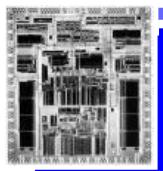


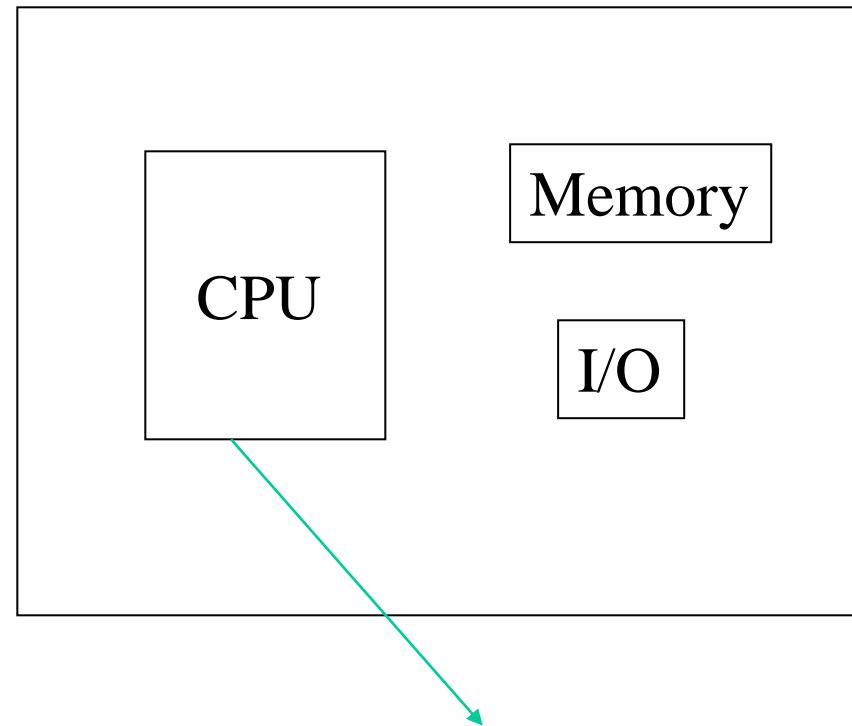
Embedded Systems



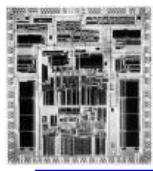
Basic Components of Computer

- CPU
- Memory
- I/O

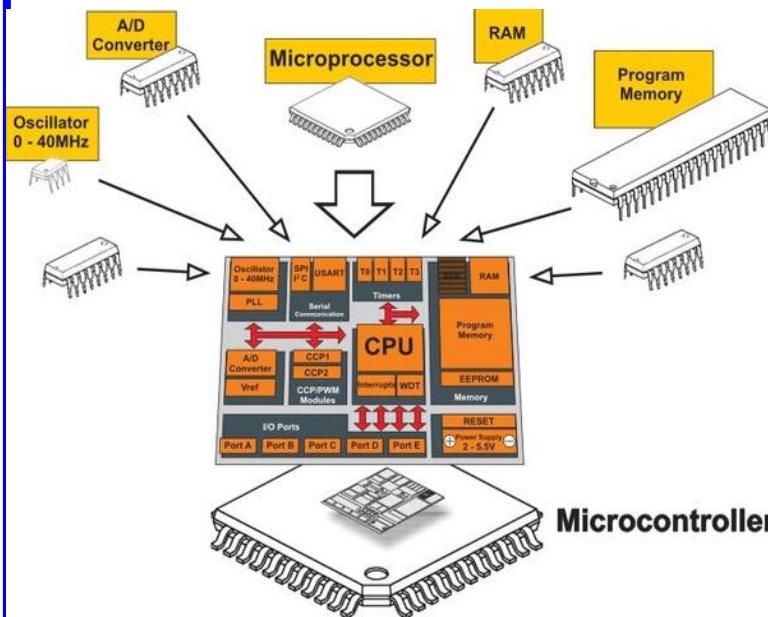
Motherboard



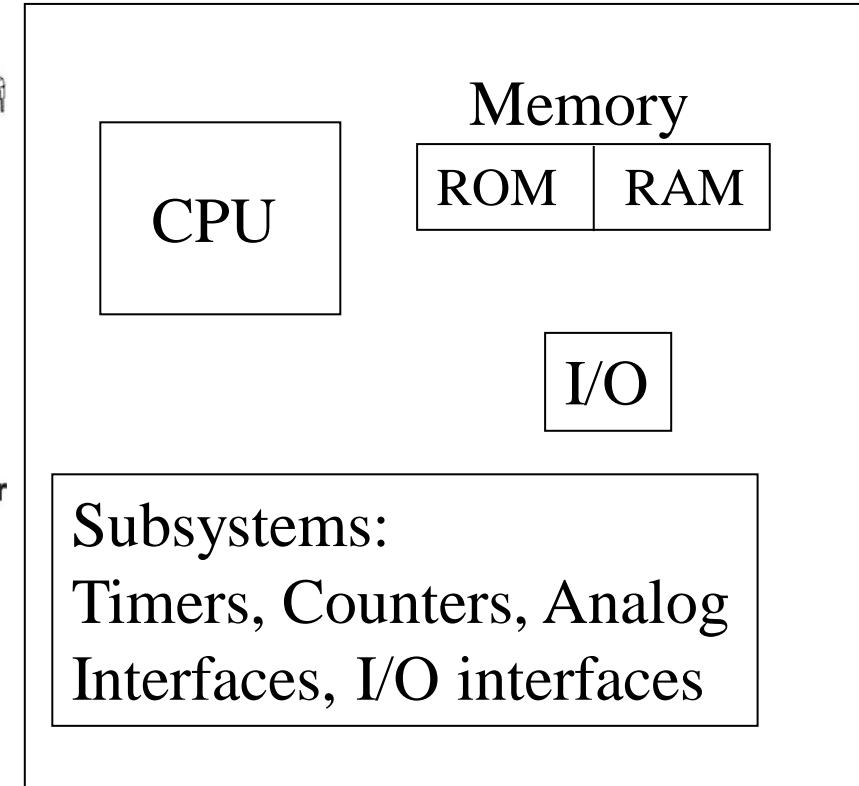
Microprocessor



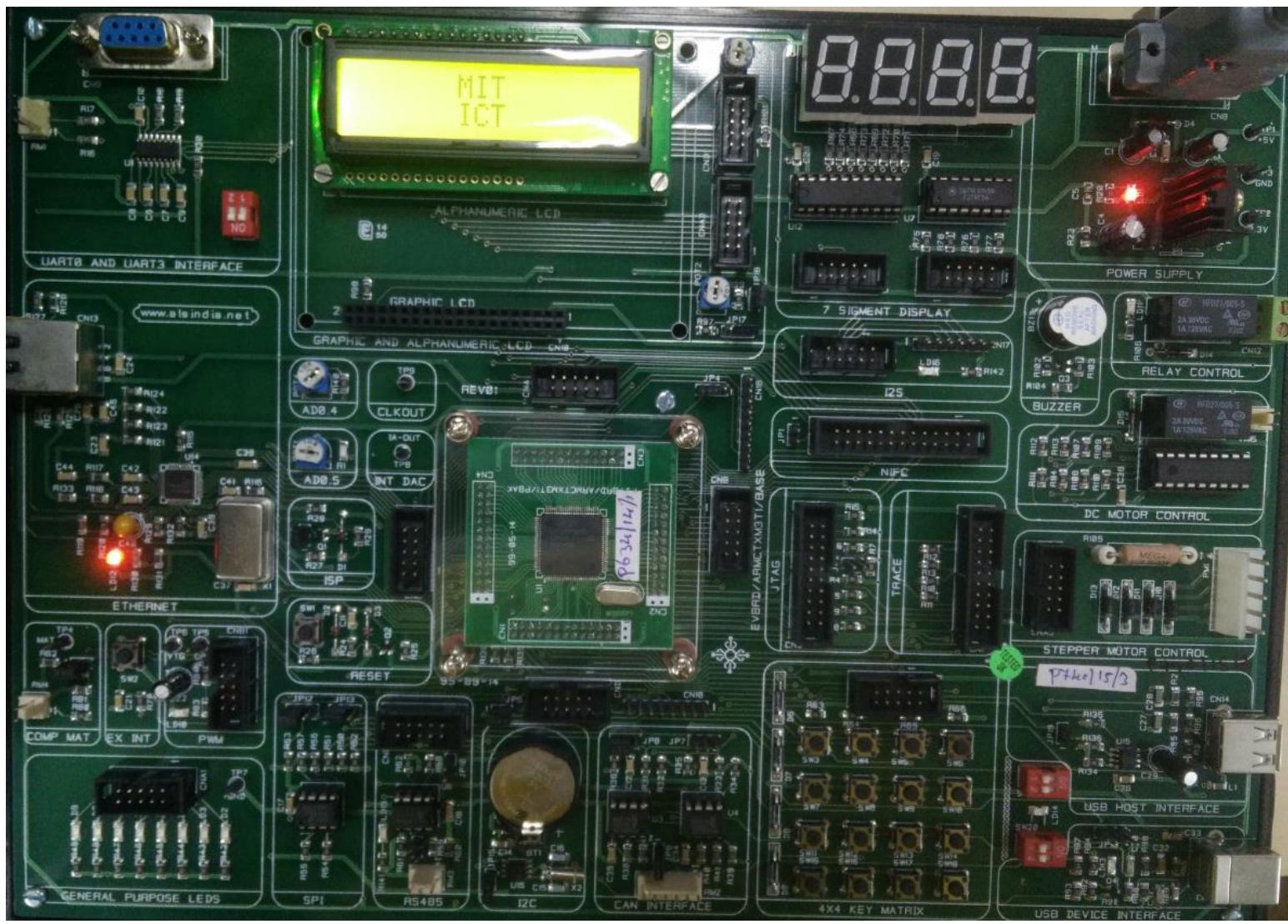
Microcontrollers



A single chip
(SoC)



LPC 1768



Microprocessor vs Microcontroller

Sl.no	Microprocessor	Microcontroller
1	General purpose processor	Specific application controller
2	Contains no RAM, no ROM, no I/O ports on chip itself.	Contains RAM, ROM, I/O ports on chip itself
3	Size of RAM/ROM can vary	Size of RAM/ROM is fixed
4	Makes the system bulkier	Make the system compact
5	More expensive	Less expensive
6	It has less bit handling instructions	It has more bit handling instructions
7	Less number of pins have multiplexed functions	more number of pins have multiplexed functions
8	More flexible in designer point of view	Less flexible in designer point of view
9	Limited power saving options compared to microcontrollers	Includes lot of power saving features
10	Eg: Desktop PC, 8086, i7	Eg: Digital Camera, 8051, msp430
11	Execution faster	Compared to µp slower
12	More general purpose registers	Less number of gen purpose registers
13	More addressing modes	Less addressing modes
14	Design time is more	Application design time less
15	Microprocessors are based on von Neumann model/architecture where program and data are stored in same memory module	Micro controllers are based on Harvard architecture where program memory and Data memory are separate
16	Cannot be used in compact systems and hence inefficient	Can be used in compact systems and hence it is an efficient technique
17	Example code: ADD AX,BX ADD AX,CX ADD AX,DX	Example code: MOV A, #2fh MOV B, #2fh ADD A, B
18	It cannot be used as stand alone	Can be used as stand alone.
19	May or may not be real-time application oriented	Real-time application oriented
20	Fig : a	Fig : b

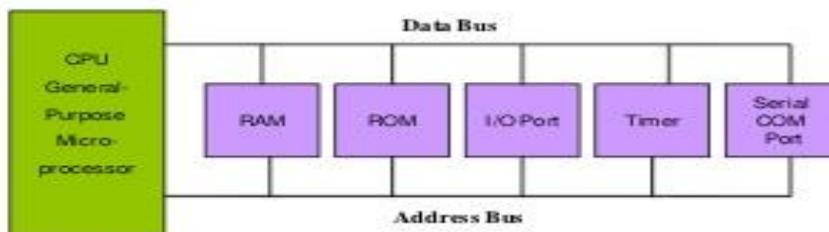


Fig : (a) Microprocessor

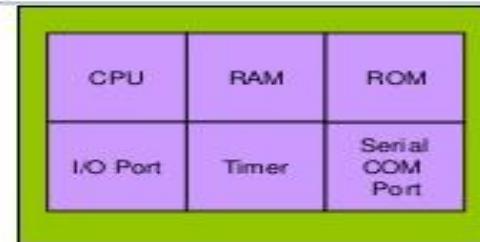
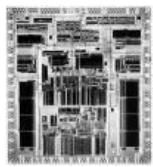
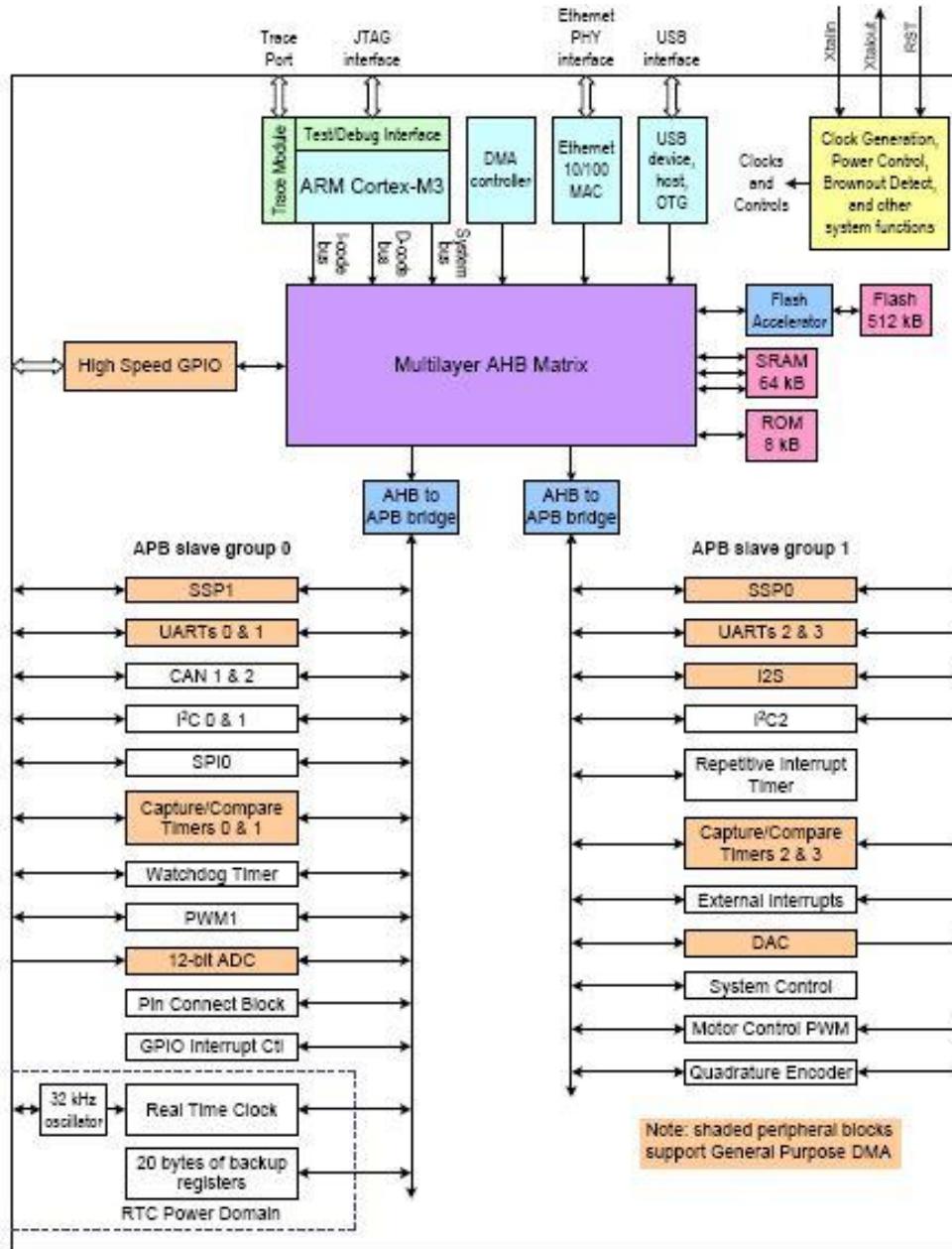
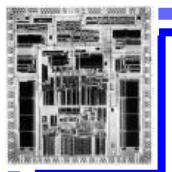


Fig : (b) Microcontroller



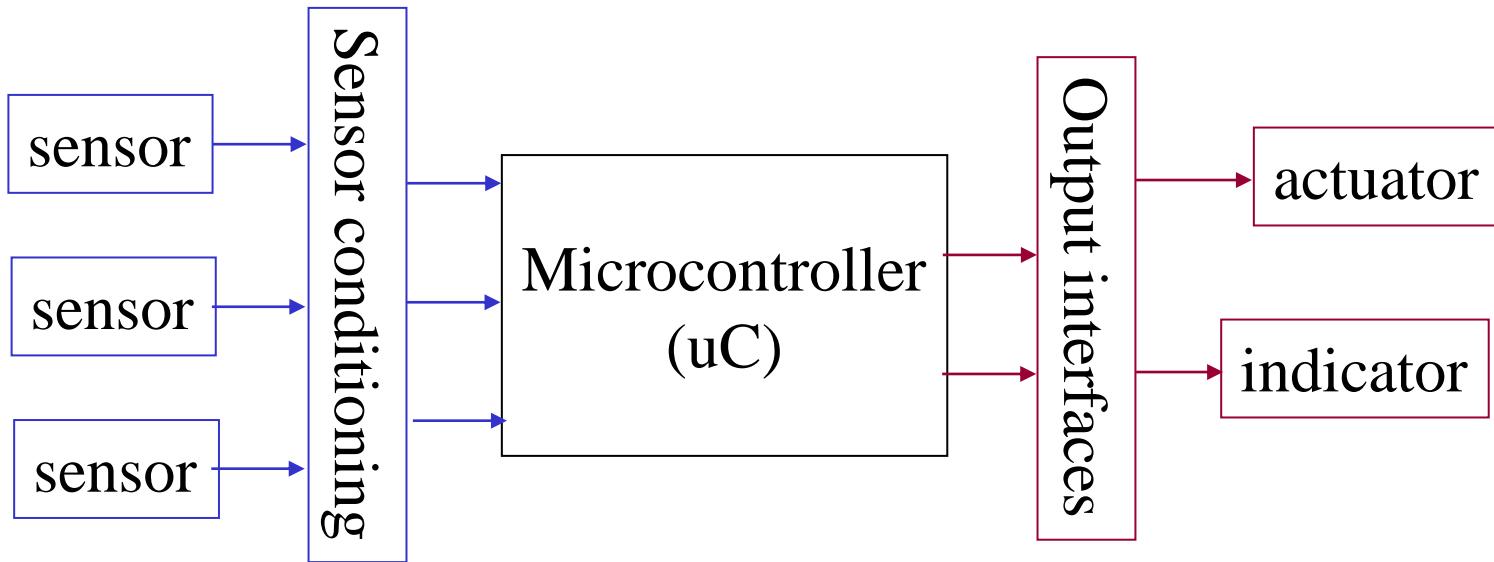
Microcontrollers

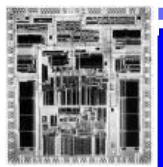




Embedded System

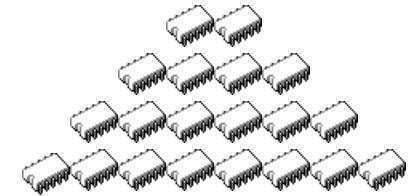
General Block Diagram



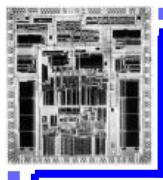


Embedded systems overview

- Embedded computing systems
 - Computing systems embedded within electronic devices
 - Nearly any computing system other than a desktop computer
 - Billions of units produced yearly, versus millions of desktop units

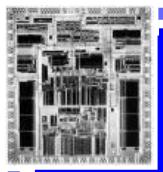


Lots more of these,
though they cost a lot
less each.

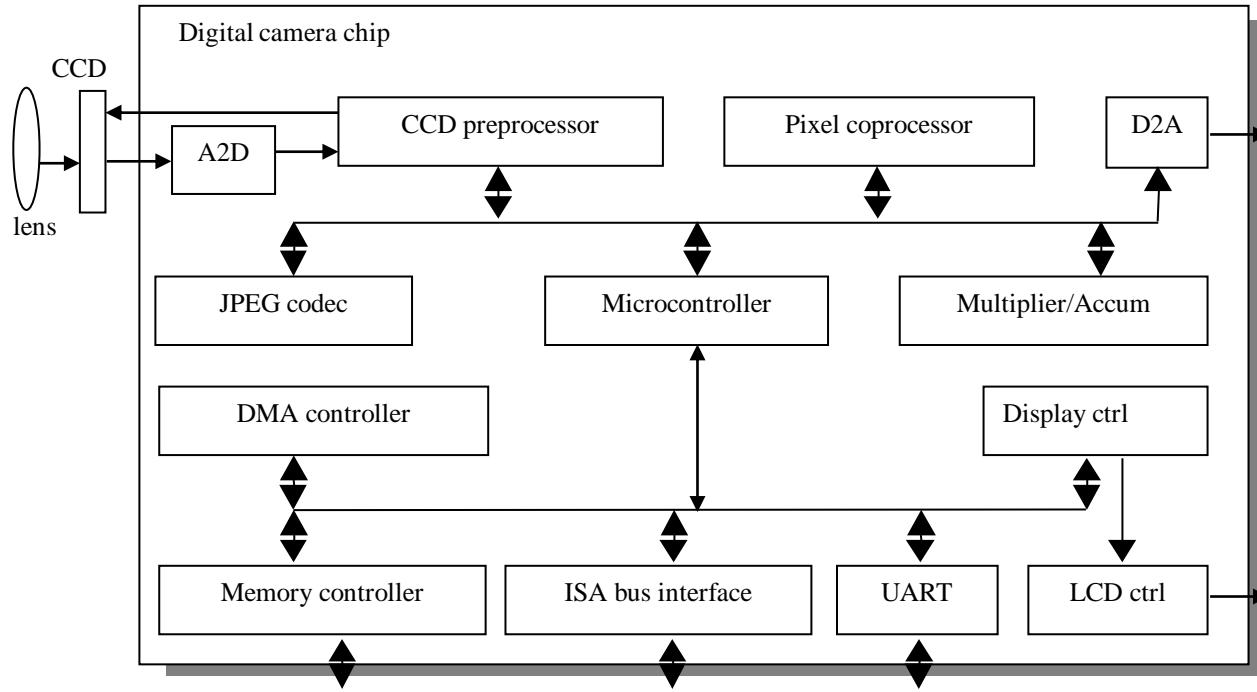


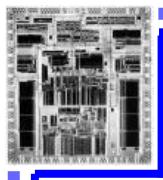
Some common characteristics of embedded systems

- Single-functioned
 - Executes a single program, repeatedly
- Tightly-constrained
 - Low cost, low power, small, fast, etc.
- Reactive and real-time
 - Continually reacts to changes in the system's environment
 - Must compute certain results in real-time without delay



An embedded system example -- a digital camera





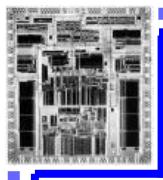
RISC vs CISC

What is RISC?

- A reduced instruction set computer is a computer that only uses simple commands that can be divided into several instructions that achieve low-level operation within a single CLK cycle, as its name proposes “Reduced Instruction Set”.

What is CISC?

- A complex instruction set computer is a computer where single instructions can perform numerous low-level operations like a load from memory, an arithmetic operation, and a memory store or are accomplished by multi-step processes or addressing modes in single instructions, as its name proposes “Complex Instruction Set”.

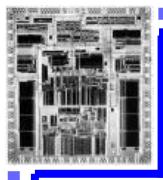


RISC vs CISC

RISC – Reduced Instruction Set Computer

CISC – Complex Instruction Set Computer

1. One of the major characteristics of RISC architecture is a large number of registers. All RISC architectures have at least 8 or 16 registers. Of these 16 registers, only a few are assigned to a dedicated function. One advantage of a large number of registers is that it avoids the need for a large stack to store parameters.
2. RISC processors have a fixed instruction size. In a CISC microprocessors such as the x86, instructions can be 1, 2, 3, or even 5 bytes. For example, look at the following instructions in the x86:

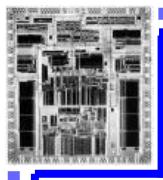


RISC vs CISC

3. RISC processors have a small instruction set. RISC processors have only basic instructions such as ADD, SUB, MUL, LOAD, STORE, AND, OR, EOR, CALL, JUMP, and so on.

The limited number of instructions is one of the criticisms leveled at the RISC processor because it makes the job of Assembly language programmers much more tedious and difficult compared to CISC Assembly language programming.

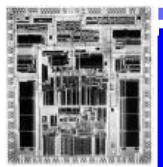
This is one reason that RISC is used more commonly in high-level language environments such as the C programming language rather than Assembly language environments.



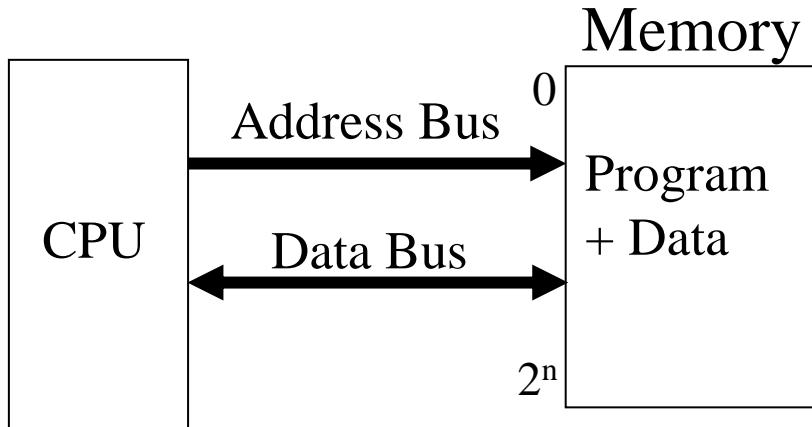
RISC vs CISC

4. The most important characteristic of the RISC processor is that more than 99% of instructions are executed with only one clock cycle, in contrast to CISC instructions.

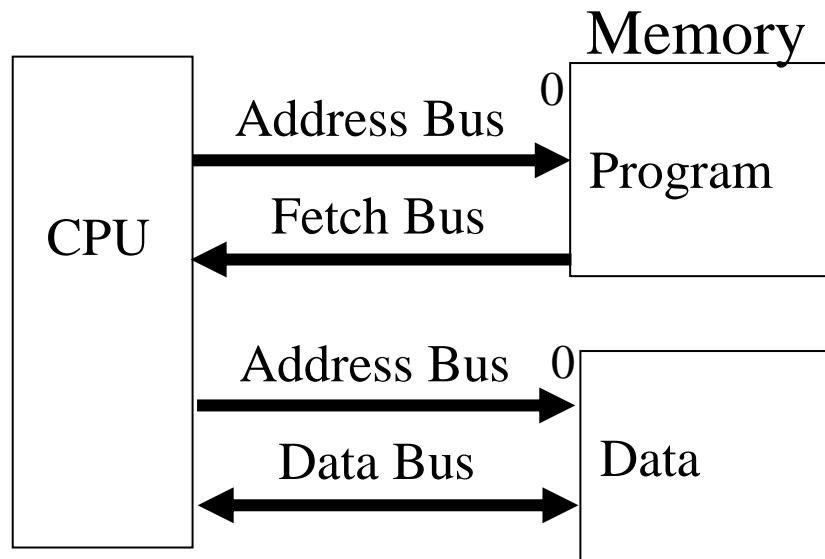
5. RISC processors have separate buses for data and code.



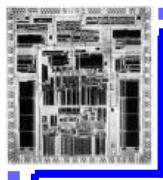
Microcontroller Architectures



Von Neumann
Architecture in CISC



Harvard
Architecture in RISC



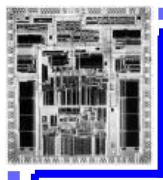
RISC vs CISC

6. Because CISC has such a large number of instructions, each with so many different addressing modes, microinstructions (microcode) are used to implement them.

RISC instructions, however, due to the small set of instructions, are implemented using the hardwire method.

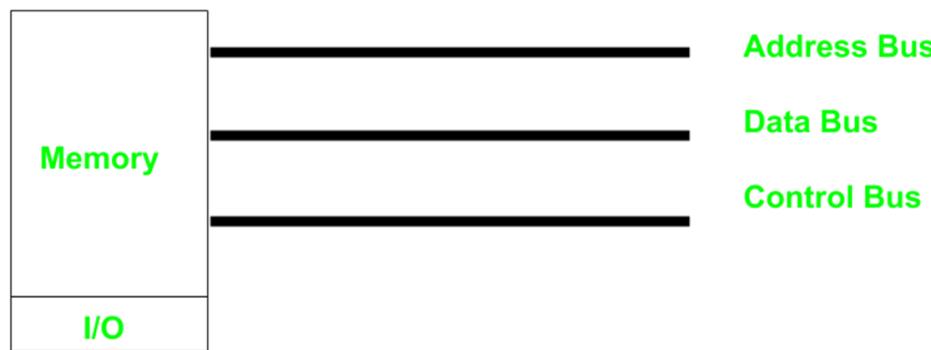
7. RISC uses load/ store architecture. In CISC microprocessors, data can be manipulated while it is still in memory.

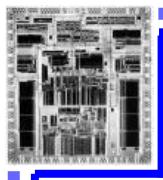
8. Memory Mapped IO in RISC and IO mapped IO in CISC



RISC vs CISC

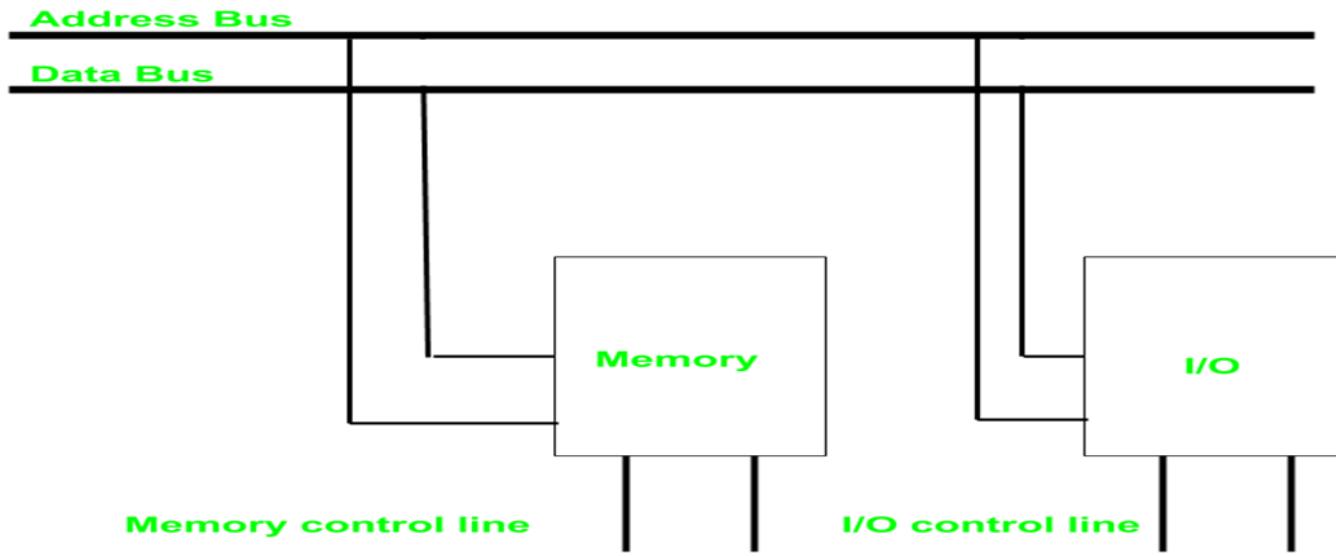
- **Memory Mapped I/O**
- In this case every bus is common due to which the same set of instructions work for memory and I/O.



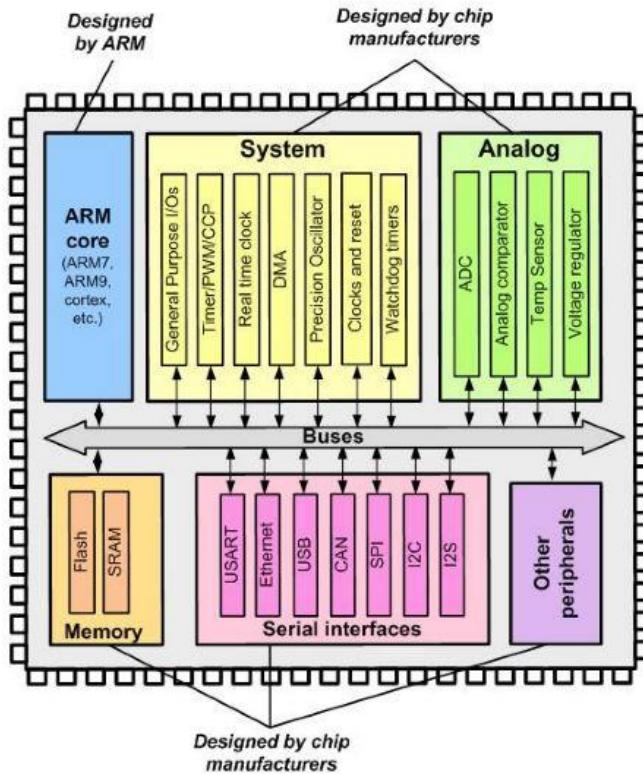


RISC vs CISC

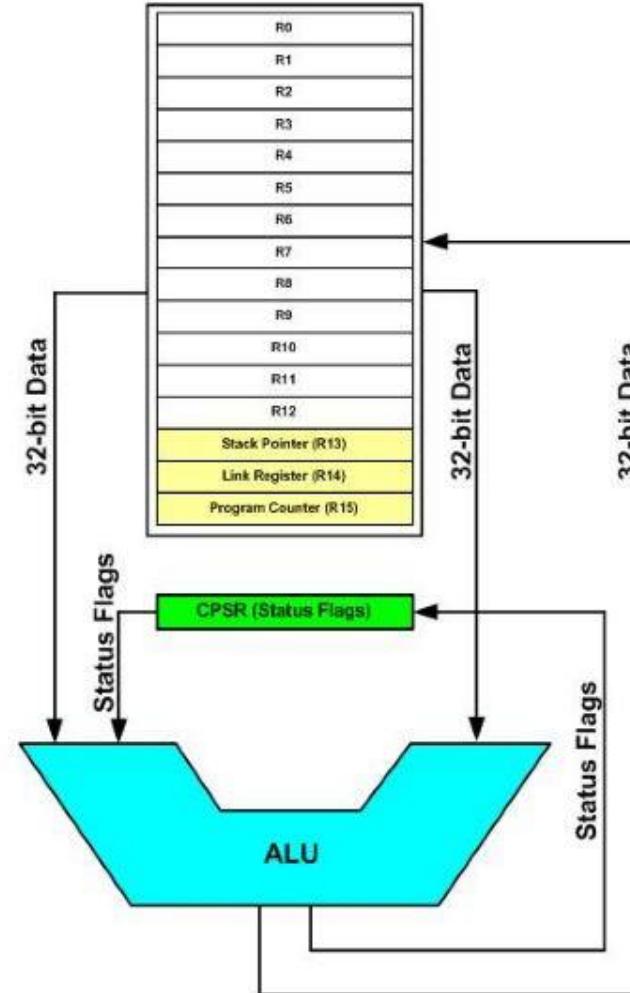
IO mapped IO (Isolated I/O) In which we have common bus(data and address) for I/O and memory but separate read and write control lines for I/O.

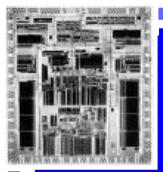


RISC vs CISC



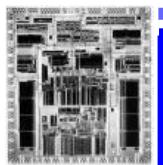
N,Z,C,V flags
Bit No. 31, 30, 29, 28



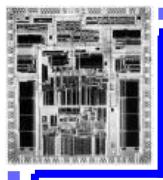


Addressing Modes

The way in which an operand is specified in an instruction is called Addressing Mode



Addressing Modes

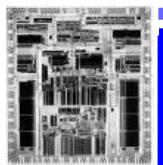


Addressing modes

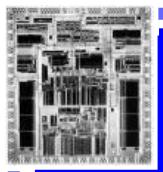
Indexed Addressing Mode	Syntax	Pointing Location in Memory	Rm Value After Execution
Preindex	LDR Rd, [Rm,#k]	Rm+#k	Rm
Preindex with WB*	LDR Rd, [Rm,#k]!	Rm+#k	Rm + #k
Postindex	LDR Rd, [Rm],#k	Rm	Rm + #k

*WB means Writeback

** Rd and Rm are any of registers and #k is a signed 12-bit immediate value between -4095 and +4095



Addressing Modes



Load/Store Byte/Half Word

LDRB – Load Byte

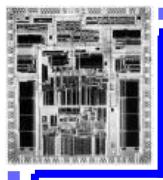
LDRH – Load Halfword

LDRSB – Load Signed byte

LDRSH – Load Signed Halfword

STRH – Store Halfword

Note : For all these instructions – Indirect/Indexed addressing modes are applicable)



MOV instruction

MOV Rd, Rn

MOV Rd, #0x12

MOVW Rd, #0x1234 (Move Word. i.e to the Lower 16 bit)

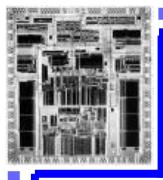
MOVT Rd, #0x1234 (Move Top. i.e. to higher 16 bit)

MVN Rd, Rn (Move Negative – 1's compliment)

MVN Rd, #0x12

MSR Special_Function_Reg, Rn (Move SFR from Register)

MRS Rn, Special_Function_Reg (Move Register from SFR)



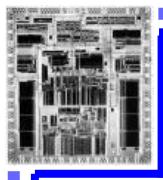
ADD instruction

ADD Rd, Rn, opr2 ; $Rd \leftarrow Rn + opr2$ (addition)

ADC Rd, Rn, opr2; $Rd \leftarrow Rn + opr2 + C$

Flags not affected.

Use S suffix to update flags: ADDS, ADCS



SUBTRACT

SUB Rd, Rn, opr2 ; $Rd \leftarrow Rn - opr2$ (**Subtract**)

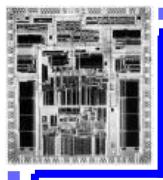
SBC Rd, Rn, opr2; $Rd \leftarrow Rn - opr2 - (1 - C)$ (**Subtract with Carry**)

RSB Rd, Rn, opr2 ; $Rd \leftarrow opr2 - Rn$ (**Reverse Subtract**)

RSC Rd, Rn, opr2 ; $Rd \leftarrow opr2 - Rn - (1 - C)$ (**Reverse Subtract with Carry**)

Flags not affected.

Use S suffix to update flags : SUBS, SBCS, RSBS, RSCS



MULTIPLICATION

MUL Rd, Rn, Rm ; $Rd = Rn \times Rm$ (**Multiply**)

MLA Rd, Rs1, Rs2, Rs3 ; $Rd = (Rs1 \times Rs2) + Rs3$ (**Multiply and Accumulate**)

MLS Rd, Rm, Rs, Rn ; $Rd = Rn - (Rs \times Rm)$ (**Multiply and Subtract**)

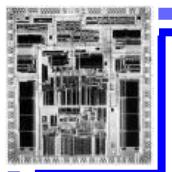
UMULL RdLo, RdHi, Rn, Rm ; $RdHi:RdLo = Rm \times Rn$ (**Unsigned Multiply Long**)

UMLAL RdLo, RdHi, Rn, Rm ; $RdHi:RdLo = (Rm \times Rn) + (RdHi:RdLo)$ (**Unsigned Multiply and Accumulate Long**)

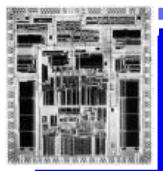
SMULL Rdlo, Rdhi, Rn, Rm ; $Rdhi:Rdlo = Rm \times Rn$ (**Signed Multiply Long**)

SMLAL Rdlo, Rdhi, Rn, Rm ; $Rdhi:Rdlo = (Rm \times Rn) + (Rdhi:Rdlo)$ (**Signed Multiply and Accumulate Long**)

Flags not affected.



MULTIPLICATION



LOGICAL INSTRUCTIONS

AND Rd, Rn, Op2

;Rd = Rn ANDed Op2

ORR Rd, Rn, Op2

;Rd = Rn ORed with 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 update flags

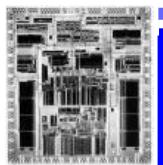
TEQ Rn, Op2

;performs Rn Ex-OR Op2

TST Rn, Op2

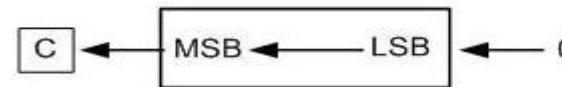
;performs Rn AND Op2

Flags N,Z affected

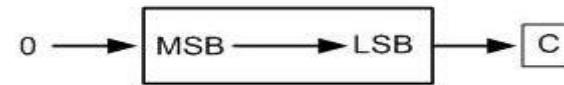


SHIFT AND ROTATE

LSL Rd, Rn, Op2

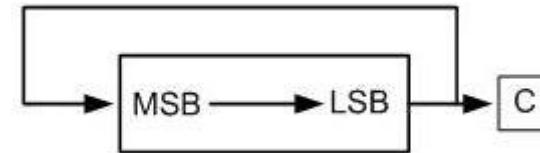


LSR Rd, Rn, Op2



ASR Rd, Rn, Op2

ROR Rd, Rn, Op2

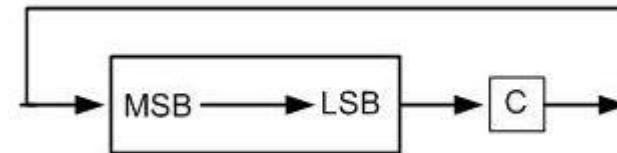


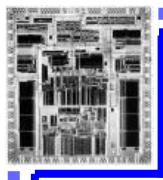
RRX Rd, Rm

;Rd = rotate Rm right 1 bit position

Flags not affected.

Use S suffix to update flags





SHIFT AND ROTATE

BCD addition

```
AREA      MYCODE, CODE, READONLY
```

```
ENTRY
```

```
EXPORT Reset_Handler
```

```
Reset_Handler
```

```
;::::::::::;User Code Starts from the next line;:::
```

```
ldr r0,=res  
ldr r3,=num1  
ldr r4,=num2  
ldr r1,[r3]  
ldr r2,[r4]
```

```
mov r0,r1
```

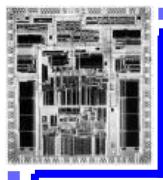
```
mov r5,#0
```

```
mov r8,#8
```

```
loop    mov r3,r1  
        mov r4,r2  
        and r3,#mask  
        and r4,#mask  
        add r6,r3,r4  
        add r6,r6,r5  
        cmp r6,#0x0a  
        blo rcarryl
```

BCD addition

```
        mov r5,#1
        sub r6,#0x0a
        b next
rcarryl mov r5,#0
next    lsr r1, #4
        lsr r2, #4
        orr r7,r6
        ror r7,#4
        sub r8,#1
        teq r8,#0
        bne loop
nextl   str r7,[r0],#4
        str r5,[r0]
stop    b stop
num1    DCD 0x99999999
num2    DCD 0x99999999
mask    equ 0x0f
        AREA data1,DATA
res     DCD 0,0
        end
```



Compare Instructions

CMP

Compare

Flags: Affected: V, N, Z, C.

Format:

CMP Rn,Op2

;sets flags as if

"Rn-Op2"

CMN

Compare Negative

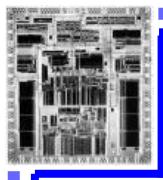
Flags: Affected: V, N, Z,C.

Format:

CMN Rn,Op2

;sets flags as if "Rn +

Op2"



BRANCH INSTRUCTIONS

B

Branch (unconditional jump)

Flags: Unchanged.

Format:

B target ;jump to target address

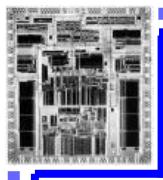
Bxx

Branch Conditional

Flags: Unaffected.

Format:

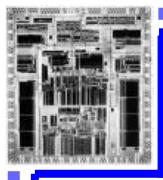
Bxx target ;jump to target upon
condition



BRANCH INSTRUCTIONS

Instruction	Condition
BCS Branch if Carry Set	jump if C=1
BCC Branch if Carry Clear	jump if C=0
BEQ Branch if Equal	jump if Z=1
BNE Branch if Not Equal	jump if Z=0
BMI Branch if Minus/Negative	jump if N=1
BPL Branch if Plus/Positive	jump if N=0
BVS Branch if Overflow	jump if V=1
BVC Branch if No overflow	jump if V=0

Based on single flag

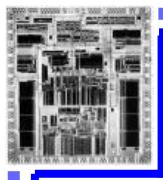


BRANCH INSTRUCTIONS

"B condition" where the condition refers to the comparison of unsigned numbers. After a compare (CMP Rn,Op2) instruction is executed, C and Z indicate the result of the comparison, as follows:

	C	Z
Rn > Op2	1	0
Rn = Op2	1	1
Rn < Op2	0	0

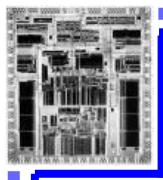
Based on multiple flags



BRANCH INSTRUCTIONS

Instruction	Condition
BHI	Branch if Higher jump if C=1 and Z=0
BEQ	Branch if Equal jump if C=1 and Z=1
BLS	Branch if Lower or same jump if C=0 or Z=1
BLO	Brach if lower C=0 and Z=0
BHS	Brach if Higher or same C=1 or Z=1

For unsigned comparison



BRANCH INSTRUCTIONS

Rn > Op2	V=N or Z=0
Rn = Op2	Z=1
Rn < Op2	V inverse of N

For the signed comparison

Instruction	
BGE	Branch Greater or Equal
BLT	Branch Less than
BGT	Branch Greater than
BLE	Branch Less or Equal
BEQ	Branch if Equal

V=N or Z=1

V is inverse N and Z=0

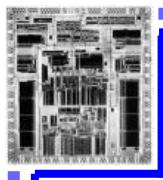
V=N and Z=0

V is inverse N or Z=1

jump if Z = 1

BCD to HEX

```
AREA      MYCODE, CODE, READONLY  
  
ENTRY  
  
EXPORT Reset_Handler  
  
Reset_Handler  
    ldr r0,=bcd  
    ldr r1,[r0]  
    and r2,r1,#0x0000000f0  
    lsr r2,r2,#4  
    and r3,r1,#0x0000000f  
    mov r4,#10 ; or ox0A  
    mla r5,r2,r4,r3  
    ldr r0,=hex  
    str r5,[r0]  
  
stop    B stop  
  
bcd     DCD 0x98  
        AREA data1,DATA  
hex     dcd 0  
        end
```



Function Call & Return

BL Branch with Link (this is Call instruction)

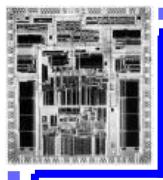
Flags: Unchanged.

Format: BL Subroutine_Addr ;transfer

**BX Branch Indirect (BX LR is used for
Return)**

Flags: Unchanged.

Format: BX Rm ;BX LR is used for Return
from a subroutine



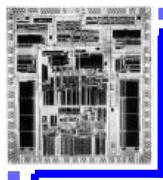
Function Call & Return

```
BL HEX_BCD ; call HEX_BCD  
MOV R0,R1
```

HERE B HERE

```
HEX_BCD MOV R4,R5
```

BX LR ; Return



PUSH and POP

PUSH **PUSH register onto stack**

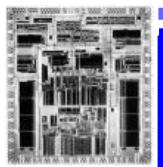
Flags: Unaffected.

Format: PUSH {reg_list} ;PUSH reg_list
onto stack

POP **POP register from Stack**

Flags: Unaffected.

Format: POP {reg_list} ;reg_reg = words off top
of stack



PUSH and POP

PUSH {R1,R3-R5}; same as PUSH {R1,R3,R4,R5}

POP {R1,R3-R5}

Example:

LDR R13,=0x10000010

LDR R1,=0x12345678

LDR R3,=0x89ABCDEF

LDR R4,=-2

LDR R5,=0x98765432

PUSH {R1,R3-R5}

POP {R2,R6-R8}

R13 = 0x10000000

R2= 0x12345678

R6=0x89ABCDEF...

→ 0x10000010

0x100000F	98
0x100000E	76
0x100000D	54
0x100000C	32
0x100000B	FF
0x100000A	FF
0x1000009	FF
0x1000008	FE
0x1000007	89
0x1000006	AB
0x1000005	CD
0x1000004	EF
0x1000003	12
0x1000002	34
0x1000001	56
0x1000000	78

BCD addition

```
AREA      MYCODE, CODE, READONLY
```

```
ENTRY
```

```
EXPORT Reset_Handler
```

```
Reset_Handler
```

```
;::::::::::;User Code Starts from the next line;:::
```

```
ldr r0,=res  
ldr r3,=num1  
ldr r4,=num2  
ldr r1,[r3]  
ldr r2,[r4]
```

```
mov r0,r1
```

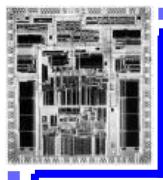
```
mov r5,#0
```

```
mov r8,#8
```

```
loop    mov r3,r1  
        mov r4,r2  
        and r3,#mask  
        and r4,#mask  
        add r6,r3,r4  
        add r6,r6,r5  
        cmp r6,#0x0a  
        blo rcarryl
```

BCD addition

```
        mov r5,#1
        sub r6,#0x0a
        b next
rcarryl mov r5,#0
next    lsr r1, #4
        lsr r2, #4
        orr r7,r6
        ror r7,#4
        sub r8,#1
        teq r8,#0
        bne loop
nextl   str r7,[r0],#4
        str r5,[r0]
stop    b stop
num1    DCD 0x99999999
num2    DCD 0x99999999
mask    equ 0x0f
        AREA data1,DATA
res     DCD 0,0
        end
```

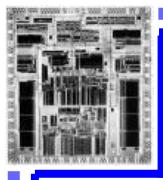


HEX to BCD

AREA Example4, CODE, READONLY ENTRY

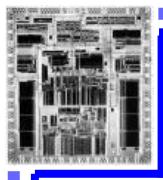
Reset_Handler

```
ldr r0,=hex
ldr r2,=rem
mov r5,#0
mov r7,#32
ldr r1,[r0]
up2    bl divide
      cmp r1,#0
      bne up2
      ldr r0,=bcd
      lsr r5, r7
      str r5,[r0]
stop    B stop
```



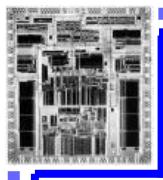
HEX to BCD

```
divide    mov r3,#0
up1        cmp r1,#0x0a
            b lo down|
            sub r1,#0x0a
            add r3,#1
            b up1
down
            orr r5, r1
            ror r5,#4
            mov r1,r3
            sub r7,#4
            bx lr
hex       DCD 0xffffe
          AREA data1,DATA
bcd      DCD 0
```



Convert NEG to POSITIVE in array

```
LDR R0, =ARRAY  
MOV R1,#10  
UP LDR R2, [R0], #4  
CMP R2, #0  
RSBLT R2, #0  
STRLT R2, [R0, #-4]  
SUB R1, #0  
TEQ R1, #0  
BNE UP
```

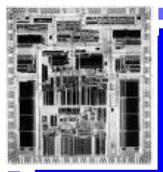


Factorial using Recursion

```
AREA Example4, CODE, READONLY  
ENTRY
```

```
Reset_Handler
```

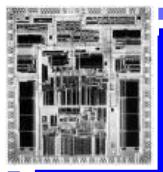
```
    ldr r1,num  
    ldr r13,=0x10001000  
    bl fact1  
    ldr r1,=fact  
    str r2,[r1]  
  
stop      b stop  
fact1     cmp r1,#1  
          beq exit  
          push {r1}  
          push {lr}  
          sub r1, #1  
          bl fact1
```



Factorial using Recursion

```
pop {lr}
pop {r1}
mul r2,r1,r2
bx lr
exit      mov r2,#1
          bx lr
num      DCD 0x07
```

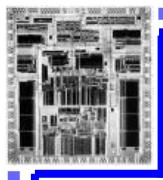
```
AREA data1,DATA
fact      DCD 0
          end
```



GCD

	LDR R0, =NUM1
	LDR R1, =NUM2
	LDR R0,[R0]
	LDR R1,[R1]
UP	CMP R0, R1
	BEQ EXIT
	SUBHI R0,R1
	SUBL0 R1,R0
	B UP
EXIT	LDR R2,=GCD
	STR R0, [R2]

```
While (a != b)
{
    If (a > b)
        a = a-b;
    Else if (b > a)
        b = b-a;
}
```



Load and Store Multiple

LDM Load Multiple registers

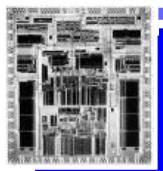
Flags: Unaffected.

Format: LDM Rn, {Rx,Ry,...}

STM Store Multiple

Flags: Unaffected.

Format: STM Rn, {Rx,Ry,...}



Load and Store Multiple

LDR R1, =0x10000000

LDM R1, {R2,R4,R6}

R2 = 0x32547698

R4 = 0xFFFFFFF

R6 = 0xEFCDAB89

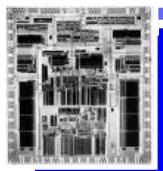
R1= 0x10000000

In case of !

LDM R1!, {R2,R4,R6}

R1=0x1000000C

0x10000000	98
0x10000001	76
0x10000002	54
0x10000003	32
0x10000004	FF
0x10000005	FF
0x10000006	FF
0x10000007	FE
0x10000008	89
0x10000009	AB
0x1000000A	CD
0x1000000B	EF
0x1000000C	
0x1000000D	
0x1000000E	
0x1000000F	

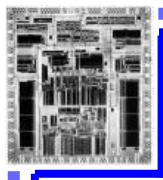


Load and Store Multiple

LDR R1, =0x10000000
LDR R2, = 0x32547698
LDR R4, =0xFFFFFFF
LDR R6, = 0xEFCDAB89
STM R1, {R2,R4,R6}
R1= 0x10000000

In case of !
STM R1!, {R2,R4,R6}
R1=0x1000000C

0x10000000	98
0x10000001	76
0x10000002	54
0x10000003	32
0x10000004	FF
0x10000005	FF
0x10000006	FF
0x10000007	FE
0x10000008	89
0x10000009	AB
0x1000000A	CD
0x1000000B	EF
0x1000000C	
0x1000000D	
0x1000000E	
0x1000000F	



Load and Store Multiple

LDMDB

Load Multiple registers and

Decrement Before each access

Flags: Unaffected.

Format:

LDMDB Rn,{Rx,Ry,...}

STMDB

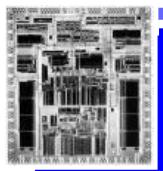
Store Multiple register and

Decrement Before

Flags: Unaffected.

Format:

STMDB Rn,{Rx,Ry,...}



Load and Store Multiple

LDR R1, =0x1000000C

LDMDB R1, {R2,R4,R6}

R2 = 0x32547698

R4 = 0xFFFFFFF

R6 = 0xEFCDAB89

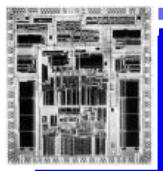
R1= 0x1000000C

In case of !

LDMDB R1!, {R2,R4,R6}

R1=0x10000000

0x10000000	98
0x10000001	76
0x10000002	54
0x10000003	32
0x10000004	FF
0x10000005	FF
0x10000006	FF
0x10000007	FE
0x10000008	89
0x10000009	AB
0x1000000A	CD
0x1000000B	EF
0x1000000C	
0x1000000D	
0x1000000E	
0x1000000F	

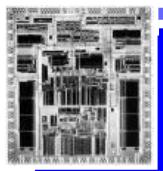


Load and Store Multiple

LDR R1, =0x1000000C
LDR R2, = 0x32547698
LDR R4, =0xFFFFFFF
LDR R6, = 0x32547698
STMDB R1, {R2,R4,R6}
R1= 0x1000000C

In case of !
STMDB R1!, {R2,R4,R6}
R1=0x10000000

0x10000000	98
0x10000001	76
0x10000002	54
0x10000003	32
0x10000004	FF
0x10000005	FF
0x10000006	FF
0x10000007	FE
0x10000008	89
0x10000009	AB
0x1000000A	CD
0x1000000B	EF
0x1000000C	
0x1000000D	
0x1000000E	
0x1000000F	



Fully Ascending Stack

LDR R13, =0x10000000

LDR R2, = 0x32547698

LDR R4, =0xFFFFFFF

LDR R6, = 0x32547698

STM R13!, {R2,R4,R6}

R13= 0x1000000C

MOV R2, #0

MOV R4, #0

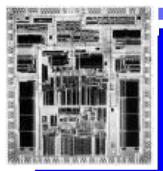
MOV R6, #0

LDMDB R13!, {R2,R4,R6}

R13=0x10000000

SP increments for PUSH
SP decrements for POP

0x10000000	98
0x10000001	76
0x10000002	54
0x10000003	32
0x10000004	FF
0x10000005	FF
0x10000006	FF
0x10000007	FE
0x10000008	89
0x10000009	AB
0x1000000A	CD
0x1000000B	EF
0x1000000C	
0x1000000D	
0x1000000E	
0x1000000F	



Fully Descending Stack

```
LDR R13, =0x1000000C  
LDR R2, = 0x32547698  
LDR R4, =0xFFFFFFF  
LDR R6, = 0x32547698  
STMDB R13!, {R2,R4,R6}  
R13= 0x10000000  
MOV R2, #0  
MOV R4, #0  
MOV R6, #0  
LDM R13!, {R2,R4,R6}  
R13=0x1000000C
```

0x10000000	98
0x10000001	76
0x10000002	54
0x10000003	32
0x10000004	FF
0x10000005	FF
0x10000006	FF
0x10000007	FE
0x10000008	89
0x10000009	AB
0x1000000A	CD
0x1000000B	EF
0x1000000C	
0x1000000D	
0x1000000E	
0x1000000F	

SP decrements for PUSH
SP increments for POP



ARM LPC1768 ASM Programs



ARM – ASM Programming

Bubble Sort



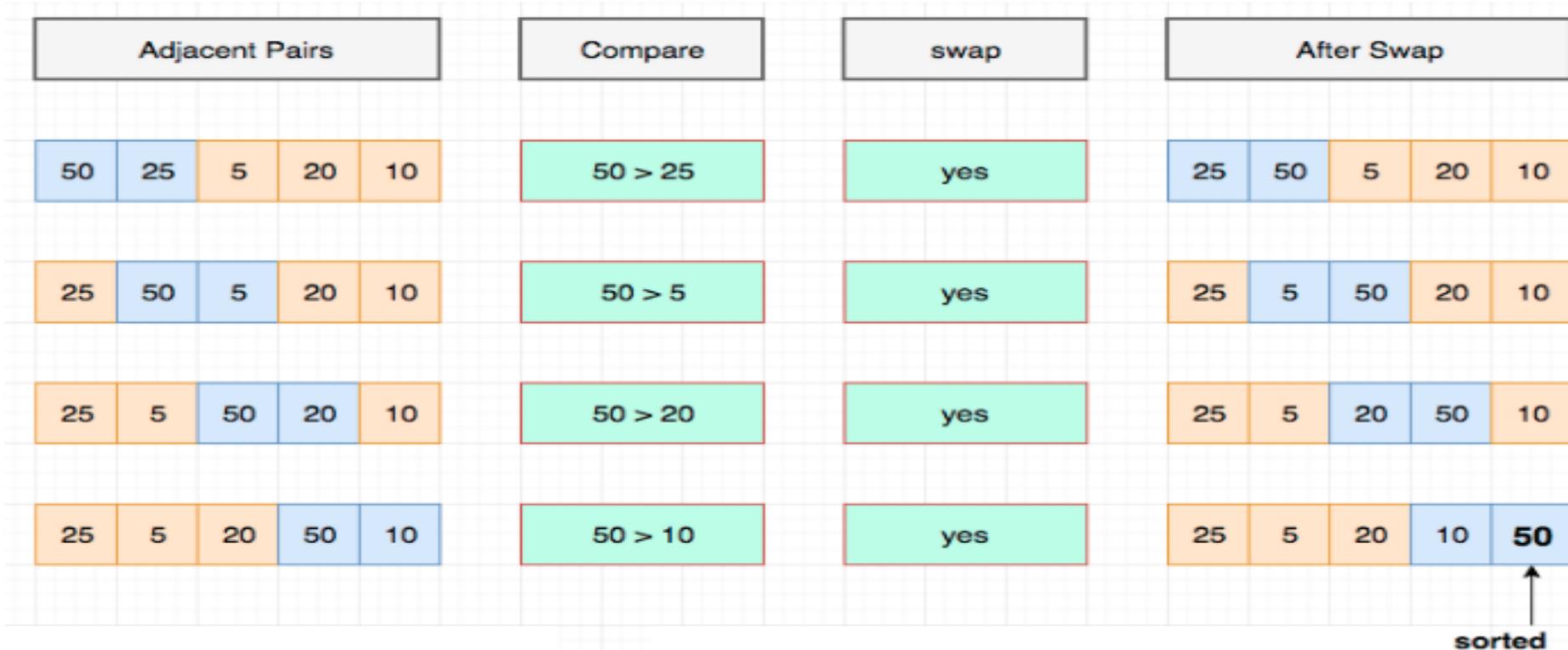
Bubble Sort Algorithm

- Two successive elements are compared with each other, and if the left element is larger than the right one, they are swapped.
- These comparison and swap operations are performed from left to right across all elements. Therefore, after the first pass, the largest element is positioned on the far right.
- Ex: Let's take an array of 5 elements = {50, 25, 5, 20, 10}



Bubble Sort Algorithm

Step 1

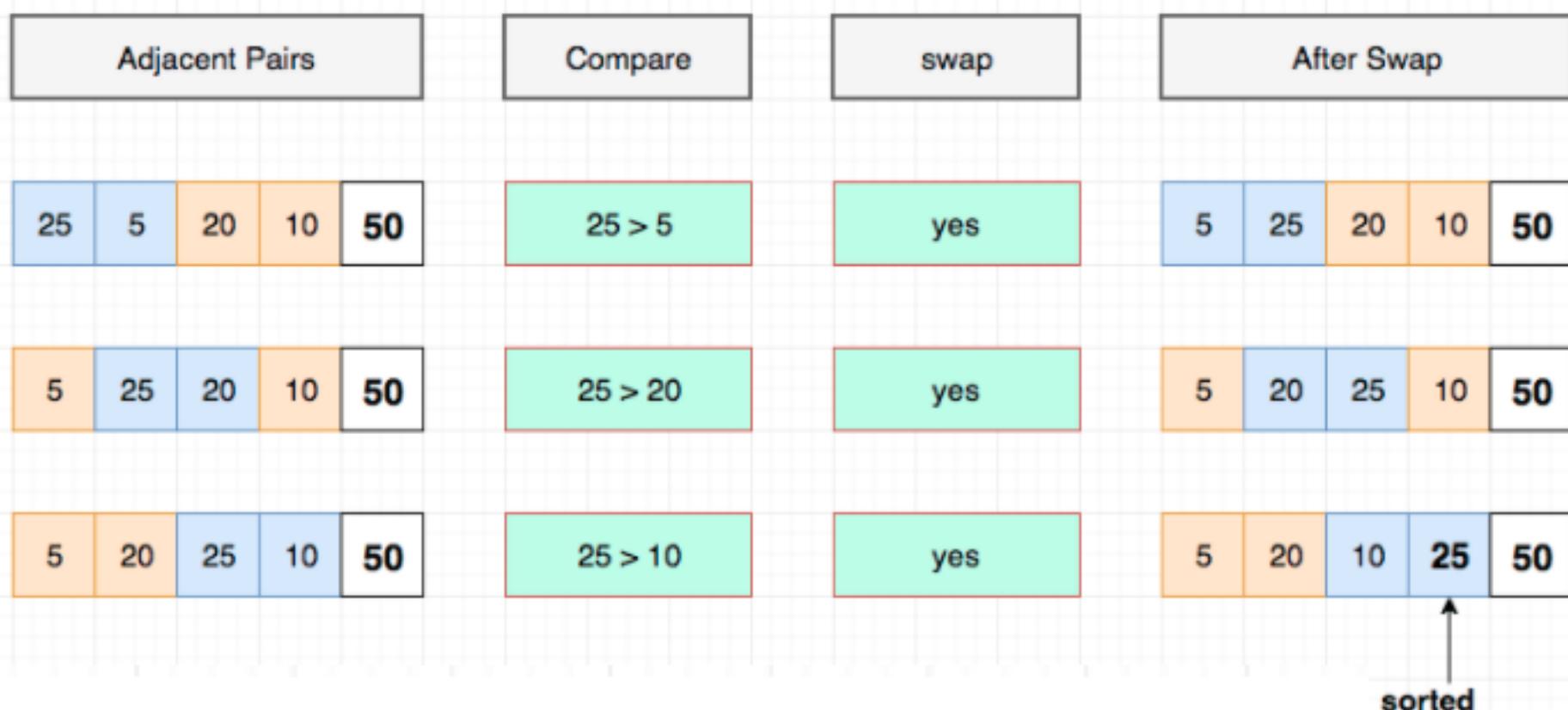


- We can notice that one element has been sorted after the above process.
- In general, to sort N element using bubble sort, we need to do the same process N-1 times.
- From next iteration onwards, we can skip the sorted elements. i.e. 50



Bubble Sort Algorithm

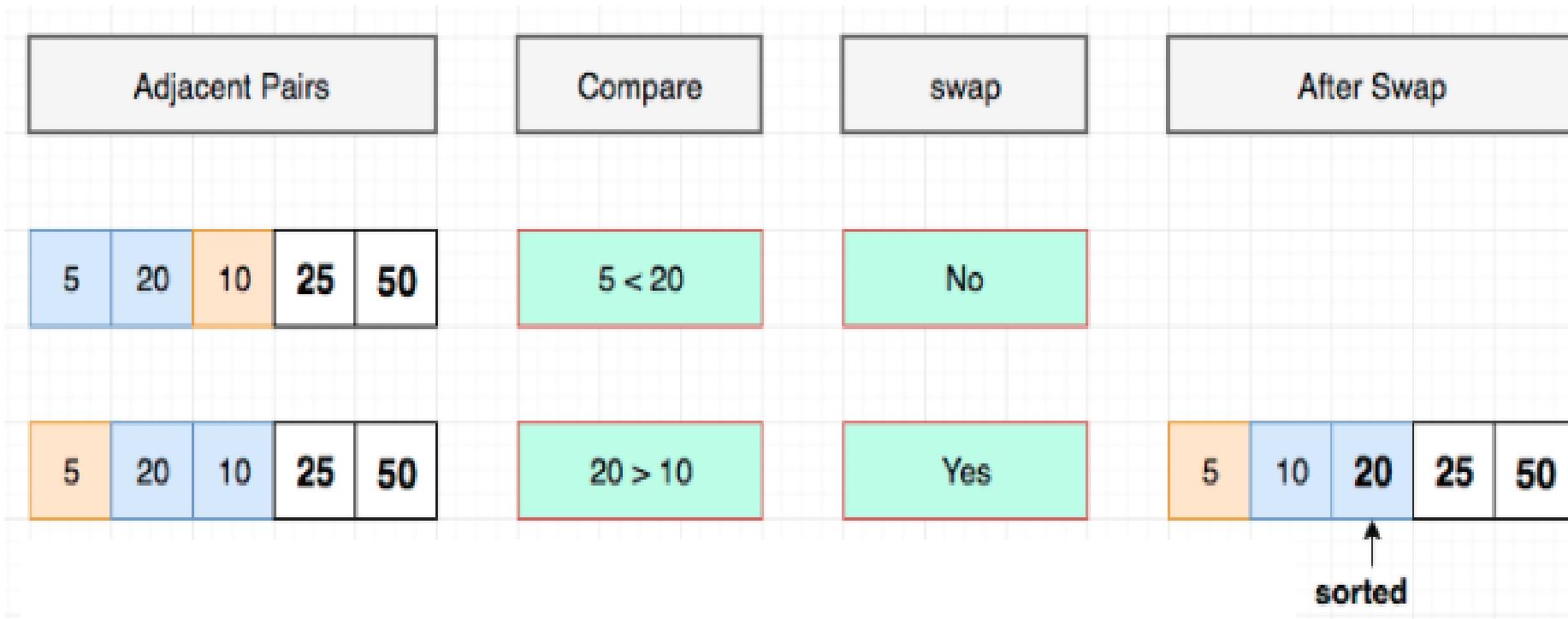
Step 2





Bubble Sort Algorithm

Step 3





Bubble Sort Algorithm

Step 4

Adjacent Pairs

5	10	20	25	50
---	----	----	----	----

Compare

$5 < 10$

swap

No

After Swap

5	10	20	25	50
---	----	----	----	----



ARM – ASM Programming

Hex to ASCII & ASCII to Hex Conversion



Hexadecimal System (Hex System)

- The hexadecimal system (shortly hex), uses the **number 16** as its **base (radix)**.
- As a base-16 numeral system, it uses 16 symbols:- 10 decimal digits (**0, 1, 2, 3, 4, 5, 6, 7, 8, 9**) and the first six letters of the English alphabet (**A, B, C, D, E, F**).
- Hex is used in **mathematics** and **information technologies** as a more **friendly way to represent binary numbers**.
- Each hex digit represents four binary digits; therefore, hex is a language to write binary in an abbreviated form.
- **Application**:- In html programming, colours can be represented by a 6-digit hexadecimal number: FFFFFF represents white whereas 000000 represents black.



ASCII Text

- ASCII abbreviated as **American Standard Code for Information Interchange**.
- ASCII is one of the most common character encoding standards **widely used in electronic communication for conveying text**.
- As computers can only understand numbers, the **ASCII code represents text (characters) with different numbers**. This is how a computer ‘understands’ and shows text.
- ASCII is based on 128 characters. These are the 26 letters of the English alphabet (both in lower and upper cases); numbers from 0 to 9; and various punctuation marks.
- In the ASCII code, each of these characters are assigned a decimal number from 0 to 127.
- Ex:- ASCII representation of upper case A is 65 and the lower case a is 97.



ASCII Table

Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	'
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D	36	24	\$	68	44	D	100	64	d
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f
7	7	Bell	BEL	CTRL-G	39	27	.	71	47	G	103	67	g
8	8	Backspace	BS	CTRL-H	40	28	(72	48	H	104	68	h
9	9	Horizontal tab	HT	CTRL-I	41	29)	73	49	I	105	69	i
10	0A	Line feed	LF	CTRL-J	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	VT	CTRL-K	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	FF	CTRL-L	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage feed	CR	CTRL-M	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	SO	CTRL-N	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	SI	CTRL-O	47	2F	/	79	4F	O	111	6F	o
16	10	Data line escape	DLE	CTRL-P	48	30	0	80	50	P	112	70	p
17	11	Device control 1	DC1	CTRL-Q	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	DC2	CTRL-R	50	32	2	82	52	R	114	72	r
19	13	Device control 3	DC3	CTRL-S	51	33	3	83	53	S	115	73	s
20	14	Device control 4	DC4	CTRL-T	52	34	4	84	54	T	116	74	t
21	15	Neg acknowledge	NAK	CTRL-U	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	SYN	CTRL-V	54	36	6	86	56	V	118	76	v
23	17	End of xmit block	ETB	CTRL-W	55	37	7	87	57	W	119	77	w
24	18	Cancel	CAN	CTRL-X	56	38	8	88	58	X	120	78	x
25	19	End of medium	EM	CTRL-Y	57	39	9	89	59	Y	121	79	y
26	1A	Substitute	SUB	CTRL-Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	ESC	CTRL-[59	3B	:	91	5B	{	123	7B	{
28	1C	File separator	FS	CTRL-\	60	3C	<	92	5C	\	124	7C	
29	1D	Group separator	GS	CTRL-]	61	3D	=	93	5D]	125	7D	}
30	1E	Record separator	RS	CTRL-^	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	US	CTRL-_	63	3F	?	95	5F	_	127	7F	DEL



ASCII Encoding of a Text using Hex Codes

HEX Code	ASCII String
68 65 6C 6C 6F	hello
47 6F 6F 64 20 6D 6F 72 6E 69 6E 67	Good morning
4D 41 48 45	MAHE
4D 61 69 70 61 6C	Manipal



ASCII - Hex Mapping

Hexadecimal Number	ASCII number (Hexadecimal)
0	30
1	31
2	32
3	33
4	34
5	35
6	36
7	37
8	38
9	39
A	41
B	42
C	43
D	44
E	45
F	46



Steps to Convert Hexadecimal number to ASCII equivalent

- **Example 1.** Hexadecimal number =1

Corresponding ASCII value =Hexadecimal number + $30_{(16)}$
= $31_{(16)}$

- **Example 2.** Hexadecimal number =D

Corresponding ASCII value =Hexadecimal number + $37_{(16)}$
= $D_{(16)} + 37_{(16)}$
= $13_{(10)} + 55_{(10)}$
= $68_{(10)}$
= $44_{(16)}$



Steps to Convert ASCII to Hexadecimal equivalent number

- **Example 1:** ASCII number = $36_{(16)}$

$$\begin{aligned}\text{Corresponding Hexadecimal number} &= 36_{(16)} - 30_{(16)} \\ &= 6\end{aligned}$$

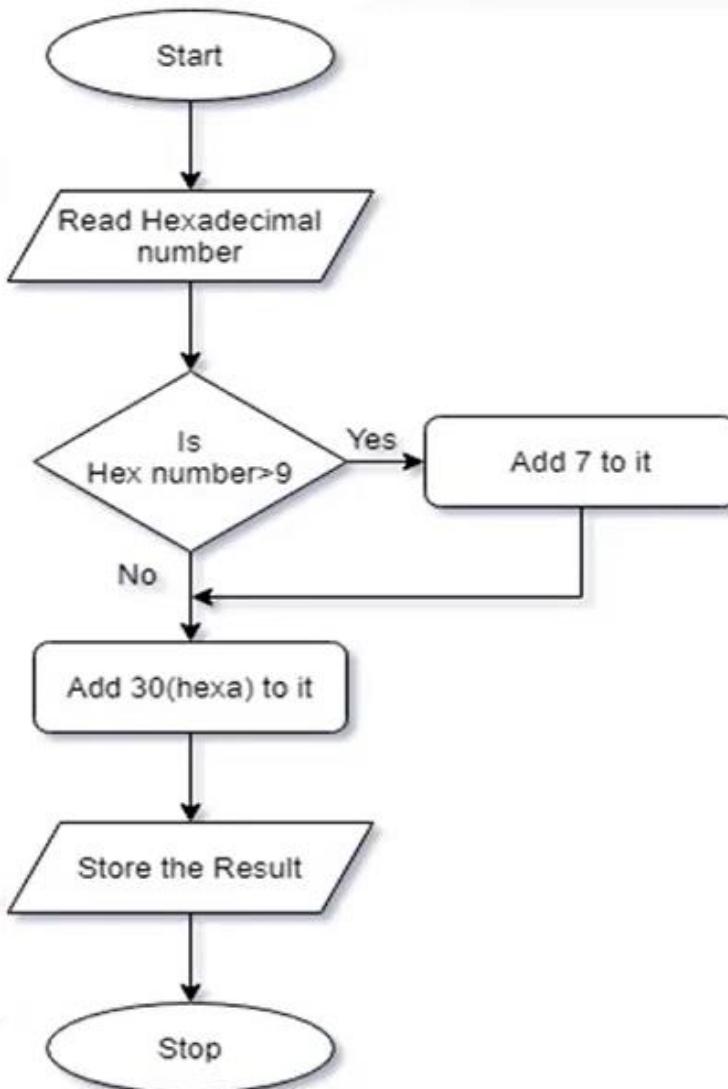
- **Example 2:** ASCII number = $45_{(16)}$

$$\begin{aligned}\text{Corresponding Hexadecimal number} &= 45_{(16)} - 37_{(16)} \\ &= 69_{(10)} - 55_{(10)} \\ &= 14_{(10)} = E_{(16)}\end{aligned}$$

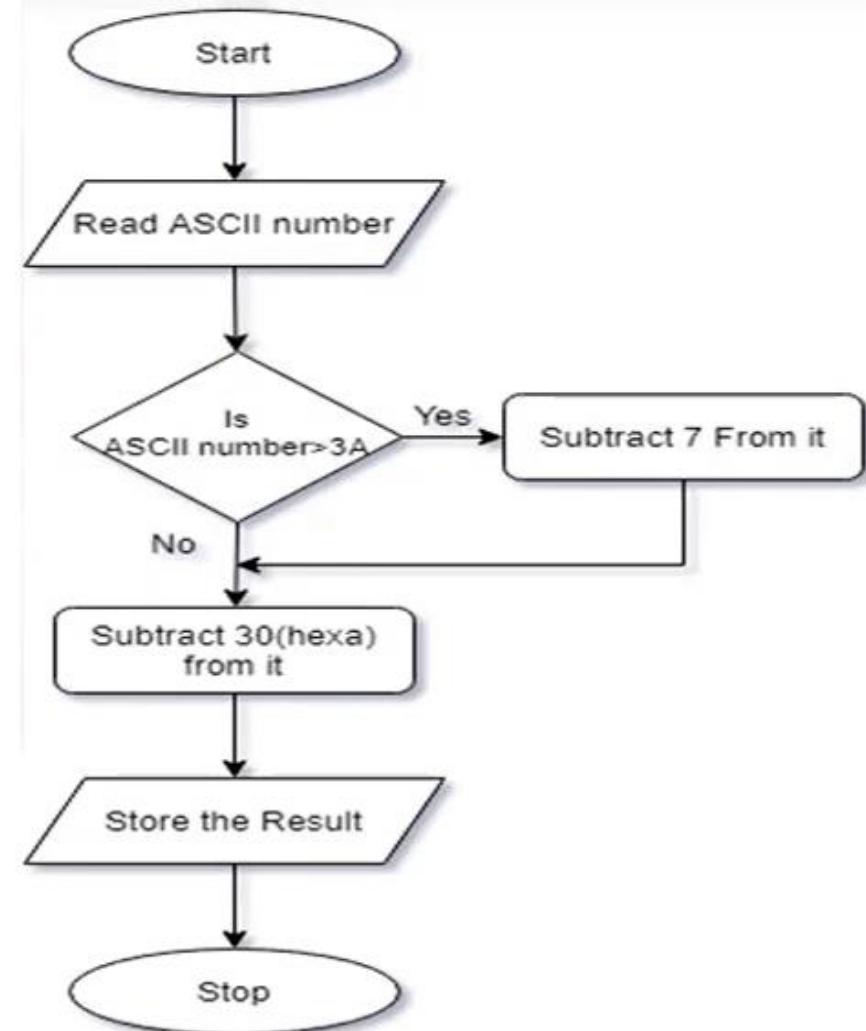


Flow Chart

Hexadecimal to ASCII



ASCII to Hexadecimal





Puzzle

- What is the **minimum number of square towels** that can be cut from a ream of cloth **20 meters in length** and **16 meters wide** without wasting any cloth ?
- The measurement of one side of required square towel is $\text{GCD}(20,16) = 4 \text{ meters}$.
- The area of ONE square towel is $4 \times 4 = 16 \text{ sqm}$
- The area of full ream = $20 \times 16 = 320 \text{ sqm}$
- Minimum number of towels possible is: $320/16 = 20 \text{ towels.}$



ARM – ASM Programming

GCD of Two Numbers



GCD of Two Numbers

- The Greatest Common Divisor (GCD) refers to the **greatest positive integer** that is a common divisor for a given set of positive integers.
- It is also termed as the Highest Common Factor (HCF) or the Greatest Common Factor (GCF).
- **Application:** Encryption Algorithms, Linear Algebra etc.

```
While (a != b)
{
    If (a > b)
        a = a-b;
    Else if (b > a)
        b = b-a;
}
```



ARM – ASM Programming

Fibonacci series

Fibonacci series

FIBONACCI SEQUENCE

A series of numbers, starting from 0 where every number is the sum of the two numbers preceding it.

0,1,1,2,3,5,8,13,21,34,55.... and so on

Named after

FIBONACCI

An Italian mathematician

Year 1202

The year it was first introduced to the western world in the book "Liber Abaci"

$$x_n = x_{n-1} + x_{n-2}$$

Mathematical formula



1.618

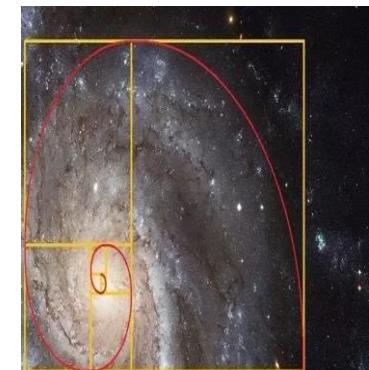
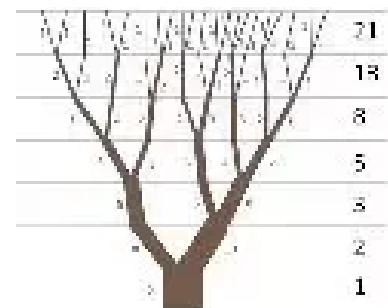
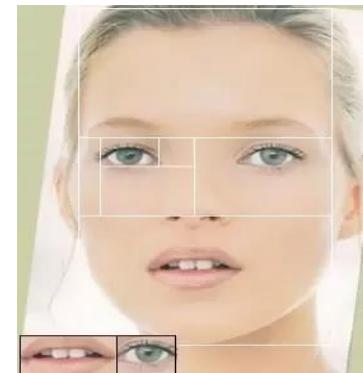
"Phi" or the "Golden Ratio"
The ratio of any two consequent numbers of the sequence

Nature's code

Because it is observed in several natural phenomena

mocomi.com

Copyright © 2016 www.mocomi.com



- Fibonacci numbers are used in Fibonacci heaps, which are a data structure that can be used to speed up some very practical algorithms.
- They are used in charting analysis (technical analysis) in stock trading to estimate the price fall and rise.



ARM – ASM Programming

Factorial of a Number



Factorial of a Number

- In programming terms, a recursive function can be defined as a routine that calls itself directly or indirectly.
- Types:
 - **Function calling itself(Direct way):** Here the function calls itself.
 - **Recursion using mutual function call(Indirect way):** Here, a function [funA()] can call another function [funB()] which in turn calls [funA()] which is the former function.
- **Application:** Encryption Algorithms.



Factorial

SP = 0x100D1000

x_1
 x_2
 x_3

```
Reset_Handler
    ldr r0, =05
    bl factorial
stop b stop
```

$j PC = y_1 \& R_{14} = x_3$

```
factorial
    push {r4,lr}
    ;push r4 and lr onto the top of the stack
    ;stack<=>smdmdb sp!, {r4,lr}

    y_1
    mov r4,r0          ;//make a copy of r0

    y_2
    cmp r0,#0          ;//compare r0 with 0

    y_3
    bne not_zero       ;//branch to not_zero if not equal
    ;//r0=1
    b last

    y_4
    not_zero
    mov r0,#1
    b last

    y_5
    not_zero
    sub r0,r0,#1
    bl factorial
    ;//call recursively factorial

    y_6
    mov r1,r4
    mul r0,r0,r1
    ;//r0=r0*r1
    ;//r0=1*2*3=6*4=24*5=120
    y_7
    last
    pop {r4,lr}
    ;//pop <=> ldmia sp! {r4,lr}
    y_8
    bx lr
```



Storage LE
R₄ = lower memory & LR = higher memory



ARM – ASM Programming

BCD to Hexadecimal Conversion

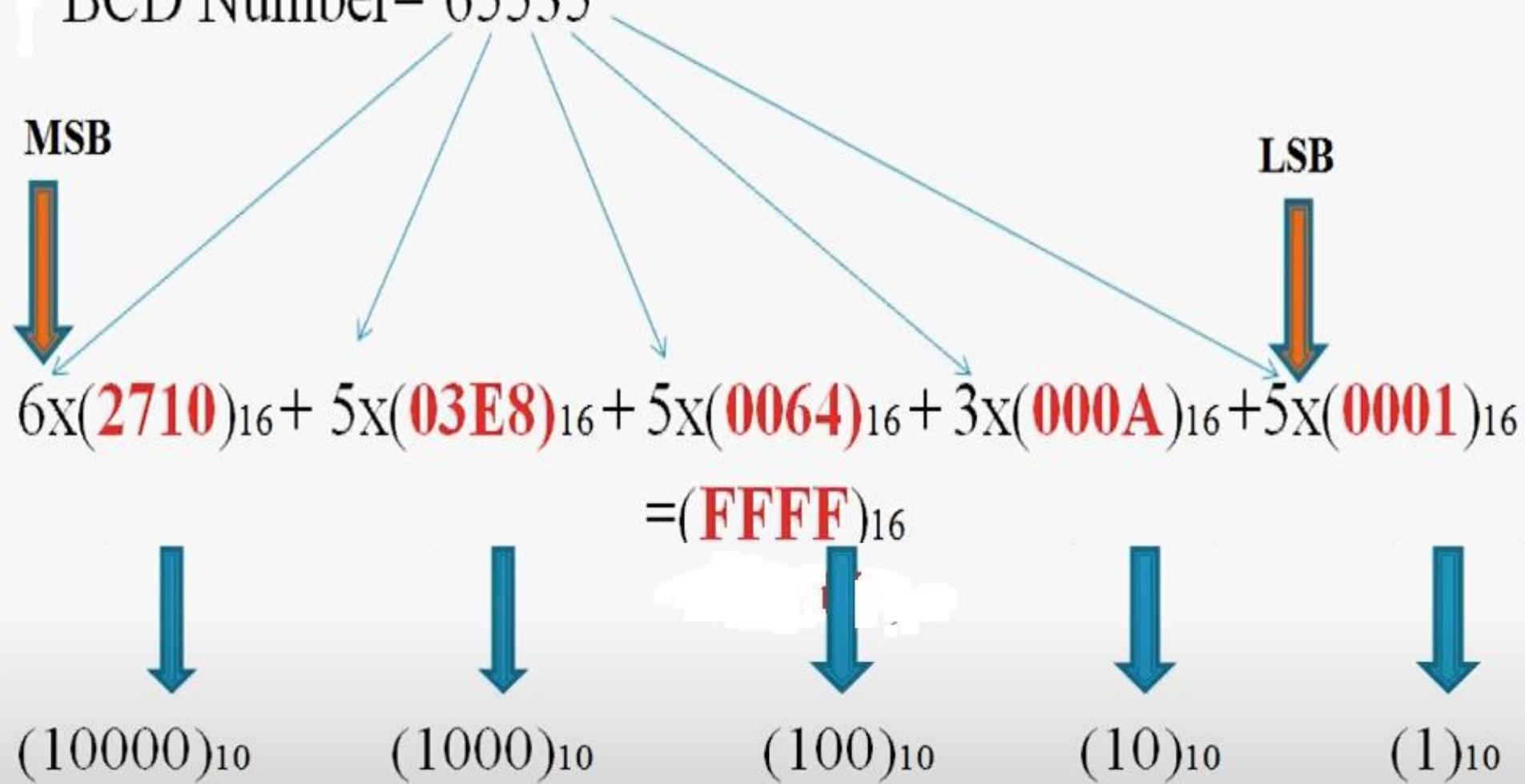


BCD - Hexadecimal

- In BCD each digit of number is coded separately using fixed number of bits usually 4.
- $29 = 0010\ 1001$

↓ ↓
2 9

BCD Number= 65535



Note: Here we need to perform positional multiplication.



ARM – ASM Programming

Hex-BCD





ARM – ASM Programming

Linear search



Linear Search

- Here, a sequential search is made over all items one by one.
- Every item is checked and if a match is found then that particular item is stored on to the data memory.

Linear Search









ARM LPC1768

External Hardware Interrupt



External HW interrupts

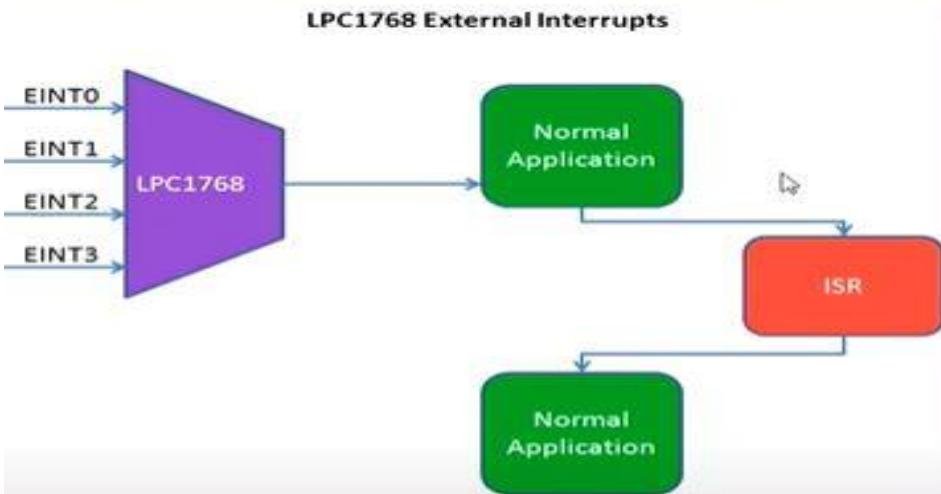
- What are Interrupts?!
- System Control block of Arm has SFR's to handle external hardware interrupts.
- LPC1768 has Four External Hardware Interrupts: EINT0 – EINT3 in general represented as EINTx Pins mapped to multifunctional pins of port 2 as shown

Port Pin	PINSEL_FUNC_0	PINSEL_FUNC_1	PINSEL_FUNC_2	PINSEL_FUNC_3
P2.10	GPIO	EINT0	NMI	
P2.11	GPIO	EINT1	I2STX_CLK	
P2.12	GPIO	EINT2	I2STX_WS	
P2.13	GPIO	EINT3	I2STX_SDA	

External HW interrupts

- There are two types of interrupt triggers namely:
 - Level Triggered: Level 0 or Level 1
 - Edge Triggered: Rising or Falling Edge

- Operational Logic:



User, SYS	FIQ	IRQ	SVC	Undef	Abort
r0					
r1					
r2					
r3					
r4					
r5					
r6					
r7					
r8	r8				
r9	r9				
r10	r10				
r11	r11				
r12	r12				
r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)
r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)
r15 (pc)					
CPSR	spsr	spsr	spsr	spsr	spsr



External HW interrupts

- Since we are accessing system control block for EHI we write **LPC_SC** similar to accessing GPIO as we write **LPC_GPIO** to access SFR's associated with External Hardware Interrupts (EHI).
- Registers associated with EHI are

EINT Registers

Register	Description
PINSELx	To configure the pins as External Interrupts
EXTINT	External Interrupt Flag Register contains interrupt flags for EINT0,EINT1, EINT2 & EINT3.
EXTMODE	External Interrupt Mode register(Level/Edge Triggered)
EXTPOLAR	External Interrupt Polarity(Falling/Rising Edge, Active Low/High)



External HW interrupts Registers

EXTINT				
31:4	3	2	1	0
RESERVED	EINT3	EINT2	EINT1	EINT0

EINTx: Bits will be set whenever the interrupt is detected on the particular interrupt pin. If the interrupts are enabled then the control goes to ISR.

Writing one to specific bit will clear the corresponding interrupt.

EXTMODE				
31:4	3	2	1	0
RESERVED	EXTMODE3	EXTMODE2	EXTMODE1	EXTMODE0

EXTMODEx: These bits are used to select whether the EINTx pin is level or edge Triggered

0: EINTx is Level Triggered.

1: EINTx is Edge Triggered.

External HW interrupts Registers

EXTPOLAR				
31:4	3	2	1	0
RESERVED	EXTPOLAR3	EXTPOLAR2	EXTPOLAR1	EXTPOLAR0

EXTPOLARx: These bits are used to select polarity(LOW/HIGH, FALLING/RISING) of the EINTx interrupt depending on the EXTMODE register.

0: EINTx is Active Low or Falling Edge (depending on EXTMODEx).

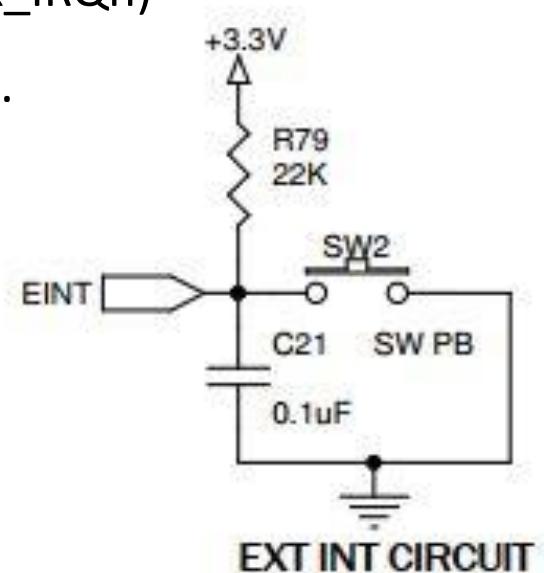
1: EINTx is Active High or Rising Edge (depending on EXTMODEx).

EXTMODEx	EXTPOLARx	EINTx
0	0	Level 0
0	1	Level 1
1	0	Falling Edge
1	1	Rising Edge

External HW interrupts

Steps to Configure External Hardware Interrupts:

- Configure the pins as external interrupts in PINSELx register.
- Configure the EINTx as Edge/Level triggered in EXTMODE register.
- Select the polarity(Falling/Rising Edge, Active Low/High) of the interrupt in EXTPOLAR register.
- Enable the interrupts by calling NVIC_EnableIRQ(EINTx_IRQn)
- Clear any pending interrupts in EXTINT on entering ISR.





External HW interrupts - Programming

Toggle LED connected to P1.23 at each negative edge of the input applied at P2.12 (EINT2, Function-01)

```
include<LPC17xx.h>
void EINT2_IRQHandler(void);
int main(void)
{
    SystemInit();
    SystemCoreClockUpdate();

    LPC_PINCON->PINSEL4 |= (1<<24);
    LPC_GPIO1->FIODIR = 0x00800000;
    LPC_SC->EXTMODE = 0x00000004;
    LPC_SC->EXTPOLAR = 0x00000000;
    NVIC_EnableIRQ(EINT2_IRQn);
    while(1) ;
}

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

*

External HW interrupts - Programming

```
#include<LPC17xx.h>
void EINT3_IRQHandler(void);
unsigned char int3_flag=0;
int main(void)
{
    LPC_PINCON->PINSEL4 |= 0x04000000;          //P2.13 as EINT3
    LPC_PINCON->PINSEL4 &= 0xFCFFFFFF;        //P2.12 GPIO for LED
    LPC_GPIO2->FIODIR = 0x00001000;           //P2.12 is assigned output
    LPC_GPIO2->FIOSET = 0x00001000;           //Initially LED is kept on
    LPC_SC->EXTINT = 0x00000008;              //writing 1 clears the interrupt, get set if there is interrupt
    LPC_SC->EXTMODE = 0x00000008;              //EINT3 is initiated as edge sensitive, 0 for level sensitive
    LPC_SC->EXTPOLAR = 0x00000000;            //EINT3 is falling edge sensitive, 1 for rising edge
    NVIC_EnableIRQ(EINT3_IRQn);                //core_cm3.h
    while(1);
}
```

PINSEL4 - bit 26 and 27 - P2.13- EINT3 is the first alternate function - value is 01
PINSEL4- Value 0xFFFFFFFF - C- 1100 making P2.12 as GPIO - bit 25 and 24 with 00, while not altering 26 and 27





External HW interrupts - Programming

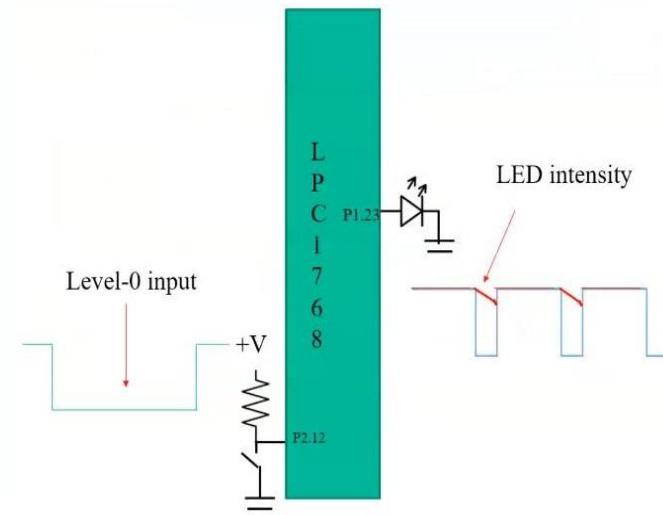
```
void EINT3_IRQHandler(void)
{
    LPC_SC->EXTINT = 0x00000008;           //clears the interrupt
    if(int3_flag == 0x00)                  //when flag is '0' off the LED
    {
        LPC_GPIO2->FIOCLR = 0x00001000;
        int3_flag = 0xff;
    }
    else                                  //when flag is FF on the LED
    {
        LPC_GPIO2->FIOSET = 0x00001000;
        int3_flag = 0;
    }
}
```

External HW interrupts - Programming

**Toggle LED connected to P1.23 whenever the switch connected to P2.12 (EINT2, Function-01) is pressed
 LED remains ON as long as the switch is pressed (Assume, when the switch is pressed Logic-0 is INPUT).**

```
#include<LPC17xx.h>
void EINT2_IRQHandler(void);
int main(void)
{
    LPC_PINCON->PINSEL4 |= (1<<24);           //P2.12 as EINT2 i.e FUNCTION-01
    LPC_GPIO1->FIODIR = 0x00800000;             //P1.23 is assigned output
    LPC_SC->EXTMODE = 0x00000000;                //EINT2 is level-0 sensitive
    LPC_SC->EXTPOLAR = 0x00000000;
    NVIC_EnableIRQ(EINT2_IRQn);
    while(1);

}
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
}
```



External HW interrupts - Programming

Program to Stop Displaying Hex digits on seven segment Display on Button Pressed



Reference: https://www.youtube.com/watch?v=FuAyGjmTNdc&ab_channel=AnandHD

Manipal School of Information Sciences, MAHE, Manipal





ARM LPC1768

IO Interrupt



GPIO Interrupts

- The **GPIO interrupt** is only available to **PORT 0 & PORT 2** of LPC1768 Microcontroller.
- The GPIO interrupts are **edge triggered**.

Falling Edge Interrupt

- **Register `LPC_GPIOINT` → `IOxIntEnF`**

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

- Example:
 - To enable Pin 0.3 falling edge interrupt
 - `LPC_GPIOINT` → `IO0IntEnF |= (1<<3);`

Rising Edge Interrupt

- **Register `LPC_GPIOINT` → `IOxIntEnR`**

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

- Example:

To enable Pin 0.3 and 0.5 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 Interrupt Register Map (SFR's)

- IntStatus → GPIO overall Interrupt status.
- IntEnR → GPIO Interrupt Enable for Rising Edge.
- IntEnF → GPIO interrupt enable for falling edge.
- IntStatR → GPIO interrupt status for rising edge.
- IntStatF → GPIO interrupt status for falling edge.
- IntClr → GPIO Interrupt Clear.

- **Note:** GPIO Interrupts are applicable only on Input port and on output port.

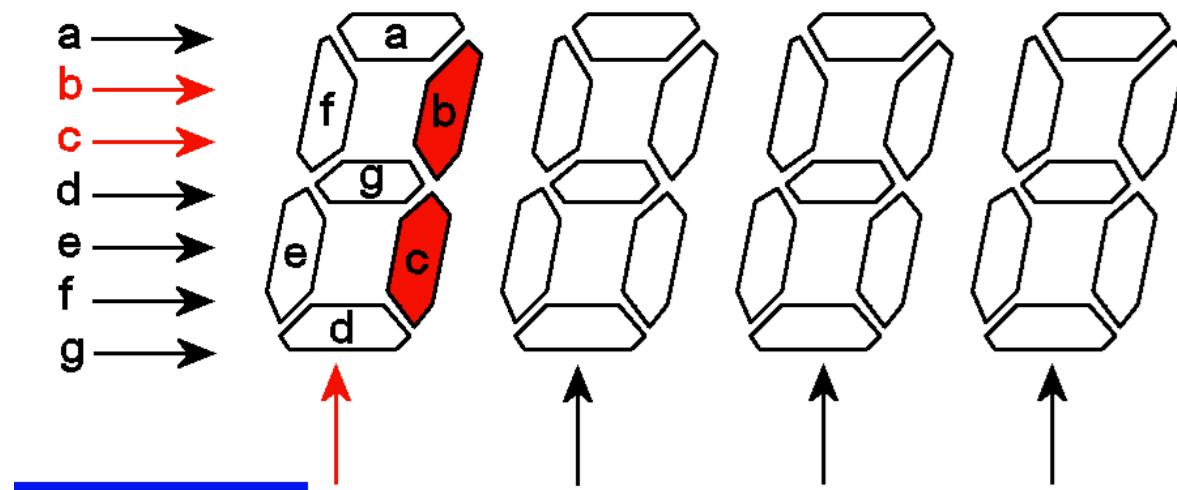


GPIO Interrupt Programming



GPIO Interrupt Programming

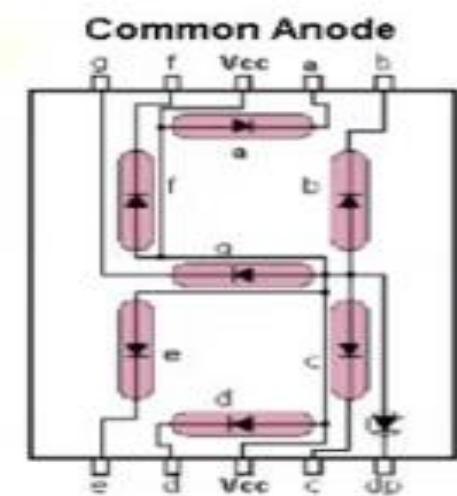
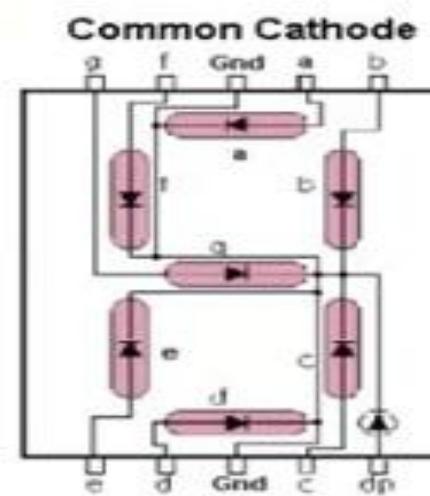
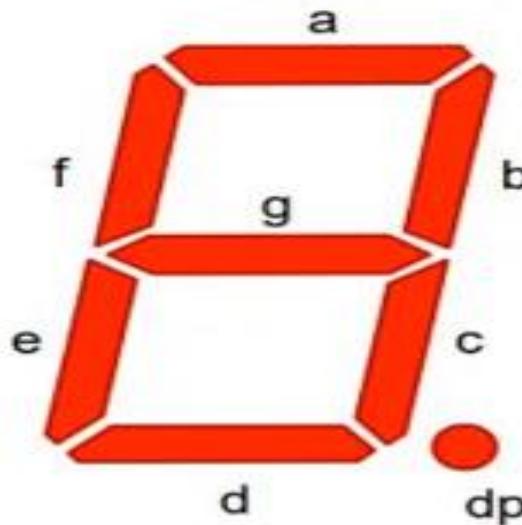
ARM LPC1768 – Seven Segment Display Interfacing



Seven Segment Display



- Common Cathode (all LED cathodes are connected)
- Common Anode (all LED anodes are connected)

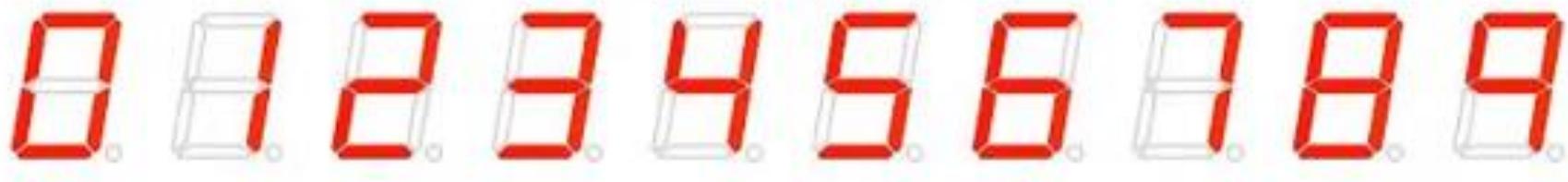




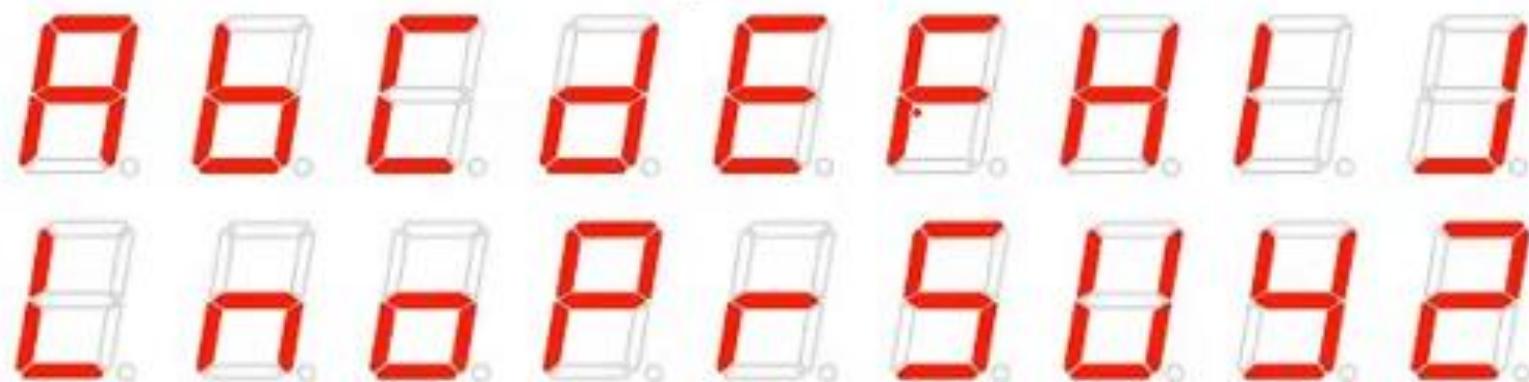


Seven Segment Display

Decimal Digits 0-9



Select Alpha Characters



Seven Segment Display



Here,
CNA
P0.11 – P0.4

FRC Cable
-----> '7' Segment
h ----- a



Seven Segment Display

Number	h g f e d c b a	Hex Code
0	0 0111111	3F
1	0 0000110	06
2	1011011	5B
3	1001111	4F
4	1100110	66
5	1101101	6D
6	1111101	7D
7	0000111	07
8	1111111	7F
9	1001111	4F

Pin CNA	Pin LPC1768	Description
1	81	P0.4/I2SRX_CLK/RD2/CAP2.0
2	80	P0.5/I2SRX_WS/TD2/CAP2.1
3	79	P0.6/I2SRX_SDA/SSEL1/MAT2.0
4	78	P0.7/I2STX_CLK/SCK1//MAT2.1
5	77	P0.8/I2STX_WS/MISO1/MAT2.2
6	76	P0.9/I2STX_SDA/MOSI1/MAT2.3
7	48	P0.10/TXD2/SDA2/MAT3.0
8	49 *	P0.11/RXD2/SCL2/MAT3.1
9	-	No connection
10	-	Ground

Pin CNB	Pin LPC1768	Description
1	37	P1.23/MCI1/PWM1.4/MISO0
2	38	P1.24/MCI2/PWM1.5/MOSIO
3	39	P1.25/MCOA1/MAT1.1
4	40	P1.26/MCOB1/PWM1.6/CAP0.0
5	53	P2.10/EINT0/NMI
6	52	P2.11/EINT1/I2STX_CLK
7	51	P2.12/EINT2/I2STX_WS
8	50	P2.13/EINT3/I2STX_SDA
9	-	No connection
10	-	Ground

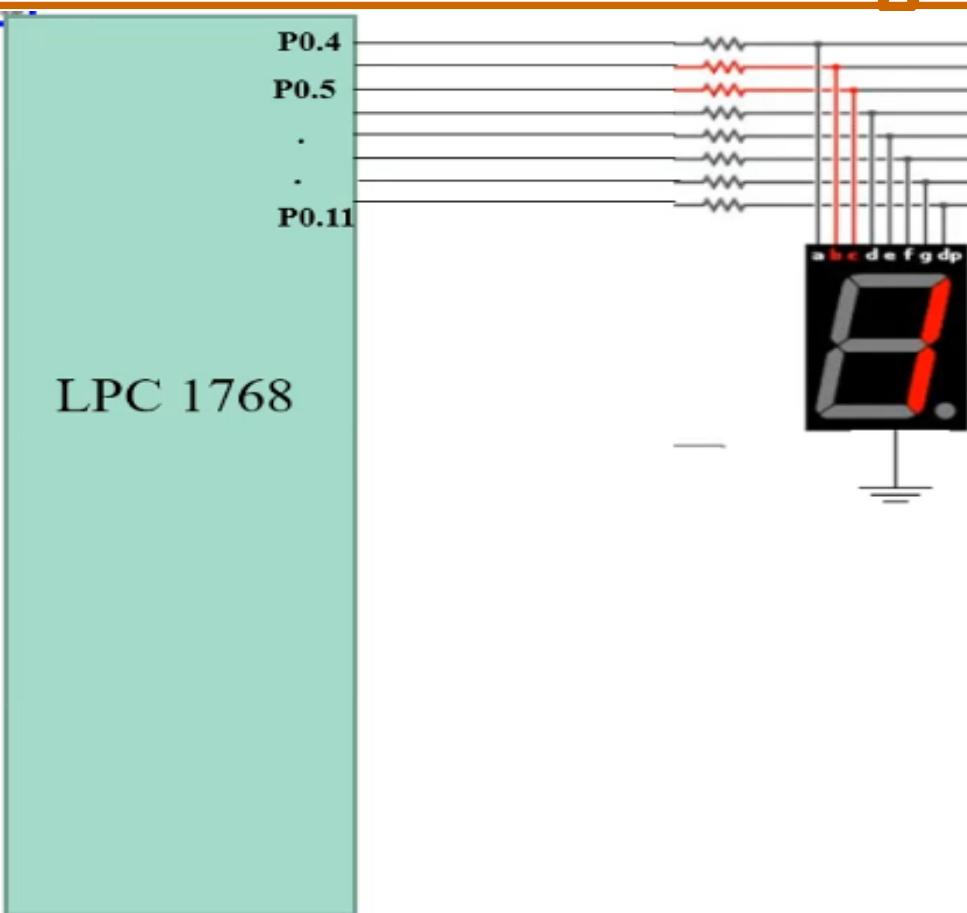
Pin CNC	Pin LPC1768	Description
1	62	P0.15/TXD1/SCK0/SCK
2	63	P0.16/RXD1/SSEL0/SSEL
3	61	P0.17/CTS1/MISO0/MISO
4	60	P0.18/DDC1/MOSI0/MOSI
5	59	P0.19/DSR1/SDA1
6	58	P0.20/DTR1/SCL1
7	57	P0.21/RI1/RD1
8	56	P0.22/RTS1/TD1
9	50	P2.13/I2STX_SDA
10	-	Ground

Pin CND	Pin LPC1768	Description
1	9	P0.23/AD0.0/I2SRX_CLK/CAP3.0
2	8	P0.24/AD0.1/I2SRX_WS/CAP3.1
3	7	P0.25/AD0.2/I2SRX_SDA/TXD3
4	6	P0.26/AD0.3/AOUT/RXD3
5	25	P0.27/SDA0/USB/SDA
6	24	P0.28/SCL0/USB_SCL
7	75	P2.0/PWM1.1/TXD1
8	74	P2.1/PWM1.2/RXD1
9	-	No connection
10	-	Ground

Assumption:

- Common Cathode.
- 'h' is MS bit (P0.11) & 'a' is LS bit (P0.4)

Seven Segment Display Interfacing



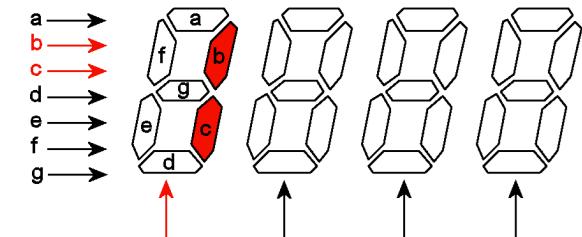
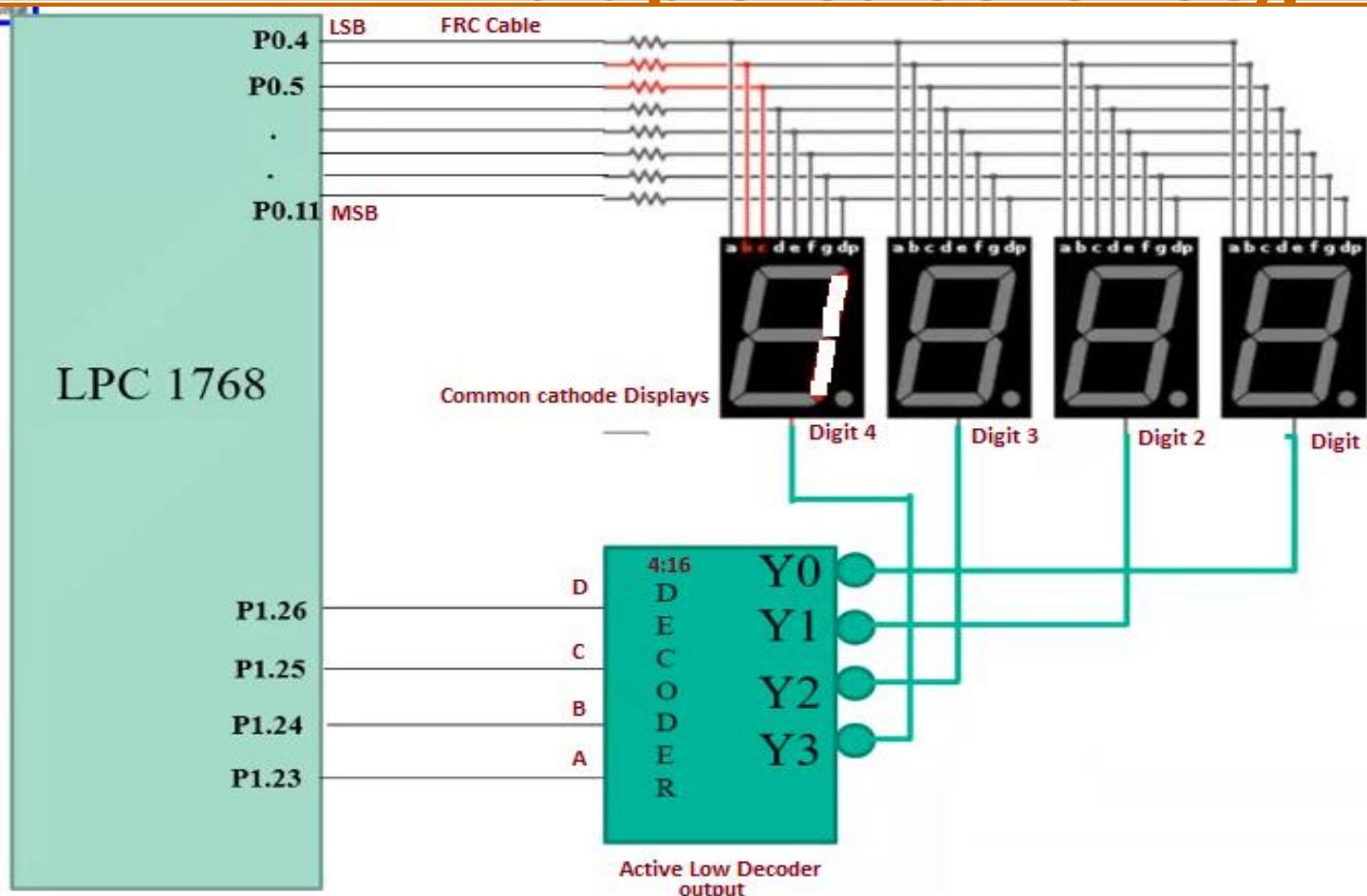


Seven Segment Display Programming

```
#include<lpc17xx.h>
unsigned char seven_seg[10]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};
unsigned int i,j;
void delay(void);
int main(void)
{
    SystemInit();
    SystemCoreClockUpdate();

    LPC_PINCON->PINSEL0 = 0      //P0.4 to P0.11 GPIO data lines
    LPC_GPIO0->FIODIR |= 0x00000FF0;      //P0.4 to P0.11 output
    while (1)
    {
        for(i=0; i<10; i++)
        {
            LPC_GPIO0->FIOPIN = seven_seg[i ] << 4;
            delay();
        }
    }
}
void delay(void)
{
    for(j=0;j<10000;j++);
}
```

Multiplexed Seven Segment Display





Multiplexed Seven Segment Display Programming – Display a Number

```
#include <LPC17xx.h>
#include <stdio.h>

#define FIRST_SEG      0<<23
#define SECOND_SEG     1<<23
#define THIRD_SEG      2<<23
#define FOURTH_SEG     3<<23

unsigned int dig_count;
unsigned int digit_value = {0, 4, 3, 2, 1};
unsigned int select_segment = {0, 0 << 23, 1<<23, 2<<23, 3<<23};
unsigned char seven_seg[10]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};
unsigned long int temp1,temp2 ,i=0;

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

    LPC_PINCON->PINSEL0 = 0;    //P0.4 to P0.11 GPIO data lines
    LPC_PINCON->PINSEL3 = 0;    //P1.23 to P1.26 GPIO enable lines

    LPC_GPIO0->FIODIR = 0x00000FF0;    //P0.4 to P0.11 output
    LPC_GPIO1->FIODIR = 0x07800000;    //P1.23 to P1.26 output
```



Multiplexed Seven Segment Display Programming – Display a Number

```
while(1)
{
    delay();

    dig_count +=1;
    if(dig_count == 0x05)
        dig_count = 0x01;

    Display();

} //end of while(1)

}//end of main

void Display(void)    //To Display on 7-segments
{
    |                                ▶
    LPC_GPIO1->FIOPIN = select_segment[dig_count];
    LPC_GPIO0->FIOPIN = seven_seg[digit_value[dig_count]] << 4;
    for(i=0;i<500;i++);
    LPC_GPIO0->FIOCLR = 0x00000FF0;
}
void delay(void)
{
for i=0;i<500;i++);
}
```



Multiplexed Seven Segment Display Programming – 4 Digit BCD UP Counter

```
while(1)
{
    delay();

    dig_count +=1;
    if(dig_count == 0x05)
        dig_count = 0x01;

    Display(); ←

} //end of while(1)
```

For every second update the digits

```
//end of main
```

```
void Display(void) //To Display on 7-segments
{
    LPC_GPIO1->FIOPIN = select_segment[dig_count];
    LPC_GPIO0->FIOPIN = seven_seg[digit_value[dig_count]] << 4;
    for(i=0;i<500;i++);
    LPC_GPIO0->FIOCLR = 0x00000FF0;
```

```
}
```

```
void delay(void)
{
    for i=0;i<500;i++);
```

After one second, set Flag



Multiplexed Seven Segment Display Programming – 4 Digit BCD UP Counter

```
void delay(void)
{
for i=0;i<500;i++);

if(count ==N)
{
    flag = 0xFF;
    count = 0;
}
else count += 1;
}
```

After one second, set Flag



Multiplexed Seven Segment Display Programming – 4 Digit BCD UP Counter

```
if(flag == 0xFF)
{
    flag = 0;
    digit_value[1] +=1;

    if(digit_value[1] == 0x0A)
    {
        digit_value[1] = 0;
        digit_value[2] +=1;

        if(digit_value[2] == 0x0A)
        {
            digit_value[2]= 0;
            digit_value[3] +=1;

            if(digit_value[3] == 0x0A)
            {
                digit_value[3] = 0;
                digit_value[4] += 1;

                if(digit_value[4]== 0x0A)
                {
                    digit_value[4]= 0;
                } //end of dig4
            } //end of dig3
        } //end of dig2
    } //end of dig1
}

} //end of one_sec if
```

For every second update the digits



PINSELECT Registers

PIN CONNECT BLOCK

Register	Controls
PINSEL0	P0[15:0]
PINSEL1	P0 [31:16]
PINSEL2	P1 [15:0] (Ethernet)
PINSEL3	P1 [31:16]
PINSEL4	P2 [15:0]
PINSEL5	P2 [31:16]
PINSEL6	P3 [15:0]
PINSEL7	P3 [31:16]
PINSEL8	P4 [15:0]
PINSEL9	P4 [31:16]

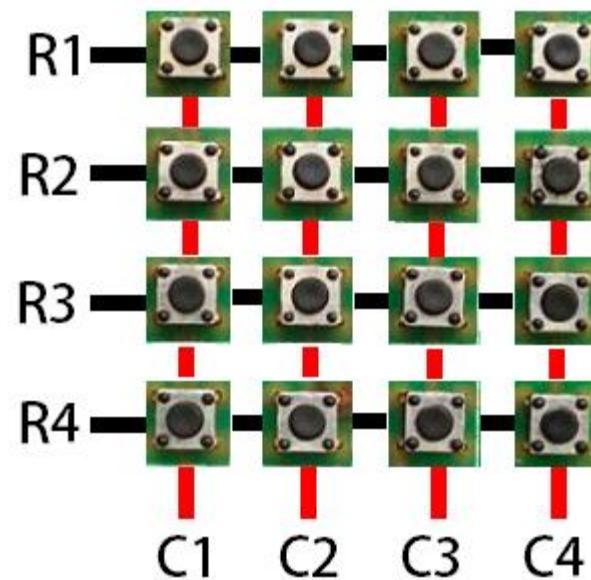
PINSEL0 to PINSEL9 Values	Function
00	Primary (default) function, typically GPIO port
01	First alternate function
10	Second alternate function
11	Third alternate function



GPIO Registers

FIODIR	Fast GPIO Port Direction control register. This register individually controls the direction of each port pin.	0	Controlled pin is input.
FIOPIN	Fast Port Pin value register using FIOMASK. The current state of digital port pins can be read from this register, regardless of pin direction or alternate function selection (as long as pins are not configured as an input to ADC). The value read is masked by ANDing with inverted FIOMASK. Writing to this register places corresponding values in all bits enabled by zeros in FIOMASK. Important: if an FIOPIN register is read, its bit(s) masked with 1 in the FIOMASK register will be read as 0 regardless of the physical pin state.	1	Controlled pin is output.
FIOSET	Fast Port Output Set register using FIOMASK. This register controls the state of output pins. Writing 1s produces highs at the corresponding port pins. Writing 0s has no effect. Reading this register returns the current contents of the port output register. Only bits enabled by 0 in FIOMASK can be altered.	0	Controlled pin output is unchanged.
FIOCLR	Fast Port Output Clear register using FIOMASK. This register controls the state of output pins. Writing 1s produces lows at the corresponding port pins. Writing 0s has no effect. Only bits enabled by 0 in FIOMASK can be altered.	1	Controlled pin output is set to HIGH.
FIOMASK	Fast Mask register for port. Writes, sets, clears, and reads to port (done via writes to FIOPIN, FIOSET, and FIOCLR, and reads of FIOPIN) alter or return only the bits enabled by zeros in this register.	0	Controlled pin is affected by writes to the port's FIOxSET, FIOxCLR, and FIOxPIN register(s). Current state of the pin can be read from the FIOxPIN register.
		1	Controlled pin is not affected by writes into the port's FIOxSET, FIOxCLR and FIOxPIN register(s). When the FIOxPIN register is read, this bit will not be updated with the state of the physical pin.

ARM LPC1768 – Hex keypad Interfacing



Hex keypad

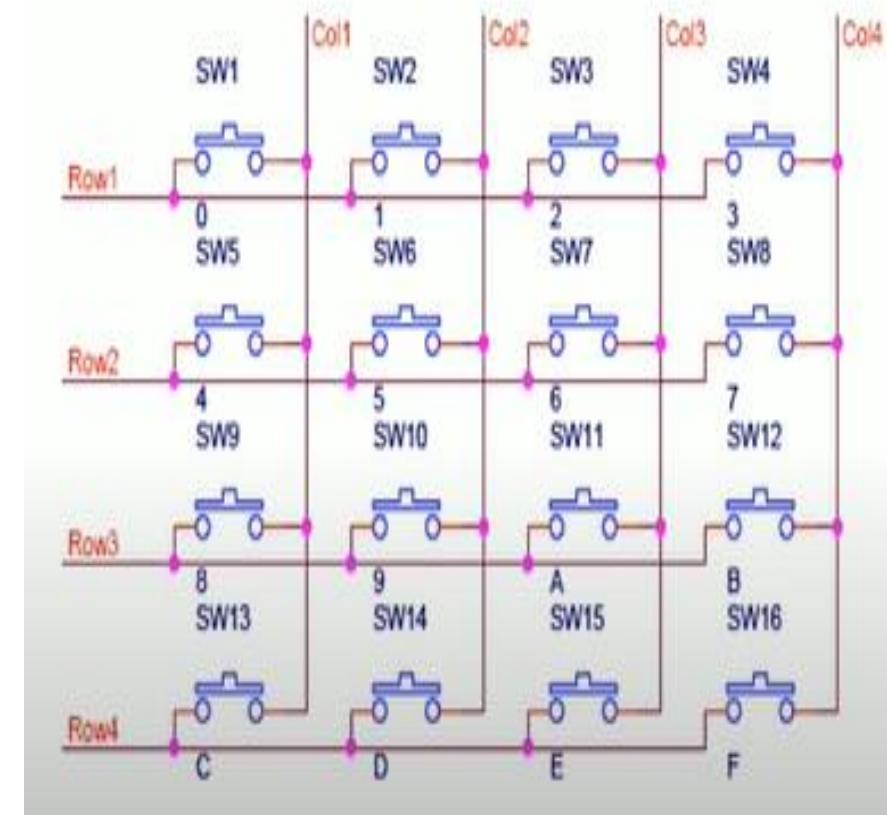
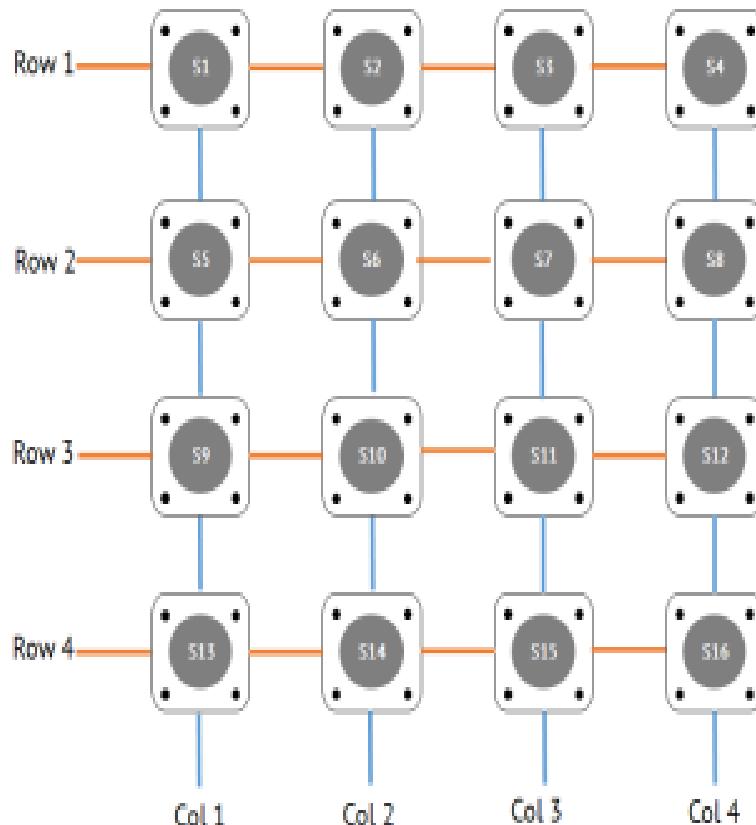
- Used to
 - Give multiple Inputs
 - Each input can have particular significance
- Easy to Interface
- Easy to procure





Hex keypad Introduction

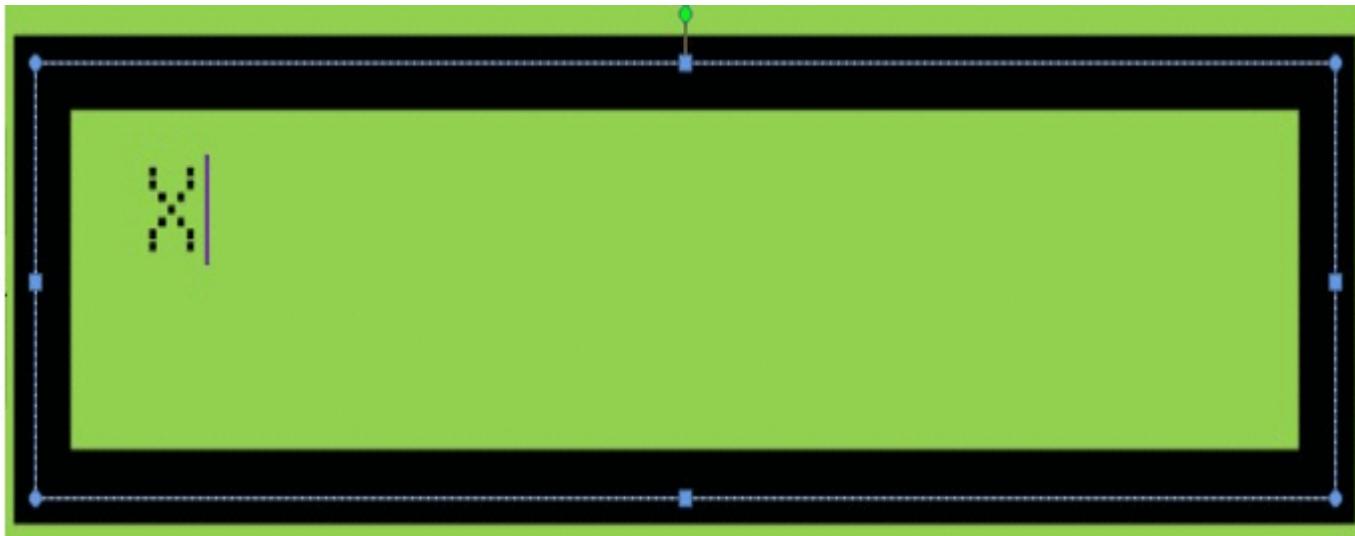
Hex keypad





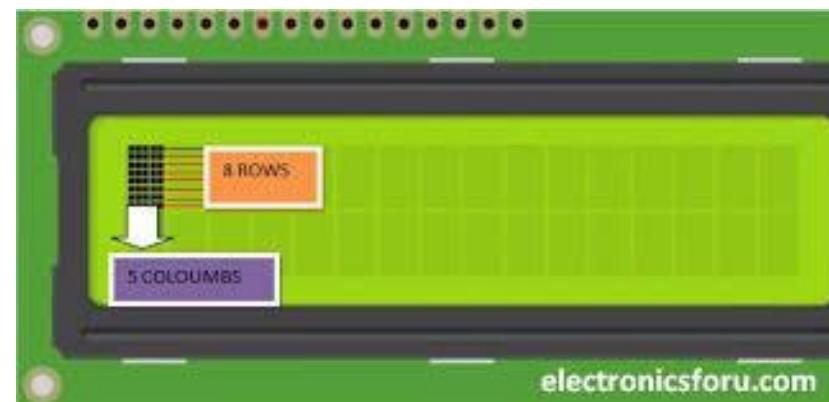
Hex keypad Programming

ARM LPC1768 – LCD Display Interfacing



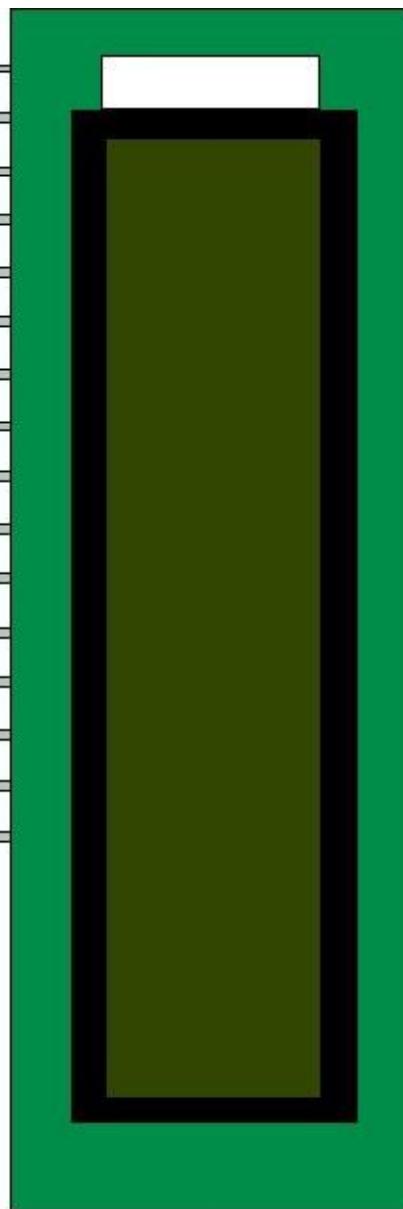
LCD 16 x 2 Features

- The operating voltage of this LCD is 4.7V-5.3V.
- It includes two rows where each row can produce 16-characters.
- The utilization of current is 1mA with no backlight.
- Every character can be built with a 5×8 pixel box.
- The alphanumeric LCDs alphabets & numbers.
- Its display can work on two modes like 4-bit & 8-bit.
- These are obtainable in Blue & Green Backlight.
- It displays a few custom generated characters.



LCD Pinouts

GND
 Vcc
 VEE
 RS
 R/W
 EN
 DB0
 DB1
 DB2
 DB3
 DB4
 DB5
 DB6
 DB7
 Led +
 Led -



Pin No.	Symbol	Function
1	Vss	Ground
2	Vdd	+5V
3	Vo	LCD contrast adjust
4	RS	Register select
5	R/W	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
+	BL+	Power Supply for BL+
-	BL-	Power Supply for BL-



LCD Pinouts

- RS – Register Select

RS=0 - Command Register

RS=1 - Data Register

- RW- Read/Write

RW=0 - Write

- EN- Enable the LCD

EN=1 for Enable



LCD Commands

- For Hex Code-01, the LCD command will be Clearing the Display Screen
- For Hex Code-02, the LCD command will be returning home
- For Hex Code-04, the LCD command will be decrement cursor
- For Hex Code-06, the LCD command will be Increment cursor
- For Hex Code-05, the LCD command will be Shift display right
- For Hex Code-07, the LCD command will be Shift display left
- For Hex Code-08, the LCD command will be Display off, cursor off
- For Hex Code-0A, the LCD command will be cursor on and display off
- For Hex Code-0C, the LCD command will be cursor off, display on
- For Hex Code-0E, the LCD command will be cursor blinking, Display on
- For Hex Code-0F, the LCD command will be cursor blinking, Display on
- For Hex Code-10, the LCD command will be Shift cursor position to left
- For Hex Code-14, the LCD command will be Shift cursor position to the right
- For Hex Code-18, the LCD command will be Shift the entire display to the left
- For Hex Code-1C, the LCD command will be Shift the entire display to the right
- For Hex Code-80, the LCD command will be Force cursor to the beginning (1st line)
- For Hex Code-C0, the LCD command will be Force cursor to the beginning (2nd line)
- For Hex Code-38, the LCD command will be 2 lines and 5×7 matrix



LCD Commands

Instruction	Code										Description	Execution Time (max) (when f_{cp} or f_{osc} is 270 kHz)
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.	
Return home	0	0	0	0	0	0	0	0	1	—	Sets DDRAM address 0 in address counter. Also returns display from being shifted to original position. DDRAM contents remain unchanged.	1.52 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 μ s
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets entire display (D) on/off, cursor on/off (C), and blinking of cursor position character (B).	37 μ s
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	—	—	Moves cursor and shifts display without changing DDRAM contents.	37 μ s
Function set	0	0	0	0	1	DL	N	F	—	—	Sets interface data length (DL), number of display lines (N), and character font (F).	37 μ s
Set CGRAM address	0	0	0	1	ACG	ACG	ACG	ACG	ACG	ACG	Sets CGRAM address. CGRAM data is sent and received after this setting.	37 μ s
Set DDRAM address	0	0	1	ADD	Sets DDRAM address. DDRAM data is sent and received after this setting.	37 μ s						
Read busy flag & address	0	1	BF	AC	Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents.	0 μ s						



LCD Commands

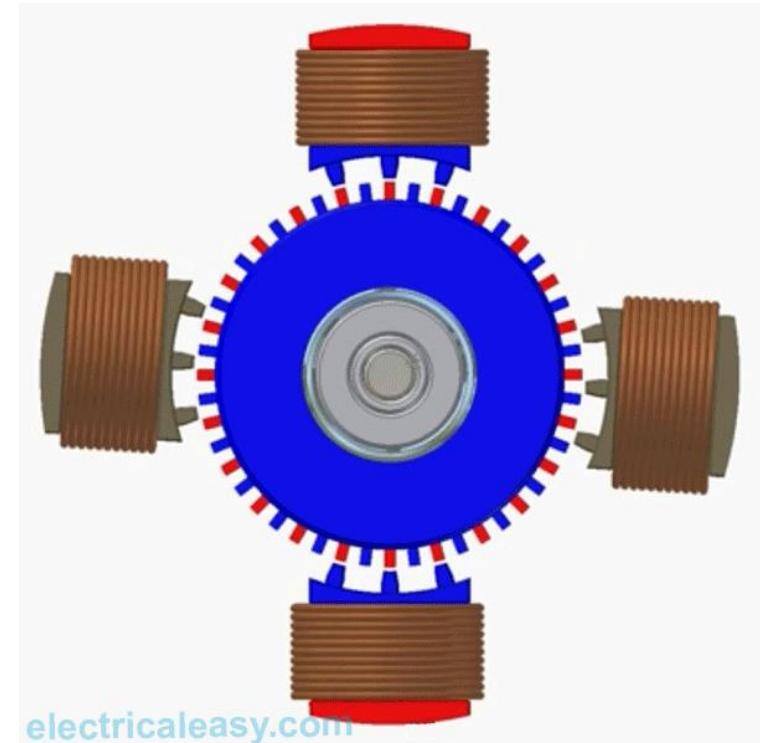
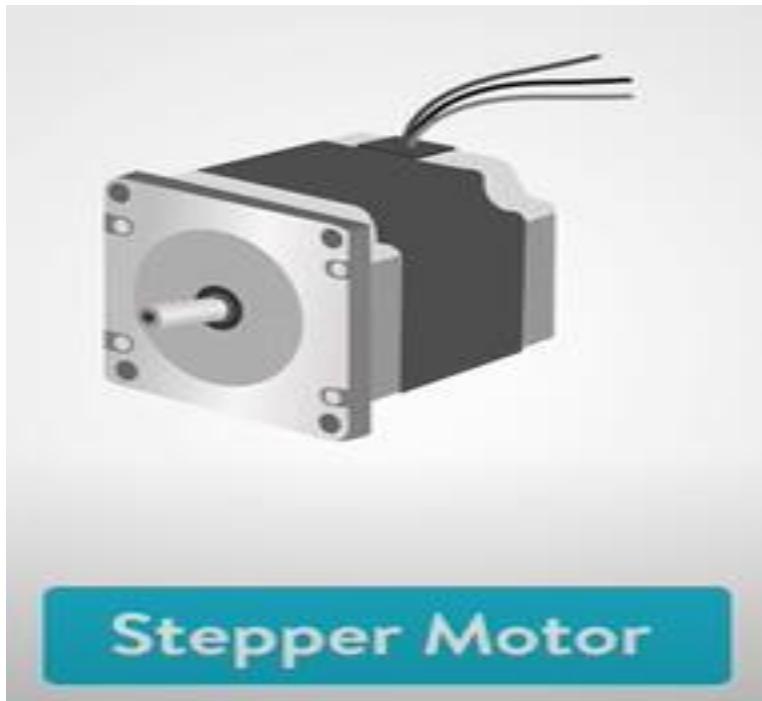
Instruction	RS	R/W	Code								Execution Time (max) (when f_{op} or f_{osc} is 270 kHz)	
			DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	
Write data to CG or DDRAM	1	0	Write data								Writes data into DDRAM or CGRAM.	37 μ s $t_{ADD} = 4 \mu$ s*
Read data from CG or DDRAM	1	1	Read data								Reads data from DDRAM or CGRAM.	37 μ s $t_{ADD} = 4 \mu$ s*
	I/D	= 1:	Increment								DDRAM: Display data RAM	Execution time
	I/D	= 0:	Decrement								CGRAM: Character generator	changes when
	S	= 1:	Accompanies display shift								RAM	frequency changes
	S/C	= 1:	Display shift								ACG: CGRAM address	Example:
	S/C	= 0:	Cursor move								ADD: DDRAM address	When f_{op} or f_{osc} is
	R/L	= 1:	Shift to the right								(corresponds to cursor	250 kHz,
	R/L	= 0:	Shift to the left								address)	
	DL	= 1:	8 bits, DL = 0: 4 bits								AC: Address counter used for	37μ s $\times \frac{270}{250} = 40 \mu$ s
	N	= 1:	2 lines, N = 0: 1 line								both DD and CGRAM	
	F	= 1:	5 x 10 dots, F = 0: 5 x 8 dots								addresses	
	BF	= 1:	Internally operating									
	BF	= 0:	Instructions acceptable									

Note: — indicates no effect.

- * After execution of the CGRAM/DDRAM data write or read instruction, the RAM address counter is incremented or decremented by 1. The RAM address counter is updated after the busy flag turns off. In Figure 10, t_{ADD} is the time elapsed after the busy flag turns off until the address counter is updated.

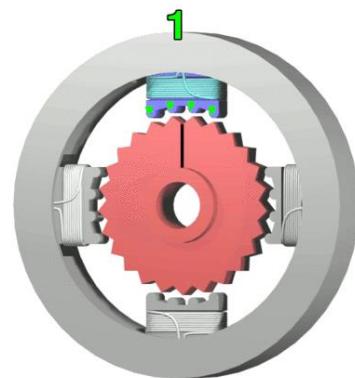


ARM LPC1768 – Stepper Motor Interfacing

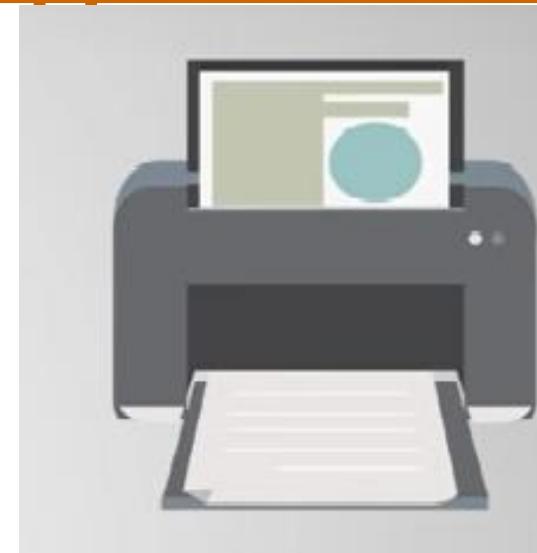


Stepper Motor Introduction

- A stepper motor (also referred to as step or stepping motor) is an electromechanical device achieving mechanical movements through conversion of electrical pulses.
- Typically, all winding in the motor are part of the stator, and the rotor is either a permanent magnet or, in the case of variable reluctance motors, a toothed block of some magnetically soft material.
- All of the commutation must be handled externally by the motor controller, and typically, the motors and controllers are designed so that the motor may be held in any fixed position as well as being rotated one way or the other.

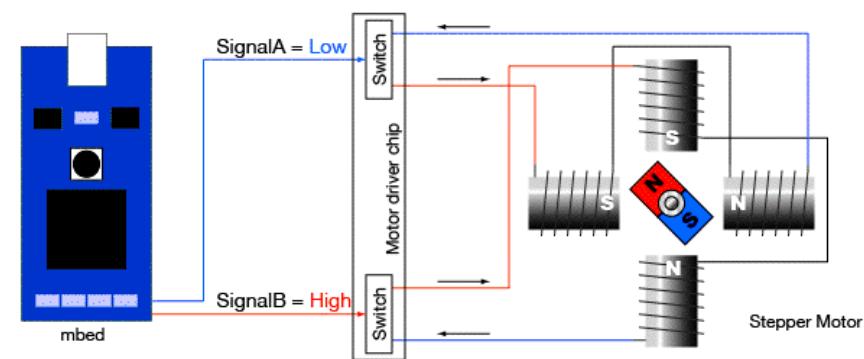
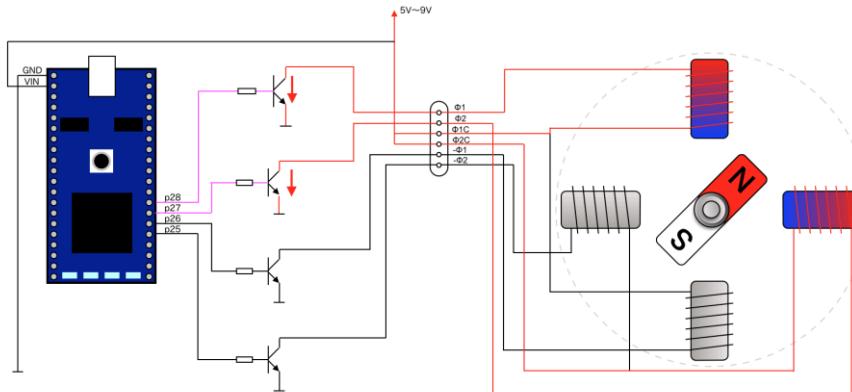


Applications



Stepper Motor - Working

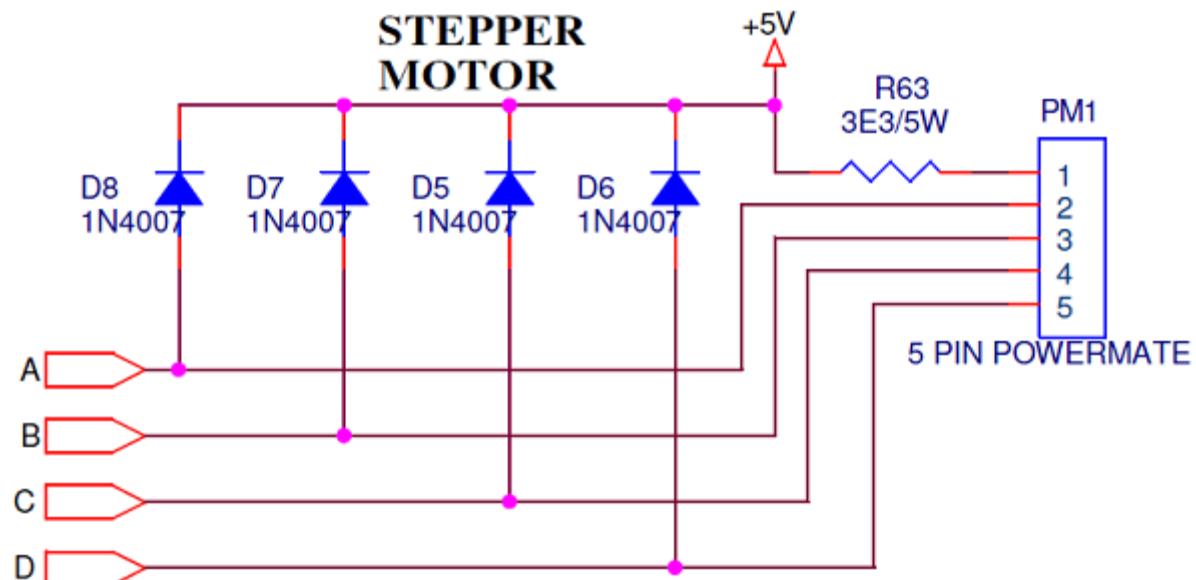
- Stepper motors are driven by digital pulses rather than by a continuous applied voltage.
- Unlike conventional electric motors which rotate continuously, stepper motors rotate or step in fixed angular increments.
- A stepper motor is most commonly used for position control.
- Most steppers can be stepped at audio frequencies, allowing them to spin quite quickly, and with an appropriate controller, they may be started and stopped at controlled orientations.



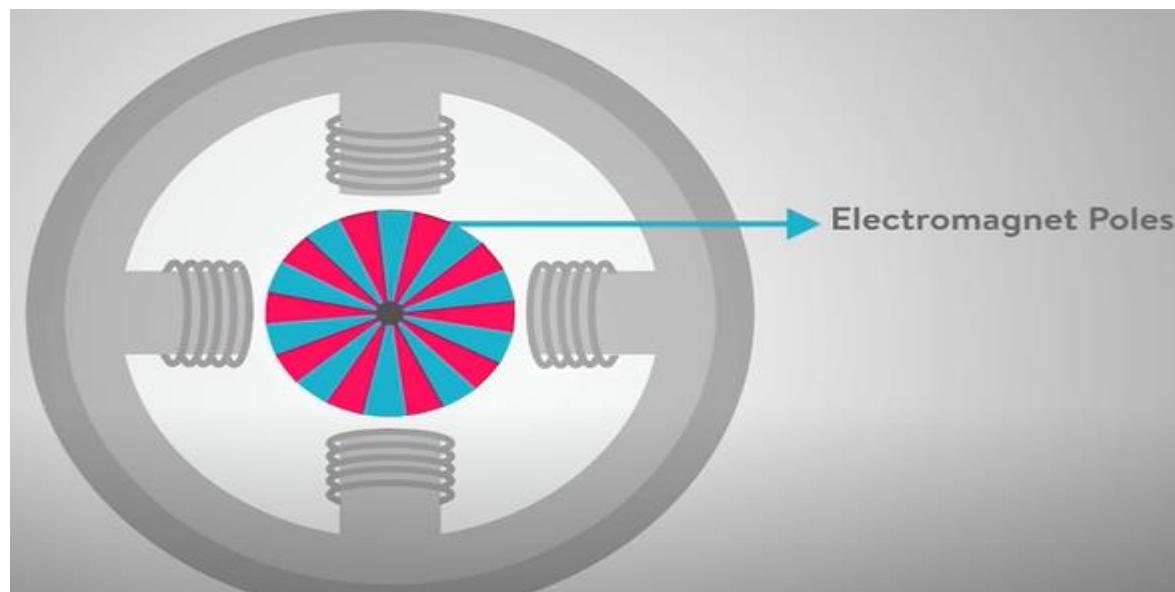
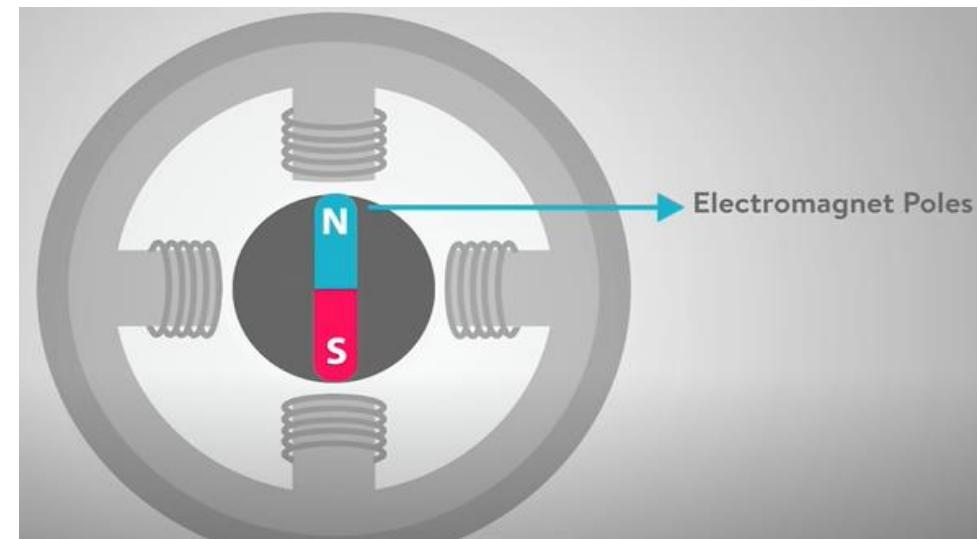
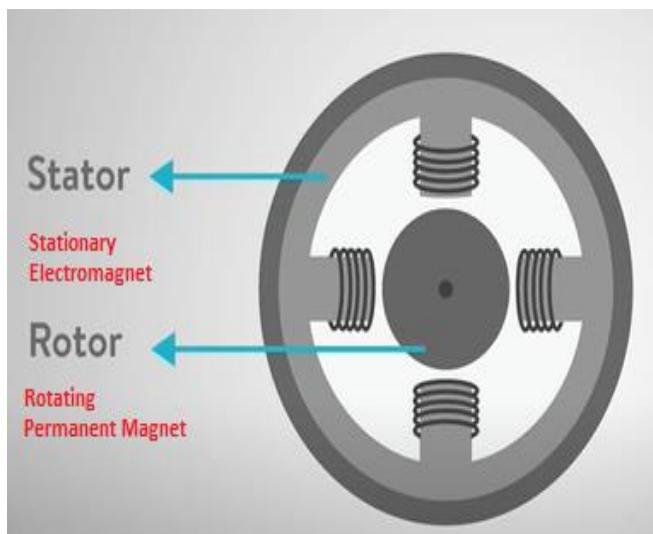
Stepper Motor - Working

- In this lab, we'll use a so called "five wire stepper" shown in the figure below. This 28BYJ48 stepper motor is driven via a driver board that contains 4 Darlington drivers (ULN2003) and 4 LEDs.
- PM1 – it's a 5 pin straight male power mate.

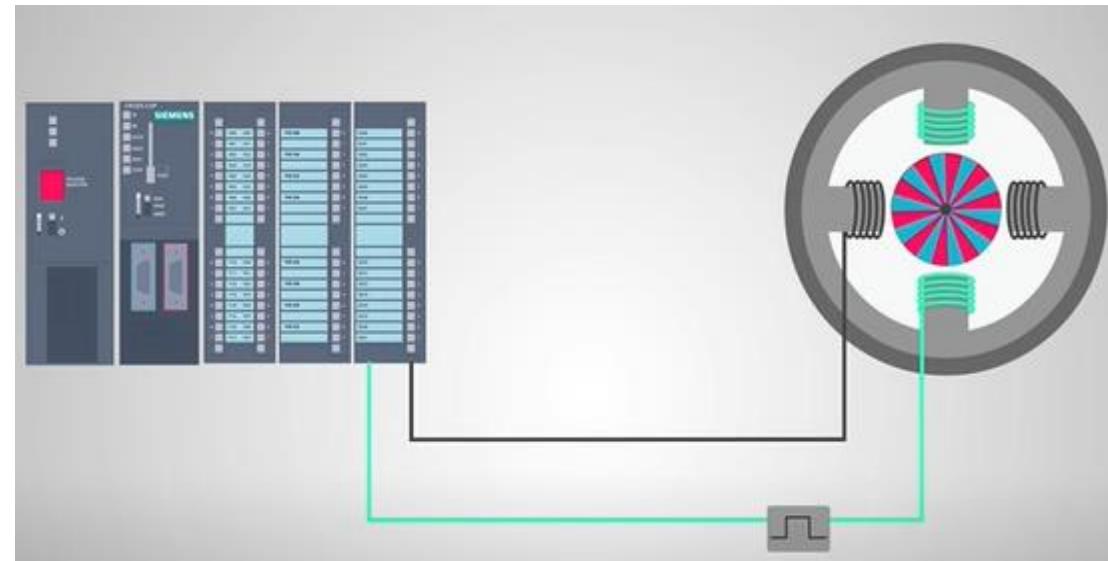
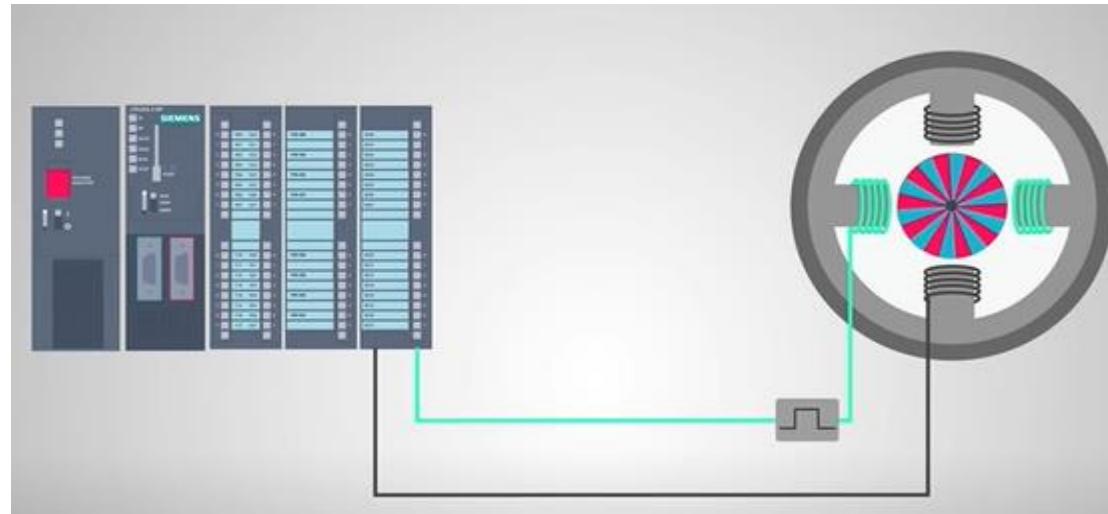
Pin no	Description
1	+5v supply
2	Phase A
3	Phase B
4	Phase C
5	Phase D



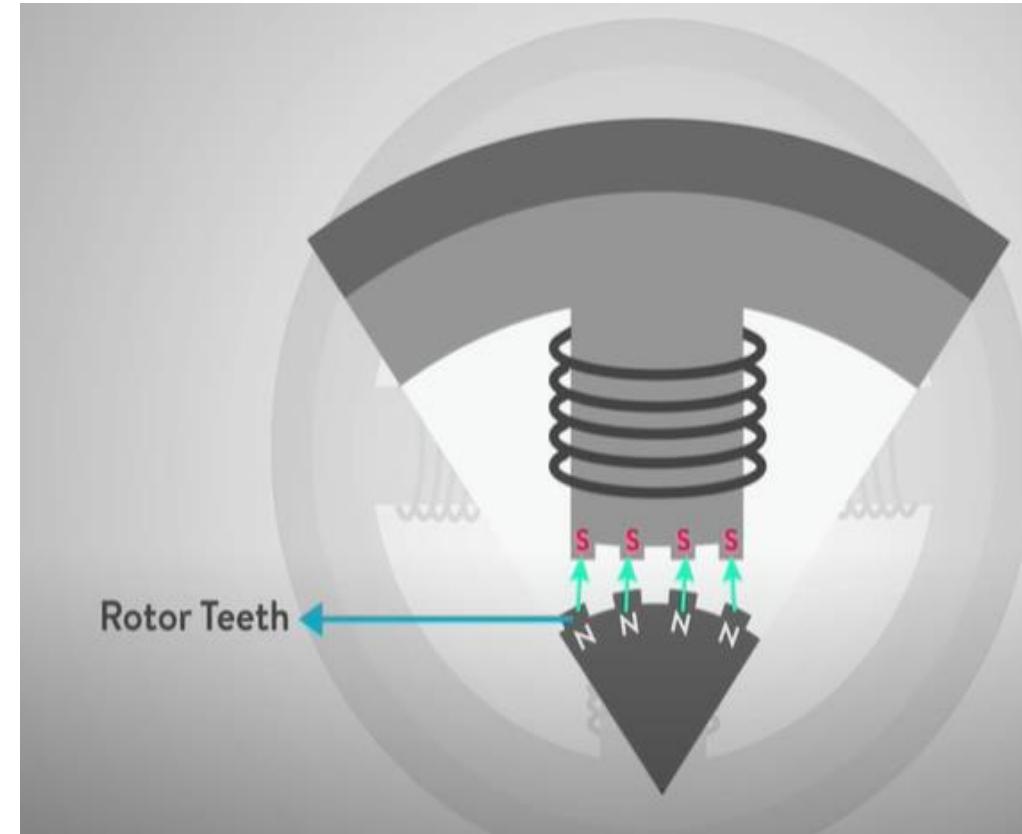
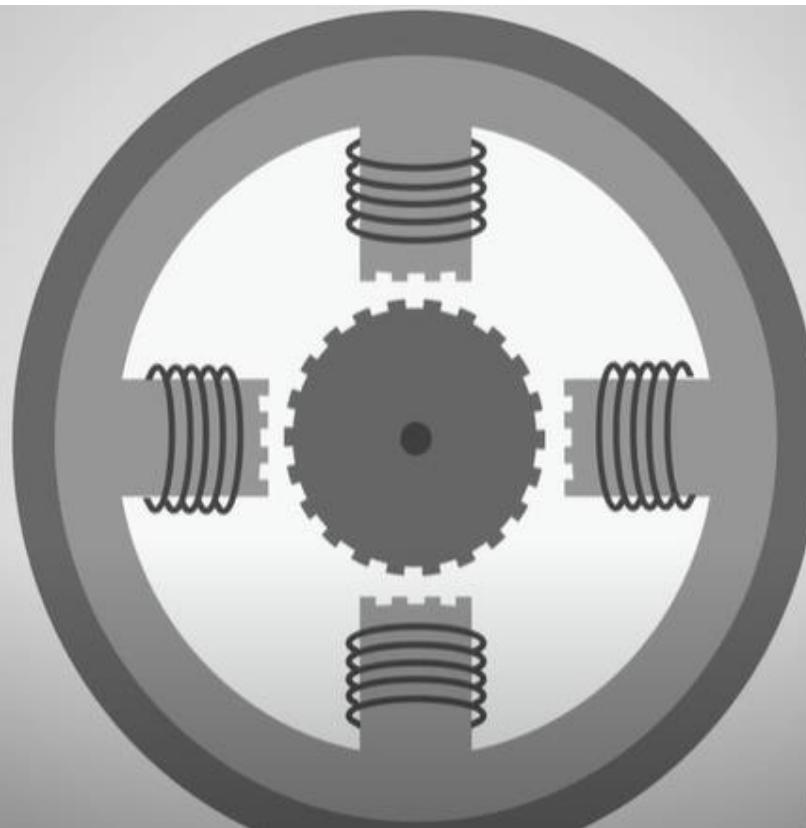
Construction



Working

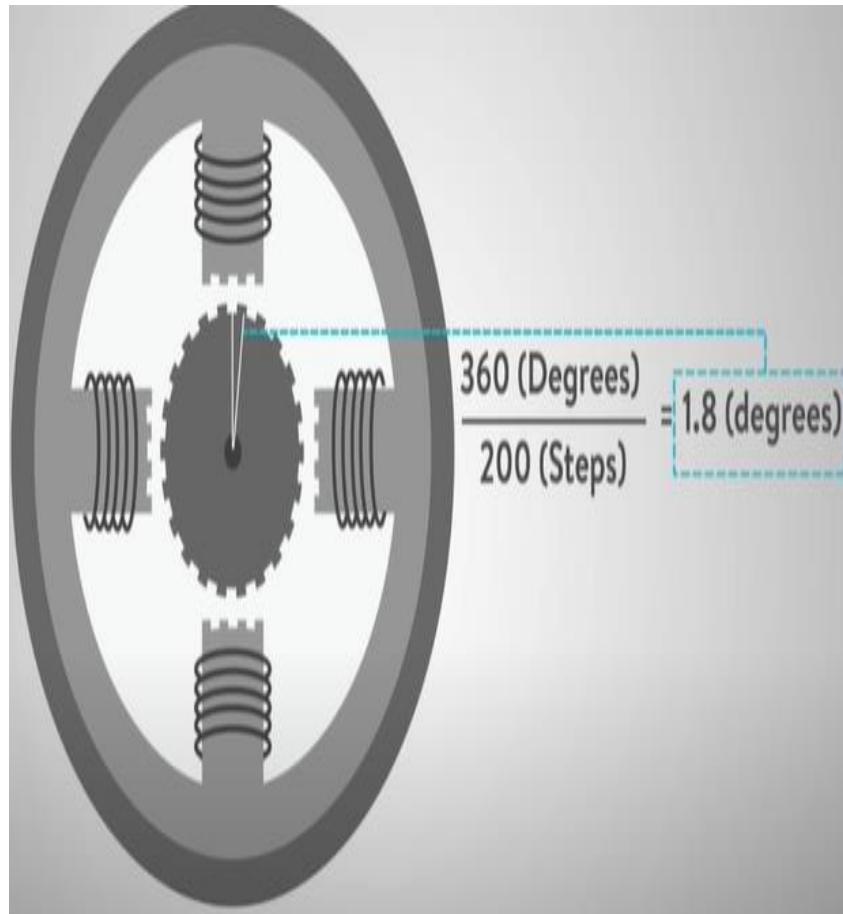


Working

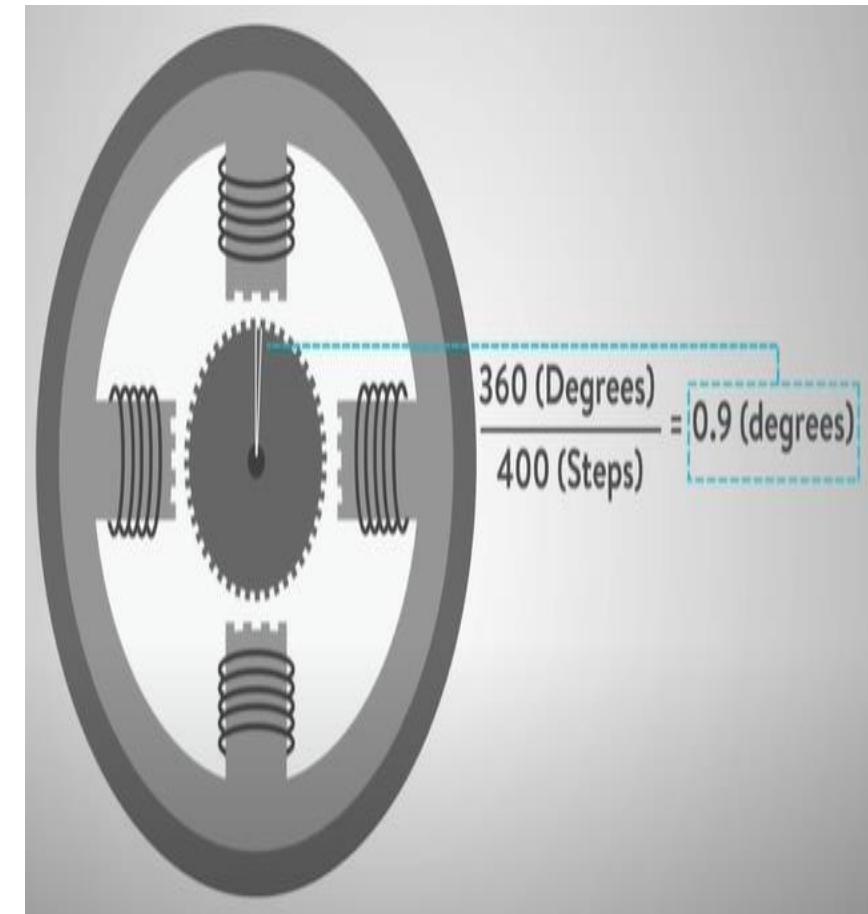


Step Angle Calculation

Full Step Rotation



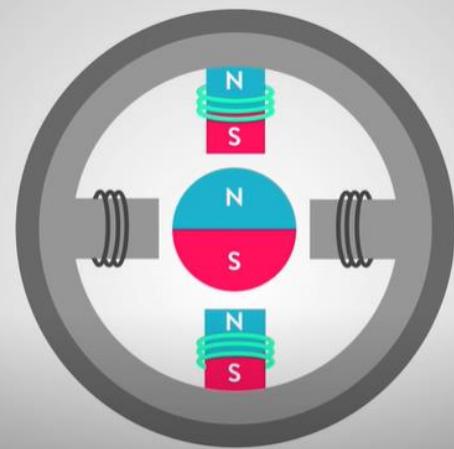
Half Step Rotation



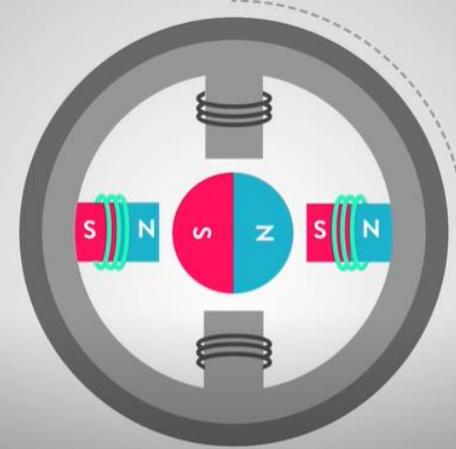


Rotational Mechanism

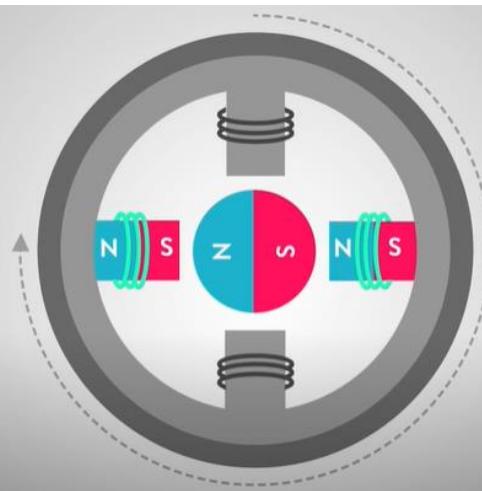
Position 1



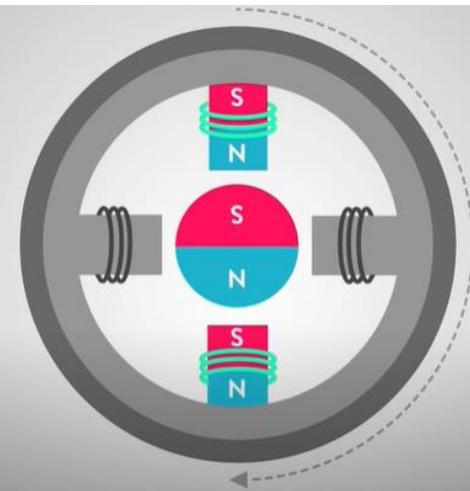
Position 2



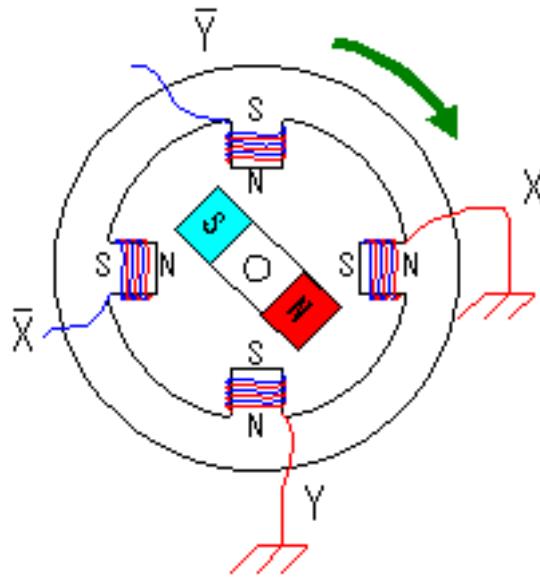
Position 4



Position 3



Rotational Mechanism – Clockwise



X	X̄	Y	Ȳ
0	1	0	1
1	0	0	1
1	0	1	0
0	1	1	0

Clockwise control

X, \bar{X} , Y and \bar{Y} are controlled in the following order.

X	\bar{X}	Y	\bar{Y}	Step angle
0	1	0	1	0°
1	0	0	1	90°
1	0	1	0	180°
0	1	1	0	270°

"0" means grounding.

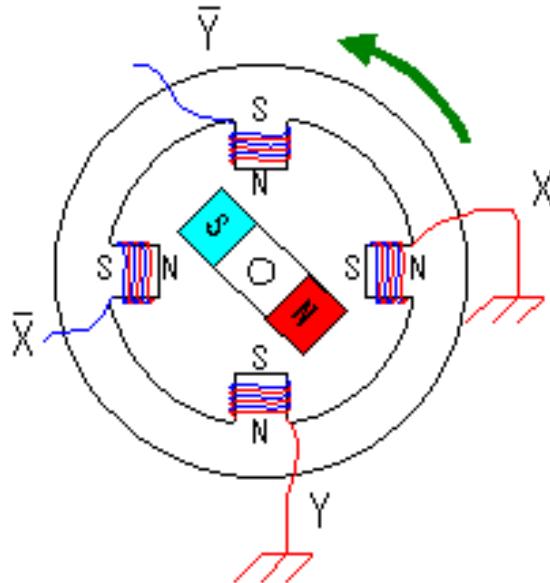


Clockwise rotation – Excitation Codes

The case of the clockwise control is shown below.
The combination of X, \bar{X} , Y and \bar{Y} repeats four patterns.

X	\bar{X}	Y	\bar{Y}	Step angle	X	\bar{X}	Y	\bar{Y}	Step angle
0	1	0	1	0.0°	0	1	0	1	180.0°
1	0	0	1	7.5°	1	0	0	1	187.5°
1	0	1	0	15.0°	1	0	1	0	195.0°
0	1	1	0	22.5°	0	1	1	0	202.5°
0	1	0	1	30.0°	0	1	0	1	210.0°
1	0	0	1	37.5°	1	0	0	1	217.5°
1	0	1	0	45.0°	1	0	1	0	225.0°
0	1	1	0	52.5°	0	1	1	0	232.5°
0	1	0	1	60.0°	0	1	0	1	240.0°
1	0	0	1	67.5°	1	0	0	1	247.5°
1	0	1	0	75.0°	1	0	1	0	255.0°
0	1	1	0	82.5°	0	1	1	0	262.5°
0	1	0	1	90.0°	0	1	0	1	270.0°
1	0	0	1	97.5°	1	0	0	1	277.5°
1	0	1	0	105.0°	1	0	1	0	285.0°
0	1	1	0	112.5°	0	1	1	0	292.5°
0	1	0	1	120.0°	0	1	0	1	300.0°
1	0	0	1	127.5°	1	0	0	1	307.5°
1	0	1	0	135.0°	1	0	1	0	315.0°
0	1	1	0	142.5°	0	1	1	0	322.5°
0	1	0	1	150.0°	0	1	0	1	330.0°
1	0	0	1	157.5°	1	0	0	1	337.5°
1	0	1	0	165.0°	1	0	1	0	345.0°
0	1	1	0	172.5°	0	1	1	0	352.5°

Rotational Mechanism – Anti-Clockwise



X	\bar{X}	Y	\bar{Y}
0	1	0	1
0	1	1	0
1	0	1	0
1	0	0	1

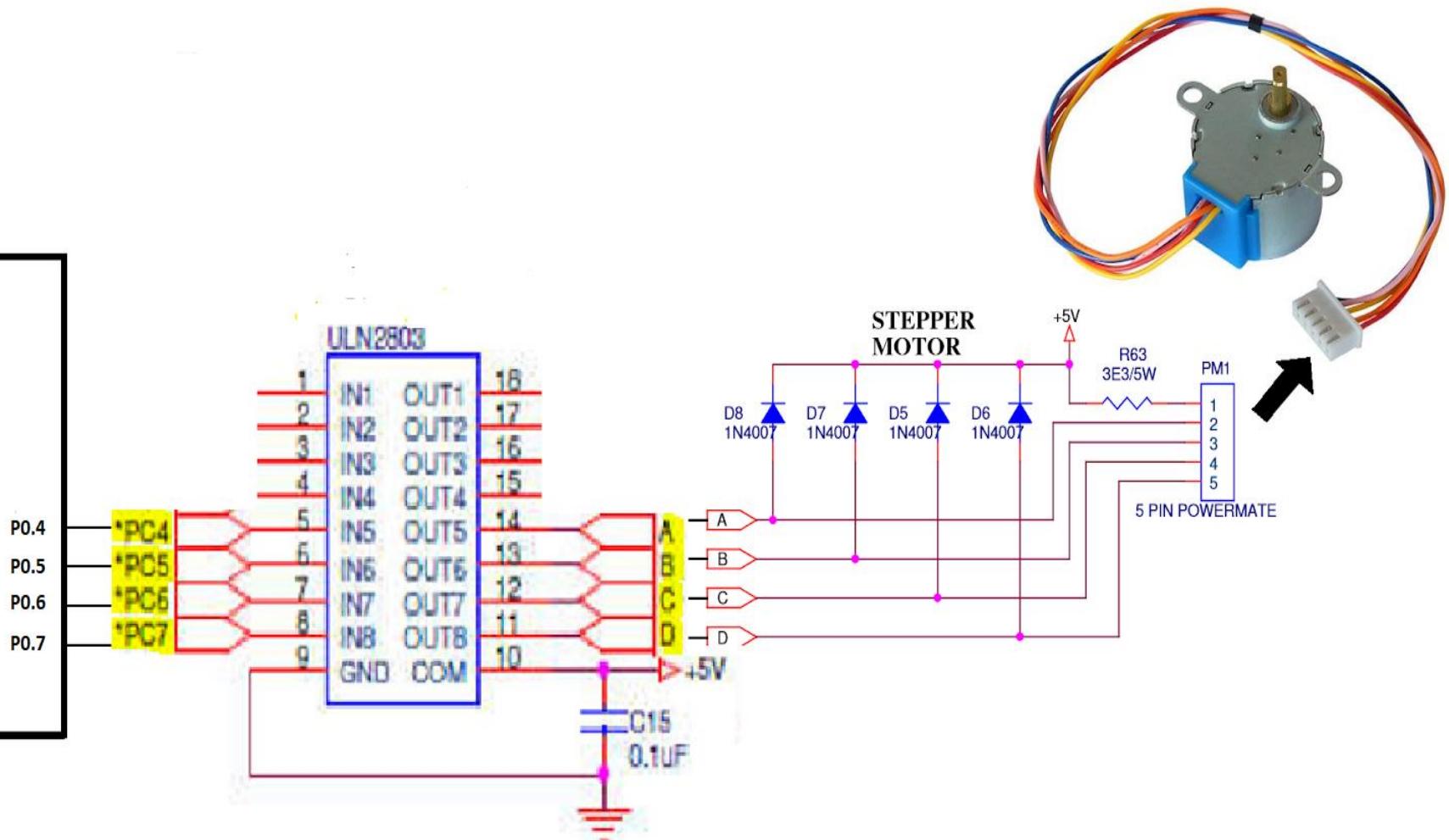
● Counterclockwise control

X , \bar{X} , Y and \bar{Y} are controlled in the following order.

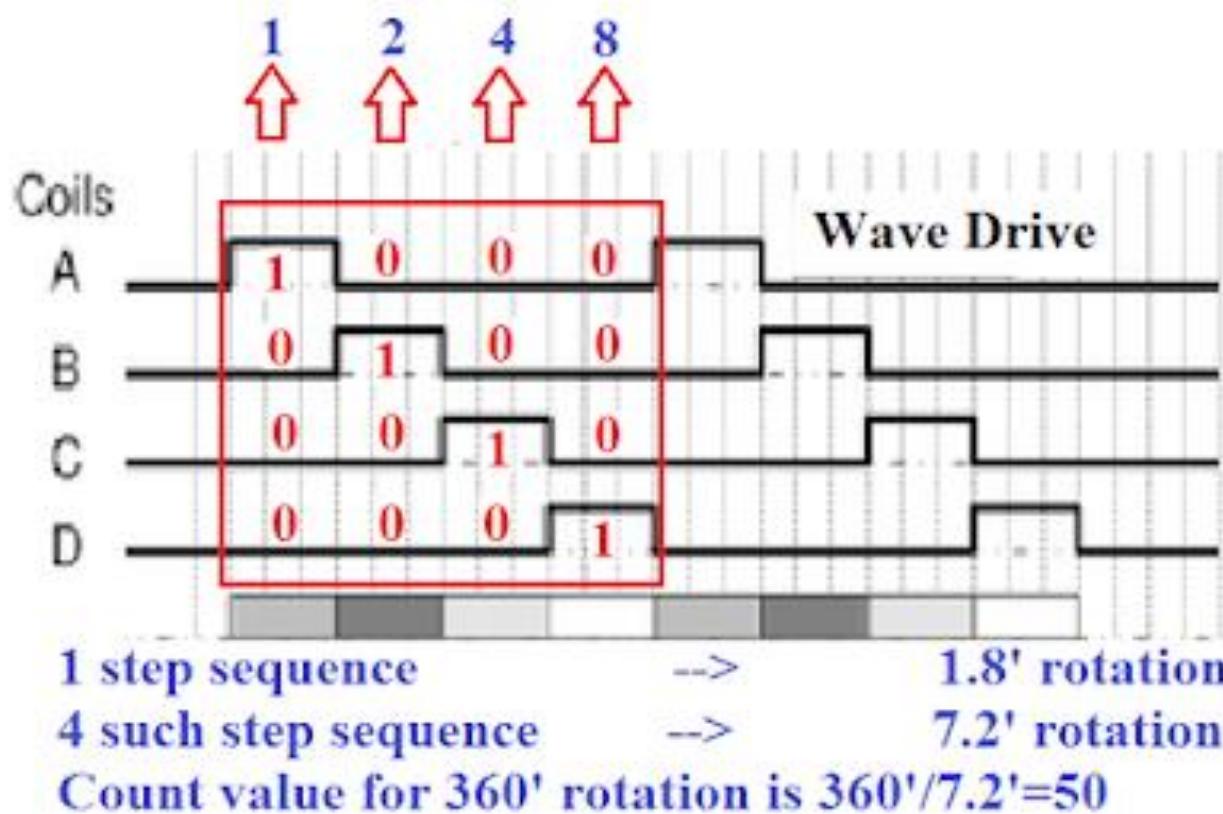
X	\bar{X}	Y	\bar{Y}	Step angle
0	1	0	1	0°
0	1	1	0	-90°
1	0	1	0	-180°
1	0	0	1	-270°

"0" means grounding.

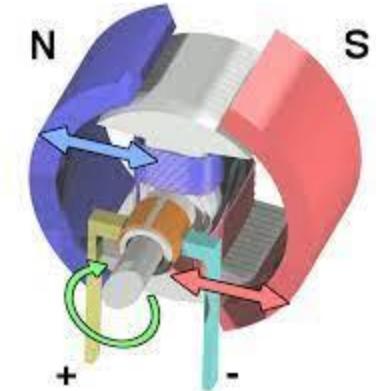
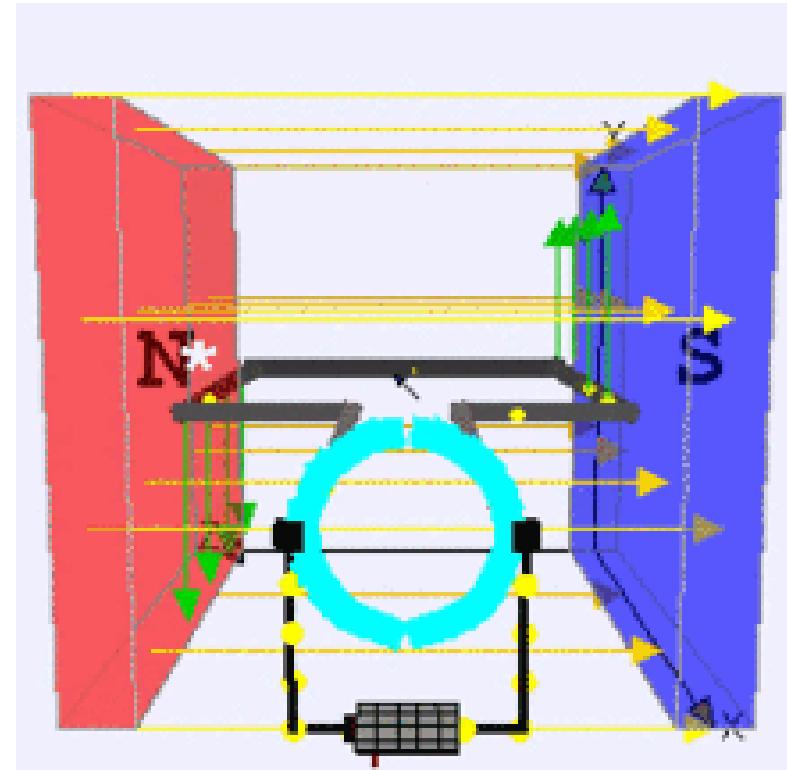
Interfacing Diagram



Interfacing Diagram



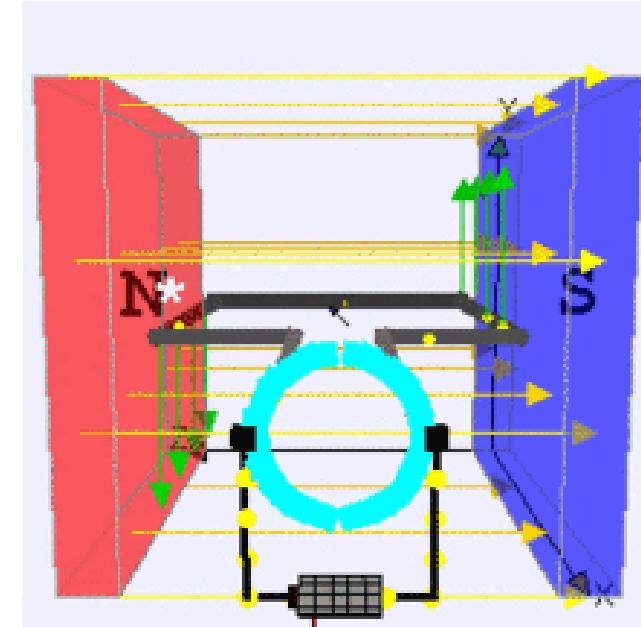
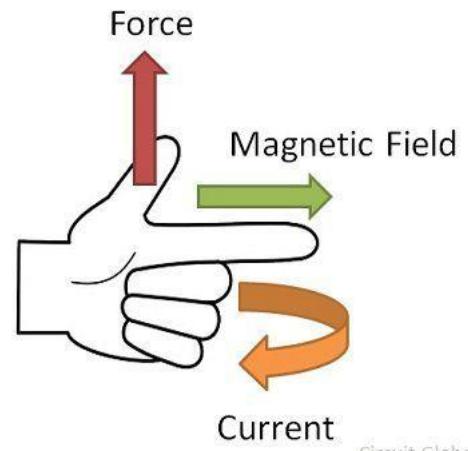
ARM LPC1768 – DC Motor Interfacing





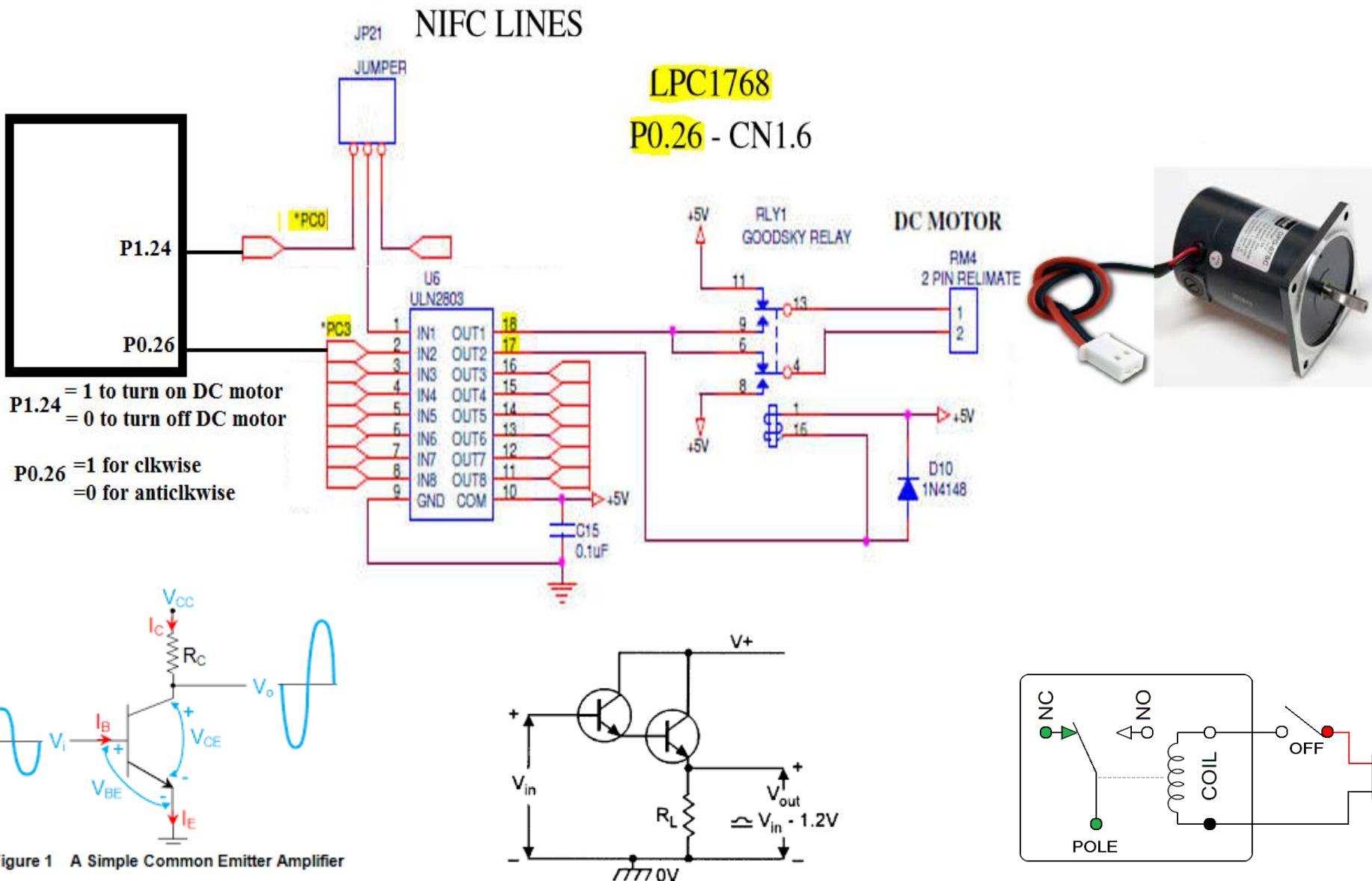
DC Motor - Introduction

- The DC motor is the device which converts the **Direct Current (DC) into the mechanical work.**
- It works on the principle of **Lorentz Law**, which states that “**the current-carrying conductor placed in a magnetic and electric field experience a force**”. The experienced force is called the Lorentz force.
- The Flemming left-hand rule gives the direction of the force.

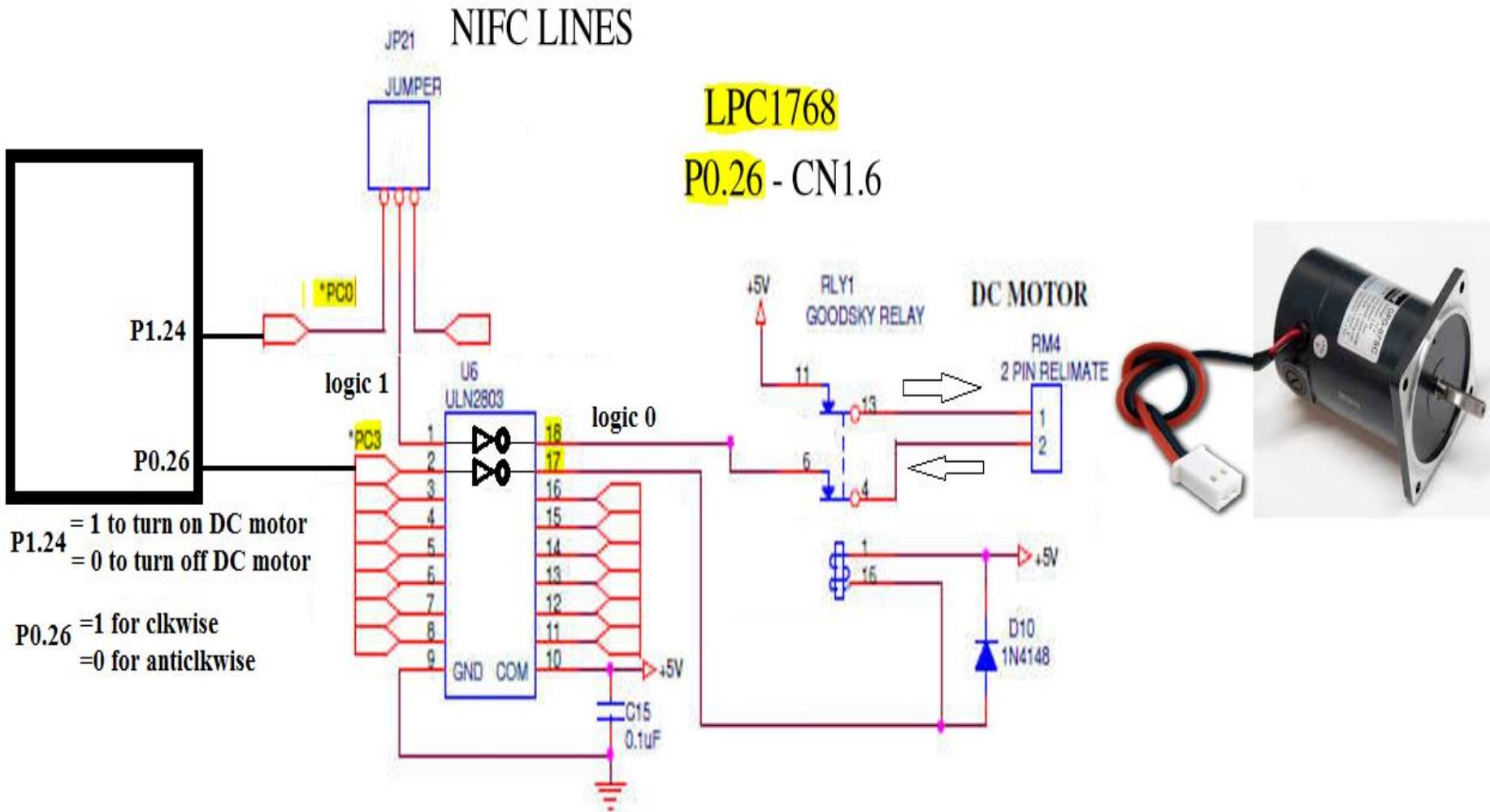




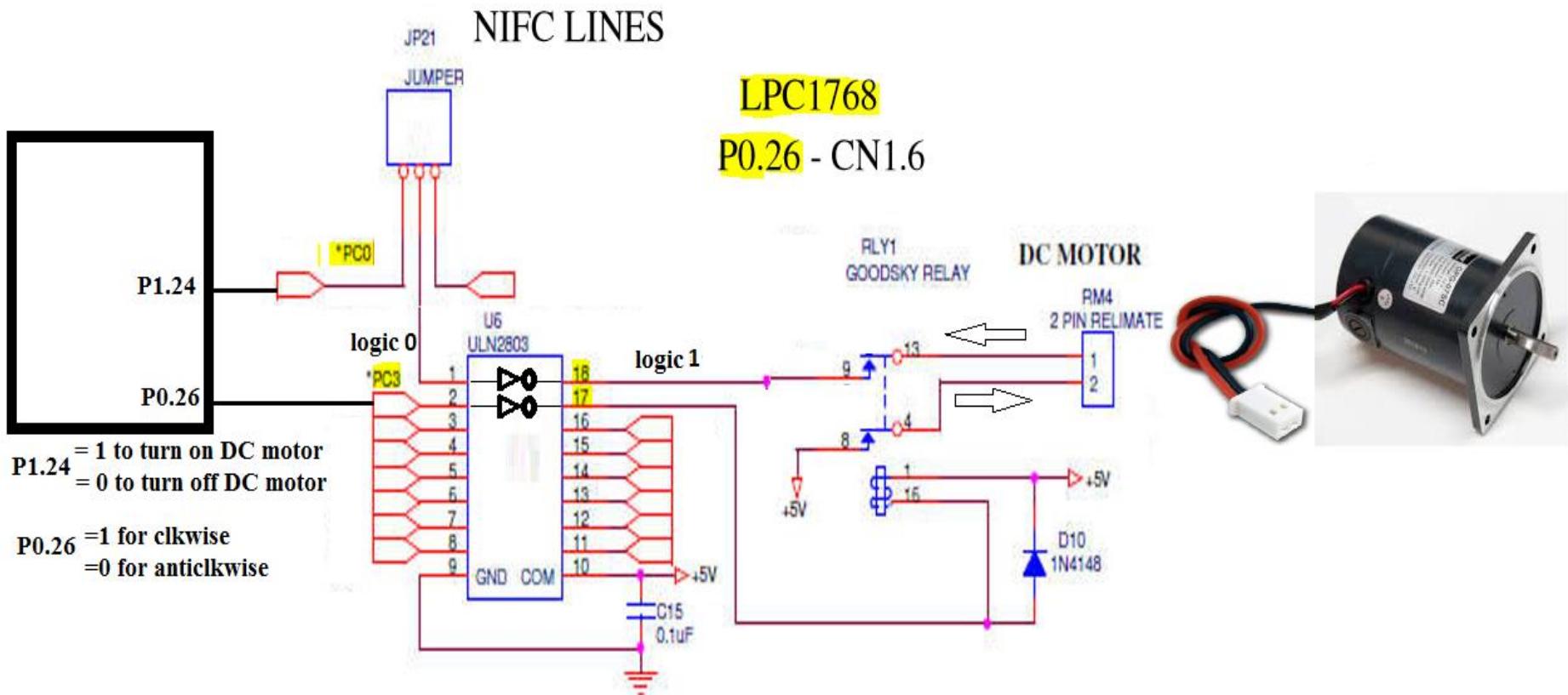
DC Motor - Interfacing



DC Motor – Interfacing – Case 1 : Clockwise

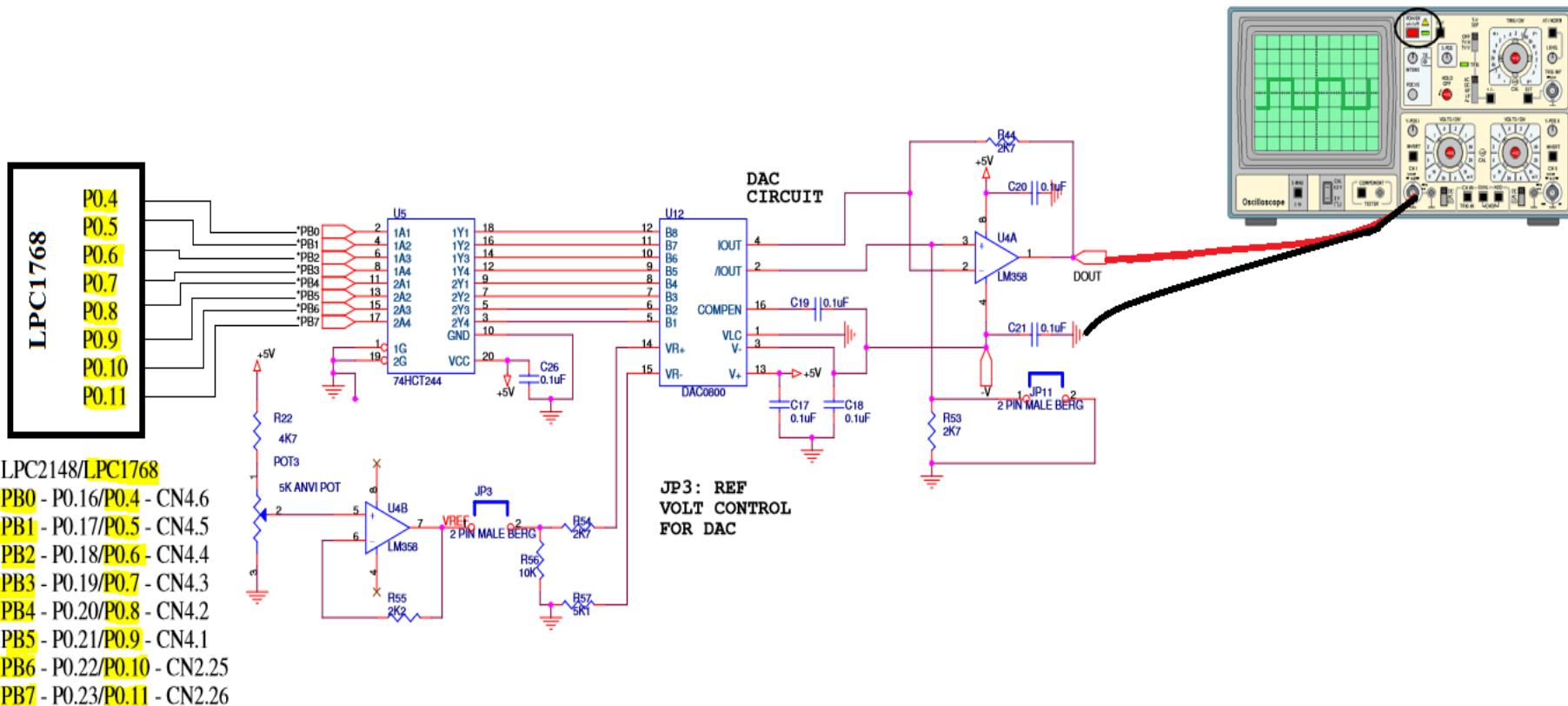


DC Motor – Interfacing – Case 2 : Anti-Clockwise





ARM LPC1768 – Digital to Analog Converter



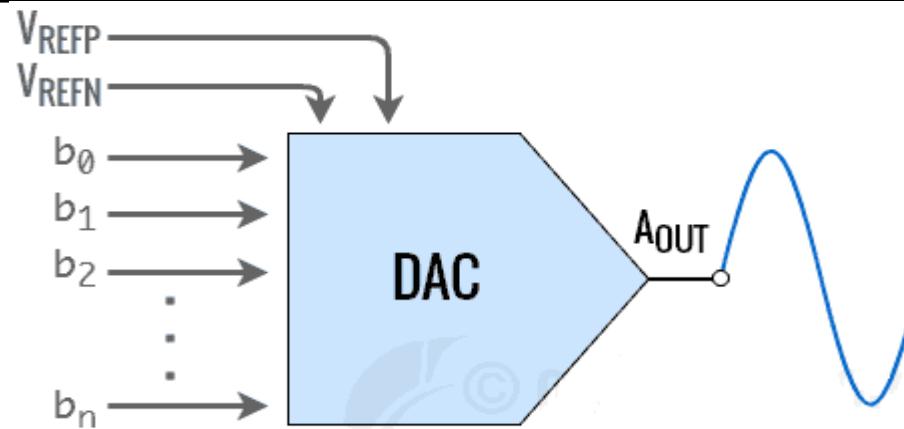


DAC Introduction

- DAC is abbreviated for Digital to Analog Converter. It is quite opposite to an ADC (Analog to Digital Converter).
- DACs are used in audio equipment like Music Players to convert the digital data into analog audio signals. Similarly, there are video DACs, for converting digital video data into analog video signals to be displayed on a screen.
- Types of **DAC implementations**: Switched Resistor DAC, R – 2R Ladder DAC, Successive Approximation DAC, etc.
- One of the simplest ways to implement a DAC is to use the PWM Functionality of a Timer Peripheral (if dedicated DAC unit is not present).
- Similar to an ADC, a DAC also has **resolution**. It indicates the number of possible values a DAC unit can produce between the voltage extremes.
- E.g.: 8-bit DAC can produce (2^8) 256 different levels for a voltage range.

- DAC is abbreviated for Digital to Analog Converter. It is quite opposite to an ADC (Analog to Digital Converter).
- It contains a 10-bit DAC peripheral based on Resistor String and the maximum update rate is 1 MHz.
- The resolution of the DAC in LPC1768 is 10-bit, it can produce (2^{10}) 1024 different values between the positive and the negative reference voltage levels.

DAC Channel	Port Pin	Pin Functions	Associated PINSEL Register
Aout	P0.26	0-GPIO, 1-AD0[3], 2-AOUT , 3-RXD3	20,21 bits of PINSEL1





DAC Calculations

- The Analog voltage at the output of this pin is given as:

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

- Here,
 - VAOUT = Output Analog Voltage of DAC
 - VREFP = Positive Reference Voltage of DAC
 - VREFN = Negative Reference Voltage of DAC
 - VALUE = 10-bit digital value, which must be converted to analog voltage.
- When we have VREFN = 0, the equation boils down to:

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



Pins Associated with DAC

- LPC1768 has only a single channel DAC, only one output pin of the DAC Peripheral. The remaining pins related to power.
- Table gives an overview of pins associated with DAC peripheral of LPC1768 MCU

Pin	Function	Description
AOUT – P0.26	Analog Output	Provides the analog output voltage with reference to the analog GND (VSSA).
V _{REFP} , V _{REFN}	Voltage References	These pins provide the positive and negative reference voltages for DAC and ADC. Usually isolated 3.3V and 0V.
V _{DDA} , V _{SSA}	Analog Power and GND	Same as V _{DD} and V _{ss} but isolated to minimize noise.

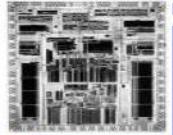


LPC1768 DAC Registers

- There are totally **three registers for DAC in LPC1768**. They are DACR, DACCTRL and DACCNTVAL.
- The DACCTRL and DACCNTVAL registers are associated with DMA operations in DAC.
- **DACR – D/A Converter Register:** It contains the digital value that must be converted to analog value. Also, this register contains a bit for a trade-off between power and performance.
- The field containing bits [15:6] is used to feed a digital value which needs to be converted and bit 16 is used to select settling time.
 - 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 the converted Analog voltage at the output. The formula for analog voltage at AOUT pin is as shown above.
 - **Bit[16] – BIAS:** Setting this bit to 0 selects settling time of 1us max with max current consumption of 700uA at max 1Mhz update rate. Setting it to 1 will select settling time of 2.5us but with reduce max current consumption of 300uA at max 400Khz update rate.
 - Bits[31:17]: Reserved



THANK YOU...

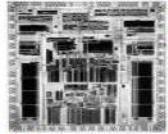


Timer/Counter Programming

Timer - Interval Timer to generate the intended delay . The Timer is designed to count cycles of the peripheral clock (PCLK)
Counter - Counting internal events. or an externally-supplied clock

There are four 32-bit Timers in LPC1768:

Timer0, Timer1, Timer2 and Timer3.



Timer/Counter Programming

PC – Prescale Counter Register: It is a 32-bit register. The value in PC is incremented on every PCLK cycle and when its value matches the value in PR, the TC is incremented and the value in PC is RESET on the next PCLK cycle.

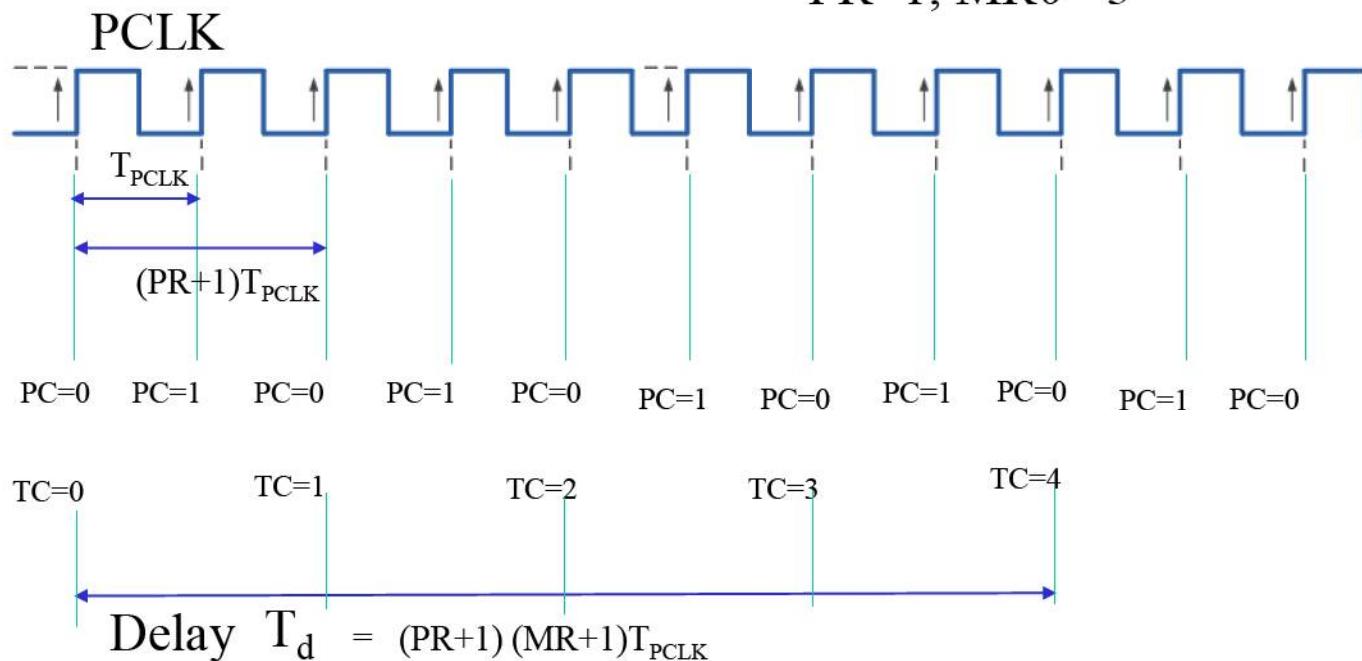
PR – Prescale Register: It is a 32-bit register and specifies the maximum value for the Prescale Counter (PC).

TC – Timer Counter Register: It is a 32-bit register and is incremented at every PR+1 cycles of PCLK.

MR0 – MR3 – Match Registers: Contains user loaded values to be compared with TC. When value in Match Register matches with TC, appropriate action can be performed.

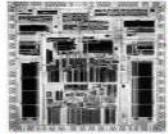
Timer/Counter Programming

PR=1, MR0= 3



$$T_{PCLK} = 1/f_{PCLK}$$

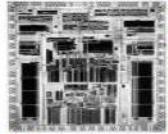
Ex: For frequency 3MHz, to get 1 second delay: PR= 999, MR0= 2999
PR= 2999, MR0= 999 etc.



Timer/Counter Programming

TCR – Timer Control register: Used to control Timer Counter functions i.e. enable, disable and reset.

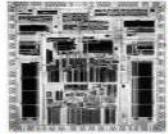
Bit 0	Counter Enable	When 1, TC and PC are enabled for counting. When 0, TC and PC are disabled.
Bit 1	Counter Reset	When 1, TC and PC are reset on next positive edge of PCLK. The counters remain reset until this bit is returned to 0.



Timer/Counter Programming

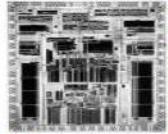
CTCR (Counter/Timer Control Register) is used to select between Timer and Counter mode, and in Counter mode to select the pin and edge(s) for counting.

Bit	Symbol	Value	Description
1:0	Counter/ Timer Mode	00	This field selects which rising PCLK edges can increment the Timer's Prescale Counter (PC), or clear the PC and increment the Timer Counter (TC).
		01	Timer Mode: the TC is incremented when the Prescale Counter matches the Prescale Register. The Prescale Counter is incremented on every rising PCLK edge.
		10	Counter Mode: TC is incremented on rising edges on the CAP input selected by bits 3:2.
		11	Counter Mode: TC is incremented on falling edges on the CAP input selected by bits 3:2.
3:2	Count Input Select	00	When bits 1:0 in this register are not 00, these bits select which CAP pin is sampled for clocking.
		01	CAPn.0 for TIMERn
		10	Reserved
		11	Reserved



Timer/Counter Programming

CAP0.0	P1.26
CAP0.1	P1.27
CAP1.0	P1.18 / P1.28 / P2.6
CAP1.1	P1.19 / P1.29
CAP2.0	P0.4
CAP2.1	P0.5
CAP3.0	P0.23
CAP3.1	P0.24
MAT0.0	P1.28 / P3.25
MAT0.1	P1.29 / P3.26
MAT1.0	P1.22
MAT1.1	P1.25
MAT2.0	P0.6 / P4.28
MAT2.1	P0.7 / P4.29
MAT2.2	P0.8
MAT2.3	P0.9
MAT3.0	P0.10
MAT3.1	P0.11



Timer/Counter Programming

Match Registers (MR0 - MR3)

- The Match register values are continuously compared to the Timer Counter value.
- When the two values are equal, actions can be triggered automatically.
- The action possibilities are to generate an interrupt, reset the Timer Counter, or stop the timer. Actions are controlled by the settings in the MCR register.

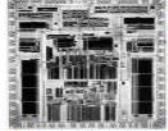


Timer/Counter Programming

Match Control Register (MCR)

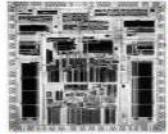
The Match Control Register is used to control what operations are performed when one of the Match Registers matches the Timer Counter.

Bit	Symbol	Value	Description
0	MR0I	1	Interrupt on MR0: an interrupt is generated when MR0 matches the value in the TC.
		0	This interrupt is disabled
1	MR0R	1	Reset on MR0: the TC will be reset if MR0 matches it.
		0	Feature disabled.
2	MR0S	1	Stop on MR0: the TC and PC will be stopped and TCR[0] will be set to 0 if MR0 matches the TC.
		0	Feature disabled.
3	MR1I	1	Interrupt on MR1: an interrupt is generated when MR1 matches the value in the TC.
		0	This interrupt is disabled
4	MR1R	1	Reset on MR1: the TC will be reset if MR1 matches it.
		0	Feature disabled.
5	MR1S	1	Stop on MR1: the TC and PC will be stopped and TCR[0] will be set to 0 if MR1 matches the TC.
		0	Feature disabled.



Timer/Counter Programming

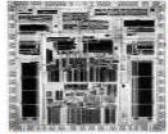
MR2I	1	Interrupt on MR2: an interrupt is generated when MR2 matches the value in the TC.
	0	This interrupt is disabled
MR2R	1	Reset on MR2: the TC will be reset if MR2 matches it.
	0	Feature disabled.
MR2S	1	Stop on MR2: the TC and PC will be stopped and TCR[0] will be set to 0 if MR2 matches the TC.
	0	Feature disabled.
MR3I	1	Interrupt on MR3: an interrupt is generated when MR3 matches the value in the TC.
	0	This interrupt is disabled
MR3R	1	Reset on MR3: the TC will be reset if MR3 matches it.
	0	Feature disabled.
MR3S	1	Stop on MR3: the TC and PC will be stopped and TCR[0] will be set to 0 if MR3 matches the TC.
	0	Feature disabled.
-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.



Timer/Counter Programming

External Match Register (EMR): It provides Match Outputs. Also, the External Match Register provides both control and status of the external match pins.

Bit	Symbol	Description
0	EM0	External Match 0. When a match occurs between the TC and MR0, this bit can either toggle, go low, go high, or do nothing, depending on bits 5:4 of this register. This bit can be driven onto a MATn.0 pin, in a positive-logic manner (0 = low, 1 = high).
1	EM1	External Match 1. When a match occurs between the TC and MR1, this bit can either toggle, go low, go high, or do nothing, depending on bits 7:6 of this register. This bit can be driven onto a MATn.1 pin, in a positive-logic manner (0 = low, 1 = high).
2	EM2	External Match 2. When a match occurs between the TC and MR2, this bit can either toggle, go low, go high, or do nothing, depending on bits 9:8 of this register. This bit can be driven onto a MATn.2 pin, in a positive-logic manner (0 = low, 1 = high).
3	EM3	External Match 3. When a match occurs between the TC and MR3, this bit can either toggle, go low, go high, or do nothing, depending on bits 11:10 of this register. This bit can be driven onto a MATn.3 pin, in a positive-logic manner (0 = low, 1 = high).
5:4	EMC0	External Match Control 0. Determines the functionality of External Match 0. Table 433 shows the encoding of these bits.
7:6	EMC1	External Match Control 1. Determines the functionality of External Match 1. Table 433 shows the encoding of these bits.
9:8	EMC2	External Match Control 2. Determines the functionality of External Match 2. Table 433 shows the encoding of these bits.
11:10	EMC3	External Match Control 3. Determines the functionality of External Match 3. Table 433 shows the encoding of these bits.

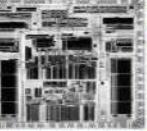


Timer/Counter Programming

**EMR[11:10], EMR[9:8], Function
EMR[7:6], or EMR[5:4]**

- | | |
|----|---|
| 00 | Do Nothing. |
| 01 | Clear the corresponding External Match bit/output to 0 (MATn.m pin is LOW if pinned out). |
| 10 | Set the corresponding External Match bit/output to 1 (MATn.m pin is HIGH if pinned out). |
| 11 | Toggle the corresponding External Match bit/output. |

MAT0.0	P1.28 / P3.25
MAT0.1	P1.29 / P3.26
MAT1.0	P1.22
MAT1.1	P1.25
MAT2.0	P0.6 / P4.28
MAT2.1	P0.7 / P4.29
MAT2.2	P0.8
MAT2.3	P0.9
MAT3.0	P0.10
MAT3.1	P0.11



Timer/Counter Programming

```
#include<stdio.h>
#include<LPC17xx.h>
void delay(void)
{
    LPC_TIM0->TCR = 0x00000002; // Timer0 Reset
    LPC_TIM0->EMR = 0X20;//Set match bit upon match
    LPC_TIM0->PR = 1000; /
    LPC_TIM0->MR0 = 3000;      //for 1 second
    LPC_TIM0->MCR = 0x00000004; // stop PC and TC on MR0
    LPC_TIM0->TCR = 0x00000001; // Timer0 Enable
    while ( !(LPC_TIM0->EMR & 0x01)); // wait until match

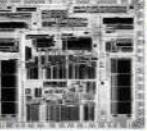
    return;
}

int main(void)
{
    LPC_GPIO0->FIODIR=0x00000004;

    while(1)
    {
        LPC_GPIO0->FIOPIN=~(LPC_GPIO0->FIOPIN & 0x00000004);
        Delay();
        LPC_GPIO0->FIOSET=0x4;
        delay();

        LPC_GPIO0->FIOCLR=0x4;
        delay();
    }
}
```

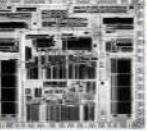
Toggle LED connected to P0.2 every second.i.e generate square wave with period 2 seconds



Timer/Counter Programming

```
#include<stdio.h>
#include<LPC17xx.h>
void delay(void)
{
    //LPC_SC->PCONP |= (1<<1); //powers the T0
    LPC_TIM0->TCR = 0x00000002; // Timer0 Reset
    LPC_TIM0->EMR = 0X20; //Set EM0 upon match
    LPC_TIM0->PR = 2999;
    LPC_TIM0->MCR = 0x00000004; // stop PC and TC on MRO
    LPC_TIM0->TCR = 0x00000001; // Timer0 Enable
    while ( !(LPC_TIM0->EMR & 0x01));// Wait until EM0 is set
    return;
}
int main(void)
{
    LPC_GPIO0->FIODIR=0x00000004;
    while(1)
    {
        LPC_GPIO0->FIOPIN=0x00000004;
        LPC_TIM0->MRO = 1500; //For 1.5 seconds
        delay();
        LPC_GPIO0->FIOPIN=0x00000000;
        LPC_TIM0->MRO = 500; //For 0.5 seconds
        delay();
    }
}
```

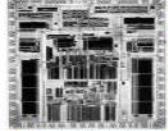
Generate square wave of period 2 seconds with 75% duty cycle on P0.2



Timer/Counter Programming

Square waveform on MAT 0.0 output line by taking EM0 on the output pin.

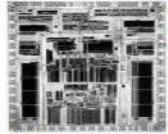
```
#include<stdio.h>
#include<LPC17xx.h>
void delay(void)
{
    LPC_TIM0->TCR = 0x00000002; // Timer0 Reset
    LPC_TIM0->CTCR = 0x00000000;
    LPC_TIM0->EMR = 0X30;//Toggle bit upon match
    LPC_TIM0->PR = 0; //
    LPC_TIM0->MRO = 3000000; //
    LPC_TIM0->MCR = 0x00000002; // Reset TC
    LPC_TIM0->TCR = 0x00000001; // Timer0 Enable
    return;
}
int main(void)
{
    LPC_PINCON->PINSEL3 |= (3<<24);//Get EM0 on MAT0.0 (P1.28) line
    delay();
    while(1);
}
```



Timer/Counter Programming

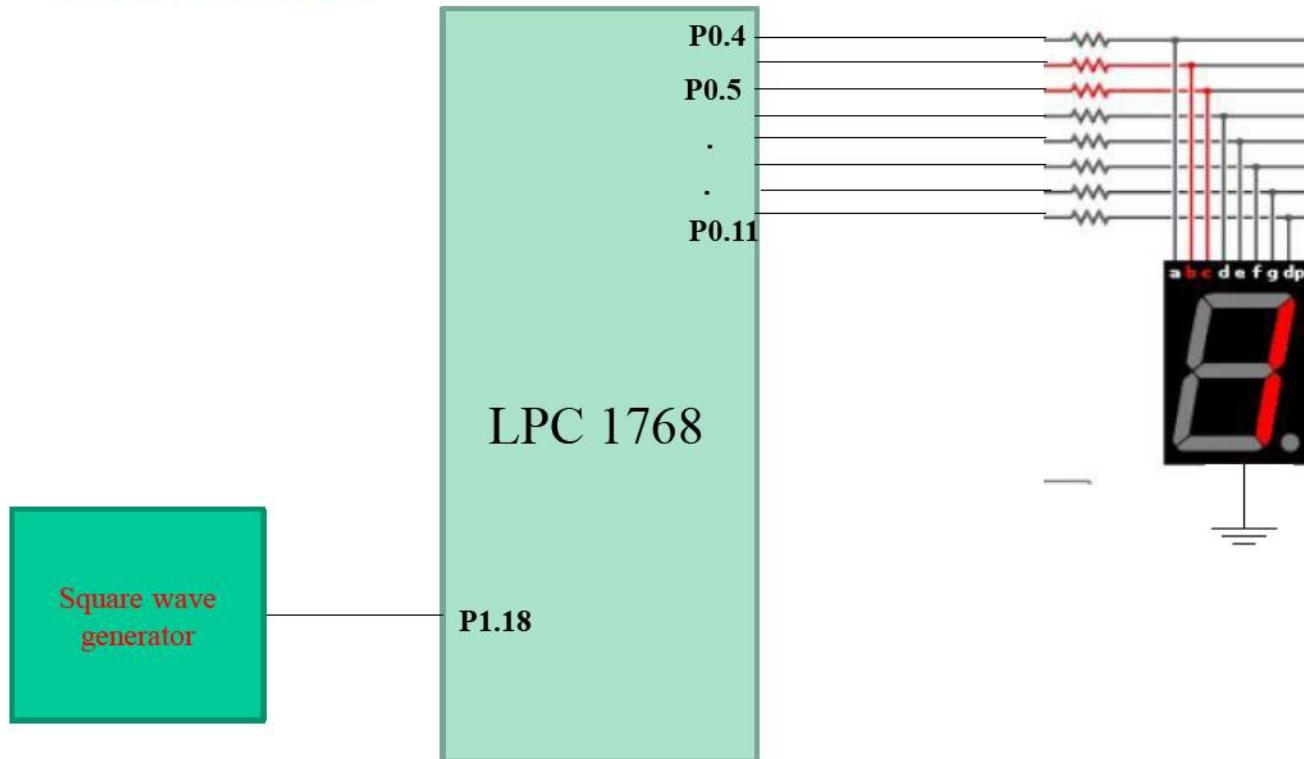
MAT 1.1(P1.25) toggles whenever count reaches 3. CAP 1.0 (P1.18) is counter clock. i.e Divide the frequency of the square waveform input at P1.18 by a factor of 8 on P1.25

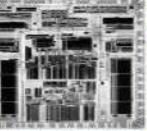
```
#include<stdio.h>
#include<LPC17xx.h>
void init_timer1(void)
{
    LPC_PINCON->PINSEL3 |=(3<<18 | 3<<4); // MAT 1.1(P1.25) and CAP 1.0 (P1.18)
    LPC_TIM1->TCR=2; //Reset Counter1
    LPC_TIM1->CTCR = 0x2; // Counter at -ve edge of CAP1.0
    LPC_TIM1->MR1=0x03; //To count 4 clock pulses
    LPC_TIM1->MCR=0x10; //Clear TC upon Match1
    LPC_TIM1->EMR=0xC0; //Toggle EM1 upon Match
    LPC_TIM1->TCR=1; //Start Counter1
}
int main(void)
{
    init_timer1();
    while(1);
}
```



Timer/Counter Programming

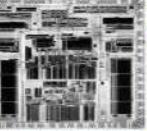
Assume that output of a square wave generator (Frequency < 10Hz) is connected to P1.18 (CAP1.0, Function-3) , write a program to display the frequency of this square waveform on the seven segment connected to P011-P0.4.





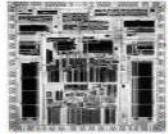
Timer/Counter Programming

```
#include<stdio.h>
#include<LPC17xx.h>
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 1 ms
    LPC_TIM0->MRO = 1000; //for 1 second
    LPC_TIM0->MCR = 0x00000004; // stop PC and TC on MRO
    LPC_TIM0->TCR = 0x00000001; // Timer0 Enable
    while ( !(LPC_TIM0->EMR & 0x01)); // wait until match
}
void init_counter1(void)
{
    LPC_PINCON->PINSEL3 = (3<<4);// cap 1.0 (P1.18)
    LPC_TIM1->CTCR = 0x01; // Counter at +ve edge of CAP1.0
}
```



Timer/Counter Programming

```
int main(void)
{
    LPC_PINCON->PINSEL0 = 0      //P0.4 to P0.11 GPIO data lines
    LPC_GPIO0->FIODIR = 0x00000FF0;      //P0.4 to P0.11 output
    init_counter1();
    while(1)
    {
        LPC_TIM1->TCR=2;//Reset Counter1
        LPC_TIM1->TCR=1;//Start Counter1
        Delay(); // wait for 1 second
        LPC_GPIO0->FIOPIN = seven_seg[LPC_TIM1->TC ] << 4; Counter1 on the
seven segment
    }
}
```

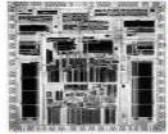


Timer/Counter Programming

Capture Registers (CR0 - CR1)

Each Capture register is associated with a device pin and may be loaded with the Timer Counter value when a specified event occurs on that pin.

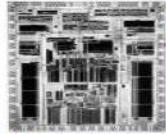
The settings in the Capture Control Register determine whether the capture function is enabled, and whether a capture event happens on the rising edge of the associated pin, the falling edge, or on both edges.



Timer/Counter Programming

Capture Control Register (CCR)

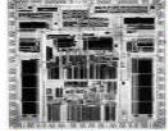
The Capture Control Register is used to control whether one of the four Capture Registers is loaded with the value in the Timer Counter when the capture event occurs, and whether an interrupt is generated by the capture event.



Timer/Counter Programming

Capture Control Register (CCR)

Bit	Symbol	Value	Description
0	CAP0RE	1	Capture on CAPn.0 rising edge: a sequence of 0 then 1 on CAPn.0 will cause CR0 to be loaded with the contents of TC.
		0	This feature is disabled.
1	CAP0FE	1	Capture on CAPn.0 falling edge: a sequence of 1 then 0 on CAPn.0 will cause CR0 to be loaded with the contents of TC.
		0	This feature is disabled.
2	CAP0I	1	Interrupt on CAPn.0 event: a CR0 load due to a CAPn.0 event will generate an interrupt.
		0	This feature is disabled.
3	CAP1RE	1	Capture on CAPn.1 rising edge: a sequence of 0 then 1 on CAPn.1 will cause CR1 to be loaded with the contents of TC.
		0	This feature is disabled.
4	CAP1FE	1	Capture on CAPn.1 falling edge: a sequence of 1 then 0 on CAPn.1 will cause CR1 to be loaded with the contents of TC.
		0	This feature is disabled.
5	CAP1I	1	Interrupt on CAPn.1 event: a CR1 load due to a CAPn.1 event will generate an interrupt.
		0	This feature is disabled.

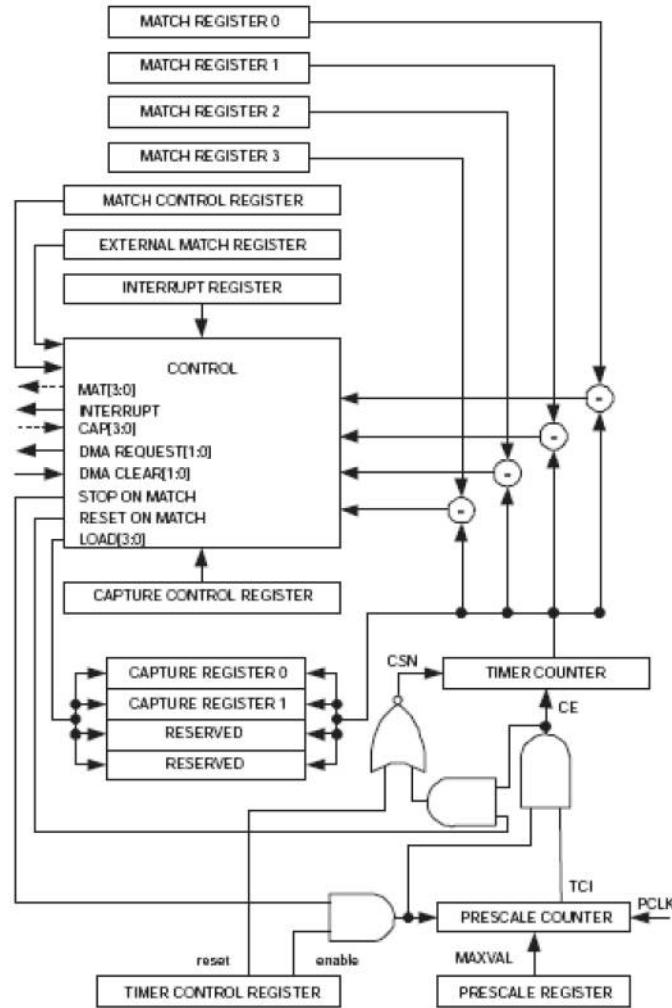


Timer/Counter Programming

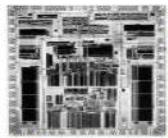
```
#include<stdio.h>           Capture TC into TC when +ve edge is applied to CAP0.0 (P1.26) or CAP0.1(P1.27)
#include<LPC17xx.h>
void delay(void)
{
    //LPC_SC->PCONP |= (1<<1); //powers the T0
    LPC_TIM0->CCR=9;//capture on positive edge
    LPC_TIM0->TCR = 0x00000002; // Timer0 Reset
    LPC_TIM0->EMR = 0X20;//Set match bit upon match
    LPC_TIM0->PR = 3000; //for 1 ms
    LPC_TIM0->MR0 = 1000;           //for 1 second
    LPC_TIM0->MCR = 0x00000004; // stop PC and TC on MR0
    LPC_TIM0->TCR = 0x00000001; // Timer0 Enable
    while ( !(LPC_TIM0->EMR & 0x01)); // wait until match
    return;
}

int main(void)
{
    LPC_GPIO0->FIODIR=0x00000004;
    LPC_PINCON->PINSEL3 |=(3<<20) | (3<<22);//select cap 0.0 and cap 0.1
    while(1)
    {
        LPC_GPIO0->FIOPIN=~(LPC_GPIO0->FIOPIN & 0x00000004);//toggle p0.2
        delay();
    }
}
```

Timer/Counter Programming

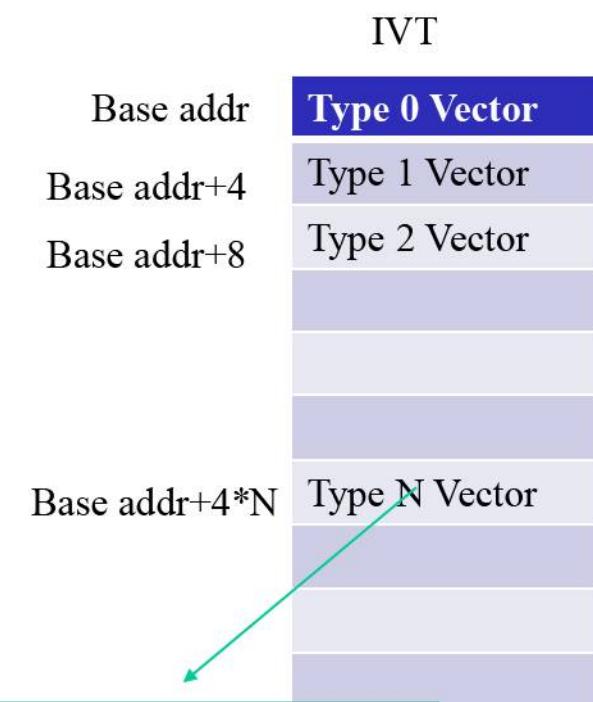


Timer Block diagram

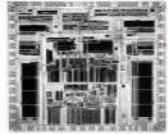


Nested Vectored Interrupt Controller(NVIC)

- Controls system exceptions and peripheral interrupts
- In the LPC176x, the NVIC supports 35 vectored interrupts
- Each peripheral device may have one or more interrupt lines to the Vectored Interrupt Controller.
- Interrupt numbers relate to where entries are stored in the Interrupt vector table.
- Interrupt Vector – Is the address of Interrupt Service Subroutine (ISR)
- Interrupt vector of Interrupt Type N is stored at an offset $N*4$ from the base address of Interrupt Vector Table(IVT)
- If the peripheral device is enabled to generate Interrupt when some event

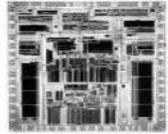


When Interrupt occurs, NVIC loads vector to PC and executes the ISR to provide the service to peripheral



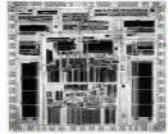
Nested Vectored Interrupt Controller(NVIC)

- If the peripheral device is enabled to generate Interrupt when some event occurs, the INTR request is sent to NVIC
- If NVIC is enabled to service the INTR request from the peripheral, it services the INTR request by executing ISR pertaining to the peripheral.(i.e Save the return address, Get the Interrupt Vector from IVT and load that address to PC. Upon completion of ISR execution resumes the calling function)



Nested Vectored Interrupt Controller(NVIC)

- In case of Timer, there are 6 INTR enable flags (4 in MCR and 2 in CCR. i.e 4 Match events and 2 Capture events can generate the Interrupt when the event occurs)
- When the event occurs the corresponding bit is set automatically in the IR register. This indicates the NVIC about the event
- If the NVIC is enabled to service the Timer Interrupt, it executes the corresponding ISR and gives the desired service to the Timer.
- In the ISR, clear the corresponding bit, by writing back 1.



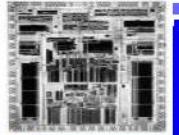
Timer/Counter Interrupt Programming

Interrupt Register (IR)

The Interrupt Register consists of 4 bits for the match interrupts and 2 bits for the capture interrupts. If an interrupt is generated then the corresponding bit in the IR will be high. Otherwise, the bit will be low.

Writing a logic one to the corresponding IR bit will reset the interrupt. Writing a zero has no effect.

Bit	Symbol	Description
0	MR0 Interrupt	Interrupt flag for match channel 0.
1	MR1 Interrupt	Interrupt flag for match channel 1.
2	MR2 Interrupt	Interrupt flag for match channel 2.
3	MR3 Interrupt	Interrupt flag for match channel 3.
4	CR0 Interrupt	Interrupt flag for capture channel 0 event.
5	CR1 Interrupt	Interrupt flag for capture channel 1 event.

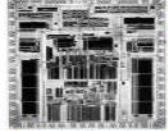


Timer/Counter Interrupt Programming

```
#include<stdio.h>
#include<LPC17xx.h>
unsigned int ticks=0,x;
void TIMER0_IRQHandler(void)
{
    LPC_TIM0->IR = 1;
    ticks++;
    if(ticks==1000)
    {
        ticks=0;
        LPC_GPIO0->FIOPIN=~(LPC_GPIO0->FIOPIN & 0x00000004);
    }
}
void init_timer0(void)
{
    LPC_TIM0->TCR = 0x00000002; // Timer0 Reset
    LPC_TIM0->CTCR =0x00;//Timer
    LPC_TIM0->MR0 = 2999; // For 1ms
    LPC_TIM0->EMR = 0X00;//Do nothing for EM0
    LPC_TIM0->PR = 0;
    LPC_TIM0->MCR = 0x00000003; //Reset TC upon Match-0 and generate INTR
    LPC_TIM0->TCR = 0x00000001; // Timer0 Enable

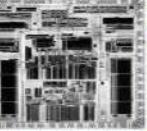
    return;
}
```

Toggle LED connected to p0.2 every second while displaying the status of switch connected to P1.0 on the LED connected to P2.0



Timer/Counter Interrupt Programming

```
int main(void)
{
    LPC_GPIO0->FIODIR=0x00000004;
    LPC_GPIO2->FIODIR=0x00000001;
    init_timer0();
    NVIC_EnableIRQ(TIMER0_IRQn);//timer 0 intr enabled in NVIC
    while(1)
    {
        LPC_GPIO2->FIOPIN=(LPC_GPIO1->FIOPIN & 0x01);
    }
}
```



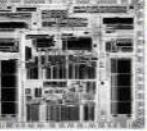
Timer/Counter Interrupt Programming

Toggle P0.2 whenever counter value reaches 3. I. e for every 4 edges using counter interrupt.

```
#include<stdio.h>
#include<LPC17xx.h>
void TIMER0_IRQHandler(void)
{
    LPC_TIM0->IR = 1; //Clear the interrupt
    LPC_GPIO0->FIOPIN=~(LPC_GPIO0->FIOPIN & 0x00000004);

}
void init_timer0(void)
{
    LPC_TIM0->TCR = 0x00000002; // Timer0 Reset
    LPC_TIM0->CTCR =0x05; // Counter at +ve edge of CAP0.1
    LPC_TIM0->MRO = 3;
    LPC_TIM0->EMR = 0X00;
    LPC_TIM0->PR = 0;
    LPC_TIM0->MCR = 0x00000003;
    LPC_TIM0->TCR = 0x00000001; // Timer0 Enable
    return;
}

int main(void)
{
    LPC_GPIO0->FIODIR=0x00000004;
    LPC_PINCON->PINSEL3 |=((3<<22)|(3<<24));
    init_timer0();
    NVIC_EnableIRQ(TIMER0_IRQn);
    while(1);
}
```

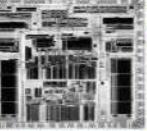


Timer/Counter Interrupt Programming

Timer interrupt for rectangular waveform generation (1.5 second HIGH and 0.5 second LOW)

```
include<stdio.h>
#include<LPC17xx.h>
unsigned char flag=1;
void TIMER0_IRQHandler(void)
{
    LPC_TIM0->IR = 1;

    if(flag)
    {
        flag=0;
        LPC_TIM0->TCR = 0x00000002; // Timer0 Reset
        LPC_GPIO0->FIOCLR=0x00000004;
        LPC_TIM0->MRO = 500;
        LPC_TIM0->TCR = 0x00000001; // Timer0 Enable
    }
    else
    {
        flag=1;
        LPC_TIM0->TCR = 0x00000002; // Timer0 Reset
        LPC_GPIO0->FIOSET=0x00000004;
        LPC_TIM0->MRO = 1500;
        LPC_TIM0->TCR = 0x00000001; // Timer0 Enable
    }
}
```



Timer/Counter Interrupt Programming

```
void init_timer0(void)
{
    LPC_TIM0->TCR = 0x00000002; // Timer0 Reset
    LPC_TIM0->CTCR =0x00;
    LPC_TIM0->MRO = 1500;
    LPC_TIM0->EMR = 0X00;
    LPC_TIM0->PR = 3000;
    LPC_TIM0->MCR = 0x00000005;
    LPC_TIM0->TCR = 0x00000001; // Timer0 Enable
    LPC_GPIO0->FIOSET=0x00000004;
    return;
}
int main(void)
{
    LPC_GPIO0->FIODIR=0x00000004;
    init_timer0();
    NVIC_EnableIRQ(TIMERO IRQn);
    while(1);
}
```

XIAMEN AMOTEC DISPLAY CO.,LTD

**SPECIFICATIONS OF
LCD MODULE**

MODULE NO : ADM1602K-NSW-FBS/3.3V

DOC.REVISION: 00

	SIGNATURE	DATE
PREPARED BY (RD ENGINEER)	QIU	2008-10-29
CHECKED BY	<i>Chen</i>	2008-10-29
APPROVED BY	<i>yf</i>	2008-10-29

DOCUMENT REVISION HISTORY

VERSINO	DATE	DESCRIPTION	CHANGED BY
00	Oct-29-2008	First issue	

CONTENTS

Item	Page
Functions & Features	3
Mechanical specifications	3
Dimensional Outline	4
Absolute maximum ratings	5
Block diagram	5
Pin description	5
Contrast adjust	6
Optical characteristics	6
Electrical characteristics	6
Timing Characteristics	7-8
Instruction description	9-12
Display character address code: character pattern	12 13
Quality Specifications	14--21

1. Features

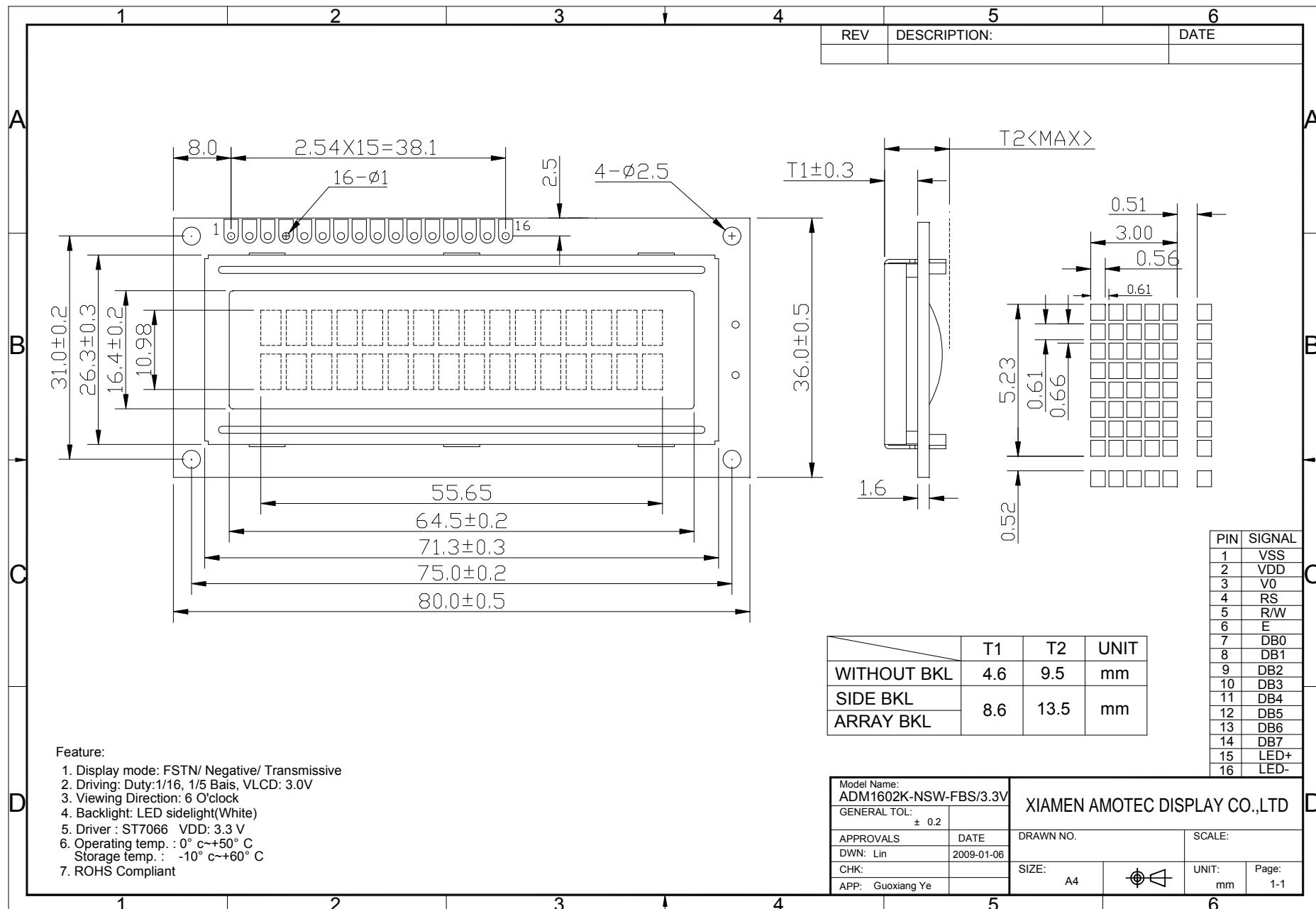
1. 5x8 dots with cursor
2. 16characters *2lines display
3. 4-bit or 8-bit MPU interfaces
4. Built-in controller (ST7066 or equivalent)
5. Display Mode & Backlight Variations
6. ROHS Compliant

LCD type	<input type="checkbox"/> TN			
	<input type="checkbox"/> FSTN	<input checked="" type="checkbox"/> FSTN Negative		
	<input type="checkbox"/> STN Yellow Green		<input type="checkbox"/> STN Gray	<input type="checkbox"/> STN Blue Negative
View direction	<input checked="" type="checkbox"/> 6 O'clock		<input type="checkbox"/> 12 O'clock	
Rear Polarizer	<input type="checkbox"/> Reflective		<input type="checkbox"/> Transflective	<input checked="" type="checkbox"/> Transmissive
Backlight Type	<input checked="" type="checkbox"/> LED	<input type="checkbox"/> EL	<input type="checkbox"/> Internal Power	<input checked="" type="checkbox"/> 3.3V Input
		<input type="checkbox"/> CCFL	<input checked="" type="checkbox"/> External Power	<input type="checkbox"/> 5.0V Input
Backlight Color	<input checked="" type="checkbox"/> White	<input type="checkbox"/> Blue	<input type="checkbox"/> Amber	<input type="checkbox"/> Yellow-Green
Temperature Range	<input checked="" type="checkbox"/> Normal		<input type="checkbox"/> Wide	<input type="checkbox"/> Super Wide
DC to DC circuit	<input type="checkbox"/> Build-in		<input checked="" type="checkbox"/> Not Build-in	
Touch screen	<input type="checkbox"/> With		<input checked="" type="checkbox"/> Without	
Font type	<input checked="" type="checkbox"/> English-Japanese		<input type="checkbox"/> English-Europen	<input type="checkbox"/> English-Russian
			<input type="checkbox"/> Other	

2. MECHANICAL SPECIFICATIONS

Module size	80.0mm(L)*36.0mm(W)* Max13.5(H)mm
Viewing area	64.5mm(L)*16.4mm(W)
Character size	3.00mm(L)*5.23mm(W)
Character pitch	3.51mm(L)*5.75mm(W)
Weight	Approx.

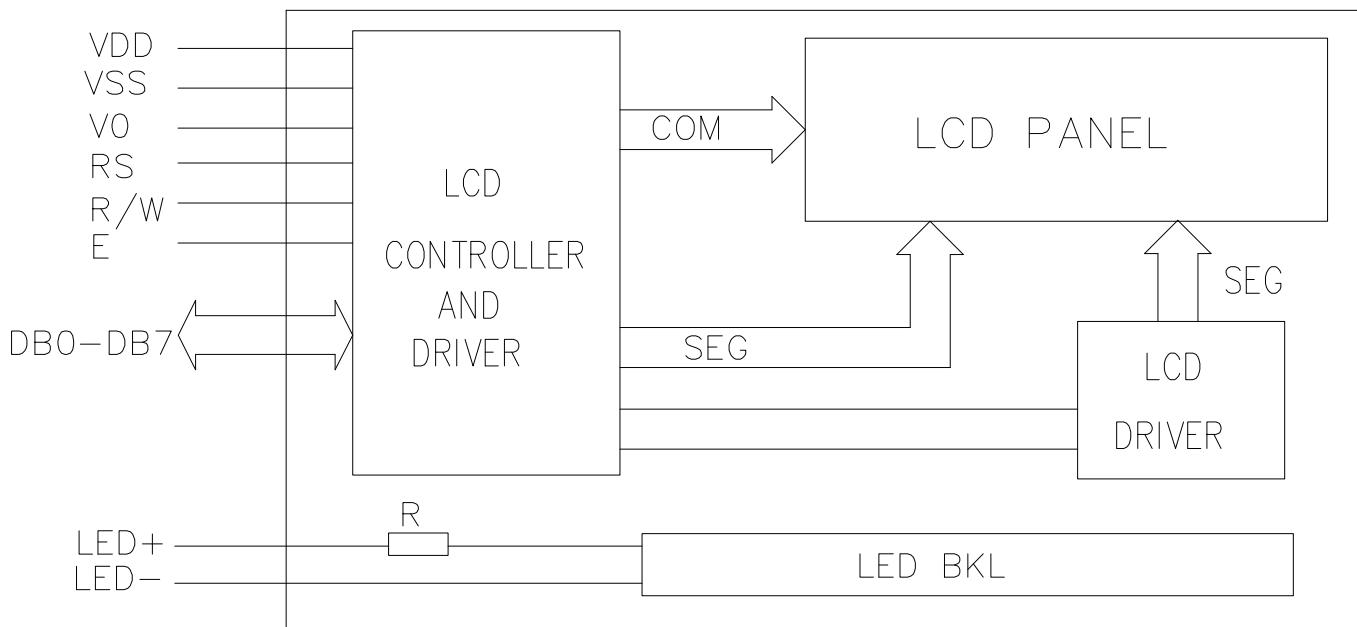
3. Outline dimension



4. Absolute maximum ratings

Item	Symbol	Standard			Unit
Power voltage	$V_{DD}-V_{SS}$	0	-	7.0	V
Input voltage	V_{IN}	V_{SS}	-	V_{DD}	
Operating temperature range	V_{OP}	0	-	+50	°C
Storage temperature range	V_{ST}	-10	-	+60	

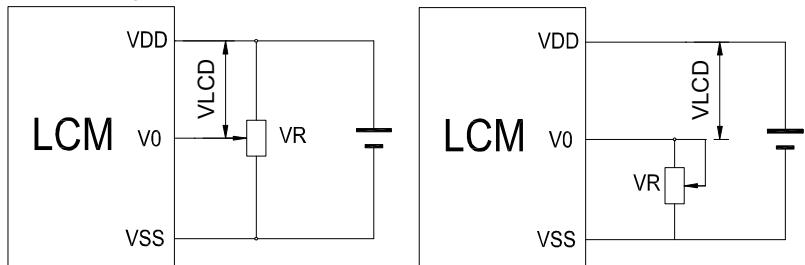
5. Block diagram



6. Interface pin description

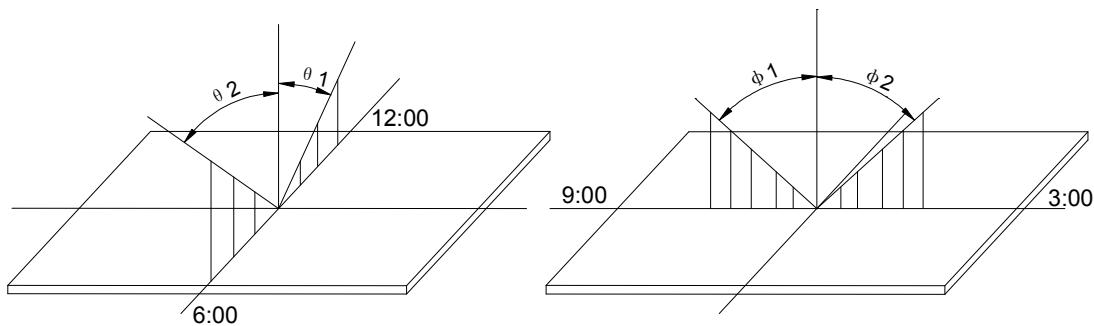
Pin no.	Symbol	External connection	Function
1	V_{SS}	Power supply	Signal ground for LCM
2	V_{DD}		Power supply for logic for LCM
3	V_0		Contrast adjust
4	RS	MPU	Register select signal
5	R/W	MPU	Read/write select signal
6	E	MPU	Operation (data read/write) enable signal
7~10	DB0~DB3	MPU	Four low order bi-directional three-state data bus lines. Used for data transfer between the MPU and the LCM. These four are not used during 4-bit operation.
11~14	DB4~DB7	MPU	Four high order bi-directional three-state data bus lines. Used for data transfer between the MPU
15	LED+	LED BKL power supply	Power supply for BKL
16	LED-		Power supply for BKL

7. Contrast adjust



$V_{DD} \sim V_0$: LCD Driving voltage VR: 10k~20k

8. Optical characteristics



STN type display module ($T_a=25^\circ\text{C}$, $VDD=3.3\text{V}$)

Item	Symbol	Condition	Min.	Typ.	Max.	Unit
Viewing angle	$\theta 1$	$C_r \geq 3$		20		deg
	$\theta 2$			40		
	$\Phi 1$			35		
	$\Phi 2$			35		
Contrast ratio	C_r		-	10	-	-
Response time (rise)	T_r	-	-	200	250	ms
Response time (fall)	T_f	-	-	300	350	

9. Electrical characteristics

DC characteristics

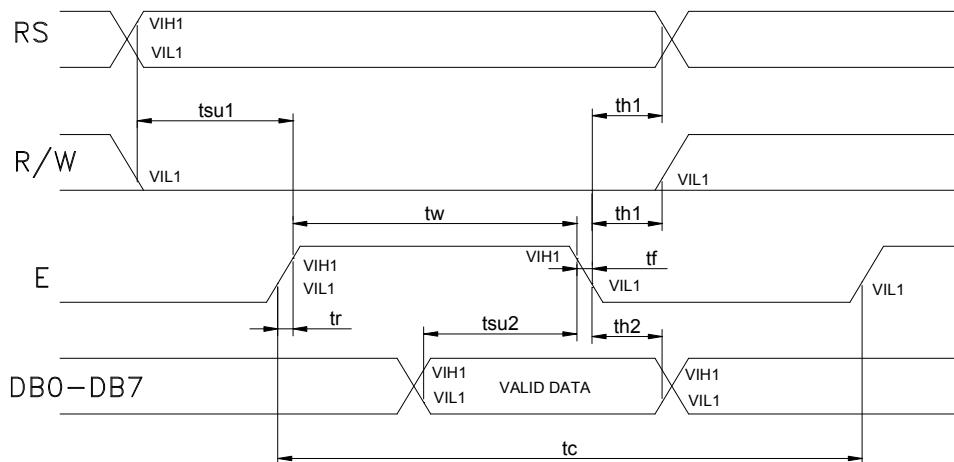
Parameter	Symbol	Conditions	Min.	Typ.	Max.	Unit
Supply voltage for LCD	$V_{DD}-V_0$	$T_a = 25^\circ\text{C}$	-	3.0	-	V
Input voltage	V_{DD}		3.1	3.3	3.5	
Supply current	I_{DD}	$T_a=25^\circ\text{C}$, $V_{DD}=3.3\text{V}$	-	1.5	2.5	mA
Input leakage current	I_{LKG}		-	-	1.0	uA
"H" level input voltage	V_{IH}		2.2	-	V_{DD}	V
"L" level input voltage	V_{IL}	Twice initial value or less	0	-	0.6	
"H" level output voltage	V_{OH}	$LOH=-0.25\text{mA}$	2.4	-	-	
"L" level output voltage	V_{OL}	$LOH=1.6\text{mA}$	-	-	0.4	
Backlight supply voltage	V_F		-	3.0		
Backlight supply current	I_{LED}	$V_{LED}=3.3\text{ V}$ $R=25\Omega$			16	mA

10. Timing Characteristics

Write cycle ($T_a=25^\circ C$, $VDD=3.3V$)

Parameter	Symbol	Test pin	Min.	Typ.	Max.	Unit
Enable cycle time	t_c	E	500	-	-	ns
Enable pulse width	t_w		300	-	-	
Enable rise/fall time	t_r, t_f		-	-	25	
RS; R/W setup time	t_{su1}		100	-	-	
RS; R/W address hold time	t_h1		10	-	-	
Read data output delay	t_{su2}		60	-	-	
Read data hold time	t_h2		10	-	-	

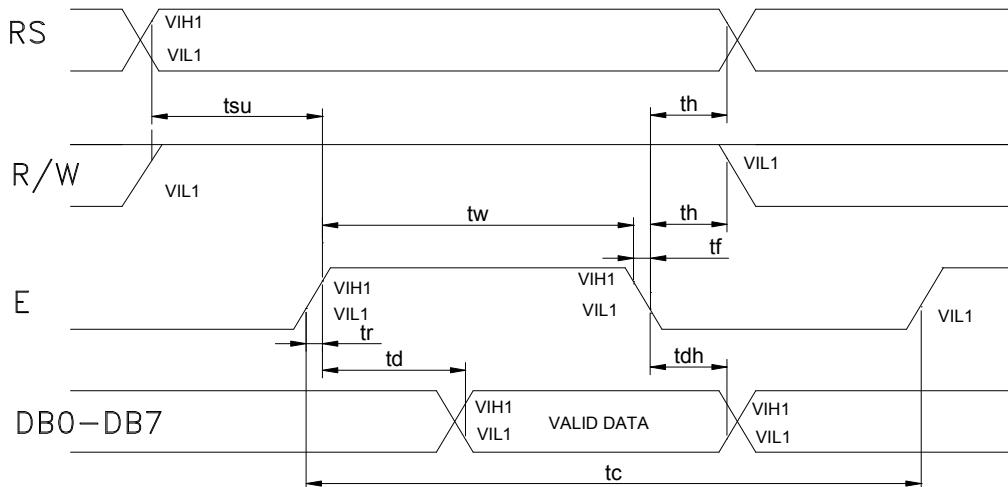
Write mode timing diagram



Read cycle ($T_a=25^\circ C$, $VDD=3.3V$)

Parameter	Symbol	Test pin	Min.	Typ.	Max.	Unit
Enable cycle time	t_c	E	500	-	-	ns
Enable pulse width	t_w		300	-	-	
Enable rise/fall time	t_r, t_f		-	-	25	
RS; R/W setup time	t_{su}		100	-	-	
RS; R/W address hold time	t_h		10	-	-	
Read data output delay	t_d		60	-	90	
Read data hold time	t_{dh}		20	-	-	

Read mode timing diagram



11. FUNCTION DESCRIPTION

11.1 System Interface

This chip has all two kinds of interface type with MPU : 4-bit bus and 8-bit bus. 4-bit bus and 8-bit bus is selected by DL bit in the instruction register.

11.2 Busy Flag (BF)

When BF = "High", it indicates that the internal operation is being processed. So during this time the next instruction cannot be accepted. BF can be read, when RS = Low and R/W = High (Read Instruction Operation), through DB7 port. Before executing the next instruction, be sure that BF is not high.

11.3 Address Counter (AC)

Address Counter (AC) stores DDRAM/CGRAM address, transferred from IR. After writing into (reading from) DDRAM/CGRAM, AC is automatically increased (decreased) by 1. When RS = "Low" and R/W = "High", AC can be read through DB0 – DB6 ports.

11.4 Display Data RAM (DDRAM)

DDRAM stores display data of maximum 80 x 8 bits (80 characters). DDRAM address is set in the address counter (AC) as a hexadecimal number.

Display position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
DDRAM address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
DDRAM address	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

11.5 CGROM (Character Generator ROM)

CGROM has a 5 x 8 dots 204 characters pattern and a 5 x 10 dots 32 characters pattern. CGROM has 204 character patterns of 5 x 8 dots.

11.6 CGRAM (Character Generator RAM)

CGRAM has up to 5 . 8 dot, 8 characters. By writing font data to CGRAM, user defined characters can be used.

Character Code (DDRAM Data)								CGRAM Address						Character Patterns (CGRAM Data)										
b8	b7	b6	b5	b4	b3	b2	b1	b0	b5	b4	b3	b2	b1	b0	b7	b6	b5	b4	b3	b2	b1	b0		
0	0	0	0	0	-	0	0	0	0	0	0	0	0	0	-	-	-	-	-	1	1	1	1	1
						0	0	0				0	0	1						0	0	1	0	0
						0	0	0				0	1	0						0	0	1	0	0
						0	0	0				0	1	1	-	-	-	-	-	0	0	1	0	0
						0	0	0				1	0	0						0	0	1	0	0
						0	0	0				1	0	1						0	0	1	0	0
						0	0	0				1	0	1						0	0	1	0	0
						0	0	0				1	1	0						0	0	1	0	0
						0	0	0				1	1	1						0	0	0	0	0
						0	0	0				1	1	1	-	-	-	-	-	1	1	1	1	0
						0	0	0				0	0	0						1	1	1	1	0
						0	0	0				0	0	1						1	0	0	0	1
						0	0	0				0	1	0						1	0	0	0	1
						0	0	0				0	1	1						1	1	1	1	0
						0	0	0				1	0	0						1	0	1	0	0
						0	0	0				1	0	1						1	0	0	1	0
						0	0	0				1	1	0						1	0	0	0	1
						0	0	0				1	1	1						0	0	0	0	0

Relationship between CGRAM Addresses, Character Codes (DDRAM) and Character patterns (CGRAM Data)

Notes:

1. Character code bits 0 to 2 correspond to CGRAM address bits 3 to 5 (3 bits: 8 types).
2. CGRAM address bits 0 to 2 designate the character pattern line position. The 8th line is the cursor position

and its display is formed by a logical OR with the cursor. Maintain the 8th line data, corresponding to the cursor display position, at 0 as the cursor display. If the 8th line data is 1, 1 bit will light up the 8th line regardless of the cursor presence.

3. Character pattern row positions correspond to CGRAM data bits 0 to 4 (bit 4 being at the left).
4. As shown Table, CGRAM character patterns are selected when character code bits 4 to 7 are all 0. However, since character code bit 3 has no effect, the R display example above can be selected by either character code 00H or 08H.
5. 1 for CGRAM data corresponds to display selection and 0 to non-selection.
“-”: Indicates no effect.

11.7 Cursor/Blink Control Circuit

It controls cursor/blink ON/OFF at cursor position.

11.8 Outline

To overcome the speed difference between the internal clock of ST7066 and the MPU clock, ST7066 performs internal operations by storing control formations to IR or DR. The internal operation is determined according to the signal from MPU, composed of read/write and data bus (Refer to Table7).

Instructions can be divided largely into four groups:

- 1) ST7066 function set instructions (set display methods, set data length, etc.)
- 2) Address set instructions to internal RAM
- 3) Data transfer instructions with internal RAM
- 4) Others

The address of the internal RAM is automatically increased or decreased by 1.

Note: during internal operation, busy flag (DB7) is read “High”.

Busy flag check must be preceded by the next instruction.

11.9 Instruction Table

Instruction	Instruction code										Description	Execution time (fosc= 270 KHZ)
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Write "20H" to DDRA and set DDRAM address to "00H" from AC	1.53ms
Return Home	0	0	0	0	0	0	0	0	1	-	Set DDRAM address to "00H" From AC and return cursor to Its original position if shifted. The contents of DDRAM are not changed.	1.53ms
Entry mode Set	0	0	0	0	0	0	0	1	I/D	SH	Assign cursor moving direction And blinking of entire display	39us
Display ON/OFF control	0	0	0	0	0	0	1	D	C	B	Set display (D), cursor (C), and Blinking of cursor (B) on/off Control bit.	
Cursor or Display shift	0	0	0	0	0	1	S/C	R/L	-	-	Set cursor moving and display Shift control bit, and the Direction, without changing of DDRAM data.	39us
Function set	0	0	0	0	1	DL	N	F	-	-	Set interface data length (DL: 8-Bit/4-bit), numbers of display Line (N: =2-line/1-line) and, Display font type (F: 5x11/5x8)	39us
Set CGRAM Address	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	Set CGRAM address in address Counter.	39us
Set DDRAM Address	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Set DDRAM address in address Counter.	39us
Read busy Flag and Address	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Whether during internal Operation or not can be known By reading BF. The contents of Address counter can also be read.	0us
Write data to Address	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Write data into internal RAM (DDRAM/CGRAM).	43us
Read data From RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Read data from internal RAM (DDRAM/CGRAM).	43us

NOTE:

When an MPU program with checking the busy flag (DB7) is made, it must be necessary 1/2fosc is necessary for executing the next instruction by the falling edge of the "E" signal after the busy flag (DB7) goes to "Low".

11.3Contents

1) Clear display

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	0	0	1

Clear all the display data by writing "20H" (space code) to all DDRAM address, and set DDRAM address to "00H" into AC (address counter).

Return cursor to the original status, namely, bring the cursor to the left edge on the fist line of the display. Make the entry mode increment (I/D="High").

2) Return home

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	0	1	-

Return home is cursor return home instruction.

Set DDRAM address to “00H” into the address counter.

Return cursor to its original site and return display to its original status, if shifted.

Contents of DDRAM does not change.

3) Entry mode set

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	1	I/D	SH

Set the moving direction of cursor and display.

I/D: increment / decrement of DDRAM address (cursor or blink)

When I/D=“high”, cursor/blink moves to right and DDRAM address is increased by 1.

When I/D=“Low”, cursor/blink moves to left and DDRAM address is increased by 1.

*CGRAM operates the same way as DDRAM, when reading from or writing to CGRAM.

SH: shift of entire display

When DDRAM read (CGRAM read/write) operation or SH=“Low”, shifting of entire display is not performed. If SH =“High” and DDRAM write operation, shift of entire display is performed according to I/D value. (I/D=“high”. shift left, I/D=“Low”. Shift right).

4) Display ON/OFF control

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	1	D	C	B

Control display/cursor/blink ON/OFF 1 bit register.

D: Display ON/OFF control bit

When D=“High”, entire display is turned on.

When D=“Low”, display is turned off, but display data remains in DDRAM.

C: cursor ON/OFF control bit

When D=“High”, cursor is turned on.

When D=“Low”, cursor is disappeared in current display, but I/D register preserves its data.

B: Cursor blink ON/OFF control bit

When B=“High”, cursor blink is on, which performs alternately between all the “High” data and display characters at the cursor position.

When B=“Low”, blink is off.

5) Cursor or display shift

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	1	S/C	R/L	-	-

Shifting of right/left cursor position or display without writing or reading of display data.

This instruction is used to correct or search display data.

During 2-line mode display, cursor moves to the 2nd line after the 40th digit of the 1st line.

Note that display shift is performed simultaneously in all the lines.

When display data is shifted repeatedly, each line is shifted individually.

When display shift is performed, the contents of the address counter are not changed.

Shift patterns according to S/C and R/L bits

S/C	R/L	Operation
0	0	Shift cursor to the left, AC is decreased by 1
0	1	Shift cursor to the right, AC is increased by 1
1	0	Shift all the display to the left, cursor moves according to the display
1	1	Shift all the display to the right, cursor moves according to the display

6) Function set

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	DL	N	F	-	-

DL: Interface data length control bit

When DL="High", it means 8-bit bus mode with MPU.

When DL="Low", it means 4-bit bus mode with MPU. Hence, DL is a signal to select 8-bit or 4-bit bus mode.

When 4-bit bus mode, it needs to transfer 4-bit data twice.

N: Display line number control bit

When N="Low", 1-line display mode is set.

When N="High", 2-line display mode is set.

F: Display line number control bit

When F="Low", 5x8 dots format display mode is set.

When F="High", 5x11 dots format display mode.

7) Set CGRAM address

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0

Set CGRAM address to AC.

The instruction makes CGRAM data available from MPU.

8) Set DDRAM address

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0

Set DDRAM address to AC.

This instruction makes DDRAM data available from MPU.

When 1-line display mode (N=LOW), DDRAM address is from "00H" to "4FH". In 2-line display mode (N=High), DDRAM address in the 1st line from "00H" to "27H", and DDRAM address in the 2nd line is from "40H" to "67H".

9) Read busy flag & address

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0

This instruction shows whether SPLC780D is in internal operation or not.

If the resultant BF is "High", internal operation is in progress and should wait BF is to be LOW, which by then the next instruction can be performed. In this instruction you can also read the value of the address counter.

10) Write data to RAM

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
1	0	D7	D6	D5	D4	D3	D2	D1	D0

Write binary 8-bit data to DDRAM/CGRAM.

The selection of RAM from DDRAM, and CGRAM, is set by the previous address set instruction (DDRAM address set, CGRAM address set).

RAM set instruction can also determine the AC direction to RAM.

After write operation. The address is automatically increased/decreased by 1, according to the entry mode.

11) Read data from RAM

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
1	1	D7	D6	D5	D4	D3	D2	D1	D0

Read binary 8-bit data from DDRAM/CGRAM.

The selection of RAM is set by the previous address set instruction. If the address set instruction of RAM is not performed before this instruction, the data that has been read first is invalid, as the direction of AC is not yet determined. If RAM data is read several times without RAM address instructions set before, read operation, the correct RAM data can be obtained from the second. But the first data would be incorrect, as there is no time margin to transfer RAM data.

In case of DDRAM read operation, cursor shift instruction plays the same role as DDRAM address set

instruction, it also transfers RAM data to output data register.

After read operation, address counter is automatically increased/decreased by 1 according to the entry mode.

After CGRAM read operation, display shift may not be executed correctly.

NOTE: In case of RAM write operation, AC is increased/decreased by 1 as in read operation.

At this time, AC indicates next address position, but only the previous data can be read by the read instruction.

12. Standard character pattern

Upper 4bit Lower 4bit	LLLL	LLLH	LLHL	LLHH	LHLL	LHLH	LHHL	LHHH	HLLL	HLLH	HLHL	HLHH	HHLL	HHLH	HHHL	HHHH
LLLL	CG RAM (1)															
LLLH	(2)															
LLHL	(3)															
LLHH	(4)															
LHLL	(5)															
LHLH	(6)															
LHHL	(7)															
LHHH	(8)															
HLLL	(1)															
HLLH	(2)															
HLHL	(3)															
HLHH	(4)															
HHLL	(5)															
HHLH	(6)															
HHHL	(7)															
HHHH	(8)															

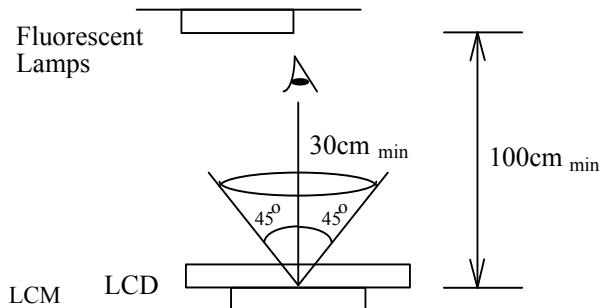
13. QUALITY SPECIFICATIONS

13.1 Standard of the product appearance test

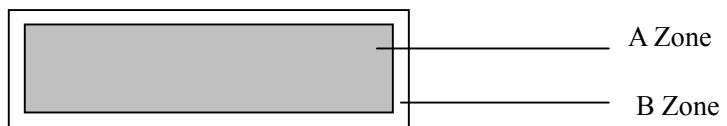
Manner of appearance test: The inspection should be performed in using 20W x 2 fluorescent lamps.

Distance between LCM and fluorescent lamps should be 100 cm or more. Distance between LCM and inspector eyes should be 30 cm or more.

Viewing direction for inspection is 45° from vertical against LCM.



Definition of zone:



A Zone: Active display area (minimum viewing area).

B Zone: Non-active display area (outside viewing area).

13.2 Specification of quality assurance

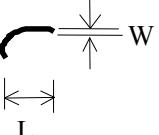
AQL inspection standard

Sampling method: MIL-STD-105E, Level II, single sampling

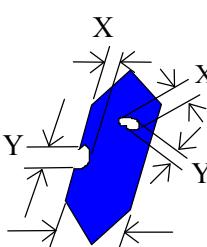
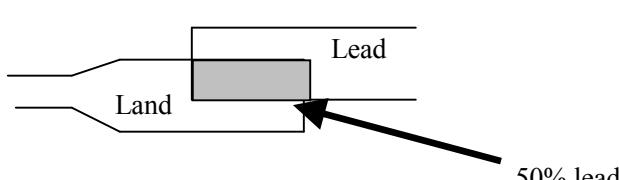
Defect classification **(Note: * is not including)**

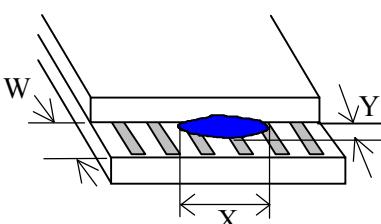
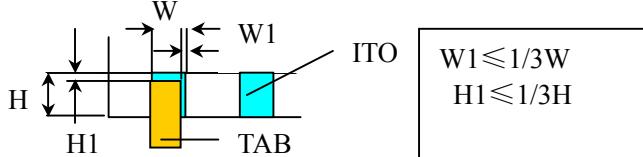
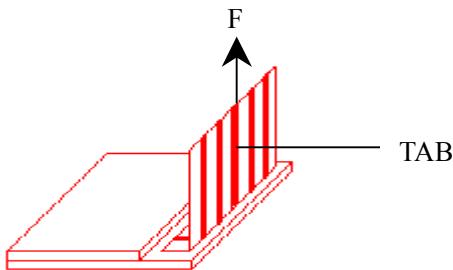
Classify	Item		Note	AQL
Major	Display state	Short or open circuit	1	0.65
		LC leakage		
		Flickering		
		No display		
		Wrong viewing direction		
		Contrast defect (dim, ghost)	2	
	Non-display	Back-light	1,8	
Minor	Display state	Flat cable or pin reverse	10	1.0
		Wrong or missing component	11	
		Background color deviation	2	
		Black spot and dust	3	
		Line defect, Scratch	4	
		Rainbow	5	
		Chip	6	
		Pin hole	7	
	Polarizer	Protruded	12	
		Bubble and foreign material	3	
	Soldering	Poor connection	9	
	Wire	Poor connection	10	
	TAB	Position, Bonding strength	13	

Note on defect classification

No.	Item	Criterion																			
1	Short or open circuit LC leakage Flickering No display Wrong viewing direction Wrong Back-light	Not allow																			
2	Contrast defect Background color deviation	Refer to approval sample																			
3	Point defect, Black spot, dust (including Polarizer) $\phi = (X+Y)/2$	 <table border="1"> <thead> <tr> <th>Point Size</th> <th>Acceptable Qty.</th> </tr> </thead> <tbody> <tr> <td>$\phi \leq 0.10$</td> <td>Disregard</td> </tr> <tr> <td>$0.10 < \phi \leq 0.20$</td> <td>3</td> </tr> <tr> <td>$0.20 < \phi \leq 0.25$</td> <td>2</td> </tr> <tr> <td>$0.25 < \phi \leq 0.30$</td> <td>1</td> </tr> <tr> <td>$\phi > 0.30$</td> <td>0</td> </tr> </tbody> </table> <p style="text-align: right;">Unit: mm</p>	Point Size	Acceptable Qty.	$\phi \leq 0.10$	Disregard	$0.10 < \phi \leq 0.20$	3	$0.20 < \phi \leq 0.25$	2	$0.25 < \phi \leq 0.30$	1	$\phi > 0.30$	0							
Point Size	Acceptable Qty.																				
$\phi \leq 0.10$	Disregard																				
$0.10 < \phi \leq 0.20$	3																				
$0.20 < \phi \leq 0.25$	2																				
$0.25 < \phi \leq 0.30$	1																				
$\phi > 0.30$	0																				
4	Line defect, Scratch	 <table border="1"> <thead> <tr> <th>Line</th> <th>Acceptable Qty.</th> </tr> <tr> <th>L</th> <th>W</th> <th></th> </tr> </thead> <tbody> <tr> <td>---</td> <td>$0.015 \geq W$</td> <td>Disregard</td> </tr> <tr> <td>$3.0 \geq L$</td> <td>$0.03 \geq W$</td> <td rowspan="2">2</td> </tr> <tr> <td>$2.0 \geq L$</td> <td>$0.05 \geq W$</td> </tr> <tr> <td>$1.0 \geq L$</td> <td>$0.1 > W$</td> <td>1</td> </tr> <tr> <td>---</td> <td>$0.05 < W$</td> <td>Applied as point defect</td> </tr> </tbody> </table> <p style="text-align: right;">Unit: mm</p>	Line	Acceptable Qty.	L	W		---	$0.015 \geq W$	Disregard	$3.0 \geq L$	$0.03 \geq W$	2	$2.0 \geq L$	$0.05 \geq W$	$1.0 \geq L$	$0.1 > W$	1	---	$0.05 < W$	Applied as point defect
Line	Acceptable Qty.																				
L	W																				
---	$0.015 \geq W$	Disregard																			
$3.0 \geq L$	$0.03 \geq W$	2																			
$2.0 \geq L$	$0.05 \geq W$																				
$1.0 \geq L$	$0.1 > W$	1																			
---	$0.05 < W$	Applied as point defect																			
5	Rainbow	Not more than two color changes across the viewing area.																			

No	Item	Criterion																																	
6	Chip	<p>Remark:</p> <p>X: Length direction</p> <p>Y: Short direction</p> <p>Z: Thickness direction</p> <p>t: Glass thickness</p> <p>W: Terminal Width</p> <p>Acceptable criterion</p> <table border="1"> <thead> <tr> <th>X</th> <th>Y</th> <th>Z</th> </tr> </thead> <tbody> <tr> <td>≤ 2</td> <td>0.5mm</td> <td>$\leq t/2$</td> </tr> </tbody> </table> <p>Acceptable criterion</p> <table border="1"> <thead> <tr> <th>X</th> <th>Y</th> <th>Z</th> </tr> </thead> <tbody> <tr> <td>≤ 2</td> <td>0.5mm</td> <td>$\leq t$</td> </tr> </tbody> </table> <p>Acceptable criterion</p> <table border="1"> <thead> <tr> <th>X</th> <th>Y</th> <th>Z</th> </tr> </thead> <tbody> <tr> <td>≤ 3</td> <td>≤ 2</td> <td>$\leq t$</td> </tr> <tr> <td colspan="3">shall not reach to ITO</td> </tr> </tbody> </table> <p>Acceptable criterion</p> <table border="1"> <thead> <tr> <th>X</th> <th>Y</th> <th>Z</th> </tr> </thead> <tbody> <tr> <td>Disregard</td> <td>≤ 0.2</td> <td>$\leq t$</td> </tr> </tbody> </table> <p>Acceptable criterion</p> <table border="1"> <thead> <tr> <th>X</th> <th>Y</th> <th>Z</th> </tr> </thead> <tbody> <tr> <td>≤ 5</td> <td>≤ 2</td> <td>$\leq t/3$</td> </tr> </tbody> </table>	X	Y	Z	≤ 2	0.5mm	$\leq t/2$	X	Y	Z	≤ 2	0.5mm	$\leq t$	X	Y	Z	≤ 3	≤ 2	$\leq t$	shall not reach to ITO			X	Y	Z	Disregard	≤ 0.2	$\leq t$	X	Y	Z	≤ 5	≤ 2	$\leq t/3$
X	Y	Z																																	
≤ 2	0.5mm	$\leq t/2$																																	
X	Y	Z																																	
≤ 2	0.5mm	$\leq t$																																	
X	Y	Z																																	
≤ 3	≤ 2	$\leq t$																																	
shall not reach to ITO																																			
X	Y	Z																																	
Disregard	≤ 0.2	$\leq t$																																	
X	Y	Z																																	
≤ 5	≤ 2	$\leq t/3$																																	

No.	Item	Criterion								
7	Segment pattern W = Segment width $\phi = (X+Y)/2$	(1) Pin hole $\phi < 0.10\text{mm}$ is acceptable.  <table border="1" data-bbox="841 415 1301 595"> <thead> <tr> <th>Point Size</th><th>Acceptable Qty</th></tr> </thead> <tbody> <tr> <td>$\phi \leq 1/4W$</td><td>Disregard</td></tr> <tr> <td>$1/4W < \phi \leq 1/2W$</td><td>1</td></tr> <tr> <td>$\phi > 1/2W$</td><td>0</td></tr> </tbody> </table> <p style="text-align: right;">Unit: mm</p>	Point Size	Acceptable Qty	$\phi \leq 1/4W$	Disregard	$1/4W < \phi \leq 1/2W$	1	$\phi > 1/2W$	0
Point Size	Acceptable Qty									
$\phi \leq 1/4W$	Disregard									
$1/4W < \phi \leq 1/2W$	1									
$\phi > 1/2W$	0									
8	Back-light	(1) The color of backlight should correspond its specification. (2) Not allow flickering								
9	Soldering	(1) Not allow heavy dirty and solder ball on PCB. (The size of dirty refer to point and dust defect) (2) Over 50% of lead should be soldered on Land. 								
10	Wire	(1) Copper wire should not be rusted (2) Not allow crack on copper wire connection. (3) Not allow reversing the position of the flat cable. (4) Not allow exposed copper wire inside the flat cable.								
11*	PCB	(1) Not allow screw rust or damage. (2) Not allow missing or wrong putting of component.								

No	Item	Criterion
12	Protruded W: Terminal Width	 <p>Acceptable criteria: $Y \leq 0.4$</p>
13	TAB	<p>1. Position</p>  <p>2 TAB bonding strength test</p>  <p>$P (=F/\text{TAB bonding width}) \geq 650\text{gf/cm}$, (speed rate: 1mm/min) 5pcs per SOA (shipment)</p>
14	Total no. of acceptable Defect	<p>A. Zone</p> <p>Maximum 2 minor non-conformities per one unit. Defect distance: each point to be separated over 10mm</p> <p>B. Zone</p> <p>It is acceptable when it is no trouble for quality and assembly in customer's end product.</p>

13.3 Reliability of LCM

Reliability test condition:

Item	Condition	Time (hrs)	Assessment
High temp. Storage	80°C	48	No abnormalities in functions and appearance
High temp. Operating	70°C	48	
Low temp. Storage	-30°C	48	
Low temp. Operating	-20°C	48	
Humidity	40°C/ 90%RH	48	
Temp. Cycle	0°C ← 25°C → 50°C (30 min ← 5 min → 30min)	10cycles	

Recovery time should be 24 hours minimum. Moreover, functions, performance and appearance shall be free from remarkable deterioration within 50,000 hours under ordinary operating and storage conditions room temperature ($20\pm8^\circ\text{C}$), normal humidity (below 65% RH), and in the area not exposed to direct sun light.

13.4 Precaution for using LCD/LCM

LCD/LCM is assembled and adjusted with a high degree of precision. Do not attempt to make any alteration or modification. The followings should be noted.

General Precautions:

1. LCD panel is made of glass. Avoid excessive mechanical shock or applying strong pressure onto the surface of display area.
2. The polarizer used on the display surface is easily scratched and damaged. Extreme care should be taken when handling. To clean dust or dirt off the display surface, wipe gently with cotton, or other soft material soaked with isopropyl alcohol, ethyl alcohol or trichlorotrifluoroethane, do not use water, ketone or aromatics and never scrub hard.
3. Do not tamper in any way with the tabs on the metal frame.
4. Do not make any modification on the PCB without consulting AMOTEC
5. When mounting a LCM, make sure that the PCB is not under any stress such as bending or twisting. Elastomer contacts are very delicate and missing pixels could result from slight dislocation of any of the elements.
6. Avoid pressing on the metal bezel, otherwise the elastomer connector could be deformed and lose contact, resulting in missing pixels and also cause rainbow on the display.
7. Be careful not to touch or swallow liquid crystal that might leak from a damaged cell. Any liquid crystal adheres to skin or clothes, wash it off immediately with soap and water.

Static Electricity Precautions:

1. CMOS LSI is used for the module circuit; therefore operators should be grounded whenever he/she comes into contact with the module.
2. Do not touch any of the conductive parts such as the LSI pads; the copper leads on the PCB and the interface terminals with any parts of the human body.

-
- 3. Do not touch the connection terminals of the display with bare hand; it will cause disconnection or defective insulation of terminals.
 - 4. The modules should be kept in anti-static bags or other containers resistant to static for storage.
 - 5. Only properly grounded soldering irons should be used.
 - 6. If an electric screwdriver is used, it should be grounded and shielded to prevent sparks.
 - 7. The normal static prevention measures should be observed for work clothes and working benches.
 - 8. Since dry air is inductive to static, a relative humidity of 50-60% is recommended.

Soldering Precautions:

- 1. Soldering should be performed only on the I/O terminals.
- 2. Use soldering irons with proper grounding and no leakage.
- 3. Soldering temperature: $280^{\circ}\text{C} \pm 10^{\circ}\text{C}$
- 4. Soldering time: 3 to 4 second.
- 5. Use eutectic solder with resin flux filling.
- 6. If flux is used, the LCD surface should be protected to avoid spattering flux.
- 7. Flux residue should be removed.

Operation Precautions:

- 1. The viewing angle can be adjusted by varying the LCD driving voltage V_o .
- 2. Since applied DC voltage causes electro-chemical reactions, which deteriorate the display, the applied pulse waveform should be a symmetric waveform such that no DC component remains. Be sure to use the specified operating voltage.
- 3. Driving voltage should be kept within specified range; excess voltage will shorten display life.
- 4. Response time increases with decrease in temperature.
- 5. Display color may be affected at temperatures above its operational range.
- 6. Keep the temperature within the specified range usage and storage. Excessive temperature and humidity could cause polarization degradation, polarizer peel-off or generate bubbles.
- 7. For long-term storage over 40°C is required, the relative humidity should be kept below 60%, and avoid direct sunlight.

Limited Warranty

AMOTEC LCDs and modules are not consumer products, but may be incorporated by AMOTEC's customers into consumer products or components thereof. AMOTEC does not warrant that its LCDs and components are fit for any such particular purpose.

- 1. The liability of AMOTEC is limited to repair or replacement on the terms set forth below. AMOTEC will not be responsible for any subsequent or consequential events or injury or damage to any personnel or user including third party personnel and/or user. Unless otherwise agreed in writing between AMOTEC and the customer, AMOTEC will only replace or repair any of its LCD which is found defective electrically or visually when inspected in accordance with AMOTEC general LCD inspection standard. (Copies available on request)
- 2. No warranty can be granted if any of the precautions state in handling liquid crystal display above has been disregarded. Broken glass, scratches on polarizer mechanical damages as well as defects that are caused accelerated environment tests are excluded from warranty.
- 3. In returning the LCD/LCM, they must be properly packaged; there should be detailed description of the failures or defect.