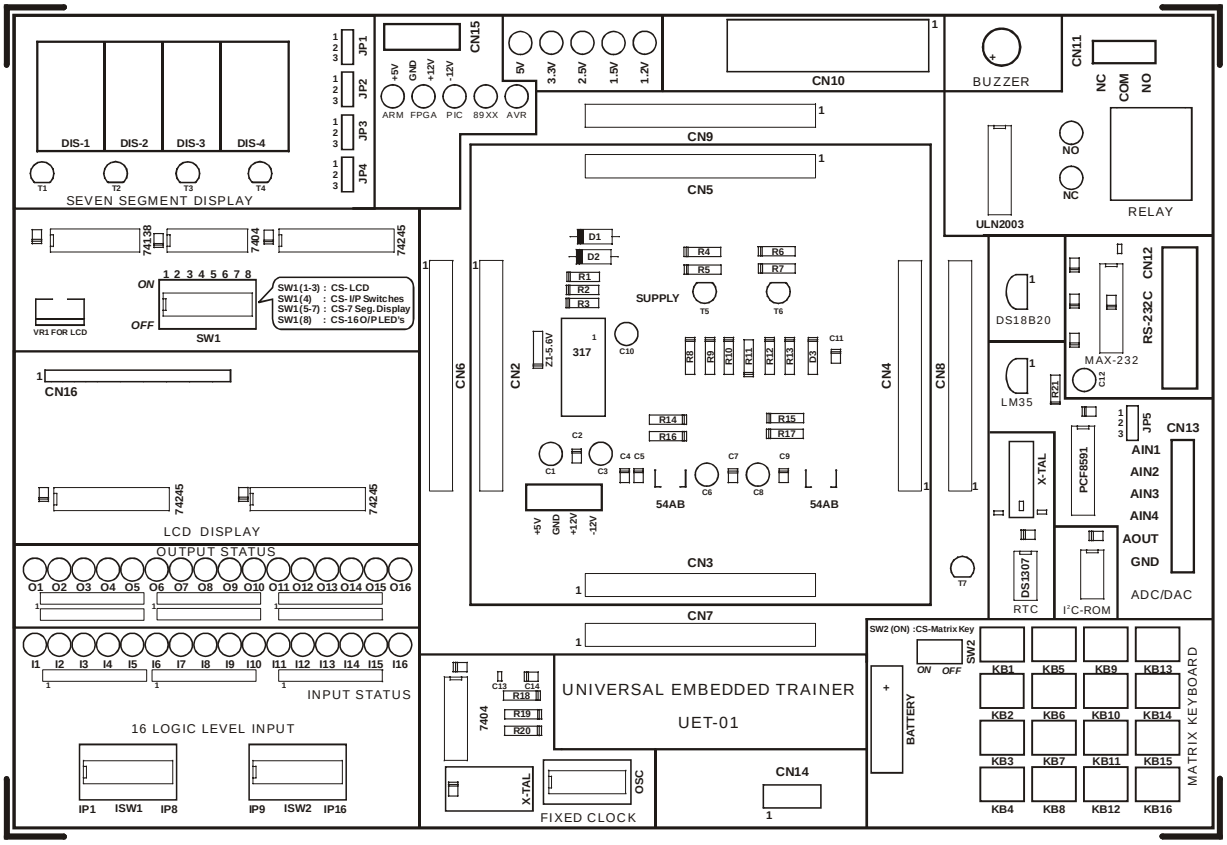
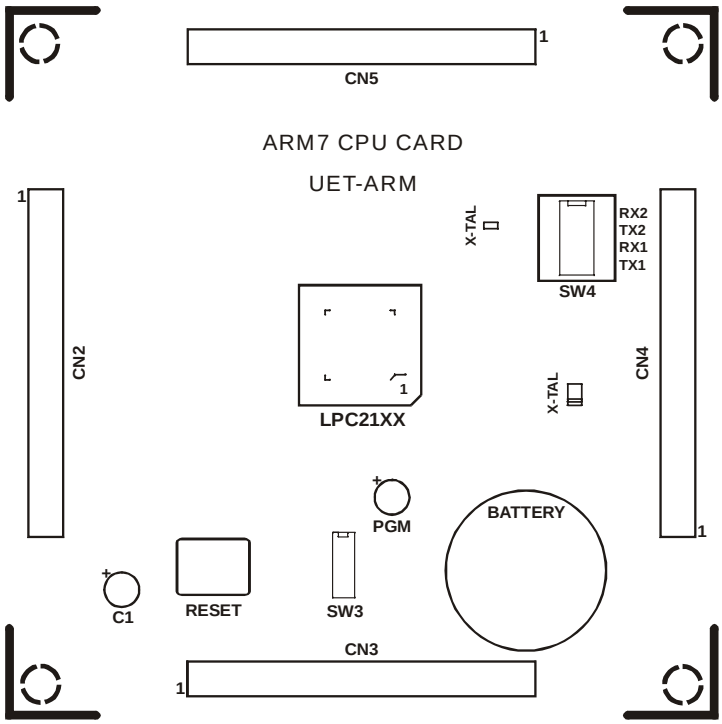
SPECIFICATION

ARM 7 EMBEDDED TRAINER

- CPU: Philips LPC2148 operating @ 12MHz.
- 16 bit / 32bit ARM7 TDMI-S Microcontroller.
- On-chip 512KB Flash memory and 40KB RAM.
- On-chip 14 Ch. 10 bit ADC and 1 Ch. 10bit DAC.
- On-chip Real Time Clock, UART, SPI, SSP, I2C, PWM
- On-chip 48 I/O Lines are provided in 40 pin Conn.
  - ISP Programming facility
  - On-board Reset Key
- Onboard Applications
- 16 LED's to display Digital Output.
- 16 Switches to give Digital Input indicated by LED's.
- 16x2 Alphanumeric LCD backlit display.
- 4x4 Matrix Keyboard.
- Miniature Buzzer.
- 12V SPDT Relay.
- 4 digit seven segment displays.
  - I^2C compatible:
    - 24C512 EEPROM (64KB)
    - DS1307 RTC with suitable battery
    - 4Ch. 8bit ADC & 1Ch. 8 bit DAC using PCF8591
  - Temperature sensor interface using LM35.
  - Temperature sensor interface using DS18B20.
  - 24 I/O Lines Provided on a 26 pin FRC Connector for external interface.
- On board supply +/- 12V, 5V, 3.3V, 2.5V, 1.5V & 1.2V is provided.
- Supply Input Voltage: 230V AC.
- All ICS are mounted on IC Sockets.
- Bare board Tested Glass Epoxy SMOBC PCB is used.
- Attractive Wooden enclosures of Light weight Australian Pine Wood.
- User's Manual with `C' source code sample programs for all on board applications

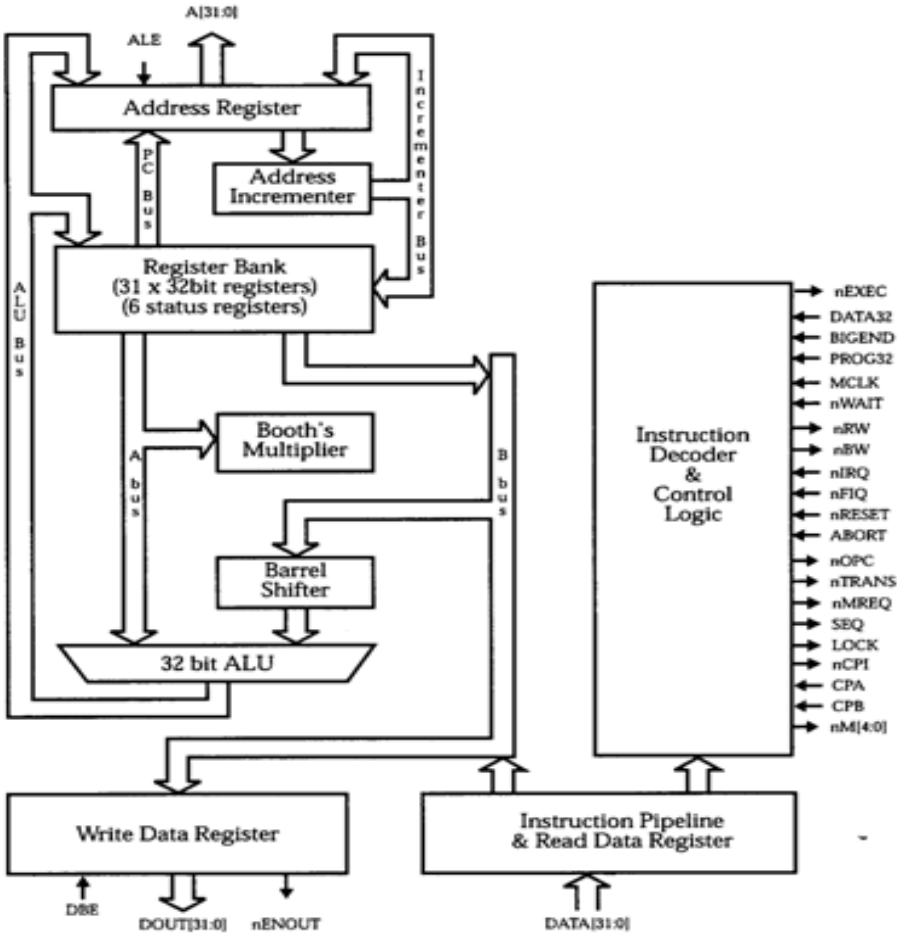
BLOCK DESCRIPTION





INTRODUCTION

Advanced RISC Machine (ARM) was developed by Acorn Company. ARM is intended for use of applications, which require power efficient processors, such as Telecommunications, Data Communication (protocol converter), Portable Instrument, Portable Computer and Smart Card. ARM is basically a 32-bit RISC processor (32-bit data bus and address bus) with fast interrupt response for use in real time applications. A block diagram of ARM7 processor is shown in figure.



Block diagram of ARM7 architecture.

Instruction Decoder and Logic Control: The function of instruction decoder and logic control is to decode instructions and generate control signals to other parts of processor for execution of instructions.

Address Register: To hold a 32-bit address for address bus.

Address Increment: It is used to increment an address by four and place it in address register.

Register Bank: Register bank contains thirty one 32-bit registers and six status registers.

Barrel Shifter: it is used for fast shift operation.

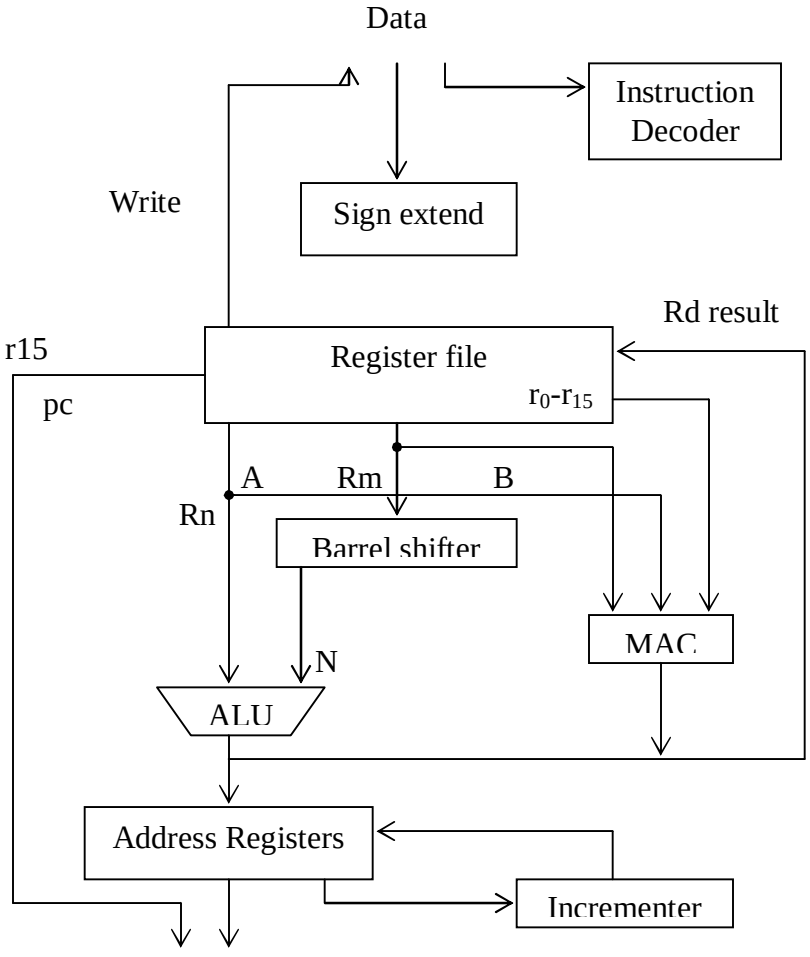
ALU: 32-bit ALU is used for Arithmetic and Logic Operation.

Write Data Register: The processor put the data in Write Data Register for write operation.

Read Data Register: When processor reads from memory it places the result in this register.

ARM CORE INTRODUCTION:

The ARM Processor, like all RISC processors, uses load store architecture. This means that it has two instruction types for transferring data in and out of the processor. Load instructions copy data from memory to registers in the core and the store instructions copy data from registers to memory. There are n o data processing instructions that directly manipulate data in memory. The data processing is carried out only in registers.



ARM Core data flow model

Data items are placed in the register file. The register file is a storage bank made up of 32 bit registers. The registers hold 32 bit signed or unsigned numbers. The sign extend converts 8 bit or 16 bit numbers to 32 bit values as they are read from memory and placed in register.

ARM instructions typically have two source registers Rn and Rm, and result register Rd. Source operands are read from the register file via internal buses A & B.

The ALU or MAC takes the register values Rn and Rm from the A and B buses and compute result. Data processing instructions write the result in Rd. Load and store instructions use the ALU to generate an address.

Rm can be preprocessed in the barrel shifter before it enters ALU. The Rd is written back to the register file using result bus.

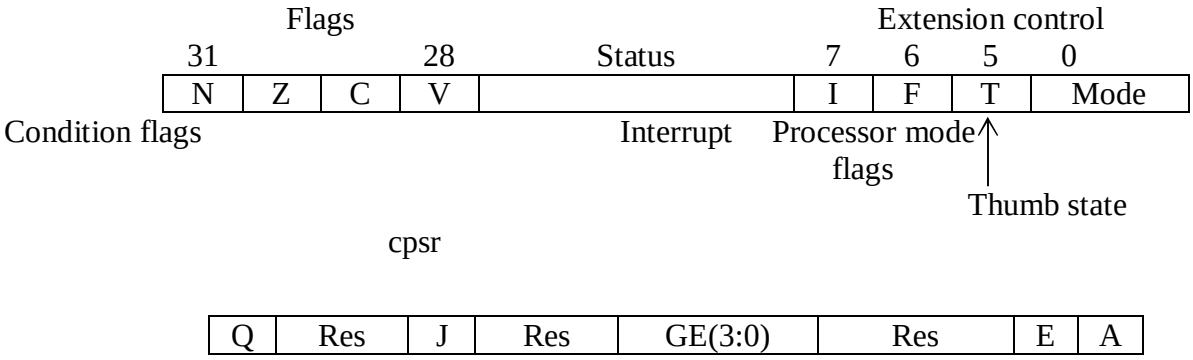
For load, store instructions the incrementer updates the address register before the core reads or writes next register or memory.

The processor continues executing instructions until an exception or interrupt changes the normal flow.

ARM Registers

General purpose registers hold either data or an address. The active registers available in user mode are r0-r15 and cpsr. spsr is also available in other than user mode.

r0-r12 are 32 bit general purpose registers.
r13-r15 are special purpose 32 bit registers.
r13 – stack pointer which holds top of stack address
r14 – Link register where the core puts return address during call.
r15 – program counter contains the address of next instruction to be fetched.
cpsr – Current program status register which is used by the ARM core for monitoring and control internal operations. The fields of cpsr are flag, status, extension and control.



Q	Res	J	Res	GE(3:0)	Res	E	A

Q – saturation flag (saturated signed & unsigned arithmetic)

J = 1 Java execution

Res = reserved

GE(3:0) – Greater or equal flags (parallel modulo add & subtract)

E Controls the data endiness

A = 1 disables imprecise data aborts

I = 1 disables IRQ interrupts

F = 1 disables FIQ interrupts

T = 1 Thumb state ; = 0 ARM state.

Mode	10000	user mode
	10001	FIQ mode
	10010	IRQ mode
	10011	supervisor mode
	10111	abort mode
	11011	undefined mode
	11111	system mode

Except user mode all others are privileged modes. Processor exerts abort mode when there is a failed attempt to access memory.

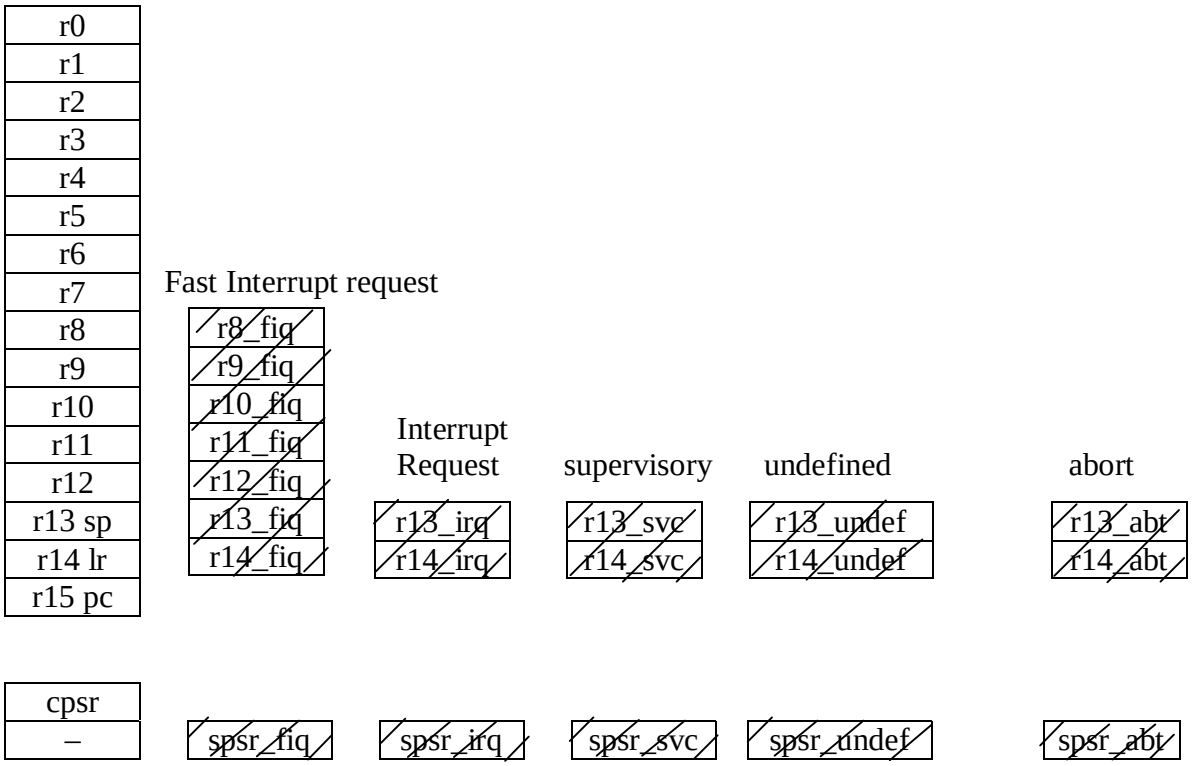
Fast interrupt request and Interrupt request modes correspond to two interrupt levels of ARM processor.

Supervisor mode is the mode that the processor is in after reset and OS Kernel operates in.

System mode is special version of user mode that allows full read-write access to the cpsr.

undefined mode is used when processor encounters an instruction that is undefined.

Actually there are 37 registers in register file. 20 registers are hidden from a program at different times. These registers are called banked registers, are identified by shading in the diagram.



A banked register maps one to one onto a user mode register. If we change processor mode, a banked register from the new mode will replace an existing register.

The saved program status register (spsr) stores the previous mode's cpsr. cpsr is not copied into spsr when a mode change is forced due to a program writing directly to cpsr.

The state of the core determines which instruction set is being executed.

1. ARM 2. Thumb 3. Jazelle

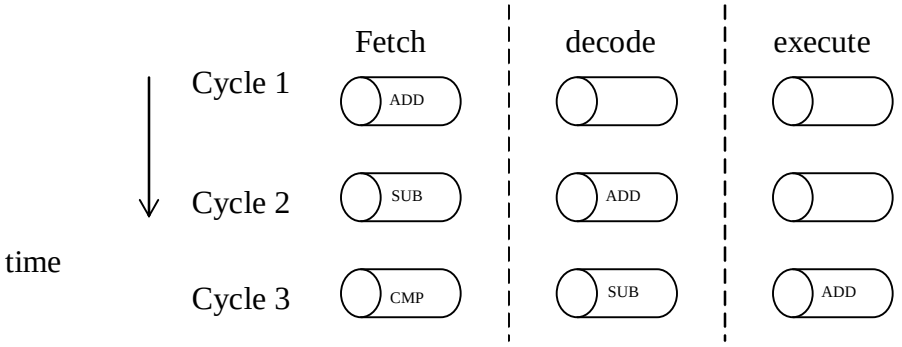
We cannot intermingle sequential ARM, Thumb and Jazelle instructions.

ARM 32 bit
Thumb 16 bit
Jazelle 8 bit and is hybrid mix of software and hardware designed to speed up the execution of Java byte codes.

Pipe line :

Using a pipe line speeds up execution by fetching the next instruction while other instructions are being decoded and executed. ARM7 has 3 stages pipe line.

Example: Instruction 1 ADD
Instruction 2 SUB
Instruction 3 CMP



In the first cycle, the core fetches ADD instruction from memory. In the second cycle the core fetches SUB instruction and decodes ADD instruction. In the third cycle the ARM core fetches CMP instruction, decode SUB instruction and execute ADD instruction.

ARM9 has five stage pipe line including memory write and ARM10 has 6 stage pipe line.

The ARM pipe line has not processed an instruction until passes completely through the execute stage. For example, ARM7 has executed an instruction only when the fourth instruction is fetched.

The program counter always points to the address of the instruction being executed plus two instructions ahead. This is important when the pc is used for calculating a relative offset. When the processor is in thumb state the pc is the instruction address plus 4.

Characteristics of pipe line :

1. The execution of a branch instruction or branching by direct modification of pc causes the ARM core to flush its pipe line.
2. An instruction in execute stage will complete even though an interrupt has been raised. Other instructions in the pipe line will be abandoned, the processor start filling the pipe line from the appropriate entry in the vector table.
3. ARM10 uses branch prediction which reduces the effect of pipe line flush.

Exceptions, Interrupts and Vector Table

When an exception (unplanned changes occur in flow of control) or interrupt (immediate program needed apart from regular program sequence) occurs, the processor sets the pc to a specified memory address. The address in with in special address range called Vector table. The entries in the Vector table are instructions that branches to specific routines designed to handle a particular exception or interrupt.

When an exception or interrupt occurs the processor suspends normal execution and starts loading instructions from the exception vector table.

The memory map address 0 x 00000000is reserved for vector table. On some processors the vector table can be optionally located at higher address 0 x FFFF 0000. Operating systems such as Linux and MS embedded products can take advantage of this feature.

1. reset vector : Is the location of first instruction executed by processor when power is supplied to processor. This instruction branches to the initialization code.
2. Undefined instruction vector : Used when the processor can not decode a instruction.
3. Software interrupt vector : Called when SWI instruction is executed.
4. Prefetch abort vector : occurs when the processor attempts to fetch an instruction from address without access permissions.
5. Data abort vector : Occurs when the processor attempts to access an data (from data memory) without access permission.
6. Interrupt request vector : Used by external hardware to interrupt normal execution. (IRQ are not masked)
7. Fast Interrupt request vector : FIQ are not masked

Vector Table

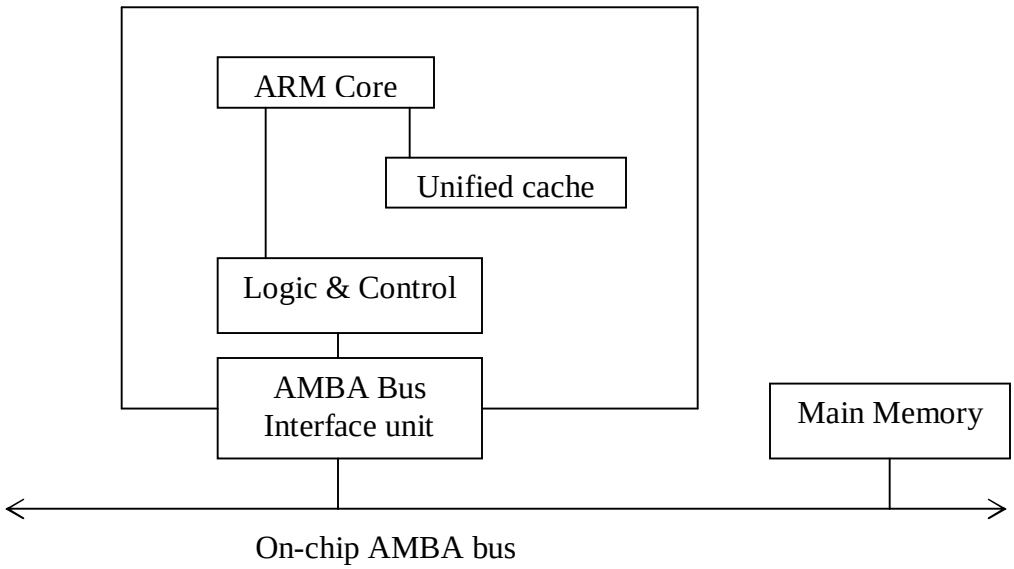
Exception / Interrupt	Address
Reset (RESET)	0 x 0000 0000
Undefined Instruction (UNDEF)	0 x 0000 0004
Software interrupt (SWI)	0 x 0000 0008
Prefetch abort (PABT)	0 x 0000 000C
Data abort (DABT)	0 x 0000 0010
Reserved	0 x 0000 0014
Interrupt request (IRQ)	0 x 0000 0018
Fast interrupt request (FIQ)	0 x 0000 001C

Core Extensions

There are three hardware extensions ARM wraps around the core, cache and tightly coupled Memory, Memory Management and the Coprocessor interface.

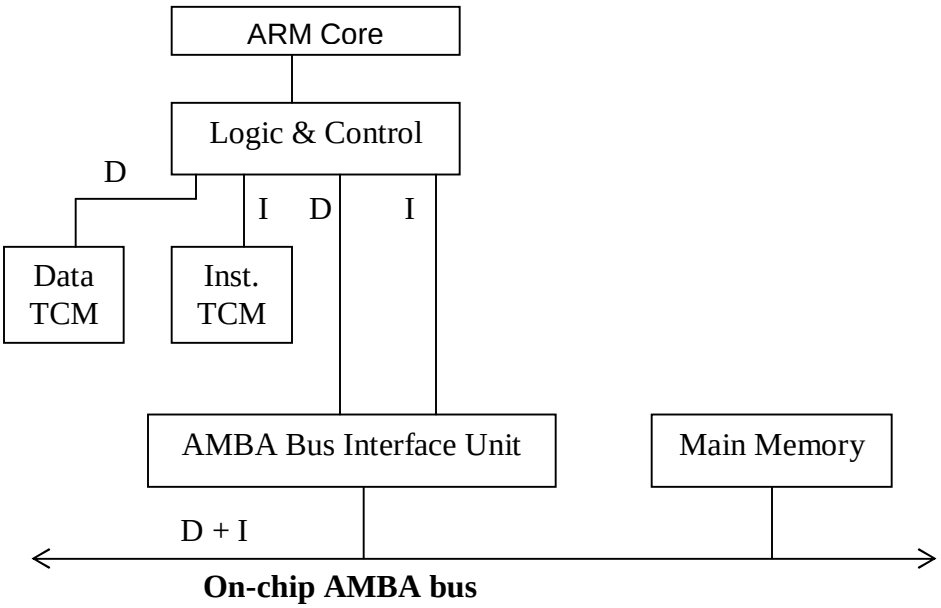
1. Cache and Tightly couple Memory

Cache is the fast memory placed in between main memory and core which speed up fetching. Von Neumann architecture has unified cache for both data and instruction.



Harvard style core provides separate cache for data and instruction.

A cache provides an overall increase in performance but at the expense of predictable execution. but for real time systems it is paramount that code execution is deterministic-time taken for load, store data or instruction must be predictable. This is fast SRAM located close to the core and guarantees the clock required to fetch instruction or data.



Memory Management :

Embedded systems use multiple memory devices. Memory management hardware helps to organize these devices and protection.

ARM cores have three different types of memory management hardware 1) non protected memory, Memory protection unit (MPU) and Memory Management unit (MMU) full protection.

- Non protected memory is fixed and provides little flexibility used for small embedded systems.
- MPU employs a simple system that uses limited number of memory regions. These regions are controlled with a set of special coprocessor registers and each region is defined with specific access permissions.
- MMU uses a set of translation tables to provide fine grained control over memory. These tables are stored in main memory and provide a Virtual to physical address map as well as access permissions.

Coprocessors :

Coprocessor extends the processing features of a core by extending the instruction set or by providing configuration registers. More than one coprocessor can be added to the ARM core via the coprocessor interface.

The coprocessor can be accessed through dedicated ARM instructions.

The coprocessor can extend ARM instructions by providing instructions for special operations. Ex: Vector Floating Point.

Architecture Revisions :

Every ARM implementation executes a specific Instruction Set Architecture (ISA). The code written to earlier architecture revision will also execute on as latter version of architecture.

The ARM nomenclature indicates the feature of the processor and not includes architecture revision information.

Nomenclature

ARM[x] [y] [z] [T] [D] [M] [I] [E] [J] [F] [S]

X – family**;** y – memory management / protection unit**;** z – cache

T – thumb 16 bit decoder

D – JTAG debug

M – Fast Multiplier

I – Embedded JCE Macro cell

E – enhanced instructions (assume TDMI)

J – Jazelle

F – Vector floating point unit

S – Synthesizable version

All ARM cores after ARM7TDMI include the TDMI features even though not include in letters.

Family – group of processor that shares the same hardware characteristics.

JTAG – IEE1149.1 Standard Test Access Port and boundary scan Architecture (developed by Joint Test Action Group)

Embedded ICE macro cell – debut hardware built into the processor that allows Breakpoints and watch points to be set.

Synthesizable – processor core is supplied as source code that can be compiled into a form easily used by EDA tools.

Architecture Revision

Revision	Example core	ISA Enhancement
ARMv1	ARM 1	first arm ; 26 bit addressing
ARMv2	ARM 2	32 bit multiplier ; 32 bit coprocessor support
ARMv2a	ARM 3	on chip cache ; SWAP Instruction coprocessor 15 for cache Management
ARMv3	ARM 6 & ARM 7DI	32 bit addressing ; cpsr & spsr undefined instruction and abort MMU support – Virtual memory
ARMv3M	ARM 7M	signed & unsigned long multiple instructions
ARMv4	Strong ARM	Load / store instructions for signed / unsigned half words / bytes new mode – system reserve SWI space for architecturally defined operations 26 bit addressing mode no longer supported
ARMv4T	ARM 7 TDMI & ARM 97	Thumb
ARMv5TE	ARM 9E and ARM 10E	superset of ARMv4T Extra instructions added to change state between thumb & ARM Enhanced multiple instruction Extra DSP type instructions Faster Multiply accumulate
ARMv5 TEJ	ARM 7 EJ, ARM 926EJ	Java acceleration
ARMv6	ARM 11	improved multiprocess instructions unaligned and mixed endian data handling new multimedia instructions

ARM processor families :

ARM has designed a number of processors that are grouped into different families according to the core they use. The families are based on ARM 7, ARM 9, ARM 10 and ARM 11 cores. ARM 8 was developed but was soon superseded.

ARM family attribute comparison

	ARM 7	ARM 9	ARM 10	ARM 11
Pipe line depth	3	5	6	8
Typical MHz	80	150	260	335
mW / MHz	0.06	0.19	0.5	0.4
MIPS / MHz	0.97	1.1	1.3	1.2
Architecture	Von Neumann	Harvard	Harvard	Harvard
Multiplier	8 x 32	8 x 32	16 x 32	16 x 32

ARM Processor Variants

CPU Core	MMU / MPU	Cache	Jazelle	Thumb	E
ARM 7 TDMI	none	none	no	yes	no
ARM 7 EJ-S	none	none	yes	yes	yes
ARM 7 20T	MMU	undefined 8K	no	yes	No
ARM 9207	MMU	16K / 16K	no	yes	no
ARM 992T	MMU	8K / 8K	no	yes	no
ARM 926EJ-S	MMU	Cache & TCM	yes	yes	yes
ARM 940T	MPU	4K / 4K	no	yes	no
ARM 946 E-S	MPU	Cache & TCM	no	yes	yes
ARM 966 E-S	none	TCM	no	yes	Yes
ARM 1020E	MMU	32K / 32K	no	yes	yes
ARM 1022 E	MMU	16K / 16K	no	yes	yes
ARMM 1026 EJ-S	MMU & MPU	Cache & TCM	yes	yes	Yes
ARM 1136 J-S	MMU	Cache & TCM	yes	yes	yes
ARM 1136 JFS	MMU	Cache & TCM	yes	yes	yes

ARM Instruction Set

ARM instructions are classified as**:**

1) Data processing instructions
2) Branch instructions (control flow instructions)
3) Load – store instructions (data transfer instructions)

in addition
software interrupt instructions and
program status register instructions.

Data Processing instructions:
Enable the programmer to perform arithmetic and logical operations on data in registers. Generally, these instructions require two operands and produce a single result.

***** Rules apply to data processing instructions**:**

- All operands are 32 bit wide and come from registers or are specified as literals in the instruction itself.
- Each of the operand registers and the result register are independently specified in the instruction.

Arithmetic Operations :

```
ADD   r0, r1, r2      ;     r0 = r1 + r2
ADC   r0, r1, r2      ;     r0 = r1 + r2 + c
SUB   r0, r1, r2      ;     r0 = r1 – r2
SBC   r0, r1, r2      ;     r0 = r1 – r2 + c – 1
RSB   r0, r1, r2      ;     r0 = r2 – r1
RSC   r0, r1, r2      ;     r0 = r2 – r1 + c – 1
```

Bit Wise Logical Operations

```
AND   r0, r1, r2      ;     r0 = r1 AND r2
ORR   r0, r1, r2      ;     r0 = r1 OR r2
EOR   r0, r1, r2      ;     r0 = r1 XOR r2
BIC   r0, r1, r2      ;     r0 = r1 AND NOT r2
```

Register movement operations

```
MOV   r0, r2          ;     r0 = r2
MVN   r0, r2          ;     r0 = NOT r2
```

Comparison Operations

```
CMP   r1, r2          ;     set cc on r1 – r2
CMN   r1, r2          ;     set cc on r1 + r2
TST   r1, r2          ;     set cc on r1 AND r2
TEQ   r1, r2          ;     set cc on r1 XOR r2
```

These instructions do not produce a result. Just set the condition code bits (N, Z, C and V) in the cpsr according to the selected operation.

The mnemonics stand for ‘compare’ (CMP) compare negated (CMN), bit test (TST) and test equal (TEQ).

Immediate Operands: If, instead of adding two registers second operand can be replaced by immediate value.

```
ADD  r3, r3, # 1      ;     r3 = r3 + 1
```

Shifted register operands: By using barrel shifter, the second operand may be shifted before it is combined with the first operand.

```
ADD   r3, r2, r1, LSL # 3     ;     r3 = r2 + 8 x r1
```

Shift Operations:

LSL : Logical shift left by 0 to 31 places; fill the vacated bits at LS and of the word with zero.
LSR : Logical shift right by 0 to 32 places; fill the vacated bits at MS end of the word with zero.
ASL : Arithmetic shift left; Synonym for LSL
ASR : Arithmetic shift right by o to 32 places; fill the vacated bits at MS end of the word with zeros if the source operand was positive, or with ones if negative.
ROR : rotate right by 0 to 32 places;
RRX : rotate right extended by 1 place (with carry)

Setting the condition codes:

Any data processing instruction can set condition codes (N, Z, C and V) if the programmer wishes it to. At the assembly language level this request is indicated by adding 'S'. S stands for set condition codes.

Example: 64 bit addition

```
ADDS  r2, r2, r0;
ADC r3, r3, r2
```

Data Transfer Instructions :

There are three basic forms of data transfer instructions. Data transfer instructions move data between ARM registers and memory.

Single register load & store : data may be byte, 16 bit or 32 bit.
Multiple register load & store : large data transfer in efficient manner.
Single register swap : Exchange register with memory content.

```
LDR    r0, [r1]          ;   r0 ⇦ mem32[r1]

STR r0,[r1]              ;   mem32[r1] ⇔ r0

LDR r0, [r1, # 4]        ;   r0 ⇦ mem32[r1+4] pre indexing
                             (base + offset)

LDR r0, [r1, # 4]!       ;   r0 ⇦ mem32[r1+4] with pre index + write back
                         ;   r1 ⇦ r1 + 4
```

The exclamation mark indicates that the instruction should update the base register.

```
LDR r0, [r1], #4      ;     r0 ⇦ mem32[r1]
                      ;     r1 = r1 + 4 post index
LDR B r0, [r1]        ;     r0 ⇦ mem8 [r1]
                            8 bit data in memory
STR B
LDR H       half word
STR H
LDR SB      load signed byte
LDR SH      load signed half word
The offset may be immediate, register, scaled register.
```

Multiple Load store Instructions :

These instructions allow any subset (or all) of the 16 registers to be transferred with a single instruction.

		Instructions	
LDM	Load Multiple registers		
STM	Save Multiple registers	LDM 1A	LDM DA
1A	increment after	LDM 1B	LDM DB
1B	increment before	STM 1A	STM DA
DA	decrement after	STM 1B	STM DB
DB	decrement before		

Co-processor Instructions

There are two types of undefined instruction. The first set are totally illegal, and cause the undefined instruction vector to be called whenever they are encountered. The second set are only classed as illegal if there is no co-processor in the system which can execute them. Unsurprisingly, these are called the co-processor instructions. Note that the ARM itself treats all undefined instructions as possible co-processor ones. The only thing which distinguishes the first class from the second is that no co-processor will ever indicate that it can execute the first type.

CPI Co-processor instruction

The ARM asserts this when it encounters any undefined instruction. All co-processors in the system monitor it, ready to leap into action when they see it become active.

CPA Co-processor absent

When a co-processor sees that an undefined instruction has been fetched (by reading the CPI signal), it uses CPA to tell the ARM if it can execute it. There are two parts to the instruction which determine whether it is 'co-process able'. The first is a single bit which indicates whether a particular undefined instruction is a proper co-processor one (the bit is set) or an undefined one (the bit is clear).

The second part is the co-processor id. This is a four-bit field which determines which of 16 possible co-processors the instruction is aimed at. If this field matches the co-processor's id, *and* the instruction is a true co-processor one, the co-processor lets the ARM know by setting the CPA signal low. If none of the co-processors recognizes the id, or there aren't any, the CPA line will remain in its normal high state, and the ARM will initiate an undefined instruction trap.

CPB Co-processor busy

Once a co-processor claims an instruction, the ARM must wait for it to become available to actually execute it. This is necessary because the co-processor might still be in the middle of executing a previous undefined instruction. The ARM waits for the co-processor to become ready by monitoring CPB. This is high while the co-processor is tied up performing some internal operation, and is taken low when it is ready to accept the new instruction.

Software interrupt instruction

The final group is the most simple, and the most complex. It is very simple because it contains just one instruction, **SWI**, whose assembler format has absolutely no variants or options.

The general form of **SWI** is:

SWI{cond} <expression>

SWI is the user's access to the operating system of the computer. When a **SWI** is executed, the CPU enters supervisor mode, saves the return address in R14_SVC, and jumps to location 8. From here, the operating system takes over. The way in which the **SWI** is used depends on **<expression>**. This is encoded as a 24-bit field in the instruction.

HARDWARE DESCRIPTION

Kitek's ARM 7 Embedded Trainer UET 01 ARM consists of Piggyback Module of ARM 7 CPU and a Base board consisting of Fourteen Applications. The connection details of **Connector CN-2** with both Application Modules and Processor Pins is as below

Application Pins terminated on CN2	CN2 Pin Details	ARM 7 Pin Connection (CPU Card)
8051 LED	PIN 1	--
PIC LED	PIN 2	--
FPGA LED	PIN 3	--
ARM LED	PIN 4	GND
138/1– RS(LCD)	PIN 5	P0.0
138/2– RW(LCD)	PIN 6	P0.1
138/3– E(LCD)	PIN 7	P0.12
7 LED-4/SELECT	PIN 8	NC
D0-LED/LCD	PIN 9	P0.4
D1-LED/LCD	PIN 10	P0.5
D2-LED/LCD	PIN 11	P0.6
D3-LED/LCD	PIN 12	P0.7
D4-LED/LCD	PIN 13	P0.8
D5-LED/LCD	PIN 14	P0.9
D6-LED/LCD	PIN 15	P0.10
D7-LED/LCD	PIN 16	P0.11
GND	PIN 17	--
GND	PIN 18	--
O-LED1	PIN 19	P0.0
I-LED1	PIN 20	P1.26
O-LED2	PIN 21	P0.1
I-LED2	PIN 22	P1.25
O-LED3	PIN 23	P0.2
I-LED3	PIN 24	P1.24
O-LED4	PIN 25	P0.4
I-LED4	PIN 26	P1.23
O-LED5	PIN 27	P0.5
I-LED5	PIN 28	P1.22
O-LED6	PIN 29	P0.5
I-LED6	PIN 30	P1.21
O-LED7	PIN 31	P0.7
I-LED7	PIN 32	P1.20
O-LED8	PIN 33	P0.8
I-LED8	PIN 34	P1.30
O-LED9	PIN 35	P0.9
I-LED9	PIN 36	P1.29
O-LED10	PIN 37	P0.10
I-LED10	PIN 38	P1.28
NC	PIN 39	--
NC	PIN 40	--

These are the connection details of **Connector CN-3** with both Application Modules and Processor Pins

Application Pins terminated on CN3	CN3 Pin Details	ARM 7 Pin Connection
GND	PIN 1	--
GND	PIN 2	--
--	PIN 3	--
--	PIN 4	--
O-LED11	PIN 5	P0.11
I-LED11	PIN 6	P1.27
O-LED12	PIN 7	P0.12
I-LED12	PIN 8	P1.19
O-LED13	PIN 9	P0.13
I-LED13	PIN 10	P1.18
O-LED14	PIN 11	P0.14
I-LED14	PIN 12	P1.17
O-LED15	PIN 13	P0.15
I-LED15	PIN 14	P1.16
O-LED16	PIN 15	P0.16
I-LED16	PIN 16	P1.31
--	PIN 17	--
--	PIN 18	--
--	PIN 19	--
--	PIN 20	--
--	PIN 21	--
--	PIN 22	--
--	PIN 23	--
--	PIN 24	--
--	PIN 25	--
--	PIN 26	--
--	PIN 27	--
--	PIN 28	--
--	PIN 29	--
--	PIN 30	--
CN14-1	PIN 31	D+
CN14-2	PIN 32	D-
CN14-3	PIN 33	P0.31
CN14-4	PIN 34	NC
CN14-5	PIN 35	P0.4/ (SCK)
CN14-6	PIN 36	P0.5 (MIS10)
CN14-7	PIN 37	P0.6 (MOS10)
CN14-8	PIN 38	P0.7 (SSEL)
NC	PIN 39	--
NC	PIN 40	--

These are the connection details of **Connector CN-4** with both Application Modules and Processor Pins

Application Pins terminated on CN4	CN4 Pin Details	ARM 7 Pin Connection
GND	PIN 1	--
GND	PIN 2	--
--	PIN 3	--
--	PIN 4	--
--	PIN 5	--
--	PIN 6	--
--	PIN 7	--
--	PIN 8	--
--	PIN 9	--
--	PIN 10	--
--	PIN 11	--
--	PIN 12	--
--	PIN 13	--
--	PIN 14	--
--	PIN 15	--
--	PIN 16	--
--	PIN 17	--
DS1820	PIN 18	P0.18
WP-I2C	PIN 19	P0.17
SCL	PIN 20	P0.2
SDA	PIN 21	P0.3
ROW-1	PIN 22	P1.16
ROW-2	PIN 23	P1.17
ROW-3	PIN 24	P1.18
ROW-4	PIN 25	P1.19
COLUMN-1	PIN 26	P1.20
COLUMN-2	PIN 27	P1.21
COLUMN-3	PIN 28	P1.22
COLUMN-4	PIN 29	P1.24
Tx1	PIN 30	P0.0
Rx1	PIN 31	P0.1
Tx2	PIN 32	P0.8
Rx2	PIN 33	P0.9
--	PIN 34	--
--	PIN 35	--
+12V	PIN 36	--
RELAY	PIN 37	P0.19
BUZZER	PIN 38	P0.20
NC	PIN 39	--
NC	PIN 40	--

Connection details of **Connector CN-5** with both Application Modules and Processor Pins

Application Pins terminated on CN5	**CN5 Pin Details**	**ARM 7 Pin Connection**
NC	PIN 1	--
NC	PIN 2	--
1.2V	PIN 3	--
2.5V	PIN 4	--
1.5V	PIN 5	--
3.3V	PIN 6	--
5V	PIN 7	--
APLN- SUPPLY	PIN 8	--
AVR LED	PIN 9	--
--	PIN 10	--
--	PIN 11	--
--	PIN 12	--
CN10-1	PIN 13	P0.12
CN10-2	PIN 14	P0.13
CN10-3	PIN 15	P0.10
CN10-4	PIN 16	P0.11
CN10-5	PIN 17	P0.8
CN10-6	PIN 18	P0.9
CN10-7	PIN 19	P0.30
CN10-8	PIN 20	P0.31
CN10-9	PIN 21	P0.28
CN10-10	PIN 22	P0.29
CN10-11	PIN 23	P0.23
CN10-12	PIN 24	P0.25
CN10-13	PIN 25	P0.21
CN10-14	PIN 26	P0.22
CN10-15	PIN 27	P0.6
CN10-16	PIN 28	P0.7
CN10-17	PIN 29	P0.4
CN10-18	PIN 30	P0.5
CN10-19	PIN 31	P0.2
CN10-20	PIN 32	P0.3
CN10-21	PIN 33	P0.0
CN10-22	PIN 34	P0.1
CN10-23	PIN 35	P0.14
CN10-24	PIN 36	P0.15
--	PIN 37	--
--	PIN 38	--
GND	PIN 39	--
GND	PIN 40	--

SWITCH SETTINGS OF ARM 7 DAUGHTER BOARD

SWITCH SW4				SWITCH SW3	
1	**2**	**3**	**4**	**For ISP SW3 is**	**For IAP SW3 is**
OFF	OFF	ON	ON	PRESSED	RELEASED

ISP = ISP PROGRAMMING
IAP = APPLICATION RUNNING

Switch Position Details of ARM 7 Embedded Kit.

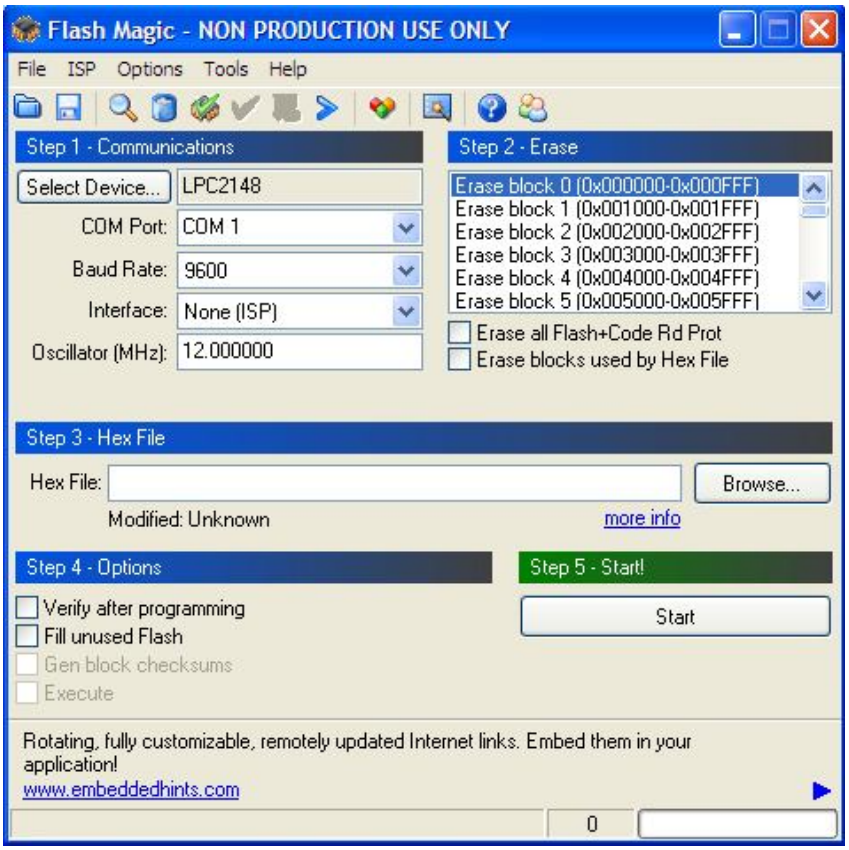
S.N	NAME OF PROGRAM	SWITCH (SW1)		SWITCH SETTINGS				JUMPER SETTINGS					
		ON	OFF	SW2	SW3	SW4	JP1	JP2	JP3	JP4	JP5	JP6	JP8
1	Seven Seg. Display	5,6 & 7	1,2,3, 4 & 8	OFF	OFF	OFF	2 & 3	2 & 3	2 & 3	2 & 3	---	---	---
2	LCD Display	1,2 & 3	4,5,6, 7 & 8	OFF	OFF	OFF	---	---	---	---	---	---	---
3	16 Input LED/Switch	4	1,2,3,5, 6,7 & 8	OFF	OFF	OFF	---	---	---	---	---	---	---
4	16 Output LED	8	1,2,3,4 5,6, & 7	OFF	OFF	OFF	---	---	---	---	---	---	---
5	Key Matrix.	1,2 & 3	4,5,6, 7 & 8	ON	OFF	OFF	---	---	---	---	---	---	---
6	Relay	1,2 & 3	4,5,6, 7 & 8	OFF	OFF	ON(2)	---	---	---	---	---	---	---
7	Buzzer	1,2 & 3	4,5,6, 7 & 8	OFF	OFF	ON(1)	---	---	---	---	---	---	---
8	RTC	1,2 & 3	4,5,6, 7 & 8	OFF	OFF	ON (3 & 4)	---	---	---	---	---	---	---
9	I2C ROM	1,2 & 3	4,5,6, 7 & 8	OFF	OFF	ON (3 & 4)	---	---	---	---	---	---	---
10	ADC/DAC	1,2 & 3	4,5,6, 7 & 8	OFF	OFF	ON (3 & 4)	---	---	**Note**	---	2 & 3	2 & 3	---
11	LM35 TEMP	1,2 & 3	4,5,6, 7 & 8	OFF	OFF	ON (3 & 4)	---	---	1 & 2	---	1 & 2	2 & 3	2 & 3
12	DS 18B20 TEMP	1,2 & 3	4,5,6, 7 & 8	OFF	ON	OFF	---	---	---	---	---	---	---

Note:

ADC/DAC - When JP5 in 1 & 2 - Input is taken to LM35 from Analog Input 1 (AIN1) of PCF8591
- When JP5 in 2 & 3 - Input is taken from External Source for Analog Input (AIN1) for PCF8591

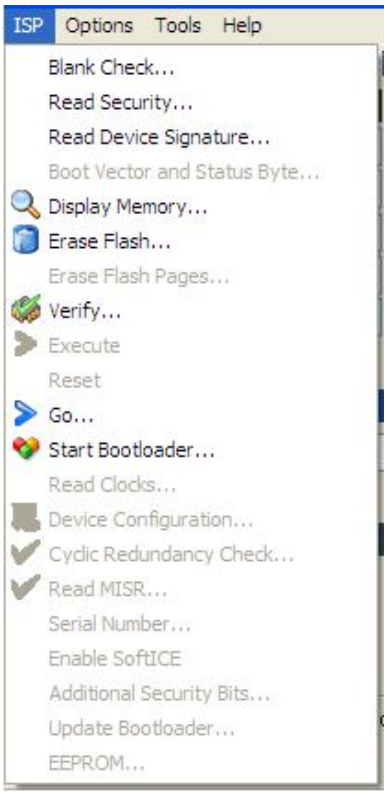
ARM ISP PROGRAMMER SETUP

1. Install flash magic in your computer.
2. Run flash Magic from desktop.
3. Select "Device" → ARM 7 → LPC 2148 in **STEP 1 - Communication** window
   Com port – com 1
   Baud rate – 9600
   Interface – None (ISP)
   Oscillator – 12.00000



4. Connect Serial cable between CN12 (9 pin connector) to PC - COM port using 9 Pin Serial Cable. **Note:** Ensure SW4 (3, 4) in ARM module is in ON Position
5. To check the if the Kit is communicating with PC or not, Select ISP → Read device signature
   Note: If the kit is communicating it will display device signature, otherwise it will show failed.
6. Now your Kit is ready to program the device
7. Select the Erase Block to be used by your hex file in **STEP 2 – Erase** window and Click on Erase blocks used by hex file icon
8. Now select your file to download using browse in **STEP 3 – Hex file** window.
   Note: Keep SW3 on ARM module in pressed condition for programming the device & then Press Reset Key which resets the Processor and gets into ISP Mode
9. After loading your file press "Start" in **STEP 5 – Start** window.
10. After completion of programming keep "SW3" in ARM module in released condition and press reset Key. This resets the Processor and Mode gets changed

11. As the system is in Application Mode the downloaded application program starts executing.
    Note: All sample Codes are available in Code directory.
12. Various other functions as shown can be performed by clicking ISP Menu.



Features

- Straightforward and intuitive user interface
- Five simple steps to erasing and programming a device and setting key options
- Programs Intel Hex Files
- Automatic verifying after programming
- Fills unused Flash to increase firmware security
- Automatically program checksums. Using the supplied checksum calculation routine your firmware can easily verify the integrity of a Flash block, ensuring no unauthorized or corrupted code can ever be executed
- Program security bits
- Check which Flash blocks are blank or in use with the ability to easily erase all blocks in use
- Read any section of Flash and save as an Intel Hex File
- Reprogram the Boot Vector and Status Byte with the help of confirmation features that prevent accidentally programming incorrect values
- Display the contents of Flash in ASCII and Hexadecimal formats
- Single-click access to the manual, Flash Magic / NXP Microcontrollers home page
- Use high-speed serial communications on devices that support it.
- Command Line interface allowing use in IDEs and Batch Files
- Manual in PDF format
- Supports half-duplex communications for many devices

- Verify Hex Files previously programmed
- Save and open settings
- Control the DTR and RTS RS232 signals to place the device into BootROM and Execute modes automatically (requires hardware support)
- Send commands to place the device in Bootloader mode
- Play any Wave file when finished programming
- Powerful, flexible Just In Time Code feature. Write your own JIT Modules to generate last minute code for programming, for example serial number generation.
- Displays information about the selected Hex File, including the creation and modification dates, flash memory used, percentage of the current device used
- Ethernet bootloader for LPC2xxx devices
- Support programming certain LPC2xxx devices via Ethernet
- Read the device signature
- Build your own Flash Magic based applications using the DLLs

Requirements

- Flash Magic works on Windows 2000, XP and Vista.
- 10Mb of disk space is required

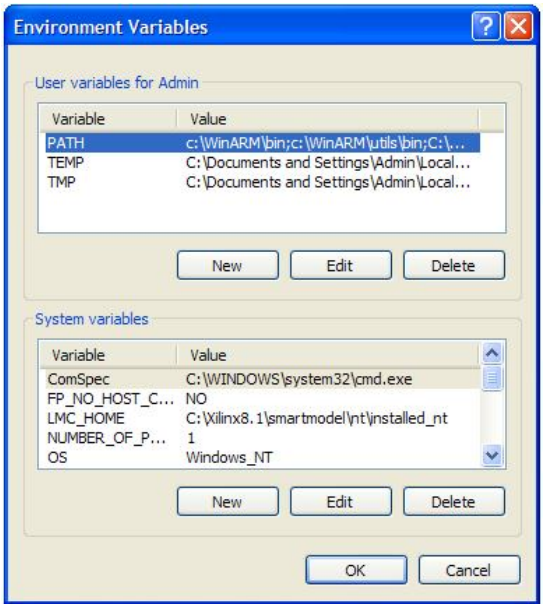
INSTALLATION WINARM COMPLIER

1) Extract WinArm-20060606 file in **"WinArm"** folder in main drive i.e. C drive.
2) Set Environmental variables.

Select **"Control Panel"** → **"System"** → Click on **"Advanced"**

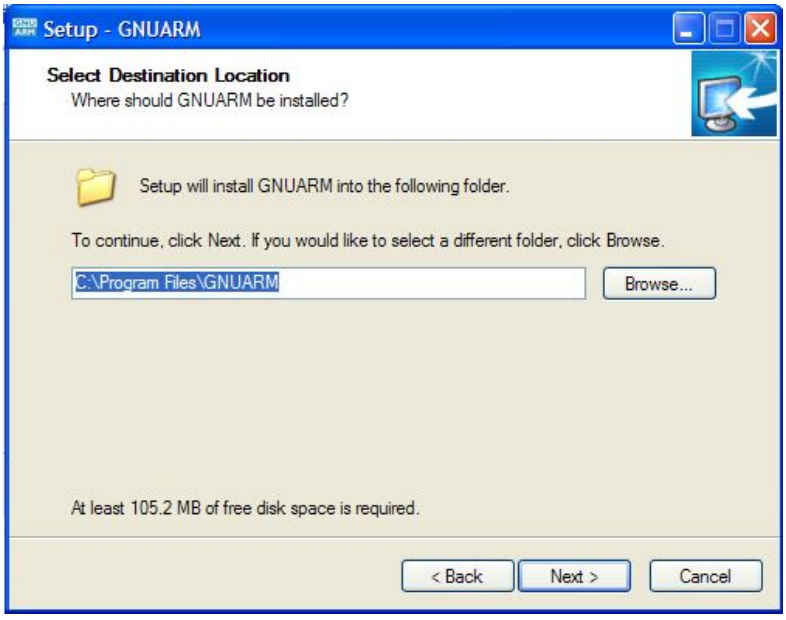
Select **"Environmental variables"** →

Edit **"User Variable for Admin"** → Path

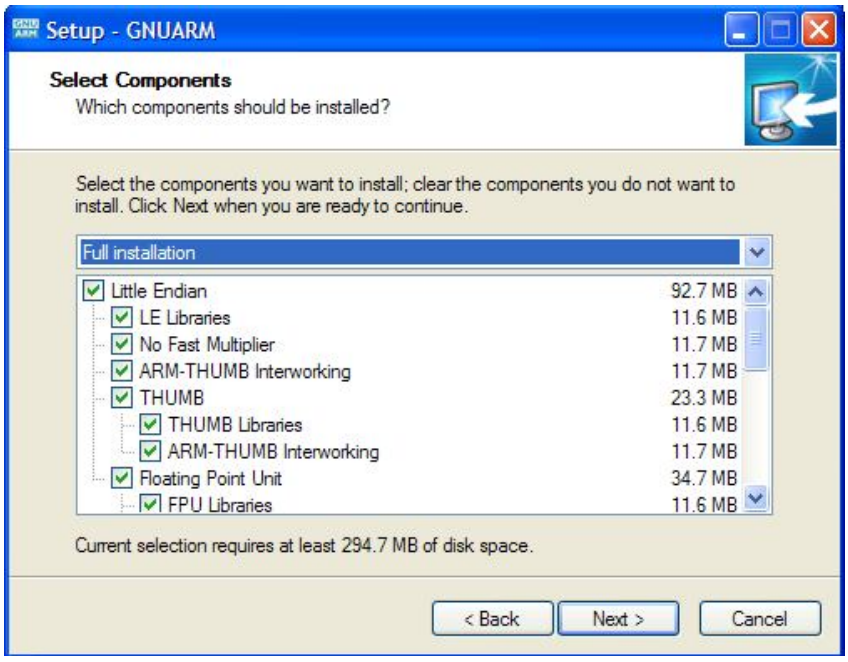


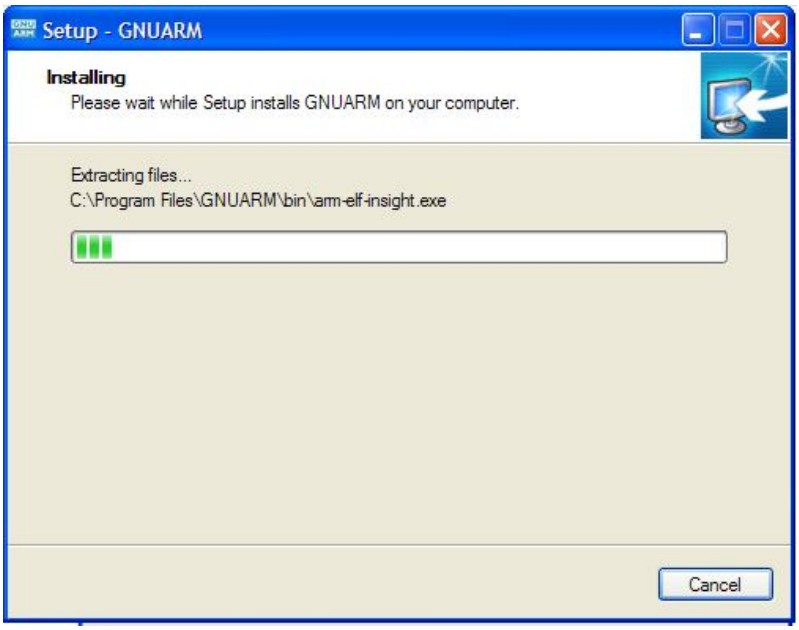
3) Install GNUARM setup file.

Click on **"I accept the agreement"** → **"Next"**
Then **"Browse"** the location where the Software is to be installed → **"Next"**



Select **"Full Installation"** → "**Next"** then → "**Next"** again → "**Next"** then → **"Install"**





4) Now the installation of WINARM compiler is over.
5) Run **"Pn.exe"** from WinArm folder.
6) To compile existing C file, open any existing project file using
   File → **"Open project"** → **"prjSrArithmatic.pnproj"** → **"open"**
7) Now your existing project file will be opened.
8) Now to compile click **"Tool"** → **"Make all".**
9) After compilation if there are no errors, the "**Main.hex"** will be generated.

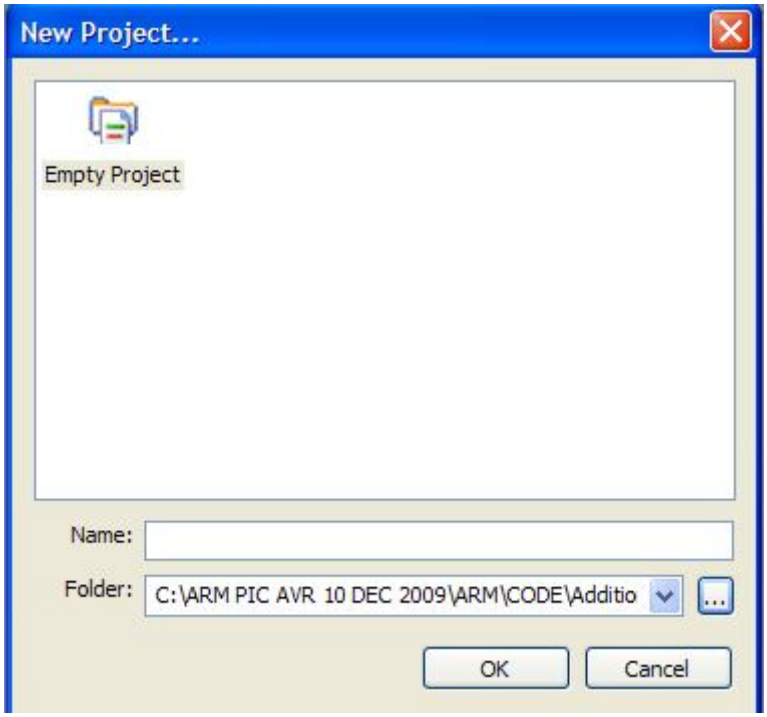
New project

If New project is to be created then follow the below procedure**:** Eg Arithmatic program

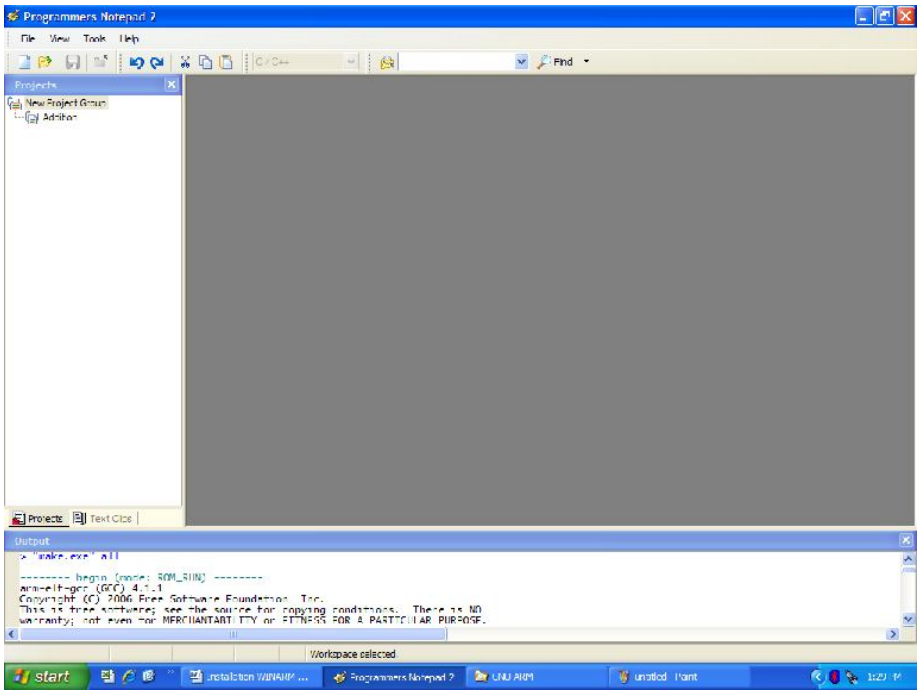
10) Select **"File"** → **"Project"** - CODE directory-Select Folder CODE

11) Click "Make New Folder"

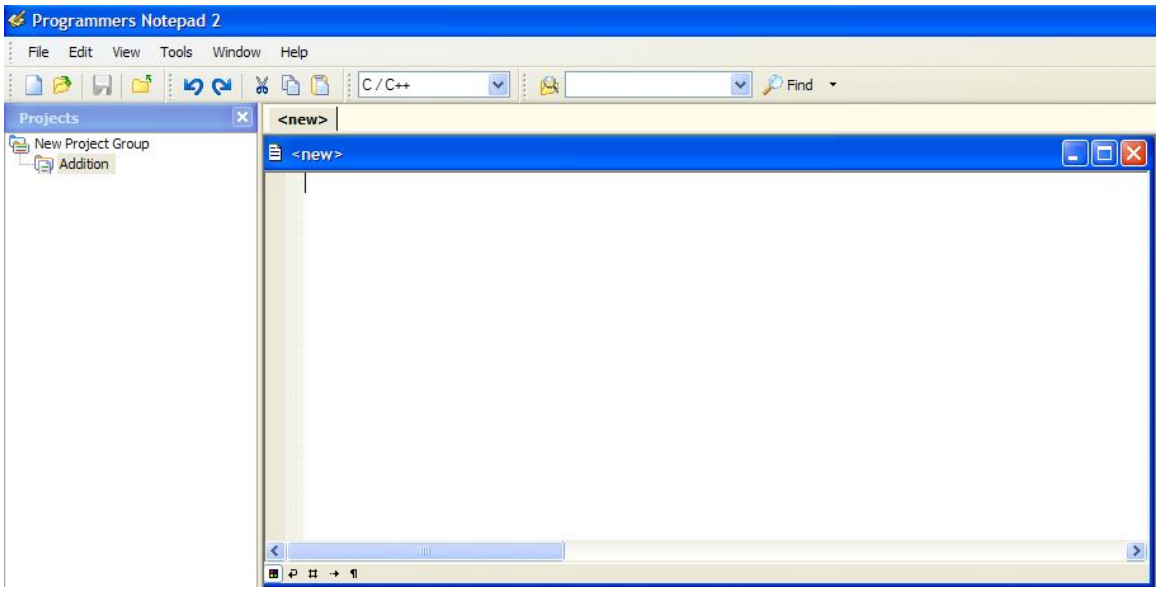
12) Write your project name folder Box example: **Addition** → click OK



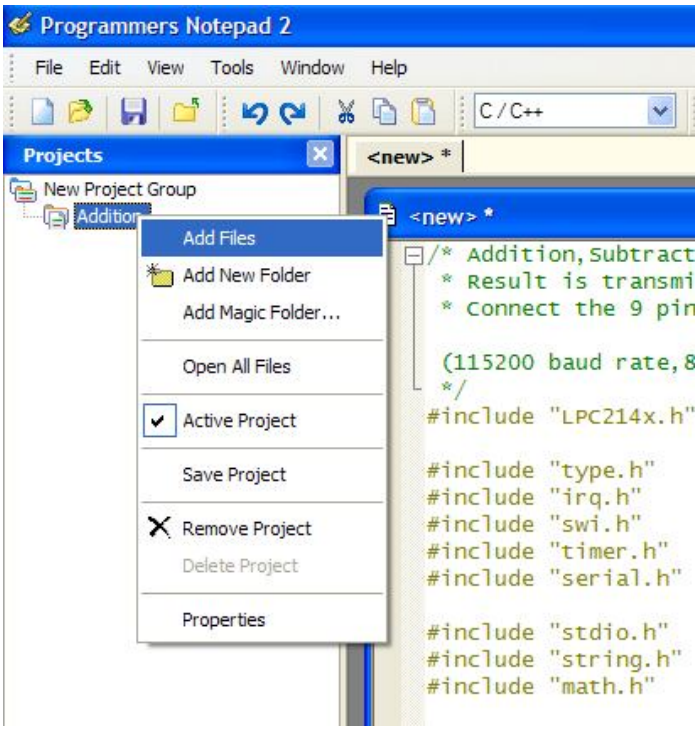
13) Now you write Project name in Name folder example: **Addition** & Click OK



14) Now Select “File” → “C/C++” you will get a new worksheet.

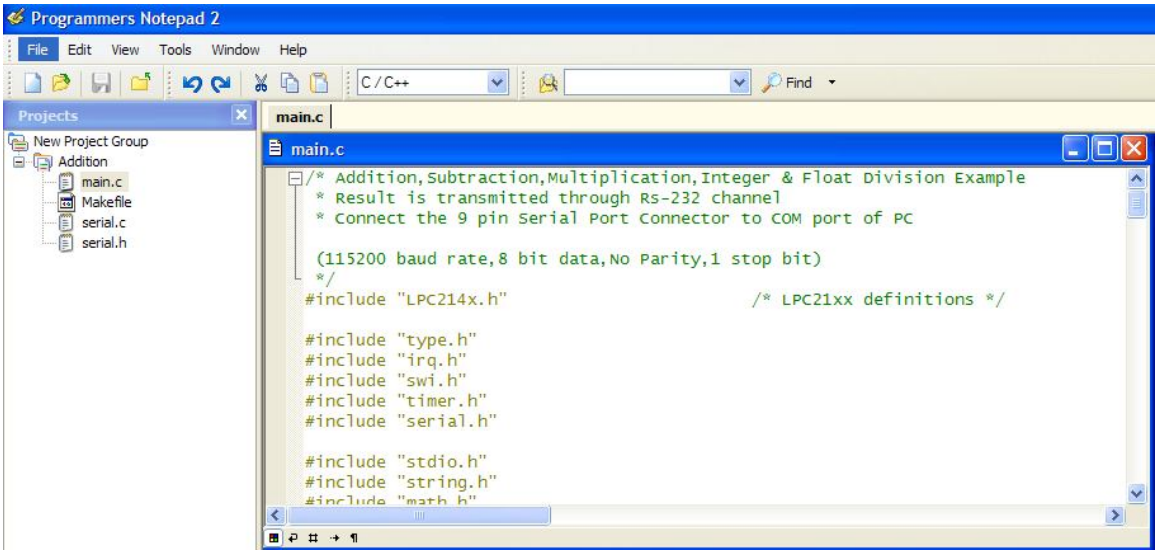


15) One can write a program in this new worksheet & save the file name in the new project folder.
16) To load the file in the project click right button on Addition icon project → click Add files



17) One has to load all *.C files, *.H files & make file form Addition Project folder.

(Note Makefile is important file, it has to add in every new project because it defines the ARM controller number, Supporting C file & Header file.)



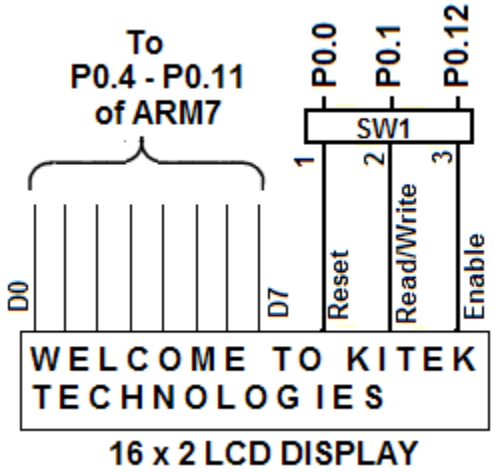
18) Now to compile click **"Tool"** → **"Make all".**
19) After compilation if there are no errors, the Main. hex will be generated.

Note: "Common_WinArm" directory should be there on "Code" directory.This directory contains all supporting header files of LPC2148.

CIRCUIT DESCRIPRION FOR ONBOARD MODULES:

1. 16 x 2 CHARACTER LCD:

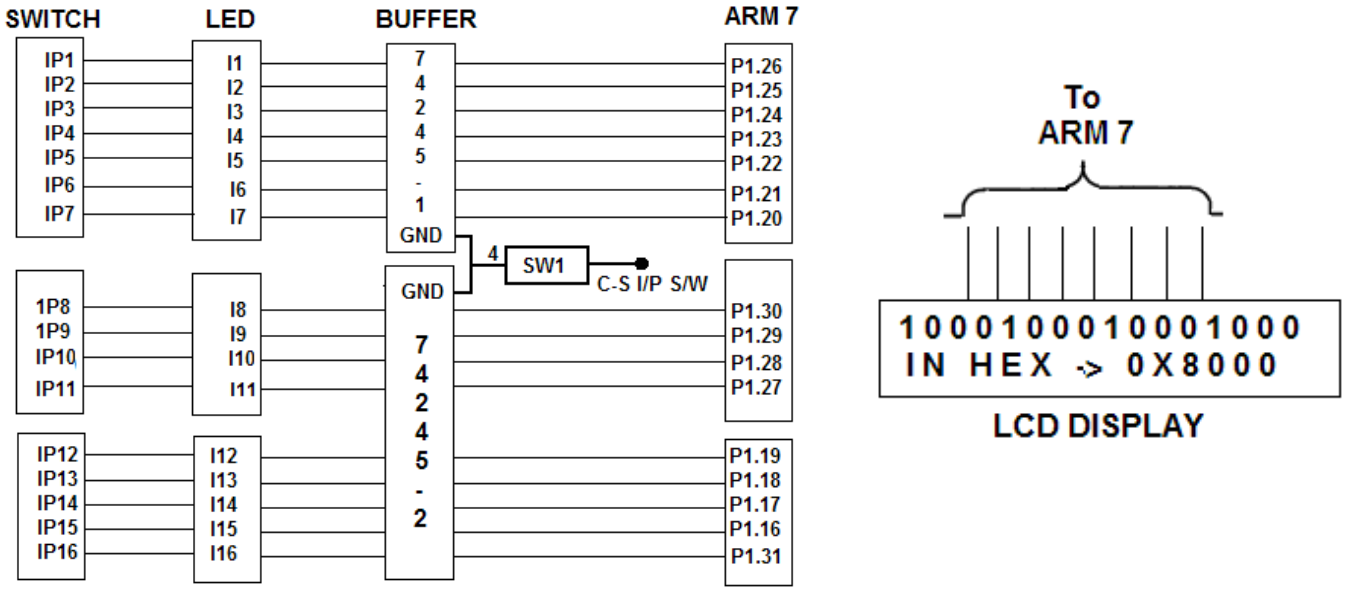
Reset (RST) of LCD is connected with P0.0 of ARM 7, READ/WRITE (R/W) of LCD is connected with P0.1 of ARM 7, Enable (E) of LCD is connected with P0.12 of ARM 7 and Data Lines of LCD (D0-D7) is connected with P0.4 - P0.11 of ARM 7. Set the switches as explained in table1. Message **"WELCOME TO KITEK TECHNOLOGIES MUM"** will be displayed on LCD on execution of Program.



2. 16 INPUT DIP switches:

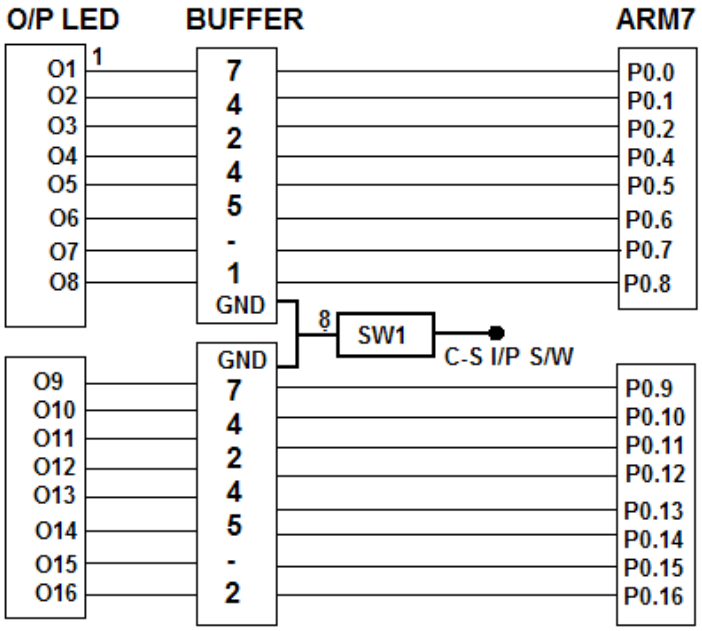
Input Switches IP1-7 is connected to its corresponding LEDs (I1-7) P1.26 – P1.20 of ARM 7, similarly IP 8-11 is connected to LEDs (I8-11) and to P1.30 – P1.27 of ARM 7, IP 12-15 is connected LEDs (I12-15) and to P1.19 – P1.16 of ARM 7 and IP 16 is connected LEDs (I16) and to P1.31 of ARM 7. Set the switches as explained in table1

On execution of Program the Output will be displayed on LCD. The output will in Binary Form. It will indicate the status of Switch by displaying 0 and 1 where "Switch On" is 1 and "Switch Off" is 0. It also displays Hex equivalent of the input



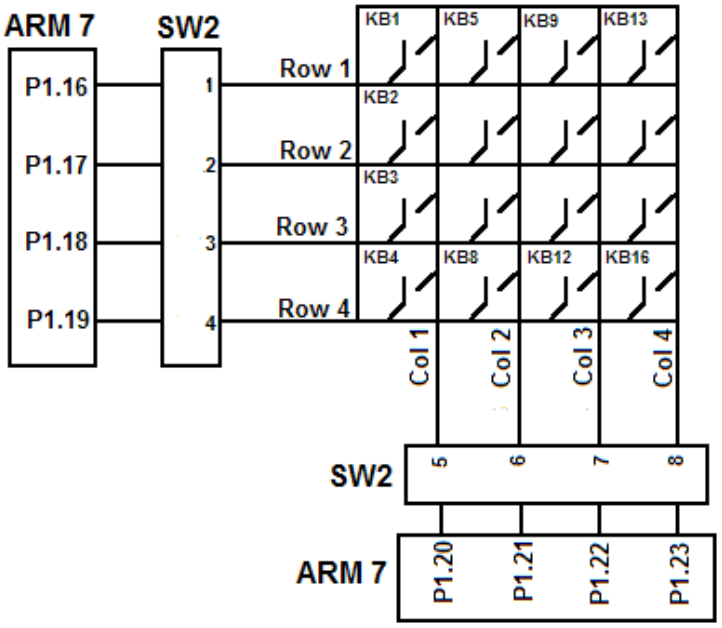
3. 16 LED indicators:

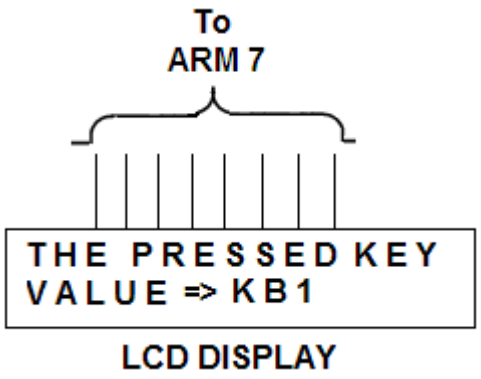
Output LED O1-O16 is connected to P0.0 – P0.16 of ARM 7 .Set the switches as explained in table1. On execution of Program, the Four LEDs, i.e. O1 - O4 will glow and after fixed interval the other four i.e. O5-O8 will glow then other four O9 - O12 and then O13 - O16 and so on.



4. KEY MATRIX:

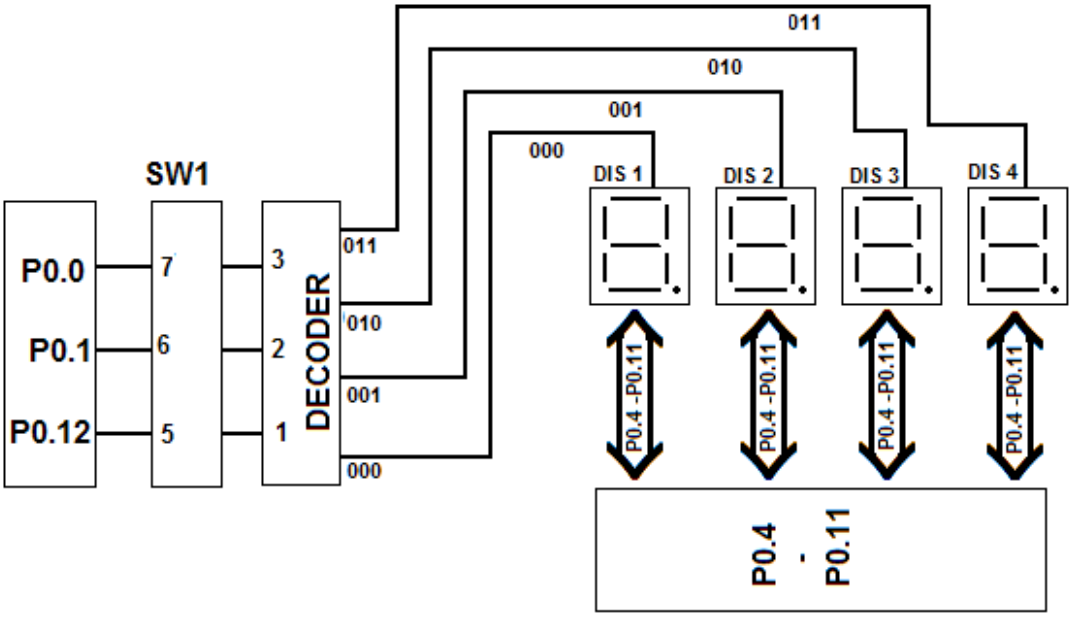
Rows of the Matrix are connected to P1.16 – P1.19 of ARM 7 and Columns of the Matrix is connected to P1.20 – P1.23 of ARM 7. Set the switches as explained in table1. On execution of Program, Initially LCD will display the message **"KEYPAD DRIVER TO DETECT KEY PRESS".** Then it will ask "**PRESS A KEY ….!"** Initially it will display "**THE PRESSED KEY VALUE => NONE"** then as we press a key say KB1 then it will display "**THE PRESSED KEY VALUE=> KB1"** on 16 X 2 LCD display.





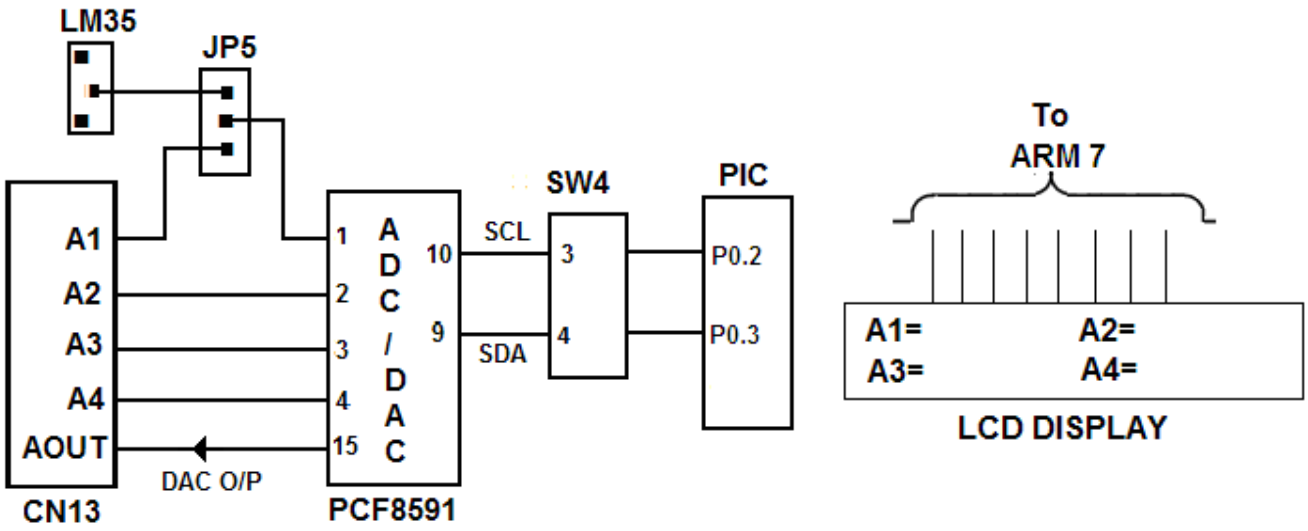
5. 7-SEGMENT:

The seven segments, D0- D7 of 7 segment display is connected to RB0- RB7 of PIC. Selection of display is done through a decoder 74138. Input to the decoder is provided by RD0, RD1 and RD2 of PIC. When 000 is given display-1 is selected, 001 then display -2, 010 then display-3 and 011 then display - 4. Set the switches as explained in table1. On execution of Program **"GOOD"** will be displayed by the four seven segments.



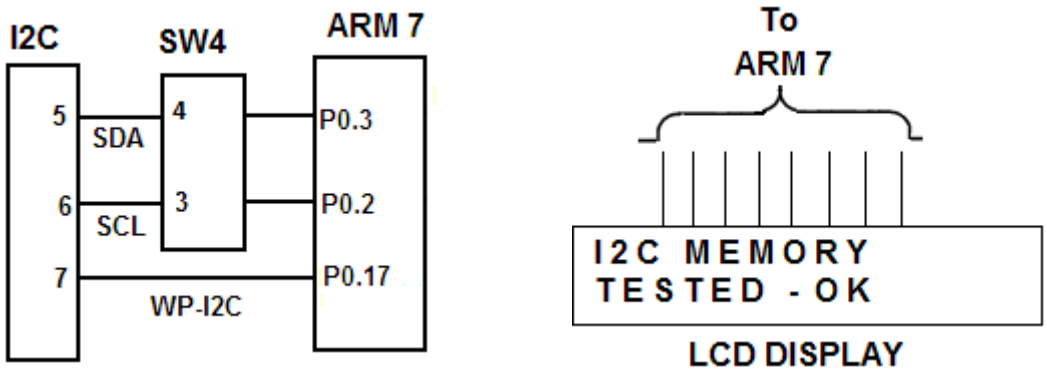
6. ADC_DAC:

Channel 1 of ADC (PCF8591) is connected to analog IN (AIN) through JP5. Input to channel is selectable i.e. can be external input through AIN1 or can be LM35. The output is provided through I2C BUS to PIC microcontroller at RC2, RC3 & RC4. Four channel Data will be displayed on LCD. Set the switches as explained in table1. On execution of Program it will display **PCF 8591 ADC & DAC …** then it will show **OBSERVE DAC O/P AT 'AOUT'** then it will show **4 SINGLE ENDED ANALOG I/P …** then finally it will show "**AI1 = AI2 = AI3 = AI4 =**" on 16 X 2 LCD display



7. SERIAL EEPROM:

The output is provided through I2C BUS to PIC microcontroller at RC2, RC3 & RC4. Set the switches as explained in table1. On execution of Program it will display "**I2C EEPROM IC MEMORY TEST"** then it will show **PAGE SIZE 128 BYTE** then **WRITING 0…127 IN EACH PAGE** then it will display **PAGE NO = - 511 WRITING DONE** then **NOW READING ALL PAGES** then **PAGE NO = 400 – 511 READING MATCHES** then finally it will show **I2C MEMORY TEST OK** on 16 X 2 LCD display



8. DS18B20 Temp sensor:

DS18B20Temperature Sensor is connected to the PIC **Microcontroller through Pin RC1**. Keep SW3 ON condition. Set the switches as explained in table1. On execution of Program it will display "**TEMPERATURE MEASUREMENT"** and then "**1 WIRE PROTOCOL DS18B20"** and then "**TEMPERATURE in CELSIUS =** ROOM Temperature" will be displayed on 16 X 2 LCD display.

9. BUZZER:

Buzzer is connected to the PIC Microcontroller through Pin RC5. Set the switches as explained in table1. On execution of Program it will display **BUZZER STATE DEENERGIZED** and a toggle time of 2 sec it will show **BUZZER STATE ENERGIZED** on 16 X 2 LCD display**.** This is a continuous process.

10. LM35 Temp sensor:

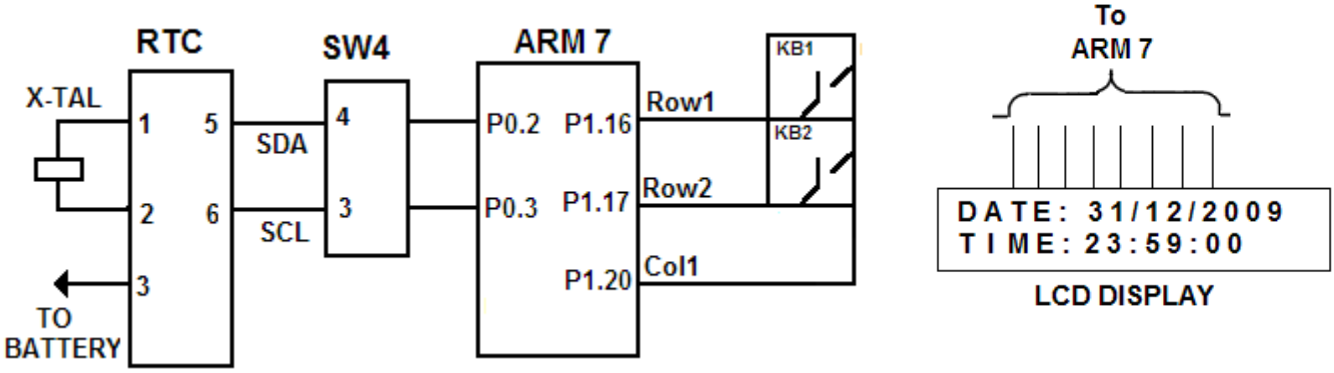
Set the switches as explained in table1. The Jumper JP5 should be fixed at 1 &2. On execution of Program it will display "**LM35 TEMPERATURE SENSOR** and then **LM35 TEMPERATURE D CELSIUS: 37** will be displayed on 16 X 2 LCD display

11. RELAY:

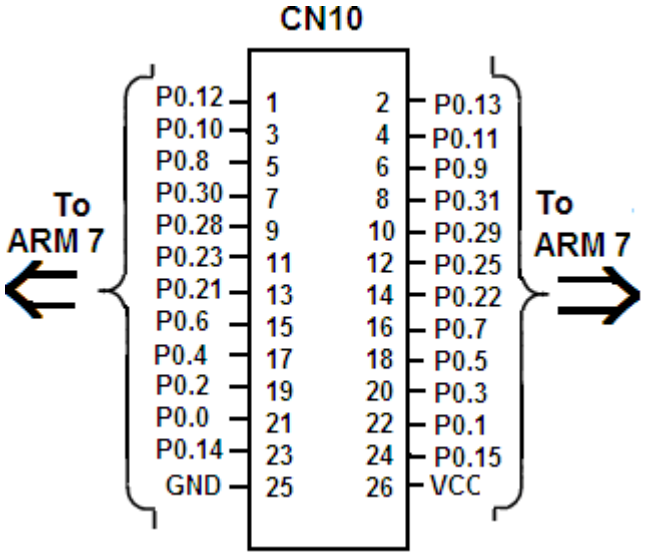
Relay is connected to the PIC Microcontroller through Pin RC0. Set the switches as explained in table1. On execution of Program it will display RELAY STATE DEENERGIZED and a toggle time of 2 sec it will show **RELAY STATE ENERGIZED** on 16 X 2 LCD display **.** This is a continuous process.

12. RTC:

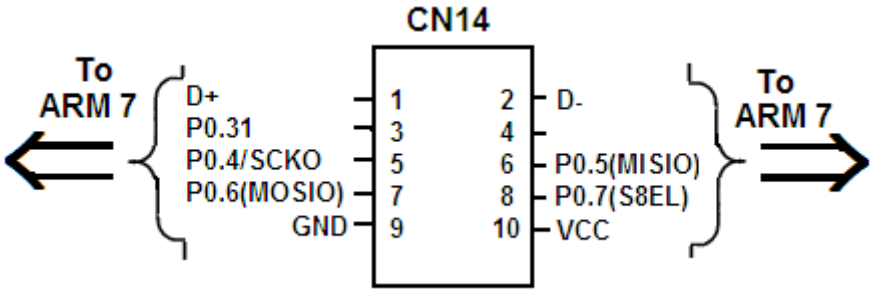
The output is provided through I2C ROM to PIC microcontroller at RC2, RC3 & RC4. Between Pin 1, & Pin 2 Crystal is connected Pin 3 is connected to Battery and keys KB1 and KB2 of Keyboard Matrix to PIC microcontroller at RA1, RA2 & RA3. Set the switches as explained in table1. On execution of Program it will display "**REAL TIME CLOCK DS1307"** then it will show **WANT TO SET NEW DATE/TIME Y/N** to enter new Date and Time press key KB1 else KB2. On pressing KB1 then it will show **SETTING NEW DATE & TIME DATE: 31/12/2009 TIME 23:59:00** if KB2 pressed then it will display **READING RTC DATE & TIME** on 16 X 2 LCD display



CONNECTOR CN10 DETAILS



CONNECTOR CN14 DETAILS

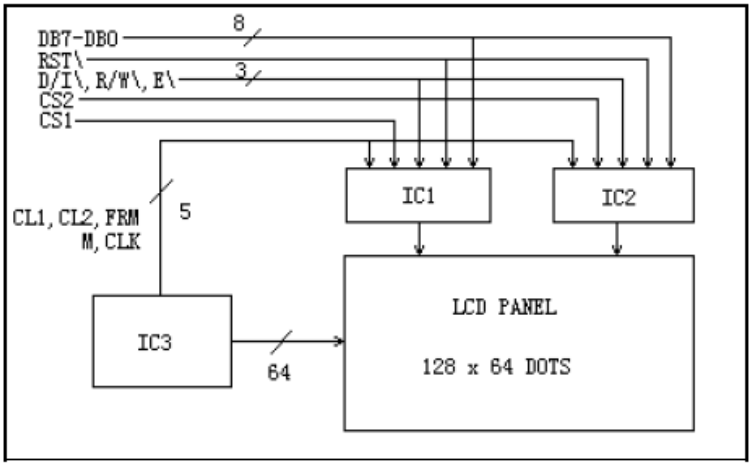


GRAPHIC LCD

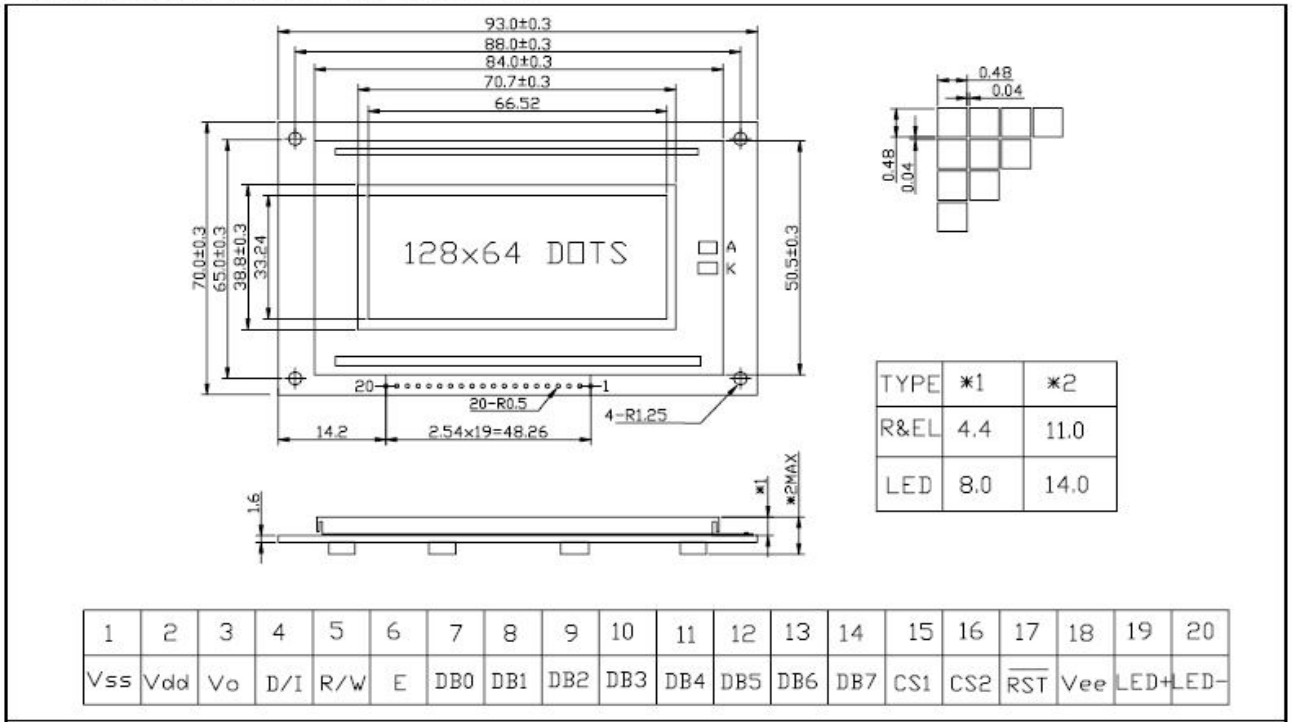
CHARACTERISTICS

DISPLAY CONTENT：128 x 64 DOTS
DRIVING MODE：1/64D
AVAILABLE TYPES：
STN(YELLOW GREEN、GREY、B/W)
REFLECTIVE、WITH EL OR LED
EL/100VAC，400HZ
LED/4.2VDC

APPLICATION CIRCUIT



DIMENSIONS/DISPLAY CONTENT



TYPE	*1	*2
R&EL	4.4	11.0
LED	8.0	14.0

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Vss	Vdd	Vo	D/I	R/W	E	DB0	DB1	DB2	DB3	DB4	DB5	DB6	DB7	CS1	CS2	$\overline{RST}$	Vee	LED+	LED-

LIMIT PARAMETER

PARAMETER	Symbol	Testing Criteria	Standard Values		UNIT
			MIN	MAX	
Supply Voltage	VDD-VSS	Ta=25℃	0	6.5	V
LCD Voltage	VDD-V0		0	15.0	V
Input Voltage	V1		0	VDD	V

ELECTRIC PARAMETER

PARAMETER		Symbol	Testing Criteria	Standard Values			UNIT
				MIN	Typical	MAX	
Voltage	LOGIC	VDD-VSS	-	4.75	5.0	5.25	V
	LCD	VDD-V0	-	-	13.0	-	V
Current	LOGIC	IDD	-	-	4.0	-	mA
	LCD	IEE	-	-	3.0	-	mA
LCD Drive Voltage (recommend)			0℃	-	14.0	-	V
			25℃	-	13.0	-	V
			40℃	-	12.0	-	V
Input Voltage	'H' Level	VIH	High	0.7VDD	-	VDD	V
	'L' Level	VIIL	Low	0	-	0.3VDD	V

PIN CONFIGURATION

PIN	SYMBOL	LEVEL	INSTRUCTION
1	VSS	0V	Ground contact (GND)
2	VDD	5.0V	Power Supply Voltage
3	V0	LCD Drive Voltage	Adjust Contrast
4	D/I	H/L	H:DATA; L:COMMAND
5	R/W	H/L	H:READ; L:WRITE
6	E	H,H→L	IC select signal
7	DB0	H/L	DATA 0
8	DB1	H/L	DATA 1
9	DB2	H/L	DATA 2
10	DB3	H/L	DATA 3
11	DB4	H/L	DATA 4
12	DB5	H/L	DATA 5
13	DB6	H/L	DATA 6
14	DB7	H/L	DATA 7
15	CS1	H	Select Signal 1,High effective
16	CS2	H	Select Signal 2,High effective
17	RST	L	RESET signal, low effective
18	VEE	-10.0V	LCD Drive negative voltage
19	LED+		Back LED Anode
20	LED-		Back LED Negative