PWM)

- Used to control amount of power delivered through
- The width of the pulse ie the duration for which pulse stays on in a period is varied. Hence it is called pulse width modulation

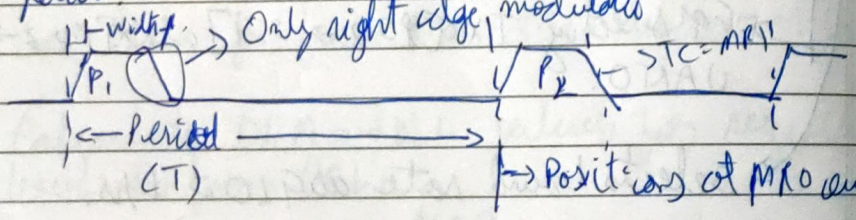
Duty cycle = $T_{on}/(T_{on}+T_{off})$

Duty cycle % = $T_{on} * 100/(T_{on}+T_{off})$

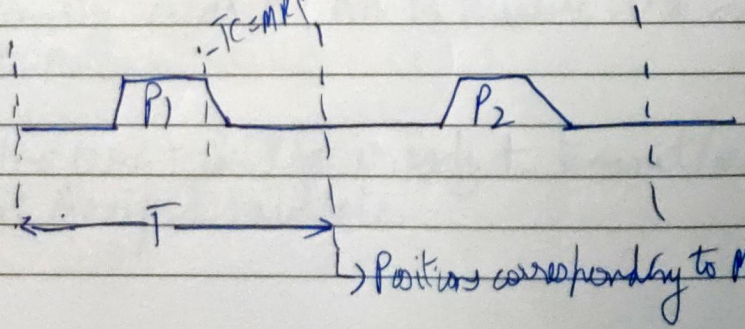
Duty cycle is varied to vary intensity of bulb

Types of PWM

1) Single Edge PWM: Pulse starts at beginning of period



2) Double edge PWM: Both the edges can be modulated and hence pulse is placed anywhere in period.



V_{Avg} = Duty cycle * V_H

In LPC1768 6 PWM O/P

PWM1.1	P1.18 / P2.0
PWM1.2	P1.20 / P2.1 / P3.25
PWM1.3	P1.21 / P2.2 / P3.26
PWM1.4	P1.23 / P2.3
PWM1.5	P1.24 / P2.4
PWM1.6	P1.26 / P2.5

Max 6 single edge PWM
Max 3 double edge PWM

PWM channel	Single Edge PWM set by	Reset By	Double Edge PWM set by	Reset By
1	M0	M1	M0	N/A
2	M0	M2	M1	M2
3	M0	M3	M2	M3
4	M0	M4	M3	M4
5	M0	M5	M4	M5
6	M0	M6	M5	M6

PWM Timer Control Register (PWMTCR)

Bit 0 Counter Enable 1 → PWM timer counter and prescale counter enabled

Bit 1 Counter Reset 1 → PWM Timer counter and prescale counter reset on next rising edge of PCLK

Bit 3 PWM enable 1 - PWM mode enabled. TC will reset to 1

MR0 → PWM cycle

PWM count control register (PWMCTCR)

Bit	Symbol	Value	Desc
1:0	Counter/ Timer	00	Timer mode: TC incremented when ~~PCLK~~ PC counter matches PC register
		01	Counter mode: TC incremented on rising edges
		10	TC incremented on falling edge
		11	TC incremented on both falling and rising
3:2	Count input select		When bits 1:0 are not 00 these bits select which PCAP pin carries signal used to increment TC.
		00	PCAP1:0
		01	PCAP1.1

PWM Match control Register

Bit	Symbol	val	Descrip
0	PWMMR0I	1	Interrupt on PWMMR0 matches PWMTC
		0	Interrupt is disabled
1	PWMMR0R	1	Reset on PWMMR0
		0	Disabled
2	PWMMR0S	1	Stop on PWMMR0: the PWMTC will be stopped and PWMTCR[0] will be set to 0 if PWMMR0 matches PWMTC.
		0	Disabled

/// 4 lines exist till MR6 till bit 20

PWM Capture control Register

Bit	Symbol	Value	Descrip
0	Capture on PCAP1.0 rising	0	Disabled
		1	Synchronously sampled rising edge on the (PCAP1.0) will cause CR0 to be loaded with contents of TC
1	Capture on PCAP1.0 falling edge	0	Disabled
		1	"Button-falling edge
2	Interrupt on PCAP1.0 event	0	Disabled
		1	ACR0 load due to a (PCAP1.0) event will generate an interrupt.

:

5 Till PCAP1.1 — CR1

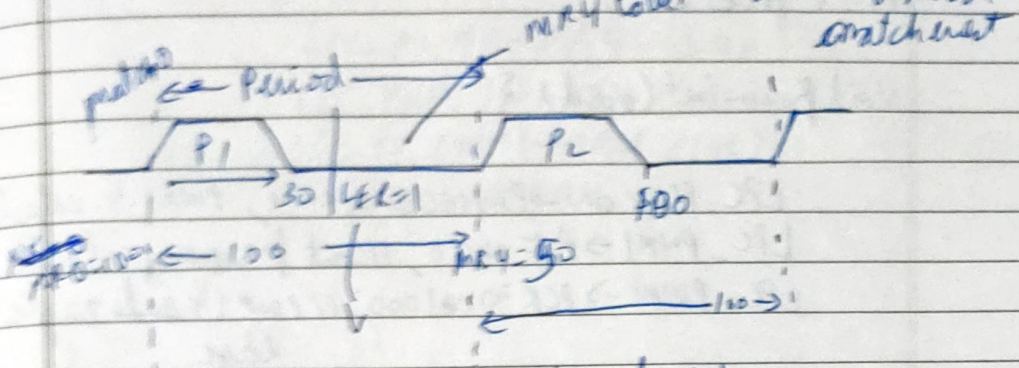
PWM control Register (PWMICR

Bit	Symbol	Value	Descrip
1:0	Unused	–	–
2	PWMSE2	1	Select double edge controlled mode PWM o/p
		0	" Single edge
6	PWMSEL6	1	" PWM6 o/p
			" Single edge PWM6
8:7	Reserved		
9	PWMENA1	1	PWM1 o/p enabled
		0	" disabled
14	PWMENA6	1	PWM6 o/p enabled
		0	" disabled.
31:15	Unused		

PWM Latch Enable Register (PWMLER)

MR0	0	Enable PWM0 match0 latch	Writing one to this bit allows last value written to PWM match 0 register to become effective when next time is reset by PWM match event
MR6	6	Enable PWM match6 latch	

Single edge PWM 1, 4



match event — PWMMR4 will be loaded

PWM Interrupt Register (PWMIR)

Bit	Symbol	Description
0	PWMMR0	Interrupt flag for PWM match channel 0
:		
3	PWMMR3	" channel 3
4	PWMCAP0	Interrupt flag for capture input 0.
5	PWMCAP1	" 1
7:6	reserved	
8	PWMMR4 interrupt	Interrupt flag for PWM match channel 4
:		
10	PWMMR6	" channel 6
31:11	reserved	

To generate intensity of LED connected to PWM1.4
P1.23 func^n 2

```
Void pwm_init(void) {

    LPC_PINCON->PINSEL3 = 2<<4;  PWM1.4 selected
    LPC_PWM1->PR = 0x00;  //Count frequency : Fpclk
    LPC_PWM1->PCR = 0x1000;  // PWM1.4 Single edge
    LPC_PWM1->MCR = 0x03;  //Reset and interrupt on PWMMR0
    LPC_PWM1->MR0 = 30000;  //Set up match register MR0
    LPC_PWM1->MR4 = 50;  //Set up match register MR4
    LPC_PWM1->LER = 0x7F;  // enable shadow copy register
    LPC_PWM1->TCR = 0x09;  // Enable PWM and counter
    LPC_PWM1->TCR = 0x02;  //Reset TC and PC
    NVIC_EnableIRQ(PWM1_IRQn);
    return;
}
Void PWM1_IRQHandler(void) {

    LPC_PWM1->IR = 0x01;  //Clear interrupt
    if(flag == 0x00) {
        LPC_PWM1->MR4 += 50;
        LPC_PWM1->LER = 0x7F;
        if(LPC_PWM1->MR4 >= 29900) {
            flag = 0xff;
        }
    }
}
```

Increasing

Enable all counters

→ Decreasing

```
else if (flag=0xbb){
        LPC_PWM1->MR4 -=50;
        LPC_PWM1->LER = 0x7F
        if(LPC_PWM1->MR4 <100){
            flag = 0x00;
        }
    }
}
```

Received transmit code

Q Message - "Failures are stepping stones to success"
TxD0(P0.2, fn2) at 9600 baud, 1-start, 1-Stop and 8 bit data, P_{CLK} = 3MHz

A.) include
#include <LPC17xx.h>

```
Void delay(unsigned int n1);
Void UART0_Init(void);
Void UART0_IRQHandler(Void);

unsigned long int i=0, j=0;
unsigned char tx0_flag = 0;
unsigned char *ptr, arr[] = "Failures are stepping stones to success";

int main(Void){

    SystemInit();
    SystemCoreClockUpdate();
    UART0_Init();
    while(1){
      ptr = arr;
      while(*ptr != '\0'){
          LPC_UART0->THR = *ptr++;
          //THR contains data to be transmitted
        while(tx0_flag == 0x00){
          tx0_flag = 0x00;
        }
        for(i=0; i<500; i++)
          delay(625);
      }
}
```

```
Void UART0_Init(Void){

    LPC_SC->PCONP |= 0x08; //UART0 peripheral
    LPC_PINCON->PINSEL0 |= 0x 2<<2;
    LPC_UART0->LCR = 0x83; //Enable division
                           latch, parity disable, 1 stop bit,
                           8 bit word length line control reg

    LPC_UART0->DLM = 0x00;
    LPC_UART0->DLL = 0x13;
    LPC_UART0->LCR = 0x03; //For both receive
                           and transmit
    LPC_UART0->FCR = 0x07; // Transmitter and receiver
    LPC_UART0->IER = 0x03; //select transmit
                           and receive interrupt
    NVIC_EnableIRQ(UART0_IRQn);
}

Void UART0_IRQHandler(Void){

    unsigned long int stat;
    Int_Stat = LPC_UART0->IIR; //Reading data from IIR
    Int_Stat = IntStat & 0x06; //masking other than
    // Tramit int & rvce data indicator

    if((Int_Stat & 0x02)==0x02)
        tx0_flag = 0xff;
    }

    Void delay(unsigned int n){
        for(i=0; i<n; i++)
    }
}
```

papergrid
Date: / /

Q. LPC1768, Switch P2.10, Aout P0.26 in 3
Peak Amp = 3.3,
Switch ON → 2 kHz
Switch OF → 4 kHz

Mid voltage = 1.65

$Y = A + A\sin(\theta)$ Sampling for every 30°

$Y = 1.65 + 1.65 \times \sin(30) = 2.475$

Considering output as 8 bit
Max O/P = 0xff

so 3.3V = 0xff
then $2.475 = \frac{255 \times 2.475}{3.3} = 191.25 = 191 = 0xBF$

Similarly Sine table can be computed.

$f = \frac{1}{2 \times 10^{-3}} = \frac{1000}{2} = 500$ 0.5×10^{-3} s = Time Period

★ As per slides, $T_d = T/12 \Rightarrow \frac{0.5 \times 10^{-3}}{12} = 0.042 \times 10^{-3}$
$= 42$ ns

Td for 4kHz ⇒ $\frac{0.25 \times 10^{-3}}{12} = 20$ ns

Code

```
#include <LPC17xx.h>

int m, count=0, sinevalue, value;
unsigned char sine_tab[12]={ 0x80, 0xBF
    .... };

int main (void)
{
    SystemInit();
    SystemCoreClockUpdate();

    LPC_PINCON→ PINSEL= 3<<20, P1.26 -3

    while(1) {
        for( count=0; count<=11; count++){
            sine value = sine_tab[count];
            value = sinevalue << 8;
            LPC_DAC→DACR = value;
            if( LPC_GPIO2 →FIOPIN A (1<<10))
            //If key is pressed
              LPC_TIM0 → MR0 = 125
        // (MR+1)x = 3x10^-6 x 42x10^-6  126, MR = 126-1
              delay();
        }
        else
        {
          LPC_TIM0→MR0 = 60
          MR+1=3x10^-6 x 20x10^-6
          delay();
        }
    }
}
```

```
void delay() {
    LPC_TIM0->TCR = 0x02;
    LPC_TIM0->EMR = 0x20;
    LPC_TIM0->PR = 0;
    LPC_TIM0->MCR = 0x04;
    LPC_TIM0->TCR = 0x01;
    while(!(LPC_TIM0->EMR & 0x01));
    return;
}
```