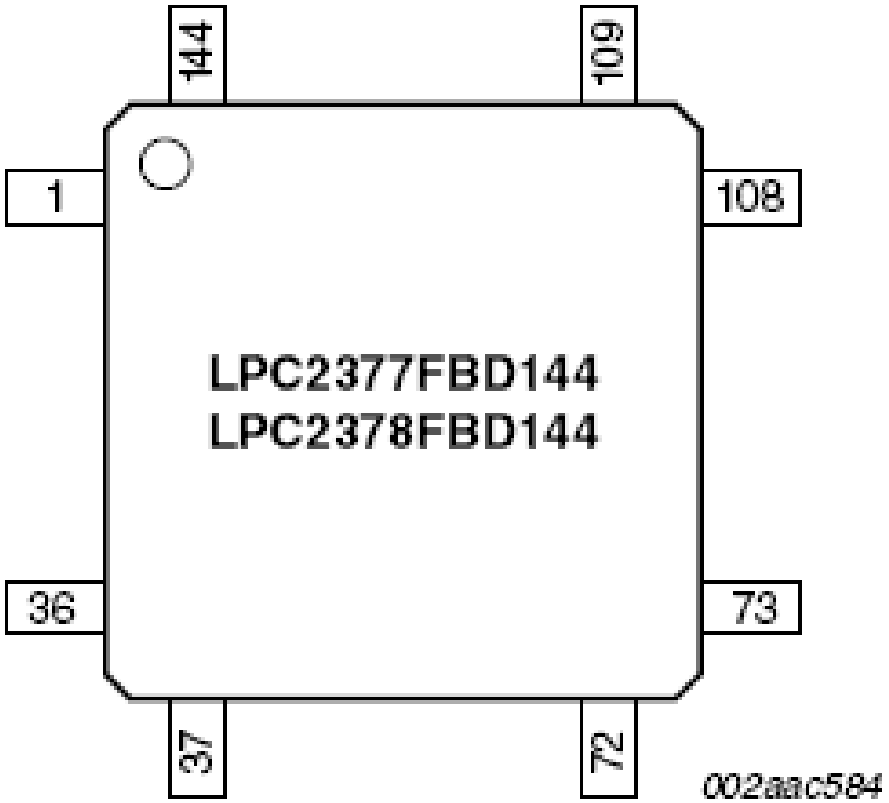
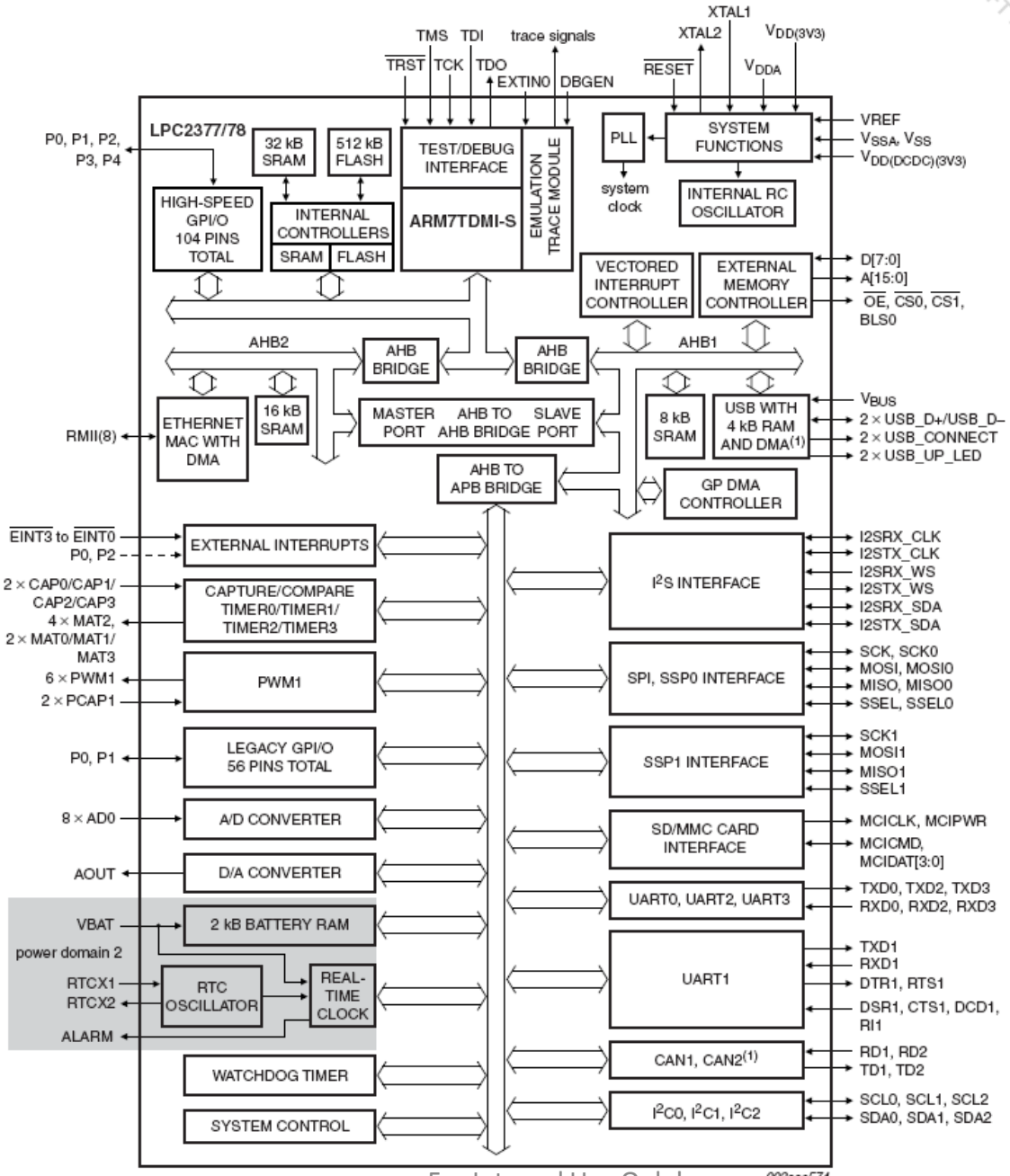
Embedded Systems and Microprocessor Systems

review

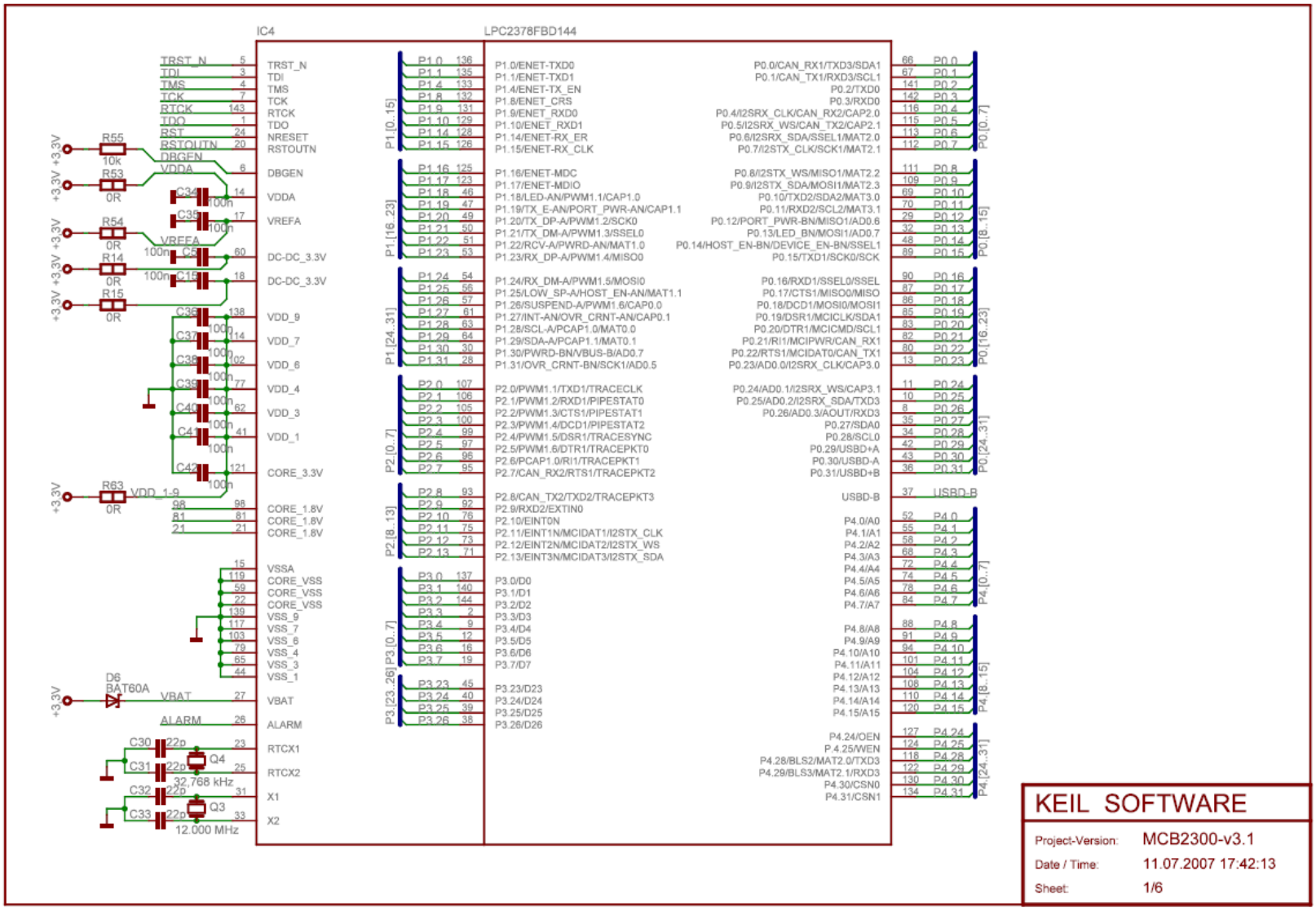
- ✓ Organization/Architecture
- ✓ Mnemonics
- ✓ Flags
- ✓ Representations
- ✓ Subroutines
- ✓ Interrupt
- ✓ Pipeline
- ✓ Encoding
- ✓ Memory
- ✓ Processor Modes
- ✓ Peripheral Communication

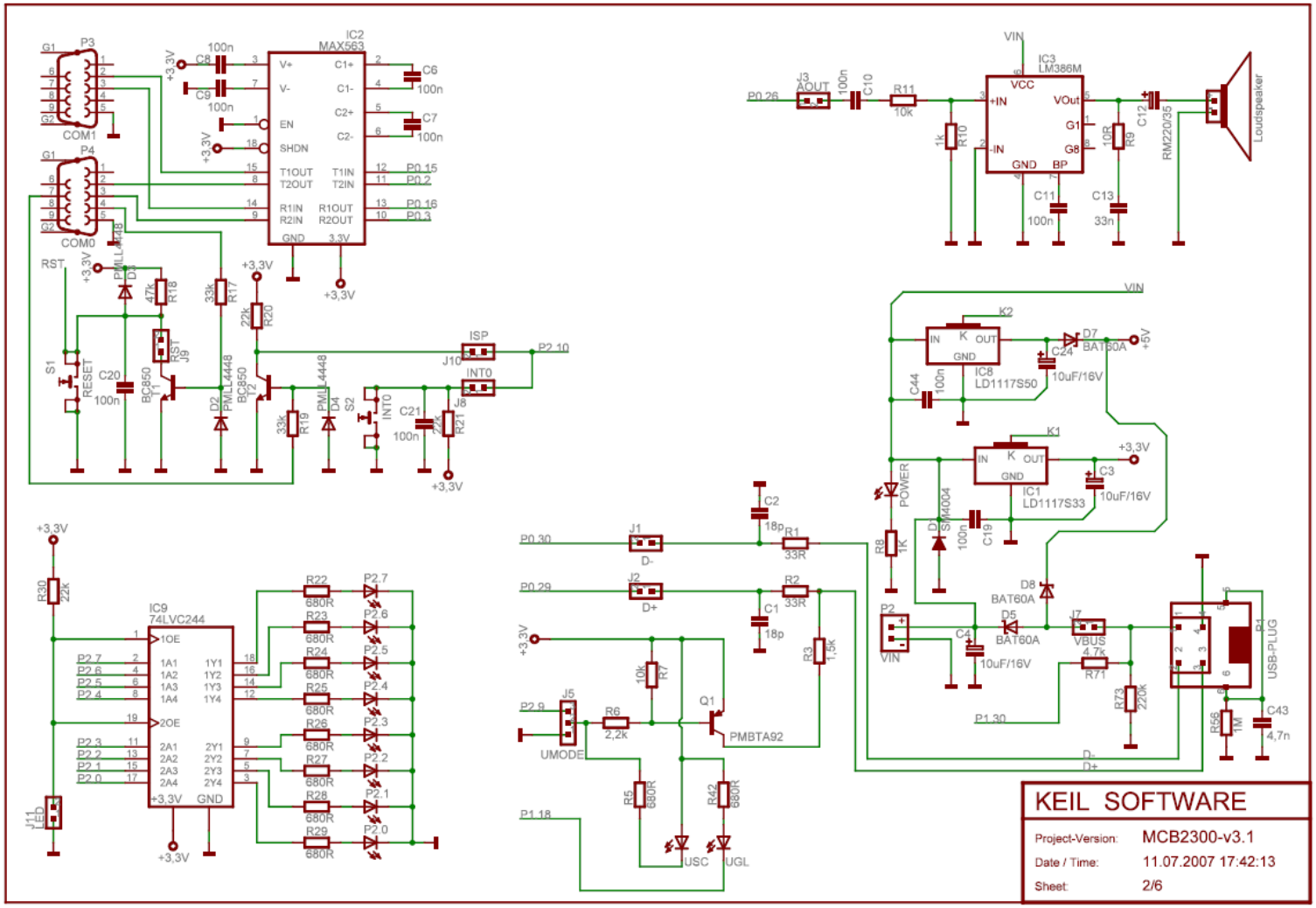
- ✓ **Organization/Architecture**
- ✓ Mnemonics
- ✓ Flags
- ✓ Representations
- ✓ Subroutines
- ✓ Interrupt
- ✓ Pipeline
- ✓ Encoding
- ✓ Memory
- ✓ Processor Modes
- ✓ Peripheral Communication


144
109
1
108
LPC2377FBD144
LPC2378FBD144
36
73
37
72
002aac584



For Internal Use Only!


IC4
LPC2378FBD144
TRST_N
TDI
TMS
TCK
RTCK
TDO
RST
RSTOUTN
DBGEN
VDDA
VREFA
DC-DC_3.3V
VDD_9
VDD_7
VDD_6
VDD_4
VDD_3
VDD_1
CORE_3.3V
VDD_1-9
CORE_1.8V
VSSA
CORE_VSS
VSS_9
VSS_7
VSS_6
VSS_4
VSS_3
VSS_1
VBAT
ALARM
RTCX1
RTCX2
X1
X2
32.768 kHz
12.000 MHz
KEIL SOFTWARE
Project-Version: MCB2300-v3.1
Date / Time: 11.07.2007 17:42:13
Sheet: 1/6


KEIL SOFTWARE
Project-Version: MCB2300-v3.1
Date / Time: 11.07.2007 17:42:13
Sheet: 2/6

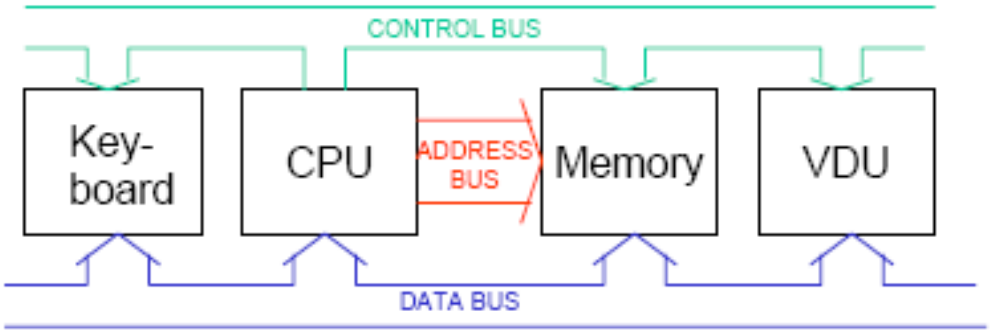
ARM7TDMI microprocessor

- 74,209 transistors
- 32 bit address bus
- 0.13 µm gate length
- upto 133 MHz clock
- 0.26 mm^2 die area
- max. power 8 mW

T: 16-bit **T**humb code
D: on-chip **D**ebug support
M: an enhanced **M**ultiplier
I: Embedded**I**CE hardware to give on-chip breakpoint and watchpoint support

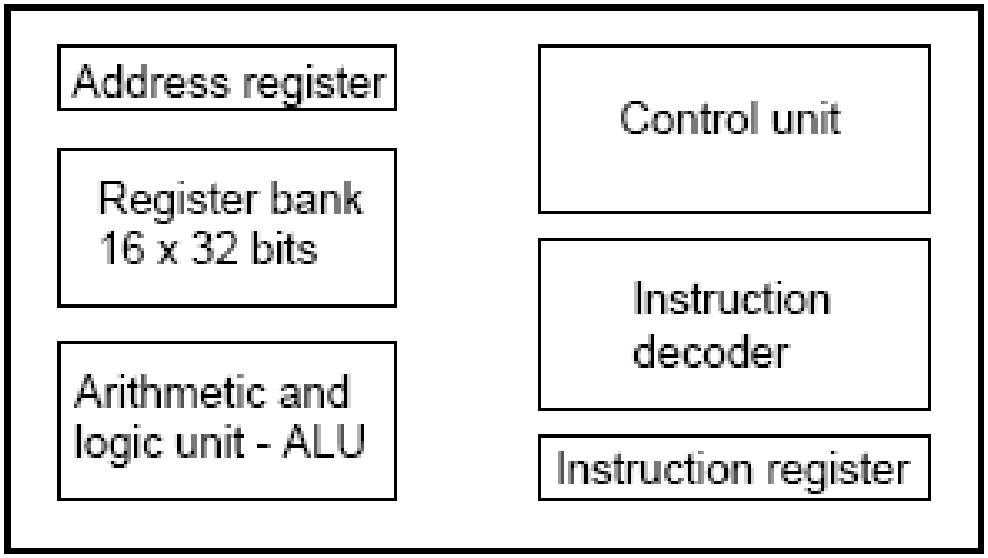
A Third Bus

- In addition to the data bus and control bus there is a third bus called the address bus.
- The address bus is used by the CPU to determine which location in memory is sending or receiving data.



The ARM7 core

- The basic building blocks of the ARM7 core are:

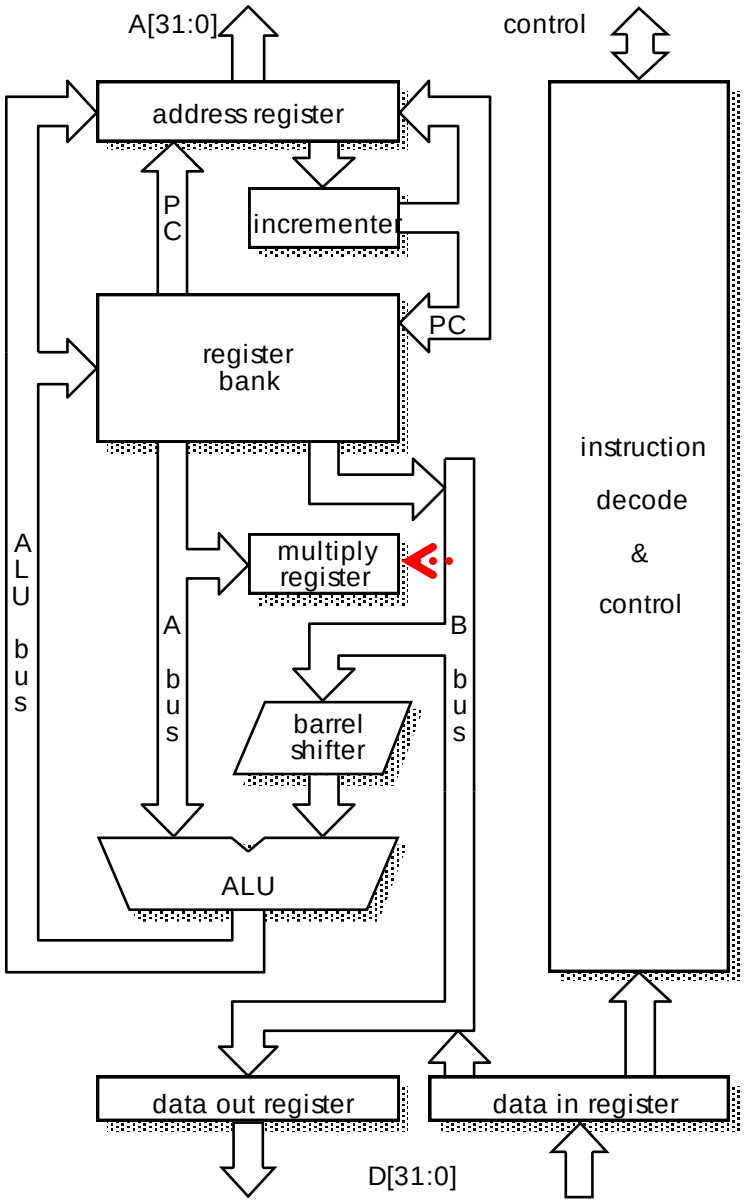


ARM7 CPU:

many internal registers;
- address register
- and register bank.

3 internal datapaths;
- A bus,
- B bus,
- ALU bus.

additional ALU functionality;
- multiplier
- and barrel shifter.



The barrel shifter

- The barrel shifter is a very useful feature of the ARM7 microprocessor which allows bit patterns to be rotated.
- E.g. the instruction

  MOV r1, r2, LSL #5

  – would take the bit pattern in register r2 and shift it 5 places to the left before placing it in register r1. So if r2 held:

    1010 0101 1100 0011 1001 0110 1110 0111

  – after the instruction was executed r1 would hold:

    1011 1000 0111 0010 1101 1100 1110 0000

ALU adder design

- ripple-carry adder
- carry look-ahead
- carry select adder

Booth's algorithm: Nth cycle

Booth's algorithm works by replacing $\times 3$ by $\times(4\text{-}1)$ so that each cycle multiples by a 2 bit value.

N bit multiplications can be completed in N/2 cycles.

The table right summarizes the action on the Nth cycle.

Table 4.3 in Furber also replaces $\times 2$ by $\times(4\text{-}2)$ for $C_{in=0}$ as this makes the control logic slightly simpler.

C_{in}	Multiplier	LSL #	ALU	C_{out}
0	$\times 00_2$	-	A+0	0
0	$\times 01_2$	2N	A+B	0
0	$\times 10_2$	(2N+1)	A+B	0
0	$\times 11_2$	2N	A–B	1
1	$\times 00_2$	2N	A+B	0
1	$\times 01_2$	(2N+1)	A+B	0
1	$\times 10_2$	2N	A–B	1
1	$\times 11_2$	-	A+0	1

For Internal Use Only!

- ✓ Organization/Architecture
- ✓ **Mnemonics**
- ✓ Flags
- ✓ Representations
- ✓ Subroutines
- ✓ Interrupt
- ✓ Pipeline
- ✓ Encoding
- ✓ Memory
- ✓ Processor Modes
- ✓ Peripheral Communication

Program counter.

- Instructions are stored in memory so:
  - How does the CPU know the memory address for the next instruction in the computer program?
  - Register r15 always holds the memory address of the next instruction to be executed.
  - An alternative name for register r15 is the 'program counter'.

Instructions

- For a human to perform a task he/she needs instructions e.g.
  - to knit a jumper
    - the instructions are the knitting pattern
  - to bake a cake
    - the instructions are in the recipe.
- Computers also need instructions
  - so that the data is processed into information correctly.

Instructions stored in memory.

- Instructions are 32 bits long, whereas memory locations are 8 bits long so one instruction occupies four locations in memory, e.g.
  - one instruction would be stored in 4 memory locations with addresses 0x00008000,0x00008001, 0x00008002 and 0x00008003.
  - The next instruction would be stored at addresses 0x00008004, 0x00008005, 0x00008006 and 0x00008007 and so on.

Simple instructions.

- One of the simplest instructions is to move a value into a register e.g.
  - move 114 into register r12
  - The machine code for this instruction is 0xE3A0C072
  - After the instruction is executed
    - register r12 will hold the value 0x00000072 (hexadecimal for 114) and
    - value in register r15, the program counter, will have increased by 4.
    - All other registers remain unchanged.

Machine code.

- Look at the machine code for the instruction;
  - move 114 into register r12, again.
    - 0xE3A0C072
    - 1110 0011 1010 0000 1100 0000 0111 0010
  - The value 114 is given in the least significant byte
    - 0xE3A0C072
    - 114 in decimal is 72 in hexadecimal.
  - Register r12 is given by the 5th digit
    - 0xE3A0C072
    - 12 in decimal is C in hexadecimal.

Mnemonics

- In general nobody remembers all of the machine code for any particular processor (or indeed any).
- Instead we use mnemonics
  - mnemonics are words or phrases which are easy to remember and
  - can replace something which is difficult to remember.
- The instruction to move the value 114 into register r12 has the mnemonic
  - MOV r12, #114.
  - This is much easier to remember than 0xE3A0C072

Instructions for Arithmetic

- The ARM7 can add, subtract and multiply numbers (but not divide).
- The mnemonic for add the value in register x to the value in register y and place the sum in register z is: ADD rz, ry, rx
- E.g. to add the value in register r1 to the value in register r2 and leave the sum in register r3
  - the mnemonic is: ADD r3, r2, r1
  - We don't need to know the machine code.

Instructions using logic

- As well as arithmetic the ARM7 can also do logic such as AND and OR.
- AND value in rx with value in ry and leave the result in rz has the mnemonic:

  AND rz, ry, rx
- OR value in rx with value in ry and leave the result in rz has the mnemonic:

  ORR rz, ry, rx

- ✓ Organization/Architecture
- ✓ Mnemonics
- ✓ **Flags**
- ✓ Representations
- ✓ Subroutines
- ✓ Interrupt
- ✓ Pipeline
- ✓ Encoding
- ✓ Memory
- ✓ Processor Modes
- ✓ Peripheral Communication

Flags

Flags are used to give a signal that something either has or has not happened.

For example, the flag is raised to half mast on in front of buildings as a signal that someone important is dead.

Microprocessors also use “flags” to signify what has happened.

Flags and Processors

Flags are essentially 1 bit memory devices and different microprocessors have different flags.

The ARM microprocessor has four flags that are commonly found in the majority of microprocessors.

These flags are the zero flag (Z), the negative flag (N), the carry flag (C), and the overflow flag (V).

Flags are also know as ‘condition codes’.

What are flags used for?

Flags are used in two ways:

The main use of flags is to determine if another instruction is executed or not. This is called conditional execution.

The carry flag can also be used in some arithmetic instructions as an additional value (we will return to this later).

Branches

- An important use of conditional execution is in branches.
- Conditional branches are used when a processor performs one function rather than another
  - e.g.
    - display text messages rather than receive an incoming call.
- First consider the unconditional branch.

Flags - summary.

- The zero flag, Z, is set when the result (not including any carry out) is 0x00000000.
- The negative flag, N, is set when the most significant bit of the 32 bit result is 1.
- The carry flag, C, is set when the result (taken as an unsigned integer) is greater than $(2^{32} - 1)$.
- The overflow flag, V, is set when the result (taken as a two's compliment number) is greater than $(2^{31} - 1)$ or less than -2^{31}

- ✓ Organization/Architecture
- ✓ Mnemonics
- ✓ Flags
- ✓ **Representations**
- ✓ Subroutines
- ✓ Interrupt
- ✓ Pipeline
- ✓ Encoding
- ✓ Memory
- ✓ Processor Modes

Representing characters

- As well as manipulating numbers, computers also manipulate characters e.g.
  - in a word processing package like Microsoft Word.
  - Characters must be coded in binary before computers can process them
  - the standard coding for characters is ASCII (American Standard Code for Information Interchange).

Negative numbers

- There are two main methods for representing negative numbers in microprocessors.
- These are:
  1) Sign magnitude.
  2) Two's complement.
- In each case the most significant bit - 'm.s.b.' - indicates the sign (1 for -ve and 0 for +ve).

2's complement

- The two's complement method automatically sets the m.s.b. or 'sign bit' to 1.
- The following method can be used to find a 2's complement representation of a negative number,
  - e.g.
    - -20640
    - First find the positive value: 0x000050A0 or 0000 0000 0000 0000 0101 0000 1010 0000_2
    - Next invert all bits (0→ 1, 1 → 0).
      - 1111 1111 1111 1111 1010 1111 0101 1111_2 or
      - 0xFFFFAF5F.
    - And then add 1 ➔0xFFFFAF60

Floating point numbers

- The floating point format uses the equation $A \times 2^n$ to represent numbers.
- Using the IEEE 754 standard the A value has a sign and 24 bits and the exponent value, n, has 8 bits.
- The number $5{,}637{,}144{,}576_{10}$ ($= 2^{32} + 2^{30} + 2^{28}$) is too big for a 32 bit unsigned integer.
- In binary this number is
  - $1\ 0101\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2$
- First we must normalize this number as follows:

5,637,144,57610 in IEEE 754

- $1\ 0101\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = 1.0101\ 0000\ 0000\ 0000\ 0000\ 000_2 \times 2^{32}$
- The exponent is $32_{10} = 100000_2$ but in the IEEE 754 format we must add $127_{10} = 1111111_2$to this to give 10011111_2 $(=159_{10})$
- In 32 bits this becomes:

  0100 1111 1010 1000 0000 0000 0000 0000
- Note that the normalization means that the m.s.b. of the A value is always 1 so this can be omitted.
- The m.s.b. of the 32 bit word is a sign bit.

- ✓ Organization/Architecture
- ✓ Mnemonics
- ✓ Flags
- ✓ Representations
- ✓ **Subroutines**
- ✓ Interrupt
- ✓ Pipeline
- ✓ Encoding
- ✓ Memory
- ✓ Processor Modes
- ✓ Peripheral Communication

Subroutines

- Many microprocessor programs include groups of instructions which are repeated many times.
- It is wasteful of memory to include these instructions in the program again and again.
- Instead they can be included once in a special structure known as a subroutine.
- The main program branches to the start of the subroutine when it requires those particular instructions.
- At the end of the subroutine there is another branch back to the main program.

Problem

- At the end of the subroutine how does the processor know which instruction to return to?
- This problem is solved by using a 'link register'.
- A link register holds the memory address of the instruction in the main program to which the subroutine returns to.
- Register r14 in the ARM microprocessor is designated as the link register and 'r14' can be replaced by 'lr' in mnemonics.

Branch and link

- The mnemonic for ‘branch and link’ is BL;
  - a simple branch with mnemonic B does not update the link register.
- To return from the subroutine the value in the link register is moved into the program counter;
  - MOV pc, lr.
- What happens if a branch and link occurs in a subroutine?
  - The value held in the link register will be overwritten by a new return address so before one subroutine calls another the link register value must be stored elsewhere.

Nested subroutines

- Complicated programs will have several subroutines and some subroutines will call other subroutines
  - this is standard practice and it is known as nesting.
- If subroutine A calls subroutine B the link register can not store the return address for both subroutines at the same time.
- In order to preserve the return address of all subroutines an area of computer memory called the stack is used.
- The stack is a 'last in first out' queue.

The stack

- The stack is a last in first out queue
  - that means whatever data was added to the stack (‘pushed’) last is taken from the stack (‘popped’) first.
  - E.g. if the values pushed onto a stack were 0x00FF, 0xFF00, 0xAAAA in that order then they would be popped from the stack in the reverse order.
- In memory the stack is held as a list:

top of stack 0xAAAA

0xFF00

bottom of stack 0x00FF

Another problem

- The stack is a dynamic memory structure
  - Meaning that the amount of data held in the stack can vary.
- The bottom of the stack can be fixed at a particular memory address.
- How do we know where the top of the stack is?
  - We need to know so that we can pop the correct data.
- Another register is used to identify the top of the stack - it is known as the stack pointer.

- ✓ Organization/Architecture
- ✓ Mnemonics
- ✓ Flags
- ✓ Representations
- ✓ Subroutines
- ✓ **Interrupt**
- ✓ Pipeline
- ✓ Encoding
- ✓ Memory
- ✓ Processor Modes
- ✓ Peripheral Communication

Interrupts

- Many applications for microprocessor systems require things to happen when an event occurs unexpectedly.
- These unexpected events can interrupt the normal operation of the microprocessor.
- During an ‘interrupt’ the execution of the main program is halted and a special program is executed instead.
- When the interrupt program has finished, the microprocessor returns to the main program.

Interrupts

- Interrupts provide a very convenient method for dealing with events and this method is often used for events that are not unexpected
  - e.g.
    - when a mobile phone receives an incoming call.
- An interrupt is triggered when the microprocessor receives a (voltage) signal on a special connection within the control bus.
- The ARM7 has two types of interrupts,
  - a normal interrupt (IRQ) and
  - a fast interrupt (FIQ).

Interrupt Handling

- When an interrupt occurs the following happens:
  ① The registers for IRQ mode or FIQ mode are activated.
  ② The current program status register, CPSR, (this contains the flags and other information) is saved into a saved program status register, SPSR.
     a) There are two SPSR registers, one for each mode.
  ③ The return address of the next instruction to be executed in the main program is stored in the link register for the appropriate mode.
  ④ The program counter is set to either 0x00000018 for an IRQ or 0x0000001C for a FIQ.

Interrupt Vectors

- The first instruction executed by an interrupt program is the instruction stored at memory address 0x00000018 for an IRQ or 0x0000001C for a FIQ.
- These memory addresses are known as ‘vectors’.
- For an IRQ the instruction at address 0x00000018 must be a branch to another part of memory because the following memory location, 0x0000001C contains the first instruction of the FIQ handler.

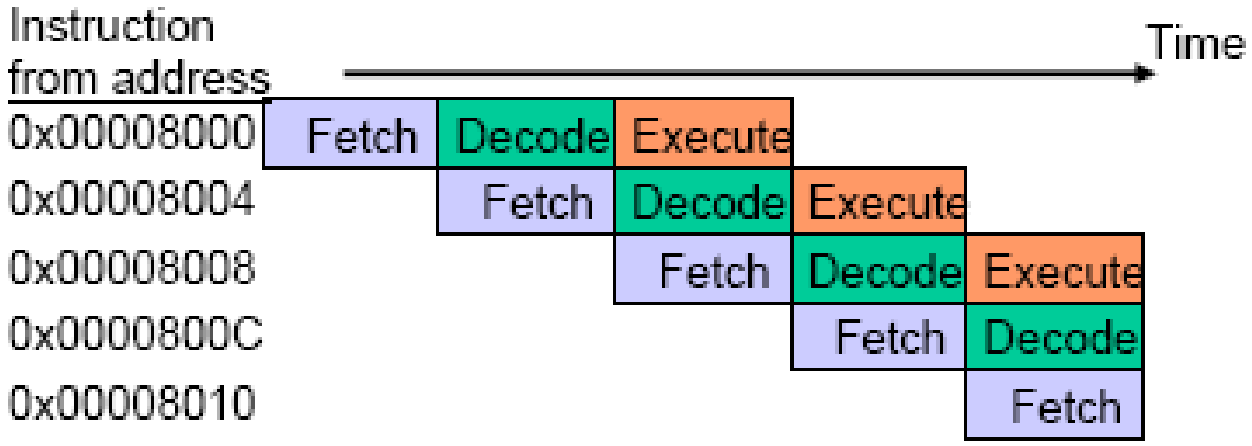
- ✓ Organization/Architecture
- ✓ Mnemonics
- ✓ Flags
- ✓ Representations
- ✓ Subroutines
- ✓ Interrupt
- ✓ **Pipeline**
- ✓ Encoding
- ✓ Memory
- ✓ Processor Modes
- ✓ Peripheral Communication

Instruction Pipelines

- Instruction pipelines are an important feature of all modern microprocessors.
- For the same basic speed of transistor operation, an n stage instruction pipeline allows the microprocessor to execute up to n times as many instructions in a given time.
- The ARM7 microprocessor has a three stage pipeline
  - one stage for each of the CPU cycles;
    - fetch, decode and execute.

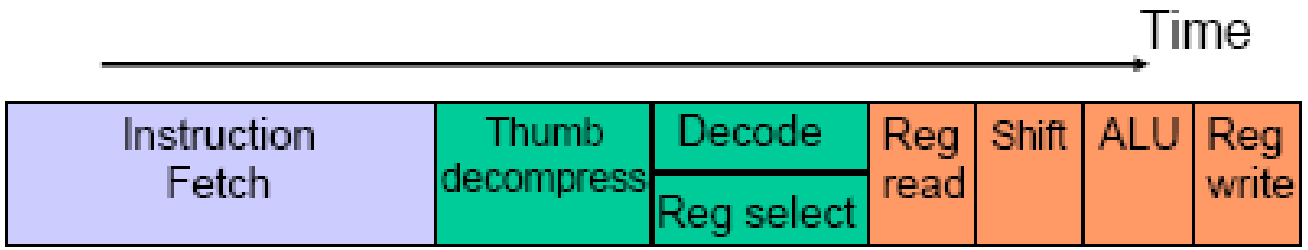
Instruction Pipelines

- In a three stage pipeline, the CPU can simultaneously execute an instruction, decode the next instruction and fetch the next instruction.



ARM7 3 stage pipeline: detail

In each stage of the ARM7 pipeline, several things happen; normally consecutively:



FETCH
Instruction fetched from memory.

DECODE
Thumb to ARM decompression.
Decode.
Register selection

EXECUTE
Register read.
Barrel shift.
ALU operation.
Register write.

For Internal Use Only!

- ✓ Organization/Architecture
- ✓ Mnemonics
- ✓ Flags
- ✓ Representations
- ✓ Subroutines
- ✓ Interrupt
- ✓ Pipeline
- ✓ **Encoding**
- ✓ Memory
- ✓ Processor Modes
- ✓ Peripheral Communication

ARM7 pipeline example

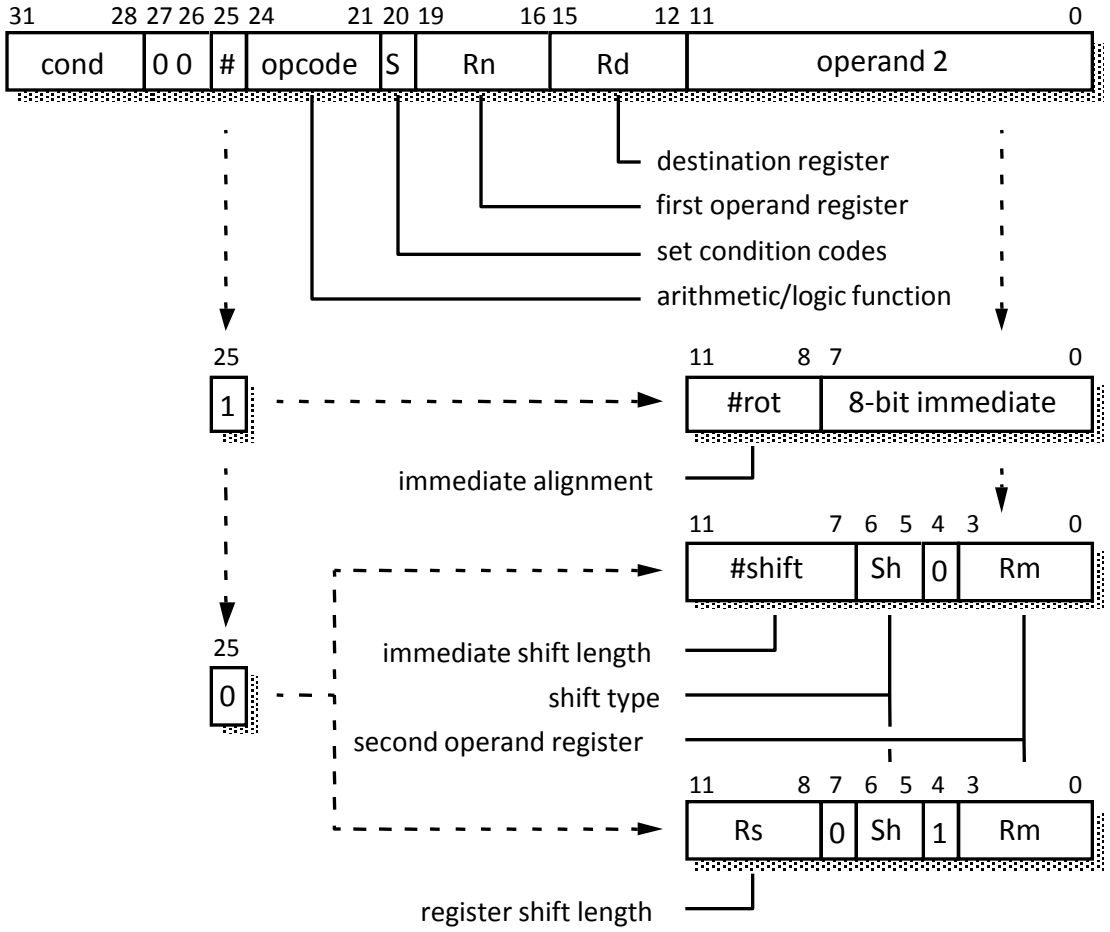
Consider a typical three operand ARM7 instruction e.g.

ADDNE r11, r3, r7 LSR #8

Add the contents of register r3 to the contents of register r7 shifted right by 8 bits and put the sum in register r11 if the zero flag is clear (Z=0).

The machine code for this instruction is 0x1083B427

Data processing instruction binary encoding



For Internal Use Only!

ARM data processing instructions

Opcode [24:21]	Mnemonic	Meaning	Effect
0000	AND	Logical bit-wise AND	Rd := Rn AND Op2
0001	EOR	Logical bit-wise exclusive OR	Rd := Rn EOR Op2
0010	SUB	Subtract	Rd := Rn - Op2
0011	RSB	Reverse subtract	Rd := Op2 - Rn
0100	ADD	Add	Rd := Rn + Op2
0101	ADC	Add with carry	Rd := Rn + Op2 + C
0110	SBC	Subtract with carry	Rd := Rn - Op2 + C - 1
0111	RSC	Reverse subtract with carry	Rd := Op2 - Rn + C - 1
1000	TST	Test	Scc on Rn AND Op2
1001	TEQ	Test equivalence	Scc on Rn EOR Op2
1010	CMP	Compare	Scc on Rn - Op2
1011	CMN	Compare negated	Scc on Rn + Op2
1100	ORR	Logical bit-wise OR	Rd := Rn OR Op2
1101	MOV	Move	Rd := Op2
1110	BIC	Bit clear	Rd := Rn AND NOT Op2
1111	MVN	Move negated	Rd := NOT Op2

For Internal Use Only!

- ✓ Organization/Architecture
- ✓ Mnemonics
- ✓ Flags
- ✓ Representations
- ✓ Subroutines
- ✓ Interrupt
- ✓ Pipeline
- ✓ Encoding
- ✓ **Memory**
- ✓ Processor Modes
- ✓ Peripheral Communication

What is computer memory?

- Computer memory is a very big sequential logic circuit made up of thousands or millions of simple logic gates, such as a D type latch, which can remember a 0 or a 1, that is one bit of data.
- Groups of these gates are collected together in a memory ‘location’.
- There are typically 8 bits of data in one location.
- Each memory location has a unique memory address.

Memory organization.

- Taking the ARM7TDMI microprocessor as an example
  - at each memory location
  - it has 8 bits of data
    - 8 bits is known as a byte.
- The ARM is a 32 bit processor and addresses are 32 bits long from 0x00000000 to 0xFFFFFFFF.
  - That means there can be up to 4,294,967,296 (or 2^{32}) different memory locations all with a unique memory address.
- In practice not all addresses are used for memory.

Addressing modes

- The concept of addressing mode is common to all microprocessors and an understanding of addressing mode is important for users of microprocessors.
- The ARM7 microprocessor supports many different addressing modes and we will concentrate on just four.

Load and Store

- When a load instruction is executed
  - the data travels from memory to register
- whereas when a store instruction is executed
  - the data travels from register to memory.
- Each memory location holds one byte or 8 bits of data whereas each register holds 4 bytes of data.
- So LDR and STR use four consecutive memory addresses but which byte goes to which location?

Little endian

- Microprocessors can be either ‘little endian’ or ‘big endian’
- If the processor is ‘little endian’ then the instruction:
  - STR r6, [r11]
  - with 0xFFAABB11 in r6 and 0x00008000 in r11
  - would store
    - byte 0x11 at address 0x00008000
    - 0xBB at 0x00008001
    - 0xAA at 0x00008002
    - 0xFF at 0x00008003

Big endian

- Whereas if the processor is ‘big endian’ then the instruction:
  - STR r6, [r11]
  - with 0xFFAABB11 in r6 and 0x00008000 in r11
  - would store
    - byte 0xFF at address 0x00008000
    - 0xAA at 0x00008001
    - 0xBB at 0x00008002
    - 0x11 at 0x00008003

Types of Silicon Memory

- Silicon IC memory can be classified as either read-only or read-write.
- Read-only memory (or ROM) is used for the operating system on a desktop computer or an application program in an embedded system such as a mobile phone.
- Read-write memory (which is normally called RAM) is used for an application program on a desktop computer and for temporary data.

Types of ROM

- There are several generations of silicon ROM.
- Programmable ROM (PROM) could be electrically programmed once only and then the contents were fixed.
- Erasable PROM (EPROM) could also have its contents erased by exposure to ultraviolet (UV) light and could then be reprogrammed.
- Electrically erasable PROM (EEPROM) could have its contents erased by an electrical signal.

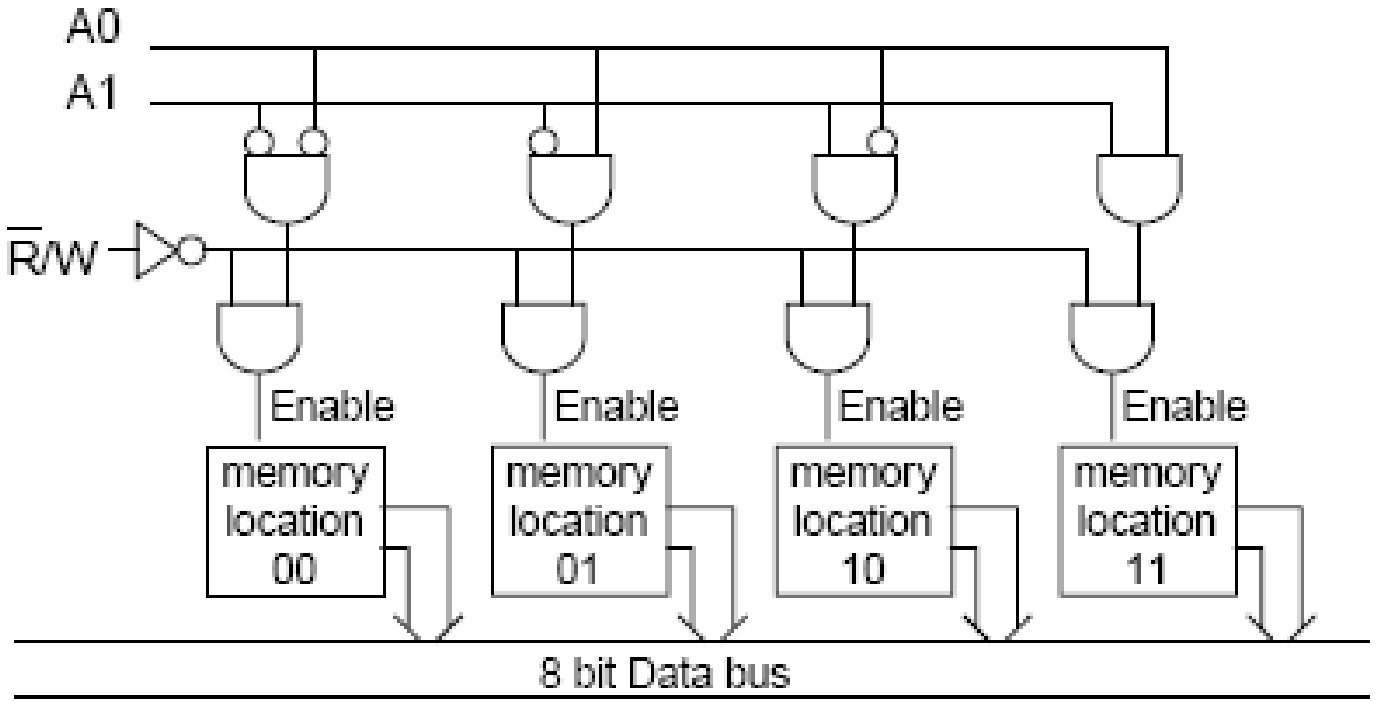
Types of ROM

- Flash Erasable PROM (FEPROM or flash memory) is similar to EEPROM but erasing could only be done over a significant part of (maybe all) the integrated circuit.
- All ROM is non-volatile - meaning the information stored in ROM memory is not lost when it is disconnected from a power source. The data stored in ROM will remain almost indefinitely e.g. many tens of years.
- In contrast the contents of RAM memory will be lost almost as soon as power is switched off - RAM is volatile.

Cache memory

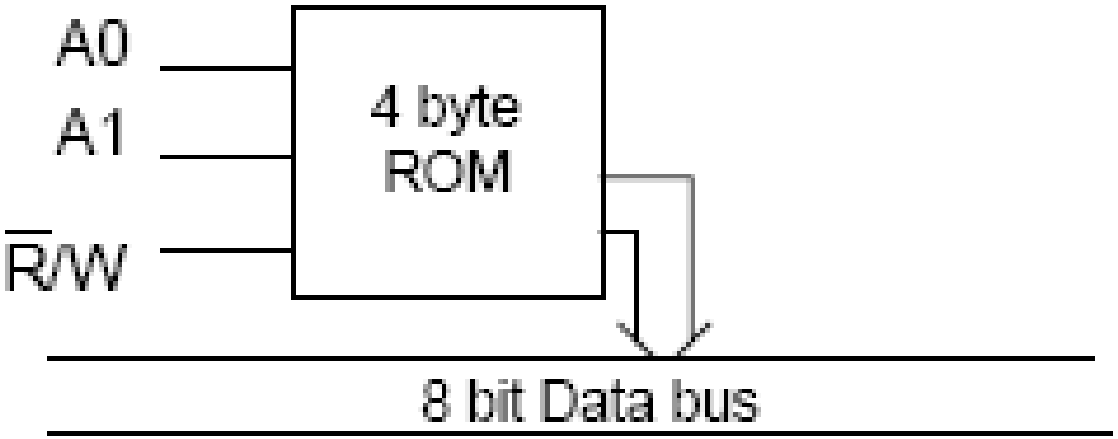
- When data is read from, or written to, main memory a copy of the data is saved in the memory cache (along with the main memory address).
- When a subsequent read occurs the cache can be checked to see if the required data is already there.
- If it is (a cache ‘hit’) then it can be supplied immediately.
- If not (a cache ‘miss’) then it must be fetched from main memory.

Simple 4 byte ROM addressing



Simple ROM addressing

- So memory location 00 is ‘enabled’ and connected to the data bus only when R*/W = 0, A0 = 0 and A1 = 0.
- The previous circuit can be represented by a single block as follows:



Tri-state gates

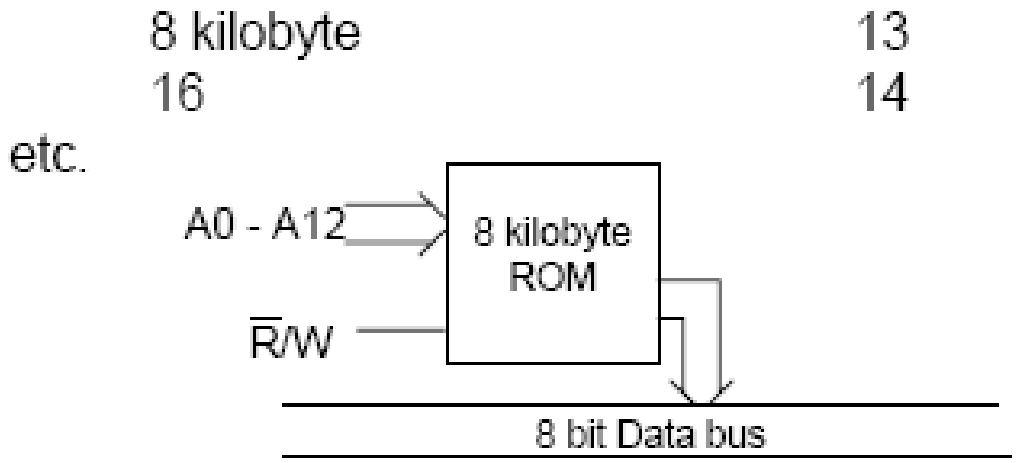
- Only one memory location should be connected to the bus at one time – one device can not drive a logic 0 onto a bus line at the same time as another device drives a logic 1 onto the same line.
- This is achieved using ‘tri-state gates’ which have an ‘enable’ input – when not enabled the output acts like an ‘open circuit’ which is referred to as ‘high impedance’.

Tri-state gates

enable	input	output
0	0	high-impedance
0	1	high-impedance
1	0	0
1	1	1

Simple ROM addressing

- Bigger memories would have similar connections but there would be more address lines.
- A 2^N byte memory would need N address lines:
- E.g. a 4 kilobyte memory would need 12 address lines.



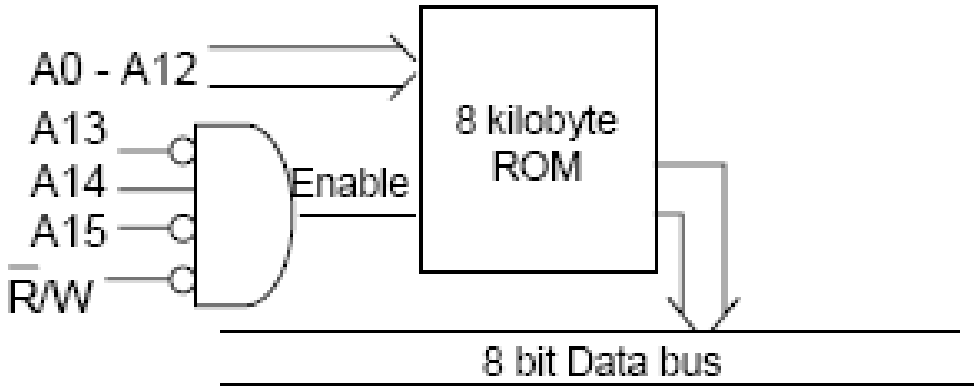
Which address?

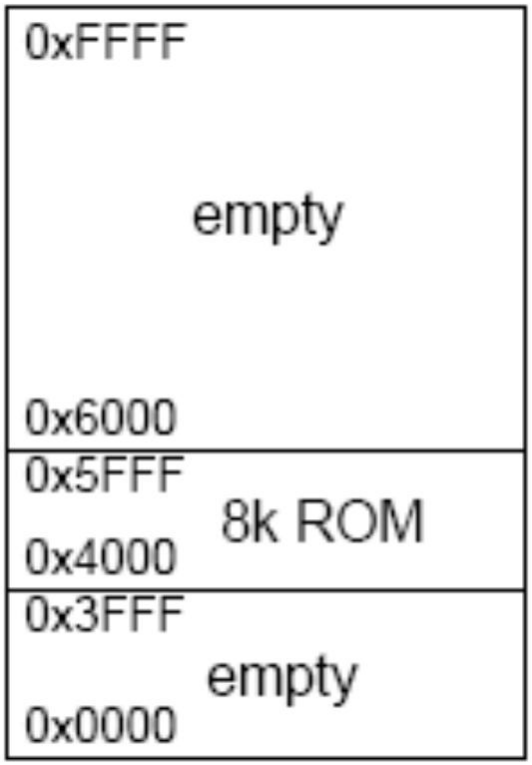
- If an 8 kilobyte memory chip is connected to a processor with a 16 bit address bus whereabouts is the memory located within in the 64 kilobyte address space?
- The lowest N address lines are connected to the 2^N byte memory; in this case A0 to A12. The other address lines, A13 to A15, must be decoded e.g. if the memory range is from 0x4000 to 0x5FFF or in binary:

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1

8 kilobyte ROM addressing

- The memory is only enabled when A13 = 0, A14 = 1, A15 = 0 and R*/W = 0.

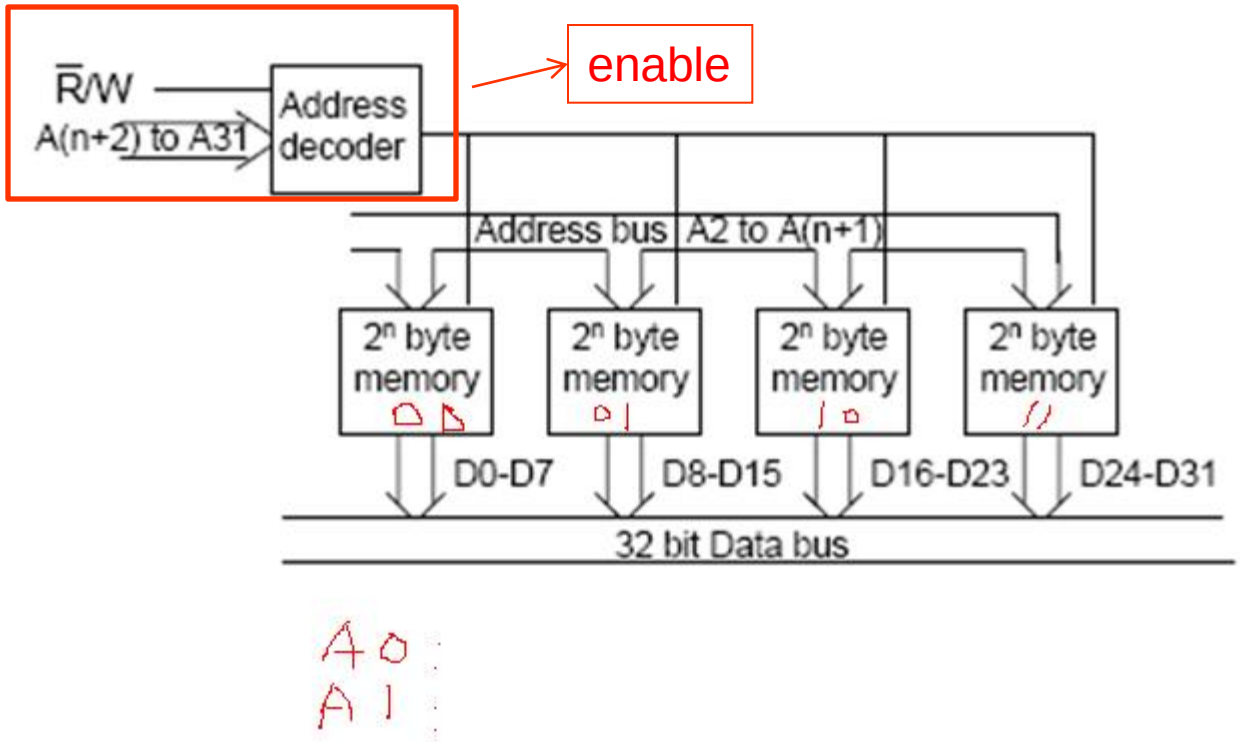




- If the address bus is 32 bits wide then all the address lines not connected to the memory chip, A13 to A31, are decoded.

Connecting to a 32 bit data bus

- Memory with 8 data bits output can be connected to a 32 bit data bus by using 4 separate memory chips; one for each byte

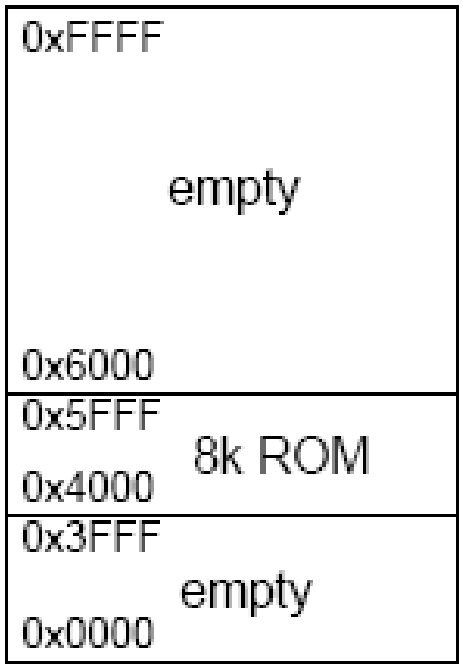


$A_0 – A_{n+1}$: memory size

$A_{n+2} – A_{31}$: location in the memory map

Memory maps

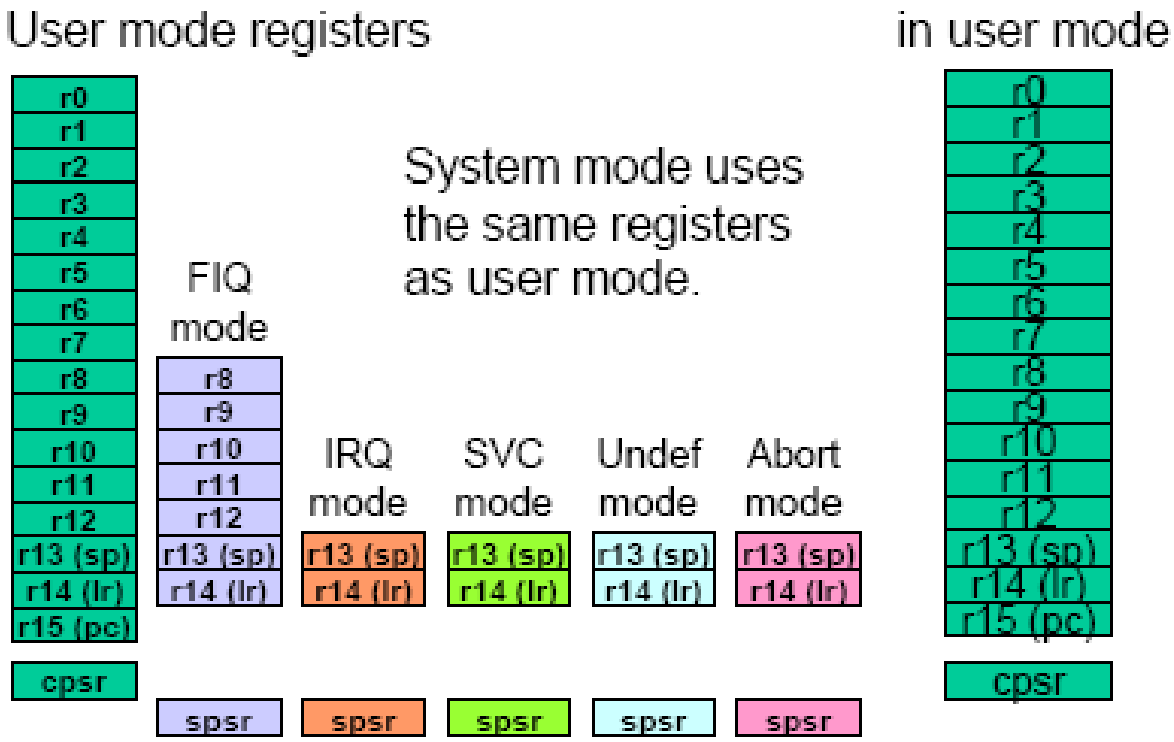
- The location of a memory chip within the complete addressable space can be represented by a memory map.
- E.g. for the 8 kilobyte ROM memory between addresses 0x4000 and 0x5FFF in a 64 kilobyte addressable space:



For Internal Use Only!

- ✓ Organization/Architecture
- ✓ Mnemonics
- ✓ Flags
- ✓ Representations
- ✓ Subroutines
- ✓ Interrupt
- ✓ Pipeline
- ✓ Encoding
- ✓ Memory
- ✓ **Processor Modes**
- ✓ Peripheral Communication

Different registers in each mode



- ✓ Organization/Architecture
- ✓ Mnemonics
- ✓ Flags
- ✓ Representations
- ✓ Subroutines
- ✓ Interrupt
- ✓ Pipeline
- ✓ Encoding
- ✓ Memory
- ✓ Processor Modes
- ✓ **Peripheral Communication**

Input/Output (I/O) ports

- I/O ports are the specific components of a microcomputer that allow it to interact with its environment.
- A **device driver** is a collection of software functions that allow higher level software to utilize an I/O device.
- In other words, the set of low-level functions that input/output directly with the hardware are grouped together in a single module and called a device driver.

Peripheral Communication (through I/O port)

Abbreviation	Full Name
UART	Universal asynchronous receiver/transmitter
SSI/SPI	Synchronous serial interface/Serial peripheral interface
I^2C	Inter-integrated circuit
USB	Universal serial bus
Ethernet	High-speed network
CAN	Controller area network
…	

Universal Asynchronous Receiver/Transmitter (UART)

The **UART** can be used for serial communication between computers. It is asynchronous and allows for simultaneous communication in both directions.

Serial transmission involves sending one bit at a time, such that the data is spread out over time

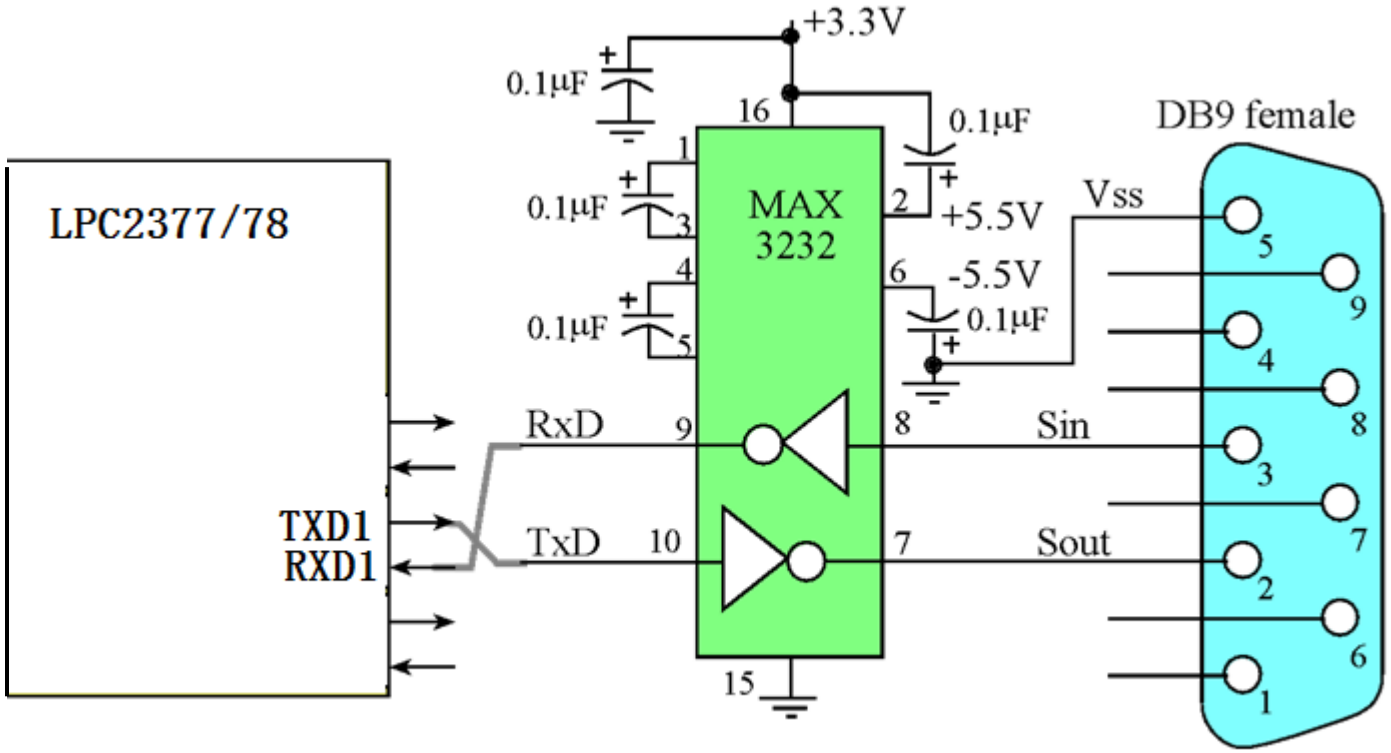
When designing systems with any I/O module, you must also refer to the reference manual of your specific microcontroller

NXP Semiconductors: LPC2377/78

pin number

P2[0]/PWM1[1]/ TXD1/ TRACECLK	107[1]	I/O	**P2[0]** — General purpose digital input/output pin.
		O	**PWM1[1]** — Pulse Width Modulator 1, channel 1 output.
		O	**TXD1** — Transmitter output for UART1.
		O	**TRACECLK** — Trace Clock.
P2[1]/PWM1[2]/ RXD1/ PIPESTAT0	106[1]	I/O	**P2[1]** — General purpose digital input/output pin.
		O	**PWM1[2]** — Pulse Width Modulator 1, channel 2 output.
		I	**RXD1** — Receiver input for UART1.
		O	**PIPESTAT0** — Pipeline Status, bit 0.
P2[2]/PWM1[3]/ CTS1/ PIPESTAT1	105[1]	I/O	**P2[2]** — General purpose digital input/output pin.
		O	**PWM1[3]** — Pulse Width Modulator 1, channel 3 output.
		I	**CTS1** — Clear to Send input for UART1.
		O	**PIPESTAT1** — Pipeline Status, bit 1.
P2[3]/PWM1[4]/ DCD1/ PIPESTAT2	100[1]	I/O	**P2[3]** — General purpose digital input/output pin.
		O	**PWM1[4]** — Pulse Width Modulator 1, channel 4 output.
		I	**DCD1** — Data Carrier Detect input for UART1.
		O	**PIPESTAT2** — Pipeline Status, bit 2.
P2[4]/PWM1[5]/ DSR1/ TRACESYNC	99[1]	I/O	**P2[4]** — General purpose digital input/output pin.
		O	**PWM1[5]** — Pulse Width Modulator 1, channel 5 output.
		I	**DSR1** — Data Set Ready input for UART1.
		O	**TRACESYNC** — Trace Synchronization.

RS-232 Serial Port



DB25 Pin	RS232 Name	DB9 Pin	EIA-574 Name	Signal	Description	True	DTE	DCE
2	BA	3	103	TxD	Transmit Data	-5.5V	out	in
3	BB	2	104	RxD	Receive Data	-5.5V	in	out
7	AB	5	102	SG	Signal Ground			

LPC23XX User manual

2.3 APB peripheral addresses

The following table shows the APB address map. No APB peripheral uses all of the 16 kB space allocated to it. Typically each device's registers are "aliased" or repeated at multiple locations within each 16 kB range.

Table 10. APB peripherals and base addresses

APB Peripheral	Base Address	Peripheral Name
0	0xE000 0000	Watchdog Timer
1	0xE000 4000	Timer 0
2	0xE000 8000	Timer 1
3	0xE000 C000	UART0
4	0xE001 0000	UART1
5	0xE001 4000	Not used
6	0xE001 8000	PWM1
7	0xE001 C000	I^2C0
8	0xE002 0000	SPI
9	0xE002 4000	RTC
10	0xE002 8000	GPIO
11	0xE002 C000	Pin Connect Block
12	0xE003 0000	SSP1
13	0xE003 4000	ADC
14	0xE003 8000	CAN Acceptance Filter RAM[1]
15	0xE003 C000	CAN Acceptance Filter Registers[1]
16	0xE004 0000	CAN Common Registers[1]
17	0xE004 4000	CAN Controller 1[1]
18	0xE004 8000	CAN Controller 2[1]
19 to 22	0xE004 C000 to 0xE005 8000	Not used
23	0xE005 C000	I^2C1
24	0xE006 0000	Not used
25	0xE006 4000	Not used
26	0xE006 8000	SSP0
27	0xE006 C000	DAC
28	0xE007 0000	Timer 2
29	0xE007 4000	Timer 3
30	0xE007 8000	UART2
31	0xE007 C000	UART3
32	0xE008 0000	I^2C2
33	0xE008 4000	Battery RAM
34	0xE008 8000	I^2S
35	0xE008 C000	SD/MMC Card Interface[2]
36 to 126	0xE009 0000 to 0xE01F BFFF	Not used
127	0xE01F C000	System Control Block

[1] CAN interface is available in LPC2364/66/68, LPC2378, LPC2387, and LPC2388.

Table 369. UART1 register map

Name	Description	Bit functions and addresses								Access	Reset Value[1]	Address
		MSB							LSB			
		BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0			
U1RBR	Receiver Buffer Register	8 bit Read Data								RO	NA	0xE001 0000 (DLAB=0)
U1THR	Transmit Holding Register	8 bit Write Data								WO	NA	0xE001 0000 (DLAB=0)
U1DLL	Divisor Latch LSB	8 bit Data								R/W	0x01	0xE001 0000 (DLAB=1)
U1DLM	Divisor Latch MSB	8 bit Data								R/W	0x00	0xE001 0004 (DLAB=1)
U1IER	Interrupt Enable Register	Reserved						Enable Autobaud Time-Out Interrupt	Enable End of Autobaud Interrupt	R/W	0x00	0xE001 0004 (DLAB=0)
		Enable CTS Interrupt		0		Enable Modem Status interrupt	Enable RX Line Status Interrupt	Enable THRE Interrupt	Enable RX Data Available Interrupt			
U1IIR	Interrupt ID Register	Reserved						ABTO Itn	ABEO int	RO	0x01	0xE001 0008
		FIFOs Enabled		0		IIR3	IIR2	IIR1	IIR0			
U1FCR	FIFO Control Register	RX Trigger		Reserved			TX FIFO Reset	RX FIFO Reset	FIFO Enable	WO	0x00	0xE001 0008
U1LCR	Line Control Register	DLAB	Set Break	Stick Parity	Even Parity Select	Parity Enable	Number of Stop Bits	Word Length Select		R/W	0x00	0xE001 000C
U1MCR	Modem Control Register	CTSen	RTSen	0	Loop Back	0		RTS	DTR	R/W	0x00	0xE001 0010
U1LSR	Line Status Register	RX FIFO Error	TEMT	THRE	BI	FE	PE	OE	DR	RO	0x60	0xE001 0014
U1MSR	Modem Status Register	DCD	RI	DSR	CTS	Delta DCD	Trailing Edge RI	Delta DSR	Delta CTS	RO	0x00	0xE001 0018

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2012. All rights reserved.